







Diese Edition bezieht sich auf Version 7, Release 0 von WebSphere eXtreme Scale und alle nachfolgenden Releases und Bearbeitungen, sofern kein anderer Hinweis in den neuen Editionen enthalten ist.

© Copyright International Business Machines Corporation 2009, 2009.

# Inhaltsverzeichnis

<b>Abbildungsverzeichnis</b> . . . . .	<b>v</b>
<b>Tabellen</b> . . . . .	<b>vii</b>
<b>Informationen zu <i>Programmierhandbuch</i></b>	<b>ix</b>
<b>Kapitel 1. Einführung in WebSphere eXtreme Scale</b> . . . . .	<b>1</b>
<b>Kapitel 2. WebSphere eXtreme Scale programmieren.</b> . . . . .	<b>7</b>
<b>Kapitel 3. Auf Daten in WebSphere eXtreme Scale zugreifen.</b> . . . . .	<b>9</b>
Interaktion mit einem ObjectGrid über ObjectGrid-Manager . . . . .	12
createObjectGrid-Methoden . . . . .	12
getObjectGrid-Methoden . . . . .	16
removeObjectGrid-Methoden . . . . .	17
Verbindung zu einem verteilten ObjectGrid herstellen . . . . .	18
Lebenszyklus eines ObjectGrids steuern . . . . .	23
Auf das ObjectGrid-Shard zugreifen . . . . .	25
Session-Objekte für den Zugriff auf Daten im Grid verwenden . . . . .	25
Handhabung von Sperren . . . . .	30
Transaktionsisolation . . . . .	40
SessionHandle für Routing . . . . .	42
Optimistische Kollisionsausnahme. . . . .	42
ObjectMap-API . . . . .	44
Einführung in ObjectMap. . . . .	44
Dynamische Maps . . . . .	48
ObjectMap und JavaMap . . . . .	50
Maps als FIFO-Warteschlangen . . . . .	51
Anwendungsprogrammierschnittstelle "EntityManager" . . . . .	54
API "ObjectMap" versus API "EntityManager". . . . .	57
Entitätsschema definieren. . . . .	57
Integrierte Server-API . . . . .	65
Leistungseinfluss der Schnittstelle "EntityManager" . . . . .	67
EntityManager in einer verteilten Umgebung . . . . .	71
Abfragewarteschlangen für Entitäten . . . . .	74
Schnittstelle "EntityTransaction" . . . . .	78
Lebenszyklen von Entitäteninstanzen. . . . .	79
API "Query" . . . . .	85
API "ObjectQuery" verwenden . . . . .	89
EntityManager-API "Query". . . . .	94
Referenzinformationen zu eXtreme-Scale-Abfragen . . . . .	98
Optimierung der Abfrageleistung. . . . .	109
Indexierung . . . . .	121
Indexierung für den Datenzugriff ohne Schlüssel verwenden . . . . .	123

Zusammengesetzter Hash-Index . . . . .	127
DataGrid-API . . . . .	129
DataGrid-APIs und Partitionierung . . . . .	129
DataGrid-Agenten und entitätsbasierte Maps . . . . .	130
Beispiel für die DataGrid-APIs . . . . .	130
API-Dokumentation . . . . .	135

<b>Kapitel 4. Systemanwendungsprogrammierschnittstellen und -Plug-ins</b> . . . . .	<b>137</b>
Einführung in Plug-ins . . . . .	137
Ereignis-Listener . . . . .	139
MapEventListener-Plug-in . . . . .	140
ObjectGridEventListener-Plug-in . . . . .	141
Index-Plug-in schreiben . . . . .	143
TransactionCallback-Plug-in . . . . .	146
Einführung in Plug-in-Slots. . . . .	151
Externe Transaktionsmanager . . . . .	153
Loader verwenden . . . . .	156
Loader schreiben . . . . .	158
Hinweise zur Programmierung von JPA-Loadern . . . . .	162
JPAEntityLoader-Plug-in. . . . .	164
Loader mit Entitäts-Maps und Tupeln verwenden. . . . .	166
Loader mit einem Preload-Controller für Replikate schreiben . . . . .	172
LogElement und LogSequence. . . . .	176
eXtreme Scale mit JPA verwenden . . . . .	179
Übersicht über das clientbasierte JPA-Preload-Dienstprogramm . . . . .	180
Zeitbasierte JPA-Datenaktualisierungskomponente. . . . .	188
OptimisticCallback-Plug-in . . . . .	193
ObjectTransformer-Plug-in . . . . .	197
WebSphereTransactionCallback-Plug-in . . . . .	202

<b>Kapitel 5. Integration mit dem Spring-Framework</b> . . . . .	<b>203</b>
Native Transaktionen. . . . .	204
Spring-Erweiterungs-Beans und Unterstützung von Namespaces . . . . .	208

<b>Kapitel 6. Sicherheits-API</b> . . . . .	<b>213</b>
Programmierung der Clientauthentifizierung . . . . .	214
Programmierung der Clientberechtigung . . . . .	232
Grid-Authentifizierung . . . . .	240
Lokale Sicherheit . . . . .	241

<b>Kapitel 7. Verwaltungs-API zum Starten eines integrierten eXtreme-Scale-Containerservers verwenden.</b> . . . . .	<b>247</b>
--	------------

<b>Kapitel 8. Leistungsaspekte</b> . . . . .	<b>251</b>
Optimierung von JVMs . . . . .	251

Bewährte Verfahren für CopyMode . . . . .	253
Maps für Bytefeldgruppen . . . . .	258
Bereinigung . . . . .	261
Bewährte Verfahren für Standardbereinigungs- programme . . . . .	263
Angepasstes Bereinigungsprogramm schreiben	265
Bewährte Verfahren zum Erreichen einer opti- malen Leistung der Plug-in-Bereinigungs- programme . . . . .	270
Bewährte Verfahren für die Sperrleistung . . . . .	272
Sperrungen von Map-Einträgen mit Abfragen und Indizes . . . . .	273
Bewährte Verfahren für die Schnittstelle "Object- Transformer" . . . . .	276
Serialisierungsleistung . . . . .	277
Optimierung der Abfrageleistung. . . . .	278
Abfrageoptimierung mit Indizes . . . . .	280

Abfrageplan . . . . .	287
-----------------------	-----

**Kapitel 9. Fehlerbehebung . . . . . 291**

Protokolle und Trace . . . . .	291
Trace-Optionen . . . . .	293
Nachrichten . . . . .	295
Releaseinformationen. . . . .	295

**Kapitel 10. Glossar . . . . . 297**

**Bemerkungen . . . . . 323**

**Marken . . . . . 325**

**Index . . . . . 327**

---

## Abbildungsverzeichnis

1.	objectgrid.xml. . . . .	72	9.	Interaktion der Abfrage mit den ObjectGrid-Objekt-Maps, Definition und Zuordnung des Entitätsschemas zu einer ObjectGrid-Map . . .	95
2.	entity.xml . . . . .	72	10.	Loader. . . . .	156
3.	Verbindung zu einem verteilten Server	72	11.	Clientladeprogramme, die die JPA-Implementierung zum Laden des ObjectGrids verwenden . . . . .	180
4.	ServerPerson.java . . . . .	73	12.	Regelmäßige Aktualisierung . . . . .	189
5.	ClientPerson.java. . . . .	73	13.	Ablauf der Clientauthentifizierung und -berechtigung . . . . .	213
6.	Beispiel für eine verwaltete Entität . . . . .	80			
7.	Beispiel für eine freigegebene Entität . . . . .	80			
8.	Interaktion der Abfrage mit den ObjectGrid-ObjectMaps, Definition eines Schemas für Klassen und Zuordnung des Schemas zu einer ObjectGrid-Map . . . . .	90			





---

## Tabellen

1. Kompatibilitätsmatrix für Sperrmodi . . . . .	32	8. Nichteinhaltung der Reihenfolge mit U-Sperre	38
2. Deadlock-Szenario mit einem einzelnen Schlüssel . . . . .	35	9. Weitere Methoden . . . . .	89
3. Deadlocks mit einem einzigen Schlüssel, Fortsetzung . . . . .	36	10. Zusammenfassung der BNF-Notation	107
4. Deadlocks mit einem einzigen Schlüssel, Fortsetzung . . . . .	36	11. Modi des Clientladeprogramms . . . . .	181
5. Deadlocks mit einem einzigen Schlüssel, Fortsetzung . . . . .	36	12. Liste der Methoden und der erforderlichen MapPermissions . . . . .	234
6. Deadlock-Szenario mit mehreren Schlüsseln unter Einhaltung der Reihenfolge . . . . .	37	13. Liste der Methoden und der erforderlichen ObjectGridPermission . . . . .	235
7. Deadlock-Szenario mit mehreren Schlüsseln unter Einhaltung der Reihenfolge, Fortsetzung .	37	14. Berechtigungen für eine ObjectMap in einem Server . . . . .	235



---

## Informationen zu *Programmierhandbuch*

Das Dokumentationsset von WebSphere eXtreme Scale umfasst drei Handbücher, die die erforderlichen Informationen zur Verwendung des Produkts WebSphere eXtreme Scale, die Programmierung für das Produkt und die Verwaltung des Produkts enthalten.

### **Bibliothek von WebSphere eXtreme Scale**

Die Bibliothek von WebSphere eXtreme Scale enthält die folgenden Bücher:

- Das *Administratorhandbuch* enthält die für Systemadministratoren erforderlichen Informationen, z. B. Planung von Anwendungsimplementierungen, Kapazitätsplanung, Installation und Konfiguration des Produkts, Starten und Stoppen von Servern, Überwachung der Umgebung und Sicherung der Umgebung.
- Das *Programmierhandbuch* enthält Informationen für Anwendungsentwickler zur Entwicklung von Anwendungen für WebSphere eXtreme Scale unter Verwendung der bereitgestellten API-Informationen.
- Das Handbuch *Produktübersicht* enthält einer Übersicht über die Konzepte von WebSphere eXtreme Scale, einschließlich Anwendungsfallszenarios und Lernprogrammen.

Zum Herunterladen der Handbücher rufen Sie die Bibliotheksseite von WebSphere eXtreme Scale auf.

Sie finden die Informationen aus dieser Bibliothek auch im Information Center von WebSphere eXtreme Scale.

### **Zielgruppe**

Dieses Handbuch ist hauptsächlich für Anwendungsentwickler bestimmt.

### **Aufbau des Handbuchs**

Das Buch enthält Informationen zu den folgenden wichtigen Themen:

- **Kapitel 1** enthält einführende Informationen zu WebSphere eXtreme Scale.
- **Kapitel 2** enthält Informationen zur Programmierung von WebSphere eXtreme Scale.
- **Kapitel 3** enthält Informationen zum Zugriff auf Daten.
- **Kapitel 4** enthält Informationen zu den System-APIs und -Plug-ins.
- **Kapitel 5** enthält Informationen zur Integration des Spring-Frameworks.
- **Kapitel 6** enthält Informationen zur Sicherheits-API.
- **Kapitel 7** enthält Informationen zur Verwaltungs-API.
- **Kapitel 8** enthält Informationen zu Leistungsaspekten.
- **Kapitel 9** enthält Informationen zur Fehlerbehebung.
- **Kapitel 10** enthält das Produktglossar.

### **Aktualisierungen für dieses Handbuch**

Sie erhalten Aktualisierungen zu diesem Handbuch, indem Sie die jeweils aktuelle Version von der Bibliotheksseite von WebSphere eXtreme Scale herunterladen.

## **Hinweise zu Rückmeldungen**

Wenden Sie sich an das Dokumentationsteam. Haben Sie die benötigten Informationen gefunden? Sind die Informationen präzise und vollständig? Senden Sie Ihre Kommentare zu dieser Dokumentation per E-Mail an [wasdoc@us.ibm.com](mailto:wasdoc@us.ibm.com).

---

## Kapitel 1. Einführung in WebSphere eXtreme Scale

Nach der Installation von WebSphere eXtreme Scale in einer eigenständigen Umgebung verwenden Sie die folgenden Schritte als einfache Einführung in die Funktionalität des Produkts als speicherinternes Daten-Grid.

Die eigenständige Installation von WebSphere eXtreme Scale enthält ein Beispiel, das Sie verwenden können, um Ihre Installation zu prüfen und eigene Erfahrungen mit der Verwendung eines einfachen eXtreme-Scale-Grids und -Clients sammeln können. Das Einführungsbeispiel ist im Verzeichnis *Installationsstammverzeichnis/ObjectGrid/gettingstarted* enthalten.

Das Einführungsbeispiel ist eine Schnelleinführung in die Funktionen und die Basisoperationen von eXtreme Scale. Das Beispiel setzt sich aus Shell- und Stapel-scripts zusammen, mit denen ein einfaches Grid ohne umfassende Anpassungen gestartet werden kann. Außerdem wird ein Clientprogramm, einschließlich des Quellcodes, bereitgestellt, mit dem Sie einfache Erstellungs-, Lese-, Aktualisierungs- und Löschoptionen (CRUD, create, read, update, and delete) für dieses Basis-Grid ausführen können.

### Scripts und ihre Funktionen

Dieses Beispiel stellt die folgenden vier Scripts bereit:

Das Script `env.sh|bat` wird von den anderen Scripts aufgerufen, um die erforderlichen Umgebungsvariablen zu setzen. Normalerweise müssen Sie dieses Script nicht ändern.

- `./env.sh`
- `env.bat`

Das Script `runcat.sh|bat` startet den Katalogserviceprozess von eXtreme Scale auf dem lokalen System.

- `./runcat.sh`
- `runcat.bat`

Das Script `runcontainer.sh|bat` startet einen Containerserverprozess. Sie können dieses Script mehrfach mit eindeutigen Servernamen starten, um eine beliebige Anzahl an Containern zu starten. Diese Instanzen können zusammenarbeiten, um partitionierte und redundante Informationen im Grid zu speichern.

- `./runcontainer.sh eindeutiger_Servername`
- `runcontainer.bat eindeutiger_Servername`

Das Script `runclient.sh|bat` führt den einfachen CRUD-Client aus und startet die angegebene Operation.

- `./runclient.sh Befehl Wert1 Wert2`
- `runclient.sh Befehl Wert1 Wert2`

Für *Befehl* können Sie eine der folgenden Optionen einsetzen:

- Geben Sie *i* ein, um *Wert2* in das Grid mit dem Schlüssel *Wert1* einzufügen.
- Geben Sie *u* ein, um das Objekt mit dem Schlüssel *Wert1* in *Wert2* zu aktualisieren.

- Geben Sie `d` ein, um das Objekt mit dem Schlüssel `Wert1` zu löschen.
- Geben Sie `g` ein, um das Objekt mit dem Schlüssel `value1` abzurufen und anzuzeigen.

**Anmerkung:** Die Datei `Installationsstammverzeichnis/ObjectGrid/gettingstarted/src/Client.java` ist das Clientprogramm, das demonstriert, wie eine Verbindung zu einem Katalogserver hergestellt, eine `ObjectGrid`-Instanz abgerufen und die API "ObjectMap" verwendet wird.

## Grundlegende Schritte

Verwenden Sie die folgenden Schritte, um das erste Grid zu starten und einen Client für die Interaktion mit dem Grid auszuführen.

1. Öffnen Sie eine Terminalsitzung oder ein Befehlszeilenfenster.
2. Verwenden Sie den folgenden Befehl, um zum Verzeichnis `gettingstarted` zu navigieren:
 

```
cd Installationsstammverzeichnis/ObjectGrid/gettingstarted
```

 Ersetzen Sie `Installationsstammverzeichnis` durch den Pfad des Installationsstammverzeichnisses von eXtreme Scale bzw. durch den Stammpfad des `Installationsstammverzeichnisses` der extrahierten Testversion von eXtreme Scale.
3. Setzen oder exportieren Sie die Umgebungsvariable `JAVA_HOME` so, dass sie auf das Installationsverzeichnis eines gültigen JDK bzw. einer gültigen JRE Version 1.5 oder höher verweist.

```
.. export JAVA_HOME=Java-Ausgangsverzeichnis
.. set JAVA_HOME=Java-Ausgangsverzeichnis
```

4. Führen Sie das folgende Script aus, um einen Katalogserviceprozess auf localhost zu starten:
  - `.. ./runcat.sh`
  - `.. runcat.bat`

Der Katalogserviceprozess wird im aktuellen Terminalfenster ausgeführt.

5. Öffnen Sie eine weitere Terminalsitzung bzw. ein weiteres Befehlszeilenfenster, und führen Sie den folgenden Befehl aus, um eine Containerserverinstanz zu starten:

```
.. ./runcontainer.sh server0
.. runcontainer.bat server0
```

Der Containerserver wird im aktuellen Terminalfenster ausgeführt. Sie können die Schritte 5 und 6 wiederholen, wenn Sie weitere Containerserverinstanzen für die Unterstützung der Replikation starten möchten.

6. Öffnen Sie eine weitere Terminalsitzung bzw. ein weiteres Befehlszeilenfenster, um Clientbefehle auszuführen.
  - Fügen Sie dem Grid Daten hinzu:
 

```
- .. ./runclient.sh i key1 helloWorld
- .. runclient.bat i key1 helloWorld
```
  - Suchen und zeigen Sie den Wert an:
 

```
- .. ./runclient.sh g key1
- .. runclient.bat g key1
```
  - Aktualisieren Sie den Wert:
 

```
- .. ./runclient.sh u key1 goodbyeWorld
- .. runclient.bat u key1 goodbyeWorld
```

- Löschen Sie den Wert:
    - ... ./runclient.sh d key1
    - ... runclient.bat d key1
7. Verwenden Sie <STRG+c>, um den Katalogserviceprozess und die Containerserver in den entsprechenden Fenstern zu stoppen.

## ObjectGrid definieren

Das Beispiel verwendet die Dateien `objectgrid.xml` und `deployment.xml` aus dem Verzeichnis `Installationsstammverzeichnis/ObjectGrid/gettingstarted/xml`, um einen Containerserver zu starten. Die Datei `objectgrid.xml` ist die ObjectGrid-XML-Deskriptordatei und die Datei `deployment.xml` die ObjectGrid-XML-Deskriptordatei für Implementierungsrichtlinien. Beide Dateien zusammen definieren eine verteilte ObjectGrid-Topologie.

### ObjectGrid-XML-Deskriptordatei

Eine ObjectGrid-XML-Deskriptordatei wird verwendet, um die Struktur des ObjectGrids definieren, das von der Anwendung verwendet wird. Sie enthält eine Liste mit BackingMap-Konfigurationen. Diese BackingMaps sind der eigentliche Datenspeicher für zwischengespeicherte Daten. Im Folgenden sehen Sie eine Beispieldatei `objectgrid.xml`. Die ersten Zeilen der Datei enthalten den erforderlichen Header, den jede ObjectGrid-XML-Datei enthalten muss. Diese Beispieldatei definiert das Grid "ObjectGrid" mit den BackingMaps "Map1" und "Map2".

```
<objectGridConfig xmlns:xsi="http://www.w3.org/2001/XMLSchema-instance"
xsi:schemaLocation="http://ibm.com/ws/objectgrid/config ../objectGrid.xsd"
xmlns="http://ibm.com/ws/objectgrid/config">
  <objectGrids>
    <objectGrid name="Grid">
      <backingMap name="Map1" />
      <backingMap name="Map2" />
    </objectGrid>
  </objectGrids>
</objectGridConfig>
```

### XML-Implementierungsrichtliniendeskriptordatei

Eine XML-Deskriptordatei für Implementierungsrichtlinien wird während des Starts an einen ObjectGrid-Containerserver übergeben. Eine Implementierungsrichtlinie muss zusammen mit einer ObjectGrid-XML-Datei verwendet werden und mit der ObjectGrid-XML kompatibel sein, mit der sie verwendet wird. Für jedes `objectgridDeployment`-Element in der Implementierungsrichtlinie muss ein entsprechendes ObjectGrid-Element in der ObjectGrid-XML vorhanden sein. Die `backingMap`-Elemente, die im `objectgridDeployment`-Element definiert werden, müssen mit den `backingMap`-Elementen in der ObjectGrid-XML konsistent sein. Jedes `backingMap`-Element darf nur in einem einzigen `mapSet`-Element referenziert werden.

Die Implementierungsrichtliniendeskriptordatei ist für die Verwendung mit der entsprechenden ObjectGrid-XML-Datei `objectgrid.xml` bestimmt. Im folgenden Beispiel enthalten die ersten Zeilen der Datei `deployment.xml` den erforderlichen Header, den jede XML-Implementierungsrichtliniendatei enthalten muss. Die Datei definiert das Element "objectgridDeployment" für das Grid "ObjectGrid", das in der Datei `objectgrid.xml` definiert ist. Beide im Grid "ObjectGrid" definierten BackingMaps, Map1 und Map2, sind im MapSet "mapSet" enthalten, in dem die Attribute "numberOfPartitions", "minSyncReplicas" und "maxSyncReplicas" konfiguriert sind.

```

<deploymentPolicy xmlns:xsi="http://www.w3.org/2001/XMLSchema-instance"
xsi:schemaLocation="http://ibm.com/ws/objectgrid/deploymentPolicy ../deploymentPolicy.xsd"
xmlns="http://ibm.com/ws/objectgrid/deploymentPolicy">
    <objectgridDeployment objectgridName="Grid">
        <mapSet name="mapSet" numberOfPartitions="13" minSyncReplicas="0" maxSyncReplicas="1" >
            <map ref="Map1"/>
            <map ref="Map2"/>
        </mapSet>
    </objectgridDeployment>
</deploymentPolicy>

```

Mit dem Attribut "numberOfPartitions" des Elements "mapSet" wird die Anzahl der Partitionen für das MapSet angegeben. Es ist ein optionales Attribut und hat standardmäßig den Wert 1. Die Anzahl der Partitionen muss der geplanten Grid-Kapazität angemessen sein.

Mit dem Attribut "minSyncReplicas" des MapSets wird die Mindestanzahl synchroner Replikate für jede Partition im MapSet angegeben. Es ist ein optionales Attribut und hat standardmäßig den Wert 0. Es werden erst dann primäre Shards und Replikat-Shards verteilt, wenn die Domäne die Mindestanzahl synchroner Replikate unterstützen kann. Für die Unterstützung des minSyncReplicas-Werts benötigen Sie einen Container mehr, als der minSyncReplicas-Wert vorgibt. Wenn die Anzahl synchroner Replikate unter den Wert von minSyncReplicas fällt, werden keine Schreiboperationen für diese Partitionen mehr zugelassen.

Mit dem Attribut "maxSyncReplicas" des MapSets wird die maximale Anzahl synchroner Replikate für jede Partition im MapSet angegeben. Es ist ein optionales Attribut und hat standardmäßig den Wert 0. Es werden keine weiteren synchronen Replikate für eine Partition verteilt, wenn eine Domäne diese Anzahl synchroner Replikate für diese bestimmte Partition erreicht. Das Hinzufügen von Containern, die dieses ObjectGrid unterstützen, kann zu einer höheren Anzahl synchroner Replikate führen, wenn der maxSyncReplicas-Wert noch nicht erreicht ist. Das Beispiel setzt maxSyncReplicas auf 1, d. h., die Domäne verteilt maximal ein synchrones Replikat. Wenn Sie mehrere Containerserverinstanzen starten, wird nur ein einziges synchrones Replikat in einer der Containerserverinstanzen verwendet.

## ObjectGrid verwenden

Die Datei Client.java im Verzeichnis *Installationsstammverzeichnis/*ObjectGrid/gettingstarted/src/" ist das Clientprogramm, das demonstriert, wie eine Verbindung zum Katalogserver hergestellt, eine ObjectGrid-Instanz abgerufen und die API "ObjectMap" verwendet wird.

Aus der Perspektive einer Clientanwendung kann die Verwendung von WebSphere eXtreme Scale in die folgenden Schritte unterteilt werden:

1. Durch den Abruf einer ClientClusterContext-Instanz eine Verbindung zum Katalogservice herstellen,
2. Abruf einer ObjectGrid-Clientinstanz,
3. Abruf einer Session-Instanz,
4. Abruf einer ObjectMap-Instanz,
5. Verwendung der ObjectMap-Methoden.

### 1. Durch den Abruf einer ClientClusterContext-Instanz eine Verbindung zum Katalogservice herstellen

Zum Herstellen einer Verbindung zum Katalogserver können Sie die Methode "connect" der API "ObjectGridManager" verwenden. Die Methode "connect", die



von diesem Beispiel verwendet wird, erfordert nur den Katalogserverendpunkt im Format `Hostname:Port`, z. B. `localhost:2809`. Wenn die Verbindungsherstellung zum Katalogserver erfolgreich ist, gibt die Methode `connect` eine `ClientClusterContext`-Instanz zurück. Die `ClientClusterContext`-Instanz ist erforderlich, um das `ObjectGrid` über die API `ObjectGridManager` abzurufen. Das folgende Code-Snippet demonstriert, wie die Verbindung zu einem Katalogserver hergestellt und eine `ClientClusterContext`-Instanz abgerufen wird.

```
ClientClusterContext ccc = ObjectGridManagerFactory.getObjectGridManager().connect("localhost:2809", null, null);
```

## 2. Abruf einer `ObjectGrid`-Instanz

Zum Abrufen einer `ObjectGrid`-Instanz können Sie die Methode `getObjectGrid` der API `ObjectGridManager` verwenden. Die Methode `getObjectGrid` erfordert die `ClientClusterContext`-Instanz und den Namen der `ObjectGrid`-Instanz. Die `ClientClusterContext`-Instanz wird während der Verbindungsherstellung zum Katalogserver abgerufen. Der Name des `ObjectGrids` ist `Grid` und wurde in der Datei `objectgrid.xml` angegeben. Das folgende Code-Snippet demonstriert, wie Sie ein `ObjectGrid` durch Aufruf der Methode `getObjectGrid` der API `ObjectGridManager` abrufen können:

```
ObjectGrid grid = ObjectGridManagerFactory.getObjectGridManager().getObjectGrid(ccc, "Grid");
```

## 3. Abruf einer `Session`-Instanz

Sie können eine `Session`-Instanz über die abgerufene `ObjectGrid`-Instanz abrufen. Eine `Session`-Instanz ist erforderlich, um die `ObjectMap`-Instanz abzurufen und die Transaktionsabgrenzung durchzuführen. Das folgende Code-Snippet demonstriert, wie Sie eine `Session`-Instanz durch Aufruf der Methode `getSession` der API `ObjectGrid` abrufen:

```
Session sess = grid.getSession();
```

## 4. Abruf einer `ObjectMap`-Instanz

Nach dem Abruf einer `Session`-Instanz können Sie über diese eine `ObjectMap`-Instanz abrufen, indem Sie die Methode `getMap` der API `Session` aufrufen. Sie müssen den Namen der `Map` als Parameter an die Methode `getMap` übergeben, um die `ObjectMap`-Instanz abzurufen. Das folgende Code-Snippet demonstriert, wie Sie eine `ObjectMap`-Instanz durch den Aufruf der Methode `getMap` der API `Session` abrufen:

```
ObjectMap map1 = sess.getMap("Map1");
```

## 5. Verwendung der `ObjectMap`-Methoden

Nach dem Abruf einer `ObjectMap`-Instanz können Sie sie über die API `ObjectMap` starten. Beachten Sie, dass die `ObjectMap` eine Transaktions-`Map` ist und eine Transaktionsabgrenzung durch die Verwendung der Methoden `begin` und `commit` der API `Session` erfordert. Wenn es keine explizite Transaktionsabgrenzung gibt, werden `ObjectMap`-Operationen mit Transaktionen mit automatischem Festschreiben ausgeführt.

Das folgende Code-Snippet demonstriert, wie Sie die API `ObjectMap` mit Transaktionen mit automatischem Festschreiben verwenden:

```
map1.insert(key1, value1);
```

Das folgende Code-Snippet demonstriert, wie Sie die API `ObjectMap` mit expliziter Transaktionsabgrenzung verwenden:

```
sess.begin();
map1.insert(key1, value1);
sess.commit();
```

## Weitere Informationen

Dieses Beispiel demonstriert, wie Sie einen Katalogserver und einen Containerserver starten und die API "ObjectMap" in einer eigenständigen Umgebung verwenden. Sie können auch die API "EntityManager" verwenden.

In einer Umgebung mit WebSphere Application Server, in der WebSphere eXtreme Scale installiert oder aktiviert ist, ist das gebräuchlichste Szenario eine Topologie mit Netzanschluss. In einer Topologie mit Netzanschluss befindet sich der Katalogserver im Deployment-Manager-Prozess von WebSphere Application Server, und jede Instanz von WebSphere Application Server enthält automatisch einen eXtreme-Scale-Server. Java-EE-Anwendungen (Java™ Platform, Enterprise Edition) müssen nur die ObjectGrid-XML-Deskriptordatei und die ObjectGrid-XML-Deskriptordatei für Implementierungsrichtlinien im Verzeichnis META-INF jedes Moduls enthalten, und das ObjectGrid ist automatisch verfügbar. Anschließend kann eine Anwendung eine Verbindung zu einem lokal verfügbaren Katalogserver herstellen und eine ObjectGrid-Instanz zur Verwendung abrufen.

---

## Kapitel 2. WebSphere eXtreme Scale programmieren

WebSphere eXtreme Scale stellt mehrere Features bereit, die programmgesteuert mit der Programmiersprache Java über Anwendungsprogrammierschnittstellen (API, Application Programming Interfaces) und Systemprogrammierschnittstellen (SPI, System Programming Interfaces) aufgerufen werden.

### Programmiermodell

Die folgende Abbildung zeigt eine Übersicht über das Programmiermodell von eXtreme Scale.

### APIs von WebSphere eXtreme Scale

Wenn Sie Anwendungsprogrammierschnittstellen (API, Application Programming Interface) von eXtreme Scale verwenden, müssen Sie zwischen transaktionsorientierten und nicht transaktionsorientierten Operationen unterscheiden. Eine transaktionsorientierte Operation ist eine Operation, die innerhalb einer Transaktion durchgeführt wird. In der Abbildung sind ObjectMap, EntityManager, Query und DataGrid transaktionsorientierte APIs, die im Session-Objekt enthalten sind, das ein transaktionsorientierter Container ist. Nicht transaktionsorientierte Operationen haben nichts mit einer Transaktion zu tun, wie z. B. Konfigurationsoperationen.

ObjectGrid, BackingMap und Plug-in-APIs sind nicht transaktionsorientiert. ObjectGrid, BackingMap und andere Konfigurations-APIs werden in die Kategorie der ObjectGrid-Kern-APIs eingeordnet. Plug-ins sind für die Anpassung des Caches bestimmt, um gewünschte Funktionen ausführen zu können, und werden in die Kategorie der Systemprogrammier-APIs eingeordnet. Ein Plug-in in eXtreme Scale ist eine Komponente, die einen bestimmten Typ von Funktion für die Plug-in-Komponenten von eXtreme Scale bereitstellt, zu denen ObjectGrid und BackingMap gehören. Ein Feature stellt eine bestimmte Funktion oder ein bestimmtes Leistungsmerkmal einer eXtreme-Scale-Komponente, einschließlich ObjectGrid, Session, BackingMap usw., dar. Gewöhnlich sind Features über Konfigurations-APIs konfigurierbar. Plug-ins können integriert werden, erfordern aber in manchen Situationen möglicherweise, dass Sie eigene Plug-ins entwickeln.

In der Regel können Sie ObjectGrid und BackingMap für Ihre Anwendungsanforderungen konfigurieren. Wenn die Anwendung spezielle Anforderungen hat, sollten Sie die Verwendung spezieller Plug-ins in Betracht ziehen. WebSphere eXtreme Scale bietet Ihnen integrierte Plug-ins, die Ihre Anforderungen möglicherweise erfüllen. Wenn Sie beispielsweise ein Modell für die Peer-to-Peer-Replikation zwischen zwei lokalen ObjectGrid-Instanzen oder zwei verteilten eXtreme-Scale-Grids benötigen, ist das integrierte JMSObjectGridEventListener-Plug-in verfügbar. Wenn keines der integrierten Plug-ins Ihre Geschäftsprobleme lösen kann, ziehen Sie die Dokumentation zu den Systemprogrammier-APIs zu Rate, um eigene Plug-ins zu schreiben.

ObjectMap ist eine einfache Map-basierte API. Wenn die zwischengespeicherten Objekte einfach sind und keine Beziehungen haben, eignet sich die API "ObjectMap" ideal für Ihre Anwendung. Wenn Objektbeziehungen vorhanden sind, verwenden Sie die API "EntityManager", die graphenähnliche Beziehungen unterstützt.

Die API "Query" ist ein leistungsstarker Mechanismus für das Suchen von Daten im ObjectGrid. Die APIs "Session" und "EntityManager" bieten die traditionellen Abfragefunktionen.

Die API "DataGrid" ist eine leistungsstarke Datenverarbeitungsfunktion in einer verteilten eXtreme-Scale-Umgebung mit viele Maschinen, Replikaten und Partitionen. Anwendungen können Geschäftslogik parallel auf allen Knoten in der verteilten eXtreme-Scale-Umgebung ausführen. Die Anwendung kann die API "DataGrid" über die API "ObjectMap" abrufen.

---

## Kapitel 3. Auf Daten in WebSphere eXtreme Scale zugreifen

Wenn eine Anwendung eine Referenz auf eine ObjectGrid-Instanz hat, können Sie mit Daten in WebSphere eXtreme Scale interagieren. Verwenden Sie eine der createObjectGrid-Methoden in der Anwendungsprogrammierschnittstelle "ObjectGridManager", um eine lokale Instanz zu erstellen, bzw. die Methode "getObjectGrid" für eine Clientinstanz in einem verteilten Grid.

Ein Thread in einer Anwendung benötigt eine eigene Sitzung. Wenn eine Anwendung das ObjectGrid in einem Thread verwenden möchten, muss sie lediglich eine der getSession-Methoden aufrufen, um einen Thread anzufordern. Diese Operation ist kostengünstig, weil die Operationen in den meisten Fällen nicht in einen Pool gestellt werden müssen. Wenn die Anwendung ein Framework für Abhängigkeitsinjektion wie Spring verwendet, können Sie bei Bedarf eine Sitzung in eine Anwendungs-Bean injizieren.

Nach dem Erhalt der Sitzung kann die Anwendung auf die Daten zugreifen, die in Maps im ObjectGrid gespeichert sind. Wenn das ObjectGrid Entitäten verwendet, können Sie die Anwendungsprogrammierschnittstelle "EntityManager" verwenden, die Sie mit der Methode "Session.getEntityManager" abrufen können. Da sie näher an die Java-Spezifikationen angelehnt ist, ist die Schnittstelle "EntityManager" einfacher zu verwenden als die Map-basierte Anwendungsprogrammierschnittstelle. Die Anwendungsprogrammierschnittstelle "EntityManager" bringt jedoch Leistungseinbußen mit sich, weil sie Änderungen in Objekten verfolgt. Die Map-basierte Anwendungsprogrammierschnittstelle wird über die Methode "Session.getMap" abgerufen.

WebSphere eXtreme Scale verwendet Transaktionen. Wenn eine Anwendung mit einer Sitzung interagiert, muss sie im Kontext einer Transaktion enthalten sein. Eine Transaktion wird über die Methoden "Session.begin", "Session.commit" und "Session.rollback" im Session-Objekt gestartet und festgeschrieben bzw. rückgängig gemacht. Anwendungen können auch im Modus für automatische Festschreibung ausgeführt werden. In diesem Modus startet und schreibt die Sitzung eine Transaktion automatisch fest, wenn die Anwendung mit Maps interagiert. Allerdings ist der Modus für automatische Festschreibung langsamer.

### Logik für die Verwendung von Transaktionen

Transaktionen scheinen langsam zu sein, aber eXtreme Scale verwendet Transaktionen aus drei Gründen:

1. Änderungen können rückgängig gemacht werden, wenn eine Ausnahme eintritt oder wenn die Geschäftslogik Statusänderungen widerrufen muss.
2. Es können Datensperren für die Dauer einer Transaktion gesetzt und freigegeben werden, was eine atomare Durchführung von Änderungen ermöglicht, d. h., es werden entweder alle Änderungen oder gar keine Änderungen an den Daten vorgenommen.
3. Es kann eine atomare Replikationseinheit erzeugt werden.

Mit WebSphere eXtreme Scale kann eine Sitzung dynamisch festlegen, wie viel Transaktion tatsächlich erforderlich ist. Eine Anwendung kann die Rollback-Unterstützung und Sperren inaktivieren, aber dies geht zu Lasten der Anwendung. Die Anwendung muss das Fehlen dieser Features ausgleichen.

Beispiel: Eine Anwendung kann Sperren inaktivieren, indem Sie in der `BackingMap` die Einstellung für die Sperrstrategie auf `NONE` (Keine) setzt. Diese Strategie ist zwar schnell, ermöglicht aber, dass verschiedene Transaktionen dieselben Daten ungeschützt voreinander gleichzeitig ändern. Die Anwendung ist für alle Sperren und für die Datenkonsistenz zuständig, wenn `NONE` als Einstellung für die Sperrstrategie verwendet wird.

Eine Anwendung kann auch die Art und Weise ändern, in der Objekte beim Zugriff über die Transaktion kopiert werden. Die Anwendung kann festlegen, wie Objekte mit der Methode `ObjectMap.setCopyMode` kopiert werden. Mit dieser Methode können Sie die Einstellung `CopyMode` inaktivieren. Die Einstellung `CopyMode` wird normalerweise für Transaktionen inaktiviert, die im Lesezugriff arbeiten, wenn innerhalb einer Transaktion verschiedene Werte für dasselbe Objekt zurückgegeben werden können. Innerhalb einer Transaktion können verschiedene Werte für dasselbe Objekt zurückgegeben werden.

Wenn die Transaktion beispielsweise die Methode `ObjectMap.get` für das Objekt zum Zeitpunkt T1 aufruft, empfängt sie den Wert, der zu diesem Zeitpunkt gültig ist. Wenn sie die Methode `get` später zum Zeitpunkt T2 erneut aufruft, kann ein anderer Thread den Wert in der Zwischenzeit geändert haben. Da der Wert von einem anderen Thread geändert wurde, sieht die Anwendung einen anderen Wert. Wenn die Anwendung ein Objekt ändert, das mit dem `CopyMode`-Wert `NONE` abgerufen wurde, ändert sie die festgeschriebene Kopie dieses Objekts direkt. Ein Rollback der Transaktion hat in diesem Modus keine Bedeutung. Sie ändern die einzige Kopie im `ObjectGrid`. Obwohl die `CopyMode`-Einstellung `NONE` schnell ist, müssen Sie sich über die Konsequenzen, die diese Einstellung hat, genau im Klaren sein. Eine Anwendung, die die `CopyMode`-Einstellung `NONE` verwendet, darf die Transaktion nicht rückgängig machen. Wenn die Anwendung die Transaktion rückgängig macht, werden die Indizes nicht mit den Änderungen aktualisiert, *und* die Änderungen werden nicht repliziert, wenn die Replikation aktiviert ist. Die Standardwerte sind problemlos zu verwenden und weniger fehleranfällig. Wenn Sie die Leistung der Zuverlässigkeit von Daten vorziehen, muss die Anwendung wissen, was sie tut, um unbeabsichtigte Probleme zu vermeiden.

#### **Vorsicht:**

**Gehen Sie vorsichtig vor, wenn Sie die Sperrstrategie oder den `CopyMode`-Wert ändern. Wenn Sie diese Werte ändern, ist das Anwendungsverhalten unvorhersehbar.**

## **Interaktion mit gespeicherten Daten**

Nach dem Erhalt einer Sitzung können Sie das folgende Codefragment verwenden, um die Anwendungsprogrammierschnittstelle `Map` für das Einfügen von Daten zu verwenden.

```
Session session = ...;
ObjectMap personMap = session.getMap("PERSON");
session.begin();
Person p = new Person();
p.name = "John Doe";
personMap.insert(p.name, p);
session.commit();
```

Im Folgenden sehen Sie dasselbe Beispiel mit der Anwendungsprogrammierschnittstelle `EntityManager`. In diesem Codemuster wird davon ausgegangen, dass das `Person`-Objekt einer Entität zugeordnet ist.

```

Session session = ...;
EntityManager em = session.getEntityManager();
session.begin();
Person p = new Person();
p.name = "John Doe";
em.persist(p);
session.commit();

```

Das Muster ist so konzipiert, dass es Referenzen auf die ObjectMaps für die Maps abrufen, mit denen der Thread arbeitet, eine Transaktion startet, die Daten bearbeitet und dann die Transaktion festschreibt.

Die Schnittstelle "ObjectMap" enthält die typischen Map-Operationen, wie z. B. put, get und remove. Verwenden Sie jedoch spezifischere Operationsnamen, wie z. B. get, getForUpdate, insert, update und remove. Diese Methodennamen vermitteln die Absicht deutlicher als die traditionellen Map-APIs.

Sie können auch die flexible Indexierungsunterstützung verwenden.

Im Folgenden sehen Sie ein Beispiel für die Aktualisierung eines Objekts:

```

session.begin();
Person p = (Person)personMap.getForUpdate("John Doe");
p.name = "John Doe";
p.age = 30;
personMap.update(p.name, p);
session.commit();

```

Die Anwendung verwendet normalerweise die Methode "getForUpdate" an Stelle einer einfachen get-Methode, um den Datensatz zu sperren. Die update-Methode muss aufgerufen werden, um den aktualisierten Wert in der Map bereitzustellen. Wenn die update-Methode nicht aufgerufen wird, bleibt die Map unverändert. Im Folgenden sehen Sie dasselbe Fragment mit der Anwendungsprogrammierschnittstelle "EntityManager":

```

session.begin();
Person p = (Person)em.findForUpdate(Person.class, "John Doe");
p.age = 30;
session.commit();

```

Die Anwendungsprogrammierschnittstelle "EntityManager" ist einfacher als der Ansatz mit Map. In diesem Fall sucht eXtreme Scale die Entität und gibt ein verwaltetes Objekt an die Anwendung zurück. Die Anwendung ändert das Objekt und schreibt die Transaktion fest, und eXtreme Scale verfolgt die Änderungen an verwalteten Objekten automatisch während der Festschreibung und nimmt die erforderlichen Aktualisierungen vor.

## Transaktionen und Partitionen

Transaktionen von WebSphere eXtreme Scale können nur eine einzige Partition aktualisieren. Transaktionen eines Clients können in mehreren Partitionen lesen, aber nur eine einzige Partition aktualisieren. Wenn eine Anwendung versucht, zwei Partitionen zu aktualisieren, scheitert die Transaktion und wird rückgängig gemacht. Eine Transaktion, die ein integriertes ObjectGrid (Grid-Logik) verwendet, hat keine Routing-Funktionen und kann nur die Daten in der lokalen Partition sehen. Diese Geschäftslogik kann immer eine zweite Sitzung abrufen, die eine reine Clientsitzung ist, um auf andere Partitionen zuzugreifen. Diese Transaktion ist jedoch eine unabhängige Transaktion.

## Abfragen und Partitionen

Wenn eine Transaktion bereits nach einer Entität gesucht hat, wird die Transaktion der Partition für diese Entität zugeordnet. Alle Abfragen, die in einer Transaktion ausgeführt werden, die einer Entität zugeordnet ist, werden an die zugeordnete Partition weitergeleitet.

Wenn eine Abfrage in einer Transaktion ausgeführt wird, bevor sie einer Partition zugeordnet wurde, müssen Sie die für die Abfrage zu verwendende Partitions-ID angeben. Die Partitions-ID ist ein ganzzahliger Wert. Die Abfrage wird dann an diese Partition weitergeleitet.

Abfragen suchen nur innerhalb einer einzigen Partition. Über die DataGrid-Anwendungsprogrammierschnittstellen können Sie dieselbe Abfrage jedoch parallel in allen Partitionen oder einem Teil der Partitionen ausführen. Verwenden Sie die DataGrid-Anwendungsprogrammierschnittstellen, um einen Eintrag zu suchen, der sich in jeder der Partitionen befinden könnte.

---

## Interaktion mit einem ObjectGrid über ObjectGridManager

Die Klasse "ObjectGridManagerFactory" und die Schnittstelle "ObjectGridManager" stellen einen Mechanismus für das Erstellen, Zugreifen auf und Zwischenspeichern von ObjectGrid-Instanzen bereit. Die Klasse "ObjectGridManagerFactory" ist eine statische Helper-Klasse für den Zugriff auf die Schnittstelle "ObjectGridManager", ein Singleton. Die Schnittstelle "ObjectGridManager" enthält mehrere Methoden zur Vereinfachung der Erstellung von Instanzen eines ObjectGrid-Objekts. Außerdem vereinfacht die Schnittstelle "ObjectGridManager" die Erstellung und Zwischenspeicherung von ObjectGrid-Instanzen, auf die von mehreren Benutzern zugegriffen wird.

### Programmiermodell

Vor der Verwendung der Funktionalität von eXtreme Scale als speicherinternes Daten-Grid müssen Sie mit Methoden wie den folgenden ObjectGrid-Instanzen erstellen und mit diesen interagieren:

- createObjectGrid-Methoden
- getObjectGrid-Methoden
- removeObjectGrid-Methoden
- Lebenszyklus eines ObjectGrids steuern

### createObjectGrid-Methoden

In diesem Abschnitt werden die sieben createObjectGrid-Methoden in der Schnittstelle "ObjectGridManager" beschrieben. Jede dieser Methoden erstellt eine lokale Instanz eines ObjectGrids.

### Lokale speicherinterne Instanz

Das folgende Code-Snippet veranschaulicht, wie eine lokale ObjectGrid-Instanz mit eXtreme Scale erstellt und konfiguriert wird.

```
// Lokale ObjectGrid-Referenz anfordern
// Sie können ein neues ObjectGrid erstellen oder ein konfiguriertes ObjectGrid abrufen,
// das in der ObjectGrid-XML-Datei definiert ist.
ObjectGridManager objectGridManager =
    ObjectGridManagerFactory.getObjectGridManager();
ObjectGrid ivObjectGrid =
```



```

objectGridManager.createObjectGrid("objectgridName");

// Dem ObjectGrid einen TransactionCallback hinzufügen.
HeapTransactionCallback tcb = new HeapTransactionCallback();
ivObjectGrid.setTransactionCallback(tcb);

// BackingMap definieren.
// Wenn die BackingMap in der ObjectGrid-XML-Datei
// konfiguriert ist, können Sie sie einfach abrufen.
BackingMap ivBackingMap = ivObjectGrid.defineMap("myMap");

// Der BackingMap einen Loader hinzufügen.
Loader ivLoader = new HeapCacheLoader();
ivBackingMap.setLoader(ivLoader);

// ObjectGrid initialisieren.
ivObjectGrid.initialize();

// Sitzung für den aktuellen Thread anfordern.
// Die Sitzung kann nicht von mehreren Threads gemeinsam genutzt werden.
Session ivSession = ivObjectGrid.getSession();

// ObjectMap vom ObjectGrid-Session-Objekt anfordern.
ObjectMap objectMap = ivSession.getMap("myMap");

```

## Gemeinsam genutzte Standardkonfiguration

Der folgende Code zeigt einen einfachen Fall für die Erstellung eines ObjectGrids, das von vielen Benutzern gemeinsam verwendet werden kann.

```

import com.ibm.websphere.objectgrid.ObjectGrid;
import com.ibm.websphere.objectgrid.ObjectGridException;
import com.ibm.websphere.objectgrid.ObjectGridManagerFactory;
import com.ibm.websphere.objectgrid.ObjectGridManager;
final ObjectGridManager oGridManager=
    ObjectGridManagerFactory.getObjectGridManager();
ObjectGrid employees =
    oGridManager.createObjectGrid("Employees",true);
employees.initialize();
employees.
/*sample continues..*/

```

Das vorherige Java-Code-Snippet erstellt und das ObjectGrid "Employees" und stellt es in den Cache. Das ObjectGrid "Employees" wird mit der Standardkonfiguration initialisiert und kann dann verwendet werden. Der zweite Parameter in der Methode "createObjectGrid" wird auf "true" gesetzt. Damit wird ObjectGrid-Manager angewiesen, die erstellte ObjectGrid-Instanz zwischenspeichern. Wenn dieser Parameter auf "false" gesetzt wird, wird die s Instanz nicht zwischengespeichert. Jede ObjectGrid-Instanz hat einen Namen, und die Instanz kann von vielen Clients oder Benutzern auf der Basis dieses Namens gemeinsam genutzt werden.

Wenn die ObjectGrid-Instanz im Peer-to-Peer-Modus gemeinsam genutzt wird, muss das Caching auf "true" gesetzt werden. Weitere Informationen zur gemeinsamen Nutzung im Peer-to-Peer-Modus finden Sie im Abschnitt zur Verteilung von Änderungen an Peer-JVMs.

## XML-Konfiguration

WebSphere eXtreme Scale ist hoch konfigurierbar. Das vorherige Beispiel veranschaulicht, wie ein einfaches ObjectGrid ohne Konfiguration erstellt wird. Dieses Beispiel zeigt, wie Sie eine vorkonfigurierte ObjectGrid-Instanz erstellen, die auf

einer XML-Konfigurationsdatei basiert. Sie können eine ObjectGrid-Instanz programmgesteuert oder durch die Verwendung einer XML-basierten Konfigurationsdatei konfigurieren. Sie können ObjectGrid auch über eine Kombination beider Ansätze konfigurieren. Die Schnittstelle "ObjectGridManager" lässt die Erstellung einer ObjectGrid-Instanz auf der Basis der XML-Konfiguration zu. Die Schnittstelle "ObjectGridManager" hat mehrere Methoden, die einen URL als Argument akzeptieren. Jede XML-Datei, die an die Schnittstelle "ObjectGridManager" übergeben wird, muss anhand des Schemas validiert werden. Die XML-Validierung kann nur inaktiviert werden, wenn die Datei zuvor validiert wurde und seit der letzten Validierung keine Änderungen mehr an der Datei vorgenommen wurden. Durch die Inaktivierung kann ein geringer Teil der Kosten eingespart werden, bringt aber das Risiko der Verwendung einer ungültigen XML-Datei mit sich. IBM® Java Developer Kit (JDK) 1.4.2 unterstützt die XML-Validierung. Wenn Sie ein JDK verwenden, das diese Unterstützung nicht besitzt, kann Apache Xerces für die Validierung der XML erforderlich sein.

Das folgende Java-Code-Snippet veranschaulicht, wie eine XML-Konfigurationsdatei zum Erstellen eines ObjectGrids übergeben wird.

```
import java.net.MalformedURLException;
import java.net.URL;
import com.ibm.websphere.objectgrid.ObjectGrid;
import com.ibm.websphere.objectgrid.ObjectGridException;
import com.ibm.websphere.objectgrid.ObjectGridManager;
import com.ibm.websphere.objectgrid.ObjectGridManagerFactory;
boolean validateXML = true; // XML-Validierung aktivieren
boolean cacheInstance = true; // Instanz zwischenspeichern
String objectGridName="Employees"; // Name des ObjectGrid-URL
allObjectGrids = new URL("file:test/myObjectGrid.xml");
final ObjectGridManager oGridManager=
    ObjectGridManagerFactory.getObjectGridManager();
ObjectGrid employees =
    oGridManager.createObjectGrid(objectGridName, allObjectGrids,
        bvalidateXML, cacheInstance);
```

Die XML-Datei kann Konfigurationsdaten für mehrere ObjectGrids enthalten. Das vorherige Code-Snippet gibt das ObjectGrid "Employees" zurück, vorausgesetzt, dass die Employees-Konfiguration in der Datei definiert ist. Informationen zur XML-Syntax finden Sie in der Beschreibung der ObjectGrid-Konfiguration. Es sind sieben createObjectGrid-Methoden vorhanden, die im folgenden Codeblock dokumentiert sind.

```
/**
 * Eine einfache Factory-Methode, mit der eine Instanz eines
 * ObjectGrids zurückgegeben werden kann. Es wird ein eindeutiger Name zugeordnet.
 * Die ObjectGrid-Instanz wird nicht zwischengespeichert.
 * Benutzer können anschließend {@link ObjectGrid#setName(String)} verwenden,
 * um den ObjectGrid-Namen zu ändern.
 *
 * @return ObjectGrid Eine ObjectGrid-Instanz mit einem eindeutigen zugeordneten Namen.
 * @throws ObjectGridException bei allen Fehlern, die während der Erstellung
 * des ObjectGrids auftreten.
 */
public ObjectGrid createObjectGrid() throws ObjectGridException;

/**
 * Eine einfache Factory-Methode, mit der eine ObjectGrid-Instanz des angegebenen
 * Namens zurückgegeben werden kann. Die ObjectGrid-Instanzen können zwischengespeichert
 * werden. Wenn bereits eine ObjectGrid-Instanz mit diesem Namen zwischengespeichert
 * ist, wird eine Ausnahme des Typs "ObjectGridException" ausgelöst.
 *
 * @param objectGridName Der Name der zu erstellenden ObjectGrid-Instanz.
 * @param cacheInstance true, wenn die ObjectGrid-Instanz zwischengespeichert werden soll.
```

```

* @return Eine ObjectGrid-Instanz.
* @throws ObjectGridException, wenn dieser Name bereits zwischengespeichert wurde oder ein
* Fehler während der ObjectGrid-Erstellung aufgetreten ist.
*/
public ObjectGrid createObjectGrid(String objectGridName, boolean cacheInstance)
    throws ObjectGridException;

/**
 * ObjectGrid-Instanz mit dem angegebenen ObjectGrid-Namen erstellen. Die
 * erstellte ObjectGrid-Instanz wird zwischengespeichert.
 * @param objectGridName Der Name der zu erstellenden ObjectGrid-Instanz.
 * @return Eine ObjectGrid-Instanz.
 * @throws ObjectGridException, wenn bereits ein ObjectGrid mit diesem Namen
 * zwischengespeichert wurde oder ein Fehler während der Erstellung des ObjectGrids
 * aufgetreten ist.
 */
public ObjectGrid createObjectGrid(String objectGridName)
    throws ObjectGridException;

/**
 * ObjectGrid-Instanz auf der Basis des angegebenen ObjectGrid-Namens und der
 * XML-Datei erstellen. Die in der XML-Datei definierte ObjectGrid-Instanz wird
 * mit dem angegebenen ObjectGrid-Namen erstellt und zurückgegeben. Wird ein
 * solches ObjectGrid nicht in der XML-Datei gefunden, wird eine Ausnahme ausgelöst.
 *
 * Diese ObjectGrid-Instanz kann zwischengespeichert werden.
 *
 * Wenn der URL null ist, wird er einfach ignoriert. In diesem Fall, verhält sich
 * diese Methode genauso wie {@link #createObjectGrid(String, boolean)}.
 *
 * @param objectGridName Der Name der zurückzugebenden ObjectGrid-Instanz. Er
 * darf nicht null sein.
 * @param xmlFile Der URL einer gemäß ObjectGrid-Schema korrekt formatierten XML-Datei.
 * @param enableXmlValidation Wenn true, wird die XML validiert.
 * @param cacheInstance Ein boolescher Wert, der anzeigt, ob die ObjectGrid-Instanzen,
 * die in der XML definiert sind, zwischengespeichert werden oder nicht. Wenn true,
 * werden die Instanzen zwischengespeichert.
 *
 * @throws ObjectGridException, wenn bereits ein ObjectGrid mit demselben
 * Namen zwischengespeichert wurde, kein ObjectGrid-Name in der XML-Datei gefunden
 * wird oder ein anderer Fehler während der Erstellung des ObjectGrids auftritt.
 * @return Eine ObjectGrid-Instanz.
 * @see ObjectGrid
 */
public ObjectGrid createObjectGrid(String objectGridName, final URL xmlFile,
    final boolean enableXmlValidation, boolean cacheInstance)
    throws ObjectGridException;

/**
 * XML-Datei verarbeiten und eine Liste mit ObjectGrid-Objekten auf der Basis
 * dieser Datei erstellen.
 * Diese ObjectGrid-Instanzen können zwischengespeichert werden.
 * Es wird eine Ausnahme des Typs "ObjectGridException" ausgelöst, wenn versucht
 * wird, eine neu erstellte ObjectGrid-Instanz zwischenzuspeichern, die
 * denselben Namen hat wie eine bereits zwischengespeicherte ObjectGrid-Instanz.
 *
 * @param xmlFile Die Datei, die ein ObjectGrid oder mehrere
 * ObjectGrids definiert.
 * @param enableXmlValidation Bei true wird die XML anhand des
 * Schemas validiert.
 * @param cacheInstances Bei true werden alle auf der Basis dieser Datei
 * erstellen ObjectGrid-Instanzen zwischengespeichert.
 * @return Eine ObjectGrid-Instanz.
 * @throws ObjectGridException, wenn versucht wird, eine ObjectGrid-Instanz
 * zu erstellen und zwischenzuspeichern, die denselben Namen wie eine bereits
 * zwischengespeicherte ObjectGrid-Instanz hat, oder wenn während der
 * Erstellung der ObjectGrid-Instanz ein anderer Fehler auftritt.

```

```

*/
public List createObjectGrids(final URL xmlFile, final boolean enableXmlValidation,
boolean cacheInstances) throws ObjectGridException;

/** Alle ObjectGrids erstellen, die in der XML-Datei enthalten sind. Die
 * XML-Datei wird anhand des Schemas validiert. Alle erstellten ObjectGrid-Instanzen
 * werden zwischengespeichert. Es wird eine Ausnahme des Typs "ObjectGridException"
 * ausgelöst, wenn versucht wird, eine neu erstellte ObjectGrid-Instanz zwischenzuspeichern,
 * die denselben Namen hat wie eine bereits zwischengespeicherte ObjectGrid-Instanz.
 * @param xmlFile Die zu verarbeitende XML-Datei. ObjectGrids werden auf der Basis
 * der Angaben in der Datei erstellt.
 * @return Eine Liste mit ObjectGrid-Instanzen, die erstellt wurden.
 * @throws ObjectGridException, wenn bereits eine ObjectGrid-Instanz zwischengespeichert
 * wurde, die einen der in der XML-Datei definierten Namen hat, oder wenn während
 * der Erstellung der ObjectGrid-Instanz ein Fehler auftritt.
*/
public List createObjectGrids(final URL xmlFile) throws ObjectGridException;

/**
 * XML-Datei verarbeiten und nur dann eine einzige ObjectGrid-Instanz mit dem
 * angegebenen ObjectGrid-Namen erstellen, wenn ein ObjectGrid mit diesem Namen
 * in der Datei enthalten ist. Wenn kein ObjectGrid mit diesem Namen in der
 * XML-Datei vorhanden ist, wird eine Ausnahme des Typs "ObjectGridException"
 * ausgelöst. Die erstellte ObjectGrid-Instanz wird zwischengespeichert.
 * @param objectGridName Name des erstellenden ObjectGrids. Dieses ObjectGrid
 * muss in der XML-Datei definiert sein.
 * @param xmlFile Die zu verarbeitende XML-Datei.
 * @return Ein neu erstelltes ObjectGrid.
 * @throws ObjectGridException, wenn bereits ein ObjectGrid mit demselben
 * Namen zwischengespeichert wurde, kein ObjectGrid-Name in der XML-Datei gefunden
 * wird oder ein anderer Fehler während der Erstellung des ObjectGrids auftritt.
*/
public ObjectGrid createObjectGrid(String objectGridName, URL xmlFile)
throws ObjectGridException;

```

## Client blockiert beim Aufruf der Methode "getObjectGrid"

Ein Client kann blockieren, wenn die Methode "getObjectGrid" in ObjectGridManager aufgerufen wird oder eine Ausnahme des Typs "com.ibm.websphere.projector.MetadataException" auslösen. Das EntityMetadata-Repository ist nicht verfügbar und der Zeitlimitschwellenwert ist erreicht. Der Grund hierfür ist, dass der Client auf die Verfügbarkeit der Entitätsmetadaten im ObjectGrid-Server wartet. Dieser Fehler kann auftreten, wenn ein Container gestartet wurde, aber die anfängliche Anzahl der Container oder die Mindestanzahl synchroner Replikate nicht erreicht ist. Untersuchen Sie die Implementierungsrichtlinie für das ObjectGrid, und stellen Sie sicher, dass die Anzahl aktiver Container größer-gleich der Werte der Attribute "numInitialContainers" und "minSyncReplicas" in der Deskriptordatei der Implementierungsrichtlinie ist.

## getObjectGrid-Methoden

Verwenden Sie die ObjectGridManager.getObjectGrid-Methoden, um zwischengespeicherte Instanzen abzurufen.

### Zwischengespeicherte Instanz abrufen

Da die ObjectGrid-Instanz "Employees" über die Schnittstelle "ObjectGridManager" zwischengespeichert wurde, kann ein anderer Benutzer mit dem folgenden Code-Snippet auf sie zugreifen:

```
ObjectGrid myEmployees = oGridManager.getObjectGrid("Employees");
```

Im Folgenden sehen Sie die beiden getObjectGrid-Methoden, die zwischengespeicherte ObjectGrid-Instanzen zurückgeben:

- **Alle zwischengespeicherten Instanzen abrufen**

Zum Abrufen aller ObjectGrid-Instanzen, die zuvor zwischengespeichert wurden, verwenden Sie die Methode "getObjectGrids", die eine Liste mit allen Instanzen zurückgibt. Wenn keine zwischengespeicherten Instanzen vorhanden sind, gibt die Methode "null" zurück.

- **Zwischengespeicherte Instanzen nach Namen abrufen**

Wenn Sie eine einzelne zwischengespeicherte Instanz eines ObjectGrids abrufen möchten, verwenden Sie getObjectGrid(String objectGridName), und übergeben Sie den Namen der zwischengespeicherten Instanz an die Methode. Die Methode gibt entweder die ObjectGrid-Instanz mit dem angegebenen Namen oder "null" zurück, falls keine ObjectGrid-Instanz mit diesem Namen vorhanden ist.

**Anmerkung:** Sie können die Methode "getObjectGrid" verwenden, um eine Verbindung zu einem verteilten Grid herzustellen. Weitere Informationen finden Sie im Abschnitt „Verbindung zu einem verteilten ObjectGrid herstellen“ auf Seite 18.

## removeObjectGrid-Methoden

Sie können zwei unterschiedliche removeObjectGrid-Methoden verwenden, um ObjectGrid-Instanzen aus dem Cache zu entfernen.

### ObjectGrid-Instanz entfernen

Verwenden Sie zum Entfernen von ObjectGrid-Instanzen aus dem Cache eine der removeObjectGrid-Methoden. Der ObjectGridManager behält keine Referenz auf die entfernten Instanzen. Es sind zwei Methoden zum Entfernen einer Instanz vorhanden. Eine Methode akzeptiert einen booleschen Parameter. Wenn der boolesche Parameter auf true gesetzt ist, wird die Methode "destroy" für das ObjectGrid aufgerufen. Der Aufruf der Methode "destroy" für das ObjectGrid beendet das ObjectGrid und gibt die vom ObjectGrid verwendeten Ressourcen frei. Im Folgenden wird die Verwendung der beiden removeObjectGrid-Methoden beschrieben:

```
/**
 * ObjectGrid aus dem Cache der ObjectGrid-Instanzen entfernen
 *
 * @param objectGridName Name der ObjectGrid-Instanz, die aus dem Cache
 * entfernt werden soll
 *
 * @throws ObjectGridException, wenn ein ObjectGrid mit dem angegebenen objectGridName-Wert
 * nicht im Cache gefunden wird
 */
public void removeObjectGrid(String objectGridName) throws ObjectGridException;

/**
 * ObjectGrid aus dem Cache der ObjectGrid-Instanzen entfernen und
 * die zugehörigen Ressourcen löschen
 *
 * @param objectGridName Name der ObjectGrid-Instanz, die aus dem Cache
 * entfernt werden soll
 *
 * @param destroy ObjectGrid-Instanz und die zugehörigen Ressourcen
 * löschen
 *
 * @throws ObjectGridException, wenn ein ObjectGrid mit dem angegebenen objectGridName-Wert
```

```

    * nicht im Cache gefunden wird
    */
    public void removeObjectGrid(String objectGridName, boolean destroy)
        throws ObjectGridException;

```

## Verbindung zu einem verteilten ObjectGrid herstellen

Sie können eine Verbindung zu einem verteilten ObjectGrid über einen Verbindungsendpunkt des Katalogservice herstellen. Sie müssen den Hostnamen und den Endpunktport des Katalogservice kennen, zu dem Sie die Verbindung herstellen möchten.

Um eine Verbindung zu einem verteilten Grid herzustellen, müssen Sie Ihre serverseitige Umgebung mit einem Katalogservice und Containerservern konfiguriert haben.

Die Methode "getObjectGrid(ClientClusterContext ccc, String objectGridName)" stellt die Verbindung zum angegebenen Katalogservice her und gibt eine ObjectGrid-Clientinstanz zurück, die einer serverseitigen ObjectGrid-Instanz entspricht.

Das folgende Code-Snippet ist ein Beispiel für die Verbindungsherstellung zu einem verteilten Grid.

```

    // ObjectGridManager-Instanz erstellen.

    ObjectGridManager ogm = ObjectGridManagerFactory.getObjectGridManager();

    // ClientClusterContext durch Verbindungsherstellung zu
    // einem katalogserverbasierten verteilten ObjectGrid anfordern.
    // Sie müssen einen Verbindungsendpunkt für den Katalogserver
    // im Format Hostname:Endpunktport angeben. Der Hostname steht
    // für die Maschine, auf der sich der Katalogserver befindet,
    // und der Endpunktport ist der Empfangsport des Katalogservers,
    // der standardmäßig 2809 ist.
    ClientClusterContext ccc = ogm.connect("localhost:2809", null, null);

    // Verteiltes ObjectGrid über ObjectGridManager abrufen und
    // das ClientClusterContext-Objekt angeben.
    ObjectGrid og = ogm.getObjectGrid(ccc, "objectgridName");

```

## eXtreme-Scale-Client konfigurieren

Sie können einen eXtreme-Scale-Client auf der Basis Ihrer Anforderungen, einschließlich Korrektoreinstellungen, konfigurieren.

Sie können einen eXtreme-Scale-Client auf die folgenden Arten konfigurieren:

- XML-Konfiguration
- Programmgesteuerte Konfiguration
- Konfiguration des Spring-Frameworks
- Nahen Cache inaktivieren

Sie können die folgenden Plug-ins in einem Client überschreiben:

- **ObjectGrid-Plug-ins**
  - TransactionCallback-Plug-in
  - ObjectGridEventListener-Plug-in
- **BackingMap-Plug-ins**
  - Evictor-Plug-in
  - MapEventListener-Plug-in
  - Attribut "numberOfBuckets"

- Attribut "ttlEvictorType"
- Attribut "timeToLive"

## Konfiguration des Clients mit XML

Zum Ändern von Einstellungen auf der Clientseite kann eine ObjectGrid-XML-Datei verwendet werden. Wenn Sie die Einstellungen in einem eXtreme-Scale-Client müssen Sie eine ObjectGrid-XML-Datei erstellen, die in ihrer Struktur der Datei gleich, die für den eXtreme-Scale-Server verwendet wurde.

Angenommen, die folgende XML-Datei wurde mit einer XML-Implementierungsrichtliniendatei kombiniert und zum Starten eines eXtreme-Scale-Servers verwendet.

**companyGridServerSide.xml**

```
<?xml version="1.0" encoding="UTF-8"?>
<objectGridConfig xmlns:xsi="http://www.w3.org/2001/XMLSchema-instance"
  xsi:schemaLocation="http://ibm.com/ws/objectgrid/config ../objectGrid.xsd"
  xmlns="http://ibm.com/ws/objectgrid/config">

  <objectGrids>
    <objectGrid name="CompanyGrid">
      <bean id="TransactionCallback"
        className="com.company.MyTxCallback" />
      <bean id="ObjectGridEventListener"
        className="com.company.MyOgEventListener" />
      <backingMap name="Customer"
        pluginCollectionRef="customerPlugins" />
      <backingMap name="Item" />
      <backingMap name="OrderLine" numberOfBuckets="1049"
        timeToLive="1600" ttlEvictorType="LAST_ACCESS_TIME" />
      <backingMap name="Order" lockStrategy="PESSIMISTIC"
        pluginCollectionRef="orderPlugins" />
    </objectGrid>
  </objectGrids>

  <backingMapPluginCollections>
    <backingMapPluginCollection id="customerPlugins">
      <bean id="Evictor"
        className="com.ibm.websphere.objectgrid.plugins.builtins.LRUEvictor" />
      <bean id="MapEventListener"
        className="com.company.MyMapEventListener" />
    </backingMapPluginCollection>
    <backingMapPluginCollection id="orderPlugins">
      <bean id="MapIndexPlugin"
        className="com.company.MyMapIndexPlugin" />
    </backingMapPluginCollection>
  </backingMapPluginCollections>
</objectGridConfig>
```

In einem eXtreme-Scale-Server verhält sich das CompanyGrid gemäß der Definition in der Datei companyGridServerSide.xml. Standardmäßig hat der CompanyGrid-Client dieselben Einstellungen wie das im Server ausgeführte CompanyGrid. Einige dieser Einstellungen können jedoch im Client überschrieben werden.

Erstellen Sie ein clientspezifisches ObjectGrid, um einige dieser Einstellungen zu überschreiben. Kopieren Sie die ObjectGrid-XML-Datei, die zum Öffnen des Servers verwendet wurde, und passen Sie die Kopie für die Clientseite an. Zum Entfernen eines Plug-ins aus dem Client verwenden Sie eine leere Zeichenfolge als Wert für das Attribut "className". Wenn Sie ein vorhandenes Plug-in ändern möchten, geben Sie einen neuen Wert für das Attribut "className" an. Sie können der Datei auch ein Plug-in hinzufügen, falls es sich um eines der Plug-ins handelt, die überschrieben werden können.

Wenn Sie eines der Attribute im Client setzen möchten, geben Sie einen neuen Wert an.

Die folgende ObjectGrid-XML-Datei kann verwendet werden, um einige Attribute und Plug-ins im CompanyGrid-Client anzugeben:

```

companyGridClientSide.xml
<?xml version="1.0" encoding="UTF-8"?>
<objectGridConfig xmlns:xsi="http://www.w3.org/2001/XMLSchema-instance"
  xsi:schemaLocation="http://ibm.com/ws/objectgrid/config ../objectGrid.xsd"
  xmlns="http://ibm.com/ws/objectgrid/config">
  <objectGrids>
    <objectGrid name="CompanyGrid">
      <bean id="TransactionCallback"
        className="com.company.MyClientTxCallback" />
      <bean id="ObjectGridEventListener" className="" />
      <backingMap name="Customer" numberOfBuckets="1429"
        pluginCollectionRef="customerPlugins" />
      <backingMap name="Item" />
      <backingMap name="OrderLine" numberOfBuckets="701"
        timeToLive="800" ttlEvictorType="LAST_ACCESS_TIME" />
      <backingMap name="Order" lockStrategy="PESSIMISTIC"
        pluginCollectionRef="orderPlugins" />
    </objectGrid>
  </objectGrids>
  <backingMapPluginCollections>
    <backingMapPluginCollection id="customerPlugins">
      <bean id="Evictor"
        className="com.ibm.websphere.objectgrid.plugins.builtins.LRUEvictor" />
      <bean id="MapEventListener" className="" />
    </backingMapPluginCollection>
    <backingMapPluginCollection id="orderPlugins">
      <bean id="MapIndexPlugin"
        className="com.company.MyMapIndexPlugin" />
    </backingMapPluginCollection>
  </backingMapPluginCollections>
</objectGridConfig>

```

Die Datei `companyGridClientSide.xml` überschreibt mehrere Attribute und Plug-ins im CompanyGrid-Client. Im Client ist `"com.company.MyClientTxCallback"` der der TransactionCallback, wohingegen der TransactionCallback, der im Server ausgeführt wird, `"com.company.MyTxCallback"` ist. Der `ObjectGridEventListener` wird im Client entfernt, weil der `className`-Wert eine leere Zeichenfolge ist.

Für die BackingMap `"Customer"` wird der `numberOfBuckets`-Wert im Client auf 1429 gesetzt. Die BackingMap `"Customer"` übernimmt das Bereinigungsprogramm (Evictor) aus der Serverkonfiguration, aber der `MapEventListener` wird aus dem Client entfernt.

Die Attribute `numberOfBuckets` und `timeToLive` wurde auf der Clientseite der BackingMap `"OrderLine"` angepasst.

Das Attribut `"lockStrategy"` in dieser Datei wird ignoriert, egal welcher Wert angegeben wird. Das Überschreiben dieses Attribut auf Clientseite wird nicht unterstützt.

Zum Erstellen eines CompanyGrid-Clients über die Datei `companyGridClientSide.xml` übergeben Sie die ObjectGrid-XML-Datei als URL an eine der `connect`-Methoden im `ObjectGridManager`.

#### Client für XML erstellen

```

ObjectGridManager ogManager =
  ObjectGridManagerFactory.ObjectGridManager();
ClientClusterContext clientClusterContext =
  ogManager.connect("MyServer1.company.com:2809", null, new URL(
    "file:xml/companyGridClientSide.xml"));

```

#### Client über das Programm konfigurieren

Sie können die clientseitigen ObjectGrid-Einstellungen auch über das Programm überschreiben. Erstellen Sie ein `ObjectGridConfiguration`-Objekt, das in der Struktur dem serverseitigen `ObjectGrid` gleicht. Der folgende Code erstellt ein clientseitiges `ObjectGrid`, das funktional äquivalent zu dem `ObjectGrid` ist, das mit der Datei `"companyGridClientSide.xml"` aus dem vorherigen Abschnitt erstellt wird.

```

Clientseitige Einstellungen über das Programm überschreiben
ObjectGridConfiguration companyGridConfig = ObjectGridConfigFactory
  .createObjectGridConfiguration("CompanyGrid");
Plugin txCallbackPlugin = ObjectGridConfigFactory.createPlugin(

```



```

        PluginType.TRANSACTION_CALLBACK, "com.company.MyClientTxCallback");
companyGridConfig.addPlugin(txCallbackPlugin);

Plugin ogEventListenerPlugin = ObjectGridConfigFactory.createPlugin(
    PluginType.OBJECTGRID_EVENT_LISTENER, "");
companyGridConfig.addPlugin(ogEventListenerPlugin);

BackingMapConfiguration customerMapConfig = ObjectGridConfigFactory
    .createBackingMapConfiguration("Customer");
customerMapConfig.setNumberOfBuckets(1429);
Plugin evictorPlugin = ObjectGridConfigFactory.createPlugin(PluginType.EVICTOR,
    "com.ibm.websphere.objectgrid.plugins.builtins.LRUEvictor");
customerMapConfig.addPlugin(evictorPlugin);

companyGridConfig.addBackingMapConfiguration(customerMapConfig);

BackingMapConfiguration orderLineMapConfig = ObjectGridConfigFactory
    .createBackingMapConfiguration("OrderLine");
orderLineMapConfig.setNumberOfBuckets(701);
orderLineMapConfig.setTimeToLive(800);
orderLineMapConfig.setTtlEvictorType(TTLType.LAST_ACCESS_TIME);

companyGridConfig.addBackingMapConfiguration(orderLineMapConfig);

List ogConfigs = new ArrayList();
ogConfigs.add(companyGridConfig);

Map overrideMap = new HashMap();
overrideMap.put(CatalogServerProperties.DEFAULT_DOMAIN, ogConfigs);

ogManager.setOverrideObjectGridConfigurations(overrideMap);
ClientClusterContext client = ogManager.connect(catalogServerAddresses, null, null);
ObjectGrid companyGrid = ogManager.getObjectGrid(client, objectGridName);

```

Es werden nur die ObjectGridConfiguration- und BackingMapConfiguration-Objekte, die in der BackingMap "overrideMap" enthalten sind, auf überschriebene Plug-ins und Attribute hin überprüft. Der vorherige Code überschreibt beispielsweise die Anzahl der Buckets in der Map "OrderLine". Die Map "Order" bleibt auf der Clientseite jedoch unverändert, weil keine entsprechende Konfiguration eingeschlossen ist.

## Client in Spring Framework konfigurieren

Clientseitige ObjectGrid-Einstellungen können auch über Spring Framework überschrieben werden. Die folgende XML-Beispieldatei veranschaulicht, wie eine ObjectGridConfiguration erstellt und zum Überschreiben einige clientseitiger Einstellungen verwendet wird. Diese Beispieldatei ruft dieselben APIs auf, die auch in der Konfiguration über das Programm verwendet werden. Das Beispiel ist funktional äquivalent zu zu dem Beispiel in der ObjectGrid-XML-Konfiguration.

```

Clientkonfiguration mit Spring
<?xml version="1.0" encoding="UTF-8"?>
<!DOCTYPE beans PUBLIC "-//SPRING//DTD BEAN//EN"
    "http://www.springframework.org/dtd/spring-beans.dtd">
<beans>
  <bean id="companyGrid" factory-bean="manager" factory-method="getObjectGrid"
    singleton="true">
    <constructor-arg type="com.ibm.websphere.objectgrid.ClientClusterContext">
      <ref bean="client" />
    </constructor-arg>
    <constructor-arg type="java.lang.String" value="CompanyGrid" />
  </bean>

  <bean id="manager" class="com.ibm.websphere.objectgrid.ObjectGridManagerFactory"
    factory-method="getObjectGridManager" singleton="true">
    <property name="overrideObjectGridConfigurations">
      <map>
        <entry key="DefaultDomain">
          <list>
            <ref bean="ogConfig" />
          </list>
        </entry>
      </map>
    </property>
  </bean>

  <bean id="ogConfig"
    class="com.ibm.websphere.objectgrid.config.ObjectGridConfigFactory"
    factory-method="createObjectGridConfiguration">
    <constructor-arg type="java.lang.String">
      <value>CompanyGrid</value>
    </constructor-arg>
  </bean>

```

```

</constructor-arg>
<property name="plugins">
  <list>
<bean class="com.ibm.websphere.objectgrid.config.ObjectGridConfigFactory"
  factory-method="createPlugin">
  <constructor-arg type="com.ibm.websphere.objectgrid.config.PluginType"
  value="TRANSACTION_CALLBACK" />
  <constructor-arg type="java.lang.String"
  value="com.company.MyClientTxCallback" />
  </bean>
<bean class="com.ibm.websphere.objectgrid.config.ObjectGridConfigFactory"
  factory-method="createPlugin">
  <constructor-arg type="com.ibm.websphere.objectgrid.config.PluginType"
  value="OBJECTGRID_EVENT_LISTENER" />
  <constructor-arg type="java.lang.String" value="" />
  </bean>
  </list>
</property>
<property name="backingMapConfigurations">
  <list>
  <bean class="com.ibm.websphere.objectgrid.config.ObjectGridConfigFactory"
  factory-method="createBackingMapConfiguration">
  <constructor-arg type="java.lang.String" value="Customer" />
  <property name="plugins">
  <bean class="com.ibm.websphere.objectgrid.config.ObjectGridConfigFactory"
  factory-method="createPlugin">
  <constructor-arg type="com.ibm.websphere.objectgrid.config.PluginType"
  value="EVICTOR" />
  <constructor-arg type="java.lang.String"
  value="com.ibm.websphere.objectgrid.plugins.builtins.LRUEvictor" />
  </bean>
  </property>
  <property name="numberOfBuckets" value="1429" />
  </bean>
  <bean class="com.ibm.websphere.objectgrid.config.ObjectGridConfigFactory"
  factory-method="createBackingMapConfiguration">
  <constructor-arg type="java.lang.String" value="OrderLine" />
  <property name="numberOfBuckets" value="701" />
  </bean>
  </list>
</property>
<property name="timeToLive" value="800" />
<property name="ttlEvictorType">
  <value type="com.ibm.websphere.objectgrid.
  TTLType">LAST_ACCESS_TIME</value>
</property>
</bean>
</list>
</property>
</bean>

<bean id="client" factory-bean="manager" factory-method="connect"
  singleton="true">
  <constructor-arg type="java.lang.String">
  <value>localhost:2809</value>
  </constructor-arg>
<constructor-arg
  type="com.ibm.websphere.objectgrid.security.
  config.ClientSecurityConfiguration">
  <null />
  </constructor-arg>
<constructor-arg type="java.net.URL">
  <null />
  </constructor-arg>
</bean>
</beans>

```

Nachdem Sie die XML-Datei erstellt haben, laden Sie die Datei und erstellen das ObjectGrid mit dem folgenden Code-Snippet:

```

BeanFactory beanFactory = new XmlBeanFactory(new
  UrlResource("file:test/companyGridSpring.xml"));

```

```

ObjectGrid companyGrid = (ObjectGrid) beanFactory.getBean("companyGrid");

```

Weitere Informationen finden Sie im Abschnitt Kapitel 5, „Integration mit dem Spring-Framework“, auf Seite 203.

## Nahen Cache inaktivieren

Der nahe Cache wird standardmäßig aktiviert, wenn eine optimistische Sperrstrategie oder keine Sperrstrategie konfiguriert ist, und kann nicht verwendet werden, wenn eine pessimistische Sperrstrategie konfiguriert ist.

Zum Inaktivieren des nahen Caches müssen Sie das Attribut "numberOfBuckets" in der ObjectGrid-Deskriptordatei mit den Korrekturwerten des Clients auf "0" setzen.

Weitere Informationen finden Sie in der Dokumentation zum Sperren von Map-Einträgen im *Administratorhandbuch*.

## Lebenszyklus eines ObjectGrids steuern

Sie können die Schnittstelle "ObjectGridManager" verwenden, um den Lebenszyklus einer ObjectGrid-Instanz über eine Startup-Bean oder ein Servlet zu steuern.

### Lebenszyklus über eine Startup-Bean steuern

Eine Startup-Bean wird verwendet, um den Lebenszyklus einer ObjectGrid-Instanz zu steuern. Eine Startup-Bean wird geladen, wenn eine Anwendung gestartet wird. Mit einer Startup-Bean kann Code ausgeführt werden, sobald eine Anwendung erwartungsgemäß gestartet oder gestoppt wird. Zum Erstellen einer Startup-Bean verwenden Sie die Home-Schnittstelle "com.ibm.websphere.startupservice.AppStartUpHome" und die Remote-Schnittstelle "com.ibm.websphere.startupservice.AppStartUp". Implementieren Sie die Methoden "start" und "stop" in der Bean. Die Methode "start" wird aufgerufen, wenn die Anwendung gestartet wird. Die Methode "stop" wird aufgerufen, wenn die Anwendung beendet wird. Die Methode "start" wird verwendet, um ObjectGrid-Instanzen zu erstellen. Die Methode "stop" wird verwendet, um ObjectGrid-Instanzen zu entfernen. Im Folgenden sehen Sie ein Code-Snippet, das dieses Management des ObjectGrid-Lebenszyklus in einer Startup-Bean demonstriert:

```
public class MyStartupBean implements javax.ejb.SessionBean {
    private ObjectGridManager objectGridManager;

    /* Die Methoden in der Schnittstelle "SessionBean" wurden
     * in diesem Beispiel weggelassen, damit das Beispiel nicht
     * zu lang wird. */
    public boolean start(){
        // Startup-Bean starten.
        // Diese Methode wird aufgerufen, wenn die Anwendung gestartet wird.
        objectGridManager = ObjectGridManagerFactory.getObjectGridManager();
        try {
            // 2 ObjectGrids erstellen und die Instanzen zwischenspeichern.
            ObjectGrid bookstoreGrid =
            objectGridManager.createObjectGrid("bookstore", true);
            bookstoreGrid.defineMap("book");
            ObjectGrid videostoreGrid =
            objectGridManager.createObjectGrid("videostore", true);
            // In der JVM könne diese ObjectGrids jetzt über die Methode
            // getObjectGrid(String) vom ObjectGridManager abgerufen werden.
        } catch (ObjectGridException e) {
            e.printStackTrace();
            return false;
        }

        return true;
    }

    public void stop(){
        // Startup-Bean stoppen.
        // Diese Methode wird aufgerufen, wenn die Anwendung gestoppt wird.
        try {
            // Zwischengespeicherte ObjectGrids entfernen und löschen.
            objectGridManager.removeObjectGrid("bookstore", true);
            objectGridManager.removeObjectGrid("videostore", true);
        } catch (ObjectGridException e) {
```

```

        e.printStackTrace();
    }
}

```

Nach dem Aufruf der Methode "start" werden die neu erstellten ObjectGrid-Instanzen von der Schnittstelle "ObjectGridManager" abgerufen. Wenn beispielsweise ein Servlet in der Anwendung enthalten ist, greift das Servlet über das folgende Code-Snippet auf eXtreme Scale zu:

```

ObjectGridManager objectGridManager =
    ObjectGridManagerFactory.getObjectGridManager();
ObjectGrid bookstoreGrid = objectGridManager.getObjectGrid("bookstore");
ObjectGrid videostoreGrid = objectGridManager.getObjectGrid("videostore");

```

## Lebenszyklus über ein Servlet steuern

Wenn Sie den Lebenszyklus eines ObjectGrid über ein Servlet verwalten möchten, können Sie die Methode "init" verwenden, um eine ObjectGrid-Instanz zu erstellen, und die Methode "destroy", um die ObjectGrid-Instanz zu entfernen. Wenn die ObjectGrid-Instanz zwischengespeichert ist, wird sie im Servlet-Code abgerufen und bearbeitet. Im Folgenden sehen Sie einen Mustercode, der das Erstellen, Bearbeiten und Löschen eines ObjectGrids demonstriert:

```

public class MyObjectGridServlet extends HttpServlet implements Servlet {
    private ObjectGridManager objectGridManager;

    public MyObjectGridServlet() {
        super();
    }

    public void init(ServletConfig arg0) throws ServletException {
        super.init();
        objectGridManager = ObjectGridManagerFactory.getObjectGridManager();
        try {
            // ObjectGrid mit dem Namen "bookstore" erstellen und zwischenspeichern.
            ObjectGrid bookstoreGrid =
                objectGridManager.createObjectGrid("bookstore", true);
            bookstoreGrid.defineMap("book");
        } catch (ObjectGridException e) {
            e.printStackTrace();
        }
    }

    protected void doGet(HttpServletRequestRequest req, HttpServletResponse res)
        throws ServletException, IOException {
        ObjectGrid bookstoreGrid = objectGridManager.getObjectGrid("bookstore");
        Session session = bookstoreGrid.getSession();
        ObjectMap bookMap = session.getMap("book");
        // Operationen für das zwischengespeicherte ObjectGrid durchführen.
        // ...
    }

    public void destroy() {
        super.destroy();
        try {
            // Zwischengespeichertes ObjectGrid "bookstore" entfernen und löschen.
            objectGridManager.removeObjectGrid("bookstore", true);
        } catch (ObjectGridException e) {
            e.printStackTrace();
        }
    }
}

```

---

## Auf das ObjectGrid-Shard zugreifen

Anwendungen wie eXtreme Scale erreichen hohe Verarbeitungsraten, indem sie die Logik dorthin verschieben, wo sich die Daten befinden, und nur das Ergebnis an den Client zurückgeben.

Anwendungslogik in einer Client-JVM muss Daten aus der Server-JVM extrahieren, die die Daten enthält, und mit Push zurückgeben, wenn die Transaktion festgeschrieben wird. Dieser Prozess senkt die Geschwindigkeit, in der die Daten verarbeitet werden können. Wenn sich die Anwendungslogik in derselben JVM befindet wie das Shard, das die Daten enthält, fallen Netzlatenzzeit und Marshaling-Kosten weg, und dies kann zu einem erheblichen Leistungsgewinn führen.

### Lokale Referenz auf Shard-Daten

Die ObjectGrid-APIs übertragen ein Session-Objekt an die serverseitige Methode. Dieses Session-Objekt ist eine direkte Referenz auf die Daten für dieses Shard. Es ist keine Routing-Logik in diesem Pfad enthalten. Die Anwendungslogik kann mit den Daten für dieses Shard direkt arbeiten. Das Session-Objekt kann nicht verwendet werden, um auf die Daten in einer anderen Partition zuzugreifen, weil keine Routing-Logik vorhanden ist.

Ein Loader-Plug-in ist ebenfalls eine Methode, ein Ereignis zu empfangen, wenn ein Shard zu einer primären Partition wird. Eine Anwendung kann einen Loader und die Schnittstelle "ReplicaPreloadController" implementieren. Die Methode zum Überprüfen des Preload-Status wird nur aufgerufen, wenn ein Shard zu einem primären Shard wird. Das an die Methode übergebene Session-Objekt ist eine lokale Referenz auf die Daten des Shards. Dieser Ansatz wird normalerweise verwendet, wenn eine Partition Threads starten oder eine Nachrichtenstruktur für partitionsbezogenen Datenverkehr subscribieren muss. Möglicherweise muss sie einen Thread starten, um Nachrichten in einer lokalen Map über die API "getNextKey" zu empfangen.

### Optimierung der Client/Server-Co-Location

Wenn eine Anwendung die Client-APIs verwendet, um auf eine Partition zuzugreifen, die sich in derselben JVM befindet, die auch den Client enthält (dies wird als Co-Location bezeichnet), wird das Netz umgangen, aber es findet aufgrund der derzeitigen Implementierungsprobleme noch ein gewisses Marshaling statt. Wenn ein partitioniertes Grid verwendet wird, sind keine Einbußen bei der Anwendungsleistung zu verzeichnen, weil (N-1)/N Aufrufe an eine andere JVM weitergeleitet werden. Verwenden Sie die APIs "Loader" und "ObjectGrid" für den Aufruf dieser Logik, wenn Sie immer mit lokalen Zugriffen auf ein Shard arbeiten müssen.

---

## Session-Objekte für den Zugriff auf Daten im Grid verwenden

Anwendungen können Transaktionen über die Schnittstelle "Session" starten und beenden. Die Schnittstelle "Session" ermöglicht auch den Zugriff auf die anwendungsbasierten Schnittstellen "ObjectMap" und "JavaMap".

Jede ObjectMap- bzw. JavaMap-Instanz ist direkt an ein bestimmtes Session-Objekt gebunden. Jeder Thread, der auf eXtreme Scale zugreifen möchte, muss zuerst ein Session-Objekt vom ObjectGrid-Objekt abrufen. Eine Session-Instanz kann nicht gleichzeitig von mehreren Threads genutzt werden. WebSphere eXtreme Scale ver-

wendet keinen Thread-eigenen Speicher, aber Plattformbeschränkungen können die Möglichkeit, ein Session-Objekt von einem Thread an den anderen zu übergeben, einschränken.

## Methoden

Die folgenden Methoden werden mit der Schnittstelle "Session" bereitgestellt. Weitere Informationen zu diesen Methoden finden Sie in der API-Dokumentation.

```
public interface Session {
    ObjectMap getMap(String cacheName) throws UndefinedMapException;

    void begin() throws TransactionAlreadyActiveException, TransactionException;

    void beginNoWriteThrough() throws TransactionAlreadyActiveException, TransactionException;
    public void commit() throws NoActiveTransactionException, TransactionException;

    public void rollback() throws NoActiveTransactionException, TransactionException;

    public void flush() throws TransactionException;

    TxID getTxID() throws NoActiveTransactionException;

    boolean isWriteThroughEnabled();

    void setTransactionType(String tranType);

    public void processLogSequence(LogSequence logSequence) throws
    NoActiveTransactionException, UndefinedMapException, ObjectGridException;

    ObjectGrid getObjectGrid();

    public void setTransactionTimeout(int timeout);
    public int getTransactionTimeout();
    public boolean transactionTimedOut();

    public boolean isCommitting();
    public boolean isFlushing();

    public void markRollbackOnly(Throwable t) throws NoActiveTransactionException;
    public boolean isMarkedRollbackOnly();
}
```

### Methode "get"

Eine Anwendung ruft eine Session-Instanz von einem ObjectGrid-Objekt über die Methode "ObjectGrid.getSession" ab. Das folgende Beispiel veranschaulicht, wie eine Session-Instanz abgerufen wird:

```
ObjectGrid objectGrid = ...; Session sess = objectGrid.getSession();
```

Nach dem Abruf einer Session-Instanz verwaltet der Thread eine Referenz zur eigenen Verwendung auf das Session-Objekt. Wenn Sie die Methode "getSession" mehrfach aufrufen, wird jedesmal ein neues Session-Objekt zurückgegeben.

### Transaktionen und Session-Methoden

Ein Session-Objekt kann verwendet werden, um Transaktionen zu starten, festzuschreiben oder rückgängig zu machen. Operationen für BackingMaps über ObjectMaps und JavaMaps werden am effizientesten in einer Session-Transaktion durchgeführt. Nach dem Starten einer Transaktion werden alle Änderungen, die an einer oder mehreren BackingMaps im Geltungsbereich dieser Transaktion vorgenommen, in einem speziellen Transaktionscache gespeichert, bis die Transaktion festgeschrieben wird. Wenn eine Transaktion festgeschrieben wird, werden die offenen Änderungen auf die BackingMaps und Loader angewendet und werden damit für andere Clients dieses ObjectGrids sichtbar.

WebSphere eXtreme Scale unterstützt auch die Möglichkeit, Transaktionen automatisch festzuschreiben (auto-commit). Wenn ObjectMap-Operationen außerhalb des

Kontextes einer aktiven Transaktion durchgeführt werden, wird eine implizite Transaktion gestartet, bevor die Operation und die Transaktion vor der Rückgabe der Steuerung an die Anwendung festgeschrieben werden.

```
Session session = objectGrid.getSession();
ObjectMap objectMap = session.getMap("someMap");
session.begin();
objectMap.insert("key1", "value1");
objectMap.insert("key2", "value2");
session.commit();
objectMap.insert("key3", "value3"); // auto-commit
```

### **Methode "Session.flush"**

Die Methode "Session.flush" ist nur sinnvoll, wenn einer BackingMap ein Loader zugeordnet ist. Die Flush-Methode ruft den Loader mit dem aktuellen Änderungssatz im Transaktionscache auf. Der Loader wendet die Änderungen auf das Back-End an. Diese Änderungen werden beim Aufruf der Flush-Operation nicht festgeschrieben. Wenn eine Session-Transaktion nach dem Aufruf einer Flush-Operation festgeschrieben wird, werden nur die Aktualisierungen auf den Loader angewendet, die nach dem Aufruf der Flush-Methode vorgenommen wurden. Wenn eine Session-Transaktion nach dem Aufruf einer Flush-Methode rückgängig gemacht wird, werden die mit Flush übertragenen Änderungen zusammen mit allen anderen offenen Änderungen in der Transaktion verworfen. Verwenden Sie die Flush-Methode sparsam, weil sie die Möglichkeit von Stapeloperationen für einen Loader beschränkt. Im Folgenden sehen Sie ein Beispiel für die Verwendung der Methode "Session.flush":

```
Session session = objectGrid.getSession();
session.begin();
// Änderungen vornehmen
...
session.flush(); // Änderungen mit Push an den Loader übertragen, aber noch nicht festschreiben
// Weitere Änderungen vornehmen
...
session.commit();
```

### **Methode "NoWriteThrough"**

Einige Maps von eXtreme Scale werden durch einen Loader gestützt, der einen persistenten Speicher für die Daten in der Map bereitstellt. Manchmal ist es hilfreich, Daten nur in der eXtreme-Scale-Map festzuschreiben und nicht mit Push an den Loader zu übertragen. Die Schnittstelle "Session" stellt zu diesem Zweck die Methode "beginNoWriteThrough" bereit. Die Methode "beginNoWriteThrough" startet wie die Methode "begin" eine Transaktion. Wenn Sie die Methode "beginNoWriteThrough" verwenden und die Transaktion festgeschrieben wird, werden die Daten nur in der speicherinternen Map von eXtreme Scale festgeschrieben und nicht in dem vom Loader bereitgestellten persistenten Speicher. Diese Methode ist sehr hilfreich beim vorherigen Laden von Daten in die Map.

Wenn Sie eine verteilte ObjectGrid-Instanz verwenden, ist die Methode "beginNoWriteThrough" hilfreich, um Änderungen nur im nahen Cache vorzunehmen, ohne den fernen Cache auf dem Server zu ändern. Wenn bekannt ist, dass die Daten im nahen Cache veraltet sind, können mit der Methode "beginNoWriteThrough" Einträge im nahen Cache ungültig gemacht werden, ohne sie auch im Server ungültig zu machen.

Die Schnittstelle "Session" stellt auch die Methode "isWriteThroughEnabled" bereit, mit der Sie feststellen können, welcher Typ von Transaktion momentan aktiv ist:

```

Session session = objectGrid.getSession();
session.beginNoWriteThrough();
// Änderungen vornehmen
session.commit(); // Diese Änderungen werden nicht mit Push an den Loader übertragen

```

### **Methode zum Abrufen des TxID-Objekts**

Das TxID-Objekt ist ein nicht transparentes Objekt, das die aktive Transaktion identifiziert. Verwenden Sie das TxID-Objekt für die folgenden Zwecke:

- für Vergleiche, wenn Sie eine bestimmte Transaktion suchen,
- zum Speichern der von den TransactionCallback- und Loader-Objekten gemeinsam genutzten Daten.

Weitere Informationen zum Objekt-Slot-Feature finden Sie in den Beschreibungen von TransactionCallback-Plug-ins und Loadern.

### **Methode für Leistungsüberwachung**

Wenn Sie eXtreme Scale in WebSphere Application Server verwenden, kann es erforderlich sein, den Transaktionstyp für die Leistungsüberwachung zurückzusetzen. Sie können den Transaktionstyp mit der Methode "setTransactionType" festlegen. Weitere Informationen zur Methode "setTransactionType" finden Sie im Abschnitt zur Überwachung der ObjectGrid-Leistung mit WebSphere Application Server Performance Monitoring Infrastructure (PMI).

### **Methode zur Verarbeitung eines vollständigen LogSequence-Objekts**

WebSphere eXtreme Scale kann ganze Sätze von Map-Änderungen an ObjectGrid-Listener zum Verteilen von Maps von einer Java Virtual Machine an eine andere weitergeben. Um dem Listener die Verarbeitung empfangener LogSequence-Objekte zu erleichtern, stellt die Schnittstelle "Session" die Methode "processLogSequence" bereit. Diese Methode untersucht jedes LogElement-Objekt im LogSequence-Objekt und führt die entsprechende Operation, z. B. insert (Einfügen), update (Aktualisieren), invalidate (Ungültigmachen) usw. für die BackingMap aus, die mit dem Argument "MapName" des LogSequence-Objekts angegeben wurde. Es muss eine ObjectGrid-Sitzung verfügbar sein, bevor die Methode "processLogSequence" aufgerufen wird. Die Anwendung ist auch für das Absetzen der entsprechenden Commit- und Rollback-Aufrufe zuständig, um die Sitzung zu beenden. Die automatische Festschreibungsverarbeitung ist für diesen Methodenaufruf nicht verfügbar. Normalerweise startet der empfangende ObjectGridEventListener der fernen JVM ein Session-Objekt mit der Methode "beginNoWriteThrough", wodurch eine endlose Weitergabe von Änderungen verhindert wird. Anschließend setzt er einen Aufruf an die Methode "processLogSequence" ab und schreibt dann die Transaktion fest bzw. macht sie rückgängig.

```

// Bei der Initialisierung des ObjectGridEventListeners
// übergebenes Session-Objekt verwenden...
session.beginNoWriteThrough();
// Empfangenes LogSequence-Objekt verarbeiten
try {
    session.processLogSequence(receivedLogSequence);
} catch (Exception e) {
    session.rollback(); throw e;
}
// Änderungen festschreiben
session.commit();

```

### **Methode "markRollbackOnly"**



Diese Methode wird verwendet, um die aktuelle Transaktion als "rollback only" (Nur Rollback) kennzuzeichnen. Das Kennzeichnen einer Transaktion als "rollback only" stellt sicher, dass die Transaktion auch dann rückgängig gemacht wird, wenn die Methode "commit" von der Anwendung aufgerufen wird. Diese Methode wird gewöhnlich von ObjectGrid selbst oder von der Anwendung verwendet, wenn diese weiß, dass Daten beschädigt werden könnten, wenn die Transaktion festgeschrieben wird. Nach dem Aufruf dieser Methode wird das an diese Methode übergebene Throwable-Objekt mit der Ausnahme "com.ibm.websphere.objectgrid.TransactionException" verkettet, die von der Methode "commit" ausgelöst wird, wenn diese für ein Session-Objekt aufgerufen wird, das zuvor als "rollback only" gekennzeichnet wurde. Alle nachfolgenden Aufrufe dieser Methode für eine Transaktion, die bereits mit "rollback only" gekennzeichnet ist, werden ignoriert, d. h., es wird nur der erste Aufruf, der eine Throwable-Referenz ungleich null übergibt, verwendet. Sobald die gekennzeichnete Transaktion abgeschlossen ist, wird die Markierung "rollback only" entfernt, so dass die nächste Transaktion, die vom Session-Objekt gestartet wird, festgeschrieben werden kann.

#### **Methode "isMarkedRollbackOnly"**

Diese Methode kehrt zurück, wenn das Session-Objekt momentan mit "rollback only" gekennzeichnet ist. Diese Methode gibt den booleschen Wert "true" zurück, wenn die Methode "markRollbackOnly" vorher in diesem Session-Objekt aufgerufen wurde und die vom Session-Objekt gestartete Transaktion immer noch aktiv ist.

#### **Methode "setTransactionTimeout"**

Setzen Sie das Transaktionszeitlimit für die nächste von diesem Session-Objekt gestartete Transaktion auf eine bestimmte Anzahl von Sekunden. Diese Methode hat keine Auswirkung auf das Transaktionszeitlimit von Transaktionen, die bereits von diesem Session-Objekt gestartet wurden. Sie beeinflusst nur die Transaktionen, die nach dem Aufruf dieser Methode gestartet werden. Wenn Sie diese Methode nicht aufrufen, wird der Zeitlimitwert verwendet, der an die Methode "setTxTimeout" der Methode "com.ibm.websphere.objectgrid.ObjectGrid" übergeben wurde.

#### **Methode "getTransactionTimeout"**

Diese Methode gibt den Transaktionszeitlimitwert in Sekunden zurück. Es wird der Wert, der als Zeitlimitwert an die Methode "setTransactionTimeout" übergeben wurde, von dieser Methode zurückgegeben. Wenn Sie die Methode "setTransactionTimeout" nicht aufrufen, wird der Zeitlimitwert verwendet, der an die Methode "setTxTimeout" der Methode "com.ibm.websphere.objectgrid.ObjectGrid" übergeben wurde.

#### **Methode "transactionTimedOut"**

Diese Methode gibt den booleschen Wert "true" zurück, wenn die aktuelle Transaktion, die von diesem Session-Objekt gestartet wurde, das zulässige Zeitlimit überschreitet.

#### **Methode "isFlushing"**

Diese Methode gibt den booleschen Wert "true" zurück, wenn alle Transaktionsänderungen auf den Aufruf der Methode "flush" in der Schnittstelle "Session" hin an den Loader übertragen wurden. Für ein Loader-Plug-in kann diese Methode hilfreich sein, wenn es wissen muss, warum seine Methode "batchUpdate" aufgerufen wurde.

### Methode "isCommitting"

Diese Methode gibt den booleschen Wert "true" zurück, wenn alle Transaktionsänderungen auf den Aufruf der Methode "commit" der Schnittstelle "Session" hin festgeschrieben wurden. Für ein Loader-Plug-in kann diese Methode hilfreich sein, wenn es wissen muss, warum seine Methode "batchUpdate" aufgerufen wurde.

### Methode "setRequestRetryTimeout"

Diese Methode legt den Zeitlimitwert für Anforderungswiederholungen in Millisekunden fest. Wenn der Client ein Zeitlimit für Anforderungswiederholungen gesetzt hat, überschreibt die Sitzungseinstellung den Clientwert.

### Methode "getRequestRetryTimeout"

Diese Methode ruft die aktuelle Zeitlimiteinstellung für Anforderungswiederholungen in der Sitzung ab. Der Wert -1 zeigt an, dass kein Zeitlimit definiert wurde. Der Wert 0 zeigt an, dass der Modus für schnelles Fehlschlagen (fast-fail) aktiv ist. Ein Wert größer als 0 zeigt die Zeitlimiteinstellung in Millisekunden an.

---

## Handhabung von Sperren

Sperren haben einen Lebenszyklus, und unterschiedliche Typen von Sperren sind auf verschiedene Arten miteinander kompatibel. Sperren müssen in der richtigen Reihenfolge verarbeitet werden, um Deadlock-Szenarios zu vermeiden.

### Lebenszyklus einer Sperre

**Zeitlimits für Sperren**Jede BackingMap hat ein Standardzeitlimit für Sperren. Das Zeitlimit wird verwendet, um sicherzustellen, dass eine Anwendung nicht endlos auf die Erteilung eines Sperrmodus wartet, weil eine Deadlock-Situation aufgrund eines Anwendungsfehlers eingetreten ist. Die Anwendung kann die Schnittstelle "BackingMap" verwenden, um das Standardzeitlimit für Sperren zu überschreiben. Das folgende Beispiel veranschaulicht, wie das Wartezeitlimit für Sperren für die BackingMap "map1" auf 60 Sekunden gesetzt wird:

```
import com.ibm.websphere.objectgrid.BackingMap;
import com.ibm.websphere.objectgrid.LockStrategy;
import com.ibm.websphere.objectgrid.ObjectGrid;
import com.ibm.websphere.objectgrid.ObjectGridManagerFactory;
...
ObjectGrid og =
    ObjectGridManagerFactory.getObjectGridManager().createObjectGrid("test");
BackingMap bm = og.defineMap("map1");
bm.setLockStrategy( LockStrategy.PESSIMISTIC );
bm.setLockTimeout( 60 );
```

Um eine Ausnahme des Typs "java.lang.IllegalStateException" zu vermeiden, rufen Sie die Methode "setLockStrategy" und die Methode "setLockTimeout" auf, bevor Sie die Methode "initialize" oder "getSession" für die ObjectGrid-Instanz aufrufen. Der Parameter für die Methode "setLockTimeout" ist ein primitiver Java-Integer, der die Anzahl der Sekunden angibt, die eXtreme Scale auf die Erteilung des Sperrmodus wartet. Wenn die Wartezeit einer Transaktion den für die BackingMap konfigurierten Wert für das Wartezeitlimit für Sperren überschreitet, wird eine Ausnahme des Typs "com.ibm.websphere.objectgrid.LockTimeoutException" ausgelöst.

Wenn eine Ausnahme des Typs "LockTimeoutException" eintritt, muss die Anwendung feststellen, ob die Zeitlimitüberschreitung aufgetreten ist, weil die Anwen-

dung langsamer als erwartet arbeitet oder weil eine Deadlock-Situation vorliegt. Wenn eine echte Deadlock-Situation eingetreten ist, kann die Ausnahme durch Erhöhen des Wartezeitlimits für Sperren nicht behoben werden. Das Erhöhen des Zeitlimits führt lediglich dazu, dass die Ausnahme später ausgelöst wird. Falls die Ausnahme jedoch durch Erhöhung des Wartezeitlimits für Sperren behoben werden können, ist das Problem darauf zurückzuführen, dass die Anwendung langsamer gearbeitet hat als erwartet. Die Anwendung muss in diesem Fall feststellen, warum die Leistung nicht den Erwartungen entspricht.

### Wartezeitlimit für Sperren für ObjectMaps

Das Wartezeitlimit für Sperren kann für eine einzelne ObjectMap-Instanz mit der Methode "ObjectMap.setLockTimeout" überschrieben werden. Das Zeitlimit für Sperren wirkt sich auf alle Transaktionen aus, die nach der Festlegung des neuen Zeitlimits gestartet werden. Diese Methode kann hilfreich sein, wenn Sperrkollisionen möglich oder in Auswahltransaktionen erwartet werden.

### Gemeinsam genutzte, aktualisierbare und exklusive Sperren

Wenn eine Anwendung eine Methode der Schnittstelle "ObjectMap" aufruft, die find-Methoden für einen Index verwendet oder eine Abfrage durchführt, versucht eXtreme Scale automatisch, eine Sperre für den Map-Eintrag zu setzen, auf den zugegriffen wird. WebSphere eXtreme Scale verwendet je nach Methode, die die Anwendung in der Schnittstelle "ObjectMap" aufruft, die folgenden Sperrmodi.

- Die Methoden "get" und "getAll" in der Schnittstelle "ObjectMap", Indexmethoden und Abfragen erfordern eine *S-Sperre* bzw. einen gemeinsamen Sperrmodus für den Schlüssel eines Map-Eintrags. Wie lange diese S-Sperre gehalten wird, richtet sich nach der verwendeten Isolationsstufe. Eine S-Sperre lässt gemeinsame Zugriffe mehrerer Transaktionen zu, die versuchen, eine S- oder U-Sperre für denselben Schlüssel anzufordern, blockiert aber alle anderen Transaktionen, die versuchen, eine exklusive Sperre für denselben Schlüssel abzurufen.
- Die Methoden "getForUpdate" und "getAllForUpdate" fordern eine *U-Sperre* (das "U" steht für "Upgradeable", d. h. aktualisierbar) oder einen aktualisierbaren Sperrmodus für den Schlüssel eines Map-Eintrags an. Die U-Sperre wird gehalten, bis die Transaktion abgeschlossen wird. Eine U-Sperre lässt gemeinsame Zugriffe mehrerer Transaktionen zu, die eine S-Sperre für denselben Schlüssel anfordern, blockiert aber anderen Transaktionen, die versuchen, eine U-Sperre oder eine exklusive Sperre für denselben Schlüssel abzurufen.
- Die Methoden "put", "putAll", "remove", "removeAll", "insert", "update" und "touch" fordern eine *X-Sperre* (das "X" steht für "Exclusive", d. h. exklusiv) oder einen exklusiven Sperrmodus für den Schlüssel eines Map-Eintrags an. Die X-Sperre wird gehalten, bis die Transaktion abgeschlossen wird. Eine X-Sperre stellt sicher, dass nur eine einzige Transaktion einen Map-Eintrag mit einem bestimmten Schlüsselwert einfügt, aktualisiert oder entfernt. Eine X-Sperre blockiert alle anderen Transaktionen, die versuchen, eine S-, U- oder X-Sperre für denselben Schlüssel anzufordern.
- Die globale Methode "global" und die globale Methode "invalidateAll" fordern eine X-Sperre für jeden Map-Eintrag an, der ungültig gemacht wird. Die X-Sperre wird gehalten, bis die Transaktion abgeschlossen wird. Es werden keine Sperren für die lokale Methode "invalidate" und die lokale Methode "invalidateAll" angefordert, weil keiner der BackingMap-Einträge durch den Aufruf einer lokalen Methode "invalidate" ungültig gemacht wird.

Aus den vorherigen Definitionen geht eindeutig hervor, dass eine S-Sperre schwächer ist als eine U-Sperre, weil sie beim Zugriff auf denselben Map-Eintrag mehr Transaktionen gleichzeitig ausgeführt werden können. Die U-Sperre ist geringfügig stärker als die S-Sperre, weil sie andere Transaktionen blockiert, die eine U- oder X-Sperre anfordern. Im gemeinsamen Sperrmodus werden nur solche Transaktionen blockiert, die eine X-Sperre anfordern. Dieser kleine Unterschied ist wichtig, um bestimmte Deadlocks zu verhindern. Die X-Sperre ist die stärkste Sperre, weil sie alle anderen Transaktionen blockiert, die versuchen, eine S-, U- oder X-Sperre für denselben Map-Eintrag anzufordern. Der Reineffekt einer X-Sperre ist die Gewährleistung, dass nur eine einzige Transaktion einen Map-Eintrag einfügen, aktualisieren oder entfernen kann und keine Aktualisierungen verloren gehen, wenn mehrere Transaktionen versuchen, denselben Map-Eintrag zu aktualisieren.

Die folgende Tabelle ist eine Kompatibilitätsmatrix für Sperrmodi, die Sie verwenden können, um festzustellen, welche Sperrmodi miteinander kompatibel sind. Informationen zum Lesen dieser Matrix: In der Zeile wird ein Sperrmodus angegeben, der bereits erteilt wurde. In der Spalte wird der Sperrmodus angezeigt, der von einer anderen Transaktion angefordert wird. Wenn in der Spalte das Wort "Ja" steht, wird der von der anderen Transaktion angeforderte Sperrmodus erteilt, weil er mit dem bereits erteilten Sperrmodus kompatibel ist. Das Wort "Nein" gibt an, dass der angeforderte Sperrmodus nicht kompatibel ist und die andere Transaktion warten muss, bis die erste Transaktion die gehaltene Sperre freigibt.

Tabelle 1. Kompatibilitätsmatrix für Sperrmodi

Sperre	Sperrtyp S (gemeinsam genutzt)	Sperrtyp U (aktualisierbar)	Sperrtyp X (exklusiv)	Stärke
S (gemeinsam genutzt)	Ja	Ja	Nein	Am schwächsten
U (aktualisierbar)	Ja	Nein	Nein	Normal
X (exklusiv)	Nein	Nein	Nein	Am stärksten

## Sperr-Deadlocks

Stellen Sie sich die folgende Folge von Sperrmodusanforderungen vor:

1. Der Transaktion 1 wird eine X-Sperre für key1 erteilt.
2. Der Transaktion 2 wird eine X-Sperre für key2 erteilt.
3. Transaktion 1 fordert eine X-Sperre für key2 an. (Transaktion 1 wird blockiert und muss auf die von der Transaktion 2 gehaltene Sperre warten).
4. Transaktion 2 fordert eine X-Sperre für key1 an. (Transaktion 2 wird blockiert und muss auf die von der Transaktion 1 gehaltene Sperre warten).

Die vorherige Folge ist das klassische Deadlock-Beispiel, in dem zwei Transaktionen versuchen, mehrere Sperren anzufordern, und jede Transaktion die Sperren in einer anderen Reihenfolge anfordert. Um dieses Deadlock zu vermeiden, müssen die beiden Transaktionen die Sperren jeweils in derselben Reihenfolge anfordern. Wenn die optimistische Sperrstrategie verwendet wird und die Methode "flush" in der Schnittstelle "ObjectMap" von der Anwendung nie verwendet wird, werden Sperrmodi von der Transaktion nur während des Festschreibungszyklus angefordert. Während des Festschreibungszyklus legt eXtreme Scale die Schlüssel für die Map-Einträge fest, die gesperrt werden müssen, und fordert die Sperrmodi in Schlüsselreihenfolge an (deterministisches Verhalten). Mit dieser Methode verhindert eXtreme Scale die meisten der klassischen Deadlocks. eXtreme Scale kann jedoch nicht alle möglichen Deadlock-Szenarios verhindern. Es gibt ein paar Szenario-

rios, die die Anwendung berücksichtigen muss. Im Folgenden sind die Szenarios beschrieben, die die Anwendung beachten und für die sie entsprechende vorbeugende Maßnahmen ergreifen muss.

Es gibt ein Szenario, in dem eXtreme Scale ein Deadlock erkennen kann, ohne auf den Ablauf des Wartezeitlimits für Sperren zu warten. Wenn dieses Szenario eintritt, wird eine Ausnahme des Typs "com.ibm.websphere.objectgrid.LockDeadlockException" ausgelöst. Sehen Sie sich das folgende Code-Snippet an:

```
Session sess = ...;
ObjectMap person = sess.getMap("PERSON");
sess.begin();
Person p = (IPerson)person.get("Lynn");
// Lynn hat Geburtstag, also machen wir sie ein Jahr älter.
p.setAge( p.getAge() + 1 );
person.put( "Lynn", p );
sess.commit();
```

In dieser Situation möchte Lynns Freund Lynn älter machen, als sie ist, und Lynn und ihr Freund führen diese Transaktion gleichzeitig aus. In dieser Situation halten beide Transaktionen eine S-Sperre für den Eintrag "Lynn" in der Map "PERSON", weil die Methode "person.get("Lynn")" ausgerufen wurde. Wegen des Aufrufs der Methode "person.put("Lynn", p)" versuchen beide Transaktionen, die S-Sperre in eine X-Sperre zu aktualisieren. Beide Transaktionen werden blockiert und warten auf die Freigabe der S-Sperre durch die jeweils andere Transaktion. Demzufolge tritt eine Deadlock-Situation ein, weil eine zirkuläre Wartebedingung zwischen den beiden Transaktionen besteht. Eine zirkuläre Wartebedingung ergibt sich, wenn mehrere Transaktionen versuchen, eine schwächere Sperre auf eine stärkere Sperre für denselben Map-Eintrag hochzustufen. In diesem Szenario wird eine Ausnahme des Typs "LockDeadlockException" an Stelle einer Ausnahme des Typs "LockTimeoutException" ausgelöst.

Die Anwendung kann die Ausnahme des Typs "LockDeadlockException" für das vorherige Beispiel verhindern, indem sie die optimistische Sperrstrategie an Stelle der pessimistischen Sperrstrategie verwendet. Die Verwendung der optimistischen Sperrstrategie ist die bevorzugte Lösung, wenn die Map im Wesentlichen nur gelesen wird und nur wenige Aktualisierungen in der Map vorgenommen werden. Wenn Sie die pessimistische Sperrstrategie verwenden müssen, können Sie die Methode "getForUpdate" an Stelle der Methode "get" aus dem vorherigen Beispiel oder die Transaktionsisolationsstufe TRANSACTION\_READ\_COMMITTED verwenden.

Weitere Informationen finden Sie im Abschnitt zu den Sperrstrategien im Handbuch *Produktübersicht*.

Die Verwendung der Transaktionsisolationsstufe TRANSACTION\_READ\_COMMITTED verhindert, dass die S-Sperre, die von der Methode "get" angefordert wird, gehalten wird, bis die Transaktion abgeschlossen ist. Solange der Schlüssel im Transaktionscache nicht ungültig gemacht wird, ist ein wiederholbares Lesen garantiert.

Weitere Informationen finden Sie im Abschnitt zum Sperren von Einträgen im *Administratorhandbuch*.

Eine Alternative zum Ändern der Transaktionsisolationsstufe ist die Verwendung der Methode "getForUpdate". Die erste Transaktion, die die Methode "getForUpdate" aufruft, fordert eine U-Sperre an Stelle einer S-Sperre an. Dieser Sperrmodus bewirkt, dass die zweite Transaktion blockiert wird, wenn diese die Methode "get-

ForUpdate" aufruft, weil eine U-Sperre nur einer einzigen Transaktion erteilt wird. Da die zweite Transaktion blockiert ist, hält sie keine Sperre für den Map-Eintrag "Lynn". Die erste Transaktion wird nicht blockiert, wenn sie versucht, über den Aufruf der Methode "put" die U-Sperre in eine X-Sperre zu aktualisieren. Dieses Feature demonstriert, warum eine U-Sperre auch als *aktualisierbarer* Sperrmodus bezeichnet wird. Nach Abschluss der ersten Transaktion wird die Blockierung der zweiten Transaktion aufgehoben, und sie erhält die U-Sperre. Eine Anwendung kann das Deadlock-Szenario bei der Hochstufung von Sperrern verhindern, indem sie die Methode "getForUpdate" an Stelle der Methode "get" verwendet, wenn die pessimistische Sperrstrategie verwendet wird.

**Wichtig:** Diese Lösung verhindert nicht, dass Transaktionen, die mit Lesezugriff arbeiten, einen Map-Eintrag lesen können. Transaktionen mit Lesezugriff rufen die Methode "get" auf, rufen aber nie die Methoden "put", "insert", "update" und "remove" auf. Die Anzahl der gemeinsamen Zugriffe ist genauso hoch, wenn die reguläre Methode "get" verwendet wird. Eine Verringerung der gemeinsamen Zugriffe ist nur zu verzeichnen, wenn die Methode "getForUpdate" von mehreren Transaktionen für denselben Map-Eintrag aufgerufen wird.

Sie müssen erkennen, wenn eine Transaktion die Methode "getForUpdate" für mehrere Map-Einträge aufruft, um sicherzustellen, dass die U-Sperren von jeder Transaktion in derselben Reihenfolge angefordert werden. Angenommen, die erste Transaktion ruft die Methode "getForUpdate" für den Schlüssel 1 (key1) und die Methode "getForUpdate" für den Schlüssel 2 (key2) auf. Eine andere Transaktion ruft die Methode "getForUpdate" für dieselben Schlüssel, aber in umgekehrter Reihenfolge auf. Diese Folge löst die klassische Deadlock-Situation aus, weil mehrere Sperrern in unterschiedlicher Reihenfolge von unterschiedlichen Transaktionen angefordert werden. Die Anwendung muss weiterhin sicherstellen, dass jede Transaktion beim Zugriff auf mehrere Map-Einträge die Schlüsselreihenfolge einhält, um diese Deadlock-Situation zu verhindern. Da die U-Sperre angefordert wird, wenn die Methode "getForUpdate" aufgerufen wird, und nicht, wenn die Transaktion festgeschrieben wird, kann eXtreme Scale die Sperranforderungen nicht wie im Festschreibungszyklus sortieren. Die Anwendung muss die Reihenfolge der Sperrern in diesem Fall steuern.

Der Aufruf der Methode "flush" in der Schnittstelle "ObjectMap" vor einer Fest-schreibung kann weitere Überlegungen bezüglich der Sperrernreihenfolge mit sich bringen. Die Methode "flush" wird gewöhnlich verwendet, um Änderungen, die an der Map vorgenommen wurden, über das Loader-Plug-in im Back-End zu erzwingen. In dieser Situation verwendet das Back-End seinen eigenen Sperrernmanager für die Steuerung des gemeinsamen Zugriffs, so dass das Warten auf eine Sperre und die Deadlock-Situation im Back-End und nicht im Sperrernmanager von eXtreme Scale auftreten. Sehen Sie sich die folgende Transaktion an:

```
Session sess = ...;
ObjectMap person = sess.getMap("PERSON");
boolean activeTran = false;
try
{
    sess.begin();
    activeTran = true;
    Person p = (IPerson)person.get("Lynn");
    p.setAge( p.getAge() + 1 );
    person.put( "Lynn", p );
    person.flush();
    ...
    p = (IPerson)person.get("Tom");
    p.setAge( p.getAge() + 1 );
    sess.commit();    activeTran = false;
}
finally
{
    if ( activeTran ) sess.rollback();
}
```

Angenommen, eine andere Transaktion hat auch die Person "Tom" (so genannte Flush-Methode) und anschließend die Person "Lynn" aktualisiert. In dieser Situation führt die Verschachtelung der beiden Transaktionen zu einer Deadlock-Bedingung in der Datenbank.

Eine X-Sperre wird der Transaktion für "Lynn" erteilt, wenn die Flush-Operation durchgeführt wird.  
 Eine X-Sperre wird der Transaktion 2 für "Tom" erteilt, wenn die Flush-Operation durchgeführt wird.  
 Eine X-Sperre wird von Transaktion 1 für "Tom" während der Commit-Verarbeitung angefordert.  
 (Transaktion 1 wird blockiert und muss auf die von Transaktion 2 gehaltene Sperre warten.)  
 Eine X-Sperre wird von Transaktion 2 für "Lynn" während der Commit-Verarbeitung angefordert.  
 (Transaktion 2 wird blockiert und muss auf die von der Transaktion 1 gehaltene Sperre warten).

Dieses Beispiel demonstriert, dass die Verwendung der Flush-Methode eine Deadlock-Bedingung in der Datenbank und nicht in eXtreme Scale hervorrufen kann. Eine solche Deadlock-Bedingung kann bei jeder Sperrstrategie auftreten. Die Anwendung muss darauf achten, solche Deadlock-Bedingungen zu verhindern, wenn sie die Flush-Methode verwendet und wenn ein Loader in die BackingMap integriert ist. Das vorherige Beispiel veranschaulicht auch einen weiteren Grund, warum eXtreme Scale einen Wartezeitlimitmechanismus für Sperren hat. Eine Transaktion, die auf eine Datenbanksperre wartet, kann warten, während sie Eigentümer einer Sperre für einen eXtreme-Scale-Eintrag ist. Deshalb können Probleme auf Datenbankebene zu übermäßig langen Wartezeiten für einen eXtreme-Scale-Sperrmodus führen und zu einer Ausnahme des Typs "LockTimeoutException" führen.

## Häufige Deadlock-Szenarios

In den folgenden Abschnitten werden einige der häufigsten Deadlock-Szenarios beschrieben und Maßnahmen zu deren Vermeidung vorgeschlagen.

### Szenario: Deadlocks bei einem einzigen Schlüssel

In den folgenden Szenarios wird beschrieben, wie Deadlocks auftreten können, wenn auf einen einzelnen Schlüssel mit einer S-Sperre zugegriffen wird, die später aktualisiert wird. Wenn dies zwei Transaktionen gleichzeitig tun, kann ein Deadlock auftreten.

Tabelle 2. Deadlock-Szenario mit einem einzelnen Schlüssel

	Thread 1	Thread 2	
1	session.begin()	session.begin()	Jeder Thread erstellt eine unabhängige Transaktion.
2	map.get(key1)	map.get(key1)	Beiden Transaktionen wird eine S-Sperre für key1 erteilt.
3	map.update(Key1,v)		Keine U-Sperre. Die Aktualisierung wird im Transaktionscache durchgeführt.
4		map.update(key1,v)	Keine U-Sperre. Die Aktualisierung wird im Transaktionscache durchgeführt.
5	session.commit()		Blockiert: Die S-Sperre für key1 kann nicht in eine X-Sperre aktualisiert werden, weil Thread 2 eine S-Sperre hat.
6		session.commit()	Deadlock: Die S-Sperre für key1 kann nicht in eine X-Sperre aktualisiert werden, weil T1 eine S-Sperre hat.

Tabelle 3. Deadlocks mit einem einzigen Schlüssel, Fortsetzung

	Thread 1	Thread 2	
1	session.begin()	session.begin()	Jeder Thread erstellt eine unabhängige Transaktion.
2	map.get(key1)		Es wird eine S-Sperre für key1 erteilt.
3	map.getForUpdate(key1,v)		Die S-Sperre wird in eine U-Sperre für key1 aktualisiert.
4		map.get(key1)	Es wird eine S-Sperre für key1 erteilt.
5		map.getForUpdate(key1,v)	Blockiert: T1 hat bereits eine U-Sperre.
6	session.commit()		Deadlock: Die U-Sperre für key1 kann nicht aktualisiert werden.
7		session.commit()	Deadlock: Die S-Sperre für key1 kann nicht aktualisiert werden.

Tabelle 4. Deadlocks mit einem einzigen Schlüssel, Fortsetzung

	Thread 1	Thread 2	
1	session.begin()	session.begin()	Jeder Thread erstellt eine unabhängige Transaktion.
2	map.get(key1)		Es wird eine S-Sperre für key1 erteilt.
3	map.getForUpdate(key1,v)		Die S-Sperre wird in eine U-Sperre für key1 aktualisiert.
4		map.get(key1)	Es wird eine S-Sperre für key1 erteilt.
5		map.getForUpdate(key1,v)	Blockiert: Thread 1 hat bereits eine U-Sperre.
6	session.commit()		Deadlock: Die U-Sperre für key1 kann nicht in eine X-Sperre aktualisiert werden, weil Thread 2 eine S-Sperre hat.

Wenn die Methode "ObjectMap.getForUpdate" zur Vermeidung der S-Sperre verwendet wird, kann die Deadlock-Bedingung vermieden werden:

Tabelle 5. Deadlocks mit einem einzigen Schlüssel, Fortsetzung

	Thread 1	Thread 2	
1	session.begin()	session.begin()	Jeder Thread erstellt eine unabhängige Transaktion.
2	map.getForUpdate(key1)		Thread 1 wird eine U-Sperre für key1 erteilt.
3		map.getForUpdate(key1)	Die Anforderung der U-Sperre wird blockiert.
4	map.update(key1,v)	<blocked>	
5	session.commit()	<blocked>	Die U-Sperre für key1 kann erfolgreich in eine X-Sperre aktualisiert werden.
6		<released>	Die U-Sperre wird schließlich Thread 2 für key1 erteilt.



Tabelle 5. Deadlocks mit einem einzigen Schlüssel, Fortsetzung (Forts.)

	Thread 1	Thread 2	
7		map.update(key2,v)	Die U-Sperre wird Thread 2 für key2 erteilt.
8		session.commit()	Die U-Sperre für key1 kann erfolgreich in eine X-Sperre aktualisiert werden.

### Lösungen

1. Verwenden Sie die Methode "getForUpdate" an Stelle von "get", um eine U-Sperre an Stelle einer S-Sperre anzufordern.
2. Verwenden Sie die Transaktionsisoliationsstufe "read committed" (Lesen mit COMMIT), um S-Sperren zu vermeiden. Eine Verringerung der Transaktionsisoliationsstufe erhöht das Risiko nicht wiederholbarer Leseoperationen. Nicht wiederholbare Leseoperationen sind jedoch nur möglich, wenn der Transaktionscache explizit ungültig gemacht wird.
3. Verwenden Sie die optimistische Sperrstrategie. Die optimistische Sperrstrategie erfordert eine optimistische Behandlung von Kollisionsausnahmen.

### Szenario: Deadlock mit mehreren Schlüsseln unter Einhaltung der Reihenfolge

In diesem Szenario wird beschrieben, was geschieht, wenn zwei Transaktionen versuchen, denselben Eintrag direkt zu aktualisieren und S-Sperren für andere Einträge halten.

Tabelle 6. Deadlock-Szenario mit mehreren Schlüsseln unter Einhaltung der Reihenfolge

	Thread 1	Thread 2	
1	session.begin()	session.begin()	Jeder Thread erstellt eine unabhängige Transaktion.
2	map.get(key1)	map.get(key1)	Beiden Transaktionen wird eine S-Sperre für key1 erteilt.
3	map.get(key2)	map.get(key2)	Beiden Transaktionen wird eine S-Sperre für key2 erteilt.
4	map.update(key1,v)		Keine U-Sperre. Die Aktualisierung wird im Transaktionscache durchgeführt.
5		map.update(key2,v)	Keine U-Sperre. Die Aktualisierung wird im Transaktionscache durchgeführt.
6.	session.commit()		Blockiert: Die S-Sperre für key1 kann nicht in eine X-Sperre aktualisiert werden, weil Thread 2 eine S-Sperre hat.
7		session.commit()	Deadlock: Die S-Sperre für key2 kann nicht aktualisiert werden, weil Thread 1 eine S-Sperre hat.

Sie können die Methode "ObjectMap.getForUpdate" verwenden, um die S-Sperre und damit das anschließende Deadlock zu vermeiden.

Tabelle 7. Deadlock-Szenario mit mehreren Schlüsseln unter Einhaltung der Reihenfolge, Fortsetzung

	Thread 1	Thread 2	
1	session.begin()	session.begin()	Jeder Thread erstellt eine unabhängige Transaktion.

Tabelle 7. Deadlock-Szenario mit mehreren Schlüsseln unter Einhaltung der Reihenfolge, Fortsetzung (Forts.)

	Thread 1	Thread 2	
2	map.getForUpdate(key1)		Transaktion T1 wird eine U-Sperre für key1 erteilt.
3		map.getForUpdate(key1)	Die Anforderung der U-Sperre wird blockiert.
4	map.get(key2)	<blocked>	T1 wird eine S-Sperre für key2 erteilt.
5	map.update(key1,v)	<blocked>	
6	session.commit()	<blocked>	Die U-Sperre für key1 kann erfolgreich in eine X-Sperre aktualisiert werden.
7		<released>	Die U-Sperre wird schließlich T2 für key1 erteilt.
8		map.get(key2)	T2 wird eine S-Sperre für key2 erteilt.
9		map.update(key2,v)	T2 wird eine U-Sperre für key2 erteilt.
10		session.commit()	Die U-Sperre für key1 kann erfolgreich in eine X-Sperre aktualisiert werden.

### Lösungen

1. Verwenden Sie die Methode "getForUpdate" an Stelle der Methode "get", um eine U-Sperre für den ersten Schlüssel direkt anzufordern. Diese Strategie funktioniert nur, wenn die Methodenreihenfolge deterministisch ist.
2. Verwenden Sie die Transaktionsisolationsstufe "read committed" (Lesen mit COMMIT), um S-Sperren zu vermeiden. Diese Lösung ist am einfachsten zu implementieren, wenn die Methodenreihenfolge nicht deterministisch ist. Eine Verringerung der Transaktionsisolationsstufe erhöht das Risiko nicht wiederholbarer Leseoperationen. Nicht wiederholbare Leseoperationen sind jedoch nur möglich, wenn der Transaktionscache explizit ungültig gemacht wird.
3. Verwenden Sie die optimistische Sperrstrategie. Die optimistische Sperrstrategie erfordert eine optimistische Behandlung von Kollisionsausnahmen.

### Szenario: Nichteinhaltung der Reihenfolge mit U-Sperre

Wenn die Reihenfolge, in der die Schlüssel angefordert werden, nicht gewährleistet werden kann, kann ein Deadlock auftreten:

Tabelle 8. Nichteinhaltung der Reihenfolge mit U-Sperre

	Thread 1	Thread 2	
1	session.begin()	session.begin()	Jeder Thread erstellt eine unabhängige Transaktion.
2	map.getforUpdate(key1)	map.getForUpdate(key2)	Es werden erfolgreich U-Sperren für key1 und key2 erteilt.
3	map.get(key2)	map.get(key1)	Es wird eine S-Sperre für key1 und key2 erteilt.
4	map.update(key1,v)	map.update(key2,v)	
5	session.commit()		Die U-Sperre kann nicht in eine X-Sperre aktualisiert werden, weil T2 eine S-Sperre hat.

Tabelle 8. Nichteinhaltung der Reihenfolge mit U-Sperre (Forts.)

	Thread 1	Thread 2	
6		session.commit()	Die U-Sperre kann nicht in eine X-Sperre aktualisiert werden, weil T1 eine S-Sperre hat.

### Lösungen

1. Schließen Sie alle Arbeiten in eine einzige globale U-Sperre (Mutex) ein. Diese Methode verringert zwar die gemeinsamen Zugriffe, deckt aber alle Szenarios ab, in denen Zugriff und Reihenfolge nicht deterministisch ist.
2. Verwenden Sie die Transaktionsisolationsstufe "read committed" (Lesen mit COMMIT), um S-Sperren zu vermeiden. Diese Lösung ist am einfachsten zu implementieren, wenn die Methodenreihenfolge nicht deterministisch ist, und unterstützt die höchste Anzahl gemeinsamer Zugriffe. Eine Verringerung der Transaktionsisolationsstufe erhöht das Risiko nicht wiederholbarer Leseoperationen. Nicht wiederholbare Leseoperationen sind jedoch nur möglich, wenn der Transaktionscache explizit ungültig gemacht wird.
3. Verwenden Sie die optimistische Sperrstrategie. Die optimistische Sperrstrategie erfordert eine optimistische Behandlung von Kollisionsausnahmen.

### Ausnahmebehandlung in Sperrszenarios

Die vorherigen Beispiele enthalten keine Ausnahmebehandlung. Um zu verhindern, dass Sperren übermäßig lange gehalten werden, wenn eine Ausnahme des Typs "LockTimeoutException" oder "LockDeadlockException" eintritt, muss eine Anwendung sicherstellen, dass unerwartete Ausnahmen abgefangen werden und die Rollback-Methode aufgerufen wird, wenn etwas Unerwartetes eintritt. Ändern Sie das vorherige Code-Snippet, wie im folgenden Beispiel gezeigt:

```

Session sess = ...;
ObjectMap person = sess.getMap("PERSON");
boolean activeTran = false;
try
{
    sess.begin();
    activeTran = true;
    Person p = (IPerson)person.get("Lynn");
    // Lynn hat Geburtstag, also machen wir sie ein Jahr älter.
    p.setAge( p.getAge() + 1 );
    person.put( "Lynn", p );
    sess.commit();    activeTran = false;
}
finally
{
    if ( activeTran ) sess.rollback();
}

```

Der Block "finally" im Code-Snippet stellt sicher, dass eine Rollback-Operation für eine Transaktion durchgeführt wird, wenn eine unerwartete Ausnahme eintritt. Sie behandelt nicht nur Ausnahmen des Typs "LockDeadlockException", sondern auch alle anderen unerwarteten Ausnahmen, die eintreten können. Der Block "finally" behandelt Ausnahmen, die während des Aufrufs der Methode "commit" eintreten. Dieses Beispiel ist nicht einzige Möglichkeit für die Behandlung unerwarteter Ausnahmen, und es kann Fälle geben, in denen die Anwendung einige der unerwarteten Ausnahmen abfangen und eine ihrer eigenen Ausnahmen anzeigen möchte. Sie können nach Bedarf Catch-Blocks hinzufügen, aber die Anwendung muss sicherstellen, dass das Code-Snippet nicht ohne Abschluss der Transaktion beendet wird.

---

## Transaktionsisolation

Für Transaktionen können Sie in jeder BackingMap-Konfiguration eine von drei Sperrstrategien konfigurieren: PESSIMISTIC, OPTIMISTIC oder NONE. Wenn Sie pessimistisches oder optimistisches Sperren verwenden, verwendet eXtreme Scale gemeinsame (S), aktualisierbare (U) und exklusive (X) Sperren, um die Konsistenz zu verwalten. Dieses Sperrverhalten besonders beachtenswert, wenn pessimistisches Sperren verwendet wird, weil optimistische Sperren nicht gehalten werden. Sie können eine von drei Transaktionsisolationsstufen verwenden, um die Sperrsemantik zu optimieren, die eXtreme Scale für die Verwaltung der Konsistenz in den einzelnen Cache-Maps verwendet: repeatable read (wiederholbares Lesen), read committed (Lesen mit COMMIT) oder read uncommitted (Lesen ohne COMMIT).

### Übersicht über die Transaktionsisolation

Die Transaktionsisolation definiert, wie die von einer Operation vorgenommenen Änderungen für andere gleichzeitig ausgeführte Operationen sichtbar werden.

WebSphere eXtreme Scale unterstützt drei Transaktionsisolationsstufen, mit denen Sie die Sperrsemantik, die eXtreme Scale für die Verwaltung in den einzelnen Cache-Maps verwendet (repeatable read, read committed oder read uncommitted) weiter optimieren. Die Transaktionsisolationsstufe wird in der Schnittstelle "Session" mit der Methode "setTransactionIsolation" festgelegt. Die Transaktionsisolationsstufe kann jederzeit im Verlauf der Sitzung geändert werden, wenn keine Transaktion in Bearbeitung ist.

Das Produkt setzt die verschiedenen Transaktionsisolationsemantiken um, indem es die Art und Weise anpasst, in der gemeinsame Sperren (S-Sperren) angefordert und gehalten werden. Die Transaktionsisolation hat keine Auswirkungen auf Maps, die für die Verwendung der optimistischen Sperrstrategie bzw. der Strategie ohne Sperren konfiguriert sind, wenn aktualisierbare Sperren (U-Sperren) angefordert werden.

### Wiederholbares Lesen mit pessimistischem Sperren

Die Transaktionsisolationsstufe "repeatable read" (wiederholbares Lesen) ist die Standardeinstellung. Diese Isolationsstufe verhindert fehlerhafte und nicht wiederholbare Leseoperationen, aber keine Scheinleseoperationen. Eine fehlerhafte Leseoperation ist eine Leseoperation, die für Daten ausgeführt werden, die von einer Transaktion geändert, aber nicht festgeschrieben wurden. Eine nicht wiederholbare Leseoperation kann auftreten, wenn bei einer Leseoperation keine Lesesperren angefordert werden. Eine Scheinleseoperation kann auftreten, wenn zwei identische Leseoperationen durchgeführt werden, aber zwei unterschiedliche Ergebnismengen zurückgegeben werden, weil zwischen den Leseoperationen eine Aktualisierung stattgefunden hat. Das Produkt erreicht eine wiederholbare Leseoperation, indem es alle S-Sperren so lange hält, bis die Transaktion, die Eigner der Sperre ist, abgeschlossen wird. Da keine X-Sperre erteilt wird, bis alle S-Sperren freigegeben sind, sehen alle Transaktionen, die eine S-Sperre halten, bei einer Wiederholung der Leseoperation garantiert denselben Wert.

```
map = session.getMap("Order");
session.setTransactionIsolation(Session.TRANSACTION_REPEATABLE_READ);
session.begin();
```

```
// Es wird eine S-Sperre angefordert und gehalten, und der Wert wird in den
// Transaktionscache kopiert.
Order order = (Order) map.get("100");
```

```

// Der Eintrag wird aus dem Transaktionscache entfernt.
map.invalidate("100", false);

// Derselbe Wert wird erneut angefordert. Für den Eintrag ist bereits
// eine Sperre gesetzt, und deshalb wird derselbe Wert abgerufen und in
// den Transaktionscache kopiert.
Order order2 (Order) = map.get("100");

// Alle Sperren werden freigegeben, nachdem die Transaktion mit der
// Cache-Map synchronisiert wurde.
session.commit();

```

Scheinleseoperationen sind möglich, wenn Sie Abfragen oder Indizes verwenden, weil keine Sperren für Datenbereiche angefordert werden, sondern nur für die Cacheeinträge, die dem Index bzw. den Abfragekriterien entsprechen. Beispiel:

```

session1.setTransactionIsolation(Session.TRANSACTION_REPEATABLE_READ);
session1.begin();

// Es wird eine Abfrage ausgeführt, die einen Bereich von Werten auswählt.
ObjectQuery query = session1.createObjectQuery
    ("SELECT o FROM Order o WHERE o.itemName='Widget'");

// In diesem Fall stimmt nur ein einziger Auftrag (oder) mit dem
// Abfragefilter überein.
// Der Auftrag hat den Schlüssel "100".
// Die Abfragesteuerkomponente fordert eine S-Sperre für den Auftrag "100" an.
Iterator result = query.getResultIterator();

// Eine zweite Transaktion fügt einen Auftrag ein, der der Abfrage ebenfalls entspricht.
Map orderMap = session2.getMap("Order");
orderMap.insert("101", new Order("101", "Widget"));

// Wenn die Abfrage erneut in der aktuellen Transaktion ausgeführt wird,
// ist der neue Auftrag sichtbar und gibt die Aufträge "100" und "101" zurück.
result = query.getResultIterator();

// Alle Sperren werden freigegeben, nachdem die Transaktion mit der
// Cache-Map synchronisiert wurde.
session.commit();

```

## Lesen mit COMMIT mit pessimistischem Sperren

Die Transaktionsisolationssperre "read committed" (Lesen mit COMMIT) kann mit eXtreme Scale verwendet werden. Sie verhindert fehlerhafte Leseoperationen, aber keine wiederholbaren Leseoperationen oder Scheinleseoperationen, und so verwendet eXtreme Scale weiterhin S-Sperren für das Lesen von Daten aus der Cache-Map, die aber sofort wieder freigegeben werden.

```

map1 = session1.getMap("Order");
session1.setTransactionIsolation(Session.TRANSACTION_READ_COMMITTED);
session1.begin();

// Es wird eine S-Sperre angefordert, die aber sofort wieder freigegeben wird,
// und der Wert wird in den Transaktionscache kopiert.

Order order = (Order) map1.get("100");

// Der Eintrag wird aus dem Transaktionscache entfernt.
map1.invalidate("100", false);

// Eine zweite Transaktion aktualisiert denselben Auftrag (order).
// Sie fordert eine U-Sperre an, aktualisiert den Wert und wird dann festgeschrieben.
// ObjectGrid fordert während der Festschreibung erfolgreich eine X-Sperre an, da die
// erste Transaktion die Isolationsstufe "read committed" (Lesen mit COMMIT) verwendet.

```

```

Map orderMap2 = session2.getMap("Order");
session2.begin();
order2 = (Order) orderMap2.getForUpdate("100");
order2.quantity=2;
orderMap2.update("100", order2);
session2.commit();

// Derselbe Wert wird erneut angefordert. Dieses Mal soll der Wert
// aktualisiert werden, aber es wird schon der neue Wert angezeigt.
Order order1Copy (Order) = map1.getForUpdate("100");

```

## Lesen ohne COMMIT mit pessimistischem Sperren

Die Transaktionsisolationsstufe "read uncommitted" (Lesen ohne COMMIT) kann mit eXtreme Scale verwendet werden. Diese Stufe lässt fehlerhafte Leseoperationen, nicht wiederholbare Leseoperationen und Scheinleseoperationen zu.

---

## SessionHandle für Routing

Wenn Sie für jeden Container eine eigene Partitionsverteilungsrichtlinie verwenden, können Sie eine SessionHandle-Instanz verwenden. Eine SessionHandle-Instanz enthält Partitionsinformationen für das aktuelle Session-Objekt und kann für ein neues Session-Objekt wiederverwendet werden.

Eine SessionHandle-Instanz enthält Informationen für die Partition, an die das aktuelle Session-Objekt gebunden ist. Die SessionHandle-Instanz ist äußerst hilfreich für die containerbezogene Partitionsverteilungsrichtlinie und kann durch Java-Standardserialisierung serialisiert werden.

Wenn Sie eine SessionHandle-Instanz haben, können Sie diese Handle-Instanz mit der Methode "setSessionHandle(SessionHandle target)" auf ein Session-Objekt anwenden, indem Sie die Handle-Instanz als Ziel übergeben. Sie können die SessionHandle-Instanz mit der Methode "Session.getSessionHandle" abrufen.

Da die SessionHandle-Instanz nur in einem containerbezogenen Verteilungsszenario anwendbar ist, wird beim Abrufen der SessionHandle-Instanz eine Ausnahme des Typs "IllegalStateException" ausgelöst, wenn ein bestimmtes ObjectGrid mehrere MapSets pro Container oder kein MapSet hat. Wenn Sie die Methode "setSessionHandle" nicht vor der Methode "getSessionHandle" aufrufen, wird die entsprechende SessionHandle-Instanz auf der Basis der Clienteigenschaftenkonfiguration ausgewählt.

Sie können auch die Helper-Klasse "SessionHandleTransformer" verwenden, um die SessionHandle-Instanz in andere Formate zu konvertieren. Die Methoden dieser Klasse können die Darstellung einer SessionHandle-Instanz von einer Bytefeldgruppe in eine Instanzzeichenfolge, von einer Zeichenfolge in eine Instanz und jeweils umgekehrt konvertieren und außerdem den Inhalt der SessionHandle-Instanz in den Ausgabedatenstrom schreiben.

Ein Beispiel für die Verwendung eines SessionHandle finden Sie im Abschnitt zu den bevorzugten Routing-Zonen im Handbuch *Produktübersicht*.

---

## Optimistische Kollisionsausnahme

Sie können eine Ausnahme des Typs "OptimisticCollisionException" direkt oder über eine Ausnahme des Typs "ObjectGridException" empfangen.

Der folgende Code ist ein Beispiel für das Abfangen der Ausnahme und die anschließende Anzeige der entsprechenden Nachricht:

```
try {
    ...
} catch (ObjectGridException oe) {
    System.out.println(oe);
}
```

## Ursache für die Ausnahme

Es wird eine Ausnahme des Typs "OptimisticCollisionException" erstellt, wenn zwei verschiedene Clients versuchen, denselben Map-Eintrag fast zur selben Zeit zu aktualisieren. Wenn beispielsweise ein Client versucht, eine Sitzung festzuschreiben und den Map-Eintrag zu aktualisieren, nachdem ein anderer Client die Client die Daten vor dem Festschreiben liest, sind diese Daten falsch. Die Ausnahme wird erstellt, wenn der andere Client versucht, die falschen Daten festzuschreiben.

## Schlüssel abrufen, der die Ausnahme ausgelöst hat

Für die Fehlerbehebung einer solchen Ausnahme kann es hilfreich sein, den Schlüssel abzurufen, der dem Eintrag entspricht, der die Ausnahme ausgelöst hat. Der Vorteil der Ausnahme "OptimisticCollisionException" besteht darin, dass sie die Methode "getKey" enthält, die das Objekt zurückgibt, das diesen Schlüssel darstellt. Das folgende Beispiel veranschaulicht, wie der Schlüssel beim Abfangen der Ausnahme "OptimisticCollisionException" abgerufen und ausgegeben wird:

```
try {
    ...
} catch (OptimisticCollisionException oce) {
    System.out.println(oce.getKey());
}
```

## ObjectGridException löst eine Ausnahme des Typs "OptimisticCollisionException" aus

Die Ausnahme "OptimisticCollisionException" kann die Ursache für die Anzeige einer Ausnahme des Typs "ObjectGridException" sein. In diesem Fall können Sie den folgenden Code verwenden, um den Ausnahmetyp zu bestimmen und den Schlüssel auszugeben. Im folgenden Code wird das Dienstprogramm "findRootCause" gemäß der Beschreibung im folgenden Abschnitt verwendet:

```
try {
    ...
}
catch (ObjectGridException oe) {
    Throwable root = findRootCause( oe );
    if (root instanceof OptimisticCollisionException) {
        OptimisticCollisionException oce = (OptimisticCollisionException)root;
        System.out.println(oce.getKey());
    }
}
```

## Allgemeine Technik für die Behandlung von Ausnahmen

Die eigentliche (Fehler-)Ursache für ein Throwable-Objekt zu kennen, ist für die Eingrenzung der Fehlerquelle hilfreich. Das folgende Codebeispiel zeigt, wie eine Ausnahmebehandlungsroutine eine Dienstprogrammmethode verwendet, um die eigentliche (Fehler-)Ursache für das Throwable-Objekt zu ermitteln.

Beispiel:

```
static public Throwable findRootCause( Throwable t )
{
    // Mit dem Stamm-Throwable beginnen.
    Throwable root = t;

    // Ursachenkette verfolgen, bis das letzte Throwable-Objekt in der Kette gefunden wird.
    Throwable cause = root.getCause();
    while ( cause != null )
    {
        root = cause;
        cause = root.getCause();
    }

    // Letztes Throwable-Objekt in der Kette als eigentliche (Fehler-)Ursache zurückgeben.
    return root;
}
```

---

## ObjectMap-API

ObjectMaps gleichen Java-Maps, in denen Daten in Form von Schlüssel/Wert-Paaren gespeichert werden können. ObjectMaps sind eine einfache und intuitive Methode, mit der die Anwendung Daten speichern kann. Eine ObjectMap eignet sich ideal für die Zwischenspeicherung von Objekten ohne Beziehungen. Wenn Objektbeziehungen vorliegen, müssen Sie die API "EntityManager" verwenden.

Weitere Informationen zur API "EntityManager" finden Sie im Abschnitt „Anwendungsprogrammierschnittstelle "EntityManager"“ auf Seite 54.

Anwendungen fordern gewöhnlich eine eXtreme-Scale-Referenz und anschließend aus der Referenz ein Session-Objekt für jeden Thread an. Session-Objekte können nicht von mehreren Threads gemeinsam genutzt werden. Die Methode "getMap" der Schnittstelle "Session" gibt eine Referenz auf eine für den jeweiligen Thread zu verwendende ObjectMap zurück.

### Einführung in ObjectMap

Die Schnittstelle "ObjectMap" wird für transaktionsorientierte Interaktionen zwischen Anwendungen und BackingMaps verwendet.

#### Zweck

Eine ObjectMap-Instanz wird von einem Session-Objekt abgerufen, das dem aktuellen Thread entspricht. Die Schnittstelle "ObjectMap" ist das wichtigste Mittel, das Anwendungen einsetzen, um Änderungen an Einträgen in einer BackingMap vorzunehmen.

#### ObjectMap-Instanz abrufen

Eine Anwendung ruft eine ObjectMap-Instanz von einem Session-Objekt über die Methode "Session.getMap(String)" ab. Das folgende Code-Snippet veranschaulicht, wie eine ObjectMap-Instanz abgerufen wird:

```
ObjectGrid objectGrid = ...;
BackingMap backingMap = objectGrid.defineMap("mapA");
Session sess = objectGrid.getSession();
ObjectMap objectMap = sess.getMap("mapA");
```

Jede ObjectMap-Instanz entspricht einem bestimmten Session-Objekt. Wenn Sie die Methode "getMap" mehrfach für ein bestimmtes Session-Objekt mit demselben



BackingMap-Namen aufrufen, wird immer dieselbe ObjectMap-Instanz zurückgegeben.

## Transaktionen automatisch festschreiben

Operationen für BackingMaps, die ObjectMaps und JavaMaps verwenden, werden am effizientesten in einer Session-Transaktion durchgeführt. WebSphere eXtreme Scale stellt die Unterstützung für automatische Festschreibung (autocommit) bereit, wenn Methoden in den Schnittstellen "ObjectMap" und "JavaMap" außerhalb einer Session-Transaktion aufgerufen werden. Die Methoden starten eine implizite Transaktion, führen die angeforderte Operation durch und schreiben die implizite Transaktion fest.

## Methodensemantik

Im Folgenden wird die Semantik der einzelnen Methoden in den Schnittstellen "ObjectMap" und "JavaMap" erläutert. Die Methode "setDefaultKeyword", die Methode "invalidateUsingKeyword" und die Methoden, die ein Argument "Serializable" haben, werden im Abschnitt zu den Schlüsselwörtern erläutert. Die Methode "setTimeToLive" wird im Abschnitt zu den Bereinigungsprogrammen (Evictor) beschrieben. Weitere Informationen zu diesen Methoden finden Sie in der API-Dokumentation.

### Methode "containsKey"

Die Methode "containsKey" bestimmt, ob ein Schlüssel einen Wert in der BackingMap oder im Loader hat. Wenn Nullwerte von einer Anwendung unterstützt werden, kann diese Methode verwendet werden, um zu bestimmen, ob eine Nullreferenz, die von einer get-Operation zurückgegeben wird, auf einen Nullwert verweist oder anzeigt, dass die BackingMap und der Loader den Schlüssel nicht enthalten.

### Methode "flush"

Die Semantik der Methode "flush" gleicht der Semantik der Methode "flush" in der Schnittstelle "Session". Der nennenswerte Unterschied ist der, dass die Methode "flush" der Schnittstelle "Session" die aktuellen offenen Änderungen für alle Maps anwendet, die in der aktuellen Sitzung geändert werden. Diese Methode "flush" hingegen überträgt nur die Änderungen in dieser ObjectMap-Instanz an den Loader.

### Methode "get"

Die Methode "get" ruft den Eintrag aus der BackingMap-Instanz ab. Wenn der Eintrag nicht in der BackingMap-Instanz gefunden wird, der BackingMap-Instanz aber ein Loader zugeordnet ist, versucht die BackingMap-Instanz, den Eintrag vom Loader abzurufen. Die Methode "getAll" wird bereitgestellt, um den Abrufprozess im Stapelbetrieb durchzuführen.

### Methode "getForUpdate"

Die Methode "getForUpdate" entspricht der Methode "get", aber wenn Sie die Methode "getForUpdate" verwenden, teilen Sie der BackingMap und dem Loader mit, dass der Eintrag aktualisiert werden soll. Ein Loader kann diesen Hinweis verwenden, um eine SELECT- oder UPDATE-Abfrage an ein Datenbank-Back-End abzusetzen. Wenn eine pessimistische Sperrstrategie für die BackingMap definiert ist, sperrt der Sperrenmanager den Eintrag. Die Methode "getAllForUpdate" wird bereitgestellt, um den Abrufprozess im Stapelbetrieb durchzuführen.

### Methode "insert"

Die Methode "insert" fügt einen Eintrag in die BackingMap und in den Loader ein. Mit der Verwendung dieser Methode teilen Sie der BackingMap und dem Loader mit, dass Sie einen Eintrag einfügen möchten, der noch nicht vorhanden ist. Wenn Sie diese Methode für einen vorhandenen Eintrag verwenden, wird beim Aufruf der Methode bzw. beim Festschreiben der aktuellen Transaktion eine Ausnahme ausgelöst.

### Methode "invalidate"

Die Semantik der Methode "invalidate" ist von dem Wert des Parameters **isGlobal** abhängig, der an die Methode übergeben wird. Die Methode "invalidateAll" wird bereitgestellt, um den invalidate-Prozess im Stapelbetrieb durchzuführen.

Ein lokales Ungültigmachen wird angegeben, wenn der Wert "false" mit dem Parameter **isGlobal** der Methode "invalidate" übergeben wird. Beim lokalen Ungültigmachen werden alle Änderungen am Eintrag im Transaktionscache verworfen. Wenn die Anwendung eine Methode "get" absetzt, wird der Eintrag aus dem zuletzt festgeschriebenen Wert in der BackingMap abgerufen. Wenn kein Eintrag in der BackingMap vorhanden ist, wird der Eintrag aus dem zuletzt mit Flush übertragenen oder festgeschriebenen Wert im Loader abgerufen. Wenn eine Transaktion festgeschrieben wird, haben alle Einträge mit der Markierung für lokales Ungültigmachen keine Auswirkung auf die BackingMap. Alle Änderungen, die mit Flush an den Loader übertragen wurden, werden festgeschrieben, selbst wenn der Eintrag ungültig gemacht wurde.

Ein globales Ungültigmachen wird angegeben, wenn der Wert "true" mit dem Parameter **isGlobal** der Methode "invalidate" übergeben wird. Beim globalen Ungültigmachen werden alle offenen Änderungen am Eintrag im Transaktionscache verworfen, und der BackingMap-Wert wird bei nachfolgenden Operationen, die für den Eintrag durchgeführt wird, umgangen. Wenn eine Transaktion festgeschrieben wird, werden alle Einträge mit der Markierung für globales Ungültigmachen aus der BackingMap entfernt. Stellen Sie sich beispielsweise den folgenden Anwendungsfall für das Ungültigmachen vor: Die BackingMap wird durch eine Datenbanktabelle gestützt, die eine Spalte für automatische Erhöhung hat. Inkrementspalten sind hilfreich, um Datensätzen eindeutige Nummern zuzuordnen. Die Anwendung fügt einen Eintrag ein. Nach dem Einfügen muss die Anwendung die Folgenummer für die eingefügte Zeile kennen. Sie weiß, dass ihre Kopie des Objekts veraltet ist, und verwendet deshalb das globale Ungültigmachen, um den Wert vom Loader abzurufen. Der folgende Code demonstriert diesen Anwendungsfall:

```
Session sess = objectGrid.getSession();
ObjectMap map = sess.getMap("mymap");
sess.begin();
map.insert("Billy", new Person("Joe", "Bloggs", "Manhattan"));
sess.flush();
map.invalidate("Billy", true);
Person p = map.get("Billy");
System.out.println("Version column is: " + p.getVersion());
map.commit();
```

Dieser Mustercode fügt einen Eintrag für Billy hinzu. Das Attribut "version" des Person-Objekts wird über eine Spalte für automatische Erhöhung in der Datenbank gesetzt. Die Anwendung führt zunächst einen Befehl "insert" aus. Anschließend setzt sie einen Befehl "flush" ab, der bewirkt, dass der eingefügte Eintrag an den Loader und an die Datenbank gesendet wird. Die Datenbank setzt die Spalte "version" auf die nächste Nummer in

der Folge, woraufhin das Person-Objekt in der Transaktion veraltet ist. Zum Aktualisieren des Objekts wird die Anwendung global ungültig gemacht. Die nächste Methode "get", die abgesetzt wird, ruft den Eintrag vom Loader ab und ignoriert dabei den Transaktionswert. Der Eintrag wird aus der Datenbank mit dem aktualisierten Versionswert abgerufen.

#### **Methode "put"**

Die Semantik der Methode "put" ist davon abhängig, ob zuvor eine Methode "get" in der Transaktion für den Schlüssel aufgerufen wurde. Wenn die Anwendung eine Operation "get" absetzt, die einen Eintrag zurückgibt, der in der BackingMap oder im Loader vorhanden ist, wird die Methode "put" als Aktualisierung interpretiert und gibt den vorherigen Wert in der Transaktion zurück. Wenn ein Aufruf der Methode "put" ohne vorherigen Aufruf der Methode "get" durchgeführt wird oder wenn bei einem vorherigen Aufruf der Methode "get" kein Eintrag gefunden wurde, wird die Operation als Einfügeoperation interpretiert. Die Semantik der Methoden "insert" und "update" kommt zur Anwendung, wenn die Operation "put" festgeschrieben wird. Die Methode "putAll" wird zur Unterstützung von Einfüge- und Aktualisierungsoperationen im Stapelbetrieb bereitgestellt.

#### **Methode "remove"**

Die Methode "remove" entfernt den Eintrag aus der BackingMap und aus dem Loader, wenn ein Loader integriert ist. Der Wert des Objekts, das entfernt wurde, wird von dieser Methode zurückgegeben. Wenn das Objekt nicht vorhanden ist, gibt diese Methode einen Nullwert zurück. Die Methode "removeAll" wird zur Unterstützung von Löschoptionen im Stapelbetrieb ohne Rückgabewerte bereitgestellt.

#### **Methode "setCopyMode"**

Die Methode "setCopyMode" gibt einen CopyMode-Wert für diese ObjectMap an. Mit dieser Methode kann eine Anwendung den CopyMode-Wert überschreiben, der in der BackingMap definiert ist. Der angegebene CopyMode-Wert bleibt so lange wirksam, bis die Methode "clearCopyMode" aufgerufen wird. Beide Methoden können außerhalb der Transaktionsgrenzen aufgerufen werden. Ein CopyMode-Wert kann nicht mitten in einer Transaktion geändert werden.

#### **Methode "touch"**

Die Methode "touch" aktualisiert die letzte Zugriffszeit für einen Eintrag. Diese Methode ruft den Wert nicht aus der BackingMap ab. Verwenden Sie diese Methode in einer eigenen Transaktion. Wenn der bereitgestellte Schlüssel nicht in der BackingMap vorhanden ist, weil er ungültig gemacht oder entfernt wurde, wird während der Festschreibung eine Ausnahme ausgelöst.

#### **Methode "update"**

Die Methode "update" aktualisiert explizit einen Eintrag in der BackingMap und im Loader. Die Verwendung dieser Methode zeigt der BackingMap und dem Loader an, dass Sie einen vorhandenen Eintrag aktualisieren möchten. Beim Aufruf der Methode bzw. bei der Festschreibung wird eine Ausnahme ausgelöst, wenn Sie die Methode für einen Eintrag aufrufen, der nicht vorhanden ist.

#### **Methode "getIndex"**

Die Methode "getIndex" versucht, einen benannten Index abzurufen, der für die BackingMap erstellt wurde. Der Index kann nicht von mehreren Threads gemeinsam genutzt werden und funktioniert nach denselben Regeln wie ein Session-Objekt. Das zurückgegebene Indexobjekt muss in

die richtige Anwendungsindexschnittstelle, z. B. `MapIndex`, `MapRangeIndex` oder eine angepasste Indexschnittstelle, umgesetzt werden.

#### **Methode "clear"**

Die Methode "clear" entfernt alle Cacheinträge aus einer Map in allen Partitionen. Diese Operation ist eine Funktion für automatische Festschreibung. Deshalb darf keine aktive Transaktion vorhanden sein, wenn die Methode "clear" aufgerufen wird.

**Anmerkung:** Die Methode "clear" löscht nur die Map, für die sie aufgerufen wird. Alle zugehörigen Entitäts-Maps bleiben von dieser Methode unberührt. Diese Methode ruft das Loader-Plug-in nicht auf.

## **Dynamische Maps**

Sie können Maps nach der Initialisierung des Grids erstellen.

In früheren Versionen mussten Sie in eXtreme Scale die Maps vor der Initialisierung des ObjectGrids erstellen, d. h., Sie mussten alle zu verwendenden Maps erstellen, bevor Sie Transaktionen für die Maps ausgeführt haben.

### **Vorteile dynamischer Maps**

Die Einführung dynamischer Maps lockert die Einschränkung, dass alle Maps vor der Initialisierung des Grids erstellt werden müssen. Durch die Verwendung von Schablonen-Maps können Maps jetzt nach der Initialisierung des ObjectGrids erstellt werden.

Schablonen-Maps werden in der ObjectGrid-XML-Datei definiert. Es werden Schablonenvergleiche durchgeführt, wenn ein Session-Objekt eine Map anfordert, die noch nicht definiert wurde. Wenn der neue Map-Name dem regulären Ausdruck einer Schablonen-Map entspricht, wird die Map dynamisch mit dem Namen der angeforderten Map erstellt. Diese neu erstellte Map übernimmt alle Einstellungen der Schablonen-Map, die in der ObjectGrid-XML-Datei definiert sind.

### **Dynamische Maps erstellen**

Die Erstellung dynamischer Maps ist an die Methode "Session.getMap(String)" gebunden. Beim Aufruf dieser Methode wird eine ObjectMap zurückgegeben, die auf der BackingMap basiert, die in der ObjectGrid-XML-Datei konfiguriert wurde.

Die Übergabe einer Zeichenfolge, die dem regulären Ausdruck einer Schablonen-Map entspricht, führt zur Erstellung einer ObjectMap und einer zugehörigen BackingMap.

Weitere Informationen zur Methode "Session.getMap(String cacheName)" finden Sie in der API-Dokumentation.

Zum Definition einer Schablonen-Map in XML muss einfach ein boolesches Attribut "template" im Element "backingMap" festgelegt werden. Wenn Sie "template" auf "true" setzen, wird der Name der BackingMap als regulärer Ausdruck interpretiert.

WebSphere eXtreme Scale verwendet die Mustererkennung für reguläre Java-Ausdrücke. Weitere Informationen zur Steuerkomponente für reguläre Ausdrücke in Java finden Sie in der API-Dokumentation zu dem Paket "java.util.regex" und den zugehörigen Klassen.

Im Folgenden finden Sie eine ObjectGrid-XML-Beispieldatei mit einer Schablonen-Map-Definition:

```
<?xml version="1.0" encoding="UTF-8"?>
<objectGridConfig xmlns:xsi="http://www.w3.org/2001/XMLSchema-instance"
  xsi:schemaLocation="http://ibm.com/ws/objectgrid/config ../objectGrid.xsd"
  xmlns="http://ibm.com/ws/objectgrid/config">
  <objectGrids>
    <objectGrid name="accounting">
      <backingMap name="payroll" readOnly="false" />
      <backingMap name="templateMap.*" template="true"
        pluginCollectionRef="templatePlugins" lockStrategy="PESSIMISTIC" />
    </objectGrid>
  </objectGrids>

  <backingMapPluginCollections>
    <backingMapPluginCollection id="templatePlugins">
      <bean id="Evictor"
        className="com.ibm.websphere.objectgrid.plugins.builtins.LFUEvictor" />
    </backingMapPluginCollection>
  </backingMapPluginCollections>
</objectGridConfig>
```

In der vorherigen XML-Datei werden eine Schablonen-Map und eine Map definiert, die keine Schablone ist. Der Name der Schablonen-Map ist ein regulärer Ausdruck: `templateMap.*`. Wenn die Methode `Session.getMap(String)` mit einem Map-Namen aufgerufen wird, der diesem regulären Ausdruck entspricht, erstellt die Anwendung eine neue Map.

**Anmerkung:** Wenn Sie mehrere Schablonen-Maps definiert haben, müssen Sie sicherstellen, dass der Name jedes Arguments für die Methode `Session.getMap(String)` nur einer einzigen Schablonen-Map entspricht.

## Beispiel

Die Konfiguration einer Schablonen-Map ist die Voraussetzung für die Erstellung einer dynamischen Map. Fügen Sie das boolesche Attribut `template` einem Element `backingMap` in der ObjectGrid-XML-Datei hinzu.

```
<backingMap name="templateMap.*" template="true" />
```

Der Name der Schablonen-Map wird als regulärer Ausdruck behandelt.

Der Aufruf der Methode `Session.getMap(String cacheName)` mit einem cacheName-Wert, der diesem regulären Ausdruck entspricht, führt zur Erstellung der dynamischen Map. Der Methodenaufruf gibt ein ObjectMap-Objekt zurück, und das zugehörige BackingMap-Objekt wird erstellt.

```
Session session = og.getSession();
ObjectMap map = session.getMap("templateMap1");
```

Die neu erstellte Map ist mit allen Attributen und Plug-ins konfiguriert, die in der Definition der Schablonen-Map festgelegt wurden. Angenommen, die folgende Map wurde zum Definieren des ObjectGrids verwendet:

```
<?xml version="1.0" encoding="UTF-8"?>
<objectGridConfig xmlns:xsi="http://www.w3.org/2001/XMLSchema-instance"
  xsi:schemaLocation="http://ibm.com/ws/objectgrid/config ../objectGrid.xsd"
  xmlns="http://ibm.com/ws/objectgrid/config">
  <objectGrids>
    <objectGrid name="accounting">
      <backingMap name="payroll" readOnly="false" />
      <backingMap name="templateMap.*" template="true"
        pluginCollectionRef="templatePlugins" lockStrategy="PESSIMISTIC" />
    </objectGrid>
  </objectGrids>
```

```

</objectGrid>
</objectGrids>

<backingMapPluginCollections>
<backingMapPluginCollection id="templatePlugins">
  <bean id="Evictor"
    className="com.ibm.websphere.objectgrid.plugins.builtins.LFUEvictor" />
</backingMapPluginCollection>
</backingMapPluginCollections>
</objectGridConfig>

```

Für eine auf der Basis der Schablonen-Map in dieser XML-Datei erstellte dynamische Map wird ein Bereinigungsprogramm (Evictor) und die pessimistische Sperrstrategie konfiguriert.

Eine Schablone ist keine echte BackingMap, d. h. für dieses Beispiel, dass das ObjectGrid "accounting" keine echte Map "templateMap.\*" enthält. Die Schablone wird nur als Basis für die Erstellung dynamischer Maps verwendet.

Berücksichtigen Sie das geänderte Verhalten der Methode "Session.getMap(String cacheName)", wenn Schablonen-Maps verwendet werden. Vor WebSphere eXtreme Scale Version 7.0 wird bei allen Aufrufen der Methode "Session.getMap(String cacheName)" eine Ausnahme vom Typ "UndefinedMapException" ausgelöst, wenn die angeforderte Map nicht vorhanden ist. Mit dynamischen Maps wird für jeden Namen, der dem regulären Ausdruck für eine Schablonen-Map entspricht, eine Map erstellt. Notieren Sie sich die Anzahl der Maps, die Ihre Anwendung erstellt, insbesondere, wenn der verwendete reguläre Ausdruck generisch ist.

Außerdem ist ObjectGridPermission.DYNAMIC\_MAP für die Erstellung dynamischer Maps erforderlich, wenn die Sicherheit von eXtreme Scale aktiviert ist. Diese Berechtigung wird geprüft, wenn die Methode "Session.getMap(String)" aufgerufen wird. Weitere Informationen finden Sie in den Informationen zur Berechtigung von Anwendungsclicents im Handbuch *Produktübersicht*.

## Einschränkungen und Hinweise

Einschränkungen:

- Mit der API "Query" können keine dynamischen Maps verwendet werden.
- QuerySchema unterstützt das Attribut "template" für "mapName" nicht.
- Mit dynamischen Maps können keine Entitäten verwendet werden.
- Es wird implizit eine Entitäts-BackingMap definiert, die der Entität über den Klassennamen zugeordnet ist.

Hinweise:

- Viele Plug-ins haben keine Möglichkeit, die Map zu bestimmen, der sie zugeordnet sind.
- Andere Plug-ins unterscheiden sich anhand eines Map-Namens oder eines BackingMap-Namens als Argument von den anderen.

## ObjectMap und JavaMap

Eine JavaMap-Instanz wird von einem ObjectMap-Objekt abgerufen. Die Schnittstelle "JavaMap" hat dieselben Methodensignaturen wie die Schnittstelle "ObjectMap", aber mit einer anderen Ausnahmebehandlung. JavaMap erweitert die Schnittstelle "java.util.Map", so dass alle Ausnahmen Instanzen der Klasse "java.lang.RuntimeException" sind. Da JavaMap die Schnittstelle "java.util.Map" erwei-

tert, kann WebSphere eXtreme Scale ohne großen Aufwand mit einer vorhandenen Anwendung eingesetzt werden, die eine Schnittstelle "java.util.Map" für das Objekt-Caching verwendet.

## JavaMap-Instanz abrufen

Eine Anwendung ruft eine JavaMap-Instanz von einem ObjectMap-Objekt über die Methode "ObjectMap.getJavaMap" ab. Das folgende Code-Snippet veranschaulicht, wie eine JavaMap-Instanz angefordert wird.

```
ObjectGrid objectGrid = ...;
BackingMap backingMap = objectGrid.defineMap("mapA");
Session sess = objectGrid.getSession();
ObjectMap objectMap = sess.getMap("mapA");
java.util.Map map = objectMap.getJavaMap();
JavaMap javaMap = (JavaMap) javaMap;
```

Eine JavaMap-Instanz wird durch das ObjectMap-Objekt gestützt, von dem die Instanz abgerufen wurde. Wenn Sie die Methode "getJavaMap" mehrfach mit einem bestimmten ObjectMap-Objekt aufrufen, wird immer dieselbe JavaMap-Instanz zurückgegeben.

## Methoden

Die Schnittstelle "JavaMap" unterstützt nur einen Teil der Methoden in der Schnittstelle "java.util.Map". Die Schnittstelle "java.util.Map" unterstützt die folgenden Methoden:

- containsKey(java.lang.Object)**
- get(java.lang.Object)**
- put(java.lang.Object, java.lang.Object)**
- putAll(java.util.Map)**
- remove(java.lang.Object)**
- clear()**

Alle anderen Methoden, die aus der Schnittstelle "java.util.Map" übernommen werden, führen zu einer Ausnahme des Typs "java.lang.UnsupportedOperationException".

## Maps als FIFO-Warteschlangen

Mit WebSphere eXtreme Scale können Sie eine Funktion für alle Maps bereitstellen, die einer FIFO-Warteschlange (First In/First Out) gleicht. WebSphere eXtreme Scale überwacht die Einfügereihenfolge für alle Maps. Ein Client kann eine Map nach dem nächsten nicht gesperrten Eintrag in der Einfügereihenfolge abfragen und den Eintrag sperren. Dieser Prozess ermöglicht mehreren Clients, Einträge effizient aus der Map zu konsumieren.

## FIFO-Beispiel

Das folgende Code-Snippet zeigt einen Client, der in eine Schleife eintritt, um Einträge aus der Map zu verarbeiten, bis die Map abgearbeitet ist. Die Schleife startet eine Transaktion und ruft dann die Methode "ObjectMap.getNextKey(5000)" auf. Diese Methode gibt den Schlüssel des nächsten verfügbaren, nicht gesperrten Eintrags zurück und sperrt den Eintrag. Wenn die Transaktion länger als 5000 Millisekunden blockiert, gibt die Methode null zurück.

```

Session session = ...;
ObjectMap map = session.getMap("xxx");
// Zum Stoppen der Schleife muss folgende Zeile irgendwo definiert werden.
boolean timeToStop = false;

while(!timeToStop)
{
    session.begin();
    Object msgKey = map.getNextKey(5000);
    if(msgKey == null)
    {
        // Aktuelle Partition ist abgearbeitet. Erneut in einer neuen
        // Transaktion aufrufen, um mit der nächsten Partition fortzufahren.
    session.rollback();
        continue;
    }
    Message m = (Message)ma.get(msgKey);
    // Jetzt die Nachricht konsumieren.
    ...
    // Eintrag entfernen
    map.remove(msgKey);
    session.commit();
}

```

## Lokaler Modus versus Clientmodus

Wenn die Anwendung einen lokalen Kern verwendet, d. h., wenn sie kein Client ist, funktioniert der Mechanismus wie zuvor beschrieben.

Wenn die Java Virtual Machine (JVM) ein Client ist, stellt der Client zunächst eine Verbindung zu einem zufällig ausgewählten primären Shard der Partition her. Wenn keine Arbeit in dieser Partition vorhanden ist, sucht der Client in der nächsten Partition nach Arbeit. Entweder findet der Client eine Partition mit Einträgen, oder er kehrt wieder zur ersten zufällig ausgewählten Partition zurück. Wenn der Client wieder zur ersten Partition zurückkehrt, gibt er einen Nullwert an die Anwendung zurück. Findet der Client eine Map, die Einträge enthält, konsumiert er bis zum Ablauf des Zeitlimits Einträge aus dieser Map, bis keine Einträge mehr verfügbar sind. Nach Ablauf des Zeitlimits wird null zurückgegeben. Wenn null zurückgegeben und eine partitionierte Map verwendet wird, bedeutet dies, dass Sie eine neue Transaktion starten und den Empfangsvorgang fortsetzen müssen. Das vorherige Mustercodefragment hat dieses Verhalten.

## Beispiel

Wenn Sie einen Client ausführen und ein Schlüssel zurückgegeben wird, ist diese Transaktion an die Partition gebunden, die den Eintrag für diesen Schlüssel enthält. Wenn Sie in dieser Transaktion keine weiteren Maps aktualisieren möchten, ist kein Problem vorhanden. Wenn Sie weitere Maps aktualisieren möchten, können Sie nur Maps aktualisieren, die zu derselben Partition gehören wie die Map, aus der Sie den Schlüssel abgerufen haben. Der Eintrag, der von der Methode "getNextKey" zurückgegeben wird, muss der Anwendung die Möglichkeit bieten, relevante Daten in dieser Partition zu finden. Sie haben beispielsweise zwei Maps: eine für Ereignisse und eine andere für Jobs, auf die sich die Ereignisse auswirken. Sie definieren die beiden Maps mit den folgenden Entitäten:

```

Job.java
package tutorial.fifo;

import com.ibm.websphere.projector.annotations.Entity;
import com.ibm.websphere.projector.annotations.Id;

@Entity

```



```

public class Job
{
    @Id String jobId;

    int jobState;
}

JobEvent.java
package tutorial.fifo;

import com.ibm.websphere.projector.annotations.Entity;
import com.ibm.websphere.projector.annotations.Id;
import com.ibm.websphere.projector.annotations.OneToOne;

@Entity
public class JobEvent
{
    @Id String eventId;
    @OneToOne Job job;
}

```

Der Job hat eine ID und einen Status (eine ganze Zahl). Angenommen, Sie möchten den Statuswert um jeweils eins erhöhen, wenn ein Ereignis eintritt. Die Ereignisse werden in der Map "JobEvent" gespeichert. Jeder Eintrag hat eine Referenz auf den Job, den das Ereignis betrifft. Der Code für den entsprechenden Listener gleicht dem folgenden Beispiel:

```

JobEventListener.java
package tutorial.fifo;

import com.ibm.websphere.objectgrid.ObjectGridException;
import com.ibm.websphere.objectgrid.ObjectMap;
import com.ibm.websphere.objectgrid.Session;
import com.ibm.websphere.objectgrid.em.EntityManager;

public class JobEventListener
{
    boolean stopListening;

    public synchronized void stopListening()
    {
        stopListening = true;
    }

    synchronized boolean isStopped()
    {
        return stopListening;
    }

    public void processJobEvents(Session session)
        throws ObjectGridException
    {
        EntityManager em = session.getEntityManager();
        ObjectMap jobEvents = session.getMap("JobEvent");
        while(!isStopped())
        {
            em.getTransaction().begin();

            Object jobEventKey = jobEvents.getNextKey(5000);
            if(jobEventKey == null)
            {
                em.getTransaction().rollback();
                continue;
            }
            JobEvent event = (JobEvent)em.find(JobEvent.class, jobEventKey);
            // Ereignis verarbeiten. Hier wird nur der Jobstatus erhöht.

```

```

    event.job.jobState++;
    em.getTransaction().commit(); }
}
}

```

Der Listener wird von der Anwendung in einem Thread gestartet. Der Listener wird ausgeführt, bis die Methode "stopListening" aufgerufen wird. Die Methode "processJobEvents" wird im Thread ausgeführt, bis die Methode "stopListening" aufgerufen wird. Die Schleife blockiert beim Warten auf ein eventKey-Objekt aus der Map "JobEvent" und verwendet dann den EntityManager, um auf das Ereignisobjekt zuzugreifen, den Job zu dereferenzieren und den Status zu erhöhen.

Die API "EntityManager" hat keine Methode "getNextKey", wohl aber die API "ObjectMap". Deshalb verwendet der Code die API "ObjectMap" für JobEvent, um den Schlüssel abzurufen. Wenn eine Map mit Entitäten verwendet wird, werden keine Objekte mehr gespeichert. Stattdessen werden Tupel gespeichert: Ein Tupelobjekt für den Schlüssel und ein Tupelobjekt für den Wert. Die Methode "EntityManager.find" akzeptiert ein Tupel für den Schlüssel.

Der Code zum Erstellen eines Ereignisses gleicht dem folgenden Beispiel:

```

em.getTransaction().begin();
Job job = em.find(Job.class, "Job Key");
JobEvent event = new JobEvent();
event.id = Random.toString();
event.job = job;
em.persist(event); // Einfügen
em.getTransaction().commit();

```

Sie finden den Job für das Ereignis, erstellen ein Ereignis, zeigen auf den Job, fügen ihn in die Map "JobEvent" ein und schreiben dann die Transaktion fest.

## Loader und FIFO-Maps

Wenn Sie eine Map, die als FIFO-Warteschlange verwendet wird, mit einem Loader stützen möchten, kann zusätzliche Arbeit Ihrerseits erforderlich sein. Wenn die Reihenfolge der Einträge in der Map keine Rolle spielt, haben Sie keine zusätzliche Arbeit. Wenn die Reihenfolge von Bedeutung ist, müssen Sie allen eingefügten Datensätzen eine Folgenummer hinzufügen, wenn Sie die Datensätze persistent im Back-End speichern. Der Preload-Mechanismus muss so geschrieben werden, dass die Datensätze beim Starten in dieser Reihenfolge eingefügt werden.

---

## Anwendungsprogrammierschnittstelle "EntityManager"

Die Anwendungsprogrammierschnittstelle (API, Application Programming Interface) "EntityManager" vereinfacht die Interaktion mit dem eXtreme-Scale-Cache, indem sie eine einfache Methode für die Deklaration und Interaktion mit einem komplexen Graphen zusammengehöriger Objekte bereitstellt.

### Übersicht über EntityManager

Anwendungen rufen gewöhnlich zuerst eine ObjectGrid-Referenz und anschließend über diese Referenz ein Session-Objekt für jeden Thread ab. Session-Objekte können nicht von mehreren Threads gemeinsam genutzt werden. Es ist eine zusätzliche Methode im Session-Objekt, die Methode "getEntityManager". Diese Methode gibt eine Referenz auf einen EntityManager für diesen Thread zurück. Die Schnittstelle "EntityManager" kann die Schnittstellen "Session" und "ObjectMap" für alle Anwendungen ersetzen.

## EntityManager-Instanz über eine Sitzung anfordern

Die Methode "getEntityManager" ist in einem Session-Objekt verfügbar. Das folgende Codebeispiel veranschaulicht, wie eine lokale ObjectGrid-Instanz erstellt und auf den EntityManager zugegriffen wird. Einzelheiten zu allen unterstützten Methoden finden Sie in der Beschreibung der Schnittstelle "EntityManager" in der API-Dokumentation.

```
ObjectGrid og =  
ObjectGridManagerFactory.getObjectGridManager().createObjectGrid("intro-grid");  
Session s = og.getSession();  
EntityManager em = s.getEntityManager();
```

Es besteht eine Eins-zu-eins-Beziehung zwischen dem Session-Objekt und dem EntityManager-Objekt. Sie können das EntityManager-Objekt mehrfach verwenden. Codebeispiele und weitere Einzelheiten finden Sie in dem Lernprogramm zum EntityManager im Handbuch *Produktübersicht*.

## Entität persistent speichern

Eine Entität persistent zu speichern bedeutet, dass der Status einer neuen Entität in einem ObjectGrid-Cache gespeichert wird. Nach Aufruf der Methode "persist" befindet sich die Entität im Status "Verwaltet". Die Operation "persist" ist eine transaktionsorientierte Operation, und die neue Entität wird nach der Festschreibung der Transaktion im ObjectGrid-Cache gespeichert.

Jede Entität hat eine entsprechende BackingMap, in der die Tupel gespeichert werden. Die BackingMap hat denselben Namen wie die Entität und wird bei der Registrierung der Klasse erstellt. Das folgende Codebeispiel veranschaulicht, wie ein Order-Objekt mit Hilfe der Operation "persist" erstellt wird:

```
Order order = new Order(123);  
em.persist(order);  
order.setX();  
...
```

Das Order-Objekt wird mit dem Schlüssel 123 erstellt, und das Objekt wird an die Methode "persist" übergeben. Sie können mit dem Ändern des Objektstatus fortfahren, bevor Sie die Transaktion festschreiben.

**Anmerkung:** Das vorherige Beispiel enthält keine erforderlichen Transaktionsgrenzen, wie z. B. "begin" und "commit". Weitere Informationen finden Sie in dem Lernprogramm zum EntityManager im Handbuch *Produktübersicht*.

## Entität suchen

Sie können die Entität im ObjectGrid-Cache mit der Methode "find" suchen, indem Sie nach dem Speichern der Entität im Cache einen Schlüssel angeben. Diese Methode erfordert keine Transaktionsgrenzen, was für eine schreibgeschützte Semantik hilfreich ist. Das folgende Beispiel veranschaulicht, dass nur eine einzige Codezeile erforderlich ist, um die Entität zu suchen.

```
Order foundOrder = (Order)em.find(Order.class, new Integer(123));
```

## Entität entfernen

Die Methode "remove" ist wie die Methode "persist" eine transaktionsorientierte Operation. Das folgende Beispiel zeigt die Transaktionsgrenzen durch den Aufruf von begin und commit auf.

```
em.getTransaction().begin();
Order foundOrder = (Order)em.find(Order.class, new Integer(123));
em.remove(foundOrder );
em.getTransaction().commit();
```

Die Entität muss verwaltet sein, bevor sie entfernt werden kann. Sie können dies erreichen, indem Sie die Methode "find" innerhalb der Transaktionsgrenzen aufrufen. Rufen Sie anschließend die Methode "remove" in der Schnittstelle "EntityManager" auf. Weitere Informationen zu den verschiedenen Status einer Entität (z. B. neu, verwaltet, freigegeben oder entfernt) finden Sie im Abschnitt „Lebenszyklen von Entitäteninstanzen“ auf Seite 79.

## Entität ungültig machen

Die Methode "invalidate" verhält sich ähnlich wie die Methode "remove", ruft aber keine Ladeprogramm-Plug-ins auf. Verwenden Sie diese Methode, um Entitäten aus dem ObjectGrid zu entfernen, aber sie im Back-End-Datenspeicher beizubehalten.

```
em.getTransaction().begin();
Order foundOrder = (Order)em.find(Order.class, new Integer(123));
em.invalidate(foundOrder );
em.getTransaction().commit();
```

Die Entität muss verwaltet sein, bevor sie ungültig gemacht werden kann. Sie können dies erreichen, indem Sie die Methode "find" innerhalb der Transaktionsgrenzen aufrufen. Nach dem Aufruf der Methode "find" können Sie die Methode "invalidate" in der Schnittstelle "EntityManager" aufrufen. Weitere Informationen zu den verschiedenen Status einer Entität finden Sie im Abschnitt „Lebenszyklen von Entitäteninstanzen“ auf Seite 79.

## Entität aktualisieren

Die Methode "update" ist ebenfalls eine transaktionsorientierte Operation. Die Entität muss verwaltet sein, damit Aktualisierungen angewendet werden können.

```
em.getTransaction().begin();
Order foundOrder = (Order)em.find(Order.class, new Integer(123));
foundOrder.date = new Date(); // Datum des Auftrags aktualisieren
em.getTransaction().commit();
```

Im vorherigen Beispiel wird die Methode "persist" nach der Aktualisierung der Entität nicht aufgerufen. Die Entität wird im ObjectGrid-Cache aktualisiert, wenn die Transaktion festgeschrieben wird.

## Abfragen verwenden

Mit der flexiblen Abfragesteuerkomponente können Sie Entitäten über die Anwendungsprogrammierschnittstelle "EntityManager" abrufen. Erstellen Sie mit Hilfe der Abfragesprache von ObjectGrid Abfragen vom Typ SELECT über eine Entität oder ein objektbasiertes Schema. Unter "Abfrageschnittstelle" wird detailliert erläutert, wie Sie die Abfragen mit Hilfe der Anwendungsprogrammierschnittstelle "EntityManager" ausführen können. Weitere Informationen zur Verwendung von Abfragen finden Sie im Abschnitt „API "Query"“ auf Seite 85.

## Abfragewarteschlangen verwenden

Eine Entitätsabfragewarteschlange ist eine warteschlangenähnliche Datenstruktur, die einer Entitätsabfrage zugeordnet ist. Sie wählt alle Entitäten aus, die der WHE-

RE-Bedingung im Abfragefilter entsprechen, und reiht sie in eine Warteschlange ein. Anschließend können Clients interaktiv Entitäten aus dieser Warteschlange abrufen. Weitere Informationen zur Verwendung von Abfragewarteschlangen mit Entitäten finden Sie im Abschnitt „Abfragewarteschlangen für Entitäten“ auf Seite 74.

#### **Zugehörige Verweise**

API-Dokumentation: Schnittstelle EntityManager

„Schnittstelle "EntityManager"“ auf Seite 78

Sie können die Schnittstelle "EntityManager" verwenden, um Transaktionen abzugrenzen.

## **API "ObjectMap" versus API "EntityManager"**

Die meisten Cacheprodukte verwenden Map-basierte APIs, um Daten in Form von Schlüssel/Wert-Paaren zu speichern. Dieser Ansatz wird unter anderem von der API "ObjectMap" und vom dynamischen Cache in WebSphere Application Server verwendet. Die Map-basierten APIs sind zwar erfolgreich, haben aber einige Einschränkungen.

### **Einschränkungen Map-basierter APIs und der API "ObjectMap"**

Wenn Sie eine Map-basierte API verwenden, wie z. B. den dynamischen Cache in WebSphere Application Server oder die API "ObjectMap", sind die folgenden Einschränkungen zu beachten:

- Der Cache muss mit Reflexion arbeiten, um Daten aus den Objekten im Cache zu extrahieren, was zu Leistungseinbußen führt.
- Ein Cache kann nicht von zwei Anwendungen gemeinsam genutzt werden, wenn die beiden Anwendungen unterschiedliche Objekte für dieselben Daten verwenden.
- Eine Weiterentwicklung der Daten ist nicht möglich. Einem zwischengespeicherten Java-Objekt kann nicht so einfach ein Attribut hinzugefügt werden.
- Das Arbeiten mit Objektgraphen ist mühsam. Die Anwendung muss künstliche Referenzen zwischen Objekten speichern und sie manuell verknüpfen.

### **API "EntityManager"**

Die API "EntityManager" verwendet die vorhandene Map-basierte Infrastruktur, konvertiert aber Entitätsobjekte in Tupel und Tupel in Entitätsobjekte, bevor sie sie in der Map speichert bzw. aus der Map liest. Ein Entitätsobjekt wird in ein Schlüsseltupel und ein Werttupel umgesetzt, die dann als Schlüssel/Wert-Paar gespeichert werden. Ein Tupel ist ein Bereich primitiver Attribute.

Diese Gruppe von APIs vereinfacht die Verwendung von eXtreme Scale durch die POJO-Programmierung, die von den meisten Frameworks übernommen wird, erheblich.

## **Entitätsschema definieren**

Jedes eXtreme Scale kann eine unbegrenzte Anzahl logischer Entitätsschemas haben. Entitäten werden über annotierte Java-Klassen, XML oder eine Kombination von XML und Java-Klassen definiert. Definierte Entitäten werden anschließend bei einem eXtreme Scale registriert und an BackingMaps, Indizes und andere Plug-ins gebunden.

Beim Design eines Entitätsschemas müssen Sie die folgenden Tasks ausführen:

1. Entitäten und ihre Beziehungen definieren,
2. eXtreme Scale konfigurieren,
3. Entitäten bei eXtreme Scale registrieren,
4. Anwendungen erstellen, die mit eXtreme Scale interagieren, EntityManager und Entitäten erstellen.

## Konfiguration des Entitätsschemas

Ein Entitätsschema setzt sich aus einer Gruppe von Entitäten und den Beziehungen zwischen diesen Entitäten zusammen. In einem eXtreme Scale mit mehreren Partitionen gelten die folgenden Einschränkungen und Optionen für Entitätsschemas:

- Für jedes Entitätsschema muss ein einziges Stammelement definiert werden. Dieses Element wird als der Schemastamm bezeichnet.
- Alle Entitäten für ein bestimmtes Schema müssen in demselben MapSet enthalten sein, d. h., alle Entitäten, die vom Schemastamm über Beziehungen mit und ohne Schlüsselfunktion erreichbar sind, müssen in demselben MapSet wie der Schemastamm definiert sein.
- Jede Entität kann nur zu einem einzigen Entitätsschema gehören.
- Jedes eXtreme Scale kann mehrere Schemas haben.

Entitäten werden bei einem eXtreme Scale registriert, bevor sie initialisiert werden. Jede definierte Entität muss eindeutig benannt werden und wird automatisch an eine BackingMap von eXtreme Scale desselben Namens gebunden. Die Initialisierungsmethode variiert je nach verwendeter Konfiguration:

### Eigenständiges eXtreme Scale

Wenn Sie eine eigenständige eXtreme-Scale-Konfiguration verwenden, können Sie das Entitätsschema über das Programm konfigurieren. In diesem Modus können Sie die Methoden "ObjectGrid.registerEntities" verwenden, um annotierte Entitätsklassen oder eine Deskriptordatei für die Entitätsmetadaten zu registrieren.

### Verteilte eXtreme-Scale-Konfiguration

Wenn Sie eine verteilte eXtreme-Scale-Konfiguration verwenden, müssen Sie eine Deskriptordatei für die Entitätsmetadaten mit dem Entitätsschema angeben.

## Voraussetzungen für Entitätsklassen

Entitäten werden über die Zuordnung verschiedener Metadaten mit einer Java-Klasse identifiziert. Die Metadaten können mit Annotationen von Java Platform, Standard Edition 5, einer Deskriptordatei für Entitätsmetadaten oder einer Kombination von Annotationen und Deskriptordatei angegeben werden. Entitätsklassen müssen die folgenden Kriterien erfüllen:

- Sie müssen eine definierte Annotation @Entity haben oder in der XML-Deskriptordatei der Entität angegeben sein.
- Sie müssen einen öffentlichen oder geschützten Konstruktor ohne Argumente haben.
- Sie müssen eine Ausgangsklasse sein. Schnittstellen und Auflistungen (enum-Typen) sind keine gültigen Entitätsklassen.
- Sie dürfen das Schlüsselwort "final nicht verwenden.
- Sie dürfen keine Vererbung verwenden.

- Sie müssen einen eindeutigen Namen und Typ für jedes eXtreme Scale haben.

Entitäten haben alle einen eindeutigen Namen und Typ. Der Name ist bei der Verwendung von Annotationen standardmäßig der einfache (Kurz-)Name der Klasse, der jedoch mit dem Attribut "name" der Annotation `@Entity` überschrieben werden kann.

## Persistente Attribute

Der Zugriff auf den persistenten Zustand einer Entität durch Clients und Entity-Manager erfolgt über Felder (Instanzvariablen) oder Zugriffsmethoden, die mit Enterprise-JavaBeans-Eigenschaften angegeben werden. Für jede Entität muss der feld- oder eigenschaftsbasierte Zugriff definiert werden. Annotierte Entitäten sind Entitäten mit Feldzugriff, weil die Klassenfelder annotiert sind, und Entitäten mit Eigenschaftszugriff, wenn die Getter-Methode der Eigenschaft annotiert ist. Eine Mischung von Feld- und Eigenschaftszugriff ist nicht zulässig. Wenn der Typ nicht automatisch bestimmt werden kann, kann das Attribut **accessType** in der Annotation `@Entity` oder eine funktional entsprechende XML verwendet werden, um den Zugriffstyp zu identifizieren.

### Persistente Felder

Instanzvariablen für Entitäten mit Feldzugriff werden direkt über den EntityManager und die Clients aufgerufen. Felder, die mit dem Modifikator oder der Annotation "transient" gekennzeichnet sind, werden ignoriert. Persistente Felder dürfen die Modifikatoren "final" und "static" nicht enthalten.

### Persistente Eigenschaften

Entitäten mit Eigenschaftenzugriff müssen die JavaBeans™-Signaturkonventionen für Lese- und Schreiboperationen einhalten. Methoden, die die JavaBeans-Konventionen nicht einhalten, oder die Annotation "transient" in der Getter-Methode enthalten, werden ignoriert. Für eine Eigenschaft des Typs T muss eine Getter-Methode vorhanden sein (T getProperty()) und eine Setter-Methode (void setProperty(T)) vorhanden sein. Für boolesche Typen kann die Getter-Methode mit boolean isProperty() ausgedrückt werden. Persistente Eigenschaften dürfen den Modifikator "static" nicht enthalten.

### Unterstützte Attributtypen

Folgenden Typen werden für persistente Felder und Eigenschaften unterstützt:

- primitive Java-Typen, einschließlich Wrappern
- java.lang.String
- java.math.BigInteger
- java.math.BigDecimal
- java.util.Date
- java.util.Calendar
- java.sql.Date
- java.sql.Time
- java.sql.Timestamp
- byte[]
- java.lang.Byte[]
- char[]
- java.lang.Character[]

- enum

Benutzerdefinierte serialisierbare Attributtypen werden zwar unterstützt, aber für diese gelten Einschränkungen bezüglich der Leistung, Abfrage und Änderungserkennung. Persistente Daten, die nicht über einen Proxy weitergeleitet werden können, wie z. B. Feldgruppen und benutzerdefinierte serialisierbare Objekte, müssen der Entität erneut zugeordnet werden, wenn sie geändert werden.

## Entitätsassoziationen

Bidirektionale und unidirektionale Entitätsassoziationen oder Beziehungen zwischen Entitäten können als 1:1, n:1, 1:n oder n:n definiert werden. Der EntityManager löst die Entitätsbeziehungen automatisch in die entsprechenden Schlüsselreferenzen auf, wenn die Entitäten in eXtreme Scale gespeichert werden.

eXtreme Scale ist ein Datencache und erzwingt keine referenzielle Integrität wie eine Datenbank. Obwohl Beziehungen die Operationen "cascading persist" und "remove" für untergeordnete Entitäten zulassen, werden keine fehlerhaften Verknüpfungen mit Objekten erkannt oder umgesetzt. Wenn ein untergeordnetes Objekt entfernt wird, muss die Referenz auf dieses Objekt aus dem übergeordneten Objekt entfernt werden.

Wenn Sie eine bidirektionale Assoziation zwischen zwei Entitäten definieren, müssen Sie den Eigner der Beziehung angeben. In einer Viele-Beziehung (oder n-Beziehung) ist die n-Seite der Beziehung immer der Eigner. Wenn der Eigner nicht automatisch bestimmt werden kann, muss das Attribut **mappedBy** der Annotation bzw. das funktional entsprechende XML-Element angegeben werden. Das Attribut **mappedBy** gibt das Feld in der Zielentität an, das Eigner der Beziehung ist. Über diese Attribut können auch die zugehörigen Felder ermittelt werden, wenn es mehrere Attribute mit demselben Typ und derselben Kardinalität gibt.

## Einzelwertassoziationen

1:1- und N:1-Assoziationen werden mit den Annotationen "@OneToOne" bzw. "@ManyToOne" oder funktional entsprechenden XML-Attributen gekennzeichnet. Der Typ der Zielentität wird über den Attributtyp bestimmt. Im folgenden Beispiel wird eine unidirektionale Assoziation zwischen Person und Address verwendet. Die Entität "Customer" hat eine Referenz auf eine einzige Entität, die Entität "Address". In diesem Fall könnte die Assoziation auch eine n:1-Assoziation sein, das es keine Umkehrbeziehung gibt.

```
@Entity
public class Customer {
    @Id id;
    @OneToOne Address homeAddress;
}

@Entity
public class Address {
    @Id id
    @Basic String city;
}
```

Wenn Sie eine bidirektionale Beziehung zwischen den Klassen "Customer" und "Address" angeben möchten, fügen Sie der Klasse "Customer" über die Klasse "Address" eine Referenz hinzu, und fügen Sie anschließend die entsprechende Annotation hinzu, um die Gegenseite der Beziehung zu kennzeichnen. Da diese



Assoziation eine 1:1-Assoziation ist, müssen Sie einen Eigner für die Beziehung mit dem Attribut "mappedBy" in der Annotation "@OneToOne" angeben.

```
@Entity
public class Address {
    @Id id
    @Basic String city;
    @OneToOne(mappedBy="homeAddress") Customer customer;
}
```

## Collection-Assoziationen

1:n- und n:n-Assoziationen werden mit den Annotationen @OneToMany und @ManyToOne oder funktional entsprechenden XML-Attributen gekennzeichnet. Alle Viele-Beziehungen (oder n-Beziehungen) werden mit den Typen java.util.Collection, java.util.List oder java.util.Set dargestellt. Der Typ der Zielentität wird über den generischen Typ des Collection-, List- oder Set-Objekt oder explizit mit dem Attribut **targetEntity** in der Annotation @OneToMany bzw. @ManyToOne (oder den funktional entsprechenden XML-Elementen) bestimmt.

Im vorherigen Beispiel ist es nicht praktisch, ein einziges Address-Objekt pro Customer-Objekt zu haben, da es viele Kunden geben kann, die dieselbe Adresse oder mehrere Adressen haben können. Diese Situation lässt sich besser über eine Viele-Beziehung lösen:

```
@Entity
public class Customer {
    @Id id;
    @ManyToOne Address homeAddress;
    @ManyToOne Address workAddress;
}

@Entity
public class Address {
    @Id id
    @Basic String city;
    @OneToMany(mappedBy="homeAddress") Collection<Customer> homeCustomers;

    @OneToMany(mappedBy="workAddress", targetEntity=Customer.class)
    Collection workCustomers;
}
```

In diesem Beispiel existieren zwei verschiedene Beziehungen zwischen denselben Entitäten: eine Adressbeziehung "Home" und eine Adressbeziehung "Work". Es wird ein nicht generisches Collection-Objekt für das Attribut **workCustomers** verwendet, um zu veranschaulichen, wie das Attribut **targetEntity** verwendet wird, wenn kein generisches Objekt vorhanden ist.

## Primärschlüssel

Alle Entitäten müssen einen Primärschlüssel haben, der ein einfacher (Einzelattribut) oder ein Verbundschlüssel (mehrere Attribute) sein kann. Die Schlüsselattribute werden mit der Annotation "Id" gekennzeichnet oder in der XML-Deskriptordatei der Entität definiert. Für Schlüsselattribute gelten die folgenden Voraussetzungen:

- Der Wert eines Primärschlüssels darf nicht geändert werden.
- Ein Primärschlüsselattribut muss einen der folgenden Typen haben: primitiver Java-Typ oder Wrapper, java.lang.String, java.util.Date oder java.sql.Date.

- Ein Primärschlüssel kann eine beliebige Anzahl an Einzelwertassoziationen haben. Die Zielentität der Primärschlüsselassoziation darf keine direkte oder indirekte Umkehrassoziation zur Quellenentität haben.

Verbundprimärschlüssel können optional eine Primärschlüsselklasse definieren. Eine Entität wird einer Primärschlüsselklasse über die Annotation `@IdClass` oder durch Definition in der XML-Deskriptordatei der Entität zugeordnet. Eine Annotation `@IdClass` ist hilfreich, wenn sie zusammen mit der Methode "EntityManager.find" verwendet wird.

Für Primärschlüsselklassen gelten die folgenden Voraussetzungen:

- Die Primärschlüsselklasse muss öffentlich sein und einen öffentlichen Konstruktor ohne Argumente haben.
- Der Zugriffstyp der Primärschlüsselklasse wird über die Entität bestimmt, die die Primärschlüsselklasse deklariert.
- Wenn der Zugriff über Eigenschaften erfolgt, müssen die Eigenschaften der Primärschlüsselklasse öffentlich oder geschützt sein.
- Die Primärschlüsselfelder und -eigenschaften müssen den Schlüsselattributnamen und -typen entsprechen, die in der referenzierenden Entität definiert sind.
- Primärschlüsselklassen müssen die Methoden "equals" und "hashCode" implementieren.

```
@Entity
@IdClass(CustomerKey.class)
public class Customer {
    @Id @ManyToOne Zone zone;
    @Id int custId;
    String name;
    ...
}

@Entity
public class Zone{
    @Id String zoneCode;
    String name;
}

public class CustomerKey {
    Zone zone;
    int custId;

    public int hashCode() {...}
    public boolean equals(Object o) {...}
}
```

## Entitäts-Proxys und Feld-Interceptor

Entitätsklassen und veränderliche unterstützte Attributtypen werden durch Proxy-Klassen für Entitäten mit Eigenschaftenzugriff und Bytecode für Entitäten mit Feldzugriff in Java Development Kit (JDK) 5 erweitert. Alle Zugriffe auf die Entität, selbst Zugriffe durch interne Geschäftsmethoden und die equals-Methoden, müssen die entsprechenden Feld- bzw. Eigenschaftenzugriffsmethoden verwenden.

Proxys und Feld-Interceptor werden verwendet, um dem EntityManager zu ermöglichen, den Status der Entität zu verfolgen, zu bestimmen, ob sich die Entität geändert hat und die Leistung zu verbessern. Feld-Interceptor sind nur in Plattformen des Typs Java SE 5 verfügbar, wenn der Instrumentierungsagent für Entitäten konfiguriert ist.

**Achtung:** Wenn Sie Entitäten mit Eigenschaftenzugriff verwenden, muss die Methode "equals" den Operator "instanceof" verwenden, um die aktuelle Instanz mit dem Eingabeobjekt zu vergleichen. Die gesamte Introspektion des Zielobjekts muss über die Eigenschaften des Objekts und nicht die Felder selbst erfolgen, da die Objektinstanz der Proxy ist.

## Datei emd.xsd

Verwenden Sie die XML-Schemadefinition für Entitätsmetadaten, um eine XML-Deskriptordatei zu erstellen und ein Entitätsschema für WebSphere eXtreme Scale zu definieren.

Beschreibungen der einzelnen Elemente und Attribute in der Datei emd.xsd finden Sie im Abschnitt zur Deskriptordatei für Entitätsmetadaten im *Administratorhandbuch*.

## Datei emd.xsd

```
<?xml version="1.0" encoding="UTF-8"?>
<xsd:schema xmlns:emd="http://ibm.com/ws/projector/config/emd"
  xmlns:xsd="http://www.w3.org/2001/XMLSchema"
  targetNamespace="http://ibm.com/ws/projector/config/emd"
  elementFormDefault="qualified" attributeFormDefault="unqualified"
  version="1.0">

  <xsd:element name="entity-mappings"
    <xsd:complexType>
      <xsd:sequence>
        <xsd:element name="description" type="xsd:string" minOccurs="0"/>
        <xsd:element name="entity" type="emd:entity" minOccurs="1" maxOccurs="unbounded"/>
      </xsd:sequence>
    </xsd:complexType>
    <xsd:unique name="uniqueEntityClassName">
      <xsd:selector xpath="emd:entity"/>
      <xsd:field xpath="@class-name"/>
    </xsd:unique>
  </xsd:element>

  <xsd:complexType name="entity">
    <xsd:sequence>
      <xsd:element name="description" type="xsd:string" minOccurs="0"/>
      <xsd:element name="id-class" type="emd:id-class" minOccurs="0"/>
      <xsd:element name="attributes" type="emd:attributes" minOccurs="0"/>
      <xsd:element name="entity-listeners" type="emd:entity-listeners" minOccurs="0"/>
      <xsd:element name="pre-persist" type="emd:pre-persist" minOccurs="0"/>
      <xsd:element name="post-persist" type="emd:post-persist" minOccurs="0"/>
      <xsd:element name="pre-remove" type="emd:pre-remove" minOccurs="0"/>
      <xsd:element name="post-remove" type="emd:post-remove" minOccurs="0"/>
      <xsd:element name="pre-invalidate" type="emd:pre-invalidate" minOccurs="0"/>
      <xsd:element name="post-invalidate" type="emd:post-invalidate" minOccurs="0"/>
      <xsd:element name="pre-update" type="emd:pre-update" minOccurs="0"/>
      <xsd:element name="post-update" type="emd:post-update" minOccurs="0"/>
      <xsd:element name="post-load" type="emd:post-load" minOccurs="0"/>
    </xsd:sequence>
    <xsd:attribute name="name" type="xsd:string" use="required"/>
    <xsd:attribute name="class-name" type="xsd:string" use="required"/>
    <xsd:attribute name="access" type="emd:access-type"/>
    <xsd:attribute name="schemaRoot" type="xsd:boolean"/>
  </xsd:complexType>

  <xsd:complexType name="attributes">
    <xsd:sequence>
      <xsd:choice>
        <xsd:element name="id" type="emd:id" minOccurs="0" maxOccurs="unbounded"/>
      </xsd:choice>
      <xsd:element name="basic" type="emd:basic" minOccurs="0" maxOccurs="unbounded"/>
      <xsd:element name="version" type="emd:version" minOccurs="0" maxOccurs="unbounded"/>
      <xsd:element name="many-to-one" type="emd:many-to-one" minOccurs="0" maxOccurs="unbounded"/>
      <xsd:element name="one-to-many" type="emd:one-to-many" minOccurs="0" maxOccurs="unbounded"/>
      <xsd:element name="one-to-one" type="emd:one-to-one" minOccurs="0" maxOccurs="unbounded"/>
      <xsd:element name="many-to-many" type="emd:many-to-many" minOccurs="0" maxOccurs="unbounded"/>
      <xsd:element name="transient" type="emd:transient" minOccurs="0" maxOccurs="unbounded"/>
    </xsd:sequence>
  </xsd:complexType>

  <xsd:simpleType name="access-type">
    <xsd:restriction base="xsd:token">
      <xsd:enumeration value="PROPERTY"/>
      <xsd:enumeration value="FIELD"/>
    </xsd:restriction>
  </xsd:simpleType>

  <xsd:complexType name="id-class">
```

```

<xsd:attribute name="class-name" type="xsd:string" use="required"/>
</xsd:complexType>

<xsd:complexType name="id">
<xsd:attribute name="name" type="xsd:string" use="required"/>
<xsd:attribute name="alias" type="xsd:string" use="optional"/>
</xsd:complexType>

<xsd:complexType name="transient">
<xsd:attribute name="name" type="xsd:string" use="required"/>
</xsd:complexType>

<xsd:complexType name="basic">
<xsd:attribute name="name" type="xsd:string" use="required"/>
<xsd:attribute name="alias" type="xsd:string"/>
<xsd:attribute name="fetch" type="emd:fetch-type"/>
</xsd:complexType>

<xsd:simpleType name="fetch-type">
<xsd:restriction base="xsd:token">
<xsd:enumeration value="LAZY"/>
<xsd:enumeration value="EAGER"/>
</xsd:restriction>
</xsd:simpleType>

<xsd:complexType name="many-to-one">
<xsd:sequence>
<xsd:element name="cascade" type="emd:cascade-type" minOccurs="0"/>
</xsd:sequence>
<xsd:attribute name="name" type="xsd:string" use="required"/>
<xsd:attribute name="alias" type="xsd:string"/>
<xsd:attribute name="target-entity" type="xsd:string"/>
<xsd:attribute name="fetch" type="emd:fetch-type"/>
<xsd:attribute name="id" type="xsd:boolean"/>
</xsd:complexType>

<xsd:complexType name="one-to-one">
<xsd:sequence>
<xsd:element name="cascade" type="emd:cascade-type" minOccurs="0"/>
</xsd:sequence>
<xsd:attribute name="name" type="xsd:string" use="required"/>
<xsd:attribute name="alias" type="xsd:string"/>
<xsd:attribute name="target-entity" type="xsd:string"/>
<xsd:attribute name="fetch" type="emd:fetch-type"/>
<xsd:attribute name="mapped-by" type="xsd:string"/>
<xsd:attribute name="id" type="xsd:boolean"/>
</xsd:complexType>

<xsd:complexType name="one-to-many">
<xsd:sequence>
<xsd:element name="order-by" type="emd:order-by" minOccurs="0"/>
<xsd:element name="cascade" type="emd:cascade-type" minOccurs="0"/>
</xsd:sequence>
<xsd:attribute name="name" type="xsd:string" use="required"/>
<xsd:attribute name="alias" type="xsd:string"/>
<xsd:attribute name="target-entity" type="xsd:string"/>
<xsd:attribute name="fetch" type="emd:fetch-type"/>
<xsd:attribute name="mapped-by" type="xsd:string"/>
</xsd:complexType>

<xsd:complexType name="many-to-many">
<xsd:sequence>
<xsd:element name="order-by" type="emd:order-by" minOccurs="0"/>
<xsd:element name="cascade" type="emd:cascade-type" minOccurs="0"/>
</xsd:sequence>
<xsd:attribute name="name" type="xsd:string" use="required"/>
<xsd:attribute name="alias" type="xsd:string"/>
<xsd:attribute name="target-entity" type="xsd:string"/>
<xsd:attribute name="fetch" type="emd:fetch-type"/>
<xsd:attribute name="mapped-by" type="xsd:string"/>
</xsd:complexType>

<xsd:simpleType name="order-by">
<xsd:restriction base="xsd:string"/>
</xsd:simpleType>

<xsd:complexType name="cascade-type">
<xsd:sequence>
<xsd:element name="cascade-all" type="emd:emptyType" minOccurs="0"/>
<xsd:element name="cascade-persist" type="emd:emptyType" minOccurs="0"/>
<xsd:element name="cascade-remove" type="emd:emptyType" minOccurs="0"/>
<xsd:element name="cascade-invalidate" type="emd:emptyType" minOccurs="0"/>
<xsd:element name="cascade-merge" type="emd:emptyType" minOccurs="0"/>
<xsd:element name="cascade-refresh" type="emd:emptyType" minOccurs="0"/>
</xsd:sequence>
</xsd:complexType>

<xsd:complexType name="emptyType"/>

<xsd:complexType name="version">
<xsd:attribute name="name" type="xsd:string" use="required"/>

```

```

</xsd:complexType>

<xsd:complexType name="entity-listeners">
  <xsd:sequence>
    <xsd:element name="entity-listener" type="emd:entity-listener" minOccurs="0" maxOccurs="unbounded"/>
  </xsd:sequence>
</xsd:complexType>

<xsd:complexType name="entity-listener">
  <xsd:sequence>
    <xsd:element name="pre-persist" type="emd:pre-persist" minOccurs="0"/>
    <xsd:element name="post-persist" type="emd:post-persist" minOccurs="0"/>
    <xsd:element name="pre-remove" type="emd:pre-remove" minOccurs="0"/>
    <xsd:element name="post-remove" type="emd:post-remove" minOccurs="0"/>
    <xsd:element name="pre-invalidate" type="emd:pre-invalidate" minOccurs="0"/>
    <xsd:element name="post-invalidate" type="emd:post-invalidate" minOccurs="0"/>
    <xsd:element name="pre-update" type="emd:pre-update" minOccurs="0"/>
    <xsd:element name="post-update" type="emd:post-update" minOccurs="0"/>
    <xsd:element name="post-load" type="emd:post-load" minOccurs="0"/>
  </xsd:sequence>
  <xsd:attribute name="class-name" type="xsd:string" use="required"/>
</xsd:complexType>

<xsd:complexType name="pre-persist">
  <xsd:attribute name="method-name" type="xsd:string" use="required"/>
</xsd:complexType>

<xsd:complexType name="post-persist">
  <xsd:attribute name="method-name" type="xsd:string" use="required"/>
</xsd:complexType>

<xsd:complexType name="pre-remove">
  <xsd:attribute name="method-name" type="xsd:string" use="required"/>
</xsd:complexType>

<xsd:complexType name="post-remove">
  <xsd:attribute name="method-name" type="xsd:string" use="required"/>
</xsd:complexType>

<xsd:complexType name="pre-invalidate">
  <xsd:attribute name="method-name" type="xsd:string" use="required"/>
</xsd:complexType>

<xsd:complexType name="post-invalidate">
  <xsd:attribute name="method-name" type="xsd:string" use="required"/>
</xsd:complexType>

<xsd:complexType name="pre-update">
  <xsd:attribute name="method-name" type="xsd:string" use="required"/>
</xsd:complexType>

<xsd:complexType name="post-update">
  <xsd:attribute name="method-name" type="xsd:string" use="required"/>
</xsd:complexType>

<xsd:complexType name="post-load">
  <xsd:attribute name="method-name" type="xsd:string" use="required"/>
</xsd:complexType>

</xsd:schema>

```

## Integrierte Server-API

WebSphere eXtreme Scale stellt mehrere Features bereit, die programmgesteuert mit der Programmiersprache Java über mehrere Anwendungsprogrammierschnittstellen (API, Application Programming Interfaces) und Systemprogrammierschnittstellen (SPI, System Programming Interfaces) aufgerufen werden.

### ServerFactor.getInstance

Setzen Sie die Methode "ServerFactory.getInstance" ab, so dass die Laufzeitumgebung des eXtreme-Scale-Servers in einer beliebigen JVM ausgeführt werden kann. Diese Methode gibt das Server-Singleton zurück und startet die Laufzeitumgebung des Servers. Sie müssen die Servereigenschaften festlegen, bevor Sie die Methode "getInstance" absetzen. Andernfalls haben alle Änderungen, die Sie nach dem Absetzen der Methode vornehmen, keine Auswirkung auf die Laufzeitumgebung.

## eXtreme-Scale-Server instanzieren

```
Server server = ServerFactory.getInstance();
```

Sie können verschiedene Eigenschaften verwenden, um die eXtreme-Scale-Serverinstanz zu konfigurieren, die Sie über die Methode "ServerFactory.getServerProperties" abrufen können. Das ServerProperties-Objekt ist das Singleton, und deshalb ruft jeder Aufruf der Methode "getServerProperties" dieselbe Instanz ab. Alle Eigenschaften, die beim ersten Aufruf von "getInstance" gesetzt werden, werden zum Initialisieren des Servers verwendet.

## Servereigenschaften abrufen

Sie können die Servereigenschaften festlegen, bis die Methode "ServerFactory.getInstance" zum ersten Mal aufgerufen wird. Der erste Aufruf der Methode "getInstance" instanziert den eXtreme-Scale-Server und liest alle konfigurierten Eigenschaften. Das Festlegen von Eigenschaften nach der Erstellung der Instanz hat keine Auswirkung.

## Eigenschaften für den integrierten eXtreme-Scale-Server festlegen

```
// Servereigenschaften abrufen, die diesem Prozess zugeordnet sind.
ServerProperties serverProperties = ServerFactory.getServerProperties();

// Servernamen für diesen Prozess festlegen.
serverProperties.setServerName("EmbeddedServerA");

// Namen der Zone festlegen, in der sich dieser Prozess befindet.
serverProperties.setZoneName("EmbeddedZone1");

// Erforderliche Endpunktinformationen zum Booten des Katalogservice festlegen.
serverProperties.setCatalogServiceBootstrap("localhost:2809");

// Hostnamen des ORB-Listeners festlegen, zu dem die Bindung hergestellt werden soll.
serverProperties.setListenerHost("host.local.domain");

// ORB-Listener-Port festlegen, zu dem die Bindung hergestellt werden soll.
serverProperties.setListenerPort(9010);

// Alle MBeans für diesen Prozess inaktivieren.
serverProperties.setMBeansEnabled(false);

Server server = ServerFactory.getInstance();
```

## Integrierter Katalogservice

Jede JVM, die von der Methode "CatalogServerProperties.setCatalogServer" markiert wird, kann den Katalogservice für eXtreme Scale ausführen. Diese Methode weist die Laufzeitumgebung des eXtreme-Scale-Servers an, den Katalogservice zu instanzieren, wenn der Server gestartet wird. Der folgende Code veranschaulicht, wie der Katalogserver von eXtreme Scale instanziiert wird:

```
CatalogServerProperties catalogServerProperties =
    ServerFactory.getCatalogProperties();
catalogServerProperties.setCatalogServer(true);

Server server = ServerFactory.getInstance();
```

## eXtreme-Scale-Container integrieren

Setzen Sie die Methode "Server.createContainer" für jede ab, die mehrere eXtreme-Scale-Container ausführen soll. Der folgende Code veranschaulicht, wie ein eXtreme-Scale-Container instanziiert wird:

```
Server server = ServerFactory.getInstance();
DeploymentPolicy policy = DeploymentPolicyFactory.createDeploymentPolicy(
    new File("META-INF/embeddedDeploymentPolicy.xml").toURL(),
    new File("META-INF/embeddedObjectGrid.xml").toURL());
Container container = server.createContainer(policy);
```

## Eigenständiger Serverprozess

Sie können alle Services gemeinsam starten, was für die Entwicklung hilfreich und auch in einer Produktionsumgebung praktisch ist. Wenn die Services gemeinsam gestartet werden, führt ein einziger Prozess alle folgenden Aufgaben aus: Er startet den Katalogservice, er startet eine Gruppe von Containern und er führt die Clientverbindungslogik aus. Mit dieser Art des Servicestarts können Programmierungsprobleme festgestellt werden, bevor die Services in einer verteilten Umgebung implementiert werden. Der folgende Code veranschaulicht, wie ein eigenständiger eXtreme-Scale-Server instanziiert wird:

```
CatalogServerProperties catalogServerProperties =
    ServerFactory.getCatalogProperties();
catalogServerProperties.setCatalogServer(true);

Server server = ServerFactory.getInstance();
DeploymentPolicy policy = DeploymentPolicyFactory.createDeploymentPolicy(
    new File("META-INF/embeddedDeploymentPolicy.xml").toURL(),
    new File("META-INF/embeddedObjectGrid.xml").toURL());
Container container = server.createContainer(policy);
```

## eXtreme Scale in WebSphere Application Server integrieren

Die Konfiguration für eXtreme Scale wird automatisch vorgenommen, wenn Sie WebSphere Extended Deployment DataGrid in einer Umgebung mit WebSphere Application Server installieren. Sie müssen keine Eigenschaften festlegen, bevor Sie auf den Server zugreifen, um einen Container zu erstellen. Der folgende Code veranschaulicht, wie Sie einen eXtreme-Scale-Server in WebSphere Application Server instanziiieren:

```
Server server = ServerFactory.getInstance();
DeploymentPolicy policy = DeploymentPolicyFactory.createDeploymentPolicy(
    new File("META-INF/embeddedDeploymentPolicy.xml").toURL(),
    new File("META-INF/embeddedObjectGrid.xml").toURL());
Container container = server.createContainer(policy);
```

## Leistungseinfluss der Schnittstelle "EntityManager"

Die Schnittstelle "EntityManager" grenzt Anwendungen vom Zustand im Grid ab. Dies ist ein wichtiger Faktor, wenn Sie das Grid nicht mit Anwendungen koppeln möchten. Stellen Sie sich beispielsweise eine Anwendung vor, die jede Anwendung zwingt, dieselben Datenzugriffsobjekte für eine Datenbank zu verwenden. Diese Implementierung ist nicht praktisch. Der hohe Bedienungskomfort geht jedoch zu Lasten der Leistung.

Die Kosten für die Verwendung der Schnittstelle "EntityManager" sind nicht sehr hoch und richten sich nach dem Typ der durchgeführten Arbeiten. Verwenden Sie immer die Schnittstelle "EntityManager", und optimieren Sie die entscheidende Geschäftslogik nach Abschluss der Anwendung. Sie können jeden Code, der EntityManager-Schnittstellen verwendet, so nachbearbeiten, dass er Maps und Tupel

verwendet. Diese Nachbearbeitung des Codes kann für 10 % des Codes erforderlich sein, wenn man die normale 90/10-Regel anwendet, mit der berechnet wird, wie viel Code wirklich kritisch für die Leistung ist.

Wenn Sie Beziehungen zwischen Objekten verwenden, ist der Leistungseinfluss geringer, weil eine Anwendung, die Maps verwendet, diese Beziehungen ähnlich wie die Schnittstelle "EntityManager" verwalten muss.

Anwendungen, die die Schnittstelle "EntityManager" verwenden, müssen keinen ObjectTransformer bereitstellen, weil die Objektumsetzung automatisch optimiert wird.

## Beispiel: EntityManager-Code für die Verwendung von Maps nachbearbeiten

Es folgt eine Musterentität:

```
@Entity
public class Person
{
    @Id String ssn;
    String firstName;
    @Index
    String middleName;
    String surname;
}
```

Im Folgenden sehen Sie Mustercode, mit dem die Entität gesucht und aktualisiert wird:

```
Person p = null;
s.begin();
p = (Person)em.find(Person.class, "1234567890");
p.middleName = String.valueOf(inner);
s.commit();
```

Im Folgenden sehen Sie denselben Code mit Maps und Tupeln:

```
Tuple key = null;
key = map.getEntityMetadata().getKeyMetadata().createTuple();
key.setAttribute(0, "1234567890");
```

```
// Der Kopiermodus ist für Entitäts-Maps immer NO_COPY, sofern nicht COPY_TO_BYTES verwendet wird.
// Das Tupel muss entweder kopiert werden, oder das ObjectGrid muss dazu aufgefordert werden.
map.setCopyMode(CopyMode.COPY_ON_READ);
s.begin();
Tuple value = (Tuple)map.get(key);
value.setAttribute(1, String.valueOf(inner));
map.update(key, value);
value = null;
s.commit();
```

Beide Code-Snippets haben dasselbe Ergebnis, und eine Anwendung kann beide Code-Snippets gleichzeitig verwenden. Das zweite Code-Snippet zeigt, wie Maps direkt verwendet werden und wie mit den Tupeln gearbeitet wird. Die Schlüssel/Wert-Paare sind Tupel. Das Werttupel (value) hat drei Attribute: firstName, middleName und surname, indexiert mit 0, 1 und 2. Das Schlüsseltupel (key) hat ein einziges Attribut, die ID-Nummer, indexiert mit null. Sie können sehen, wie Tupel mit der Methode "EntityManager#getKeyMetadata" oder "EntityManager#getValueMetadata" erstellt werden. Sie müssen diese Methoden verwenden, um Tupel für eine Entität zu erstellen. Sie können die Schnittstelle



"Tuple" nicht implementieren, und Sie können keine Instanz Ihrer Tuple-Implementierung übergeben.

## Instrumentierungsagent

Die Leistung von Entitäten mit Feldzugriff kann durch Aktivierung des Agenten von WebSphere eXtreme Scale verbessert werden, wenn Java Development Kit (JDK) Version 1.5 oder höher verwendet wird. Der Agent wird nur in einer lokalen eXtreme-Scale-Umgebung oder in einem eXtreme-Scale-Client in einer verteilten eXtreme-Scale-Umgebung verwendet.

## Agenten von eXtreme Scale in JDK Version 1.5 oder höher aktivieren

Der ObjectGrid-Agent kann über eine Java-Befehlszeilenoption mit der folgenden Syntax aktiviert werden:

```
-javaagent:jarpath[=Optionen]
```

In dieser Syntax steht *jarpath* für den Pfad zu einer JAR-Datei (Java-Archiv) der eXtreme-Scale-Laufzeitumgebung, die eine eXtreme-Scale-Agentenklasse und unterstützende Klassen enthält, wie z. B. `objectgrid.jar`, `wobjectgrid.jar`, `ogclient.jar`, `wsogclient.jar` oder `ogagent.jar`. In einem eigenständigen Java-Programm oder in einer Java-EE-Umgebung (Java Platform, Enterprise Edition), in der WebSphere Application Server nicht ausgeführt wird, verwenden Sie gewöhnlich die Datei `objectgrid.jar` oder die Datei `ogclient.jar`. In einer Umgebung mit WebSphere Application Server oder mehreren Klassenladeprogrammen müssen Sie die Datei `ogagent.jar` in der Java-Befehlszeilenoption für den Agenten angeben.

Zusätzlich zur JAR-Datei für den Agenten muss die Datei `cglib.jar` in den Bootstrap-Klassenpfad eingefügt werden. Obwohl in WebSphere Application Server die Datei `cglib.jar` unter Umständen bereits im Verzeichnis "lib" und im Klassenpfad des Anwendungsservers enthalten ist, müssen Sie die Datei `cglib.jar` in der Eigenschaft "class" der Java Virtual Machine (JVM) angeben, die dem Anwendungsserver zugeordnet ist.

Geben Sie die Datei `ogagent.config` im Klassenpfad an, oder verwenden Sie die Agentenoptionen, um weitere Informationen anzugeben.

## Optionen für den Agenten von eXtreme Scale

**config** Überschreibt den Namen der Konfigurationsdatei.

### include

Gibt die Definition der Umsetzungsdomäne an bzw. überschreibt sie. Diese Definition ist der erste Teil der Konfigurationsdatei.

### exclude

Gibt die @Exclude-Definition an bzw. überschreibt sie.

### fieldAccessEntity

Gibt die @FieldAccessEntity-Definition an bzw. überschreibt sie.

**trace** Gibt eine Trace-Stufe an. Die gültigen Stufen sind ALL, CONFIG, FINE, FINER, FINEST, SEVERE, WARNING, INFO und OFF.

### trace.file

Gibt die Position der Trace-Datei an.

Das Semikolon (;) wird als Trennzeichen zwischen den Optionen verwendet. Das Komma (,) wird als Trennzeichen zwischen den Elementen in einer Option verwendet. Das folgende Beispiel demonstriert die eXtreme-Scale-Agentenoption für ein Java-Programm:

```
-javaagent:objectgridRoot/lib/objectgrid.jar=config=myConfigFile;  
include=includedPackage;exclude=excludedPackage;  
fieldAccessEntity=package1,package2
```

## Datei ogagent.config

Die Datei ogagent.config ist der vordefinierte Name für die Konfigurationsdatei des eXtreme-Scale-Agenten. Wenn der Dateiname im Klassenpfad enthalten ist, findet der eXtreme-Scale-Agent die Datei und analysiert sie syntaktisch. Sie können den vordefinierten Dateinamen mit der Option "config" des eXtreme-Scale-Agenten überschreiben. Das folgende Beispiel veranschaulicht, wie die Konfigurationsdatei angegeben wird:

```
-javaagent:objectgridRoot/lib/objectgrid.jar=config=myOverrideConfigFile
```

Die Konfigurationsdatei eines eXtreme-Scale-Agenten setzt sich aus den folgenden Teilen zusammen:

- **Umsetzungsdomäne:** Der Teil für die Umsetzungsdomäne ist der erste Teil in der Konfigurationsdatei. Die Umsetzungsdomäne ist eine Liste mit Paketen und Klassen, die am Klassenumsetzungsprozess beteiligt sind. Diese Umsetzungsdomäne muss alle Klassen enthalten, die Entitätsklassen mit Feldzugriff sind, und alle Klassen, die auf diese Entitätsklassen mit Feldzugriff verweisen. Entitätsklassen mit Feldzugriff und die Klassen, die auf diese Entitätsklassen mit Feldzugriff verweisen, bilden die Umsetzungsdomäne. Wenn Sie Entitätsklassen mit Feldzugriff im Teil "@FieldAccessEntity" angeben möchten, müssen Sie die Entitätsklassen mit Feldzugriff hier nicht angeben. Die Umsetzungsdomäne muss vollständig sein. Andernfalls wird eine Ausnahme des Typs "FieldAccessEntityNotInstrumentedException" ausgelöst.
- **@Exclude:** Das Token "@Exclude" zeigt an, dass Pakete und Klassen, die hinter diesem Token aufgelistet werden, aus der Umsetzungsdomäne ausgeschlossen werden.
- **@FieldAccessEntity:** Das Token "@FieldAccessEntity" zeigt an, dass Pakete und Klassen, die hinter diesem Token aufgelistet werden, Entitätspakete und -klassen mit Feldzugriff sind. Wenn dem Token "@FieldAccessEntity" keine Zeile folgt, ist das dasselbe, als würden Sie @FieldAccessEntity nicht angeben. Der eXtreme-Scale-Agent bestimmt, dass keine Entitätspakete und -klassen mit Feldzugriff definiert sind. Wenn dem Token "@FieldAccessEntity" Zeilen folgen, stellen diese die benutzerdefinierten Entitätspakete und -klassen dar, z. B. "Entitätsdomäne mit Feldzugriff". Die Entitätsdomäne mit Feldzugriff ist eine Unterdomäne der Umsetzungsdomäne. Pakete und Klassen, die in der Entitätsdomäne mit Feldzugriff aufgelistet sind, gehören zur Umsetzungsdomäne, selbst wenn sie nicht in der Umsetzungsdomäne aufgelistet sind. Das Token "@Exclude", das Pakete und Klassen auflistet, die von der Umsetzung ausgeschlossen werden, hat keine Auswirkung auf die Entitätsdomäne mit Feldzugriff. Wenn Sie das Token "@FieldAccessEntity" angeben, müssen alle Entitäten mit Feldzugriff in dieser Entitätsdomäne mit Feldzugriff enthalten sein. Andernfalls wird eine Ausnahme des Typs "FieldAccessEntityNotInstrumentedException" ausgelöst.

## Beispiel für eine Agentenkonfigurationsdatei (ogagent.config)

```
#####  
# Das Zeichen # kennzeichnet eine Kommentarzeile.  
#####  
# Dies ist eine ObjectGrid-Agentenkonfigurationsdatei (der vordefinierte Dateiname ist "ogagent.config"), die  
# von einem ObjectGrid-Agenten gefunden und syntaktisch analysiert werden kann, wenn sie im Klassenpfad enthalten ist.  
# Wenn die Datei den Namen "ogagent.config" hat und im Klassenpfad enthalten ist, wird für Java-Programme,  
# die mit "-javaagent:objectgridRoot/ogagent.jar" ausgeführt werden, der
```

```

# ObjectGrid-Agent aktiviert.
# Wenn die Datei nicht den Namen "ogagent.config" hat, aber im Klassenpfad enthalten ist,
# können Sie den Dateinamen mit der Option "config" des ObjectGrid-Agenten angeben.
# -javaagent:objectgridRoot/lib/objectgrid.jar=config=myOverrideConfigFile
# In den folgenden Kommentaren finden Sie weitere Informationen zum Überschreiben
# der Instrumentierungseinstellung.

# Der erste Teil der Konfiguration ist die Liste der Pakete und Klassen, die in die
# Umsetzungsdomäne eingeschlossen werden sollen.
# Die einzuschließenden Elemente (Pakete/Klassen), aus denen sich die Instrumentierungsdomäne
# zusammensetzt, müssen am Anfang der Datei stehen.
com.testpackage
com.testClass

# Umsetzungsdomäne: Die vorherigen Zeilen sind Pakete/Klassen, aus denen sich die Umsetzungsdomäne zusammensetzt.
# Das System verarbeitet Klassen mit Namen, die mit den zuvor genannten Paketen/Klassen beginnen, bei der Umsetzung.
#
# Token @Exclude: Schließt Elemente aus der Umsetzungsdomäne aus.
# Das Token @Exclude zeigt an, dass die nachfolgenden Pakete/Klassen aus der Umsetzungsdomäne ausgeschlossen werden sollen.
# Es wird verwendet, wenn der Benutzer einige Pakete/Klassen der zuvor angegebenen eingeschlossenen Pakete ausschließen möchte.
#
# Token @FieldAccessEntity: Entitätsdomäne mit Feldzugriff
# Das Token @FieldAccessEntity gibt an, dass die nachfolgenden Pakete/Klassen Entitätspakete/-klassen mit Feldzugriff sind.
# Wenn dem Token @FieldAccessEntity keine Zeile folgt, ist dies dasselbe, als würden Sie das Token @FieldAccessEntity nicht angeben.
# Die Laufzeitumgebung berücksichtigt es, wenn der Benutzer keine Entitätsklassen/-pakete mit Feldzugriff angibt.
# Die Domäne "Entitätsdomäne mit Feldzugriff" ist eine Unterdomäne der Umsetzungsdomäne.
#
# Pakete/Klassen, die in der "Entitätsdomäne mit Feldzugriff" aufgelistet sind, gehören immer zur Umsetzungsdomäne,
# selbst wenn sie nicht in der Umsetzungsdomäne aufgelistet sind.
# Das Token @Exclude, unter dem Pakete/Klassen aufgelistet werden, die von der Umsetzung ausgeschlossen sind, hat
# keine Auswirkung auf die "Entitätsdomäne mit Feldzugriff".
# Anmerkung: Wenn Sie @FieldAccessEntity angeben, müssen alle Entitäten mit Feldzugriff in der Entitätsdomäne
# mit Feldzugriff enthalten sein, ansonsten wird eine Ausnahme des Typs "FieldAccessEntityNotInstrumentedException" ausgegeben.
#
# Die Standardkonfigurationsdatei für den ObjectGrid-Agenten ist "ogagent.config".
# Die Laufzeitumgebung sucht diese Datei als Ressource im Klassenpfad und verarbeitet sie.
# Benutzer können diese vordefinierte ObjectGrid-Agentenkonfigurationsdatei mit der Option
# "config" des Agenten überschreiben.
#
# Beispiel:
# javaagent:objectgridRoot/lib/objectgrid.jar=config=myOverrideConfigFile
#
# Die Instrumentierungsdefinition, einschließlich Umsetzungsdomäne, @Exclude und @FieldAccessEntity, kann
# individuell mit entsprechenden vordefinierten Agentenoptionen überschrieben werden.
# Zu den vordefinierten Agentenoptionen gehören:
# include      -> Wird verwendet, um die Definition der Instrumentierungsdomäne (erster Teil der Konfigurationsdatei) zu überschreiben.
# exclude     -> Wird verwendet, um die @Exclude-Definition zu überschreiben.
# fieldAccessEntity -> Wird verwendet, um die @FieldAccessEntity-Definition zu überschreiben.
#
# Die Agentenoptionen müssen durch ein Semikolon (;) voneinander getrennt werden.
# In der Agentenoption müssen die Pakete und Klassen ein Komma (",") voneinander getrennt werden.
#
# Im Folgenden sehen Sie ein Beispiel, das den Namen der Konfigurationsdatei nicht überschreibt:
# -javaagent:objectgridRoot/lib/objectgrid.jar=include=includedPackage;exclude=excludedPackage;fieldAccessEntity=package1,package2
#
#####
@Exclude
com.excludedPackage
com.excludedClass

@FieldAccessEntity

```

## Leistungshinweis

Um eine bessere Leistung zu erzielen, geben Sie die Umsetzungsdomäne und die Entitätsdomäne mit Feldzugriff an.

## EntityManager in einer verteilten Umgebung

Sie können die Anwendungsprogrammierschnittstelle "EntityManager" nicht nur in einer einer lokalen eigenständigen Umgebung von eXtreme Scale verwenden, sondern auch in einer verteilten und partitionierten Umgebung von WebSphere eXtreme Scale. Der Hauptunterschied besteht darin, wie auf die ferne Umgebung von eXtreme Scale zugegriffen wird und wie die Verbindung zu dieser Verbindung hergestellt wird. Nach der Herstellung einer Verbindung zu einer fernen Umgebung von eXtreme Scale unterscheidet sich der Zugriff auf den EntityManager über das Session-Objekt in keiner Weise von der Verwendung der API "EntityManager".

## Erforderliche Konfigurationsdateien

Die folgenden XML-Konfigurationsdateien sind erforderlich:

- ObjectGrid-XML-Deskriptordatei
- XML-Deskriptordatei der Entität
- XML-Deskriptordatei der Implementierung oder des Grids

Dem Server muss mitgeteilt werden, welche Entitäten und BackingMaps während des Starts ausgeführt werden sollen. Erstellen Sie eine XML-Datei für das ObjectGrid und eine XML-Datei für die Entitäten.

Die Deskriptordatei mit den Metadaten für die Entitäten enthält eine Beschreibung der verwendeten Entitäten. Sie müssen mindestens die Entitätsklasse und den Entitätsnamen angeben. Wenn Sie in einer Umgebung mit Java Platform, Standard Edition 5 arbeiten, liest eXtreme Scale automatisch die Entitätsklasse und die zugehörigen Annotationen. Sie können weitere XML-Attribute definieren, wenn die Entitätsklasse keine Annotationen hat oder wenn bestimmte Werte überschrieben werden müssen. Sie können das folgende XML-Konfigurations-Snippet verwenden, um eXtreme Scale mit Entitäten zu definieren. In diesem Snippet erstellt der Server eine eXtreme-Scale-Instanz mit dem Namen bookstore und eine zugehörige BackingMap mit dem Namen order. Das Snippet verweist auf die Datei entity.xml. In diesem Fall enthält die Datei entity.xml nur eine einzige Entität, die Entität "order".

```
<objectGridConfig xmlns:xsi="http://www.w3.org/2001/XMLSchema-instance"
xsi:schemaLocation="http://ibm.com/ws/objectgrid/config ../objectGrid.xsd"
xmlns="http://ibm.com/ws/objectgrid/config">
  <objectGrids>
    <objectGrid name="bookstore" entityMetadataXMLFile="entity.xml">
      <backingMap name="Order"/>
    </objectGrid>
  </objectGrids>
</objectGridConfig>
```

Abbildung 1. objectgrid.xml

Diese Datei objectgrid.xml verweist auf die Datei entity.xml mit dem Attribut "entityMetadataXMLFile". Die Position dieser Datei ist relativ zur Position der Datei objectgrid.xml. Es folgt ein Beispiel für die Datei entity.xml:

```
<entity-mappings xmlns="http://ibm.com/ws/projector/config/emd"
xmlns:xsi="http://www.w3.org/2001/XMLSchema-instance"
xsi:schemaLocation="http://ibm.com/ws/projector/config/emd ./emd.xsd">
  <entity class-name="com.ibm.websphere.tutorials.objectgrid.em.distributed.step1.Order" name="Order"/>
</entity-mappings>
```

Abbildung 2. entity.xml

Informationen zum Starten eines Servers von eXtreme Scale finden Sie im Abschnitt *Serverprozesse von WebSphere eXtreme Scale starten im Administratorhandbuch*, in dem die Dateien deployment.xml und objectgrid.xml zum Starten des Katalogservers verwendet werden.

## Verbindung zu einem verteilten Server von eXtreme Scale herstellen

Der folgende Code aktiviert den Verbindungsmechanismus für einen Client und einen Server auf demselben Computer:

```
String catalogEndpoints="localhost:2809";
URL clientOverrideURL= new URL("file:etc/emtutorial/distributed/step1/objectgrid.xml");
ClientClusterContext clusterCtx = ogMgr.connect(catalogEndpoints, null, clientOverrideURL);
ObjectGrid objectGrid=ogMgr.getObjectGrid(clusterCtx, "bookstore");
```

Abbildung 3. Verbindung zu einem verteilten Server

Beachten Sie im vorherigen Code-Snippet die Referenz auf den fernen Server von eXtreme Scale. Nach der Herstellung der Verbindung können Sie Operationen der API "EntityManager" aufrufen, wie z. B. die Methoden "persist", "update", "remove" oder "find".

**Achtung:** Wenn Sie Entitäten verwenden, übergeben Sie die neue XML-Deskriptordatei des Clients für das ObjectGrid an die Methode "connect". Wird ein Nullwert an die Eigenschaft "clientOverrideURL" übergeben und hat der Client eine andere Verzeichnisstruktur als der Server, kann der Client die ObjectGrid-XML-Deskriptordatei oder die XML-Deskriptordatei der Entität möglicherweise nicht finden. Zumindest können die XML-Dateien für das ObjectGrid und die Entitäten für den Server in den Client kopiert werden.

## Client- und serverseitige Schemas

Das serverseitige Schema definiert den Typ der Daten, von Daten, die in den Maps in einem Server gespeichert werden. Das clientseitige Schema ist eine Zuordnung zu den Anwendungsobjekten aus dem Schema im Server. Sie könnten beispielsweise das folgende serverseitige Schema haben:

```
@Entity
class ServerPerson
{
    @Id String ssn;
    String firstName;
    String surname;
    int age;
    int salary;
}
```

Abbildung 4. *ServerPerson.java*

Ein Client könnte ein Objekt haben, das wie im folgenden Beispiel annotiert ist:

```
@Entity(name="ServerPerson")
class ClientPerson
{
    @Id @Basic(alias="ssn") String socialSecurityNumber;
    String surname;
}
```

Abbildung 5. *ClientPerson.java*

Dieser Client verwendet anschließend eine serverseitige Entität und projiziert das Subset der Entität in das Clientobjekt. Diese Projektion führt zu Bandbreiten- und Speichereinsparungen auf einem Client, weil der Client nur die Informationen besitzt, die er benötigt, und nicht alle Informationen, die in der serverseitigen Entität enthalten sind. Anwendungen können ihre eigenen Objekte verwenden, anstatt sie dazu zu zwingen, einen Satz von Klassen für den Datenzugriff gemeinsam zu nutzen.

## Woher stammt das Schema?

Wenn die Anwendung Java SE 5 verwendet, kann die Anwendung den Objekten über Annotationen hinzugefügt werden. Der EntityManager kann das Schema aus den Annotationen in diesen Objekten lesen. Der EntityManager muss wissen, welche Objekte untersucht werden müssen. Die Anwendung informiert die Laufzeitumgebung von eXtreme Scale über diese Objekte mit Hilfe der Datei `entity.xml`, die über die Datei `objectgrid.xml` referenziert wird. In der Datei `entity.xml` sind alle Entitäten aufgelistet. Die Datei enthält für jede Entität einen Klassennamen oder ein Schema. Wenn ein Klassename angegeben ist, wird versucht, die Annotationen der Java SE Version 5 aus diesen Klassen zu lesen, um das Schema zu bestimmen. Ist die Klassendatei nicht annotiert, wird das Schema der XML-Datei entnommen. Die XML-Datei wird verwendet, um alle Attribute, Schlüssel und Beziehungen für jede Entität anzugeben.

Eine eigenständige Umgebung von eXtreme Scale benötigt keine XML-Dateien. Das Programm kann eine eXtreme-Scale-Referenz anfordern und die Methode "Object-

Grid.registerEntities" aufrufen, um eine Liste mit annotierten Klassen der Java SE Version 5 oder eine XML-Datei anzugeben.

Die Laufzeitumgebung verwendet die XML-Datei oder eine Liste mit annotierten Klassen, um Entitätsnamen, Attributnamen und -typen, Schlüsselfelder und -typen sowie Beziehungen zwischen Entitäten zu suchen. Wenn eXtreme Scale in einem Server oder im eigenständigen Modus ausgeführt wird, wird automatisch eine Map erstellt, die nach der jeweiligen Entität benannt wird. Diese Maps können über die Datei objectgrid.xml oder APIs, die von der Anwendung oder von Injektions-Frameworks wie Spring definiert werden, weiter angepasst werden.

## Deskriptordateien für Entitätsmetadaten

Weitere Informationen zur Deskriptordatei für Metadaten finden Sie im Abschnitt „Datei emd.xsd“ auf Seite 63.

### Zugehörige Verweise

API-Dokumentation: Schnittstelle EntityManager

„Schnittstelle "EntityTransaction"“ auf Seite 78

Sie können die Schnittstelle "EntityTransaction" verwenden, um Transaktionen abzugrenzen.

## Abfragewarteschlangen für Entitäten

Abfragewarteschlangen ermöglichen Anwendungen, eine durch Abfrage im serverseitigen oder lokalen eXtreme Scale über eine Entität qualifiziert Warteschlange zu erstellen. Entitäten aus dem Abfrageergebnis werden in dieser Warteschlange gespeichert. Derzeit werden Abfragewarteschlangen nur in Maps unterstützt, die die pessimistische Sperrstrategie verwenden.

Eine Abfragewarteschlange wird von mehreren Transaktionen/Clients gemeinsam genutzt. Wenn die Abfragewarteschlange leer ist, wird die Entitätenabfrage, die dieser Warteschlange zugeordnet ist, erneut ausgeführt, und die neuen Ergebnisse werden der Warteschlange hinzugefügt. Eine Abfragewarteschlange wird über die Entitätsabfragezeichenfolge und -parameter eindeutig identifiziert. Es gibt nur eine einzige Instanz jeder eindeutigen Abfragewarteschlange in einer ObjektGrid-Instanz. Weitere Informationen finden Sie in der API-Dokumentation zu EntityManager.

## Beispiel für Abfragewarteschlange

Das folgende Beispiel veranschaulicht, wie eine Abfragewarteschlange verwendet werden kann.

```
/**
 * Nicht zugordnete Task vom Typ "Frage" abrufen
 */
private void getUnassignedQuestionTask() throws Exception {
    EntityManager em = og.getSession().getEntityManager();
    EntityTransaction tran = em.getTransaction();

    QueryQueue queue = em.createQueryQueue("SELECT t FROM Task t
    WHERE t.type=?1 AND t.status=?2", Task.class);
    queue.setParameter(1, new Integer(Task.TYPE_QUESTION));
    queue.setParameter(2, new Integer(Task.STATUS_UNASSIGNED));

    tran.begin();
    Task nextTask = (Task) queue.getNextEntity(10000);
    System.out.println("next task is " + nextTask);
    if (nextTask != null) {
```

```

        assignTask(em, nextTask);
    }
    tran.commit();
}

```

Der vorherige Beispielcode erstellt zunächst eine Abfragewarteschlange (Query-Queue) mit der Entitätsabfragezeichenfolge "SELECT t FROM Task t WHERE t.type=?1 AND t.status=?2". Anschließend setzt er die Parameter des QueryQueue-Objekts. Diese Abfragewarteschlange stellt alle "nicht zugeordneten" Tasks des Typs "Frage" dar. Das QueryQueue-Objekt ist dem Query-Objekt einer Entität sehr ähnlich.

Nach dem Erstellen des QueryQueue-Objekts wird eine Entitätstransaktion gestartet und die Methode "getNextEntity" aufgerufen, die die nächste verfügbare Entität mit einem definierten Zeitlimit von 10 Sekunden aufruft. Nachdem die Entität abgerufen wurde, wird sie in der Methode "assignTask" verarbeitet. Die Methode "assignTask" modifiziert die Task-Entitätsinstanz und ändert den Status in "zugeordnet"(assigned), woraufhin die Instanz aus der Warteschlange entfernt wird, weil sie dem Filter von QueryQueue nicht mehr entspricht. Nach der Zuordnung wird die Transaktion festgeschrieben.

Anhand dieses einfachen Beispiels lässt sich erkennen, dass eine Abfragewarteschlange einer Entitätsabfrage gleicht. Sie unterscheiden sich jedoch in folgender Hinsicht:

1. Entitäten in der Abfragewarteschlange können iterativ abgerufen werden. Die Benutzeranwendung legt die Anzahl der abzurufenden Entitäten fest. Wenn beispielsweise "QueryQueue.getNextEntity(timeout)" verwendet wird, wird nur eine einzige Entität abgerufen, wird "QueryQueue.getNextEntities(5, timeout)" verwendet, werden 5 Entitäten abgerufen. In einer verteilten Umgebung entscheidet die Anzahl der Entitäten direkt über die Anzahl der Bytes, die vom Server an den Client übertragen werden.
2. Beim Abruf einer Entität aus der Abfragewarteschlange wird eine U-Sperre für die Entität gesetzt, so dass keine anderen Transaktionen auf die Entität zugreifen können.

## Entitäten in einer Schleife abrufen

Sie können Entitäten in einer Schleife abrufen. Im Folgenden sehen Sie ein Beispiel, das veranschaulicht, wie alle nicht zugeordneten Tasks vom Typ "Frage" abgerufen werden:

```

/**
 * Alle nicht zugeordneten Tasks vom Typ "Frage" abrufen
 */
private void getAllUnassignedQuestionTask() throws Exception {
    EntityManager em = og.getSession().getEntityManager();
    EntityTransaction tran = em.getTransaction();

    QueryQueue queue = em.createQueryQueue("SELECT t FROM Task t WHERE
t.type=?1 AND t.status=?2", Task.class);
    queue.setParameter(1, new Integer(Task.TYPE_QUESTION));
    queue.setParameter(2, new Integer(Task.STATUS_UNASSIGNED));

    Task nextTask = null;

    do {
        tran.begin();
        nextTask = (Task) queue.getNextEntity(10000);
        if (nextTask != null) {
            System.out.println("next task is " + nextTask);
        }
    }
}

```

```

    }
    tran.commit();
  } while (nextTask != null);
}

```

Wenn es 10 nicht zugeordnete Tasks vom Typ "Frage" in der Entitäts-Map gibt, erwarten Sie wahrscheinlich, dass 10 Entitäten in der Konsole ausgegeben werden. Wenn Sie jedoch dieses Beispiel ausführen, werden Sie feststellen, dass das Programm entgegen Ihren Erwartungen nie beendet wird.

Wenn eine Abfragewarteschlange erstellt und die Methode "getNextEntity" aufgerufen wird, wird die Entitätsabfrage ausgeführt, die der Warteschlange zugeordnet ist, und die 10 Ergebnisse werden in die Warteschlange eingetragen. Beim Aufruf von "getNextEntity" wird der Warteschlange eine Entität entnommen. Nach der Ausführung von zehn getNextEntity-Aufrufen ist die Warteschlange leer. Die Entitätsabfrage wird automatisch erneut ausgeführt. Da diese 10 Entitäten immer noch vorhanden sind und den Filterkriterien der Abfragewarteschlange entsprechen, werden sie erneut in die Warteschlange eingetragen.

Wenn die folgende Zeile hinter der Anweisung "println()" hinzugefügt wird, werden nur 10 Entitäten ausgegeben:

```
em.remove(nextTask);
```

## In allen Partitionen implementierte Abfragewarteschlangen

In einer verteilten eXtreme-Scale-Umgebung kann eine Abfragewarteschlange für eine einzige oder für alle Partitionen erstellt werden. Wenn eine Abfragewarteschlange für alle Partitionen erstellt wird, gibt es in jeder Partition eine einzige Instanz der Abfragewarteschlange.

Wenn ein Client versucht, die nächste Entität mit der Methode "QueryQueue.getNextEntity" oder "QueryQueue.getNextEntities" abzurufen, sendet der Client eine Anforderung an eine der Partitionen. Ein Client sendet so genannte Peek- und Pin-Anforderungen an den Server:

- Bei einer Peek-Anforderung sendet der Client eine Anforderung an eine einzige Partition, und der Server gibt das Ergebnis sofort zurück. Ist eine Entität in der Warteschlange vorhanden, sendet der Server eine Antwort mit der Entität, wenn nicht, sendet der Server eine Antwort ohne Entität. In beiden Fällen gibt der Server sofort ein Ergebnis zurück.
- Bei einer Pin-Anforderung sendet der Client eine Anforderung an eine einzige Partition, und der Server wartet, bis eine Entität verfügbar ist. Ist eine Entität in der Warteschlange enthalten, sendet der Server unverzüglich eine Antwort mit der Entität, wenn nicht, wartet der Server in der Warteschlange, bis eine Entität verfügbar ist oder bis das zulässige Anforderungszeitlimit abläuft.

Im Folgenden sehen Sie ein Beispiel, in dem veranschaulicht wird, wie eine Entität für eine Abfragewarteschlange abgerufen wird, die in allen Partitionen (n) implementiert ist:

1. Beim Aufruf einer Methode "QueryQueue.getNextEntity" oder "QueryQueue.getNextEntities" verwendet der Client wahlfrei eine Partitionsnummer zwischen 0 und n-1.
2. Der Client sendet eine Peek-Anforderung an die wahlfreie Partition.
  - Ist eine Entität verfügbar, wird die Methode "QueryQueue.getNextEntity" bzw. "QueryQueue.getNextEntities" durch Rückgabe der Entität beendet.



- Ist keine Entität verfügbar und handelt es sich nicht um die letzte noch nicht besuchte Partition, sendet der Client eine Peek-Anforderung an die nächste Partition.
- Ist keine Entität verfügbar und handelt es sich um die letzte noch nicht besuchte Partition, sendet der Client stattdessen eine Pin-Anforderung.
- Wenn das zulässige Zeitlimit für die Pin-Anforderung an die letzte Partition abläuft und noch immer keine Daten verfügbar sind, unternimmt der Client einen letzten Versuch, indem er noch einmal nacheinander eine Peek-Anforderung an alle Partitionen sendet. Deshalb ist der Client in der Lage, eine mittlerweile in einer der früheren Partitionen verfügbare Entität abzurufen.

## Unterstützung von Teilentitäten und keinen Entitäten

Im Folgenden wird die Methode zum Erstellen eines QueryQueue-Objekt im EntityManager veranschaulicht:

```
public QueryQueue createQueryQueue(String qlString, Class entityClass);
```

Das Ergebnis in der Abfragewarteschlange muss an das Objekt weitergegeben werden, das mit dem zweiten Parameter der Methode definiert ist, Class entityClass.

Wenn dieser Parameter angegeben ist, muss die Klasse denselben Entitätsnamen haben, der auch in der Abfragezeichenfolge enthalten ist. Dies ist hilfreich, wenn Sie eine Entität an eine Teilentität weitergeben möchten. Wird ein Nullwert als Entitätsklasse verwendet, wird das Ergebnis nicht weitergegeben. Der in der Map gespeicherte Wert hat das Entitätstupelformat.

## Clientseitige Schlüsselkollision

In einer verteilten eXtreme-Scale-Umgebung werden Abfragewarteschlangen nur für eXtreme-Scale-Maps mit pessimistischem Sperrmodus unterstützt. Deshalb gibt es auf der Clientseite keinen nahen Cache. Ein Client kann jedoch Daten (Schlüssel und Wert) in einer Transaktions-Map verwalten. Dies könnte zu einer Schlüsselkollision führen, wenn eine vom Server abgerufene Entität denselben Schlüssel wie ein Eintrag verwendet, der bereits in der Transaktions-Map enthalten ist.

Bei einer Schlüsselkollision verwendet die Laufzeitumgebung des eXtreme-Scale-Clients die folgende Regel, um entweder eine Ausnahme auszulösen oder die Daten unbeaufsichtigt zu überschreiben.

1. Wenn der von der Kollision betroffene Schlüssel der Schlüssel der Entität ist, die in der Entitätenabfrage für die Abfragewarteschlange angegeben ist, wird eine Ausnahme ausgelöst. In diesem Fall wird die Transaktion rückgängig gemacht und eine U-Sperre für diesen Entitätsschlüssel auf Serverseite freigegeben.
2. Wenn der betroffene Schlüssel der Schlüssel der Entitätsassoziation ist, werden die Daten in der Transaktions-Map ohne Warnung überschrieben.

Die Schlüsselkollision kommt nur vor, wenn die Transaktions-Map Daten enthält. Anders ausgedrückt, es kommt nur dann zu einer Schlüsselkollision, wenn ein Aufruf der Methode "getNextEntity" oder "getNextEntities" in einer Transaktion vorgenommen wird, die bereits genutzt wurde (es wurden neue Daten eingefügt oder aktualisiert). Wenn eine Anwendung Schlüsselkollisionen vermeiden möchte, muss sie getNextEntity oder getNextEntities immer in einer noch nicht genutzten Transaktion aufrufen.

## Clientfehler

Nachdem ein Client eine getNextEntity- oder getNextEntities-Anforderung an den Server gesendet hat, könnte einer der folgenden Fehler im Client auftreten:

1. Der Client sendet eine Anforderung an den Server und stürzt dann ab.
2. Der Client empfängt eine oder mehrere Entitäten vom Server und stürzt dann ab.

Im ersten Fall erkennt der Server, dass der Client nicht mehr verfügbar ist, wenn er versucht, die Antwort an den Client zurückzusenden. Im zweiten Fall wird eine X-Sperre für diese Entitäten gesetzt. Wenn der Client abstürzt, läuft irgendwann das Transaktionszeitlimit ab, und die X-Sperre wird freigegeben.

Abfrage mit der Klausel ORDER BY

Im Allgemeinen wird die Klausel ORDER BY von Abfragewarteschlangen nicht berücksichtigt. Wenn Sie getNextEntity oder getNextEntities über die Abfragewarteschlange aufrufen, besteht keine Garantie, dass die Entitäten in der richtigen Reihenfolge zurückgegeben werden. Der Grund hierfür ist, dass die Entitäten in den Partitionen nicht sortiert werden können. Wenn die Abfragewarteschlange in allen Partitionen implementiert ist und ein getNextEntity- oder getNextEntities-Aufruf ausgeführt wird, wird eine Partition für die Verarbeitung der Anforderung zufällig ausgewählt. Deshalb ist die Reihenfolge nicht garantiert.

ORDER BY wird berücksichtigt, wenn eine Abfragewarteschlange nur in einer einzigen Partition implementiert ist.

### Zugehörige Verweise

API-Dokumentation: Schnittstelle EntityManager

„Schnittstelle "EntityTransaction"“

Sie können die Schnittstelle "EntityTransaction" verwenden, um Transaktionen abzugrenzen.

## Schnittstelle "EntityTransaction"

Sie können die Schnittstelle "EntityTransaction" verwenden, um Transaktionen abzugrenzen.

### Zweck

Zum Abgrenzen einer Transaktion können Sie die Schnittstelle "EntityTransaction" verwenden, die einer EntityManager-Instanz zugeordnet wird. Verwenden Sie die Methode "EntityManager.getTransaction()", um die EntityTransaction-Instanz für den EntityManager abzurufen. Jede EntityManager- und jede EntityTransaction-Instanz wird dem Session-Objekt zugeordnet. Sie können Transaktionen über das EntityTransaction-Objekt oder das Session-Objekt abgrenzen. Methoden in der Schnittstelle "EntityTransaction" haben keine geprüften Ausnahmen. Es können nur Laufzeitausnahmen des Typs "PersistenceException" oder zugehöriger Unterklassen ausgegeben werden.

Weitere Informationen zur Schnittstelle "EntityTransaction" finden Sie in der Beschreibung der Schnittstelle "EntityTransaction" in der API-Dokumentation.

## Zugehörige Konzepte

„Anwendungsprogrammierschnittstelle "EntityManager"“ auf Seite 54

Die Anwendungsprogrammierschnittstelle (API, Application Programming Interface) "EntityManager" vereinfacht die Interaktion mit dem eXtreme-Scale-Cache, indem sie eine einfache Methode für die Deklaration und Interaktion mit einem komplexen Graphen zusammengehöriger Objekte bereitstellt.

„Abfragewarteschlangen für Entitäten“ auf Seite 74

Abfragewarteschlangen ermöglichen Anwendungen, eine durch Abfrage im serverseitigen oder lokalen eXtreme Scale über eine Entität qualifiziert Warteschlange zu erstellen. Entitäten aus dem Abfrageergebnis werden in dieser Warteschlange gespeichert. Derzeit werden Abfragewarteschlangen nur in Maps unterstützt, die die pessimistische Sperrstrategie verwenden.

„EntityManager in einer verteilten Umgebung“ auf Seite 71

Sie können die Anwendungsprogrammierschnittstelle "EntityManager" nicht nur in einer lokalen eigenständigen Umgebung von eXtreme Scale verwenden, sondern auch in einer verteilten und partitionierten Umgebung von WebSphere eXtreme Scale. Der Hauptunterschied besteht darin, wie auf die ferne Umgebung von eXtreme Scale zugegriffen wird und wie die Verbindung zu dieser Verbindung hergestellt wird. Nach der Herstellung einer Verbindung zu einer fernen Umgebung von eXtreme Scale unterscheidet sich der Zugriff auf den EntityManager über das Session-Objekt in keiner Weise von der Verwendung der API "EntityManager".

## Lebenszyklen von Entitäteninstanzen

Jede Entitäteninstanz durchläuft in einem Lebenszyklus verschiedene Status.

### Lebenszyklus einer Entitäteninstanz

Eine Entitäteninstanz hat die folgenden Status:

- **Neu:** Eine neu erstellte Entitäteninstanz, die noch nicht im eXtreme-Scale-Cache vorhanden ist.
- **Verwaltet:** Die Entitäteninstanz ist im eXtreme-Scale-Cache vorhanden und wird über den EntityManager abgerufen oder als persistent definiert. Im Status "Verwaltet" muss eine Entität einer aktiven Transaktion zugeordnet sein.
- **Freigegeben:** Die Entitäteninstanz ist im eXtreme-Scale-Cache vorhanden, aber keiner aktiven Transaktion mehr zugeordnet.
- **Entfernt:** Die Entitäteninstanz wurde bereits aus dem eXtreme-Scale-Cache entfernt bzw. soll entfernt werden, wenn eine Flush- oder Commit-Operation für die Transaktion ausgeführt wird.
- **Ungültig gemacht:** Die Entitäteninstanz wurde im eXtreme-Scale-Cache ungültig gemacht bzw. soll ungültig gemacht werden, wenn eine Flush- oder Commit-Operation für die Transaktion ausgeführt wird.

Wenn sich der Status einer Entität ändert, können Sie Callback-Methoden für den Lebenszyklus aufrufen.

In den folgenden Abschnitten werden die Status "Neu", "Verwaltet", "Freigegeben", "Entfernt" und "Ungültig gemacht" für Entitäten ausführlicher beschrieben.

### Entität suchen

Die Schnittstelle "EntityManager" unterstützt die Suchoperation mit und ohne Transaktionsabgrenzung:

```

em.getTransaction().begin();
Order o=(Order)em.find(Order.class,"ASD100");
// Entitäteninstanz "o" ist jetzt verwaltet
em.getTransaction().commit()
// Entitäteninstanz "o" ist jetzt freigegeben

```

Abbildung 6. Beispiel für eine verwaltete Entität

```

Order o=(Order)em.find(Order.class,"ASD100");
// Entitäteninstanz "o" freigegeben. Nicht innerhalb der Transaktionsgrenzen

```

Abbildung 7. Beispiel für eine freigegebene Entität

## Entität als persistent definieren

Mit der Methode "persist" der Schnittstelle "EntityManager" können Sie Entitäten als persistent definieren. Sie können nur neue Entitäteninstanzen als persistent definieren, und Sie müssen Sie mit einem Mustercode beschreiben.

```

//Ein Kunde (Customer) mit der ID "10" ist im eXtreme-Scale-Cache vorhanden, und
//der Kunde erteilt einen neuen Auftrag (Order).
Order newOrder = createNewOrder();
// Angenommen, diese Methode definiert Beziehungen zwischen Item, OrderLine und Order
em.getTransaction().begin();
Customer customer10=(Customer)em.find(Customer.class, "10");
// Entitäteninstanz customer10 ist jetzt verwaltet
newOrder.customer=customer10;
// newOrder als persistent definieren. newOrder hat den Status "Neu"
em.persist(newOrder);
// newOrder hat jetzt den Status "Verwaltet"
// Bei der Ausführung der Commit-Operation für die Transaktion werden Order, OrderLines und Item als
// Da der Kunde bereits vorhanden ist und den Status "Verwaltet" hat, ignoriert die
// Operation "persist" die Customer-Entität in diesem Fall.
// Wenn Kundenobjektwerte geändert werden, wird das Objekt im
// eXtreme-Scale-Cache aktualisiert.
em.getTransaction().commit();// Jetzt werden alle Instanzen freigegeben.

// Der Kunde erteilt einen weiteren Auftrag, und dieses Mal verwendet der
// Code die freigegebene Entitäteninstanz. Dies ist ein Fehler, und es
// wird eine Ausnahme ausgelöst.

Order newOrder2 = createNewOrder();
// Angenommen, diese Methode definiert Beziehungen zwischen Item, OrderLine und Order
em.getTransaction().begin();
newOrder.customer=customer10; // Verwendung einer freigegebenen customer10-Instanz
em.persist(newOrder2);
// Der Commit-Aufruf definiert den Auftrag (Order) im ObjectGrid als persistent.
// Die Instanz customer10 ist bereits in eXtreme Scale vorhanden, aber da
// die Instanz freigegeben ist, löst die Commit-Operation eine Ausnahme
// vom Typ EntityExistsException für den Customer-Objekt aus.
em.getTransaction().commit();

```

Wenn Sie eine Entität als persistent definieren, beachten Sie folgende Punkte:

- Sie können nur neue Entitäten als persistent definieren. Nach dem Aufruf von "em.persist(object)" haben die Entitäten den Status "Verwaltet". Die Entität bleibt so lange im Status "Verwaltet", bis die Transaktion festgeschrieben (Commit) oder rückgängig gemacht (Rollback) wird. Im Wesentlichen handelt es sich um eine Einfügeoperation.
- Die Operation "persist" wird für bereits verwaltete Entitäten ignoriert, und alle Wertänderungen werden aktualisiert. Wenn die Transaktion zunächst rückgängig gemacht wurde, entspricht der Aufruf der Methode "persist" einer Aktualisierungsoperation.

- Wenn "CascadeType" den Wert PERSIST oder ALL hat, wird die Operation an die Assoziationen weitergegeben.
- Wenn eine neu erstellte Entität oder eine freigegebene Entität als persistent definiert wird, wird eine Ausnahme vom Typ "EntityExistsException" ausgelöst, wenn der Schlüssel im eXtreme-Scale-Cache vorhanden ist. Die Persistenz gleicht der einer Einfügeoperation, bei der eine Ausnahme ausgelöst wird, weil der Schlüssel bereits vorhanden ist.

## Entität entfernen oder ungültig machen

Beim Entfernen oder Ungültigmachen einer Entität wird eine ähnliche Semantik wie bei der Methode "persist" verwendet. Beim Entfernen einer Entität wird die Entität aus eXtreme Scale und aus dem Betriebssystem entfernt. Beim Ungültigmachen einer Entität wird die Entität nur aus eXtreme Scale, entfernt, und es werden keine Ladeprogramm-Plug-ins aufgerufen.

- Zum Entfernen einer Entität muss die Entität den Status "Verwaltet" haben.
- Wenn "CascadeType" den Wert REMOVE oder ALL hat, wird die Operation "remove" an die Assoziationen weitergegeben.
- Wenn "CascadeType" den Wert INVALIDATE oder ALL hat, wird die Operation "invalidate" an die Assoziationen weitergegeben.
- Es kann eine Ausnahme vom Typ "IllegalArgumentException" ausgelöst werden, wenn eine freigegebene Entität entfernt oder ungültig gemacht wird.
- Wenn eine Entität neu ist, ignoriert die Operation "remove" bzw. "invalidate" die Entität. Je nach Wert von CascadeType wird die Operation weitergegeben.

```
// Eine richtige Methode für das Entfernen einer Entität.
// Es ist ein Auftrag (Order) mit der Auftragsnummer (OrderNumber) ASD100 vorhanden, der entfernt
em.getTransaction().begin();
Order o=(Order)em.find(Order.class,"ASD100"); // Verwaltet
em.remove(o);
em.getTransaction().commit();

// Eine ungültige Methode zum Entfernen einer Entität:
Order o=(Order)em.find(Order.class,"ASD100");
// Freigegeben; nicht innerhalb der Transaktionsabgrenzung
em.getTransaction().begin();
em.remove(o); // Kann eine Ausnahme vom Typ "IllegalArgumentException" auslösen
em.getTransaction().commit();
```

## Entitäts-Listener und Callback-Methoden

Anwendungen können benachrichtigt werden, wenn Entitätstransaktionen ihren Status wechseln. Es sind zwei Callback-Mechanismen für Statusänderungsereignisse vorhanden: Callback-Methoden für den Lebenszyklus, die in einer Entitätsklasse definiert und aufgerufen werden, wenn sich der Entitätsstatus ändert, und Entitäts-Listener, die allgemeiner sind, weil der Entitäts-Listener bei mehreren Entitäten registriert werden kann.

### Callback-Methoden für den Lebenszyklus der Entität

Callback-Methoden für den Lebenszyklus der Entität können in der Entitätsklasse definiert werden und werden aufgerufen, wenn sich der Entitätsstatus ändert. Diese Methoden sind hilfreich für die Validierung von Entitätsfeldern und die Aktualisierung eines Übergangszustand, der gewöhnlich nicht persistent für die Entität definiert wird. Callback-Methoden für den Lebenszyklus der Entität können auch in Klassen definiert werden, die keine Entitäten verwenden. Solche Klassen sind Entitäts-Listener-Klassen, die mehreren Entitätstypen zugeordnet werden können.

nen. Callback-Methoden für den Lebenszyklus können über Metadatenannotationen und eine XML-Deskriptordatei für die Entitätsmetadaten definiert werden:

- **Annotations:** Callback-Methoden für den Lebenszyklus können mit den Annotationen "PrePersist", "PostPersist", "PreRemove", "PostRemove", "PreUpdate", "PostUpdate" und "PostLoad" in einer Entitätsklasse gekennzeichnet werden.
- **XML-Entitätsdeskriptor:** Callback-Methoden für den Lebenszyklus können mit XML beschrieben werden, wenn keine Annotationen verfügbar sind.

## Entitäts-Listener

Eine Entitäts-Listener-Klasse ist eine Klasse, die keine Entitäten verwendet und eine oder mehrere Callback-Methoden für den Lebenszyklus einer Entität definiert. Entitäts-Listener sind hilfreich für allgemeine Prüf- und Protokollierungsanwendungen. Entitäts-Listener können über Metadatenannotationen und eine XML-Deskriptordatei für die Entitätsmetadaten definiert werden:

- **Annotation:** Die Annotation "EntityListeners" kann verwendet werden, um eine oder mehrere Entitäts-Listener-Klassen in einer Entitätsklasse zu kennzeichnen. Wenn mehrere Entitäts-Listener definiert sind, wird die Reihenfolge, in der diese aufgerufen werden, durch die Reihenfolge bestimmt, in der sie in der Annotation "EntityListeners" angegeben sind.
- **XML-Entitätsdeskriptor:** Der XML-Deskriptor kann als Alternative zur Angabe der Aufrufreihenfolge von Entitäts-Listenern oder zum Überschreiben der in den Metadatenannotationen festgelegten Reihenfolge verwendet werden.

## Voraussetzungen für Callback-Methoden

Sie können einen beliebigen Teil oder eine Kombination von Annotationen in einer Entitätsklasse oder Listener-Klasse angegeben werden. Eine Klasse darf nicht mehr als eine einzige Callback-Methode für dasselbe Lebenszyklusereignis enthalten. Dieselbe Methode kann jedoch für mehrere Callback-Ereignisse verwendet werden. Die Entitäts-Listener-Klasse muss einen öffentlichen Konstruktor ohne Argumente haben. Entitäts-Listener sind statusunabhängig. Der Lebenszyklus eines Entitäts-Listeners ist unspezifiziert. eXtreme Scale unterstützt keine Entitätsvererbung. Deshalb können Callback-Methoden nur in der Entitätsklasse und nicht in der Superklasse definiert werden.

## Signatur von Callback-Methoden

Callback-Methoden für den Lebenszyklus einer Entität können in einer Entitäts-Listener-Klasse und/oder direkt in einer Entitätsklasse definiert werden. Callback-Methoden für den Lebenszyklus einer Entität können über Metadatenannotationen und über den XML-Entitätsdeskriptor definiert werden. Die Annotationen, die für die Callback-Methoden in der Entitätsklasse und in der Entitäts-Listener-Klasse verwendet werden, sind identisch. Die Signaturen der Callback-Methoden sind bei der Definition in einer Entitätsklasse anders als bei der Definition in einer Entitäts-Listener-Klasse. Callback-Methoden, die in einer Entitätsklasse oder zugeordneten Superklasse definiert werden, haben die folgende Signatur:

```
void <METHOD>()
```

Callback-Methoden, die in einer Entitäts-Listener-Klasse definiert werden, haben die folgende Signatur:

```
void <METHOD>(Object)
```

Das Argument "Object" ist die Entitätsinstanz, für die die Callback-Methode aufgerufen wird. Das Argument "Object" kann als `java.lang.Object`-Objekt oder als der eigentliche Entitätstyp deklariert werden.

Callback-Methoden können öffentlich, privat, geschützt oder auf Paketebene zugänglich sein, dürfen aber weder statisch (`static`) noch endgültig (`final`) sein.

Die folgenden Annotationen werden definiert, um Callback-Methoden für Lebenszyklusereignisse der entsprechenden Typen zu kennzeichnen:

- `com.ibm.websphere.projector.annotations.PrePersist`
- `com.ibm.websphere.projector.annotations.PostPersist`
- `com.ibm.websphere.projector.annotations.PreRemove`
- `com.ibm.websphere.projector.annotations.PostRemove`
- `com.ibm.websphere.projector.annotations.PreUpdate`
- `com.ibm.websphere.projector.annotations.PostUpdate`
- `com.ibm.websphere.projector.annotations.PostLoad`

Weitere Einzelheiten finden Sie in der API-Dokumentation. Jede Annotation hat ein funktional entsprechendes XML-Attribut, das in der XML-Deskriptordatei für die Entitätsmetadaten definiert wird.

## Semantik der Callback-Methoden für den Lebenszyklus

Jede der verschiedenen Callback-Methoden für den Lebenszyklus hat einen anderen Zweck und wird in einer jeweils anderen Phase des Lebenszyklus einer Entität aufgerufen:

### **PrePersist**

Wird für eine Entität aufgerufen, bevor die Entität als persistent im Speicher definiert wird. Dazu gehören auch Entitäten, die während einer Kaskadierungsoperation als persistent definiert werden. Diese Methode wird in dem Thread der Operation `"EntityManager.persist"` aufgerufen.

### **PostPersist**

Wird für eine Entität aufgerufen, nachdem die Entität als persistent im Speicher definiert wurde. Dazu gehören auch Entitäten, die während einer Kaskadierungsoperation als persistent definiert wurden. Diese Methode wird in dem Thread der Operation `"EntityManager.persist"` aufgerufen. Sie wird nach dem Aufruf von `"EntityManager.flush"` oder `"EntityManager.commit"` aufgerufen.

### **PreRemove**

Wird für eine Entität aufgerufen, bevor die Entität entfernt wird. Dazu gehören auch Entitäten, die während einer Kaskadierungsoperation entfernt werden. Diese Methode wird in dem Thread der Operation `"EntityManager.remove"` aufgerufen.

### **PostRemove**

Wird für eine Entität aufgerufen, nachdem die Entität entfernt wurde. Dazu gehören auch Entitäten, die während einer Kaskadierungsoperation entfernt wurden. Diese Methode wird in dem Thread der Operation `"EntityManager.remove"` aufgerufen. Sie wird nach dem Aufruf von `"EntityManager.flush"` oder `"EntityManager.commit"` aufgerufen.

### PreUpdate

Wird für eine Entität aufgerufen, bevor die Entität im Speicher aktualisiert wird. Diese Methode wird im Thread der Operation "flush" oder "commit" für die Transaktion aufgerufen.

### PostUpdate

Wird für eine Entität aufgerufen, nachdem die Entität im Speicher aktualisiert wurde. Diese Methode wird im Thread der Operation "flush" oder "commit" für die Transaktion aufgerufen.

### PostLoad

Wird für eine Entität aufgerufen, nachdem die Entität aus dem Speicher geladen wurde. Dazu gehören auch Entitäten, die über eine Assoziation geladen wurden. Diese Methode wird im Thread der Ladeoperation, z. B. "EntityManager.find" oder einer Abfrage, aufgerufen.

## Mehrfach vorhandene Callback-Methoden für den Lebenszyklus

Wenn mehrere Callback-Methoden für ein Lebenszyklusereignis einer Entität definiert sind, werden die Methoden in der folgenden Reihenfolge aufgerufen:

1. **In Entitäts-Listnern definierte Callback-Methoden für den Lebenszyklus:** Die Callback-Methoden für den Lebenszyklus, die in den Entitäts-Listener-Klassen für eine Entitätsklasse definiert sind, werden in derselben Reihenfolge aufgerufen, in der sie in den Entitäts-Listener-Klassen in der Annotation "EntityListeners" oder im XML-Deskriptor angegeben wurden.
2. **Listener-Superklasse:** Callback-Methoden, die in der Superklasse des Entitäts-Listeners definiert sind, werden vor den untergeordneten aufgerufen.
3. **Methoden für den Lebenszyklus der Entität:** WebSphere eXtreme Scale unterstützt keine Entitätsvererbung. Deshalb können die Methoden für den Lebenszyklus der Entität nur in der Entitätsklasse definiert werden.

## Ausnahmen

Callback-Methoden für den Lebenszyklus können zu Laufzeitausnahmen führen. Wenn eine Callback-Methode für den Lebenszyklus eine Laufzeitausnahme in einer Transaktion auslöst, wird die Transaktion rückgängig gemacht. Nach dem Eintreten einer Laufzeitausnahme werden keine weiteren Callback-Methoden für den Lebenszyklus aufgerufen.

## Beispiele für Entität-Listener

Dieser Abschnitt enthält Beispiele für Entitäts-Listener.

### Beispiel für einen Entitäts-Listener mit Annotationen

Das folgende Beispiel veranschaulicht die Aufrufe von Callback-Methoden für den Lebenszyklus und die Reihenfolge der Aufrufe. Angenommen, es sind eine Entitätsklasse "Employee" und zwei Entitäts-Listener vorhanden: EmployeeListener und EmployeeListener2.

```
@Entity
@EntityListeners(EmployeeListener.class, EmployeeListener2.class)
public class Employee {
    @PrePersist
    public void checkEmployeeID() {
        ....
    }
}

public class EmployeeListener {
```



```

    @PrePersist
    public void onEmployeePrePersist(Employee e) {
        ....
    }
}

public class PersonListener {
    @PrePersist
    public void onPersonPrePersist(Object person) {
        ....
    }
}

public class EmployeeListener2 {
    @PrePersist
    public void onEmployeePrePersist2(Object employee) {
        ....
    }
}

```

Wenn ein PrePersist-Ereignis in einer Employee-Instanz eintritt, werden die folgenden Methoden nacheinander aufgerufen:

1. onEmployeePrePersist
2. onPersonPrePersist
3. onEmployeePrePersist2
4. checkEmployeeID

### Beispiel für einen Entitäts-Listener mit XML

Das folgende Beispiel veranschaulicht, wie Sie einen Entitäts-Listener in einer Entität über die XML-Deskriptordatei der Entität definieren:

```

<entity
  class-name="com.ibm.websphere.objectgrid.sample.Employee"
  name="Employee" access="FIELD">
  <attributes>
    <id name="id" />
    <basic name="value" />
  </attributes>
  <entity-listeners>
    <entity-listener
      class-name="com.ibm.websphere.objectgrid.sample.EmployeeListener">
      <pre-persist method-name="onListenerPrePersist" />
      <post-persist method-name="onListenerPostPersist" />
    </entity-listener>
  </entity-listeners>
  <pre-persist method-name="checkEmployeeID" />
</entity>

```

Die Entität "Employee" ist mit einer Entitäts-Listener-Klasse "com.ibm.websphere.objectgrid.sample.EmployeeListener" definiert, in der zwei Callback-Methoden für den Lebenszyklus definiert sind. Die Methode "onListenerPrePersist" ist für das Ereignis "PrePersist" bestimmt und die Methode "onListenerPostPersist" für das Ereignis "PostPersist". Außerdem ist die Methode "checkEmployeeID" in der Klasse "Employee" definiert, die auf das Ereignis "PrePersist" wartet.

---

## API "Query"

WebSphere eXtreme Scale stellt eine flexible Abfragesteuerkomponente bereit, mit der Entitäten über die API "EntityManager" und Java-Objekte über die API "ObjectQuery" abgerufen werden können.

## Abfragefunktionen von WebSphere eXtreme Scale

Mit der Abfragesteuerkomponente von eXtreme Scale können Sie unter Verwendung der Abfragesprache von eXtreme Scale Abfragen vom Typ SELECT für eine Entität oder ein objektbasiertes Schema durchführen.

Diese Abfragesprache hat die folgenden Leistungsmerkmale:

- Sie unterstützt Ergebnisse mit einem Wert und mit mehreren Werten.
- Sie stellt Aggregatfunktionen bereit.
- Sie unterstützt Sortierung und Gruppierung.
- Sie unterstützt Verknüpfungen (Joins).
- Sie unterstützt Bedingungsausdrücke mit Unterabfragen.
- Sie unterstützt benannte und positionsgebundene Parameter.
- Sie unterstützt die Verwendung von eXtreme-Scale-Indizes.
- Sie unterstützt die Pfadausdruckssyntax für die Objektnavigation.
- Sie unterstützt Seitenaufteilung.

### Abfrageschnittstelle

Verwenden Sie die Abfrageschnittstelle, um die Ausführung von Entitätsabfragen zu steuern.

Verwenden Sie die Methode "EntityManager.createQuery(String)", um eine Abfrage zu erstellen. Sie können jede Abfrageinstanz mehrfach für die EntityManager-Instanz verwenden, in der sie abgerufen wurde.

Jedes Abfrageergebnis erzeugt eine Entität, deren Entitätsschlüssel die Zeilen-ID (vom Typ "long") und deren Entitätswert die Feldergebnisse der SELECT-Klausel ist. Sie können jedes Abfrageergebnis in nachfolgenden Abfragen verwenden.

Die folgenden Methoden sind in der Schnittstelle "com.ibm.websphere.objectgrid.em.Query" verfügbar.

#### **public ObjectMap getResultMap()**

Die Methode "getResultMap" führt eine SELECT-Abfrage durch und gibt die Ergebnisse in einem ObjectMap-Objekt in der in der Abfrage festgelegten Reihenfolge zurück. Das ObjectMap-Objekt ist nur für die aktuelle Transaktion gültig.

Der Map-Schlüssel ist die Ergebnisnummer (vom Typ "long"), beginnend bei 1. Der Map-Wert hat den Typ "com.ibm.websphere.projector.Tuple", wobei jedes Attribut und jede Assoziation nach der Ordinalposition in der SELECT-Klausel der Abfrage benannt wird. Verwenden Sie die Methode, um das EntityMetadata-Objekt für das Tuple-Objekt abzurufen, das in der Map gespeichert ist.

Die Methode "getResultMap" ist die schnellste Methode für das Abrufen von Abfrageergebnisdaten, wenn mehrere Ergebnisse verfügbar sein können. Sie können den Namen der Entität mit den Methoden "ObjectMap.getEntityMetadata()" und "EntityMetadata.getName()" abrufen.

Die folgende Abfrage gibt beispielsweise zwei Zeilen zurück:

```
String q1 = SELECT e.name, e.id, d from Employee e join e.dept d WHERE d.number=5
Query q = em.createQuery(q1);
ObjectMap resultMap = q.getResultMap();
long rowID = 1; // starts with index 1
```

```

Tuple tResult = (Tuple) resultMap.get(new Long(rowID));
while(tResult != null) {
    // Das erste Attribut ist der Name und hat den Attributnamen 1,
    // aber die Ordinalposition 0.
String name = (String)tResult.getAttribute(0);
Integer id = (String)tResult.getAttribute(1);

    // Dept ist eine Assoziation mit dem Namen 3, aber der
    // Ordinalposition 0, da es sich um die erste Assoziation handelt.
// Die Assoziation ist immer eine 1:1-Beziehung, und deshalb
// gibt es nur einen einzigen Schlüssel.
Tuple deptKey = tResult.getAssociation(0,0);
    ...
    ++rowID;
    tResult = (Tuple) resultMap.get(new Long(rowID));
}

```

### public Iterator getResultIterator

Die Methode "getResultIterator" führt eine SELECT-Abfrage durch und gibt die Abfrageergebnisse über einen Iterator zurück. Jedes Ergebnis ist entweder ein Objekt (bei einer Abfrage mit einem einzigen Wert) oder ein Objektbereich (für eine Abfrage mit mehreren Werten). Die Werte in "Object[]" werden in der Abfrage-reihenfolge gespeichert. Der Ergebnisiterator ist nur für die aktuelle Transaktion gültig.

Diese Methode ist die bevorzugte Methode für das Abrufen von Abfrage-ergebnissen im EntityManager-Kontext. Sie können die optionale Methode "setResultEntityName(String)" verwenden, um die ermittelte Entität zu benennen, so dass sie in weiteren Abfragen verwendet werden kann.

Die folgende Abfrage gibt beispielsweise zwei Zeilen zurück:

```

String q1 = SELECT e.name, e.id, e.dept from Employee e WHERE e.dept.number=5
Query q = em.createQuery(q1);
Iterator results = q.getResultIterator();
while(results.hasNext()) {
    Object[] curEmp = (Object[]) results.next();
    String name = (String) curEmp[0];
    Integer id = (Integer) curEmp[1];
    Dept d = (Dept) curEmp[2];
    ...
}

```

### public Iterator getResultIterator(Class resultType)

Die Methode "getResultIterator(Class resultType)" führt eine SELECT-Abfrage durch und gibt die Abfrageergebnisse über einen Entitätsiterator zurück. Der Entitätstyp wird mit dem Parameter "resultType" bestimmt. Der Ergebnisiterator ist nur für die aktuelle Transaktion gültig.

Verwenden Sie diese Methode, wenn Sie die EntityManager-APIs für den Zugriff auf die ermittelten Entitäten verwenden möchten.

Die folgende Abfrage gibt beispielsweise alle Mitarbeiter (Employees) und die Abteilung (Department), zu der sie gehören, für einen einzigen Unternehmensbereich (Division), sortiert nach Gehalt (Salary), zurück. Wenn Sie die fünf Mitarbeiter mit den höchsten Gehältern ausgeben und anschließend nur mit Mitarbeitern aus einer einzigen Abteilung in demselben Arbeitsbereich arbeiten möchten, verwenden Sie den folgenden Code:

```

String string_q1 = "SELECT e.name, e.id, e.dept from Employee e WHERE e.dept.division='Manufacturing' ORDER BY e.salary DESC";
Query query1 = em.createQuery(string_q1);
query1.setResultEntityName("AllEmployees");
Iterator results1 = query1.getResultIterator(EmployeeResult.class);
int curEmployee = 0;
System.out.println("Highest paid employees");
while (results1.hasNext() && curEmployee++ < 5) {
    EmployeeResult curEmp = (EmployeeResult) results1.next();
    System.out.println(curEmp);
}

```

```

// Mitarbeiter aus der Ergebnismenge entfernen
em.remove(curEmp);
}

// Änderungen in einer Flush-Operation in die Ergebnismenge schreiben
em.flush();

// Abfrage für den lokalen Arbeitsbereich ohne die entfernten
// Mitarbeiter ausführen
String string_q2 = "SELECT e.name, e.id, e.dept from AllEmployees e WHERE e.dept.name='Hardware'";
Query query2 = em.createQuery(string_q2);
Iterator results2 = query2.getResultIterator(EmployeeResult.class);
System.out.println("Subset list of Employees");
while (results2.hasNext()) {
    EmployeeResult curEmp = (EmployeeResult) results2.next();
    System.out.println(curEmp);
}

```

### **public Object getSingleResult**

Die Methode "getSingleResult" führt eine SELECT-Abfrage durch, die ein einziges Ergebnis zurückgibt.

Wenn in der SELECT-Klausel mehrere Felder definiert sind, ist das Ergebnis ein Objektbereich, in dem jedes Element auf der Ordinalposition in der SELECT-Klausel der Abfrage basiert.

```
String q1 = "SELECT e from Employee e WHERE e.id=100"
Employee e = em.createQuery(q1).getSingleResult();
```

```
String q1 = "SELECT e.name, e.dept from Employee e WHERE e.id=100"
Object[] empData = em.createQuery(q1).getSingleResult();
String empName= (String) empData[0];
Department empDept = (Department) empData[1];
```

### **public Query setResultEntityName(String entityName)**

Die Methode "setResultEntityName(String entityName)" gibt den Namen der Abfrageergebnisentität an.

Jedesmal, wenn die Methode "getResultIterator" oder "getResultMap" aufgerufen wird, wird dynamisch eine Entität mit einer ObjectMap erstellt, in der die Ergebnisse der Abfrage gespeichert werden. Wenn die Entität nicht angegeben oder null ist, werden der Entitäts- und ObjectMap-Name automatisch generiert.

Da alle Abfrageergebnisse für die Dauer einer Transaktion verfügbar sind, kann ein Abfragenname innerhalb einer Transaktion nicht wiederverwendet werden.

### **public Query setPartition(int partitionId)**

Diese Methode legt die Partition fest, an die die Abfrage weitergeleitet wird.

Diese Methode ist erforderlich, wenn die Maps in der Abfrage partitioniert sind und der EntityManager keine Affinität zu einer Stammentitätspartition mit einem einzelnen Schema hat.

Verwenden Sie die Schnittstelle "PartitionManager", um die Anzahl der Partitionen für die BackingMap einer bestimmten Entität festzulegen.

Die folgende Tabelle enthält Beschreibungen weiterer Methoden, die über die Abfrageschnittstelle verfügbar sind.

Tabelle 9. Weitere Methoden

Methodenname	Ergebnis
public Query setMaxResults(int maxResult)	Legt die maximale Anzahl abzurufender Ergebnisse fest.
public Query setFirstResult(int startPosition)	Legt die Position des ersten abzurufenden Ergebnisses fest.
public Query setParameter(String name, Object value)	Bindet ein Argument an einen benannten Parameter.
public Query setParameter(int position, Object value)	Bindet ein Argument an einen positionsgebundenen Parameter.
public Query setFlushMode(FlushModeType flushMode)	Legt den bei der Durchführung der Abfrage zu verwendenden Flush-Modustyp fest, der den im EntityManager festgelegten Flush-Modustyp überschreibt.

## Elemente von eXtreme-Scale-Abfragen

Mit der Abfragesteuerkomponente von eXtreme Scale können Sie eine einzige Abfragesprache verwenden, um den eXtreme-Scale-Cache zu durchsuchen. Diese Abfragesprache kann Java-Objekte abfragen, die in ObjectMap-Objekten und Entity-Objekten gespeichert sind. Verwenden Sie die folgende Syntax, um eine Abfragezeichenfolge zu erstellen.

Eine eXtreme-Scale-Abfrage ist eine Zeichenfolge, die die folgenden Elemente enthält:

- eine SELECT-Klausel, die die zurückzugebenden Objekte oder Werte angibt,
- eine FROM-Klausel, die die Objektsammlungen benennt,
- eine optionale WHERE-Klausel, die Suchprädikate für die Sammlungen enthält,
- eine optionale GROUP-BY- oder HAVING-Klausel (weitere Informationen hierzu finden Sie in der Beschreibung der Aggregationsfunktionen für eXtreme-Scale-Abfragen),
- eine optionale ORDER-BY-Klausel, die die Sortierung der Ergebnissammlung angibt.

Sammlungen von eXtreme-Scale-Objekten werden in Abfragen anhand ihres in der FROM-Klausel der Abfrage angegebenen Namens identifiziert.

Die Elemente der Abfragesprache werden ausführlicher in den folgenden Referenzinformationen beschrieben:

- „Backus-Naur-Form für ObjectGrid-Abfragen“ auf Seite 107
- „Referenzinformationen zu eXtreme-Scale-Abfragen“ auf Seite 98

In den folgenden Informationen werden die Mittel zur Verwendung der API "Query" beschrieben:

- „EntityManager-API "Query"“ auf Seite 94
- „API "ObjectQuery" verwenden“

## API "ObjectQuery" verwenden

Die API "ObjectQuery" stellt Methoden für die Abfrage von Daten im ObjectGrid bereit, die mit der API "ObjectMap" gespeichert wurden. Wenn ein Schema in der

ObjectGrid-Instanz definiert ist, kann die API "ObjectQuery" verwendet werden, um Abfragen für die heterogenen Objekte, die in den ObjectMaps gespeichert sind, zu erstellen und auszuführen.

## Abfrage und ObjectMaps

Sie können eine erweiterte Abfragefunktion für Objekte verwenden, die mit der API "ObjectMap" gespeichert wurden. Diese Abfragen unterstützen den Abruf von Objekten über Attribute ohne Schlüsselfunktion und führen einfache Aggregationen wie sum, avg, min und max für alle Daten aus, die einer Abfrage entsprechen. Anwendungen können eine Abfrage mit der Methode "Session.createObjectQuery" erstellen. Diese Methode gibt ein ObjectQuery-Objekt zurück, das anschließend abgefragt werden kann, um die Abfrageergebnisse zu erhalten. Das Query-Objekt lässt auch die Anpassung der Abfrage zu, bevor die Abfrage ausgeführt wird. Die Abfrage wird automatisch ausgeführt, wenn eine Methode aufgerufen wird, die das Ergebnis zurückgibt.

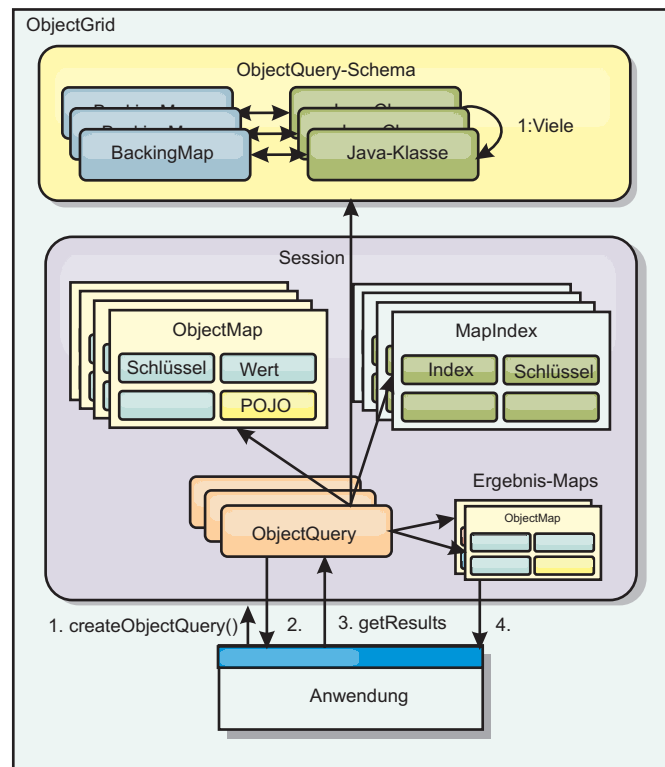


Abbildung 8. Interaktion der Abfrage mit den ObjectGrid-ObjectMaps, Definition eines Schemas für Klassen und Zuordnung des Schemas zu einer ObjectGrid-Map

## ObjectMap-Schema definieren

ObjectMaps werden zum Speichern von Objekten in verschiedenen Formen verwendet und kennen das Format im Großen und Ganzen nicht. Es muss ein Schema im ObjectGrid definiert werden, das das Format der Daten definiert. Ein Schema setzt sich aus den folgenden Teilen zusammen:

- dem Typ des in der ObjectMap gespeicherten Objekts,
- den Beziehungen zwischen ObjectMaps,
- der Methode, für die jede Abfrage auf die Datenattribute in den Objekten zugreifen muss (Feld- oder Eigenschaftsmethoden),

- dem Namen des Primärschlüsselattributs im Objekt.

Einzelheiten hierzu finden Sie im Abschnitt ObjectQuery-Schema konfigurieren.

Ein Beispiel für das Erstellen eines Schemas über das Programm oder mit der ObjectGrid-XML-Deskriptordatei finden Sie im Lernprogramm zu ObjectQuery im Handbuch *Produktübersicht*.

## Objekte mit der API "ObjectQuery" abfragen

Die Schnittstelle "ObjectQuery" unterstützt die Abfrage von Objekten, die keine Entitäten sind, d. h. heterogenen Objekten, die direkt in den ObjectGrid-ObjectMaps gespeichert werden. Die API "ObjectQuery" ist eine einfache Methode für die direkte Suche von ObjectMap-Objekten über die Schlüsselwort- und Indexmechanismen.

Es gibt zwei Methoden für das Abrufen von Ergebnissen aus einer ObjectQuery: getResultIterator und getResultMap.

### Abfrageergebnisse mit getResultIterator abrufen

Abfrageergebnisse sind im Wesentlichen eine Liste mit Attributen. Angenommen, die Abfrage lautet "select a,b,c from X where y=z". Diese Abfrage gibt eine Liste mit Zeilen zurück, zu denen a, b und c gehören. Diese Liste ist eigentlich in einer transaktionsbezogenen Map gespeichert, d. h., Sie müssen jeder Zeile einen künstlichen Schlüssel zuordnen und einen Integer-Wert verwenden, der sich mit jeder Zeile erhöht. Diese Map wird mit der Methode "ObjectQuery.getResultMap()" abgerufen. Sie können auf die Elemente jeder Zeile mit Code wie dem folgenden zugreifen:

```
ObjectQuery q = session.createQuery(
    "select c.id, c.firstName, c.surname from Customer c where c.surname=?1");

q.setParameter(1, "Claus");

Iterator iter = q.getResultIterator();
while(iter.hasNext())
{
    Object[] row = (Object[])iter.next();
    System.out.println("Found a Claus with id "
        + row[objectgrid: 0 ] + ", firstName: "
        + row[objectgrid: 1 ] + ", surname: "
        + row[objectgrid: 2 ]);
}
```

### Abfrageergebnisse mit getResultMap abrufen

Abfrageergebnisse können auch direkte über die Ergebnis-Map abgerufen werden. Das folgende Beispiel zeigt eine Abfrage, die bestimmte Komponenten der übereinstimmenden Customer-Objekte abrufen, und veranschaulicht, wie auf die Ergebniszeilen zugegriffen wird. Wenn Sie das ObjectQuery-Objekt für den Zugriff auf die Daten verwenden, müssen Sie beachten, dass die generierte lange Zeilenkennung verborgen bleibt. Die lange Zeilenkennung ist nur sichtbar, wenn Sie die ObjectMap für den Zugriff auf das Ergebnis verwenden.

Nach Abschluss der Transaktion ist auch diese Map nicht mehr vorhanden. Die Map ist außerdem nur für die verwendete Sitzung sichtbar, d. h. normalerweise nur für den Thread, der sie erstellt hat. Die Map verwendet einen Schlüssel vom Typ "long", der die Zeilen-ID darstellt. Die in der Map gespeicherten Werte haben

den Typ "Object" oder "Object[]", wobei jedes Element dem Typ des Elements in der Klausel SELECT der Abfrage entspricht.

```
ObjectQuery q = em.createQuery(
    "select c.id, c.firstName, c.surname from Customer c where c.surname=?1");
q.setParameter(1, "Claus");
ObjectMap qmap = q.getResultMap();
for(long rowId = 0; true; ++rowId)
{
    Object[] row = (Object[]) qmap.get(new Long(rowId));
    if(row == null) break;
    System.out.println(" I Found a Claus with id " + row[0]
        + ", firstName: " + row[1]
        + ", surname: " + row[2]);
}
```

Beispiele zur Verwendung von ObjectQuery finden Sie im Lernprogramm zur API "ObjectQuery" im Handbuch *Produktübersicht*.

## ObjectQuery-Schema konfigurieren

ObjectQuery stützt sich bei der Durchführung der Semantikprüfung und der Auswertung von Pfadausdrücken auf Schema- bzw. Forminformationen. In diesem Abschnitt wird beschrieben, wie das Schema in XML oder über das Programm definiert werden kann.

### Schema definieren

Das ObjectMap-Schema wird in der ObjectGrid-XML-Implementierungsdeskriptor-datei oder über das Programm mit Hilfe der herkömmlichen eXtreme-Scale-Konfigurationstechniken definiert. Ein Beispiel zum Erstellen eines Schemas finden Sie im Abschnitt „ObjectQuery-Schema konfigurieren“.

Die Schemainformationen beschreiben POJOs (Plain Old Java Objects): aus welchen Attributen sie sich zusammensetzen und welche Typen von Attributen vorhanden sein können, ob die Attribute Primärschlüsselfelder, Beziehungen mit einem einzigen oder mehreren Werten oder bidirektionale Beziehungen sind. Die Schema-informationen weisen ObjectQuery an, Feldzugriff oder Eigenschaftszugriff zu verwenden.

### Abfragbare Attribute

Wenn das Schema im ObjectGrid definiert ist, werden die Objekte im Schema durch Reflexion überwacht, um festzustellen, welche Attribute abgefragt werden können. Sie können die folgenden Attributtypen abfragen:

- primitive Java-Typen, einschließlich Wrappern
- java.lang.String
- java.math.BigInteger
- java.math.BigDecimal
- java.util.Date
- java.sql.Date
- java.sql.Time
- java.sql.Timestamp
- java.util.Calendar
- byte[]
- java.lang.Byte[]
- char[]



- java.lang.Character[]
- J2SE enum

Es können auch andere integrierte serialisierbare Typen als die zuvor genannten in ein Abfrageergebnis, aber nicht in die WHERE- oder FROM-Klausel der Abfrage eingeschlossen werden. Eine Navigation für serialisierbare Attribute ist nicht möglich.

Attributtypen können aus dem Schema ausgeschlossen werden, wenn das Feld bzw. die Eigenschaft statisch oder das Feld transient ist. Da alle Map-Objekte serialisierbar sein müssen, enthält das ObjectGrid nur Attribute aus dem Objekt, die als persistent definiert werden können. Andere Objekte werden ignoriert.

### Feldattribute

Wenn das Schema für den Zugriff auf das Objekt über Felder konfiguriert ist, werden alle serialisierbaren, nicht transienten Felder automatisch in das Schema integriert. Zum Auswählen eines Feldattributs in einer Abfrage verwenden Sie den Feldkennzeichnungsamen aus der Klassendefinition.

Alle öffentlichen, privaten, geschützten und paketgeschützten Felder werden in das Schema eingeschlossen.

### Eigenschaftsattribute

Wenn das Schema für den Zugriff auf das Objekt über Eigenschaften konfiguriert ist, werden alle serialisierbaren Methoden, die den Namenskonventionen für JavaBeans-Eigenschaften entsprechen, automatisch in das Schema eingeschlossen. Zum Auswählen eines Eigenschaftsattributs für die Abfrage verwenden Sie die Namenskonventionen für JavaBeans-Eigenschaften.

Alle öffentlichen, privaten, geschützten und paketgeschützten Eigenschaften werden in das Schema eingeschlossen.

Im folgenden Beispiel werden einer Klasse die folgenden Attribute zum Schema hinzugefügt: name, birthday, valid.

```
public class Person {
    public String getName(){
    private java.util.Date getBirthday(){
    boolean isValid(){
    public NonSerializableObject getData(){
    }
```

Wenn Sie den Kopiermodus "COPY\_ON\_WRITE" verwenden, muss das Schema immer den eigenschaftsbasierten Zugriff verwenden. COPY\_ON\_WRITE erstellt Proxy-Objekte, wenn Objekte aus der Map abgerufen werden, und der Zugriff auf diese Objekte ist nur mit Eigenschaftsmethoden möglich. Erfolgt der Zugriff auf andere Weise wird jedes Abfrageergebnis auf null gesetzt.

### Beziehungen

Jede Beziehung muss in der Schemakonfiguration explizit definiert werden. Die Kardinalität der Beziehung wird automatisch über den Typ des Attributs bestimmt. Wenn das Attribut die Schnittstelle "java.util.Collection" implementiert, ist die Beziehung entweder eine Eins-zu-viele- oder eine Viele-zu-viele-Beziehung.

Anders als Entitätsabfragen dürfen Attribute, die auf andere zwischengespeicherte Objekte verweisen, keine direkten Referenzen auf das Objekt enthalten. Referenzen auf andere Objekte werden als Teil Daten des übergeordneten Objekts serialisiert. Stattdessen muss der Schlüssel im zugehörigen Objekt gespeichert werden.

Beispiel mit einer Viele-zu-eins-Beziehung zwischen einem Kunden (Customer) und einem Auftrag (Order):

**Falsch.**

**Es wird eine Objektreferenzreferenz gespeichert.**

```
public class Customer {
    String customerId;
    Collection<Order> orders;
}
```

```
public class Order {
    String orderId;
    Customer customer;
}
```

**Richtig. Der Schlüssel wird im zugehörigen Objekt gespeichert.**

```
public class Customer {
    String customerId;
    Collection<String> orders;
}
```

```
public class Order {
    String orderId;
    String customer;
}
```

Wenn eine Abfrage ausgeführt wird, die zwei Map-Objekte miteinander verknüpft, wird der schlüssel automatisch dekomprimiert. Die folgende Abfrage gibt beispielsweise Customer-Objekte zurück:

```
SELECT c FROM Order o JOIN Customer c WHERE orderId=5
```

## Indizes verwenden

ObjectGrid verwendet Index-Plug-ins, um Maps Indizes hinzuzufügen. Die Abfragesteuerkomponente integriert automatisch alle Indizes, die in einem Schema-Map-Element des Typs "com.ibm.websphere.objectgrid.plugin.index.HashIndex" definiert sind, wenn die Eigenschaft "rangeIndex" auf "true" gesetzt ist. Wenn der Indextyp nicht "HashIndex" ist und die Eigenschaft "rangeIndex" nicht auf "true" gesetzt ist, wird der Index von der Abfrage ignoriert. Ein Beispiel für das Hinzufügen eines Index zum Schema finden Sie im Lernprogramm zu ObjectQueryLernprogramm zu ObjectQuery im Handbuch *Produktübersicht*.

## EntityManager-API "Query"

Die API "EntityManager" stellt Methoden für die Abfrage von Daten im ObjectGrid bereit, die mit der API "EntityManager" gespeichert wurden. Die API "EntityManager Query" wird verwendet, um Abfragen über eine oder mehrere in eXtreme Scale definierte Entitäten zu erstellen und auszuführen.

## Abfragen und ObjectMaps für Entitäten

In WebSphere Extended Deployment Version 6.1 wurden erweiterte Abfragefunktionen für Entitäten eingeführt, die in eXtreme Scale gespeichert sind. Mit Hilfe dieser Abfragen können Objekte über Attribute ohne Schlüsselfunktion abge-

rufen werden und einfache Spaltenberechnungen wie Summe, Durchschnitt, Minimum und Maximum für alle Daten, die einer Abfrage entsprechen, durchgeführt werden. Anwendungen erstellen eine Abfrage über die API "EntityManager.createQuery". Diese Abfrage gibt ein Query-Objekt zurück, das dann abgefragt werden kann, um die Abfrageergebnisse zu erhalten. Das Query-Objekt lässt auch die Anpassung der Abfrage zu, bevor die Abfrage ausgeführt wird. Die Abfrage wird automatisch ausgeführt, wenn eine Methode aufgerufen wird, die das Ergebnis zurückgibt.

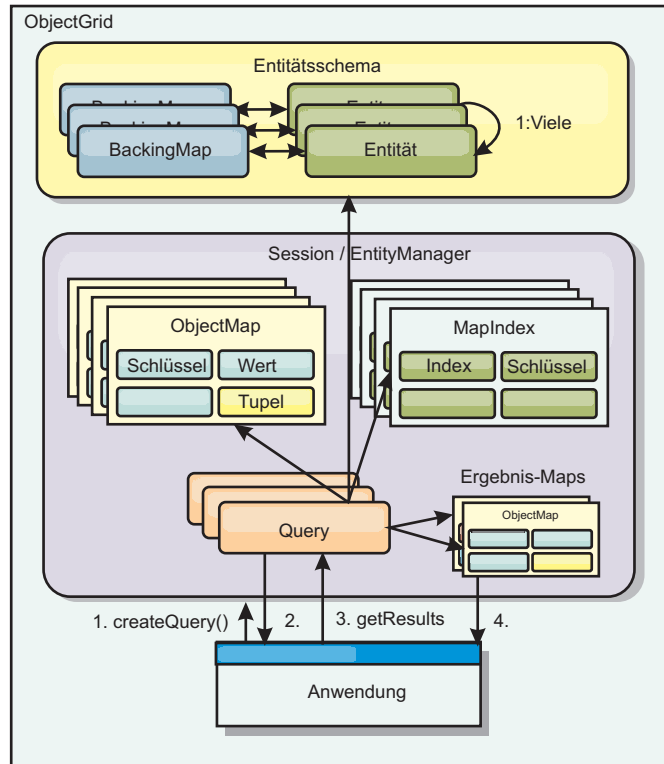


Abbildung 9. Interaktion der Abfrage mit den ObjectGrid-Objekt-Maps, Definition und Zuordnung des Entitätsschemas zu einer ObjectGrid-Map

## Abfrageergebnisse mit der Methode "getResultIterator" abrufen

Abfrageergebnisse sind eine Liste mit Attributen. Wenn die Abfrage "select a,b,c from X where y=z" lautet, wird eine Liste mit Zeilen zurückgegeben, die a, b und c enthalten. Diese Liste wird in einer transaktionsbezogenen Map gespeichert, d. h., Sie müssen jeder Zeile einen künstlichen Schlüssel zuordnen und eine ganze Zahl verwenden, die mit jeder Zeile um eins steigt. Diese Map wird mit der Methode "Query.getResultMap" abgerufen. Die Map enthält Entitätsmetadaten, die jede Zeile in der zugehörigen Map beschreiben. Sie können auf die Elemente jeder Zeile mit Code wie dem folgenden zugreifen:

```
Query q = em.createQuery("select c.id, c.firstName, c.surname from Customer c where c.surname=?1");
q.setParameter(1, "Claus");

Iterator iter = q.getResultIterator();
while(iter.hasNext())
{
    Object[] row = (Object[])iter.next();
    System.out.println("Found a Claus with id " + row[objectgrid: 0 ]
        + ", firstName: " + row[objectgrid: 1 ]
        + ", surname: " + row[objectgrid: 2 ]);
}
```

## Abfrageergebnisse mit "getResultMap" abrufen

Der folgende Code veranschaulicht, wie bestimmte Teile der übereinstimmenden Customer-Objekte abgerufen und auf die zurückgegebenen Zeilen zugegriffen wird. Wenn Sie das Query-Objekt für den Zugriff auf die Daten verwenden, wird die generierte Zeilenkennung vom Typ "long" verdeckt. Die Zeilenkennung vom Typ "long" ist nur sichtbar, wenn Sie die ObjectMap für den Zugriff auf das Ergebnis verwenden. Nach Abschluss der Transaktion wird diese Map entfernt. Die Map ist nur für die verwendete Sitzung sichtbar, d. h. normalerweise nur für den Thread, der sie erstellt hat. Die Map verwendet für den Schlüssel ein Tupel mit einem einzigen Attribut vom Typ "long" mit der Zeilen-ID. Der Wert ist ein weiteres Tupel mit einem Attribut für jede Spalte in der Ergebnismenge.

Der folgende Mustercode veranschaulicht dies:

```
Query q = em.createQuery("select c.id, c.firstName, c.surname from
Customer c where c.surname=?1");
q.setParameter(1, "Claus");
ObjectMap qmap = q.getResultMap();
Tuple keyTuple = qmap.getEntityMetadata().getKeyMetadata().createTuple();
for(long i = 0; true; ++i)
{
    keyTuple.setAttribute(0, new Long(i));
    Tuple row = (Tuple)qmap.get(keyTuple);
    if(row == null) break;
    System.out.println(" I Found a Claus with id " + row.getAttribute(0)
        + ", firstName: " + row.getAttribute(1)
        + ", surname: " + row.getAttribute(2));
}
```

## Abfrageergebnisse mit einem Entitätsergebnisiterator abrufen

Der folgende Code veranschaulicht die Abfrage und die Schleife, mit denen die einzelnen Ergebniszeilen über die herkömmlichen Map-APIs abgerufen werden. Der Schlüssel für die Map ist ein Tupel. Sie müssen also einen der richtigen Typen mit dem Ergebnis der Methode "createTuple" in "keyTuple" erstellen. Versuchen Sie, alle Zeilen mit Zeile-IDs ab 0 aufwärts abzurufen. Wenn die Abfrage null zurückgibt (d. h., der Schlüssel wurde nicht gefunden), wird die Schleife beendet. Sie definieren, dass das erste Attribut von keyTuple der Wert vom Typ "long" ist, den Sie suchen möchten. Der von der get-Methode zurückgegebene Wert ist ebenfalls ein Tupel mit einem Attribut für jede Spalte im Abfrageergebnis. Anschließend extrahieren Sie mit getAttribute jedes Attribut aus dem Werttupel.

Sehen Sie sich das nächste Codefragment an:

```
Query q2 = em.createQuery("select c.id, c.firstName, c.surname from Customer c where c.surname=?1");
q2.setResultEntityName("CustomerQueryResult");
q2.setParameter(1, "Claus");

Iterator iter2 = q2.getResultIterator(CustomerQueryResult.class);
while(iter2.hasNext())
{
    CustomerQueryResult row = (CustomerQueryResult)iter2.next();
    // firstName is the id not the firstName.
    System.out.println("Found a Claus with id " + row.id
        + ", firstName: " + row.firstName
        + ", surname: " + row.surname);
}

em.getTransaction().commit();
```

In der Abfrage ist ein ResultEntityName-Wert angegeben. Dieser Wert teilt der Abfragesteuerkomponente mit, dass Sie jede Zeile einem bestimmten Objekt zuordnen möchten, in diesem Fall CustomerQueryResult. Im Folgenden sehen Sie die Klasse:

```

@Entity
public class CustomerQueryResult {
    @Id long rowId;
    String id;
    String firstName;
    String surname;
};

```

Im ersten Snippet wird jede Abfragezeile als CustomerQueryResult-Objekt und nicht als Object[] zurückgegeben. Die Ergebnisspalten der Abfrage werden dem CustomerQueryResult-Objekt zugeordnet. Die Projektion des Ergebnisses ist zur Laufzeit geringfügig langsamer, aber lesbarer. Abfrageergebnisentitäten sollten beim Start nicht bei eXtreme Scale registriert werden. Wenn die Entitäten registriert werden, wird eine globale Map desselben Namens erstellt, und die Abfrage scheitert mit einem Fehler, der auf den doppelt vorhandenen Map-Namen hinweist.

## Beispielabfragen mit EntityManager

In diesem Abschnitt wird veranschaulicht, wie einfache Abfragen mit der API "EntityManager Query" ausgeführt werden.

Die API "EntityManager Query" ist der SQL anderer Abfragesteuerkomponenten, die Abfragen über Objekte ausführen, sehr ähnlich. Es wird eine Abfrage definiert, und anschließend wird das Ergebnis über verschiedene getResult-Methoden aus der Abfrage abgerufen.

Die folgenden Beispiele beziehen sich auf die Entitäten, die im Lernprogramm zu EntityManager im Handbuch "Produktübersicht" verwendet werden.

## Einfache Abfrage ausführen

In diesem Beispiel werden Kunden (Customer) mit dem Nachnamen "Claus" abgefragt:

```

em.getTransaction().begin();

Query q = em.createQuery("select c from Customer c where c.surname='Claus'");

Iterator iter = q.getResultIterator();
while(iter.hasNext())
{
    Customer c = (Customer)iter.next();
    System.out.println("Found a claus with id " + c.id);
}

em.getTransaction().commit();

```

## Parameter verwenden

Da Sie alle Kunden mit dem Nachnamen Claus suchen möchten, wird ein Parameter zur Angabe des Nachnamens (surname) verwendet, weil Sie diese Abfrage möglicherweise mehrfach verwenden möchten.

### Beispiel für positionsgebundene Parameter

```

Query q = em.createQuery("select c from Customer c where c.surname=?1");
q.setParameter(1, "Claus");

```

Die Verwendung von Parametern ist sehr wichtig, wenn die Abfrage mehrfach verwendet wird. EntityManager muss die Abfragezeichenfolge syntaktisch analysieren und einen Plan für die Abfrage erstellen, was kostenintensiv ist. Wenn Sie einen

Parameter verwenden, stellt EntityManager den Plan für die Abfrage in den Cache, wodurch sich die Ausführungsdauer der Abfrage verkürzt.

Es werden sowohl positionsgebundene als auch benannten Parameter verwendet.

#### Beispiel für benannte Parameter

```
Query q = em.createQuery("select c from Customer c where c.surname=:name");
q.setParameter("name", "Claus");
```

#### Index für die Verbesserung der Leistung verwenden

Wenn es Millionen von Kunden gibt, muss die vorherige Abfrage alle Zeilen in der Map "Customer" durchsuchen. Dies ist nicht sehr effizient. eXtreme Scale stellt jedoch einen Mechanismus für die Definition von Indizes für einzelne Attribute in einer Entität bereit. Die Abfrage verwendet diesen Index gegebenenfalls automatisch, was Anfragen erheblich beschleunigen kann.

Sie können ohne viel Aufwand angeben, welche Attribute indexiert werden sollen, indem Sie einfach die Annotation "@Index" im Entitätsattribut verwenden:

```
@Entity
public class Customer
{
    @Id String id;
    String firstName;
    @Index String surname;
    String address;
    String phoneNumber;
}
```

EntityManager erstellt einen entsprechenden ObjectGrid-Index für das Attribut "surname" in der Entität "Customer", und die Abfragesteuerkomponente verwendet den Index automatisch, was die Abfragezeit erheblich verringert.

#### Seitenaufteilung zur Verbesserung der Leistung verwenden

Wenn es eine Million von Kunden mit dem Namen Claus gibt, wollen Sie wahrscheinlich keine Seite anzeigen, auf der eine Million Kunden angezeigt werden. Wahrscheinlich finden Sie eine Anzeige von jeweils 10 bis 25 Kunden sehr viel angemessener.

Die Abfragemethoden "setFirstResult" und "setMaxResults" helfen hierbei, weil sie nur einen Teil der Ergebnisse zurückgeben.

#### Beispiel für Seitenaufteilung

```
Query q = em.createQuery("select c from Customer c where c.surname=:name");
q.setParameter("name", "Claus");
// Erste Seite anzeigen
q.setFirstResult=1;
q.setMaxResults=25;
displayPage(q.getResultIterator());

// Zweite Seite anzeigen
q.setFirstResult=26;
displayPage(q.getResultIterator());
```

## Referenzinformationen zu eXtreme-Scale-Abfragen

WebSphere eXtreme Scale hat eine eigene eindeutige Sprache, mit der der Benutzer Daten abfragen kann.

## FROM-Klausel in ObjectGrid-Abfragen

Die FROM-Klausel gibt die Sammlungen von Objekten an, auf die die Abfrage angewendet werden soll. Jede Sammlung wird anhand eines abstrakten Schemanamens und einer Identifikationsvariablen, einer so genannten Bereichsvariablen, oder anhand der Deklaration eines Sammlungselements identifiziert, die eine Beziehung mit einem einzelnen oder mehreren Werten und eine Identifikationsvariable angibt.

Konzeptionell ist die Semantik der Abfrage so, dass zuerst eine temporäre Sammlung von Tupeln (R) erstellt wird. Tupel setzen sich aus Elementen aus Sammlungen zusammen, die in der FROM-Klausel angegeben sind. Jedes Tupel enthält ein einziges Element aus jeder der Sammlungen in der FROM-Klausel. Alle möglichen Kombinationen werden auf der Basis der Vorgaben in den Deklarationen der Sammlungselemente erstellt. Wenn ein Schemaname eine Sammlung angibt, für die keine Datensätze im persistenten Speicher vorhanden sind, ist die temporäre Sammlung R leer.

### Beispiele mit FROM

Das Objekt "DeptBean" enthält die Datensätze 10, 20 und 30. Das Objekt "EmpBean" enthält die Datensätze 1, 2 und 3, die sich auf Abteilung (Department) 10 beziehen, und die Datensätze 4 und 5, die sich auf Abteilung 20 beziehen. Abteilung 30 hat keine zugehörigen Mitarbeiterobjekte (employee).

```
FROM DeptBean d, EmpBean e
```

Diese Klausel erstellt eine temporäre Sammlung R, die 15 Tupel enthält.

```
FROM DeptBean d, DeptBean d1
```

Diese Klausel erstellt eine temporäre Sammlung R, die 9 Tupel enthält.

```
FROM DeptBean d, IN (d.emps) AS e
```

Diese Klausel erstellt eine temporäre Sammlung R, die 5 Tupel enthält. Abteilung 30 ist nicht in der temporären Sammlung R enthalten, weil es keine Mitarbeiterobjekte enthält. Abteilung 10 ist dreimal und Abteilung 20 zweimal in der temporären Sammlung R enthalten.

An Stelle von "IN(d.emps) as e" können Sie ein JOIN-Prädikat verwenden:

```
FROM DeptBean d JOIN d.emps as e
```

Nach der Erstellung der temporären Sammlung werden die Suchbedingungen der WHERE-Klausel auf die temporäre Sammlung R angewendet. Das Ergebnis ist eine neue temporäre Sammlung R1. Die ORDER-BY- und SELECT-Klauseln werden auf R1 angewendet, um die endgültige Ergebnismenge zu erhalten.

Eine Identifikationsvariable ist eine Variable, die in der FROM-Klausel über den Operator IN oder den optionalen Operator AS deklariert wird.

```
FROM DeptBean AS d, IN (d.emps) AS e
```

entspricht:

```
FROM DeptBean d, IN (d.emps) e
```

Eine Identifikationsvariable, die als abstrakter Schemaname deklariert wird, ist eine so genannte Bereichsvariable. In der vorherigen Abfrage ist "d" eine Bereichsvariable. Eine Identifikationsvariable, die als Pfadausdruck mit mehreren Werten deklariert wird, ist eine so genannte Sammlungselementdeklaration. Die Werte "d" und "e" im vorherigen Beispiel sind Sammlungselementdeklarationen.

Im Folgenden sehen Sie ein Beispiel für die Verwendung eines Pfadausdrucks mit einem einzelnen Wert in der FROM-Klausel:

```
FROM EmpBean e, IN(e.dept.mgr) as m
```

## SELECT-Klausel in ObjectGrid-Abfragen

Die Syntax der SELECT-Klausel wird im folgenden Beispiel veranschaulicht:

```
SELECT { ALL | DISTINCT } [ Auswahl , ]* Auswahl  
selection ::= { Pfadausdruck_mit_einem_einzelnen_Wert |  
                Identifikationsvariable |  
                OBJECT ( Identifikationsvariable) |  
                Aggregatfunktionen } [[ AS ] id ]
```

Die SELECT-Klausel setzt sich aus einem oder mehreren der folgenden Elemente zusammen: einer einzelnen Identifikationsvariablen, die in der FROM-Klausel definiert ist, einem Pfadausdruck mit einem einzelnen Wert, der in Objektreferenzen oder -werte ausgewertet wird, und einer Aggregatfunktion. Sie können das Schlüsselwort DISTINCT verwenden, um doppelte Referenzen auszuschließen.

Ein skalares Subselect ist ein Subselect, das einen Einzelwert zurückgibt.

### Beispiele mit SELECT

Alle Mitarbeiter (employees) suchen, die mehr als der Mitarbeiter Job verdienen:

```
SELECT OBJECT(e) FROM EmpBean ej, EmpBean eWHERE ej.name = 'John' and  
e.salary > ej.salary
```

Alle Abteilungen (departments) suchen, die einen oder mehrere Mitarbeiter haben, die weniger als 20000 verdienen:

```
SELECT DISTINCT e.dept FROM EmpBean e where e.salary < 20000
```

Eine Abfrage kann einen Pfadausdruck enthalten, der in einen beliebigen Wert ausgewertet wird:

```
SELECT e.dept.name FROM EmpBean e where e.salary < 20000
```

Die vorherige Abfrage gibt eine Sammlung von Namenswerten für Abteilungen zurück, die Mitarbeiter haben, die weniger als 20000 verdienen.

Eine Abfrage kann einen Ergebniswert zurückgeben:

```
SELECT avg(e.salary) FROM EmpBean e
```



Im Folgenden sehen Sie eine Abfrage, die die Namen und Objektreferenzen für unterbezahlte Mitarbeiter zurückgibt:

```
SELECT e.name as name, object(e) as emp from EmpBean e where e.salary < 50000
```

## WHERE-Klausel in ObjectGrid-Abfragen

Die WHERE-Klausel enthält Suchbedingungen, die sich aus den im Folgenden beschriebenen Elementen zusammensetzen. Wenn eine Suchbedingung mit TRUE ausgewertet wird, wird das Tupel der Ergebnismenge hinzugefügt.

### ObjectGrid-Abfrageliterale

Ein Zeichenfolgeliteral wird in einfache Anführungszeichen eingeschlossen. Ein einfaches Anführungszeichen, das in einem Zeichenfolgeliteral enthalten ist, wird durch zwei einfache Anführungszeichen dargestellt, z. B. 'Tom''s'.

Ein numerisches Literal kann jeder der folgenden Werte sein:

- ein genauer Wert wie 57, -957 oder +66,
- ein vom Java-Typ "long" unterstützter Wert,
- ein Dezimalliteral wie 57,5 oder -47,02,
- ein ungefährender numerischer Wert wie 7E3 oder -57,4E-2

Boolesche Literale sind TRUE und FALSE.

Temporale Literale folgen der JDBC-Escape-Syntax auf der Basis des Attributtyps:

- java.util.Date: yyyy-mm-ss
- java.sql.Date: yyyy-mm-ss
- java.sql.Time: hh-mm-ss
- java.sql.Timestamp: yyyy-mm-tt hh:mm:ss.f...
- java.util.Calendar: yyyy-mm-tt hh:mm:ss.f...

Auflistungsliterale (Enum-Literale) werden in der Java-Syntax für Enum-Literale mit dem vollständig qualifizierten Namen der Enum-Klasse ausgedrückt.

### Eingabeparameter für ObjectGrid-Abfragen

Sie können Eingabeparameter über eine Ordinalposition oder über einen Variablenamen angeben. Es wird dringend empfohlen, Abfragen zu schreiben, die Eingabeparameter verwenden, weil die Verwendung von Eingabeparametern die Leistung erhöht, da das ObjectGrid auf diese Weise den Abfrageplan zwischen aktiven Aktionen abfangen kann.

Ein Eingabeparameter kann jeder der folgenden Typen sein: Byte, Short, Integer, Long, Float, Double, BigDecimal, BigInteger, String, Boolean, Char, java.util.Date, java.sql.Date, java.sql.Time, java.sql.Timestamp, java.util.Calendar, ein Enum-Typ von Java SE 5, ein Entitäts- oder POJO-Objekt oder eine binäre Datenzeichenfolge im Format Java byte[].

Ein Eingabeparameter darf keinen Nullwert haben. Wenn Sie nach dem Vorkommen eines Nullwerts suchen möchten, verwenden Sie das Prädikat NULL.

*Positionsgebundene Parameter*

Positionsgebundene Eingabeparameter werden mit einem Fragezeichen, gefolgt von einer positiven Zahl definiert:

?[positive ganze Zahl].

Positionsgebundene Eingabeparameter werden, angefangen bei 1, nummeriert und entsprechen den Argumenten der Abfrage. Deshalb darf eine Abfrage keinen Eingabeparameter enthalten, der die Anzahl der Eingabeargumente überschreitet.

Beispiel: `SELECT e FROM Employee e WHERE e.city = ?1 and e.salary >= ?2`

#### *Benannte Parameter*

Benannte Eingabeparameter werden mit einem Variablennamen im folgenden Format definiert: `:[Parametername]`.

Beispiel: `SELECT e FROM Employee e WHERE e.city = :city and e.salary >= :salary`

### **Prädikat BETWEEN in ObjectGrid-Abfragen**

Das Prädikat BETWEEN bestimmt, ob ein bestimmter Wert zwischen zwei angegebenen Werten liegt.

`expression [NOT] BETWEEN expression-2 AND expression-3`

#### *Beispiel 1*

`e.salary BETWEEN 50000 AND 60000`

entspricht:

`e.salary >= 50000 AND e.salary <= 60000`

#### *Beispiel 2*

`e.name NOT BETWEEN 'A' AND 'B'`

entspricht:

`e.name < 'A' OR e.name > 'B'`

### **Prädikat IN in ObjectGrid-Abfragen**

Das Prädikat IN vergleicht einen Wert mit einer Gruppe von Werten. Sie können das Prädikat IN in einem der folgenden beiden Formen verwenden:

`Ausdruck [NOT] IN ( Subselect )` `Ausdruck [NOT] IN ( Wert1, Wert2, .... )`

Der Wert WertN kann entweder der Literalwert oder ein Eingabeparameter sein. Der Ausdruck kann nicht in einen Referenztyp ausgewertet werden.

#### *Beispiel 1*

`e.salary IN ( 10000, 15000 )`

entspricht:

```
( e.salary = 10000 OR e.salary = 15000 )
```

*Beispiel 2*

```
e.salary IN ( select e1.salary from EmpBean e1 where e1.dept.deptno = 10)
```

entspricht:

```
e.salary = ANY ( select e1.salary from EmpBean e1 where e1.dept.deptno = 10)
```

*Beispiel 3*

```
e.salary NOT IN ( select e1.salary from EmpBean e1 where e1.dept.deptno = 10)
```

entspricht:

```
e.salary <> ALL ( select e1.salary from EmpBean e1 where e1.dept.deptno = 10)
```

### **Prädikat LIKE in ObjectGrid-Abfragen**

Das Prädikat LIKE sucht einen Zeichenfolgewert für ein bestimmtes Muster.

```
Zeichenfolgeausdruck [NOT] LIKE Muster [ ESCAPE Escape-Zeichen ]
```

Der Musterwert ist ein Zeichenfolgeliteral oder eine Parametermarke des Typs "string" (Zeichenfolge), in dem bzw. der das Unterstreichungszeichen ( \_ ) für ein beliebiges einzelnes Zeichen und das Prozentzeichen ( % ) für eine Folge von Zeichen, einschließlich einer leeren Folge stehen kann. Jedes andere Zeichen steht für sich selbst. Das Escape-Zeichen kann verwendet werden, um das Zeichen \_ oder % zu suchen. Das Escape-Zeichen kann als Zeichenfolgeliteral oder als Eingabeparameter angegeben werden.

Wenn der Zeichenfolgeausdruck null ist, ist das Ergebnis unbekannt.

Wenn Zeichenfolgeausdruck und Muster leer sind, ist das Ergebnis "true".

*Beispiel*

```
' ' LIKE ' ' ist true
' ' LIKE '%' ist true
e.name LIKE '12%3' ist true für '123' '12993' und false für '1234'
e.name LIKE 's_me' ist true für 'some' und 'same', false für 'soome'
e.name LIKE '/_foo' escape '/' ist true für '_foo', false für 'afoo'
e.name LIKE '//_foo' escape '/' ist true für '/afoo' und für '/bfoo'
e.name LIKE '///_foo' escape '/' ist true für '/_foo', aber false für '/afoo'
```

### **Prädikat NULL in ObjectGrid-Abfragen**

Das Prädikat NULL prüft, ob Nullwerte vorhanden sind.

```
{Pfadausdruck_mit_einem_einzelnen_Wert | Eingabeparameter} IS [NOT] NULL
```

*Beispiel*

```
e.name IS NULL
e.dept.name IS NOT NULL
e.dept IS NOT NULL
```

### Sammlungsprädikat EMPTY in ObjectGrid-Abfragen

Verwenden Sie das Sammlungsprädikat EMPTY, um zu prüfen, ob eine Sammlung leer ist.

Um festzustellen, ob eine Beziehung mit mehreren Werten leer ist, verwenden Sie die folgende Syntax:

```
Pfadausdruck_mit_Sammlungswert IS [NOT] EMPTY
```

#### *Beispiel*

Alle Abteilungen suchen, die keine Mitarbeiter haben:

```
SELECT OBJECT(d) FROM DeptBean d WHERE d.emps IS EMPTY
```

### Prädikat MEMBER OF in ObjectGrid-Abfragen

Der folgende Ausdruck prüft, ob die Objektreferenz, die mit dem Pfadausdruck mit einem einzelnen Wert oder Eingabeparameter angegeben wurde, zur angegebenen Sammlung gehört. Wenn der Pfadausdruck mit Sammlungswert eine leere Sammlung bezeichnet, ist der Wert des MEMBER-OF-Ausdrucks FALSE.

```
{ Pfadausdruck_mit_einem_einzelnen_Wert | Eingabeparameter } [ NOT ] MEMBER
[ OF ] Pfadausdruck_mit_Sammlungswert
```

#### *Beispiel*

Mitarbeiter suchen, die nicht zu einer Abteilung mit einer bestimmten Nummer gehören:

```
SELECT OBJECT(e) FROM EmpBean e , DeptBean d
WHERE e NOT MEMBER OF d.emps AND d.deptno = ?1
```

Mitarbeiter suchen, deren Manager zu einer Abteilung mit einer bestimmten Nummer gehört:

```
SELECT OBJECT(e) FROM EmpBean e, DeptBean d
WHERE e.dept.mgr MEMBER OF d.emps and d.deptno=?1
```

### Prädikat EXISTS in ObjectGrid-Abfragen

Das Prädikat EXISTS prüft, ob eine in einem Subselect angegebene Bedingung vorhanden ist oder nicht.

```
EXISTS ( Subselect )
```

Das Ergebnis von EXISTS ist "true", wenn das Subselect mindestens einen Wert zurückgibt, andernfalls ist das Ergebnis "false".

Zum Verneinen eines Prädikats EXISTS, stellen Sie dem Prädikat den logischen Operator NOT voran.

#### *Beispiel*

Abteilungen zurückgeben, die mindestens einen Mitarbeiter haben, der mehr als 1000000 verdient:

```
SELECT OBJECT(d) FROM DeptBean d
WHERE EXISTS ( SELECT e FROM IN (d.emps) e WHERE e.salary > 1000000 )
```

Abteilungen zurückgeben, die keine Mitarbeiter haben:

```
SELECT OBJECT(d) FROM DeptBean d
WHERE NOT EXISTS ( SELECT e FROM IN (d.emps) e)
```

Sie können die vorherige Abfrage auch wie im folgenden Beispiel schreiben:

```
SELECT OBJECT(d) FROM DeptBean d WHERE SIZE(d.emps)=0
```

## ORDER-BY-Klausel in ObjectGrid-Abfragen

Die ORDER-BY-Klausel gibt die Reihenfolge der Objekte in der Ergebnissammlung an. Es folgt ein Beispiel:

```
ORDER BY [ Sortierelement ,]* Sortierelement Sortierelement ::= { Pfadausdruck } [
ASC | DESC ]
```

Der Pfadausdruck muss ein Einzelwertfeld angeben, das einen primitiven Typ (byte, short, int, long, float, double, char) oder einen Wrapper-Typ (Byte, Short, Integer, Long, Float, Double, BigDecimal, String, Character, java.util.Date, java.sql.Date, java.sql.Time, java.sql.Timestamp oder java.util.Calendar) hat. Das Sortierelement ASC gibt an, dass die Ergebnisse in aufsteigender Reihenfolge angezeigt werden sollen. Dies ist die Standardeinstellung. Ein Sortierelement DESC gibt an, dass die Ergebnisse in absteigender Reihenfolge angezeigt werden sollen.

*Beispiel*

Abteilungsobjekte zurückgeben und die Abteilungsnummern in absteigender Reihenfolge anzeigen:

```
SELECT OBJECT(d) FROM DeptBean d ORDER BY d.deptno DESC
```

Mitarbeiterobjekte, sortiert nach Abteilungsnummer und Namen zurückgeben:

```
SELECT OBJECT(e) FROM EmpBean e ORDER BY e.dept.deptno ASC, e.name DESC
```

## Aggregationsfunktionen in ObjectGrid-Abfragen

Aggregationsfunktionen arbeiten mit einer Gruppe von Werten, um einen einzelnen skalaren Wert zurückzugeben. Sie können diese Funktionen in SELECT- und Subselect-Methoden verwenden. Das folgende Beispiel veranschaulicht eine Aggregation:

```
SELECT SUM (e.salary) FROM EmpBean e WHERE e.dept.deptno =20
```

Diese Aggregation berechnet das Gesamtgehalt für die Abteilung 20.

Die Aggregationsfunktionen sind AVG, COUNT, MAX, MIN und SUM. Die Syntax einer Aggregationsfunktion wird im folgenden Beispiel veranschaulicht:

```
Aggregationsfunktion ( [ ALL | DISTINCT ] Ausdruck )
```

oder:

COUNT( [ ALL | DISTINCT ] Identifikationsvariable )

Die Option DISTINCT schließt doppelte Werte aus, bevor die Funktion angewendet wird. Die Option ALL ist die Standardoption und schließt keine doppelten Werte aus. Nullwerte werden in den Berechnungen der Aggregatfunktion ignoriert, es sei denn, Sie verwenden die Funktion "COUNT(Identifikationsvariable)", die die Anzahl aller Elemente in der Gruppe zurückgibt.

### Rückgabotyp definieren

Die Funktionen MAX und MIN können auf jeden numerischen, Zeichenfolge- oder Datum/Zeit-Datentyp angewendet werden und geben den entsprechenden Datentyp zurück. Die Funktionen SUM und AVG akzeptieren einen numerischen Typ als Eingabe. Die Funktion AVG gibt den Datentyp "double" zurück. Die Funktion SUM gibt den Datentyp "long" zurück, wenn der Eingabetyp ein ganzzahliger Typ ist, es sei denn, die Eingabe ist ein Java-Typ "BigInteger" (große ganze Zahl). In diesem Fall gibt die Funktion einen Java-Typ "BigInteger" zurück. Die Funktion SUM gibt den Datentyp "double" zurück, wenn der Eingabetyp kein ganzzahliger Typ ist, es sei denn, die Eingabe ist ein Java-Typ "BigDecimal" (große Dezimalzahl). In diesem Fall gibt die Funktion einen Java-Typ "BigDecimal" zurück. Die Funktion COUNT akzeptiert jeden Datentyp mit Ausnahme von Sammlungen und gibt den Datentyp "long" zurück.

Wenn die Funktionen SUM, AVG, MAX und MIN auf eine leere Gruppe angewendet werden, können diese einen Nullwert zurückgeben. Die Funktion COUNT gibt null (0) zurück, wenn sie auf eine leere Gruppe angewendet wird.

### GROUP-BY- und HAVING-Klauseln anwenden

Die Gruppe von Werten, die für die Aggregatfunktion verwendet wird, wird von der Sammlung bestimmt, die das Ergebnis der FROM- und WHERE-Klauseln der Abfrage ist. Sie können die Gruppe in weitere Gruppen einteilen und die Aggregationsfunktion auf jede einzelne Gruppe anwenden. Zum Durchführen dieser Aktion verwenden Sie eine GROUP-BY-Klausel in der Abfrage. Die GROUP-BY-Klausel definiert die Gruppierungselemente, die sich aus einer Liste von Pfadausdrücken zusammensetzen. Jeder Pfadausdruck gibt ein Feld an, das einen primitiven Datentyp (byte, short, int, long, float, double, boolean, char) oder einen Wrapper-Typ (Byte, Short, Integer, Long, Float, Double, BigDecimal, String, Boolean, Character, java.util.Date, java.sql.Date, java.sql.Time, java.sql.Timestamp, java.util.Calendar oder Enum-Typ von Java SE 5) hat.

Das folgende Beispiel veranschaulicht die Verwendung der GROUP-BY-Klausel in einer Abfrage, die das Durchschnittsgehalt für jede Abteilung berechnet:

```
SELECT e.dept.deptno, AVG ( e.salary) FROM EmpBean e GROUP BY e.dept.deptno
```

Bei der Aufteilung einer Gruppe in weitere Gruppen wird ein Nullwert mit einem anderen Nullwert gleich gesetzt.

Gruppen können mit einer HAVING-Klausel gefiltert werden, die die Gruppeneigenschaften prüft, bevor Aggregatfunktionen eingesetzt oder Elemente gruppiert werden. Dieser Filtervorgang gleicht dem, den die WHERE-Klausel für die Tupel (d. h. Datensätze der zurückgegebenen Sammlungswerte) aus der FROM-Klausel durchführt. Im Folgenden sehen Sie ein Beispiel für die HAVING-Klausel:

```
SELECT e.dept.deptno, AVG ( e.salary) FROM EmpBean e
GROUP BY e.dept.deptno
HAVING COUNT(e) > 3 AND e.dept.deptno > 5
```

Diese Abfrage gibt das Durchschnittsgehalt für Abteilungen zurück, die mehr als drei Mitarbeiter haben und deren Abteilungsnummer größer als fünf ist.

Sie können eine HAVING-Klausel ohne eine GROUP-BY-Klausel verwenden. In diesem Fall wird die gesamte Gruppe als eine einzige Gruppe behandelt, auf die die HAVING-Klausel angewendet wird.

## Backus-Naur-Form für ObjectGrid-Abfragen

Im Folgenden finden Sie eine Zusammenfassung der BNF-Notation (Backus-Naur-Form) für ObjectGrid-Abfragen.

*Tabelle 10. Zusammenfassung der BNF-Notation*

Darstellung	Beschreibung
{...}	Gruppierung
[...]	Optionale Konstrukte
<b>Fettschrift</b>	Schlüsselwörter
*	0 oder mehr
	Alternativen

```
ObjectGrid QL ::=select_clause from_clause [where_clause] [group_by_clause]
[having_clause] [order_by_clause]
from_clause ::=FROM identification_variable_declaration
[,identification_variable_declaration]*
identification_variable_declaration ::=collection_member_declaration |
range_variable_declaration
collection_member_declaration ::=IN ( collection_valued_path_expression |
single_valued_navigation) [AS] identifier | [LEFT [OUTER]
| INNER] JOIN collection_valued_path_expression |
single_valued_navigation [AS] identifier
range_variable_declaration ::=abstract_schema_name [AS] identifier
single_valued_path_expression ::= {single_valued_navigation | identification_variable}.
{ state_field | state_field.value_object_attribute } | single_valued_navigation
single_valued_navigation ::=identification_variable.[ single_valued_association_field. ]*
single_valued_association_field
collection_valued_path_expression ::=identification_variable.[
single_valued_association_field. ]* collection_valued_association_field
select_clause ::= SELECT [DISTINCT] [ selection , ]* selection
selection ::= {single_valued_path_expression | identification_variable | OBJECT
( identification_variable) | aggregate_functions } [[ AS ] id ]
order_by_clause ::= ORDER BY [ {identification_variable.[ single_valued_association_field.
]*state_field} [ASC|DESC],]* {identification_variable.[
single_valued_association_field. ]*state_field}[ASC|DESC]
where_clause ::= WHERE conditional_expression
conditional_expression ::= conditional_term | conditional_expression OR conditional_term
conditional_term ::= conditional_factor | conditional_term AND conditional_factor
conditional_factor ::= [NOT] conditional_primary
conditional_primary ::= simple_cond_expression | (conditional_expression)
simple_cond_expression ::= comparison_expression | between_expression | like_expression |
in_expression | null_comparison_expression | empty_collection_comparison_expression |
exists_expression | collection_member_expression
between_expression ::= numeric_expression [NOT] BETWEEN numeric_expression
AND numeric_expression | string_expression [NOT] BETWEEN
string_expression AND string_expression | datetime_expression [NOT]
BETWEEN datetime_expression AND datetime_expression
```

```

in_expression ::= identification_variable.[ single_valued_association_field. ]state_field
  [ *NOT ] IN { (subselect) | ( atom ,)* atom }
atom ::= { string_literal | numeric_literal | input_parameter }
like_expression ::= string_expression [ NOT ] LIKE { string_literal | input_parameter }
  [ ESCAPE { string_literal | input_parameter } ]
null_comparison_expression ::= { single_valued_path_expression | input_parameter } IS
  [ NOT ] NULL
empty_collection_comparison_expression ::= collection_valued_path_expression IS
  [ NOT ] EMPTY
collection_member_expression ::= { single_valued_path_expression | input_parameter } [
  NOT ] MEMBER [ OF ] collection_valued_path_expression
exists_expression ::= EXISTS { (subselect) }
subselect ::= SELECT [ { ALL | DISTINCT } ] subselection from_clause
  [ where_clause ] [ group_by_clause ] [ having_clause ]
subselection ::= { single_valued_path_expression | identification_variable |
  aggregate_functions }
group_by_clause ::= GROUP BY [ single_valued_path_expression , ] *
  single_valued_path_expression
having_clause ::= HAVING conditional_expression
comparison_expression ::= numeric_expression comparison_operator { numeric_expression
  | { SOME | ANY | ALL } (subselect) } | string_expression
  comparison_operator {
string_expression | { SOME | ANY | ALL } (subselect) } |
datetime_expression comparison_operator {
datetime_expression { SOME | ANY | ALL } (subselect) } |
boolean_expression { = | <> } {
boolean_expression { SOME | ANY | ALL } (subselect) } |
entity_expression { = | <> } {
entity_expression { SOME | ANY | ALL } (subselect) }
comparison_operator ::= = | > | >= | < | <= | <>
string_expression ::= string_primary | (subselect)
string_primary ::= state_field_path_expression | string_literal | input_parameter |
  functions_returning_strings
datetime_expression ::= datetime_primary | (subselect)
datetime_primary ::= state_field_path_expression | string_literal | long_literal
  | input_parameter | functions_returning_datetime
boolean_expression ::= boolean_primary | (subselect)
boolean_primary ::= state_field_path_expression | boolean_literal | input_parameter
entity_expression ::= single_valued_association_path_expression |
  identification_variable | input_parameter
numeric_expression ::= simple_numeric_expression | (subselect)
simple_numeric_expression ::= numeric_term | numeric_expression (+|-) numeric_term
numeric_term ::= numeric_factor | numeric_term (*|/) numeric_factor
numeric_factor ::= {+|-} numeric_primary
numeric_primary ::= single_valued_path_expression | numeric_literal |
  ( numeric_expression ) | input_parameter | functions
aggregate_functions :=
AVG([ALL|DISTINCT] identification_variable.[
  single_valued_association_field. ]*state_field) |
COUNT([ALL|DISTINCT] {single_valued_path_expression |
  identification_variable}) |
MAX([ALL|DISTINCT] identification_variable.[
  single_valued_association_field. ]*state_field) |
MIN([ALL|DISTINCT] identification_variable.[
  single_valued_association_field. ]*state_field) |
SUM([ALL|DISTINCT] identification_variable.[
  single_valued_association_field. ]*state_field)
functions ::=
ABS (simple_numeric_expression) |

```



**CONCAT** (string\_primary , string\_primary) |  
**LOWER** (string\_primary) |  
**LENGTH**(string\_primary) |  
**LOCATE**(string\_primary, string\_primary [, simple\_numeric\_expression]) |  
**MOD** (simple\_numeric\_expression, simple\_numeric\_expression) |  
**SIZE** (collection\_valued\_path\_expression) |  
**SQRT** (simple\_numeric\_expression) |  
**SUBSTRING** (string\_primary, simple\_numeric\_expression[, simple\_numeric\_expression]) |  
**UPPER** (string\_primary) |  
**TRIM** ([[**LEADING** | **TRAILING** | **BOTH**] [trim\_character]  
**FROM**] string\_primary)

## Optimierung der Abfrageleistung

Verwenden Sie zum Optimieren der Leistung Ihrer Abfragen die folgenden Techniken und Tipps.

### Parameter verwenden

Wenn eine Abfrage ausgeführt wird, muss die Abfragezeichenfolge syntaktisch analysiert und ein Plan für die Abfrage entwickelt werden, was beides kostenintensiv sein kann. WebSphere eXtreme Scale führt die Zwischenspeicherung von Abfrageplänen über die Abfragezeichenfolge durch. Da der Cache eine begrenzte Größe hat, ist es wichtig, Abfragezeichenfolgen nach Möglichkeit wiederzuverwenden. Die Verwendung benannter Parameter und positionsgebundener Parameter kann durch die Förderung der Wiederverwendung von Abfrageplänen ebenfalls zu einer Leistungsverbesserung beitragen.

Beispiel für positionsgebundene Parameter: Query q = em.createQuery("select c from Customer c where c.surname=?1"); q.setParameter(1, "Claus");

### Indizes verwenden

Eine ordnungsgemäße Indexierung in einer Map kann erhebliche Auswirkungen auf die Abfrageleistung haben, auch wenn die Indexierung einen gewissen Einfluss auf die Gesamtleistung der Map hat. Ohne Indexierung der an Abfragen beteiligten Objektattribute führt die Abfragesteuerkomponente eine Tabellensuche für jedes Attribut durch. Die Tabellensuche ist die kostenintensivste Operation während eines Abfragelaufs. Die Indexierung der an Abfragen beteiligten Objektattribute ermöglicht der Abfragesteuerkomponente, unnötige Tabellensuchen zu vermeiden, was die Gesamtabfrageleistung verbessert. Wenn die Anwendung so konzipiert ist, dass sie häufig Abfragen in Maps durchführt, in denen hauptsächlich Leseoperationen durchgeführt werden, konfigurieren Sie Indizes für die an der Abfrage beteiligten Objektattribute. Wenn die Map hauptsächlich aktualisiert wird, müssen Sie einen Kompromiss zwischen Verbesserung der Abfrageleistung und Indexierungsaufwand finden. Weitere Informationen hierzu finden Sie im Abschnitt „Indexierung“ auf Seite 121.

Wenn POJOs (Plain Old Java Objects) in einer Map gespeichert werden, kann durch eine ordnungsgemäße Indexierung eine Java-Reflexion vermieden werden. Im folgenden Beispiel ersetzt die Abfrage die WHERE-Klausel mit Bereichssuche, wenn für das Feld "budget" ein Index erstellt wurde. Andernfalls scannt die Abfrage die gesamte Map und wertet die WHERE-Klausel aus, indem sie zuerst das Budget durch Java-Reflexion abrufen und anschließend das Budget mit dem Wert 50000 vergleicht:

```
SELECT d FROM DeptBean d WHERE d.budget=50000
```

Im Abschnitt „Abfrageplan“ wird ausführlich beschrieben, wie Sie einzelne Abfragen am besten optimieren und wie sich verschiedene Syntax, Objektmodelle und Indizes auf die Abfrageleistung auswirken können.

## Seitenaufteilung verwenden

In Client/Server-Umgebungen überträgt die Abfragesteuerkomponente die vollständige Ergebnis-Map an den Client. Die zurückgegebenen Daten müssen in angemessene Blöcke unterteilt werden. Die Schnittstellen "EntityManager Query" und "ObjectMap ObjectQuery" unterstützen beide die Methoden "setFirstResult" und "setMaxResults", mit denen der Abfrage ermöglicht wird, einen Teil der Ergebnisse zurückzugeben.

## Primitive Werte an Stelle von Entitäten zurückgeben

Mit der API "EntityManager Query" werden Entitäten als Abfrageparameter zurückgegeben. Die Abfragesteuerkomponente gibt die Schlüssel für diese Entitäten an den Client zurück. Wenn der Client mit dem Iterator aus der Methode "getResultIterator" über diese Entitäten iteriert, wird jede Entität automatisch dekomprimiert und so verwaltet, als wäre sie mit der Methode "find" der Schnittstelle "EntityManager" erstellt worden. Der vollständige Entitäts-Graph wird aus der Entitäts-ObjectMap im Client erstellt. Die Wertattribute der Entität und alle zugehörigen Entitäten werden aufgelöst.

Um die Erstellung dieses kostenintensiven Graphen zu vermeiden, ändern Sie die Abfrage so, dass sie die einzelnen Attribute mit Pfadnavigation zurückgibt.

Beispiel:

```
// Gibt eine Entität zurück.  
SELECT p FROM Person p  
// Gibt die Attribute SELECT p.name, p.address.street, p.address.city, p.gender FROM Person p zurück.
```

## Abfrageplan

Alle Abfragen von eXtreme Scale haben einen Abfrageplan. Der Plan beschreibt, wie die Abfragesteuerkomponente mit ObjectMaps und Indizes interagiert. Zeigen Sie den Abfrageplan an, um festzustellen, ob die Abfragezeichenfolge oder Indizes ordnungsgemäß verwendet werden. Der Abfrageplan kann auch verwendet werden, um die Unterschiede zu erkennen, die geringfügige Änderungen einer Abfragezeichenfolge in der Art und Weise, wie eXtreme Scale eine Abfrage ausführt, bewirken.

Der Abfrageplan kann auf die folgenden beiden Arten angezeigt werden:

- Mit der Methode "getplan" der APIs "EntityManager Query" und "ObjectQuery"
- Diagnose-Trace für ObjectGrid

## Methode "getPlan"

Die Methode "getPlan" in den Schnittstellen "ObjectQuery" und "Query" gibt eine Zeichenfolge zurück, die den Abfrageplan beschreibt. Diese Zeichenfolge kann in der Standardausgabe oder in einem Protokoll angezeigt werden. Anmerkung: In einer verteilten Umgebung wird die Methode "getPlan" nicht für den Server ausgeführt und gibt keine definierten Indizes zurück. Zum Anzeigen des Plans auf dem Server verwenden Sie einen Agenten.

## Trace für Abfrageplan

Der Abfrageplan kann über einen ObjectGrid-Trace angezeigt werden. Zum Aktivieren des Trace für den Abfrageplan verwenden Sie die folgende Trace-Spezifikation:

```
QueryEnginePlan=debug=enabled
```

Einzelheiten zum Aktivieren des Trace und zum Auffinden der Trace-Protokoll-dateien finden Sie im Abschnitt „Protokolle und Trace“ auf Seite 291.

## Beispiele für Abfragepläne

Im Abfrageplan wird das Wort "for" verwendet, um anzuzeigen, dass die Abfrage durch eine ObjectMap-Sammlung oder durch eine abgeleitete Sammlung, wie z. B. "q2.getEmps(), q2.dept" oder eine temporäre Sammlung, die von einer inneren Schleife zurückgegeben wird, iteriert. Wenn die Sammlung aus einer ObjectMap stammt, zeigt der Abfrageplan an, ob eine sequenzielle Suche (angegeben mit INDEX SCAN), einen eindeutigen Index oder einen nicht eindeutigen Index verwendet wird. Der Abfrageplan verwendet eine Filterzeichenfolge, um die Bedingungsausdrücke aufzulisten, die auf eine Sammlung angewendet werden.

Ein kartesisches Produkt wird in der Objektanfrage gewöhnlich nicht verwendet. Die folgende Abfrage scannt die gesamte Map "EmpBean" in der äußeren Schleife und die gesamte Map "DeptBean" in der inneren Schleife:

```
SELECT e, d FROM EmpBean e, DeptBean d
```

Plan trace:

```
for q2 in EmpBean ObjectMap using INDEX SCAN
  for q3 in DeptBean ObjectMap using INDEX SCAN
    returning new Tuple( q2, q3 )
```

Die folgende Abfrage ruft alle Mitarbeiternamen (employee) aus einer bestimmten Abteilung (department) ab, indem sie die Map "EmpBean" sequenziell durchsucht, um ein employee-Objekt abzurufen. Über das employee-Objekt navigiert die Abfrage zum zugehörigen department-Objekt und wendet den Filter "d.no=1" an. In diesem Beispiel hat jeder Mitarbeiter nur eine Referenz auf ein department-Objekt, so dass die innere Schleife nur ein einziges Mal ausgeführt wird:

```
SELECT e.name FROM EmpBean e JOIN e.dept d WHERE d.no=1
```

Plan trace:

```
for q2 in EmpBean ObjectMap using INDEX SCAN
  for q3 in q2.dept
    filter ( q3.getNo() = 1 )
    returning new Tuple( q2.name )
```

Die folgende Abfrage entspricht der vorherigen Abfrage. Sie ist jedoch schneller, weil sie das Ergebnis über die Verwendung eines eindeutigen Index, der für die Primärschlüsselfeldnummer "DeptBean" auf ein einziges department-Objekt eingrenzt. Über das department-Objekt navigiert die Abfrage zu den zugehörigen employee-Objekten, um die Namen abzurufen:

```
SELECT e.name FROM DeptBean d JOIN d.emps e WHERE d.no=1
```

Plan trace:

```

for q2 in DeptBean ObjectMap using UNIQUE INDEX key=(1)
  for q3 in q2.getEmps()
    returning new Tuple( q3.name )

```

Die folgende Abfrage sucht alle Mitarbeiter (employee), die für die Entwicklung (development) oder den Vertrieb (sales) arbeiten. Die Abfrage scannt die gesamte Map "EmpBean" und führt weitere Filterungen durch, indem Sie die Ausdrücke "d.name = 'Sales'" und "d.name='Dev'" auswertet:

```

SELECT e FROM EmpBean e, in (e.dept) d WHERE d.name = 'Sales'
or d.name='Dev'

```

Plan trace:

```

for q2 in EmpBean ObjectMap using INDEX SCAN
  for q3 in q2.dept
    filter (( q3.getName() = Sales ) OR ( q3.getName() = Dev ) )
    returning new Tuple( q2 )

```

Die folgende Abfrage entspricht der vorherigen Abfrage, führt aber einen anderen Abfrageplan aus und verwendet den über das Feld "name" erstellten Bereichsindex. Im Allgemeinen ist diese Abfrage schneller, weil der Index über das Feld "name" verwendet wird, um die department-Objekte einzugrenzen, was schnell ist, wenn es nur wenige Entwicklungs- oder Vertriebsabteilungen gibt.

```

SELECT e FROM DeptBean d, in(d.emps) e WHERE d.name='Dev' or d.name='Sales'

```

Plan trace:

IteratorUnionIndex of

```

  for q2 in DeptBean ObjectMap using INDEX on name = (Dev)
    for q3 in q2.getEmps()

```

```

  for q2 in DeptBean ObjectMap using INDEX on name = (Sales)
    for q3 in q2.getEmps()

```

Die folgende Abfrage sucht Abteilungen, die keine Mitarbeiter haben:

```

SELECT d FROM DeptBean d WHERE NOT EXISTS(select e from d.emps e)

```

Plan trace:

```

for q2 in DeptBean ObjectMap using INDEX SCAN
  filter ( NOT EXISTS ( correlated collection defined as

    for q3 in q2.getEmps()
      returning new Tuple( q3 )

    returning new Tuple( q2 )

```

Die folgende Abfrage entspricht der vorherigen Abfrage, verwendet aber die Skalarfunktion SIZE. Diese Abfrage hat eine ähnliche Leistung, ist aber einfacher zu schreiben:

```

SELECT d FROM DeptBean d WHERE SIZE(d.emps)=0
for q2 in DeptBean ObjectMap using INDEX SCAN
  filter (SIZE( q2.getEmps()) = 0 )
  returning new Tuple( q2 )

```

Das folgende Beispiel zeigt eine weitere einfachere Methode, dieselbe Abfrage wie zuvor mit ähnlicher Leistung zu schreiben:

```
SELECT d FROM DeptBean d WHERE d.emps IS EMPTY
```

Plan trace:

```
for q2 in DeptBean ObjectMap using INDEX SCAN
  filter ( q2.getEmps() IS EMPTY )
  returning new Tuple( q2 )
```

Die folgende Abfrage sucht alle Mitarbeiter mit einer Privatadresse, die mindestens einer der Adressen des Mitarbeiters entspricht, dessen Name mit dem Wert des Parameters übereinstimmt. Die innere Schleife ist nicht von der äußeren Schleife abhängig. Die Abfrage führt die innere Schleife ein einziges Mal aus:

```
SELECT e FROM EmpBean e WHERE e.home = any (SELECT e1.home FROM EmpBean e1
  WHERE e1.name=?1)
for q2 in EmpBean ObjectMap using INDEX SCAN
  filter ( q2.home =ANY      temp collection defined as
    for q3 in EmpBean ObjectMap using INDEX on name = ( ?1)
    returning new Tuple( q3.home      )
  )
  returning new Tuple( q2 )
```

Die folgende Abfrage entspricht der vorherigen Abfrage, hat aber eine Unterabfrage mit Korrelationsbezug. Außerdem wird die innere Schleife mehrfach ausgeführt:

```
SELECT e FROM EmpBean e WHERE EXISTS(SELECT e1 FROM EmpBean e1 WHERE
  e.home=e1.home and e1.name=?1)
```

Plan trace:

```
for q2 in EmpBean ObjectMap using INDEX SCAN
  filter ( EXISTS (      correlated collection defined as
    for q3 in EmpBean ObjectMap using INDEX on name = (?1)
    filter ( q2.home = q3.home )
    returning new Tuple( q3      )
  )
  returning new Tuple( q2 )
```

## Abfrageoptimierung mit Indizes

Durch die ordnungsgemäße Definition und Verwendung von Indizes kann die Abfrageleistung erheblich verbessert werden.

eXtreme-Scale-Abfragen können integrierte HashIndex-Plug-ins verwenden, um die Leistung von Abfragen zu verbessern. Indizes können für Entitäts- oder Objektattribute definiert werden. Die Abfragesteuerkomponente verwendet automatisch die definierten Indizes, wenn in der WHERE-Klausel eine der folgenden Zeichenfolgen verwendet wird:

- ein Vergleichsausdruck mit den folgenden Operatoren: =, <, >, <= oder >= (jeder Vergleichsausdruck mit Ausnahme von ungleich (<>),
- ein BETWEEN-Ausdruck,
- Konstanten oder einfache Bedingungen als Operanden des Ausdrucks.

## Voraussetzungen

Indizes müssen die folgenden Voraussetzungen erfüllen, wenn sie in einer Abfrage verwendet werden:

- Alle Indizes müssen das integrierte HashIndex-Plug-in verwenden.

- Alle Indizes müssen statisch definiert sein. Dynamische Indizes werden nicht unterstützt.
- Die Annotation "@Index" kann verwendet werden, um statische HashIndex-Plug-ins automatisch zu erstellen.
- Für alle Einzelattributindizes muss die Eigenschaft "RangeIndex" auf "true" gesetzt sein.
- Für alle zusammengesetzten Indizes muss die Eigenschaft "RangeIndex" auf "false" gesetzt sein.
- Für alle Assoziations- oder Beziehungsindizes muss die Eigenschaft "RangeIndex" auf "false" gesetzt sein.

Eine effizientere Methode für die Suche zwischengespeicherter Objekte ist im Abschnitt „Zusammengesetzter Hash-Index“ auf Seite 127 beschrieben.

### Hinweise zur Auswahl eines Index

Ein Index kann manuell über die Methode "setHint" in den Schnittstellen "Query" und "ObjectQuery" mit der Konstanten HINT\_USEINDEX ausgewählt werden. Dies kann hilfreich sein, wenn eine Abfrage für die Verwendung des Index mit der besten Leistung optimiert wird.

### Abfragebeispiele, in denen Attributindizes verwendet werden

In den folgenden Beispielen werden einfache Bedingungen verwendet: e.empid, e.name, e.salary, d.name, d.budget und e.isManager. In den Beispielen wird davon ausgegangen, dass Indizes über die Felder "name", "salary" und "budget" einer Entität oder eines Wertobjekts definiert werden. Das Feld "empid" ist ein Primärschlüssel, und für "isManager" ist kein Index definiert.

Die folgende Abfrage verwendet beide Indizes über die Felder "name" und "salary". Sie gibt alle Mitarbeiter (employees) zurück, deren Name dem Wert des ersten Parameters oder deren Gehalt (salary) dem Wert des zweiten Parameters entspricht:

```
SELECT e FROM EmpBean e where e.name=?1 or e.salary=?2
```

Die folgende Abfrage verwendet beide Indizes über die Felder "name" und "budget". Die Abfrage gibt alle Abteilungen (departments) mit dem Namen 'DEV' und einem Budget größer als 2000 zurück.

```
SELECT d FROM DeptBean d where d.name='DEV' and d.budget>2000
```

Die folgende Abfrage gibt alle Mitarbeiter mit einem Gehalt größer als 3000 zurück, deren isManager-Flag-Wert dem Wert des Parameters entspricht. Die Abfrage verwendet den Index, der über das Feld "salary" definiert wurde, und führt eine weitere Filterung durch, indem sie den folgenden Vergleichsausdruck auswertet: e.isManager=?1.

```
SELECT e FROM EmpBean e where e.salary>3000 and e.isManager=?1
```

Die folgende Abfrage sucht alle Mitarbeiter, deren Gehalt höher ist als der erste Parameter, bzw. alle Mitarbeiter, die Manager sind. Obwohl für das Feld "salary" ein Index definiert ist, scannt die Abfrage den integrierten Index, der über die Primärschlüssel des Felds "EmpBean" erstellt wurde, und wertet den folgenden Ausdruck aus: e.salary>?1 or e.isManager=TRUE.

```
SELECT e FROM EmpBean e WHERE e.salary>?1 or e.isManager=TRUE
```

Die folgende Abfrage gibt Mitarbeiter zurück, deren Name den Buchstaben a enthält. Obwohl für das Feld "name" ein Index definiert ist, verwendet die Abfrage den Index nicht, weil das Feld "name" im LIKE-Ausdruck verwendet wird.

```
SELECT e FROM EmpBean e WHERE e.name LIKE '%a%'
```

Die folgende Abfrage sucht alle Mitarbeiter, deren Name nicht "Smith" ist. Obwohl für das Feld "name" ein Index definiert ist, verwendet die Abfrage den Index nicht, weil die Abfrage den Vergleichsoperator ungleich (<>) verwendet.

```
SELECT e FROM EmpBean e where e.name<>'Smith'
```

Die folgende Abfrage sucht alle Abteilungen, deren Budget kleiner ist als der Wert des Parameters und deren Mitarbeitergehälter größer als 3000 sind. Die Abfrage verwendet einen Index für das Gehalt (salary), aber keinen Index für das Budget, weil dept.budget keine einfache Bedingung ist. Die dept-Objekte werden aus der Sammlung e abgeleitet. Sie müssen den Budgetindex nicht verwenden, um dept-Objekte zu suchen.

```
SELECT dept from EmpBean e, in (e.dept) dept where e.salary>3000 and dept.budget<?
```

Die folgende Abfrage sucht alle Mitarbeiter, deren Gehälter größer sind als die Gehälter der Mitarbeiter mit empid 1, 2 und 3. Der Index für das Feld "salary" wird nicht verwendet, weil der Vergleich eine Unterabfrage enthält. Der empid-Wert ist jedoch ein Primärschlüssel und wird für eine eindeutige Indextsuche verwendet, weil für alle Primärschlüssel ein integrierter Index definiert ist.

```
SELECT e FROM EmpBean e WHERE e.salary > ALL (SELECT e1.salary FROM EmpBean e1 WHERE e1.empid=1 or e1.empid =2 or e1.empid=99)
```

Um zu prüfen, ob der Index von der Abfrage verwendet wird, können Sie den Abschnitt „Abfrageplan“ auf Seite 110 verwenden. Im Folgenden sehen Sie einen Beispielabfrageplan für die vorherige Abfrage:

```
for q2 in EmpBean ObjectMap using INDEX SCAN
  filter ( q2.salary >ALL temp collection defined as
    IteratorUnionIndex of
      for q3 in EmpBean ObjectMap using UNIQUE INDEX key=(1)
      )
      for q3 in EmpBean ObjectMap using UNIQUE INDEX key=(2)
      )
      for q3 in EmpBean ObjectMap using UNIQUE INDEX key=(99)
      )
  returning new Tuple( q3.salary )
returning new Tuple( q2 )

for q2 in EmpBean ObjectMap using RANGE INDEX on salary with range(3000,)
  for q3 in q2.dept
  filter ( q3.budget < ?1 )
  returning new Tuple( q3 )
```

## Attribute indexieren

Indizes können über einen Einzelattributtyp mit den zuvor definierten Einschränkungen definiert werden.

### Entitätsindizes mit @Index definieren

Wenn Sie einen Index für eine Entität definieren möchten, definieren Sie einfach eine Annotation:

#### Entitäten mit Annotationen

```
@Entity
public class Employee {
    @Id int empid;
    @Index String name
    @Index double salary
    @ManyToOne Department dept;
}
@Entity
public class Department {
    @Id int deptid;
    @Index String name;
    @Index double budget;
    boolean isManager;
    @OneToMany Collection<Employee> employees;
}
```

### Mit XML

Indizes können auch mit XML definiert werden:

#### Entitäten ohne Annotationen

```
public class Employee {
    int empid;
    String name
    double salary
    Department dept;
}

public class Department {
    int deptid;
    String name;
    double budget;
    boolean isManager;
    Collection employees;
}
```

#### ObjectGrid-XML mit Attributindizes

```
<?xml version="1.0" encoding="UTF-8"?>
<objectGridConfig xmlns:xsi="http://www.w3.org/2001/XMLSchema-instance"
xsi:schemaLocation="http://ibm.com/ws/objectgrid/config ../objectGrid.xsd"
xmlns="http://ibm.com/ws/objectgrid/config">
<objectGrids>
<objectGrid name="DepartmentGrid" entityMetadataXMLFile="entity.xml">
<backingMap name="Employee" pluginCollectionRef="Emp"/>
<backingMap name="Department" pluginCollectionRef="Dept"/>
</objectGrid>
</objectGrids>
<backingMapPluginCollections>
<backingMapPluginCollection id="Emp">
<bean id="MapIndexPlugin" className="com.ibm.websphere.objectgrid.plugins.index.HashIndex">
<property name="Name" type="java.lang.String" value="Employee.name"/>
<property name="AttributeName" type="java.lang.String" value="name"/>
<property name="RangeIndex" type="boolean" value="true"
description="Ranges are must be set to true for attributes." />
</bean>
<bean id="MapIndexPlugin" className="com.ibm.websphere.objectgrid.plugins.index.HashIndex">
<property name="Name" type="java.lang.String" value="Employee.salary"/>
<property name="AttributeName" type="java.lang.String" value="salary"/>
<property name="RangeIndex" type="boolean" value="true">
```



```

description="Ranges are must be set to true for attributes." />
</bean>
</backingMapPluginCollection>
<backingMapPluginCollection id="Dept">
<bean id="MapIndexPlugin" className="com.ibm.websphere.objectgrid.plugins.index.HashIndex">
<property name="Name" type="java.lang.String" value="Department.name"/>
<property name="AttributeName" type="java.lang.String" value="name"/>
<property name="RangeIndex" type="boolean" value="true"
description="Ranges are must be set to true for attributes." />
</bean>
<bean id="MapIndexPlugin" className="com.ibm.websphere.objectgrid.plugins.index.HashIndex">
<property name="Name" type="java.lang.String" value="Department.budget"/>
<property name="AttributeName" type="java.lang.String" value="budget"/>
<property name="RangeIndex" type="boolean" value="true"
description="Ranges are must be set to true for attributes." />
</bean>
</backingMapPluginCollection>
</backingMapPluginCollections>
</objectGridConfig>

```

#### Entitäts-XML

```

<?xml version="1.0" encoding="UTF-8"?>
<entity-mappings xmlns="http://ibm.com/ws/projector/config/emd"
xmlns:xsi="http://www.w3.org/2001/XMLSchema-instance"
xsi:schemaLocation="http://ibm.com/ws/projector/config/emd ./emd.xsd">

<description>Department entities</description>
<entity class-name="acme.Employee" name="Employee" access="FIELD">
<attributes>
<id name="empid" />
<basic name="name" />
<basic name="salary" />
<many-to-one name="department"
target-entity="acme.Department"
fetch="EAGER">
<cascade><cascade-persist/></cascade>
</many-to-one>
</attributes>
</entity>
<entity class-name="acme.Department" name="Department" access="FIELD">
<attributes>
<id name="deptid" />
<basic name="name" />
<basic name="budget" />
<basic name="isManager" />
<one-to-many name="employees"
target-entity="acme.Employee"
fetch="LAZY" mapped-by="parentNode">
<cascade><cascade-persist/></cascade>
</one-to-many>
</attributes>
</entity>
</entity-mappings>

```

### Indizes mit XML für Objekte definieren, die keine Entitäten sind

Indizes für Objekte, die keine Entitäten sind, werden in XML definiert. Das MapIndexPlugin für Maps, die keine Entitäts-Maps sind, wird auf dieselbe Weise wie für Entitäts-Maps erstellt.

#### Java-Bean

```

public class Employee {
    int empid;
    String name;
    double salary;
    Department dept;

    public class Department {
        int deptid;
        String name;
        double budget;
        boolean isManager;
        Collection employees;
    }
}

```

#### ObjectGrid-XML mit Attributindizes

```

<?xml version="1.0" encoding="UTF-8"?>
<objectGridConfig xmlns:xsi="http://www.w3.org/2001/XMLSchema-instance"
xsi:schemaLocation="http://ibm.com/ws/objectgrid/config ../objectGrid.xsd"
xmlns="http://ibm.com/ws/objectgrid/config">
<objectGrids>
<objectGrid name="DepartmentGrid">

```

```

<backingMap name="Employee" pluginCollectionRef="Emp"/>
<backingMap name="Department" pluginCollectionRef="Dept"/>
<querySchema>
<mapSchemas>
<mapSchema mapName="Employee" valueClass="acme.Employee"
primaryKeyField="empid" />
<mapSchema mapName="Department" valueClass="acme.Department"
primaryKeyField="deptid" />
</mapSchemas>
<relationships>
<relationship source="acme.Employee"
target="acme.Department"
relationField="dept" invRelationField="employees" />
</relationships>
</querySchema>
</objectGrid>
</objectGrids>
<backingMapPluginCollections>
<backingMapPluginCollection id="Emp">
<bean id="MapIndexPlugin" className="com.ibm.websphere.objectgrid.plugins.index.HashIndex">
<property name="Name" type="java.lang.String" value="Employee.name"/>
<property name="AttributeName" type="java.lang.String" value="name"/>
<property name="RangeIndex" type="boolean" value="true"
description="Ranges are must be set to true for attributes." />
</bean>
<bean id="MapIndexPlugin" className="com.ibm.websphere.objectgrid.plugins.index.HashIndex">
<property name="Name" type="java.lang.String" value="Employee.salary"/>
<property name="AttributeName" type="java.lang.String" value="salary"/>
<property name="RangeIndex" type="boolean" value="true"
description="Ranges are must be set to true for attributes." />
</bean>
</backingMapPluginCollection>
<backingMapPluginCollection id="Dept">
<bean id="MapIndexPlugin" className="com.ibm.websphere.objectgrid.plugins.index.HashIndex">
<property name="Name" type="java.lang.String" value="Department.name"/>
<property name="AttributeName" type="java.lang.String" value="name"/>
<property name="RangeIndex" type="boolean" value="true"
description="Ranges are must be set to true for attributes." />
</bean>
<bean id="MapIndexPlugin" className="com.ibm.websphere.objectgrid.plugins.index.HashIndex">
<property name="Name" type="java.lang.String" value="Department.budget"/>
<property name="AttributeName" type="java.lang.String" value="budget"/>
<property name="RangeIndex" type="boolean" value="true"
description="Ranges are must be set to true for attributes." />
</bean>
</backingMapPluginCollection>
</backingMapPluginCollections>
</objectGridConfig>

```

## Beziehungen indexieren

WebSphere eXtreme Scale speichert die Fremdschlüssel für zusammengehörige Entitäten im übergeordneten Objekt. Für Entitäten werden die Schlüssel im zugrunde liegenden Tupel gespeichert. Für Objekte, die keine Entitäten sind, werden die Schlüssel explizit im übergeordneten Objekt gespeichert.

Das Hinzufügen eines Index für ein Beziehungsattribut kann Abfragen beschleunigen, die zyklische Referenzen oder die Abfragefilter IS NULL, IS EMPTY, SIZE und MEMBER OF verwenden. Sowohl Assoziationen mit einem Wert als auch Assoziationen mit mehreren Werten können die Annotation "@Index" oder eine HashIndex-Plug-in-Konfiguration in einer ObjectGrid-XML-Deskriptordatei enthalten.

## Entitätsbeziehungsindizes mit @Index definieren

Im folgenden Beispiel werden Entitäten mit @Index-Annotationen definiert:

### Entität mit Annotation

```

@Entity
public class Node {
    @ManyToOne @Index
    Node parentNode;

    @OneToMany @Index
    List<Node> childrenNodes = new ArrayList();
}

```

```

    @OneToMany @Index
    List<BusinessUnitType> businessUnitTypes = new ArrayList();
}

```

## Entitätsbeziehungsindizes mit XML definieren

Im folgenden Beispiel werden dieselben Entitäten und Indizes mit XML mit Hash-Index-Plug-ins definiert:

### Entität ohne Annotationen

```

public class Node {
    int nodeId;
    Node parentNode;
    List<Node> childrenNodes = new ArrayList();
    List<BusinessUnitType> businessUnitTypes = new ArrayList();
}

```

### ObjectGrid XML

```

<?xml version="1.0" encoding="UTF-8"?>
<objectGridConfig xmlns:xsi="http://www.w3.org/2001/XMLSchema-instance"
xsi:schemaLocation="http://ibm.com/ws/objectgrid/config ../objectGrid.xsd"
xmlns="http://ibm.com/ws/objectgrid/config">
  <objectGrids>
    <objectGrid name="ObjectGrid_Entity" entityMetadataXMLFile="entity.xml">
      <backingMap name="Node" pluginCollectionRef="Node"/>
      <backingMap name="BusinessUnitType" pluginCollectionRef="BusinessUnitType"/>
    </objectGrid>
  </objectGrids>
  <backingMapPluginCollections>
    <backingMapPluginCollection id="Node">
      <bean id="MapIndexPlugin" className="com.ibm.websphere.objectgrid.plugins.index.HashIndex">
        <property name="Name" type="java.lang.String" value="parentNode"/>
        <property name="AttributeName" type="java.lang.String" value="parentNode"/>
        <property name="RangeIndex" type="boolean" value="false"
description="Ranges are not supported for association indexes." />
      </bean>
      <bean id="MapIndexPlugin" className="com.ibm.websphere.objectgrid.plugins.index.HashIndex">
        <property name="Name" type="java.lang.String" value="businessUnitType"/>
        <property name="AttributeName" type="java.lang.String" value="businessUnitTypes"/>
        <property name="RangeIndex" type="boolean" value="false"
description="Ranges are not supported for association indexes." />
      </bean>
    </backingMapPluginCollection>
  </backingMapPluginCollections>
</objectGridConfig>

```

### Entitäts-XML

```

<?xml version="1.0" encoding="UTF-8"?>
<entity-mappings xmlns="http://ibm.com/ws/projector/config/emd"
xmlns:xsi="http://www.w3.org/2001/XMLSchema-instance"
xsi:schemaLocation="http://ibm.com/ws/projector/config/emd ../emd.xsd">
  <description>My entities</description>
  <entity class-name="acme.Node" name="Account" access="FIELD">
    <attributes>
      <id name="nodeId" />
      <one-to-many name="childrenNodes"
target-entity="acme.Node"
fetch="EAGER" mapped-by="parentNode">
        <cascade><cascade-all/></cascade>
      </one-to-many>
      <many-to-one name="parentNode"
target-entity="acme.Node"
fetch="LAZY" mapped-by="childrenNodes">
        <cascade><cascade-none/></cascade>
      </many-to-one>
      <many-to-one name="businessUnitTypes"
target-entity="acme.BusinessUnitType"
fetch="EAGER">
        <cascade><cascade-persist/></cascade>
      </many-to-one>
    </attributes>
  </entity>
  <entity class-name="acme.BusinessUnitType" name="BusinessUnitType" access="FIELD">
    <attributes>

```

```

<id name="build" />
<basic name="TypeDescription" />
</attributes>
</entity>
</entity-mappings>

```

Mit den zuvor definierten Indizes werden die folgenden Entitätsabfragebeispiele optimiert:

```

SELECT n FROM Node n WHERE n.parentNode is null
SELECT n FROM Node n WHERE n.businessUnitTypes is EMPTY
SELECT n FROM Node n WHERE size(n.businessUnitTypes)>=10
SELECT n FROM BusinessUnitType b, Node n WHERE b member of n.businessUnitTypes and b.name='TELECOM'

```

## Beziehungsindizes für Objekte definieren, die keine Entitäten sind

Im folgenden Beispiel wird ein HashIndex-Plug-in für Maps, die keine Entitäts-Maps sind, in einer ObjectGrid-XML-Deskriptordatei definiert:

```

<?xml version="1.0" encoding="UTF-8"?>
<objectGridConfig xmlns:xsi="http://www.w3.org/2001/XMLSchema-instance"
  xsi:schemaLocation="http://ibm.com/ws/objectgrid/config ../objectGrid.xsd"
  xmlns="http://ibm.com/ws/objectgrid/config">
  <objectGrids>
    <objectGrid name="ObjectGrid_P0J0">
      <backingMap name="Node" pluginCollectionRef="Node"/>
      <backingMap name="BusinessUnitType" pluginCollectionRef="BusinessUnitType"/>
      <querySchema>
        <mapSchemas>
          <mapSchema mapName="Node"
            valueClass="com.ibm.websphere.objectgrid.samples.entity.Node"
            primaryKeyField="id" />
          <mapSchema mapName="BusinessUnitType"
            valueClass="com.ibm.websphere.objectgrid.samples.entity.BusinessUnitType"
            primaryKeyField="id" />
        </mapSchemas>
        <relationships>
          <relationship source="com.ibm.websphere.objectgrid.samples.entity.Node"
            target="com.ibm.websphere.objectgrid.samples.entity.Node"
            relationField="parentNodeId" invRelationField="childrenNodeIds" />
          <relationship source="com.ibm.websphere.objectgrid.samples.entity.Node"
            target="com.ibm.websphere.objectgrid.samples.entity.BusinessUnitType"
            relationField="businessUnitTypeKeys" invRelationField="" />
        </relationships>
      </querySchema>
    </objectGrid>
  </objectGrids>
  <backingMapPluginCollections>
    <backingMapPluginCollection id="Node">
      <bean id="MapIndexPlugin" className="com.ibm.websphere.objectgrid.plugins.index.HashIndex">
        <property name="Name" type="java.lang.String" value="parentNode"/>
      </bean>
    <backingMapPluginCollection id="BusinessUnitType">
      <bean id="MapIndexPlugin" className="com.ibm.websphere.objectgrid.plugins.index.HashIndex">
        <property name="Name" type="java.lang.String" value="businessUnitType"/>
      </bean>
    </backingMapPluginCollections>
  </objectGridConfig>

```

Mit den zuvor gezeigten Indexkonfigurationen werden die folgenden Objektabfragebeispiele optimiert:

```

SELECT n FROM Node n WHERE n.parentNodeId is null
SELECT n FROM Node n WHERE n.businessUnitTypeKeys is EMPTY
SELECT n FROM Node n WHERE size(n.businessUnitTypeKeys)>=10
SELECT n FROM BusinessUnitType b, Node n WHERE
  b member of n.businessUnitTypeKeys and b.name='TELECOM'

```

---

## Indexierung

Verwenden Sie das Indexierungsfeature, das vom MapIndexPlugin-Plug-in dargestellt wird, um einen Index oder mehrere Indizes für eine BackingMap zu erstellen, damit Datenzugriffe ohne Schlüssel unterstützt werden.

### Indextypen und Konfiguration

Das Indexierungsfeature wird durch "MapIndexPlugin" oder kurz "Index" dargestellt. Index ist ein BackingMap-Plug-in. Für eine BackingMap können mehrere Index-Plug-ins konfiguriert werden, solange jedes Plug-in den Index-Konfigurationsregeln entspricht.

Sie können das Indexierungsfeature verwenden, um einen Index oder mehrere Indizes für eine BackingMap zu erstellen. Ein Index wird aus einem Attribut oder einer Liste von Attributen eines Objekts in der BackingMap erstellt. Das Feature bietet Anwendungen eine Möglichkeit, bestimmte Objekte schneller zu finden. Mit dem Indexierungsfeature können Anwendungen mit einem bestimmten Wert oder innerhalb eines bestimmten Wertebereichs indexierter Attribute finden.

Es gibt zwei Typen von Indexierung: statische Indexierung und dynamische Indexierung. Bei der statischen Indexierung müssen Sie das Index-Plug-in in der BackingMap konfigurieren, bevor Sie die ObjectGrid-Instanz initialisieren. Sie können diese Konfiguration durch XML- oder programmgesteuerte Konfiguration der BackingMap vornehmen. Die statische Indexierung beginnt mit der Erstellung eines Index während der ObjectGrid-Initialisierung. Der Index ist immer mit der BackingMap synchronisiert und zur Verwendung bereit. Nach dem Start des statischen Indexierungsprozesses erfolgt die Verwaltung des Index im Rahmen des Transaktionsverwaltungsprozesses von eXtreme Scale. Wenn Transaktionen Änderungen festschreiben, werden diese Änderungen auch im statischen Index durchgeführt, und Indexänderungen werden rückgängig gemacht, wenn die Transaktion rückgängig gemacht wird.

Bei der dynamischen Indexierung können Sie einen Index in einer BackingMap vor oder nach der Initialisierung der übergeordneten ObjectGrid-Instanz erstellen. Anwendungen haben eine Lebenszykluskontrolle über den dynamischen Indexierungsprozess, d. h., Sie können einen dynamischen Index entfernen, wenn er nicht mehr benötigt wird. Wenn eine Anwendung einen dynamischen Index erstellt, ist der Index möglicherweise nicht zur sofortigen Verwendung bereit, weil die Erstellung des Index eine gewisse Zeit dauert. Da die Erstellungsdauer vom Volumen der zu indexierenden Daten abhängig ist, wird die Schnittstelle "DynamicIndexCallback" für Anwendungen bereitgestellt, die Benachrichtigungen empfangen möchten, wenn bestimmte Indexierungsereignisse eintreten. Zu diesen Ereignissen gehören die Bereitschaft des Index (ready), Fehler (error) und das Löschen des Index (destroy). Anwendungen können diese Callback-Schnittstelle implementieren und sich beim dynamischen Indexierungsprozess registrieren.

Wenn eine BackingMap ein konfiguriertes Index-Plug-in hat, können Sie das Proxy-Objekt für den Anwendungsindex von der entsprechenden ObjectMap abrufen. Wenn Sie die Methode "getIndex" in der Schnittstelle "ObjectMap" aufrufen und den Namen des Index-Plug-ins übergeben, wird das Index-Proxy-Objekt zurückgegeben. Sie müssen das Index-Proxy-Objekt in die entsprechende Anwendungsschnittstelle, z. B. MapIndex, MapRangeIndex oder eine angepasste Indexschnittstelle, umsetzen. Nach dem Abrufen des Index-Proxy-Objekts können Sie in der Anwendungsindexschnittstelle definierte Methoden verwenden, um zwischengespeicherte Objekte zu suchen.

Die Schritte zur Verwendung der Indexierung sind in der folgenden Liste zusammengefasst:

- Fügen Sie statische oder dynamische Index-Plug-ins in der BackingMap hinzu.
- Rufen Sie mit der Methode "getIndex" von ObjectMap ein Proxy-Objekt für den Anwendungsindex ab.
- Setzen Sie das Proxy-Objekt für den Index in eine entsprechende Anwendungsindexschnittstelle um, wie z. B. MapIndex, MapRangeIndex oder eine angepasste Indexschnittstelle.
- Verwenden Sie die in der Anwendungsindexschnittstelle definierten Methoden, um zwischengespeicherte Objekte zu suchen.

Weitere Informationen zum Schreiben eigener Index-Plug-ins finden Sie in den Informationen zum Schreiben eines Index-Plug-ins im *Programmierhandbuch*.

Informationen zur Verwendung der Indexierung finden Sie in den Informationen zur Verwendung der Indexierung für den Datenzugriff ohne Schlüssel im *Programmierhandbuch* und im Abschnitt „Zusammengesetzter Hash-Index“ auf Seite 127.

## Hinweis zur Datenqualität

Die Ergebnisse der Indexabfragemethoden stellen nur eine Momentaufnahme der Daten zu einem bestimmten Zeitpunkt dar. Es werden keine Sperren für Dateneinträge angefordert, nachdem die Ergebnisse an die Anwendung zurückgegeben wurden. Die Anwendung muss sich darüber im Klaren sein, dass Datenaktualisierungen für eine zurückgegebene Datengruppe vorgenommen werden können. Beispiel: Die Anwendung ruft den Schlüssel eines zwischengespeicherten Objekts mit der Methode "findAll" von MapIndex ab. Dieses zurückgegebene Schlüsselobjekt ist einem Dateneintrag im Cache zugeordnet. Die Anwendung muss in der Lage sein, die Methode "get" in ObjectMap auszuführen, um ein Objekt durch Übergabe des Schlüsselobjekts zu suchen. Wenn eine andere Transaktion das Datenobjekt aus dem Cache entfernt, kurz bevor die Methode "get" aufgerufen wird, ist das zurückgegebene Ergebnis null.

## Hinweise zur Leistung der Indexierung

Eine der Hauptzielsetzungen des Indexierungsfeatures ist die Verbesserung der Gesamtleistung der BackingMap. Wenn die Indexierung nicht ordnungsgemäß verwendet wird, kann dies die Leistung der Anwendung beeinträchtigen. Berücksichtigen Sie vor der Verwendung dieses Features die folgenden Faktoren.

- **Anzahl gleichzeitiger Transaktionen mit Schreibzugriff:** Die Indexverarbeitung kann jedesmal stattfinden, wenn eine Transaktion Daten in eine BackingMap schreibt. Schreiben viele Transaktionen gleichzeitig Daten in die Map, kann es zu Leistungseinbußen kommen, wenn eine Anwendung versucht, Indexabfrageoperationen durchzuführen.
- **Größe der von einer Abfrageoperation zurückgegebenen Ergebnismenge:** Je größer die Ergebnismenge wird, desto mehr nimmt die Abfrageleistung ab. Ab einer Ergebnismengengröße von 15 % der Gesamtgröße der BackingMap beginnen sich Leistungseinbußen abzuzeichnen.
- **Anzahl der für dieselbe BackingMap erstellten Indizes:** Jeder Index belegt Systemressourcen. Mit steigender Indexanzahl für die BackingMap nimmt die Leistung ab.

Die Indexierungsfunktion kann die Leistung einer BackingMap erheblich verbessern. Die besten Ergebnisse lassen sich erzielen, wenn hauptsächlich Lese-

operationen für die BackingMap durchgeführt werden, wenn die Abfrageergebnismenge nur einen kleinen Prozentsatz der BackingMap-Einträge enthält und wenn nur einige wenige Indizes für die BackingMap erstellt werden.

## Indexierung für den Datenzugriff ohne Schlüssel verwenden

Die Verwendung der Indexierung als Alternative zum Zugriff auf Daten über Schlüssel ist wesentlich effizienter.

### Erforderliche Schritte

1. Fügen Sie der BackingMap statische oder dynamische Index-Plug-ins hinzu.
2. Rufen Sie mit der Methode "getIndex" von ObjectMap ein Proxy-Objekt für den Anwendungsindex ab.
3. Setzen Sie das Proxy-Objekt für den Index in eine entsprechende Anwendungsschnittstelle um, wie z. B. MapIndex, MapRangeIndex oder eine angepasste Indexschnittstelle.
4. Verwenden Sie die in der Anwendungsschnittstelle definierten Methoden, um zwischengespeicherte Objekte zu suchen.

Die Klasse HashIndex ist die Index-Plug-in-Implementierung, die beide integrierten Anwendungsschnittstellen, MapIndex und MapRangeIndex, unterstützen kann. Sie können auch eigene Indizes erstellen.

**Anmerkung:** Wenn das Indexobjekt in einer verteilten ObjectGrid-Umgebung von einem Client-ObjectGrid abgerufen wird, hat es den Typ "Clientindexobjekt", und alle Indexoperationen werden in einem fernen Server-ObjectGrid ausgeführt. Wenn die Map partitioniert ist, wird die Indexoperation über Fernzugriff in jeder Partition ausgeführt, und die Ergebnisse aller Partitionen werden zusammengeführt, bevor sie an die Anwendung zurückgegeben werden. Die Leistung richtet sich nach der Anzahl der Partitionen und der Größe des von jeder einzelnen Partition zurückgegebenen Ergebnisses. Die Leistung kann schwach sein, wenn beide Faktoren hoch sind.

Wenn Sie ein eigenes Index-Plug-in schreiben möchten, sehen Sie sich den Abschnitt „Index-Plug-in schreiben“ auf Seite 143 an.

Informationen zur Indexierung finden Sie in den Abschnitten „Indexierung“ auf Seite 121 und „Zusammengesetzter Hash-Index“ auf Seite 127.

## Statische Index-Plug-ins hinzufügen

Sie können beide Ansätze verwenden, um der BackingMap-Konfiguration statische Index-Plug-ins hinzuzufügen: XML-Konfiguration und Konfiguration programmgesteuerte Konfiguration. Das folgende Beispiel veranschaulicht den Ansatz mit der XML-Konfiguration.

### Statische Index-Plug-ins hinzufügen: XML-Konfiguration

```
<backingMapPluginCollection id="person">
  <bean id="MapIndexplugin"
    className="com.ibm.websphere.objectgrid.plugins.index.HashIndex">
    <property name="Name" type="java.lang.String" value="CODE"
      description="index name" />
    <property name="RangeIndex" type="boolean" value="true"
      description="true for MapRangeIndex" />
    <property name="AttributeName" type="java.lang.String" value="employeeCode"
      description="attribute name" />
  </bean>
</backingMapPluginCollection>
```

In diesem XML-Konfigurationsbeispiel wird die integrierte Klasse "HashIndex" als Index-Plug-in verwendet. Die Klasse "HashIndex" unterstützt Eigenschaften, die die Benutzer konfigurieren können, wie z. B. Name, RangeIndex und Attribute-Name aus dem vorherigen Beispiel.

- Die Eigenschaft "Name" ist als "CODE" konfiguriert, einer Zeichenfolge, die dieses Index-Plug-in identifiziert. Der Wert der Eigenschaft "Name" muss innerhalb des Geltungsbereichs der BackingMap eindeutig sein und kann verwendet werden, um das Indexobjekt nach Namen von der ObjectMap-Instanz für die BackingMap abzurufen.
- Die Eigenschaft "RangeIndex" ist mit "true" konfiguriert, d. h., die Anwendung kann das abgerufene Indexobjekt in die Schnittstelle "MapRangeIndex" umsetzen. Wird die Eigenschaft "RangeIndex" mit dem Wert "false" konfiguriert, kann die Anwendung das abgerufene Indexobjekt nur in die Schnittstelle "MapIndex" umsetzen. MapRangeIndex unterstützt Funktionen, mit denen Sie Daten über Bereichsfunktionen, wie z. B. größer als und/oder kleiner als, suchen können, wohingegen die Schnittstelle "MapIndex" nur Vergleichsfunktionen unterstützt. Wenn der Index von einer Abfrage verwendet wird, muss die Eigenschaft "RangeIndex" für Einzelattributindizes mit "true" konfiguriert werden. Für einen Beziehungsindex oder einen zusammengesetzten Index muss die Eigenschaft "RangeIndex" mit "false" konfiguriert werden.
- Die Eigenschaft "AttributeName" ist mit "employeeCode" konfiguriert, d. h., das Attribut "employeeCode" des zwischengespeicherten Objekts wird verwendet, um einen Einzelattributindex zu erstellen. Wenn eine Anwendung zwischengespeicherte Objekte mit mehreren Attributen suchen muss, kann die Eigenschaft "AttributeName" auf eine durch Kommas begrenzte Liste mit Attributen gesetzt werden. Dies ergibt dann einen zusammengesetzten Index.

Weitere Informationen finden Sie in den Informationen zur Konfiguration von HashIndex im *Administratorhandbuch*.

Die Schnittstelle "BackingMap" hat zwei Methoden, die Sie verwenden können, um statische Index-Plug-ins hinzuzufügen: addMapIndexplugin und setMapIndexplugins. Weitere Informationen finden Sie in der API-Dokumentation.

Das folgende Codebeispiel veranschaulicht den programmgesteuerten Konfigurationsansatz:

### Statische Index-Plug-ins hinzufügen: Programmgesteuerte Konfiguration

```
import com.ibm.websphere.objectgrid.ObjectGridManagerFactory;
import com.ibm.websphere.objectgrid.ObjectGridManager;
import com.ibm.websphere.objectgrid.ObjectGrid;
import com.ibm.websphere.objectgrid.BackingMap;

ObjectGridManager ogManager = ObjectGridManagerFactory.getObjectGridManager();
ObjectGrid ivObjectGrid = ogManager.createObjectGrid( "grid" );
BackingMap personBackingMap = ivObjectGrid.getMap("person");

// Integrierte Klasse "HashIndex" als Index-Plug-in-Klasse verwenden
HashIndex mapIndexplugin = new HashIndex();
mapIndexplugin.setName("CODE");
mapIndexplugin.setAttributeName("EmployeeCode");
mapIndexplugin.setRangeIndex(true);
personBackingMap.addMapIndexplugin(mapIndexplugin);
```

### Statische Indizes verwenden

Nachdem Sie einer BackingMap-Konfiguration ein statisches Index-Plug-in hinzugefügt und die übergeordnete ObjectGrid-Instanz initialisiert haben, können Anwendungen das Indexobjekt nach Namen von der ObjectMap-Instanz für die BackingMap abrufen. Setzen Sie das Indexobjekt in die Anwendungsindex-



schnittstelle um. Operationen, die von der Anwendungsindexschnittstelle unterstützt werden, können jetzt ausgeführt werden.

Das folgende Codebeispiel veranschaulicht, wie statische Indizes abgerufen und verwendet werden.

### Beispiel für die Verwendung statischer Indizes

```
Session session = ivObjectGrid.getSession();
ObjectMap map = session.getMap("person ");
MapRangeIndex codeIndex = (MapRangeIndex) m.getIndex("CODE");
Iterator iter = codeIndex.findLessEqual(new Integer(15));
while (iter.hasNext()) {
    Object key = iter.next();
    Object value = map.get(key);
}
```

### Dynamische Indizes hinzufügen, entfernen und verwenden

Sie können dynamische Indizes jederzeit über das Programm in einer BackingMap erstellen und entfernen. Ein dynamischer Index unterscheidet sich insofern von einem statischen Index, dass der dynamische Index auch nach der Initialisierung der übergeordneten ObjectGrid-Instanz erstellt werden kann. Anders als die statische Indexierung ist die dynamische Indexierung ein asynchroner Prozess, der im Status "Bereit" sein muss, damit Sie ihn verwenden können. Diese Methode verwendet denselben Ansatz für das Abrufen und Verwenden der dynamischen Indizes wie für statische Indizes. Sie können einen dynamischen Index entfernen, wenn er nicht mehr benötigt wird. Die Schnittstelle "BackingMap" hat Methoden zum Erstellen und Entfernen dynamischer Indizes.

Weitere Informationen zu den Methoden "createDynamicIndex" und "removeDynamicIndex" finden Sie in der Dokumentation zur API "BackingMap".

### Beispiel für die Verwendung dynamischer Indizes

```
import com.ibm.websphere.objectgrid.ObjectGridManagerFactory;
import com.ibm.websphere.objectgrid.ObjectGridManager;
import com.ibm.websphere.objectgrid.ObjectGrid;
import com.ibm.websphere.objectgrid.BackingMap;

ObjectGridManager ogManager = ObjectGridManagerFactory.getObjectGridManager();
ObjectGrid og = ogManager.createObjectGrid("grid");
BackingMap bm = og.getMap("person");
og.initialize();

// Index nach der Initialisierung des ObjectGrids ohne DynamicIndexCallback erstellen
bm.createDynamicIndex("CODE", true, "employeeCode", null);

try {
    // Wenn DynamicIndexCallback nicht verwendet wird, warten, bis der Index bereit ist.
    // Die Wartezeit richtet sich nach der aktuellen Größe der Map.
    Thread.sleep(3000);
} catch (Throwable t) {
    // ...
}

// Wenn der Index bereit ist, können Anwendungen versuchen, eine Instanz der
// Anwendungsindexschnittstelle abzurufen.
// Anwendungen müssen einen Weg finden, um sicherzustellen, dass der Index
// zur Verwendung bereit ist, wenn die Schnittstelle "DynamicIndexCallback"
// nicht verwendet wird.
// Das folgende Beispiel demonstriert eine Methode, mit der auf
// die Bereitschaft des Index gewartet wird.
// Berücksichtigen Sie die Größe der Map bei der Berechnung der Gesamtwartzeit.

Session session = og.getSession();
ObjectMap m = session.getMap("person");
MapRangeIndex codeIndex = null;

int counter = 0;
int maxCounter = 10;
boolean ready = false;
while (!ready && counter < maxCounter) {
    try {
        counter++;
        codeIndex = (MapRangeIndex) m.getIndex("CODE");
        ready = true;
    } catch (IndexNotReadyException e) {
        // Impliziert, dass der Index nicht bereit ist...
        System.out.println("Index is not ready. continue to wait.");
    }
}
```

```

        try {
            Thread.sleep(3000);
        } catch (Throwable tt) {
            // ...
        }
    } catch (Throwable t) {
        // unexpected exception
        t.printStackTrace();
    }
}

if (!ready) {
    System.out.println("Index is not ready. Need to handle this situation.");
}

// Verwenden Sie den Index für Abfragen.
// Die unterstützen Operationen finden Sie in den Beschreibungen der
// Schnittstellen "MapIndex" und "MapRangeIndex".
// Das Objektattribut, nach dem der Index erstellt wird, ist EmployeeCode.
// Nehmen Sie an, dass das Attribut "EmployeeCode" den Datentyp "Integer" hat.
// Der Parameter, der an Indexoperationen übergeben wird, hat diesen Datentyp.

Iterator iter = codeIndex.findLessEqual(new Integer(15));

// Entfernen Sie den dynamischen Index, wenn er nicht mehr benötigt wird.
bm.removeDynamicIndex("CODE");

```

## Schnittstelle "DynamicIndexCallback"

Die Schnittstelle "DynamicIndexCallback" ist für Anwendungen bestimmt, die Benachrichtigungen über die Indexierungsereignisse "ready" (Bereit), "error" (Fehler) oder "destroy" (Löschen) empfangen möchten. DynamicIndexCallback ist ein optionaler Parameter für die Methode "createDynamicIndex" der Schnittstelle "BackingMap". Mit einer registrierten DynamicIndexCallback-Instanz können Anwendungen beim Empfang von Benachrichtigungen über ein Indexierungsereignis Geschäftslogik ausführen. Das Ereignis "ready" bedeutet beispielsweise, dass der Index zur Verwendung bereit ist. Wenn eine Benachrichtigung über dieses Ereignis empfangen wird, kann eine Anwendung versuchen, die Instanz der Anwendungsindexschnittstelle abzurufen und zu verwenden. Weitere Informationen finden Sie in der Beschreibung der API "DynamicIndexCallback" in der API-Dokumentation.

Das folgende Codebeispiel veranschaulicht die Verwendung der Schnittstelle "DynamicIndexCallback":

### Schnittstelle "DynamicIndexCallback" verwenden

```

BackingMap personBackingMap = ivObjectGrid.getMap("person");
DynamicIndexCallback callback = new DynamicIndexCallbackImpl();
personBackingMap.createDynamicIndex("CODE", true, "employeeCode", callback);

class DynamicIndexCallbackImpl implements DynamicIndexCallback {
    public DynamicIndexCallbackImpl() {
    }

    public void ready(String indexName) {
        System.out.println("DynamicIndexCallbackImpl.ready() -> indexName = " + indexName);

        // Simulieren, was eine Anwendung tut, wenn sie über die Bereitschaft des Index benachrichtigt wird.
        // Normalerweise wartet die Anwendung, bis der Bereitschaftsstatus erreicht ist, und fährt
        // dann mit der Logik zur Verwendung des Index fort.
        if("CODE".equals(indexName)) {
            ObjectGridManager ogManager = ObjectGridManagerFactory.getObjectGridManager();
            ObjectGrid og = ogManager.createObjectGrid("grid");
            Session session = og.getSession();
            ObjectMap map = session.getMap("person");
            MapIndex codeIndex = (MapIndex) map.getIndex("CODE");
            Iterator iter = codeIndex.findAll(codeValue);
        }
    }

    public void error(String indexName, Throwable t) {
        System.out.println("DynamicIndexCallbackImpl.error() -> indexName = " + indexName);
        t.printStackTrace();
    }

    public void destroy(String indexName) {
        System.out.println("DynamicIndexCallbackImpl.destroy() -> indexName = " + indexName);
    }
}

```

## Zusammengesetzter Hash-Index

Der zusammengesetzte Hash-Index verbessert die Abfrageleistung und unterbindet das kostenintensiv Durchsuchen von Maps. Außerdem bietet das Feature der Anwendungsprogrammierschnittstelle "HashIndex" eine komfortable Möglichkeit, zwischengespeicherte Objekte zu suchen, wenn die Suchkriterien sehr viele Attribute enthalten.

### Verbesserte Leistung

Ein zusammengesetzter Hash-Index ist eine schnelle und komfortable Methode für das Suchen zwischengespeicherter Objekte mit mehreren Attributen in Suchkriterien. Der zusammengesetzte Index unterstützt Suchoperationen mit vollständiger Attributübereinstimmung, aber keine Bereichssuche.

**Anmerkung:** Zusammengesetzte Indizes unterstützen den Operator BETWEEN in der ObjectGrid-Abfragesprache nicht, da BETWEEN die Unterstützung von Bereichssuchen voraussetzt. Die Bedingungen "größer als" (>) und "kleiner als" (<) funktionieren nicht, weil sie Bereichsindizes erfordern.

Ein zusammengesetzter Index kann die Leistung von Abfragen verbessern, wenn der entsprechende zusammengesetzte Index für die WHERE-Bedingung verfügbar ist. Das bedeutet, dass der zusammengesetzte Index exakt dieselben Attribute hat, die auch in der WHERE-Bedingung verwendet werden, und eine vollständige Attributübereinstimmung erzielt wird.

Eine Abfrage kann viele Attribute in einer Bedingung enthalten. Sehen Sie sich das folgende Beispiel an:

```
SELECT a FROM Address a WHERE a.city='Rochester' AND a.state='MN' AND a.zipcode='55901'
```

Ein zusammengesetzter Index kann die Anfrageleistung verbessern, indem das Durchsuchen von Maps oder das Verknüpfen mehrerer Einzelattributindexergebnisse vermieden wird. Wenn in dem Beispiel ein zusammengesetzter Index mit Attributen (city,state,zipcode) definiert wird, kann die Abfragesteuerkomponente den zusammengesetzten Index verwenden, um den Eintrag mit city='Rochester', state='MN' und zipcode='55901' zu suchen. Sie keinen zusammengesetzten Index und Attributindizes für die Attribute "city", "state" und "zipcode" verwenden, muss die Abfragesteuerkomponente die Map durchsuchen oder mehrere Einzelattributsuchen verknüpfen, was gewöhnlich Kosten und Aufwand verursacht. Außerdem wird bei Abfragen des zusammengesetzten Index nur das Muster mit vollständiger Übereinstimmung unterstützt.

### Zusammengesetzten Index konfigurieren

Sie können einen zusammengesetzten Index auf drei Arten konfigurieren: mit XML, über das Programm oder (nur für Entitäts-Maps) mit Entitätsannotationen.

#### Mit XML

Zum Konfigurieren eines zusammengesetzten Index mit XML müssen Sie Code wie den folgenden in das Element "backingMapPluginCollections" der Konfigurationsdatei einfügen.

#### Zusammengesetzter Index - Konfigurationsansatz mit XML

```
<bean id="MapIndexPlugin" className="com.ibm.websphere.objectgrid.plugins.index.HashIndex">
<property name="Name" type="java.lang.String" value="Address.CityStateZip"/>
<property name="AttributeName" type="java.lang.String" value="city,state,zipcode"/>
</bean>
```

## Programmgesteuerte Konfiguration

Der folgende Beispielcode für die programmgesteuerte Konfiguration erstellt denselben zusammengesetzten Index wie der Beispielcode für die XML-Konfiguration.

```
HashIndex mapIndex = new HashIndex();
mapIndex.setName("Address.CityStateZip");
mapIndex.setAttributeName(("city,state,zipcode"));
mapIndex.setRangeIndex(true);

BackingMap bm = objectGrid.defineMap("mymap");
bm.addMapIndexPlugin(mapIndex);
```

Die Konfiguration eines zusammengesetzten Index entspricht abgesehen von der Definition des Werts für die Eigenschaft "attributeName" der Konfiguration eines regulären Index mit XML. Bei einem zusammengesetzten Index ist der Wert von "attributeName" eine durch Kommas begrenzte Liste mit Attributen. Die Werteklasse "Address" hat beispielsweise 3 Attribute: city, state und zipcode. Ein zusammengesetzter Index kann mit dem Wert "city,state,zipcode" für die Eigenschaft "attributeName" definiert werden. Dieser Wert zeigt an, dass city, state und zipcode in den zusammengesetzten Index eingeschlossen werden sollen.

Beachten Sie auch, dass zusammengesetzte Hash-Indizes keine Bereichssuchen unterstützen und die Eigenschaft "RangeIndex" deshalb nicht auf "true" gesetzt werden kann.

## Mit Entitätsannotationen

Für Entitäts-Maps kann der Annotationsansatz zum Definieren eines zusammengesetzten Index verwendet werden. Sie können eine Liste zusammengesetzter Indizes in der Annotation "CompositeIndexes" auf der Ebene der Entitätsklasse definieren. Ein zusammengesetzter Index hat eine Eigenschaft "name" und eine Eigenschaft "attributeNames". Jeder zusammengesetzte Index wird einer HashIndex-Instanz zugeordnet, die auf die BackingMap der jeweiligen Entität angewendet wird. Die HashIndex-Instanz wird als Index ohne Bereichsunterstützung konfiguriert.

```
@Entity
@CompositeIndexes({
    @CompositeIndex(name="CityStateZip", attributeNames="city,state,zipcode"),
    @CompositeIndex(name="lastnameBirthday", attributeNames="lastname,birthday")
})
public class Address {
    @Id int id;
    String street;
    String city;
    String state;
    String zipcode;
    String lastname;
    Date birthday;
}
```

Die Eigenschaft "name" jedes zusammengesetzten Index muss in der Entität und in der BackingMap eindeutig sein. Wenn Sie den Namen nicht angeben, wird ein generierter Name verwendet. Die Eigenschaft "attributeNames" wird verwendet, um die Eigenschaft "attributeName" der HashIndex-Instanz mit der durch Kommas begrenzten Liste von Attributen zu füllen. Die Attributnamen stimmen mit den persistenten Feldnamen überein, wenn die Entitäten für die Verwendung von Entitäten mit Feldzugriff konfiguriert sind, bzw. mit dem Eigenschaftsnamen gemäß Definition in den JavaBeans-Namenskonventionen für Entitäten mit Eigenschaftszugriff. Beispiel: Wenn der Attributname "street" lautet, wird die Getter-Methode für die Eigenschaft "getStreet" benannt.

## Suchoperationen in zusammengesetzten Indizes durchführen

Nach der Konfiguration eines zusammengesetzten Index kann eine Anwendung die Methode "findAll(Object)" in der Schnittstelle "MapIndex" wie folgt verwenden, um Suchoperationen durchzuführen:

```
Object[] compositeValue = new Object[]{
    MapIndex.EMPTY_VALUE, "MN", "55901"};
Iterator iter = mapIndex.findAll(compositeValue);
```

MapIndex.EMPTY\_VALUE wird compositeValue[ 0 ] zugeordnet, d. h., dass das Attribut "city" von der Auswertung ausgeschlossen wird. Es werden nur Objekte in das Ergebnis eingeschlossen, deren Attribut "state" den Wert "MN" und deren Attribut "zipcode" den Wert "55901" hat.

Die folgenden Abfragen profitieren von der zuvor beschriebenen Konfiguration des zusammengesetzten Index:

```
SELECT a FROM Address a WHERE a.city='Rochester' AND a.state='MN' AND
a.zipcode='55901'
```

```
SELECT a FROM Address a WHERE a.state='MN' AND a.zipcode='55901'
```

Die Abfragesteuerkomponente sucht den entsprechenden zusammengesetzten Index und verwendet diesen zur Leistungsverbesserung in den Fällen mit vollständiger Attributübereinstimmung.

In einigen Szenarios muss die Anwendung mehrere zusammengesetzte Indizes mit überlappenden Attributen definieren, um alle Abfragen mit vollständiger Attributübereinstimmung abzudecken. Ein Nachteil einer höheren Anzahl zusammengesetzter Indizes sind die möglichen Leistungseinbußen in Map-Operationen.

## Migration und Interoperabilität

Bezüglich der Verwendung zusammengesetzter Indizes ist eine einzige Einschränkung zu beachten: Eine Anwendung kann einen zusammengesetzten Index nicht in einer verteilten Umgebung mit heterogenen Containern konfigurieren. Alte und neue Container können nicht gemischt werden, da ältere Container die Konfiguration eines zusammengesetzten Index nicht erkennen. Der zusammengesetzte Index gleicht abgesehen davon, dass er die Indexierung über mehrere Attribute unterstützt, dem vorhandenen regulären Attributindex. Wenn Sie ausschließlich den regulären Attributindex verwenden, ist eine heterogene Containerumgebung trotzdem realisierbar.

---

## DataGrid-API

Die DataGrid-API ist eine einfache Programmierschnittstelle für die Ausführung von Geschäftslogik über das gesamte ObjectGrid oder über einen Teil des ObjectGrids parallel zu der Position, an der sich die Daten befinden.

## DataGrid-APIs und Partitionierung

Mit den DataGrid-APIs kann ein Client Anforderungen an eine einzige Partition, an einen Teil aller Partitionen oder an alle Partitionen in einem Grid senden. Der Client kann eine Liste mit Schlüsseln angeben, und WebSphere eXtreme Scale bestimmt die Gruppe von Partitionen, die die Schlüssel enthalten. Anschließend wird die Anforderung gleichzeitig an alle Partitionen in der Gruppe gesendet, und

der Client wartet auf die Ergebnisse. Der Client kann Anforderungen auch ohne die Angabe von Schlüssel senden. In diesem Fall werden die Anforderungen an alle Partitionen gesendet.

Agenten, die im Grid implementiert sind, funktionieren nicht im Clientmodus. Diese Agenten arbeiten direkt mit dem primären Shard. Die direkte Arbeit mit dem primären Shard führt zu einer maximalen Leistung. Sie ermöglicht die Verarbeitung Zehntausender und mehr Transaktionen pro Sekunde, weil die Agenten in der kürzest möglichen Zugriffszeit auf den Speicher mit den Daten arbeiten können. Die direkte Arbeit mit dem primären Shard bedeutet auch, dass der Agent nur Daten sehen kann, die sich in diesem Shard befinden. Sie bietet einige interessante Möglichkeiten, die in einem Client nicht realisierbar sind.

Ein typischer eXtreme-Scale-Client muss in der Lage sein, die Partition über die Transaktion zu bestimmen, weil der Client die Anforderung weiterleiten muss. Wenn ein Agent direkt mit einem Shard verbunden ist, ist kein Routing erforderlich. Alle Anforderungen werden direkt über dieses Shard abgewickelt. Da der Agent direkt mit einem Shard verbunden ist, ist der Zugriff auf Daten in anderen Maps im Shard möglich, ohne sich um die allgemeinen Partitionierungsschlüssel kümmern zu müssen, weil kein Routing stattfindet.

## DataGrid-Agenten und entitätsbasierte Maps

Eine Map enthält Schlüsselobjekte und Wertobjekte. Das Schlüsselobjekt ist wie der Wert ein generiertes Tupel. Gewöhnlich wird ein Agent mit den anwendungsdefinierten Schlüsselobjekten bereitgestellt.

Das Schlüsselobjekt ist wie der Wert ein generiertes Tupel. Gewöhnlich wird ein Agent mit den anwendungsdefinierten Schlüsselobjekten bereitgestellt. Dies sind die Schlüsselobjekte, die von der Anwendung verwendet werden, bzw. Tupel, wenn es sich um eine Entitäts-Map handelt. Eine Anwendung, die Entitäten verwendet, möchte nicht direkt mit Tupeln arbeiten, sondern bevorzugt die Arbeit mit den Java-Objekten, die der Entität zugeordnet sind.

Deshalb kann eine Agentenklasse die Schnittstelle "EntityAgentMixin" implementieren. Dies zwingt die Klasse, eine weitere Methode zu implementieren, die Methode "getClassForEntity()". Diese Methode gibt die Entitätsklasse zurück, die mit dem Agenten auf der Serverseite zu verwenden ist. Die Schlüssel werden in diese Entität konvertiert, bevor die Methoden "process" und "reduce" aufgerufen werden.

Diese Semantik unterscheidet sich von einem Agenten, der kein EntityAgentMixin-Agent ist, bei dem diese Methoden nur für die Schlüssel bereitgestellt werden. Ein Agent, der EntityAgentMixin implementiert, empfängt das Entity-Objekt, das Schlüssel und Werte in einem einzigen Objekt enthält.

**Anmerkung:** Wenn die Entität nicht auf dem Server vorhanden ist, sind die Schlüssel das unbearbeitete Tupelformat des Schlüssels und nicht die verwaltete Entität.

## Beispiel für die DataGrid-APIs

Die DataGrid-APIs unterstützen zwei gebräuchliche Programmiermuster für Grids. Das erste Muster ist eine parallele Map und das zweite Muster eine Reduktion.

### Parallele Map

Die parallele Map lässt die Verarbeitung der Einträge für eine Gruppe von Schlüsseln zu und gibt für jeden verarbeiteten Eintrag ein Ergebnis zurück. Die Anwendung erstellt eine Liste mit Schlüsseln und empfängt nach dem Aufruf einer Map-Operation eine Map mit Schlüssel/Ergebnis-Paaren. Das Ergebnis ist das Ergebnis der Anwendung einer Funktion auf den Eintrag jedes Schlüssels. Die Funktion wird von der Anwendung bereitgestellt.

### Ablauf des MapGridAgent-Aufrufs

Wenn die Methode "AgentManager.callMapAgent" mit einer Sammlung von Schlüsseln aufgerufen wird, wird die MapGridAgent-Instanz serialisiert und an jede primäre Partition gesendet, in die die Schlüssel aufgelöst werden. Das bedeutet, dass alle Instanzdaten, die im Agenten gespeichert sind, an den Server gesendet werden können. Jede primäre Partition besitzt damit eine Instanz des Agenten. Die Methode "process" wird für jede Instanz einmal für jeden Schlüssel aufgerufen, der in die Partition aufgelöst wird. Das Ergebnis jeder Methode "process" wird anschließend zurück in den Client serialisiert und an den Aufrufenden in einer Map-Instanz zurückgegeben, wo das Ergebnis als Wert dargestellt wird.

Wenn die Methode "AgentManager.callMapAgent" ohne Sammlung von Schlüsseln aufgerufen wird, wird die MapGridAgent-Instanz serialisiert und an jede primäre Partition gesendet. Das bedeutet, dass alle Instanzdaten, die im Agenten gespeichert sind, an den Server gesendet werden können. Jede primäre Partition besitzt damit eine Instanz (Partition) des Agenten. Die Methode "processAllEntries" wird für jede Partition aufgerufen. Das Ergebnis jeder Methode "processAllEntries" wird anschließend zurück in den Client serialisiert und an den Aufrufenden in einer Map-Instanz zurückgegeben. Im folgenden Beispiel wird davon ausgegangen, dass es eine Entität "Person" in der folgenden Form gibt:

```
import com.ibm.websphere.projector.annotations.Entity;
import com.ibm.websphere.projector.annotations.Id;
@Entity
public class Person
{
    @Id String ssn;
    String firstName;
    String surname;
    int age;
}
```

Die von der Anwendung bereitgestellte Funktion ist als Klasse geschrieben, die die Schnittstelle "MapAgentGrid" implementiert. Im Folgenden sehen Sie einen Beispielagenten mit einer Funktion, die das Alter einer Person mit zwei multipliziert zurückgibt.

```
public class DoublePersonAgeAgent implements MapGridAgent, EntityAgentMixin
{
    private static final long serialVersionUID = -2006093916067992974L;

    int lowAge;
    int highAge;

    public Object process(Session s, ObjectMap map, Object key)
    {
        Person p = (Person)key;
        return new Integer(p.age * 2);
    }

    public Map processAllEntries(Session s, ObjectMap map)
    {
        EntityManager em = s.getEntityManager();    Query q = em.createQuery("select p from Person p where p.age > ?1 and p.age < ?2");
        q.setParameter(1, lowAge);
        q.setParameter(2, highAge);
        Iterator iter = q.getResultIterator();
        Map<Person, Integer> rc = new HashMap<Person, Integer>();
        while(iter.hasNext())
        {
            Person p = (Person)iter.next();
        }
    }
}
```

```

        rc.put(p, (Integer)process(s, map, p));
    }
    return rc;
}
public Class getClassForEntity()
{
    return Person.class;
}
}

```

Dieses Beispiel zeigt den Map-Agenten für die Verdopplung des Alters einer Person. Sehen Sie sich zuerst die process-Methoden an. Mit der ersten process-Methode wird die Person angegeben, die bearbeitet werden soll. Sie gibt einfach das verdoppelte Alter dieses Eintrags zurück. Die zweite process-Methode wird für jede Partition ausgeführt. Sie sucht alle Person-Objekte, deren Alter zwischen "lowAge" und "highAge" liegt und gibt deren Alter verdoppelt zurück.

```

Session s = grid.getSession();
ObjectMap map = s.getMap("Person");
AgentManager amgr = map.getAgentManager();

DoublePersonAgeAgent agent = new DoublePersonAgeAgent();

// Liste mit Schlüssel erstellen
ArrayList<Person> keyList = new ArrayList<Person>();
Person p = new Person();
p.ssn = "1";
keyList.add(p);
p = new Person ();
p.ssn = "2";
keyList.add(p);

// Ergebnisse für diese Einträge abrufen
Map<Tuple, Object> = amgr.callMapAgent(agent, keyList);

```

Dieses Beispiel zeigt einen Client, der ein Session-Objekt und eine Referenz auf die Map "Person" abrufen. Die Agentenoperation wird für eine bestimmte Map durchgeführt. Die Schnittstelle "AgentManager" wird von dieser Map abgerufen. Es wird eine aufzurufende Instanz des Agenten erstellt, und alle erforderlichen Statusinformationen werden dem Objekt durch das Setzen von Attributen hinzugefügt. In diesem Fall gibt es keine. Anschließend wird eine Liste mit Schlüssel erstellt. Zurückgegeben werden eine Map mit den verdoppelten Werten für Person 1 und dieselben Werte für Person 2.

Anschließend wird der Agent für diese Gruppe von Schlüssel aufgerufen. Die Methode "process" des Agenten wird mit einigen der angegebenen Schlüssel für jede Partition im Grid parallel aufgerufen. Es wird eine Map zurückgegeben, die die zusammengefassten Ergebnisse für den angegebenen Schlüssel enthält. In diesem Fall werden eine Map mit verdoppelten Werten für das Alter der Person 1 und dieselben Werte für Person 2 zurückgegeben.

Wenn der Schlüssel nicht vorhanden ist, wird der Agent trotzdem aufgerufen. Dies gibt dem Agenten die Möglichkeit, den Map-Eintrag zu erstellen. Wenn EntityAgentMixin verwendet wird, ist der zu verarbeitende Schlüssel nicht die Entität, sondern der eigentliche Tupelschlüsselwert für die Entität. Wenn die Schlüssel unbekannt sind, können alle Partitionen abgefragt werden, um Person-Objekte einer bestimmten Form zu suchen und deren Alter verdoppelt zurückzugeben. Es folgt ein Beispiel:

```

Session s = grid.getSession();
ObjectMap map = s.getMap("Person");
AgentManager amgr = map.getAgentManager();

DoublePersonAgeAgent agent = new DoublePersonAgeAgent();

```



```
agent.lowAge = 20;
agent.highAge = 9999;

Map m = amgr.callMapAgent(agent);
```

Das vorherige Beispiel zeigt, wie der AgentManager für die Map "Person" abgerufen und der Agent mit dem Mindest- und Maximalalter für die Personen von Interesse erstellt und initialisiert wird. Anschließend wird der Agent mit der Methode "callMapAgent" aufgerufen. Beachten Sie, dass keine Schlüssel bereitgestellt werden. Dies bewirkt, dass das ObjectGrid den Agenten auf jeder Partition im Grid parallel aufruft und anschließend die zusammengefassten Ergebnisse an den Client zurückgibt. Der Agent sucht alle Person-Objekte im Grid, deren Alter zwischen dem Mindest- und dem Maximalwert liegt, und verdoppelt das Alter dieser Person-Objekte. Dies zeigt, wie die Grid-APIs verwendet werden können, um eine Abfrage auszuführen, die Entitäten sucht, die einer bestimmten Abfrage entsprechen. Der Agent wird einfach serialisiert und vom ObjectGrid auf die Partitionen mit den benötigten Einträgen transportiert. Die Ergebnisse werden für den Rücktransport an den Client auf ähnliche Weise serialisiert. Bei der Verwendung der Map-APIs muss mit Vorsicht vorgegangen werden. Wenn das ObjectGrid Terabytes von Objekten enthält und auf vielen Servern ausgeführt wird, könnte dies möglicherweise nur auf den größten Maschinen zu schaffen sein, auf denen der Client ausgeführt wird. Verwenden Sie dies nur für die Verarbeitung einer kleinen Teilmenge. Wenn eine große Teilmenge verarbeitet werden muss, empfiehlt sich die Verwendung eines Reduktionsagenten, um die Verarbeitung auf das Grid auszulagern, anstatt sie in einem Client auszuführen.

### Parallele Reduktions- oder Aggregationsagenten

Bei diesem Programmierstil wird eine Teilmenge der Einträge verarbeitet und ein einziges Ergebnis für die Eintragsgruppe ermittelt. Beispiele für ein solches Ergebnis sind

- ein Mindestwert,
- ein Maximalwert,
- eine andere geschäftsspezifische Funktion.

Ein Reduktionsagent wird auf ähnlich Weise wie die Map-Agenten codiert und aufgerufen.

### Ablauf des ReduceGridAgent-Aufrufs

Wenn die Methode "AgentManager.callReduceAgent" mit einer Sammlung von Schlüsseln aufgerufen wird, wird die ReduceGridAgent-Instanz serialisiert und an jede primäre Partition gesendet, in die die Schlüssel aufgelöst werden. Das bedeutet, dass alle Instanzdaten, die im Agenten gespeichert sind, an den Server gesendet werden können. Jede primäre Partition besitzt damit eine Instanz des Agenten. Die Methode "reduce(Session s, ObjectMap map, Collection keys)" wird einmal pro Instanz (Partition) mit dem Teil der Schlüssel aufgerufen, die in die Partition aufgelöst werden. Das Ergebnis jeder Methode "reduce" wird anschließend zurück in den Client serialisiert. Die Methode "reduceResults" wird in der ReduceGridAgent-Clientinstanz mit der Sammlung der Ergebnisse jedes fernen reduce-Aufrufs aufgerufen. Das Ergebnis der Methode "reduceResults" wird an den Aufrufenden der Methode "callReduceAgent" zurückgegeben.

Wenn die Methode "AgentManager.callReduceAgent" ohne eine Sammlung von Schlüsseln aufgerufen wird, wird ReduceGridAgentinstance serialisiert und an jede primäre Partition gesendet. Das bedeutet, dass alle Instanzdaten, die im Agenten

gespeichert sind, an den Server gesendet werden können. Jede primäre Partition besitzt damit eine Instanz des Agenten. Die Methode "reduce(Session s, ObjectMap map)" wird einmal pro Instanz (Partition) aufgerufen. Das Ergebnis jeder Methode "reduce" wird anschließend zurück in den Client serialisiert. Die Methode "reduce-Results" wird in der ReduceGridAgent-Clientinstanz mit der Sammlung der Ergebnisse jedes fernen reduce-Aufrufs aufgerufen. Das Ergebnis der Methode "reduce-Results" wird an den Aufrufenden der Methode "callReduceAgent" zurückgegeben. Im Folgenden sehen Sie ein Beispiel für einen Reduktionsagenten, der einfach die Alter der übereinstimmenden Einträge hinzufügt:

```
public class SumAgeReduceAgent implements ReduceGridAgent, EntityAgentMixin
{
    private static final long serialVersionUID = 2521080771723284899L;

    int lowAge;
    int highAge;

    public Object reduce(Session s, ObjectMap map, Collection keyList)
    {
        Iterator<Person> iter = keyList.iterator();
        int sum = 0;
        while(iter.hasNext())
        {
            Person p = iter.next();
            sum += p.age;
        }
        return new Integer(sum);
    }

    public Object reduce(Session s, ObjectMap map)
    {
        EntityManager em = s.getEntityManager ();
        Query q = em.createQuery("select p from Person p where p.age > ?1 and p.age < ?2");
        q.setParameter(1, lowAge);
        q.setParameter(2, highAge);
        Iterator<Person> iter = q.getResultIterator();
        int sum = 0;
        while(iter.hasNext())
        {
            sum += iter.next().age;
        }
        return new Integer(sum);
    }

    public Class getClassForEntity()
    {
        return Person.class;
    }
}
```

Das vorherige Beispiel zeigt den Agenten. Der Agent setzt sich aus drei wichtigen Teilen zusammen. Im ersten Teil kann eine bestimmte Gruppe von Einträgen ohne Abfrage verarbeitet werden. Es findet lediglich eine Iteration durch die Gruppe der Einträge statt, bei der die Alter hinzugefügt werden. Die Methode gibt die Summe zurück. Im zweiten Teil wird eine Abfrage verwendet, um die zusammenzufassenden Einträge auszuwählen. Anschließend werden die Altersangaben aller übereinstimmenden Personen summiert. Die dritte Methode wird verwendet, um die Ergebnisse aller Partitionen zu einem einzigen Ergebnis zusammenzufassen. Das ObjectGrid führt die Eintragsaggregation parallel im Grid durch. Jede Partition liefert ein Zwischenergebnis, das mit den Zwischenergebnissen der anderen Partitionen zusammengefasst werden muss. Diese dritte Methode führt diese Task aus. Im folgenden Beispiel wird der Agent aufgerufen, und die Altersangaben aller Personen mit einem Alter zwischen 10 und 20 ausschließlich werden zusammengefasst:

```
Session s = grid.getSession();
ObjectMap map = s.getMap("Person");
AgentManager amgr = map.getAgentManager();

SumAgeReduceAgent agent = new SumAgeReduceAgent();

Person p = new Person();
p.ssn = "1";
ArrayList<Person> list = new ArrayList<Person>();
list.add(p);
```

```
p = new Person ();  
p.ssn = "2";  
list.add(p);  
Integer v = (Integer)amgr.callReduceAgent(agent, list);
```

### Welche Aufgaben kann ein Agent übernehmen?

Der Agent kann beliebige ObjectMap- oder EntityManager-Operationen in dem lokalen Shard ausführen, in dem er aktiv ist. Der Agent erhält ein Session-Objekt und kann Daten in der Partition, die vom Session-Objekt dargestellt wird, hinzufügen, aktualisieren, lesen oder entfernen. Einige Anwendungen fragen nur Daten vom Grid ab, aber Sie können einen Agenten auch so schreiben, dass das Alter aller Personen, die einer bestimmten Abfrage entsprechen, um 1 erhöht wird. Beim Aufruf des Agenten wird eine Transaktion im Session-Objekt erstellt, die festgeschrieben wird, wenn der Agent zurückkehrt, sofern keine Ausnahme ausgelöst wird.

### Fehlerbehandlung

Wenn ein Map-Agent mit einem unbekanntem Schlüssel aufgerufen wird, ist der zurückgegebene Wert ein Fehlerobjekt, das die Schnittstelle "EntryErrorValue" implementiert.

### Transaktionen

Ein Map-Agent wird in einer anderen Transaktion als der Client ausgeführt. Agentenaufrufe können zu einer einzigen Transaktion gruppiert werden. Wenn ein Fehler im Agenten auftritt (eine Ausnahme ausgelöst wird), wird die Transaktion rückgängig gemacht. Alle Agenten, die in einer Transaktion erfolgreich ausgeführt wurden, werden zusammen mit dem fehlgeschlagenen Agenten rückgängig gemacht. AgentManager führt die rückgängig gemachten Agenten, die erfolgreich ausgeführt wurden, in einer neuen Transaktion erneut aus.

---

## API-Dokumentation

Die API-Dokumentation zu WebSphere eXtreme Scale enthält Informationen, in denen Sie nach Paket- oder Klassennamen suchen können, um Einzelheiten über System- oder Anwendungsprogrammierschnittstellen zu finden.

Sie finden die API-Dokumentation im Information Center von WebSphere eXtreme Scale Information Center.



---

## Kapitel 4. Systemanwendungsprogrammierschnittstellen und -Plug-ins

Ein Plug-in ist eine Komponente, die den Plug-in-Komponenten, zu denen ObjectGrid und BackingMap gehören, eine Funktion bereitstellt. Um eXtreme Scale möglichst effizient als speicherinternes Daten-Grid oder Datenbankverarbeitungsbereich zu verwenden, müssen Sie sorgfältig überlegen, wie Sie die Leistung mit den verfügbaren Plug-ins am besten maximieren können.

---

### Einführung in Plug-ins

Ein Plug-in in WebSphere eXtreme Scale ist eine Komponente, die einen bestimmten Typ von Funktion für die Plug-in-Komponenten bereitstellt, zu denen ObjectGrid und BackingMap gehören. WebSphere eXtreme Scale stellt mehrere Plug-in-Punkte bereit, über die Anwendungen und Cacheprovider in verschiedene Datenspeicher und alternative Client-APIs integriert werden können und die Gesamtleistung des Caches verbessert werden kann. Im Lieferumfang des Produkts sind mehrere vordefinierte Plug-ins enthalten, aber Sie können auch eigene Plug-ins für die Anwendung erstellen.

Alle Plug-ins sind konkrete Klassen, die ein oder mehrere Plug-in-Schnittstellen von eXtreme Scale implementieren. Diese Klassen werden anschließend von ObjectGrid zur entsprechenden Zeit instanziiert und aufgerufen. Bei ObjectGrid und BackingMap können jeweils angepasste Plug-ins registriert werden.

#### ObjectGrid-Plug-ins

Die folgenden Plug-ins sind für eine ObjectGrid-Instanz verfügbar:

- **TransactionCallback:** Ein TransactionCallback-Plug-in stellt Transaktionslebenszyklusereignisse bereit.
- **ObjectGridEventListener:** Ein ObjectGridEventListener-Plug-in stellt ObjectGrid-Lebenszyklusereignisse für das ObjectGrid, Shards und Transaktionen bereit.
- **SubjectSource, MapAuthorization, SubjectValidation:** eXtreme Scale bietet mehrere Sicherheitsendpunkte, an denen angepasste Authentifizierungsverfahren mit eXtreme Scale integriert werden können.

#### Allgemeine Voraussetzungen für ObjectGrid-Plug-ins

Das ObjectGrid instanziiert und initialisiert Plug-in-Instanzen unter Verwendung von JavaBeans-Konventionen. Für alle Implementierungen der zuvor aufgeführten Plug-ins gelten die folgenden Voraussetzungen:

- Die Plug-in-Klasse muss eine öffentliche Ausgangsklasse sein.
- Die Plug-in-Klasse muss einen öffentlichen Konstruktor ohne Argumente haben.
- Die Plug-in-Klasse muss (gegebenenfalls) im Klassenpfad für Server und Clients verfügbar sein.
- Attribute müssen über JavaBeans-Eigenschaftenmethoden gesetzt werden.
- Plug-ins werden, sofern nicht anders angegeben, vor der Initialisierung des ObjectGrids initialisiert und können nach der Initialisierung des ObjectGrids nicht mehr geändert werden.

## BackingMap-Plug-ins

Die folgenden Plug-ins sind für eine BackingMap verfügbar:

- **Evictor:** Ein Evictor-Plug-in (Bereinigungsprogramm) ist ein Standardmechanismus für das Entfernen von Cacheeinträgen und ein Plug-in für das Erstellen angepasster Bereinigungsprogramme.
- **Loader:** Ein Loader-Plug-in (Ladeprogramm) in einer ObjectGrid-Map dient als Speicher-cache für Daten, die gewöhnlich in einem persistenten Speicher auf demselben System oder einem anderen System gespeichert werden.
- **ObjectTransformer:** Mit einem ObjectTransformer-Plug-in können Sie Objekte im Cache serialisieren, entserialisieren und kopieren.
- **OptimisticCallback:** Mit einem OptimisticCallback-Plug-in können Sie Versionssteuerungs- und Vergleichsoperationen für Cacheobjekte anpassen, wenn Sie die optimistische Sperrstrategie verwenden.
- **MapEventListener:** Ein MapEventListener-Plug-in unterstützt Callback-Benachrichtigungen und signifikante Cachestatusänderungen für eine BackingMap.
- **Indexing:** Verwenden Sie das Indexierungsfeature, das vom MapIndexPlugin-Plug-in dargestellt wird, um einen Index oder mehrere Indizes für eine BackingMap zu erstellen, damit Datenzugriffe ohne Schlüssel unterstützt werden.

## Lebenszyklen von Plug-ins

Die meisten Plug-ins haben zusätzlich zu den Methoden, für die sie entworfen wurden, initialize- und destroy-Methoden oder funktional entsprechende Methoden. Diese speziellen Methoden jedes Plug-ins können an bestimmten Funktionspunkten aufgerufen werden. Die initialize- und destroy-Methoden definieren den Lebenszyklus von Plug-ins, die über ihre "Eigenerobjekte" gesteuert werden. Ein Eigenerobjekt ist das Objekt, das das jeweilige Plug-in tatsächlich verwendet. Ein Eigener kann ein Grid-Client, ein Server oder eine BackingMap sein.

Bei der Initialisierung von Eigenerobjekten rufen diese Objekte die initialize-Methode ihrer Plug-ins auf. Während des destroy-Zyklus von Eigenerobjekten wird deshalb auch die destroy-Methode der Plug-ins aufgerufen. Einzelheiten zu den speziellen Eigenschaften der initialize- und destroy-Methoden und weiteren Methoden, die mit jedem Plug-in ausgeführt werden können, finden Sie in den entsprechenden Abschnitten zum jeweiligen Plug-in.

Stellen Sie sich beispielsweise eine verteilte Umgebung vor. Die clientseitigen ObjectGrids und die serverseitigen ObjectGrids können jeweils eigene Plug-ins haben. Der Lebenszyklus eines clientseitigen ObjectGrids und somit der zugehörigen Plug-in-Instanzen ist von allen serverseitigen ObjectGrids und Plug-in-Instanzen unabhängig.

Angenommen, Sie haben in einer solchen verteilten Topologie ein ObjectGrid mit dem Namen "myGrid" in der Datei "objectGrid.xml" definiert und mit einem angepassten ObjectGridEventListener mit dem Namen "myObjectGridEventListener" konfiguriert. Die Datei "objectGridDeployment.xml" definiert die Implementierungsrichtlinie für das ObjectGrid "myGrid". Die Dateien "objectGrid.xml" und "objectGridDeployment.xml" werden beide zum Starten der Containerserver verwendet. Während des Starts des Containerservers wird die serverseitige Instanz des ObjectGrids "myGrid" initialisiert und die initialize-Methode der myObjectGridEventListener-Instanz aufgerufen, deren Eigener die myObject-

Grid-Instanz ist. Nach dem Start des Containerservers kann Ihre Anwendung eine Verbindung zur serverseitigen Instanz des ObjectGrids "myGrid" herstellen und eine clientseitige Instanz abrufen.

Beim Abrufen der clientseitigen Instanz des ObjectGrids "myGrid" durchläuft die clientseitige myGrid-Instanz ihren eigenen Initialisierungszyklus und ruft die initialize-Methode ihrer eigenen clientseitigen myObjectGridEventListener-Instanz auf. Diese clientseitige myObjectGridEventListener-Instanz ist von der serverseitigen myObjectGridEventListener-Instanz unabhängig. Ihr Lebenszyklus wird vom Instanzeigener gesteuert, der die clientseitige Instanz des ObjectGrids "myGrid" ist.

Wenn Ihre Anwendung die Verbindung trennt oder die clientseitige Instanz des ObjectGrids "myGrid" löscht, wird automatisch die destroy-Methode der eigenen clientseitigen myObjectGridEventListener-Instanz aufgerufen. Dies hat jedoch keine Auswirkung auf die serverseitige myObjectGridEventListener-Instanz. Die destroy-Methode der serverseitigen myObjectGridEventListener-Instanz wird während des destroy-Zyklus der serverseitigen Instanz des ObjectGrids "myGrid" nur dann aufgerufen, wenn ein Containerserver gestoppt wird. Das heißt, wenn ein Containerserver gestoppt wird, werden die enthaltenen ObjectGrid-Instanzen gelöscht, und die destroy-Methode aller zugehörigen Plug-ins aufgerufen.

Das vorherige Beispiel bezieht sich speziell für den Fall einer Client- und einer Serverinstanz eines ObjectGrids. Der Eigner eines Plug-ins kann aber auch eine BackingMap sein, und Sie müssen bei der Festlegung der Konfigurationen für Plug-ins, die Sie auf der Basis dieser Lebenszyklushinweise schreiben, sorgfältig vorgehen.

---

## Ereignis-Listener

Sie können die ObjectGridEventListener- und MapEventListener-Plug-ins verwenden, um Benachrichtigungen für verschiedene Ereignisse im eXtreme-Scale-Cache zu konfigurieren. Listener-Plug-ins werden wie andere eXtreme-Scale-Plug-ins bei einer ObjectGrid- oder BackingMap-Instanz registriert und fügen Integrations- und Anpassungspunkte für Anwendung und Cacheprovider hinzu.

### ObjectGridEventListener-Plug-in

Ein ObjectGridEventListener-Plug-in stellt eXtreme-Scale-Lebenszyklusereignisse für die ObjectGrid-Instanz, Shards und Transaktionen bereit. Verwenden Sie das ObjectGridEventListener-Plug-in, um Benachrichtigungen zu empfangen, wenn wichtige Ereignisse in einem ObjectGrid eintreten. Zu diesen Ereignissen gehören die Initialisierung von ObjectGrid, der Beginn einer Transaktion, das Ende einer Transaktion und das Löschen eines ObjectGrids. Wenn Sie diese Ereignisse empfangen möchten, erstellen Sie eine Klasse, die die Schnittstelle "ObjectGridEventListener" implementiert, und fügen Sie sie eXtreme Scale hinzu.

Weitere Informationen zum Schreiben eines ObjectGridEventListener-Plug-ins finden Sie im Abschnitt „ObjectGridEventListener-Plug-in“ auf Seite 141. Auch in der API-Dokumentation finden Sie weitere Informationen.

### ObjectGridEventListener-Instanzen hinzufügen und entfernen

Ein ObjectGrid kann mehrere ObjectGridEventListener-Listener haben. Verwenden Sie zum Hinzufügen und Entfernen der Listener die Methoden "addEventListener", "setEventListeners" und "removeEventListener" in der Schnittstelle "ObjectGrid". Sie können ObjectGridEventListener-Plug-ins auch deklarativ mit der ObjectGrid-

Deskriptordatei registrieren. Diesbezügliche Beispiele finden Sie im Abschnitt „ObjectGridEventListener-Plug-in“ auf Seite 141.

## MapEventListener-Plug-in

Ein MapEventListener-Plug-in stellt Callback-Benachrichtigungen und Benachrichtigungen über wichtige Cachestatusänderungen, die für eine BackingMap-Instanz eintreten, bereit. Einzelheiten zum Schreiben eines MapEventListener-Plug-ins finden Sie im Abschnitt „MapEventListener-Plug-in“. Weitere Informationen finden Sie auch in der API-Dokumentation.

### MapEventListener-Instanzen hinzufügen und entfernen

eXtreme Scale kann mehrere MapEventListener-Listener haben. Verwenden Sie zum Hinzufügen und Entfernen von Listnern die Methoden "addMapEventListener", "setMapEventListeners" und "removeMapEventListener" in der Schnittstelle "BackingMap". Sie können MapEventListener-Listener auch deklarativ mit der ObjectGrid-Deskriptordatei registrieren. Diesbezügliche Beispiele finden Sie im Abschnitt „MapEventListener-Plug-in“.

## MapEventListener-Plug-in

Ein MapEventListener-Plug-in stellt Callback-Benachrichtigungen und Benachrichtigungen über wichtige Cachestatusänderungen für ein BackingMap-Objekt, für den Abschluss des Preload-Prozesses für eine Map oder für das Entfernen eines Eintrags aus der Map bereit. Ein MapEventListener-Plug-in ist eine angepasste Klasse, die die Schnittstelle "MapEventListener" implementiert.

### Konventionen für MapEventListener-Plug-ins

Wenn Sie ein MapEventListener-Plug-in entwickeln, müssen Sie allgemeine Plug-in-Konventionen einhalten. Weitere Informationen zu den Plug-in-Konventionen finden Sie im Abschnitt „Einführung in Plug-ins“ auf Seite 137. Informationen zu anderen Typen von Listener-Plug-ins finden Sie im Abschnitt „Ereignis-Listener“ auf Seite 139.

Nachdem Sie eine MapEventListener-Implementierung geschrieben haben, können Sie sie der BackingMap-Konfiguration über das Programm oder über eine XML-Konfiguration hinzufügen.

### MapEventListener-Implementierung schreiben

Ihre Anwendung kann eine Implementierung des MapEventListener-Plug-ins enthalten. Das Plug-in muss die Schnittstelle "MapEventListener" implementieren, um wichtige Ereignisse zu einer Map empfangen zu können. Ereignisse werden an das MapEventListener-Plug-in gesendet, wenn ein Eintrag aus der Map entfernt wird oder wenn der Preload-Prozess für eine Map abgeschlossen ist.

### MapEventListener-Implementierung über XML integrieren

Eine MapEventListener-Implementierung kann mit XML konfiguriert werden. Die folgende XML muss in der Datei myGrid.xml enthalten sein:

```
<?xml version="1.0" encoding="UTF-8"?>
<objectGridconfig xmlns:xsi="http://www.w3.org/2001/XMLSchema-instance"
  xsi:schemaLocation="http://ibm.com/ws/objectgrid/config ../objectGrid.xsd"
  xmlns="http://ibm.com/ws/objectgrid/config">
  <objectGrids>
```



```

        <objectGrid name="myGrid">
            <backingMap name="myMap" pluginCollectionRef="myPlugins" />
        </objectGrid>
    </objectGrids>
</backingMapPluginCollections>
    <backingMapPluginCollection id="myPlugins">
        <bean id="MapEventListener" className=
            "com.company.org.MyMapEventListener" />
    </backingMapPluginCollection>
</backingMapPluginCollections>
</objectGridConfig>

```

Die Bereitstellung dieser Datei für eine ObjectGridManager-Instanz vereinfacht die Erstellung der Konfiguration. Das folgende Code-Snippet veranschaulicht, wie eine ObjectGrid-Instanz mit dieser XML-Datei erstellt wird. Die neu erstellte ObjectGrid-Instanz hat ein definiertes MapEventListener-Plug-in für die BackingMap "myMap":

```

ObjectGridManager objectGridManager =
    ObjectGridManagerFactory.getObjectGridManager();
ObjectGrid myGrid =
    objectGridManager.createObjectGrid("myGrid", new URL("file:etc/test/myGrid.xml"),
        true, false);

```

## MapEventListener-Implementierung programmgesteuert integrieren

Der Klassenname für das angepasste MapEventListener-Plug-in ist com.company.org.MyMapEventListener. Diese Klasse implementiert die Schnittstelle "MapEventListener". Das folgende Code-Snippet erstellt das angepasste MapEventListener-Objekt und fügt es einem BackingMap-Objekt hinzu:

```

ObjectGridManager objectGridManager =
    ObjectGridManagerFactory.getObjectGridManager();
ObjectGrid myGrid = objectGridManager.createObjectGrid("myGrid", false);
BackingMap myMap = myGrid.defineMap("myMap");
MyMapEventListener myListener = new MyMapEventListener();
myMap.addMapEventListener(myListener);

```

## ObjectGridEventListener-Plug-in

Ein ObjectGridEventListener-Plug-in stellt Lebenszykluseignisse von WebSphere eXtreme Scale für das ObjectGrid, Shards und Transaktionen bereit. Ein ObjectGridEventListener-Plug-in stellt Benachrichtigungen bereit, wenn ein ObjectGrid initialisiert oder gelöscht wird und wenn eine Transaktion gestartet oder beendet wird. ObjectGridEventListener-Plug-ins sind angepasste Klassen, die die Schnittstelle "ObjectGridEventListener" implementieren. Die Plug-ins folgen den allgemeinen Konventionen für eXtreme-Scale-Plug-ins und enthalten optional ObjectGridEventGroup-Unterschnittstellen.

### Übersicht

Ein ObjectGridEventListener-Plug-in ist hilfreich, wenn ein Loader-Plug-in verfügbar ist und Sie JDBC-Verbindungen (Java Database Connectivity) oder Verbindungen zu einem Back-End herstellen müssen, wenn Transaktionen gestartet und beendet werden. Gewöhnlich werden ein ObjectGridEventListener-Plug-in und ein Loader-Plug-in zusammen geschrieben.

### ObjectGridEventListener-Plug-in schreiben

Ein ObjectGridEventListener-Plug-in muss die Schnittstelle "ObjectGridEventListener" implementieren, um Benachrichtigungen über wichtige eXtreme-Scale-Ereig-

nisse empfangen zu können. Wenn Sie zusätzliche Ereignisbenachrichtigungen empfangen möchten, können Sie die folgenden Schnittstellen implementieren. Diese Unterschnittstellen sind in der Schnittstelle "ObjectGridEventGroup" enthalten:

- Schnittstelle "ShardEvents"
- Schnittstelle "ShardLifecycle"
- Schnittstelle "TransactionEvents"

Weitere Informationen zu diesen Schnittstellen finden Sie in der API-Dokumentation.

## Shard-Ereignisse

Wenn der Katalogservice primäre Shards und oder Replikat-Shards einer Partition an eine JVM verteilt, wird eine neue ObjectGrid-Instanz in dieser JVM erstellt, die diese Shards aufnimmt. Einige Anwendungen, die Threads in der JVM mit dem primären Shard starten müssen, müssen über diese Ereignisse benachrichtigt werden. Die Schnittstelle "ObjectGridEventGroup.ShardEvents" deklariert die Methoden "shardActivate" und "shardDeactivate". Diese Methoden werden nur aufgerufen, wenn ein Shard als primäres Shard aktiviert wird und wenn das Shard als primäres Shard inaktiviert wird. Diese beiden Ereignisse ermöglichen der Anwendung, zusätzliche Threads zu starten, wenn das Shard ein primäres Shard ist, und die Threads zu stoppen, wenn das Shard wird zu einem Replikat heruntergestuft wird oder wenn das Shard einfach außer Betrieb genommen wird.

Eine Anwendung kann feststellen, welche Partition aktiviert wurde, indem Sie mit der Methode "ObjectGrid#getMap" eine bestimmte BackingMap in der ObjectGrid-Referenz sucht, die an die Methode "shardActivate" übergeben wurde. Die Anwendung kann die Partitionsnummer dann mit Hilfe der Methode "BackingMap#getPartitionId()" anzeigen. Die Partitionen sind von 0 bis zur Anzahl der Partitionen im Implementierungsdeskriptor minus eins nummeriert.

## Lebenszykluseignisse für Shards

Ereignisse der Methoden "ObjectGridEventListener.initialize" und "ObjectGridEventListener.destroy" werden über die Schnittstelle "ObjectGridEventGroup.ShardLifecycle" bereitgestellt.

## Transaktionsereignisse

Ereignisse der Methoden "ObjectGridEventListener.transactionBegin" und "ObjectGridEventListener.transactionEnd" werden über die Schnittstelle "ObjectGridEventGroup.TransactionEvents" bereitgestellt.

## Vorteile dieses Ansatzes

Wenn ein ObjectGridEventListener-Plug-in die Schnittstellen "ObjectGridEventListener" und "ShardLifecycle" implementiert, sind die Lebenszykluseignisse für Shards die einzigen Ereignisse, die dem Listener zugestellt werden. Nach der Implementierung der neuen inneren ObjectGridEventGroup-Schnittstellen, stellt diese eXtreme-Scale-Instanz nur diese speziellen Ereignisse über die neuen Schnittstellen bereit. Mit diesem Implementierungscode wird die Abwärtskompatibilität gewährleistet. Wenn Sie die neuen inneren Schnittstellen verwenden, können jetzt nur die speziellen Ereignisse empfangen werden, die benötigt werden.

## ObjectGridEventListener-Plug-in verwenden

Wenn Sie ein angepasstes ObjectGridEventListener-Plug-in verwenden möchten, müssen Sie zuerst eine Klasse erstellen, die die Schnittstelle "ObjectGridEventListener" und alle optionalen ObjectGridEventGroup-Unterschnittstellen implementiert. Fügen Sie einen angepassten Listener einer eXtreme-Scale-Instanz hinzu, um Benachrichtigungen über wichtige Ereignisse zu empfangen. Sie können beim Hinzufügen eines ObjectGridEventListener-Plug-ins zur eXtreme-Scale-Konfiguration zwischen zwei Methoden wählen: programmgesteuerte Konfiguration und XML-Konfiguration.

### ObjectGridEventListener-Plug-in programmgesteuert konfigurieren

Angenommen, der Klassenname des Ereignis-Listeners eXtreme Scale ist com.company.MyObjectGridEventListener. Diese Klasse implementiert die Schnittstelle "ObjectGridEventListener". Das folgende Code-Snippet erstellt das angepasste ObjectGridEventListener-Plug-in und fügt es einer eXtreme-Scale-Instanz hinzu:

```
ObjectGridManager objectGridManager = ObjectGridManagerFactory.getObjectGridManager();
ObjectGrid myGrid = objectGridManager.createObjectGrid("myGrid", false);
MyObjectGridEventListener myListener = new MyObjectGridEventListener();
myGrid.addEventListener(myListener);
```

### ObjectGridEventListener-Plug-in mit XML konfigurieren

Sie können ein ObjectGridEventListener-Plug-in auch mit XML konfigurieren. Die folgende XML erstellt eine Konfiguration, die dem zuvor beschriebenen ObjectGrid-Ereignis-Listener entspricht, der über das Programm erstellt wird: Der folgende Text muss in der Datei myGrid.xml enthalten sein:

```
<?xml version="1.0" encoding="UTF-8"?>
<objectGridConfig xmlns:xsi="http://www.w3.org/2001/XMLSchema-instance"
  xsi:schemaLocation="http://ibm.com/ws/objectgrid/config ../objectGrid.xsd"
  xmlns="http://ibm.com/ws/objectgrid/config">
  <objectGrids>
    <objectGrid name="myGrid">
      <bean id="ObjectGridEventListener"
        className="com.company.org.MyObjectGridEventListener" />
      <backingMap name="Book"/>
    </objectGrid>
  </objectGrids>
</objectGridConfig>
```

Beachten Sie, dass die Bean-Deklarationen vor den BackingMap-Deklarationen stehen müssen. Stellen Sie diese Datei dem ObjectGridManager-Plug-in bereit, um die Erstellung der Konfiguration zu vereinfachen. Das folgende Code-Snippet veranschaulicht, wie eine ObjectGrid-Instanz mit dieser XML-Datei erstellt wird. In der erstellten ObjectGrid-Instanz ist ein ObjectGridEventListener-Plug-in für das ObjectGrid "myGrid" definiert.

```
ObjectGridManager objectGridManager = ObjectGridManagerFactory.getObjectGridManager();
ObjectGrid myGrid = objectGridManager.createObjectGrid("myGrid",
  new URL("file:etc/test/myGrid.xml"), true, false);
```

---

## Index-Plug-in schreiben

Mit einem MapIndexPlugin-Plug-in oder Index können Sie angepasste Indexierungsstrategien erstellen, die über die integrierten Indizes hinausgehen, die von eXtreme Scale bereitgestellt werden.

Allgemeine Informationen zur Indexierung finden Sie im Abschnitt „Indexierung“ auf Seite 121.

Informationen zur Verwendung der Indexierung finden Sie im Abschnitt „Indexierung für den Datenzugriff ohne Schlüssel verwenden“ auf Seite 123.

MapIndexPlugin-Implementierungen müssen die Schnittstelle "MapIndexPlugin verwenden und die allgemeinen eXtreme-Scale-Konventionen für Plug-ins einhalten.

In den folgenden Abschnitten werden einige wichtige Methoden der Schnittstelle "Index" beschrieben.

## Methode "setProperties"

Verwenden Sie die Methode "setProperties", um das Index-Plug-in programmgesteuert zu initialisieren. Der an die Methode übergebene Objektparameter "Properties" muss die erforderlichen Konfigurationsdaten enthalten, damit das Index-Plug-in ordnungsgemäß initialisiert werden kann. Die Implementierung der Methode "setProperties" und die Methode "getProperties" sind in einer verteilten Umgebung erforderlich, weil die Konfiguration des Index-Plug-ins zwischen den Client- und Serverprozessen übergeben wird. Es folgt ein Implementierungsbeispiel für diese Methode:

```
setProperties(Properties properties)

// Mustercode für die Methode "setProperties"
public void setProperties(Properties properties) {
    ivIndexProperties = properties;

    String ivRangeIndexString = properties.getProperty("rangeIndex");
    if (ivRangeIndexString != null && ivRangeIndexString.equals("true")) {
        setRangeIndex(true);
    }
    setName(properties.getProperty("indexName"));
    setAttributeName(properties.getProperty("attributeName"));

    String ivFieldAccessAttributeString = properties.getProperty("fieldAccessAttribute");
    if (ivFieldAccessAttributeString != null && ivFieldAccessAttributeString.equals("true")) {
        setFieldAccessAttribute(true);
    }

    String ivPOJOKeyIndexString = properties.getProperty("POJOKeyIndex");
    if (ivPOJOKeyIndexString != null && ivPOJOKeyIndexString.equals("true")) {
        setPOJOKeyIndex(true);
    }
}
```

## Methode "getProperties"

Die Methode "getProperties" extrahiert die Konfiguration des Index-Plug-ins aus einer MapIndexPlugin-Instanz. Sie können die extrahierten Eigenschaften verwenden, um eine weitere MapIndexPlugin-Instanz mit denselben internen Status zu initialisieren. Die Implementierungen der Methode "getProperties" und der Methode "setProperties" sind in einer verteilten Umgebung erforderlich. Es folgt ein Implementierungsbeispiel für die Methode "getProperties":

```
getProperties()

// Mustercode für die Methode "getProperties"
public Properties getProperties() {
    Properties p = new Properties();
    p.put("indexName", indexName);
    p.put("attributeName", attributeName);
    p.put("rangeIndex", ivRangeIndex ? "true" : "false");
    p.put("fieldAccessAttribute", ivFieldAccessAttribute ? "true" : "false");
    p.put("POJOKeyIndex", ivPOJOKeyIndex ? "true" : "false");
    return p;
}
```

## Methode "setEntityMetadata"

Die Methode "setEntityMetadata" wird von der Laufzeitumgebung von WebSphere eXtreme Scale während der Initialisierung aufgerufen, um das EntityMetadata-Objekt der zugeordneten BackingMap in der MapIndexPlugin-Instanz zu setzen. Das EntityMetadata-Objekt ist für die Unterstützung der Indexierung von Tupelobjekten erforderlich. Ein Tupel ist eine Datenmenge, die ein Entitätsobjekt oder dessen Schlüssel darstellt. Wenn die BackingMap für eine Entität bestimmt ist, müssen Sie diese Methode implementieren.

Der folgende Mustercode implementiert die Methode "setEntityMetadata":

```
setEntityMetadata(EntityMetadata entityMetadata)

// Mustercode für die Methode "setEntityMetadata"
public void setEntityMetadata(EntityMetadata entityMetadata) {
    ivEntityMetadata = entityMetadata;
    if (ivEntityMetadata != null) {
        // Die ist eine Tupel-Map
        TupleMetadata valueMetadata = ivEntityMetadata.getValueMetadata();
        int numAttributes = valueMetadata.getNumAttributes();
        for (int i = 0; i < numAttributes; i++) {
            String tupleAttributeName = valueMetadata.getAttribute(i).getName();
            if (attributeName.equals(tupleAttributeName)) {
                ivTupleValueIndex = i;
                break;
            }
        }

        if (ivTupleValueIndex == -1) {
            // Das Attribut wurde nicht im Tupelwert gefunden. Versuchen, es im Schlüssel-Tupel zu finden.
            // Wenn es im Schlüssel-Tupel vorhanden ist, implizite Schlüsselindexierung nach
            // einem der Schlüsselattribute des Tupels
            TupleMetadata keyMetadata = ivEntityMetadata.getKeyMetadata();
            numAttributes = keyMetadata.getNumAttributes();
            for (int i = 0; i < numAttributes; i++) {
                String tupleAttributeName = keyMetadata.getAttribute(i).getName();
                if (attributeName.equals(tupleAttributeName)) {
                    ivTupleValueIndex = i;
                    ivKeyTupleAttributeIndex = true;
                    break;
                }
            }
        }

        if (ivTupleValueIndex == -1) {
            // Wenn das entityMetadata-Objekt ungleich null ist und
            // attributeName nicht in entityMetadata-Objekt gefunden wird,
            // ist dies ein Fehler.
            throw new ObjectGridRuntimeException("Invalid attributeName. Entity: " +
                ivEntityMetadata.getName());
        }
    }
}
```

## Methoden für Attributnamen

Die Methode "setAttributeName" legt den Namen des zu indexierenden Attributs fest. Die zwischengespeicherte Objektklasse muss die Methode "get" für das indexierte Attribut bereitstellen. Wenn das Objekt beispielsweise ein Attribut "employeeName" oder "EmployeeName" hat, ruft der Index die Methode "getEmployeeName" für das Objekt auf, um den Attributwert zu extrahieren. Der Attributname muss mit dem Namen in der get-Methode identisch sein, und das Attribut muss die Schnittstelle "Comparable" implementieren. Wenn das Attribut ein boolesches Attribut ist, können Sie auch das Methodenmuster "isAttributeName" verwenden.

Die Methode "getAttributeName" gibt den Namen des indexierten Attributs zurück.

## Methode "getAttribute"

Die Methode "getAttribute" gibt den Wert des indexierten Attributs aus dem angegebenen Objekt zurück. Wenn ein Employee-Objekt beispielsweise ein Attribut mit

dem Namen "employeeName" hat, das indexiert ist, können Sie die Methode "getAttribute" verwenden, um den Wert des Attributs "employeeName" aus dem angegebenen Employee-Objekt zu extrahieren. Diese Methode ist in einer verteilten Umgebung von WebSphere eXtreme Scale erforderlich.

```
getAttribute(Object value)

// Mustercode für die Methode "getAttribute"
public Object getAttribute(Object value) throws ObjectGridRuntimeException {
    if (ivPOJOKeyIndex) {
        // Bei der Indexierung von POJO-Schlüsseln muss das Attribut nicht aus dem Wertobjekt abgerufen werden.
        // Der Schlüssel selbst ist der zum Erstellen des Index verwendete Attributwert.
        return null;
    }

    try {
        Object attribute = null;
        if (value != null) {
            // Tupelwert bearbeiten, wenn ivTupleValueIndex != -1
            if (ivTupleValueIndex == -1) {
                // regulärer Wert
                if (ivFieldAccessAttribute) {
                    attribute = this.getAttributeField(value).get(value);
                } else {
                    attribute = getAttributeMethod(value).invoke(value, emptyArray);
                }
            } else {
                // Tupelwert
                attribute = extractValueFromTuple(value);
            }
        }
        return attribute;
    } catch (InvocationTargetException e) {
        throw new ObjectGridRuntimeException(
            "Caught unexpected Throwable during index update processing,
            index name = " + indexName + ": " + t,
            t);
    } catch (Throwable t) {
        throw new ObjectGridRuntimeException(
            "Caught unexpected Throwable during index update processing,
            index name = " + indexName + ": " + t,
            t);
    }
}
```

---

## TransactionCallback-Plug-in

Verwenden Sie das TransactionCallback-Plug-in, um Versionssteuerungs- und Vergleichsoperationen für Cacheobjekte anzupassen, wenn Sie die optimistische Sperrstrategie verwenden.

Sie können ein Plug-in-fähiges optimistisches Callback-Objekt bereitstellen, das die Schnittstelle "com.ibm.websphere.objectgrid.plugins.OptimisticCallback" implementiert. Für Entitäts-Maps wird automatisch ein OptimisticCallback-Plug-in mit hoher Leistung konfiguriert.

### Zweck

Verwenden Sie die Schnittstelle "OptimisticCallback" für die Unterstützung optimistischer Vergleichsoperationen für die Werte einer Map. Eine OptimisticCallback-Implementierung ist erforderlich, wenn Sie die optimistische Sperrstrategie verwenden. WebSphere eXtreme Scale stellt eine Standardimplementierung von OptimisticCallback bereit. Gewöhnlich muss die Anwendung eine eigene Implementierung der Schnittstelle "OptimisticCallback" integrieren. Weitere Informationen finden Sie im Abschnitt in den Informationen zu Sperrstrategien im Handbuch *Produktübersicht*.

### Standardimplementierung

Das eXtreme-Scale-Framework stellt eine Standardimplementierung der Schnittstelle "OptimisticCallback" bereit, die verwendet wird, wenn die Anwendung kein anwendungsdefiniertes OptimisticCallback-Objekt bereitstellt, wie im folgenden

Abschnitt veranschaulicht wird. Die Standardimplementierung gibt immer den Sonderwert `NULL_OPTIMISTIC_VERSION` als Versionsobjekt für den Wert zurück und aktualisiert das Versionsobjekt nie. Diese Aktion macht einen optimistischen Vergleich zu einer Funktion mit "Nulloperation". In den meisten Fällen ist die Funktion mit "Nulloperation" nicht angebracht, wenn die optimistische Sperrstrategie verwendet wird. Ihre Anwendungen müssen die Schnittstelle "OptimisticCallback" implementieren und eigene OptimisticCallback-Implementierungen integrieren, damit die Standardimplementierung nicht verwendet wird. Es gibt jedoch mindestens ein Szenario, in dem die bereitgestellte OptimisticCallback-Standardimplementierung hilfreich ist. Stellen Sie sich die folgende Situation vor:

- Es wird ein Loader (Ladeprogramm) für die BackingMap integriert.
- Der Loader weiß ohne Hilfe eines OptimisticCallback-Plug-ins, wie der optimistische Vergleich durchgeführt wird.

Wie kann der Loader nun ohne Hilfe eines OptimisticCallback-Objekts wissen, wie mit der optimistischen Versionssteuerung zu verfahren ist? Der Loader kennt das Wertobjekt für die Klasse und weiß, welches Feld des Wertobjekts als Wert für die optimistische Versionssteuerung verwendet wird. Angenommen, die folgende Schnittstelle wird für das Wertobjekt der Map "Employee" verwendet:

```
public interface Employee
{
    // Für die optimistische Versionssteuerung verwendete Folgennummer.
    public long getSequenceNumber();
    public void setSequenceNumber(long newSequenceNumber);
    // Weitere get/set-Methoden für andere Felder des Employee-Objekts.
}
```

In diesem Fall weiß der Loader, dass er die Methode "getSequenceNumber" verwenden kann, um die aktuellen Versionsinformationen für ein Employee-Wertobjekt abzurufen. Der Loader erhöht den zurückgegebenen Wert um eins, um eine neue Versionsnummer zu generieren, bevor er den persistenten Speicher mit dem neuen Employee-Wert aktualisiert. Für einen JDBC-Loader (Java Database Connectivity) wird die aktuelle Folgennummer in der WHERE-Klausel einer überqualifizierten SQL-Anweisung "update" verwendet. Der Loader verwendet die neu generierte Folgennummer, um die Folgennummernspalte auf den neuen Folgennummernwert zu setzen.

Eine weitere Möglichkeit ist die, dass der Loader eine vom Back-End bereitgestellte Funktion verwendet, die eine verdeckte Spalte, die für die optimistische Versionssteuerung verwendet werden kann, automatisch aktualisiert. In manchen Fällen kann unter Umständen eine gespeicherte Prozedur oder ein Trigger verwendet werden, um eine Spalte zu verwalten, die Informationen zur Versionssteuerung enthält. Wenn der Loader eine dieser Techniken für die Verwaltung der Informationen zur optimistischen Versionssteuerung verwendet, muss die Anwendung keine eigene OptimisticCallback-Implementierung bereitstellen. Sie können die OptimisticCallback-Standardimplementierung verwenden, weil der Loader die optimistische Versionssteuerung ohne Hilfe eines OptimisticCallback-Objekts übernehmen kann.

## Standardimplementierung für Entitäten

Entitäten werden im ObjectGrid mit Hilfe von Tupelobjekten gespeichert. Die OptimisticCallback-Standardimplementierung verhält sich ähnlich wie bei Maps, die keine Entitäts-Maps sind. Das Versionsfeld in der Entität wird jedoch mit der Annotation "@Version" bzw. dem Versionsattribut in der XML-Deskriptordatei der Entität angegeben.

Die gültigen Datentypen für das Versionsattribut sind int, Integer, short, Short, long, Long und java.sql.Timestamp. Für eine Entität darf nur ein einziges Versionsattribut definiert werden. Das Versionsattribut darf nur während der Erstellung definiert werden. Sobald die Entität als persistent definiert wird, darf der Wert des Versionsattributs nicht mehr geändert werden.

Wenn kein Versionsattribut konfiguriert ist und die optimistische Sperrstrategie verwendet wird, wird das vollständige Tupel implizit über den Status des Tupels versionsgesteuert.

Im folgenden Beispiel hat die Entität "Employee" ein Versionsattribut mit dem Namen "SequenceNumber" und dem Typ "long":

```
@Entity
public class Employee
{
    private long sequence;
    // Für die optimistische Versionssteuerung verwendete Folgennummer.
    @Version
    public long getSequenceNumber() {
        return sequence;
    }
    public void setSequenceNumber(long newSequenceNumber) {
        this.sequence = newSequenceNumber;
    }
    // Weitere get/set-Methoden für andere Felder des Employee-Objekts.
}
```

## OptimisticCallback-Implementierung schreiben

Ein OptimisticCallback-Plug-in muss die Schnittstelle "OptimisticCallback" implementieren und die folgenden Konventionen für ObjectGrid-Plug-ins einhalten.

Die folgende Liste enthält Beschreibungen und Hinweise für alle Methoden in der Schnittstelle "OptimisticCallback":

### NULL\_OPTIMISTIC\_VERSION

Dieser Sonderwert wird von der Methode "getVersionedObjectForValue" zurückgegeben, wenn die OptimisticCallback-Standardimplementierung an Stelle einer anwendungsdefinierten OptimisticCallback-Implementierung verwendet wird.

### Methode "getVersionedObjectForValue"

Die Methode "getVersionedObjectForValue" kann eine Kopie des Werts oder ein Attribut des Werts zurückgeben, das für Versionssteuerungszwecke verwendet werden kann. Diese Methode wird aufgerufen, wenn ein Objekt einer Transaktion zugeordnet wird. Wenn in einer BackingMap kein Loader definiert ist, verwendet die BackingMap diesen Wert während der Festschreibung, um einen optimistischen Versionsvergleich durchzuführen. Der optimistische Versionsvergleich wird von der BackingMap verwendet, um sicherzustellen, dass die Version des Map-Eintrags seit dem ersten Zugriff der Transaktion, die den Map-Eintrag geändert hat, nicht geändert wurde. Wenn eine andere Transaktion die Version für diesen Map-Eintrag bereits geändert hat, schlägt der Versionsvergleich fehl, und die BackingMap zeigt eine Ausnahme des Typs "OptimisticCollisionException" an, um eine Rollback-Operation für die Transaktion zu erzwingen. Wenn ein Loader integriert ist, verwendet die BackingMap die Informationen für die optimistische Versionssteuerung nicht. Stattdessen ist der Loader für die Durchführung der optimistischen Versionssteuerung und die Aktualisierung der Versionssteuerungsinformationen zuständig,



sofern dies erforderlich ist. Der Loader ruft gewöhnlich das erste Versionssteuerungsobjekt von dem LogElement-Objekt ab, das an die Methode "batchUpdate" im Loader übergeben wurde, die aufgerufen wird, wenn eine Flush-Operation durchgeführt oder eine Transaktion festgeschrieben wird.

Der folgende Code zeigt die vom EmployeeOptimisticCallbackImpl-Objekt verwendete Implementierung:

```
public Object getVersionedObjectForValue(Object value)
{
    if (value == null)
    {
        return null;
    }
    else
    {
        Employee emp = (Employee) value;
        return new Long( emp.getSequenceNumber() );
    }
}
```

Wie im vorherigen Beispiel gezeigt, wird das Attribut "sequenceNumber" in einem vom Loader erwarteten Objekt des Typs "java.lang.Long" zurückgegeben. Dies impliziert, dass die Person, die den Loader geschrieben hat, auch die EmployeeOptimisticCallbackImpl-Implementierung geschrieben hat bzw. eng mit der Person zusammengearbeitet hat, die EmployeeOptimisticCallbackImpl implementiert hat, z. B. mit dieser Person den von der Methode "getVersionedObjectForValue" zurückgegebenen Wert vereinbart hat. Wie zuvor beschrieben, gibt die OptimisticCallback-Standardimplementierung den Sonderwert NULL\_OPTIMISTIC\_VERSION als Versionsobjekt zurück.

## Methode "updateVersionedObjectForValue"

Die Methode "updateVersionedObjectForValue" wird aufgerufen, wenn eine Transaktion einen Wert aktualisiert hat und ein neues versionsgesteuertes Objekt erforderlich ist. Wenn die Methode "getVersionedObjectForValue" ein Attribut des Werts zurückgibt, aktualisiert diese Methode gewöhnlich den Attributwert mit einem neuen Versionsobjekt. Wenn die Methode "getVersionedObjectForValue" eine Kopie des Werts zurückgibt, führt diese Methode gewöhnlich keine Aktualisierung durch. Die OptimisticCallback-Standardimplementierung führt keine Aktualisierung durch, da die Standardimplementierung der Methode "getVersionedObjectForValue" immer den Sonderwert NULL\_OPTIMISTIC\_VERSION als Versionsobjekt zurückgibt. Der folgende Beispielcode zeigt die vom EmployeeOptimisticCallbackImpl-Objekt im Abschnitt "OptimisticCallback" verwendete Implementierung:

```
public void updateVersionedObjectForValue(Object value)
{
    if ( value != null )
    {
        Employee emp = (Employee) value;
        long next = emp.getSequenceNumber() + 1;
        emp.updateSequenceNumber( next );
    }
}
```

Wie im vorherigen Beispiel gezeigt, wird das Attribut "sequenceNumber" um eins erhöht, so dass beim nächsten Aufruf der Methode "getVersionedObjectForValue" der zurückgegebene Wert vom Typ "java.lang.Long" einen langen Wert hat, der dem ursprünglichen Folgenummernwert plus eins entspricht, z. B. dem nächsten Versionswert für diese Employee-Instanz. Auch hier impliziert das Beispiel, dass die Person, die den Loader geschrieben hat, auch die EmployeeOptimisticCallback-

Impl-Implementierung geschrieben hat bzw. eng mit der Person zusammengearbeitet hat, die `EmployeeOptimisticCallbackImpl` implementiert hat.

### **Methode "serializeVersionedValue"**

Diese Methode schreibt den versionsgesteuerten Wert in den angegebenen Datenstrom. Je nach Implementierung kann der versionsgesteuerte Wert verwendet werden, um optimistische Aktualisierungskollisionen zu identifizieren. In einigen Implementierungen ist der versionsgesteuerte Wert eine Kopie des ursprünglichen Werts. Andere Implementierungen können eine Folgenummer oder ein anderes Objekt haben, um die Version des Werts anzugeben. Da die tatsächliche Implementierung nicht bekannt ist, wird diese Methode bereitgestellt, damit die richtige Serialisierung durchgeführt werden kann. Die Standardimplementierung ruft die Methode "writeObject" auf.

### **Methode "inflateVersionedValue"**

Diese Methode akzeptiert die serialisierte Version des versionsgesteuerten Werts und gibt das tatsächliche versionsgesteuerte Wertobjekt zurück. Je nach Implementierung kann der versionsgesteuerte Wert verwendet werden, um optimistische Aktualisierungskollisionen zu identifizieren. In einigen Implementierungen ist der versionsgesteuerte Wert eine Kopie des ursprünglichen Werts. Andere Implementierungen können eine Folgenummer oder ein anderes Objekt haben, um die Version des Werts anzugeben. Da die tatsächliche Implementierung nicht bekannt ist, wird diese Methode bereitgestellt, damit die richtige Entserialisierung durchgeführt werden kann. Die Standardimplementierung ruft die Methode "readObject" auf.

## **Anwendungsdefinierte OptimisticCallback-Implementierung verwenden**

Sie können zum Hinzufügen einer anwendungsdefinierten `OptimisticCallback`-Implementierung zur `BackingMap`-Konfiguration zwischen zwei Ansätzen wählen: der programmgesteuerten Konfiguration und der XML-Konfiguration.

### **OptimisticCallback-Implementierung über das Programm integrieren**

Das folgende Beispiel veranschaulicht, wie eine Anwendung über das Programm ein `OptimisticCallback`-Objekt für die `BackingMap` "Employee" in der `ObjectGrid`-Instanz "grid1" integriert:

```
import com.ibm.websphere.objectgrid.ObjectGridManagerFactory;
import com.ibm.websphere.objectgrid.ObjectGridManager;
import com.ibm.websphere.objectgrid.ObjectGrid;
import com.ibm.websphere.objectgrid.BackingMap;
ObjectGridManager ogManager = ObjectGridManagerFactory.getObjectGridManager();
ObjectGrid og = ogManager.createObjectGrid( "grid1" );
BackingMap bm = dg.defineMap("employees");
EmployeeOptimisticCallbackImpl cb = new EmployeeOptimisticCallbackImpl();
bm.setOptimisticCallback( cb );
```

### **OptimisticCallback-Implementierung durch XML-Konfiguration integrieren**

Das `EmployeeOptimisticCallbackImpl`-Objekt im vorherigen Beispiel muss die Schnittstelle "OptimisticCallback" integrieren. Die Anwendung kann auch eine XML-Datei verwenden, um ihr `OptimisticCallback`-Objekt zu integrieren, wie im folgenden Beispiel gezeigt wird:

```
<?xml version="1.0" encoding="UTF-8"?>
<objectGridConfig xmlns:xsi="http://www.w3.org/2001/XMLSchema-instance"
xsi:schemaLocation="http://ibm.com/ws/objectgrid/config ../objectGrid.xsd"
```

```

xmlns="http://ibm.com/ws/objectgrid/config">
<objectGrids>
  <objectGrid name="grid1">
    <backingMap name="employees" pluginCollectionRef="employees" lockStrategy="OPTIMISTIC" />
  </objectGrid>
</objectGrids>

<backingMapPluginCollections>
  <backingMapPluginCollection id="employees">
    <bean id="OptimisticCallback" className="com.xyz.EmployeeOptimisticCallbackImpl" />
  </backingMapPluginCollection>
</backingMapPluginCollections>
</objectGridConfig>

```

## Einführung in Plug-in-Slots

Ein Plug-in-Slot ist ein transaktionsorientierter Speicherbereich, der für Plug-ins reserviert ist, die einen Transaktionskontext gemeinsam nutzen. Diese Slots bieten eXtreme-Scale-Plug-ins die Möglichkeit, miteinander zu kommunizieren, einen Transaktionskontext gemeinsam zu nutzen und sicherzustellen, dass Transaktionsressourcen innerhalb einer Transaktion korrekt und konsistent verwendet werden.

Ein Plug-in kann einen Transaktionskontext, z. B. eine Datenbankverbindung, eine JMS-Verbindung (Java Message Service) usw., in einem Plug-in-Slot speichern. Der gespeicherte Transaktionskontext kann von jedem Plug-in abgerufen werden, das die Nummer des Plug-in-Slots kennt, die als Schlüssel für den Abruf des Transaktionskontexts dient.

## Plug-in-Slots verwenden

Plug-in-Slots gehören zur Schnittstelle "TxID". Weitere Informationen zu dieser Schnittstelle finden Sie in der API-Dokumentation. Diese Slots sind Einträge in einer ArrayList-Feldgruppe. Plug-ins können einen Eintrag in der ArrayList-Feldgruppe reservieren, indem Sie die Methode "ObjectGrid.reserveSlot" aufrufen und anzeigen, dass sie einen Slot in allen TxID-Objekten verwenden möchten. Nachdem die Slots reserviert wurden, können Plug-ins einen Transaktionskontext in den Slots jedes TxID-Objekts speichern und den Kontext später abrufen. Die Operationen "put" und "get" werden über die Slot-Nummern koordiniert, die von der Methode "ObjectGrid.reserveSlot" zurückgegeben werden.

Ein Plug-in hat gewöhnlich einen Lebenszyklus. Die Verwendung von Plug-in-Slots muss in den Lebenszyklus des Plug-ins passen. Gewöhnlich muss das Plug-in Plug-in-Slots in der Initialisierungsphase reservieren und eine Slot-Nummer für jeden Slot anfordern. Zur normalen Laufzeit speichert das Plug-in zum entsprechenden Zeitpunkt einen Transaktionskontext im reservierten Slot im TxID-Objekt. Normalerweise geschieht dies am Anfang der Transaktion. Das Plug-in oder andere Plug-ins können dann den gespeicherten Transaktionskontext über die Slot-Nummer aus dem TxID-Objekt innerhalb der Transaktion abrufen.

Das Plug-in führt gewöhnlich eine Bereinigung durch, indem es den Transaktionskontext und die Slots entfernt. Das folgende Code-Snippet veranschaulicht, wie Plug-in-Slots in einem TransactionCallback-Plug-in verwendet werden:

```

public class DatabaseTransactionCallback implements TransactionCallback {
    int connectionSlot;
    int autoCommitConnectionSlot;
    int psCacheSlot;
    Properties ivProperties = new Properties();

    public void initialize(ObjectGrid objectGrid) throws TransactionCallbackException {
        // In der Initialisierungsphase die gewünschten Plug-in-Slots durch Aufruf
        // der Methode reserveSlot von ObjectGrid reservieren und den
        // gewünschten Slot-Namen mit TxID.SLOT_NAME übergeben.
        // Anmerkung: Sie müssen mit TxID.SLOT_NAME den Slot-Namen übergeben,
        // der für die Anwendung bestimmt ist.
    }

    try {
        // Zurückgegebene Slot-Nummern zwischenspeichern.
        connectionSlot = objectGrid.reserveSlot(TxID.SLOT_NAME);
    }
}

```

```

        psCacheSlot = objectGrid.reserveSlot(TxID.SLOT_NAME);
        autoCommitConnectionSlot = objectGrid.reserveSlot(TxID.SLOT_NAME);
    } catch (Exception e) {
    }
}

public void begin(TxID tx) throws TransactionCallbackException {
    // Am Anfang der Transaktion Transaktionskontexte in den reservierten
    // Slots speichern.
try {
    Connection conn = null;
    conn = DriverManager.getConnection(ivDriverUrl, ivProperties);
    tx.putSlot(connectionSlot, conn);
    conn = DriverManager.getConnection(ivDriverUrl, ivProperties);
    conn.setAutoCommit(true);
    tx.putSlot(autoCommitConnectionSlot, conn);
    tx.putSlot(psCacheSlot, new HashMap());
    } catch (SQLException e) {
        SQLException ex = getLastSQLException(e);
        throw new TransactionCallbackException("unable to get connection", ex);
    }
}

public void commit(TxID id) throws TransactionCallbackException {
    // Gespeicherte Transaktionskontexte abrufen, verwenden und anschließend
    // alle Transaktionsressourcen bereinigen.
try {
    Connection conn = (Connection) id.getSlot(connectionSlot);
    conn.commit();
    cleanUpSlots(id);
    } catch (SQLException e) {
        SQLException ex = getLastSQLException(e);
        throw new TransactionCallbackException("commit failure", ex);
    }
}

void cleanUpSlots(TxID tx) throws TransactionCallbackException {
    closePreparedStatements((Map) tx.getSlot(psCacheSlot));
    closeConnection((Connection) tx.getSlot(connectionSlot));
    closeConnection((Connection) tx.getSlot(autoCommitConnectionSlot));
}

/**
 * @param map
 */
private void closePreparedStatements(Map psCache) {
    try {
        Collection statements = psCache.values();
        Iterator iter = statements.iterator();
        while (iter.hasNext()) {
            PreparedStatement stmt = (PreparedStatement) iter.next();
            stmt.close();
        }
    } catch (Throwable e) {
    }
}

/**
 * Verbindung schließen und alle Throwable-Objekte abfangen.
 *
 * @param connection
 */
private void closeConnection(Connection connection) {
    try {
        connection.close();
    } catch (Throwable e1) {
    }
}

public void rollback(TxID id) throws TransactionCallbackException
    // Gespeicherte Transaktionskontexte abrufen, verwenden und anschließend
    // alle Transaktionsressourcen bereinigen.
try {
    Connection conn = (Connection) id.getSlot(connectionSlot);
    conn.rollback();
    cleanUpSlots(id);
    } catch (SQLException e) {
    }
}

public boolean isExternalTransactionActive(Session session) {
    return false;
}

// Getter-Methoden für die Slot-Nummern. Andere Plug-ins können die Slot-Nummern
// über diese Getter-Methoden anfordern.
public int getConnectionSlot() {
    return connectionSlot;
}

public int getAutoCommitConnectionSlot() {

```

```

        return autoCommitConnectionSlot;
    }
    public int getPreparedStatementSlot() {
        return psCacheSlot;
    }
}

```

Das folgende Code-Snippet veranschaulicht, wie ein Loader (Ladeprogramm) den gespeicherten Transaktionskontext abrufen kann, der mit dem vorherigen TransactionCallback-Plug-in-Beispielcode gespeichert wurde:

```

public class DatabaseLoader implements Loader
{
    DatabaseTransactionCallback tcb;
    public void preloadMap(Session session, BackingMap backingMap) throws LoaderException
    {
        // Die Methode "preload" ist die Initialisierungsmethode des Loaders.
        // Gewünschtes Plug-in von der Session- bzw. ObjectGrid-Instanz anfordern.
        tcb =
        (DatabaseTransactionCallback)session.getObjectGrid().getTransactionCallback();
    }
    public List get(TxID txid, List keyList, boolean forUpdate) throws LoaderException
    {
        // Gespeicherte Transaktionskontexte abrufen, die mit der TCB-Methode "begin" gespeichert wurden.
        Connection conn = (Connection)txid.getSlot(tcb.getConnectionSlot());
        // Abruf hier implementieren.
        return null;
    }
    public void batchUpdate(TxID txid, LogSequence sequence) throws LoaderException,
    OptimisticCollisionException
    {
        // Gespeicherte Transaktionskontexte abrufen, die mit der TCB-Methode "begin" gespeichert wurden.
        Connection conn = (Connection)txid.getSlot(tcb.getConnectionSlot());
        // Aktualisierung im Stapelbetrieb hier implementieren.
    }
}

```

## Externe Transaktionsmanager

Gewöhnlich beginnen eXtreme-Scale-Transaktionen mit der Methode "session.begin" und enden mit der Methode "session.commit". Wenn jedoch ein eXtreme Scale integriert ist, können Transaktionen von einem externen Transaktionskoordinator gestartet und beendet werden. In diesem Fall müssen die Methoden "session.begin" und "session.commit" nicht aufrufen.

### Externe Transaktionskoordination

Das TransactionCallback-Plug-in wurde mit der Methode "isExternalTransactionActive(Session session)" erweitert, die die eXtreme-Scale-Sitzung einer externen Transaktion zuordnet. Der Methoden-Header sieht wie folgt aus:

```
public synchronized boolean isExternalTransactionActive(Session session)
```

eXtreme Scale kann beispielsweise für die Integration mit WebSphere Application Server und WebSphere Extended Deployment konfiguriert werden.

Außerdem stellt eXtreme Scale ein integriertes WebSphere-Plug-in bereit. Im Abschnitt „TransactionCallback-Plug-in“ auf Seite 146 wird beschrieben, wie Sie das Plug-in für Umgebungen mit WebSphere Application Server erstellen, aber Sie können das Plug-in auch für andere Frameworks anpassen.

Der Schlüssel zu dieser nahtlosen Integration ist die Nutzung der API "ExtendedJTATransaction" in WebSphere Application Server Version 5.x und Version 6.x. Wenn Sie jedoch WebSphere Application Server Version 6.0.2 verwenden, müssen Sie für die Unterstützung dieser Methode APAR PK07848 anwenden. Verwenden Sie den folgenden Beispielcode, um einer ObjectGrid-Sitzung eine Transaktions-ID von WebSphere Application Server zuzuordnen:

```

/**
 * Diese Methode ist erforderlich, um einer ObjectGrid-Sitzung eine Transaktions-ID von
 * WebSphere Application Server zuzuordnen.
 */

```

```

Map/**/ localIdToSession;
public synchronized boolean isExternalTransactionActive(Session session)
{
    // Denken Sie daran, dass localid bedeutet, dass die Sitzung für spätere Verwendung gespeichert v
    localIdToSession.put(new Integer(jta.getLocalId()), session);
    return true;
}

```

## Externe Transaktion abrufen

Manchmal müssen Sie ein externes Transaktionserviceobjekt für das TransactionCallback-Plug-in abrufen. Suchen Sie im Server von WebSphere Application Server das ExtendedJTATransaction-Objekt über den zugehörigen Namespace, wie im folgenden Beispiel gezeigt wird:

```

public J2EETransactionCallback() {
    super();
    localIdToSession = new HashMap();
    String lookupName="java:comp/websphere/ExtendedJTATransaction";
    try
    {
        InitialContext ic = new InitialContext();
        jta = (ExtendedJTATransaction)ic.lookup(lookupName);
        jta.registerSynchronizationCallback(this);
    }
    catch(NotSupportedException e)
    {
        throw new RuntimeException("Cannot register jta callback", e);
    }
    catch(NamingException e){
        throw new RuntimeException("Cannot get transaction object");
    }
}

```

Für andere Produkte können Sie einen ähnlichen Ansatz verwenden, um das Transaktionserviceobjekt abzurufen.

## Festschreibung durch externen Callback steuern

Das TransactionCallback-Plug-in muss ein externes Signal empfangen, um die eXtreme-Scale-Sitzung festzuschreiben (Commit) oder rückgängig zu machen (Roll-back). Zum Empfangen dieses externen Signals verwenden Sie den Callback des externen Transaktionservice. Implementieren Sie die Schnittstelle für externe Callbacks, und registrieren Sie sie beim externen Transaktionservice. Implementieren Sie beispielsweise für WebSphere Application Server die Schnittstelle "SynchronizationCallback", wie im folgenden Beispiel gezeigt wird:

```

public class J2EETransactionCallback implements
com.ibm.websphere.objectgrid.plugins.TransactionCallback, SynchronizationCallback {
    public J2EETransactionCallback() {
        super();
        String lookupName="java:comp/websphere/ExtendedJTATransaction";
        localIdToSession = new HashMap();
        try {
            InitialContext ic = new InitialContext();
            jta = (ExtendedJTATransaction)ic.lookup(lookupName);
            jta.registerSynchronizationCallback(this);
        } catch(NotSupportedException e) {
            throw new RuntimeException("Cannot register jta callback", e);
        }
        catch(NamingException e) {
            throw new RuntimeException("Cannot get transaction object");
        }
    }

    public synchronized void afterCompletion(int localId, byte[] arg1,boolean didCommit) {
        Integer lid = new Integer(localId);
        // Session-Objekt für localId suchen
        Session session = (Session)localIdToSession.get(lid);
        if(session != null) {
            try {

```

```

        // Wenn WebSphere Application Server beim
        // Abschotten der Transaktion in der BackingMap festgeschrieben wird
// Flush wurde bereits in beforeCompletion durchgeführt
        if (didCommit) {
            session.commit();
        } else {
            // otherwise rollback
            session.rollback();
        }
    } catch (NoActiveTransactionException e) {
        // In der Theorie nicht möglich
    } catch (TransactionException e) {
        // da bereits ein Flush durchgeführt wurde, sollte dies nicht fehlschlagen
    } finally {
        // Sitzung immer aus der Zuordnungs-Map entfernen
        localIdToSession.remove(localId);
    }
}

}

public synchronized void beforeCompletion(int localId, byte[] arg1) {
    Session session = (Session)localIdToSession.get(new Integer(localId));
    if (session != null) {
        try {
            session.flush();
        } catch (TransactionException e) {
            // WebSphere Application Server definiert formal keine Methode,
            // um zu signalisieren, dass die Transaktion die Operation
            // nicht durchführen konnte
            throw new RuntimeException("Cache flush failed", e);
        }
    }
}
}
}

```

## eXtreme-Scale-APIs mit dem TransactionCallback-Plug-in verwenden

Das TransactionCallback-Plug-in inaktiviert die automatische Festschreibung in eXtreme Scale. Im Folgenden sehen Sie das normale Verwendungsmuster für eXtreme Scale:

```

Session ogSession = ...;
ObjectMap myMap = ogSession.getMap("MyMap");
ogSession.begin();
MyObject v = myMap.get("key");
v.setAttribute("newValue");
myMap.update("key", v);
ogSession.commit();

```

Wenn dieses TransactionCallback-Plug-in verwendet wird, geht eXtreme Scale davon aus, dass die Anwendung eXtreme Scale verwendet, wenn eine containerverwaltete Transaktion vorhanden ist. Das vorherige Code-Snippet ändert den folgenden Code in dieser Umgebung:

```

public void myMethod() {
    UserTransaction tx = ...;
    tx.begin();
    Session ogSession = ...;
    ObjectMap myMap = ogSession.getMap("MyMap");
    yObject v = myMap.get("key");
    v.setAttribute("newValue");
    myMap.update("key", v);
    tx.commit();
}

```

Die Methode "myMethod" gleicht einem Webanwendungsszenario. Die Anwendung verwendet die normale Schnittstelle "UserTransaction", um Transaktionen zu starten, festzuschreiben und rückgängig zu machen. eXtreme Scale wird automatisch um die Containertransaktion herum gestartet und festgeschrieben. Wenn die Methode eine EJB-Methode ist, die das Attribut TX\_REQUIRES verwendet, entfernen Sie die UserTransaction-Referenz, woraufhin die Aufrufe zum Starten und Festschreiben von Transaktionen und die Methode auf dieselbe Weise funktionieren.

ren. In diesem Fall ist der Container für das Starten und Beenden der Transaktion zuständig.

## Loader verwenden

Mit einem Loader-Plug-in von eXtreme Scale kann sich eine ObjectGrid wie ein Speichercache für Daten verhalten, die gewöhnlich in einem persistenten Speicher auf demselben System oder einem anderen System gespeichert werden. Gewöhnlich wird eine Datenbank oder ein Dateisystem als persistenter Speicher verwendet. Es kann auch eine ferne Java Virtual Machine (JVM) als Datenquelle verwendet werden, was die Erstellung Hub-basierter Caches mit ObjectGrid ermöglicht. Ein Loader enthält die Logik für das Lesen aus einem und das Schreiben in einem persistenten Speicher.

Loader (Ladeprogramme) sind BackingMap-Plug-ins, die aufgerufen werden, wenn Änderungen an der BackingMap vorgenommen werden oder wenn die BackingMap eine Datenanforderungen nicht bedienen kann (Cachefehler).

Weitere Informationen finden Sie in der Beschreibung der Caching-Konzepte im Handbuch *Produktübersicht*.

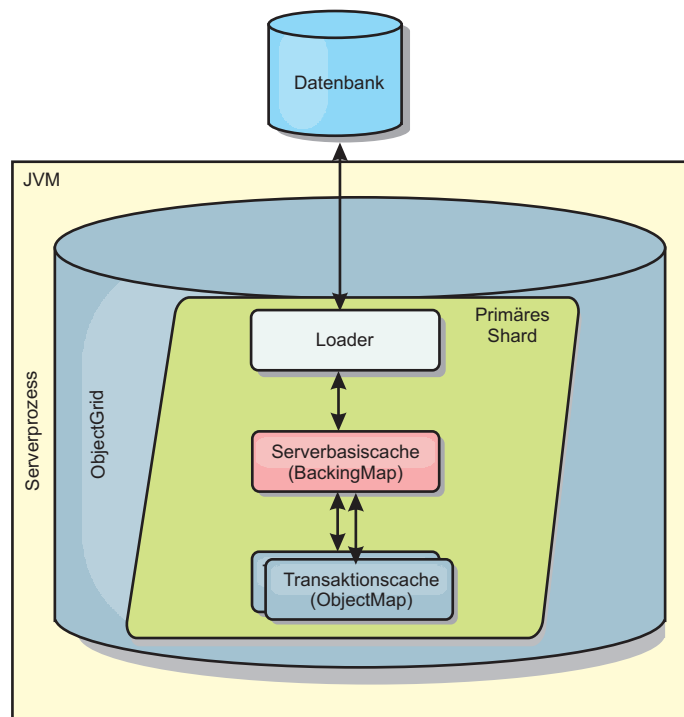


Abbildung 10. Loader

WebSphere eXtreme Scale enthält zwei integrierte Loader für die Integration relationaler Datenbank-Back-Ends. Die JPA-Loader (Java Persistence API) verwenden die ORB-Funktionen (Object-Relational Mapping, objektrelationale Zuordnung) der OpenJPA- und Hibernate-Implementierungen der JPA-Spezifikation.



## Loader verwenden

Wenn Sie der BackingMap-Konfiguration einen Loader hinzufügen möchten, können Sie die programmgesteuerte Konfiguration oder die XML-Konfiguration verwenden. Ein Loader steht mit einer BackingMap in folgender Beziehung:

- Eine BackingMap kann nur einen einzigen Loader haben.
- Eine Client-BackingMap (naher Cache) kann keinen Loader haben.
- Eine Loader-Definition kann auf mehrere BackingMaps angewendet werden, aber jede BackingMap hat eine eigene Loader-Instanz.

## Loader programmgesteuert integrieren

Das folgende Code-Snippet veranschaulicht, wie ein anwendungsdefinierter Loader in die BackingMap für map1 über die API "ObjectGrid" integriert wird:

```
import com.ibm.websphere.objectgrid.ObjectGridManagerFactory;
import com.ibm.websphere.objectgrid.ObjectGridManager;
import com.ibm.websphere.objectgrid.ObjectGrid;
import com.ibm.websphere.objectgrid.BackingMap;
ObjectGridManager ogManager = ObjectGridManagerFactory.getObjectGridManager();
ObjectGrid og = ogManager.createObjectGrid( "grid" );
BackingMap bm = og.defineMap( "map1" );
MyLoader loader = new MyLoader();
loader.setDataBaseName("testdb");
loader.setIsolationLevel("read committed");
bm.setLoader( loader );
```

In diesem Snippet wird davon ausgegangen, dass die Klasse "MyLoader" die anwendungsdefinierte Klasse ist, die die Schnittstelle "com.ibm.websphere.objectgrid.plugins.Loader" definiert. Da die Assoziation eines Loaders zu einer BackingMap nach der Initialisierung von ObjectGrid nicht mehr geändert werden kann, muss der Code vor dem Aufruf der Methode "initialize" der aufgerufenen ObjectGrid-Schnittstelle aufgerufen werden. Es wird eine Ausnahme vom Typ "IllegalStateException" in einem Aufruf der Methode "setLoader" ausgelöst, wenn diese nach der Initialisierung aufgerufen wird.

Der anwendungsdefinierte Loader kann definierte Eigenschaften haben. In dem Beispiel wird der Loader "MyLoader" verwendet, um Daten aus einer Tabelle in einer relationalen Datenbank zu lesen und Daten in diese Tabelle zu schreiben. Der Loader muss den Namen der Datenbank und die SQL-Isolationsstufe angeben. Der Loader "MyLoader" enthält die Methoden "setDataBaseName" und "setIsolationLevel", mit denen die Anwendung diese beiden Loader-Eigenschaften setzen kann.

## XML-Konfiguration für die Integration eines Loaders

Ein anwendungsdefinierter Loader kann auch über eine XML-Datei integriert werden. Das folgende Beispiel veranschaulicht, wie der Loader "MyLoader" in die BackingMap "map1" mit demselben Datenbanknamen und denselben Loader-Eigenschaften für die Isolationsstufe integriert wird:

```
<?xml version="1.0" encoding="UTF-8"?>
<objectGridConfig xmlns:xsi="http://www.w3.org/2001/XMLSchema-instance"
  xsi:schemaLocation="http://ibm.com/ws/objectgrid/config ../objectGrid.xsd"
  xmlns="http://ibm.com/ws/objectgrid/config">
<objectGrids>
  <objectGrid name="grid">
    <backingMap name="map1" pluginCollectionRef="map1" lockStrategy="OPTIMISTIC" />
  </objectGrid>
</objectGrids>
<backingMapPluginCollections>
  <backingMapPluginCollection id="map1">
    <bean id="Loader" className="com.myapplication.MyLoader">
```

```

        <property name="dataBaseName"
            type="java.lang.String"
            value="testdb"
            description="database name" />
        <property name="isolationLevel"
            type="java.lang.String"
            value="read committed"
            description="iso level" />
    </bean>
</backingMapPluginCollection>
</backingMapPluginCollections>
</objectGridConfig>

```

## Loader schreiben

Sie können eine Loader-Plug-in-Implementierung in Ihre Anwendungen einschließen. Die Loader-Implementierungsklasse muss die Schnittstelle "Loader" implementieren und die allgemeinen Konventionen für Plug-ins von WebSphere eXtreme Scale einhalten.

### Loader-Plug-in einschließen

Die Schnittstelle "Loader" hat die folgende Definition:

```

public interface Loader
{
    static final SpecialValue KEY_NOT_FOUND;
    List get(TxID txid, List keyList, boolean forUpdate) throws LoaderException;
    void batchUpdate(TxID txid, LogSequence sequence) throws LoaderException, OptimisticCollisionException;
    void preloadMap(Session session, BackingMap backingMap) throws LoaderException;
}

```

Weitere Informationen finden Sie in den Informationen zu Loadern im *Administratorhandbuch*.

### Methode "get"

Die BackingMap ruft die Methode "get" des Loaders auf, um die Werte abzurufen, die einer Schlüsselliste zugeordnet sind, die als Argument "keyList" übergeben wird. Die Methode "get" ist erforderlich, um eine Werteliste des Typs "java.lang.util.List" für jeden Schlüssel in der Schlüsselliste zurückzugeben. Der erste Wert, der in der Werteliste zurückgegeben wird, entspricht dem ersten Schlüssel in der Schlüsselliste, der zweite zurückgegebene Wert in der Werteliste dem zweiten Schlüssel in der Schlüsselliste usw. Wenn der Loader den Wert für einen Schlüssel in der Schlüsselliste nicht findet, muss der Loader das Sonderwertobjekt KEY\_NOT\_FOUND zurückgeben, das in der Schnittstelle "Loader" definiert ist. Da eine BackingMap so konfiguriert werden kann, dass sie null als gültigen Wert zulässt, ist es sehr wichtig, dass der Loader das Sonderobjekt KEY\_NOT\_FOUND zurückgibt, wenn er den Schlüssel nicht finden kann. Anhand dieses Sonderwerts kann die BackingMap zwischen einem Nullwert und einem Wert unterscheiden, der nicht vorhanden ist, weil der Schlüssel nicht gefunden wurde. Wenn eine BackingMap keine Nullwerte unterstützt, führt ein Loader, der einen Nullwert an Stelle des Objekts KEY\_NOT\_FOUND für einen nicht vorhandenen Schlüssel zurückgibt, zu einer Ausnahme.

Das Argument "forUpdate" teilt dem Loader mit, ob die Anwendung eine Methode "get" für die Map oder eine Methode "getForUpdate" für die Map aufgerufen hat. Weitere Informationen finden Sie in der Dokumentation der Schnittstelle "ObjectMap" in der API-Dokumentation. Der Loader ist für die Implementierung einer Richtlinie für die Steuerung des gemeinsamen Zugriffs zuständig, die den gleichzeitigen Zugriff auf den persistenten Speicher steuert. Viele Verwaltungssysteme für relationale Datenbanken unterstützten beispielsweise die Syntax "for update" in der SQL-Anweisung SELECT, die zum Lesen von Daten aus einer relati-

onalen Tabelle verwendet wird. Der Loader kann die Syntax "for update" in der SQL-Anweisung SELECT verwenden, wenn der boolesche Wert true als Argumentwert für den Parameter "forUpdate" dieser Methode übergeben wird. Gewöhnlich verwendet der Loader die Syntax "for update" nur, wenn eine pessimistische Richtlinie für die Steuerung des gemeinsamen Zugriffs verwendet wird. Bei einer optimistischen Steuerung des gemeinsamen Zugriffs verwendet der Loader die Syntax for update nie in der SQL-Anweisung SELECT. Der Loader muss auf der Basis der von ihm verwendeten Richtlinie für die Steuerung des gemeinsamen Zugriffs entscheiden, ob das Argument "forUpdate" verwendet wird.

Eine Erläuterung des Parameters "txid" finden Sie im Abschnitt „TransactionCallback-Plug-in“ auf Seite 146.

## Methode "batchUpdate"

Die Methode "batchUpdate" ist eine kritische Methode in der Schnittstelle "Loader". Diese Methode wird aufgerufen, wenn eXtreme Scale alle aktuellen Änderungen auf den Loader anwenden muss. Der Loader erhält eine Liste der Änderungen für die ausgewählte Map. Die Änderungen werden iteriert und auf das Back-End angewendet. Die Methode empfängt den aktuellen TxID-Wert und die anzuwendenden Änderungen. Der folgende Beispielcode iteriert durch die Gruppe der Änderungen und führt drei JDBC-Anweisungen (Java Database Connectivity) im Stapelbetrieb aus (eine mit "insert", eine weitere mit "update" und eine weitere mit "delete").

```
import java.util.Collection;
import java.util.Map;
import java.sql.PreparedStatement;
import java.sql.SQLException;
import com.ibm.websphere.objectgrid.TxID;
import com.ibm.websphere.objectgrid.plugins.Loader;
import com.ibm.websphere.objectgrid.plugins.LoaderException;
import com.ibm.websphere.objectgrid.plugins.LogElement;
import com.ibm.websphere.objectgrid.plugins.LogSequence;

public void batchUpdate(TxID tx, LogSequence sequence) throws LoaderException {
    // Zu verwendende SQL-Verbindung abrufen.
    Connection conn = getConnection(tx);
    try {
        // Liste der Änderungen verarbeiten und eine Gruppe vorbereiteter
        // Anweisungen für die Ausführung in einer SQL-Operation update, insert oder delete
        // im Stapelbetrieb erstellen.
        Iterator iter = sequence.getPendingChanges();
        while (iter.hasNext()) {
            LogElement logElement = (LogElement) iter.next();
            Object key = logElement.getKey();
            Object value = logElement.getCurrentValue();
            switch (logElement.getType().getCode()) {
                case LogElement.CODE_INSERT:
                    buildBatchSQLInsert(tx, key, value, conn);
                    break;
                case LogElement.CODE_UPDATE:
                    buildBatchSQLUpdate(tx, key, value, conn);
                    break;
                case LogElement.CODE_DELETE:
                    buildBatchSQLDelete(tx, key, conn);
                    break;
            }
        }
        // Die Stapelanweisungen ausführen, die mit der vorherigen Schleife erstellt wurden.
        Collection statements = getPreparedStatementCollection(tx, conn);
        iter = statements.iterator();
        while (iter.hasNext()) {
            PreparedStatement pstmt = (PreparedStatement) iter.next();
            pstmt.executeBatch();
        }
    } catch (SQLException e) {
        LoaderException ex = new LoaderException(e);
        throw ex;
    }
}
```

Das vorherige Beispiel veranschaulicht die übergeordnete Logik für die Verarbeitung des Arguments "LogSequence", aber die Details der Erstellung einer SQL-An-

weisung "insert", "update" und "delete" werden nicht gezeigt. Im Folgenden sind einige Schlüsselaspekte aufgeführt, die veranschaulicht werden:

- Die Methode "getPendingChanges" wird für das Argument "LogSequence" aufgerufen, um einen Iterator für die Liste der LogElement-Objekte abzurufen, die der Loader verarbeiten muss.
- Die Methode "LogElement.getType().getCode()" wird verwendet, um festzustellen, ob das LogElement-Objekt für eine SQL-Anweisung "insert", "update" oder "delete" bestimmt ist.
- Es wird eine Ausnahme des Typs "SQLException" abgefangen und mit einer Ausnahme des Typs "LoaderException" verkettet, die ausgegeben wird, um zu berichten, dass während der Aktualisierung im Stapelbetrieb eine Ausnahme eingetreten ist.
- Die JDBC-Unterstützung für Aktualisierungen im Stapelbetrieb wird verwendet, um die Anzahl der erforderlichen Abfragen an das Back-End zu minimieren.

## Methoden "preloadMap"

Während der Initialisierung von eXtreme Scale wird jede definierte BackingMap initialisiert. Wenn ein Loader in eine BackingMap integriert ist, ruft die BackingMap die Methode "preloadMap" in der Schnittstelle "Loader" auf, damit der Loader Daten vorab aus dem zugehörigen Back-End abrufen und in die Map laden kann. Im folgenden Beispiel wird angenommen, dass die ersten 100 Zeilen einer Tabelle "Employee" aus der Datenbank gelesen und in die Map geladen werden. Die Klasse "EmployeeRecord" ist eine anwendungsdefinierte Klasse, die die Mitarbeiterdaten enthält, die aus der Tabelle "employee" gelesen werden.

```
import java.sql.PreparedStatement;
import java.sql.SQLException;
import com.ibm.websphere.objectgrid.Session;
import com.ibm.websphere.objectgrid.TxID;
import com.ibm.websphere.objectgrid.plugins.Loader;
import com.ibm.websphere.objectgrid.plugins.LoaderException;

public void preloadMap(Session session, BackingMap backingMap) throws LoaderException {
    boolean tranActive = false;
    ResultSet results = null;
    Statement stmt = null;
    Connection conn = null;
    try {
        session.beginNoWriteThrough();
        tranActive = true;
        ObjectMap map = session.getMap(backingMap.getName());
        TxID tx = session.getTxID();
        // Verbindung mit automatischem Festschreiben abrufen, die
        // auf die Isolationsstufe "Lesen mit COMMIT" gesetzt ist.
        conn = getAutoCommitConnection(tx);
        // EmployeeRecord-Objekte vorher in die Map "Employee"
        // laden. Alle Employee-Datensätze aus der Tabelle lesen,
        // aber das vorherige Laden auf die ersten 100 Zeilen beschränken.
        stmt = conn.createStatement();
        results = stmt.executeQuery(SELECT_ALL);
        int rows = 0;
        while (results.next() && rows < 100) {
            int key = results.getInt(EMPNO_INDEX);
            EmployeeRecord emp = new EmployeeRecord(key);
            emp.setLastName(results.getString(LASTNAME_INDEX));
            emp.setFirstName(results.getString(FIRSTNAME_INDEX));
            emp.setDepartmentName(results.getString(DEPTNAME_INDEX));
            emp.updateSequenceNumber(results.getLong(SEQNO_INDEX));
            emp.setManagerNumber(results.getInt(MGRNO_INDEX));
            map.put(new Integer(key), emp);
            ++rows;
        }
        // Transaktion festschreiben.
        session.commit();
        tranActive = false;
    } catch (Throwable t) {
        throw new LoaderException("preload failure: " + t, t);
    } finally {
        if (tranActive) {
            try {
                session.rollback();
            } catch (Throwable t2) {
                // Rollback-Fehler tolerieren und Auslösung
            }
        }
    }
}
```

```

        // des ursprünglichen Elements der Throwable-Klasse zulassen.
    }
    }
    // Sicherstellen, dass hier auch andere Datenbankressourcen
    // bereinigt werden, z. B. Anweisungen, Ergebnismengen usw. schließen.
}
}

```

Dieses Beispiel veranschaulicht die folgenden Schlüsselaspekte:

- Die BackingMap "preloadMap" verwendet das Session-Objekt, das als Sitzungsargument an sie übergeben wurde.
- Die Methode "Session.beginNoWriteThrough" wird an Stelle der Methode "begin" verwendet, um die Transaktion zu starten.
- Der Loader kann nicht für jede Operation "put" aufgerufen werden, die in dieser Methode zum Laden der Map ausgeführt wird.
- Der Loader kann Spalten der Tabelle "Employee" einem Feld im Java-Objekt "EmployeeRecord" zuordnen. Der Loader fängt alle Throwable-Ausnahmen ab, die eintreten, und löst eine Ausnahme des Typs "LoaderException" aus, die mit der abgefangenen Throwable-Ausnahme verkettet wird.
- Der Block "finally" stellt sicher, dass alle Throwable-Ausnahmen, die in der Zeit zwischen dem Aufruf der Methode "beginNoWriteThrough" und dem Aufruf der Methode "commit" eintreten, dazu führen, dass der Block "finally" eine Roll-back-Operation für die aktive Transaktion durchführt. Diese Aktion ist kritisch, um sicherzustellen, dass jede von der Methode "preloadMap" gestartete Transaktion abgeschlossen ist, bevor sie zum Aufrufenden zurückkehrt. Der Block "finally" eignet sich bestens für die Ausführung weiterer Bereinigungsaktionen, die erforderlich sein könnten, wie z. B. das Schließen der JDBC-Verbindung und anderer JDBC-Objekte.

Im Beispiel "preloadMap" wird eine SQL-Anweisung SELECT verwendet, die alle Zeilen der Tabelle auswählt. In Ihrem anwendungsdefinierten Loader müssen Sie möglicherweise eine oder mehrere Loader-Eigenschaften definieren, um zu steuern, wie viele Zeilen der Tabelle vorher in die Map geladen werden müssen.

Da die Methode "preloadMap" nur einmal während der BackingMap-Initialisierung aufgerufen wird, eignet sich auch bestens für die einmalige Ausführung des Initialisierungscode für den Loader. Selbst wenn ein Loader beschließt, Daten nicht vorab aus dem Back-End abzurufen und die Map zu laden, muss er wahrscheinlich irgendeine andere einmalige Initialisierung durchführen, um andere Methoden des Loaders effizienter zu machen. Das folgende Beispiel veranschaulicht das Caching des TransactionCallback-Objekts und des OptimisticCallback-Objekts als Instanzvariablen des Loaders, so dass die anderen Methoden des Loaders keine Methodenaufrufe absetzen müssen, um Zugriff auf diese Objekte zu erhalten. Dieses Caching der ObjectGrid-Plug-in-Werte kann durchgeführt werden, weil die TransactionCallback- und OptimisticCallback-Objekte nach der Initialisierung der BackingMap nicht mehr geändert oder ersetzt werden können. Es ist zulässig, diese Objektreferenzen als Instanzvariablen des Loaders zwischenspeichern.

```

import com.ibm.websphere.objectgrid.Session;
import com.ibm.websphere.objectgrid.BackingMap;
import com.ibm.websphere.objectgrid.plugins.OptimisticCallback;
import com.ibm.websphere.objectgrid.plugins.TransactionCallback;

// Instanzvariablen des Loaders
MyTransactionCallback ivTcb; // MyTransactionCallback

// Erweitert TransactionCallback
MyOptimisticCallback ivOcb; // MyOptimisticCallback

// Implementiert OptimisticCallback
// ...
public void preloadMap(Session session, BackingMap backingMap) throws LoaderException
[Replication programming]
// TransactionCallback- und OptimisticCallback-Objekte

```

```

// in Instanzvariablen dieses Loaders zwischenspeichern
ivTcb = (MyTransactionCallback) session.getObjectGrid().getTransactionCallback();
ivOcb = (MyOptimisticCallback) backingMap.getOptimisticCallback();
// Restlicher preloadMap-Code (z. B. wie im vorherigen Beispiel)
}

```

Informationen zum vorherigen Laden und wiederherstellbaren vorherigen Laden beim Replikations-Failover finden Sie in der Dokumentation zur Replikation im Handbuch *Produktübersicht*.

## Loader mit Entitäts-Maps

Wenn der Loader in eine Entitäts-Map integriert ist, muss der Loader mit Tupelobjekten arbeiten. Tupelobjekte sind ein spezielles Format von Entitätsdaten. Der Loader muss eine Konvertierung zwischen dem Tupelformat und anderen Datenformaten durchführen. Die Methode "get" gibt beispielsweise eine Liste mit Werten zurück, die der Gruppe von Schlüssel entspricht, die an die Methode übergeben werden. Die übergebenen Schlüssel haben den Typ "Tupel", d. h., sie sind Schlüssel-tupel. Angenommen, der Loader definiert die Map über JDBC in einer Datenbank als persistent. In diesem Fall muss die Methode "get" jedes Schlüssel-tupel in eine Liste von Attributwerten konvertieren, die den Primärschlüsselspalten der Tabelle entsprechen, die der Entitäts-Map zugeordnet ist, die Anweisung SELECT mit der WHERE-Klausel ausführen, die konvertierte Attributwerte als Kriterien für den Abruf von Daten aus der Datenbank verwendet, und anschließend die zurückgegebenen Daten in Werttupel konvertieren. Die Methode "get" ruft Daten aus der Datenbank ab, konvertiert die Daten in Werttupel für die übergebenen Schlüssel-tupel und gibt dann eine Liste mit Werttupeln zurück, die den Schlüssel-tupeln entsprechen, die an den Aufrufenden übergeben werden. Die Methode "get" kann eine einzige Anweisung SELECT ausführen, um alle Daten gleichzeitig abzurufen, oder sie kann für jedes Schlüssel-tupel eine eigene Anweisung SELECT ausführen. Ausführliche Informationen zur Programmierung, die zeigen, wie der Loader verwendet wird, wenn die Daten über einen EntityManager gespeichert werden, finden Sie im Abschnitt „Loader mit Entitäts-Maps und Tupeln verwenden“ auf Seite 166.

## Hinweise zur Programmierung von JPA-Loadern

Ein JPA-Loader (Java Persistence API (JPA)) ist eine Ladeprogramm-Plug-in-Implementierung, die JPA für die Interaktion mit der Datenbank verwendet. Verwenden Sie die folgenden Hinweise, wenn Sie eine Anwendung entwickeln, die einen JPA-Loader verwendet.

### eXtreme-Scale-Entität und JPA-Entität

Sie können eine beliebige POJO-Klasse über eXtreme-Scale-Annotationen und/oder XML-Konfiguration als eXtreme-Scale-Entität festlegen. Dieselbe POJO-Klasse kann über JPA-Entitätsannotationen und/oder XML-Konfiguration auch als JPA-Entität festgelegt werden.

**eXtreme-Scale-Entität:** Eine eXtreme-Scale-Entität stellt persistente Daten dar, die in ObjectGrid-Maps gespeichert werden. Ein Entitätsobjekt wird in ein Schlüssel-tupel und ein Werttupel umgesetzt, die dann als Schlüssel/Wert-Paar in den Maps gespeichert werden. Ein Tupel ist ein Bereich primitiver Attribute.

**JPA-Entität:** Eine JPA-Entität stellt persistente Daten dar, die über Container-managed Persistence (CMP) automatisch in einer relationalen Datenbank gespeichert werden. Die Daten werden in Form eines Datenspeichersystems entsprechenden Formats persistent gespeichert, z. B. als Datenbanktupel in einer Datenbank.

Wenn eine eXtreme-Scale-Entität persistent gespeichert wird, werden ihre Relationen in anderen Entitäts-Maps gespeichert. Wenn Sie beispielsweise eine Entität "Consumer" mit einer 1:n-Relation zu einer Entität "ShippingAddress" persistent speichern und "cascade-persist" aktiviert ist, wird die Entität "ShippingAddress" in der Map "shippingAddress" im Tupelformat gespeichert. Wenn Sie eine JPA-Entität persistent speichern, werden auch die zugehörigen JPA-Entitäten persistent in Datenbanktabellen gespeichert, wenn "cascade-persist" aktiviert ist. Wenn eine POJO-Klasse sowohl als eXtreme-Scale-Entität als auch als JPA-Entität definiert ist, können die Daten in ObjectGrid-Entitäts-Maps und in Datenbanken persistent gespeichert werden. Im Folgenden sind verschiedene gängige Anwendungsfälle beschrieben:

- **Preload-Szenario:** Eine Entität wird aus einer Datenbank über einen JPA-Provider geladen und in ObjectGrid-Entitäts-Maps persistent gespeichert.
- **Loader-Szenario:** Eine Loader-Implementierung wird für die ObjectGrid-Entitäts-Maps integriert, so dass eine in ObjectGrid-Entitäts-Maps gespeicherte Entität über JPA-Provider in einer Datenbank persistent gespeichert oder aus einer Datenbank geladen werden kann.

Es ist auch gängig, dass eine POJO-Klasse nur als JPA-Entität definiert wird. In diesem Fall werden POJO-Instanzen in den ObjectGrid-Maps gespeichert und keine Entitätstupel, wie es bei einer ObjectGrid-Entität der Fall ist.

## Hinweise zum Anwendungsdesign für Entitäts-Maps

Wenn Sie eine JPALoader-Schnittstelle integrieren, werden die Objektinstanzen direkt in den ObjectGrid-Maps gespeichert.

Wenn Sie jedoch ein JPAEntityLoader-Plug-in integrieren, wird die Entitätsklasse sowohl als eXtreme-Scale-Entität als auch als JPA-Entität definiert. In diesem Fall behandeln Sie diese Entität so, als hätte sie zwei persistente Speicher: die ObjectGrid-Entitäts-Maps und den JPA-Persistenzspeicher. Die Architektur wird komplexer als beim JPALoader-Plug-in.

Weitere Informationen zum JPAEntityLoader-Plug-in und zum Anwendungsdesign finden Sie in der Beschreibung des JPAEntityLoader-Plug-ins im *Administratorhandbuch*. Diese Informationen können auch helfen, wenn Sie planen, einen eigenen Loader für die Entitäts-Maps zu implementieren.

## Leistungsaspekte

Stellen Sie sicher, dass Sie den richtigen Abruftyp (eager oder lazy) für Beziehungen festlegen. Die Leistungsunterschiede lassen sich beispielsweise an einer bidirektionalen 1:n-Beziehung zwischen "Consumer" und "ShippingAddress" mit OpenJPA besser erklären. In diesem Beispiel versucht eine JPA-Abfrage mit `select o from Consumer o where . . .` einen Masseneinlesevorgang durchzuführen und auch alle zugehörigen ShippingAddress-Objekte zu laden. Im Folgenden sehen Sie eine 1:n-Beziehung, die in der Klasse "Consumer" definiert ist:

```
@Entity
public class Consumer implements Serializable {

    @OneToMany(mappedBy="consumer", cascade=CascadeType.ALL, fetch = FetchType.EAGER)
    ArrayList <ShippingAddress> addresses;
```

Im Folgenden sehen Sie eine n:1-Beziehung, die in der Klasse "ShippingAddress" definiert ist:

```

@Entity
public class ShippingAddress implements Serializable{

    @ManyToOne(fetch=FetchType.EAGER)
    Consumer consumer;
}

```

Wenn die Abruftypen beider Beziehungen mit eager definiert sind, verwendet OpenJPA N+1+1-Abfragen, um alle Consumer-Objekte und ShippingAddress-Objekte abzurufen, wobei N für die Anzahl der ShippingAddress-Objekte steht. Wird das ShippingAddress-Objekt jedoch geändert, so dass, wie im folgenden Beispiel gezeigt, der Abruftyp "lazy" verwendet wird, werden nur zwei Abfragen zum Abrufen aller Daten benötigt:

```

@Entity
public class ShippingAddress implements Serializable{

    @ManyToOne(fetch=FetchType.LAZY)
    Consumer consumer;
}

```

Obwohl die Abfrage in beiden Fällen dieselben Ergebnisse zurückgibt, nimmt mit einer geringeren Anzahl an Abfragen die Interaktion mit der Datenbank erheblich ab, was die Anwendungsleistung steigern kann.

## JPAEntityLoader-Plug-in

Das JPAEntityLoader-Plug-in ist eine integrierte Loader-Implementierung, die Java Persistence API (JPA) verwendet, um mit der Datenbank zu kommunizieren, wenn Sie die API "EntityManager" verwenden. Wenn Sie die API "ObjectMap" verwenden, verwenden Sie den Loader "JPALoader".

### Details zum Loader

Verwenden Sie das JPALoader-Plug-in, wenn Sie Daten mit der API "ObjectMap" speichern. Verwenden Sie das JPAEntityLoader-Plug-in, wenn Sie Daten mit der API "EntityManager" speichern.

Loader (Ladeprogramme) stellen zwei Hauptmethoden bereit:

1. **get:** In der Methode "get" ruft das JPAEntityLoader-Plug-in zunächst die Methode "javax.persistence.EntityManager.find(Class entityClass, Object key)" auf, um die JPA-Entität zu suchen. Anschließend projiziert das Plug-in diese JPA-Entität auf Entitätstupel. Während der Projektion werden die Tupelattribute und die Assoziationsschlüssel im Werttupel gespeichert. Nach der Verarbeitung der einzelnen Schlüssel gibt die Methode "get" eine Liste mit Entitätswerttupel zurück.
2. **batchUpdate:** Die Methode "batchUpdate" verwendet ein LogSequence-Objekt, das eine Liste mit LogElement-Objekten enthält. Jedes LogElement-Objekt enthält ein Schlüsseltupel und ein Werttupel. Für die Interaktion mit dem JPA-Provider muss zunächst auf der Basis des Schlüsseltupels die eXtreme-Scale-Entität gesucht werden. Basierend auf dem LogElement-Typ werden die folgenden JPA-Aufrufe ausgeführt:
  - **insert:** javax.persistence.EntityManager.persist(Object o)
  - **update:** javax.persistence.EntityManager.merge(Object o)
  - **remove:** javax.persistence.EntityManager.remove(Object o)

Ein LogElement-Objekt mit dem Typ **update** veranlasst JPAEntityLoader, die Methode "javax.persistence.EntityManager.merge(Object o)" aufzurufen, um die



Entität aufzunehmen. Ein LogElement-Objekt des Typs **update** kann das Ergebnis eines Aufrufs der Methode "com.ibm.websphere.objectgrid.em.EntityManager.merge(object o)" oder einer Attributänderung in der von der eXtreme-Scale-API "EntityManager" verwalteten Instanz sein. Sehen Sie sich dazu das folgende Beispiel an:

```
com.ibm.websphere.objectgrid.em.EntityManager em = og.getSession().getEntityManager();
em.getTransaction().begin();
Consumer c1 = (Consumer) em.find(Consumer.class, c.getConsumerId());
c1.setName("New Name");
em.getTransaction().commit();
```

In diesem Beispiel wird ein LogElement-Objekt des Typs "update" an den JPAEntityLoader des Map-Konsumenten gesendet. Die Methode "javax.persistence.EntityManager.merge(Object o)" im JPA-EntityManager wird an Stelle einer Attributaktualisierung in der von JPA verwalteten Entität aufgerufen. Aufgrund dieses geänderten Verhaltens sind eine Einschränkungen bei der Verwendung dieses Programmiermodells zu beachten.

## Regeln für das Anwendungsdesign

Entitäten haben Beziehungen mit anderen Entitäten. Beim Design einer Anwendung mit Beziehungen und mit dem JPAEntityLoader-Plug-in müssen weitere Aspekte berücksichtigt werden. Die Anwendung muss vier Regeln einhalten, die im Folgenden beschrieben werden.

### Eingeschränkte Unterstützung für die Beziehungstiefe

JPAEntityLoader wird nur unterstützt, wenn Entitäten ohne Beziehungen bzw. Beziehungen mit einer einzigen Beziehungsebene verwendet werden. Beziehungen mit mehreren Ebenen, wie z. B. Company > Department > Employee, werden nicht unterstützt.

### Ein einziges Ladeprogramm pro Map

Angenommen Sie haben eine Entitätsbeziehung "Consumer-ShippingAddress". Wenn Sie einen Konsumenten laden, der sehr viele Abrufe durchführt, können Sie alle zugehörigen ShippingAddress-Objekte laden. Wenn Sie ein Customer-Objekt als persistent definieren oder aufnehmen, können Sie die zugehörigen ShippingAddress-Objekte als persistent definieren oder aufnehmen, wenn "cascade-persist" oder "cascade-merge" aktiviert ist.

Sie können kein Ladeprogramm für die Stammentitäts-Map integrieren, in der die Consumer-Entitätstupel gespeichert werden. Sie müssen für jede einzelne Entitäts-Map einen Loader konfigurieren.

### Derselbe Kaskadierungstyp für JPA und eXtreme Scale

Stellen Sie sich wieder den Fall vor, dass der Entitätskonsument (Consumer) eine Eins-zu-viele-Beziehung zu ShippingAddress hat. Angenommen, für diese Beziehung ist "cascade-persist" aktiviert. Wenn ein Consumer-Objekt in eXtreme Scale als persistent definiert wird, werden die zugehörigen N ShippingAddress-Objekte in eXtreme Scale ebenfalls als persistent definiert.

Ein Aufruf der Methode "persist" für das Consumer-Objekt mit einer Beziehung vom Typ "cascade-persist" zu ShippingAddress von der JPAEntityLoader-Schicht in einen Aufruf der Methode "javax.persistence.EntityManager.persist(consumer)" und N Aufrufe der Methode "javax.persistence.EntityManager.persist(shippingAddress)"

übersetzt. Diese N zusätzlichen Aufrufe von "persist" an die ShippingAddress-Objekte sind jedoch wegen der Einstellung "cascade-persist" aus der Sicht des JPA-Providers nicht erforderlich. Zur Lösung dieses Problems stellt eXtreme Scale eine neue Methode "isCascaded" in der Schnittstelle "LogElement" bereit. Die Methode "isCascaded" gibt an, ob das LogElement-Objekt das Ergebnis einer Operation "cascade" des EntityManager von eXtreme Scale ist. In diesem Beispiel empfängt der JPAEntityLoader der Map "ShippingAddress" wegen der cascade-persist-Aufrufe N LogElement-Objekte. Der JPAEntityLoader stellt fest, dass die Methode "isCascaded" den Wert "true" zurückgibt und ignoriert sie daraufhin, ohne JPA-Aufrufe abzusetzen. Deshalb wird aus der JPA-Sicht nur ein einziger Aufruf der Methode "javax.persistence.EntityManager.persist(consumer)" abgerufen.

Dasselbe Verhalten zeigt sich, wenn Sie eine Entität aufnehmen oder eine Entität entfernen, wenn die Kaskadierung aktiviert ist. Die kaskadierten Operationen werden vom JPAEntityLoader-Plug-in ignoriert.

Die Kaskadierungsunterstützung ist so konzipiert, dass die Operationen des EntityManager von eXtreme Scale an die JPA-Provider wiederholt werden. Zu diesen Operationen gehören die Operationen "persist", "merge" und "remove". Damit dies funktioniert, müssen Sie sicherstellen, dass die Kaskadierungseinstellungen von JPA und EntityManager von eXtreme Scale identisch sind.

## Entitätsaktualisierung mit Vorsicht verwenden

Wie bereits beschrieben, ist die Kaskadierungsunterstützung so konzipiert, dass die Operationen des EntityManager von eXtreme Scale an die JPA-Provider wiederholt werden. Wenn Ihre Anwendung die Methode "ogEM.persist(consumer)" im EntityManager von eXtreme Scale aufruft, werden selbst die zugeordneten ShippingAddress-Objekte aufgrund der Einstellung "cascade-persist" persistent gespeichert, und der JPAEntityLoader ruft nur die Methode "jpAEM.persist(consumer)" in den JPA-Providern auf.

Wenn Ihre Anwendung jedoch eine verwaltete Entität aktualisiert, wird diese Aktualisierung vom JPAEntityLoader-Plug-in in einen JPA-Aufruf "merge" übersetzt. In diesem Szenario ist die Unterstützung für mehrere Beziehungsebenen und Schlüsselzuordnungen nicht gewährleistet. In diesem Fall bewährt es sich, die Methode "javax.persistence.EntityManager.merge(o)" zu verwenden, anstatt eine verwaltete Entität zu aktualisieren.

## Loader mit Entitäts-Maps und Tupeln verwenden

Der EntityManager konvertiert alle Entitätsobjekte in Tupelobjekte, bevor sie in einer eXtreme-Scale-Map gespeichert werden. Für jede Entität gibt es ein Schlüssel-tupel und ein Werttupel. Dieses Schlüssel/Wert-Paar wird in der eXtreme-Scale-Map gespeichert, die der Entität zugeordnet ist. Wenn Sie eine eXtreme-Scale-Map mit einem Loader verwenden, muss der Loader mit den Tupelobjekten interagieren.

## Übersicht

eXtreme Scale enthält Loader-Plug-ins, die die Integration mit relationalen Datenbanken vereinfachen. Die JPA-Loader (Java Persistence API) verwenden eine Java Persistence API, um mit der Datenbank zu interagieren und die Entitätsobjekte zu erstellen. Die JPA-Loader sind mit eXtreme-Scale-Entitäten kompatibel.

## Tupel

Ein Tupel enthält Informationen zu den Attributen und Assoziationen einer Entität. Primitive Werte werden über ihre primitiven Wrapper gespeichert. Andere unterstützte Objekttypen werden in ihrem nativen Format gespeichert. Assoziationen zu anderen Entitäten werden als Sammlung von Schlüsseltupelobjekten gespeichert, die die Schlüssel der Zielentitäten darstellen.

Jedes Attribut und jede Assoziation wird über einen nullbasierten Index gespeichert. Sie können den Index jedes Attributs mit der Methode "getAttributePosition" oder "getAssociationPosition" abrufen. Nachdem Sie die Position abgerufen haben, bleibt diese für die Dauer des Lebenszyklus von eXtreme Scale unverändert. Die Position kann sich ändern, wenn eXtreme Scale erneut gestartet wird. Die Methoden "setAttribute", "setAssociation" und "setAssociations" werden verwendet, um die Elemente im Tupel zu aktualisieren.

**Achtung:** Wenn Sie Tupelobjekte erstellen oder aktualisieren, müssen Sie jedes primitive Feld mit einem Wert ungleich null aktualisieren. Primitive Werte wie int dürfen nicht null sein. Wenn sie den Wert nicht in einen Standardwert ändern, treten Leistungsprobleme auf, die sich auch auf Felder auswirken, die mit der Annotation "@Version" oder einem Versionsattribut in der XML-Entitätsdeskriptordatei markiert sind.

Im folgenden Beispiel wird ausführlicher erläutert, wie Tupel verarbeitet werden. Weitere Informationen zum Definieren von Entitäten für dieses Beispiel finden Sie in der Beschreibung des Schemas der Entität "Order" im Lernprogramm zur Schnittstelle "EntityManager" im Handbuch *Produktübersicht*. WebSphere eXtreme Scale ist so konfiguriert, dass für jede Entität ein Loader verwendet wird. Außerdem wird nur die Entität "Order" verwendet, und diese Entität hat eine Viele-zu-eins-Beziehung zur Entität "Customer". Der Attributname ist customer, und dieses Attribut hat eine Eins-zu-viele-Beziehung zur Entität "OrderLine".

Verwenden Sie den Projektor, um Tupelobjekte automatisch aus Entitäten zu erstellen. Die Verwendung des Projektors kann Loader vereinfachen, wenn ein ORM-Dienstprogramm (Object-Relational Mapping, objektbezogene Zuordnung) wie Hibernate oder JPA verwendet wird.

### order.java

```
@Entity
public class Order
{
    @Id String orderNumber;
    java.util.Date date;
    @OneToOne(cascade=CascadeType.PERSIST) Customer customer;
    @OneToMany(cascade=CascadeType.ALL, mappedBy="order") @OrderBy("lineNumber") List<OrderLine> lines;
}
```

### customer.java

```
@Entity
public class Customer {
    @Id String id;
    String firstName;
    String surname;
    String address;
    String phoneNumber;
}
```

## orderLine.java

```
@Entity
public class OrderLine
{
    @Id @ManyToOne(cascade=CascadeType.PERSIST) Order order;
    @Id int lineNumber;
    @OneToOne(cascade=CascadeType.PERSIST) Item item;
    int quantity;
    double price;
}
```

Eine Klasse "OrderLoader", die die Schnittstelle "Loader" implementiert, wird im folgenden Code gezeigt. Im folgenden Beispiel wird angenommen, dass ein zugehöriges TransactionCallback-Plug-in definiert ist.

## orderLoader.java

```
public class OrderLoader implements com.ibm.websphere.objectgrid.plugins.Loader {

    private EntityMetadata entityMetadata;
    public void batchUpdate(TxID txid, LogSequence sequence)
        throws LoaderException, OptimisticCollisionException {
        ...
    }
    public List get(TxID txid, List keyList, boolean forUpdate)
        throws LoaderException {
        ...
    }
    public void preloadMap(Session session, BackingMap backingMap)
        throws LoaderException {
        this.entityMetadata=backingMap.getEntityMetadata();
    }
}
```

Die Instanzvariable "entityMetadata" wird während des Aufrufs der Methode "preloadMap" über eXtreme Scale initialisiert. Die Variable *entityMetadata* ist nicht null, wenn die Map für die Verwendung von Entitäten konfiguriert ist. Andernfalls ist der Wert null.

## Methode "batchUpdate"

Mit der Methode "batchUpdate" kann festgestellt werden, welche Aktion die Anwendung ausführen wollte. Auf der Basis einer Operation "insert", "update" oder "delete" kann eine Verbindung zur Datenbank geöffnet und die Arbeit ausgeführt werden. Da der Schlüssel und die Werte vom Typ "Tupel" sind, müssen sie umgesetzt werden, so dass sie in der SQL-Anweisung Sinn machen.

Die Tabelle ORDER wurde mit der DDL-Definition (Data Definition Language) erstellt, die Sie im folgenden Code sehen:

```
CREATE TABLE ORDER (ORDERNUMBER VARCHAR(250) NOT NULL, DATE TIMESTAMP, CUSTOMER_ID VARCHAR(250))
ALTER TABLE ORDER ADD CONSTRAINT PK_ORDER PRIMARY KEY (ORDERNUMBER)
```

Der folgende Code veranschaulicht wie Sie ein Tupel in ein Objekt konvertieren:

```
public void batchUpdate(TxID txid, LogSequence sequence)
    throws LoaderException, OptimisticCollisionException {
    Iterator iter = sequence.getPendingChanges();
    while (iter.hasNext()) {
        LogElement logElement = (LogElement) iter.next();
        Object key = logElement.getKey();
        Object value = logElement.getCurrentValue();

        switch (logElement.getType().getCode()) {
        case LogElement.CODE_INSERT:

1)            if (entityMetadata!=null) {

// Der Auftrag (order) hat nur einen einzigen Schlüssel, die Auftragsnummer (orderNumber).
2)                String ORDERNUMBER=(String) getKeyAttribute("orderNumber", (Tuple) key);
// Wert von date abrufen.
3)                java.util.Date unFormattedDate = (java.util.Date) getValueAttribute("date", (Tuple) value);
```

```

// Die Werte sind 2 Assoziationen. Kunden verarbeiten, weil die Tabelle
// customer.id als Primärschlüssel enthält.
4)      Object[] keys= getForeignKeyForValueAssociation("customer","id",(Tuple) value);
        //Beziehung zwischen Order und Customer ist M zu 1. Es kann nur einen einzigen Schlüssel geben.
5)      String CUSTOMER_ID=(String)keys[0];
// Variable unformattedDate syntaktisch analysieren und für die Datenbank als formattedDate formatieren
6)      String formattedDate = "2007-05-08-14.01.59.780272"; // formatted for DB2
// Abschließend die folgende SQL-Anweisung, um den Datensatz einzufügen
7) //INSERT INTO ORDER (ORDERNUMBER, DATE, CUSTOMER_ID) VALUES(ORDERNUMBER,formattedDate, CUSTOMER_ID)
        }
        break;
    case LogElement.CODE_UPDATE:
        break;
    case LogElement.CODE_DELETE:
        break;
    }
}

// Gibt den Wert für das Attribut zurück, der im Schlüsseltuple gespeichert ist
private Object getKeyAttribute(String attr, Tuple key) {
    // Metadaten des Schlüssels abrufen
    TupleMetadata keyMD = entityMetaData.getKeyMetadata();
    // Position des Attributs abrufen
    int keyAt = keyMD.getAttributePosition(attr);
    if (keyAt > -1) {
        return key.getAttribute(keyAt);
    } else { // attribute undefined
        throw new IllegalArgumentException("Invalid position index for "+attr);
    }
}

// Gibt den Wert für das Attribut zurück, der im Werttuple gespeichert ist
private Object getValueAttribute(String attr, Tuple value) {
    // Ähnlich wie oben, abgesehen davon, dass mit den Metadaten des Werts gearbeitet wird
    TupleMetadata valueMD = entityMetaData.getValueMetadata();

    int keyAt = valueMD.getAttributePosition(attr);
    if (keyAt > -1) {
        return value.getAttribute(keyAt);
    } else {
        throw new IllegalArgumentException("Invalid position index for "+attr);
    }
}

// Gibt einen Bereich von Schlüsseln zurück, die auf die Assoziation verweisen
private Object[] getForeignKeyForValueAssociation(String attr, String fk_attr, Tuple value) {
    TupleMetadata valueMD = entityMetaData.getValueMetadata();
    Object[] ro;

    int customerAssociation = valueMD.getAssociationPosition(attr);
    TupleAssociation tupleAssociation = valueMD.getAssociation(customerAssociation);

    EntityMetadata targetEntityMetaData = tupleAssociation.getTargetEntityMetadata();

    Tuple[] customerKeyTuple = ((Tuple) value).getAssociations(customerAssociation);

    int numberOfKeys = customerKeyTuple.length;
    ro = new Object[numberOfKeys];

    TupleMetadata keyMD = targetEntityMetaData.getKeyMetadata();
    int keyAt = keyMD.getAttributePosition(fk_attr);
    if (keyAt < 0) {
        throw new IllegalArgumentException("Invalid position index for " + attr);
    }
    for (int i = 0; i < numberOfKeys; ++i) {
        ro[i] = customerKeyTuple[i].getAttribute(keyAt);
    }

    return ro;
}
}

```

1. Stellen Sie sicher, dass entityMetaData nicht null ist, was impliziert, dass die Cacheinträge für Schlüssel und Wert Tuple sind. Aus den entityMetaData werden nur die TupleMetadata für den Schlüssel abgerufen, die wirklich nur den Schlüsselteil der Auftragsmetadaten enthalten.
2. Verarbeiten Sie das Schlüsseltuple (KeyTuple), und rufen Sie den Wert des Schlüsselattributs "orderNumber" ab.
3. Verarbeiten Sie das Werttuple (ValueTuple), und rufen Sie den Wert des Attributs "date" ab.
4. Verarbeiten Sie das Werttuple (ValueTuple), und rufen Sie den Wert der Schlüssel aus der Assoziation "customer" ab.

5. Extrahieren Sie CUSTOMER\_ID. Basierend auf der Beziehung kann ein Auftrag (Order) nur einen einzigen Kunden (Customer) haben. Deshalb gibt es auch nur einen einzigen Schlüssel. Die Größe der Schlüssel ist damit 1. Zur Einfachheit wird die Syntaxanalyse des Datums auf das richtige Format übersprungen.
6. Da es sich um eine Einfügeoperation (insert) handelt, wird die SQL-Anweisung in der Datenquellenverbindung übergeben, um die Einfügeoperation durchzuführen.

Informationen zur Transaktionsabgrenzung und zum Zugriff auf die Datenbank finden Sie im Abschnitt „Loader schreiben“ auf Seite 158.

## Methode "get"

Wenn der Schlüssel nicht im Cache gefunden wird, rufen Sie die Methode "get" im Loader-Plug-in auf, um den Schlüssel zu suchen.

Der Schlüssel ist ein Tupel. Der erste Schritt ist die Konvertierung des Tupels in primitive Werte, die an die SQL-Anweisung SELECT übergeben werden können. Nachdem alle Attribute aus der Datenbank abgerufen wurden, müssen Sie sie in Tupel konvertieren. Der folgende Code demonstriert die Klasse "Order":

```
public List get(TxID txid, List keyList, boolean forUpdate) throws LoaderException {
    System.out.println("OrderLoader: Get called");
    ArrayList returnList = new ArrayList();

1)  if (entityMetaData != null) {
        int index=0;
        for (Iterator iter = keyList.iterator(); iter.hasNext();) {
2)      Tuple orderKeyTuple=(Tuple) iter.next();

            // Der Auftrag (order) hat nur einen einzigen Schlüssel, die Auftragsnummer (orderNumber).
3)      String ORDERNUMBERKEY = (String) getKeyAttribute("orderNumber",orderKeyTuple);
            //Es muss eine Abfrage ausgeführt werden, um die Werte von
4)      // SELECT CUSTOMER_ID, date FROM ORDER WHERE ORDERNUMBER='ORDERNUMBERKEY' abzurufen
5)      //1) Fremdschlüssel: CUSTOMER_ID
6)      //2) date
            // Annehmen, dass diese beiden wie folgt zurückgegeben werden:ls
7)      String CUSTOMER_ID = "C001"; // Annehmen, dass sie abrufen und initialisiert wurden
8)      java.util.Date retrievedDate = new java.util.Date();
            // Annehmen, dass dieses Datum das Datum in der Datenbank widerspiegelt

            // Jetzt müssen diese Daten vor der Rückgabe in eine Tupel konvertiert werden.

            // Werttupel erstellen
9)      TupleMetadata valueMD = entityMetaData.getValueMetadata();
            Tuple valueTuple=valueMD.createTuple();

            // retrievedDate-Objekt dem Tupel hinzufügen
            int datePosition = valueMD.getAttributePosition("date");
10)     valueTuple.setAttribute(datePosition, retrievedDate);

            // Jetzt muss die Assoziation hinzugefügt werden.
11)     int customerPosition=valueMD.getAssociationPosition("customer");
            TupleAssociation customerTupleAssociation =
                valueMD.getAssociation(customerPosition);
            EntityMetadata customerEMD = customerTupleAssociation.getTargetEntityMetadata();
            TupleMetadata customerTupleMDForKey=customerEMD.getKeyMetadata();
12)     int customerKeyAt=customerTupleMDForKey.getAttributePosition("id");

            Tuple customerKeyTuple=customerTupleMDForKey.createTuple();
            customerKeyTuple.setAttribute(customerKeyAt, CUSTOMER_ID);
13)     valueTuple.addAssociationKeys(customerPosition, new Tuple[] {customerKeyTuple});

14)     int linesPosition = valueMD.getAssociationPosition("lines");
            TupleAssociation linesTupleAssociation = valueMD.getAssociation(linesPosition);
            EntityMetadata orderLineEMD = linesTupleAssociation.getTargetEntityMetadata();
            TupleMetadata orderLineTupleMDForKey = orderLineEMD.getKeyMetadata();
            int lineNumberAt = orderLineTupleMDForKey.getAttributePosition("lineNumber");
            int orderAt = orderLineTupleMDForKey.getAssociationPosition("order");

            if (lineNumberAt < 0 || orderAt < 0) {
                throw new IllegalArgumentException(
                    "Invalid position index for lineNumber or order "+
                    lineNumberAt + " " + orderAt);
            }
15) // SELECT LINENUMBER FROM ORDERLINE WHERE ORDERNUMBER='ORDERNUMBERKEY'
```

```

// Annehmen, dass zwei lineNumber-Zeilen mit den Werten 1 und 2 zurückgegeben werden

Tuple orderLineKeyTuple1 = orderLineTupleMDForKEY.createTuple();
orderLineKeyTuple1.setAttribute(lineNumberAt, new Integer(1)); // Schlüssel setzen
orderLineKeyTuple1.addAssociationKey(orderAt, orderKeyTuple);

Tuple orderLineKeyTuple2 = orderLineTupleMDForKEY.createTuple();
orderLineKeyTuple2.setAttribute(lineNumberAt, new Integer(2)); // Schlüssel initialisieren
orderLineKeyTuple2.addAssociationKey(orderAt, orderKeyTuple);

16) valueTuple.addAssociationKeys(linesPosition, new Tuple[]
    {orderLineKeyTuple1, orderLineKeyTuple2 });

returnList.add(index, valueTuple);

index++;

}
} else {
// Unterstützt keine Tuple
}
}
return returnList;
}

```

1. Die Methode "get" wird aufgerufen, wenn der ObjectGrid-Cache den Schlüssel nicht findet und den Loader auffordert, den Schlüssel abzurufen. Suchen Sie den entityMetaData-Wert, und fahren Sie fort, wenn der Wert ungleich null ist.
2. Die Schlüsselliste enthält Tupel.
3. Rufen Sie den Wert des Attributs "orderNumber" ab.
4. Führen Sie die Abfrage aus, um das Datum (Wert) und die Kunden-ID (Fremdschlüssel) abzurufen.
5. CUSTOMER\_ID ist ein Fremdschlüssel, der im Assoziationstupel gesetzt werden muss.
6. Das Datum ist ein Wert und muss bereits gesetzt sein.
7. Da in diesem Beispiel keine JDBC-Aufrufe ausgeführt werden, wird CUSTOMER\_ID angenommen.
8. Da in diesem Beispiel keine JDBC-Aufrufe ausgeführt werden, wird date angenommen.
9. Erstellen Sie das Werttupel.
10. Setzen Sie den Wert von "date" im Tupel, je nach Position.
11. Das Order-Objekt hat zwei Assoziationen. Beginnen Sie mit dem Attribut "customer", das auf die Kundenentität verweist. Sie müssen den Wert von ID haben, um ihn im Tupel zu setzen.
12. Ermitteln Sie die Position von ID in der Entity "Customer".
13. Setzen Sie nur die Werte der Assoziationsschlüssel.
14. "lines" ist auch eine Assoziation, die als Gruppe von Assoziationsschlüsseln konfiguriert werden muss (auf dieselbe Weise wie bei der customer-Assoziation).
15. Da Sie Schlüssel für die Positionsnummer (lineNumber) festlegen müssen, die diesem Auftrag zugeordnet ist, führen Sie die SQL zum Abrufen der lineNumber-Werte aus.
16. Definieren Sie die Assoziationsschlüssel im Werttupel (valueTuple). Diese Aktion schließt die Erstellung des Tupels ab, das an die BackingMap zurückgegeben wird.

Dieser Abschnitt beschreibt die Schritte zum Erstellen von Tupeln und enthält nur eine Beschreibung der Entität "Order". Führen Sie ähnliche Schritte für die anderen Entitäten und den gesamten Prozess aus, der in Zusammenhang mit dem TransactionCallback-Plug-in steht. Weitere Einzelheiten finden Sie im Abschnitt „TransactionCallback-Plug-in“ auf Seite 146.

## Loader mit einem Preload-Controller für Replikate schreiben

Ein Loader mit einem Preload-Controller für Replikate ist ein Loader, der auch die Schnittstelle "com.ibm.websphere.objectgrid.plugins.ReplicaPreloadController" implementiert.

### Übersicht

Die Schnittstelle "ReplicaPreloadController" bietet einem Replikate, das zum primären Shard wird, die Möglichkeit festzustellen, ob das vorherige primäre Shard den Preload-Prozess (Prozess für vorheriges Laden) vollständig abgeschlossen hat. Wenn der Preload-Prozess nur teilweise abgeschlossen ist, werden Informationen bereitgestellt, anhand derer das neue primäre Replikate die Verarbeitung dort fortsetzen kann, wo das vorherige Replikate aufgehört hat. Mit der Implementierung der Schnittstelle "ReplicaPreloadController" setzt ein Replikate, das zum primären Replikate wird, den Preload-Prozess dort fort, wo das vorherige Replikate aufgehört hat, und beendet den gesamten Preload-Prozess.

In einer verteilten WebSphere eXtreme Scale-Umgebung kann eine Map Replikate haben und ein hohes Datenvolumen während der Initialisierung vorher laden. Der Preload-Prozess ist eine Loader-Aktivität und findet nur in der primären Map während der Initialisierung statt. Die Ausführung des Preload-Prozesses kann lange dauern, wenn ein hohes Datenvolumen vorher geladen werden muss. Wenn die primäre Map bereits einen großen Teil der Preload-Daten geladen hat, aber dann aus einem unbekanntem Grund während der Initialisierung gestoppt wird, wird ein Replikate zur primären Map. In dieser Situation gehen die von der vorherigen Map bereits geladenen Preload-Daten verloren, weil die neue primäre Map normalerweise einen unbedingten Preload durchführt. Bei einem unbedingten Preload startet die neue primäre Map den Preload-Prozess von vorn, und die bereits geladenen Daten werden ignoriert. Wenn die neue primäre Map dort weitermachen soll, wo die vorherige primäre Map während des Preload-Prozesses aufgehört hat, stellen Sie einen Loader bereit, der die Schnittstelle "ReplicaPreloadController" implementiert. Weitere Informationen finden Sie in der API-Dokumentation.

Weitere Informationen zu Loadern finden Sie in der Dokumentation zu Loadern im *Administratorhandbuch*. Wenn Sie ein reguläres Loader-Plug-in schreiben möchten, lesen Sie den Abschnitt „Loader schreiben“ auf Seite 158.

Die Schnittstelle "ReplicaPreloadController" hat die folgende Definition:

```
public interface ReplicaPreloadController
{
    public static final class Status
    {
        static public final Status PRELOADED_ALREADY = new Status(K_PRELOADED_ALREADY);
        static public final Status FULL_PRELOAD_NEEDED = new Status(K_FULL_PRELOAD_NEEDED);
        static public final Status PARTIAL_PRELOAD_NEEDED = new Status(K_PARTIAL_PRELOAD_NEEDED);
    }

    Status checkPreloadStatus(Session session, BackingMap bmap);
}
```

In den folgenden Abschnitten werden einige Methoden der Schnittstellen "Loader" und "ReplicaPreloadController" beschrieben.

### Methode "checkPreloadStatus"

Wenn ein Loader die Schnittstelle "ReplicaPreloadController" implementiert, wird während der Map-Initialisierung die Methode "checkPreloadStatus" vor der Methode "preloadMap" aufgerufen. Der Rückkehrstatus dieser Methode bestimmt,



ob die Methode "preloadMap" aufgerufen wird. Wenn diese Methode Status#PRELOADED\_ALREADY zurückgibt, wird die Preload-Methode nicht aufgerufen. Andernfalls wird die Methode "preload" aufgerufen. Aufgrund dieses Verhaltens sollte diese Methode als Methode für die Loader-Initialisierung dienen. Sie müssen in dieser Methode die Loader-Eigenschaften initialisieren. Diese Methode muss den richtigen Status zurückgeben, oder der Preload-Prozess funktioniert nicht wie erwartet.

```
public Status checkPreloadStatus(Session session, BackingMap backingMap) {
    // Wenn ein Loader die Schnittstelle "ReplicaPreloadController" implementiert, wird diese Methode
    // während der Map-Initialisierung vor der Methode "preloadMap" aufgerufen.
    // Ob die Methode "preloadMap" aufgerufen wird, richtet sich nach dem Status, den diese Methode zurückgibt.
    // Deshalb dient diese Methode auch als Methode für die Initialisierung des Loaders.
    // Diese Methode muss den richtigen Status zurückgeben, da der Preload-Prozess ansonsten nicht
    // wie erwartet funktioniert.

    // Anmerkung: Hier muss die Loader-Instanz initialisiert werden.
    ivOptimisticCallback = backingMap.getOptimisticCallback();
    ivBackingMapName = backingMap.getName();
    ivPartitionId = backingMap.getPartitionId();
    ivPartitionManager = backingMap.getPartitionManager();
    ivTransformer = backingMap.getObjectTransformer();
    preloadStatusKey = ivBackingMapName + "_" + ivPartitionId;

    try {
        // preloadStatusMap abrufen, um den Preload-Status abzurufen, der von anderen JVMs
        // gesetzt werden kann.
        ObjectMap preloadStatusMap = session.getMap(ivPreloadStatusMapName);

        // Index des zuletzt aufgezeichneten Preload-Datenblocks abrufen.
        Integer lastPreloadedDataChunk = (Integer) preloadStatusMap.get(preloadStatusKey);

        if (lastPreloadedDataChunk == null) {
            preloadStatus = Status.FULL_PRELOAD_NEEDED;
        } else {
            preloadedLastDataChunkIndex = lastPreloadedDataChunk.intValue();
            if (preloadedLastDataChunkIndex == preloadCompleteMark) {
                preloadStatus = Status.PRELOADED_ALREADY;
            } else {
                preloadStatus = Status.PARTIAL_PRELOAD_NEEDED;
            }
        }

        System.out.println("TupleHeapCacheWithReplicaPreloadControllerLoader.checkPreloadStatus()
-> map = " + ivBackingMapName + ", preloadStatusKey = " + preloadStatusKey
+ ", retrieved lastPreloadedDataChunk = " + lastPreloadedDataChunk + ", determined preloadStatus = "
+ getStatusString(preloadStatus));

    } catch (Throwable t) {
        t.printStackTrace();
    }

    return preloadStatus;
}
```

## Methode "preloadMap"

Die Ausführung der Methode "preloadMap" ist von dem von der Methode "checkPreloadStatus" zurückgegebenen Ergebnis abhängig. Wenn die Methode "preloadMap" aufgerufen wird, muss sie gewöhnlich Informationen zum Preload-Status aus der angegebenen Preload-Status-Map abrufen und die weitere Vorgehensweise bestimmen. Idealerweise sollte die Methode "preloadMap" wissen, ob der Preload-Prozess teilweise abgeschlossen wurde und wo genau begonnen werden muss. Während des Daten-Preloads muss die Methode "preloadMap" den Preload-Status in der angegebenen Preload-Status-Map aktualisieren. Der Preload-Status, der in der Preload-Status-Map gespeichert ist, wird von der Methode "checkPreloadStatus" abgerufen, wenn diese den Preload-Status überprüfen muss.

```
public void preloadMap(Session session, BackingMap backingMap) throws LoaderException {
    EntityMetadata emd = backingMap.getEntityMetadata();
    if (emd != null && tupleHeapPreloadData != null) {
        // Die Methode "getPreLoadData" gleicht dem Abruf von Daten aus der Datenbank.
    }
}
```

```

// Diese Daten werden als Preload-Prozess mit Push in den Cache übertragen.
ivPreloadData = tupleHeapPreloadData.getPreLoadData(emd);

ivOptimisticCallback = backingMap.getOptimisticCallback();
ivBackingMapName = backingMap.getName();
ivPartitionId = backingMap.getPartitionId();
ivPartitionManager = backingMap.getPartitionManager();
ivTransformer = backingMap.getObjectTransformer();
Map preloadMap;

if (ivPreloadData != null) {
    try {
        ObjectMap map = session.getMap(ivBackingMapName);

        // preloadStatusMap abrufen, um den Preload-Status aufzuzeichnen.
        ObjectMap preloadStatusMap = session.getMap(ivPreloadStatusMapName);

        // Anmerkung: Wenn diese Methode "preloadMap" aufgerufen wird,
        // wurde checkPreloadStatus aufgerufen.
        // preloadStatus und preloadedLastDataChunkIndex wurden gesetzt.
        // Der preloadStatus muss PARTIAL_PRELOAD_NEEDED oder
        // FULL_PRELOAD_NEEDED sein. Diese erfordern einen erneuten Preload.

        // Wenn sehr viele Daten vorher geladen werden, werden die Daten gewöhnlich
        // in Blöcke eingeteilt, und der Preload-Prozess verarbeitet jeden Block in
        // einer gesonderten Transaktion.
        // In diesem Beispiel werden nur ein paar Einträge vorher geladen, und jeder
        // Eintrag wird als Block betrachtet.
        // Der Preload-Prozess verarbeitet also jeweils einen Eintrag in einer
        // einer Transaktion, um das vorherige Laden von Datenblöcken zu simulieren.

        Set entrySet = ivPreloadData.entrySet();
        preloadMap = new HashMap();
        ivMap = preloadMap;

        // dataChunkIndex stellt den Datenblock dar, der verarbeitet wird.
        int dataChunkIndex = -1;
        boolean shouldRecordPreloadStatus = false;
        int numberOfDataChunk = entrySet.size();
        System.out.println("    numberOfDataChunk to be preloaded = " + numberOfDataChunk);
        Iterator it = entrySet.iterator();
        int whileCounter = 0;
        while (it.hasNext()) {
            whileCounter++;
            System.out.println("preloadStatusKey = " + preloadStatusKey + " , whileCounter = " + whileCounter);
            dataChunkIndex++;

            // Wenn aktueller dataChunkIndex <= preloadedLastDataChunkIndex
            // ist, ist keine Verarbeitung erforderlich, weil der Datenblock
            // bereits von einer anderen JVM vorher geladen wurde.
            // Verarbeitung nur für dataChunkIndex > preloadedLastDataChunkIndex erforderlich.
            if (dataChunkIndex <= preloadedLastDataChunkIndex) {
                System.out.println("ignore current dataChunkIndex = " + dataChunkIndex
                    + " that has been previously preloaded.");
                continue;
            }

            // Anmerkung: Dieses Beispiel simuliert einen Datenblock mit einem Eintrag.
            // Jeder Schlüssel stellt zur Einfachheit einen Datenblock dar.
            // Wenn der primäre Server oder das primäre Shard aus einem unbekanntem
            // Grund gestoppt wird, muss der Preload-Status, der den Fortschritt des
            // Preload-Prozesses anzeigt, in der preloadStatusMap verfügbar sein.
            // Ein Replikat, das zu einem primären Shard wird kann den Preload-Status
            // abrufen und bestimme, wird der erneute Preload-Prozess durchgeführt
            // werden muss.
            // Anmerkung: Der Preload-Status sollte in derselben Transaktion aufgezeichnet
            // werden, in der auch die Daten in den Cache übertragen werden, so dass
            // der aufgezeichnete Preload-Status der tatsächliche Status ist,
            // falls eine Rollback durchgeführt werden muss oder ein Fehler auftritt.
            Map.Entry entry = (Entry) it.next();
            Object key = entry.getKey();
            Object value = entry.getValue();
            boolean tranActive = false;

            System.out.println("processing data chunk. map = " + this.ivBackingMapName
                + " , current dataChunkIndex = " + dataChunkIndex + " , key = " + key);

            try {
                shouldRecordPreloadStatus = false; // re-set to false

```

```

session.beginNoWriteThrough();
tranActive = true;

if (ivPartitionManager.getNumOfPartitions() == 1) {
    // Wenn nur eine einzige Partition vorhanden ist, Partitionierung übergehen.
    // Die Daten nur mit Push in den Cache übertragen.
    map.put(key, value);
    preloadMap.put(key, value);
    shouldRecordPreloadStatus = true;
} else if (ivPartitionManager.getPartition(key) == ivPartitionId) {
    // Wenn die Map partitioniert ist, muss der Partitionsschlüssel berücksichtigt werden.
    // Nur Daten vorher laden, die zu dieser Partition gehören.
    preloadMap.put(key, value);
    shouldRecordPreloadStatus = true;
} else {
    // Entität ignorieren, weil sie nicht zu dieser Partition gehört.
}

map.put(key, value);

if (shouldRecordPreloadStatus) {
    System.out.println("record preload status. map = " + this.ivBackingMapName
        + ", preloadStatusKey = " + preloadStatusKey + ", current dataChunkIndex = "
        + dataChunkIndex);
    if (dataChunkIndex == numberOfDataChunk) {
        System.out.println("record preload status. map = " + this.ivBackingMapName
            + ", preloadStatusKey = " + preloadStatusKey + ", mark complete = "
            + preloadCompleteMark);
        // Bedeutet, dass dies der letzte Datenblock ist. Bei erfolgreicher Festschreibung,
        // aufzeichnen, dass Preload abgeschlossen ist.
        // Jetzt ist der Preload abgeschlossen.
        // -99 als Sondermarkierung für den Status "Preload abgeschlossen" verwenden.
        preloadStatusMap.get(preloadStatusKey);
        // Ein put nach einem get wird zu update, wenn get ein Objekt zurückgibt,
        // andernfalls wird es zu insert.
        preloadStatusMap.put(preloadStatusKey, new Integer(preloadCompleteMark));
    } else {
        // Datensatz hat vorher geladenen aktuellen dataChunkIndex in preloadStatusMap geladen
        // Ein put nach einem get wird zu update, wenn get ein Objekt zurückgibt,
        // andernfalls wird es zu insert.
        preloadStatusMap.get(preloadStatusKey);
        preloadStatusMap.put(preloadStatusKey, new Integer(dataChunkIndex));
    }
}

session.commit();
tranActive = false;

// Zum Simulieren eines Preloads großer Datenmengen
// diesen Thread für 30 Sekunden inaktivieren.
// Die echte Anwendung darf diesen Thread NICHT inaktivieren.
Thread.sleep(10000);

} catch (Throwable e) {
    e.printStackTrace();
    throw new LoaderException("preload failed with exception: " + e, e);
} finally {
    if (tranActive && session != null) {
        try {
            session.rollback();
        } catch (Throwable e1) {
            // Preload durchführen und Ausnahme der Rollback-Operation ignorieren
        }
    }
}

// Jetzt ist der Preload-Prozess wirklich abgeschlossen.
// -99 als Sondermarkierung für den Status "Preload abgeschlossen" verwenden.
// Dies gewährleistet, dass die Abschlussmarkierung gesetzt wird.
// Außerdem weiß bei der Partitionierung nicht jede Partition, wann es sich
// um den letzten Datenblock handelt.
// Deshalb dient der folgende Block als Bericht für den Bericht, dass der gesamte
// Preload-Prozess abgeschlossen ist.
System.out.println("Overall preload status complete -> record preload status. map = "
    + this.ivBackingMapName + ", preloadStatusKey = " + preloadStatusKey + ", mark complete = "
    + preloadCompleteMark);
session.begin();
preloadStatusMap.get(preloadStatusKey);

```



Der Typ wird durch eine der in der Schnittstelle "LogElement" definierten Konstanten dargestellt: INSERT, UPDATE, DELETE, EVICT, FETCH oder TOUCH.

Der aktuelle Wert stellt den neuen Wert für die Operation dar, wenn diese eine Operation vom Typ INSERT, UPDATE oder FETCH ist. Wenn es sich bei der Operation um eine des Typs TOUCH, DELETE oder EVICT handelt, ist der aktuelle Wert null. Dieser Wert kann in ValueProxyInfo umgesetzt werden, wenn ein ValueInterface verwendet wird.

Weitere Einzelheiten zur Schnittstelle "LogElement" finden Sie in der API-Dokumentation.

## Schnittstelle "LogSequence"

In den meisten Transaktionen werden Operationen für mehrere Einträge in einer Map ausgeführt, so dass mehrere LogElement-Objekte erstellt werden. Sie müssen ein Objekt erstellen, das sich wie ein Verbund mehrerer LogElement-Objekte verhält. Die Schnittstelle "LogSequence" dient diesem Zweck und enthält eine Liste von LogElement-Objekten.

Weitere Einzelheiten zur Schnittstelle "LogSequence" finden Sie in der API-Dokumentation.

## LogElement und LogSequence verwenden

LogElement und LogSequence werden in eXtreme Scale und von ObjectGrid-Plugins, die von Benutzern geschrieben werden, häufig verwendet, wenn Operationen von einer Komponente oder einem Server an eine andere Komponente bzw. einen anderen Server weitergegeben werden. Beispielsweise kann ein LogSequence-Objekt von der ObjectGrid-Funktion für verteilte Transaktionsweitergabe verwendet werden, um die Änderungen an andere Server weiterzugeben, oder vom Loader auf den persistenten Speicher angewendet werden. LogSequence-Objekte werden hauptsächlich von den folgenden Schnittstellen verwendet:

- com.ibm.websphere.objectgrid.plugins.ObjectGridEventListener
- com.ibm.websphere.objectgrid.plugins.Loader
- com.ibm.websphere.objectgrid.plugins.Evictor
- com.ibm.websphere.objectgrid.Session

## Loader-Beispiel

Dieser Abschnitt beschreibt, wie die LogSequence- und LogElement-Objekte in einem Loader verwendet werden. Ein Loader (Ladeprogramm) wird verwendet, um Daten aus einem und in einen persistenten Speicher zu laden. Die Methode "batchUpdate" der Schnittstelle "Loader" verwendet ein LogSequence-Objekt:

```
void batchUpdate(TxID txid, LogSequence sequence) throws LoaderException, OptimisticCollisionException
```

Die Methode "batchUpdate" wird aufgerufen, wenn ein ObjectGrid alle aktuellen Änderungen auf den Loader anwenden muss. Der Loader erhält, gekapselt in einem LogSequence-Objekt, eine Liste der LogElement-Objekte für die Map. Die Implementierung der Methode "batchUpdate" muss durch die Änderungen iterieren und sie auf das Back-End anwenden. Das folgende Code-Snippet veranschaulicht, wie ein Loader ein LogSequence-Objekt verwendet. Das Snippet iteriert durch den Änderungssatz und erstellt drei JDBC-Stapelanweisungen: inserts, updates und deletes:

```

public void batchUpdate(TxID tx, LogSequence sequence) throws LoaderException {
// Zu verwendende SQL-Verbindung abrufen.
    Connection conn = getConnection(tx);
    try
    {
        // Die Liste der Änderungen verarbeiten und vorbereitete Anweisungen für
        // die Ausführung von SQL-Aktualisierungs-, Einfüge- oder Löschooperationen
        // im Stapelbetrieb erstellen. Die Anweisungen werden in stmtCache
        // zwischengespeichert.
        Iterator iter = sequence.getPendingChanges();
        while ( iter.hasNext() )
        {
            LogElement logElement = (LogElement)iter.next();
            Object key = logElement.getCacheEntry().getKey();
            Object value = logElement.getCurrentValue();
            switch ( logElement.getType().getCode() )
            {
                case LogElement.CODE_INSERT:
                    buildBatchSQLInsert( key, value, conn );
                    break;
                case LogElement.CODE_UPDATE:
                    buildBatchSQLUpdate( key, value, conn );
                    break;
                case LogElement.CODE_DELETE:
                    buildBatchSQLDelete( key, conn );
                    break;
            }
        }
        // Die Stapelanweisungen ausführen, die mit der Schleife erstellt wurden
        Collection statements = getPreparedStatementCollection( tx, conn );
        iter = statements.iterator();
        while ( iter.hasNext() )
        {
            PreparedStatement pstmt = (PreparedStatement) iter.next();
            pstmt.executeBatch();
        }
    } catch (SQLException e)
    {
        LoaderException ex = new LoaderException(e);
        throw ex;
    }
}

```

Das vorherige Muster veranschaulicht die übergeordnete Verarbeitungslogik für das Argument "LogSequence". Das Muster zeigt jedoch nicht die Details der Erstellung einer SQL-Anweisung "insert", "update" oder "delete". Die Methode "getPendingChanges" wird für das Argument "LogSequence" aufgerufen, um einen Iterator für die LogElement-Objekte abzurufen, die ein Loader verarbeiten muss, und die Methode "LogElement.getType().getCode()" wird verwendet, um festzustellen, ob ein LogElement für eine SQL-Operation "insert", "update" oder "delete" bestimmt ist.

## Evictor-Beispiel

Sie können LogSequence- und LogElement-Objekte auch mit einem Evictor (Bereinigungsprogramm) verwenden. Ein Evictor wird verwendet, um Map-Einträge auf der Basis bestimmter Kriterien aus der BackingMap zu löschen. Die Methode "apply" der Schnittstelle "Evictor" verwendet LogSequence.

```

/**
 * Diese Methode wird während der Cachefestschreibung verwendet, damit das
 * Bereinigungsprogramm (Evictor) die Verwendung des Objekts in einer BackingMap-Instanz
 * verfolgen kann. Diese Methode meldet außerdem alle Einträge, die gelöscht
 * bereinigt.

```

```

*
* @param sequence LogSequence-Objekt mit den an der Map vorgenommenen Änderungen
*/
void apply(LogSequence sequence);

```

## Schnittstellen "LogSequenceFilter" und "LogSequenceTransformer"

Manchmal müssen die LogElement-Objekte gefiltert werden, damit nur LogElement-Objekte mit bestimmten Kriterien akzeptiert und andere Objekte zurückgewiesen werden. Sie können beispielsweise ein bestimmtes LogElement auf der Basis eines bestimmten Kriteriums serialisieren.

LogSequenceFilter löst dieses Problem mit der folgenden Methode.

```
public boolean accept (LogElement logElement);
```

Diese Methode gibt "true" zurück, wenn das angegebene LogElement-Objekt in der Operation verwendet werden soll. Andernfalls gibt die Methode den Wert "false" zurück.

LogSequenceTransformer ist eine Klasse, die die Funktion "LogSequenceFilter" verwendet. Diese Klasse verwendet LogSequenceFilter, um LogElement-Objekte zu filtern und die akzeptierten LogElement-Objekte anschließend zu serialisieren. Diese Klasse hat zwei Methoden. Die erste Methode sehen Sie im Folgenden.

```
public static void serialize(Collection logSequences, ObjectOutputStream stream,
    LogSequenceFilter filter, DistributionMode mode) throws IOException
```

Diese Methode erlaubt dem Aufrufenden, einen Filter zu definieren, um die LogElement-Objekte zu ermitteln, die in den Serialisierungsprozess aufgenommen werden. Der Parameter "DistributionMode" ermöglicht dem Aufrufenden, den Serialisierungsprozess zu steuern. Der Wert muss nicht serialisiert werden, wenn der Verteilungsmodus auf Invalidierung eingestellt ist. Die zweite Methode dieser Klasse ist die Methode "inflate", die im Folgenden gezeigt wird:

```
public static Collection inflate(ObjectInputStream stream, ObjectGrid
    objectGrid) throws IOException, ClassNotFoundException
```

Die Methode "inflate" liest die serialisierte Form der Protokollfolge, die mit der Methode "serialize" aus dem bereitgestellten Objekteingabedatenstrom erstellt wurde.

---

## eXtreme Scale mit JPA verwenden

Java Persistence API (JPA) ist eine Spezifikation, die die Zuordnung von Java-Objekten zu relationalen Datenbank ermöglicht. JPA enthält eine vollständige ORM-Spezifikation (Object-Relational Mapping, objektbezogene Zuordnung) mit Metadatenannotationen für die Sprache Java und XML-Deskriptoren für die Definition der Zuordnung von Java-Objekten zu einer relationalen Datenbank und umgekehrt. Es gibt eine Reihe von Open-Source- und kostenpflichtigen Implementierungen.

Zur Verwendung von JPA müssen Sie einen unterstützten JPA-Provider wie OpenJPA oder Hibernate, JAR-Dateien und eine Datei META-INF/persistence.xml in Ihrem Klassenpfad haben.

## Übersicht über das clientbasierte JPA-Preload-Dienstprogramm

Das clientbasierte JPA-Preload-Dienstprogramm (Java Persistence API) lädt Daten über eine Clientverbindung zum ObjectGrid in die BackingMaps von eXtreme Scale.

Diese Funktion kann das Laden der eXtreme-Scale-Maps vereinfachen, wenn die Datenbankabfragen nicht partitioniert werden können. Es kann auch Ladeprogramm, wie z. B. ein JPA-Loader, verwendet werden, das sich ideal eignet, wenn die Daten parallel geladen werden können.

Das clientbasierte JPA-Preload-Dienstprogramm kann die OpenJPA- und Hibernate-JPA-Implementierungen verwenden, um das ObjectGrid aus einer Datenbank zu laden. Da WebSphere eXtreme Scale nicht direkt mit der Datenbank bzw. mit Java Database Connectivity (JDBC) interagiert, kann jede von OpenJPA bzw. Hibernate unterstützte Datenbank zum Laden des ObjectGrids verwendet werden.

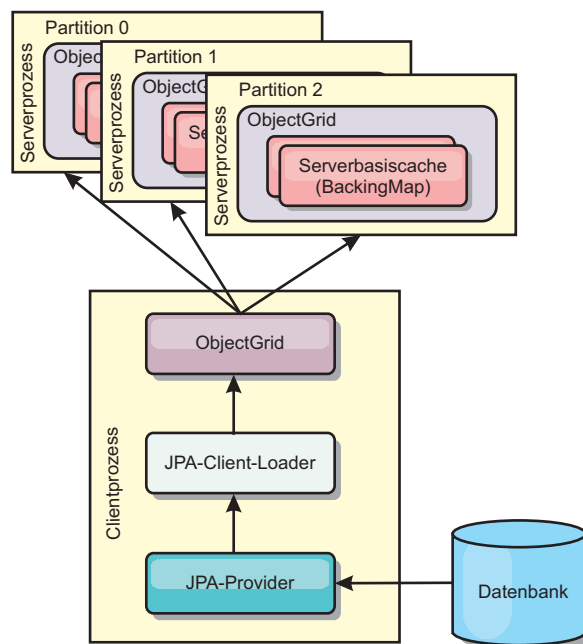


Abbildung 11. Clientladeprogramme, die die JPA-Implementierung zum Laden des ObjectGrids verwenden

Gewöhnlich übergibt eine Benutzeranwendung den Namen einer Persistenzeinheit, den Namen einer Entitätsklasse und eine JPA-Abfrage an das Clientladeprogramm. Das Clientladeprogramm ruft den JPA-EntityManager anhand des Namens der Persistenzeinheit ab, verwendet den EntityManager, um Daten aus der Datenbank über die angegebene Entitätsklasse und JPA-Abfrage abzufragen, und lädt schließlich die Daten in die verteilten ObjectGrid-Maps. Wenn mehrschichtige Relationen an der Abfrage beteiligt sind, können Sie eine angepasste Abfragezeichenfolge verwenden, um die Leistung zu optimieren. Optional können Sie eine Map für Persistenzeigenschaften angeben, um die konfigurierten Persistenzeigenschaften zu überschreiben.

Ein Clientladeprogramm kann Daten in zwei verschiedenen Modi laden, wie in der folgenden Tabelle gezeigt wird:



Tabelle 11. Modi des Clientladeprogramms

Modus	Beschreibung
<i>Preload (Vorheriges Laden)</i>	Löscht und lädt alle Einträge in die BackingMap. Wenn die Map eine Entity-Map ist, werden auch alle zugehörigen Entity-Maps gelöscht, wenn die ObjectGrid-Option "CascadeType.REMOVE" aktiviert ist.
<i>Reload (Erneutes Laden)</i>	Die JPA-Abfrage wird für das ObjectGrid ausgeführt, um alle Entitäten in der Map ungültig zu machen, die der Abfrage entsprechen. Wenn die Map eine Entity-Map ist, werden auch alle zugehörigen Entity-Maps gelöscht, wenn die ObjectGrid-Option "CascadeType.INVALIDATE" aktiviert ist.

In beiden Fällen wird eine JPA-Abfrage verwendet, um die gewünschten Entitäten aus der Datenbank auszuwählen und zu laden und in den ObjectGrid-Maps zu speichern. Wenn die ObjectGrid-Map keine Entity-Map ist, werden die JPA-Entitäten freigegeben und direkt gespeichert. Wenn die ObjectGrid-Map eine Entity-Map ist, werden die JPA-Entitäten als ObjectGrid-Entitätstupel gespeichert. Sie können eine JPA-Abfrage angeben oder die Standardabfrage `select o from EntityName o` verwenden.

Weitere Informationen zum Konfigurieren des clientbasierten JPA-Preload-Dienstprogramms finden Sie in dem entsprechenden Abschnitt im *Programmierhandbuch*

### Programmierung eines clientbasierten JPA-Preload-Dienstprogramms

Das clientbasierte JPA-Preload-Dienstprogramm (Java Persistence API) lädt Daten über eine Clientverbindung zum ObjectGrid in die BackingMaps von eXtreme Scale. Sie können das vorherige Laden (Preload) und das erneute Laden (Reload) von Daten in Ihrer Anwendung implementieren.

#### Schnittstelle "StateManager" verwenden

Verwenden Sie die Methode "setObjectGridState" der Schnittstelle "StateManager", um den ObjectGrid-Status auf einen der folgenden Werte zu setzen: OFFLINE, ONLINE, QUIESCE oder PRELOAD. Die Schnittstelle "StateManager" verhindert, dass andere Clients auf das ObjectGrid zugreifen, wenn es noch nicht online ist.

Setzen Sie den ObjectGrid-Status beispielsweise auf PRELOAD, bevor Sie Daten in die Maps laden. Nach Abschluss des Datenladevorgangs setzen Sie den ObjectGrid-Status zurück auf ONLINE. Weitere Informationen finden Sie in den Informationen zum Festlegen der Verfügbarkeit eines ObjectGrids im *Administratorhandbuch*.

Wenn Sie einen Preload-Prozess für verschiedene Maps in einem einzigen ObjectGrid durchführen, setzen Sie den ObjectGrid-Status einmal auf PRELOAD und dann wieder zurück auf ONLINE, wenn die Daten in alle Maps geladen wurden. Diese Koordination kann über die Schnittstelle "ClientLoadCallback" vorgenommen werden. Setzen Sie den ObjectGrid-Status nach dem Erhalt der ersten preStart-Benachrichtigung von der ObjectGrid-Instanz auf PRELOAD und nach dem Erhalt der letzten postFinish-Benachrichtigung zurück auf ONLINE. Wenn Sie mehrere Java Virtual Machines koordinieren müssen, muss Ihre Anwendung die Koordination übernehmen.

## Beispiel für einen clientbasierten Preload-Prozess

Der Ablauf beim vorherigen Laden von Daten ist wie folgt:

1. Löschen Sie die Map, in die die Daten vorher geladen sollen. Wenn es sich um eine Entitäts-Map handelt und eine Beziehung mit "cascade-remove" konfiguriert ist, werden auch die zugehörigen Maps gelöscht.
2. Führen Sie die JPA-Abfrage für die Entitäten in einem Stapel aus. Die Stapelgröße ist 1000.
3. Erstellen Sie für jeden Stapel eine Schlüsselliste und eine Werteliste für jede Partition.
4. Rufen Sie für jede Partition den DataGrid-Agenten aus, um die Daten auf der Serverseite direkt einzufügen bzw. zu aktualisieren, wenn es sich um einen eXtreme-Scale-Client handelt. Wenn das Grid eine lokale Instanz ist, werden die Daten direkt in den ObjectGrid-Maps eingefügt bzw. aktualisiert.

Das folgende Mustercode-Snippet zeigt einen einfachen Client-Preload-Prozess:

```
// StateManager-Objekt abrufen
StateManager stateMgr = StateManagerFactory.getStateManager();

// ObjectGrid-Status vor dem Aufruf von ClientLoader.load auf PRELOAD setzen
stateMgr.setObjectGridState(AvailabilityState.PRELOAD, objectGrid);

ClientLoader c = ClientLoaderFactory.getClientLoader();

// Daten laden
c.load(objectGrid, "CUSTOMER", "custPU", null,
    null, null, null, true, null);

// ObjectGrid-Status zurück auf ONLINE setzen
stateMgr.setObjectGridState(AvailabilityState.ONLINE, objectGrid);
```

In diesem Beispiel ist die Map CUSTOMER als Entitäts-Map konfiguriert. Die Entitätsklasse "Customer", die in der XML-Deskriptordatei für die ObjectGrid-Entitätsmetadaten konfiguriert ist, hat eine 1:n-Beziehung mit Order-Entitäten. Die Entität "Customer" hat eine aktivierte Option "CascadeType.ALL" in der Beziehung zur Entität "Order".

Vor dem Aufruf der Methode "ClientLoader.load" wird der ObjectGrid-Status auf PRELOAD gesetzt.

Die folgenden Parameter werden in der Methode "ClientLoader.load" verwendet:

1. **objectGrid**: Die ObjectGrid-Instanz. Es handelt sich hier um eine clientseitige ObjectGrid-Instanz.
2. **CUSTOMER**: Die zu ladende Map. Da Customer eine Beziehung des Typs "cascade-all" mit Order-Entitäten hat, werden auch die Order-Entitäten geladen.
3. **custPU**: Der Name der JPA-Persistenzeinheit für die Customer- und Order-Entitäten.
4. **null**: Die Map "persistenceProps" ist null, d. h., es werden die in der Datei "persistence.xml" konfigurierten Standardpersistenzeigenschaften verwendet.
5. **null**: Die entityClass ist mit null konfiguriert. Diese Einstellung wird für die Map CUSTOMER auf die in der XML-Deskriptordatei für die ObjectGrid-Entitätsmetadaten konfigurierte Entitätsklasse gesetzt, in diesem Fall Customer.class.
6. **null**: Die loadSql ist null, d. h., die Standardanweisung "select o from CUSTOMER o" wird zum Abfragen der JPA-Entitäten verwendet.

7. **null**: Die Map für die Abfrageparameter ist null.
8. **true**: Gibt an, dass PRELOAD als Modus für das Laden der Daten gesetzt ist. Deshalb werden Löschoperationen (clear) für die Maps CUSTOMER und ORDER aufgerufen, um alle Daten vor dem Laden zu löschen, weil eine Beziehung des Typs "cascade-remove" zwischen den Maps definiert ist.
9. **null**: Der ClientLoaderCallback ist null.

Weitere Informationen zu den erforderlichen Parametern finden Sie in der Beschreibung der ClientLoader-API in der API-Dokumentation.

## Beispiel für erneutes Laden

Das erneute Laden einer Map (Reload) entspricht dem vorherigen Laden einer Map (Preload) mit der Ausnahme, dass das Argument "isPreload" in der Methode "ClientLoader.load" auf "false" gesetzt wird.

Im Reload-Modus ist der Ablauf für das Laden der Daten wie folgt:

1. Führen Sie die bereitgestellte Abfrage für die ObjectGrid-Map aus, und machen Sie alle Ergebnisse ungültig. Wenn es sich um eine Entitäts-Map handelt und eine Beziehung mit der Option CascadeType.INVALIDATE konfiguriert wurde, werden auch die zugehörigen Entitäten in den entsprechenden Maps ungültig gemacht.
2. Führen Sie die bereitgestellte JPA-Abfrage aus, um die JPA-Entitäten im Stapelbetrieb abzurufen. Die Stapelgröße ist 1000.
3. Erstellen Sie für jeden Stapel eine Schlüsselliste und eine Werteliste für jede Partition.
4. Rufen Sie für jede Partition den DataGrid-Agenten aus, um die Daten auf der Serverseite direkt einzufügen bzw. zu aktualisieren, wenn es sich um einen eXtreme-Scale-Client handelt. Wenn das Grid eine lokale eXtreme-Scale-Konfiguration ist, werden die Daten direkt in den ObjectGrid-Maps eingefügt bzw. aktualisiert.

Im Folgenden sehen Sie ein Reload-Beispiel:

```
// StateManager-Objekt abrufen
StateManager stateMgr = StateManagerFactory.getStateManager();

// ObjectGrid-Status vor dem Aufruf von ClientLoader.load auf PRELOAD setzen
stateMgr.setObjectGridState(AvailabilityState.PRELOAD, objectGrid);

ClientLoader c = ClientLoaderFactory.getClientLoader();

// Daten laden
String loadSql = "select c from CUSTOMER c
  where c.custId >= :startCustId and c.custId < :endCustId ";
Map<String, Long> params = new HashMap<String, Long>();
params.put("startCustId", 1000L);
params.put("endCustId", 2000L);

c.load(objectGrid, "CUSTOMER", "customerPU", null, null,
  loadSql, params, false, null);

// ObjectGrid-Status zurück auf ONLINE setzen
stateMgr.setObjectGridState(AvailabilityState.ONLINE, objectGrid);
```

Verglichen mit dem Preload-Beispiel besteht der Hauptunterschied darin, dass eine loadSql und Parameter angegeben sind. Dieser Mustercode lädt nur die Customer-Daten mit einer ID zwischen 1000 und 2000 erneut.

Beachten Sie, dass diese Abfragezeichenfolge sowohl die JPA-Abfragesyntax als auch die Syntax für eXtreme-Scale-Entitäten einhält. Diese Abfragezeichenfolge ist wichtig, weil sie zweimal ausgeführt wird, einmal für das ObjectGrid, um die übereinstimmenden ObjectGrid-Entitäten ungültig zu machen, und einmal für JPA, um die übereinstimmenden JPA-Entitäten zu laden.

## Client-Loader in einer Loader-Implementierung aufrufen

Die Schnittstelle "Loader" enthält eine Methode "preload":

```
void preloadMap(Session session, BackingMap backingMap) throws  
LoaderException;
```

Diese Methode signalisiert dem Loader, die Daten vorher in die Map zu laden. Eine Loader-Implementierung kann einen Client-Loader verwenden, um die Daten vorher in alle seine Partitionen zu laden. Der JPA-Loader verwendet beispielsweise den Client-Loader, um Daten vorher in die Map zu laden.

Weitere Informationen finden Sie in der Übersicht über JPA-Loader im Handbuch *Produktübersicht*.

Im Folgenden sehen Sie ein Beispiel für das vorherige Laden einer Map mit dem Client-Loader in der Methode "preloadMap". Der Mustercode prüft zuerst, ob die aktuelle Partitionsnummer mit der Preload-Partition identisch ist. Wenn die Partitionsnummer der Preload-Partition nicht entspricht, findet keine Aktion statt. Stimmen die Partitionsnummern überein, wird der Client-Loader aufgerufen, um die Daten in die Maps zu laden. Es ist wichtig, den Client-Loader in einer einzigen Partition aufzurufen.

```
ObjectGrid og = session.getObjectGrid();  
int partitionId = backingMap.getPartitionId();  
int numPartitions = backingMap.getPartitionManager().getNumOfPartitions();  
  
// Client-Loader nur in einer einzigen Partition aufrufen  
if (partitionId == preloadPartition) {  
    ClientLoader loader = ClientLoaderFactory.getClientLoader();  
  
    // Client-Loader zum Laden der Daten aufrufen  
    try {  
        loader.load(og, backingMap.getName(), txCallback.getPersistenceUnitName(),  
            null, entityClass, null, null, true, null);  
    } catch (ObjectGridException e) {  
        LoaderException le = new LoaderException("Exception caught in ObjectMap " +  
            ogName + "." + mapName);  
        le.initCause(e);  
        throw le;  
    }  
}
```

## Manuelle Clientladefunktion

Die Methode "ClientLoader.load" stellt eine Clientladefunktion bereit, die den meisten Szenarios genügt. Wenn Sie jedoch Daten ohne die Methode "ClientLoader.load" laden möchten, können Sie eine eigene Preload-Methode implementieren.

Im Folgenden sehen Sie eine Schablone für eine manuelle Clientladefunktion:

```
// StateManager-Objekt abrufen  
StateManager stateMgr = StateManagerFactory.getStateManager();  
  
// ObjectGrid-Status vor dem Aufruf von ClientLoader.Loader auf PRELOAD setzen  
stateMgr.setObjectGridState(AvailabilityState.PRELOAD, objectGrid);  
  
// Daten laden
```

...

```
// ObjectGrid-Status zurück auf ONLINE setzen
stateMgr.setObjectGridState(AvailabilityState.ONLINE, objectGrid);
```

Wenn Sie die Daten über die Clientseite laden, könnte der Einsatz eines DataGrid-Agenten die Leistung erhöhen. Wenn Sie einen DataGrid-Agenten verwenden, finden alle Lese- und Schreiboperationen für die Daten im Serverprozess statt. Sie können Ihre Anwendung auch so gestalten, dass sichergestellt wird, dass DataGrid-Agenten für mehrere Partitionen parallel ausgeführt werden, um so die Leistung noch weiter zu erhöhen.

Verwenden Sie zum Implementieren des Daten-Preloads mit einem DataGrid-Agenten das folgende Beispiel.

Nachdem Sie die Daten-Preload-Implementierung erstellt haben, können Sie einen generischen Loader für die Ausführung der folgenden Tasks erstellen:

1. Daten aus der Datenbank im Stapelbetrieb abrufen.
2. Schlüsselliste und Werteliste für jede Partition erstellen.
3. Für jede Partition die Methode "agentMgr.callReduceAgent(agent, aKey)" aufrufen, um den Agenten in einem Server-Thread auszuführen. Wenn Sie einen Thread verwenden, können Sie Agenten gleichzeitig für mehrere Partitionen ausführen.

### **Beispiel: Daten-Preload mit einem DataGrid-Agenten**

Wenn Sie die Daten über die Clientseite laden, könnte der Einsatz eines DataGrid-Agenten die Leistung erhöhen. Wenn Sie einen DataGrid-Agenten verwenden, finden alle Lese- und Schreiboperationen für die Daten im Serverprozess statt. Sie können Ihre Anwendung auch so gestalten, dass sichergestellt wird, dass DataGrid-Agenten für mehrere Partitionen parallel ausgeführt werden, um so die Leistung noch weiter zu erhöhen.

Im Folgenden sehen Sie ein Beispiel für das Laden der Daten mit einem DataGrid-Agenten:

```
import java.io.Externalizable;
import java.io.IOException;
import java.io.ObjectInput;
import java.io.ObjectOutput;
import java.util.ArrayList;
import java.util.Collection;
import java.util.Iterator;
import java.util.List;

import com.ibm.websphere.objectgrid.NoActiveTransactionException;
import com.ibm.websphere.objectgrid.ObjectGridException;
import com.ibm.websphere.objectgrid.ObjectGridRuntimeException;
import com.ibm.websphere.objectgrid.ObjectMap;
import com.ibm.websphere.objectgrid.Session;
import com.ibm.websphere.objectgrid.TransactionException;
import com.ibm.websphere.objectgrid.datagrid.ReduceGridAgent;
import com.ibm.websphere.objectgrid.em.EntityManager;

public class InsertAgent implements ReduceGridAgent, Externalizable {

    private static final long serialVersionUID = 6568906743945108310L;

    private List keys = null;

    private List vals = null;
```

```

protected boolean isEntityMap;

public InsertAgent() {
}

public InsertAgent(boolean entityMap) {
    isEntityMap = entityMap;
}

public Object reduce(Session sess, ObjectMap map) {
    throw new UnsupportedOperationException(
        "ReduceGridAgent.reduce(Session, ObjectMap)");
}

public Object reduce(Session sess, ObjectMap map, Collection arg2) {
    Session s = null;
    try {
        s = sess.getObjectGrid().getSession();
        ObjectMap m = s.getMap(map.getName());
        s.beginNoWriteThrough();
        Object ret = process(s, m);
        s.commit();        return ret;
    } catch (ObjectGridRuntimeException e) {
        if (s.isTransactionActive()) {
            try {
                s.rollback();
            } catch (TransactionException e1) {
            } catch (NoActiveTransactionException e1) {
            }
        }
        throw e;
    } catch (Throwable t) {
        if (s.isTransactionActive()) {
            try {
                s.rollback();
            } catch (TransactionException e1) {
            } catch (NoActiveTransactionException e1) {
            }
        }
        throw new ObjectGridRuntimeException(t);
    }
}

public Object process(Session s, ObjectMap m) {
    try {

        if (!isEntityMap) {
            // In the POJO case, it is very straightforward,
            // we can just put everything in the
            // map using insert
            insert(m);
        } else {
            // 2. Entity case.
            // In the Entity case, we can persist the entities
            EntityManager em = s.getEntityManager();
            persistEntities(em);
        }

        return Boolean.TRUE;
    } catch (ObjectGridRuntimeException e) {
        throw e;
    } catch (ObjectGridException e) {
        throw new ObjectGridRuntimeException(e);
    } catch (Throwable t) {
        throw new ObjectGridRuntimeException(t);
    }
}

```

```

}

/**
 * Im Prinzip ist dies ein neuer Ladevorgang.
 * @param s
 * @param m
 * @throws ObjectGridException
 */
protected void insert(ObjectMap m) throws ObjectGridException {

    int size = keys.size();

    for (int i = 0; i < size; i++) {
        m.insert(keys.get(i), vals.get(i));
    }

}

protected void persistEntities(EntityManager em) {
    Iterator<Object> iter = vals.iterator();

    while (iter.hasNext()) {
        Object value = iter.next();
        em.persist(value);
    }
}

public Object reduceResults(Collection arg0) {
    return arg0;
}

public void readExternal(ObjectInput in)
    throws IOException, ClassNotFoundException {
    int v = in.readByte();
    isEntityMap = in.readBoolean();
    vals = readList(in);
    if (!isEntityMap) {
        keys = readList(in);
    }
}

public void writeExternal(ObjectOutput out) throws IOException {
    out.write(1);
    out.writeBoolean(isEntityMap);

    writeList(out, vals);
    if (!isEntityMap) {
        writeList(out, keys);
    }
}

public void setData(List ks, List vs) {
    vals = vs;
    if (!isEntityMap) {
        keys = ks;
    }
}

/**
 * @return Gibt das isEntityMap-Objekt zurück.
 */
public boolean isEntityMap() {
    return isEntityMap;
}

```

```

static public void writeList(ObjectOutput oo, Collection l)
throws IOException {
    int size = l == null ? -1 : l.size();
    oo.writeInt(size);
    if (size > 0) {
        Iterator iter = l.iterator();
        while (iter.hasNext()) {
            Object o = iter.next();
            oo.writeObject(o);
        }
    }
}

public static List readList(ObjectInput oi)
throws IOException, ClassNotFoundException {
    int size = oi.readInt();
    if (size == -1) {
        return null;
    }

    ArrayList list = new ArrayList(size);
    for (int i = 0; i < size; ++i) {
        Object o = oi.readObject();
        list.add(o);
    }
    return list;
}
}

```

## Zeitbasierte JPA-Datenaktualisierungskomponente

Eine zeitbasierte JPA-Datenbankaktualisierungskomponente (Java Persistence API) aktualisiert die ObjectGrid-Maps mit den letzten Änderungen, die in der Datenbank vorgenommen wurden.

Wenn Änderungen direkt in einer Datenbank mit WebSphere eXtreme Scale als Front-End vorgenommen werden, werden diese Änderungen nicht gleichzeitig im eXtreme-Scale-Grid widerspiegelt. Für eine ordnungsgemäße Implementierung von eXtreme Scale als speicherinterner Datenbankverarbeitungsbereich müssen Sie berücksichtigen, dass es vorkommen kann, dass Ihr Grid nicht synchron mit der Datenbank ist. Die zeitbasierte Datenbankaktualisierungskomponente verwendet die SCN-Funktion (System Change Number) in Oracle 10g und das Format ROW CHANGE TIMESTAMP (Zeitmarke für Zeilenänderung) in DB2 9.5, um Änderungen in der Datenbank im Hinblick ungueltige und aktualisierte Einträge zu überwachen. Die Aktualisierungskomponente ermöglicht Anwendungen auch die Verwendung eines benutzerdefinierten Felds für denselben Zweck.



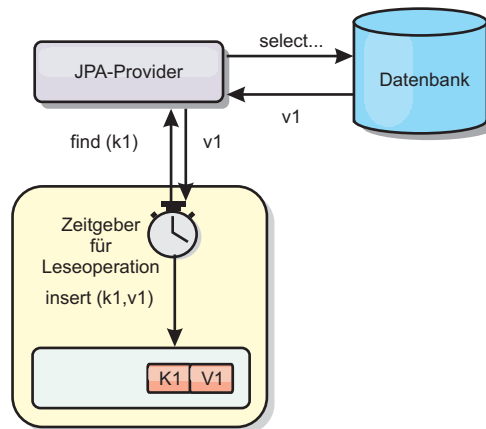


Abbildung 12. Regelmäßige Aktualisierung

Die zeitbasierte Datenbankaktualisierungskomponente fragt die Datenbank regelmäßig über JPA-Schnittstellen ab, um die JPA-Entitäten zu ermitteln, die die neu eingefügten oder aktualisierten Datensätze in der Datenbank darstellen. Um die Datensätze regelmäßig aktualisieren zu können, muss jeder Datensatz in der Datenbank eine Zeitmarke haben, die angibt, wann bzw. in welcher Folge der Datensatz zuletzt aktualisiert bzw. eingefügt wurde. Es ist nicht erforderlich, dass die Zeitmarke ein Zeitmarkenformat hat. Der Zeitmarkenwert kann eine ganze Zahl sein oder ein ausführliches Format haben, solange ein eindeutiger zunehmender Wert generiert wird.

Diese Funktionalität wird von mehreren kostenpflichtigen Datenbanken bereitgestellt.

In DB2 9.5 können Sie beispielsweise eine Spalte mit dem ROW CHANGE TIMESTAMP wie folgt definieren:

```

ROWCHGTS TIMESTAMP NOT NULL
GENERATED ALWAYS
FOR EACH ROW ON UPDATE AS
ROW CHANGE TIMESTAMP
  
```

In Oracle können Sie die Pseudospalte **ora\_rowscn** verwenden, die die SCN (System Change Number, Systemänderungsnummer) des Datensatzes darstellt.

Die zeitbasierte Datenbankaktualisierungskomponente aktualisiert die ObjectGrid-Maps auf drei verschiedene Arten:

1. **INVALIDATE\_ONLY**. Die Einträge in der ObjectGrid-Map werden nur ungültig gemacht, wenn die entsprechenden Datensätze in der Datenbank geändert wurden.
2. **UPDATE\_ONLY**. Die Einträge in der ObjectGrid-Map werden aktualisiert, wenn die entsprechenden Datensätze in der Datenbank geändert wurden. Alle neu in die Datenbank eingefügten Datensätze werden jedoch ignoriert.
3. **INSERT\_UPDATE**. Die vorhandenen Einträge in der ObjectGrid-Map werden mit den aktuellen Werten aus der Datenbank aktualisiert. Außerdem werden alle neu in die Datenbank eingefügten Datensätze in die ObjectGrid-Map eingefügt.

Weitere Informationen zum Konfigurieren der zeitbasierten JAP-Datenaktualisierungskomponente finden Sie in den Informationen im *Administratorhandbuch*.

## Zeitbasierte JPA-Aktualisierungskomponente starten

Wenn die zeitbasierte JPA-Aktualisierungskomponente (Java Persistence API) starten, werden die ObjectGrid-Maps mit den letzten Änderungen aus der Datenbank aktualisiert.

### Vorbereitungen

Konfigurieren Sie die zeitbasierte Aktualisierungskomponente. Weitere Informationen finden Sie in der Beschreibung der Konfiguration einer zeitbasierten JPA-Aktualisierungskomponente im *Administratorhandbuch*.

### Warum und wann dieser Vorgang ausgeführt wird

Weitere Informationen zur Funktionsweise der zeitbasierten JAP-Datenaktualisierungskomponente finden Sie im Abschnitt „Zeitbasierte JPA-Datenaktualisierungskomponente“ auf Seite 188.

- Starten Sie eine zeitbasierte Datenbankaktualisierungskomponente.

- **Automatisch für ein verteiltes eXtreme Scale:**

Wenn Sie die `timeBasedDBUpdate`-Konfiguration für die `BackingMap` erstellen, wird die zeitbasierte Datenbankaktualisierungskomponente automatisch gestartet, wenn ein verteiltes primäres `ObjectGrid-Shard` aktiviert wird. Für ein `ObjectGrid` mit mehreren Partitionen wird die zeitbasierte Datenbankaktualisierungskomponente nur in Partition 0 gestartet.

- **Automatisch für ein lokales eXtreme Scale:**

Wenn Sie die `timeBasedDBUpdate`-Konfiguration für die `BackingMap` erstellen, wird die zeitbasierte Datenbankaktualisierungskomponente automatisch gestartet, wenn die lokale `Map` aktiviert wird.

- **Manuell:**

Sie können die zeitbasierte Datenbankaktualisierungskomponente auch manuell über die API `TimeBasedDBUpdater` starten.

```
public synchronized void startDBUpdate(ObjectGrid objectGrid, String mapName,
    String punitName, Class entityClass, String timestampField, DBUpdateMode mode) {
```

1. **ObjectGrid:** Die `ObjectGrid`-Instanz (lokal oder Client).
2. **mapName:** Der Name der `BackingMap`, für die die zeitbasierte Datenbankaktualisierungskomponente gestartet wird.
3. **punitName:** Der Name der JPA-Persistenzeinheit für das Erstellen einer JPA-`EntityManager-Factory`. Der Standardwert ist der Name der ersten in der Datei `persistence.xml` definierten Persistenzeinheit.
4. **entityClass:** Der Name der Entitätsklasse, die für die Interaktion mit dem JPA-Provider verwendet wird. Der Name der Entitätsklasse wird verwendet, um JPA-Entitäten über Entitätsabfragen abzurufen.
5. **timestampField:** Ein Zeitmarkenfeld der Entitätsklasse, in dem die Zeit oder Folge gespeichert wird, zu der bzw. in der ein Datenbank-Back-End-Datensatz zuletzt aktualisiert oder eingefügt wurde.
6. **mode:** Der Modus der zeitbasierten Datenbankaktualisierung. Der Typ `INVALIDATE_ONLY` bewirkt, dass die Einträge in der `ObjectGrid-Map` ungültig gemacht werden, wenn die entsprechenden Datensätze in der Datenbank geändert wurden. Der Typ `UPDATE_ONLY` zeigt an, dass die vorhandenen Einträge in der `ObjectGrid-Map` mit den aktuellen Werten aus der Datenbank aktualisiert werden sollen. Alle neu in die Datenbank eingefügten Datensätze werden jedoch ignoriert. Der Typ `INSERT_UPDATE` zeigt an, dass die vorhandenen Einträge in der `ObjectGrid-Map` mit

den aktuellen Werten aus der Datenbank aktualisiert und auch alle neu in die Datenbank eingefügten Datensätze in die ObjectGrid-Map eingefügt werden sollen.

Wenn Sie die zeitbasierte Datenbankaktualisierungskomponente stoppen möchten, können Sie dazu die folgende Methode aufrufen:

```
public synchronized void stopDBUpdate(ObjectGrid objectGrid, String mapName)
```

Die Parameter "ObjectGrid" und "mapName" müssen dieselben sein, die auch in der Methode "startDBUpdate" übergeben werden.

- Erstellen Sie das Zeitmarkenfeld in Ihrer Datenbank.

#### – DB2

Im Rahmen des Features für optimistisches Sperren stellt DB2 9.5 ein Zeitmarkenfeature für geänderte Zeilen (row change timestamp) zur Verfügung. Sie können eine Spalte ROWCHGTS im ROW-CHANGE-TIMESTAMP-Format wie folgt erstellen:

```
ROWCHGTS TIMESTAMP NOT NULL
GENERATED ALWAYS
FOR EACH ROW ON UPDATE AS
ROW CHANGE TIMESTAMP
```

Anschließend können Sie das Entitätsfeld, das dieser Spalte entspricht, durch Annotation oder Konfiguration als Zeitmarkenfeld angeben. Es folgt ein Beispiel:

```
@Entity(name = "USER_DB2")
@Table(name = "USER1")
public class User_DB2 implements Serializable {

    private static final long serialVersionUID = 1L;

    public User_DB2() {
    }

    public User_DB2(int id, String firstName, String lastName) {
        this.id = id;
        this.firstName = firstName;
        this.lastName = lastName;
    }

    @Id
    @Column(name = "ID")
    public int id;

    @Column(name = "FIRSTNAME")
    public String firstName;

    @Column(name = "LASTNAME")
    public String lastName;

    @com.ibm.websphere.objectgrid.jpa.dbupdate.annotation.Timestamp
    @Column(name = "ROWCHGTS", updatable = false, insertable = false)
    public Timestamp rowChgTs;
}
```

#### – Oracle

In Oracle gibt es eine Pseudospalte ora\_rowscn für die Systemänderungsnummer des Datensatzes. Sie können diese Spalte für denselben Zweck verwenden. Es folgt ein Beispiel für eine Entität, die das Feld "ora\_rowscn" als Zeitmarkenfeld für die zeitbasierte Datenbankaktualisierung verwendet:

```
@Entity(name = "USER_ORA")
@Table(name = "USER1")
public class User_ORA implements Serializable {
```

```

private static final long serialVersionUID = 1L;

public User_ORA() {
}

public User_ORA(int id, String firstName, String lastName) {
    this.id = id;
    this.firstName = firstName;
    this.lastName = lastName;
}

@Id
@Column(name = "ID")
public int id;

@Column(name = "FIRSTNAME")
public String firstName;

@Column(name = "LASTNAME")
public String lastName;

@com.ibm.websphere.objectgrid.jpa.dbupdate.annotation.Timestamp
@Column(name = "ora_rowscn", updatable = false, insertable = false)
public long rowChgTs;
}

```

#### – Andere Datenbanken

Für andere Typen von Datenbanken können Sie eine Tabellenspalte erstellen, in der die Änderungen verfolgt werden. Die Spaltenwerte müssen von der Anwendung, die die Tabelle aktualisiert, manuell geändert werden.

Nehmen Sie eine Apache-Derby-Datenbank als Beispiel: Sie können eine Spalte "ROWCHGTS" erstellen, um die Änderungsnummern zu verfolgen. Für diese Tabelle wird auch die aktuelle Änderungsnummer verfolgt. Jedesmal, wenn ein Datensatz eingefügt oder aktualisiert wird, wird die letzte Änderungsnummer für die Tabelle um eins erhöht, und der Wert der Spalte ROWCHGTS für den Datensatz wird mit dieser erhöhten Nummer aktualisiert:

```

@Entity(name = "USER_DER")
@Table(name = "USER1")
public class User_DER implements Serializable {

    private static final long serialVersionUID = 1L;

    public User_DER() {
    }

    public User_DER(int id, String firstName, String lastName) {
        this.id = id;
        this.firstName = firstName;
        this.lastName = lastName;
    }

    @Id
    @Column(name = "ID")
    public int id;

    @Column(name = "FIRSTNAME")
    public String firstName;

    @Column(name = "LASTNAME")
    public String lastName;
}

```

```

@com.ibm.websphere.objectgrid.jpa.dbupdate.annotation.Timestamp
@Column(name = "ROWCHGTS", updatable = true, insertable = true)
public long rowChgTs;
}

```

---

## OptimisticCallback-Plug-in

Verwenden Sie das OptimisticCallback-Plug-in, um Versionssteuerungs- und Vergleichsoperationen für Cacheobjekte anzupassen, wenn Sie die optimistische Sperrstrategie verwenden.

Sie können ein Plug-in-fähiges optimistisches Callback-Objekt bereitstellen, das die Schnittstelle "com.ibm.websphere.objectgrid.plugins.OptimisticCallback" implementiert. Für Entitäts-Maps wird automatisch ein OptimisticCallback-Plug-in mit hoher Leistung konfiguriert.

### Zweck

Verwenden Sie die Schnittstelle "OptimisticCallback" für die Unterstützung optimistischer Vergleichsoperationen für die Werte einer Map. Ein OptimisticCallback-Plug-in ist erforderlich, wenn Sie die optimistische Sperrstrategie verwenden. Das Produkt stellt eine Standardimplementierung der Schnittstelle "OptimisticCallback" bereit. Gewöhnlich muss die Anwendung eine eigene Implementierung der Schnittstelle "OptimisticCallback" integrieren.

### Standardimplementierung

Das eXtreme-Scale-Framework stellt eine Standardimplementierung der Schnittstelle "OptimisticCallback" bereit, die verwendet wird, wenn die Anwendung kein anwendungsdefiniertes OptimisticCallback-Objekt bereitstellt. Die Standardimplementierung gibt immer den Sonderwert NULL\_OPTIMISTIC\_VERSION als Versionsobjekt für den Wert zurück und aktualisiert das Versionsobjekt nie. Diese Aktion macht einen optimistischen Vergleich zu einer Funktion mit "Nulloperation". In den meisten Fällen ist die Funktion mit "Nulloperation" nicht angebracht, wenn die optimistische Sperrstrategie verwendet wird. Ihre Anwendungen müssen die Schnittstelle "OptimisticCallback" implementieren und eigene OptimisticCallback-Implementierungen integrieren, damit die Standardimplementierung nicht verwendet wird. Es gibt jedoch mindestens ein Szenario, in dem die bereitgestellte OptimisticCallback-Standardimplementierung hilfreich ist. Stellen Sie sich die folgende Situation vor:

- Es wird ein Loader (Ladeprogramm) für die BackingMap integriert.
- Der Loader weiß ohne Hilfe eines OptimisticCallback-Plug-ins, wie der optimistische Vergleich durchgeführt wird.

Wie kann der Loader nun ohne Hilfe eines OptimisticCallback-Objekts die optimistische Versionssteuerung durchführen? Der Loader kennt das Wertobjekt für die Klasse und weiß, welches Feld des Wertobjekts als Wert für die optimistische Versionssteuerung verwendet wird. Angenommen, die folgende Schnittstelle wird für das Wertobjekt der Map "Employee" verwendet:

```

public interface Employee
{
    // Für die optimistische Versionssteuerung verwendete Folgennummer.
    public long getSequenceNumber();
    public void setSequenceNumber(long newSequenceNumber);
    // Weitere get/set-Methoden für andere Felder des Employee-Objekts.
}

```

In diesem Beispiel weiß der Loader, dass er die Methode "getSequenceNumber" verwenden kann, um die aktuellen Versionsinformationen für ein Employee-Wertobjekt abzurufen. Der Loader erhöht den zurückgegebenen Wert um eins, um eine neue Versionsnummer zu generieren, bevor er den persistenten Speicher mit dem neuen Employee-Wert aktualisiert. Für einen JDBC-Loader (Java Database Connectivity) wird die aktuelle Folgennummer in der WHERE-Klausel einer überqualifizierten SQL-Anweisung "UPDATE" verwendet. Der Loader verwendet die neu generierte Folgennummer, um die Folgennummernspalte auf den neuen Folgennummernwert zu setzen. Eine weitere Möglichkeit ist die, dass der Loader eine vom Back-End bereitgestellte Funktion verwendet, die eine verdeckte Spalte, die für die optimistische Versionssteuerung verwendet werden kann, automatisch aktualisiert.

In manchen Fällen kann unter Umständen eine gespeicherte Prozedur oder ein Trigger verwendet werden, um eine Spalte zu verwalten, die Informationen zur Versionssteuerung enthält. Wenn der Loader eine dieser Techniken für die Verwaltung der Informationen zur optimistischen Versionssteuerung verwendet, muss die Anwendung keine eigene OptimisticCallback-Implementierung bereitstellen. Die OptimisticCallback-Standardimplementierung kann in diesem Szenario verwendet werden, weil der Loader die optimistische Versionssteuerung ohne Hilfe eines OptimisticCallback-Objekts durchführen kann.

## Standardimplementierung für Entitäten

Entitäten werden im ObjectGrid mit Hilfe von Tupelobjekten gespeichert. Die OptimisticCallback-Standardimplementierung gleicht dem Verhalten bei Maps, die keine Entitäts-Maps sind. Das Versionsfeld in der Entität wird jedoch mit der Annotation "@Version" bzw. dem Versionsattribut in der XML-Deskriptordatei der Entität angegeben.

Die gültigen Datentypen für das Versionsattribut sind int, Integer, short, Short, long, Long und java.sql.Timestamp. Für eine Entität darf nur ein einziges Versionsattribut definiert werden. Das Versionsattribut darf nur während der Erstellung definiert werden. Sobald die Entität als persistent definiert wird, darf der Wert des Versionsattributs nicht mehr geändert werden.

Wenn kein Versionsattribut konfiguriert ist und die optimistische Sperrstrategie verwendet wird, wird das vollständige Tupel implizit über den Status des Tupels versionsgesteuert, was sehr viel kostenintensiver ist.

Im folgenden Beispiel hat die Entität "Employee" ein Versionsattribut mit dem Namen "SequenceNumber" und dem Typ "long":

```
@Entity
public class Employee
{
    private long sequence;
    // Für die optimistische Versionssteuerung verwendete Folgennummer.
    @Version
    public long getSequenceNumber() {
        return sequence;
    }
    public void setSequenceNumber(long newSequenceNumber) {
        this.sequence = newSequenceNumber;
    }
    // Weitere get/set-Methoden für andere Felder des Employee-Objekts.
}
```

## OptimisticCallback-Plug-in schreiben

Ein OptimisticCallback-Plug-in muss die Schnittstelle "OptimisticCallback" implementieren und die folgenden Konventionen für ObjectGrid-Plug-ins einhalten. Weitere Informationen finden Sie in der Dokumentation zur Schnittstelle "OptimisticCallback" in der API-Dokumentation.

Die folgende Liste enthält Beschreibungen und Hinweise für alle Methoden in der Schnittstelle "OptimisticCallback":

### NULL\_OPTIMISTIC\_VERSION

Dieser Sonderwert wird von der Methode "getVersionedObjectForValue" zurückgegeben, wenn die OptimisticCallback-Implementierung keine Versionsprüfung erfordert. Die integrierte Plug-in-Implementierung der Klasse "com.ibm.websphere.objectgrid.plugins.builtins.NoVersioningOptimisticCallback" verwendet diesen Wert, weil die Versionssteuerung inaktiviert ist, wenn Sie diese Plug-in-Implementierung angeben.

### Methode "getVersionedObjectForValue"

Die Methode "getVersionedObjectForValue" kann eine Kopie des Werts oder ein Attribut des Werts zurückgeben, das für Versionssteuerungszwecke verwendet werden kann. Diese Methode wird aufgerufen, wenn ein Objekt einer Transaktion zugeordnet wird. Wenn in eine BackingMap kein Loader integriert ist, verwendet die BackingMap diesen Wert während der Festschreibung, um einen optimistischen Versionsvergleich durchzuführen. Der optimistische Versionsvergleich wird von der BackingMap verwendet, um sicherzustellen, dass die Version des Map-Eintrags seit dem ersten Zugriff der Transaktion, die den Map-Eintrag geändert hat, nicht geändert wurde. Wenn eine andere Transaktion die Version für diesen Map-Eintrag bereits geändert hat, schlägt der Versionsvergleich fehl, und die BackingMap zeigt eine Ausnahme des Typs "OptimisticCollisionException" an, um eine Rollback-Operation für die Transaktion zu erzwingen. Wenn ein Loader integriert ist, verwendet die BackingMap die Informationen für die optimistische Versionssteuerung nicht. Stattdessen ist der Loader für die Durchführung der optimistischen Versionssteuerung und die Aktualisierung der Versionssteuerungsinformationen zuständig, sofern dies erforderlich ist. Der Loader ruft gewöhnlich das erste Versionssteuerungsobjekt von dem LogElement-Objekt ab, das an die Methode "batchUpdate" im Loader übergeben wurde, die aufgerufen wird, wenn eine Flush-Operation durchgeführt oder eine Transaktion festgeschrieben wird.

Der folgende Code zeigt die vom EmployeeOptimisticCallbackImpl-Objekt verwendete Implementierung:

```
public Object getVersionedObjectForValue(Object value)
{
    if (value == null)
    {
        return null;
    }
    else
    {
        Employee emp = (Employee) value;
        return new Long( emp.getSequenceNumber() );
    }
}
```

Wie im vorherigen Beispiel gezeigt, wird das Attribut "sequenceNumber" in einem vom Loader erwarteten Objekt des Typs "java.lang.Long" zurückgegeben. Dies

impliziert, dass die Person, die den Loader geschrieben hat, auch die EmployeeOptimisticCallbackImpl-Implementierung geschrieben hat bzw. eng mit der Person zusammengearbeitet hat, die EmployeeOptimisticCallbackImpl implementiert hat, z. B. mit dieser Person den von der Methode "getVersionedObjectForValue" zurückgegebenen Wert vereinbart hat. Das OptimisticCallback-Standard-Plug-in gibt den Sonderwert NULL\_OPTIMISTIC\_VERSION als Versionsobjekt zurück.

### **Methode "updateVersionedObjectForValue"**

Diese Methode wird aufgerufen, wenn eine Transaktion einen Wert aktualisiert hat und ein neues versionsgesteuertes Objekt erforderlich ist. Wenn die Methode "getVersionedObjectForValue" ein Attribut des Werts zurückgibt, aktualisiert diese Methode gewöhnlich den Attributwert mit einem neuen Versionsobjekt. Wenn die Methode "getVersionedObjectForValue" eine Kopie des Werts zurückgibt, führt diese Methode gewöhnlich keine Aktionen aus. Das OptimisticCallback-Standard-Plug-in führt keine Aktionen mit dieser Methode aus, da die Standardimplementierung von getVersionedObjectForValue immer den Sonderwert NULL\_OPTIMISTIC\_VERSION als Versionsobjekt zurückgibt. Der folgende Beispielcode zeigt die vom EmployeeOptimisticCallbackImpl-Objekt im Abschnitt "OptimisticCallback" verwendete Implementierung:

```
public void updateVersionedObjectForValue(Object value)
{
    if ( value != null )
    {
        Employee emp = (Employee) value;
        long next = emp.getSequenceNumber() + 1;
        emp.updateSequenceNumber( next );
    }
}
```

Wie im vorherigen Beispiel gezeigt, wird das Attribut "sequenceNumber" um eins erhöht, so dass beim nächsten Aufruf der Methode "getVersionedObjectForValue" der zurückgegebene Wert vom Typ "java.lang.Long" einen langen Wert hat, der dem ursprünglichen Folgenummernwert plus eins entspricht, z. B. dem nächsten Versionswert für diese Employee-Instanz. Dieses Beispiel impliziert, dass die Person, die den Loader geschrieben hat, auch die EmployeeOptimisticCallbackImpl-Implementierung geschrieben hat bzw. eng mit der Person zusammengearbeitet hat, die EmployeeOptimisticCallbackImpl implementiert hat.

### **Methode "serializeVersionedValue"**

Diese Methode schreibt den versionsgesteuerten Wert in den angegebenen Datenstrom. Je nach Implementierung kann der versionsgesteuerte Wert verwendet werden, um optimistische Aktualisierungskollisionen zu identifizieren. In einigen Implementierungen ist der versionsgesteuerte Wert eine Kopie des ursprünglichen Werts. Andere Implementierungen können eine Folge Nummer oder ein anderes Objekt haben, um die Version des Werts anzugeben. Da die tatsächliche Implementierung nicht bekannt ist, wird diese Methode bereitgestellt, damit die richtige Serialisierung durchgeführt werden kann. Die Standardimplementierung ruft die Methode "writeObject" auf.

### **Methode "inflateVersionedValue"**

Diese Methode akzeptiert die serialisierte Version des versionsgesteuerten Werts und gibt das tatsächliche versionsgesteuerte Wertobjekt zurück. Je nach Implementierung kann der versionsgesteuerte Wert verwendet werden, um optimistische Aktualisierungskollisionen zu identifizieren. In einigen Implementierungen ist der



versionsgesteuerte Wert eine Kopie des ursprünglichen Werts. Andere Implementierungen können eine Folgennummer oder ein anderes Objekt haben, um die Version des Werts anzugeben. Da die tatsächliche Implementierung nicht bekannt ist, wird diese Methode bereitgestellt, damit die richtige Entserialisierung durchgeführt werden kann. Die Standardimplementierung ruft die Methode "readObject" auf.

## Anwendungsdefiniertes OptimisticCallback-Objekt verwenden

Sie können zum Hinzufügen eines anwendungsdefinierten OptimisticCallback-Objekts zur BackingMap-Konfiguration zwischen zwei Ansätzen wählen: der XML-Konfiguration und der programmgesteuerten Konfiguration.

## OptimisticCallback-Objekt durch XML-Konfiguration integrieren

Die Anwendung kann eine XML-Datei verwenden, um ihr OptimisticCallback-Objekt zu integrieren, wie im folgenden Beispiel gezeigt wird:

```
<?xml version="1.0" encoding="UTF-8"?>
<objectGridConfig xmlns:xsi="http://www.w3.org/2001/XMLSchema-instance"
  xsi:schemaLocation="http://ibm.com/ws/objectgrid/config ../objectGrid.xsd"
  xmlns="http://ibm.com/ws/objectgrid/config">
<objectGrids>
  <objectGrid name="grid1">
    <backingMap name="employees" pluginCollectionRef="employees" lockStrategy="OPTIMISTIC" />
  </objectGrid>
</objectGrids>

<backingMapPluginCollections>
  <backingMapPluginCollection id="employees">
    <bean id="OptimisticCallback" className="com.xyz.EmployeeOptimisticCallbackImpl" />
  </backingMapPluginCollection>
</backingMapPluginCollections>
</objectGridConfig>
```

## OptimisticCallback-Objekt über das Programm integrieren

Das folgende Beispiel veranschaulicht, wie eine Anwendung über das Programm ein OptimisticCallback-Objekt für die BackingMap "Employee" in der lokalen ObjectGrid-Instanz "grid1" integriert:

```
import com.ibm.websphere.objectgrid.ObjectGridManagerFactory;
import com.ibm.websphere.objectgrid.ObjectGridManager;
import com.ibm.websphere.objectgrid.ObjectGrid;
import com.ibm.websphere.objectgrid.BackingMap;
ObjectGridManager ogManager = ObjectGridManagerFactory.getObjectGridManager();
ObjectGrid og = ogManager.createObjectGrid( "grid1" );
BackingMap bm = dg.defineMap("employees");
EmployeeOptimisticCallbackImpl cb = new EmployeeOptimisticCallbackImpl();
bm.setOptimisticCallback( cb );
```

---

## ObjectTransformer-Plug-in

Mit dem ObjectTransformer-Plug-in können Sie Objekte im Cache serialisieren, entserialisieren und kopieren. Verwenden Sie das ObjectTransformer-Plug-in, wenn Sie eine hohe Leistung benötigen.

Wenn Sie Leistungsprobleme in Bezug auf die Prozessorbelegung lesen, fügen Sie jeder Map ein ObjectTransformer-Plug-in hinzu. Wenn Sie kein ObjectTransformer-Plug-in bereitstellen, werden bis zu 60-70 % der gesamten Prozessorzeit mit der Serialisierung und das Kopieren von Einträgen verbracht.

### Zweck

Wenn Sie das ObjectTransformer-Plug-in verwenden, können Ihre Anwendung angepasste Methoden für die folgenden Operationen bereitstellen:

- Serialisierung und Entserialisierung des Schlüssels für einen Eintrag,
- Serialisierung und Entserialisierung des Werts für einen Eintrag,
- Kopieren eines Schlüssels oder eines Werts für einen Eintrag.

Wenn kein ObjectTransformer-Plug-in bereitgestellt wird, müssen Sie in der Lage sein, die Schlüssel und Werte zu serialisieren, weil das ObjectGrid eine Serialisierungs- und Entserialisierungsfolge verwendet, um die Objekte zu kopieren. Diese Methode ist kostenintensiv, und deshalb sollten Sie ein ObjectTransformer-Plug-in verwenden, wenn die Leistung kritisch ist. Der Kopiervorgang findet statt, wenn eine Anwendung ein Objekt in einer Transaktion zu ersten Mal sucht. Sie können diesen Kopiervorgang vermeiden, indem Sie den Kopiermodus der Map auf NO\_COPY setzen. Mit der Kopiermoduseinstellung COPY\_ON\_READ können Sie die Anzahl der Kopiervorgänge reduzieren. Optimieren Sie die Kopieroperation, wenn diese von der Anwendung benötigt wird, indem Sie eine angepasste Kopiermethode in diesem Plug-in bereitstellen. Ein solches Plug-in kann den Kopieraufwand von 65–70 % auf 2/3 % der gesamten Prozessorzeit reduzieren.

Die Standardimplementierungen der Methoden "copyKey" und "copyValue" versuchen zuerst, die Methode "clone" zu verwenden, sofern diese verfügbar ist. Wenn keine Implementierung der Methode "clone" bereitgestellt wird, wird standardmäßig mit Serialisierung gearbeitet.

Die Objektserialisierung wird auch direkt verwendet, wenn eXtreme Scale im verteilten Modus ausgeführt wird. LogSequence verwendet das ObjectTransformer-Plug-in für die Serialisierung von Schlüsseln und Werten, bevor die Änderungen an die Peers im ObjectGrid übertragen werden. Sie müssen sorgfältig vorgehen, wenn Sie eine angepasste Serialisierungsmethode als Ersatz für die integrierte JDK-Serialisierung bereitstellen. Die Versionssteuerung von Objekten ist eine komplexes Thema, und es können Probleme mit der Versionskompatibilität auftreten, wenn Sie nicht sicherstellen, dass Ihre angepassten Methoden für die Versionssteuerung konzipiert sind.

Die folgende Liste enthält Details zur Serialisierung von Schlüsseln und Werten durch eXtreme Scale:

- Wenn ein angepasstes ObjectTransformer-Plug-in geschrieben und integriert wird, ruft eXtreme Scale Methoden in der Schnittstelle "ObjectTransformer" auf, um Schlüssel und Werte zu serialisieren und Kopien von Objektschlüsseln und -werten abzurufen.
- Wenn kein angepasstes ObjectTransformer-Plug-in verwendet wird, führt eXtreme Scale die Serialisierung und Entserialisierung von Werten nach dem Standardverfahren durch. Wenn das Standard-Plug-in verwendet wird, wird jedes Objekt als extern bereitstellbares (Externalizable) oder als serialisierbares (Serializable) Objekt implementiert.
  - Wenn das Objekt die Schnittstelle "Externalizable" unterstützt, wird die Methode "writeExternal" aufgerufen. Objekte, die als extern bereitstellbare Objekte implementiert werden, tragen zu einer besseren Leistung bei.
  - Wenn das Objekt die Schnittstelle "Externalizable" nicht unterstützt und die Schnittstelle "Serializable" implementiert, wird das Objekt mit der Methode "ObjectOutputStream" gespeichert.

## ObjectTransformer-Objekt schreiben

Ein ObjectTransformer-Objekt muss die Schnittstelle "ObjectTransformer" implementieren und die folgenden allgemeinen Konventionen für ObjectGrid-Plug-ins einhalten.

## Schnittstelle "ObjectTransformer" verwenden

Es gibt zwei Ansätze (programmgesteuerte Konfiguration und XML-Konfiguration), mit denen ein ObjectTransformer-Objekt der BackingMap-Konfiguration hinzugefügt werden kann. Beide Ansätze werden in den folgenden Abschnitten beschrieben:

## ObjectTransformer durch XML-Konfiguration integrieren

Angenommen, der Klassenname der ObjectTransformer-Implementierung ist "com.company.org.MyObjectTransformer". Diese Klasse implementiert die Schnittstelle "ObjectTransformer". Eine ObjectTransformer-Implementierung kann mit der folgenden XML konfiguriert werden:

```
<?xml version="1.0" encoding="UTF-8"?>
<objectGridConfig xmlns:xsi="http://www.w3.org/2001/XMLSchema-instance"
  xsi:schemaLocation="http://ibm.com/ws/objectgrid/config ../objectGrid.xsd"
  xmlns="http://ibm.com/ws/objectgrid/config">
  <objectGrids>
    <objectGrid name="myGrid">
      <backingMap name="myMap" pluginCollectionRef="myMap" />
    </objectGrid>
  </objectGrids>

  <backingMapPluginCollections>
    <backingMapPluginCollection id="myMap">
      <bean id="ObjectTransformer" className="com.company.org.MyObjectTransformer" />
    </backingMapPluginCollection>
  </backingMapPluginCollections>
</objectGridConfig>
```

## ObjectTransformer-Objekt programmgesteuert integrieren

Das folgende Code-Snippet erstellt das angepasste ObjectTransformer-Objekt und fügt es einer BackingMap hinzu:

```
ObjectGridManager objectGridManager = ObjectGridManagerFactory.getObjectGridManager();
ObjectGrid myGrid = objectGridManager.createObjectGrid("myGrid", false);
BackingMap backingMap = myGrid.getMap("myMap");
MyObjectTransformer myObjectTransformer = new MyObjectTransformer();
backingMap.setObjectTransformer(myObjectTransformer);
```

## Einsatzszenarios für ObjectTransformer

Sie können das ObjectTransformer-Plug-in in den folgenden Situationen verwenden:

- Nicht serialisierbares Objekt
- Serialisierbares Objekt, aber Serialisierungsleistung muss verbessert werden
- Schlüssel- oder Wertkopie

Im folgenden Beispiel wird ObjectGrid verwendet, um die Klasse "Stock" zu speichern:

```
/**
 * Stock-Objekt für ObjectGrid-Demonstration
 *
 */
public class Stock implements Cloneable {
    String ticket;
```

```

double price;
String company;
String description;
int serialNumber;
long lastTransactionTime;
/**
 * @return Gibt die Beschreibung zurück.
 */
public String getDescription() {
    return description;
}
/**
 * @param description Die festzulegende Beschreibung.
 */
public void setDescription(String description) {
    this.description = description;
}
/**
 * @return Gibt den lastTransactionTime-Wert zurück.
 */
public long getLastTransactionTime() {
    return lastTransactionTime;
}
/**
 * @param lastTransactionTime Der festzulegende lastTransactionTime-Wert.
 */
public void setLastTransactionTime(long lastTransactionTime) {
    this.lastTransactionTime = lastTransactionTime;
}
/**
 * @return Gibt den Preis zurück.
 */
public double getPrice() {
    return price;
}
/**
 * @param price Der festzulegende Preis.
 */
public void setPrice(double price) {
    this.price = price;
}
/**
 * @return Gibt den serialNumber-Wert zurück.
 */
public int getSerialNumber() {
    return serialNumber;
}
/**
 * @param serialNumber Der festzulegende serialNumber-Wert.
 */
public void setSerialNumber(int serialNumber) {
    this.serialNumber = serialNumber;
}
/**
 * @return Gibt das Ticket zurück.
 */
public String getTicket() {
    return ticket;
}
/**
 * @param ticket Das festzulegende Ticket.
 */
public void setTicket(String ticket) {
    this.ticket = ticket;
}
/**
 * @return Gibt die Firma zurück.
 */
public String getCompany() {
    return company;
}
/**
 * @param company Die festzulegende Firma.
 */
public void setCompany(String company) {
    this.company = company;
}
}
//clone
public Object clone() throws CloneNotSupportedException

```

```

    {
        return super.clone();
    }
}

```

Sie können eine angepasste ObjectTransformer-Klasse für die Klasse "Stock" schreiben:

```

/**
 * Angepasste Implementierung des ObjectGrid-ObjectTransformer-Plug-ins für Stock-Objekt
 */
public class MyStockObjectTransformer implements ObjectTransformer {
    /* (keine Javadoc)
    * @see
    * com.ibm.websphere.objectgrid.plugins.ObjectTransformer#serializeKey
    * (java.lang.Object,
    * java.io.ObjectOutputStream)
    */
    public void serializeKey(Object key, ObjectOutputStream stream) throws IOException {
        String ticket= (String) key;
        stream.writeUTF(ticket);
    }

    /* (keine Javadoc)
    * @see com.ibm.websphere.objectgrid.plugins.
    ObjectTransformer#serializeValue(java.lang.Object,
    java.io.ObjectOutputStream)
    */
    public void serializeValue(Object value, ObjectOutputStream stream) throws IOException {
        Stock stock= (Stock) value;
        stream.writeUTF(stock.getTicket());
        stream.writeUTF(stock.getCompany());
        stream.writeUTF(stock.getDescription());
        stream.writeDouble(stock.getPrice());
        stream.writeLong(stock.getLastTransactionTime());
        stream.writeInt(stock.getSerialNumber());
    }

    /* (keine Javadoc)
    * @see com.ibm.websphere.objectgrid.plugins.
    ObjectTransformer#inflateKey(java.io.ObjectInputStream)
    */
    public Object inflateKey(ObjectInputStream stream) throws IOException, ClassNotFoundException {
        String ticket=stream.readUTF();
        return ticket;
    }

    /* (keine Javadoc)
    * @see com.ibm.websphere.objectgrid.plugins.
    ObjectTransformer#inflateValue(java.io.ObjectInputStream)
    */
    public Object inflateValue(ObjectInputStream stream) throws IOException, ClassNotFoundException {
        Stock stock=new Stock();
        stock.setTicket(stream.readUTF());
        stock.setCompany(stream.readUTF());
        stock.setDescription(stream.readUTF());
        stock.setPrice(stream.readDouble());
        stock.setLastTransactionTime(stream.readLong());
        stock.setSerialNumber(stream.readInt());
        return stock;
    }

    /* (keine Javadoc)
    * @see com.ibm.websphere.objectgrid.plugins.
    ObjectTransformer#copyValue(java.lang.Object)
    */
    public Object copyValue(Object value) {
        Stock stock = (Stock) value;
        try {
            return stock.clone();
        }
        catch (CloneNotSupportedException e)
        {
            // Ausnahmenachricht anzeigen
        }
    }

    /* (keine Javadoc)
    * @see com.ibm.websphere.objectgrid.plugins.
    ObjectTransformer#copyKey(java.lang.Object)
    */
    public Object copyKey(Object key) {
        String ticket=(String) key;
        String ticketCopy= new String (ticket);
        return ticketCopy;
    }
}

```

Integrieren Sie anschließend diese angepasste Klasse "MyStockObjectTransformer" in die BackingMap:

```
ObjectGridManager ogf=ObjectGridManagerFactory.getObjectGridManager();
ObjectGrid og = ogf.getObjectGrid("NYSE");
BackingMap bm = og.defineMap("NYSEStocks");
MyStockObjectTransformer ot = new MyStockObjectTransformer();
bm.setObjectTransformer(ot);
```

---

## WebSphereTransactionCallback-Plug-in

Wenn Sie das WebSphereTransactionCallback-Plug-in verwenden, können Unternehmensanwendungen, die in einer Umgebung von WebSphere Application Server ausgeführt werden, ObjectGrid-Transaktionen verwalten.

Wenn Sie eine ObjectGrid-Sitzung in einer Methode verwenden, die für die Verwendung containerverwalteter Transaktionen konfiguriert ist, wird die ObjectGrid-Transaktion vom Unternehmenscontainer gestartet, festgeschrieben oder rückgängig gemacht. Wenn Sie JTA-UserTransaction-Objekte verwenden, wird die ObjectGrid-Transaktion automatisch vom UserTransaction-Objekt verwaltet.

Eine ausführliche Beschreibung der Implementierung dieses Plug-ins finden Sie im Abschnitt „Externe Transaktionsmanager“ auf Seite 153.

**Anmerkung:** ObjectGrid unterstützt keine zweiphasigen XA-Transaktionen. Dieses Plug-in registriert die ObjectGrid-Transaktion nicht beim Transaktionsmanager. Wenn das ObjectGrid nicht festgeschrieben werden kann, werden deshalb alle anderen Ressourcen, die von der XA-Transaktion verwaltet werden, nicht rückgängig gemacht.

### WebSphereTransactionCallback-Plug-in aktivieren

Sie können den WebSphereTransactionCallback in der ObjectGrid-Konfiguration durch programmgesteuerte Konfiguration oder XML-Konfiguration aktivieren.

### XML-Konfigurationsansatz für die Integration des WebSphereTransactionCallback-Objekts

Die folgende XML-Konfiguration erstellt das WebSphereTransactionCallback-Objekt und fügt es einem ObjectGrid hinzu. Der folgende Text muss in der Datei myGrid.xml enthalten sein:

```
<?xml version="1.0" encoding="UTF-8"?>
<objectGridConfig xmlns:xsi="http://www.w3.org/2001/XMLSchema-instance"
  xsi:schemaLocation="http://ibm.com/ws/objectgrid/config ../objectGrid.xsd"
  xmlns="http://ibm.com/ws/objectgrid/config">
  <objectGrids>
    <objectGrid name="myGrid">
      <bean id="TransactionCallback" className=
        "com.ibm.websphere.objectgrid.plugins.builtins.WebSphereTransactionCallback" />
    </objectGrid>
  </objectGrids>
</objectGridConfig>
```

### WebSphereTransactionCallback-Objekt über das Programm integrieren

Das folgende Code-Snippet erstellt das WebSphereTransactionCallback-Objekt und fügt es einem ObjectGrid hinzu:

```
ObjectGridManager objectGridManager = ObjectGridManagerFactory.getObjectGridManager();
ObjectGrid myGrid = objectGridManager.createObjectGrid("myGrid", false);
WebSphereTransactionCallback wsTxCallback= new WebSphereTransactionCallback ();
myGrid.setTransactionCallback(wsTxCallback);
```

---

## Kapitel 5. Integration mit dem Spring-Framework

Spring ist ein vielfach eingesetztes Framework für die Entwicklung von Java-Anwendungen. WebSphere eXtreme Scale unterstützt den Einsatz von Spring für die Verwaltung von eXtreme-Scale-Transaktionen und die Konfiguration der Clients und Server, aus denen sich das implementierte speicherinterne Daten-Grid zusammensetzt.

### Über Spring verwaltete native Transaktionen

Spring unterstützt containerverwaltete Transaktionen, die einem Java-EE-Anwendungsserver gleichen. Der Spring-Mechanismus kann jedoch in verschiedene Implementierungen integriert werden. WebSphere eXtreme Scale unterstützt die Integration eines Transaktionsmanagers, der Spring die Verwaltung der Lebenszyklen von ObjectGrid-Transaktionen ermöglicht. Weitere Einzelheiten finden Sie in den Informationen zu nativen Transaktionen im *Programmierhandbuch*.

### Über Spring verwaltete Erweiterungs-Beans und Unterstützung von Namespaces

Spring kann auch in eXtreme Scale integriert werden, um die Definition von Spring-Beans für Erweiterungspunkte und Plug-ins zu ermöglichen. Dieses Feature unterstützt fortgeschrittene Konfigurationen und mehr Flexibilität für die Konfiguration der Erweiterungspunkte.

Zusätzlich zu den über Spring verwalteten Erweiterungs-Beans stellt eXtreme Scale einen Spring-Namespace mit dem Namen "objectgrid" bereit. Beans und integrierte Implementierungen sind in diesem Namespace vordefiniert. Dies erleichtert Benutzern die Konfiguration von eXtreme Scale. Weitere Einzelheiten zu diesen Themen und ein Beispiel zum Starten eines eXtreme-Scale-Servers mit Spring-Konfigurationen finden Sie im Abschnitt „Spring-Erweiterungs-Beans und Unterstützung von Namespaces“ auf Seite 208.

### Unterstützung des Geltungsbereichs "Shard"

Mit der traditionellen Spring-Konfiguration kann eine ObjectGrid-Bean ein Singleton oder ein Prototyp sein. ObjectGrid unterstützt außerdem einen neuen Geltungsbereich, den Geltungsbereich "Shard". Wenn eine Bean mit dem Geltungsbereich "Shard" definiert wird, kann pro Shard nur eine einzige Bean erstellt werden. Auf alle Anforderungen für Beans mit IDs, die der Bean-Definition im selben Shard entsprechen, wird eine bestimmte Bean-Instanz vom Spring-Container zurückgegeben.

Das folgende Beispiel zeigt eine definierte `com.ibm.ws.objectgrid.jpa.plugins.JPAPropFactoryImpl`-Bean mit dem Geltungsbereich "Shard". Deshalb wird nur eine einzige Instanz der Klasse "JPAPropFactoryImpl" pro Shard erstellt.

```
<bean id="jpaPropFactory" class="com.ibm.ws.objectgrid.jpa.plugins.JPAPropFactoryImpl" scope="shard" />
```

### Spring Web Flow

Spring Web Flow speichert seinen Sitzungsstatus standardmäßig in der HTTP-Sitzung. Wenn in einer Webanwendung die Verwendung von eXtreme Scale für die Sitzungsverwaltung konfiguriert ist, verwendet Spring automatisch eXtreme Scale

für das Speichern seines Status und übernimmt die Fehlertoleranz der Sitzung.

## Packen

Die Spring-Erweiterungen für eXtreme Scale sind in der Datei `ogspring.jar` enthalten. Diese JAR-Datei (Java-Archiv) muss im Klassenpfad enthalten sein, damit die Spring-Unterstützung funktioniert. Wenn eine Java-EE-Anwendung, die in WebSphere Application Server Network Deployment mit der Erweiterung WebSphere Extended Deployment ausgeführt wird, muss die Anwendung die Datei `spring.jar` und die zugehörigen Dateien in die EAR-Module stellen. Außerdem müssen Sie die Datei `ogspring.jar` an dieselbe Position kopieren.

---

## Native Transaktionen

Spring ist ein vielfach eingesetztes Framework für die Entwicklung von Java-Anwendungen. WebSphere eXtreme Scale unterstützt den Einsatz von Spring für die Verwaltung von eXtreme-Scale-Transaktionen und die Konfiguration von eXtreme-Scale-Clients und -Servern.

### Native Transaktionen mit WebSphere eXtreme Scale

Spring stellt containerverwaltete Transaktionen im Stil eines Java-EE-Anwendungsservers (Java Platform, Enterprise Edition) bereit, hat aber den Vorteil, dass in den Spring-Mechanismus verschiedene Implementierungen integriert werden können. In diesem Abschnitt wird ein eXtreme Scale Plattform Transaction Manager beschrieben, der mit Spring verwendet werden kann. Dies ermöglicht Programmierern, ihre POJOs (Plain Old Java Objects) zu annotieren und anschließend Session-Objekte über Spring von eXtreme Scale abrufen und eXtreme-Scale-Transaktionen starten, festschreiben, rückgängig machen, aussetzen und fortsetzen zu lassen. Eine Erläuterung des Spring-Transaktionsmechanismus ist in diesem Abschnitt nicht enthalten. Spring-Transaktionen sind in Kapitel 9 des Benutzerhandbuchs von Spring beschrieben. In diesem Abschnitt wird beschrieben, wie ein eXtreme-Scale-Transaktionsmanager erstellt und mit annotierten POJOs verwendet wird. Es wird auch erläutert, wie dieser Ansatz mit einer Client- oder lokalen eXtreme-Scale-Umgebung und mit einer kollokierten Data-Grid-Anwendung verwendet wird.

### Transaktionsmanager erstellen

eXtreme Scale stellt eine Implementierung eines Spring-PlatformTransactionManager bereit. Dieser Manager kann POJOs, die von Spring verwaltet werden, verwaltete eXtreme-Scale-Sitzungen bereitstellen. Mit Annotationen verwaltet Spring diese Sitzungen für die POJOs in einem Transaktionslebenszyklus. Das folgende XML-Snippet veranschaulicht, wie ein Transaktionsmanager erstellt wird:

```
<aop:aspectj-autoproxy/>
<tx:annotation-driven transaction-manager="transactionManager"/>

<bean id="ObjectGridManager"
      class="com.ibm.websphere.objectgrid.ObjectGridManagerFactory"
      factory-method="getObjectGridManager"/>

<bean id="ObjectGrid"
      factory-bean="ObjectGridManager"
      factory-method="createObjectGrid"/>

<bean id="transactionManager"
      class="com.ibm.websphere.objectgrid.spring.ObjectGridSpringFactory"
      factory-method="getLocalPlatformTransactionManager"/>
</bean>

<bean id="Service" class="com.ibm.websphere.objectgrid.spring.test.TestService">
  <property name="txManager" ref="transactionManager"/>
</bean>
```



Dieses Snippet zeigt, dass die Bean "transactionManager" deklariert und mit der Bean "Service", die Spring-Transaktionen verwendet, verbunden wird. Dies wird mit Hilfe von Annotationen veranschaulicht, und dies ist auch der Grund für die Klausel "tx:annotation" am Anfang.

## ObjectGrid-Session-Objekt für die aktuelle Spring-Transaktion anfordern

Ein POJO, dessen Methoden von Spring verwaltet werden, kann jetzt das ObjectGrid-Session-Objekt für die aktuelle Transaktion wie folgt anfordern:

```
Session s = txManager.getSession();
```

Diese Methode gibt die Sitzung zurück, die das POJO verwenden soll. An derselben Transaktion beteiligte Beans empfangen dasselbe Session-Objekt, wenn sie diese Methode aufrufen. Spring startet das Session-Objekt automatisch und ruft auch bei Bedarf automatisch die Methode "commit" oder "rollback" auf. Sie können auch ein ObjectGrid-EntityManager-Objekt anfordern, indem Sie einfach "getEntityManager" über das Session-Objekt aufrufen.

## Muster-POJO mit Annotationen

Im Folgenden sehen Sie ein POJO, das Annotationen verwendet, um seine Transaktionsabsichten für Spring zu deklarieren. Sie sehen, dass die Klasse eine Annotation auf Klassenebene hat, die anzeigt, dass alle Methoden standardmäßig die Transaktionssemantik REQUIRED verwenden sollen. Die Klasse implementiert eine Schnittstelle mit einer Methode für alle Methoden in der Klasse. Dies ist erforderlich, damit Spring AOP funktioniert, wenn kein Bytecode-Weaving möglich ist. Die Klasse hat eine Instanzvariable "txManager", die mit dem ObjectGrid-Transaktionsmanager über die Spring-XML-Datei verbunden wird. Jede Methode ruft einfach die Methode "txManager.getSession" auf, um das für die Methode zu verwendende Session-Objekt anzufordern. Die Methode "queryNewTx" ist annotiert und zeigt an, dass die Semantik REQUIRES\_NEW verwendet werden soll. Dies bedeutet, dass alle vorhandenen Transaktionen ausgesetzt werden und eine neue unabhängige Transaktion für diese Methode erstellt wird.

```
@Transactional(propagation=Propagation.REQUIRED)
public class TestService implements ITestService
{
    SpringLocalTxManager txManager;

    public TestService()
    {
    }

    public void initialize()
        throws ObjectGridException
    {
        Session s = txManager.getSession();
        ObjectMap m = s.getMap("TEST");
        m.insert("Hello", "Billy");
    }

    public void update(String updatedValue)
        throws ObjectGridException
    {
        Session s = txManager.getSession();
        System.out.println("Update using " + s);
        ObjectMap m = s.getMap("TEST");
        String v = (String)m.get("Hello");
        m.update("Hello", updatedValue);
    }

    public String query()
        throws ObjectGridException
    {
        Session s = txManager.getSession();
        System.out.println("Query using " + s);
        ObjectMap m = s.getMap("TEST");
        return (String)m.get("Hello");
    }
}
```

```

    }

    @Transactional(propagation=Propagation.REQUIRES_NEW)
    public String queryNewTx()
        throws ObjectGridException
    {
        Session s = txManager.getSession();
        System.out.println("QueryTX using " + s);
        ObjectMap m = s.getMap("TEST");
        return (String)m.get("Hello");
    }

    public void testRequiresNew(ITestService bean)
        throws ObjectGridException
    {
        update("1");
        String txValue = bean.query();
        if(!txValue.equals("1"))
        {
            System.out.println("Requires didnt work");
            throw new IllegalStateException("requires didn't work");
        }
        String committedValue = bean.queryNewTx();
        if(committedValue.equals("1"))
        {
            System.out.println("Requires new didnt work");
            throw new IllegalStateException("requires new didn't work");
        }
    }

    public SpringLocalTxManager getTxManager() {
        return txManager;
    }

    public void setTxManager(SpringLocalTxManager txManager) {
        this.txManager = txManager;
    }
}

```

## ObjectGrid-Instanz für einen Thread festlegen

In einer einzelnen Java Virtual Machine (JVM) können viele ObjectGrid-Instanzen ausgeführt werden. Jedes primäre Shard in einer JVM hat eine eigene ObjectGrid-Instanz. Eine JVM, die als Client für ein fernes ObjectGrid auftritt, verwendet eine ObjectGrid-Instanz, die über das ClientClusterContext-Objekt der Methode "connect" zurückgegeben wird, um mit diesem Grid zu interagieren. Vor dem Aufruf einer Methode in einem POJO über Spring-Transaktionen für das ObjectGrid muss der Thread mit der zu verwendenden ObjectGrid-Instanz verbunden werden. Die TransactionManager-Instanz hat eine Methode, mit der eine bestimmte ObjectGrid-Instanz angegeben werden kann. Wenn diese Methode angegeben wird, geben alle nachfolgenden Aufrufe von "txManager.getSession" Session-Objekte für diese ObjectGrid-Instanz zurück.

## Einfaches Bootstrapping für Tests

Das folgende Beispiel zeigt eine Musterfunktion "main" für das Testen dieser Funktionalität:

```

ClassPathXmlApplicationContext ctx = new ClassPathXmlApplicationContext(new String[]
    {"applicationContext.xml"});
SpringLocalTxManager txManager = (SpringLocalTxManager)ctx.getBean("transactionManager");
txManager.setObjectGridForThread(og);

ITestService s = (ITestService)ctx.getBean("Service");
s.initialize();
assertEquals(s.query(), "Billy");
s.update("Bobby");
assertEquals(s.query(), "Bobby");
System.out.println("Requires new test");
s.testRequiresNew(s);
assertEquals(s.query(), "1");

```

Hier wird ein Spring-ApplicationContext-Objekt verwendet. Das ApplicationContext-Objekt wird verwendet, um eine Referenz auf das txManager-Objekt anzufordern und um ein in diesem Thread zu verwendendes ObjectGrid anzugeben. Anschließend fordert der Code eine Referenz auf den Service an und ruft Metho-

den in diesem Service auf. Jeder Methodenaufruf auf dieser Ebene bewirkt, dass Spring ein Session-Objekt erstellt und begin/commit-Aufrufe um den Methodenaufruf herum absetzt. Alle Ausnahmen führen zu einem Rollback.

## Neue eXtreme-Scale-Schnittstellen

Es gibt eine einzelne neue Schnittstelle, die Schnittstelle "SpringLocalTxManager". Diese Schnittstelle wird von ObjectGrid Platform Transaction Manager implementiert und enthält alle allgemein zugänglichen Schnittstellen. Die Methoden in dieser Schnittstelle sind für die Auswahl der ObjectGrid-Instanz, die in einem Thread verwendet werden soll, und für das Abrufen eines Session-Objekts für den Thread bestimmt. Alle POJOs, die lokale ObjectGrid-Transaktionen verwenden, müssen über eine Referenz auf diese Managerinstanz injiziert werden, und es muss nur eine einzige Instanz erstellt werden, d. h., der Geltungsbereich muss "Singleton" sein. Diese Instanz wird mit einer statischen Methode in ObjectGridSpringFactory.getLocalPlatformTransactionManager() erstellt.

## eXtreme Scale für JTA- und globale Transaktionen

eXtreme Scale unterstützt aus verschiedenen Gründen, die sich hauptsächlich auf die Skalierbarkeit beziehen, weder JTA noch zweiphasige Festschreibung. Deshalb interagiert ObjectGrid bis auf einen letzten einphasigen Teilnehmer nicht in globalen XA- oder JTA-Transaktionen. Dieser Plattformmanager soll lokale ObjectGrid-Transaktionen für Spring-Entwickler so einfach wie möglich machen.

## Über Spring verwaltete Erweiterungs-Beans

ObjectGrid enthält eine Methode, mit der POJOs als Erweiterungspunkte in der Datei "objectgrid.xml" verwendet werden können. ObjectGrid bietet die Möglichkeit, die Beans zu benennen und anschließend den Klassennamen anzugeben. ObjectGrid erstellt normalerweise Instanzen der angegebenen Klasse und verwendet diese Instanzen als Plug-in. ObjectGrid kann jetzt Spring als Bean-Factory für den Abruf von Instanzen dieser Plug-in-Objekte einsetzen. Wenn eine Anwendung Spring verwendet, müssen solche POJOs gewöhnlich mit dem Rest der Anwendung verbunden werden.

ObjectGrid wurde geändert, um einer Anwendung zu ermöglichen, eine Spring-Bean-Factory-Instanz für ein bestimmtes benanntes ObjectGrid zu registrieren. Die Anwendung muss eine Instanz von BeanFactory oder einen Spring-Anwendungskontext erstellen und diese anschließend bei ObjectGrid mit der folgenden statischen Methode registrieren:

```
void registerSpringBeanFactoryAdapter(String objectGridName, Object springBeanFactory)
```

Diese Methode gibt Folgendes an: Wenn ObjectGrid eine Erweiterungs-Bean (z. B. ObjectTransformer, Loader, TransactionCallback usw.) findet, deren Klassename mit dem Präfix "{spring}" beginnt, wird der Rest des Namens als Name der Spring-Bean verwendet und die Bean-Instanz über die Spring-Bean-Factory angefordert. ObjectGrid kann eine Spring-Bean-Factory auch über eine Standard-Spring-XML-Konfigurationsdatei erstellen. Wenn keine Bean-Factory für ein bestimmtes ObjectGrid registriert wurde, versucht ObjectGrid eine XML-Datei mit dem Namen "ObjectGridName'\_spring.xml" zu finden. Wenn Ihr Grid beispielsweise den Namen GRID hat, hat die XML-Datei den Namen "/GRID\_spring.xml" und sollte sich im Klassenpfad des Stammpakets befinden. Ist diese Datei vorhanden, erstellt ObjectGrid ein ApplicationContext-Objekt über diese Datei und Beans über diese Bean-Factory. Ein Beispielklassenname ist:

```
"{spring}MyLoaderBean"
```

Dies veranlasst ObjectGrid, Spring nach einer Bean mit dem Namen "MyLoader-Bean" abzufragen. Dieser Ansatz kann verwendet werden, um über Spring verwaltete POJOs für jeden Erweiterungspunkt anzugeben, solange die Bean-Factory zuvor registriert wurde. Die Spring-Erweiterungen von ObjectGrid sind in der Datei "ogspring.jar" enthalten. Diese JAR-Datei muss im Klassenpfad enthalten sein, damit die Spring-Unterstützung funktioniert. Wenn eine Java-EE-Anwendung, die in einer mit Extended Deployment erweiterten Network-Deployment-Umgebung ausgeführt wird, muss die Anwendung die Datei "spring.jar" und die zugehörigen Dateien in EAR-Module packen. Die Datei "ogspring.jar" muss an dieselbe Position kopiert werden.

## Spring Webflow

Spring Webflow speichert seinen Sitzungsstatus standardmäßig in der HTTP-Sitzung. Wenn in einer Webanwendung die Verwendung von ObjectGrid für die Sitzungsverwaltung konfiguriert ist, verwendet Spring automatisch eXtreme Scale für das Speichern seines Status und übernimmt die Fehlertoleranz der Sitzung.

---

## Spring-Erweiterungs-Beans und Unterstützung von Namespaces

WebSphere eXtreme Scale stellt ein Feature für die Deklaration von POJOs (Plain Old Java Object) als Erweiterungspunkte in der Datei objectgrid.xml und eine Methode für die Benennung der Beans und die anschließende Spezifikation des Klassennamens bereit. Normalerweise werden Instanzen der angegebenen Klasse erstellt, und diese Objekte werden dann als Plug-ins verwendet. Jetzt kann eXtreme Scale das Abrufen von Instanzen dieser Plug-in-Objekte an Spring delegieren. Wenn eine Anwendung Spring verwendet, müssen solche POJOs gewöhnlich mit dem Rest der Anwendung verbunden werden.

In manchen Fällen müssen Sie Spring für die Konfiguration bestimmter Plug-in-Objekte verwenden. Verwenden Sie die folgende Konfiguration als Beispiel:

```
<objectGrid name="Grid">
  <bean id="TransactionCallback" className="com.ibm.websphere.objectgrid.jpa.JPATxCallback">
    <property name="persistenceUnitName" type="java.lang.String" value="employeePU" />
  </bean>
  ...
</objectGrid>
```

Die integrierte TransactionCallback-Implementierung, die Klasse "com.ibm.websphere.objectgrid.jpa.JPATxCallback", ist als TransactionCallback-Klasse konfiguriert. Diese Klasse wird, wie im vorherigen Beispiel gezeigt, mit einer einzigen Eigenschaft, der Eigenschaft "persistenceUnitName", konfiguriert. Die Klasse "JPATxCallback" besitzt auch das Attribut "JPAPropertyFactory", das den Typ "java.lang.Object" hat. Die ObjectGrid-XML-Konfiguration unterstützt diesen Typ von Konfiguration nicht.

Die Integration von Spring in eXtreme Scale löst dieses Problem, indem die Bean-Erstellung an das Spring-Framework delegiert wird. Die überarbeitete Konfiguration folgt:

```
<objectGrid name="Grid">
  <bean id="TransactionCallback" className="{spring}jpaTxCallback"/>
  ...
</objectGrid>
```

Die Spring-Datei für das Objekt "Grid" enthält die folgenden Informationen:

```
<bean id="jpaTxCallback" class="com.ibm.websphere.objectgrid.jpa.JPATxCallback" scope="shard">
  <property name="persistenceUnitName" value="employeeEMPU"/>
  <property name="JPAPropertyFactory" ref="jpaPropFactory"/>
</bean>
```

```
<bean id="jpaPropFactory" class="com.ibm.ws.objectgrid.jpa.plugins.JPAPropFactoryImpl" scope="shard">
</bean>
```

Hier ist TransactionCallback mit {spring}jpaTxCallback angegeben, und die Beans "jpaTxCallback" und "jpaPropFactory" werden in der Spring-Datei, wie im vorherigen Beispiel gezeigt, konfiguriert. Mit der Spring-Konfiguration kann eine JPAPropertyFactory-Bean als Parameter des JPATxCallback-Objekts konfiguriert werden.

### Standard-Spring-Bean-Factory

Wenn eXtreme Scale ein Plug-in oder eine Erweiterungs-Bean (z. B. ObjectTransformer, Loader, TransactionCallback usw.) mit einem classname-Wert findet, der mit dem Präfix {spring} beginnt, verwendet eXtreme Scale den restlichen Teil des Namens als Namen für die Spring-Bean und ruft die Bean-Instanz über die Spring-Bean-Factory ab.

Wenn keine Bean-Factory für ein bestimmtes ObjectGrid registriert wurde, wird standardmäßig versucht, eine Datei ObjectGridName\_spring.xml zu finden. Wenn Ihr Grid beispielsweise "Grid" heißt, hat die XML-Datei den Namen /Grid\_spring.xml. Diese Datei muss im Klassenpfad oder in einem Verzeichnis META-INF im Klassenpfad enthalten sein. Wenn diese Datei gefunden wird, erstellt eXtreme Scale über diese Datei einen Anwendungskontext und über diese Bean-Factory die Beans.

### Angepasste Spring-Bean-Factory

WebSphere eXtreme Scale stellt auch eine API "ObjectGridSpringFactory" bereit, über die eine Spring-Bean-Factory-Instanz für ein bestimmtes ObjectGrid registriert werden kann. Diese API registriert eine Instanz von BeanFactory mit der folgenden statischen Methode bei eXtreme Scale:

```
void registerSpringBeanFactoryAdapter(String objectGridName, Object
springBeanFactory)
```

### Unterstützung von Namespaces

Seit Version 2.0 hat Spring einen Mechanismus für schemabasierte Erweiterungen für das Spring-XML-Basisformat für die Definition und Konfiguration von Beans. ObjectGrid verwendet dieses neue Feature, um ObjectGrid-Beans zu definieren und zu konfigurieren. Mit der Spring-XML-Schemaerweiterung sind einige der integrierten Implementierungen von eXtreme-Scale-Plug-ins und einige ObjectGrid-Beans im Namespace "objectgrid" vordefiniert. Wenn Sie die Spring-Konfigurationsdateien schreiben, müssen Sie den vollständigen Klassennamen der integrierten Implementierungen nicht angeben. Stattdessen können Sie die vordefinierten Beans referenzieren.

Außerdem sinkt mit den Attributen der Beans, die im XML-Schema definiert sind, das Risiko, dass falsche Attributnamen angegeben werden. Die XML-Validierung, die auf dem XML-Schema basiert, kann diese Art von Fehlern früher im Entwicklungszyklus abfangen.

Die folgenden Beans sind in den XML-Schemaerweiterungen definiert:

- transactionManager
- register
- server

- catalog
- container
- JPALoader
- JPATxCallback
- JPAEntityLoader
- LRUEvictor
- LFUEvictor
- HashIndex

Diese Beans sind in der XML-Schemadatei "objectgrid.xsd" definiert. Diese XSD-Datei wird als Datei com/ibm/ws/objectgrid/spring/namespace/objectgrid.xsd in der Datei ogspring.jar geliefert. Ausführliche Beschreibungen der XSD-Datei und der in der XSD-Datei definierten Beans finden Sie in den Informationen zur Spring-Deskriptordatei im *Administratorhandbuch*.

Verwenden Sie weiterhin das JPATxCallback-Beispiel aus dem vorherigen Abschnitt. Im vorherigen Abschnitt wurde die JPATxCallback-Bean wie folgt konfiguriert:

```
<bean id="jpaTxCallback" class="com.ibm.websphere.objectgrid.jpa.JPATxCallback" scope="shard">
  <property name="persistenceUnitName" value="employeeEMPU"/>
  <property name="JPAPropertyFactory" ref="jpaPropFactory"/>
</bean>

<bean id="jpaPropFactory" class="com.ibm.ws.objectgrid.jpa.plugins.JPAPropFactoryImpl" scope="shard">
</bean>
```

Mit dem Namespace-Feature kann die Spring-XML-Konfiguration wie folgt geschrieben werden:

```
<objectgrid:JPATxCallback id="jpaTxCallback" persistenceUnitName="employeeEMPU"
  jpaPropertyFactory="jpaPropFactory" />

<bean id="jpaPropFactory" class="com.ibm.ws.objectgrid.jpa.plugins.JPAPropFactoryImpl"
  scope="shard">
</bean>
```

Beachten Sie, dass hier die Klasse "com.ibm.websphere.objectgrid.jpa.JPATxCallback" nicht wie im vorherigen Beispiel angegeben wird, sondern dass die vordefinierte Bean "objectgrid:JPATxCallback" direkt verwendet wird. Wie Sie sehen, ist diese Konfiguration weniger ausführlich und in Bezug auf die Fehlerprüfung komfortabler.

## Containerserver mit Spring-Erweiterungs-Beans starten

In diesem Beispiel wird veranschaulicht, wie Sie einen ObjectGrid-Server über Spring-verwaltete Erweiterungs-Beans und die Unterstützung von Namespaces von ObjectGrid starten.

### ObjectGrid-XML-Datei

Zuerst wird eine sehr einfache ObjectGrid-XML-Datei definiert, die ein einziges ObjectGrid mit dem Namen "Grid" und eine einzige Map mit dem Namen "Test" enthält. Das ObjectGrid hat ein ObjectGridEventListener-Plug-in mit dem Namen "partitionListener" und die Map "Test" ein Evictor-Plug-in mit dem Namen "testLRUEvictor". Beachten Sie, dass sowohl das ObjectGridEventListener-Plug-in als auch das Evictor-Plug-in mit Spring konfiguriert sind, da ihre Namen "{spring}" enthalten:

```
<?xml version="1.0" encoding="UTF-8"?>
<objectGridConfig xmlns:xsi="http://www.w3.org/2001/XMLSchema-instance"
  xsi:schemaLocation="http://ibm.com/ws/objectgrid/config ../objectGrid.xsd"
  xmlns="http://ibm.com/ws/objectgrid/config">
```

```

<objectGrids>
  <objectGrid name="Grid">
    <bean id="ObjectGridEventListener" className="{spring}partitionListener" />
    <backingMap name="Test" pluginCollectionRef="test" />
  </objectGrid>
</objectGrids>

<backingMapPluginCollections>
  <backingMapPluginCollection id="test">
    <bean id="Evictor" className="{spring}testLRUEvictor"/>
  </backingMapPluginCollection>
</backingMapPluginCollections>
</objectGridConfig>

```

## ObjectGrid-XML-Implementierungsdatei

Jetzt wird eine einfache ObjectGrid-XML-Implementierungsdatei erstellt. Sie partitioniert das ObjectGrid in 5 Partitionen, und es ist kein Replikat erforderlich.

```

<?xml version="1.0" encoding="UTF-8"?>
<deploymentPolicy xmlns:xsi="http://www.w3.org/2001/XMLSchema-instance"
  xsi:schemaLocation="http://ibm.com/ws/objectgrid/deploymentPolicy ../deploymentPolicy.xsd"
  xmlns="http://ibm.com/ws/objectgrid/deploymentPolicy">
  <objectgridDeployment objectgridName="Grid">
    <mapSet name="mapSet" numInitialContainers="1" numberOfPartitions="5" minSyncReplicas="0"
      maxSyncReplicas="1" maxAsyncReplicas="0">
      <map ref="Test"/>
    </mapSet>
  </objectgridDeployment>
</deploymentPolicy>

```

## ObjectGrid-Spring-XML-Datei

Jetzt werden die ObjectGrid-Spring-verwalteten Erweiterungs-Beans und die Unterstützung für Namespaces verwendet, um die ObjectGrid-Beans zu konfigurieren. Die Spring-XML-Datei hat den Namen "Grid\_spring.xml". Beachten Sie, dass zwei Schemas in der XML-Datei enthalten sind: spring-beans-2.0.xsd ist für die Verwendung der Spring-verwalteten Beans bestimmt und objectgrid.xsd für die im Namespace "objectgrid" vordefinierten Beans:

```

<beans xmlns="http://www.springframework.org/schema/beans"
  xmlns:xsi="http://www.w3.org/2001/XMLSchema-instance"
  xmlns:aop="http://www.springframework.org/schema/aop"
  xmlns:tx="http://www.springframework.org/schema/tx"
  xmlns:objectgrid="http://www.ibm.com/schema/objectgrid"
  xsi:schemaLocation="
    http://www.ibm.com/schema/objectgrid
    http://www.ibm.com/schema/objectgrid/objectgrid.xsd
    http://www.springframework.org/schema/beans
    http://www.springframework.org/schema/beans/spring-beans-2.0.xsd">

  <objectgrid:register id="ogregister" gridname="Grid"/>

  <objectgrid:server id="server" isCatalog="true" name="server">
    <objectgrid:catalog host="localhost" port="2809"/>
  </objectgrid:server>

  <objectgrid:container id="container"
    objectgridxml="com/ibm/ws/objectgrid/test/springshard/objectgrid.xml"
    deploymentxml="com/ibm/ws/objectgrid/test/springshard/deployment.xml"
    server="server"/>

  <objectgrid:LRUEvictor id="testLRUEvictor" numberOfLRUQueues="31"/>

  <bean id="partitionListener"
    class="com.ibm.websphere.objectgrid.springshard.ShardListener" scope="shard"/>
</beans>

```

Es wurden 6 in dieser Spring-XML-Datei definiert:

1. *objectgrid:register*: Diese Bean registriert die Standard-Bean-Factory für das ObjectGrid "Grid".
2. *objectgrid:server*: Diese Bean definiert einen ObjectGrid-Server mit dem Namen "server". Dieser Server stellt auch Katalogservices bereit, da er eine verschachtelte Bean "objectgrid:catalog" enthält.
3. *objectgrid:catalog*: Diese Bean definiert einen ObjectGrid-Katalogserviceendpunkt, der auf "localhost:2809" gesetzt ist.

4. *objectgrid:container*: Diese Bean definiert einen ObjectGrid-Container mit der angegebenen Objectgrid-XML-Datei und der XML-Implementierungsdatei, die zuvor beschrieben wurden. Die Eigenschaft "server" gibt an, in welchem Server dieser Container ausgeführt wird.
5. *objectgrid:LRUEvictor*: Diese Bean definiert einen LRUEvictor mit der einer LRU-Warteschlangenanzahl von 31.
6. *partitionListener*: Diese Bean definiert ein ShardListener-Plug-in. Diese Klasse ist eine Klasse, die von Benutzern integriert wird. Deshalb kann sie die vordefinierten Beans nicht verwenden. Außerdem hat die Bean den Geltungsbereich "shard", d. h., es gibt nur eine einzige Instanz dieses ShardListeners pro ObjectGrid-Shard.

## Server starten

Das folgende Snippet startet den ObjectGrid-Server, in dem der Containerservice und der Katalogservice ausgeführt werden. Wie zu sehen, ist die einzige Methode, die zum Starten des Servers aufgerufen werden muss, eine Methode "get", mit der ein Bean-Container von der Bean-Factory abgerufen wird. Dies vereinfacht die Programmierungskomplexität, weil die meiste Logik in die Spring-Konfiguration verschoben wird:

```
public class ShardServer extends TestCase
{
    Container container;
    org.springframework.beans.factory.BeanFactory bf;

    public void startServer(String cep)
    {
        try
        {
            bf = new org.springframework.context.support.ClassPathXmlApplicationContext(
                "/com/ibm/ws/objectgrid/test/springshard/Grid_spring.xml", ShardServer.class);
            container = (Container)bf.getBean("container");
        }
        catch (Exception e)
        {
            throw new ObjectGridRuntimeException("Cannot start OG container", e);
        }
    }

    public void stopServer()
    {
        if(container != null)
            container.teardown();
    }
}
```



## Kapitel 6. Sicherheits-API

WebSphere eXtreme Scale verwendet eine offene Sicherheitsarchitektur. Sie bildet das Basissicherheits-Framework für die Authentifizierung, Berechtigung und Transportsicherheit und fordert Benutzer auf, Plug-ins für die Vervollständigung der Informationstechnologie zu implementieren.

Die folgende Abbildung zeigt den Basisablauf der Clientauthentifizierung und -berechtigung für einen eXtreme-Scale-Server.

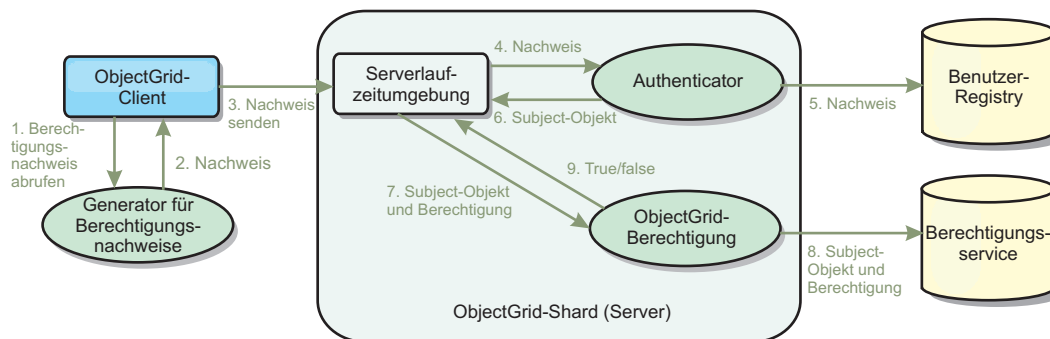


Abbildung 13. Ablauf der Clientauthentifizierung und -berechtigung

Der Authentifizierungsablauf und der Berechtigungsablauf sind im Folgenden beschrieben:

### Authentifizierungsablauf

1. Der Authentifizierungsablauf beginnt damit, dass ein eXtreme-Scale-Client einen Berechtigungsnachweis abrufen. Der Abruf erfolgt über das Plug-in "com.ibm.websphere.objectgrid.security.plugins.CredentialGenerator".
2. Ein CredentialGenerator-Objekt weiß, wie ein gültiger Clientberechtigungs-nachweis, z. B. eine Kombination von Benutzer-ID und Kennwort, ein Kerberos-Ticket usw., generiert wird. Dieser generierte Berechtigungsnachweis wird an den Client zurückgesendet.
3. Nachdem der Client das Credential-Objekt über das CredentialGenerator-Objekt abgerufen hat, wird dieses Credential-Objekt zusammen mit der eXtreme-Scale-Anforderung an den eXtreme-Scale-Server gesendet.
4. Der eXtreme-Scale-Server authentifiziert das Credential-Objekt, bevor er die eXtreme-Scale-Anforderung verarbeitet. Anschließend verwendet der Server das Authenticator-Plug-in, um das Credential-Objekt zu authentifizieren.
5. Das Authenticator-Plug-in stellt eine Schnittstelle zur Benutzer-Registry dar, z. B. einem LDAP-Server (Lightweight Directory Access Protocol) oder einer Benutzer-Registry des Betriebssystems. Das Authenticator-Plug-in tätigt Rückfragen bei der Benutzer-Registry und trifft Authentifizierungsentscheidungen.
6. Wenn die Authentifizierung erfolgreich ist, wird ein Subject-Objekt zurückgegeben, das diesem Client präsentiert wird.

### Berechtigungsablauf

WebSphere eXtreme Scale verwendet einen berechtigungsbasierten Berechtigungsmechanismus und hat verschiedene Berechtigungskategorien, die von verschiedenen Berechtigungsklassen dargestellt werden. Ein com.ibm.webs-

phere.objectgrid.security.MapPermission-Objekt stellt beispielsweise die Berechtigungen zum Lesen, Schreiben, Einfügen, Ungültigmachen und Entfernen der Dateneinträge in einer ObjectMap dar. Da WebSphere eXtreme Scale die JAAS-Berechtigung (Java Authentication and Authorization Service) direkt unterstützt, können Sie JAAS für die Berechtigungsverarbeitung verwenden, indem Sie Berechtigungsrichtlinien festlegen.

Außerdem unterstützt eXtreme Scale angepasste Berechtigungen. Angepasste Berechtigungen werden über das Plug-in "com.ibm.websphere.objectgrid.security.plugins.ObjectGridAuthorization" integriert. Der Ablauf der Kundenberechtigung wird im Folgenden beschrieben.

7. Die Laufzeitumgebung des Servers sendet das Subject-Objekt und die erforderliche Berechtigung an das Berechtigungs-Plug-in.
8. Das Berechtigungs-Plug-in tätigt Rückfragen beim Berechtigungsservice und trifft eine Berechtigungsentscheidung. Wenn die Berechtigung für dieses Subject-Objekt erteilt ist, wird der Wert true zurückgegeben, andernfalls der Wert false.
9. Diese Berechtigungsentscheidung (true oder false) wird an die Laufzeitumgebung des Servers zurückgegeben.

### Sicherheitsimplementierung

In diesem Abschnitt wird beschrieben, wie Sie eine eXtreme-Scale-Implementierung und die Plug-in-Implementierungen programmieren. Der Abschnitt ist nach den verschiedenen Sicherheitsfeatures aufgebaut. In jedem Unterabschnitt erfahren Sie etwas über die relevanten Plug-ins und deren Implementierung. Im Abschnitt zur Authentifizierung lernen Sie, wie Sie eine Verbindung zu einer sicheren Implementierungsumgebung von WebSphere eXtreme Scale herstellen.

*Clientauthentifizierung:* Im Abschnitt zur Clientauthentifizierung wird beschrieben, wie ein eXtreme-Scale-Client einen Berechtigungsnachweis abrufen und wie ein Server den Client authentifiziert. Es wird erläutert, wie ein Client von WebSphere eXtreme Scale eine Verbindung zu einem sicheren Server von WebSphere eXtreme Scale herstellt.

*Berechtigung:* Im Abschnitt zur Berechtigung wird erläutert, wie Sie die Schnittstelle "ObjectGridAuthorization" verwenden, um zusätzlich zur JAAS-Berechtigung eine Kundenberechtigung durchführen.

*Grid-Authentifizierung:* Im Abschnitt zur Grid-Authentifizierung wird beschrieben, wie Sie die Schnittstelle "SecureTokenManager" verwenden, um den geheimen Schlüssel eines Servers sicher zu transportieren.

*JMX-Programmierung (Java Management Extensions):* Wenn der eXtreme-Scale-Server gesichert ist, muss der JMX-Client unter Umständen einen JMX-Berechtigungsnachweis an den Server senden.

---

## Programmierung der Clientauthentifizierung

Für die Authentifizierung stellt WebSphere eXtreme Scale eine Laufzeitumgebung bereit, in der der Berechtigungsnachweis vom Client an die Serverseite gesendet und anschließend das Authenticator-Plug-in für die Authentifizierung der Benutzer aufgerufen wird.

WebSphere eXtreme Scale setzt die Implementierung der folgenden Plug-ins für die Durchführung der Authentifizierung voraus.

- **Credential:** Ein Credential-Plug-in stellt einen Clientberechtigungs-nachweis dar, wie z. B. eine Kombination von Benutzer-ID und Kennwort.
- **CredentialGenerator:** Ein CredentialGenerator-Plug-in stellt eine Berechtigungs-nachweis-Factory für die Generierung des Berechtigungs-nachweises dar.
- **Authenticator:** Ein Authenticator-Plug-in authentifiziert den Clientberechtigungs-nachweis und ruft Clientinformationen ab.

## Credential- und CredentialGenerator-Plug-ins

Wenn ein eXtreme-Scale-Client eine Verbindung zu einem Server herstellt, der eine Authentifizierung voraussetzt, muss der Client einen Clientberechtigungs-nachweis vorlegen. Ein Clientberechtigungs-nachweis wird durch eine Schnittstelle `com.ibm.websphere.objectgrid.security.plugins.Credential` dargestellt. Gültige Clientberechtigungs-nachweise sind eine Kombination von Benutzername und Kennwort, ein Kerberos-Ticket, ein Clientzertifikat oder Daten in einem beliebigen Format, auf das sich Client und Server geeinigt haben. Weitere Einzelheiten finden Sie in den Informationen zur API `Credential` in der API-Dokumentation. Diese Schnittstelle definiert explizit die Methoden `equals(Object)` und `hashCode`. Diese beiden Methoden sind wichtig, weil die authentifizierten Subject-Objekte mit dem Credential-Objekt als Schlüssel auf der Serverseite zwischengespeichert werden. WebSphere eXtreme Scale stellt auch ein Plug-in für die Generierung eines Berechtigungs-nachweises bereit. Dieses Plug-in wird durch die Schnittstelle `com.ibm.websphere.objectgrid.security.plugins.CredentialGenerator` dargestellt und ist hilfreich, wenn der Berechtigungs-nachweis eine Verfallszeit haben kann. In diesem Fall wird die Methode `getCredential` aufgerufen, um einen Berechtigungs-nachweis zu erneuern.

Die Schnittstelle `Credential` definiert explizit die Methoden `equals(Object)` und `hashCode`. Diese beiden Methoden sind wichtig, weil die authentifizierten Subject-Objekte mit dem Credential-Objekt als Schlüssel auf der Serverseite zwischengespeichert werden.

Sie können das bereitgestellte Plug-in verwenden, um einen Berechtigungs-nachweis zu erstellen. Dieses Plug-in wird durch die Schnittstelle `com.ibm.websphere.objectgrid.security.plugins.CredentialGenerator` dargestellt und ist hilfreich, wenn der Berechtigungs-nachweis eine Verfallszeit haben kann. In diesem Fall wird die Methode `getCredential` aufgerufen, um einen Berechtigungs-nachweis zu erneuern. Weitere Informationen finden Sie in der API-Dokumentation.

Es werden drei Standardimplementierungen für die Schnittstelle `Credential` bereitgestellt:

- Die Implementierung `com.ibm.websphere.objectgrid.security.plugins.builtins.UserPasswordCredential` enthält eine Kombination von Benutzer-ID und Kennwort.
- Die Implementierung `com.ibm.websphere.objectgrid.security.plugins.builtins.WSTokenCredential` enthält spezielle Authentifizierungs- und Berechtigungstoken für WebSphere Application Server. Diese Token können verwendet werden, um die Sicherheitsattribute an die Anwendungsserver in derselben Sicherheitsdomäne weiterzugeben.

WebSphere eXtreme Scale stellt auch ein Plug-in für die Generierung eines Berechtigungs-nachweises bereit. Dieses Plug-in wird durch die Schnittstelle `com.ibm.websphere.objectgrid.security.plugins.CredentialGenerator` dargestellt. WebSphere eXtreme Scale stellt zwei integrierte Standardimplementierungen bereit:

- Der Konstruktor `com.ibm.websphere.objectgrid.security.plugins.builtins.UserPasswordCredentialGenerator` akzeptiert eine Benutzer-ID und ein Kennwort. Beim Aufruf der Methode `"getCredential"` wird ein `UserPasswordCredential`-Objekt zurückgegeben, das die Benutzer-ID und das Kennwort enthält.
- Die Implementierung `com.ibm.websphere.objectgrid.security.plugins.builtins.WSTokenCredentialGenerator` stellt einen Berechtigungsnachweisgenerator (bzw. Sicherheitstokengenerator) dar, wenn Sie mit WebSphere Application Server arbeiten. Beim Aufruf der Methode `"getCredential"` wird das Subject-Objekt abgerufen, das dem aktuellen Thread zugeordnet ist. Anschließend werden die Sicherheitsinformationen in diesem Subject-Objekt in ein `WSTokenCredential`-Objekt konvertiert. Sie können angeben, ob mit der Konstanten `"WSTokenCredentialGenerator.RUN_AS_SUBJECT"` oder `"WSTokenCredentialGenerator.CALLER_SUBJECT"` ein `RunAs-Subject`- oder ein `Caller-Subject`-Objekt vom Thread abgerufen werden soll.

## UserPasswordCredential und UserPasswordCredentialGenerator

Für Testzwecke stellt WebSphere eXtreme Scale die folgenden Plug-in-Implementierungen bereit:

1. `com.ibm.websphere.objectgrid.security.plugins.builtins.UserPasswordCredential`
2. `com.ibm.websphere.objectgrid.security.plugins.builtins.UserPasswordCredentialGenerator`

Das `UserPasswordCredential` enthält eine Benutzer-ID und ein Kennwort. Der `UserPasswordCredentialGenerator` speichert diese Benutzer-ID und dieses Kennwort anschließend.

Der folgende Mustercode veranschaulicht, wie diese beiden Plug-ins implementiert werden.

```

UserPasswordCredential.java
// Dieses Musterprogramm wird ohne Wartung (auf "as-is"-Basis)
// bereitgestellt und kann vom Kunden (a) zu Schulungs- und Studienzwecken,
// (b) zum Entwickeln von Anwendungen für ein IBM WebSphere-Produkt zur
// internen Nutzung beim Kunden oder Weitergabe im Rahmen einer solchen
// Anwendung in kundeneigenen Produkten gebührenfrei genutzt, ausgeführt,
// kopiert und geändert werden.
// Lizenziertes Material - Eigentum von IBM
// 5724-J34 © COPYRIGHT International Business Machines Corp. 2007
package com.ibm.websphere.objectgrid.security.plugins.builtins;

import com.ibm.websphere.objectgrid.security.plugins.Credential;

/**
 * This class represents a credential containing a user ID and password.
 *
 * @ibm-api
 * @since WAS XD 6.0.1
 *
 * @see Credential
 * @see UserPasswordCredentialGenerator#getCredential()
 */
public class UserPasswordCredential implements Credential {

    private static final long serialVersionUID = 1409044825541007228L;

    private String ivUserName;

    private String ivPassword;

    /**
     * Creates a UserPasswordCredential with the specified user name and
     * password.
     *
     * @param userName the user name for this credential
     * @param password the password for this credential
     *
     * @throws IllegalArgumentException if userName or password is <code>null</code>
     */
    public UserPasswordCredential(String userName, String password) {
        super();
        if (userName == null || password == null) {

```

```

        throw new IllegalArgumentException("User name and password cannot be null.");
    }
    this.ivUserName = userName;
    this.ivPassword = password;
}

/**
 * Gets the user name for this credential.
 *
 * @return the user name argument that was passed to the constructor
 *         or the <code>setUserName(String)</code>
 *         method of this class
 *
 * @see #setUserName(String)
 */
public String getUserName() {
    return ivUserName;
}

/**
 * Sets the user name for this credential.
 *
 * @param userName the user name to set.
 *
 * @throws IllegalArgumentException if userName is <code>null</code>
 */
public void setUserName(String userName) {
    if (userName == null) {
        throw new IllegalArgumentException("User name cannot be null.");
    }
    this.ivUserName = userName;
}

/**
 * Gets the password for this credential.
 *
 * @return the password argument that was passed to the constructor
 *         or the <code>setPassword(String)</code>
 *         method of this class
 *
 * @see #setPassword(String)
 */
public String getPassword() {
    return ivPassword;
}

/**
 * Sets the password for this credential.
 *
 * @param password the password to set.
 *
 * @throws IllegalArgumentException if password is <code>null</code>
 */
public void setPassword(String password) {
    if (password == null) {
        throw new IllegalArgumentException("Password cannot be null.");
    }
    this.ivPassword = password;
}

/**
 * Checks two UserPasswordCredential objects for equality.
 *
 * <p>
 * Two UserPasswordCredential objects are equal if and only if their user names
 * and passwords are equal.
 *
 * @param o the object we are testing for equality with this object.
 *
 * @return <code>true</code> if both UserPasswordCredential objects are equivalent.
 *
 * @see Credential#equals(Object)
 */
public boolean equals(Object o) {
    if (this == o) {
        return true;
    }
    if (o instanceof UserPasswordCredential) {
        UserPasswordCredential other = (UserPasswordCredential) o;
        return other.ivPassword.equals(ivPassword) && other.ivUserName.equals(ivUserName);
    }
    return false;
}

/**
 * Returns the hashCode of the UserPasswordCredential object.
 *
 * @return the hash code of this object
 *
 * @see Credential#hashCode()
 */

```

```

        public int hashCode() {
            return ivUserName.hashCode() + ivPassword.hashCode();
        }
    }
}

UserPasswordCredentialGenerator.java
// Dieses Musterprogramm wird ohne Wartung (auf "as-is"-Basis)
// bereitgestellt und kann vom Kunden (a) zu Schulungs- und Studienzwecken,
// (b) zum Entwickeln von Anwendungen für ein IBM WebSphere-Produkt zur
// internen Nutzung beim Kunden oder Weitergabe im Rahmen einer solchen
// Anwendung in kundeneigenen Produkten gebührenfrei genutzt, ausgeführt,
// kopiert und geändert werden.
// Lizenziertes Material - Eigentum von IBM
// 5724-J34 © COPYRIGHT International Business Machines Corp. 2007
package com.ibm.websphere.objectgrid.security.plugins.builtins;

import java.util.StringTokenizer;

import com.ibm.websphere.objectgrid.security.plugins.Credential;
import com.ibm.websphere.objectgrid.security.plugins.CredentialGenerator;

/**
 * This credential generator creates <code>UserPasswordCredential</code> objects.
 * <p>
 * UserPasswordCredentialGenerator has a one to one relationship with
 * UserPasswordCredential because it can only create a UserPasswordCredential
 * representing one identity.
 *
 * @since WAS XD 6.0.1
 * @ibm-api
 *
 * @see CredentialGenerator
 * @see UserPasswordCredential
 */
public class UserPasswordCredentialGenerator implements CredentialGenerator {

    private String ivUser;

    private String ivPwd;

    /**
     * Creates a UserPasswordCredentialGenerator with no user name or password.
     *
     * @see #setProperties(String)
     */
    public UserPasswordCredentialGenerator() {
        super();
    }

    /**
     * Creates a UserPasswordCredentialGenerator with a specified user name and
     * password
     *
     * @param user the user name
     * @param pwd the password
     */
    public UserPasswordCredentialGenerator(String user, String pwd) {
        ivUser = user;
        ivPwd = pwd;
    }

    /**
     * Creates a new <code>UserPasswordCredential</code> object using this
     * object's user name and password.
     *
     * @return a new <code>UserPasswordCredential</code> instance
     *
     * @see CredentialGenerator#getCredential()
     * @see UserPasswordCredential
     */
    public Credential getCredential() {
        return new UserPasswordCredential(ivUser, ivPwd);
    }

    /**
     * Gets the password for this credential generator.
     *
     * @return the password argument that was passed to the constructor
     */
    public String getPassword() {
        return ivPwd;
    }

    /**
     * Gets the user name for this credential.
     *
     * @return the user argument that was passed to the constructor
     *         of this class
     */
    public String getUserName() {
        return ivUser;
    }
}

```

```

/**
 * Sets additional properties namely a user name and password.
 *
 * @param properties a properties string with a user name and
 * a password separated by a blank.
 *
 * @throws IllegalArgumentException if the format is not valid
 */
public void setProperties(String properties) {
    StringTokenizer token = new StringTokenizer(properties, " ");
    if (token.countTokens() != 2) {
        throw new IllegalArgumentException(
            "The properties should have a user name and password and separated by a blank.");
    }

    ivUser = token.nextToken();
    ivPwd = token.nextToken();
}
/**
 * Checks two UserPasswordCredentialGenerator objects for equality.
 * <p>
 * Two UserPasswordCredentialGenerator objects are equal if and only if
 * their user names and passwords are equal.
 *
 * @param obj the object we are testing for equality with this object.
 *
 * @return <code>true</code> if both UserPasswordCredentialGenerator objects
 * are equivalent.
 */
public boolean equals(Object obj) {
    if (obj == this) {
        return true;
    }

    if (obj != null && obj instanceof UserPasswordCredentialGenerator) {
        UserPasswordCredentialGenerator other = (UserPasswordCredentialGenerator) obj;

        boolean bothUserNull = false;
        boolean bothPwdNull = false;

        if (ivUser == null) {
            if (other.ivUser == null) {
                bothUserNull = true;
            } else {
                return false;
            }
        }

        if (ivPwd == null) {
            if (other.ivPwd == null) {
                bothPwdNull = true;
            } else {
                return false;
            }
        }

        return (bothUserNull || ivUser.equals(other.ivUser)) && (bothPwdNull || ivPwd.equals(other.ivPwd));
    }

    return false;
}

/**
 * Returns the hashCode of the UserPasswordCredentialGenerator object.
 *
 * @return the hash code of this object
 */
public int hashCode() {
    return ivUser.hashCode() + ivPwd.hashCode();
}
}

```

Die Klasse "UserPasswordCredential" enthält zwei Attribute: den Benutzernamen und das Kennwort. Die Klasse "UserPasswordCredentialGenerator" dient als Factory, die die UserPasswordCredential-Objekte enthält.

### WSTokenCredential und WSTokenCredentialGenerator

Wenn Clients und Server von WebSphere eXtreme Scale alle in WebSphere Application Server implementiert sind, kann die Clientanwendung diese beiden integrierten Implementierungen verwenden, wenn die folgenden Bedingungen erfüllt sind:

1. Die globale Sicherheit von WebSphere Application Server ist aktiviert.

2. Alle eXtreme-Scale-Clients und -Server werden in JVMs von WebSphere Application Server ausgeführt.
3. Alle Anwendungsserver befinden sich in derselben Sicherheitsdomäne.
4. Der Client ist bereits in WebSphere Application Server authentifiziert.

In dieser Situation kann der Client die Klasse `com.ibm.websphere.objectgrid.security.plugins.builtins.WSTokenCredentialGenerator` verwenden, um einen Berechtigungsnachweis zu generieren. Der Server verwendet die `WSAuthenticator-Implementierungsklasse`, um den Berechtigungsnachweis zu authentifizieren.

In diesem Szenario wird die Tatsache genutzt, dass der eXtreme-Scale-Client bereits authentifiziert wurde. Da sich die Anwendungsserver, die Server haben, in derselben Sicherheitsdomäne wie die Anwendungsserver befinden, die Clients haben, können die Sicherheitstoken vom Client an den Server weitergegeben werden, so dass dieselbe Benutzer-Registry nicht erneut authentifiziert werden muss.

**Anmerkung:** Gehen Sie nicht davon aus, dass ein `CredentialGenerator` immer denselben Berechtigungsnachweis generiert. Für ein Berechtigungsnachweis mit Verfallszeit und aktualisierbare Berechtigungsnachweise sollte der `CredentialGenerator` in der Lage sein, einen aktuellen gültigen Berechtigungsnachweis zu generieren, um sicherzustellen, dass die Authentifizierung erfolgreich ist. Ein Beispiel ist die Verwendung des Kerberos-Tickets als Credential-Objekt. Wenn das Kerberos-Ticket aktualisiert wird, muss der `CredentialGenerator` das aktualisierte Ticket abrufen, wenn `CredentialGenerator.getCredential` aufgerufen wird.

## Authenticator-Plug-in

Nachdem der eXtreme-Scale-Client das Credential-Objekt mit dem `CredentialGenerator-Objekt` abgerufen hat, wird dieses Client-Credential-Objekt zusammen mit der Clientanforderung an den eXtreme-Scale-Server gesendet. Der Server authentifiziert das Credential-Objekt, bevor er die Anforderung verarbeitet. Bei erfolgreicher Authentifizierung des Credential-Objekts wird ein `Subject-Objekt` zurückgegeben, das diesen Client repräsentiert.

Dieses `Subject-Objekt` wird zwischengespeichert und verfällt erst, wenn seine Lebensdauer das festgelegte Sitzungszeitlimit erreicht. Das Zeitlimit für die Anmeldesitzung kann mit der Eigenschaft `"loginSessionExpirationTime"` in der XML-Datei des Clusters definiert werden. Wenn Sie beispielsweise `loginSessionExpirationTime="300"` definieren, verfällt das `Subject-Objekt` nach 300 Sekunden.

Dieses `Subject-Objekt` wird anschließend für die Berechtigung der Anforderung verwendet, was später noch erläutert wird. Ein eXtreme-Scale-Server verwendet das `Authenticator-Plug-in`, um das Credential-Objekt zu authentifizieren. Weitere Einzelheiten finden Sie in den Informationen zu `Authenticator` in der `API-Dokumentation`.

Im `Authenticator-Plug-in` authentifiziert die Laufzeitumgebung von eXtreme Scale das Credential-Objekt aus der Benutzer-Registry des Clients, z. B. einem LDAP-Server.

WebSphere eXtreme Scale stellt keine sofort einsatzfähige Benutzer-Registry-Konfiguration bereit. Die Konfiguration und Verwaltung der Benutzer-Registry erfolgt aus Gründen der Einfachheit und Flexibilität außerhalb von WebSphere eXtreme Scale. Dieses `Plug-in` implementiert die Verbindungsherstellung zur und die



Authentifizierung über die Benutzer-Registry. Eine Authenticator-Implementierung extrahiert beispielsweise die Benutzer-ID und das Kennwort aus dem Berechtigungsnachweis, verwendet diese Informationen für die Herstellung einer Verbindung zu einem und Validierung bei einem LDAP-Server und erstellt als Ergebnis der Authentifizierung ein Subject-Objekt. Die Implementierung kann JAAS-Anmeldemodule verwenden. Als Ergebnis der Authentifizierung wird ein Subject-Objekt zurückgegeben.

Beachten Sie, dass diese Methode zwei Ausnahmen erstellt: `InvalidCredentialException` und `ExpiredCredentialException`. Die Ausnahme `"InvalidCredentialException"` zeigt an, dass der Berechtigungsnachweis nicht gültig ist. Die Ausnahme `"ExpiredCredentialException"` zeigt an, dass der Berechtigungsnachweis abgelaufen ist. Wenn eine dieser Ausnahmen in der Methode `"authenticate"` eintritt, wird sie an den Client zurückgesendet. Die Laufzeitumgebung des Clients behandelt diese Ausnahmen jedoch auf unterschiedliche Weise:

- Wenn der Fehler eine Ausnahme `"InvalidCredentialException"` ist, zeigt die Laufzeitumgebung des Clients diese Ausnahme an. Ihre Anwendung muss die Ausnahme behandeln. In diesem Fall können Sie den `CredentialGenerator` beispielsweise korrigieren und die Operation anschließend wiederholen.
- Wenn der Fehler eine Ausnahme `"ExpiredCredentialException"` ist und der Wiederholungszähler nicht 0 ist, ruft die Laufzeitumgebung des Clients die Methode `"CredentialGenerator.getCredential"` erneut auf und sendet das neue Credential-Objekt an den Server. Wenn die Authentifizierung des neuen Berechtigungsnachweises erfolgreich ist, verarbeitet der Server die Anforderung. Schlägt die Authentifizierung des neuen Berechtigungsnachweises fehl, wird die Ausnahme an den Client zurückgesendet. Wenn die Anzahl der Authentifizierungswiederholungen den unterstützten Wert erreicht und der Client immer noch eine Ausnahme `"ExpiredCredentialException"` empfängt, wird die Ausnahme `"ExpiredCredentialException"` ausgelöst. Ihre Anwendung muss den Fehler behandeln.

Die Schnittstelle `"Authenticator"` bietet hohe Flexibilität. Sie können die Schnittstelle `"Authenticator"` auf Ihre eigene spezielle Weise implementieren. Sie können diese Schnittstelle beispielsweise für die Unterstützung von zwei unterschiedlichen Benutzer-Registries implementieren.

WebSphere eXtreme Scale stellt Beispielimplementierungen des Authenticator-Plugins bereit. Mit Ausnahme des Authenticator-Plugins von WebSphere Application Server sind die anderen Implementierungen nur Beispiele für Testzwecke.

### KeyStoreLoginAuthenticator

In diesem Beispiel wird eine integrierte eXtreme-Scale-Implementierung verwendet, die Implementierung `"KeyStoreLoginAuthenticator"`, die für Test- und Beispielszwecke bestimmt ist (ein Keystore ist eine einfache Benutzer-Registry und sollte nicht für eine Produktionsumgebung verwendet werden). Auch hier sehen Sie die Klasse, um ausführlicher zu demonstrieren, wie ein Authentifikator implementiert wird:

```
KeyStoreLoginAuthenticator.java
// Dieses Musterprogramm wird ohne Wartung (auf "as-is"-Basis)
// bereitgestellt und kann vom Kunden (a) zu Schulungs- und Studienzwecken,
// (b) zum Entwickeln von Anwendungen für ein IBM WebSphere-Produkt zur
// internen Nutzung beim Kunden oder Weitergabe im Rahmen einer solchen
// Anwendung in kundeneigenen Produkten gebührenfrei genutzt, ausgeführt,
// kopiert und geändert werden.
// Lizenziertes Material - Eigentum von IBM
// 5724-J34 © COPYRIGHT International Business Machines Corp. 2007

package com.ibm.websphere.objectgrid.security.plugins.builtins;
```

```

import javax.security.auth.Subject;
import javax.security.auth.login.LoginContext;
import javax.security.auth.login.LoginException;

import com.ibm.websphere.objectgrid.security.plugins.Authenticator;
import com.ibm.websphere.objectgrid.security.plugins.Credential;
import com.ibm.websphere.objectgrid.security.plugins.ExpiredCredentialException;
import com.ibm.websphere.objectgrid.security.plugins.InvalidCredentialException;
import com.ibm.ws.objectgrid.Constants;
import com.ibm.ws.objectgrid.ObjectGridManagerImpl;
import com.ibm.ws.objectgrid.security.auth.callback.UserPasswordCallbackHandlerImpl;

/**
 * This class is an implementation of the <code>Authenticator</code> interface
 * when a user name and password are used as a credential.
 * <p>
 * When user ID and password authentication is used, the credential passed to the
 * <code>authenticate(Credential)</code> method is a UserPasswordCredential object.
 * <p>
 * This implementation will use a <code>KeyStoreLoginModule</code> to authenticate
 * the user into the key store using the JAAS login module "KeyStoreLogin". The key
 * store can be configured as an option to the <code>KeyStoreLoginModule</code>
 * class. Please see the <code>KeyStoreLoginModule</code> class for more details
 * about how to set up the JAAS login configuration file.
 * <p>
 * This class is only for sample and quick testing purpose. Users should
 * write your own Authenticator implementation which can fit better into
 * the environment.
 *
 * @ibm-api
 * @since WAS XD 6.0.1
 *
 * @see Authenticator
 * @see KeyStoreLoginModule
 * @see UserPasswordCredential
 */
public class KeyStoreLoginAuthenticator implements Authenticator {

    /**
     * Creates a new KeyStoreLoginAuthenticator.
     */
    public KeyStoreLoginAuthenticator() {
        super();
    }

    /**
     * Authenticates a <code>UserPasswordCredential</code>.
     * <p>
     * Uses the user name and password from the specified UserPasswordCredential
     * to login to the KeyStoreLoginModule named "KeyStoreLogin".
     *
     * @throws InvalidCredentialException if credential isn't a
     *         UserPasswordCredential or some error occurs during processing
     *         of the supplied UserPasswordCredential
     *
     * @throws ExpiredCredentialException if credential is expired. This exception
     *         is not used by this implementation
     *
     * @see Authenticator#authenticate(Credential)
     * @see KeyStoreLoginModule
     */
    public Subject authenticate(Credential credential) throws InvalidCredentialException, ExpiredCredentialException {
        if (credential == null) {
            throw new InvalidCredentialException("Supplied credential is null");
        }

        if ( ! (credential instanceof UserPasswordCredential) ) {
            throw new InvalidCredentialException("Supplied credential is not a UserPasswordCredential");
        }

        UserPasswordCredential cred = (UserPasswordCredential) credential;
        LoginContext lc = null;
        try {
            lc = new LoginContext("KeyStoreLogin",
                new UserPasswordCallbackHandlerImpl(cred.getUserName(), cred.getPassword().toCharArray()));

            lc.login();

            Subject subject = lc.getSubject();

            return subject;
        } catch (LoginException le) {
            throw new InvalidCredentialException(le);
        } catch (IllegalArgumentException ile) {
            throw new InvalidCredentialException(ile);
        }
    }
}

```

```

KeyStoreLoginModule.java
// Dieses Musterprogramm wird ohne Wartung (auf "as-is"-Basis)
// bereitgestellt und kann vom Kunden (a) zu Schulungs- und Studienzwecken,
// (b) zum Entwickeln von Anwendungen für ein IBM WebSphere-Produkt zur
// internen Nutzung beim Kunden oder Weitergabe im Rahmen einer solchen
// Anwendung in kundeneigenen Produkten gebührenfrei genutzt, ausgeführt,
// kopiert und geändert werden.
// Lizenziertes Material – Eigentum von IBM
// 5724-J34 © COPYRIGHT International Business Machines Corp. 2007
package com.ibm.websphere.objectgrid.security.plugins.builtins;

import java.io.File;
import java.io.FileInputStream;
import java.security.KeyStore;
import java.security.KeyStoreException;
import java.security.NoSuchAlgorithmException;
import java.security.PrivateKey;
import java.security.UnrecoverableKeyException;
import java.security.cert.Certificate;
import java.security.cert.CertificateException;
import java.security.cert.CertificateFactory;
import java.security.cert.X509Certificate;
import java.util.Arrays;
import java.util.HashSet;
import java.util.Map;
import java.util.Set;

import javax.security.auth.Subject;
import javax.security.auth.callback.Callback;
import javax.security.auth.callback.CallbackHandler;
import javax.security.auth.callback.NameCallback;
import javax.security.auth.callback.PasswordCallback;
import javax.security.auth.login.LoginException;
import javax.security.auth.spi.LoginModule;
import javax.security.auth.x500.X500Principal;
import javax.security.auth.x500.X500PrivateCredential;

import com.ibm.websphere.objectgrid.ObjectGridRuntimeException;
import com.ibm.ws.objectgrid.Constants;
import com.ibm.ws.objectgrid.ObjectGridManagerImpl;
import com.ibm.ws.objectgrid.util.ObjectGridUtil;

/**
 * A KeyStoreLoginModule is keystore authentication login module based on
 * JAAS authentication.
 * <p>
 * A login configuration should provide an option "<code>keyStoreFile</code>" to
 * indicate where the keystore file is located. If the <code>keyStoreFile</code>
 * value contains a system property in the form, <code>${system.property}</code>,
 * it will be expanded to the value of the system property.
 * <p>
 * If an option "<code>keyStoreFile</code>" is not provided, the default keystore
 * file name is <code>${java.home}/.keystore</code>.
 * <p>
 * Here is a Login module configuration example:
 * <pre><code>
 *     KeyStoreLogin {
 *         com.ibm.websphere.objectgrid.security.plugins.builtins.KeystoreLoginModule required
 *         keyStoreFile="${user.dir}/security/.keystore";
 *     };
 * </code></pre>
 *
 * @ibm-api
 * @since WAS XD 6.0.1
 *
 * @see LoginModule
 */
public class KeyStoreLoginModule implements LoginModule {

    private static final String CLASS_NAME = KeyStoreLoginModule.class.getName();

    /**
     * Key store file property name
     */
    public static final String KEY_STORE_FILE_PROPERTY_NAME = "keyStoreFile";

    /**
     * Key store type. Only JKS is supported
     */
    public static final String KEYSTORE_TYPE = "JKS";

    /**
     * The default key store file name
     */
    public static final String DEFAULT_KEY_STORE_FILE = "${java.home}/.keystore";

    private CallbackHandler handler;

    private Subject subject;

    private boolean debug = false;

```

```

private Set principals = new HashSet();
private Set publicCreds = new HashSet();
private Set privateCreds = new HashSet();

protected KeyStore keyStore;

/**
 * Creates a new KeyStoreLoginModule.
 */
public KeyStoreLoginModule() {
}

/**
 * Initializes the login module.
 * @see LoginModule#initialize(Subject, CallbackHandler, Map, Map)
 */
public void initialize(Subject sub, CallbackHandler callbackHandler,
    Map mapSharedState, Map mapOptions) {

    // initialize any configured options
    debug = "true".equalsIgnoreCase((String) mapOptions.get("debug"));

    if (sub == null)
        throw new IllegalArgumentException("Subject is not specified");

    if (callbackHandler == null)
        throw new IllegalArgumentException(
            "CallbackHandler is not specified");

    // Get the key store path
    String sKeyStorePath = (String) mapOptions
        .get(KEY_STORE_FILE_PROPERTY_NAME);

    // If there is no key store path, the default one is the .keystore
    // file in the java home directory
    if (sKeyStorePath == null) {
        sKeyStorePath = DEFAULT_KEY_STORE_FILE;
    }

    // Replace the system environment variable
    sKeyStorePath = ObjectGridUtil.replaceVar(sKeyStorePath);

    File fileKeyStore = new File(sKeyStorePath);

    try {
        KeyStore store = KeyStore.getInstance("JKS");
        store.load(new FileInputStream(fileKeyStore), null);

        // Save the key store
        keyStore = store;

        if (debug) {
            System.out.println("[KeyStoreLoginModule] initialize: Successfully loaded key store");
        }
    }
    catch (Exception e) {
        ObjectGridRuntimeException re = new ObjectGridRuntimeException(
            "Failed to load keystore: " + fileKeyStore.getAbsolutePath());
        re.initCause(e);
        if (debug) {
            System.out.println("[KeyStoreLoginModule] initialize: Key store loading failed with exception "
                + e.getMessage());
        }
    }

    this.subject = sub;
    this.handler = callbackHandler;
}

/**
 * Authenticates a user based on the keystore file.
 * @see LoginModule#login()
 */
public boolean login() throws LoginException {

    if (debug) {
        System.out.println("[KeyStoreLoginModule] login: entry");
    }

    String name = null;
    char pwd[] = null;

    if (keyStore == null || subject == null || handler == null) {
        throw new LoginException("Module initialization failed");
    }
}

```

```

NameCallback nameCallback = new NameCallback("Username:");
PasswordCallback pwdCallback = new PasswordCallback("Password:", false);

try {
    handler.handle(new Callback[] { nameCallback, pwdCallback });
}
catch (Exception e) {
    throw new LoginException("Callback failed: " + e);
}

name = nameCallback.getName();
char[] tempPwd = pwdCallback.getPassword();

if (tempPwd == null) {
    // treat a NULL password as an empty password
    tempPwd = new char[0];
}
pwd = new char[tempPwd.length];
System.arraycopy(tempPwd, 0, pwd, 0, tempPwd.length);

pwdCallback.clearPassword();

if (debug) {
    System.out.println("[KeyStoreLoginModule] login: "
        + "user entered user name: " + name);
}

// Validate the user name and password
try {
    validate(name, pwd);
}
catch (SecurityException se) {
    principals.clear();
    publicCreds.clear();
    privateCreds.clear();
    LoginException le = new LoginException(
        "Exception encountered during login");
    le.initCause(se);

    throw le;
}

if (debug) {
    System.out.println("[KeyStoreLoginModule] login: exit");
}
return true;
}

/**
 * Indicates the user is accepted.
 * <p>
 * This method is called only if the user is authenticated by all modules in
 * the login configuration file. The principal objects will be added to the
 * stored subject.
 *
 * @return false if for some reason the principals cannot be added; true
 *         otherwise
 *
 * @exception LoginException
 *         LoginException is thrown if the subject is readonly or if
 *         any unrecoverable exceptions is encountered.
 *
 * @see LoginModule#commit()
 */
public boolean commit() throws LoginException {
    if (debug) {
        System.out.println("[KeyStoreLoginModule] commit: entry");
    }

    if (principals.isEmpty()) {
        throw new IllegalStateException("Commit is called out of sequence");
    }

    if (subject.isReadOnly()) {
        throw new LoginException("Subject is Readonly");
    }

    subject.getPrincipals().addAll(principals);
    subject.getPublicCredentials().addAll(publicCreds);
    subject.getPrivateCredentials().addAll(privateCreds);

    principals.clear();
    publicCreds.clear();
    privateCreds.clear();

    if (debug) {
        System.out.println("[KeyStoreLoginModule] commit: exit");
    }
    return true;
}

```

```

/**
 * Indicates the user is not accepted
 *
 * @see LoginModule#abort()
 */
public boolean abort() throws LoginException {
    boolean b = logout();
    return b;
}

/**
 * Logs the user out. Clear all the maps.
 *
 * @see LoginModule#logout()
 */
public boolean logout() throws LoginException {

    // Clear the instance variables
    principals.clear();
    publicCreds.clear();
    privateCreds.clear();

    // clear maps in the subject
    if (!subject.isReadOnly()) {
        if (subject.getPrincipals() != null) {
            subject.getPrincipals().clear();
        }

        if (subject.getPublicCredentials() != null) {
            subject.getPublicCredentials().clear();
        }

        if (subject.getPrivateCredentials() != null) {
            subject.getPrivateCredentials().clear();
        }
    }
    return true;
}

/**
 * Validates the user name and password based on the keystore.
 *
 * @param userName user name
 * @param password password
 * @throws SecurityException if any exceptions encountered
 */
private void validate(String userName, char password[])
    throws SecurityException {

    PrivateKey privateKey = null;

    // Get the private key from the keystore
    try {
        privateKey = (PrivateKey) keyStore.getKey(userName, password);
    }
    catch (NoSuchAlgorithmException nsae) {
        SecurityException se = new SecurityException();
        se.initCause(nsae);
        throw se;
    }
    catch (KeyStoreException kse) {
        SecurityException se = new SecurityException();
        se.initCause(kse);
        throw se;
    }
    catch (UnrecoverableKeyException uke) {
        SecurityException se = new SecurityException();
        se.initCause(uke);
        throw se;
    }

    if (privateKey == null) {
        throw new SecurityException("Invalid name: " + userName);
    }

    // Check the certificats
    Certificate certs[] = null;
    try {
        certs = keyStore.getCertificateChain(userName);
    }
    catch (KeyStoreException kse) {
        SecurityException se = new SecurityException();
        se.initCause(kse);
        throw se;
    }

    if (debug) {
        System.out.println(" Print out the certificates:");
        for (int i = 0; i < certs.length; i++) {
            System.out.println(" certificate " + i);
        }
    }
}

```



```

        catch (IllegalArgumentException ile) {
            throw new InvalidCredentialException(ile);
        }
    }
}

```

Außerdem wird mit eXtreme Scale das Anmeldemodul "com.ibm.websphere.objectgrid.security.plugins.builtins.LDAPLoginModule" für diesen Zweck bereitgestellt. Sie müssen die folgenden beiden Optionen in der JAAS-Anmeldekonfigurationsdatei angeben:

- providerURL: Der Provider-URL des LDAP-Servers.
- factoryClass: Die Implementierungsklasse der LDAP-Kontext-Factory.

Das Anmeldemodul "LDAPLoginModule" ruft die Methode com.ibm.websphere.objectgrid.security.plugins.builtins.LDAPAuthenticationHelper.authenticate auf. Das folgende Code-Snippet zeigt, wie Sie die Methode "authenticate" von LDAPAuthenticationHelper implementieren:

```

/**
 * Benutzer über das LDAP-Verzeichnis authentifizieren.
 * @param user the user ID, e.g., uid=xxxxxx,c=us,ou=bluepages,o=ibm.com
 * @param pwd the password
 *
 * @throws NamingException
 */
public String[] authenticate(String user, String pwd)
throws NamingException {
    Hashtable env = new Hashtable();
    env.put(Context.INITIAL_CONTEXT_FACTORY, factoryClass);
    env.put(Context.PROVIDER_URL, providerURL);
    env.put(Context.SECURITY_PRINCIPAL, user);
    env.put(Context.SECURITY_CREDENTIALS, pwd);
    env.put(Context.SECURITY_AUTHENTICATION, "simple");

    InitialContext initialContext = new InitialContext(env);

    // Benutzer suchen.
    DirContext dirCtx = (DirContext) initialContext.lookup(user);

    String uid = null;
    int iComma = user.indexOf(",");
    int iEqual = user.indexOf("=");
    if (iComma > 0 && iComma > 0) {
        uid = user.substring(iEqual + 1, iComma);
    }
    else {
        uid = user;
    }

    Attributes attributes = dirCtx.getAttributes("");

    // UID prüfen.
    String thisUID = (String) (attributes.get(UID).get());

    String thisDept = (String) (attributes.get(HR_DEPT).get());

    if (thisUID.equals(uid)) {
        return new String[] { thisUID, thisDept };
    }
    else {
        return null;
    }
}

```

Wenn die Authentifizierung erfolgreich ist, werden ID und Kennwort als gültig eingestuft. Anschließend ruft das Anmeldemodul die ID-Informationen und



Abteilungsinformationen über diese Methode "authenticate" ab. Das Anmelde-  
modul erstellt zwei Principals: SimpleUserPrincipal und SimpleDeptPrincipal. Sie  
können das authentifizierte Subject-Objekte für die Gruppenberechtigung (in die-  
sem Fall ist die Abteilung eine Gruppe) und Berechtigung von Einzelpersonen ver-  
wenden.

Das folgende Beispiel zeigt eine Anmeldemodulkonfiguration, die für die Anmel-  
dung beim LDAP-Server verwendet wird:

```
LDAPLogin { com.ibm.websphere.objectgrid.security.plugins.builtins.LDAPLoginModule required
    providerURL="ldap://directory.acme.com:389/"
    factoryClass="com.sun.jndi.ldap.LdapCtxFactory";
};
```

In der vorherigen Konfiguration zeigt der LDAP-Server auf ldap://  
directory.acme.com:389/server. Ändern Sie diese Einstellung in Ihren LDAP-Server.  
Dieses Anmeldemodul verwendet die bereitgestellte ID und das bereitgestellte  
Kennwort für die Verbindungsherstellung zum LDAP-Server. Diese Implementie-  
rung ist nur für Testzwecke bestimmt.

### **Authenticator-Plug-in für WebSphere Application Server verwenden**

eXtreme Scale stellt auch die integrierte Implementierung "com.ibm.websphere.ob-  
jectgrid.security.plugins.builtins.WSTokenAuthenticator" bereit, mit der Sie die  
Sicherheitsinfrastruktur von WebSphere Application Server verwenden können.  
Diese integrierte Implementierung kann verwendet werden, wenn die folgenden  
Bedingungen zutreffen:

1. Die globale Sicherheit von WebSphere Application Server ist aktiviert.
2. Alle eXtreme-Scale-Clients und -Server sind in JVMs von WebSphere Applica-  
tion Server gestartet.
3. Diese Anwendungsserver befinden sich in derselben Sicherheitsdomäne.
4. Der eXtreme-Scale-Client ist bereits in WebSphere Application Server authenti-  
fiziert.

Der Client kann die Klasse com.ibm.websphere.objectgrid.security.plugins.builtins.  
WSTokenCredentialGenerator verwenden, um einen Berechtigungsnachweis zu  
generieren. Der Server verwendet diese Authenticator-Implementierungsklasse, um  
den Berechtigungsnachweis zu authentifizieren. Bei erfolgreicher Authentifizierung  
des Tokens wird ein Subject-Objekt zurückgegeben.

In diesem Szenario wird die Tatsache genutzt, dass der Client bereits authenti-  
fiziert wurde. Da sich die Anwendungsserver, die Server haben, in derselben  
Sicherheitsdomäne wie die Anwendungsserver befinden, die Clients haben, können  
die Sicherheitstoken vom Client an den Server weitergegeben werden, so dass die-  
selbe Benutzer-Registry nicht erneut authentifiziert werden muss.

### **Authenticator-Plug-in für Tivoli Access Manager verwenden**

Tivoli Access Manager wird weithin als Sicherheitsserver eingesetzt. Sie können  
Authenticator auch über die mit Tivoli Access Manager bereitgestellten Anmelde-  
module implementieren.

Zum Authentifizieren eines Benutzers für Tivoli Access Manager wenden Sie das  
Anmeldemodul "com.tivoli.mts.PDLoginModule" an, das erfordert, dass die aufru-  
fende Anwendung die folgenden Informationen bereitstellt:

1. einen Principal-Namen (als Kurznamen oder als X.500-Namen (DN)),
2. ein Kennwort.

Das Anmeldemodul authentifiziert den Principal und gibt den Berechtigungsnachweis von Tivoli Access Manager zurück. Das Anmeldemodul erwartet, dass die aufrufende Anwendung die folgenden Informationen bereitstellt:

1. den Benutzernamen in einem `javax.security.auth.callback.NameCallback`-Objekt,
2. das Kennwort in einem `javax.security.auth.callback.PasswordCallback`-Objekt.

Nachdem der Berechtigungsnachweis von Tivoli Access Manager erfolgreich abgerufen wurde, erstellt das JAAS-Anmeldemodul ein `Subject`- und ein `PDPrincipal`-Objekt. Es wird keine integrierte Lösung für die Authentifizierung in Tivoli Access Manager bereitgestellt, weil die Authentifizierung nur über das Modul "PDLogin-Module" erfolgt. Weitere Einzelheiten finden Sie in der Veröffentlichung IBM Tivoli Access Manager Authorization Java Classes Developer Reference.

## Sichere Verbindung zu WebSphere eXtreme Scale herstellen

Um eine sichere Verbindung zwischen einem eXtreme-Scale-Client und einem Server herzustellen, können Sie jede beliebige `connect`-Methode in der Schnittstelle "ObjectGridManager" verwenden, die ein `ClientSecurityConfiguration`-Objekt akzeptiert. Im Folgenden sehen Sie ein kurzes Beispiel:

```
public ClientClusterContext connect(String catalogServerAddresses,
    ClientSecurityConfiguration securityProps,
    URL overRideObjectGridXml) throws ConnectException;
```

Diese Methode akzeptiert einen Parameter des Typs "ClientSecurityConfiguration", der eine Schnittstelle ist, die eine Clientsicherheitskonfiguration darstellt. Sie können die allgemein zugängliche API "com.ibm.websphere.objectgrid.security.config.ClientSecurityConfigurationFactory" verwenden, um eine Instanz mit Standardwerten zu erstellen, oder Sie können eine Instanz erstellen, indem Sie die Eigenschaftendatei des eXtreme-Scale-Clients übergeben. Diese Datei enthält die folgenden Eigenschaften, die sich auf die Authentifizierung beziehen. Der mit einem Pluszeichen (+) markierte Wert ist der Standardwert.

- `securityEnabled` (true, false+): Diese Eigenschaft zeigt an, ob die Sicherheit aktiviert ist. Wenn ein Client eine Verbindung zu einem Server herstellt, muss die Eigenschaft "securityEnabled" auf der Client- und auf der Serverseite denselben Wert haben: true oder false. Sollte die Sicherheit auf dem verbindungsherstellenden Server beispielsweise aktiviert sein, muss die Eigenschaft auch auf dem Client auf "true" gesetzt werden, damit die Verbindung zum Server hergestellt werden kann.
- `authenticationRetryCount` (ganzzahliger Wert, 0+): Diese Eigenschaft bestimmt, wie oft die Anmeldung wiederholt wird, wenn ein Berechtigungsnachweis verfallen ist. Beim Wert 0 wird die Anmeldung nicht wiederholt. Die Authentifizierungswiederholung gilt nur für den Fall, dass der Berechtigungsnachweis verfallen ist. Wenn der Berechtigungsnachweis nicht gültig ist, findet keine Wiederholung statt. Ihre Anwendung ist für die Wiederholung der Operation verantwortlich.

Nachdem Sie ein `com.ibm.websphere.objectgrid.security.config.ClientSecurityConfiguration`-Objekt erstellt haben, definieren Sie das `credentialGenerator`-Objekt auf dem Client mit der folgenden Methode:

```
/**
 * {@link CredentialGenerator}-Objekt für diesen Client definieren.
 * @param generator Das CredentialGenerator-Objekt, das dem Client zugeordnet wird.
 */
void setCredentialGenerator(CredentialGenerator generator);
```

Sie können das CredentialGenerator-Objekt auch wie folgt in der Eigenschaftendatei des eXtreme-Scale-Clients setzen:

- credentialGeneratorClass: Der Name der Implementierungsklasse für das CredentialGenerator-Objekt. Diese Klasse muss einen Standardkonstruktor haben.
- credentialGeneratorProps: Die Eigenschaften für die Klasse "CredentialGenerator". Bei einem Wert ungleich null, wird diese Eigenschaft mit der Methode "setProperty(String)" auf das erstellte CredentialGenerator-Objekt gesetzt.

Es folgt ein Beispiel für die Instanzierung einer ClientSecurityConfiguration, die anschließend für die Verbindungsherstellung zum Server verwendet wird.

```
/**
 * Gesicherten ClientClusterContext abrufen.
 * @return Ein sicheres ClientClusterContext-Objekt.
 */
protected ClientClusterContext connect() throws ConnectException {
    ClientSecurityConfiguration csConfig = ClientSecurityConfigurationFactory
        .getClientSecurityConfiguration("/properties/security.ogclient.props");

    UserPasswordCredentialGenerator gen= new
        UserPasswordCredentialGenerator("manager", "manager1");

    csConfig.setCredentialGenerator(gen);

    return objectGridManager.connect(csConfig, null);
}
```

Wenn die connect-Methode aufgerufen wird, ruft der eXtreme-Scale-Client die Methode "CredentialGenerator.getCredential" auf, um den Clientberechtigungs-nachweis abzurufen. Dieser Berechtigungsnachweis wird zusammen mit der Verbindungsanforderung zur Authentifizierung an den Server gesendet.

## Für jede Sitzung eine andere CredentialGenerator-Instanz verwenden

In manchen Fällen stellt ein eXtreme-Scale-Client nur eine einzige Clientidentität dar. In anderen Fällen wiederum kann er mehrere Identitäten darstellen. Es folgt ein Szenario für den letzten Fall: Es wird ein eXtreme-Scale-Client erstellt und in einem Webserver gemeinsam genutzt. Alle Servlets in diesem Webserver verwenden diesen einen eXtreme-Scale-Client. Da jedes Servlet einen anderen Webclient darstellt, verwenden Sie unterschiedliche Berechtigungsnachweise, wenn Sie Anforderungen an eXtreme-Scale-Server senden.

WebSphere eXtreme Scale unterstützt die Änderung des Berechtigungsnachweises auf Sitzungsebene. Jede Sitzung kann ein anderes CredentialGenerator-Objekt verwenden. Deshalb können die zuvor beschriebenen Szenarios realisiert werden, indem Sie das Servlet eine Sitzung mit einem anderen CredentialGenerator-Objekt abrufen lassen. Das folgende Beispiel veranschaulicht die Methode "ObjectGrid.getSession(CredentialGenerator)" in der Schnittstelle "ObjectGridManager".

```
/**
 * Sitzung mit <code>CredentialGenerator</code> abrufen.
 * <p>
 * Diese Methode kann nur vom ObjectGrid-Client in einer Client/Server-Umgebung
 * aufgerufen werden. Wenn ObjectGrid in einem lokalen Modell verwendet wird, d. h.
 * in derselben JVM ohne vorhandenen Client oder Server, muss die Methode
 * <code>getSession(Subject)</code> oder das Plug-in <code>SubjectSource</code>
 * zum Sichern des ObjectGrids verwendet werden.
 *
 * <p>Wenn die Methode <code>initialize()</code> nicht vor dem ersten
 * Aufruf von <code>getSession</code> aufgerufen wird, findet eine
 * implizite Implementierung statt. Auf diese Weise wird sichergestellt, dass die
 * gesamte Konfiguration abgeschlossen ist, bevor sie zur Laufzeit benötigt wird.</p>
 *
 * @param credGen Ein <code>CredentialGenerator</code> für die Generierung
```

```

*           eines Berechtigungsnachweises für die zurückgegebene Sitzung.
*
* @return Eine Instanz von <code>Session</code>.
*
* @throws ObjectGridException, wenn während der Verarbeitung ein Fehler auftritt.
*         * @throws TransactionCallbackException, wenn <code>TransactionCallback</code>
*           eine Ausnahme auslöst.
* @throws IllegalStateException, wenn diese Methode nach dem Aufruf der
*         Methode <code>destroy()</code> aufgerufen wird.
*
*     * @see #destroy()
*     * @see #initialize()
*     * @see CredentialGenerator
*     * @see Session
*     * @since WAS XD 6.0.1
*/
Session getSession(CredentialGenerator credGen) throws
ObjectGridException, TransactionCallbackException;

```

### Beispiel:

```

ObjectGridManager ogManager = ObjectGridManagerFactory.getObjectGridManager();

CredentialGenerator credGenManager = new UserPasswordCredentialGenerator("manager", "xxxxxx");
CredentialGenerator credGenEmployee = new UserPasswordCredentialGenerator("employee", "xxxxxx");

ObjectGrid og = ogManager.getObjectGrid(ctx, "accounting");

// Sitzung mit CredentialGenerator abrufen
Session session = og.getSession(credGenManager );

// Mitarbeiterzuordnung abrufen
ObjectMap om = session.getMap("employee");

// Transaktion starten
session.begin();

Object rec1 = map.get("xxxxxx");

session.commit();

// Weitere Sitzung mit einem anderen CredentialGenerator abrufen
session = og.getSession(credGenEmployee );

// Mitarbeiterzuordnung abrufen
om = session.getMap("employee");

// Transaktion starten
session.begin();

Object rec2 = map.get("xxxxxx");

session.commit();

```

Wenn Sie die Methode "ObjectGrid.getSession" verwenden, um ein Session-Objekt abzurufen, verwendet die Sitzung das CredentialGenerator-Objekt, das im Client-ConfigurationSecurity-Objekt definiert ist. Die Methode "ObjectGrid.getSession(CredentialGenerator)" überschreibt den im ClientSecurityConfiguration-Objekt definierten CredentialGenerator.

Wenn Sie das Session-Objekt wiederverwenden können, findet eine Leistungssteigerung statt. Der Aufruf der Methode "ObjectGrid.getSession(CredentialGenerator)" ist jedoch nicht sehr kostenintensiv. Die Hauptkosten entstehen durch die längere Dauer der Objekt-Garbage-Collection. Stellen Sie sicher, dass die Referenzen nach der Arbeit mit den Session-Objekten freigegeben werden. Wenn Ihr Session-Objekt die Identität freigeben kann, können Sie im Allgemeinen versuchen, das Session-Objekt wiederzuverwenden. Wenn nicht, verwenden Sie die Methode "ObjectGrid.getSession(CredentialGenerator)".

---

## Programmierung der Clientberechtigung

WebSphere eXtreme Scale unterstützt die vordefinierte JAAS-Berechtigung (Java Authentication and Authorization) und auch die angepasste Berechtigung mit über die Schnittstelle "ObjectGridAuthorization".

Das ObjectGridAuthorization-Plug-in wird verwendet, um ObjectGrid-, ObjectMap- und JavaMap-Zugriff auf die Principals, die durch ein Subject-Objekt dargestellt werden, auf angepasste Weise zu berechtigen. Eine typische Implementierung dieses Plug-ins ist der Abruf der Principals aus dem Subject-Objekt mit anschließender dahingehender Prüfung, ob die angegebenen Berechtigungen den Principals erteilt wurden oder nicht.

Die folgenden Berechtigungen können an die Methode "checkPermission(Subject, Permission)" übergeben werden:

1. MapPermission
2. ObjectGridPermission
3. ServerMapPermission
4. AgentPermission

Weitere Einzelheiten finden Sie in der Dokumentation zur API "ObjectGridAuthorization".

### MapPermission

Die öffentliche Klasse "com.ibm.websphere.objectgrid.security.MapPermission" stellt Berechtigungen für die ObjectGrid-Ressourcen, insbesondere die Methoden der Schnittstellen "ObjectMap" und "JavaMap", dar. WebSphere eXtreme Scale definiert die folgenden Berechtigungszeichenfolgen für den Zugriff auf die Methoden der Schnittstellen "ObjectMap" und "JavaMap":

1. **read**: Berechtigung zum Lesen der Daten aus der Map. Die ganzzahlige Konstante wird mit `MapPermission.READ` definiert.
2. **write**: Berechtigung zum Aktualisieren der Daten in der Map. Die ganzzahlige Konstante wird mit `MapPermission.WRITE` definiert.
3. **insert**: Berechtigung zum Einfügen der Daten in die Map. Die ganzzahlige Konstante wird mit `MapPermission.INSERT` definiert.
4. **remove**: Berechtigung zum Entfernen der Daten aus der Map. Die ganzzahlige Konstante wird mit `MapPermission.REMOVE` definiert.
5. **invalidate**: Berechtigung zum Ungültigmachen der Daten in der Map. Die ganzzahlige Konstante wird mit `MapPermission.INVALIDATE` definiert.
6. **all**: Alle zuvor beschriebenen Berechtigungen: read, write, insert, remote und invalidate. Die ganzzahlige Konstante wird mit `MapPermission.ALL` definiert.

Weitere Einzelheiten finden Sie in der Dokumentation zur API "MapPermission".

Sie können ein MapPermission-Objekt erstellen, indem Sie den vollständig qualifizierten Namen der ObjectGrid-Map (im Format [ObjectGrid\_name].[ObjectMap\_name]) und die Berechtigungszeichenfolge oder den ganzzahligen Wert übergeben. Eine Berechtigungszeichenfolge kann eine durch Kommas getrennte Zeichenfolge der zuvor beschriebenen Berechtigungszeichenfolgen wie read, insert oder all sein. Ein ganzzahliger Berechtigungswert kann jeder der zuvor genannten ganzzahligen Berechtigungskonstanten oder ein mathematischer Wert sein, der sich aus mehreren ganzzahligen Berechtigungskonstanten zusammensetzt, z. B. `MapPermission.READ|MapPermission.WRITE`.

Die Berechtigung findet statt, wenn eine ObjectMap- oder JavaMap-Methode aufgerufen wird. Die Laufzeitumgebung von eXtreme Scale prüft die verschiedenen

Berechtigungen für verschiedene Methoden. Wenn dem Client die erforderlichen Berechtigungen nicht erteilt wurden, wird eine Ausnahme des Typs "AccessControlException" ausgegeben.

Table 12. Liste der Methoden und der erforderlichen MapPermissions

Berechtigung	ObjectMap/JavaMap
read	boolean containsKey(Object)
	boolean equals(Object)
	Object get(Object)
	Object get(Object, Serializable)
	List getAll(List)
	List getAll(List keyList, Serializable)
	List getAllForUpdate(List)
	List getAllForUpdate(List, Serializable)
	Object getForUpdate(Object)
	Object getForUpdate(Object, Serializable)
	public Object getNextKey(long)
write	Object put(Object key, Object value)
	void put(Object, Object, Serializable)
	void putAll(Map)
	void putAll(Map, Serializable)
	void update(Object, Object)
	void update(Object, Object, Serializable)
insert	public void insert (Object, Object)
	void insert(Object, Object, Serializable)
remove	Object remove (Object)
	void removeAll(Collection)
	void clear()
invalidate	public void invalidate (Object, boolean)
	void invalidateAll(Collection, boolean)
	void invalidateUsingKeyword(Serializable)
	int setTimeToLive(int)

Die Berechtigung basiert allein darauf, welche Methode verwendet wird, und nicht darauf, was die Methode wirklich tut. Eine Methode "put" kann beispielsweise einen Datensatz einfügen oder aktualisieren, je nachdem, ob der Datensatz vorhanden ist oder nicht. Diese Fälle werden jedoch nicht unterschieden.

Ein Operationstyp kann durch Kombinationen anderer Typen erreicht werden. Eine Aktualisierung kann beispielsweise durch die Kombination einer Entfernungs- und einer anschließenden Einfügeoperation erreicht werden. Berücksichtigen Sie diese Kombinationen, wenn Sie Ihre Berechtigungsrichtlinien gestalten.

### ObjectGridPermission

com.ibm.websphere.objectgrid.security.ObjectGridPermission stellt Berechtigungen für das ObjectGrid dar:

- Query: Berechtigung zum Erstellen einer Objektanfrage oder Entitätsanfrage. Die ganzzahlige Konstante wird mit ObjectGridPermission.QUERY definiert.
- Dynamic Map: Berechtigung zum Erstellen einer dynamischen Map auf der Basis der Map-Schablone. Die ganzzahlige Konstante wird mit ObjectGridPermission.DYNAMIC\_MAP definiert.

Weitere Einzelheiten finden Sie in der Dokumentation zur API "ObjectGridPermission".

In der folgenden Tabelle sind die Methoden und die jeweils erforderliche ObjectGridPermission zusammengefasst:

*Tabelle 13. Liste der Methoden und der erforderlichen ObjectGridPermission*

Berechtigungsaktion	Methoden
query	com.ibm.websphere.objectgrid.Session.createObjectQuery(String)
query	com.ibm.websphere.objectgrid.em.EntityManager.createQuery(String)
dynamicmap	com.ibm.websphere.objectgrid.Session.getMap(String)

### ServerMapPermission

ServerMapPermission stellt Berechtigungen für eine ObjectMap in einem Server dar. Der Name der Berechtigung ist der vollständige Name der ObjectGrid-Map. Sie hat drei Aktionen:

1. replicate: Berechtigung zum Replizieren einer Server-Map im nahen Cache.
2. dynamicIndex: Berechtigung für einen Client zum Erstellen oder Entfernen eines dynamischen Index in einem Server.

Weitere Einzelheiten finden Sie in der Dokumentation zur API "ServerMapPermission". Die detaillierten Methoden, die unterschiedliche ServerMapPermissions erfordern, sind in der folgenden Tabelle aufgelistet:

*Tabelle 14. Berechtigungen für eine ObjectMap in einem Server*

Berechtigungsaktion	Methoden
replicate	com.ibm.websphere.objectgrid.ClientReplicableMap.enableClientReplication(Mode, int[], ReplicationMapListener)
dynamicIndex	com.ibm.websphere.objectgrid.BackingMap.createDynamicIndex(String, boolean, String, DynamicIndexCallback)
dynamicIndex	com.ibm.websphere.objectgrid.BackingMap.removeDynamicIndex(String)

### AgentPermission

AgentPermission stellt Berechtigungen für die DataGrid-Agenten dar. Der Name der Berechtigung ist der vollständige Name der ObjectGrid-Map, und die Aktion ist eine durch Kommas getrennte Zeichenfolge, die sich aus den Namen der Agentenimplementierungsklassen bzw. Paketnamen zusammensetzt.

Weitere Informationen finden Sie in der Dokumentation zur API "AgentPermission".

Die folgenden Methoden in der Klasse "com.ibm.websphere.objectgrid.data-grid.AgentManager" erfordern AgentPermission.

```
com.ibm.websphere.objectgrid.datagrid.AgentManager#callMapAgent(MapGridAgent, Collection)
com.ibm.websphere.objectgrid.datagrid.AgentManager#callMapAgent(MapGridAgent)
```

```
com.ibm.websphere.objectgrid.datagrid.AgentManager#callReduceAgent(ReduceGridAgent, Collection)
com.ibm.websphere.objectgrid.datagrid.AgentManager#callReduceAgent(ReduceGridAgent, Collection)
```

## Berechtigungsmechanismus

WebSphere eXtreme Scale unterstützt zwei Arten von Berechtigungsmechanismen: JAAS-Berechtigung (Java Authentication and Authorization Service) und angepasste Berechtigung. Diese Mechanismen gelten für alle Berechtigungen. Die JAAS-Berechtigung erweitert die Java-Sicherheitsrichtlinien mit benutzerorientierten Zugriffsteuerungselementen. Berechtigungen können nicht nur auf der Basis des ausgeführten Codes, sondern auch danach erteilt werden, wer den Code ausführt. Die JAAS-Berechtigung ist Teil von SDK Version 1.4 und höher.

Außerdem unterstützt WebSphere eXtreme Scale die angepasste Berechtigung mit dem folgenden Plug-in:

- `ObjectGridAuthorization`: Angepasste Methode zur Berechtigung des Zugriffs auf alle Artefakte.

Sie können einen eigenen Berechtigungsmechanismus implementieren, wenn Sie die JAAS-Berechtigung nicht verwenden möchten. Wenn Sie einen angepassten Berechtigungsmechanismus verwenden, können Sie die Richtliniendatenbank, den Richtlinienserver oder Tivoli Access Manager für die Verwaltung der Berechtigungen verwenden.

Sie können den Berechtigungsmechanismus auf zwei Arten konfigurieren:

1. *XML-Konfiguration*: Sie können die ObjectGrid-XML-Datei verwenden, um ein ObjectGrid zu definieren und den Berechtigungsmechanismus `AUTHORIZATION_MECHANISM_JAAS` oder `AUTHORIZATION_MECHANISM_CUSTOM` festzulegen. Im Folgenden sehen Sie die Datei `secure-objectgrid-definition.xml`, die in der Unternehmensanwendung `ObjectGridSample` verwendet wird:

```
<objectGrids>
  <objectGrid name="secureClusterObjectGrid" securityEnabled="true"
    authorizationMechanism="AUTHORIZATION_MECHANISM_JAAS">
    <bean id="TransactionCallback"
      classname="com.ibm.websphere.samples.objectgrid.HeapTransactionCallback" />
    ...
  </objectGrids>
```

2. *Programmgesteuerte Konfiguration*: Wenn Sie ein ObjectGrid mit der Methode `ObjectGrid.setAuthorizationMechanism(int)` erstellen möchten, können Sie die folgende Methode aufrufen, um den Berechtigungsmechanismus festzulegen. Der Aufruf dieser Methode gilt nur dann für das lokale eXtreme-Scale-Programmiermodell, wenn Sie die ObjectGrid-Instanz direkt instanzieren:

```
/**
 * Berechtigungsmechanismus festlegen. Die Standardeinstellung ist
 * com.ibm.websphere.objectgrid.security.SecurityConstants.
 * AUTHORIZATION_MECHANISM_JAAS.
 * @param authMechanism Der Berechtigungsmechanismus für die Map.
 */
void setAuthorizationMechanism(int authMechanism);
```

## JAAS-Berechtigung

Ein `javax.security.auth.Subject`-Objekt stellt einen authentifizierten Benutzer dar. Ein Subject-Objekt setzt sich aus einer Gruppe von Principals zusammen, und jeder



Principal stellt eine Identität für diesen Benutzer dar. Beispielsweise kann ein Subject-Objekt einen Namens-Principal, z. B. Joe Smith, und einen Gruppen-Principal, z. B. manager, haben.

Mit der JAAS-Berechtigungsrichtlinie können Berechtigungen bestimmten Principals erteilt werden. WebSphere eXtreme Scale ordnet das Subject-Objekt dem aktuellen Zugriffssteuerungskontext zu. Für jeden Aufruf der ObjectMap- oder JavaMap-Methode bestimmt die Java-Laufzeitumgebung automatisch, ob die Richtlinie die erforderliche Berechtigung nur einem bestimmten Principal erteilt, und wenn ja, wird die Operation nur zugelassen, wenn das Subject-Objekt, das dem Zugriffssteuerungskontext zugeordnet ist, den angegebenen Principal enthält.

Sie müssen mit der Richtliniensyntax der Richtliniendatei vertraut sein. Eine ausführliche Beschreibung der JAAS-Berechtigung finden Sie in der Veröffentlichung "JAAS Reference Guide".

WebSphere eXtreme Scale hat eine spezielle Codebasis, die für die Überprüfung der JAAS-Berechtigung für die ObjectMap- und JavaMap-Methodenaufrufe verwendet wird. Diese spezielle Codebasis ist <http://www.ibm.com/com/ibm/ws/objectgrid/security/PrivilegedAction>. Verwenden Sie diese Codebasis, wenn Sie Principals ObjectMap- oder JavaMap-Berechtigungen erteilen. Dieser spezielle Code wurde erstellt, weil die JAR-Datei (Java-Archiv) für eXtreme Scale mit allen Berechtigungen ausgestattet ist.

Die Richtlinienschablone für die Erteilung der MapPermission-Berechtigung sieht wie folgt aus:

```
grant codeBase "http://www.ibm.com/com/ibm/ws/objectgrid/security/PrivilegedAction"
  <Principal field(s)>{
    permission com.ibm.websphere.objectgrid.security.MapPermission
      "[ObjectGrid_name].[ObjectMap_name]", "action";
    ....
    permission com.ibm.websphere.objectgrid.security.MapPermission
      "[ObjectGrid_name].[ObjectMap_name]", "action";
  };
```

Ein Beispiel für ein Principal-Feld sehen Sie im Folgenden:

```
principal Principal_class "principal_name"
```

In dieser Richtlinie werden einem bestimmten Principal nur Einfüge- und Leseberechtigungen für diese vier Maps erteilt. In der anderen Richtliniendatei, fullAccessAuth.policy, werden einem Principal alle Berechtigungen für diese Maps erteilt. Ändern Sie vor dem Ausführen der Anwendung "principal\_name" und "principal\_class" in die entsprechenden Werte. Der Wert für "principal\_name" richtet sich nach der Benutzer-Registry. Wird beispielsweise das lokale Betriebssystem als Benutzer-Registry verwendet, ist der Maschinenname MACH1, die Benutzer-ID user1 und der Principal-Name MACH1/user1.

Die JAAS-Berechtigungsrichtlinie kann direkt in die Java-Richtliniendatei oder eine separate JAAS-Berechtigungsdatei gestellt und dann mit dem JVM-Argument

```
-Djava.security.auth.policy=file:[JAAS_AUTH_POLICY_FILE]
```

oder mit der Eigenschaft

```
-Dauth.policy.url.x=file:[JAAS_AUTH_POLICY_FILE]
```

in der Datei "java.security" gesetzt werden.

## Angepasste ObjectGrid-Berechtigung

Das ObjectGridAuthorization-Plug-in wird verwendet, um ObjectGrid-, ObjectMap- und JavaMap-Zugriff auf die Principals, die durch ein Subject-Objekt dargestellt werden, auf angepasste Weise zu berechtigen. Eine typische Implementierung dieses Plug-ins ist der Abruf der Principals aus dem Subject-Objekt mit anschließender dahingehender Prüfung, ob die angegebenen Berechtigungen den Principals erteilt wurden oder nicht.

Die folgenden Berechtigungen können an die Methode "checkPermission(Subject, Permission)" übergeben werden:

1. MapPermission
2. ObjectGridPermission
3. AgentPermission
4. ServerMapPermission

Weitere Einzelheiten finden Sie in der Dokumentation zur API "ObjectGridAuthorization".

Das ObjectGridAuthorization-Plug-in kann auf die folgenden Arten konfiguriert werden:

1. *XML-Konfiguration*: Sie können die ObjectGrid-XML-Datei verwenden, um ein ObjectAuthorization-Plug-in zu definieren. Es folgt ein Beispiel:

```
<objectGrids>
  <objectGrid name="secureClusterObjectGrid" securityEnabled="true"
    authorizationMechanism="AUTHORIZATION_MECHANISM_CUSTOM">
    ...
    <bean id="ObjectGridAuthorization"
      className="com.acme.ObjectGridAuthorizationImpl" />
</objectGrids>
```

2. *Programmgesteuerte Konfiguration*: Wenn Sie ein ObjectGrid mit der API-Methode "ObjectGrid.setObjectGridAuthorization(ObjectGridAuthorization)" erstellen möchten, können Sie die folgende Methode aufrufen, um das Berechtigungs-Plug-in zu definieren. Diese Methode gilt nur für das lokale Programmiermodell von eXtreme Scale, wenn Sie die ObjectGrid-Instanz direkt instanzieren.

```
/**
 * Setzt die <code>ObjectGridAuthorization</code> für diese ObjectGrid-Instanz.
 * <p>
 * Bei der Übergabe von <code>null</code> an diese Methode wird ein zuvor definiertes
 * <code>ObjectGridAuthorization</code>-Objekt aus einem früheren Aufruf dieser Methode
 * entfernt. Außerdem wird damit angezeigt, dass dieses <code>ObjectGrid</code> keinem
 * <code>ObjectGridAuthorization</code>-Objekt zugeordnet ist.
 * <p>
 * Diese Methode sollte nur verwendet werden, wenn die ObjectGrid-Sicherheit aktiviert ist. Wenn
 * die ObjectGrid-Sicherheit inaktiviert ist, wird das bereitgestellte <code>ObjectGridAuthorization</code>-Objekt
 * nicht verwendet.
 * <p>
 * Ein <code>ObjectGridAuthorization</code>-Plug-in kann verwendet werden, um den
 * Zugriff auf das ObjectGrid und die Maps zu berechtigen. Weitere Einzelheiten finden Sie
 * in der Dokumentation zu <code>ObjectGridAuthorization</code>.
 *
 * <p>
 * Ab Extended Deployment 6.1 ist <code>setMapAuthorization</code> veraltet und
 * <code>setObjectGridAuthorization</code> die empfohlene Methode. Wenn jedoch
 * das <code>MapAuthorization</code>-Plug-in und das <code>ObjectGridAuthorization</code>-Plug-in
 * verwendet werden, verwendet ObjectGrid das bereitgestellte <code>MapAuthorization</code>-Plug-in,
 * um Map-Zugriffe zu berechtigen, obwohl es veraltet ist.
```

```

* <p>
* Zur Vermeidung von Ausnahmen des Typs <code>IllegalStateException</code> muss
* diese Methode vor der Methode <code>initialize()</code> aufgerufen werden.
* Beachten Sie außerdem, dass die <code>getSession</code>-Methoden die Methode
* <code>initialize()</code> implizit aufrufen, wenn sie noch von der Anwendung
* aufgerufen werden muss.
*
* @param ogAuthorization Das <code>ObjectGridAuthorization</code>-Plug-in.
*
* @throws IllegalStateException, wenn diese Methode nach dem Aufruf der
* Methode <code>initialize()</code> aufgerufen wird.
*
* @see #initialize()
* @see ObjectGridAuthorization
* @since WAS XD 6.1
*/
void setObjectGridAuthorization(ObjectGridAuthorization ogAuthorization);

```

## ObjectGridAuthorization implementieren

Die Methode "boolean checkPermission(Subject subject, Permission permission)" der Schnittstelle "ObjectGridAuthorization" wird von der Laufzeitumgebung von WebSphere eXtreme Scale aufgerufen, um zu prüfen, ob das übergebene Subject-Objekt die übergebenen Berechtigungen besitzt. Die Implementierung der Schnittstelle "ObjectGridAuthorization" gibt "true" zurück, wenn das Objekt die Berechtigung besitzt, und "false", wenn nicht.

Eine typische Implementierung dieses Plug-ins ist der Abruf der Principals aus dem Subject-Objekt mit anschließender dahingehender Prüfung, ob die angegebenen Berechtigungen den Principals erteilt wurden oder nicht, unter Verwendung der angegebenen Richtlinien. Diese Richtlinien werden von Benutzern definiert. Die Richtlinien können beispielsweise in einer Datenbank, einer unverschlüsselten Datei oder in einem Richtlinienserver von Tivoli Access Manager definiert werden.

Sie können einen Richtlinienserver von Tivoli Access Manager beispielsweise verwenden, um die Berechtigungsrichtlinie zu verwalten, und die zugehörige API, um den Zugriff zu berechtigen. Informationen zur Verwendung der Berechtigungs-APIs von Tivoli Access Manager Authorization finden Sie in der Veröffentlichung "IBM Tivoli Access Manager Authorization Java Classes Developer Reference".

Diese Beispielimplementierung basiert auf den folgenden Annahmen:

1. Es wird nur die Berechtigung für MapPermission geprüft. Für andere Berechtigungen wird immer "true" (wahr) zurückgegeben.
2. Das Subject-Objekt enthält einen com.tivoli.mts.PDPrincipal-Principal.
3. Im Richtlinienserver von Tivoli Access Manager sind die folgenden Berechtigungen für das ObjectMap- bzw. JavaMap-Namensobjekt definiert. Der Name des im Richtlinienserver definierten Objekts muss dem ObjectMap- bzw. JavaMap-Namen entsprechen und das Format [ObjectGrid\_name].[ObjectMap\_name] haben. Die Berechtigung ist das erste Zeichen der Berechtigungszeichenfolgen, die in der MapPermission-Berechtigung definiert sind. Die im Richtlinienserver definierte Berechtigung "r" stellt beispielsweise die Leseberechtigung (read) für die ObjectMap-Map dar.

Das folgende Code-Snippet veranschaulicht, wie die Methode "checkPermission" implementiert wird:

```

/**
 * @see com.ibm.websphere.objectgrid.security.plugins.
 * MapAuthorization#checkPermission
 * (javax.security.auth.Subject, com.ibm.websphere.objectgrid.security.
 * MapPermission)
 */
public boolean checkPermission(final Subject subject,
    Permission p) {

```

```

// Für andere Berechtigungen als MapPermission-Berechtigungen, immer Berechtigung erteilen.
if (!(p instanceof MapPermission)){
    return true;
}

MapPermission permission = (MapPermission) p;

String[] str = permission.getParsedNames();

StringBuffer pdPermissionStr = new StringBuffer(5);
for (int i=0; i<str.length; i++) {
    pdPermissionStr.append(str[i].substring(0,1));
}

PDPermission pdPerm = new PDPermission(permission.getName(),
pdPermissionStr.toString());

Set principals = subject.getPrincipals();

Iterator iter= principals.iterator();
while (iter.hasNext()) {
    try {
        PDPrincipal principal = (PDPrincipal) iter.next();
        if (principal.implies(pdPerm)) {
            return true;
        }
    }
    catch (ClassCastException cce) {
        // Ausnahme behandeln
    }
}
return false;
}

```

---

## Grid-Authentifizierung

Das sichere Tokenmanager-Plug-in ist ein weiteres Plug-in, das für die Authentifizierung zwischen Servern verwendet wird. Das sichere Tokenmanager-Plug-in wird durch die Schnittstelle "com.ibm.websphere.objectgrid.security.plugins.SecureTokenManager" dargestellt.

Die Methode "generateToken(Object)" verwendet ein Objekt und generiert anschließend ein Token, das von anderen nicht interpretiert werden kann. Die Methode "verifyTokens(byte[])" führt den umgekehrten Prozess aus. Sie konvertiert das Token zurück in das ursprüngliche Objekt.

Eine einfache SecureTokenManager-Implementierung verwendet einen einfachen Verschlüsselungsalgorithmus, wie z. B. einen XOR-Algorithmus (exklusives Oder), um das Objekt in serialisierter Form zu verschlüsseln, und anschließend den entsprechenden Entschlüsselungsalgorithmus, um das Token zu entschlüsseln. Diese Implementierung ist nicht sicher und anfällig für Attacken.

### Standardimplementierung von WebSphere eXtreme Scale

WebSphere eXtreme Scale stellt eine sofort verfügbare Implementierung für diese Schnittstelle bereit. Diese Standardimplementierung verwendet ein Schlüsselpaar, um die Signatur zu signieren und zu prüfen, und einen geheimen Schlüssel, um den Inhalt zu verschlüsseln. Jeder Server hat einen JCKES-Keystore, in dem das Schlüsselpaar (privater und öffentlicher Schlüssel) und der geheime Schlüssel gespeichert werden. Der Keystore muss ein JCKES-Keystore sein, damit geheime

Schlüssel gespeichert werden können. Diese Schlüssel werden verwendet, um die Shared-Secret-Zeichenfolge auf Senderseite zu verschlüsseln und zu signieren bzw. zu prüfen. Außerdem wird dem Token eine Verfallszeit zugeordnet. Auf Empfängerseite werden die Daten geprüft, entschlüsselt und mit der Shared-Secret-Zeichenfolge des Empfängers verglichen. Es sind keine SSL-Kommunikationsprotokolle (Secure Sockets Layer) zwischen einem Serverpaar für die Authentifizierung erforderlich, weil die privaten und öffentlichen Schlüssel demselben Zweck dienen. Wenn die Serverkommunikation jedoch nicht verschlüsselt ist, können die Daten einfach durch Ansicht der Kommunikation gestohlen werden. Da das Token relativ bald verfällt, ist das Sicherheitsrisiko durch Attacken durch Nachrichtenaufzeichnung und -wiederholung minimal. Das Risiko ist erheblich geringer, wenn alle Server hinter einer Firewall implementiert werden.

Dieser Ansatz hat den Nachteil, dass die Administratoren von WebSphere eXtreme Scale Schlüssel generieren und an alle Server übermitteln müssen, was während des Transports der Schlüssel zu Sicherheitsverletzungen führen kann.

---

## Lokale Sicherheit

WebSphere eXtreme Scale stellt mehrere Sicherheitsendpunkte für die Integration angepasster Mechanismen bereit. Im lokalen Programmiermodell ist die Hauptsicherheitsfunktion Berechtigung. Authentifizierung wird nicht unterstützt. Sie müssen die Authentifizierung außerhalb von WebSphere Application Server durchführen. Es werden jedoch Plug-ins für das Anfordern und Validieren von Subject-Objekten bereitgestellt.

### Sicherheit aktivieren

In der folgenden Liste sind die beiden Methoden für die Aktivierung der lokalen Sicherheit aufgeführt:

- **XML-Konfiguration:** Sie können die ObjectGrid-XML-Datei verwenden, um ein ObjectGrid zu definieren und die Sicherheit für dieses ObjectGrid zu aktivieren. Die folgende Datei ist die Datei "secure-objectgrid-definition.xml", die im Unternehmensanwendungsbeispiel "ObjectGridSample" verwendet wird. In dieser XML-Datei wird die Sicherheit durch Setzen des Attributs "securityEnabled" auf "true" aktiviert:

```
<objectGrids>
  <objectGrid name="secureClusterObjectGrid" securityEnabled="true"
    authorizationMechanism="AUTHORIZATION_MECHANISM_JASS">
    ...
  </objectGrids>
```

- **Programmierung:** Wenn Sie ein ObjectGrid über die API-Methode "ObjectGrid.setSecurityEnabled" erstellen möchten, rufen Sie die folgende Methode in der Schnittstelle "ObjectGrid" auf, um die Sicherheit zu aktivieren.

```
/**
 * ObjectGrid-Sicherheit aktivieren
 */
void setSecurityEnabled();
```

### Authentifizierung

Im lokalen Programmiermodell stellt eXtreme Scale kein Authentifizierungsverfahren bereit, sondern stützt sich bei der Authentifizierung auf die Umgebung, d. h. Anwendungsserver oder Anwendungen. Wenn eXtreme Scale in WebSphere Application Server oder WebSphere Extended Deployment verwendet wird, können Anwendungen das Sicherheitsauthentifizierungsverfahren von WebSphere Applica-

tion Server verwenden. Wenn eXtreme Scale in einer J2SE-Umgebung (Java 2 Platform, Standard Edition) ausgeführt wird, muss die Anwendung die Authentifizierung mit JAAS-Authentifizierung (Java Authentication and Authorization Service) oder anderen Authentifizierungsverfahren verwalten. Weitere Informationen zur Verwendung der JAAS-Authentifizierung finden Sie in der Veröffentlichung JAAS Reference Guide. Der Vertrag zwischen einer Anwendung und einer ObjectGrid-Instanz ist das Objekt "javax.security.auth.Subject". Nachdem der Client vom Anwendungsserver oder von der Anwendung authentifiziert wurde, kann die Anwendung das authentifizierte Objekt "javax.security.auth.Subject" abrufen und dieses Subject-Objekt verwenden, um eine Sitzung von der ObjectGrid-Instanz abzurufen, indem sie die Methode "ObjectGrid.getSession(Subject)" aufruft. Dieses Subject-Objekt wird verwendet, um Zugriff auf die Map-Daten zu berechtigen. Dieser Vertrag wird als Subject-Übergabemechanismus bezeichnet. Das folgende Beispiel veranschaulicht die API "ObjectGrid.getSession(Subject)".

```
/**
 * Diese API ermöglicht dem Cache die Verwendung eines bestimmten Subject-Objekts an Stelle des
 * im ObjectGrid konfigurierten Subject-Objekts für den Abruf einer Sitzung.
 * @param Subject-Objekt
 * @return Instanz des Session-Objekts
 * @throws ObjectGridException
 * @throws TransactionCallbackException
 * @throws InvalidSubjectException Das übergebene Subject-Objekt ist nach dem
 * SubjectValidation-Mechanismus nicht gültig.
 */
public Session getSession(Subject subject)
throws ObjectGridException, TransactionCallbackException, InvalidSubjectException;
```

Die Methode "ObjectGrid.getSession()" in der Schnittstelle "ObjectGrid" kann auch für das Abrufen eines Session-Objekts verwendet werden:

```
/**
 * Diese Methode gibt ein Session-Objekt zurück, das jeweils von einem einzigen Thread verwendet werden kann.
 * Dieses Session-Objekt kann nicht von Threads gemeinsam genutzt werden, ohne ein kritisches
 * Abschnitt um es herum zu erstellen. Während das Basis-Framework das Verschieben des Objekts
 * zwischen Threads zulässt, können die Schnittstellen "TransactionCallback" und "Loader" diese
 * Verwendung verhindern, insbesondere in J2EE-Umgebungen. Wenn die Sicherheit aktiviert ist,
 * verwendet diese Methode SubjectSource, um ein Subject-Objekt abzurufen.
 *
 * Wenn die Methode "initialize" nicht vor dem ersten Aufruf von "getSession"
 * aufgerufen wurde, findet eine implizite Initialisierung statt. Diese
 * Initialisierung stellt sicher, dass die gesamte Konfiguration abgeschlossen
 * ist, bevor sie zur Laufzeit benötigt wird.
 *
 * @see #initialize()
 * @return Instanz des Session-Objekts
 * @throws ObjectGridException
 * @throws TransactionCallbackException
 * @throws IllegalStateException, wenn diese Methode nach dem Aufruf
 * der Methode destroy() aufgerufen wird.
 */
public Session getSession()
throws ObjectGridException, TransactionCallbackException;
```

Wie in der API-Dokumentation beschrieben, verwendet diese Methode bei aktivierter Sicherheit das SubjectSource-Plug-in, um ein Subject-Objekt abzurufen. Das SubjectSource-Plug-in ist eines der Sicherheits-Plug-ins, die in eXtreme Scale für die Unterstützung der Weitergabe von Subject-Objekten definiert sind. Weitere Informationen hierzu finden Sie in der Dokumentation zu den sicherheitsbezogenen Plug-ins. Die Methode "getSession(Subject)" kann nur für die lokale ObjectGrid-Instanz aufgerufen werden. Wenn Sie die Methode "getSession(Subject)" auf Client-seite in einer verteilten eXtreme-Scale-Konfiguration aufrufen, wird eine Ausnahme des Typs "IllegalStateException" ausgelöst.

## Sicherheits-Plug-ins

WebSphere eXtreme Scale stellt zwei Sicherheits-Plug-ins bereit, die sich auf den Subject-Übergabemechanismus beziehen: das SubjectSource-Plug-in und das SubjectValidation-Plug-in.

### SubjectSource-Plug-in

Das SubjectSource-Plug-in, das von der Schnittstelle "com.ibm.websphere.objectgrid.security.plugins.SubjectSource" dargestellt wird, ist ein Plug-in, das verwendet wird, um ein Subject-Objekt von einer Umgebung mit eXtreme Scale abzurufen. Diese Umgebung kann eine Anwendung sein, die ObjectGrid verwendet, oder ein Anwendungsserver, in dem die Anwendung ausgeführt wird. Betrachten Sie das SubjectSource-Plug-in als Alternative zum Subject-Übergabemechanismus. Mit dem Subject-Übergabemechanismus ruft die Anwendung das Subject-Objekt ab und verwendet es, um das ObjectGrid-Session-Objekt abzurufen. Mit dem SubjectSource-Plug-in ruft die Laufzeitumgebung von eXtreme Scale das Subject-Objekt ab und verwendet es, um das Session-Objekt abzurufen. Der Subject-Übergabemechanismus übergibt die Steuerung der Subject-Objekte an die Anwendungen, wohingegen das SubjectSource-Plug-in die Anwendungen vom Abrufen des Subject-Objekts befreit. Sie können das SubjectSource-Plug-in verwenden, um ein Subject-Objekt abzurufen, das einen eXtreme-Scale-Client darstellt, der für die Berechtigung verwendet wird. Wenn die Methode "ObjectGrid.getSession" aufgerufen wird, löst die Subject-Methode "getSubject" eine Ausnahme des Typs "ObjectGridSecurityException" aus, wenn die Sicherheit aktiviert ist. WebSphere eXtreme Scale stellt eine Standardimplementierung dieses Plug-ins bereit: com.ibm.websphere.objectgrid.security.plugins.builtins.WSSubjectSourceImpl. Diese Implementierung kann verwendet werden, um ein Caller-Subject-Objekt oder ein RunAs-Subject-Objekt vom Thread abzurufen, wenn eine Anwendung in WebSphere Application Server ausgeführt wird. Sie können diese Klasse als SubjectSource-Implementierungsklasse in Ihrer ObjectGrid-XML-Deskriptordatei konfigurieren, wenn eXtreme Scale in WebSphere Application Server verwendet wird. Das folgende Code-Snippet zeigt den Hauptablauf der Methode "WSSubjectSourceImpl.getSubject":

```
Subject s = null;
try {
    if (finalType == RUN_AS_SUBJECT) {
        // get the RunAs subject
        s = com.ibm.websphere.security.auth.WSSubject.getRunAsSubject();
    }
    else if (finalType == CALLER_SUBJECT) {
        // get the callersubject
        s = com.ibm.websphere.security.auth.WSSubject.getCallerSubject();
    }
}
catch (WSSecurityException wse) {
    throw new ObjectGridSecurityException(wse);
}

return s;
```

Weitere Einzelheiten finden Sie in der API-Dokumentation zum SubjectSource-Plug-in und zur WSSubjectSourceImpl-Implementierung.

### SubjectValidation-Plug-in

Das SubjectValidation-Plug-in, das durch die Schnittstelle "com.ibm.websphere.objectgrid.security.plugins.SubjectValidation" dargestellt wird, ist ein weiteres Sicherheits-Plug-in. Das SubjectValidation-Plug-in kann verwendet werden, um zu prü-

fen, ob ein `javax.security.auth.Subject`-Objekt, das an das `ObjectGrid` übergeben oder vom `SubjectSource`-Plug-in abgerufen wird, ein gültiges `Subject`-Objekt ist, das nicht manipuliert wurde.

Die Methode `SubjectValidation.validateSubject(Subject)` in der Schnittstelle `SubjectValidation` akzeptiert ein `Subject`-Objekt und gibt ein `Subject`-Objekt zurück. Ob ein `Subject`-Objekt als gültig eingestuft wird und welches `Subject`-Objekt zurückgegeben wird, liegt rein im Ermessen Ihrer Implementierungen. Wenn das `Subject`-Objekt nicht gültig ist, wird eine Ausnahme des Typs `InvalidSubjectException` ausgelöst.

Sie können dieses Plug-in verwenden, wenn Sie das `Subject`-Objekt, das an diese Methode übergeben wird, nicht anerkennen. Dieser Fall tritt nur selten ein, wenn man bedenkt, dass Sie den Anwendungsentwicklern vertrauen, die den Code zum Abrufen des `Subject`-Objekts entwickeln.

Für die Implementierung dieses Plug-ins wird die Unterstützung des `Subject`-Objekterstellers benötigt, da nur der Ersteller weiß, ob das `Subject`-Objekt manipuliert wurde. Es kann jedoch vorkommen, dass ein `Subject`-Ersteller nicht weiß, ob das `Subject`-Objekt manipuliert wurde. In diesem Fall ist das Plug-in nicht hilfreich.

WebSphere eXtreme Scale stellt eine Standardimplementierung von `SubjectValidation` bereit: `com.ibm.websphere.objectgrid.security.plugins.builtins.WSSubjectValidationImpl`. Sie können diese Implementierung verwenden, um das von WebSphere Application Server authentifizierte `Subject`-Objekt zu prüfen. Sie können diese Klasse als `SubjectValidation`-Implementierungsklasse konfigurieren, wenn Sie eXtreme Scale in WebSphere Application Server verwenden. Die `WSSubjectValidationImpl`-Implementierung stuft ein `Subject`-Objekt nur dann als gültig ein, wenn das Berechtigungsnachweistoken, das diesem `Subject`-Objekt zugeordnet ist, nicht manipuliert wurde. Andere Komponenten des `Subject`-Objekts können geändert werden. Die `WSSubjectValidationImpl`-Implementierung fordert das ursprüngliche `Subject`-Objekt, das dem Berechtigungsnachweistoken entspricht, von WebSphere Application Server an und gibt das ursprüngliche `Subject`-Objekt als validiertes `Subject`-Objekt zurück. Deshalb haben Änderungen, die an anderen Inhaltskomponenten des `Subject`-Objekts als dem Berechtigungsnachweistoken vorgenommen werden, keine Auswirkungen. Das folgende Code-Snippet veranschaulicht den Basisablauf von `WSSubjectValidationImpl.validateSubject(Subject)`.

```
// LoginContext-Objekt mit Schema-WSLogin erstellen
// und einen Callback-Handler zurückgeben.
LoginContext lc = new LoginContext("WSLogin",
new WSCredTokenCallbackHandlerImpl(subject));

// Wenn diese Methode aufgerufen wird, werden Methoden des Callback-Handlers aufgerufen,
// um den Benutzer anzumelden.
lc.login();

// Subject-Objekt vom LoginContext abrufen
return lc.getSubject();
```

Im vorherigen Code-Snippet wird ein `Callback-Handler`-Objekt für das Berechtigungsnachweistoken, `WSCredTokenCallbackHandlerImpl`, mit dem zu validierenden `Subject`-Objekt erstellt. Anschließend wird ein `LoginContext`-Objekt mit dem `WSLogin`-Objekt für das Anmeldeschema erstellt. Wenn die Methode `lc.login` aufgerufen wird, ruft die Sicherheit von WebSphere Application Server das Berechtigungsnachweistoken aus dem `Subject`-Objekt ab und gibt dann das entsprechende `Subject`-Objekt als validiertes `Subject`-Objekt zurück.



Weitere Einzelheiten finden Sie in den Java-API-Beschreibungen für die SubjectValidation- und WSSubjectValidationImpl-Implementierungen.

## Plug-in-Konfiguration

Sie können das SubjectValidation-Plug-in und das SubjectSource-Plug-in auf zwei Arten konfigurieren:

- **XML-Konfiguration:** Sie können die ObjectGrid-XML-Datei verwenden, um ein ObjectGrid zu definieren und diese beiden Plug-ins zu konfigurieren. Im Folgenden sehen Sie ein Beispiel, in dem die Klasse "WSSubjectSourceImpl" als SubjectSource-Plug-in und die Klasse "WSSubjectValidation" als SubjectValidation-Plug-in konfiguriert wird:

```
<objectGrids>
  <objectGrid name="secureClusterObjectGrid" securityEnabled="true"
    authorizationMechanism="AUTHORIZATION_MECHANISM_JAAS">
    <bean id="SubjectSource"
      className="com.ibm.websphere.objectgrid.security.plugins.builtins.
        WSSubjectSourceImpl" />
    <bean id="SubjectValidation"
      className="com.ibm.websphere.objectgrid.security.plugins.builtins.
        WSSubjectValidationImpl" />
    <bean id="TransactionCallback"
      className="com.ibm.websphere.samples.objectgrid.
        HeapTransactionCallback" />
    ...
  </objectGrids>
```

- **Programmierung:** Wenn Sie ein ObjectGrid über APIs erstellen möchten, können Sie die folgenden Methoden aufrufen, um das SubjectSource- oder SubjectValidation-Plug-in zu konfigurieren.

```
**
* SubjectValidation-Plug-in für diese ObjectGrid-Instanz konfigurieren.
* Ein SubjectValidation-Plug-in kann verwendet werden, um das Subject-Objekt
* zu validieren, das als gültiges Subject-Objekt übergeben wird. Weitere
* Einzelheiten finden Sie unter {@link SubjectValidation}.
* @param subjectValidation Das SubjectValidation-Plug-in
*/
void setSubjectValidation(SubjectValidation subjectValidation);

/**
* SubjectSource-Plug-in konfigurieren. Ein SubjectSource-Plug-in kann verwendet
* werden, um ein Subject-Objekt von der Umgebung abzurufen, um den
* ObjectGrid-Client darzustellen.
*
* @param source Das SubjectSource-Plug-in
*/
void setSubjectSource(SubjectSource source);
```

## Eigenen JAAS/Authentifizierungscode schreiben

Sie können Ihren eigenen JAAS-Authentifizierungscode (Java Authentication and Authorization Service) schreiben, der für die Authentifizierung zuständig ist. Sie müssen eigene Anmeldemodule schreiben und die Anmeldemodule anschließend für Ihr Authentifizierungsmodul konfigurieren.

Das Anmeldemodul empfängt Informationen über einen Benutzer und authentifiziert den Benutzer. Diese Informationen können beliebige Informationen sein, die den Benutzer identifizieren, z. B. eine Benutzer-ID und ein Kennwort, ein Client-zertifikat usw. Nach dem Empfang der Informationen prüft das Anmeldemodul, ob

die Informationen ein gültiges Subject-Objekt darstellen, und erstellt dann ein Subject-Objekt. Derzeit sind mehrere Implementierungen von Anmeldemodulen öffentlich verfügbar.

Nachdem Sie ein Anmeldemodul geschrieben haben, konfigurieren Sie dieses für die Verwendung in der Laufzeitumgebung. Sie müssen ein JAAS-Anmeldemodul konfigurieren. Dieses Anmeldemodul enthält das Anmeldemodul selbst und das zugehörige Authentifizierungsschema. Beispiel:

```
FileLogin
{
    com.acme.auth.FileLoginModule required
};
```

Das Authentifizierungsschema ist "FileLogin", und das Anmeldemodul ist "com.acme.auth.FileLoginModule". Das Token "required" zeigt an, dass das Modul "FileLoginModule" diese Anmeldung authentifizieren muss, da ansonsten das gesamte Schema scheitert.

Die Definition der Konfigurationsdatei für das JAAS-Anmeldemodul kann auf eine der folgenden Arten erfolgen:

- Sie können die Konfigurationsdatei für das JAAS-Anmeldemodul in der Eigenschaft "login.config.url" in der Datei "java.security" definieren. Beispiel:  
login.config.url.1=file:\${java.home}/lib/security/file.login
- Sie können die Konfigurationsdatei für das JAAS-Anmeldemodul über die Befehlszeile mit den JVM-Argumenten **-Djava.security.auth.login.config** konfigurieren. Beispiel: `-Djava.security.auth.login.config ==$JAVA_HOME/lib/security/file.login`

Wenn Ihr Code in WebSphere Application Server ausgeführt wird, müssen Sie die JAAS-Anmeldung über die Administrationskonsole konfigurieren und diese Anmeldekonfiguration in der Konfiguration des Anwendungsservers speichern. Weitere Einzelheiten hierzu finden Sie in der Beschreibung der Anmeldekonfiguration für Java Authentication and Authorization Service.

---

## Kapitel 7. Verwaltungs-API zum Starten eines integrierten eXtreme-Scale-Containerservers verwenden

Mit WebSphere eXtreme Scale können Sie eine programmgesteuerte API verwenden, um den Lebenszyklus integrierter Server und Container zu verwalten. Sie können den Server über das Programm mit jeder der Optionen konfigurieren, die Sie auch über die Befehlszeilenoptionen oder dateibasierten Servereigenschaften konfigurieren können. Sie können den integrierten Server als Containerserver und/oder Katalogservice konfigurieren.

### Vorbereitungen

Sie müssen eine Methode für die Ausführung von Code über eine bereits vorhandene Java Virtual Machine haben. Die eXtreme-Scale-Klassen müssen über die Baumstruktur der Klassenlader verfügbar sein.

### Warum und wann dieser Vorgang ausgeführt wird

Viele Verwaltungs-Tasks können über die Verwaltungs-API ausgeführt werden. Die API wird häufig als interner Server für die Speicherung des Webanwendungsstatus eingesetzt. Der Webserver kann als integrierter WebSphere eXtreme Scale-Server gestartet werden, den Containerserver dem Katalogservice melden, und anschließend wird der Server als Member eines größeren verteilten Grids hinzugefügt. Diese Verwendung kann aus einem ansonsten flüchtigen Datenspeicher einen skalierbaren und hoch verfügbaren Datenspeicher machen.

Sie können den vollständigen Lebenszyklus eines integrierten eXtreme-Scale-Servers über das Programm steuern. Die Beispiele sind so generisch wie möglich und enthalten nur direkte Codemuster für die beschriebenen Schritte.

1. Rufen Sie das Objekt "ServerProperties" aus der Klasse "ServerFactory" ab, und konfigurieren Sie alle erforderlichen Optionen.

Jeder eXtreme-Scale-Server besitzt eine Reihe konfigurierbarer Eigenschaften. Wenn ein Server über die Befehlszeile gestartet wird, werden diese Eigenschaften auf Standardwerte gesetzt, aber Sie können mehrere Eigenschaften überschreiben, indem Sie eine externe Quelle oder Datei angeben. Im integrierten Bereich können Sie die Eigenschaften direkt mit einem ServerProperties-Objekt setzen. Sie müssen diese Eigenschaften setzen, bevor Sie eine Serverinstanz aus der Klasse "ServerFactory" abrufen. Das folgende Beispiel-Snippet ruft ein ServerProperties-Objekt ab, setzt das Feld "CatalogServiceBootstrap" und initialisiert mehrere optionale Servereinstellungen. Eine Liste der konfigurierbaren Einstellungen finden Sie in der API-Dokumentation.

```
ServerProperties props = ServerFactory.getServerProperties();
props.setCatalogServiceBootstrap("host:port"); // Für die Verbindungsherstellung zu einem bestimmten Katalogserver erforderlich
props.setServerName("ServerOne"); // Server benennen
props.setTraceSpecification("com.ibm.ws.objectgrid=all=enabled"); // Trace-Spezifikation festlegen
```

2. Wenn der Server ein Katalogserver sein soll, rufen Sie das Objekt "CatalogServerProperties" ab.

Jeder integrierte Server kann ein Katalogserver und/oder ein Containerserver sein. Der folgende Beispielcode ruft das Objekt "CatalogServerProperties" ab, aktiviert die Katalogserviceoption und konfiguriert verschiedene Einstellungen des Katalogservice.

```
CatalogServerProperties catalogProps = ServerFactory.getCatalogProperties();
catalogProps.setCatalogServer(true); // standardmäßig false; erforderlich für die Einstellung als Katalogservice
catalogProps.setQuorum(true); // Quorum aktivieren/inaktivieren
```

3. Rufen Sie eine Server-Instanz aus der Klasse "ServerFactory" ab. Die Server-Instanz ist ein prozessbezogenes Singleton, das für die Verwaltung der Zugehörigkeiten im Grid zuständig ist. Nach der Instanzierung dieser Instanz ist dieser Prozess verbunden und zusammen mit den anderen Servern im Grid hoch verfügbar. Das folgende Beispiel veranschaulicht, wie die Server-Instanz erstellt wird:

```
Server server = ServerFactory.getInstance();
```

Wenn Sie sich das vorherige Beispiel ansehen, stellen Sie fest, dass die Klasse "ServerFactory" eine statische Methode bereitstellt, die eine Server-Instanz zurückgibt. Die Klasse "ServerFactory" ist die einzige geplante Schnittstelle für das Abrufen einer Server-Instanz. Deshalb stellt die Klasse sicher, dass die Instanz ein Singleton bzw. die einzige Instanz für jede JVM bzw. jeden isolierten Klassenlader ist. Die Methode "getInstance" initialisiert die Server-Instanz. Sie müssen alle Servereigenschaften konfigurieren, bevor Sie die Instanz initialisieren. Die Klasse "Server" ist für die Erstellung neuer Container-Instanzen zuständig. Sie können die Klassen "ServerFactory" und "Server" verwenden, um den Lebenszyklus der integrierten Server-Instanz zu verwalten.

4. Starten Sie eine Container-Instanz über die Server-Instanz.

Bevor Shards an einen integrierten Server verteilt werden können, müssen Sie einen Container im Server erstellen. Die Schnittstelle "Server" besitzt eine Methode "createContainer", die das Argument "DeploymentPolicy" akzeptiert. Im folgenden Beispiel wird die Serverinstanz, die Sie zum Erstellen eines Containers abgerufen haben, mit einer erstellten DeploymentPolicy-Datei verwendet. Beachten Sie, dass Container für die Serialisierung einen Klassenlader erfordern, dem die Binärdateien der Anwendung zur Verfügung stehen. Sie können diese Binärdateien bereitstellen, indem Sie die Methode "createContainer" aufrufen und in diesem Aufruf den Klassenlader für den Thread-Kontext auf den Klassenlader setzen, den Sie verwenden möchten.

```
DeploymentPolicy policy = DeploymentPolicyFactory.createDeploymentPolicy(new URL("file://urltodeployment.xml"),
    new URL("file://urltoobjectgrid.xml"));
Container container = server.createContainer(policy);
```

5. Entfernen und bereinigen Sie einen Container.

Sie können einen Containerserver entfernen und bereinigen, indem Sie die Methode "teardown" für die abgerufene Container-Instanz ausführen. Bei der Ausführung der Methode "teardown" für einen Container wird der Container bereinigt und aus dem integrierten Server entfernt.

Die Bereinigung des Containers beinhaltet die Verlagerung und Umrüstung aller Shards dieses Containers. Jeder Server kann mehrere Container und Shards enthalten. Die Bereinigung eines Containers hat keine Auswirkung auf den Lebenszyklus der übergeordneten Server-Instanz. Das folgende Beispiel veranschaulicht, wie die Methode "teardown" für einen Server ausgeführt wird. Die Methode "teardown" wird über die Schnittstelle "ContainerMBean" bereitgestellt. Wenn Sie keinen Zugriff mehr über das Programm auf diesen Container haben und die Schnittstelle "ContainerMBean" verwenden, können Sie den Container mit der zugehörigen MBean trotzdem bereinigen. Die Schnittstelle "Container" enthält auch eine Methode "terminate". Verwenden Sie diese Methode nur, wenn es unbedingt erforderlich ist. Diese Methode ist konsequenter und koordiniert die entsprechende Shard-Verlagerung und -Bereinigung nicht.

```
container.teardown();
```

6. Stoppen Sie den integrierten Server.

Wenn Sie einen integrierten Server stoppen, stoppen Sie auch alle Container und Shards, die im Server ausgeführt werden. Wenn Sie einen integrierten Server stoppen, müssen Sie alle offenen Verbindungen bereinigen und alle Shards verlagern oder umrüsten. Das folgende Beispiel veranschaulicht, wie ein Server gestoppt wird und die Methode "waitFor" in der Schnittstelle "Server" verwendet wird, um sicherzustellen, dass die Server-Instanz vollständig beendet wird. Ähnlich wie im Containerbeispiel wird die Methode "stopServer" über die Schnittstelle "ServerMBean" bereitgestellt. Mit dieser Schnittstelle können Sie einen Server über die entsprechende Managed Bean (MBean) stoppen.

```
ServerFactory.stopServer(); // Factory zum Beenden des Server-Singletons
// oder
server.stopServer(); // Server-Instanz direkt verwenden
server.waitFor(); // Kehrt zurück, wenn die Beendigungsprozedur für den Server ordnungsgemäß abgeschlossen wurde
```

### Vollständiges Codebeispiel:

```
import java.net.MalformedURLException;
import java.net.URL;

import com.ibm.websphere.objectgrid.ObjectGridException;
import com.ibm.websphere.objectgrid.deployment.DeploymentPolicy;
import com.ibm.websphere.objectgrid.deployment.DeploymentPolicyFactory;
import com.ibm.websphere.objectgrid.server.Container;
import com.ibm.websphere.objectgrid.server.Server;
import com.ibm.websphere.objectgrid.server.ServerFactory;
import com.ibm.websphere.objectgrid.server.ServerProperties;

public class ServerFactoryTest {

    public static void main(String[] args) {

        try {

            ServerProperties props = ServerFactory.getServerProperties();
            props.setCatalogServiceBootstrap("catalogservice-hostname:catalogservice-port");
            props.setServerName("ServerOne"); // Server benennen
            props.setTraceSpecification("com.ibm.ws.objectgrid=all=enabled"); // TraceSpec

            /*
             * In den meisten Fällen dient der Server nur als Containerserver und
             * stellt eine Verbindung zu einem externen Katalogserver her. Auf diese
             * Weise wird eine höhere Verfügbarkeit erreicht. Der folgende, auf Kommentar
             * gesetzte Codeauszug aktiviert diesen Server als Katalogserver.
             */
            *
            * CatalogServerProperties catalogProps =
            * ServerFactory.getCatalogProperties();
            * catalogProps.setCatalogServer(true); // Katalogservice aktivieren
            * catalogProps.setQuorum(true); // Quorum aktivieren
            */

            Server server = ServerFactory.getInstance();

            DeploymentPolicy policy = DeploymentPolicyFactory.createDeploymentPolicy(new URL("url to deployment xml"),
                new URL("url to objectgrid xml file"));
            Container container = server.createContainer(policy);

            /*
             * Das Shard wird jetzt an diesen Container verteilt, wenn die
             * Implementierungsanforderungen erfüllt sind.
             * Dies umfasst die Erstellung des integrierten Servers und des Containers.
             * Die folgenden Zeilen demonstrieren lediglich den Aufruf der Bereinigungsmethoden.
             */

            container.teardown();
            server.stopServer();
            int success = server.waitFor();

        } catch (ObjectGridException e) {
            // Container konnte nicht initialisiert werden.
        } catch (MalformedURLException e2) {
            // Ungültiger URL für XML-Datei(en)
        }
    }
}
}
```



---

## Kapitel 8. Leistungsaspekte

Zur Verbesserung der Leistung Ihres speicherinternen Daten-Grids oder Datenbankverarbeitungsbereichs können Sie mehrere Überlegungen in Betracht ziehen, z. B. Optimierung Ihrer JVM-Einstellungen oder Einsatz bewährter Verfahren für Produktfeatures wie Sperren, Serialisierung und Abfrageleistung.

---

### Optimierung von JVMs

In diesem Abschnitt werden einige spezielle Fragen zur Optimierung von Java Virtual Machines (JVM) für WebSphere eXtreme Scale beantwortet.

Empfohlen werden 1- bis 2-Gb-Heap-Speicher mit einer JVM pro 4 Kernen. Die Größe der Heap-Speicher richtet sich nach der Art der Objekte, die in den Servern gespeichert werden. Eine diesbezügliche Erläuterung finden Sie weiter hinten in diesem Dokument.

#### **Empfehlungen zur Größe von Heap-Speichern und zur Garbage-Collection**

Die optimale Größe der Heap-Speicher ist von drei Faktoren abhängig:

1. Anzahl der Liveobjekte im Heap-Speicher,
2. Komplexität der Liveobjekte im Heap-Speicher,
3. Anzahl verfügbarer Kerne für die JVM.

Eine Anwendung, die beispielsweise 10-KB-Feldgruppen speichert, kann einen viel größeren Heap-Speicher haben als eine Anwendung, die komplexe POJO-Graphen verwendet.

Alle modernen JVMs verwenden Algorithmen für parallele Garbage-Collection, d. h., durch den Einsatz weiterer Kerne können die Pausen zwischen den Garbage-Collections reduziert werden. Das bedeutet also, dass die Garbage-Collection auf Maschinen mit 8 Kernen schneller ist als auf einer Maschine mit 4 Kernen.

#### **Realspeicherbelegung versus Heap-Speicherspezifikation**

Eine JVM mit einem 1-Gb-Heap-Speicher belegt ungefähr 1,3 Gb Realspeicher. Unter Laborbedingungen wurde festgestellt, dass es nicht möglich ist, zehn 1-Gb-JVMs auf einer Maschine mit 16 Gb Arbeitsspeicher auszuführen. Sobald die JVM eine Belegung von 800 MB erreicht, beginnt sie mit dem Paging.

#### **Garbage-Collection**

Verwenden Sie für IBM JVMs den Collector "avgoptpause" in Szenarios mit einer hohen Aktualisierungsrate (100 % geänderter Transaktionseinträge). Der Collector "gencon" funktioniert in Szenarios, in denen die Daten nur relativ selten aktualisiert werden (10 % der Zeit oder weniger), viel besser als der Collector "avgoptpause". Experimentieren Sie mit beiden Collector, um festzustellen, welcher sich besser in Ihrem Szenario eignet. Wenn Sie ein Leistungsproblem erkennen, aktivieren Sie die ausführliche Garbage-Collection, um die für die Garbage-Collection aufgebrauchte Zeit in Prozent zu überprüfen. Es wurden Szenarios gefunden, in denen

80 % der Zeit für die Garbage-Collection aufgebracht wurde, bis das Problem durch Optimierung behoben wurde.

## JVM-Leistung

WebSphere eXtreme Scale kann unter verschiedenen Versionen von Java 2 Platform, Standard Edition (J2SE) ausgeführt werden. ObjectGrid Version 6.1 unterstützt J2SE Version 1.4.2 und höher. Zur Steigerung der Entwicklerproduktivität und der Leistung verwenden Sie J2SE 5 oder höher, um die Vorteile von Annotationen und verbesserter Garbage-Collection zu nutzen. ObjectGrid funktioniert in 32-Bit- und 64-Bit-JVMs.

Clients von ObjectGrid Version 6.0.2 können eine Verbindung zu einem Grid der ObjectGrid Version 6.1 herstellen. Verwenden Sie Clients der ObjectGrid Version 6.1 für Clients der J2SE Version 1.4.2 oder höher. Der einzige Grund für die Verwendung eines Clients der ObjectGrid Version 6.0.2 ist die Unterstützung von J2SE Version 1.3.

WebSphere eXtreme Scale wurde mit einem Teil der verfügbaren virtuellen Maschinen getestet, aber die Liste der unterstützten JVMs ist nicht exklusiv. Sie können WebSphere eXtreme Scale in jeder Version 1.4.2 oder höher ausführen, aber wenn ein Problem in der JVM auftritt, müssen Sie sich für Unterstützung an den jeweiligen JVM-Anbieter wenden. Verwenden Sie, sofern möglich, die JVM aus der WebSphere-Laufzeitumgebung auf jeder Plattform, die von WebSphere Application Server unterstützt wird.

Java Platform, Standard Edition 6 ist die beste JVM. Die Leistung von Java 2 Platform, Standard Edition, Version 1.4 ist insbesondere in Szenarios, in denen der Collector "gencon" den Ausschlag gibt, sehr schwach. Java Platform Standard Edition 5 bietet eine gute Leistung, aber Java Platform, Standard Edition 6 eine bessere.

## Optimierung der Datei "orb.properties"

Es wird empfohlen, die folgende Datei "orb.properties" für Produktionsumgebungen zu verwenden. Diese Datei wurde unter Laborbedingungen in Grids mit bis zu 1500 JVMs verwendet. Die Datei "orb.properties" befindet sich im Ordner "lib" der verwendeten JRE.

```
# Eigenschaften des IBM JDK für den ORB
org.omg.CORBA.ORBClass=com.ibm.CORBA.iiop.ORB
org.omg.CORBA.ORBSingletonClass=com.ibm.rmi.corba.ORBSingleton

# WS-Interceptor
org.omg.PortableInterceptor.ORBInitializerClass=com.ibm.ws.objectgrid.corba.ObjectGridInitializer

# Eigenschaften des WS-ORB und der Plug-ins
com.ibm.CORBA.ForceTunnel=never
com.ibm.CORBA.RequestTimeout=10
com.ibm.CORBA.ConnectTimeout=10

# Erforderlich, wenn sehr viele JVMs gleichzeitig eine Verbindung zum Katalog herstellen
com.ibm.CORBA.ServerSocketQueueDepth=2048

# Clients und Katalogserver können offene Sockets zu allen JVMs haben
com.ibm.CORBA.MaxOpenConnections=1016

# Thread-Pool für die Verarbeitung eingehender Anforderungen, hier 200 Threads
com.ibm.CORBA.ThreadPool.IsGrowable=false
com.ibm.CORBA.ThreadPool.MaximumSize=200
com.ibm.CORBA.ThreadPool.MinimumSize=200
com.ibm.CORBA.ThreadPool.InactivityTimeout=180000

# Keine Aufteilung großer Anforderungen/Antworten in kleinere Blöcke
com.ibm.CORBA.FragmentSize=0
```



## Thread-Anzahl

Die Thread-Anzahl ist von verschiedenen Faktoren abhängig. Die Anzahl der Threads, die von einem einzelnen Shard verwaltet werden können, ist begrenzt. Je mehr Shards jede JVM enthält, desto mehr Threads und mehr gemeinsame Zugriffe sind möglich. Jedes zusätzliche Shard liefert weitere nebenläufige Pfade für die Daten. Jedes Shard ist so nebenläufig wie möglich, aber nichtsdestotrotz gibt es einen Grenzwert.

---

## Bewährte Verfahren für CopyMode

WebSphere eXtreme Scale erstellt eine Kopie des Werts auf der Basis der CopyMode-Einstellung.

Sie können die Methode `setCopyMode(CopyMode, valueInterfaceClass)` der API `BackingMap` verwenden, um den Kopiermodus auf eines der folgenden Felder des Typs `final static` setzen, die in der Klasse `com.ibm.websphere.objectgrid.CopyMode` definiert sind.

Wenn eine Anwendung die Schnittstelle `ObjectMap` verwendet, um eine Referenz auf einen Map-Eintrag anzufordern, verwenden Sie diese Referenz nur in der Transaktion von WebSphere eXtreme Scale, in der die Referenz angefordert wurde. Wenn Sie die Referenz in einer anderen Transaktion verwenden, kann dies Fehler zur Folge haben. Bei der Verwendung der pessimistischen Sperrstrategie für die `BackingMap` wird beispielsweise über einen Aufruf der Methode `get` oder `getForUpdate` je nach Transaktion eine S- (gemeinsame) bzw. U-Sperre (Aktualisierung) angefordert. Die Methode `get` gibt die Referenz auf den Wert zurück, und die angeforderte Sperre wird freigegeben, sobald die Transaktion abgeschlossen ist. Die Transaktion muss die Methode `get` oder `getForUpdate` aufrufen, um den Map-Eintrag in einer anderen Transaktion zu sperren. Jede Transaktion muss eine eigene Referenz auf den Wert über die Methode `get` bzw. `getForUpdate` anfordern, anstatt dieselbe Wertreferenz einer anderen Transaktion wiederzuverwenden.

### CopyMode-Einstellung für Entitäts-Maps

Wenn Sie eine Map verwenden, die einer Entität der API `EntityManager` zugeordnet ist, gibt die Map immer die Tupelobjekte der Entität wieder, ohne eine Kopie zu erstellen, sofern Sie nicht den Kopiermodus `COPY_TO_BYTES` verwenden. Es ist wichtig, dass die CopyMode-Einstellung aktualisiert wird bzw. das Tupel entsprechend kopiert wird, wenn Änderungen vorgenommen werden.

### COPY\_ON\_READ\_AND\_COMMIT

Der Modus `COPY_ON_READ_AND_COMMIT` ist der Standardmodus. Das Argument `valueInterfaceClass` wird ignoriert, wenn dieser Modus verwendet wird. Dieser Modus stellt sicher, dass eine Anwendung keine Referenz auf das Wertobjekt enthält, das in der `BackingMap` enthalten ist. Stattdessen arbeitet die Anwendung immer mit einer Kopie des Werts, der in der `BackingMap` enthalten ist. Der Modus `COPY_ON_READ_AND_COMMIT` stellt sicher, dass die Anwendung die Daten, die in der `BackingMap` zwischengespeichert sind, nicht unabsichtlich beschädigen kann. Wenn eine Anwendungstransaktion eine Methode `ObjectMap.get` für einen bestimmten Schlüssel aufruft und es sich um den ersten Zugriff auf den `ObjectMap`-Eintrag für diesen Schlüssel handelt, wird eine Kopie des Werts zurückgegeben. Beim Festschreiben der Transaktion werden alle von der Anwendung festgeschriebenen Änderungen in die `BackingMap` kopiert, um sicher-

zustellen, dass die Anwendung keine Referenz auf den festgeschriebenen Wert in der BackingMap hat.

## **COPY\_ON\_READ**

Der Modus `COPY_ON_READ` bietet im Vergleich mit dem Modus `COPY_ON_READ_AND_COMMIT` eine bessere Leistung, weil in diesem Modus beim Festschreiben einer Transaktion keine Daten kopiert werden. Das Argument `"valueInterfaceClass"` wird ignoriert, wenn dieser Modus verwendet wird. Zur Bewahrung der Integrität der BackingMap-Daten stellt die Anwendung sicher, dass jede Referenz auf einen Eintrag gelöscht wird, wenn die Transaktion festgeschrieben wird. In diesem Modus gibt die Methode `"ObjectMap.get"` eine Kopie des Werts an Stelle einer Referenz auf den Wert zurück, um sicherzustellen, dass Änderungen, die von der Anwendung am Wert vorgenommen werden, solange keine Auswirkungen auf den BackingMap-Wert haben, bis die Transaktion festgeschrieben wird. Beim Festschreiben der Transaktion wird jedoch keine Kopie der Änderungen erstellt. Stattdessen wird die Referenz auf die Kopie, die von der Methode `"ObjectMap.get"` zurückgegeben wurde, in der BackingMap gespeichert. Die Anwendung löscht alle Referenzen auf Map-Einträge, wenn die Transaktion festgeschrieben wird. Wenn die Anwendung die Referenzen auf die Map-Einträge nicht löscht, kann die Anwendung die in der BackingMap zwischengespeicherten Daten beschädigen. Falls eine Anwendung diesen Modus verwendet und Probleme auftreten, wechseln Sie in den Modus `COPY_ON_READ_AND_COMMIT`, um festzustellen, ob die Probleme weiterhin auftreten. Sollten die Probleme nicht mehr auftreten, ist dies ein Hinweis darauf, dass die Anwendung nach dem Festschreiben der Transaktion nicht alle Referenzen löscht.

## **COPY\_ON\_WRITE**

Der Modus `COPY_ON_WRITE` bietet im Vergleich mit dem Modus `COPY_ON_READ_AND_COMMIT` eine bessere Leistung, weil in diesem Modus beim ersten Aufruf der Methode `"ObjectMap.get"` für einen bestimmten Schlüssel in einer Transaktion keine Daten kopiert werden. Die Methode `"ObjectMap.get"` gibt einen Proxy auf den Wert an Stelle einer direkten Referenz auf das Wertobjekt zurück. Der Proxy stellt sicher, dass keine Kopie des Werts erstellt wird, sofern die Anwendung nicht eine Methode `"set"` für die Wertschnittstelle aufruft, die mit dem Argument `"valueInterfaceClass"` angegeben wurde. Der Proxy verwendet eine Implementierung von `"copy on write"` (Kopieren beim Schreiben). Wenn eine Transaktion festgeschrieben wird, prüft die BackingMap den Proxy, um festzustellen, ob auf die aufgerufene Methode `"set"` hin eine Kopie erstellt wurde. Wurde eine Kopie erstellt, wird diese Referenz auf die Kopie in der BackingMap gespeichert. Dieser Modus hat den großen Vorteil, dass beim Lesen oder Festschreiben niemals ein Wert kopiert wird, wenn die Transaktion keine Methode `"set"` aufruft, um den Wert zu ändern.

In den Modi `COPY_ON_READ_AND_COMMIT` und `COPY_ON_READ` wird eine tiefe Kopie erstellt, wenn ein Wert aus der ObjectMap abgerufen wird. Wenn eine Anwendung nur einige der Werte, die in einer Transaktion abgerufen werden, aktualisiert, ist dieser Modus nicht optimal. Der Modus `COPY_ON_WRITE` unterstützt dieses Verhalten zwar effizient, erfordert aber, dass die Anwendung ein einfaches Muster verwendet. Die Wertobjekte sind erforderlich, um eine Schnittstelle zu unterstützen. Die Anwendung muss die Methoden in dieser Schnittstelle verwenden, wenn sie mit dem Wert in einer eXtreme-Scale-Sitzung interagiert. In diesem Fall erstellt eXtreme Scale Proxys für die Werte, die an die Anwendung zurückgegeben werden. Der Proxy hat eine Referenz auf den echten Wert. Wenn die Anwendung nur Leseoperationen durchführt, werden diese immer mit der ech-

ten Kopie durchgeführt. Wenn die Anwendung ein Attribut im Objekt ändert, erstellt der Proxy eine Kopie des echten Objekts und nimmt die Änderung dann an der Kopie vor. Von diesem Zeitpunkt an verwendet der Proxy die Kopie. Mit der Verwendung der Kopie kann die Kopieroperation für Objekte, die von der Anwendung nur gelesen werden, vollständig umgangen werden. Alle Änderungsoperationen müssen mit dem festgelegten Präfix beginnen. Enterprise JavaBeans werden normalerweise so codiert, dass sie diese Art der Methodenbenennung für Methoden verwenden, die die Objektattribute ändern. Diese Konvention muss eingehalten werden. Alle geänderten Objekte werden zu dem Zeitpunkt kopiert, zu dem sie von der Anwendung geändert werden. Dieses Lese- und Schreibszenario ist das effizienteste Szenario, das von eXtreme Scale unterstützt wird. Wenn Sie den Modus COPY\_ON\_WRITE für eine Map konfigurieren möchten, können Sie das folgende Beispiel verwenden. In diesem Beispiel speichert die Anwendung Person-Objekte, die in der Map mit dem Namen als Schlüssel verwaltet werden. Das Person-Objekt wird im folgenden Code-Snippet dargestellt.

```
class Person {
    String name;
    int age;
    public Person() {
    }
    public void setName(String n) {
        name = n;
    }
    public String getName() {
        return name;
    }
    public void setAge(int a) {
        age = a;
    }
    public int getAge() {
        return age;
    }
}
```

Die Anwendung verwendet die Schnittstelle "IPerson" nur, wenn sie mit Werten interagiert, die aus einer ObjectMap abgerufen werden. Ändern Sie das Objekt, wie im folgenden Beispiel gezeigt, um eine Schnittstelle zu verwenden:

```
interface IPerson
{
    void setName(String n);
    String getName();
    void setAge(int a);
    int getAge();
}
// Person-Objekt ändern, um die Schnittstelle "IPerson" zu implementieren.
class Person implements IPerson {
    ...
}
```

Anschließend muss die Anwendung, wie im folgenden Beispiel gezeigt, den Modus COPY\_ON\_WRITE für die BackingMap konfigurieren:

```
ObjectGrid dg = ...;
BackingMap bm = dg.defineMap("PERSON");
// COPY_ON_WRITE für diese Map mit
// IPerson als valueProxyInfo-Klasse verwenden.
bm.setCopyMode(CopyMode.COPY_ON_WRITE, IPerson.class);
// Anschließend muss die Anwendung das folgende Muster verwenden,
// wenn die Map PERSON verwendet wird.
Session sess = ...;
ObjectMap person = sess.getMap("PERSON");
...
sess.begin();
```

```

// Die Anwendung setzt den zurückgegebenen Wert in IPerson und nicht in Person um.
IPerson p = (IPerson)person.get("Billy");
p.setAge( p.getAge() + 1 );
...
// Neues Person-Objekt erstellen und der Map hinzufügen
Person p1 = new Person();
p1.setName("Bobby");
p1.setAge(12);
person.insert(p1.getName(), p1);
sess.commit();
// Das folgende Snippet funktioniert nicht. Es löst eine Ausnahme des Typs "ClassCastException" aus.
sess.begin();
// Der Fehler ist hier, dass Person an Stelle von
// IPerson verwendet wird.
Person a = (Person)person.get("Bobby");
sess.commit();

```

Im ersten Abschnitt wird gezeigt, dass die Anwendung einen Wert abrufen, der in der Map den Namen "Billy" hat. Die Anwendung setzt den zurückgegebenen Wert in das IPerson-Objekt und nicht in das Person-Objekt um, weil der zurückgegebene Proxy zwei Schnittstellen implementiert:

- die im Aufruf der Methode "BackingMap.setCopyMode" angegebene Schnittstelle,
- die Schnittstelle "com.ibm.websphere.objectgrid.ValueProxyInfo".

Sie können den Proxy in zwei Typen umsetzen. Der letzte Teil des vorherigen Code-Snippets demonstriert, was im Modus COPY\_ON\_WRITE nicht zulässig ist. Die Anwendung ruft den Datensatz "Bobby" ab und versucht, den Datensatz in ein Person-Objekt umzusetzen. Diese Aktion scheitert mit einer Ausnahme bei Klassenumsetzung, weil der zurückgegebene Proxy kein Person-Objekt ist. Der zurückgegebene Proxy implementiert das IPerson-Objekt und ValueProxyInfo.

Schnittstelle "ValueProxyInfo" und Unterstützung von Teilaktualisierungen: Diese Schnittstelle ermöglicht einer Anwendung, den festgeschriebenen schreibgeschützten Wert, der vom Proxy referenziert wird, oder die Gruppe der Attribute, die in dieser Transaktion geändert wurden, abzurufen.

```

public interface ValueProxyInfo {
    List /**/ ibmGetDirtyAttributes();
    Object ibmGetRealValue();
}

```

Die Methode "ibmGetRealValue" gibt eine schreibgeschützte Kopie des Objekts zurück. Die Anwendung darf diesen Wert nicht ändern. Die Methode "ibmGetDirtyAttributes" gibt eine Liste mit Zeichenfolgen zurück, die die Attribute darstellen, die von der Anwendung während dieser Transaktion geändert wurden. Der Hauptanwendungsfall für die Methode "ibmGetDirtyAttributes" ist in einem JDBC- (Java Database Connectivity) oder CMP-basierten Loader. Nur die in der Liste benannten Attribute müssen in der SQL-Anweisung bzw. in dem Objekt, das der Tabelle zugeordnet ist, aktualisiert werden. Dies führt zu einer effizienteren SQL, die vom Loader generiert wird. Wenn eine Copy-on-Write-Transaktion festgeschrieben wird und ein Loader integriert ist, kann der Loader die Werte der geänderten Objekte in die Schnittstelle "ValueProxyInfo" umsetzen, um diese Informationen abzurufen.

Behandlung der Methode "equals", wenn COPY\_ON\_WRITE oder Proxys verwendet werden: der folgende Code erstellt beispielsweise ein Person-Objekt und fügt es anschließend in eine ObjectMap ein. Anschließend ruft er dasselbe Objekt mit der Methode "ObjectMap.get" ab. Der Wert wird in die Schnittstelle umgesetzt.

Wenn der Wert in die Schnittstelle "Person" umgesetzt wird, wird eine Ausnahme des Typs "ClassCastException" ausgelöst, weil der zurückgegebene Wert ein Proxy ist, der die Schnittstelle "IPerson" implementiert und kein Person-Objekt ist. Die Gleichheitsprüfung scheitert, wenn die Operation "==" verwendet wird, weil es sich nicht um dasselbe Objekt handelt.

```
session.begin();
// Neues Person-Objekt
Person p = new Person(...);
personMap.insert(p.getName(), p);
// Erneut abrufen und daran denken, die Schnittstelle für die Umsetzung zu verwenden
IPerson p2 = personMap.get(p.getName());
if(p2 == p) {
    // Objekte sind identisch
} else {
    // Objekte sind nicht identisch
}
```

Ein weiterer Aspekt ist das Überschreiben der Methode "equals". Wie im vorherigen Code-Snippet veranschaulicht, muss die Methode "equals" sicherstellen, dass das Argument ein Objekt ist, das die Schnittstelle "IPerson" implementiert und das Argument in ein IPerson-Objekt umsetzt. Da das Argument ein Proxy sein kann, das die Schnittstelle "IPerson" implementiert, müssen Sie die Methoden "getAge" und "getName" verwenden, wenn Sie vergleichen, ob die Instanzvariablen identisch sind.

```
{
    if ( obj == null ) return false;
    if ( obj instanceof IPerson ) {
        IPerson x = (IPerson) obj;
        return ( age.equals( x.getAge() ) && name.equals( x.getName() ) )
    }
    return false;
}
```

Voraussetzungen für die Konfiguration von ObjectQuery und HashIndex: Wenn Sie den Modus COPY\_ON\_WRITE mit ObjectQuery oder einem HashIndex-Plug-in verwenden, müssen Sie das ObjectQuery-Schema und das HashIndex-Plug-in konfigurieren, um über Eigenschaftsmethoden (Standardeinstellung) auf die Objekte zuzugreifen. Wenn die Verwendung des Feldzugriffs konfiguriert ist, versuchen die Abfragesteuerkomponente und der Index, auf die Felder im Proxy-Objekt zuzugreifen. Daraufhin wird immer null (0) zurückgegeben, da die Objektinstanz ein Proxy ist.

## NO\_COPY

Der Modus NO\_COPY ermöglicht einer Anwendung sicherzustellen, dass sie ein Wertobjekt, das über eine Methode "ObjectMap.get" abgerufen wird, nicht ändert, um Leistungsverbesserungen zu erzielen. Das Argument "valueInterfaceClass" wird ignoriert, wenn dieser Modus verwendet wird. Wenn dieser Modus verwendet wird, wird keine Kopie des Werts erstellt. Wenn die Anwendung Werte ändert, werden die Daten in der BackingMap beschädigt. Der Modus NO\_COPY ist hauptsächlich für schreibgeschützte Maps hilfreich, in denen die Daten von der Anwendung nie geändert werden. Falls eine Anwendung diesen Modus verwendet und Probleme auftreten, wechseln Sie in den Modus COPY\_ON\_READ\_AND\_COMMIT, um festzustellen, ob die Probleme weiterhin auftreten. Sollten die Probleme nicht mehr auftreten, ist dies ein Hinweis darauf, dass die Anwendung während der Transaktion oder nach dem Festschreiben der Transaktion den von der Methode "ObjectMap.get" zurückgegebenen Wert ändert. Alle Maps, die Entitäten

der API "EntityManager" zugeordnet sind, verwenden diesen Modus automatisch, unabhängig davon, welcher Modus in der Konfiguration von eXtreme Scale definiert ist.

Alle Maps, die Entitäten der API "EntityManager" zugeordnet sind, verwenden diesen Modus automatisch, unabhängig davon, welcher Modus in der Konfiguration von eXtreme Scale definiert ist.

## **COPY\_TO\_BYTES**

Sie können Objekte in einem serialisierten Format an Stelle des POJO-Formats speichern. Mit der Einstellung COPY\_TO\_BYTES können Sie den Speicherbedarf eines großen Objektgraphen verringern. Weitere Informationen finden Sie im Abschnitt „Maps für Bytefeldgruppen“.

## **Unzulässige Verwendung von CopyMode**

Es treten Fehler auf, wenn eine Anwendung versucht, die Leistung mit dem Kopiermodus COPY\_ON\_READ, COPY\_ON\_WRITE oder NO\_COPY, wie zuvor beschrieben, zu verbessern. Diese Probleme treten nicht auf, wenn Sie den Kopiermodus in COPY\_ON\_READ\_AND\_COMMIT ändern.

### **Problem**

Das Problem kann auf beschädigte Daten in der ObjectGrid-Map zurückzuführen sein, die das Ergebnis eines Verstoßes gegen den Programmiervertrag des verwendeten Kopiermodus durch die Anwendung sind. Fehlerhafte Daten können zu unvorhersehbaren Fehlern führen, die vorübergehend, unerklärlich oder unerwartet sein können.

### **Lösung**

Die Anwendung muss den Programmiervertrag einhalten, der für den verwendeten Kopiermodus festgelegt wurde. Für die Kopiermodi COPY\_ON\_READ und COPY\_ON\_WRITE verwendet die Anwendung eine Referenz auf ein Wertobjekt außerhalb des Transaktionsbereichs, von dem die Wertreferenz abgerufen wurde. Zur Verwendung dieser Modi muss die Anwendung die Referenz auf das Wertobjekt nach Abschluss der Transaktion löschen und eine neue Referenz auf das Wertobjekt in jeder Transaktion abrufen, die auf das Wertobjekt zugreift. Im Kopiermodus NO\_COPY darf die Anwendung das Wertobjekt nie ändern. In diesem Fall müssen Sie die Anwendung so schreiben, dass sie das Wertobjekt nicht ändert, oder einen anderen Kopiermodus für die Anwendung festlegen.

## **Maps für Bytefeldgruppen**

Sie können den Wert eines Schlüssel/Wert-Paars in einer Bytefeldgruppe anstatt im zugehörigen POJO-Format speichern, was den Speicherbedarf reduziert, den ein großer Objektgraph konsumieren kann.

### **Vorteile**

Die Speicherbelegung steigt mit der Anzahl der Objekte im Objektgraphen. Indem Sie einen komplexen Objektgraphen zu einer Bytefeldgruppe reduzieren, wird nur noch ein einziges Objekt an Stelle mehrerer Objekte im Heap-Speicher verwaltet. Durch die Reduktion der Objektanzahl im Heap-Speicher muss die Java-Laufzeitumgebung während der Garbage-Collection weniger Objekte suchen.

Der von WebSphere eXtreme Scale verwendete Standardkopiermechanismus ist die Serialisierung, die kostenintensiv ist. Wenn Sie beispielsweise den Standardkopiermodus `COPY_ON_READ_AND_COMMIT` verwenden, wird jeweils beim Lesen und beim Abrufen eine Kopie erstellt. Anstatt eine Kopie beim Lesen zu erstellen, wird der Wert mit Bytefeldgruppen aus den Bytes wiederhergestellt, und anstatt eine Kopie bei der Festschreibung zu erstellen, wird der Wert in Bytes serialisiert. Die Verwendung von Bytefeldgruppen liefert dieselbe Datenkonsistenz wie die Standardeinstellung mit Reduktion der Speicherbelegung.

Wenn Sie Bytefeldgruppen verwenden, ist der Einsatz eines optimierten Serialisierungsmechanismus von entscheidender Bedeutung, um eine Reduktion der Speicherbelegung zu erzielen. Weitere Informationen finden Sie im Abschnitt „Serialisierungsleistung“ auf Seite 277.

## Maps für Bytefeldgruppen konfigurieren

Sie können Maps für Bytefeldgruppen über die ObjectGrid-XML-Datei aktivieren, indem Sie das von einer Map verwendete Attribut `CopyMode`, wie im folgenden Beispiel gezeigt, auf `COPY_TO_BYTES` setzen:

```
<backingMap name="byteMap" copyMode="COPY_TO_BYTES" />
```

Weitere Informationen finden Sie im Abschnitt zur ObjectGrid-XML-Deskriptordatei im *Administratorhandbuch*.

## Hinweise

Sie müssen prüfen, ob sich die Verwendung von Maps für Bytefeldgruppen in einem bestimmten Szenario eignet oder nicht. Die Speicherbelegung reduziert sich zwar bei der Verwendung von Bytefeldgruppen, aber die Prozessorauslastung kann zunehmen.

In der folgenden Liste sind verschiedene Faktoren beschrieben, die Sie berücksichtigen sollten, bevor Sie sich für die Verwendung von Maps für Bytefeldgruppen entscheiden.

### Objektyp

Für einige Objekttypen ist bei der Verwendung von Maps für Bytefeldgruppen unter Umständen keine Reduktion der Speicherbelegung möglich. Deshalb gibt es auch verschiedene Objekttypen, für die Sie keine Maps für Bytefeldgruppen verwenden sollten. Wenn Sie einen der primitiven Java-Wrapper als Wert verwenden oder ein POJO, das keine Referenzen auf andere Objekte enthält (sondern nur primitive Felder speichert), ist die Anzahl der Java-Objekte bereits so niedrig wie möglich, nämlich eins. Da die Speicherbelegung für das Objekt bereits optimiert ist, wird die Verwendung einer Map für Bytefeldgruppen für diese Objekttypen nicht empfohlen. Maps für Bytefeldgruppen eignen sich besser für Objekttypen, die andere Objekte oder Objektsammlungen enthalten, in denen die Gesamtanzahl der POJO-Objekte größer als eins ist.

Wenn Sie beispielsweise ein Objekt `Customer` haben, das eine Geschäftsadresse (Business Address) und eine Privatadresse (Home Address) sowie eine Sammlung von Aufträgen (Order) hat, können die Anzahl der Objekte im Heap-Speicher und die Anzahl der von diesen Objekten belegten Bytes durch die Verwendung von Maps für Bytefeldgruppen reduziert werden.

### Lokaler Zugriff

Werden andere Kopiermodi verwendet, können Anwendungen bei der Erstellung von Kopien optimiert werden, wenn Objekte mit dem Standard-ObjectTransformer klonbar sind oder wenn ein angepasster ObjectTransformer mit einer optimierten Methode "copyValue" bereitgestellt wird. Verglichen mit den anderen Kopiermodi, fallen für das Kopieren bei Lese-, Schreib- und Festschreibungsoperationen zusätzliche Kosten an, wenn lokal auf Objekte zugegriffen wird. Wenn Sie beispielsweise einen nahen Cache in einer verteilten Topologie haben oder direkt auf eine lokale oder Server-ObjectGrid-Instanz zugreifen, nehmen die Zugriffs- und Festschreibungszeiten bei der Verwendung von Maps für Bytefeldgruppen aufgrund der Serialisierungskosten zu. Ein ähnlicher Aufwand ist in einer verteilten Topologie zu verzeichnen, wenn Sie DataGrid-Agenten verwenden oder bei der Verwendung des ObjectGridEventGroup.ShardEvents-Plug-ins auf die primäre Partition des Servers zugreifen.

### **Plug-in-Interaktionen**

Wenn Sie Maps für Bytefeldgruppen verwenden, werden Objekte bei der Kommunikation zwischen einem Client und einem Server nicht dekomprimiert, es sei denn, der Server benötigt das POJO-Format. Plug-ins, die mit dem Map-Wert interagieren, verzeichnen Leistungseinbußen, weil der Wert dekomprimiert werden muss.

Jedes Plug-in, das "LogElement.getCacheEntry" oder "LogElement.getCurrentValue" verwendet, weist diesen erhöhten Aufwand auf. Wenn Sie den Schlüssel abrufen möchten, können Sie die Methode "LogElement.getKey" verwenden, die den zusätzlichen Aufwand vermeidet, den die Methode "LogElement.getCacheEntry().getKey" mit sich bringt. In den folgenden Absätzen wird die Wirkung von Bytefeldgruppen auf Plug-ins erläutert.

#### *Indizes und Abfragen*

Wenn Objekte im POJO-Format gespeichert werden, sind die Kosten für die Indizierung und Abfragen minimal, weil das Objekt nicht dekomprimiert werden muss. Wenn Sie eine Map für Bytefeldgruppen verwenden, entstehen zusätzliche Kosten für die Dekomprimierung des Objekts. Wenn Ihre Anwendung Indizes oder Abfragen verwendet, wird im Allgemeinen von der Verwendung von Maps für Bytefeldgruppen abgeraten, sofern Sie die Abfragen nicht ausschließlich für Schlüsselattribute durchführen.

#### *Optimistisches Sperren*

Wenn Sie die optimistische Sperrstrategie verwenden, entstehen zusätzliche Kosten bei Operationen zum Aktualisieren oder Ungültigmachen von Einträgen. Dies ist darauf zurückzuführen, dass der Wert im Server dekomprimiert werden muss, um den Versionswert für die optimistische Kollisionsprüfung abzurufen. Wenn Sie optimistisches Sperren nur verwenden, um Abrufoperationen (fetch) zu gewährleisten und keine optimistische Kollisionsprüfung benötigen, können Sie "com.ibm.websphere.objectgrid.plugins.builtins.NoVersioningOptimisticCallback" verwenden, um die Versionsprüfung zu inaktivieren.

#### *Loader*

Auch bei einem Loader (Ladeprogramm) fallen in der eXtreme-Scale-Laufzeitumgebung Kosten für die Dekomprimierung und erneute Serialisierung des Werts an, wenn er vom Loader verwendet wird. Sie können für Loader trotzdem Maps für Bytefeldgruppen verwenden, müssen aber die Kosten berücksichtigen, die



durch das Ändern des Werts in einem solchen Szenario anfallen. Sie können das Feature für Bytefeldgruppen beispielsweise im Kontext eines Caches verwenden, in dem hauptsächlich Leseoperationen durchgeführt werden. In diesem Fall überwiegen die Vorteile, weniger Objekte im Heap-Speicher und weniger Speicherbelegung zu haben, die Kosten, die durch die Verwendung von Bytefeldgruppen bei Einfüge- und Aktualisierungsoperationen anfallen.

#### *ObjectGridEventListener*

Wenn Sie die Methode "transactionEnd" im ObjectGridEventListener-Plug-in verwenden, entstehen zusätzliche Kosten auf der Serverseite für Fernanforderungen beim Zugriff auf den Cacheeintrag eines LogElement-Objekts oder auf den aktuellen Wert. Wenn die Implementierung der Methode nicht auf diese Felder zugreift, entstehen keine zusätzlichen Kosten.

---

## Bereinigung

WebSphere eXtreme Scale stellt einen Standardmechanismus für das Entfernen von Cacheeinträgen und ein Plug-in für das Erstellen angepasster Bereinigungsprogramme (Evictor) bereit. Ein Bereinigungsprogramm steuert die Zugehörigkeit von Einträgen in jeder BackingMap. Das Standardbereinigungsprogramm verwendet eine Bereinigungsrichtlinie für jede BackingMap, die auf der Lebensdauer (TTL, Time-to-Live) basiert. Wenn Sie ein Plug-in-fähiges Bereinigungsprogramm bereitstellen, verwendet dieses gewöhnlich eine Bereinigungsrichtlinie, die auf der Anzahl der Einträge und nicht auf der Lebensdauer basiert.

### Standard-TTL-Bereinigungsprogramm

WebSphere eXtreme Scale stellt ein TTL-Bereinigungsprogramm für jede BackingMap bereit. Das TTL-Bereinigungsprogramm verwaltet eine Verfallszeit für jeden erstellten Eintrag. Nach dem Ablauf der Verfallszeit eines Eintrags entfernt das Bereinigungsprogramm den Eintrag aus der BackingMap. Um den Leistungseinfluss zu minimieren, den das Entfernen eines Eintrags hat, kann das TTL-Bereinigungsprogramm das Entfernen eines Eintrags über den Ablauf der Verfallszeit hinweg hinauszögern. Das TTL-Bereinigungsprogramm entfernt Einträge nie vor ihrem Verfallsdatum.

Die BackingMap hat Attribute, mit denen gesteuert werden kann, wie das TTL-Bereinigungsprogramm die Verfallszeit für jeden Eintrag berechnet. Anwendungen setzen das Attribut "ttlType", um anzugeben, wie das TTL-Bereinigungsprogramm die Verfallszeit berechnet. Das Attribut "ttlType" kann auf einen der folgenden Werte gesetzt werden:

1. None: Gibt an, dass Einträge in der BackingMap nicht verfallen. Das TTL-Bereinigungsprogramm entfernt diese Einträge nicht.
2. CreationTime: Gibt an, dass die Erstellungszeit eines Eintrags für die Berechnung der Verfallszeit verwendet wird.
3. LastAccessTime: Gibt an, dass die Zeit des letzten Zugriffs auf einen Eintrag für die Berechnung der Verfallszeit verwendet wird.

Wenn das Attribut "ttlType" für eine BackingMap nicht gesetzt ist, wird der Standardtyp "None" verwendet, so dass das TTL-Bereinigungsprogramm Einträge nie entfernt. Wenn das Attribut "ttlType" auf "CreationTime" oder "LastAccessTime" gesetzt ist, wird der Wert des TTL-Attributs in der BackingMap der Erstellungszeit bzw. der letzten Zugriffszeit hinzugefügt, um die Verfallszeit zu berechnen. Die Zeit für das TTL-Attribut einer Map wird in Sekunden angegeben. Der Wert 0 für

das TTL-Attribut ist ein Sonderwert, der verwendet wird, um anzuzeigen, dass der Map-Eintrag eine unbegrenzte Lebensdauer hat, d. h., dass der Eintrag so lange in der Map bleibt, bis die Anwendung den Map-Eintrag explizit entfernt oder ungültig macht.

## Optionale Bereinigungsprogramme

Das TTL-Standardbereinigungsprogramm verwendet eine Bereinigungsrichtlinie, die auf Zeit basiert, und die Anzahl der Einträge in der BackingMap hat keine Auswirkung auf die Verfallszeit eines Eintrags. Sie können ein optionales Plug-in-Bereinigungsprogramm verwenden, um Einträge auf der Basis der Anzahl vorhandener Einträge an Stelle der Zeit zu entfernen.

Die folgenden optionalen Plug-in-Bereinigungsprogramme stellen einige gängige Algorithmen bereit, mit denen entschieden werden kann, welche Einträge entfernt werden sollen, wenn eine BackingMap ihr Größenlimit erreicht. \*

- Das Bereinigungsprogramm "LRUEvictor" verwendet einen LRU-Algorithmus (Least Recently Used), um zu entscheiden, welche Einträge entfernt werden sollen, wenn die BackingMap eine maximale Anzahl an Einträgen überschreitet.
- Das Bereinigungsprogramm "LFUEvictor" verwendet einen LFU-Algorithmus (Least Frequently Used), um zu entscheiden, welche Einträge entfernt werden sollen, wenn die BackingMap eine maximale Anzahl an Einträgen überschreitet.

Die BackingMap informiert ein Bereinigungsprogramm, wenn Einträge in einer Transaktion erstellt, geändert oder entfernt werden. Die BackingMap verfolgt diese Einträge und entscheidet, wann Einträge aus der BackingMap entfernt werden müssen.

Eine BackingMap hat keine Konfigurationsdaten für eine maximale Größe. Stattdessen werden Eigenschaften des Bereinigungsprogramms definiert, um das Verhalten des Bereinigungsprogramms zu steuern. Die Bereinigungsprogramme "LRUEvictor" und "LFUEvictor" haben beide eine Eigenschaft für die maximale Größe, die das Bereinigungsprogramm anweist, mit dem Entfernen von Einträgen zu beginnen, wenn die maximale Größe überschritten wird. Wie das TTL-Bereinigungsprogramm entfernen auch die Bereinigungsprogramme "LRUEvictor" und "LFUEvictor" einen Eintrag möglicherweise nicht sofort, wenn die maximale Anzahl an Einträgen erreicht ist, um die Auswirkungen auf die Leistung zu minimieren.

Wenn der LRU- bzw. LFU-Bereinigungsalgorithmus für eine bestimmte Anwendung nicht angemessen ist, können Sie eigene Bereinigungsprogramme schreiben, um eine eigene Bereinigungsstrategie zu erstellen.

## Speicherbasierte Bereinigung

**Wichtig:** Die speicherbasierte Bereinigung wird nur in Java Platform, Enterprise Edition Version 5 und höher unterstützt.

Alle integrierten Bereinigungsprogramme unterstützen die speicherbasierte Bereinigung, die in der Schnittstelle "BackingMap" aktiviert werden kann, indem das Attribut "evictionTriggers" der BackingMap auf MEMORY\_USAGE\_THRESHOLD gesetzt wird. Weitere Informationen zum Definieren des Attributs "evictionTriggers" in der BackingMap finden Sie in den Beschreibungen der Schnittstelle "BackingMap" und der ObjectGrid-XML-Deskriptordatei im *Administratorhandbuch*.

Die speicherbasierte Bereinigung basiert auf einem Schwellenwert für die Auslastung des Heap-Speichers. Wenn die speicherbasierte Bereinigung in der BackingMap aktiviert ist und die BackingMap ein integriertes Bereinigungsprogramm hat, wird der Schwellenwert für die Auslastung auf einen Standardprozentsatz des Gesamtspeichers gesetzt, wenn noch kein Schwellenwert definiert wurde.

Wenn Sie die speicherbasierte Bereinigung verwenden, müssen Sie den Schwellenwert für die Garbage-Collection auf denselben Wert wie die Zielauslastung des Heap-Speichers setzen. Beispiel: Wenn der Schwellenwert für die speicherbasierte Bereinigung auf 50 Prozent und der Schwellenwert für die Garbage-Collection auf den Standardwert von 70 Prozent gesetzt wird, kann die Auslastung des Heap-Speichers auf maximal 70 Prozent ansteigen. Diese Erhöhung der Heap-Speicher-auslastung findet statt, weil die speicherbasierte Bereinigung erst nach einem Garbage-Collection-Zyklus ausgelöst wird.

Der von WebSphere eXtreme Scale verwendete Algorithmus für die speicherbasierte Bereinigung reagiert auf das Verhalten des verwendeten Algorithmus für die Garbage-Collection. Der beste Algorithmus für die speicherbasierte Bereinigung ist der IBM Standarddurchsatz-Collector. Algorithmen für eine Garbage-Collection nach Objektalter können ein unerwünschtes Verhalten bewirken, und deshalb sollten Sie diese Algorithmen nicht für die speicherbasierte Bereinigung verwenden.

Wenn Sie den Prozentsatz für den Auslastungsschwellenwert ändern möchten, setzen Sie die Eigenschaft "memoryThresholdPercentage" in den Container- und Servereigenschaftendateien für eXtreme-Scale-Serverprozesse.

Wenn die Speicherbelegung zur Laufzeit den Schwellenwert für die Zielauslastung überschreitet, beginnen speicherbasierte Bereinigungsprogramme mit dem Entfernen von Einträgen und versuchen, die Speicherbelegung unterhalb des Schwellenwerts für die Zielauslastung zu halten. Es gibt jedoch keine Garantien, dass die Bereinigung schnell genug ist, um potenzielle abnormale Speicherbedingungen zu verhindern, wenn die Laufzeitumgebung des Systems weiterhin so schnell Speicher belegt.

## **Bewährte Verfahren für Standardbereinigungsprogramme**

Sie können das Verhalten des TTL-Standardbereinigungsprogramms (Time to Live, Lebensdauer) ändern, indem Sie Attribute und Eigenschaften definieren.

Zusätzlich zu den Plug-in-Bereinigungsprogrammen, die im Abschnitt "Bewährte Verfahren für Plug-in-Bereinigungsprogramme" beschrieben werden, wird mit jeder BackingMap ein TTL-Standardbereinigungsprogramm erstellt. Das Standardbereinigungsprogramm entfernt Einträge, die auf einem TTL-Konzept basieren. Dieses Verhalten wird mit dem Attribut "ttlType" definiert. Es sind drei ttlType-Attribute vorhanden:

- Ohne: Gibt an, dass Einträge nicht verfallen und deshalb nicht aus der Map entfernt werden.
- Erstellungszeit: Gibt an, dass Einträge auf der Basis der Erstellungszeit entfernt werden.
- Letzte Zugriffszeit: Gibt an, dass Einträge auf der Basis der letzten Zugriffszeit entfernt werden.

## Eigenschaft "TimeToLive"

Diese Eigenschaft ist zusammen mit der Eigenschaft "ttlType" aus Leistungsperspektive das entscheidendste. Wenn Sie das ttlType-Attribut CREATION\_TIME (Erstellungszeit) verwenden, entfernt das Bereinigungsprogramm einen Eintrag, wenn die seit der Erstellung des Eintrags vergangene Zeit dem Wert des Attributs "TimeToLive" entspricht. Wenn Sie das Attribut "TimeToLive" auf 10 Sekunden setzen, werden alle Einträge in der Map nach 10 Sekunden entfernt. Sie müssen beim Festlegen des Werts für das ttlType-Attribut CREATION\_TIME vorsichtig vorgehen. Die Verwendung dieses Bereinigungsprogramms empfiehlt sich, wenn dem Cache sehr viele Einträge hinzugefügt werden, die nur für eine bestimmte Zeit verwendet werden. Mit dieser Strategie werden alle erstellten Einträge nach der festgelegten Zeit entfernt.

Im Folgenden finden Sie ein Beispiel für einen sinnvollen Einsatz des TTL-Typs CREATION\_TIME: Sie verwenden eine Webanwendung, die Börsennotierungen abrufen. Die Topaktualität der Notierungen ist jedoch nicht kritisch. In diesem Fall werden die Börsennotierungen 20 Minuten lang in einem ObjectGrid zwischengespeichert. Nach 20 Minuten verfallen die Einträge in der ObjectGrid-Map und werden daraufhin entfernt. In einem Intervall von ungefähr 20 Minuten verwendet die ObjectGrid-Map das Ladeprogramm-Plug-in, um die Map-Daten mit aktuellen Daten aus der Datenbank zu aktualisieren. Die Datenbank wird alle 20 Minuten mit den topaktuellen Börsennotierungen aktualisiert. Somit ist die Verwendung eines TimeToLive-Werts von 20 Minuten ideal.

Wenn Sie das ttlType-Attribut LAST\_ACCESSED\_TIME verwenden, legen Sie für "TimeToLive" einen niedrigeren Wert als beim ttlType-Attribut CREATION\_TIME fest, weil das TimeToLive-Attribut eines Eintrags jedes Mal zurückgesetzt wird, wenn auf den Eintrag zugegriffen wird. Anders ausgedrückt, wenn das TimeToLive-Attribut den Wert 15 hat und ein Eintrag seit 14 Sekunden existiert, aber dann aufgerufen wird, wird seine Verfallszeit um weitere 15 Sekunden verlängert. Wenn Sie "TimeToLive" auf einen relativ hohen Wert setzen, werden viele Einträge möglicherweise nie entfernt. Setzen Sie "TimeToLive" jedoch auf einen Wert von ungefähr 15 Sekunden, können Einträge entfernt werden, wenn nicht sehr häufig auf sie zugegriffen wird.

Im Folgenden finden Sie ein Beispiel für einen sinnvollen Einsatz des TTL-Typs LAST\_ACCESSED\_TIME: Eine ObjectGrid-Map wird verwendet, um Sitzungsdaten eines Clients zu speichern. Sitzungsdaten müssen gelöscht werden, wenn der Client die Sitzungsdaten innerhalb eines bestimmten Zeitraums nicht verwendet. Die Sitzungsdaten verfallen beispielsweise, wenn innerhalb von 30 Minuten keine Aktivität des Clients erfolgt. In diesem Fall der TTL-Typ LAST\_ACCESSED\_TIME mit einem TimeToLive-Attributwert von 30 Minuten genau die Einstellung, die für diese Anwendung benötigt wird.

Im folgenden Beispiel wird eine BackingMap erstellt, das ttlType-Attribut für das Standardbereinigungsprogramm festgelegt und die Eigenschaft "TimeToLive" des Bereinigungsprogramms definiert:

```
ObjectGrid objGrid = new ObjectGrid();
BackingMap bMap = objGrid.defineMap("SomeMap");
bMap.setTtlEvictorType(TTLType.LAST_ACCESSED_TIME);
bMap.setTimeToLive(1800);
```

Die meisten Einstellungen des Bereinigungsprogramms müssen vor der Initialisierung des ObjectGrids gesetzt werden.

Sie können auch eigene Bereinigungsprogramme schreiben. Weitere Informationen hierzu finden Sie in den Informationen zum Schreiben angepasster Bereinigungsprogramme im *Programmierhandbuch*.

## Angepasstes Bereinigungsprogramm schreiben

WebSphere eXtreme Scale kann für die Verwendung jedes beliebigen Bereinigungsalgorithmus erweitert werden.

Sie müssen ein angepasstes Bereinigungsprogramm erstellen, das die Schnittstelle "Evictor" implementiert, und den allgemeinen Konventionen für eXtreme-Scale-Plug-ins folgen. Im Folgenden sehen Sie die Schnittstelle:

```
public interface Evictor
{
    void initialize(BackingMap map, EvictionEventCallback callback);
    void activate();
    void apply(LogSequence sequence);
    void deactivate();
    void destroy();
}
```

- Die Methode "initialize" wird während der Initialisierung des BackingMap-Objekts aufgerufen. Diese Methode initialisiert ein Evictor-Plug-in mit einer Referenz auf die BackingMap und eine Referenz auf ein Objekt, das die Schnittstelle "com.ibm.websphere.objectgrid.plugins.EvictionEventCallback" implementiert.
- Die Methode "activate" wird aufgerufen, um das Evictor-Plug-in zu aktivieren. Nach dem Aufruf dieser Methode kann das Evictor-Plug-in die Schnittstelle "EvictionEventCallback" verwenden, um Map-Einträge zu entfernen. Wenn das Evictor-Plug-in versucht, EvictionEventCallbackinterface vor dem Aufruf der Methode "activate" für das Entfernen von Map-Einträgen zu verwenden, wird eine Ausnahme vom Typ "IllegalStateException" ausgelöst.
- Die Methode "apply" wird aufgerufen, wenn Transaktionen, die auf einen oder mehrere Einträge der BackingMap zugreifen, festgeschrieben werden. Der Methode "apply" wird eine Referenz auf ein Objekt übergeben, die die Schnittstelle "com.ibm.websphere.objectgrid.plugins.LogSequence" implementiert. Die Schnittstelle "LogSequence" ermöglicht einem Evictor-Plug-in, die BackingMap-Einträge zu bestimmen, die von einer Transaktion erstellt, geändert oder entfernt wurden. Ein Evictor-Plug-in verwendet diese Informationen, um zu entscheiden, wann und welche Einträge zu entfernen sind.
- Die Methode "deactivate" wird aufgerufen, um das Evictor-Plug-in zu inaktivieren. Nach dem Aufruf dieser Methode darf das Evictor-Plug-in die Schnittstelle "EvictionEventCallback" nicht mehr für das Entfernen von Map-Einträgen verwenden. Wenn das Evictor-Plug-in die Schnittstelle "EvictionEventcallback" nach dem Aufruf dieser Methode verwendet, wird eine Ausnahme vom Typ "IllegalStateException" ausgelöst.
- Die Methode "destroy" wird aufgerufen, wenn die BackingMap gelöscht wird. Diese Methode ermöglicht einem Evictor-Plug-in, alle Threads zu beenden, die von ihm erstellt wurden.

Die Schnittstelle "EvictionEventCallback" hat die folgenden Methoden:

```
public interface EvictionEventCallback
{
    void evictMapEntries(List evictorDataList) throws ObjectGridException;
    void evictEntries(List keysToEvictList) throws ObjectGridException;
    void setEvictorData(Object key, Object data);
    Object getEvictorData(Object key);
}
```

Die `EvictionEventCallback`-Methoden werden wie folgt von einem Evictor-Plug-in verwendet, um einen Rückruf an das eXtreme-Scale-Framework zu senden:

- Die Methode `setEvictorData` wird von einem Evictor-Plug-in verwendet, um das Framework anzufordern, das verwendet wird, um ein von ihm erstelltes Evictor-Objekt zu speichern und dieses dem Eintrag zuzuordnen, der mit dem Schlüsselargument angegeben ist. Die Daten sind Evictor-spezifisch und werden über die Informationen bestimmt, die das Evictor-Plug-in benötigt, um den verwendeten Algorithmus zu implementieren. Bei einem LFU-Algorithmus verwaltet das Bereinigungsprogramm beispielsweise einen Zähler im Evictor-Datenobjekt, mit dem es verfolgt, wie oft die Methode `apply` mit einem `LogElement` aufgerufen wird, das auf einen Eintrag für einen bestimmten Schlüssel verweist.
- Die Methode `getEvictorData` wird von einem Evictor-Plug-in verwendet, um die Daten abzurufen, die bei einem vorherigen Aufruf der Methode `apply` an die Methode `setEvictorData` übergeben wurden. Wenn keine Evictor-Daten für das angegebene Schlüsselargument gefunden werden, wird ein spezielles Objekt `KEY_NOT_FOUND`, das in der Schnittstelle `EvictorCallback` definiert ist, zurückgegeben.
- Die Methode `evictMapEntries` wird von einem Evictor-Plug-in verwendet, um das Entfernen eines oder mehrerer Map-Einträge anzufordern. Jedes Objekt im Parameter `evictorDataList` muss die Schnittstelle `com.ibm.websphere.objectgrid.plugins.EvictorData` implementieren. Außerdem muss die `EvictorData`-Instanz, die an die Methode `setEvictorData` übergeben wird, im Parameter für die Evictor-Datenliste dieser Methode enthalten sein. Die Methode `getKey` der Schnittstelle `EvictorData` wird verwendet, um den zu entfernenden Map-Eintrag zu bestimmen. Der Map-Eintrag wird entfernt, wenn der Cacheeintrag derzeit dieselbe `EvictorData`-Instanz enthält, die in der Evictor-Datenliste für diesen Cacheeintrag enthalten ist.
- Die Methode `evictEntries` wird von einem Evictor-Programm verwendet, um das Entfernen eines oder mehrerer Map-Einträge anzufordern. Diese Methode wird nur verwendet, wenn das Objekt, das an die Methode `setEvictorData` übergeben wird, nicht die Schnittstelle `com.ibm.websphere.objectgrid.plugins.EvictorData` implementiert.

eXtreme Scale ruft die Methode `apply` der Evictor-Schnittstelle nach Abschluss einer Transaktion auf. Alle Transaktionssperren, die von der abgeschlossenen Transaktion angefordert wurden, werden freigegeben. Potenziell können mehrere Threads die Methode `apply` gleichzeitig aufrufen, und jeder Thread kann seine eigene Transaktion abschließen. Da Transaktionssperren bereits von der abgeschlossenen Transaktion freigegeben werden, muss die Methode `apply` eine eigene Synchronisation durchführen, um sicherzustellen, dass sie Thread-sicher ist.

Der Grund für die Implementierung der Schnittstelle `EvictorData` und die Verwendung der Methode `evictMapEntries` an Stelle der Methode `evictEntries` ist das Schließen eines potenziellen Zeitfensters. Stellen Sie sich die folgende Abfolge von Ereignissen vor:

1. Transaktion 1 wird abgeschlossen und ruft die Methode `apply` mit einer `LogSequence` auf, die den Map-Eintrag für Schlüssel 1 löscht.
2. Transaktion 2 wird abgeschlossen und ruft die Methode `apply` mit einer `LogSequence` auf, die einen neuen Map-Eintrag für Schlüssel 1 einfügt. Anders ausgedrückt, Transaktion 2 erstellt den Map-Eintrag erneut, der von Transaktion 1 gelöscht wurde.

Da das Bereinigungsprogramm asynchron zu Threads ausgeführt wird, die Transaktionen ausführen, ist es möglich, dass das Bereinigungsprogramm, wenn es ent-

scheidet, Schlüssel 1 zu entfernen, den Map-Eintrag entfernt, der vor Abschluss von Transaktion 1 vorhanden war, oder den Map-Eintrag, der von Transaktion 2 erneut erstellt wurde. Um Zeitfenster zu schließen und die Unsicherheit in Bezug darauf auszuschließen, welche Map-Eintragsversion von Schlüssel 1 das Bereinigungsprogramm zu entfernen beabsichtigte, implementieren Sie die Schnittstelle "EvictorData" über das Objekt, das an die Methode "setEvictorData" übergeben wird. Verwenden Sie für die gesamte Lebensdauer eines Map-Eintrags dieselbe EvictorData-Instanz. Wenn dieser Map-Eintrag gelöscht und anschließend von einer anderen Transaktion neu erstellt wird, muss das Bereinigungsprogramm eine neue Instanz der EvictorData-Implementierung verwenden. Mit Hilfe der EvictorData-Implementierung und der Methode "evictMapEntries" kann das Bereinigungsprogramm sicherzustellen, dass der Map-Eintrag nur dann entfernt wird, wenn der Cacheintrag, der dem Map-Eintrag zugeordnet ist, die richtige EvictorData-Instanz enthält.

Die Schnittstellen "Evictor" und "EvictionEventCallback" ermöglichen einer Anwendung, ein Bereinigungsprogramm zu integrieren, das einen benutzerdefinierten Algorithmus für die Bereinigung implementiert. Das folgende Code-Snippet veranschaulicht, wie Sie die Methode "initialize" der Schnittstelle "Evictor" implementieren können:

```
import com.ibm.websphere.objectgrid.BackingMap;
import com.ibm.websphere.objectgrid.plugins.EvictionEventCallback;
import com.ibm.websphere.objectgrid.plugins.Evictor;
import com.ibm.websphere.objectgrid.plugins.LogElement;
import com.ibm.websphere.objectgrid.plugins.LogSequence;
import java.util.LinkedList;
// Instanzvariablen
private BackingMap bm;
private EvictionEventCallback evictorCallback;
private LinkedList queue;
private Thread evictorThread;
public void initialize(BackingMap map, EvictionEventCallback callback)
{
    bm = map;
    evictorCallback = callback;
    queue = new LinkedList();
    // spawn evictor thread
    evictorThread = new Thread( this );
    String threadName = "MyEvictorForMap-" + bm.getName();
    evictorThread.setName( threadName );
    evictorThread.start();
}
```

Der vorherige Code speichert die Referenzen auf die Map und Callback-Objekte in Instanzvariablen, so dass sie den Methoden "apply" und "destroy" zur Verfügung stehen. In diesem Beispiel wird eine verkettete Liste erstellt, die als FIFO-Warteschlange für die Implementierung eines LRU-Algorithmus verwendet wird. Es wird ein Thread gestartet und eine Referenz auf den Thread als Instanzvariable verwaltet. Mit dieser Referenz ist die Methode "destroy" in der Lage, den gestarteten Thread zu unterbrechen und zu beenden.

Das folgende Code-Snippet veranschaulicht, wie die Methode "apply" der Schnittstelle "Evictor" implementiert werden kann. In diesem Code werden die Synchronisationsanforderungen für die Gewährleistung der Thread-Sicherheit des Codes ignoriert:

```
import com.ibm.websphere.objectgrid.BackingMap;
import com.ibm.websphere.objectgrid.plugins.EvictionEventCallback;
import com.ibm.websphere.objectgrid.plugins.Evictor;
import com.ibm.websphere.objectgrid.plugins.EvictorData;
import com.ibm.websphere.objectgrid.plugins.LogElement;
import com.ibm.websphere.objectgrid.plugins.LogSequence;
public void apply(LogSequence sequence)
{
    Iterator iter = sequence.getAllChanges();
    while(iter.hasNext())
    {
        LogElement elem = (LogElement)iter.next();
        Object key = elem.getKey();
        LogElement.Type type = elem.getType();
    }
}
```





```

// Die Methode "destroy" unterbricht einfach den von der Methode "initialize" gestarteten Thread
public void destroy() {
    evictorThread.interrupt();
}

// Es folgt die Methode "run" des Threads, der von der Methode "initialize" gestartet wurde
public void run()
{
    // Schleife, bis die Methode "destroy" diesen Thread unterbricht
    boolean continueToRun = true;
    while ( continueToRun )
    {
        try
        {
            // Eine weile inaktiv bleiben, bevor die Warteschlange bereinigt wird
            // sleepTime eignet sich bestens als Eigenschaft, die für ein
            // Bereinigungsprogramm gesetzt werden kann
            Thread.sleep( sleepTime );
            int queueSize = queue.size();
            // Einträge entfernen, wenn die Warteschlangengröße die definierte
            // Maximalgröße überschreitet. Die maximale Größe ist also
            // ebenfalls eine Eigenschaft des Bereinigungsprogramms.
            int numToEvict = queueSize - maxSize;
            if ( numToEvict > 0 )
            {
                // Eintrag am Ende der Warteschlange entfernen, da sich
                // dort der Eintrag mit dem ältesten Verwendungsdatum befindet
                List evictList = new ArrayList( numToEvict );
                while( queueSize > ivMaxSize )
                {
                    EvictorData data = null;
                    try
                    {
                        EvictorData data = (EvictorData) queue.removeLast();
                        evictList.add( data );
                        queueSize = queue.size();
                    }
                    catch ( NoSuchElementException nse )
                    {
                        // Warteschlange ist leer
                        queueSize = 0;
                    }
                }
                // Bereinigung anfordern, wenn die Schlüsselliste nicht leer ist
                if ( ! evictList.isEmpty() )
                {
                    evictorCallback.evictMapEntries( evictList );
                }
            }
        }
        catch ( InterruptedException e )
        {
            continueToRun = false;
        }
    } // end while loop
} // Ende der Methode "run"

```

## Optionale Schnittstelle "RollBackEvictor"

Die Schnittstelle "com.ibm.websphere.objectgrid.plugins.RollbackEvictor" kann von einem Evictor-Plug-in optional implementiert werden. Wenn diese Schnittstelle implementiert wird, kann ein Bereinigungsprogramm nicht nur aufgerufen werden, wenn Transaktionen festgeschrieben werden, sondern auch, wenn Transaktionen rückgängig gemacht werden (Rollback).

```

public interface RollbackEvictor
{
    void rollingBack( LogSequence ls );
}

```

Die Methode "apply" wird nur aufgerufen, wenn eine Transaktion festgeschrieben wird. Wenn eine Transaktion rückgängig gemacht wird und die Schnittstelle "RollbackEvictor" vom Bereinigungsprogramm implementiert wird, wird eine Methode "rollingBack" aufgerufen. Wird die Schnittstelle "RollbackEvictor" nicht implementiert und die Transaktion rückgängig gemacht, werden die Methoden "apply" und "rollingBack" nicht aufgerufen.

## Bewährte Verfahren zum Erreichen einer optimalen Leistung der Plug-in-Bereinigungsprogramme

Wenn Sie Plug-in-Bereinigungsprogramme (Evictor) verwenden, bleiben diese so lange inaktiv, bis Sie sie erstellen und die BackingMap anweisen, sie zu verwenden. Verwenden Sie die folgenden bewährten Verfahren und Leistungstipps für LFU- und LRU-Bereinigungsprogramme.

### LFU-Bereinigungsprogramm

Das Konzept von LFU-Bereinigungsprogrammen sieht vor, dass Einträge aus der Map entfernt werden, die nur selten verwendet werden. Die Einträge der Map sind auf eine festgelegte Menge binärer Heap-Speicher verteilt. Je öfter ein bestimmter Cacheeintrag verwendet wird, desto höher wird er im Heap-Speicher eingereiht. Wenn das Bereinigungsprogramm versucht, Bereinigungen durchzuführen, entfernt es nur die Cacheeinträge, die sich unterhalb eines bestimmten Punkts des binären Heap-Speichers befinden. Deshalb werden nur die so genannten LFU-Einträge (Last Frequently Used, am wenigsten verwendet) Einträge bereinigt.

### LRU-Bereinigungsprogramm

Das LRU-Bereinigungsprogramm folgt bis auf wenige Unterschiede denselben Konzepten wie das LFU-Bereinigungsprogramm. Der Hauptunterschied besteht darin, dass das LRU-Bereinigungsprogramm eine First In/First Out-Warteschlange (FIFO) an Stelle einer Gruppe binärer Heap-Speicher verwendet. Jedesmal, wenn auf einen Cacheeintrag zugegriffen wird, wird dieser an den Anfang der Warteschlange gestellt. Deshalb stehen oben in der Warteschlange die am häufigsten verwendeten Map-Eintrag und unten in der Warteschlange die am wenigsten verwendeten Map-Einträge. Beispiel: Der Cacheeintrag A wird 50 Mal verwendet, und der Cacheeintrag B wird nur ein einziges Mal direkt nach Cacheeintrag A verwendet. In dieser Situation steht der Cacheeintrag B am Anfang der Warteschlange, weil er zuletzt verwendet wurde, und der Cacheeintrag A steht am Ende der Warteschlange. Das LRU-Bereinigungsprogramm entfernt die Cacheeinträge, die sich hinten in der Warteschlange befinden, weil es sich hierbei um die LRU-Map-Einträge (Least Recently Used) handelt, d. h., deren Verwendungszeit am ältesten ist.

### LFU- und LRU-Eigenschaften und bewährte Verfahren zur Leistungsverbesserung

#### Anzahl der Heap-Speicher

Wenn Sie das LFU-Bereinigungsprogramm verwenden, werden alle Cacheeinträge für eine bestimmte Map über die von Ihnen angegebene Anzahl von Heap-Speichern sortiert, was die Leistung erheblich verbessert und verhindert, dass alle Bereinigungsoperationen in einem einzigen binären Heap-Speicher synchronisiert werden müssen, der die gesamte Sortierung für die Map enthält. Eine höhere Anzahl an Heap-Speichern verringert auch die erforderliche Zeit für die Neusortierung der Heap-Speicher, weil jeder Heap-Speicher weniger Einträge enthält. Setzen Sie die Anzahl der Heap-Speicher auf 10 % der Eintragsanzahl in Ihrer BaseMap.

#### Anzahl der Warteschlangen

Wenn Sie das LRU-Bereinigungsprogramm verwenden, werden alle Cacheeinträge für eine bestimmte Map über die von Ihnen angegebene Anzahl von LRU-Warteschlangen sortiert, was die Leistung erheblich verbessert und verhindert, dass alle

Bereinigungsoperationen in einer einzigen Warteschlange synchronisiert werden müssen, die die gesamte Sortierung für die Map enthält. Setzen Sie die Anzahl der Warteschlangen auf 10 % der Eintragsanzahl in Ihrer BaseMap.

### **Eigenschaft "MaxSize"**

Wenn ein LFU- oder LRU-Bereinigungsprogramm mit dem Entfernen von Einträgen beginnt, verwendet es die Bereinigungsprogrammeigenschaft "MaxSize", um festzustellen, wie viele binäre Heap-Speicher bzw. LRU-Warteschlangenelemente bereinigt werden müssen. Angenommen, Sie legen die Anzahl der Heap-Speicher bzw. Warteschlangen so fest, dass je Map-Warteschlange ungefähr 10 Map-Einträge enthält. Wenn die Eigenschaft "MaxSize" auf 7 gesetzt ist, entfernt das Bereinigungsprogramm 3 Einträge aus jedem Heap-Speicher bzw. Warteschlangenobjekt, um die Größe der einzelnen Heap-Speicher bzw. Warteschlangen wieder auf 7 zurückzusetzen. Das Bereinigungsprogramm entfernt nur dann Map-Einträge aus einem Heap-Speicher bzw. aus einer Warteschlange, wenn die Anzahl der im Heap-Speicher bzw. in der Warteschlange enthaltenen Elemente höher ist als der Wert der Eigenschaft "MaxSize". Setzen Sie die Eigenschaft "MaxSize" auf 70 % Ihrer Heap-Speicher- bzw. Warteschlangengröße. In diesem Beispiel wird die Eigenschaft auf 7 gesetzt. Sie können die ungefähre Größe jedes Heap-Speichers bzw. jeder Warteschlange bestimmen, indem Sie die Anzahl der BaseMap-Einträge durch die Anzahl der verwendeten Heap-Speicher bzw. Warteschlangen teilen.

### **Eigenschaft "SleepTime"**

Ein Bereinigungsprogramm entfernt nicht ständig Einträge aus der Map. Vielmehr ist es eine gewisse Zeit inaktiv und prüft die Map nur alle n Sekunden, wobei n für den Wert der Eigenschaft "SleepTime" steht. Diese Eigenschaft wirkt sich auch positiv auf die Leistung aus. Wird die Bereinigung zu häufig durchgeführt, verringert sich die Leistung, weil Ressourcen für die Bereinigung benötigt werden. Wird das Bereinigungsprogramm zu selten ausgeführt, können sich Einträge in einer Map ansammeln, die eigentlich nicht benötigt werden. Eine Map, die sehr viele nicht benötigte Einträge enthält, kann sich nachteilig auf den Speicherbedarf und auf die Verarbeitungsressourcen auswirken, die für Ihre Map erforderlich sind. Für die meisten Maps empfiehlt es sich, das Bereinigungsintervall auf 15 Sekunden zu setzen. Wenn eine Map viele Schreiboperationen und eine hohe Transaktionsrate aufweist, sollten Sie das Intervall verringern. Wird nur selten auf die Map zugegriffen, können Sie einen höheren Wert festlegen.

### **Beispiel**

Der Beispielcode definiert eine Map, definiert, erstellt ein neues LFU-Bereinigungsprogramm, setzt die Eigenschaften für das Bereinigungsprogramm und stellt die Map so ein, dass das Bereinigungsprogramm verwendet wird:

```
// ObjectGridManager zum Erstellen/Abrufen des ObjectGrids verwenden (siehe den
// Abschnitt zu ObjectGridManger)
ObjectGrid objGrid = ObjectGridManager.create.....
BackingMap bMap = objGrid.defineMap("SomeMap");

// Eigenschaften, ausgehend von 50.000 Map-Einträgen, definieren
LFUEvictor someEvictor = new LFUEvictor();
someEvictor.setNumberOfHeaps(5000);
someEvictor.setMaxSize(7);
someEvictor.setSleepTime(15);
bMap.setEvictor(someEvictor);
```

Die Verwendung des LRU-Bereinigungsprogramms ist der eines LFU-Bereinigungsprogramms sehr ähnlich. Es folgt ein Beispiel:

```
ObjectGrid objGrid = new ObjectGrid;  
BackingMap bMap = objGrid.defineMap("SomeMap");  
  
// Eigenschaften, ausgehend von 50.000 Map-Einträgen, definieren  
LRUEvictor someEvictor = new LRUEvictor();  
someEvictor.setNumberOfLRUQueues(5000);  
someEvictor.setMaxSize(7);  
someEvictor.setSleepTime(15);  
bMap.setEvictor(someEvictor);
```

Beachten Sie, dass sich nur zwei Zeilen vom Beispiel für das LFU-Bereinigungsprogramm unterscheiden.

---

## Bewährte Verfahren für die Sperrleistung

Sperrstrategien und Transaktionsisolationseinstellungen wirken sich auf die Leistung Ihrer Anwendungen aus.

### Zwischengespeicherte Instanz abrufen

Weitere Informationen finden Sie in der Dokumentation zum Sperren von Map-Einträgen im *Administratorhandbuch*.

### Pessimistische Sperrstrategie

Verwenden Sie die pessimistische Sperrstrategie für Lese- und Schreiboperationen in Maps, wenn die Anzahl der Schlüsselkollisionen sehr hoch ist. Die pessimistische Sperrstrategie hat die größten Auswirkungen auf die Leistung.

#### Transaktionsisolationsstufen "read committed" und "read uncommitted"

Wenn Sie eine pessimistische Sperrstrategie verwenden, setzen Sie die Transaktionsisolationsstufe mit der Methode "Session.setTransactionIsolation". Für die Isolationsstufen "read committed" und "read uncommitted" verwenden Sie das Argument "Session.TRANSACTION\_READ\_COMMITTED" bzw. das Argument "Session.TRANSACTION\_READ\_UNCOMMITTED". Wenn Sie die Transaktionsisolationsstufe auf das pessimistische Standardsperrverhalten zurücksetzen möchten, verwenden Sie die Methode "Session.setTransactionIsolation" mit dem Argument "Session.REPEATABLE\_READ".

Die Isolationsstufe "read committed" verringert die Dauer gemeinsamer Sperren, was die Anzahl gemeinsamer Zugriffe erhöhen und das Risiko von Deadlocks verringern kann. Diese Isolationsstufe sollte verwendet werden, wenn eine Transaktion keine Zusicherung benötigt, dass gelesene Werte für die Dauer der Transaktion unverändert bleiben.

Verwenden Sie die Isolationsstufe "uncommitted read", wenn die Transaktion die festgeschriebenen Daten nicht sehen muss.

### Optimistische Sperrstrategie

Optimistisches Sperren ist die Standardkonfiguration. Diese Strategie bietet im Vergleich mit der pessimistischen Sperrstrategie sowohl eine bessere Leistung als auch eine bessere Skalierbarkeit. Verwenden Sie diese Strategie, wenn Ihre Anwendungen einige Fehler bei optimistischen Aktualisierungen tolerieren können und dabei

eine bessere Leistung bieten als bei der pessimistischen Strategie. Diese Strategie eignet sich bestens für Leseoperationen und Anwendungen, die nur selten Aktualisierungen vornehmen.

### **OptimisticCallback-Plug-in**

Bei der optimistischen Sperrstrategie wird eine Kopie der Cacheeinträge erstellt, und diese werden dann bei Bedarf mit den Originaleinträgen verglichen. Diese Operation kann kostenintensiv sein, weil das Kopieren der Einträge Klon- und Serialisierungsoperationen umfassen kann. Um die beste Leistung zu erzielen, implementieren Sie das angepasste Plug-in für Maps, die keine Entitäts-Maps sind.

Weitere Informationen finden Sie im Abschnitt . Weitere Informationen zum OptimisticCallback-Plug-in finden Sie im Handbuch *Produktübersicht*.

### **Versionsfelder für Entitäten verwenden**

Wenn Sie optimistisches Sperren für Entitäten verwenden, verwenden Sie die Annotation "@Version" oder ein äquivalentes Attribut in der Metadatedeskriptor-datei der Entität. Die Versionsannotation bietet ObjectGrid eine effiziente Methode für die Verfolgung der Versionen eines Objekts. Wenn die Entität kein Versionsfeld hat und optimistisches Sperren für die Entität verwendet wird, muss die vollständige Entität kopiert und verglichen werden.

### **Strategie ohne Sperren verwenden**

Verwenden Sie die Strategie ohne Sperren für Anwendungen, die nur Leseoperationen durchführen. Bei dieser Strategie werden weder Sperren angefordert, noch wird ein Sperrenmanager verwendet. Deshalb bietet diese Strategie die höchste Anzahl gemeinsamer Zugriffe, die höchste Leistung und die höchste Skalierbarkeit.

## **Sperren von Map-Einträgen mit Abfragen und Indizes**

In diesem Abschnitt wird beschrieben, wie die Query-APIs von eXtreme Scale und das Indexierungs-Plug-in "MapRangeIndex" mit Sperren interagieren. Außerdem werden einige bewährte Verfahren vorgestellt, mit denen Sie die Anzahl der gemeinsamen Zugriffe erhöhen und die Anzahl der Deadlocks verringern können, wenn Sie die pessimistische Sperrstrategie verwenden.

### **Übersicht**

Mit der ObjectGrid-Query-API können SELECT-Abfragen von ObjectMap-Cacheobjekten und -Entitäten ausgeführt werden. Bei der Ausführung einer Abfrage verwendet die Abfragesteuerkomponente, sofern möglich, einen MapRangeIndex, um Schlüssel zu suchen, die Werten in der WHERE-Klausel der Abfrage entsprechen, oder um Beziehungen zu überbrücken. Wenn kein Index verfügbar ist, scannt die Abfragesteuerkomponente jeden Eintrag in einer oder mehreren Maps, um die entsprechenden Entitäten zu finden. Sowohl die Abfragesteuerkomponente als auch die Index-Plug-ins fordern je nach Sperrstrategie, Transaktionsisolationsstufe und Transaktionsstatus Sperren an, um die Datenkonsistenz zu prüfen.

### **Sperren mit dem HashIndex-Plug-in**

Mit dem HashIndex-Plug-in von eXtreme Scale können Schlüssel auf der Basis eines einzelnen Attributs, das im Cacheeintragswert gespeichert ist, gesucht wer-

den. Im Index wird der indexierte Wert in einer von der Cache-Map abgesonderten Datenstruktur gespeichert. Der Index vergleicht die die Schlüssel mit den Map-Einträgen, bevor er sie an den Benutzer zurückgibt, um eine genaue Ergebnismenge zu erhalten. Wenn Sie die pessimistische Sperrstrategie verwenden und der Index für eine lokale ObjectMap-Instanz (im Gegensatz zu einer Client/Server-ObjectMap) verwendet wird, fordert der Index eine Sperre für jeden übereinstimmenden Eintrag an. Wenn Sie optimistisches Sperren oder eine ferne ObjectMap verwenden, werden die Sperren immer sofort freigegeben.

Der Typ der angeforderten Sperre richtet sich nach dem Argument "forUpdate", das an die Methode "ObjectMap.getIndex" übergeben wird. Das Argument "forUpdate" gibt den Typ der Sperre an, die der Index anfordern sollte. Wenn dieses Argument den Wert "false" hat, wird eine gemeinsame Sperre (S-Sperre) angefordert, wenn es den Wert "true" hat, wird eine aktualisierbare Sperre (U-Sperre) angefordert.

Bei einer S-Sperre wird die Einstellung der Transaktionsisolationsstufe für die Sitzung angewendet, die sich auf die Dauer der Sperre auswirkt. Im Abschnitt zu den Transaktionsisolationsstufen wird ausführlich beschreiben, wie über die Transaktionsisolationsstufen die Anzahl gemeinsamer Zugriffe auf Anwendungen erhöht werden kann.

## **Gemeinsame Sperren mit Abfrage**

Die Abfragesteuerkomponente von eXtreme Scale fordert bei Bedarf S-Sperren an, um die Cacheinträge dahingehend zu überwachen, ob sie die Filterkriterien der Abfrage erfüllen. Wenn Sie die Transaktionsisolationsstufe "repeatable read" (wiederholbares Lesen) mit pessimistischem Sperren verwenden, werden die S-Sperren nur für die Elemente beibehalten, die im Abfrageergebnis enthalten sind, und für die Einträge, die nicht im Ergebnis enthalten sind, freigegeben. Wenn Sie eine niedrigere Transaktionsisolationsstufe oder optimistisches Sperren verwenden, werden die S-Sperren nicht beibehalten.

## **Gemeinsame Sperren mit Client/Server-Abfrage**

Wenn Sie die eXtreme-Scale-Abfrage über einen Client absetzen, wird die Abfrage gewöhnlich im Server ausgeführt, sofern nicht alle Maps oder Entitäten, die in der Abfrage referenziert werden, lokal im Client vorhanden sind (z. B., wenn es sich um eine im Client replizierte Map oder Abfrageergebnisentität handelt). Alle Abfragen, die in einer Transaktion mit Lese-/Schreibzugriff ausgeführt werden, halten S-Sperren so, wie es im vorherigen Abschnitt beschrieben wurde. Wenn die Transaktion keine Transaktion mit Lese-/Schreibzugriff ist, wird keine Sitzung im Server gehalten, und die S-Sperren werden freigegeben.

Eine Transaktion mit Lese-/Schreibzugriff wird nur an eine primäre Partition weitergeleitet, und es wird eine Sitzung im Server für die Clientsitzung verwaltet. Eine Transaktion kann unter den folgenden Bedingungen in eine Transaktion mit Lese-/Schreibzugriff hochgestuft werden:

1. Alle Maps, die für die Verwendung pessimistischen Sperrens konfiguriert sind, werden über die ObjectMap-API-Methoden "get" und "getAll" bzw. die Entity-Manager.find-Methoden aufgerufen.
2. Für die Transaktion wird eine Flush-Operation ausgeführt, was dazu führt, dass Aktualisierungen an den Server gesendet werden.

3. Alle Maps, die für die Verwendung optimistischen Sperrrens konfiguriert sind, werden über die Methode "ObjectMap.getForUpdate" oder "EntityManager.findForUpdate" aufgerufen.

## Aktualisierbare Sperren mit Abfrage

Gemeinsame Sperren sind hilfreich, wenn gemeinsamer Zugriff und Konsistenz wichtig sind. Sie gewährleisten, dass sich der Wert eines Eintrags während der gesamten Lebensdauer der Transaktion nicht ändert. Der Wert kann von anderen Transaktionen nicht geändert werden, während andere S-Sperren gehalten werden, und nur eine einzige andere Transaktion kann ihre Absicht zum Aktualisieren des Eintrags manifestieren. Einzelheiten zu S-, U- und X-Sperren finden Sie im Abschnitt zum pessimistischen Sperrmodus.

Aktualisierbare Sperren werden verwendet, um die Absicht zum Aktualisieren eines Cacheeintrags bekannt zu geben, wenn die pessimistische Sperrstrategie verwendet wird. Sie lassen eine Synchronisation von Transaktionen zu, die einen Cacheeintrag ändern möchten. Transaktionen können den Eintrag weiterhin über eine S-Sperre anzeigen, aber andere Transaktionen werden an der Anforderung einer U- bzw. X-Sperre gehindert. In vielen Szenarios ist die Anforderung einer U-Sperre ohne vorherige Anforderung einer S-Sperre erforderlich, um Deadlocks zu vermeiden. Beispiele für häufige Deadlocks finden Sie im Abschnitt zum pessimistischen Sperrmodus.

Die Schnittstellen "ObjectQuery" und "EntityManager Query" stellen die Methode "setForUpdate" bereit, um die beabsichtigte Verwendung für das Abfrageergebnis anzugeben. Die Abfragesteuerkomponente fordert U-Sperren an Stelle von S-Sperren für jeden Map-Eintrag an, der im Abfrageergebnis enthalten ist:

```
ObjectMap orderMap = session.getMap("Order");
ObjectQuery q = session.createQuery("SELECT o FROM Order o WHERE o.orderDate=?1");
q.setParameter(1, "20080101");
q.setForUpdate(true);
session.begin();
// Abfrage ausführen. Jeder Auftrag (order) hat eine U-Sperre.
Iterator result = q.getResultIterator();
// Für jeden Auftrag (order) den Status aktualisieren.
while(result.hasNext()) {
    Order o = (Order) result.next();
    o.status = "shipped";
    orderMap.update(o.getId(), o);
}
// Festschreibung
session.commit();

Query q = em.createQuery("SELECT o FROM Order o WHERE o.orderDate=?1");
q.setParameter(1, "20080101");
q.setForUpdate(true);
emTran.begin();
// Abfrage ausführen. Jeder Auftrag (order) hat eine U-Sperre.
Iterator result = q.getResultIterator();
// Für jeden Auftrag (order) den Status aktualisieren.
while(result.hasNext()) {
    Order o = (Order) result.next();
    o.status = "shipped";
}
tmTran.commit();
```

Wenn das Attribut **setForUpdate** aktiviert ist, wird die Transaktion automatisch in eine Transaktion mit Lese-/Schreibzugriff konvertiert, und die Sperren werden im Server wie erwartet gehalten. Falls die Abfrage keine Indizes verwenden kann, muss die Map gescannt werden, was dazu führt, dass temporäre U-Sperren für

Map-Einträge gesetzt werden, die nicht im Abfrageergebnis enthalten sind, und U-Sperren für Einträge gehalten werden, die im Ergebnis enthalten sind.

---

## Bewährte Verfahren für die Schnittstelle "ObjectTransformer"

Die Schnittstelle "ObjectTransformer" verwendet Callbacks an die Anwendung, um angepasste Implementierungen gängiger und kostenintensiver Operationen, wie z. B. Objektserialisierung und tiefe Kopien von Objekten, zu unterstützen.

### Übersicht

Einzelheiten zur Schnittstelle "ObjectTransformer" finden Sie im Abschnitt „ObjectTransformer-Plug-in“ auf Seite 197. Aus Leistungsgründen, und wie auch aus den Informationen zur Methode "CopyMode" im Abschnitt zu den bewährten Verfahren für die Methode "CopyMode" hervorgeht, kopiert eXtreme Scale die Werte in allen Fällen außer bei der Verwendung des Modus NO\_COPY. Der Standardkopiermechanismus, der in eXtreme Scale eingesetzt wird, ist die Serialisierung, die bekanntermaßen eine kostenintensive Operation ist. In dieser Situation kommt die Schnittstelle "ObjectTransformer" zur Anwendung. Die Schnittstelle "ObjectTransformer" verwendet Callbacks an die Anwendung, um eine angepasste Implementierung gängiger und kostenintensiver Operationen, wie z. B. Objektserialisierung und tiefe Kopien für Objekte, zu unterstützen.

Eine Anwendung kann eine Implementierung der Schnittstelle "ObjectTransformer" für eine Map bereitstellen. eXtreme Scale delegiert die Arbeit dann an die Methoden in diesem Objekt und verlässt sich darauf, dass die Anwendung eine optimierte Version jeder Methode in der Schnittstelle bereitstellt. Im Folgenden sehen Sie die Schnittstelle "ObjectTransformer":

```
public interface ObjectTransformer {
    void serializeKey(Object key, ObjectOutputStream stream) throws IOException;
    void serializeValue(Object value, ObjectOutputStream stream) throws IOException;
    Object inflateKey(ObjectInputStream stream) throws IOException, ClassNotFoundException;
    Object inflateValue(ObjectInputStream stream) throws IOException, ClassNotFoundException;
    Object copyValue(Object value);
    Object copyKey(Object key);
}
```

Über den folgenden Beispielcode können Sie einer BackingMap eine Schnittstelle "ObjectTransformer" zuordnen:

```
ObjectGrid g = ...;
BackingMap bm = g.defineMap("PERSON");
MyObjectTransformer ot = new MyObjectTransformer();
bm.setObjectTransformer(ot);
```

### Objektserialisierung und -dekomprimierung optimieren

Die Objektserialisierung ist gewöhnlich der wichtigste Leistungsaspekt in eXtreme Scale. Sie verwendet den Standardmechanismus "Serializable", wenn kein ObjectTransformer-Plug-in von der Anwendung angegeben wird. Eine Anwendung kann Implementierungen der Serializable-Methoden "readObject" und "writeObject" bereitstellen oder von den Objekten die Schnittstelle "Externalizable" implementieren lassen, die ungefähr zehn Mal schneller ist. Wenn die Objekte in der Map nicht geändert werden können, kann eine Anwendung der ObjectMap eine Schnittstelle "ObjectTransformer" zuordnen. Die Methoden "serialize" und "inflate" werden bereitgestellt, um der Anwendung die Bereitstellung angepassten Codes für die Optimierung dieser Operationen bereitzustellen, da ihr Leistungseinfluss auf das System sehr hoch ist. Die Methode "serialize" serialisiert das Objekt in den bereitgestellten Datenstrom. Die Methode "inflate" liefert den Eingabedatenstrom und erwartet von der Anwendung, dass diese das Objekt erstellt, das Objekt anhand



der Daten im Datenstrom dekomprimiert und das Objekt dann zurückgibt. Implementierungen der Methoden "serialize" und "inflate" müssen einander spiegeln.

## Operationen für tiefe Kopien optimieren

Wenn eine Anwendung ein Objekt von einer ObjectMap empfängt, führt eXtreme Scale eine tiefe Kopie des Objektwerts durch, um sicherzustellen, dass die Datenintegrität der Kopie in der BaseMap-Map gewahrt bleibt. Anschließend kann die Anwendung den Objektwert beruhigt ändern. Beim Festschreiben der Transaktion wird die Kopie des Objektwerts in der BaseMap-Map in den neuen, geänderten Wert aktualisiert, und die Anwendung verwendet diesen Wert von diesem Zeitpunkt an nicht mehr. Sie hätten das Objekt in der Festschreibungsphase erneut kopieren können, um eine private Kopie zu erstellen, aber in diesem Fall wurden die Leistungskosten dieser Transaktion gegen die Anweisung an den Anwendungsprogrammierer, den Wert nach der Transaktionsfestschreibung nicht zu verwenden, abgewogen. Der Standard-ObjectTransformer versucht, einen Klon oder eine Kombination der Methoden "serialize" und "inflate" zu verwenden, um eine Kopie zu generieren. Die Kombination der Methoden "serialize" und "inflate" ist das Szenario mit der schlechtesten Leistung. Wenn bei der Profilerstellung festgestellt wird, dass die Ausführung der Methoden "serialize" und "inflate" ein Problem für Ihre Anwendung darstellen, schreiben Sie eine entsprechende Methode "clone", um eine tiefe Kopie zu erstellen. Wenn Sie die Klasse nicht ändern können, erstellen Sie ein angepasstes ObjectTransformer-Plug-in, und implementieren Sie weitere effiziente copyValue- und copyKey-Methoden.

---

## Serialisierungsleistung

WebSphere eXtreme Scale verwendet mehrere Java-Prozesse, in denen Daten gespeichert werden. Die Daten, die das Format von Java-Objektinstanzen haben, werden in Bytes und bei Bedarf wieder zurück in Objekte konvertiert, um die Daten zwischen den Client- und Serverprozessen hin- und herzuschieben. Das Marshaling der Daten ist die kostenintensivste Operation und muss vom Anwendungsentwickler beim Design des Schemas, bei der Konfiguration des Grids und bei der Interaktion mit den Datenzugriffs-APIs berücksichtigt werden.

Die Java-Standardserialisierungs- und -Kopiererroutinen sind relativ langsam und können in einem typischen Setup 60 bis 70 Prozent des Prozessors belegen. In den folgenden Abschnitten sind Möglichkeiten zur Verbesserung der Serialisierungsleistung beschrieben.

### ObjectTransformer für jede BackingMap schreiben

Ein ObjectTransformer kann einer BackingMap zugeordnet werden. Ihre Anwendung kann eine Klasse haben, die die Schnittstelle "ObjectTransformer" implementiert und Implementierungen für die folgenden Operationen bereitstellt:

- Werte kopieren
- Schlüssel in und aus Datenströmen serialisieren und dekomprimieren
- Werte in und aus Datenströmen serialisieren und dekomprimieren

Die Anwendung muss keine Schlüssel kopieren, weil Schlüssel als unveränderlich betrachtet werden.

**Anmerkung:** Die Schnittstelle "ObjectTransformer" wird nur aufgerufen, wenn das ObjectGrid die Daten kennt, die umgesetzt werden sollen. Werden beispielsweise DataGrid-API-Agenten verwendet, müssen die Agenten selbst sowie die Daten der

Agenteninstanzen und Daten, die vom Agenten zurückgegeben werden, mit Hilfe angepasster Serialisierungstechniken optimiert werden. Die Schnittstelle "ObjectTransformer" wird nicht für DataGrid-API-Agenten aufgerufen.

## Entitäten verwenden

Wenn Sie die EntityManager-API mit Entitäten verwenden, speichert das ObjectGrid die Entitätsobjekte nicht direkt in den BackingMaps. Die EntityManager-API konvertiert die Entitätsobjekte in Tupelobjekte. Weitere Informationen finden Sie im Abschnitt zur Verwendung eines Loaders mit Entitäts-Maps und Tupeln im *Programmierhandbuch*. Entitäts-Maps werden automatisch einem hoch optimierten ObjectTransformer zugeordnet. Jedesmal, wenn die API "ObjectMap" oder die API "EntityManager" für die Interaktion mit Entitäts-Maps verwendet wird, wird der ObjectTransformer der Entität aufgerufen.

## Angepasste Serialisierung

Es gibt einige Fälle, in denen Objekte für die Verwendung einer angepassten Serialisierung geändert werden müssen, z. B. durch Implementierung der Schnittstelle "java.io.Externalizable" oder durch Implementierung der Methoden "writeObject" und "readObject" für Klassen, die die Schnittstelle "{java.io.Serializable}" implementieren. Sie müssen angepasste Serialisierungstechniken verwenden, wenn die Objekte mit anderen Mechanismen als den Methoden der API "ObjectGrid" oder "EntityManager" serialisiert werden.

Wenn Objekte oder Entitäten beispielsweise als Instanzdaten in einem DataGrid-API-Agenten gespeichert werden oder wenn der Agent Objekte oder Entitäten zurückgibt, werden diese Objekte nicht mit einem ObjectTransformer umgesetzt. Der Agent verwendet jedoch automatisch den ObjectTransformer, wenn Sie die Schnittstelle EntityMixin verwenden. Weitere Einzelheiten finden Sie in der Dokumentation zu DataGrid-Agenten und entitätsbasierten Maps.

## Bytefeldgruppen

Verwenden Sie API "ObjectMap" oder "DataGrid", werden die Schlüssel- und Wertobjekte serialisiert, wenn der Client mit dem Grid interagiert oder wenn die Objekte repliziert werden. Um die Kosten für die Serialisierung zu vermeiden, können Sie Bytefeldgruppen an Stelle von Java-Objekten verwenden. Bytefeldgruppen können wesentlich kostengünstiger im Hauptspeicher gespeichert werden, da das JDK für die Garbage-Collection weniger Objekte durchsuchen muss und die Objekte ausschließlich bei Bedarf dekomprimiert werden können. Bytefeldgruppen können nur verwendet werden, wenn Sie keinen Zugriff auf die Objekte über Abfragen oder Indizes benötigen. Da die Daten in Form von Bytes gespeichert werden, kann der Zugriff auf die Daten nur über ihren Schlüssel erfolgen.

WebSphere eXtreme Scale kann Daten automatisch als Bytefeldgruppen speichern, wenn die Konfigurationsoption "CopyMode.COPY\_TO\_BYTES" für die Map verwendet wird. Die Speicherung kann aber auch manuell vom Client vorgenommen werden. Diese Option speichert die Daten effizient im Speicher und kann die Objekte in der Bytefeldgruppe auch automatisch dekomprimieren, damit sie bei Bedarf für Abfragen und Indizes verwendet werden können.

---

## Optimierung der Abfrageleistung

Verwenden Sie zum Optimieren der Leistung Ihrer Abfragen die folgenden Techniken und Tipps.

## Parameter verwenden

Wenn eine Abfrage ausgeführt wird, muss die Abfragezeichenfolge syntaktisch analysiert und ein Plan für die Abfrage entwickelt werden, was beides kostenintensiv sein kann. WebSphere eXtreme Scale führt die Zwischenspeicherung von Abfrageplänen über die Abfragezeichenfolge durch. Da der Cache eine begrenzte Größe hat, ist es wichtig, Abfragezeichenfolgen nach Möglichkeit wiederzuverwenden. Die Verwendung benannter Parameter und positionsgebundener Parameter kann durch die Förderung der Wiederverwendung von Abfrageplänen ebenfalls zu einer Leistungsverbesserung beitragen.

Beispiel für positionsgebundene Parameter: `Query q = em.createQuery("select c from Customer c where c.surname=?1"); q.setParameter(1, "Claus");`

## Indizes verwenden

Eine ordnungsgemäße Indexierung in einer Map kann erhebliche Auswirkungen auf die Abfrageleistung haben, auch wenn die Indexierung einen gewissen Einfluss auf die Gesamtleistung der Map hat. Ohne Indexierung der an Abfragen beteiligten Objektattribute führt die Abfragesteuerkomponente eine Tabellensuche für jedes Attribut durch. Die Tabellensuche ist die kostenintensivste Operation während eines Abfragelaufs. Die Indexierung der an Abfragen beteiligten Objektattribute ermöglicht der Abfragesteuerkomponente, unnötige Tabellensuchen zu vermeiden, was die Gesamtabfrageleistung verbessert. Wenn die Anwendung so konzipiert ist, dass sie häufig Abfragen in Maps durchführt, in denen hauptsächlich Leseoperationen durchgeführt werden, konfigurieren Sie Indizes für die an der Abfrage beteiligten Objektattribute. Wenn die Map hauptsächlich aktualisiert wird, müssen Sie einen Kompromiss zwischen Verbesserung der Abfrageleistung und Indexierungsaufwand finden. Weitere Informationen hierzu finden Sie im Abschnitt „Indexierung“ auf Seite 121.

Wenn POJOs (Plain Old Java Objects) in einer Map gespeichert werden, kann durch eine ordnungsgemäße Indexierung eine Java-Reflexion vermieden werden. Im folgenden Beispiel ersetzt die Abfrage die WHERE-Klausel mit Bereichssuche, wenn für das Feld "budget" ein Index erstellt wurde. Andernfalls scannt die Abfrage die gesamte Map und wertet die WHERE-Klausel aus, indem sie zuerst das Budget durch Java-Reflexion abrufen und anschließend das Budget mit dem Wert 50000 vergleicht:

```
SELECT d FROM DeptBean d WHERE d.budget=50000
```

Im Abschnitt „Abfrageplan“ auf Seite 110 wird ausführlich beschrieben, wie Sie einzelne Abfragen am besten optimieren und wie sich verschiedene Syntax, Objektmodelle und Indizes auf die Abfrageleistung auswirken können.

## Seitenaufteilung verwenden

In Client/Server-Umgebungen überträgt die Abfragesteuerkomponente die vollständige Ergebnis-Map an den Client. Die zurückgegebenen Daten müssen in angemessene Blöcke unterteilt werden. Die Schnittstellen "EntityManager Query" und "ObjectMap ObjectQuery" unterstützen beide die Methoden "setFirstResult" und "setMaxResults", mit denen der Abfrage ermöglicht wird, einen Teil der Ergebnisse zurückzugeben.

## Primitive Werte an Stelle von Entitäten zurückgeben

Mit der API "EntityManager Query" werden Entitäten als Abfrageparameter zurückgegeben. Die Abfragesteuerkomponente gibt die Schlüssel für diese Entitäten an den Client zurück. Wenn der Client mit dem Iterator aus der Methode "getResultIterator" über diese Entitäten iteriert, wird jede Entität automatisch dekomprimiert und so verwaltet, als wäre sie mit der Methode "find" der Schnittstelle "EntityManager" erstellt worden. Der vollständige Entitäts-Graph wird aus der Entitäts-ObjectMap im Client erstellt. Die Wertattribute der Entität und alle zugehörigen Entitäten werden aufgelöst.

Um die Erstellung dieses kostenintensiven Graphen zu vermeiden, ändern Sie die Abfrage so, dass sie die einzelnen Attribute mit Pfadnavigation zurückgibt.

Beispiel:

```
// Gibt eine Entität zurück.  
SELECT p FROM Person p  
// Gibt die Attribute SELECT p.name, p.address.street, p.address.city, p.gender FROM Person p zurück.
```

## Abfrageoptimierung mit Indizes

Durch die ordnungsgemäße Definition und Verwendung von Indizes kann die Abfrageleistung erheblich verbessert werden.

eXtreme-Scale-Abfragen können integrierte HashIndex-Plug-ins verwenden, um die Leistung von Abfragen zu verbessern. Indizes können für Entitäts- oder Objektattribute definiert werden. Die Abfragesteuerkomponente verwendet automatisch die definierten Indizes, wenn in der WHERE-Klausel eine der folgenden Zeichenfolgen verwendet wird:

- ein Vergleichsausdruck mit den folgenden Operatoren: =, <, >, <= oder >= (jeder Vergleichsausdruck mit Ausnahme von ungleich (<>),
- ein BETWEEN-Ausdruck,
- Konstanten oder einfache Bedingungen als Operanden des Ausdrucks.

### Voraussetzungen

Indizes müssen die folgenden Voraussetzungen erfüllen, wenn sie in einer Abfrage verwendet werden:

- Alle Indizes müssen das integrierte HashIndex-Plug-in verwenden.
- Alle Indizes müssen statisch definiert sein. Dynamische Indizes werden nicht unterstützt.
- Die Annotation "@Index" kann verwendet werden, um statische HashIndex-Plug-ins automatisch zu erstellen.
- Für alle Einzelattributindizes muss die Eigenschaft "RangeIndex" auf "true" gesetzt sein.
- Für alle zusammengesetzten Indizes muss die Eigenschaft "RangeIndex" auf "false" gesetzt sein.
- Für alle Assoziations- oder Beziehungsindizes muss die Eigenschaft "RangeIndex" auf "false" gesetzt sein.

Eine effizientere Methode für die Suche zwischengespeicherter Objekte ist im Abschnitt „Zusammengesetzter Hash-Index“ auf Seite 127 beschrieben.

## Hinweise zur Auswahl eines Index

Ein Index kann manuell über die Methode "setHint" in den Schnittstellen "Query" und "ObjectQuery" mit der Konstanten HINT\_USEINDEX ausgewählt werden. Dies kann hilfreich sein, wenn eine Abfrage für die Verwendung des Index mit der besten Leistung optimiert wird.

## Abfragebeispiele, in denen Attributindizes verwendet werden

In den folgenden Beispielen werden einfache Bedingungen verwendet: e.empid, e.name, e.salary, d.name, d.budget und e.isManager. In den Beispielen wird davon ausgegangen, dass Indizes über die Felder "name", "salary" und "budget" einer Entität oder eines Wertobjekts definiert werden. Das Feld "empid" ist ein Primärschlüssel, und für "isManager" ist kein Index definiert.

Die folgende Abfrage verwendet beide Indizes über die Felder "name" und "salary". Sie gibt alle Mitarbeiter (employees) zurück, deren Name dem Wert des ersten Parameters oder deren Gehalt (salary) dem Wert des zweiten Parameters entspricht:

```
SELECT e FROM EmpBean e where e.name=?1 or e.salary=?2
```

Die folgende Abfrage verwendet beide Indizes über die Felder "name" und "budget". Die Abfrage gibt alle Abteilungen (departments) mit dem Namen 'DEV' und einem Budget größer als 2000 zurück.

```
SELECT d FROM DeptBean d where d.name='DEV' and d.budget>2000
```

Die folgende Abfrage gibt alle Mitarbeiter mit einem Gehalt größer als 3000 zurück, deren isManager-Flag-Wert dem Wert des Parameters entspricht. Die Abfrage verwendet den Index, der über das Feld "salary" definiert wurde, und führt eine weitere Filterung durch, indem sie den folgenden Vergleichsausdruck auswertet: e.isManager=?1.

```
SELECT e FROM EmpBean e where e.salary>3000 and e.isManager=?1
```

Die folgende Abfrage sucht alle Mitarbeiter, deren Gehalt höher ist als der erste Parameter, bzw. alle Mitarbeiter, die Manager sind. Obwohl für das Feld "salary" ein Index definiert ist, scannt die Abfrage den integrierten Index, der über die Primärschlüssel des Felds "EmpBean" erstellt wurde, und wertet den folgenden Ausdruck aus: e.salary>?1 or e.isManager=TRUE.

```
SELECT e FROM EmpBean e WHERE e.salary>?1 or e.isManager=TRUE
```

Die folgende Abfrage gibt Mitarbeiter zurück, deren Name den Buchstaben a enthält. Obwohl für das Feld "name" ein Index definiert ist, verwendet die Abfrage den Index nicht, weil das Feld "name" im LIKE-Ausdruck verwendet wird.

```
SELECT e FROM EmpBean e WHERE e.name LIKE '%a%'
```

Die folgende Abfrage sucht alle Mitarbeiter, deren Name nicht "Smith" ist. Obwohl für das Feld "name" ein Index definiert ist, verwendet die Abfrage den Index nicht, weil die Abfrage den Vergleichsoperator ungleich (<>) verwendet.

```
SELECT e FROM EmpBean e where e.name<>'Smith'
```

Die folgende Abfrage sucht alle Abteilungen, deren Budget kleiner ist als der Wert des Parameters und deren Mitarbeitergehälter größer als 3000 sind. Die Abfrage verwendet einen Index für das Gehalt (salary), aber keinen Index für das Budget, weil dept.budget keine einfache Bedingung ist. Die dept-Objekte werden aus der Sammlung e abgeleitet. Sie müssen den Budgetindex nicht verwenden, um dept-Objekte zu suchen.

```
SELECT dept from EmpBean e, in (e.dept) dept where e.salary>3000 and
dept.budget<?
```

Die folgende Abfrage sucht alle Mitarbeiter, deren Gehälter größer sind als die Gehälter der Mitarbeiter mit empid 1, 2 und 3. Der Index für das Feld "salary" wird nicht verwendet, weil der Vergleich eine Unterabfrage enthält. Der empid-Wert ist jedoch ein Primärschlüssel und wird für eine eindeutige Indexsuche verwendet, weil für alle Primärschlüssel ein integrierter Index definiert ist.

```
SELECT e FROM EmpBean e WHERE e.salary > ALL (SELECT e1.salary FROM
EmpBean e1 WHERE e1.empid=1 or e1.empid =2 or e1.empid=99)
```

Um zu prüfen, ob der Index von der Abfrage verwendet wird, können Sie den Abschnitt „Abfrageplan“ auf Seite 110 verwenden. Im Folgenden sehen Sie einen Beispielabfrageplan für die vorherige Abfrage:

```
for q2 in EmpBean ObjectMap using INDEX SCAN
  filter ( q2.salary >ALL temp collection defined as
    IteratorUnionIndex of
      for q3 in EmpBean ObjectMap using UNIQUE INDEX key=(1)
      )
      for q3 in EmpBean ObjectMap using UNIQUE INDEX key=(2)
      )
      for q3 in EmpBean ObjectMap using UNIQUE INDEX key=(99)
      )
  returning new Tuple( q3.salary )
returning new Tuple( q2 )

for q2 in EmpBean ObjectMap using RANGE INDEX on salary with range(3000,)
  for q3 in q2.dept
    filter ( q3.budget < ?1 )
  returning new Tuple( q3 )
```

## Attribute indexieren

Indizes können über einen Einzelattributtyp mit den zuvor definierten Einschränkungen definiert werden.

### Entitätsindizes mit @Index definieren

Wenn Sie einen Index für eine Entität definieren möchten, definieren Sie einfach eine Annotation:

#### Entitäten mit Annotationen

```
@Entity
public class Employee {
  @Id int empid;
  @Index String name
  @Index double salary
  @ManyToOne Department dept;
}
```

```

@Entity
public class Department {
    @Id int deptid;
    @Index String name;
    @Index double budget;
    boolean isManager;
    @OneToMany Collection<Employee> employees;
}

```

## Mit XML

Indizes können auch mit XML definiert werden:

### Entitäten ohne Annotationen

```

public class Employee {
    int empid;
    String name;
    double salary;
    Department dept;
}

public class Department {
    int deptid;
    String name;
    double budget;
    boolean isManager;
    Collection employees;
}

```

### ObjectGrid-XML mit Attributindizes

```

<?xml version="1.0" encoding="UTF-8"?>
<objectGridConfig xmlns:xsi="http://www.w3.org/2001/XMLSchema-instance"
xsi:schemaLocation="http://ibm.com/ws/objectgrid/config ../objectGrid.xsd"
xmlns="http://ibm.com/ws/objectgrid/config">
  <objectGrids>
    <objectGrid name="DepartmentGrid" entityMetadataXMLFile="entity.xml">
      <backingMap name="Employee" pluginCollectionRef="Emp"/>
      <backingMap name="Department" pluginCollectionRef="Dept"/>
    </objectGrid>
  </objectGrids>
  <backingMapPluginCollections>
    <backingMapPluginCollection id="Emp">
      <bean id="MapIndexPlugin" className="com.ibm.websphere.objectgrid.plugins.index.HashIndex">
        <property name="Name" type="java.lang.String" value="Employee.name"/>
        <property name="AttributeName" type="java.lang.String" value="name"/>
        <property name="RangeIndex" type="boolean" value="true"
description="Ranges are must be set to true for attributes." />
      </bean>
      <bean id="MapIndexPlugin" className="com.ibm.websphere.objectgrid.plugins.index.HashIndex">
        <property name="Name" type="java.lang.String" value="Employee.salary"/>
        <property name="AttributeName" type="java.lang.String" value="salary"/>
        <property name="RangeIndex" type="boolean" value="true"
description="Ranges are must be set to true for attributes." />
      </bean>
    </backingMapPluginCollection>
    <backingMapPluginCollection id="Dept">
      <bean id="MapIndexPlugin" className="com.ibm.websphere.objectgrid.plugins.index.HashIndex">
        <property name="Name" type="java.lang.String" value="Department.name"/>
        <property name="AttributeName" type="java.lang.String" value="name"/>
        <property name="RangeIndex" type="boolean" value="true"
description="Ranges are must be set to true for attributes." />
      </bean>
      <bean id="MapIndexPlugin" className="com.ibm.websphere.objectgrid.plugins.index.HashIndex">
        <property name="Name" type="java.lang.String" value="Department.budget"/>
        <property name="AttributeName" type="java.lang.String" value="budget"/>
        <property name="RangeIndex" type="boolean" value="true"
description="Ranges are must be set to true for attributes." />
      </bean>
    </backingMapPluginCollection>
  </backingMapPluginCollections>
</objectGridConfig>

```

### Entitäts-XML

```

<?xml version="1.0" encoding="UTF-8"?>
<entity-mappings xmlns="http://ibm.com/ws/projector/config/emd"
xmlns:xsi="http://www.w3.org/2001/XMLSchema-instance"
xsi:schemaLocation="http://ibm.com/ws/projector/config/emd ../emd.xsd">

```

```

<description>Department entities</description>
<entity class-name="acme.Employee" name="Employee" access="FIELD">
<attributes>
<id name="empid" />
<basic name="name" />
<basic name="salary" />
<many-to-one name="department"
target-entity="acme.Department"
fetch="EAGER">
<cascade><cascade-persist/></cascade>
</many-to-one>
</attributes>
</entity>
<entity class-name="acme.Department" name="Department" access="FIELD">
<attributes>
<id name="deptid" />
<basic name="name" />
<basic name="budget" />
<basic name="isManager" />
<one-to-many name="employees"
target-entity="acme.Employee"
fetch="LAZY" mapped-by="parentNode">
<cascade><cascade-persist/></cascade>
</one-to-many>
</attributes>
</entity>
</entity-mappings>

```

## Indizes mit XML für Objekte definieren, die keine Entitäten sind

Indizes für Objekte, die keine Entitäten sind, werden in XML definiert. Das MapIndexPlugin für Maps, die keine Entitäts-Maps sind, wird auf dieselbe Weise wie für Entitäts-Maps erstellt.

### Java-Bean

```

public class Employee {
    int empid;
    String name;
    double salary;
    Department dept;

    public class Department {
        int deptid;
        String name;
        double budget;
        boolean isManager;
        Collection employees;
    }
}

```

### ObjectGrid-XML mit Attributindizes

```

<?xml version="1.0" encoding="UTF-8"?>
<objectGridConfig xmlns:xsi="http://www.w3.org/2001/XMLSchema-instance"
xsi:schemaLocation="http://ibm.com/ws/objectgrid/config ../objectGrid.xsd"
xmlns="http://ibm.com/ws/objectgrid/config">
<objectGrids>
<objectGrid name="DepartmentGrid">
<backingMap name="Employee" pluginCollectionRef="Emp"/>
<backingMap name="Department" pluginCollectionRef="Dept"/>
<querySchema>
<mapSchemas>
<mapSchema mapName="Employee" valueClass="acme.Employee"
primaryKeyField="empid" />
<mapSchema mapName="Department" valueClass="acme.Department"
primaryKeyField="deptid" />
</mapSchemas>
<relationships>
<relationship source="acme.Employee"
target="acme.Department"
relationField="dept" invRelationField="employees" />
</relationships>
</querySchema>
</objectGrid>
</objectGrids>
<backingMapPluginCollections>
<backingMapPluginCollection id="Emp">
<bean id="MapIndexPlugin" className="com.ibm.websphere.objectgrid.plugins.index.HashIndex">
<property name="Name" type="java.lang.String" value="Employee.name"/>
<property name="AttributeName" type="java.lang.String" value="name"/>
<property name="RangeIndex" type="boolean" value="true"
description="Ranges are must be set to true for attributes." />
</bean>
<bean id="MapIndexPlugin" className="com.ibm.websphere.objectgrid.plugins.index.HashIndex">

```



```

<property name="Name" type="java.lang.String" value="Employee.salary"/>
<property name="AttributeName" type="java.lang.String" value="salary"/>
<property name="RangeIndex" type="boolean" value="true"
description="Ranges are must be set to true for attributes." />
</bean>
</backingMapPluginCollection>
<backingMapPluginCollection id="Dept">
<bean id="MapIndexPlugin" className="com.ibm.websphere.objectgrid.plugins.index.HashIndex">
<property name="Name" type="java.lang.String" value="Department.name"/>
<property name="AttributeName" type="java.lang.String" value="name"/>
<property name="RangeIndex" type="boolean" value="true"
description="Ranges are must be set to true for attributes." />
</bean>
<bean id="MapIndexPlugin" className="com.ibm.websphere.objectgrid.plugins.index.HashIndex">
<property name="Name" type="java.lang.String" value="Department.budget"/>
<property name="AttributeName" type="java.lang.String" value="budget"/>
<property name="RangeIndex" type="boolean" value="true"
description="Ranges are must be set to true for attributes." />
</bean>
</backingMapPluginCollection>
</backingMapPluginCollections>
</objectGridConfig>

```

## Beziehungen indexieren

WebSphere eXtreme Scale speichert die Fremdschlüssel für zusammengehörige Entitäten im übergeordneten Objekt. Für Entitäten werden die Schlüssel im zugrunde liegenden Tupel gespeichert. Für Objekte, die keine Entitäten sind, werden die Schlüssel explizit im übergeordneten Objekt gespeichert.

Das Hinzufügen eines Index für ein Beziehungsattribut kann Abfragen beschleunigen, die zyklische Referenzen oder die Abfragefilter IS NULL, IS EMPTY, SIZE und MEMBER OF verwenden. Sowohl Assoziationen mit einem Wert als auch Assoziationen mit mehreren Werten können die Annotation `@Index` oder eine HashIndex-Plug-in-Konfiguration in einer ObjectGrid-XML-Deskriptordatei enthalten.

### Entitätsbeziehungsindizes mit `@Index` definieren

Im folgenden Beispiel werden Entitäten mit `@Index`-Annotationen definiert:

#### Entität mit Annotation

```

@Entity
public class Node {
    @ManyToOne @Index
    Node parentNode;

    @OneToMany @Index
    List<Node> childrenNodes = new ArrayList();

    @OneToMany @Index
    List<BusinessUnitType> businessUnitTypes = new ArrayList();
}

```

### Entitätsbeziehungsindizes mit XML definieren

Im folgenden Beispiel werden dieselben Entitäten und Indizes mit XML mit HashIndex-Plug-ins definiert:

#### Entität ohne Annotationen

```

public class Node {
    int nodeId;
    Node parentNode;
    List<Node> childrenNodes = new ArrayList();
    List<BusinessUnitType> businessUnitTypes = new ArrayList();
}

```

#### ObjectGrid XML

```
<?xml version="1.0" encoding="UTF-8"?>
```

```

<objectGridConfig xmlns:xsi="http://www.w3.org/2001/XMLSchema-instance"
xsi:schemaLocation="http://ibm.com/ws/objectgrid/config ../objectGrid.xsd"
xmlns="http://ibm.com/ws/objectgrid/config">
<objectGrids>
<objectGrid name="ObjectGrid_Entity" entityMetadataXMLFile="entity.xml">
<backingMap name="Node" pluginCollectionRef="Node"/>
<backingMap name="BusinessUnitType" pluginCollectionRef="BusinessUnitType"/>
</objectGrid>
</objectGrids>
<backingMapPluginCollections>
<backingMapPluginCollection id="Node">
<bean id="MapIndexPlugin" className="com.ibm.websphere.objectgrid.plugins.index.HashIndex">
<property name="Name" type="java.lang.String" value="parentNode"/>
<property name="AttributeName" type="java.lang.String" value="parentNode"/>
<property name="RangeIndex" type="boolean" value="false"
description="Ranges are not supported for association indexes." /> </bean>
<bean id="MapIndexPlugin" className="com.ibm.websphere.objectgrid.plugins.index.HashIndex">
<property name="Name" type="java.lang.String" value="businessUnitType"/>
<property name="AttributeName" type="java.lang.String" value="businessUnitTypes"/>
</bean>
</backingMapPluginCollection>
</backingMapPluginCollections>
</objectGridConfig>

```

#### Entitäts-XML

```

<?xml version="1.0" encoding="UTF-8"?>
<entity-mappings xmlns="http://ibm.com/ws/projector/config/emd"
xmlns:xsi="http://www.w3.org/2001/XMLSchema-instance"
xsi:schemaLocation="http://ibm.com/ws/projector/config/emd ../emd.xsd">
<description>My entities</description>
<entity class-name="acme.Node" name="Account" access="FIELD">
<attributes>
<id name="nodeId" />
<one-to-many name="childrenNodes"
target-entity="acme.Node"
fetch="EAGER" mapped-by="parentNode">
<cascade><cascade-all/></cascade>
</one-to-many>
<many-to-one name="parentNodes"
target-entity="acme.Node"
fetch="LAZY" mapped-by="childrenNodes">
<cascade><cascade-none/></cascade>
</many-to-one>
<many-to-one name="businessUnitTypes"
target-entity="acme.BusinessUnitType"
fetch="EAGER">
<cascade><cascade-persist/></cascade>
</many-to-one>
</attributes>
</entity>
<entity class-name="acme.BusinessUnitType" name="BusinessUnitType" access="FIELD">
<attributes>
<id name="buId" />
<basic name="TypeDescription" />
</attributes>
</entity>
</entity-mappings>

```

Mit den zuvor definierten Indizes werden die folgenden Entitätsabfragebeispiele optimiert:

```

SELECT n FROM Node n WHERE n.parentNode is null
SELECT n FROM Node n WHERE n.businessUnitTypes is EMPTY
SELECT n FROM Node n WHERE size(n.businessUnitTypes)>=10
SELECT n FROM BusinessUnitType b, Node n WHERE b member of n.businessUnitTypes and b.name='TELECOM'

```

#### Beziehungsindizes für Objekte definieren, die keine Entitäten sind

Im folgenden Beispiel wird ein HashIndex-Plug-in für Maps, die keine Entitäts-Maps sind, in einer ObjectGrid-XML-Deskriptordatei definiert:

```

<?xml version="1.0" encoding="UTF-8"?>
<objectGridConfig xmlns:xsi="http://www.w3.org/2001/XMLSchema-instance"
xsi:schemaLocation="http://ibm.com/ws/objectgrid/config ../objectGrid.xsd"
xmlns="http://ibm.com/ws/objectgrid/config">
<objectGrids>

```

```

<objectGrid name="ObjectGrid_POJO">
  <backingMap name="Node" pluginCollectionRef="Node"/>
  <backingMap name="BusinessUnitType" pluginCollectionRef="BusinessUnitType"/>
  <querySchema>
    <mapSchemas>
      <mapSchema mapName="Node"
        valueClass="com.ibm.websphere.objectgrid.samples.entity.Node"
        primaryKeyField="id" />
      <mapSchema mapName="BusinessUnitType"
        valueClass="com.ibm.websphere.objectgrid.samples.entity.BusinessUnitType"
        primaryKeyField="id" />
    </mapSchemas>
    <relationships>
      <relationship source="com.ibm.websphere.objectgrid.samples.entity.Node"
        target="com.ibm.websphere.objectgrid.samples.entity.Node"
        relationField="parentNodeId" invRelationField="childrenNodeIds" />
      <relationship source="com.ibm.websphere.objectgrid.samples.entity.Node"
        target="com.ibm.websphere.objectgrid.samples.entity.BusinessUnitType"
        relationField="businessUnitTypeKeys" invRelationField="" />
    </relationships>
  </querySchema>
</objectGrid>
</objectGrids>
<backingMapPluginCollections>
  <backingMapPluginCollection id="Node">
    <bean id="MapIndexPlugin" className="com.ibm.websphere.objectgrid.plugins.index.HashIndex">
      <property name="Name" type="java.lang.String" value="parentNode"/>
    </bean>
    <property name="Name" type="java.lang.String" value="parentNodeId"/>
    <property name="AttributeName" type="java.lang.String" value="parentNodeId"/>
    <property name="RangeIndex" type="boolean" value="false"
      description="Ranges are not supported for association indexes." />
  </backingMapPluginCollection>
  <backingMapPluginCollection id="BusinessUnitType">
    <bean id="MapIndexPlugin" className="com.ibm.websphere.objectgrid.plugins.index.HashIndex">
      <property name="Name" type="java.lang.String" value="businessUnitType"/>
      <property name="AttributeName" type="java.lang.String" value="businessUnitTypeKeys"/>
    </bean>
    <property name="RangeIndex" type="boolean" value="false"
      description="Ranges are not supported for association indexes." />
  </backingMapPluginCollection>
</backingMapPluginCollections>
</objectGridConfig>

```

Mit den zuvor gezeigten Indexkonfigurationen werden die folgenden Objektbeispiele optimiert:

```

SELECT n FROM Node n WHERE n.parentNodeId is null
SELECT n FROM Node n WHERE n.businessUnitTypeKeys is EMPTY
SELECT n FROM Node n WHERE size(n.businessUnitTypeKeys)>=10
SELECT n FROM BusinessUnitType b, Node n WHERE
  b member of n.businessUnitTypeKeys and b.name='TELECOM'

```

## Abfrageplan

Alle Abfragen von eXtreme Scale haben einen Abfrageplan. Der Plan beschreibt, wie die Abfragesteuerkomponente mit ObjectMaps und Indizes interagiert. Zeigen Sie den Abfrageplan an, um festzustellen, ob die Abfragezeichenfolge oder Indizes ordnungsgemäß verwendet werden. Der Abfrageplan kann auch verwendet werden, um die Unterschiede zu erkennen, die geringfügige Änderungen einer Abfragezeichenfolge in der Art und Weise, wie eXtreme Scale eine Abfrage ausführt, bewirken.

Der Abfrageplan kann auf die folgenden beiden Arten angezeigt werden:

- Mit der Methode "getplan" der APIs "EntityManager Query" und "ObjectQuery"
- Diagnose-Trace für ObjectGrid

### Methode "getPlan"

Die Methode "getPlan" in den Schnittstellen "ObjectQuery" und "Query" gibt eine Zeichenfolge zurück, die den Abfrageplan beschreibt. Diese Zeichenfolge kann in

der Standardausgabe oder in einem Protokoll angezeigt werden. Anmerkung: In einer verteilten Umgebung wird die Methode "getPlan" nicht für den Server ausgeführt und gibt keine definierten Indizes zurück. Zum Anzeigen des Plans auf dem Server verwenden Sie einen Agenten.

## Trace für Abfrageplan

Der Abfrageplan kann über einen ObjectGrid-Trace angezeigt werden. Zum Aktivieren des Trace für den Abfrageplan verwenden Sie die folgende Trace-Spezifikation:

```
QueryEnginePlan=debug=enabled
```

Einzelheiten zum Aktivieren des Trace und zum Auffinden der Trace-Protokoll-dateien finden Sie im Abschnitt „Protokolle und Trace“ auf Seite 291.

## Beispiele für Abfragepläne

Im Abfrageplan wird das Wort "for" verwendet, um anzuzeigen, dass die Abfrage durch eine ObjectMap-Sammlung oder durch eine abgeleitete Sammlung, wie z. B. "q2.getEmps(), q2.dept" oder eine temporäre Sammlung, die von einer inneren Schleife zurückgegeben wird, iteriert. Wenn die Sammlung aus einer ObjectMap stammt, zeigt der Abfrageplan an, ob eine sequenzielle Suche (angegeben mit INDEX SCAN), einen eindeutigen Index oder einen nicht eindeutigen Index verwendet wird. Der Abfrageplan verwendet eine Filterzeichenfolge, um die Bedingungsausdrücke aufzulisten, die auf eine Sammlung angewendet werden.

Ein kartesisches Produkt wird in der Objektabfrage gewöhnlich nicht verwendet. Die folgende Abfrage scannt die gesamte Map "EmpBean" in der äußeren Schleife und die gesamte Map "DeptBean" in der inneren Schleife:

```
SELECT e, d FROM EmpBean e, DeptBean d
```

Plan trace:

```
for q2 in EmpBean ObjectMap using INDEX SCAN
  for q3 in DeptBean ObjectMap using INDEX SCAN
    returning new Tuple( q2, q3 )
```

Die folgende Abfrage ruft alle Mitarbeiternamen (employee) aus einer bestimmten Abteilung (department) ab, indem sie die Map "EmpBean" sequenziell durchsucht, um ein employee-Objekt abzurufen. Über das employee-Objekt navigiert die Abfrage zum zugehörigen department-Objekt und wendet den Filter "d.no=1" an. In diesem Beispiel hat jeder Mitarbeiter nur eine Referenz auf ein department-Objekt, so dass die innere Schleife nur ein einziges Mal ausgeführt wird:

```
SELECT e.name FROM EmpBean e JOIN e.dept d WHERE d.no=1
```

Plan trace:

```
for q2 in EmpBean ObjectMap using INDEX SCAN
  for q3 in q2.dept
    filter ( q3.getNo() = 1 )
    returning new Tuple( q2.name )
```

Die folgende Abfrage entspricht der vorherigen Abfrage. Sie ist jedoch schneller, weil sie das Ergebnis über die Verwendung eines eindeutigen Index, der für die Primärschlüsselfeldnummer "DeptBean" auf ein einziges department-Objekt eingrenzt. Über das department-Objekt navigiert die Abfrage zu den zugehörigen employee-Objekten, um die Namen abzurufen:

```
SELECT e.name FROM DeptBean d JOIN d.emps e WHERE d.no=1
```

Plan trace:

```
for q2 in DeptBean ObjectMap using UNIQUE INDEX key=(1)
  for q3 in q2.getEmps()
    returning new Tuple( q3.name )
```

Die folgende Abfrage sucht alle Mitarbeiter (employee), die für die Entwicklung (development) oder den Vertrieb (sales) arbeiten. Die Abfrage scannt die gesamte Map "EmpBean" und führt weitere Filterungen durch, indem Sie die Ausdrücke "d.name = 'Sales'" und "d.name='Dev'" auswertet:

```
SELECT e FROM EmpBean e, in (e.dept) d WHERE d.name = 'Sales'
or d.name='Dev'
```

Plan trace:

```
for q2 in EmpBean ObjectMap using INDEX SCAN
  for q3 in q2.dept
    filter (( q3.getName() = Sales ) OR ( q3.getName() = Dev ) )
    returning new Tuple( q2 )
```

Die folgende Abfrage entspricht der vorherigen Abfrage, führt aber einen anderen Abfrageplan aus und verwendet den über das Feld "name" erstellten Bereichsindex. Im Allgemeinen ist diese Abfrage schneller, weil der Index über das Feld "name" verwendet wird, um die department-Objekte einzugrenzen, was schnell ist, wenn es nur wenige Entwicklungs- oder Vertriebsabteilungen gibt.

```
SELECT e FROM DeptBean d, in(d.emps) e WHERE d.name='Dev' or d.name='Sales'
```

Plan trace:

IteratorUnionIndex of

```
for q2 in DeptBean ObjectMap using INDEX on name = (Dev)
  for q3 in q2.getEmps()

for q2 in DeptBean ObjectMap using INDEX on name = (Sales)
  for q3 in q2.getEmps()
```

Die folgende Abfrage sucht Abteilungen, die keine Mitarbeiter haben:

```
SELECT d FROM DeptBean d WHERE NOT EXISTS(select e from d.emps e)
```

Plan trace:

```
for q2 in DeptBean ObjectMap using INDEX SCAN
  filter ( NOT EXISTS ( correlated collection defined as

    for q3 in q2.getEmps()
      returning new Tuple( q3 )

    returning new Tuple( q2 )
```

Die folgende Abfrage entspricht der vorherigen Abfrage, verwendet aber die Skalarfunktion SIZE. Diese Abfrage hat eine ähnliche Leistung, ist aber einfacher zu schreiben:

```
SELECT d FROM DeptBean d WHERE SIZE(d.emps)=0
for q2 in DeptBean ObjectMap using INDEX SCAN
  filter (SIZE( q2.getEmps()) = 0 )
  returning new Tuple( q2 )
```

Das folgende Beispiel zeigt eine weitere einfachere Methode, dieselbe Abfrage wie zuvor mit ähnlicher Leistung zu schreiben:

```
SELECT d FROM DeptBean d WHERE d.emps IS EMPTY
```

Plan trace:

```
for q2 in DeptBean ObjectMap using INDEX SCAN
  filter ( q2.getEmps() IS EMPTY )
  returning new Tuple( q2 )
```

Die folgende Abfrage sucht alle Mitarbeiter mit einer Privatadresse, die mindestens einer der Adressen des Mitarbeiters entspricht, dessen Name mit dem Wert des Parameters übereinstimmt. Die innere Schleife ist nicht von der äußeren Schleife abhängig. Die Abfrage führt die innere Schleife ein einziges Mal aus:

```
SELECT e FROM EmpBean e WHERE e.home = any (SELECT e1.home FROM EmpBean e1
WHERE e1.name=?1)
for q2 in EmpBean ObjectMap using INDEX SCAN
  filter ( q2.home =ANY      temp collection defined as
          )
    for q3 in EmpBean ObjectMap using INDEX on name = ( ?1)
      returning new Tuple( q3.home      )
  )
  returning new Tuple( q2 )
```

Die folgende Abfrage entspricht der vorherigen Abfrage, hat aber eine Unterabfrage mit Korrelationsbezug. Außerdem wird die innere Schleife mehrfach ausgeführt:

```
SELECT e FROM EmpBean e WHERE EXISTS(SELECT e1 FROM EmpBean e1 WHERE
e.home=e1.home and e1.name=?1)
```

Plan trace:

```
for q2 in EmpBean ObjectMap using INDEX SCAN
  filter ( EXISTS (      correlated collection defined as
          )
    for q3 in EmpBean ObjectMap using INDEX on name = (?1)
      filter ( q2.home = q3.home )
      returning new Tuple( q3      )
  )
  returning new Tuple( q2 )
```

---

## Kapitel 9. Fehlerbehebung

Sie können Protokolle und Traces, Nachrichten und Releaseinformationen verwenden, um Fehler in der Konfiguration Ihres speicherinternen eXtreme-Scale-Daten-Grids zu beheben.

### Zugehörige Konzepte

Fehler in der XML-Konfiguration beheben

---

## Protokolle und Trace

Sie können Protokolle und Trace verwenden, um Ihre Umgebung zu überwachen und Fehler zu beheben. Protokolle werden je nach Konfiguration an unterschiedlichen Positionen gespeichert. Möglicherweise müssen Sie einen Trace für einen Server bereitstellen, wenn Sie mit der IBM Unterstützungsfunktion zusammenarbeiten.

### Protokolle mit WebSphere Application Server

Weitere Informationen finden Sie im Information Center von WebSphere Application Server.

### Protokolle mit WebSphere eXtreme Scale in einer eigenständigen Umgebung

Bei eigenständigen Katalog- und Containerservern legen Sie die Position der Protokolle und alle Trace-Spezifikationen fest. Die Katalogserverprotokolle werden dort gespeichert, wo Sie den Befehl zum Starten des Servers ausführen.

#### Protokollposition für Containerserver festlegen

Standardmäßig werden die Protokolle für einen Server in dem Verzeichnis gespeichert, in dem der Befehl zum Starten des Servers ausgeführt wird. Wenn Sie die Server im Verzeichnis `<Ausgangsverzeichnis_von_eXtreme_Scale>/bin` starten, werden die Protokoll- und Trace-Dateien in den Verzeichnissen `logs/<Servername>` des Verzeichnisses `bin` gespeichert. Wenn Sie eine andere Position für die Protokolle eines Containerservers festlegen möchten, erstellen Sie eine Eigenschaftendatei, z. B. `server.properties`, mit dem folgenden Inhalt:

```
workingDirectory=<Verzeichnis>
traceSpec=
systemStreamToFileEnabled=true
```

Die Eigenschaft "workingDirectory" ist das Stammverzeichnis für die Protokolle und die optionale Trace-Datei. WebSphere eXtreme Scale erstellt ein Verzeichnis mit dem Namen des Containerservers mit einer Datei `SystemOut.log`, einer Datei `SystemErr.log` und einer Trace-Datei, falls die Trace-Erstellung mit der Option "traceSpec" aktiviert wurde. Wenn eine Eigenschaftendatei während des Containerstarts verwendet werden soll, verwenden Sie die Option `-serverProps`, und geben Sie die Position der Servereigenschaftendatei an.

Häufige Informationsnachrichten, die in der Datei `SystemOut.log` aufgezeichnet werden, sind Bestätigungsnachrichten für den Start. Weitere Informationen zu bestimmten Nachrichten finden Sie im Abschnitt „Nachrichten“ auf Seite 295.

## Trace-Erstellung mit WebSphere Application Server

Weitere Informationen finden Sie im Information Center von WebSphere Application Server.

### Trace-Erstellung für den Katalogservice

Sie können die Trace-Erstellung für einen Katalogservice festlegen, indem Sie während des Katalogservicestarts die Parameter **-traceSpec** und **-traceFile** verwenden. Beispiel:

```
startOgServer.sh catalogServer -traceSpec
ObjectGridPlacement=all=enabled -traceFile
/home/user1/logs/trace.log
```

Wenn Sie den Katalogservice im Verzeichnis *<Ausgangsverzeichnis\_von\_eXtreme\_Scale>/bin* starten, werden die Protokolle und Trace-Dateien in einem Verzeichnis *logs/<Name\_des\_Katalogservice>* des Verzeichnisses *bin* gespeichert. Lesen Sie die Informationen zum Starten eines Katalogserviceprozesses in einer eigenständigen Umgebung im *Administratorhandbuch*.

### Trace für einen eigenständigen Containerserver erstellen

Sie können die Trace-Erstellung für einen Containerserver auf zwei Arten aktivieren. Sie können, wie im Abschnitt zu den Protokollen erläutert, eine Servereigenschaftendatei erstellen, oder Sie können die Trace-Erstellung beim Start über die Befehlszeile aktivieren. Zum Aktivieren der Trace-Erstellung für den Container über eine Servereigenschaftendatei aktualisieren Sie die Eigenschaft **traceSpec** mit der erforderlichen Trace-Spezifikation. Zum Aktivieren der Trace-Erstellung für den Container über Startparameter verwenden Sie die Parameter **-traceSpec** und **-traceFile**. Beispiel:

```
startOgServer.sh c0 -objectGridFile ../xml/myObjectGrid.xml
-deploymentPolicyFile ../xml/myDepPolicy.xml -catalogServiceEndpoints
server1.rchland.ibm.com:2809 -traceSpec
ObjectGridPlacement=all=enabled -traceFile /home/user1/logs/trace.log
```

Wenn Sie den Server im Verzeichnis *<Ausgangsverzeichnis\_von\_eXtreme\_Scale>/bin* starten, werden die Protokoll- und Trace-Dateien in den Verzeichnissen *logs/<Servername>* des Verzeichnisses *bin* gespeichert.

Weitere Informationen finden Sie

### Trace-Erstellung mit der Schnittstelle "ObjectGridManager"

Eine weitere Option ist die Definition der Trace-Erstellung zur Laufzeit über eine ObjectGridManager-Schnittstelle. Die Definition der Trace-Erstellung in einer ObjectGridManager-Schnittstelle kann verwendet werden, um einen Trace für einen eXtreme-Scale-Client zu erstellen, wenn dieser eine Verbindung zu einer eXtreme-Scale-Instanz herstellt und Transaktionen festschreibt. Wenn Sie die Trace-Erstellung in einer ObjectGridManager-Schnittstelle festlegen möchten, geben Sie eine Trace-Spezifikation und ein Trace-Protokoll an.

```
ObjectGridManager manager = ObjectGridManagerFactory.getObjectGridManager();
...
manager.setTraceEnabled(true);
manager.setTraceFileName("logs/myClient.log");
manager.setTraceSpecification("ObjectGridReplication=all=enabled");
```

### Trace-Erstellung mit dem Dienstprogramm "xsadmin" aktivieren

Zum Aktivieren der Trace-Erstellung mit dem Dienstprogramm "xsadmin" verwenden Sie die Option **setTraceSpec**. Verwenden Sie das Dienstprogramm "xsadmin",



um die Trace-Erstellung für eine eigenständige Umgebung zur Laufzeit und nicht zur Startzeit zu aktivieren. Sie können die Trace-Erstellung für alle Server und Katalogservices aktivieren, oder Sie können die Server nach dem ObjectGrid-Namen usw. filtern. Wenn Sie beispielsweise den ObjectGridReplication-Trace mit Zugriff auf den Katalogserviceserver aktivieren möchten, führen Sie den folgenden Befehl aus:

```
<Ausgangsverzeichnis_von_eXtreme_Scale>/bin>xsadmin.bat -setTraceSpec "ObjectGridReplication=all=enabled"
```

Sie können die Trace-Erstellung auch inaktivieren, indem Sie die Trace-Spezifikation auf `*=all=disabled.` setzen.

Weitere Informationen finden Sie in der Beschreibung des Dienstprogramms "xsAdmin" im *Administratorhandbuch*.

## FFDC-Verzeichnis und -Dateien

FFDC-Dateien sind als Debug-Hilfe für die IBM Unterstützungsfunktion bestimmt. Diese Dateien werden möglicherweise von der IBM Unterstützungsfunktion angefordert, wenn ein Problem auftritt.

Diese Dateien befinden sich in einem Verzeichnis mit dem Namen `ffdc`. Das Verzeichnis enthält Dateien wie die folgenden:

```
server2_exception.log  
server2_20802080_07.03.05_10.52.18_0.txt
```

## Trace-Optionen

Sie können die Trace-Erstellung aktivieren, um der IBM Unterstützungsfunktion Informationen über Ihre Umgebung bereitzustellen.

### Informationen zur Trace-Erstellung

Der WebSphere eXtreme Scale-Trace ist in mehrere Komponenten unterteilt. Ähnlich wie bei der Trace-Erstellung in WebSphere Application Server können Sie die zu verwendende Trace-Stufe angeben. Zu den gebräuchlichen Trace-Stufen gehören `all`, `debug`, `entryExit` und `event`.

Im Folgenden sehen Sie ein Beispiel für eine Trace-Zeichenfolge:

```
ObjectGridComponent=level=enabled
```

Sie können Trace-Zeichenfolgen verknüpfen. Verwenden Sie das Symbol `*` (Stern), um einen Platzhalterwert anzugeben, z. B. `ObjectGrid*=all=enabled`. Wenn Sie einen Trace für die IBM Unterstützungsfunktion bereitstellen müssen, ist eine bestimmte Trace-Zeichenfolge erforderlich. So kann beispielsweise die Trace-Zeichenfolge `ObjectGridReplication=debug=enabled` angefordert werden, wenn ein Problem mit der Replikation auftritt.

### Trace-Spezifikation

#### ObjectGrid

Allgemeine Basiccachesteuerkomponente.

#### ObjectGridCatalogServer

Allgemeiner Katalogservice.

#### ObjectGridChannel

Statische Kommunikation in der Implementierungstopologie.

- ObjectGridCORBA**  
Dynamische Kommunikation in der Implementierungstopologie.
- ObjectGridDataGrid**  
Die API "AgentManager".
- ObjectGridDynaCache**  
Der dynamische Cacheprovider von WebSphere eXtreme Scale.
- ObjectGridEntityManager**  
Die API "EntityManager". Mit der Option "Projector" zu verwenden.
- ObjectGridEvictors**  
Integrierte ObjectGrid-Bereinigungsprogramme.
- ObjectGridJPA**  
JPA-Loader (Java Persistence API).
- ObjectGridJPACache**  
JPA-Cache-Plug-ins.
- ObjectGridLocking**  
Sperrenmanager für ObjectGrid-Cacheeinträge.
- ObjectGridMBean**  
Management-Beans.
- ObjectGridPlacement**  
Catalogserverservice für Shard-Verteilung.
- ObjectGridQuery**  
ObjectGrid-Abfrage.
- ObjectGridReplication**  
Replikationsservice.
- ObjectGridRouting**  
Details zum Client/Server-Routing.
- ObjectGridSecurity**  
Sicherheits-Trace.
- ObjectGridStats**  
ObjectGrid-Statistiken.
- ObjectGridStreamQuery**  
Die API "Stream Query".
- ObjectGridWriteBehind**  
ObjectGrid-Write-Behind.
- Projector**  
Die Steuerkomponente in der API "EntityManager".
- QueryEngine**  
Die Abfragesteuerkomponente für die API "Object Query" und die API "EntityManager Query".
- QueryEnginePlan**  
Diagnose des Abfrageplans.

---

## Nachrichten

Wenn Sie in einem Protokoll oder in anderen Teilen der Produktschnittstelle eine Nachricht sehen, können Sie die Nachricht mit Hilfe des Komponentenpräfix suchen, um weitere Informationen zu erhalten.

### Nachrichten suchen

Wenn Sie eine Nachricht in einem Protokoll finden, kopieren Sie die Nachrichtennummer mit ihrem Buchstabenpräfix und ihrer Nummer, und suchen Sie im Information Center danach (z. B. CW0BJ1526I). Wenn Sie nach der Nachricht suchen, können Sie eine zusätzliche Erläuterung der Nachricht und Beschreibung möglicher Maßnahmen finden, die Sie zum Beheben des Problems verwenden können.

Einen Index der Produktnachrichten finden Sie im Information Center.

---

## Releaseinformationen

Hier finden Sie Links zur Unterstützungswebsite für das Produkt, zur Produktdokumentation und zum letzten Stand der Updates, Einschränkungen und bekannten Problemen für das Produkt.

- „Zugriff auf den letzten Stand von Updates, Einschränkungen und bekannten Problemen“
- „Zugriff auf System- und Softwarevoraussetzungen“
- „Zugriff auf die Produktdokumentation “
- „Zugriff auf die Unterstützungswebsite für das Produkt “
- „Kontaktaufnahme mit der IBM Softwareunterstützung “ auf Seite 296

### Zugriff auf den letzten Stand von Updates, Einschränkungen und bekannten Problemen

Die Releaseinformationen sind auf der Produktunterstützungssite als technische Hinweise verfügbar. Eine Liste aller technischen Hinweise für WebSphere eXtreme Scale finden Sie auf der Unterstützungswebseite.

- Eine Liste der Releaseinformationen für Version 7.0 finden Sie auf der Unterstützungswebseite.
- Eine Liste der Releaseinformationen für Version 6.1 finden Sie auf der Wiki-Site zu den Releaseinformationen.

### Zugriff auf System- und Softwarevoraussetzungen

Die Hardware- und Softwarevoraussetzungen finden Sie auf den folgenden Webseiten:

- Detailed system requirements

### Zugriff auf die Produktdokumentation

Das vollständige Informationsset finden Sie auf der Bibliotheksseite.

### Zugriff auf die Unterstützungswebsite für das Produkt

Wenn Sie nach den neuesten technischen Hinweisen, Downloads und Korrekturen sowie nach Informationen zur Unterstützung suchen, rufen Sie die Unterstützungswebseite auf.

## **Kontaktaufnahme mit der IBM Softwareunterstützung**

Wenn ein Problem mit dem Produkt auftritt, versuchen Sie zuerst, die folgenden Aktionen auszuführen:

- Führen Sie die in der Produktdokumentation beschriebenen Schritte aus.
- Suchen Sie in der Onlinehilfe nach Referenzliteratur.
- Schlagen Sie die Fehlermeldungen in der Nachrichtenreferenz nach.

Sollten Sie das Problem nicht auf diese Weise lösen können, wenden Sie sich an die technische Unterstützung der IBM.

---

## Kapitel 10. Glossar

Dieses Glossar enthält Begriffe und Definitionen für WebSphere eXtreme Scale.

In diesem Glossar werden die folgenden Querverweise verwendet:

1. 'Siehe' verweist von einem Terminus zu einem bevorzugt zu verwendenden Synonym oder von einem Akronym oder einer Abkürzung zu der Definition der vollständigen Langform des Terminus.
2. 'Siehe auch' verweist auf einen zugehörigen Terminus oder auf ein Antonym.

Wenn Sie Glossare zu anderen IBM Produkten anzeigen möchten, dann rufen Sie die Website [www.ibm.com/software/globalization/terminology](http://www.ibm.com/software/globalization/terminology) auf.

### **Abfrage.**

1. Eine Anforderung von Informationen aus einer Datenbank, die auf bestimmten Bedingungen basiert, z. B. eine Anforderung einer Liste aller Kunden in einer Kundentabelle, deren Kontostand höher als 1000 Euro ist.
2. Eine wiederverwendbare Informationsanforderung über ein oder mehr Modellelemente.

**Ableitung.** In der objektorientierten Programmierung die Verbesserung oder Erweiterung einer Klasse auf der Basis einer anderen Klasse.

**Administrator.** Eine Person, die für die Ausführung von Verwaltungstasks wie beispielsweise der Vergabe von Zugriffsberechtigungen und für das Content-Management verantwortlich ist. Administratoren sind auch für die Vergabe von Berechtigungsstufen an Benutzer zuständig.

**Agent.** Ein Programm, das eine Aktion für einen Benutzer oder ein anderes Programm ohne Benutzereingriff oder nach einem regelmäßigen Zeitplan ausführt und die Ergebnisse zurück an den Benutzer oder das Programm meldet.

**Aktualisierbare Sperre.** Eine Sperre, die die Absicht aufzeigt, einen Cacheeintrag zu aktualisieren, wenn pessimistisches Sperren verwendet wird.

**Aktualisierungspaket.** Eine kumulative Sammlung von Programmkorrekturen mit neuen Funktionen. Siehe auch Fixpack, Vorläufiger Fix.

**Allgemein.** Dieser Begriff bezeichnet das Anzeigen einer Gruppe von Objekten auf abstrakter oder höherer Ebene.

**Angepasstes Installationspaket.** Ein angepasstes Installationsimage, das mindestens ein Wartungspaket, eine Konfigurationsarchivierungsdatei aus einem eigenständigen Serverprofil, mindestens eine Unternehmensarchivdatei, Scripts und sonstige Dateien enthalten kann, die Unterstützung bei der Anpassung der daraus resultierenden Installation bieten.

**Anwendung.** Computerprogramme oder Softwarekomponenten, die eine Funktion für die direkte Unterstützung eines oder mehrerer Geschäftsprozesse bereitstellen.

**Anwendungsprogrammierschnittstelle (API).** Ein Schnittstelle, die einem in einer höheren Programmiersprache geschriebenen Anwendungsprogramm die Verwendung bestimmter Daten oder Funktionen des Betriebssystems oder eines anderen Programms ermöglicht.

**Anwendungsserver.** Ein Serverprogramm in einem verteilten Netz, das die Ausführungsumgebung für ein Anwendungsprogramm bereitstellt.

**APAR.** Siehe Authorized Program Analysis Report.

**API.** Siehe Anwendungsprogrammierschnittstelle (Application Programming Interface).

### **Arbeitsbereich.**

1. Ein Verzeichnis auf Platte, das alle Projektdateien sowie Informationen wie Benutzervorgaben enthält.

2. Ein temporäres Repository für Konfigurationsdaten, das von Clients mit Verwaltungsfunktionen verwendet wird.
3. In Eclipse die Sammlung der Projekte und anderen Ressourcen, die der Benutzer momentan in der Workbench entwickelt. Metadaten zu diesen Ressourcen befinden sich in einem Verzeichnis im Dateisystem. Die Ressourcen können sich in demselben Verzeichnis befinden.

**Asynchron.** Dieser Begriff bezeichnet Ereignisse, die zeitlich nicht synchronisiert sind oder nicht in regelmäßigen oder voraussagbaren Zeitabständen auftreten.

**Asynchrones Messaging.** Eine Methode der Kommunikation zwischen Programmen, bei der das Programm eine Nachricht in eine Nachrichtenwarteschlange stellt und dann mit seinen eigenen Anforderungen fortfährt, ohne auf eine Antwort auf die Nachricht zu warten.

**Asynchrones Replikat.** Ein Shard, das Aktualisierungen nach der Festschreibung der Transaktion empfängt. Diese Methode ist schneller als ein synchrones Replikat, birgt aber das Risiko eines Datenverlusts, weil das asynchrone Replikat aus mehreren Transaktionen hinter dem primären Shard bestehen kann.

**Aufruf.** Die Aktivierung eines Programms oder einer Prozedur.

**Ausdruck.** Ein SQL- oder XQuery-Operand oder eine Sammlung von SQL- oder XQuery-Operatoren und -Operanden, die einen einzigen Wert ergibt.

**Ausführungs-Trace.** Eine Kette von Ereignissen, die in hierarchischem Format auf der Ereignisseite des Integrations-testclients aufgezeichnet und angezeigt wird.

**Ausgeschriebener Name.** Die Eigenschaft, die den logischen Namen für den Server auf der z/OS-Plattform angibt.

**Auslösen.** Bei der objektorientierten Programmierung das Veranlassen eines Statusübergangs.

**Ausnahme.** Eine Bedingung oder ein Ereignis, die bzw. das von einem normalen Prozess nicht verarbeitet werden kann.

**Ausnahmebehandlungsroutine.** Eine Gruppe von Routinen, die auf eine abnormale Bedingung reagieren. Eine Ausnahmebehandlungsroutine kann die normale Prozessausführung unterbrechen und auch wieder aufnehmen.

**Ausstellerzertifikat.** Der anerkannte Zertifikateintrag, der in der Regel in einer Truststore-Datei enthalten ist.

**Authentifizierter Benutzer.** Ein Portalbenutzer, der für die Anmeldung beim Portal einen gültigen Benutzereintrag (Benutzer-ID und Kennwort) verwendet hat. Authentifizierte Benutzer haben Zugriff auf alle öffentlichen Bereiche.

**Authentifizierung.** Ein Sicherheitsservice, der sicherstellt, dass ein Benutzer eines Computersystems wirklich die Person ist, die er zu sein vorgibt. Allgemeine Mechanismen für die Implementierung dieses Service sind Kennwörter und digitale Signaturen. Die Authentifizierung unterscheidet sich von der Berechtigung. Die Authentifizierung dient nicht zur Erteilung oder Verweigerung von Zugriff auf Systemressourcen.

**Authentifizierungsalias.** Ein Alias, mit dem die Berechtigung für den Zugriff auf Ressourcenadapter und Datenquellen gewährt wird. Ein Authentifizierungsalias enthält Authentifizierungsdaten einschließlich einer Benutzer-ID und eines Kennworts.

**Authorized Program Analysis Report.** Eine Anforderung zur Behebung eines Fehlers in einem unterstützten Release eines von IBM gelieferten Programms.

**Automatische Erkennung.** Die Erkennung von Serviceartefakten in einem Dateisystem, einer externen Registry oder in einer anderen Quelle.

**Autonomic Manager.** Eine Gruppe von Software- oder Hardwarekomponenten, die über Richtlinien konfiguriert werden und das Verhalten anderer Software- oder Hardwarekomponenten so verwalten, wie es auch ein Benutzer tun würde. Zu einem Autonomic Manager gehört ein Regelkreis, der aus Überwachungs-, Analyse-, Plan- und Ausführungskomponenten besteht.

**Autorisierung/Berechtigung.** Der Prozess, mit dem einem Benutzer, System oder Prozess entweder uneingeschränkter oder eingeschränkter Zugriff auf ein Objekt, eine Ressource oder eine Funktion gewährt wird.

**Bean.** Eine Definition oder Instanz einer JavaBeans-Komponente. Siehe auch JavaBeans, Enterprise-Bean.

**Bean-Klasse.** Bei der EJB-Programmierung (Enterprise JavaBeans) eine Java-Klasse, die eine `javax.ejb.EntityBean`- oder eine `javax.ejb.SessionBean`-Klasse implementiert.

**Bean-managed Messaging.** Über JavaBeans realisiertes Messaging. Eine Funktion des asynchronen Messaging, durch die eine Enterprise-Bean die vollständige Steuerung der Messaging-Infrastruktur erhält.

**Bean-Managed Persistence (BMP) .** Der Mechanismus, durch den die Datenübertragung zwischen den Variablen einer Entity-Bean und einem Ressourcenmanager von der Entity-Bean verwaltet wird. (Sun)

**Bean-Managed Transaction (BMT) .** Über JavaBeans realisierte Transaktion. Die Fähigkeit einer Session-Bean, eines Servlet oder einer Anwendungsclientkomponente, eigene Transaktionen direkt ohne Unterstützung eines Containers zu verwalten.

**Bean Scripting Framework.** Eine Architektur, mit der die Funktionen von Scripting-Sprachen in Java-Anwendungen eingebunden werden können.

**Befehlszeile.** Eine leere Zeile in der Anzeige, in der Befehle, Optionsnummern oder ausgewählte Optionen eingegeben werden können.

**Berechtigung.** Die Berechtigung zur Ausführung von Aktivitäten wie Lesen und Schreiben von lokalen Dateien, Erstellen von Netzverbindungen und Laden des nativen Codes.

**Berechtigungsnaehweis.** In der JAAS-Architektur (Java Authentication and Authorization Service) eine Subjektklasse mit sicherheitsrelevanten Attributen. Diese Attribute können Informationen enthalten, die für die Authentifizierung des Subjekts für neue Services verwendet wird.

**Berechtigungsrichtlinie.** Eine Richtlinie, deren Richtlinienziel ein Geschäftsservice ist und deren Vertrag mindestens eine Zusicherung enthält, auf deren Basis die Berechtigung zur Ausführung einer Kanalaktion erteilt wird.

**Berechtigugstabelle.** Eine Tabelle, die die Informationen für die Zuordnung einer Rolle zu einem Benutzer oder zu einer Gruppe enthält, die den zulässigen Zugriff eines Clients auf eine bestimmte Ressource definieren.

**Bereinigungsprogramm.** Eine Komponente, die die Zugehörigkeit von Einträgen in jeder BackingMap-Instanz steuert. Teilcaches können Bereinigungsprogramme verwenden, um Daten ohne Beeinträchtigung der Datenbank automatisch aus dem Cache zu entfernen.

#### **Bibliothek.**

1. Eine Gruppe von Modellelementen einschließlich der zugehörigen Geschäftselemente, Prozesse, Tasks, Ressourcen und Organisationen.
2. Ein Projekt, das für Entwicklung, Versionsmanagement und Organisation gemeinsam genutzter Ressourcen verwendet wird. In einer Bibliothek kann nur ein Teil der Artefakttypen erstellt und gespeichert werden, z. B. Geschäftsobjekte und Schnittstellen.

**Binärformat.** Die Darstellung eines Dezimalwerts, bei der jedes Feld zwei oder vier Byte lang sein muss. Das Vorzeichen (+ oder -) wird im Feld in dem Bit ganz links angegeben, die restlichen Bit des Felds geben den Zahlenwert an. Positive Zahlen haben im Bit für das Vorzeichen eine 0 und werden im Standardformat ("True Format") angegeben. Negative Zahlen haben im Bit für das Vorzeichen eine 1 und werden im Zweierkomplementformat ("Two's Complement Format") angegeben.

**Bindung im Zellenbereich.** Ein Bindungsbereich, in dem die Bindung nicht spezifisch und nicht einem Knoten oder Server zugeordnet ist. Diese Art Namensbindung wird unter dem persistenten Ausgangskontext einer Zelle erstellt.

**BMP.** Siehe Bean-managed Persistence.

**BMT.** Siehe Bean-managed Transaction.

**Boot-Programm.** Ein kleines Programm, mit dem umfangreichere Programme während der Systeminitialisierung geladen werden können.

**Bootstrapping.** Der Prozess, bei dem eine Ausgangsreferenz des Namensservice abgerufen wird. Die Bootstrap-Einstellung und der Hostname bilden den Ausgangskontext für JNDI-Referenzen (Java Naming and Directory Interface).

**Bottom-up-Entwicklung.** Bei Web-Services der Prozess der Entwicklung eines Service auf der Basis eines vorhandenen Artefakts wie z. B. einer Java-Bean oder einer Enterprise-Bean und nicht auf der Basis einer WSDL-Datei (WSDL = Web Services Description Language).

**Bytecode.** Ein systemunabhängiger Code, der vom Java-Compiler generiert und vom Java-Interpreter ausgeführt wird. (Sun)

**Cacheinstanzressource.** Eine Position, unter der jede Java EE-Anwendung (Java EE = Java Platform Enterprise Edition) Daten speichern, verteilen und gemeinsam nutzen kann.

**Cachereplikation.** Die gemeinsame Nutzung von Cache-IDs, Cacheinträgen und Cacheinvalidierungen mit anderen Servern innerhalb einer Replikationsdomäne.

**CIP.** Siehe Angepasstes Installationspaket (Customized Installation Package).

**Client.** Ein Softwareprogramm oder ein Computer, der Services von einem Server anfordert. Siehe auch Host.

**Clientanwendung.** Eine Anwendung, die auf einer Workstation ausgeführt wird und über eine Verbindung zu einem Client verfügt, über die die Anwendung auf die Warteschlangensteuerservices auf einem Server zugreifen kann.

**Client/Server.** Bezeichnung des Interaktionsmodells bei der verteilten Datenverarbeitung, in dem ein Programm auf einem Computer eine Anforderung an ein Programm auf einem anderen Computer sendet und die Antwort abwartet. Das anfordernde Programm ist ein Client und das antwortende Programm ein Server.

**Cloudscape.** Ein integrierbares, vollständig Java-gestütztes, objektrelationales Datenbankverwaltungssystem (ORD-BMS, Object-Relational Database Management System).

**Cluster.** Eine Gruppe von Anwendungsservern, die zusammenarbeitet, um Funktionen wie Lastausgleich und Ausweichbetrieb bereitzustellen.

**Command-Bean.** Ein Proxy, der eine einzelne Operation mithilfe der Methode execute() aufrufen kann.

**Containerserver.** Eine Serverinstanz, die mehrere Shards haben kann. Eine Java Virtual Machine (JVM) kann mehrere Containerserver haben.

**Converter.** Bei der EJB-Programmierung (EJB = Enterprise JavaBeans) eine Klasse, die eine Datenbankdarstellung in einen Objekttyp umsetzt (und umgekehrt).

**Create-Methode.** In Enterprise-Beans eine Methode, die in der Home-Schnittstelle definiert ist und von einem Client aufgerufen wird, um eine Enterprise-Bean zu erstellen. (Sun)

**CSV-Datei.** Eine Datei, deren Datensätze durch Kommas voneinander getrennte Felder enthalten.

**Dämon.** Ein Programm, das unbeaufsichtigt ausgeführt wird und fortlaufend oder in regelmäßigen Abständen Funktionen wie die Netzsteuerung ausführt.

**Dashboard.** Eine Webseite, die einen oder auch mehrere Viewer umfassen kann, mit deren Hilfe Geschäftsdaten grafisch dargestellt werden können.

**Daten-Grid.** Ein System für den Zugriff auf Terabytes oder Petabytes von Daten.

**DB2.** Eine Familie von IBM Lizenzprogrammen für die Verwaltung relationaler Datenbanken.

**Deadlock.** Eine Bedingung, bei der zwei unabhängige Steuerungsthreads blockiert werden, wobei jeder vom anderen eine Aktion erwartet. Deadlocks treten häufig dann auf, wenn Synchronisationsmechanismen zur Vermeidung von Konkurrenzsituationen hinzugefügt werden.

**Demilitarized Zone (DMZ).** Eine Konfiguration, die mehrere Firewalls enthält, um zwischen ein Unternehmens-Intranet und ein öffentliches Netz wie das Internet mehrere Sicherheitsebenen hinzuzufügen.

**Deployment Manager.** Ein Server, der den Betrieb einer logischen Gruppe oder einer Zelle anderer Server verwaltet.

**Destination.** Ein Exitpunkt, der verwendet wird, um Dokumente an ein Back-End-System oder einen Handelspartner zu liefern.



**Differenziert.** Dieser Begriff bezeichnet das Anzeigen der Details zu einem einzelnen Objekt.

**Digitales Zertifikat.** Ein elektronisches Dokument, das zur Identifikation einer Person, eines Systems, eines Servers, eines Unternehmens oder einer anderen Entität sowie zur Zuordnung eines öffentlichen Schlüssels zu der Entität dient. Ein digitales Zertifikat wird von einer Zertifizierungsstelle ausgestellt und von dieser Instanz digital signiert.

**DMZ .** Siehe Demilitarized Zone.

**DNS.** Siehe Domain Name System.

**Document Type Definition (DTD).** Die Regeln, die die Struktur einer bestimmten Klasse von SGML- oder XML-Dokumenten festlegen. Die DTD definiert die Struktur mit Elementen, Attributen und Notationen und richtet Integritätsbedingungen dafür ein, wie die einzelnen Elemente, Attribute und Notationen in der jeweiligen Klasse von Dokumenten verwendet werden können.

**Domain Name System (DNS).** Das verteilte Datenbanksystem, das Domännennamen IP-Adressen zuordnet.

**Domäne.** Ein Objekt, ein Symbol oder ein Container, der andere Objekte enthält, die die Ressourcen einer Domäne darstellen. Die Objektdomäne kann für die Verwaltung dieser Ressourcen verwendet werden.

**Do-while-Schleife.** Eine Schleife, die die gleiche Aktivitätenfolge so lange wiederholt, bis eine bestimmte Bedingung erfüllt ist. Anders als eine While-Schleife testet eine Do-while-Schleife ihre Bedingung am Ende der Schleife. Dies bedeutet, dass ihre Aktivitätenfolge immer mindestens einmal ausgeführt wird.

**Dropdown.** Siehe Pulldown.

**DTD.** Siehe Dokumenttypdefinition.

**DTD-Dokumentdefinition.** Eine Beschreibung oder ein Layout eines XML-Dokuments, die bzw. das auf einer XML-Dokumenttypdefinition basiert.

**Durchsatz.** Die Kennzahl für das Arbeitsvolumen, das von einem Gerät, wie einem Computer oder Drucker, im Verlauf eines bestimmten Zeitraums ausgeführt wird, beispielsweise die Anzahl Jobs pro Tag.

**Dynamischer Cache.** Eine Konsolidierung mehrerer Caching-Aktivitäten einschließlich der entsprechenden Servlets, Web-Services und WebSphere-Befehle in einem Service, in dem diese Aktivitäten Konfigurationsparameter gemeinsam nutzen und zusammenwirken, um die Leistung zu verbessern.

**Dynamischer Cluster.** Ein Servercluster, der basierend auf den von den Cluster-Membren erfassten Leistungsdaten anhand von Wertigkeiten die Arbeitslast dynamisch auf seine Cluster-Member verteilt.

**EAR .** Siehe Unternehmensarchiv.

**EAR-Projekt.** Siehe Unternehmensanwendungsprojekt.

**Eclipse.** Eine Open-Source-Initiative, die unabhängigen Softwareanbietern (ISVs) und anderen Toolentwicklern eine Standardplattform für die Entwicklung von plug-kompatiblen Anwendungsentwicklungstools zur Verfügung stellt.

**Edition.** Eine Nachfolgeneration der Implementierung einer bestimmten Gruppe versionsgesteuerter Artefakte.

**Editorbereich.** In Eclipse und bei Eclipse-basierten Produkten der Bereich des Workbench-Fensters, in dem Dateien zum Bearbeiten geöffnet werden.

**Eigenschaft.** Ein Merkmal eines Objekts, durch das das Objekt beschrieben wird. Eine Eigenschaft kann geändert werden. Eigenschaften können unter anderem Namen, Typ, Wert oder Verhalten eines Objekts beschreiben.

**Eigenständig.** Unabhängig von allen anderen Einheiten, Programmen oder Systemen. In einer Netzumgebung greift ein eigenständiges System auf alle erforderlichen Ressourcen lokal zu.

**Eigenständiger Server.** Ein Katalogservice oder Containerserver, der vom Betriebssystem verwaltet wird, das den Serverprozess startet und stoppt.

**Eingabeaufforderung.** Eine Komponente einer Aktion, mit der angegeben wird, dass eine Benutzereingabe für ein Feld erforderlich ist, bevor ein Übergang zur Ausgabeanzeige erfolgt.

**Eingangsunterbrechungspunkt.** Ein Unterbrechungspunkt, der für ein Komponentenelement gesetzt ist und erreicht wird, bevor das Komponentenelement aufgerufen wird.

**EJB.** Siehe Enterprise JavaBeans.

**EJB-Abfrage.** In der EJB-Abfragesprache eine Zeichenfolge, die Folgendes enthält: eine optionale SELECT-Klausel, die die EJB-Objekte angibt, die zurückgegeben werden sollen; eine FROM-Klausel, die die Bean-Collections benennt; eine optionale WHERE-Klausel, die Suchvergleichselemente für die Collections enthält; eine optionale ORDER BY-Klausel, die die Sortierung der Ergebniscollection angibt; und Eingabeparameter, die den Argumenten der Finder-Methode entsprechen.

**EJB-Container.** Ein Container, der den EJB-Komponentenvertrag der Java EE-Architektur implementiert. In diesem Vertrag wird eine Laufzeitumgebung für Enterprise-Beans angegeben, die Services für die Sicherheit, den gemeinsamen Zugriff, das Lebenszyklusmanagement, Transaktionen, die Implementierung und anderes enthält. (Sun)

**EJB-Factory.** Eine Access-Bean, die die Erstellung einer Enterprise-Bean-Instanz oder die Suche nach einer solchen Instanz vereinfacht.

**EJB-Home-Objekt.** Bei der EJB-Programmierung (Enterprise JavaBeans) ein Objekt, das die Lebenszyklusoperationen (Erstellen, Entfernen, Suchen) für eine Enterprise-Bean bereitstellt. (Sun)

**EJB-JAR-Datei.** Ein Java-Archiv, das ein EJB-Modul enthält. (Sun)

**EJB-Kontext.** In Enterprise-Beans ein Objekt, das einer Enterprise-Bean ermöglicht, vom Container bereitgestellte Services aufzurufen und Informationen über den Aufrufenden einer Clientmethode abzurufen. (Sun)

**EJB-Modul.** Eine Softwareeinheit, die sich aus einer oder mehreren Enterprise-Beans und einem EJB-Deployment-Deskriptor besteht. (Sun)

**EJB-Objekt.** Bei Enterprise-Beans ein Objekt, dessen Klasse die ferne Schnittstelle der Enterprise-Bean implementiert (Sun).

**EJB-Projekt.** Ein Projekt, das die Ressourcen enthält, die für EJB-Anwendungen benötigt werden. Hierzu zählen Enterprise-Beans, Home-Schnittstellen, Local- und Remote-Schnittstellen, JSP-Dateien, Servlets und Implementierungsdeskriptoren.

**EJB-Referenz.** Ein logischer Name, der von einer Anwendung zur Lokalisierung der Home-Schnittstelle einer Enterprise-Bean in der Zielbetriebsumgebung verwendet wird.

**EJB-Server.** Software, die Services für einen EJB-Container bereitstellt. Ein EJB-Server kann einen oder mehrere EJB-Container aufnehmen. (Sun)

**EJB-Vererbung.** Eine Form der Vererbung, bei der eine Enterprise-Bean die Eigenschaften, Methoden und Deskriptorattribute für die Steuerung auf Methodenebene von einer anderen Enterprise-Bean in derselben Gruppe erbt.

### **Endpunkt.**

1. Eine JCA-Anwendung oder ein anderer Clientkonsument eines Ereignisses aus dem unternehmensweiten Informationssystem.
2. Das System, das den Ursprung oder das Ziel einer Sitzung darstellt.

**Endpunkt-Listener.** Der Punkt oder die Adresse, an dem bzw. der eingehende Nachrichten für einen Web-Service von einem Service Integration Bus empfangen werden.

**Engpass.** Eine Situation im System, die eintritt, wenn die Konkurrenz um eine Ressource sich auf die Leistung auswirkt.

**Enterprise-Bean.** Eine Komponente, die eine Geschäfts-Task oder Geschäftsentität implementiert und in einem EJB-Container enthalten ist. Entity-Beans, Session-Beans und Message-Driven Beans sind verschiedene Typen von Enterprise-Beans. (Sun) Siehe auch Bean.

**Enterprise JavaBeans (EJB).** Eine von Sun Microsystems definierte Komponentenarchitektur für die Entwicklung und Implementierung objektorientierter, verteilter Unternehmensanwendungen (Java EE).

**Enterprise Service Bus (ESB).** Eine flexible Konnektivitätsinfrastruktur für die Integration von Anwendungen und Services, die eine flexible und einfach zu verwaltende Strategie für die SOA-Implementierung (SOA = service-orientierte Architektur) bietet.

**Entität.**

1. Eine einfache Java-Klasse, die eine Zeile in einer Datenbanktabelle oder einen Eintrag in einer Zuordnung darstellt.
2. In Markup-Sprachen wie XML eine Sammlung von Zeichen, die als Einheit referenziert werden können, beispielsweise für die Einbindung von häufig wiederholtem Text oder Sonderzeichen in einem Dokument.

**Entity-Bean.** In der EJB-Programmierung eine Enterprise-Bean, die persistente Daten darstellt, die in einer Datenbank verwaltet werden. Jede Entity-Bean hat eine eigene Entität. (Sun)

**Entserialisierung.** Eine Methode für die Konvertierung einer serialisierten Variablen in Objektdaten.

**Ereignis.**

1. Die Änderung eines Status wie beispielsweise der Abschluss oder das Fehlschlagen einer Operation, eines Geschäftsprozesses oder einer Benutzertask, die eine nachfolgende Aktion wie beispielsweise das Speichern der Ereignisdaten in einem Datenrepository oder das Aufrufen eines anderen Geschäftsprozesses auslösen kann.
2. Eine Änderung an den in einem EIS (Enterprise Information System) gespeicherten Daten, die vom Adapter verarbeitet und zur Lieferung von Geschäftsobjekten vom EIS zu den Endpunkten (Anwendungen) verwendet wird, die über die Änderung benachrichtigt werden müssen.

**Erstellungsdefinitionsdatei.** Eine XML-Datei, die Komponenten und Merkmale für ein angepasstes Installationspaket (CIP, Customized Installation Package) angibt.

**Erstellungspfad.** Der Pfad, der während der Kompilierung des Java-Quellcodes verwendet wird, um referenzierte Klassen zu finden, die sich in anderen Projekten befinden.

**Erstellungsplan.** Eine XML-Datei, die die Verarbeitung definiert, die zum Erstellen der Generierungsausgabe erforderlich ist und die das System angibt, auf dem die Verarbeitung stattfindet.

**Erstellungszeitdaten.** Objekte, die vom Umsetzungsprogramm nicht verwendet werden, wie beispielsweise EDI-Standards, ROD-Dokumenttypen (ROD = Record Oriented Data; satzorientierte Daten) und Zuordnungen.

**ESB.** Siehe Enterprise Service Bus (ESB).

**Exklusive Sperre.** Eine Sperre, die verhindert, dass Anwendungsprozesse ausgeführt werden, die gleichzeitig auf Datenbankdaten zugreifen. Siehe auch Gemeinsame Sperre.

**Export.** Eine offene Schnittstelle eines SCA-Moduls (SCA = Service Component Architecture), die einen Geschäfts-service für externe Systeme und Benutzer bereitstellt. Ein Export verfügt über eine Bindung, die definiert, wie auf den Service von Serviceanforderern zugegriffen werden kann, z. B. als Web-Service.

**Exportdatei.**

1. Eine Datei, die während des Entwicklungsprozesses für eingehende Operationen erstellt wurde und die Konfigurationseinstellungen für die Eingangsverarbeitung enthält.
2. Eine Datei, die exportierte Daten enthält.

**Extensible Markup Language (XML).** Eine Standardmetasprache für die Definition von Markup-Sprachen, die auf Standard Generalized Markup Language (SGML) basiert.

**eXtreme-Scale-Grid.** Ein Datenmuster, das für die Interaktion mit eXtreme Scale verwendet wird, wenn sich alle Daten und Clients in einem einzigen Prozess befinden.

**Factory.** In der objektorientierten Programmierung eine Klasse, die zum Erstellen von Instanzen einer anderen Klasse verwendet wird. Eine Factory wird verwendet, um die Erstellung von Objekten einer bestimmten Klasse auf einen Bereich zu isolieren, damit neue Funktionen ohne weitreichende Codeänderungen bereitgestellt werden können.

**Failover.** Eine automatische Operation, mit der auf ein redundantes oder Bereitschaftssystem umgeschaltet werden kann, wenn eine Software-, Hardware- oder Netzunterbrechung eintritt.

**Fehler.** Eine Abweichung zwischen einem berechneten, festgestellten oder gemessenen Wert oder einer solchen Bedingung und dem wahren, definierten oder theoretisch richtigen Wert bzw. der entsprechenden Bedingung.

**Fehlerhafte Leseoperation.** Eine Leseanforderung, die keinen Sperrmechanismus besitzt. Das bedeutet, dass Daten gelesen werden können, die später zurückgesetzt werden, was zu einer Inkonsistenz zwischen den gelesenen Daten und den Daten in der Datenbank führt.

**Fehlerprotokollstrom.** Ein fortlaufender Fluss von Fehlerinformationen, die in einem vordefinierten Format übertragen werden.

**Firewall .** Eine Netzkonfiguration, die in der Regel Hardware und Software umfasst und verhindert, dass nicht autorisierter Datenverkehr in ein sicheres Netz eingeht bzw. dieses verlässt.

**Fixpack.** Eine kumulative Gruppe von Fixes (Programmkorrekturen), die zwischen den geplanten Refresh-Packs, Produktaktualisierungen oder Releases bereitgestellt wird. Fixpacks ermöglichen es den Kunden, ihre Systeme auf eine bestimmte Wartungsstufe umzustellen. Siehe auch Vorläufiger Fix.

**For-Schleife.** Eine Schleife, die eine angegebene Anzahl von Wiederholungen für eine bestimmte Aktivitätenfolge ausführt.

**Garbage-Collection.** Eine Routine, die den Hauptspeicher durchsucht, um Speicher aus Programmsegmenten oder inaktiven Daten zurückzufordern.

**Gehäuse.** Ein Metallrahmen, in den verschiedene elektronische Komponenten eingebaut werden.

#### **Geltungsbereich.**

1. Eine Spezifikation der Begrenzung, innerhalb deren Systemressourcen verwendet werden können.
2. Bei Web-Services eine Eigenschaft, die die Lebensdauer des Objekts angibt, das die Aufrufanforderung bearbeitet.

**Gemeinsame Sperre.** Eine Sperre, die gleichzeitig aktive Anwendungsprozesse auf Leseoperationen für Datenbankdaten beschränkt.

**General Inter-ORB Protocol (GIOP).** Ein Protokoll, das von der Common Object Request Broker Architecture (CORBA) zum Festlegen des Nachrichtenformats verwendet wird.

**Generisches Objekt.** Ein Objekt, mit dem in API-Aufrufen und XPATH-Ausdrücken auf Konzepte, angepasste Entitäten oder Sammlungen verwiesen wird. Beispiel: Mit dem XPATH-Ausdruck '/WSRR/GenericObject' werden alle Konzepte aus WebSphere Service Registry and Repository abgerufen.

**Gerüst.** Der Entwurf für eine Implementierungsklasse.

**Getter-Methode.** Eine Methode, die den Zweck hat, den Wert einer Instanz oder einer Klassenvariablen abzurufen. Dadurch kann ein anderes Objekt den Wert einer seiner Variablen herausfinden.

**GIOP .** Siehe General Inter-ORB Protocol.

#### **Global.**

1. Dieser Begriff bezeichnet die Eigenschaft eines Elements, das für alle Prozesse innerhalb eines Arbeitsbereichs bereitgestellt wird. Ein globales Element wird im Projektbaum aufgeführt und kann in mehreren Prozessen verwendet werden. Tasks, Prozesse, Repositories und Services können entweder global oder lokal sein. Global bedeutet hier, dass von allen Prozessen eines Projekts auf sie verwiesen werden kann, lokal bedeutet hingegen, dass sie einem bestimmten Prozess zugeordnet sind.

2. Dieser Begriff bezeichnet Informationen, die für mehr als ein Programm oder eine Unterroutine verfügbar sind.

**Globale Instanz-ID.** Eine global eindeutige ID, die entweder von der Anwendung oder vom Emitter generiert und als Primärschlüssel für die Ereignisidentifikation verwendet wird.

**Globales Attribut.** In XML ein Attribut, das als untergeordnetes Schemaelement und nicht als Teil einer komplexen Typdefinition deklariert ist. Auf globale Attribute kann unter Verwendung des Attributs ref in einem oder auch in mehreren Inhaltsmodellen verwiesen werden.

**Globales Element.** In XML ein Element, das als untergeordnetes Schemaelement und nicht als Teil einer komplexen Typdefinition deklariert ist. Auf globale Elemente kann unter Verwendung des Attributs `ref` in einem oder auch in mehreren Inhaltsmodellen verwiesen werden.

**Globale Sicherheit.** Die globale Sicherheit bezieht sich auf alle Anwendungen, die in der Umgebung ausgeführt werden, und legt fest, ob Sicherheitseinrichtungen zum Einsatz kommen. Ferner legt sie den Typ des für die Authentifizierung verwendeten Registry fest sowie andere Werte, von denen viele Standardeinstellungen sind.

**Globale Transaktion.** Eine wiederherstellbare Arbeitseinheit, die durch einen oder mehrere Ressourcenmanager in einer Umgebung für verteilte Transaktionen ausgeführt und von einem externen Transaktionsmanager koordiniert wird.

**Globale Variable.** Eine Variable, die verwendet wird, um die ihr während der Umsetzung zugeordneten Werte aufzunehmen und zu bearbeiten. Sie wird von verschiedenen Zuordnungen und Dokumentumsetzungen gemeinsam verwendet. Einer von drei Variablentypen, die von der Zuordnungsbefehlssprache von Data Interchange Services unterstützt werden.

### **Gruppe.**

1. Ein Benutzerverbund, der Zugriffsberechtigungen für geschützte Ressourcen gemeinsam benutzen kann.
2. Ein Satz zusammengehöriger Dokumente in einem Austausch. Ein Austausch muss keine Gruppen enthalten, kann aber auch viele Gruppen enthalten.
3. Zwei oder mehrere Personen in einem Bereich, die für die Zugehörigkeit zu einem Bereich in einer Gruppe zusammengefasst werden.

**HA.** Siehe Hohe Verfügbarkeit (HA, High Availability).

**HA-Gruppe.** Eine Sammlung mit mindestens einem Mitglied, die zur Bereitstellung einer hohen Verfügbarkeit für einen Prozess genutzt wird.

**HA-Richtlinie.** Eine Gruppe von Regeln, die für eine HA-Gruppe definiert sind. Diese Regelgruppe gibt vor, ob null (0) oder mehr Mitglieder aktiviert sind. Die Richtlinie wird einer bestimmten HA-Gruppe zugeordnet, indem die Übereinstimmungskriterien der Richtlinie mit dem Gruppennamen abgeglichen werden.

**High Availability Manager.** Ein Gerüst, in dem die Stammgruppenzugehörigkeit bestimmt und Statusinformationen zwischen den Stammgruppen-Mitgliedern übertragen wird.

**Hohe Verfügbarkeit (HA, High Availability).** Bezeichnung für ein Clustersystem, das neu konfiguriert wird, wenn ein Knoten- oder Dämonfehler auftritt, damit die Workloads auf die verbleibenden Knoten im Cluster umverteilt werden können.

### **Host.**

1. Ein Computer, der mit einem Netz verbunden ist und einen Zugriffspunkt auf dieses Netz bereitstellt. Der Host kann ein Client, ein Server oder Client und Server gleichzeitig sein.
2. Bei der Leistungsprofilerstellung ein System, das über Prozesse verfügt, für die ein Profil erstellt werden soll. Siehe auch Server.

### **Hostname.**

1. Bei der Internetkommunikation der Name eines Computers. Der Hostname kann ein vollständig qualifizierter Domänenname wie `mycomputer.city.company.com` oder ein bestimmter Teilname wie `mycomputer` sein.
2. Der Netzname für einen Netzadapter auf einer physischen Maschine, auf der der Knoten installiert ist.

**Hostsystem.** Ein Großrechnersystem innerhalb eines Unternehmens, das als Host für 3270-Anwendungen eingesetzt wird. Bei Entwicklungstools für 3270-Terminal-Services verwendet der Entwickler den 3270-Terminal-Service-Recorder, um eine Verbindung zum Hostsystem herzustellen.

**HTTP over SSL (HTTPS).** Ein Web-Protokoll für sichere Transaktionen, das Seitenanforderungen von Benutzern und vom Web-Server zurückgegebene Seiten verschlüsselt und entschlüsselt.

### **HTTPS.**

1. Siehe HTTP over SSL.

2. Siehe Hypertext Transfer Protocol Secure.

**Hypertext Transfer Protocol Secure (HTTPS).** Ein Internet-Protokoll, das von Webservern und Webbrowsern für die sichere Übertragung und Anzeige von Hypermedia-Dokumenten im Internet verwendet wird.

**IDE .** Siehe Integrierte Entwicklungsumgebung (Integrated Development Environment).

**If-then-Regel.** Eine Regel, in der die Aktion ('then'-Teil) nur dann ausgeführt wird, wenn die Bedingung ('if'-Teil) zutrifft.

**IIOP.** Siehe Internet Inter-ORB Protocol.

**Implementieren.** Dateien oder Software in einer Betriebsumgebung installieren. In Java EE (Java Platform Enterprise Edition) umfasst das Implementieren die Erstellung eines Implementierungsdeskriptors, der für den zu implementierenden Anwendungstyp geeignet ist.

**Implementierphase.** Siehe Implementierungsphase.

**Implementierungscode.** Zusätzlicher Code, der den Bean-Implementierungscode aktiviert, der von einem Anwendungsentwickler zur Verwendung in einer bestimmten EJB-Laufzeitumgebung geschrieben wurde. Implementierungscode kann mithilfe von Tools generiert werden, die der Anbieter des Anwendungsservers liefert.

**Implementierungsdeskriptor.** Eine XML-Datei (Extensible Markup Language), die den Einsatz eines Moduls oder einer Anwendung beschreibt, indem Sie Konfigurations- und Containeroptionen festlegt. Ein EJB-Deployment-Deskriptor übergibt beispielsweise Informationen zur Verwaltung und Steuerung einer Enterprise-Bean an einen EJB-Container.

**Implementierungsphase.** Eine Phase, in der Operationen zur Erstellung der Hosting-Umgebung für Anwendungen und zur Implementierung dieser Anwendungen ausgeführt werden. In dieser Phase werden auch die Ressourcenabhängigkeiten der Anwendung sowie Betriebsbedingungen, Kapazitätsanforderungen, Integritätsbedingungen und Zugriffsbeschränkungen aufgelöst.

**Implementierungsrichtlinie.** Eine optionale Methode für die Konfiguration einer eXtreme-Scale-Umgebung auf der Basis verschiedener Elemente, wie z. B. Anzahl der Systeme, Server, Partitionen, Replikate (einschließlich Replikattyp) und Heap-Speichergrößen für jeden Server.

**Implementierungstopologie.** Die Konfiguration von Servern und Clustern in einer Implementierungsumgebung und die physischen und logischen Beziehungen zwischen diesen Einheiten.

**Implementierungsumgebung.** Eine Sammlung konfigurierter Cluster, Server und Middlewarekomponenten, die zusammenarbeiten, um eine Umgebung bereitzustellen, in der Softwaremodule unterstützt werden können. Eine Implementierungsumgebung kann beispielsweise einen Host für Nachrichtenziele, einen Prozessor oder eine Sortierkomponente für Geschäftsereignisse und Verwaltungsprogramme umfassen.

**Implementierungsverzeichnis.** Das Verzeichnis, in dem sich die veröffentlichte Serverkonfiguration und die Webanwendung auf dem System, auf dem der Anwendungsserver installiert ist, befinden.

## **Import.**

1. Das Entwicklungsartefakt, das zum Importieren eines Service verwendet wird, der nicht in einem Modul integriert ist.

2. Der Punkt, über den ein SCA-Modul auf einen externen Service (d. h. einen Service außerhalb des SCA-Moduls) in derselben Weise zugreift, wie dies bei einem lokalen Service möglich ist. Ein Import definiert die Interaktion zwischen dem SCA-Modul und dem Serviceanbieter. Ein Import verfügt über eine Bindung sowie mindestens eine Schnittstelle.

**Index.** Eine Gruppe von Zeigern, die logisch nach den Werten eines Schlüssels sortiert sind. Indizes ermöglichen den schnellen Zugriff auf Daten und können die Eindeutigkeit der Schlüsselwerte für die Zeilen in der Tabelle erzwingen.

**Information Center.** Eine Sammlung von Informationen, die Benutzern eines oder mehrerer Produkte zur Unterstützung bereitgestellt wird, die vom Produkt gesondert gestartet werden kann und die eine Liste der Abschnitte zur Navigation sowie eine Suchmaschine enthält.

**Installationspaket.** Eine installierbare Einheit eines Softwareprodukts. Softwareproduktpakete sind separat installierbare Einheiten, die unabhängig von anderen Paketen dieses Softwareprodukts eingesetzt werden können.

**Installationsziel.** Das System, auf dem ausgewählte Installationspakete installiert werden.

**Instanz.** Ein spezielles Vorkommen eines Objekts, das einer Klasse angehört.

**Instanzieren.** Das Darstellen einer Abstraktion durch eine konkrete Instanz.

**Integrated Development Environment (IDE) .** Eine Gruppe von Softwareentwicklungstools wie Quelleneditoren, Compiler und Debugger, auf die über eine gemeinsame Benutzerschnittstelle zugegriffen werden kann.

**Integrierter Server.** Ein Katalogservice oder Containerserver, der sich in einem vorhandenen Prozess befindet und innerhalb des Prozesses gestartet und gestoppt wird.

**Internet Inter-ORB Protocol (IIOP).** Ein Protokoll, das für die Kommunikation zwischen CORBA-Object-Request-Brokern (CORBA = Common Object Request Broker Architecture) eingesetzt wird.

**Internet Protocol (IP).** Ein Protokoll, das Daten über ein Netz oder über miteinander verbundene Netze leitet. Dieses Protokoll agiert als Mittler zwischen den höheren Protokollschichten und dem physischen Netz.

**IP.** Siehe Internet Protocol.

**IP-Sprayer.** Eine Einheit, die sich zwischen den von den Benutzern eingehenden Anforderungen und den Anwendungsserverknoten befindet und Anforderungen an Knoten weiterleitet.

**Iteration.** Siehe Schleife.

**Iterator.** Eine Klasse oder Anweisung, die verwendet wird, um die Objekte einer Objektgruppe nacheinander zu durchlaufen.

**JAAS.** Siehe Java Authentication and Authorization Service.

**JAF .** Siehe JavaBeans Activation Framework.

**JAR-Datei.** Eine Java-Archivdatei. Siehe auch Webarchiv, Unternehmensarchiv.

**Java.** Eine objektorientierte Programmiersprache für portierbaren, interpretierenden Code, die die Interaktion zwischen fernen Objekten unterstützt. Java wurde von Sun Microsystems, Incorporated entwickelt und spezifiziert.

**Java API für XML (JAX).** Eine Gruppe Java-basierter Anwendungsprogrammierschnittstellen für die Verarbeitung verschiedener Operationen, an denen über Extensible Markup Language (XML) definierte Daten beteiligt sind.

**Java-Archiv.** Ein Format für komprimierte Dateien, mit dem alle Ressourcen, die zur Installation und Ausführung eines Java-Programms erforderlich sind, in einer einzigen Datei gespeichert werden können. Siehe auch Webarchiv, Unternehmensarchiv.

**Java Authentication and Authorization Service (JAAS).** In Java EE technology, a standard API for performing security-based operations. Über JAAS können Services die Authentifizierung und Berechtigung von Benutzern ausführen. Gleichzeitig bleiben die zugehörigen Anwendungen unabhängig von den zugrunde liegenden Technologien.

**JavaBeans .** Ein für Java von Sun Microsystems definiertes, portierbares und plattformunabhängiges Komponentenmodell, das wiederverwendbar ist. Siehe auch Bean.

**JavaBeans Activation Framework (JAF) .** Eine Standarderweiterung für die Java-Plattform, die beliebige Datentypen und verfügbare Operationen bestimmt und eine Bean so instanzieren kann, dass sie relevante Services ausführt.

**Java Command Language.** Eine Scripting-Sprache für die Java-Umgebung, die zur Erstellung von Webinhalten und zur Steuerung von Java-Anwendungen benutzt werden kann.

**Java Connector Security.** Eine Architektur, die zur Erweiterung des End-to-End-Sicherheitsmodells für Java EE-basierte Anwendungen entworfen wurde, so dass dieses auch unternehmensweite Informationssysteme (EIS = Enterprise Information Systems) umfassen kann.

**Java Database Connectivity (JDBC)** . Ein Industriestandard für datenbankunabhängige Konnektivität zwischen der Java-Plattform und verschiedenen Datenbanken. Die JDBC-Schnittstelle stellt eine Schnittstelle auf Aufrufebene für den SQL-basierten und den XQuery-basierten Datenbankzugriff bereit.

**Java-Datei.** Eine Quellendatei (mit der Erweiterung .java), die bearbeitet und in Bytecode (eine Datei mit der Erweiterung .class) kompiliert werden kann.

**Javadoc.**

1. Ein Tool, das die Deklarationen und Dokumentationskommentare in einer Gruppe von Quellendateien syntaktisch analysiert und eine Reihe von HTML-Seiten erstellt, die die Klassen, untergeordneten Klassen, Schnittstellen, Konstruktoren, Methoden und Felder beschreiben. (Sun)

2. Dieser Begriff bezeichnet ein Tool, das die Deklarationen und Dokumentationskommentare in einer Gruppe von Quellendateien syntaktisch analysiert und eine Reihe von HTML-Seiten erstellt, die die Klassen, untergeordneten Klassen, Schnittstellen, Konstruktoren, Methoden und Felder beschreiben.

**Java EE.** Siehe Java Platform Enterprise Edition.

**Java EE-Anwendung.** Eine beliebige implementierbare Einheit mit Java EE-Funktionalität. Bei dieser Einheit kann es sich um ein einzelnes Modul oder um eine Gruppe von Modulen handeln, die in einer EAR-Datei (EAR = Enterprise Archive) mit einem Java EE-Anwendungsimplementierungsdeskriptor gepackt sind. (Sun)

**Java EE Connector Architecture (JCA).** Eine Standardarchitektur für die Verbindung der Java EE-Plattform mit heterogenen unternehmensweiten Informationssystemen (EIS = Enterprise Information Systems).

**Java-EE-Server.** Eine Laufzeitumgebung, die EJB- oder Web-Container bereitstellt.

**Java-Klasse.** Eine Klasse, die in der Programmiersprache Java geschrieben wurde.

**JavaMail API.** Ein plattform- und protokollunabhängiges Gerüst für die Erstellung Java-basierter Mail-Clientanwendungen.

**Java Message Service (JMS).** Eine Anwendungsprogrammierschnittstelle, die Java-Sprachfunktionen für die Verarbeitung von Nachrichten zur Verfügung stellt.

**Java Naming and Directory Interface (JNDI).** Eine Erweiterung der Java-Plattform, die eine Standardschnittstelle für heterogene Namens- und Verzeichnisservices bereitstellt.

**Java Platform, Enterprise Edition (Java EE).** Eine Umgebung zum Entwickeln und Implementieren von Unternehmensanwendungen, die von Sun Microsystems Inc. definiert wurde. Die Java EE-Plattform besteht aus einer Reihe von Services, Anwendungsprogrammierschnittstellen (APIs) und Protokollen, die die Funktionalität für die Entwicklung mehrschichtiger, webbasierter Anwendungen zur Verfügung stellen. (Sun)

**Java Platform, Standard Edition (Java SE).** Die zentrale Plattform der Java-Technologie. (Sun)

**Java-Projekt.** Bei Eclipse ein Projekt, das kompilierbaren Java-Quellcode enthält und einen Container für Quellendateien oder Pakete darstellt.

**Java Runtime Environment (JRE).** Eine Untergruppe des Java Developer Kit, die die zentralen ausführbaren Programme und Dateien enthält, auf denen die Java-Standardplattform basiert. Die Java Runtime Environment (JRE) umfasst die Java Virtual Machine (JVM) sowie die wichtigsten Klassen und Unterstützungsdateien.

**JavaScript.** Eine Web-Scripting-Sprache, die sowohl von Browsern als auch von Web-Servern verwendet wird. (Sun)

**JavaScript Object Notation.** Ein einfaches Datenaustauschformat, das auf der Objekt-Literal-Notation von JavaScript basiert. JSON ist programmiersprachenneutral, verwendet allerdings Konventionen aus Sprachen, wie C, C++, C#, Java, JavaScript, Perl und Python.

**Java SE.** Siehe Java Platform Standard Edition.

**Java Secure Socket Extension (JSSE)** . Ein Java-Paket, das zur Bereitstellung der sicheren Internetkommunikation dient. Es implementiert eine Java-Version der Protokolle SSL (Secure Sockets Layer) und TLS (Transport Layer Security) und unterstützt die Datenverschlüsselung, die Serverauthentifizierung sowie die Überprüfung der Nachrichtenintegrität und optional die Clientauthentifizierung.



**Java SE Development Kit (JDK).** Der Name des Software Development Kit, das von Sun Microsystems für die Java-Plattform bereitgestellt wird.

**JavaServer Pages (JSP)** . Eine serverseitige Scripting-Technologie, mit deren Hilfe Java-Code dynamisch in Webseiten (HTML-Dateien) eingebettet und beim Bereitstellen der Seite ausgeführt werden kann, um dynamische Inhalte an den Client zurückzugeben.

**Java Specification Request (JSR).** Eine formal eingereichter Spezifikationsvorschlag für die Java-Plattform.

**Java Virtual Machine (JVM).** Die Softwareimplementierung eines Prozessors, die kompilierten Java-Code (Applets und Anwendungen) ausführt.

**Java Virtual Machine Profiler Interface (JVMPPI).** Ein Profilerstellungstool, das die Erfassung von Informationen wie z. B. von Daten zur Garbage-Collection sowie Angaben zu der Java Virtual Machine-Anwendungsprogrammierschnittstelle (JVM-API) unterstützt, die zur Ausführung des Anwendungsservers eingesetzt wird.

**JAX** . Siehe Java API for XML.

**JCA.** Siehe Java EE Connector Architecture.

**JDBC.** Siehe Java Database Connectivity.

**JDK.** Siehe Java SE Development Kit.

**JMS.** Siehe Java Message Service.

**JMS-Datenbindung.** Eine Datenbindung, die eine Zuordnung zwischen dem von einer externen JMS-Nachricht verwendeten Format und der SDO-Darstellung (SDO = Service Data Object) bereitstellt, die von einem SCA-Modul (SCA = Service Component Architecture) eingesetzt wird.

**JMX** . Siehe Java Management Extensions.

**JMX (Java Management Extensions).** Eine Methode für die Verwaltung mit Java-Technologie. JMX ist eine universelle, offene Erweiterung der Programmiersprache Java für die Verwaltung, die in jedem Unternehmen eingesetzt werden kann, in dem eine Verwaltung erforderlich ist.

**JNDI.** Siehe Java Naming and Directory Interface.

**JSP.** Siehe JavaServer Pages.

**JSP-Datei.** Eine scriptgesteuerte HTML-Datei mit der Dateierweiterung ".jsp", die den Einschluss dynamischer Inhalte in Webseiten ermöglicht. Eine JSP-Datei kann über einen URL direkt angefordert, von einem Servlet oder in einer HTML-Seite aufgerufen werden.

**JSP-Seite.** Ein textbasiertes Dokument, das feste Schablonendaten und JSP-Elemente verwendet, die beschreiben, wie eine Anforderung verarbeitet werden soll, um eine Antwort zu erstellen. (Sun)

**JSR.** Siehe Java Specification Request.

**JSSE.** Siehe Java Secure Socket Extension.

**JVM.** Siehe Java Virtual Machine.

**JVMPPI.** Siehe Java Virtual Machine Profiler Interface.

**Jython.** Eine Implementierung der Programmiersprache Python, die mit der Java-Plattform integriert ist.

**Katalog.** Ein Container, dessen Projektbaum abhängig vom jeweiligen Containertyp Prozesse, Daten, Ressourcen, Organisationen oder Berichte enthält.

**Katalogservice.** Ein Service, der die Positionierung von Shards steuert und den Status der Container erkennt und überwacht.

**Kategorie.** Ein Container, der in einem Strukturdiagramm zum Gruppieren von Elementen auf der Basis eines gemeinsamen Attributs oder einer gemeinsamen Qualität verwendet wird.

**Klasse.** In der objektorientierten Programmierung ein Modell oder eine Schablone, die verwendet werden kann, um Objekte mit einer gemeinsamen Definition und gemeinsamen Merkmalen, Operationen und Verhaltensmustern zu erstellen. Ein Objekt ist eine Instanz einer Klasse.

**Klassendatei.** Eine kompilierte Java-Quelldatei.

**Klassenhierarchie.** Die Beziehungen zwischen Klassen, die einen gemeinsamen Vorfahren (Basisklasse) haben.

**Klassenlader.** Eine Komponente der Java Virtual Machine (JVM), die für das Suchen und Laden von Klassendateien verwendet wird. Ein Klassenladeprogramm wirkt sich auf das Packen von Anwendungen und das Laufzeitverhalten von gepackten Anwendungen aus, die auf Anwendungsservern implementiert sind.

**Klassenpfad.** Eine Liste mit Verzeichnissen und JAR-Dateien, die Ressourcendateien oder Java-Klassen enthalten, die ein Programm zur Laufzeit dynamisch laden kann.

**Klassifikationsmerkmal.** Ein spezielles Attribut, das zur Gruppierung und Farbcodierung von Prozesselementen dient.

**Knoten.**

1. Eine logische Gruppierung verwalteter Server.
2. Ein Element in einer Baumstruktursteuerung, wie beispielsweise ein einfaches Element, ein Verbundelement, ein Zuordnungsbefehl, ein Kommentar oder ein Gruppenknoten.
3. In XML die kleinste Einheit der gültigen und vollständigen Struktur in einem Dokument.
4. Die grundlegenden Formen, aus denen sich ein Diagramm zusammensetzt.

**Kohärenter Cache.** Ein Cache, der die Integrität so verwaltet, dass alle Clients dieselben Daten sehen.

**Kompilereinheit.** Ein Teil eines Computerprogramms, das ausreichend vollständig ist, um ordnungsgemäß kompiliert werden zu können.

**Kompilierzeit.** Der Zeitraum, während dessen ein Computerprogramm zu einem ausführbaren Programm kompiliert wird.

**Komponente.**

1. Ein wiederverwendbares Objekt oder Programm, das eine bestimmte Funktion ausführt und mit anderen Komponenten und Anwendungen arbeitet.
2. Bei Eclipse ein bestimmtes Plug-in oder mehrere Plug-ins, die zusammenarbeiten, um eine eigenständige Funktionsgruppe bereitzustellen.

**Komponentenelement.** Eine Entität in einer Komponente, für die ein Unterbrechungspunkt gesetzt werden kann, wie beispielsweise eine Aktivität oder ein Java-Snippet in einem Geschäftsprozess bzw. ein Mediationsbasiselement oder ein Knoten in einem Mediationsablauf.

**Komponenteninstanz.** Eine aktive Komponente, die parallel zu anderen Instanzen derselben Komponente ausgeführt werden kann.

**Komponententest.** Ein automatisierter Test einer oder mehrerer Komponenten einer Unternehmensanwendung, die Java-Klassen, EJB-Beans oder Web-Services beinhalten kann.

**Lastausgleich.** Die Überwachung von Anwendungsservern und die Verwaltung der Workload auf Servern. Wenn einer der Server überlastet ist, werden die Anforderungen an einen anderen Server mit mehr Kapazität weitergeleitet.

**Laufzeit.** Der Zeitraum, während dessen ein Computerprogramm ausgeführt wird.

**Laufzeittopologie.** Eine Abbildung des momentanen Zustands der Umgebung.

**LDAP.** Siehe Lightweight Directory Access Protocol.

**LDAP-Verzeichnis.** Ein Repository-Typ, in dem Informationen zu Personen, Organisationen und anderen Ressourcen gespeichert werden und auf den über das LDAP-Protokoll zugegriffen wird. Die Einträge im Repository sind in einer hierarchischen Struktur organisiert. In manchen Fällen gibt diese hierarchische Struktur die Struktur oder Geographie einer Organisation wieder.

**Lebensdauer.** (TTL, Time-to-Live) Das Zeitintervall in Sekunden, für das ein Eintrag vor dem Löschen im Cache erhalten sein kann.

**Lebenszyklus.** Ein vollständiger Durchlauf durch die vier Softwareentwicklungsphasen Konzeption, Ausarbeitung, Konstruktion und Ablösung.

**Lightweight Directory Access Protocol (LDAP).** Ein offenes Protokoll, das TCP/IP verwendet, um den Zugriff auf Verzeichnisse bereitzustellen, die ein X.500-Modell unterstützen und bei dem weniger aufwendige Ressourcenanforderungen gelten als bei dem komplexeren X.500 Directory Access Protocol (DAP). LDAP kann beispielsweise eingesetzt werden, um Personen, Organisationen und andere Ressourcen in einem Internet- oder Intranetverzeichnis zu lokalisieren.

**Listener.** Ein Programm, das ankommende Anforderungen erkennt und den zugehörigen Channel startet.

**Listener-Port.** Ein Objekt, das die Zuordnung zwischen einer Verbindungs-Factory, einer Zieladresse und einer implementierten Message-Driven-Bean (MDB) definiert. Listener-Ports vereinfachen die Verwaltung der Zuordnungen zwischen diesen Ressourcen.

**Loader.** (Ladeprogramm) Eine Komponente, die Daten aus einem persistenten Speicher liest bzw. in diesen schreibt.

**Lokal.**

1. Bezeichnet eine Einheit, eine Datei oder ein System, auf die bzw. das direkt über das System eines Benutzers zugegriffen wird, ohne dass hierbei eine Übertragungsleitung verwendet werden muss.
2. Bezeichnet ein Element, das nur innerhalb seines eigenen Prozesses zur Verfügung steht.

**Lokale Datenbank.** Eine Datenbank, die sich auf der momentan verwendeten Workstation befindet.

**LTPA.** Siehe Lightweight Third Party Authentication.

**LTPA (Lightweight Third Party Authentication).** Ein Protokoll, das die Sicherheit in einer verteilten Umgebung durch Verschlüsselung unterstützt.

**Managed Bean (MBean).** In der Spezifikation Java Management Extensions (JMX) die Java-Objekte, die Ressourcen und die zugehörige Instrumentierung implementieren.

**Map.**

1. Eine Datenstruktur, die Schlüssel Werten zuordnet.
2. Eine Datei, die die Umsetzung zwischen Quellen und Zielen definiert.
3. In der EJB-Entwicklungsumgebung die Spezifikation, die angibt, wie die CMP-Felder (CMP = Container-managed Persistence) einer Enterprise-Bean den Spalten in einer Tabelle einer relationalen Datenbank oder einem anderen persistenten Speicher entsprechen.

**MBean.** Siehe Managed Bean.

**MBean-Provider.** Eine Bibliothek, die eine Implementierung einer JMX-MBean und die zugehörige XML-Deskriptor-datei enthält.

**Messgröße.** Ein Behältnis für Informationen in einem Überwachungskontext, normalerweise für einen Geschäftsleistungsmesswert.

**Methode.** Bei der objektorientierten Programmierung eine Operation, die von einem Objekt ausgeführt werden kann. Ein Objekt kann viele Methoden aufweisen.

**Nachgeordnet.** Dieser Begriff bezeichnet die Richtung eines Bearbeitungsablaufs. Dieser verläuft vom ersten Knoten innerhalb des Prozesses (d. h. vom vorgeordneten Knoten) zum letzten Knoten des Prozesses (d. h. zum nachgeordneten Knoten).

**Namespace.** Ein logischer Container, in dem alle Namen eindeutig sind. Die eindeutige Kennung für ein Artefakt setzt sich aus dem Namespace und dem lokalen Namen des Artefakts zusammen.

**Node Agent.** Ein Verwaltungsagent, der alle Anwendungsserver auf einem Knoten verwaltet und den Knoten in der Verwaltungszelle repräsentiert.

**ObjectGrid.** Eine Grid-fähige Speicherdatenbank für Anwendungen, die in Java geschrieben werden. ObjectGrid kann als speicherinterne Datenbank oder zum Verteilen von Daten in einem Netz verwendet werden.

**Object Request Broker.** In der objektorientierten Programmierung eine Software, die als Vermittler fungiert, indem Sie den Austausch von Anforderungen und Antworten zwischen Objekten transparent erlaubt.

**Objekt.** Im objektorientierten Design oder in der objektorientierten Programmierung eine konkrete Ausführung (Instanz) einer Klasse, die aus Daten und den zugehörigen Operationen besteht. Ein Objekt enthält die Instanzdaten, die von der Klasse definiert werden. Eigner der Operationen, die den Daten zugeordnet sind, ist aber die Klasse.

**Objektorientierte Programmierung.** Eine Programmierungsmethode auf der Basis der Konzepte zur Datenabstraktion und Vererbung. Im Gegensatz zu Verfahren der prozeduralen Programmierung liegt der Schwerpunkt der objektorientierten Programmierung nicht darauf, wie etwas erreicht wird, sondern darauf, welche Datenobjekte das Problem umfasst und wie diese bearbeitet werden.

**ODBC.** Siehe Open Database Connectivity.

**Open Database Connectivity (ODBC).** Eine standardisierte Anwendungsprogrammierschnittstelle (API = Application Programming Interface) für den Zugriff auf Daten in relationalen und nicht relationalen Datenbankmanagementsystemen. Unter Verwendung dieser API können Datenbankmanagementsysteme auf Daten zugreifen, die in Datenbankmanagementsystemen auf verschiedenen Computern gespeichert sind, auch wenn die einzelnen Datenbankmanagementsysteme unterschiedliche Formate für die Datenspeicherung und unterschiedliche Programmierschnittstellen verwenden.

**Open Source.** Software, deren Quellcode für die Verwendung oder für Änderungen allgemein zur Verfügung steht. Open-Source-Software wird normalerweise in Form einer allgemeinen Zusammenarbeit entwickelt und wird frei verfügbar gemacht, ihre Verwendung und Neuverteilung kann jedoch Lizenzbeschränkungen unterliegen. Linux ist ein sehr bekanntes Beispiel für Open-Source-Software.

**Operation.** Eine Implementierung von Funktionen oder Abfragen, zu deren Ausführung ein Objekt aufgerufen werden kann.

**ORB.** Siehe Object Request Broker.

**Ordner.** Ein Container, der zum Verwalten von Objekten verwendet wird.

**Organisation.** Eine Entität, in der Personen zusammenarbeiten, um angegebene Ziele zu erreichen, wie ein Unternehmen, eine Firma oder eine Fabrik.

**Paket.**

1. In der Java-Programmierung ein Gruppe von Typen. Pakete werden mit dem Schlüsselwort package deklariert. (Sun)
2. Der Wrapper um den Dokumentinhalt, der das zum Übertragen eines Dokuments im Internet zu verwendende Format (z. B. RNIF, AS1 oder AS2) definiert.
3. In Module zusammengefasste Komponenten und in Unternehmensanwendungen zusammengefasste Module.

**Partitionierungsfeature (WPF).** Ein Programmierungsframework und eine Systemmanagementinfrastruktur, die das Konzept der Partitionierung für Enterprise-Beans, HTTP-Datenverkehr und Datenbankzugriff unterstützt.

**Performance Monitoring Infrastructure (PMI).** Eine Gruppe von Paketen und Bibliotheken für die Erfassung, Zustellung, Verarbeitung und Anzeige von Leistungsdaten.

**Persistenter Datenspeicher.** Ein nicht flüchtiger Speicher für Ereignisdaten (wie beispielsweise ein Datenbanksystem), der über Sitzungsgrenzen hinweg existiert und nach der Ausführung des erstellenden Programms oder Prozesses erhalten bleibt.

**Persistent speichern.** Elemente über Sitzungsgrenzen hinweg verwalten, in der Regel in einem nicht flüchtigen Speicher wie einem Datenbanksystem oder einem Verzeichnis.

#### **Persistenz.**

1. Ein Merkmal von Daten, die über Sitzungsgrenzen hinweg beibehalten werden, oder eines Objekts, das auch nach der Ausführung des erstellenden Programms oder Prozesses erhalten bleibt (normalerweise in einem nicht flüchtigen Speicher, wie einem Datenbanksystem).
2. In Java EE das Protokoll, mit dem der Status einer Entity-Bean zwischen den jeweiligen Instanzvariablen und einer zugrunde liegenden Datenbank übertragen wird. (Sun)

**Pessimistisches Sperren.** Eine Sperrstrategie, bei der bei Auswahl einer Zeile eine Sperre gesetzt wird, die so lange gehalten wird, bis versucht wird, eine Aktualisierungs- oder Löschoperation mit Suche für diese Zeile durchzuführen.

**Plattform Java.** Ein Sammelbegriff für die Sprache Java zum Schreiben von Programmen. Sie umfasst eine Gruppe von APIs, Klassenbibliotheken und anderen Programmen, die bei der Entwicklung, Kompilierung und Prüfung von Programmen auf Fehler verwendet werden, sowie eine Java Virtual Machine (JVM), die die Klassendateien lädt und ausführt. (Sun)

**Plug-in.** Ein separat installierbares Softwaremodul, das einem vorhandenen Programm, einer vorhandenen Anwendung oder einer vorhandenen Schnittstelle eine Funktion hinzufügt.

**PMI .** Siehe Performance Monitoring Infrastructure.

**Port.** Gemäß Definition in einem WSDL-Dokument (Web Services Description Language) ein Endpoint, der als Kombination von Bindung und Netzadresse definiert wird.

**Portnummer.** Bei der Internetkommunikation die Kennung für einen logischen Connector zwischen einer Anwendungsentität und dem Transportservice.

#### **Primärschlüssel.**

1. Ein Objekt, das eine Entity-Bean eines bestimmten Typs eindeutig kennzeichnet.
2. In einer relationalen Datenbank ein Schlüssel, der eine einzige Zeile in einer Datenbanktabelle eindeutig kennzeichnet.

**Primitiver Datentyp.** In Java eine Datentypkategorie, die eine Variable mit einem Einzelwert beschreibt, dessen Größe und Format dem Typ entsprechen: eine Zahl, ein Zeichen oder ein boolescher Wert. Beispiele für primitive Datentypen sind byte, short, int, long, float, double, char, boolean.

**Profil.** Daten, die die Merkmale eines Benutzers, einer Gruppe, einer Ressource, eines Programms, einer Einheit oder einer fernen Position beschreiben.

**Protokollbindung.** Eine Bindung, mit der der Enterprise Service Bus Nachrichten unabhängig vom Übertragungsprotokoll verarbeiten kann.

**Protokollierung.** Die Aufzeichnung von Daten zu bestimmten Ereignissen auf dem System, wie z. B. Fehler.

**Proxy.** Ein Anwendungsgateway zwischen zwei Netzen für eine bestimmte Netzanwendung, wie Telnet oder FTP, wenn beispielsweise der Proxy-Telnet-Server einer Firewall die Authentifizierung des Benutzers ausführt und dann den Datenverkehr über den Proxy laufen lässt, als ob dieser nicht vorhanden wäre. Die Funktion wird in der Firewall ausgeführt, nicht auf der Client-Workstation. Dadurch vergrößert sich die Arbeitslast in der Firewall.

**Proxy-Cluster.** Eine Gruppe von Proxyservern, die HTTP Anforderungen über den Cluster verteilt.

**Proxy-Peer-Zugriffspunkt.** Ein Mittel, mit dem die Kommunikationseinstellungen für einen Peer-Zugriffspunkt ermittelt werden können, auf den kein direkter Zugriff möglich ist.

#### **Proxy-Server.**

1. Ein Server, der als Zwischenstation für HTTP-Webanforderungen von einer Anwendung oder einem Webserver auftritt. Ein Proxy-Server tritt stellvertretend für die Content-Server im Unternehmen auf.
2. Ein Server, der Anforderungen für einen anderen Server empfängt und für den Client agiert (als Proxy des Clients), um den angeforderten Service abzurufen. Ein Proxy-Server wird häufig verwendet, wenn der Client und der

Server für eine Direktverbindung inkompatibel sind. Beispiel: Der Client kann die Anforderungen zur Sicherheitsauthentifizierung des Servers nicht erfüllen, soll aber für einige Services berechtigt werden.

**Prozess.**

1. Eine immer weiter voranschreitende Prozedur, die aus einer Reihe gesteuerter Aktivitäten besteht, die systematisch auf ein bestimmtes Ergebnis oder Ziel gerichtet sind.
2. Die Reihenfolge der Dokumente oder Nachrichten, die zwischen den Community-Managern und den Teilnehmern ausgetauscht werden sollen, um eine Geschäftstransaktion auszuführen.

**PTF.** Siehe Vorläufige Programmkorrektur (Program Temporary Fix).

**public.**

1. Bei der objektorientierten Programmierung bezieht sich dieser Begriff auf einen Klasseneintrag, auf den alle Klassen zugreifen können.
2. In der Programmiersprache Java bezieht sich dieser Begriff auf eine Methode oder Variable, auf die Elemente aus anderen Klassen zugreifen können. (Sun)

**Punkt-zu-Punkt.** Dieser Begriff bezeichnet einen Stil einer Messaging-Anwendung, bei der die sendende Anwendung das Ziel der Nachricht kennt.

**QoS.** Siehe Servicequalität (Quality of Service).

**Qualifikationsmerkmal.** Ein einfaches Element, das einem anderen generischen Verbundelement oder einfachen Element eine bestimmte Bedeutung gibt. Qualifikationsmerkmale werden bei der Zuordnung einmaliger oder mehrfacher Vorkommen verwendet. Ein Qualifikationsmerkmal kann auch zur Bezeichnung des Namensbereichs verwendet werden, mit dem der zweite Teil des Namens interpretiert wird. Dieser Teil wird im Allgemeinen als ID bezeichnet.

**Read-Through-Cache.** Ein Teilcache, der Dateneinträge nach Schlüssel lädt, wenn sie angefordert werden. Wenn Daten nicht im Cache gefunden werden, werden die fehlenden Daten mit dem Ladeprogramm abgerufen, das die Daten aus dem Back-End-Repository lädt und in den Cache einfügt.

**Region .** Ein zusammenhängender Bereich des virtuellen Speichers, der allgemeine Merkmale aufweist und von Prozessen gemeinsam genutzt werden kann.

**Rekursion.** Eine Programmiertechnik, bei der ein Programm oder eine Routine sich selbst aufruft, um aufeinander folgende Schritte in einer Operation auszuführen, wobei jeder Schritt die Ausgabe des vorhergehenden Schrittes verwendet.

**Replikat.** Ein Server, der eine Kopie des Verzeichnisses bzw. der Verzeichnisse eines anderen Servers enthält. Replikate sichern Server, um die Leistung bzw. Antwortzeiten zu verbessern und die Datenintegrität zu gewährleisten.

**Replikation.** Der Prozess, mit dem ein definierter Satz von Daten an mehr als einer Position verwaltet wird. Zur Replikation gehören das Kopieren festgelegter Änderungen für eine Position (eine Quelle) an eine andere (ein Ziel) und die Synchronisation der Daten an beiden Positionen.

**Ressource.**

1. Eine diskrete Ressource. Beispiele: Anwendungssuiten, Anwendungen, Geschäftsservices, Schnittstellen, Endpunkte und Geschäftsereignisse.
2. Eine Einrichtung eines Computer- oder Betriebssystems, die für einen Job, eine Task oder ein aktives Programm erforderlich ist. Ressourcen können unter anderem Hauptspeicher, Ein-/Ausgabeeinheiten, die Verarbeitungseinheit, Datensätze, Dateien, Bibliotheken, Ordner, Anwendungsserver und Steuer- oder Verarbeitungsprogramme sein.
3. Eine Person, ein Bauteil oder Material, die bzw. das für die Ausführung einer Task oder eines Projekts verwendet wird. Jede Ressource ist ein bestimmtes Vorkommen oder Beispiel für eine Ressourcendefinition.

**Richtlinie.** Eine Gruppe von Aspekten, die das Verhalten einer verwalteten Ressource oder eines Benutzers beeinflussen.

**Rolle.**

1. Eine Beschreibung einer Funktion, die von einer Einzelperson oder einer Massenressource ausgeführt werden soll, sowie der zu ihrer Ausführung erforderlichen Qualifikationen. Bei der Simulation und Analyse wird der Begriff Rolle auch für die qualifizierten Ressourcen verwendet.
2. Eine Jobfunktion, die die Tasks, die ein Benutzer ausführen kann, und die Ressourcen angibt, auf die ein Benutzer Zugriff hat. Einem Benutzer können eine oder mehrere Rollen zugeordnet werden.
3. Eine logische Gruppe von Principals, die eine Gruppe von Berechtigungen bereitstellt. Der Zugriff auf Operationen wird über die Zugriffsberechtigungen für eine Rolle gesteuert.
4. In einer Beziehung legt eine Rolle die Funktion und Teilnahme von Entitäten fest. Rollen erfassen Anforderungen bezüglich Struktur und Integritätsbedingungen für teilnehmende Entitäten und ihre Art der Teilnahme. Beispielsweise lauten die Rollen in einer Beschäftigungsbeziehung "Arbeitgeber" und "Mitarbeiter".

**Root.** Der Benutzername für den Systembenutzer mit der höchsten Berechtigungsstufe.

**Schleife.** Eine Instruktionsfolge, die wiederholt ausgeführt wird.

**Schlüssel.**

1. Ein verschlüsselter mathematischer Wert, der zum digitalen Signieren, Überprüfen, Verschlüsseln oder Entschlüsseln einer Nachricht verwendet wird.
2. Die Informationen, die eine reale Entität, die von einem Überwachungskontext protokolliert wird, beschreiben und eindeutig identifizieren.

**Schlüsselwort.** Eines der vordefinierten Wörter einer Programmiersprache, einer Kunstsprache, Anwendung oder eines Befehls.

**Schnittstelle.** Eine Gruppe von Operationen, die verwendet werden, um den Service einer Klasse oder einer Komponente anzugeben.

**Schwellenwert.** Eine Einstellung, die für einen Interrupt in einer Simulation gilt, der definiert, wann eine Prozesssimulation auf der Basis einer Bedingung angehalten werden soll, die für einen angegebenen Anteil von Vorkommen eines bestimmten Ereignisses existiert.

**Script.** Eine Reihe von Befehlen, die in einer Datei zusammengefasst sind und durch die beim Ausführen der Datei eine bestimmte Funktion ausgeführt wird. Scripts werden während ihrer Ausführung interpretiert.

**Scripting.** Eine Art der Programmierung, bei der vorhandene Komponenten als Basis zum Erstellen von Anwendung wiederverwendet werden.

**SDK.** Siehe Software Development Kit.

**Secure Socket Layer (SSL).** Ein Sicherheitsprotokoll für den Schutz personenbezogener Daten bei der Datenübertragung. Mit SSL können Client/Server-Anwendungen auf eine Weise kommunizieren, die das Ausspionieren, die Manipulation von Daten während der Übertragung und die Nachrichtenfälschung verhindern soll.

**Serialisierung.** Bei der objektorientierten Programmierung das sequenzielle Schreiben von Daten aus dem Programmspeicher an ein Kommunikationsmedium.

**Serialisierungsmethode.** Eine Methode zur Konvertierung von Objektdaten in ein anderes Format, wie z. B. Binärformat oder XML.

**Servant-Region .** Ein zusammenhängender Bereich des virtuellen Speichers, der dynamisch gestartet wird, wenn die Arbeitslast größer wird, und automatisch gestoppt wird, wenn die Arbeitslast geringer wird.

**Server.** Ein Softwareprogramm oder ein Computer, das bzw. der Services für andere Softwareprogramme oder Computer bereitstellt. Siehe auch Host.

**Servercluster.** Eine Gruppe von Servern, die in der Regel auf unterschiedlichen physischen Maschinen installiert sind und mit denselben Anwendungen konfiguriert sind, aber als ein logischer Server zusammenarbeiten.

**Service-Level-Agreement (SLA).** Ein Vertrag zwischen einem Kunden und einem Serviceanbieter, der die Erwartungen hinsichtlich des Service-Levels, die Verfügbarkeit, Leistungswerte und andere messbare Zielsetzungen betreffend, angibt.

**Servicequalität (Quality of Service).** Eine Gruppe von Kommunikationsmerkmalen, die eine Anwendung erfordert. Die Servicequalität definiert bestimmte Werte für die Übertragungspriorität, die Stufe der Weiterleitungszuverlässigkeit und die Sicherheitsstufe.

**Servlet.** Ein Java-Programm, das in einem Webserver ausgeführt wird und die Funktionen des Servers durch Generierung dynamischen Inhalts als Reaktion auf Webclientanforderungen erweitert. Servlets werden häufig verwendet, um Datenbanken mit dem Web zu verbinden.

**Setter-Methode.** Eine Methode, mit der ein Wert einer Instanz- oder Klassenvariablen manipuliert werden kann. Mit dieser Methode kann ein anderes Objekt den Wert einer seiner Variablen ermitteln.

**Shard.** (Engl., Scherbe) Eine Instanz einer Partition. Ein Shard kann eine primäre Instanz oder ein Replikat sein.

**Shell-Script.** Ein Programm oder Script, das von der Shell eines Betriebssystems interpretiert wird.

**Sicherheitsadministrator.** Die Person, die den Zugriff auf Geschäftsdaten und Programmfunktionen steuert.

**Sicherheitstoken.** Die Darstellung von Anforderungen, die ein Client stellt. Beispiele für derartige Anforderungen sind Name, Kennwort, Identität, Schlüssel, Zertifikat, Gruppe, Berechtigung usw.

**Sitzung.**

1. Eine logische oder virtuelle Verbindung zwischen zwei Stationen, Softwareprogrammen oder Einheiten in einem Netz, die die Kommunikation und den Datenaustausch zwischen diesen beiden Elementen ermöglicht.
2. Eine Reihe von Anforderungen an ein Servlet, die von demselben Benutzer und demselben Browser stammen.
3. In Java EE ein Objekt, mit dem ein Servlet die Interaktion eines Benutzers mit einer Webanwendung über mehrere HTTP-Anforderungen hinweg protokollieren kann.

**Sitzungsaffinität.** Eine Methode für die Konfiguration von Anwendungen, in denen ein Client immer mit demselben Server verbunden ist. Diese Konfigurationen inaktivieren das Workload Management nach der ersten Verbindung, weil eine Clientanforderung immer an denselben Server weitergeleitet werden muss.

**Skalierbarkeit.** Die Erweiterungsmöglichkeit eines Systems, wenn Ressourcen, wie Prozessoren, Hauptspeicher oder Speicher, hinzugefügt werden.

**SLA.** Siehe Service-Level-Agreement.

**Software Development Kit (SDK).** Eine Sammlung von Tools, APIs und Dokumentation, die den Entwickler bei der Entwicklung von Software in einer bestimmten Maschinensprache oder für eine bestimmte Betriebsumgebung unterstützt.

**Speicherverlust.** Die Auswirkung eines Programms, das Referenzen auf Objekte verwaltet, die nicht mehr benötigt werden und deshalb freigegeben werden müssen.

**Sperre.** Eine Methode, mit der verhindert werden kann, dass von einem Anwendungsprozess vorgenommene nicht festgeschriebene Änderungen von einem anderen Anwendungsprozess wahrgenommen werden oder dass ein Anwendungsprozess Daten aktualisiert, auf die ein anderer Prozess gerade zugreift. Eine Sperre gewährleistet die Integrität der Daten, indem sie verhindert, dass gleichzeitig angemeldete Benutzer auf inkonsistente Daten zugreifen.

**SQL.** Siehe Structured Query Language.

**SQL-Abfrage.** Eine Komponente bestimmter SQL-Anweisungen, von der eine Ergebnistabelle angegeben wird.

**SSL.** Siehe Secure Sockets Layer.

**SSL-Channel.** Ein Typ von Channel in einer Transportkette, der der Transportkette ein SSL-Konfigurationsrepertoire zuordnet.

**Stapelspeicher.** Ein Bereich im Hauptspeicher, in dem gewöhnlich Informationen, wie temporäre Registerdaten, Werte von Parametern und Rückkehradressen von Unterroutinen, gespeichert werden und der auf dem Prinzip LIFO (Last in, First out) basiert.

**Statisch.** Ein Schlüsselwort der Programmiersprache Java, mit dem eine Variable als Klassenvariable definiert wird.



**Structured Query Language (SQL)** . Eine standardisierte Sprache für die Definition und Bearbeitung von Daten in einer relationalen Datenbank.

**Synchroner Prozess.** Ein Prozess, der durch den Aufruf einer Anforderungs-/Antwortoperation gestartet wird. Das Ergebnis des Prozesses wird von derselben Operation zurückgegeben.

**Synchrones Replikat.** Ein Shard, das Aktualisierungen im Rahmen der Transaktion für das primäre Shard empfängt, um die Datenkonsistenz zu gewährleisten, wodurch die Antwortzeit im Vergleich mit einem asynchronen Replikat verbessert werden kann.

**Synchronisieren.** Das Hinzufügen, Entfernen oder Ändern einer Funktion oder eines Artefakts, damit sie/es einer anderen Funktion bzw. einem anderen Artefakt entspricht.

**Syntax.** Die Regeln zum Erstellen eines Befehls oder einer Anweisung.

**Systemanalytiker.** Ein Spezialist, der für die Umsetzung von Geschäftsanforderungen in Systemdefinitionen und Lösungen verantwortlich ist.

**TCP.** Siehe Transmission Control Protocol.

**TCP-Channel** . Ein Typ von Channel in einer Transportkette, der Clientanwendungen ermöglicht, persistente Verbindungen in einem lokalen Netz (LAN) zu verwenden.

**TCP/IP.** Siehe Transmission Control Protocol/Internet Protocol.

**TCP/IP-Überwachungsserver.** Eine Laufzeitumgebung, die alle Anforderungen und Antworten zwischen einem Webbrowser und einem Anwendungsserver sowie TCP/IP-Aktivitäten überwacht.

**Thin Application Client.** Eine einfache, für den Download verfügbare Java-Anwendungslaufzeit, die mit Enterprise-Beans interagieren kann.

**Thin Client.** Ein Client, der über wenig oder keine installierte Software, aber über Zugriff auf Software verfügt, die von zugeordneten Netzservern verwaltet und bereitgestellt wird. Ein Thin Client ist eine Alternative zu einem Client mit vollem Funktionsumfang, wie beispielsweise einer Workstation.

**Thread.** Ein Datenstrom von Maschineninstruktionen, der von einem Prozess gesteuert wird. Bei manchen Betriebssystemen ist ein Thread die kleinste Operationseinheit in einem Prozess. Mehrere Threads können gleichzeitig laufen und dabei unterschiedliche Jobs ausführen.

**Thread-Konflikt.** Eine Bedingung, bei der ein Thread auf eine Sperre oder ein Objekt wartet, die bzw. das von einem anderen Thread gehalten wird.

**Tivoli Performance Viewer.** Ein Java-Client, der die PMI-Daten (Performance Monitoring Infrastructure) von einem Anwendungsserver abrufen und sie in verschiedenen Formaten anzeigen.

**Token.**

1. Eine Markierung, mit der der aktuelle Status einer Prozessinstanz während eines Simulationslaufs protokolliert wird.
2. Eine bestimmte Nachricht oder ein bestimmtes Bitmuster, die bzw. das die Berechtigung oder temporäre Kontrolle für die Übertragung von Daten in einem Netz angibt.

**Topologie.** Die physische oder logische Zuordnung der Position der Netzbetriebskomponenten oder Knoten in einem Netz. Bus, Ring, Stern und Baumstruktur sind gängige Beispiele für Netztopologien.

**Transaktion.** Ein Prozess, bei dem alle Datenänderungen, die während einer Transaktion vorgenommen werden, zusammen als Einheit festgeschrieben oder als Einheit rückgängig gemacht werden.

**Transmission Control Protocol/Internet Protocol (TCP/IP).** Eine Gruppe von standardisierten, nicht proprietären Übertragungsprotokollen, die über verschiedene Typen miteinander verbundener Netze zuverlässige durchgängige Verbindungen zwischen Anwendungen zur Verfügung stellt.

**Transmission Control Protocol (TCP).** Ein Übertragungsprotokoll, das im Internet sowie in allen Netzen verwendet wird, die die IETF-Standards (IETF = Internet Engineering Task Force) für netzübergreifende Protokolle verwenden. TCP bietet ein zuverlässiges Host-to-Host-Protokoll in DFV-Netzen mit Paketvermittlung und in miteinander verbundenen Systemen solcher Netze.

**Trennen von Webservern.** Eine Topologie, in der der Webserver physisch vom Anwendungsserver getrennt ist.

**Truststore-Datei.** Eine Schlüsseldatenbankdatei, die die öffentlichen Schlüssel für eine vertrauenswürdige Entität enthält.

**Typ.**

1. In der Java-Programmierung eine Klasse oder Schnittstelle.
2. In einem WSDL-Dokument ein Element, das Datentypdefinitionen enthält und hierfür ein Typsystem verwendet (z. B. XSD).

**Überwachungssignal.** Ein Signal, das eine Entität an eine andere sendet, um anzugeben, dass sie noch immer aktiv ist.

**UDDI.** Siehe Universal Description, Discovery, and Integration.

**Umgebung.** Eine benannte Gruppe von logischen und physischen Ressourcen, die verwendet wird, um die Leistung einer Funktion zu verbessern.

**Umgebungsvariable.** Eine Variable, die angibt, wie ein Betriebssystem oder ein anderes Programm ausgeführt wird, oder die die vom Betriebssystem erkannten Einheiten festlegt.

**Unbeaufsichtigte Installation.** Eine Installation, die keine Nachrichten an die Konsole sendet, sondern Nachrichten und Fehler in Protokolldateien speichert. Bei einer unbeaufsichtigten Installation können Antwortdateien für die Dateneingabe verwendet werden.

**Unbeaufsichtigter Modus.** Eine Methode für die Installation oder Deinstallation einer Produktkomponente über die Befehlszeile, ohne dass eine grafische Benutzerschnittstelle angezeigt wird. Bei Verwendung des Befehlszeilenmodus gibt der Benutzer die für das Installations- oder Deinstallationsprogramm erforderlichen Daten direkt in der Befehlszeile oder in einer Datei (mit dem Namen Options- oder Antwortdatei) an.

**Uniform Resource Identifier (URI).**

1. Eine kompakte Zeichenfolge zum Identifizieren einer abstrakten oder physischen Ressource.
2. Eine eindeutige Adresse, die zum Identifizieren von Inhalten im Web (z. B. einer Textseite, einem Video- oder Tonclip, einem Standbild oder animierten Bild bzw. einem Programm) verwendet wird. Die häufigste Form von URI ist die Adresse einer Webseite, die eine Sonderform oder eine Untergruppe der URI darstellt und Uniform Resource Locator (URL) genannt wird. Eine URI beschreibt in der Regel, wie auf die Ressource, den Computer mit der Ressource und den Namen der Ressource (einen Dateinamen) auf dem Computer zugegriffen wird.

**Uniform Resource Locator (URL).** Die eindeutige Adresse einer Informationsressource, die in einem Netz wie dem Internet zugänglich ist. Der URL enthält das Namens Kürzel für das Protokoll, mit dem auf die Informationsressource zugegriffen wird, und die Informationen, mit deren Hilfe das Protokoll die Informationsressource lokalisiert.

**Uniform Resource Name (URN).** Ein Name, der einen Web-Service eindeutig für einen Client identifiziert.

**Universal Description, Discovery, and Integration (UDDI).** Eine Reihe standardisierter Spezifikationen, mit denen Unternehmen und Anwendungen auf schnelle und einfache Weise Web-Services über das Internet finden und verwenden können.

**Universally Unique Identifier (UUID).** Die numerische 128-Bit-Kennung, mit der sichergestellt wird, dass zwei Komponenten nicht dieselbe Kennung aufweisen.

**UNIX System Services.** Ein Element von z/OS, das eine UNIX-Umgebung erstellt, die den Spezifikationen XPG4 UNIX 1995 entspricht und zwei Schnittstellen offener Systeme für das Betriebssystem z/OS bereitstellt: eine Anwendungsprogrammierschnittstelle (API = Application Programming Interface) und eine interaktive Shell-Schnittstelle.

**Unterabfrage.** In SQL ein Subselect in einem Prädikat. Beispiel: Eine Anweisung SELECT in der WHERE- oder HAVING-Klausel einer anderen SQL-Anweisung.

**Unterbrechungspunkt.** Ein markierter Punkt innerhalb eines Prozesses oder eines Programmablaufs, bei dessen Erreichen der Ablauf angehalten wird, um beispielsweise Debug- oder Überwachungsoperationen auszuführen.

**Untergeordneter Knoten.** Ein Knoten, der sich im Geltungsbereich eines anderen Knotens befindet.

**Unterklasse.** In Java eine Klasse, die über Vererbung aus einer bestimmten Klasse ableitet wird.

**Unternehmensanwendungsprojekt.** Eine Struktur und Hierarchie von Ordnern und Dateien, die einen Implementierungsdeskriptor, ein IBM Erweiterungsdokument sowie Dateien enthalten, die für alle im Implementierungsdeskriptor definierten Java EE-Module einheitlich sind.

**Unternehmensarchiv (EAR).** Ein besonderer Typ einer JAR-Datei, der durch den Java EE-Standard definiert ist und der zur Implementierung von Java EE-Anwendungen auf Java EE-Anwendungsservern verwendet wird. Eine EAR-Datei enthält EJB-Komponenten, einen Implementierungsdeskriptor sowie Webarchivdateien (WAR-Dateien) für einzelne Webanwendungen. Siehe auch Webarchiv.

**URI.** Siehe Uniform Resource Identifier.

**URL.** Siehe Uniform Resource Locator.

**URL-Schema.** Ein Format, das eine andere Objektreferenz enthält.

**URN.** Siehe Uniform Resource Name.

**UUID .** Siehe Universally Unique Identifier.

**Variable.** Eine Darstellung eines änderbaren Werts.

**Veraltet.** Dieser Begriff bezeichnet eine Entität, z. B. ein Programmierungselement oder eine Programmierungsfunktion, die zwar noch unterstützt wird, deren Verwendung aber nicht mehr empfohlen wird und die möglicherweise sehr bald nicht mehr aktuell sein wird.

**Vererbung.** Eine Technik in der objektorientierten Programmierung, bei der vorhandene Klassen als Basis für die Erstellung anderer Klassen verwendet werden. Durch die Vererbung umfassen die spezifischeren Elemente die Struktur und das Verhalten der allgemeineren Elemente.

### **Verfügbarkeit.**

1. Die Bedingung, die Benutzern den Zugriff auf und die Verwendung ihrer Anwendungen und Daten ermöglicht.
2. Die Zeiträume, während derer auf eine Ressource zugegriffen werden kann. Die Services eines Vertragsnehmers können beispielsweise werktags von 09.00 Uhr bis 17.00 Uhr sowie samstags von 09.00 Uhr bis 15.00 Uhr verfügbar sein.

### **Verknüpfung.**

1. (Join) Ein Prozesselement zur Rekombination und Synchronisation von Parallelverarbeitungspfaden nach einer Entscheidung oder Verzweigung. Eine Verknüpfung wartet an allen Eingangszweigen auf Eingabedaten, bevor die Fortführung des Prozesses zugelassen wird.
2. Eine relationale SQL-Operation, in der Daten aus zwei Tabellen, gewöhnlich über eine Verknüpfungsbedingung, die die zu verknüpfenden Spalten angibt, abgerufen werden können.
3. Die Konfiguration einer Eingangsverbindung, die das Verhalten dieser Verbindung festlegt.

**Version.** Ein gesondert lizenziertes Programm, das gewöhnlich neuen wichtigen Code oder neue wichtige Funktionen enthält.

**Verteiltes eXtreme Scale.** Ein Verwendungsmuster für die Interaktion mit eXtreme Scale, wenn Server und Clients in mehreren Prozessen existieren.

**Verzweigung.** (Fork) Ein Prozesselement, das Kopien seiner Eingabe erstellt und diese gleichzeitig über mehrere Verarbeitungspfade weiterleitet.

**Virtualisierung.** Ein Verfahren, das die Merkmale von Ressourcen auf der Basis der Art und Weise kapselt, in der andere Systeme mit diesen Ressourcen interagieren.

**Virtuelle Maschine.** Eine abstrakte Spezifikation einer Datenverarbeitungseinheit, die auf verschiedene Arten in der Software und Hardware implementiert sein kann.

**Virtueller Host.** Eine Konfiguration, mit der ein einziges Hostsystem mehreren Hostsystemen ähneln kann. Ressourcen, die einem virtuellen Host zugeordnet wurden, können nicht Daten mit Ressourcen gemeinsam nutzen, die einem anderen virtuellen Host zugeordnet wurden, auch wenn sich die virtuellen Hosts auf demselben physischen System befinden.

**Vitalität.** Der allgemeine Zustand bzw. Status der Datenbankumgebung.

**Vorgeordnet.** Dieser Begriff bezeichnet die Richtung des Bearbeitungsablaufs. Dieser verläuft vom Start des Prozesses (vorgeordnet) zum Ende des Prozesses (nachgeordnet).

**Vorläufige Programmkorrektur (Program Temporary Fix).** Bei den Produktreihen IBM System i, IBM System p und IBM System z eine Programmkorrektur, die von IBM getestet wurde und allen Kunden zur Verfügung gestellt wird. Siehe auch Fixpack.

**Vorläufiger Fix.** Ein zertifizierter Fix, der für alle Kunden zwischen regulär geplanten Fixpacks, Refresh-Packs oder Releases zur Verfügung gestellt wird. Siehe auch Fixpack.

**Waiter.** Ein Thread, der auf eine Verbindung wartet.

**WAR .** Siehe Webarchiv.

**Wartungsmodus.** Ein Status eines Knotens oder Servers, den Administratoren für dessen Diagnose, Wartung oder Optimierung verwenden können, ohne dass eingehender Datenverkehr in einer Produktionsumgebung unterbrochen wird.

**WCCM .** Siehe WebSphere Common Configuration Model.

**Webarchiv (WAR).** Ein komprimiertes Dateiformat, das vom Java EE-Standard definiert wird und mit dem alle Ressourcen, die zur Installation und Ausführung einer Webanwendung erforderlich sind, in einer einzigen Datei gespeichert werden können. Siehe auch Unternehmensarchiv.

**Webbrowser.** Ein Clientprogramm, das Anforderungen an einen Webserver einleitet und die Informationen anzeigt, die der Server zurückgibt.

**Webcontainer.** Ein Container, der den Webkomponentenvertrag der Java EE-Architektur implementiert. (Sun)

**Webcontainer-Channel.** Ein Typ von Channel in einer Transportkette, der eine Brücke in der Transportkette zwischen einem eingehenden HTTP-Channel und einer Servlet- oder JSP-Steuerkomponente erstellt.

**Web-Crawler.** Ein Crawler-Typ, der das Web durch Abrufen eines Webdokuments und Verfolgen der Links in diesem Dokument erkundet.

**Webkomponente.** Ein Servlet, eine JSP-Datei oder eine HTML-Datei. Eine oder mehrere Webkomponenten bilden ein Webmodul.

**Webserver.** Ein Softwareprogramm, das HTTP-Anforderungen verarbeiten kann.

**Webserver-Plug-in.** Ein Softwaremodul, das den Webserver bei der Weiterleitung von Anforderungen für dynamischen Inhalt, wie z. B. Servlets, an den Anwendungsserver unterstützt.

**Website.** Eine zusammengehörige Sammlung von im Web verfügbaren Dateien, die von einer einzigen Entität (einer Organisation oder einer Einzelperson) verwaltet wird und Hypertextinformationen für die Benutzer enthält. Eine Website enthält häufig Hypertext-Links zu anderen Websites.

**WebSphere.** Eine IBM Marke, die Tools für die Entwicklung von e-business-Anwendungen und Middleware für die Ausführung von Webanwendungen enthält.

**WebSphere Common Configuration Model (WCCM) .** Ein Modell, das programmgestützten Zugriff auf Konfigurationsdaten bietet.

**What You See Is What You Get (WYSIWYG).** Eine Funktionalität eines Editors, mit der Seiten ständig genau so angezeigt werden, wie sie gedruckt bzw. auf andere Weise wiedergegeben werden.

**While-Schleife.** Eine Schleife, die die gleiche Aktivitätenfolge so lange wiederholt, bis eine bestimmte Bedingung erfüllt ist. Die While-Schleife testet die zugehörige Bedingung zu Beginn jeder Schleife. Wenn die Bedingung von Beginn an nicht zutrifft, wird die Aktivitätenfolge in der Schleife nie ausgeführt.

**WLM.** Siehe Workload Manager.

**Workload-Management.** Die Optimierung der Verteilung eingehender Verarbeitungsaufträge an die Anwendungsserver, Enterprise-Beans, Servlets und anderen Objekte, die den jeweiligen Auftrag effektiv verarbeiten können.

**Workload Manager (WLM).** Eine Komponente von z/OS, mit der mehrere Workloads gleichzeitig in einem einzigen z/OS-Image oder über mehrere Images hinweg ausgeführt werden können.

**Write-Behind-Cache.** Ein Cache, der jede Schreiboperation asynchron über ein Ladeprogramm in die Datenbank schreibt.

**Write-Through-Cache.** (Durchschreibcache) Ein Cache, der jede Schreiboperation synchron über ein Ladeprogramm in die Datenbank schreibt.

**WYSIWYG.** Siehe What You See Is What You Get.

**XA.** Eine bidirektionale Schnittstelle zwischen einem oder mehreren Ressourcenmanagern, die den Zugriff auf gemeinsam genutzte Ressourcen ermöglichen, und einem Transaktionsmanager, der Transaktionen überwacht und auflöst.

**XML.** Siehe Extensible Markup Language.

**X/Open XA.** Die Schnittstelle X/Open Distributed Transaction Processing XA. Ein empfohlener Standard für die Kommunikation bei verteilten Transaktionen. Der Standard definiert eine bidirektionale Schnittstelle zwischen Ressourcenmanager, die den Zugriff auf gemeinsam genutzte Ressourcen in Transaktionen ermöglichen, und einem Transaktionsservice, der Transaktionen überwacht und auflöst.

**Zeichenfolge.** In Programmiersprachen das Datenformat, das für das Speichern und Bearbeiten von Text verwendet wird.

**Zeitbindung.** Eine angepasste Validierungsaktion, die zum Messen der Dauer eines Methodenaufrufs oder einer Folge von Methodenaufrufen verwendet wird.

**Zeitgeber.** Eine Task, die zu bestimmten Zeitpunkten eine Ausgabe generiert.

**Zeitlimit.** Ein Zeitintervall, das zugeordnet wird, damit ein Ereignis stattfinden oder beendet werden kann, bevor die Operation unterbrochen wird.

**Zelle.**

1. Eine Gruppe verwalteter Prozesse, die in denselben Deployment Manager eingebunden sind und Stammgruppen mit hoher Verfügbarkeit umfassen können.
2. Mindestens ein Prozess, der als Host für Laufzeitkomponenten eingesetzt wird. Jede Zelle hat mindestens eine benannte Stammgruppe.

**Zertifikatssammelspeicher.** Eine Gruppe vorläufiger Zertifikate oder Zertifikatwiderruflisten (CRL, Certificate Revocation List), die von einem Zertifikatpfad verwendet wird, um eine Zertifizierungskette für die Validierung zu erstellen.

**Zonenbasierte Unterstützung.** Eine Funktion, die die regelbasierte Positionierung von Shards ermöglicht, um die Grid-Verfügbarkeit durch Positionierung der Shards in unterschiedlichen Rechenzentren (auf verschiedenen Stockwerken oder sogar in verschiedenen Gebäuden oder Regionen) zu verbessern.

**z/OS.** Ein IBM Betriebssystem für Großrechner, bei dem 64-Bit-Realspeicher verwendet wird.



---

## Bemerkungen

Hinweise auf IBM Produkte, Programme und Services in dieser Veröffentlichung bedeuten nicht, dass IBM diese in allen Ländern, in denen IBM vertreten ist, anbietet. Hinweise auf IBM Lizenzprogramme oder andere IBM Produkte bedeuten nicht, dass nur Programme, Produkte oder Services von IBM verwendet werden können. An Stelle der IBM Produkte, Programme oder Services können auch andere ihnen äquivalente Produkte, Programme oder Services verwendet werden, solange diese keine gewerblichen oder anderen Schutzrechte der IBM verletzen. Die Verantwortung für den Betrieb der Produkte, Programme oder Services in Verbindung mit Fremdprodukten und Fremdservices liegt beim Kunden, soweit solche Verbindungen nicht ausdrücklich von IBM bestätigt sind.

Für in diesem Dokument beschriebene Erzeugnisse und Verfahren kann es IBM Patente oder Patentanmeldungen geben. Mit der Auslieferung dieses Handbuchs ist keine Lizenzierung dieser Patente verbunden. Lizenzanforderungen sind schriftlich an folgende Adresse zu richten (Anfragen an diese Adresse müssen auf Englisch formuliert werden):

IBM Director of Licensing  
IBM Europe, Middle East & Africa  
Tour Descartes  
2, avenue Gambetta  
92066 Paris La Defense  
France

Lizenznehmer des Programms, die Informationen zu diesem Produkt wünschen mit der Zielsetzung: (i) den Austausch von Informationen zwischen unabhängigen, erstellten Programmen und anderen Programmen (einschließlich des vorliegenden Programms) sowie (ii) die gemeinsame Nutzung der ausgetauschten Informationen zu ermöglichen, wenden sich an folgende Adresse:

IBM Corporation  
Mail Station P300  
522 South Road  
Poughkeepsie, NY 12601-5400  
USA  
Attention: Information Requests

Die Bereitstellung dieser Informationen kann unter Umständen von bestimmten Bedingungen - in einigen Fällen auch von der Zahlung einer Gebühr - abhängig sein.





---

## Marken

Folgende Namen sind Marken der IBM Corporation in den USA und/oder anderen Ländern:

- AIX
- CICS
- Cloudscape
- DB2
- Domino
- IBM
- Lotus
- RACF
- Redbooks
- Tivoli
- WebSphere
- z/OS

Java und alle auf Java basierenden Marken und Logos sind Marken von Sun Microsystems, Inc. in den USA und/oder anderen Ländern.

LINUX ist eine Marke von Linus Torvalds in den USA und/oder anderen Ländern.

Microsoft, Windows, Windows NT und das Windows-Logo sind Marken der Microsoft Corporation in den USA und/oder anderen Ländern.

UNIX ist eine eingetragene Marke von The Open Group in den USA und anderen Ländern.

Weitere Unternehmens-, Produkt- oder Servicenamen können Marken anderer Hersteller sein.



---

# Index

## A

- Abfrage 127
  - Abfrageplan 110, 287
  - Backus Naur 107
  - Beispiel 97
  - BNF 107
  - Clientfehler 74
  - Elemente suchen 86
  - Entität
    - Ergebnisse abrufen 94
  - Funktionen 99
  - gültige Attribute 92
  - Index 97, 113, 280
  - Klauseln 99
  - Methoden 86
  - ObjectMap
    - Schema 90
  - ObjectQuery-Schema 92
  - optimieren
    - Indizes 109, 279
    - Parameter 109, 279
    - Seitenaufteilung 109, 279
  - Optimierung
    - Beziehungen 113, 280
  - Parameter 97
  - Plan abrufen 110, 287
  - Prädikate 99
  - Schema 92
  - Schlüsselkollision 74
  - Seitenaufteilung 97
  - Warteschlange
    - alle Partitionen 74
    - Entitäten in einer Schleife 74
- Aktualisierbare Sperre 30
- Anwendungsprogrammierschnittstelle "EntityManager" 54
- API 65
- API-Dokumentation 135
- Ausnahmebehandlung 43

## B

- Berechtigung 130, 233
- Bereinigungsprogramm 176, 263
- Bereinigungsprogramme (Evictor) 261
- Bereinigungsprogramme schreiben
  - RollBackEvictor 265
- Bewährte Verfahren 270

## C

- Client 18
- Containerserver
  - Protokolle aktivieren 291
  - Trace aktivieren 291
- CopyMode 253

## D

- DataGrid-API 129, 130
- Daten 9
- Datenzugriff
  - Abfragen 9
  - gespeicherte Daten 9
  - Partitionen 9
  - Transaktionen 9
- Deadlocks
  - Szenarios 30
- Dynamische Maps
  - Maps 48

## E

- Entität 58
  - Lebenszyklen 79
- Entitäts-Listener 81, 84
- Entitäts-Maps
  - erstellen 166
- Entitätsmetadaten
  - Datei "emd.xsd" 63
  - XML-Konfiguration 63
- Entitätsschema
  - Entität 58
- EntityManager 71, 97
- EntityManager-API 57
- Ereignis-Listener 139
- Erweiterungs-Beans 204
- Exklusive Sperre 30
- Externer Transaktionsmanager 153

## F

- Fehlerbehebung
  - Nachrichten 295
  - Releaseinformationen 295
  - Übersicht 291
- FIFO-Warteschlangen
  - Maps 51

## G

- Gemeinsame Sperre 30
- Grid-Berechtigung 240

## H

- Heap-Speicher 270

## I

- Index
  - Callback 123
  - Datenqualität 121
  - Datenzugriff 123
  - Leistung 121
  - ohne Schlüssel 123

- Indexierung

- Hash-Index 127
  - zusammengesetzter Index 127
- Instrumentierungsagent 69
- Isolation
  - für Transaktionen 40
  - Pessimistisches Sperren 40
  - wiederholbares Lesen 40

## J

- Java Authentication and Authorization Service
  - JAAS 241
- Java Persistence API (JPA)
  - clientbasiertes Preload-Dienstprogramm
    - Programmierung 181
  - JPAEntityLoader-Plug-in
    - Einführung 164
  - mit eXtreme Scale verwenden
    - Übersicht 179
  - Preload-Dienstprogramm
    - Übersicht 180
  - zeitbasierte Aktualisierungskomponente
    - starten 190
  - Zeitbasierte Datenaktualisierungskomponente
    - Übersicht 188
- JVM 251

## K

- Katalogserver
  - Protokolle aktivieren 291
  - Trace aktivieren 291
- Konfigurieren 18

## L

- Lebenszyklus der Entität 81
- Leistung 251, 270
  - bewährte Verfahren 272
  - Sperren 272
- Listener
  - Einführung 139
  - für BackingMap-Objekte 139
  - für eXtreme Scale 139
  - MapEventListener-Plug-in 140
  - ObjectGridEventListener 141
  - ObjectGridEventListener-Plug-in 141
- Loader 176
  - Hinweise zur Programmierung von JPA 162
  - mit Entitäts-Maps und Tupeln verwenden 166
  - schreiben 158
  - Übersicht 156
- LogElement 176

LogSequence 176

## M

Maps für Bytefeldgruppen 258  
Methode "batchUpdate" 166  
Methode "get" 166

## N

Nachrichten 295  
Native Transaktionen 204

## O

ObjectGridManager 12  
ObjectMap-API  
  API 44  
  ObjectMap-API 44  
ObjectTransformer  
  bewährte Verfahren 276

## P

Plug-in  
  Einführung 137  
  Index 143  
  ObjectTransformer-Plug-in 197  
  OptimisticCallback 193  
  Plug-in-Slots 151  
  TransactionCallback 146  
  WebSphereTransactionCallback-Plug-in 202  
Programmierung von eXtreme Scale 7  
Protokolle  
  Übersicht 291  
Protokollelement 176  
Protokollfolge 176

## R

Releaseinformationen 295  
removeObjectGrid-Methoden 17

## S

Schnittstelle "EntityManager"  
  Leistung 67  
Schnittstelle "EntityTransaction" 78  
Schnittstelle "JavaMap" 51  
Schnittstelle "ObjectGridManager"  
  Lebenszyklus steuern 23  
  Trace aktivieren 291  
Schnittstelle "ObjectMap" 44  
Serialisierung  
  Leistung 277  
  Sperrern 277  
Server starten  
  programmgesteuert 247  
Server stoppen  
  programmgesteuert 247  
SessionHandle  
  Routing 42  
Sicherheit 130

Sicherheit (*Forts.*)  
  lokal 241  
  Plug-ins 241  
Sicherheits-API 213  
Sitzung  
  Kollision 43  
  Transaktion 43  
Sitzungen  
  auf Daten zugreifen  
    Flush 25  
    Grid 25  
Sperrern  
  Kompatibilität 30  
  Lebenszyklus 30  
  Zeitlimit 30  
Sperrern von Map-Einträgen  
  Abfrage 273  
  Indizes 273  
Spring 204  
  Erweiterungs-Beans 203  
  Framework 203  
  Geltungsbereich "Shard" 203  
  native Transaktionen 203  
  packen 203  
  Unterstützung von Namespaces 203  
  Web Flow 203  
Sprint-Erweiterungs-Beans 208  
System-API 137

## T

TimeToLive 263  
Trace  
  Konfigurationsoptionen 293  
  Übersicht 291  
Transaktion 153  
TTL-Bereinigungsprogramm 261  
Tupelobjekte  
  erstellen 166

## U

Unterstützung 295

## V

Verwaltungs-API 247  
Vorheriges Laden von Replikaten 172

## W

Warteschlangen 270  
Web Flow 204

## Z

Zugreifen auf 9





Gedruckt in Deutschland