WebSphere® eXtreme Scale Version 7.1
Produktübersicht

WebSphere eXtreme Scale Produktübersicht

Diese Ausgabe bezieht sich auf Version 7, Release 1 von WebSphere eXtreme Scale und, sofern in neuen Ausgaben nicht anders angegeben, auf alle nachfolgenden Releases dieses Produkts.

Diese Veröffentlichung ist eine Übersetzung des Handbuchs
IBM WebSphere eXtreme Scale Version 7.1 Product Overview,
herausgegeben von International Business Machines Corporation, USA

Informationen, die nur für bestimmte Länder Gültigkeit haben und für Deutschland, Österreich und die Schweiz nicht zutreffen, wurden in dieser Veröffentlichung im Originaltext übernommen.

Möglicherweise sind nicht alle in dieser Übersetzung aufgeführten Produkte in Deutschland angekündigt und verfügbar; vor Entscheidungen empfiehlt sich der Kontakt mit der zuständigen IBM Geschäftsstelle.

Änderung des Textes bleibt vorbehalten.

Herausgegeben von:
SW TSC Germany
Kst. 2877
August 2010

Inhaltsverzeichnis

Abbildungsverzeichnis

Tabellen

Informationen zur *Produktübersicht*

Der Dokumentationssatz zu WebSphere eXtreme Scale umfasst drei Handbücher, die die erforderlichen Informationen zur Verwendung des Produkts WebSphere eXtreme Scale, zur Programmierung für das Produkt und zur 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 Anwendungsfallszenarien 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 Buch ist für alle Benutzer bestimmt, die sich über das Produkt WebSphere eXtreme Scale informieren möchten.

Aktualisierungen für dieses Handbuch

Sie erhalten Aktualisierungen zu diesem Handbuch, indem Sie die jeweils aktuelle Version des Handbuchs 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.

Kapitel 1. Übersicht über WebSphere eXtreme Scale

WebSphere eXtreme Scale ist ein elastisches, skalierbares speicherinternes Daten-Grid. Das Produkt übernimmt folgende Aufgaben: dynamische Zwischenspeicherung, Partitionierung, Replikation und Verwaltung von Anwendungsdaten und Geschäftslogik in mehreren Servern. WebSphere eXtreme Scale unterstützt die Verarbeitung hoher Transaktionsaufkommen mit hoher Effizienz und linearer Skalierbarkeit. Außerdem können Sie mit WebSphere eXtreme Scale Servicequalitäten wie Transaktionsintegrität, hohe Verfügbarkeit und voraussagbare Antwortzeiten erreichen.

Die elastische Skalierbarkeit wird durch die Verwendung eines verteilten Objekt-Cachings erreicht. Mit elastischer Skalierbarkeit überwacht und verwaltet das Daten-Grid sich selbst. Die Topologie kann durch Hinzufügen und Entfernen von Servern horizontal in beide Richtungen skaliert werden, um sie so nach Bedarf an eine höhere bzw. geringere erforderliche Speicherkapazität, einen höheren oder geringeren erforderlichen Netzdurchsatz oder eine höhere bzw. geringere erforderliche Verarbeitungskapazität anzupassen. Diese Skalierungen werden auch als "Scale-out" und "Scale-in" bezeichnet. Beim Einleiten eines Scale-out-Prozesses wird dem aktiven Daten-Grid Kapazität hinzugefügt, ohne dass ein Neustart erforderlich ist. Beim Scale-in-Prozess werden Kapazitäten sofort entfernt. Außerdem ist das Daten-Grid selbst-heilend, d. h., es kann sich nach Fehlern automatisch wiederherstellen.

WebSphere eXtreme Scale kann auf verschiedene Arten eingesetzt werden. Das Produkt kann als extrem leistungsfähiger Cache oder als speicherinterner Datenbankverarbeitungsbereich für die Verwaltung des Anwendungsstatus oder als Plattform für die Erstellung leistungsfähiger XTP-Anwendungen (Extreme Transaction Processing) verwendet werden.

Sie müssen jedoch unbedingt beachten, dass eXtreme Scale nicht als echte speicherinterne Datenbank betrachtet werden kann, was zum größten Teil darauf zurückzuführen ist, dass eine echte speicherinterne Datenbank zu einfach ist, um die Komplexität handhaben zu können, die eXtreme Scale unterstützt. Sie haben in beiden Szenarien ein paar identische Vorteile, weil beide speicherintern sind, aber wenn die Maschine mit der speicherinternen Datenbank ausfällt, kann dieses Problem nicht sofort behoben werden. Ein solches Ereignis kann katastrophale Auswirkungen haben, wenn sich Ihre gesamte Umgebung auf dieser Maschine befindet.

Zur Vermeidung dieser Art von Fehlern teilt eXtreme Scale die jeweilige Datenmenge in Partitionen auf, die Baumstrukturschemas mit Integritätsbedingungen entsprechen. Baumstrukturschemas mit Integritätsbedingungen beschreiben die Beziehung zwischen Entitäten. Wenn Sie Partitionen verwenden, müssen die Entitätsbeziehungen eine Baumdatenstruktur modellieren, in der die Stammentität ganz oben steht und die einzige Entität ist, die partitioniert ist. Alle untergeordneten Entitäten der Stammentität sind in derselben Partition wie die Stammentität gespeichert. Jede Partition ist als primäre Kopie oder Shard vorhanden. Eine Partition enthält auch Replikat-Shards für die Sicherung der Daten. Eine speicherinterne Datenbank kann diese Art von Funktionalität nicht bereitstellen, weil sie nicht auf diese Weise strukturiert und nicht dynamisch ist, was erfordert, dass Sie die Operationen, die eXtreme Scale automatisch ausführt, manuell ausführen. Da eine speicherinterne Datenbank eine Datenbank ist, kann sie auch SQL-Operationen zulassen und bietet im Vergleich mit Datenbanken, die nicht speicherintern sind, eine wesentlich höhere Verarbeitungsgeschwindigkeit. WebSphere eXtreme Scale ver-

wendet anstelle der SQL-Unterstützung eine eigene Abfragesprache, ist aber wesentlich elastischer, unterstützt die Partitionierung von Daten und bietet eine zuverlässige Wiederherstellung nach Fehlern.

Mit dem Write-behind-Cachefeature kann WebSphere eXtreme Scale als Front-End-Cache für eine Datenbank eingesetzt werden. Durch die Verwendung dieses Front-End-Caches können Sie den Durchsatz erhöhen und die Datenbanklast und Konkurrenzsituationen in der Datenbank verringern. WebSphere eXtreme Scale ermöglicht eine horizontale Skalierung einer Topologie (Scale-out und Scale-in) zu vorhersagbaren Verarbeitungskosten.

Die folgende Abbildung zeigt eine verteilte kohärente Cacheumgebung, in der die eXtreme-Scale-Clients Daten an das Daten-Grid senden und Daten aus dem Daten-Grid empfangen, die automatisch mit einem Back-End-Datenspeicher synchronisiert werden können. Der Cache ist kohärent, weil alle Clients dieselben Daten im Cache sehen. Jede einzelne Information wird im Cache eines einzigen beschreibbaren Servers gespeichert. Auf diese Weise werden unnötige Datensatzkopien verhindert, die potenziell verschiedene Versionen der Daten enthalten könnten. Ein kohärenter Cache nimmt immer mehr Daten auf, je mehr Server dem Daten-Grid hinzugefügt werden. Die Skalierung des Caches erfolgt linear zum Wachstum des Daten-Grids. Die Daten können für zusätzliche Fehlertoleranz auch repliziert werden.

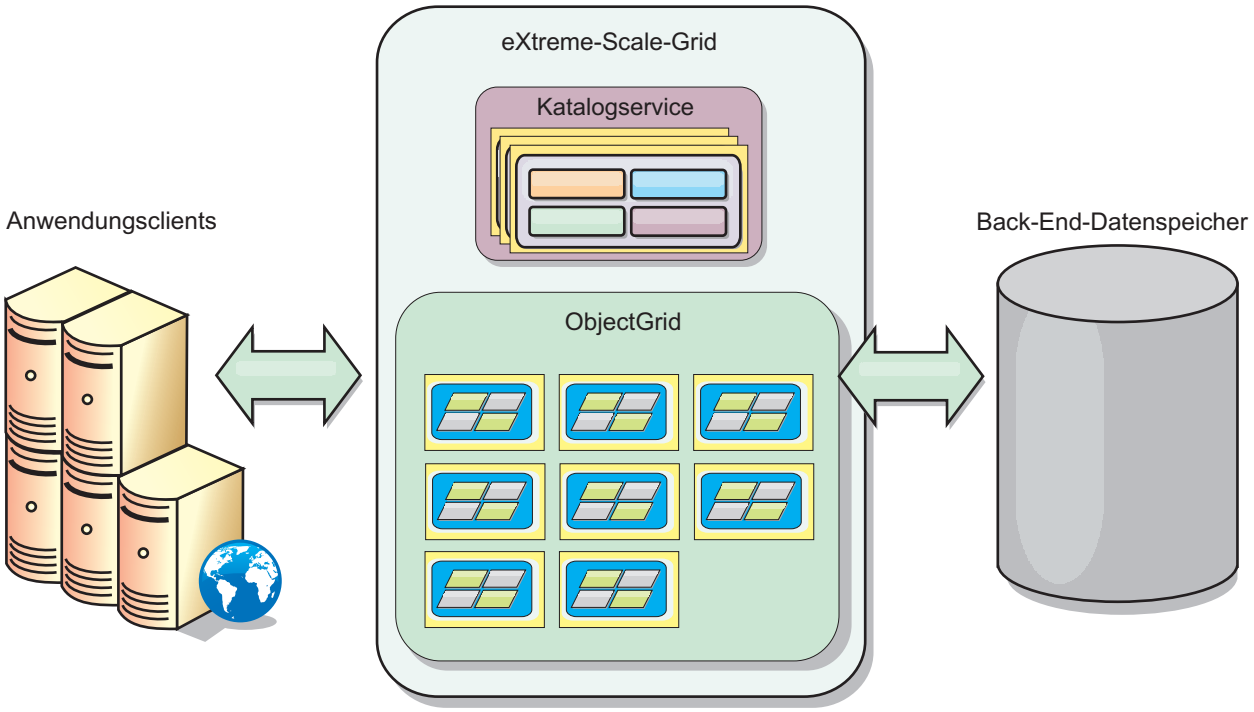


Abbildung 1. Übergeordnete Topologie

WebSphere eXtreme Scale hat Server, die das speicherinterne Daten-Grid bereitstellen. Diese Server können in WebSphere Application Server oder in einer einfachen J2SE-JVM (Java™ Standard Edition) ausgeführt werden, was die Unterstützung mehrerer dieser physischen Maschinen zulässt. Deshalb kann das speicherinterne Daten-Grid relativ groß sein. Das Daten-Grid wird weder durch den Hauptspeicher oder Adressraum der Anwendung bzw. des Anwendungsservers beschränkt noch

hat es Einfluss auf diesen. Der Hauptspeicher kann die Summe mehrerer Hundert oder Tausend Java Virtual Machines sein, die auf vielen verschiedenen Maschinen ausgeführt werden.

Als speicherinterner Datenbankverarbeitungsbereich kann WebSphere eXtreme Scale trotzdem durch eine Platte, eine Datenbank oder beides gestützt werden.

eXtreme Scale stellt viele Java-APIs bereit. Viele APIs erfordern keine Programmierung durch den Benutzer, sondern müssen nur in Ihrer WebSphere-Infrastruktur konfiguriert und implementiert werden.

Basisparadigma

Das grundlegende Daten-Grid-Paradigma ist ein Schlüssel/Wert-Paar, in dem das Daten-Grid Werte (Java-Objekte) mit einem zugehörigen Schlüssel (einem weiteren Java-Objekt) speichert. Der Schlüssel wird später zum Abrufen des Werts verwendet. In eXtreme Scale setzt sich eine Map aus Einträgen solcher Schlüssel/Wert-Paare zusammen.

WebSphere eXtreme Scale bietet verschiedene Daten-Grid-Konfigurationen, angefangen von einem einzigen einfachen lokalen Cache bis hin zu einem großen verteilten Cache, mit mehreren JVMs und/oder Servern.

Zusätzlich zu einfachen Java-Objekten können Sie Objekte mit Beziehungen speichern. Zum Abrufen dieser Objekte können Sie eine Abfragesprache, die SQL gleicht, mit `SELECT-, FROM-, und WHERE`-Klauseln verwenden. Ein Bestellobjekt (Order) kann beispielsweise ein Kundenobjekt (Customer) und mehrere zugehörige Artikelobjekte (Item) haben. WebSphere eXtreme Scale unterstützt 1:1-, 1:n- und n:n-Beziehungen.

WebSphere eXtreme Scale unterstützt auch eine EntityManager-Programmierschnittstelle für das Speichern von Entitäten im Cache. Diese Programmierschnittstelle gleicht Java-Enterprise-Edition-Entitäten. Entitätsbeziehungen können automatisch über eine XML-Entitätsdeskriptordatei oder über Annotationen in Java-Klassen erkannt werden. Eine Entität kann somit anhand des Primärschlüssels mit der Methode "find" in der Schnittstelle "EntityManager" aus dem Cache abgerufen werden. Entitäten können innerhalb von Transaktionsgrenzen persistent im Daten-Grid festgeschrieben und aus diesem entfernt werden.

WebSphere eXtreme Scale stellt XTP-Funktionen (Extreme Transaction Processing) bereit, die eine intelligentere Anwendungsinfrastruktur für die Unterstützung anspruchsvoller geschäftskritischer Anwendungen sicherstellen. Sie können herkömmliche IT-Leistungsbeschränkungen umgehen, um die Stufen von globaler Reichweite, Prozesseffektivität und Informationsmanagement zu erreichen, die für intelligentere Ergebnisse und nachhaltige Wettbewerbsvorteile in Bezug auf das Geschäft erforderlich sind.

Mit der Unterstützung für WebSphere Real Time, dem branchenführenden Java-Angebot für die Echtzeitverarbeitung, ermöglicht WebSphere eXtreme Scale XTP-Anwendungen konsistentere und voraussagbare Antwortzeiten. Sehen Sie sich die Informationen zur Unterstützung von Real Time in der Veröffentlichung *Administratorhandbuch* an.

Vor der Implementierung von eXtreme Scale in einer Produktionsumgebung müssen verschiedene Optionen berücksichtigt werden, z. B. die Anzahl der zu verwendenden Server, die Speicherkapazität jedes Servers und die synchrone oder asynchrone Replikation.

Stellen Sie sich eine verteilte Umgebung vor, in der der Schlüssel ein einfacher alphabetischer Name ist. Der Cache kann nach Schlüsseln in 4 Partitionen aufgeteilt werden: Partition 1 für Schlüssel, die mit A-E beginnen, Partition 2 für Schlüssel, die mit F-L beginnen usw. Für die Verfügbarkeit besitzt eine Partition ein primäres Shard und ein Replikat-Shard. Änderungen an den Cachedaten werden am primären Shard vorgenommen und im sekundären Shard repliziert. Für einen verteilten Cache (oder ein Daten-Grid bzw. ObjectGrid im eXtreme-Scale-Vokabular) konfigurieren Sie die Anzahl der eXtreme-Scale-Server, die die Grid-Daten enthalten, und eXtreme Scale verteilt die Daten in Shards auf diese Serverinstanzen. Für die Verfügbarkeit werden Replikat-Shards auf anderen Maschinen als die primären Shards gespeichert.

WebSphere eXtreme Scale verwendet einen Katalogservice, um das primäre Shard für jeden Schlüssel zu finden. Das Produkt verschiebt Shards zwischen eXtreme-Scale-Servern, wenn Server oder die übergeordneten physischen Maschinen dieser Server ausfallen und anschließend wiederhergestellt werden. Wenn beispielsweise der Server, der ein Replikat-Shards enthält, ausfällt, ordnet eXtreme Scale ein neues Replikat-Shard zu. Falls ein Server, der ein primäres Shard enthält, ausfällt, wird das Replikat-Shard auf ein primäres Shard hochstuft, und wie zuvor wird ein neues Replikat-Shard erstellt.

Die einfachste Programmierschnittstelle von eXtreme Scale ist "ObjectMap", die eine einfache Map-Schnittstelle ist. Sie enthält eine Methode "map.put(key,value)" zum Speichern eines Werts im Cache und eine Methode "map.get(key)" für den anschließenden Abruf des Werts.

Eine Beschreibung der bewährten Verfahren für das Entwerfen von eXtreme-Scale-Anwendungen finden Sie im folgenden Artikel auf developerWorks: Principles and best practices for building high performing and highly resilient WebSphere eXtreme Scale applications.

Neue und veraltete Features in diesem Release

WebSphere eXtreme Scale enthält zahlreiche neue Features in Version 7.1, unter anderem die Integration des dynamischen Caches, Maps für Bytefeldgruppen, und vieles mehr.

Neues in WebSphere eXtreme Scale Version 7.1

Tabelle 1. Neue Features in WebSphere eXtreme Scale Version 7.1

Feature	Beschreibung
Integration von DB2-Client-informationen	Sie können JPA-Loader-Plug-ins von eXtreme-Scale mit DB2 integrieren, so dass bei der Verwendung von DB2 als Back-End-Datenbank die eXtreme-Scale-Informationen (Benutzername, Workstationname, Anwendungsname und Abrechnungsinformationen) in DB2 Performance Monitor bereitgestellt werden können. Dieses Feature ermöglicht das Aktivieren und Inaktivieren der Konfiguration von Clientinformationen in DB2. Diese Funktion ist standardmäßig inaktiviert. Weitere Informationen finden Sie unter „Loader" auf Seite 43.
Konfiguration von Katalog-servicedomänen	Katalogservicedomänen können über die Administrationskonsole von WebSphere Application Server oder mithilfe von Verwaltungs-Tasks konfiguriert werden. Katalogservicedomänen definieren eine Gruppe. Weitere Informationen finden Sie in der Dokumentation zum Erstellen von Katalogservicedomänen im *Administratorhandbuch*.
Multimaster-Replikation	Mehrere Rechenzentren können asynchron verlinkt werden, was den lokalen Rechenzentrumszugriff auf Daten und die Aufrechterhaltung einer hohen Verfügbarkeit ermöglicht. Weitere Informationen finden Sie unter „Multmaster-Replikationstopologie (AP)" auf Seite 23.

Tabelle 1. Neue Features in WebSphere eXtreme Scale Version 7.1 (Forts.)

Feature	Beschreibung
Letzte Aktualisierungszeit	Der TTL-Evictor wurde aktualisiert und überwacht jetzt die Zeit, zu der Einträge aktualisiert werden. Weitere Informationen finden Sie in der Dokumentation zum TTL-Evictor im *Programmierhandbuch*
usedBytes-Statistik für speicherinterne Grids	Der von Cacheeinträgen in einer BackingMap belegte Speicher kann mit allen Statistikprovidern verfolgt werden. Weitere Einzelheiten finden Sie in den Informationen zur Dimensionierung der Cachenutzung im *Administratorhandbuch*.
Dynamische Statistiken	Statistiken können jetzt bei Bedarf aktiviert und inaktiviert werden. Weitere Informationen finden Sie in der Dokumentation zur Überwachung mit MBeans im *Administratorhandbuch*.
Überwachungskonsole	Die grafische Überwachungskonsole stellt aktuelle und Langzeitsichten für eXtreme-Scale-Serverstatistiken bereit. Weitere Informationen finden Sie in der Dokumentation zur Webkonsole im *Administratorhandbuch*.
Verbesserter HTTP-Sitzungsmanager	Die Konfiguration des HTTP-Sitzungsmanagers wurde vereinfacht. Sie können den HTTP-Sitzungsmanager jetzt über die Administrationskonsole von WebSphere Application Server konfigurieren. Weitere Informationen finden in den Informationen zum Konfigurieren des HTTP-Sitzungsmanagers im *Administratorhandbuch*.
Unterstützung von Clients mit mehreren Netzschnittstellen	Clients können für die Verwendung eines bestimmten Netzadapters konfiguriert werden. Weitere Informationen finden Sie in den Informationen zur Clienteigenschaftendatei im *Administratorhandbuch*.
ISA Lite	IBM® Support Assistant Lite for WebSphere eXtreme Scale unterstützt die automatische Datenerfassung und Systemanalyse für Problembestimmungsszenarien. Weitere Informationen finden Sie in den Informationen zu IBM Support Assistant for WebSphere eXtreme Scale im *Administratorhandbuch*.
REST	Der REST-Datenservice ermöglicht Clients, die keine Java-Clients sind, den Zugriff auf eXtreme-Scale-Daten mit Unterstützung des Protokolls "Open Data Protocol" (OData) und vollständiger Kompatibilität mit Microsoft® WCF Data Services. Weitere Informationen finden Sie unter Kapitel 8, „Übersicht über REST-Datenservices", auf Seite 183.
Reine Clientinstallation	eXtreme-Scale-Clients können jetzt unabhängig installiert werden, was den Speicherbedarf für die Installation von eXtreme-Scale-Anwendungen verringert. Weitere Informationen finden Sie im den Informationen zum Installieren und Implementieren von WebSphere eXtreme Scale im *Administratorhandbuch*.

Veraltete Features

Tabelle 2. Veraltete Features

Veraltet	Empfohlene Migrationsaktion
	Erstellen Sie über die Administrationskonsole von WebSphere Application Server eine Katalogservicedomäne. Damit wird dieselbe Konfiguration wie mit der angepassten Eigenschaft erstellt. Weitere Informationen finden Sie in der Dokumentation zum Erstellen von Katalogservicedomänen im *Administratorhandbuch*.
	Verwenden Sie stattdessen CatalogServiceManagementMBean.
Partitionierungsfeature (WPF): Das Partitionierungsfeature ist eine Gruppe von Programmierungs-APIs, die Java-EE-Anwendungen die Unterstützung symmetrischer Cluster ermöglicht.	Die Funktionalität von WPF kann alternativ in WebSphere eXtreme Scale realisiert werden.
StreamQuery: Eine fortlaufende Abfrage unvollständiger Daten, die in ObjectGrid-Maps gespeichert sind.	Ohne
Konfiguration statischer Grids: Eine statische, clusterbasierte Topologie, die die XML-Datei für die Clusterimplementierung verwendet.	Ersetzt durch die verbesserte dynamische Implementierungstopologie für die Verwaltung großer Daten-Grids.

Tabelle 2. Veraltete Features (Forts.)

Veraltet	Empfohlene Migrationsaktion
Veraltete Systemeigenschaften: Systemeigenschaften für die Angabe der Server- und Clienteigenschaftendateien sind veraltet.	Sie können diese Argumente zwar noch verwenden, sollten Ihre Systemeigenschaften aber auf die neuen Werte umstellen. Weitere Informationen finden Sie in den Beschreibungen der Eigenschaftendateien im *Administratorhandbuch*.

Mit WebSphere eXtreme Scale arbeiten

WebSphere eXtreme Scale ist ein elastisches, skalierbares speicherinternes Daten-Grid. Das Produkt übernimmt folgende Aufgaben: dynamische Zwischenspeicherung, Partitionierung, Replikation und Verwaltung von Anwendungsdaten und Geschäftslogik in mehreren Servern.

Da eXtreme Scale keine speicherinterne Datenbank ist, müssen Sie die speziellen Konfigurationsanforderungen für eXtreme Scale berücksichtigen. Der erste Schritt bei der Implementierung eines eXtreme-Scale-Daten-Grids ist das Starten einer Stammgruppe und eines Katalogservice, der als Koordinator für alle anderen Java Virtual Machines (JVM) auftritt, die am Grid beteiligt sind, und die Konfigurationsdaten verwaltet. Die Prozesse von WebSphere eXtreme Scale werden mit einfachen Befehlsscriptaufrufen über die Befehlszeile gestartet.

Der nächste Schritt ist das Starten von eXtreme-Scale-Serverprozessen für das Grid, um Daten zu speichern und abzurufen. Wenn Server gestartet werden, registrieren sie sich automatisch bei der Stammgruppe und beim Katalogservice, was ihnen eine Zusammenarbeit bei der Bereitstellung der Grid-Services ermöglicht. Je mehr Server gestartet werden, desto höher sind Kapazität und Zuverlässigkeit des Grids.

Ein lokales Grid ist ein einfaches Grid mit einer Instanz, d. h., alle Daten werden in einem einzigen Grid gespeichert. Wenn Sie eXtreme Scale effektiv als speicherinternen Datenbankverarbeitungsbereich nutzen möchten, können Sie ein verteiltes Grid konfigurieren und implementieren. Die Daten im verteilten Grid werden so auf die verschiedenen eXtreme-Scale-Server verteilt, dass jeder Server einen Teil der Daten, eine so genannte Partition, enthält.

Ein wichtiger Konfigurationsparameter für verteilte Grids ist die Anzahl der Grid-Partitionen. Die Grid-Daten werden in diese Anzahl von Untergruppen partitioniert, und jede dieser Untergruppen wird als Partition bezeichnet. Der Katalogservice sucht die Partition für eine bestimmte Information anhand des Schlüssels. Die Anzahl der Partitionen wirkt sich direkt auf die Kapazität und die Skalierbarkeit des Grids aus. Ein Server kann eine oder mehrere Grid-Partitionen enthalten. Somit ist die Größe einer Partition auf die Hauptspeicherkapazität des Servers beschränkt. Mit zunehmender Anzahl an Partitionen erhöht sich die Kapazität des Grids. Die maximale Kapazität eines Grids entspricht der Anzahl an Partitionen, multipliziert mit der Größe des verfügbaren Speichers eines Servers (z. B. einer JVM).

Die Daten einer Partition werden in einem so genannten Shard gespeichert. Für die Verfügbarkeit kann ein Grid mit Replikaten konfiguriert werden, die synchron oder asynchron sein können. Änderungen an den Grid-Daten werden im primären Shard vorgenommen und in den Replikat-Shards repliziert. Der von einem Grid

belegte/erforderliche Gesamtspeicher kann mit Hilfe der folgenden Formel ermittelt werden: Größe des Grids mal (1 (für das primäre Shard) + Anzahl der Replikate).

WebSphere eXtreme Scale verteilt die Shards eines Grids auf die Server, die das Grid enthalten. Diese Server können sich auf derselben und/oder auf gesonderten physischen Maschinen befinden. Für die Verfügbarkeit werden Replikat-Shards auf anderen Maschinen als die primären Shards gespeichert.

WebSphere eXtreme Scale überwacht den Status seiner Server und verschiebt Shards zwischen den Servern, falls diese und/oder deren übergeordnete physische Maschinen ausfallen und anschließend wiederhergestellt werden. Wenn beispielsweise der Server, der ein Replikat-Shards enthält, ausfällt, ordnet eXtreme Scale ein neues Replikat-Shard zu und repliziert die Daten vom primären Shard im neuen Replikat. Falls ein Server, der ein primäres Shard enthält, ausfällt, wird das Replikat-Shard auf ein primäres Shard hochstuft, und wie zuvor wird ein neues Replikat-Shard erstellt. Wenn Sie einen weiteren Server für das Grid starten, werden die Shards und somit die Last möglichst gleichmäßig auf alle Server verteilt. Dies wird als horizontale Vorwärtsskalierung (oder Scale-out) bezeichnet. Für eine horizontale Rückskalierung (Scale-in) können Sie einen der Server stoppen, um die von einem Grid konsumierten Ressourcen zu verringern. Wie bei einer Ausfallsituation werden auch hier die Shards erneut gleichmäßig auf die verbleibenden Server verteilt.

Übersicht über die Anwendungsimplementierung

Vor der Verwendung von WebSphere eXtreme Scale in einer Produktionsumgebung sollten Sie sich die folgenden Punkte zur Optimierung der Implementierung ansehen.

Anwendungsimplementierung planen

Die folgende Liste enthält die zu beachtenden Punkte:

- Anzahl der Systeme und Prozessoren: Wie viele physische Maschinen und Prozessoren sind in der Umgebung erforderlich?
- Anzahl der Server: Wie viele eXtreme-Scale-Server sind für die Speicherung der eXtreme-Scale-Maps erforderlich?
- Anzahl der Partitionen: Das in den Maps gespeicherte Datenvolumen ist ein Faktor für die Bestimmung der Anzahl erforderlicher Partitionen.
- Anzahl der Replikate: Wie viele Replikate sind für jedes primäre Shard in der Domäne erforderlich?
- Synchrone oder asynchrone Replikation: Sind die Daten elementar, so dass eine synchrone Replikation erforderlich ist? Oder hat die Leistung eine höhere Priorität, so dass die asynchrone Replikation die richtige Wahl ist?
- Größe der Heap-Speicher: Welches Datenvolumen wird auf jedem Server gespeichert?

Integration mit anderen Produkten von WebSphere Application Server

Sie können WebSphere eXtreme Scale mit anderen Serverprodukten wie WebSphere Application Server und WebSphere Application Server Community Edition integrieren.

HTTP-Sitzungsmanager für die Zusammenarbeit mit WebSphere Application Server Community Edition konfigurieren

WebSphere Application Server Community Edition kann den Sitzungsstatus zwar zur gemeinsamen Nutzung bereitstellen, aber nicht auf effiziente und skalierbare Weise. WebSphere eXtreme Scale stellt eine verteilte Persistenzschicht mit hoher Leistung bereit, die zum Replizieren des Status verwendet werden kann, sich aber nicht problemlos mit Anwendungsservern außerhalb von WebSphere Application Server integrieren lässt. Sie können diese beiden Produkte integrieren, um eine skalierbare Lösung für das Sitzungsmanagement zu erhalten. Weitere Einzelheiten hierzu finden Sie im *Administratorhandbuch*.

Sitzungsmanager von WebSphere eXtreme Scale für die Zusammenarbeit mit WebSphere Application Server konfigurieren

Der HTTP-Sitzungsmanager war erstmalig im Lieferumfang von WebSphere Extended Deployment DataGrid Version 6.1.0.0 enthalten. In den nachfolgenden Versionen bis Version 6.1.0.5 wurden die Nutzungsmethoden nicht geändert, da sie der J2EE-Spezifikation (Java 2 Enterprise Edition) für die Integration und den Abruf von Sitzungen entsprechen, aber in jedem Release wurden Verbesserungen an der Leistung und den Servicequalitäten vorgenommen. Um sicherzustellen, dass Sie die beste Servicequalität erhalten, empfiehlt es sich, das Fixpack für WebSphere eXtreme Scale Version 6.1.0.5 anzuwenden.

Weitere Einzelheiten finden Sie im *Administratorhandbuch*.

Änderung des Produktnamens

Beachten Sie bitte, dass WebSphere eXtreme Scale früher unter anderen Namen bekannt war.

Änderung des Produktnamens

Wenn auf andere Dokumentationen, Marketing-Material oder Präsentationen verwiesen wird, müssen Sie beachten, dass eXtreme Scale früher unter den folgenden Namen bekannt war.

- ObjectGrid
- WebSphere Extended Deployment Data Grid

Obwohl das Produkt selbst jetzt unter dem Namen WebSphere eXtreme Scale geführt wird, kommt der Begriff "ObjectGrid" in dieser Dokumentation und anderen Referenzen weiterhin vor, weil es der Name des Artefakts ist, das die Grid-Technologie ermöglicht.

Kostenlose Testversion

Um sich mit der Verwendung von WebSphere eXtreme Scale vertraut zu machen, laden Sie eine kostenlose Testversion herunter. Sie können innovative Hochleistungsanwendungen entwickeln, indem Sie das Daten-Caching-Konzept mit erweiterten Features erweitern.

Probedownload

Sie können eine kostenlose Testversion von eXtreme Scale von der Downloadsite mit der Testversion von eXtreme Scale herunterladen.

Nach dem Download und Entpacken der Testversion von eXtreme Scale navigieren Sie zum Verzeichnis "gettingstarted" und lesen die Datei "GETTINGSTARTED-

_README.txt". Dieses Lernprogramm ist eine Einführung in die Verwendung von eXtreme Scale. Sie erstellen ein Grid in mehreren Servern und führen verschiedene einfache Anwendungen zum Speichern und Abrufen von Daten in einem Grid aus. Vor der Implementierung von eXtreme Scale in einer Produktionsumgebung müssen verschiedene Optionen berücksichtigt werden, z. B. die Anzahl der zu verwendenden Server, die Speicherkapazität jedes Servers und die synchrone oder asynchrone Replikation.

Programmierhandbuch und Administratorhandbuch

Im Handbuch *Produktübersicht* werden die grundlegenden Konzepte für das Verständnis von WebSphere eXtreme Scale beschrieben. Es sind zwei weitere Handbücher verfügbar, die weitere Erläuterungen der Konzepte enthalten, die in diesem Handbuch beschrieben sind.

Verwenden Sie das *Administratorhandbuch* für die Konfiguration und für allgemeine Verwaltungs-Tasks und das *Programmierhandbuch* für Beschreibungen der Java-APIs für den Zugriff auf und die Konfiguration des eXtreme-Scale-Grids.

Kapitel 2. Übersicht über Caching

WebSphere eXtreme Scale kann als speicherinterner Datenbankverarbeitungsbereich eingesetzt werden, den Sie verwenden können, um integriertes Caching für ein Datenbank-Back-End oder um einen Nebencache bereitzustellen. Beim integrierten Caching wird eXtreme Scale als primäres Mittel für die Interaktion mit den Daten verwendet. Wenn eXtreme Scale als Nebencache verwendet wird, wird das Back-End zusammen mit dem Daten-Grid verwendet. In diesem This section Abschnitt sind die verschiedenen Cachekonzepte und -szenarien sowie die verfügbaren Topologien für die Implementierung eines Daten-Grids beschrieben.

Caching-Architektur: Maps, Container, Clients und Kataloge

Mit WebSphere eXtreme Scale kann Ihre Architektur speicherinternes Daten-Caching oder verteiltes Client/Server-Daten-Caching verwenden.

WebSphere eXtreme Scale erfordert für den Betrieb eine minimale zusätzliche Infrastruktur. Die Infrastruktur setzt sich aus Scripts für das Installieren, Starten und Stoppen einer Java-EE-Anwendung auf einem Server zusammen. Die zwischengespeicherten Daten werden im Server von eXtreme Scale gespeichert, und Clients stellen über Fernzugriff eine Verbindung zum Server her.

Verteilte Caches bieten eine bessere Leistung, Verfügbarkeit und Skalierbarkeit und können unter Verwendung dynamischer Topologien konfiguriert werden, in denen Server automatisch verteilt werden. Zusätzliche Server können hinzugefügt werden, ohne die vorhandenen eXtreme Scale-Server erneut starten zu müssen. Sie können einfache Implementierungen erstellen oder große Implementierungen in Terabytegröße, in denen Tausende von Servern erforderlich sind.

Maps

Eine Map ist ein Container für Schlüssel/Wert-Paare, in der eine Anwendung einen Wert nach einem Schlüssel indexiert speichern kann. Maps unterstützen Indizes, die Indexattributen im Schlüssel oder Wert hinzugefügt werden können. Diese Indizes werden automatisch von der Abfragelaufzeitumgebung verwendet, um die effizienteste Methode für die Durchführung einer Abfrage zu bestimmen.

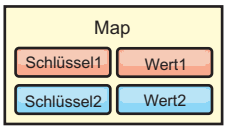


Abbildung 2. Map

Ein Mapset ist eine Sammlung von Maps mit einem gemeinsamen Partionierungsalgorithmus. Die Daten in den Maps werden auf der Basis der Richtlinie repliziert, die im Mapset definiert ist. Ein Mapset wird nur für verteilte Topologien verwendet und wird für lokale Topologien nicht benötigt.

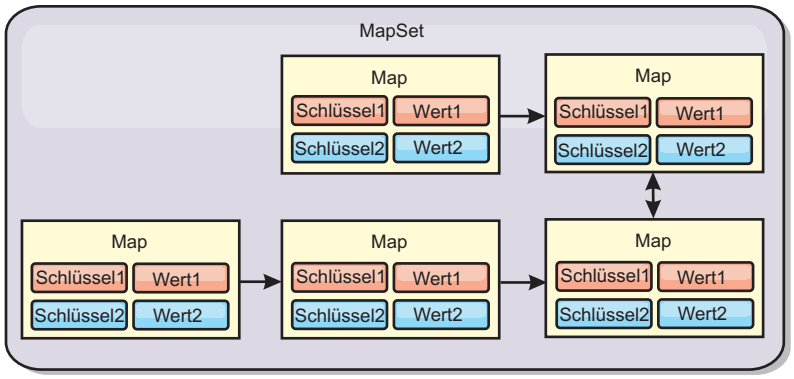


Abbildung 3. Mapsets

Einem Mapset kann ein Schema zugeordnet sein. Ein Schema setzt sich aus den Metadaten zusammen, die die Beziehungen zwischen den einzelnen Maps beschreiben, wenn homogene Objekttypen oder Entitäten verwendet werden.

WebSphere eXtreme Scale kann über die Anwendungsprogrammierschnittstelle (API, Application Programming Interface) "ObjectMap" serialisierbare Java-Objekte in jeder der Maps speichern. Es kann ein Schema für die Maps definiert werden, um die Beziehungen zwischen den Objekten in den Maps zu identifizieren, wobei jede Map Objekte eines einzelnen Typs enthält. Die Definition eines Schemas für Maps ist erforderlich, um den Inhalt der Map-Objekte abzufragen. In WebSphere eXtreme Scale können mehrere Map-Schemas definiert werden. Weitere Einzelheiten finden Sie in den Informationen zur API "ObjectMap" im *Programmierhandbuch*.

Außerdem kann WebSphere eXtreme Scale über die API "EntityManager" Entitäten speichern. Jede Entität ist einer Map zugeordnet. Das Schema für ein Entitäts-Mapset wird automatisch über eine XML-Entitätsdeskriptordatei oder über annotierte Java-Klassen erkannt. Jede Entität besitzt einen Satz von Schlüsselattributen und einen Satz von Attributen ohne Schlüsselfunktion. Eine Entität kann außerdem Beziehungen zu anderen Entitäten haben. WebSphere eXtreme Scale unterstützt 1:1-, 1:N-, N-1- und N:N-Beziehungen. Jede Entität ist physisch einer einzigen Map im Mapset zugeordnet. Entitäten ermöglichen Anwendungen die problemlose Verwendung komplexer Objektgraphen, die sich über mehrere Maps erstrecken. Eine verteilte Topologie kann mehrere Entitätsschemas haben. Weitere Einzelheiten finden Sie in den Informationen zur API "EntityManager" im *Programmierhandbuch*.

Container, Partitionen und Shards

Der Container ist ein Service, der Anwendungsdaten für das Grid speichert. Diese Daten werden gewöhnlich in Teile, so genannte Partitionen, aufgeteilt und auf mehrere Container verteilt. Jeder Container wiederum enthält einen Teil der vollständigen Daten. Eine JVM kann einen oder mehrere Container enthalten und jeder Container mehrere Shards.

Hinweis: Planen Sie die Größe des Heap-Speichers für die Container, in denen alle Ihre Daten enthalten sind. Konfigurieren Sie die Einstellungen für den Heap-Speicher entsprechend.

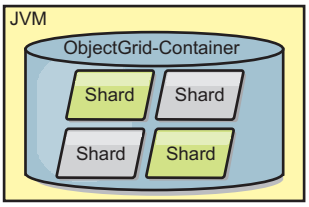


Abbildung 4. Container

Partitionen enthalten einen Teil der Daten des Grids. WebSphere eXtreme Scale stellt automatisch mehrere Partitionen in einen einzigen Container und verteilt die Partitionen dann breiter, wenn weitere Container verfügbar werden.

Wichtig: Sie müssen die Anzahl der Partitionen vor der endgültigen Implementierung sorgfältig auswählen, da sie nicht dynamisch geändert werden kann. Ein Hash-Mechanismus wird verwendet, um Partitionen im Netz zu suchen, und eXtreme Scale hat nach der Implementierung der Daten keine Möglichkeit, ein erneutes Hashing für die gesamten Daten durchzuführen. Als allgemeine Regel gilt, dass die Anzahl der Partitionen zu hoch geschätzt werden kann.

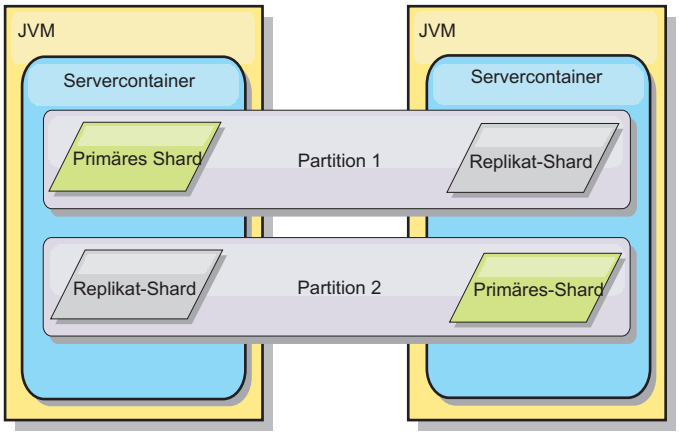


Abbildung 5. Partition

Shards sind Instanzen von Partitionen und haben eine von zwei Rollen: primäres Shard oder Replikat-Shard. Das primäre Shard und seine Replikate bilden die physische Manifestation der Partition. Jede Partition hat mehrere Shards, die jeweils alle in dieser Partition vorhandenen Daten enthalten. Ein Shard ist das primäre Shard, und die anderen Shards sind Replikate oder Replikat-Shards, d. h. redundante Kopien der Daten im primären Shard. Ein primäres Shard ist die einzige Partitionsinstanz, die Transaktionen das Schreiben in den Cache erlaubt. Ein Replikat-Shard ist eine "gespiegelte" Instanz der Partition. Es empfängt synchron oder asynchron Aktualisierungen vom primären Shard. Das Replikat-Shard erlaubt Transaktionen nur das Lesen von Daten aus dem Cache. Replikate befinden sich nie in demselben Container wie das primäre Shard und normalerweise nicht einmal auf derselben Maschine wie das primäre Shard.

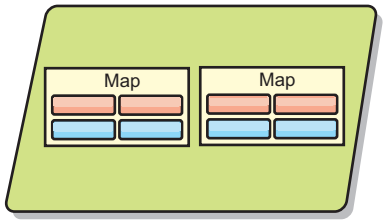


Abbildung 6. Shard

Um die Verfügbarkeit der Daten oder die Persistenzgarantien zu erhöhen, replizieren Sie die Daten. Die Replikation bedeutet jedoch einen höheren Aufwand für die Transaktion und damit Leistungseinbußen zu Gunsten der Verfügbarkeit. Mit eXtreme Scale können Sie den Aufwand steuern, da synchrone und asynchrone Replikation sowie Hybridreplikationsmodelle unterstützt werden, die synchrone und asynchrone Replikationsmodi verwenden. Ein synchrones Replikat-Shard empfängt Aktualisierungen im Rahmen der Transaktion des primären Shards, um die Datenkonsistenz zu gewährleisten. Ein synchrones Replikat kann die Antwortzeit verdoppeln, da die Transaktion sowohl im primären Shard als auch im synchronen Replikat festgeschrieben werden muss, bevor sie beendet werden kann. Ein asynchrones Replikat-Shard empfängt Aktualisierungen nach der Transaktionsfestschreibung, um die Auswirkungen auf die Leistung zu begrenzen, birgt aber das Risiko eines Datenverlusts, da das asynchrone Replikat mehrere Transaktionen hinter dem primären Shard zurückliegen kann.

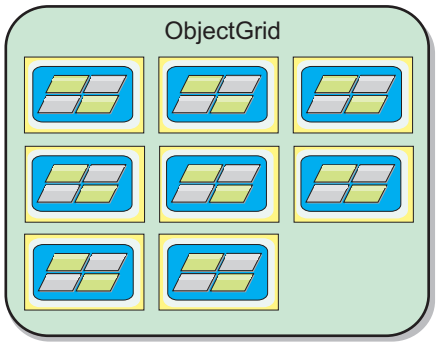


Abbildung 7. ObjectGrid

Clients

Clients stellen eine Verbindung zu einem Katalogservice her, rufen eine Beschreibung der Servertopologie ab und kommunizieren bei Bedarf mit jedem Server direkt. Wenn sich die Servertopologie ändert, weil neue Server hinzugefügt werden oder vorhandene Server ausfallen, leitet der dynamische Katalogservice den Client an den Server weiter, der die Daten enthält. Clients müssen die Schlüssel der Anwendungsdaten untersuchen, um festzustellen, an welche Partition die Anforderung weitergeleitet werden muss. Clients können in einer einzigen Transaktion Daten aus mehreren Partitionen lesen. Sie können jedoch innerhalb einer einzigen Transaktion nur eine einzige Partition aktualisieren. Nachdem der Client einige Einträge aktualisiert hat, muss die Clienttransaktion diese Partition für Aktualisierungen verwenden.

Die möglichen Implementierungskombinationen sind in der folgenden Liste aufgeführt:

- Ein Katalogservice ist in einem eigenen Grid von Java Virtual Machines vorhanden. Ein einziger Katalogservice kann verwendet werden, um mehrere Clients oder Server von eXtreme Scale zu verwalten.
- Ein Container kann ganz allein in einer JVM gestartet oder zusammen mit anderen Containern für andere ObjectGrid-Instanzen in eine beliebige JVM geladen werden.
- Ein Client kann in einer beliebigen JVM vorhanden sein und mit einer oder mehreren ObjectGrid-Instanzen kommunizieren. Ein Client kann sich auch in derselben JVM wie ein Container befinden.

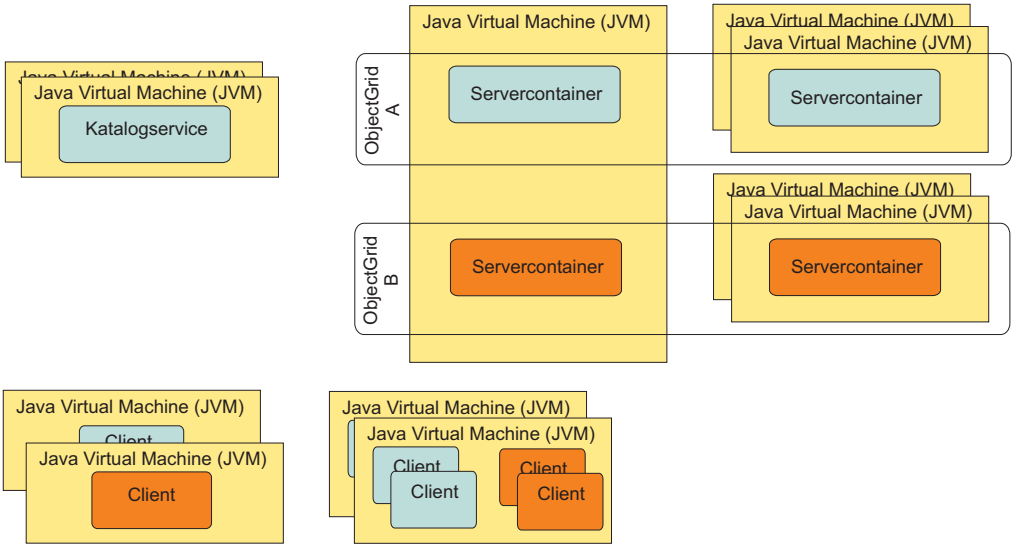


Abbildung 8. Mögliche Topologien

Katalogservices (Katalogserver)

Der Katalogservice enthält Logik, die bei stabilem Zustand der Topologie inaktiv bleibt und nur geringen Einfluss auf die Skalierbarkeit hat. Der Katalogservice ist so konzipiert, dass er Hunderte gleichzeitig verfügbarer Container bedienen kann, und führt Services für die Verwaltung der Container aus.

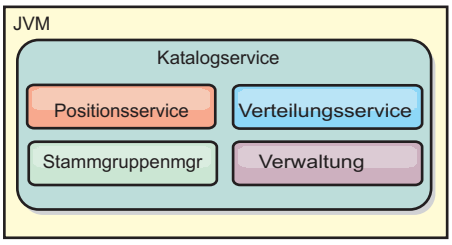


Abbildung 9. Katalogservice

Die Zuständigkeiten des Katalogs setzen sich aus den folgenden Services zusammen:

Arbeitsumgebung
Der Positionsservice stellt Positionen für Clients, die Container mit Anwendungen suchen, und für Container, die bereitgestellte Anwendungen beim

Verteilungsservice registrieren möchten, bereit. Der Positionsservice wird zur horizontalen Skalierung dieser Funktion in allen Grid-Membern ausgeführt.

Verteilungsservice
Der Verteilungsservice ist ein zentrales "Nervensystem" für das Grid und für die Zuordnung einzelner Shards zu ihrem Hostcontainer verantwortlich. Der Verteilungsservice wird als einer von N ausgewählten Services im Cluster ausgeführt. Da die Eins-von-N-Richtlinie verwendet wird, ist immer genau eine Instanz des Verteilungsservice aktiv. Wenn diese Instanz gestoppt wird, übernimmt ein anderer Prozess ihre Arbeit. Alle Zustände des Katalogservice werden für die Redundanz auf allen Servern repliziert, die den Katalogservice enthalten.

Stammgruppenmanager
Der Stammgruppenmanager verwaltet die Peer-Gruppierung für die Vitalitätsüberwachung, fasst Container zu kleinen Servergruppen zusammen und bindet die Servergruppen automatisch ein. Wenn ein Container den ersten Kontakt zum Katalogservice herstellt, wartet der Container auf die Zuordnung zu einer neuen oder vorhandenen Gruppe mehrerer Java Virtual Machines (JVM). Jede JVM-Gruppe überwacht die Verfügbarkeit ihrer Member durch den Austausch von Überwachungssignalen. Eines der Gruppen-Member übermittelt dem Katalogservice die Verfügbarkeitsdaten, damit dieser durch Neuzuordnung und Routenweiterleitung auf Fehler reagieren kann.

Verwaltung
Die vier Phasen der Verwaltung einer Umgebung mit WebSphere eXtreme Scale sind Planung, Implementierung, Verwaltung und Überwachung. Weitere Informationen zu jeder dieser Phasen finden Sie im *Administratorhandbuch*.

Für die Verfügbarkeit konfigurieren Sie eine Katalogservicedomäne. Eine Katalogservicedomäne setzt sich aus mehreren Java Virtual Machines, einschließlich einer Master-JVM und einer Reihe von Ausweich-JVMs zusammen.

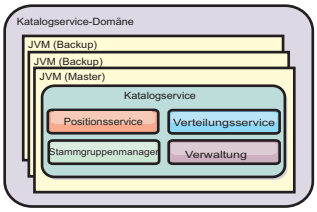


Abbildung 10. Katalogservicedomäne

Caching-Topologie: Speicherinternes und verteiltes Caching

Mit WebSphere eXtreme Scale kann Ihre Architektur speicherinternes Daten-Caching oder verteiltes Client/Server-Daten-Caching verwenden.

WebSphere eXtreme Scale erfordert für den Betrieb eine minimale zusätzliche Infrastruktur. Die Infrastruktur setzt sich aus Scripts für die Installation, das Starten und Stoppen von Java-EE-Anwendungen in einem Server zusammen. Die zwischengespeicherten Daten werden im Server von eXtreme Scale gespeichert, und Clients stellen über Fernzugriff eine Verbindung zum Server her.

Verteilte Caches bieten eine bessere Leistung, Verfügbarkeit und Skalierbarkeit und können unter Verwendung dynamischer Topologien konfiguriert werden, in denen Server automatisch verteilt werden. Zusätzliche Server können hinzugefügt werden, ohne die vorhandenen eXtreme Scale-Server erneut starten zu müssen. Sie können einfache Implementierungen erstellen oder große Implementierungen in Terabytegröße, in denen Tausende von Servern erforderlich sind.

Lokaler Speichercache

Im einfachsten Fall kann eXtreme Scale als lokaler (nicht verteilter) speicherinterner Daten-Grid-Cache verwendet werden. Dies kann insbesondere für Anwendungen mit sehr vielen gemeinsamen Zugriffen von Vorteil sein, in denen mehrere Threads auf transiente Daten zugreifen und diese ändern müssen. Die in einem lokalen eXtreme-Scale-Grid gespeicherten Daten können indexiert und über die Abfrageunterstützung von WebSphere eXtreme Scale abgerufen werden. Die Möglichkeit, die Daten abzufragen, kann Entwicklern bei der Arbeit mit großen speicherinternen Datenmengen besser helfen, als es die eingeschränkte Unterstützung für Datenstrukturen der Java Virtual Machine (JVM) tut, die sofort einsatzfähig ist.

Die lokale speicherinterne Cachetopologie für eXtreme Scale wird verwendet, um einen konsistenten, transaktionsorientierten Zugriff auf temporäre Daten in einer einzelnen Java Virtual Machine zu unterstützen.

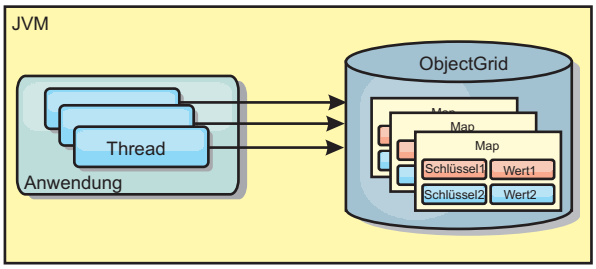


Abbildung 11. Szenario mit einem lokalen speicherinternen Speichercache

Vorteile

- Einfaches Setup: Ein ObjectGrid kann programmgesteuert oder deklarativ über die ObjectGrid-XML-Implementierungsdeskriptordatei oder mit anderen Frameworks wie Spring erstellt werden.
- Schnell: Jede BackingMap kann gesondert für eine optimale Speicherauslastung und gemeinsamen Zugriff optimiert werden.
- Ideal für Topologien mit einer einzigen JVM und einer kleinen Dateigruppe oder für das Caching von Daten, auf die häufig zugegriffen wird.
- Transaktionsorientiert: BackingMap-Aktualisierungen können zu einer einzigen Arbeitseinheit gruppiert und als letzter Teilnehmer in zweiphasige Transaktionen wie JTA-Transaktionen (Java Transaction Architecture) integriert werden.

Nachteile

- Keine Fehlertoleranz.
- Die Daten werden nicht repliziert. Speichercaches eignen sich am besten für schreibgeschützte Referenzdaten.
- Keine Skalierbarkeit. Die für die Datenbank erforderliche Speicherkapazität kann die JVM möglicherweise nicht bereitstellen.
- Es treten Probleme auf, wenn JVMs hinzugefügt werden.

- Die Daten sind nicht so einfach partitionierbar.
- Der Status muss in den JVMs manuell repliziert werden, da die einzelnen Cacheinstanzen ansonsten verschiedene Versionen derselben Daten enthalten könnten.
- Das Ungültigmachen von Einträgen ist kostenintensiv.
- Jeder Cache muss einzeln vorbereitet werden. Die Vorbereitungs- oder Aufwärmphase ist der Zeitraum, in dem eine Gruppe von Daten geladen wird, damit der Cache mit gültigen Daten gefüllt wird.

Einsatz

Die Implementierungstopologie mit dem lokalen Speichercache sollte nur verwendet werden, wenn die Menge der zwischenzuspeichernden Daten klein ist (in eine einzige JVM passt) und relativ stabil ist. Bei diesem Ansatz müssen veraltete Daten toleriert werden. Die Verwendung von Evictor (Bereinigungsprogramm), um nur die am häufigsten verwendeten oder die zuletzt verwendeten Daten im Cache zu verwalten, kann dabei helfen, den Cache klein zu halten und die Relevanz der Daten zu erhöhen.

Auf Peers replizierter lokaler Speichercache

Bei einem lokalen eXtreme-Scale-Cache müssen Sie sicherstellen, dass der Cache synchronisiert wird, wenn mehrere Prozesse mit voneinander unabhängigen Cacheinstanzen vorhanden sind. Hierfür aktivieren Sie einen replizierten Peer-Cache mit JMS.

WebSphere eXtreme Scale enthält zwei Plug-ins, die Transaktionsänderungen automatisch zwischen Peer-ObjectGrid-Instanzen weitergeben. Das JMSObjectGridEventListener-Plug-in gibt eXtreme-Scale-Änderungen automatisch über Java Messaging Service (JMS) weiter.

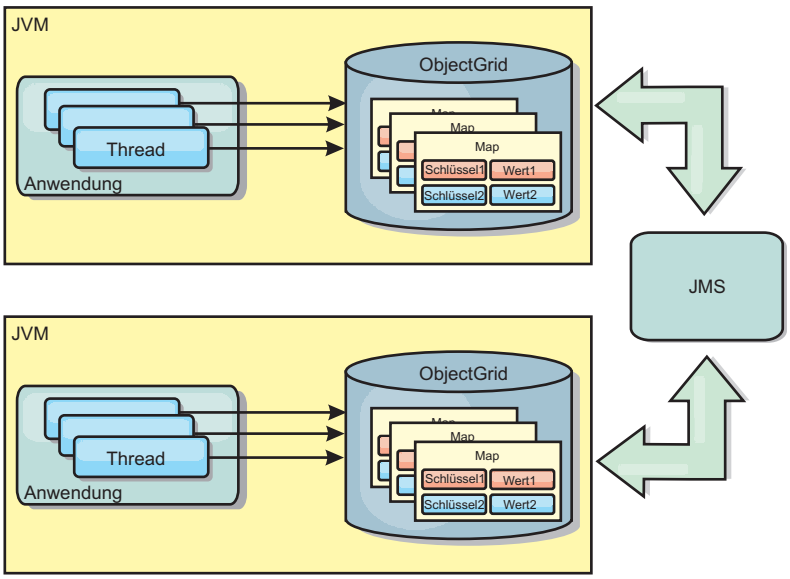


Abbildung 12. Auf Peers replizierter Cache mit Änderungen, die über JMS weitergegeben werden

Wenn Sie eine Umgebung von WebSphere Application Server ausführen, ist auch das TranPropListener-Plug-in verfügbar. Das TranPropListener-Plug-in verwendet den High Availability Manager (kurz HA-Manager), um die Änderungen an jede

Peer-Cacheinstanz von eXtreme Scale weiterzugeben.

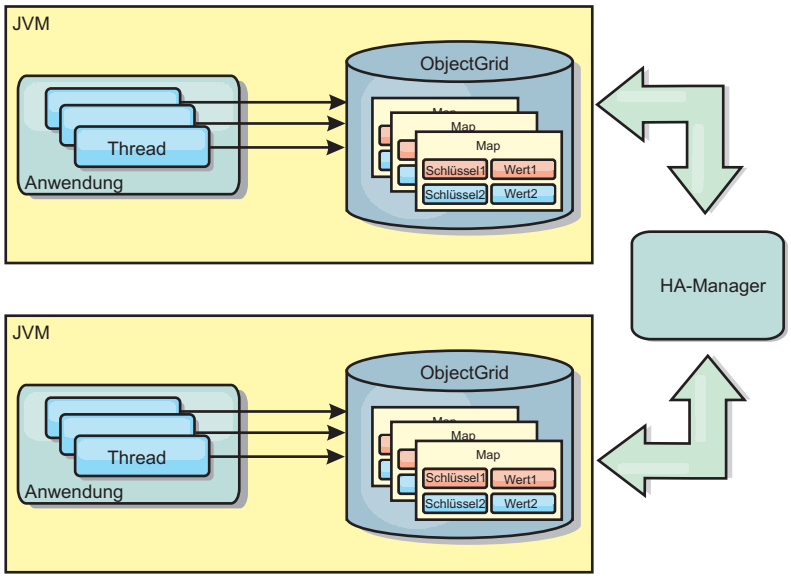


Abbildung 13. Auf Peers replizierter Cache mit Änderungen, die über den High Availability Manager weitergegeben werden

Vorteile

- Die Gültigkeit der Daten ist höher, weil sie häufiger aktualisiert werden.
- Mit dem TranPropListener-Plug-in kann eXtreme Scale wie die lokale Umgebung über das Programm oder deklarativ über die XML-Implementierungsdeskriptordatei von eXtreme Scale oder mit anderen Frameworks wie Spring erstellt werden. Die Integration mit dem High Availability Manager erfolgt automatisch.
- Jede BackingMap kann gesondert für eine optimale Speicherauslastung und gemeinsamen Zugriff optimiert werden.
- BackingMap-Aktualisierungen können zu einer einzigen Arbeitseinheit gruppiert und als letzter Teilnehmer in zweiphasige Transaktionen wie JTA-Transaktionen (Java Transaction Architecture) integriert werden.
- Ideal für Topologien mit wenigen JVMs und einer angemessen kleinen Dateigruppe oder für das Caching von Daten, auf die häufig zugegriffen wird.
- Änderungen an eXtreme Scale werden in allen Peer-Instanzen von eXtreme Scale repliziert. Die Änderungen sind so lange konsistent, wie eine permanente Subskription verwendet wird.

Nachteile

- Die Konfiguration und Verwaltung für JMSObjectGridEventListener kann komplex sein. eXtreme Scale kann über das Programm oder deklarativ über die XML-Implementierungsdeskriptordatei von eXtreme Scale oder mit anderen Frameworks wie Spring erstellt werden.
- Nicht skalierbar: Die für die Datenbank erforderliche Speicherkapazität kann die JVM möglicherweise nicht bereitstellen.
- Funktioniert nicht ordnungsgemäß, wenn Java Virtual Machines hinzugefügt werden:
  - Die Daten sind nicht so einfach partitionierbar.
  - Das Ungültigmachen von Einträgen ist kostenintensiv.
  - Jeder Cache muss einzeln vorbereitet werden.

Einsatz

Diese Implementierungstopologie sollte nur verwendet werden, wenn das zwischenzuspeichernde Datenvolumen gering (d. h. in eine einzige JVM passt) und relativ stabil ist.

Verteilter Cache

In den meisten Fällen wird WebSphere eXtreme Scale als gemeinsam genutzter Cache verwendet, um einen transaktionsgesteuerten Zugriff auf Dateien durch mehrere Komponenten zu ermöglichen, wo ansonsten eine traditionelle Datenbank verwendet werden würde. Bei der Verwendung eines gemeinsam genutzten Caches muss keine Datenbank konfiguriert werden.

Der Cache ist kohärent, weil alle Clients dieselben Daten im Cache sehen. Jede einzelne Information wird im Cache eines einzigen Servers gespeichert. Auf diese Weise werden unnötige Datensatzkopien verhindert, die potenziell verschiedene Versionen der Daten enthalten könnten. Ein kohärenter Cache kann außerdem mehr Daten aufnehmen, wenn dem Grid weitere Server hinzugefügt werden. Die Skalierung des Caches erfolgt linear zum Wachstum des Grids. Da Clients über Prozedurfernaufrufe auf Daten in diesem Grid zugreifen, wird der Cache auch als ferner Cache bezeichnet. Durch die Datenpartitionierung enthält jeder Prozess einen eindeutigen Teil der Gesamtdatengruppe. Größere Grids können mehr Daten aufnehmen und und mehr Anforderungen für diese Daten bearbeiten. Aufgrund der Kohärenz müssen auch keine Daten zum Ungültigmachen im Grid verteilt werden, weil es keine veralteten Daten gibt. Der kohärente Cache enthält jeweils nur die aktuelle Kopie jeder Information.

Wenn Sie in einer Umgebung mit WebSphere Application Server arbeiten, ist auch das TranPropListener-Plug-in verfügbar. Das TranPropListener-Plug-in verwendet die Komponente "High Availability Manager" (kurz HA-Manager) von WebSphere Application Server, um die Änderungen an die einzelnen Peer-ObjectGrid-Cacheinstanzen weiterzugeben.

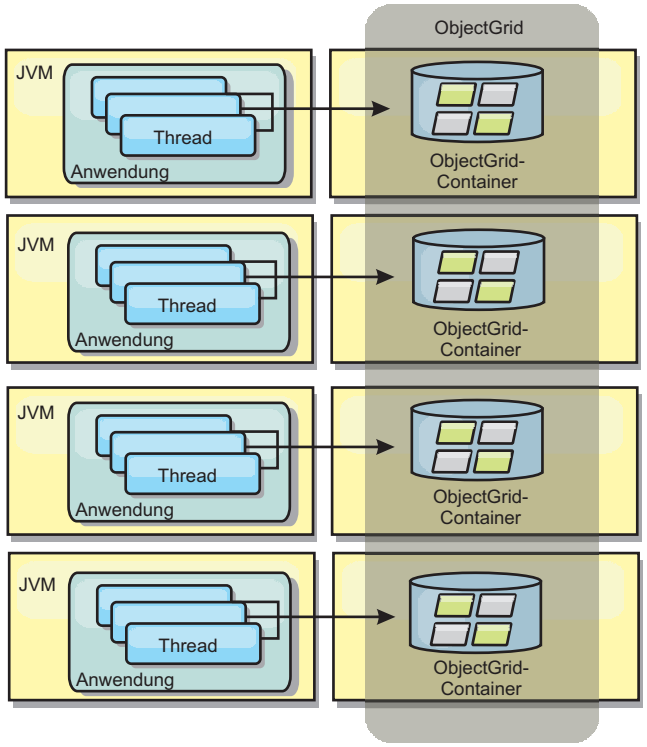


Abbildung 14. Verteilter Cache

Naher Cache

Clients können optional einen lokalen integrierten Cache haben, wenn eXtreme Scale in einer verteilten Topologie verwendet wird. Dieser optionale Cache wird als naher Cache bezeichnet. Er ist ein unabhängiges ObjectGrid in jedem Client, das als Cache für den fernen serverseitigen Cache dient. 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.

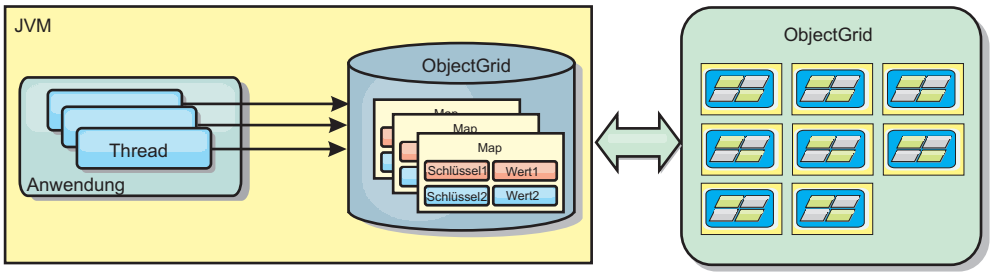


Abbildung 15. Naher Cache

Ein naher Cache ist sehr schnell, weil er den speicherinternen Zugriff auf einen Teil der gesamten zwischengespeicherten Daten ermöglicht, die fern in den Servern von eXtreme Scale gespeichert sind. Der nahe Cache ist nicht partitioniert und enthält Daten aus allen fernen eXtreme-Scale-Partitionen. WebSphere eXtreme Scale kann bis zu drei Cacheschichten haben. Diese sind im Folgenden erläutert:

1. Der Cache auf der Transaktionsschicht enthält alle Änderungen für eine einzelne Transaktion. Der Transaktionscache enthält eine Arbeitskopie der Daten, bis

die Transaktion festgeschrieben wird. Wenn eine Clienttransaktion Daten aus einer ObjectMap anfordert, wird zuerst die Transaktion geprüft.

2. Der nahe Cache auf der Clientschicht enthält einen Teil der Daten aus der Serverschicht. Wenn die Daten nicht auf Transaktionsschicht zu finden sind, werden die Daten (sofern verfügbar) aus dem nahen Cache abgerufen und in den Transaktionscache eingefügt.
3. Das Grid auf der Serverschicht enthält den Hauptteil der Daten und wird von allen Clients gemeinsam genutzt. Die Serverschicht kann partitioniert werden, was die Zwischenspeicherung großer Datenvolumen ermöglicht. Wenn der nahe Cache des Clients die Daten nicht enthält, werden die Daten von der Serverschicht abgerufen und in den Clientcache eingefügt. Die Serverschicht kann auch ein Loader-Plug-in (Ladeprogramm) haben. Wenn das Grid die angeforderten Daten nicht enthält, wird der Loader aufgerufen, der die Daten aus dem Back-End-Datenspeicher abruft und in das Grid einfügt.

Zum Inaktivieren des nahen Caches setzen Sie das Attribut "numberOfBuckets" in der eXtreme-Scale-Deskriptorkonfiguration für das Überschreiben des Clients auf "0". Einzelheiten zu den Sperrstrategien von eXtreme Scale finden Sie im Abschnitt zum Sperren von Map-Einträgen. Der nahe Cache kann auch mit einer gesonderten Bereinigungsrichtlinie und anderen Plug-ins konfiguriert werden, die die eXtreme-Scale-Deskriptorkonfiguration für das Überschreiben des Clients verwenden.

Vorteil

- Schnelle Antwortzeiten, weil alle Zugriffe auf die Daten lokal erfolgen.

Nachteile

- Veraltete Daten bleiben länger verfügbar.
- Es muss ein Evictor (Bereinigungsprogramm) zum Ungültigmachen von Daten verwendet werden, um Speicherengpässe zu verhindern.

Einsatz

Verwenden Sie diese Technik, wenn die Antwortzeiten wichtig und veraltete Daten tolerierbar sind.

Integrierter Cache

eXtreme-Scale-Grids können in vorhandenen Prozessen als integrierte eXtreme-Scale-Server ausgeführt oder als externe Prozesse verwaltet werden. Integrierte Grids sind hilfreich, wenn Sie mit einem Anwendungsserver wie WebSphere Application Server arbeiten. Sie können eXtreme-Scale-Server, die nicht integriert sind, über Befehlszeilenscripts starten und in einem Java-Prozess ausführen.

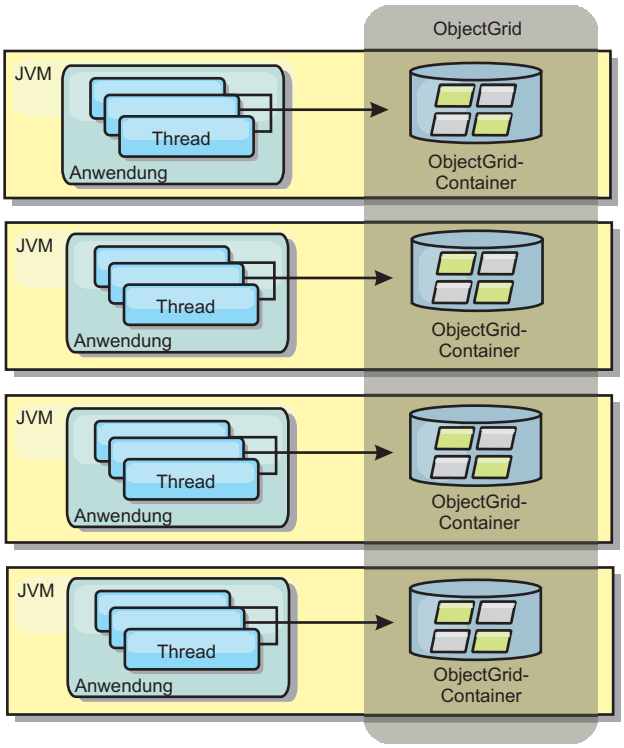


Abbildung 16. Integrierter Cache

Vorteile

- Vereinfachte Verwaltung, da weniger Prozesse zu verwalten sind
- Vereinfachte Anwendungsimplementierung, weil das Grid den Klassen-Loader der Clientanwendung verwendet
- Unterstützung von Partitionierung und hoher Verfügbarkeit

Nachteile

- Erhöhter Speicherbedarf im Clientprozess, weil alle Daten im Prozess erfasst werden
- Erhöhte CPU-Auslastung für die Bearbeitung von Clientanforderungen
- Erschwerte Verarbeitung von Anwendungsupgrades, da Clients dieselben Java-Anwendungsarchivdateien wie die Server verwenden
- Weniger Flexibilität. Die Skalierung von Clients und Grid-Servern ist nicht linear. Wenn Server extern definiert werden, haben Sie mehr Flexibilität bei der Verwaltung der Prozessanzahl.

Einsatz

Verwenden Sie integrierte Grids, wenn im Clientprozess reichlich Speicher für die Grid-Daten und potenzielle Failover-Daten frei ist.

Weitere Informationen finden Sie im Abschnitt zum Mechanismus für Clientinaktivierung im *Administratorhandbuch*.

Multmaster-Replikationstopologie (AP)

Wenn Sie das asynchrone Multimaster-Replikationsfeature verwenden, können zwei oder mehr Grids exakte Spiegel voneinander werden. Diese Spiegelung wird

durch asynchrone Replikation zwischen Links erreicht werden, die die Grids miteinander verbinden. Jedes Grid ist in einer vollständig unabhängigen "Domäne" enthalten, hat einen eigenen Katalogservice, eigene Containerserver und einen eindeutigen Domänennamen. Mit dem asynchronen Multimaster-Replikationsfeature können Sie Links verwenden, um eine Sammlung dieser Domänen miteinander zu verbinden, und diese Domänen anschließend durch Replikation über die Links synchronisieren. eXtreme Scale ermöglicht Ihnen, nahezu jede Topologie zu erstellen, weil die Definition der Links zwischen den Domänen Ihnen überlassen bleibt.

Domänen: Grids mit speziellen Merkmalen

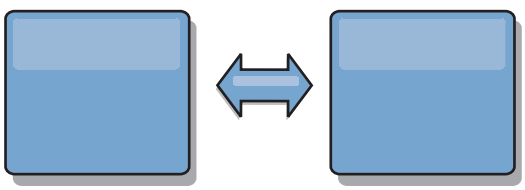
In Multimaster-Replikationstopologien verwendete Grids werden als *Domänen* bezeichnet. Jede Domäne muss die folgenden Merkmale haben:

- Hat einen privaten Katalogservice mit einem eindeutigen Domänennamen
- Hat denselben Grid-Namen wie andere Grids in der Domäne
- Hat dieselbe Anzahl an Partitionen wie andere Grids in der Domäne
- Ist ein Grid vom Typ FIXED_PARTITION (Grids vom Typ PER_CONTAINER können nicht repliziert werden)
- Hat dieselbe Anzahl an Partitionen (kann, muss aber nicht dieselbe Anzahl und dieselben Typen von Replikaten haben)
- Hat dieselben Replikationsdatentypen wie andere Grids in der Domäne
- Hat dieselben MapSet-Namen, Map-Namen und dynamischen Map-Schablonen wie andere Grids in der Domäne

Nachdem die Domänen in der Topologie gestartet wurden, werden alle Grids mit den zuvor beschriebenen Merkmalen repliziert. Beachten Sie, dass die Replikationsrichtlinien der Grids ignoriert werden.

Links, die Domänen verbinden

Eine Grid-Replikationsinfrastruktur ist ein verbundener Graph von Domänen mit bidirektionalen Links zwischen den Domänen. Über einen Link können zwei Domänen Daten miteinander austauschen. Die einfachste Topologie ist beispielsweise ein Domänenpaar mit einem einzigen Link zwischen ihnen. Die Domänen werden von links aus gesehen beginnend mit "A", dann "B" usw. benannt. Der Link kann ein Weitverkehrsnetz (WAN) durchqueren und große Distanzen überwinden. Wenn der Link unterbrochen wird, können trotzdem Änderungen an den Daten in den beiden Domänen vorgenommen werden. Die Änderungen werden später abgeglichen, wenn der Link die Domänen wieder verbindet. Links versuchen nach der Unterbrechung der Netzverbindung automatisch, die Verbindung wiederherzustellen.

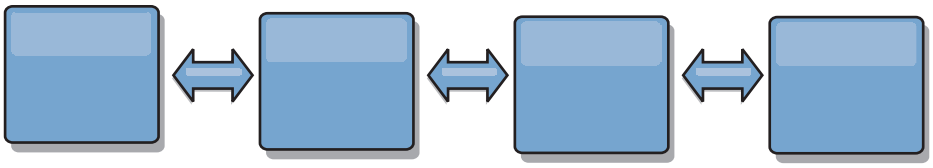


Nach dem Einrichten der Links versucht eXtreme Scale zuerst, alle Domänen abzugleichen, und dann, diese identischen Bedingungen beizubehalten, wenn Änderungen in den Domänen stattfinden. eXtreme Scale hat das Ziel, dass jede Domäne eine exakte Spiegelung jeder anderen Domäne ist, die über die Links verbunden ist. Die Replikationslinks zwischen den Domänen stellen sicher, dass alle Änderun-

gen, die in einer Domäne vorgenommen werden, in die anderen Domänen kopiert werden.

Reihentopologien

Obwohl die Reihentopologie zu den einfachsten Topologien gehört, veranschaulicht sie einige Qualitäten der Links. Zunächst ist es nicht erforderlich, dass eine Domäne direkte mit jeder anderen Domäne verbunden ist, damit sie Änderungen empfängt. Domäne B übernimmt Änderungen von Domäne A. Domäne C empfängt Änderungen von Domäne A über Domäne B, die die Domänen A und B verbindet. Domäne D empfängt Änderungen von den anderen Domänen über Domäne C. Auf diese Weise kann die Quelle der Änderungen von der Verteilung der Änderungen entlastet werden.



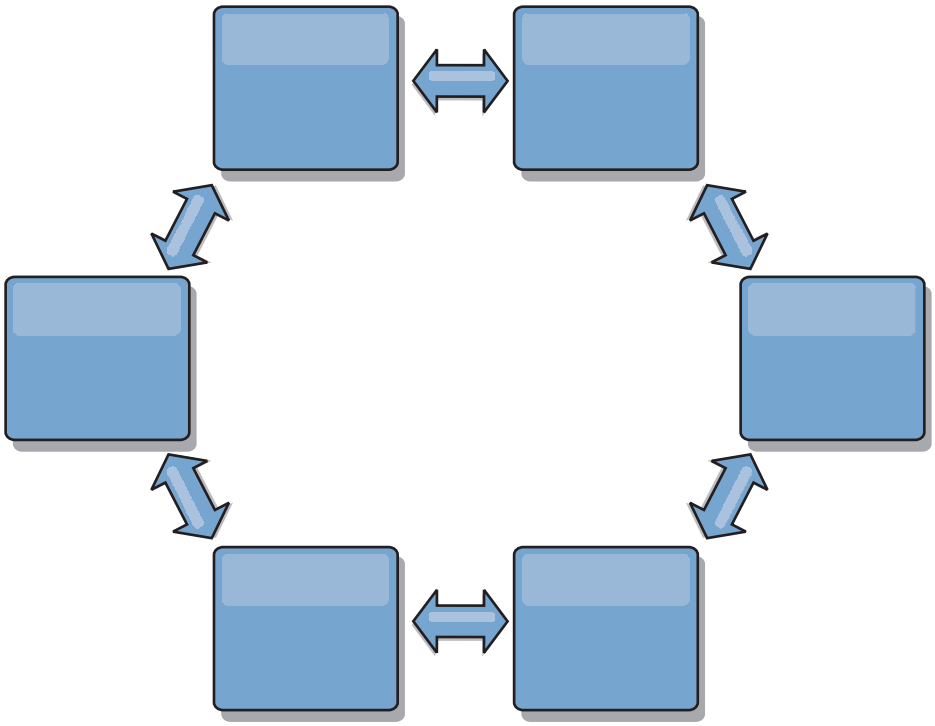
Wenn Domäne C ausfällt, treten die folgenden Ereignisse ein:

1. Domäne D ist verwaist, bis Domäne C erneut gestartet wird.
2. Domäne C synchronisiert sich selbst mit Domäne B, die eine Kopie von Domäne A ist.
3. Domäne D verwendet Domäne C, um sich mit den Änderungen in den Domänen A und B zu synchronisieren, die vorgenommen wurden, während Domäne D verwaist war (aufgrund des Ausfalls von Domäne C).

Am Ende sind die Domänen A, B, C und D wieder identisch.

Ringtopologien

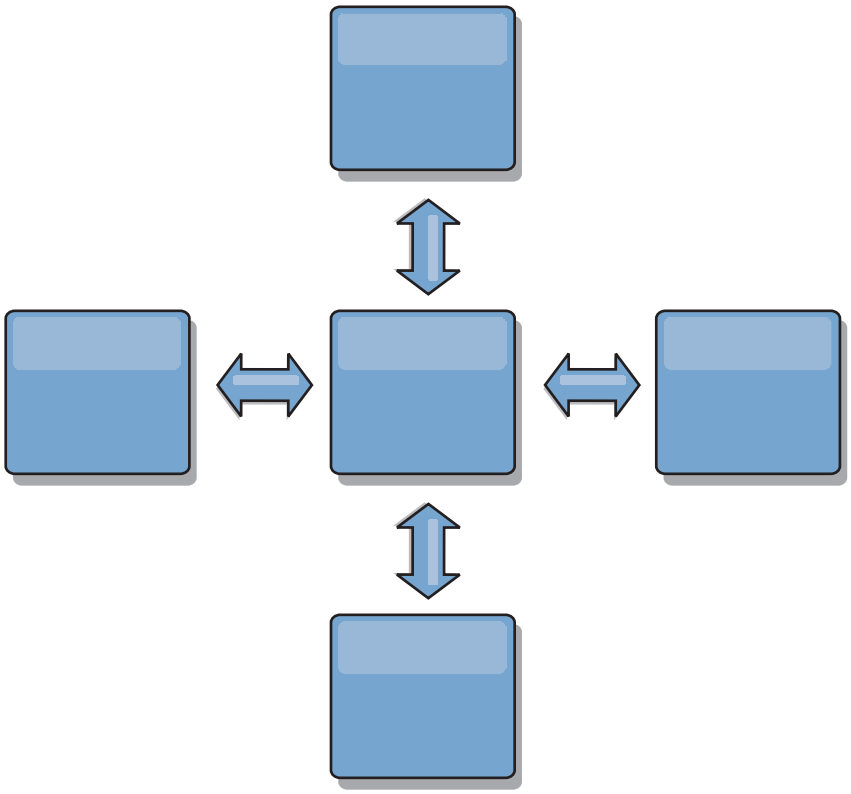
Ringtopologien sind ein Beispiel für eine Topologie mit erhöhter Ausfallsicherheit. Eine Domäne oder ein einzelner Link kann ausfallen, aber die verbleibenden Domänen können trotzdem Änderungen empfangen, weil die Änderungen im Ring in die andere Richtung (von der ausgefallenen Domäne bzw. vom ausgefallenen Link weg) weitergegeben werden. Jede Domäne hat zwei Links zu den anderen Domänen. Jede Domäne hat maximal zwei Links, unabhängig davon, wie groß eine Ringtopologie ist. Die Latenzzeit für die Weitergabe der Änderung kann hoch sein, weil Änderungen einer bestimmten Domäne unter Umständen mehrere Domänen durchqueren müssen, bevor sie von allen Domänen gesehen werden. Eine Reihentopologie hat dasselbe Problem.



Dies ist eine Abbildung einer fortgeschrittenen Ringtopologie mit einer Stammdomäne in der Mitte des Rings. Die Stammdomäne dient als zentrale Clearing-Stelle, während die anderen Domänen als ferne Clearing-Stellen für Änderungen dienen, die in der Stammdomäne vorgenommen werden. Die Stammdomäne kann Änderungen zwischen den Domänen arbitrieren. Wenn eine Ringtopologie mehrere Ringe um die Stammdomäne herum enthält, kann die Stammdomäne Änderungen nur zwischen den Domänen im innersten Ring arbitrieren. Die Ergebnisse der Arbitrierung werden jedoch an die Domänen in den anderen Ringen verteilt.

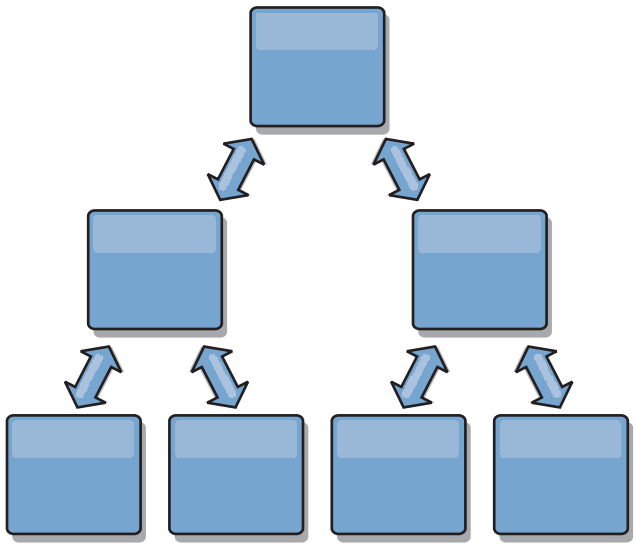
Hub- und Peripherietopologien

Eine Hub- und Peripherietopologie hat bessere Latenzzeiten, d. h., Änderungen durchqueren maximale eine zwischengeordnete Domäne (den Hub), bringt aber andere Probleme mit sich. Eine solche Topologie hat eine zentrale Domäne, die als Hub dient. Diese "Hub-Domäne" ist über einen Link mit jeder "Peripheriedomäne" verbunden. Die Last der Verteilung von Änderungen an die Domänen liegt klar auf dem Hub. Der Hub dient als Clearing-Stelle für Kollisionen. Deshalb kann dieses Setup für bestimmte Szenarien von entscheidender Bedeutung sein. In einer Umgebung mit einer hohen Aktualisierungsrate muss der Hub unter Umständen auf mehr Hardwarekomponenten als die Peripheriedomänen ausgeführt werden, damit er Schritt halten kann. eXtreme Scale ist für eine lineare Skalierung konzipiert, d. h., Sie können den Hub bei Bedarf ohne Schwierigkeit vergrößern. Wenn der Hub jedoch ausfällt, werden die Änderungen erst nach einem Neustart des Hubs wieder verteilt. Alle Änderungen in den Peripheriedomänen werden verteilt, nachdem die Hub-Verbindung wiederhergestellt ist.



Baumtopologien

Ein letztes Topologiebeispiel ist eine azyklische gerichtete Baumstruktur. Azyklisch bedeutet, dass es keine Zyklen und keine Schleifen gibt. Gerichtet bedeutet, dass Links nur zwischen übergeordneten und untergeordneten Elementen existieren. Diese Konfiguration kann für Topologien hilfreich sein, die so viele Domänen haben, dass die Verwendung eines zentralen Hubs, der mit jeder möglichen Peripheriedomäne verbunden ist, nicht praktisch ist, oder in Domänen, in denen Sie untergeordnete Domänen hinzufügen können, ohne die Stammdomäne zu aktualisieren.



Diese Topologie kann trotzdem eine zentrale Clearing-Stelle in der Stammdomäne haben, aber die zweite Ebene kann als ferne Clearing-Stelle für Änderungen die-

nen, die in den Domänen unterhalb dieser Ebene vorgenommen werden. Die Stammdomäne kann Änderungen zwischen den Domänen nur auf der zweiten Ebene arbitrieren. N-gliedrige Baumstrukturen sind ebenfalls möglich. Eine N-gliedrige Baumstruktur hat auf jeder Ebene N untergeordnete Elemente. Jede Domäne hat ein Fanout (Ausgabefächerung) von N.

Arbitrierungshinweise für das Topologiedesign

Änderungskollisionen können auftreten, wenn dieselben Datensätze gleichzeitig an zwei Stellen geändert werden können. Konfigurieren Sie alle Domänen mit denselben Werten für CPU-, Hauptspeicher und Netzressourcen. Sie können beobachten, dass Domänen, die für das Kollisions-Handling (Arbitrierung) zuständig sind, mehr Ressourcen als andere Domänen verbrauchen. Kollisionen werden automatisch erkannt. Sie werden mit einem der folgenden beiden Mechanismen behoben:

- **Standardkollisions-Arbiter**: Standardmäßig werden die Änderungen aus der Domäne verwendet, deren Name in der lexikalischer Reihenfolge am niedrigsten steht. Wenn beispielsweise Domäne A und Domäne B einen Konflikt in Bezug auf einen Datensatz verursachen, wird die Änderung aus Domäne B ignoriert. Domäne A behält ihre Version, und der Datensatz in Domäne B wird geändert, so dass er dem Datensatz aus Domäne A entspricht. Dies gilt auch für Anwendungen, in denen Benutzer oder Sitzungen normalerweise gebunden sind oder eine Affinität zu einem der Grids haben.
- **Angepasster Kollisions-Arbiter:** Anwendungen können einen angepassten Arbiter bereitstellen. Wenn eine Domäne eine Kollision erkennt, ruft sie den Arbiter auf. Informationen zum Entwickeln eines 'guten' angepassten Arbiters finden Sie unter Angepasste Arbiter für Multimaster-Replikation.

Für Topologien, in denen Kollisionen möglich sind, sollten Sie die Verwendung einer Hub- und Peripherietopologie oder einer Baumtopologie in Erwägung ziehen. Diese beiden Topologien sind dienlich, um Endloskollisionen zu vermeiden, die auftreten können, wenn

1. in mehreren Domänen eine Kollision auftritt,
2. jede Domäne die Kollision lokal behebt, was zu Überarbeitungen führt,
3. die Überarbeitungen kollidieren, was zu Überarbeitungen von Überarbeitungen führt,
4. da die Überarbeitungen zwischen den verschiedenen Domänen weitergeleitet werden, um eine Synchronizität zu erreichen.

Zur Vermeidung von Endloskollisionen wählen Sie eine bestimmte Domäne, eine *Arbitrierungsdomäne*, als Kollisions-Handler für einen Teil der Domänen aus. In einer Hub- und Peripherietopologie kann der Hub beispielsweise als Kollisions-Handler verwendet werden. Der Peripheriekollisions-Handler ignoriert alle von den Peripheriedomänen erkannten Kollisionen. Die Hub-Domäne erstellt Überarbeitungen, wodurch Kollisionsüberarbeitungen, die außer Kontrolle geraten, verhindert werden. Die für die Behandlung von Kollisionen zugeordnete Domäne muss einen Link zu allen Domänen haben, für die sie Kollisionen beheben soll. In einer Baumtopologie beheben alle internen übergeordneten Domänen Kollisionen für die ihnen unmittelbar untergeordneten Domänen. Wenn Sie eine Ringtopologie verwendet, ist es nicht möglich, eine einzige Domäne im Ring zu bestimmen, die die Kollisionen beheben soll.

In der folgenden Tabelle sind die kompatiblen Arbitrierungsansätze für die verschiedenen Topologien zusammengefasst.

Tabelle 3. Arbitrierungsansätze. Der folgenden Tabelle können Sie entnehmen, ob Anwendungsarbitrierung mit den verschiedenen Technologien kompatibel ist.

Topologie	Anwendungs-arbitrierung?	Anmerkungen
Eine Reihe von zwei Domänen	Ja	Wählen Sie eine Domäne als Arbiter aus.
Eine Reihe von drei Domänen	Ja	Die mittlere Domäne muss der Arbiter sein. Stellen Sie sich die mittlere Domäne als Hub in einer einfachen Hub- und Peripherietopologie vor.
Eine Reihe von mehr als drei Domänen	Nein	Anwendungsarbitrierung wird nicht unterstützt.
Ein Hub mit N Peripheriedomänen	Ja	Der Hub mit den Links zu allen Peripheriedomänen muss die Arbitrierungsdomäne sein.
Ein Ring mit N Domänen	Nein	Anwendungsarbitrierung wird nicht unterstützt.
Eine azyklische gerichtete Baumstruktur (N-gliedriger Baumstruktur)	Ja	Alle Stammknoten müssen nur die ihnen direkt untergeordneten Knoten arbitrieren.

Linkhinweise für das Topologiedesign

Im Idealfall enthält eine Topologie die Mindestanzahl an Links und optimiert gleichzeitig Kompromisse zwischen Latenzzeit für Änderungen, Fehlertoleranz und Leistungsmerkmale.

- **Latenzzeit bei Änderungen**

  Die Latenzzeit bei Änderungen wird durch die Anzahl zwischengeschalteter Domänen bestimmt, die eine Änderung durchlaufen muss, bevor sie in einer bestimmten Domäne ankommt.

  Eine Topologie weist die beste Latenzzeit bei Änderungen auf, wenn zwischengeschaltete Domänen wegfallen, weil jede Domäne mit jeder anderen Domäne verlinkt wird. Eine Domäne muss jedoch proportional zur Anzahl ihrer Links Replikationsarbeiten ausführen. In großen Topologien kann die reine Anzahl zu definierender Links einen hohen Verwaltungsaufwand darstellen.

  Die Geschwindigkeit, mit der eine Änderung in andere Domänen kopiert wird, richtet sich nach weiteren Faktoren, wie z. B.:

  – CPU und Netzbandbreite in der Quellendomäne,
  – Anzahl zwischengeschalteter Domänen und Links zwischen der Quellen- und der Zieldomäne,
  – CPU- und Netzressourcen, die der Quellendomäne, der Zieldomäne und den zwischengeschalteten Domänen zur Verfügung stehen.

- **Fehlertoleranz**

  Die Fehlertoleranz wird durch die Anzahl der existierenden Pfade zwischen zwei Domänen für die Änderungsreplikation bestimmt.

  Wenn nur ein einziger Link zwischen Domänen existiert und dieser Link ausfällt, werden Änderungen nicht weitergegeben. Wenn der einzige Link von einer Domäne zu einer anderen Domäne über zwischengeschaltete Domänen verläuft, werden die Änderungen nicht weitergegeben, wenn eine der zwischengeschalteten Domänen inaktiv ist.

  Stellen Sie sich eine Reihentopologie mit drei Domänen, A, B und C, vor:

```
A <-> B <-> C
```

Wenn eine der folgenden Bedingungen zutrifft, sieht die Domäne C Änderungen von Domäne A nicht:

- Domäne A ist aktiv, und Domäne B ist inaktiv.
- Der Links zwischen A und B ist inaktiv.
- Der Link zwischen B und C ist inaktiv.

Im Gegensatz dazu kann in einer Ringtopologie jede Domäne Änderungen aus beiden Richtungen übernehmen.

```
A <-> B <-> C <-> zurück zu A
```

Wenn beispielsweise Domäne B inaktiv ist, kann Domäne C trotzdem Änderungen direkt von Domäne A übernehmen.

Ein Hub- und Peripheriedesign ist anfällig, wenn der Hub ausfällt, weil alle Änderungen über den Hub verteilt werden. Es ist jedoch zu bedenken, dass eine einzige Domäne trotzdem ein vollständig fehlertolerantes Grid ist, in dem Fehler wie WAN-Probleme oder Probleme im physischen Rechenzentrum selten auftreten.

- **Leistung**

  Die Anzahl der in einer Domäne definierten Links wirkt sich auf die Leistung aus. Je mehr Links definiert werden, desto mehr Ressourcen werden benötigt, und deshalb kann die Replikationsleistung abnehmen. Die Möglichkeit, Änderungen für Domäne A über andere Domänen zu beziehen, entlastet die Domäne A effektiv von der Replikation ihrer Transaktionen in allen Domänen. *Die Last für die Verteilung der Änderungen in einer Domäne ist auf die Anzahl der von dieser Domäne verwendeten Links beschränkt. Sie hat nicht mit der Anzahl der Domänen in der Topologie zu tun.* Diese Eigenschaft bietet Skalierbarkeit und die Möglichkeit, die Last für die Verteilung von Änderungen auf die Domänen in der Topologie zu verteilen, anstatt diese auf eine einzige Domäne zu konzentrieren.

  Eine Domäne kann Änderungen indirekt über andere Domänen beziehen. Stellen Sie sich eine Reihentopologie mit fünf Domänen vor.

  ```
  A <=> B <=> C <=> D <=> E
  ```

  - A bezieht Änderungen von B, C, D und E über B.
  - B bezieht Änderungen von A und C direkt und Änderungen von D und E über C.
  - C bezieht Änderungen von B und D direkt und Änderungen von A über B und Änderungen von E über D.
  - D bezieht Änderungen von C und E direkt und Änderungen von A und B über C.
  - E bezieht Änderungen von D direkt und Änderungen von A, B und C über D.

  Die Verteilungslast ist in den Domänen A und E am geringsten, weil sie jeweils nur einen Link zu einer einzigen Domäne haben. Die Verteilungslast in den Domänen B, C und D ist doppelt so hoch wie die Laste in den Domänen A und E, weil die Domänen B, C und D jeweils einen Links zu zwei Domänen haben. Diese Verteilung der Last bliebe auch bei 1000 Domänen in der Reihe konstant, weil sich die Last nach der Anzahl der Links jeder Domäne richtet und nicht nach der Gesamtanzahl der Domänen in der Topologie.

Leistungsaspekte

Berücksichtigen Sie bei der Verwendung von Multimaster-Replikationstopologien die folgenden Einschränkungen:

- **Optimierung der Verteilung von Änderungen** (zuvor beschrieben)
- **Latenzzeit bei der Replikation** (zuvor beschrieben)
- **Leistung von Replikationslinks**: eXtreme Scaleerstellt einen einzigen TCP/IP-Socket zwischen jedem JVM-Paar. Der gesamte Datenverkehr zwischen diesen JVMs findet über diesen Socket statt, einschließlich der Multimaster-Replikation. Da sich Domänen in mindestens N Container-JVMs mit N TCP-Links zu Peer-Domänen befinden, weisen die Domänen mit einer höheren Anzahl an Containern höhere Replikationsleistungsraten auf. Mehr Container bedeuten mehr CPU- und Netzressourcen.
- **Optimierung des TCP-Sliding-Window und RFC 1323**: Wenn Sie die Unterstützung für RFC 1323 auf beiden Seiten eines Links aktivieren, werden mehr Daten in einer Austauschoperation zugelassen, was zu einem höheren Durchsatz führt. Die Technik erweitert die Fensterkapazität um einen Faktor von ca. 16.000.

  TCP-Sockets verwenden einen Sliding-Window-Mechanismus, um den Fluss von Massendaten zu steuern, der den Socket gewöhnlich auf 64 KB für ein Umlaufintervall beschränkt. Wenn ein Umlaufintervall 100 ms lang ist, ist die Bandbreite ohne Optimierung auf 640 KB/Sekunde beschränkt. Um die verfügbare Bandbreite eines Links vollständig nutzen zu können, müssen unter Umständen spezielle Optimierungs-Tasks für ein Betriebssystem ausgeführt werden. Die meisten Betriebssysteme haben Optimierungsparameter, einschließlich Optionen für RFC 1323, um den Durchsatz über Links mit hoher Latenzzeit zu verbessern.

  Es gibt mehrere Faktoren, die sich auf die Replikationsleistung auswirken können:

  – Geschwindigkeit, mit der eXtreme Scale Änderungen beziehen kann,
  – Geschwindigkeit, mit der eXtreme Scale Replikationsanforderungen bearbeiten kann,
  – Kapazität des Sliding Window,
  – Optimierung des Netzpuffers auf beiden Seiten eines Links, damit eXtreme Scale Änderungen über den Socket so schnell wie möglich beziehen kann.
- **Objektserialisierung**: Alle Daten müssen serialisierbar sein. Wenn eine Domäne COPY_TO_BYTES nicht verwendet, muss die Domäne Java-Serialisierung oder ObjectTransformer verwenden, um die Serialisierungsleistung zu optimieren.
- **Komprimierung**: eXtreme Scale komprimiert standardmäßig alle Daten, die zwischen Domänen gesendet werden. Es gibt keine Option für die Inaktivierung der Komprimierung im aktuellen Release.
- **Hauptspeicheroptimierung**: *Die Speicherbelegung für eine Multimaster-Replikationstopologie ist weitgehend unabhängig von der Anzahl der Domänen in der Topologie.*

  Die Aktivierung der Multimaster-Replikation bedeutet feste Gemeinkosten pro Map-Eintrag für die Versionssteuerung. Außerdem überwacht jeder Container ein festes Datenvolumen für jede Domäne in der Topologie. Eine Topologie mit zwei Domänen belegt ungefähr denselben Speicher wie eine Topologie mit 50 Domänen. eXtreme Scale verwendet keine Wiedergabeprotokolle oder ähnlichen Warteschlangen in der Implementierung, d. h., wenn ein Replikationslink für längere Zeit nicht verfügbar ist, gibt Datenstruktur mit zunehmender Größe, die darauf wartet, die Replikation fortzusetzen, wenn der Link erneut gestartet wird.

Mehrere Rechenzentren mit FIXED_PARTITION

Sie können jetzt ein FIXED_PARTITION-Grid zwischen zwei oder mehr Rechenzentren einsetzen. Jedes Rechenzentrum benötigt im Hinblick auf die Multimaster-Replikation eine eigene Domäne. Jedes Rechenzentrum kann Daten aus der lokalen Domäne lesen und in diese schreiben. Diese Änderungen werden an die anderen Rechenzentren über die von Ihnen definierten Links weitergegeben.

Vollständig replizierte Clients

Diese Topologievariante umfasst ein eXtreme-Scale-Serverpaar, das als Hub ausgeführt wird. Jeder Client erstellt ein eigenständiges Einzelcontainer-Grid mit einem Katalog in der Client-JVM. Ein Client verwendet sein Grid, um die Verbindung zum Hub-Katalog herzustellen, was bewirkt, dass sich der Client mit dem Hub synchronisiert, sobald die Verbindung zum Hub hergestellt ist.

Alle vom Client vorgenommenen Änderungen sind lokal und werden asynchron im Hub repliziert. Der Hub dient als Arbitrierungsdomäne und verteilt Änderungen an alle verbundenen Clients. Die Topologie mit vollständig replizierten Clients ist ein guter L2-Cache für einen objektbezogenen Mapper wie OpenJPA. Änderungen werden über den Hub schnell an die Client-JVMs verteilt. Solange die Cachegröße vom verfügbaren Heap-Speicher der Clients untergebracht werden kann, ist diese Topologie eine geeignete Architektur für diesen L2-Stil.

Verwenden Sie bei Bedarf mehrere Partitionen für die Skalierung der Hub-Domäne in mehreren JVMs. Da alle Daten weiterhin in eine einzige Client-JVM passen müssen, erhöht die Verwendung mehrerer Partitionen die Kapazität des Hubs für die Verteilung und Arbitrierung von Änderungen, aber nicht die Kapazität einer einzelnen Domäne.

Einschränkungen

Berücksichtigen Sie bei der Entscheidung, ob und wie Multimaster-Replikationstopologien zu verwenden sind, die folgenden Einschränkungen:

- **Konfiguration von Loadern mit mehreren Domänen**

  Domänen müssen Zugriff auf alle Klassen haben, die als Schlüssel und Werte verwendet werden. Alle Abhängigkeiten müssen sich in allen Klassenpfaden für Grid-Container-JVMs für alle Domänen widerspiegeln. Wenn ein CollisionArbiter-Plug-in den Wert für einen Cacheeintrag abruft, müssen die Klassen für die Werte für die Domäne vorhanden sein, die den Arbiter aufruft.
- **Verwendung von Loadern wird nicht empfohlen**

  Loader können verwendet werden, um Änderungen zwischen einem Grid und einer Datenbank zu übertragen. Es ist unwahrscheinlich, dass alle Grids (Domänen) in einer Topologie geografisch mit derselben Datenbank kollokiert sind. Aufgrund der Latenzzeit im WAN und anderen Faktoren kann dieser Anwendungsfall unerwünscht sein.

  Das vorherige Laden (Preload) des Grids ist ein weiterer Punkt, der eines sorgfältigen Designs bedarf. Beim Neustart eines Grids wird der Preload gewöhnlich erneut durchgeführt. Ein Preload ist bei der Verwendung einer Multimaster-Replikation nicht erforderlich oder sogar nicht erwünscht. Sobald eine Domäne online ist, lädt sie automatisch den Inhalt der Domänen, mit denen sie verlinkt ist. Deshalb ist es nicht erforderlich, einen manuellen Preload für ein Grid einzuleiten, das eine Domäne in einer Multimaster-Replikationstopologie ist.

Loader halten gewöhnlich Einfüge- und Aktualisierungsregeln ein. Bei der Multimaster-Replikation müssen Einfügungen als Zusammenführungen behandelt werden. Wenn die Daten nach einem Domänenneustart über Fernzugriff abgerufen werden, werden vorhandene Daten in die lokale Domäne "eingefügt". Da diese Daten unter Umständen bereits in der lokalen Datenbank enthalten sind, schlägt eine typische Einfügung mit einer Ausnahme wegen doppelt vorhandener Schlüssel in der Datenbank fehl. Stattdessen muss die Semantik für Zusammenführung (merge) verwendet werden.

eXtreme Scale kann für die Durchführung eines Shard-basierten Preloads unter Verwendung der Preload-Methoden in Loader-Plug-ins konfiguriert werden. Verwenden Sie diese Technik nicht in einer Multimaster-Replikationstopologie. Verwenden Sie stattdessen einen clientbasierte Preload, wenn die Topologie (anfänglich) gestartet wird. Lassen Sie zu, dass die Multimaster-Topologie alle neu gestarteten Domänen mit einer aktuellen Kopie der in anderen Domänen der Topologie gespeicherten Daten aktualisiert. Nach dem Start von Domänen ist die Multimaster-Topologie für die Synchronisation der Domänen zuständig.

- **Keine Unterstützung für EntityManager**

  Ein MapSet, das eine Entitäts-Map enthält, wird in Domänen nicht repliziert.
- **Keine Unterstützung für Bytefeldgruppen-Maps**

  Ein MapSet, das eine Map enthält, die mit COPY_TO_BYTES konfiguriert ist, wird in Domänen nicht repliziert.
- **Keine Unterstützung für Write-behind-Operationen**

  Ein MapSet, das eine Map enthält, die mit Write-behind-Unterstützung konfiguriert ist, wird in Domänen nicht repliziert.

Datenbankintegration: Write-behind, Inline- und Neben-Caching

WebSphere eXtreme Scale wird als Front-End für eine traditionelle Datenbank verwendet und macht Leseaktivitäten überflüssig, die normalerweise an die Datenbank übertragen werden. Ein kohärenter Cache kann direkt oder indirekt über einen ORM (Object Relational Mapper) mit einer Anwendung verwendet werden. Der kohärente Cache kann dann die Datenbank bzw. das Back-End von Leseaktivitäten entlasten. In einem geringfügig komplexeren Szenario, wie z. B. beim transaktionsorientierten Zugriff auf einen Datenbestand, in dem nur einige der Daten traditionelle Persistenzgarantien erfordern, können Sie Filter verwenden, um selbst die Schreibtransaktionen auszulagern.

Sie können eXtreme Scale als hoch flexiblen speicherinternen Datenbankverarbeitungsbereich konfigurieren. eXtreme Scale ist jedoch kein ORM. Das Produkt weiß nicht, woher die Daten in eXtreme Scale stammen. Eine Anwendung oder ein ORM kann Daten in einem eXtreme-Scale-Server ablegen. Die Datenquelle ist dafür verantwortlich sicherzustellen, dass sie mit der Datenbank, aus der die Daten stammen, konsistent bleibt. Das bedeutet, dass eXtreme Scale Daten, die automatisch aus einer Datenbank extrahiert werden, nicht ungültig machen kann. Die Anwendung bzw. der Mapper muss diese Funktion bereitstellen und die Daten verwalten, die in eXtreme Scale gespeichert werden.

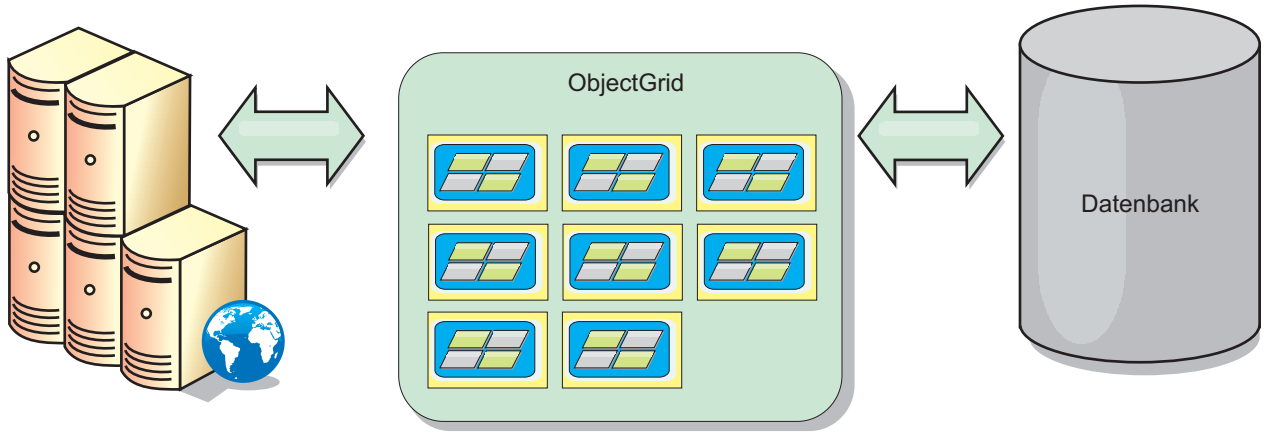


Abbildung 17. ObjectGrid als Datenbankpuffer

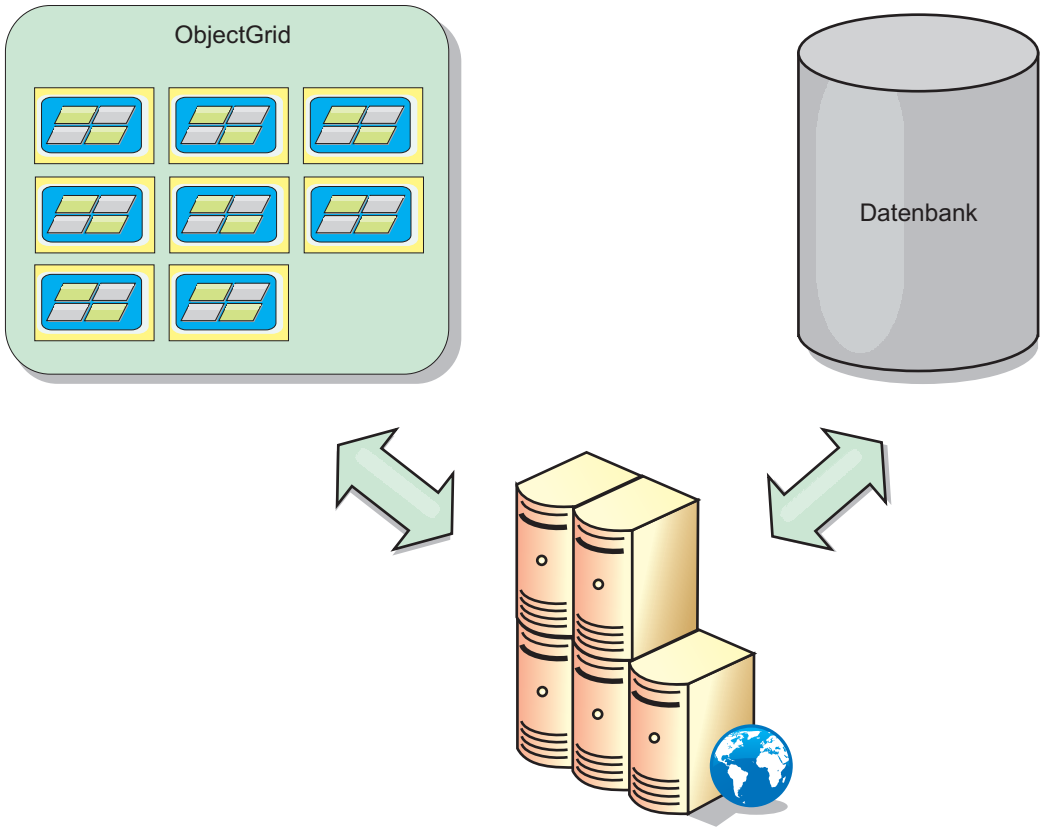


Abbildung 18. ObjectGrid als Nebencache

Teilcache und vollständiger Cache

WebSphere eXtreme Scale kann als Teilcache oder als vollständiger Cache eingesetzt werden. In einem Teilcache wird nur ein Teil der gesamten Daten gespeichert, wohingegen in einem vollständigen Cache alle Daten gespeichert werden. Ein Teilcache kann nach und nach bedarfsgesteuert gefüllt werden. Der Zugriff auf Teilcaches erfolgt gewöhnlich über Schlüssel (und nicht über Indizes oder Abfragen), da die Daten nur teilweise verfügbar sind.

Wenn ein Schlüssel nicht vorhanden ist (Cachefehler), wird die nächste Schicht aufgerufen, und die Daten werden abgerufen und in die entsprechende Cacheschicht eingefügt. Bei der Verwendung einer Abfrage oder eines Index wird nur auf die derzeit geladenen Werte zugegriffen, und die Anforderungen werden nicht an die anderen Schichten weitergeleitet. Ein vollständiger Cache enthält alle erforderlichen Daten, und der Zugriff kann über Attribute ohne Schlüsselfunktion mit Indizes oder Abfragen erfolgen.

Ein vollständiger Cache wird mit Daten gefüllt, bevor er von den Anwendungen verwendet wird, und kann als effizienter Datenbankersatz eingesetzt werden. Nach dem Laden der Daten kann der Cache ähnlich wie eine Datenbank behandelt werden. Da alle Daten verfügbar sind, können Abfragen und Indizes verwendet werden, um Daten zu suchen und zusammenzufassen.

Nebencache und integrierter Cache

WebSphere eXtreme Scale wird für die Unterstützung integrierten Cachings für ein Datenbank-Back-End oder als Nebencache für eine Datenbank verwendet. Beim integrierten Caching wird eXtreme Scale als primäres Mittel für die Interaktion mit den Daten verwendet. Wenn eXtreme Scale als Nebencache verwendet wird, wird das Back-End zusammen mit eXtreme Scale verwendet.

Nebencache

eXtreme Scale kann als Nebencache für die Datenzugriffsschicht einer Anwendung verwendet werden. In diesem Szenario wird eXtreme Scale verwendet, um Objekte, die normalerweise aus einer Back-End-Datenbank abgerufen werden, vorübergebend zu speichern. Anwendungen prüfen, ob eXtreme Scale die gewünschten Daten enthält. Wenn die Daten dort vorhanden sind, werden die Daten an den Aufrufenden zurückgegeben. Sind die Daten nicht dort vorhanden, werden die Daten vom Back-End abgerufen und in eXtreme Scale eingefügt, so dass bei der nächsten Anforderung die zwischengespeicherte Kopie verwendet werden kann. Die folgende Abbildung veranschaulicht, wie eXtreme Scale als Nebencache über eine beliebige Datenzugriffsschicht wie OpenJPA oder Hibernate verwendet werden kann.

Nebencache-Plug-ins für Hibernate und OpenJPA

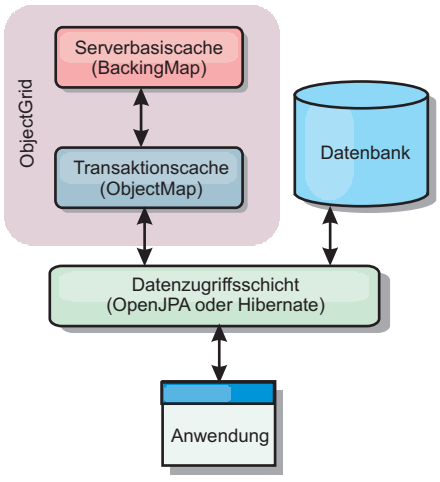


Abbildung 19. Nebencache

Cache-Plug-ins für OpenJPA und Hibernate sind in eXtreme Scale enthalten. Mit diesen Plug-ins können Sie eXtreme Scale als automatischen Nebencache verwen-

den. Durch die Verwendung von eXtreme Scale als Cacheprovider kann die Leistung beim Lesen und Abfragen von Daten verbessert und die Belastung der Datenbank verringert werden. eXtreme Scale bietet im Vergleich mit integrierten Cacheimplementierungen verschiedene Vorteile, weil der Cache automatisch in allen Prozessen repliziert wird. Wenn ein Client einen Wert zwischenspeichert, können alle anderen Clients den zwischengespeicherten Wert nutzen.

Integrierter Cache

Bei der Verwendung als integrierter Cache interagiert eXtreme Scale über ein Loader-Plug-in mit dem Back-End. Dieses Szenario kann den Datenzugriff vereinfachen, indem Anwendungen der direkte Zugriff auf die APIs von eXtreme Scale erlaubt wird. Es werden verschiedene Caching-Szenarien in eXtreme Scale unterstützt, um sicherzustellen, dass die Daten im Cache und die Daten im Back-End synchronisiert sind. Die folgende Abbildung veranschaulicht, wie ein integrierter Cache mit der Anwendung und dem Back-End interagiert.

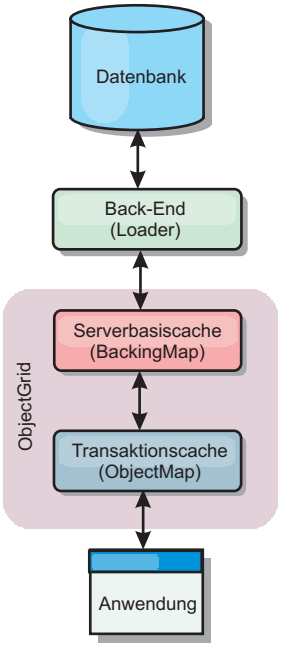


Abbildung 20. Integrierter Cache

Integriertes Caching

Beim integrierten Caching wird eXtreme Scale als primäres Mittel für die Interaktion mit den Daten verwendet. Bei der Verwendung von eXtreme Scale als integriertem Cache interagiert die Anwendung über ein Loader-Plug-in mit dem Back-End.

Die Option für integriertes Caching vereinfacht den Datenzugriff, weil sie Anwendungen den direkten Zugriff auf die eXtreme-Scale-APIs ermöglicht. WebSphere eXtreme Scale unterstützt mehrere Szenarien mit integriertem Caching:

- Read-through
- Write-Through
- Write-behind

Szenario mit Read-through-Caching

Ein Read-through-Cache ist ein Teilcache, in den nach und nach Dateneinträge nach Schlüssel geladen werden, wenn diese angefordert werden. Dies geschieht, ohne dass der Aufrufende wissen muss, wie die Einträge geladen werden. Wenn die Daten nicht im eXtreme-Scale-Cache gefunden werden, ruft eXtreme Scale die fehlenden Daten vom Loader-Plug-in ab, das die Daten aus der Back-End-Datenbank lädt und in den Cache einfügt. Nachfolgende Anforderungen für denselben Datenschlüssel werden im Cache gefunden, bis der Eintrag gelöscht, ungültig gemacht oder durch Bereinigung entfernt wird.

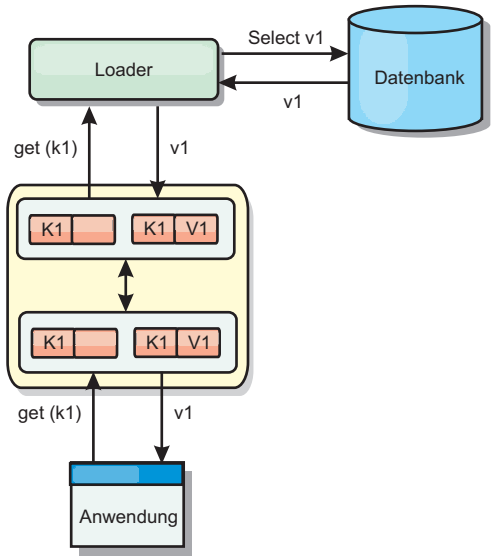


Abbildung 21. Read-through-Caching

Szenario mit Write-Through-Caching

In einem Write-Through-Cache (Durchschreibcache) erfolgt bei jedem Schreibvorgang in den Cache ein synchroner Schreibvorgang über den Loader in die Datenbank. Diese Methode gewährleistet die Konsistenz mit dem Back-End, verringert aber die Schreibleistung, weil die Datenbankoperation synchron erfolgt. Da der Cache und die Datenbank beide aktualisiert werden, werden bei nachfolgenden Leseoperationen dieselben Daten im Cache gefunden und Datenbankaufrufe vermieden. Ein Write-Through-Cache wird häufig in Kombination mit einem Read-through-Cache verwendet.

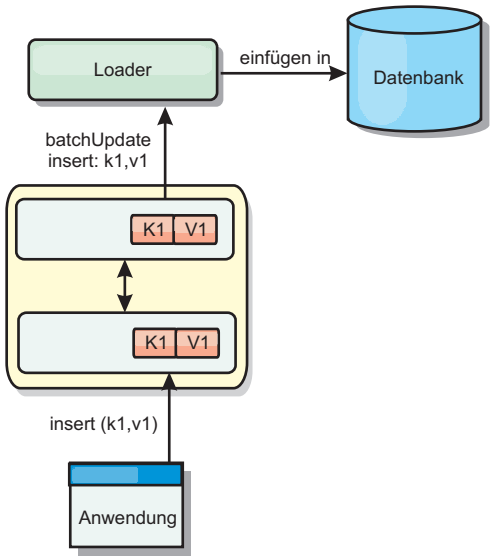


Abbildung 22. Write-Through-Caching

Szenario mit Write-behind-Caching

Die Datenbanksynchronisation kann verbessert werden, indem Änderungen asynchron geschrieben werden. Dies wird als Write-behind- oder Write-back-Cache (Rückschreibcache) bezeichnet. Änderungen, die normalerweise synchron in den Loader geschrieben werden, werden stattdessen in eXtreme Scale gepuffert und über einen Hintergrund-Thread in die Datenbank geschrieben. Die Schreibleistung wird erheblich verbessert, weil die Datenbankoperation aus der Clienttransaktion entfernt wird und die Schreibvorgänge in die Datenbank komprimiert werden können. Weitere Informationen finden Sie im Abschnitt „Write-behind-Caching” auf Seite 39.

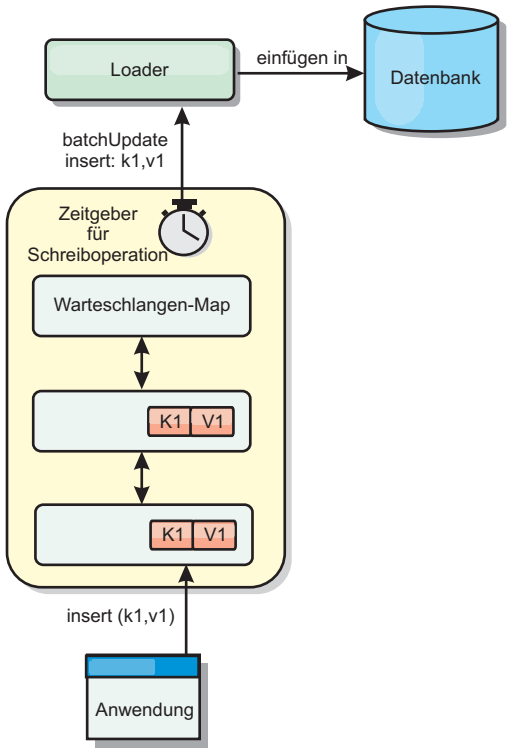


Abbildung 23. Write-behind-Caching

Weitere Informationen finden Sie im Abschnitt „Write-behind-Caching".

Write-behind-Caching

Sie können Write-behind-Caching verwenden, um die Kosten für die Aktualisierung einer Datenbank, die Sie als Back-End verwenden, zu reduzieren.

Einführung

Beim Write-behind-Caching werden Aktualisierungen für das Loader-Plug-in asynchron in die Warteschlange eingereiht. Sie können die Leistung von Aktualisierungs-, Einfüge- und Entfernungsoperationen für die Map verbessern, indem Sie die eXtreme-Scale-Transaktion von der Datenbanktransaktion entkoppeln. Die asynchrone Aktualisierung wird nach einer zeitbasierten Verzögerung (z. B. fünf Minuten) oder einer eintragsbasierten Verzögerung (z. B. 1000 Einträge) durchgeführt.

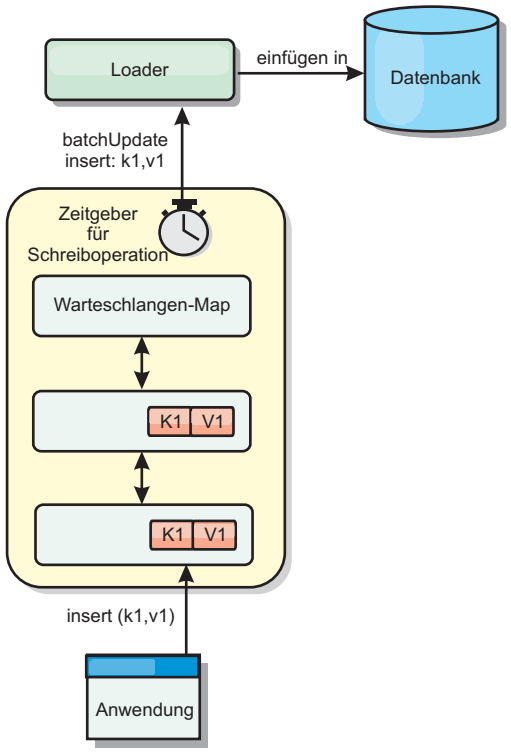


Abbildung 24. Write-behind-Caching

Bei der Write-behind-Konfiguration in einer BackingMap wird ein Thread zwischen dem Loader (Ladeprogramm) und der Map erstellt. Anschließend delegiert der Loader Datenanforderungen über den Thread gemäß den Konfigurationseinstellungen in der Methode "BackingMap.setWriteBehind". Wenn eine eXtreme-Scale-Transaktion einen Eintrag in einer Map einfügt, aktualisiert oder entfernt, wird ein LogElement-Objekt für jeden dieser Datensätze erstellt. Diese Elemente werden an den Write-behind-Loader gesendet und in eine spezielle ObjectMap, eine so genannte Warteschlangen-Map, eingereiht. Jede BackingMap mit aktivierter Write-behind-Einstellung hat ihre eigenen Warteschlangen-Maps. Ein Write-behind-Thread entfernt die in die Warteschlange eingereihten Daten aus den Warteschlangen-Maps und überträgt Sie mit Push in den echten Back-End-Loader.

Der Write-behind-Loader sendet nur LogElement-Objekte der Typen "insert" (Einfügen), "update" (Aktualisieren) und "delete" (Löschen) an den echten Loader. Alle anderen Typen von LogElement-Objekten, wie z. B. EVICT, werden ignoriert.

Vorteile

Das Aktivieren der Write-behind-Unterstützung hat die folgenden Vorteile:

- **Isolation von Back-End-Fehlern:** Durch das Write-behind-Caching können Back-End-Fehler isoliert werden. Wenn die Back-End-Datenbank ausfällt, werden Aktualisierungen in die Warteschlangen-Map eingereiht. Die Anwendungen können weiterhin Transaktionen an eXtreme Scale senden. Nach der Wiederherstellung des Back-Ends werden die Daten in der Warteschlangen-Map mit Push an das Back-End übertragen.
- **Geringere Back-End-Last:** Der Write-behind-Loader fasst die Aktualisierungen auf Schlüsselbasis so zusammen, dass nur eine einzige zusammenfasste Aktuali-

sierung pro Schlüssel in der Warteschlangen-Map vorhanden ist. Bei dieser Zusammenfassung verringert sich die Anzahl der Aktualisierungen für die Back-End-Datenbank.

- **Verbesserte Transaktionsleistung:** Die Zeiten einzelner eXtreme-Scale-Transaktionen verringern sich, weil sie nicht auf die Synchronisation der Daten mit dem Back-End warten müssen.

Hinweise zum Anwendungsdesign

Das Aktivieren der Write-behind-Unterstützung ist zwar einfach, aber eine Anwendung mit Write-behind-Unterstützung zu entwerfen, bedarf sorgfältiger Überlegungen. Ohne Write-behind-Unterstützung ist die Back-End-Transaktion in die ObjectGrid-Transaktion eingeschlossen. Die ObjectGrid-Transaktion wird vor der Back-End-Transaktion gestartet und endet erst nach Abschluss der Back-End-Transaktion.

Wenn die Write-behind-Unterstützung aktiviert ist, endet die ObjectGrid-Transaktion vor dem Start der Back-End-Transaktion. Die ObjectGrid-Transaktion und die Back-End-Transaktion sind entkoppelt.

Referenzielle Integritätsbedingungen

Jede BackingMap, die mit Write-behind-Unterstützung konfiguriert ist, hat einen eigenen Write-behind-Thread, der die Daten mit Push an das Back-End überträgt. Deshalb werden die Daten, die in einer einzigen ObjectGrid-Transaktion in verschiedenen Maps aktualisiert wurden, im Back-End in verschiedenen Back-End-Transaktionen aktualisiert. Beispiel: Transaktion T1 aktualisiert den Schlüssel "key1" in der Map "Map1" und den Schlüssel "key2" in der Map "Map2". Die Aktualisierungen von Schlüssel key1 in der Map Map1 und von Schlüssel "key2" in der Map "Map2" werden in einer jeweils anderen Back-End-Transaktion von einem jeweils anderen Write-behind-Thread durchgeführt. Wenn es Beziehungen zwischen den in Map1 und Map2 gespeicherten Daten, wie z. B. Integritätsbedingungen über Fremdschlüssel, im Back-End gibt, können die Aktualisierungen fehlschlagen.

Beim Design der referenziellen Integritätsbedingungen in Ihrer Back-End-Datenbank müssen Sie sicherstellen, dass solche nicht ausführbaren Aktualisierungen zugelassen werden.

Sperrverhalten von Warteschlangen-Maps

Ein weiterer wichtiger Unterschied im Transaktionsverhalten ist das Sperrverhalten. ObjectGrid unterstützt drei verschiedene Sperrstrategien: PESSIMISTIC (Pessimistisch), OPTIMISITIC (Optimistisch) und NONE (Keine). Die Write-behind-Warteschlangen-Map verwendet die pessimistische Sperrstrategie, unabhängig davon, welche Sperrstrategie für die zugehörige BackingMap konfiguriert ist. Es gibt zwei verschiedene Typen von Operationen, die eine Sperre für die Warteschlangen-Map anfordern:

- Wenn eine ObjectGrid-Transaktion festgeschrieben wird oder eine Flush-Operation (Map-Flush oder Sitzungs-Flush) stattfindet, liest die Transaktion den Schlüssel in der Warteschlangen-Map und setzt eine S-Sperre für den Schlüssel.
- Wenn eine ObjectGrid-Transaktion festgeschrieben wird, versucht die Transaktion die S-Sperre für den Schlüssel in eine X-Sperre zu aktualisieren.

Anhand dieses zusätzlichen Verhaltens für die Warteschlangen-Map sind einige Unterschiede im Sperrverhalten erkennbar.

- Wenn die Benutzer-Map mit einer pessimistischen Sperrstrategie konfiguriert ist, sind die Unterschiede im Sperrverhalten nicht gravierend. Bei jedem Aufruf einer Flush- oder Festschreiboperation (Commit) wird eine S-Sperre für denselben Schlüssel in der Warteschlangen-Map gesetzt. Während der Festschreibung wird nicht nur eine X-Sperre für den Schlüssel in der Benutzer-Map, sondern auch für den Schlüssel in der Warteschlangen-Map angefordert.
- Wenn die Benutzer-Map mit einer optimistischen Sperrstrategie oder ohne Sperrstrategie konfiguriert ist, folgt die Benutzertransaktion dem Muster der pessimistischen Sperrstrategie. Bei jedem Aufruf einer Flush- oder Festschreiboperation (Commit) wird eine S-Sperre für denselben Schlüssel in der Warteschlangen-Map angefordert. Während der Festschreibung wird in derselben Transaktion eine X-Sperre für den Schlüssel in der Warteschlangen-Map angefordert.

Transaktionswiederholungen im Loader

ObjectGrid unterstützt keine zweiphasigen Transaktionen und keine XA-Transaktionen. Der Write-behind-Thread entfernt Datensätze aus der Warteschlangen-Map und aktualisiert die Datensätze im Back-End. Wenn der Server mitten in der Transaktion ausfällt, können einige Back-End-Aktualisierungen verloren gehen.

Der Write-behind-Loader versucht automatisch, fehlgeschlagene Transaktionen erneut zu schreiben, und sendet eine unbestätigte Protokollfolge an das Back-End, um einen Datenverlust zu verhindern. Diese Aktion erfordert, dass der Loader idempotent ist, d. h., wenn Loader.batchUpdate(TxId, LogSequence) zweimal mit demselben Wert aufgerufen wird, liefen diese Aufrufe dasselbe Ergebnis wie ein einmaliger Aufruf. Loader-Implementierungen müssen zum Aktivieren dieses Features die Schnittstelle "RetryableLoader" implementieren. Weitere Einzelheiten finden Sie in der API-Dokumentation.

Ausfall des Loaders

Das Loader-Plug-in kann ausfallen, wenn es nicht mit dem Datenbank-Back-End kommunizieren kann. Dies kann passieren, wenn der Datenbankserver oder die Netzverbindung inaktiv ist. Der Write-behind-Loader reiht die Aktualisierungen in eine Warteschlange ein und versucht anschließend in regelmäßigen Abständen, die Datenänderungen mit Push an den Loader zu übertragen. Der Loader muss die ObjectGrid-Laufzeitumgebung darüber benachrichtigen, dass ein Problem mit der Datenbankkonnektivität vorliegt, indem es eine Ausnahme vom Typ "LoaderNotAvailableException" auslöst.

Deshalb muss die Loader-Implementierung in der Lage sein, einen Datenfehler von einem physischen Ausfall des Loaders zu unterscheiden. Bei Datenfehlern muss eine Ausnahme des Typs "LoaderException" oder "OptimisticCollisionException" ausgelöst bzw. erneut ausgelöst werden, aber beim physischen Ausfall des Loaders muss eine Ausnahme des Typs "LoaderNotAvailableException" ausgelöst werden. ObjectGrid behandelt diese beiden Ausnahmen auf unterschiedliche Weise:

- Wenn der Write-behind-Loader eine Ausnahme vom Typ "LoaderException" abfängt, geht er von einem Datenfehler aus, z. B. von einem doppelten Schlüssel. Der Write-behind-Loader löst den Aktualisierungsstapel auf und versucht, einen Datensatz nach dem anderen zu aktualisieren, um den Datenfehler zu isolieren. Wird bei dieser Aktualisierung auf Datensatzbasis erneut eine Ausnahme vom Typ "LoaderException" abgefangen, wird ein Datensatz zur fehlgeschlagenen Aktualisierung erstellt und in der Map für fehlgeschlagene Aktualisierungen protokolliert.

- Wenn der Write-behind-Loader eine Ausnahme vom Typ "LoaderNotAvailableException" abfängt, geht er von einem Ausfall aus, weil er keine Verbindung zum Datenbank-Back-End herstellen kann, z. B., weil das Datenbank-Back-End inaktiv ist, keine Datenbankverbindung verfügbar oder das Netz inaktiv ist. Der Write-behind-Loader wartet 15 Sekunden und versucht dann erneut, die Datenbankaktualisierung im Stapelbetrieb durchzuführen.

Häufig wird der Fehler gemacht, eine Ausnahme vom Typ "LoaderException" auszulösen, obwohl eigentlich eine Ausnahme vom Typ "LoaderNotAvailableException" ausgelöst werden müsste. Alle Datensätze, die in die Warteschlange für den Write-behind-Loader eingereiht sind, werden als Datensätze für eine fehlgeschlagene Aktualisierung markiert, was den eigentlich Zweck der Isolierung von Back-End-Fehlern zunichte macht.

Leistungsaspekte

Die Unterstützung des Write-behind-Cachings erhöht die Antwortzeiten, weil die Loader-Aktualisierung aus der Transaktion entfernt wird. Außerdem erhöht sich der Datenbankdurchsatz, weil Datenbankaktualisierungen kombiniert werden. Es ist wichtig, die Kosten zu kennen, die durch den Write-behind-Thread anfallen, der die Daten aus der Warteschlangen-Map extrahiert und mit Push an den Loader überträgt.

Die maximale Aktualisierungsanzahl und die maximale Aktualisierungszeit müssen den erwarteten Verwendungsmustern und der Umgebung entsprechend angepasst werden. Wenn der Wert für die maximale Aktualisierungsanzahl oder der Wert für die maximale Aktualisierungszeit zu klein gewählt wird, kann der Write-behind-Threads mehr Kosten verursachen, als er Vorteile bringt. Wenn ein sehr hoher Wert für diese beiden Parameter festgelegt wird, ist es möglich, dass die Speicherbelegung aufgrund der Einreihung der Daten zunimmt und veraltete Datensätze länger in der Datenbank verbleiben.

Um die beste Leistung zu erzielen, sollten Sie bei der Optimierung der Write-behind-Parameter die folgenden Faktoren berücksichtigen:

- Verhältnis zwischen Lese- und Schreibtransaktionen
- Aktualisierungsintervall für dieselben Datensätze
- Latenzzeit für Datenbankaktualisierung

Loader

Mit einem Loader-Plug-in von eXtreme Scale kann sich eine eXtreme-Scale-Map wie ein Speichercache für Daten verhalten, die gewöhnlich in einem persistenten Speicher auf demselben System oder einem anderen System verwaltet 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 eXtreme Scale ermöglicht. Ein Loader enthält die Logik für das Lesen aus einem und das Schreiben in einem persistenten Speicher.

Übersicht

Loader (Ladeprogramme) sind BackingMap-Plug-ins, die aufgerufen werden, wenn Änderungen an der BackingMap vorgenommen werden oder wenn die BackingMap eine Datenanforderung nicht bedienen kann (Cachefehler). Der Loader wird aufgerufen, wenn der Cache eine Anforderung für einen Schlüssel nicht bedienen kann. Er unterstützt Read-through-Funktionen und eine verzögerte Füllung des

Caches. Ein Loader lässt außerdem Aktualisierungen in der Datenbank zu, wenn sich Cachewerte ändern. Alle Änderungen in einer Transaktion werden gruppiert, um die Anzahl der Datenbankinteraktionen zu minimieren. Zusammen mit dem Loader wird ein TransactionCallback-Plug-in verwendet, um die Abgrenzung der Back-End-Transaktion auszulösen. Die Verwendung dieses Plug-ins ist wichtig, wenn mehrere Maps an einer einzelnen Transaktion beteiligt sind oder wenn Transaktionsdaten ohne Festschreibung mit Flush in den Cache übertragen werden.

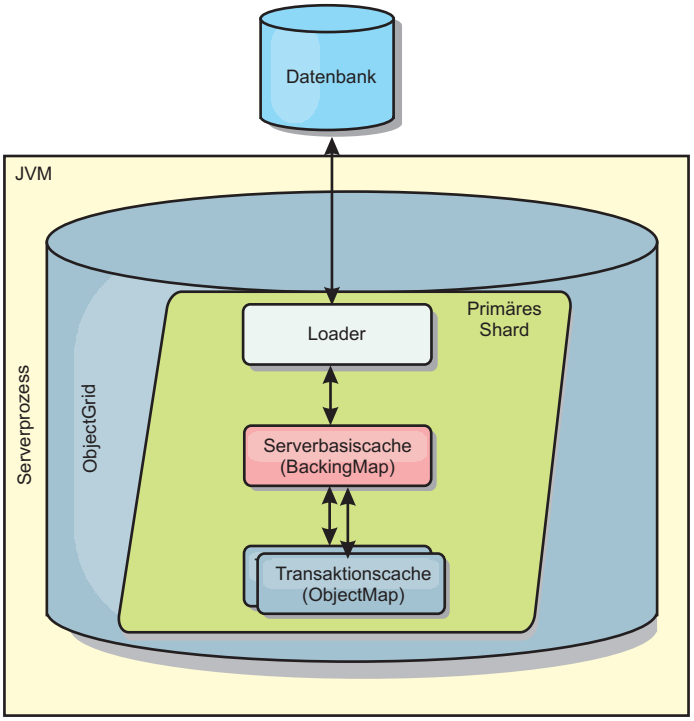


Abbildung 25. Loader

Der Loader kann auch überqualifizierte Aktualisierungen verwenden, um keine Datenbanksperren halten zu müssen. Anhand eines im Cachewert gespeicherten Versionsattributs kann der Loader das Vorher- und Nachher-Abbild des Werts erkennen, wenn dieser im Cache aktualisiert wird. Dieser Wert kann anschließend bei der Aktualisierung der Datenbank bzw. des Back-Ends verwendet werden, um sicherzustellen, dass die Daten nicht aktualisiert wurden. Ein Loader kann auch so konfiguriert werden, dass das Grid beim Start vorher geladen wird. Wenn mit Partitionierung gearbeitet wird, wird jeder Partition eine Loader-Instanz zugeordnet. Hat die Map "Company" beispielsweise zehn Partitionen, gibt es zehn Loader-Instanzen, eine für jede primäre Partition. Bei der Aktivierung des primären Shards für die Map wird die Methode "preloadMap" für den Loader synchron oder asynchron aufgerufen. Dies ermöglicht das automatische Laden von Daten aus dem Back-End in die Map-Partition. Wenn die Methode synchron aufgerufen wird, werden alle Clienttransaktionen blockiert, um einen inkonsistenten Zugriff auf das Grid zu verhindern. Alternativ kann ein Client-Preloader zum Laden des vollständigen Grids verwendet werden.

Es gibt zwei integrierte Loader, die die Integration mit relationalen Datenbank-Back-Ends erheblich vereinfachen. Die JPA-Loader nutzen die ORM-Funktionen (Object-Relational Mapping, objektbezogene Zuordnung) der OpenJPA- und Hibernate-Implementierungen der Spezifikation Java Persistence API (JPA). Weitere In-

formationen finden Sie in den Informationen zu JPA-Loadern in der *Produktübersicht*.

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.

Weitere Informationen finden Sie im Abschnitt zum Schreiben eines Loaders in der *Produktübersicht*.

XML-Konfiguration für die Integration eines Loaders

Ein anwendungsdefinierter Loader kann über eine XML-Datei integriert werden. Das folgende Beispiel veranschaulicht, wie der Loader "MyLoader" in die BackingMap "map1" integriert wird. Sie müssen den Klassennamen für Ihren Loader, den Datenbanknamen und die Verbindungsdetails sowie die Eigenschaften für die Isolationsstufe angeben. Sie können dieselbe XML-Struktur verwenden, wenn Sie lediglich einen Preloader verwenden. Geben Sie in diesem Fall nur den Klassennamen des Preloaders an Stelle des vollständigen Loader-Klassennamens an.

Loader-Konfiguration mit 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="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 programmgesteuert integrieren

Sie können eine programmgesteuerte Konfiguration mit lokalen speicherinternen Grids verwenden. Das folgende Code-Snippet veranschaulicht, wie ein anwendungsdefinierter Loader in die BackingMap für map1 über die API "ObjectGrid" integriert wird:

Programmgesteuerte Konfiguration eines Loaders

```
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 des ObjectGrids 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.

- Der Benutzer "bob" wird als Benutzer von eXtreme Scale authentifiziert. Die Anwendung greift auf das Grid "mygrid" unter Verwendung des Persistenzeinheitennamens "DB2Hibernate" zu. Der Containerserver ist "XS_Server1". Das Ergebnis ist wie folgt:
  - **Benutzer**=bob
  - **WorkstationName**=XS_Server1,192.168.1.101
  - **Anwendungname**=mygrid,DB2Hibernate
  - **Abrechnungsinformationen**=1, DEFAULT,FE7954BD-0126-4000-E000-2298094151DB,com.ibm.db2.jcc.t4.b@71787178
- Benutzer "bob" wird mit einem WAS-Token authentifiziert. Die Anwendung greift auf das Grid "mygrid" unter Verwendung des Persistenzeinheitennamens "DB2OpenJPA" zu. Der Containerserver ist "XS_Server2". Das Ergebnis ist wie folgt:
  - **Benutzer**

    =acme.principal.UserPrincipal[Bob],acme.principal.GroupPrincipal[admin]
  - **Workstationname**=XS_Server2,192.168.1.102
  - **Anwendungsname**=mygrid,DB2OpenJPA
  - **Abrechnungsinformationen**=188,DEFAULT,FE72BC63-0126-4000-E000-851C092A4E33,com.ibm.ws.rsadapter.jdbc.WSJccSQLJConnection@2b432b43

Lesen Sie die Dokumentation zu DB2 Performance Expert, um mehr über die Überwachung des Datenbankzugriffs zu erfahren.

Vorheriges Laden von Daten und Vorbereitung

In vielen Szenarien, die die Verwendung eines Loaders (Ladeprogramms) beinhalten, können Sie Ihr Grid durch vorheriges Laden von Daten (Preload) vorbereiten.

Wenn das Grid als vollständiger Cache verwendet wird, muss das Grid alle Daten aufnehmen und geladen werden, bevor Clients eine Verbindung zum Grid herstellen können. Wenn das Grid als Teilcache verwendet wird, müssen Sie den Cache mit Daten vorbereiten (Aufwärmphase), so dass Clients sofortigen Zugriff auf die Daten haben, wenn sie eine Verbindung zum Grid herstellen.

Es gibt zwei Ansätze für das vorherige Laden von Daten in das Grid: Verwendung eines Loader-Plug-ins (Ladeprogramm) oder Verwendung eines Client-Loaders. Diese beiden Ansätze werden in den folgenden Abschnitten beschrieben.

Loader-Plug-in

Das Loader-Plug-in wird jeder Map zugeordnet und ist für die Synchronisation eines einzelnen primären Partitions-Shards mit der Datenbank zuständig. Die Methode "preloadMap" des Loader-Plug-ins wird automatisch aufgerufen, wenn ein Shard aktiviert wird. Wenn Sie beispielsweise 100 Partitionen haben, sind 100 Loader-Instanzen vorhanden, die jeweils die Daten für ihre Partition laden. Wenn die Loader-Instanzen synchron ausgeführt werden, werden alle Clients blockiert, bis das vorherige Laden der Daten (der so genannte Preload-Prozess) abgeschlossen ist.

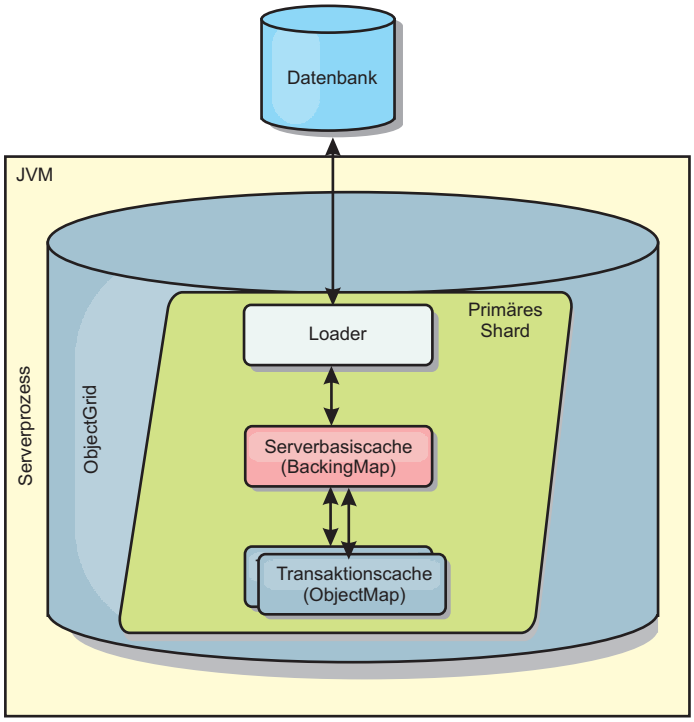


Abbildung 26. Loader-Plug-in

Weitere Informationen finden Sie in den Einzelheiten zur Verwendung von Loadern im *Programmierhandbuch*.

Client-Loader

Ein Client-Loader ist ein Muster für die Verwendung eines oder mehrerer Clients, um Daten in das Grid zu laden. Die Verwendung mehrerer Clients zum Laden von Grid-Daten kann effektiv sein, wenn das Partitionsschema nicht in der Datenbank gespeichert ist. Sie können Client-Loader manuell oder automatisch aufrufen, wenn das Grid gestartet wird. Client-Loader können optional die Schnittstelle "StateManager" verwenden, um den Status des Grids auf den Preload-Modus zu setzen, so dass Clients nicht auf das Grid zugreifen können, wenn das vorherige Laden der Daten in das Grid durchgeführt wird. WebSphere eXtreme Scale enthält einen JPA-basierten (Java Persistence API) Loader, den Sie verwenden können, um das Grid

automatisch über die OpenJPA- oder Hibernate-JPA-Provider zu laden.

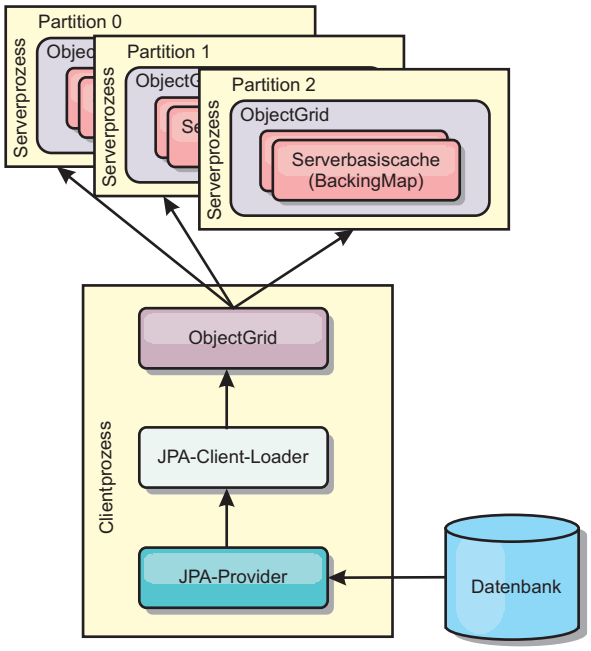


Abbildung 27. Client-Loader

Vorheriges Laden von Maps

Maps können so genannte Loader (Ladeprogramme) zugeordnet werden. Ein Loader wird verwendet, um Objekte abzurufen, wenn diese nicht in der Map gefunden werden (Cachefehler), und um Änderungen in ein Back-End zu schreiben, wenn eine Transaktion festgeschrieben wird. Loader können auch für das vorherige Laden von Daten (Preload) in eine Map verwendet werden. Die Methode "preloadMap" der Schnittstelle "Loader" wird für jede Map aufgerufen, wenn die zugehörige Partition im MapSet zu einem primären Shard wird. Die Methode "preloadMap" wird nicht für Replikate aufgerufen. Sie versucht, alle geplanten referenzierten Daten über die bereitgestellte Sitzung aus dem Back-End in die Map zu laden. Die jeweilige Map wird mit dem Argument "BackingMap" angegeben, das an die Methode "preloadMap" übergeben wird.

```
void preloadMap(Session session, BackingMap backingMap) throws LoaderException;
```

Vorheriges Laden in einem partitionierten MapSet

Maps können in N Partitionen partitioniert werden. Deshalb können Maps auf mehrere Server verteilt werden, wobei jeder Eintrag mit einem Schlüssel gekennzeichnet wird, der nur in einem einzigen dieser Server gespeichert wird. Sehr große Maps können in eXtreme Scale verwaltet werden, weil die Anwendung nicht mehr durch die Heap-Speichergröße einer einzigen JVM beschränkt ist, die alle Einträge einer Map enthält. Anwendungen, die den Preload-Prozess mit der Methode "preloadMap" der Schnittstelle "Loader" ausführen möchten, müssen den Teil der Daten angeben, die vorher geladen werden sollen. Es ist immer eine feste Anzahl an Partitionen vorhanden. Sie können diese Zahl anhand des folgenden Codebeispiels bestimmen:

```
int numPartitions = backingMap.getPartitionManager().getNumOfPartitions();
int myPartition = backingMap.getPartitionId();
```

Dieses Codebeispiel zeigt, wie eine Anwendung den Teil der Daten angeben kann, der vorher aus der Datenbank geladen werden soll. Anwendungen müssen diese Methoden auch dann verwenden, wenn die Map zunächst nicht partitioniert ist. Diese Methoden bieten Flexibilität: Wenn die Map später von den Administratoren partitioniert wird, funktioniert der Loader weiterhin ordnungsgemäß.

Die Anwendung muss Abfragen absetzen, um den Teil *myPartition* aus dem Back-End abzurufen. Wenn eine Datenbank verwendet wird, kann es unter Umständen einfacher sein, eine Spalte mit der Partitionskennung für einen bestimmten Datensatz zu haben, sofern es keine natürliche Abfrage gibt, mit der die Daten in der Tabelle einfach partitioniert werden können.

Ein Beispiel für die Implementierung eines Loaders für eine replizierte eXtreme Scale-Umgebung finden Sie im Abschnitt zum Schreiben eines Loaders mit einem Preload-Controller für Replikate im *Programmierhandbuch*.

Leistung

Die Preload-Implementierung kopiert Daten aus dem Back-End in die Map, indem sie mehrere Objekte in der Map in einer einzigen Transaktion speichert. Die optimale Anzahl der pro Transaktion zu speichernden Datensätze richtet sich nach mehreren Faktoren, einschließlich der Komplexität und der Größe. Wenn die Transaktion beispielsweise Blöcke mit mehr als 100 Einträgen enthält, nehmen die Leistungsgewinne ab, wenn Sie die Anzahl der Einträge erhöhen. Zur Bestimmung der optimalen Anzahl beginnen Sie mit 100 Einträgen, und erhöhen Sie dann die Anzahl, bis keine Leistungsgewinne mehr zu verzeichnen sind. Mit größeren Transaktionen kann eine bessere Replikationsleistung erzielt werden. Denken Sie daran, dass der Preload-Code nur im primären Shard ausgeführt wird. Die vorher geladenen Daten werden über das primäre Shard in allen Replikaten repliziert, die online sind.

MapSets vorher laden

Wenn die Anwendung ein MapSet mit mehreren Maps verwendet, hat jede Map einen eigenen Loader. Jeder Loader besitzt eine Preload-Methode. Alle Maps werden nacheinander von eXtreme Scale geladen. Es kann effizienter sein, den Preload-Prozess für die Maps so zu gestalten, dass eine einzige Map als Map bestimmt wird, in die die Daten vorher geladen werden. Dieser Prozess ist eine Anwendungskonvention. Beispiel: Die beiden Maps "department" (Abteilung) und "employee" (Mitarbeiter) könnten beide den Loader der Map "department" verwenden. Bei dieser Prozedur wird über Transaktionen gewährleistet, dass in dem Fall, dass eine Anwendung eine Abteilung abrufen möchte, sich die Mitarbeiter für diese Abteilung im Cache befinden. Wenn der Loader der Map "department" eine Abteilung vorher aus dem Back-End lädt, ruft er auch die Mitarbeiter für diese Abteilung ab. Das department-Objekt und die zugehörigen employee-Objekte werden dann der Map in einer einzigen Transaktion hinzugefügt.

Wiederherstellbares vorheriges Laden

Einige Kunden haben sehr große Datenmengen, die zwischengespeichert werden müssen. Das vorherige Laden dieser Daten kann sehr zeitaufwendig sein. Manchmal muss das vorherige Laden abgeschlossen sein, bevor die Anwendung online gehen kann. In diesem Fall können Sie von einem wiederherstellbaren vorherigen Laden profitieren. Angenommen, es gibt Millionen Datensätze, die vorher geladen werden müssen. Die Daten werden vorab in das primäre Shard geladen, und der Prozess scheitert bei Datensatz 800.000. Normalerweise löscht das als neue primäre

Shard ausgewählte Replikat den Replikationsstatus und beginnt von vorne. eXtreme Scale kann eine Schnittstelle "ReplicaPreloadController" verwenden. Der Loader für die Anwendung muss auch die Schnittstelle "ReplicaPreloadController" implementieren. Das folgende Beispiel fügt dem Loader eine einzige Methode hinzu: `Status checkPreloadStatus(Session session, BackingMap bmap);`. Diese Methode wird von der Laufzeitumgebung von eXtreme Scale aufgerufen, bevor die Methode "preload" der Schnittstelle "Loader" aufgerufen wird. eXtreme Scale prüft das Ergebnis dieser Methode (Status), um das Verhalten festzulegen, wenn ein Replikat in ein primäres Shard hochgestuft werden muss.

Tabelle 4. Statuswert und Antwort

Zurückgegebener Statuswert	Antwort von eXtreme Scale
Status.PRELOADED_ALREADY	eXtreme Scale ruft die Methode "preload" gar nicht auf, weil dieser Statuswert anzeigt, dass der Preload-Prozess für die Map bereits vollständig abgeschlossen ist.
Status.FULL_PRELOAD_NEEDED	eXtreme Scale löscht die Map und ruft ganz regulär die Methode "preload" auf.
Status.PARTIAL_PRELOAD_NEEDED	eXtreme Scale belässt die Map im aktuellen Zustand und ruft die Methode "preload" auf. Diese Strategie ermöglicht dem Loader der Anwendung, den Preload-Prozess an der Stelle fortzusetzen, an der er zuvor abgebrochen wurde.

Es ist logisch, dass ein primäres Shard beim vorherigen Laden von Daten in die Map einen Status in einer Map im MapSet hinterlassen muss, das repliziert wird, so dass das Replikat den zurückzugebenden Status bestimmen kann. Sie können dazu eine zusätzliche Map verwenden, die z. B. den Namen RecoveryMap hat. Diese RecoveryMap muss zu demselben MapSet gehören, das vorher geladen wird, um sicherzustellen, dass die replizierte Map mit den vorher geladenen Daten konsistent ist. Eine empfohlene Implementierung folgt.

Während der Preload-Prozess die einzelnen Datensatzblöcke festschreibt, aktualisiert er im Rahmen dieser Transaktion auch einen Zähler oder Wert in der RecoveryMap. Die vorher geladenen Daten und die RecoveryMap-Daten werden automatisch in den Replikaten repliziert. Wenn das Replikat in ein primäres Shard hochgestuft wird, kann es anhand der RecoveryMap prüfen, was passiert ist.

Die RecoveryMap kann einen einzigen Eintrag mit dem Statusschlüssel enthalten. Wenn kein Objekt für diesen Schlüssel vorhanden ist, müssen Sie einen vollständigen Preload-Prozess durchführen (checkPreloadStatus gibt FULL_PRELOAD_NEEDED zurück). Wenn ein Objekt für diesen Statusschlüssel vorhanden ist und der Wert COMPLETE lautet, ist der Preload-Prozess abgeschlossen, und die Methode "checkPreloadStatus" gibt PRELOADED_ALREADY zurück. Andernfalls zeigt das Wertobjekt an, wo der Preload-Prozess erneut gestartet werden muss, und die Methode "checkPreloadStatus" gibt PARTIAL_PRELOAD_NEEDED zurück. Der Loader kann den Wiederherstellungspunkt in einer Instanzvariablen speichern, so dass der Loader beim Aufruf der Methode "preload" diesen Ausgangspunkt kennt. Die RecoveryMap kann auch einen Eintrag pro Map enthalten, wenn jede Map gesondert vorher geladen wird.

Handhabung der Wiederherstellung im synchronen Replikationsmodus mit einem Loader

Die Laufzeitumgebung von eXtreme Scale ist so konzipiert, dass festgeschriebene Daten beim Ausfall des primären Shards nicht verloren gehen. Im folgenden Ab-

schnitt werden der verwendeten Algorithmen beschrieben. Diese Algorithmen gelten nur, wenn eine Replikationsgruppe die synchrone Replikation verwendet. Ein Loader ist optional.

Die Laufzeitumgebung von eXtreme Scale kann so konfiguriert werden, dass alle Änderungen in einem primären Shard synchron in den Replikaten repliziert werden. Wenn ein synchrones Replikat verteilt wird, erhält es eine Kopie der vorhandenen Daten im primären Shard. In dieser Zeit empfängt das primäre Shard weiterhin Transaktionen und kopiert sie asynchron in das Replikat. Das Replikat wird in dieser Zeit nicht als online eingestuft.

Wenn das Replikat denselben Stand wie das primäre Shard hat, wechselt das Replikat in den Peer-Modus, und die synchrone Replikation beginnt. Jede im primären Shard festgeschriebene Transaktion wird an die synchronen Replikate gesendet, und das primäre Shard wartet auf eine Antwort jedes Replikats. Eine synchrone Festschreibungsfolge mit einem Loader im primären Shard setzt sich aus den folgenden Schritten zusammen:

Tabelle 5. Festschreibungsfolge im primären Shard

Schritt mit Loader	**Schritt ohne Loader**
Sperren für Einträge abrufen	Identisch
Änderung mit Flush an den Loader übertragen	Nulloperation
Änderungen im Cache speichern	Identisch
Änderungen an Replikate senden und auf Bestätigung warten	Identisch
Festschreibung im Loader über das TransactionCallback-Plug-in	Methode "commit" des Plug-ins wird zwar aufgerufen, führt aber keine Aktion aus
Sperren für Einträge freigeben	Identisch

Beachten Sie, dass die Änderungen an das Replikat gesendet werden, bevor sie im Loader festgeschrieben werden. Um zu bestimmen, wann die Änderungen im Replikat festgeschrieben werden, ändern Sie diese Folge. Initialisieren Sie während der Initialisierung die Transaktionslisten im primären Shard wie folgt:

```
CommitedTx = {}, RolledBackTx = {}
```

Für eine synchrone Commit-Verarbeitung verwenden Sie die folgende Folge:

Tabelle 6. Synchrone Commit-Verarbeitung

Schritt mit Loader	**Schritt ohne Loader**
Sperren für Einträge abrufen	Identisch
Änderung mit Flush an den Loader übertragen	Nulloperation
Änderungen im Cache speichern	Identisch
Änderungen mit einer festgeschriebenen Transaktion senden, Rollback der Transaktion an das Replikat durchführen und auf Bestätigung warten	Identisch
Liste festgeschriebener und rückgängig gemachter Transaktionen löschen	Identisch

Tabelle 6. Synchrone Commit-Verarbeitung (Forts.)

Schritt mit Loader	Schritt ohne Loader
Festschreibung im Loader über das TransactionCallBack-Plug-in	Methode des TransactionCallBack-Plug-ins wird weiterhin aufgerufen, führt aber gewöhnlich keine Aktion aus
Bei erfolgreicher Festschreibung Transaktion der Liste festgeschriebener Transaktionen hinzufügen, andernfalls der Liste rückgängig gemachter Transaktionen hinzufügen	Nulloperation
Sperren für Einträge freigeben	Identisch

Für die Replikatverarbeitung verwenden Sie die folgende Folge:

1. Änderungen empfangen
2. Alle empfangenen Transaktionen in der Liste festgeschriebener Transaktionen festschreiben
3. Alle empfangenen Transaktionen in der Liste rückgängig gemachter Transaktionen rückgängig machen
4. Transaktion oder Sitzung starten
5. Änderung auf die Transaktion oder Sitzung anwenden
6. Transaktion oder Sitzung in der Liste offener Transaktionen speichern
7. Antwort zurücksenden

Beachten Sie, dass im Replikat keine Loader-Interaktionen stattfinden, während sich das Replikat im Replikatmodus befindet. Das primäre Shard muss alle Änderungen mit Push über den Loader übertragen. Das Replikat nimmt keine Änderungen vor. Dieser Algorithmus hat den Nebeneffekt, dass das Replikat immer die Transaktionen hat, diese aber erst festgeschrieben werden, wenn die nächste primäre Transaktion den Festschreibungsstatus dieser Transaktionen sendet. Erst dann werden die Transaktionen im Replikat festgeschrieben oder rückgängig gemacht. Bis dahin sind die Transaktionen nicht festgeschrieben. Sie können einen Zeitgeber für das primäre Shard hinzufügen, der das Transaktionsergebnis nach kurzer Zeit (ein paar Sekunden) sendet. Dieser Zeitgeber verringert, schließt aber das Risiko veralteter Daten in diesem Zeitfenster nicht ganz aus. Veraltete Daten sind nur dann ein Problem, wenn der Lesemodus für Replikate verwendet wird. Andernfalls haben die veralteten Daten keine Auswirkungen auf die Anwendung.

Wenn das primäre Shard ausfällt, ist es wahrscheinlich, dass einige Transaktionen zwar im primären Shard festgeschrieben oder rückgängig gemacht wurden, aber die Nachricht mit diesen Ergebnissen nicht mehr an das Replikat gesendet werden konnte. Wenn ein Replikat als neues primäres Shard hochgestuft wird, ist eine der ersten Aktionen die Behandlung dieser Bedingung. Jede offene Transaktion wird erneut für die Map-Gruppe des neuen primären Shards ausgeführt. Wenn ein Loader vorhanden ist, wird die erste Transaktion an den Loader übergeben. Diese Transaktionen werden in strikter First In/First Out-Reihenfolge angewendet. Wenn eine Transaktion scheitert, wird sie ignoriert. Sind drei Transaktionen, A, B und C, offen, kann A festgeschrieben, B rückgängig gemacht und C ebenfalls festgeschrieben werden. Keine der Transaktionen hat Auswirkung auf die anderen. Gehen Sie davon aus, dass sie voneinander unabhängig sind.

Ein Loader kann eine geringfügig andere Logik verwenden, wenn er sich im Modus für Fehlerbehebung durch Funktionsübernahme und nicht im normalen Mo-

dus befindet. Der Loader kann problemlos feststellen, wann er sich im Modus für Fehlerbehebung durch Funktionsübernahme befindet, indem er die Schnittstelle "ReplicaPreloadController" implementiert. Die Methode "checkPreloadStatus" wird erst aufgerufen, wenn der Loader wieder aus dem Modus für Fehlerbehebung durch Funktionsübernahme in den normalen Modus wechselt. Wenn die Methode "apply" der Schnittstelle "Loader" vor der Methode "checkPreloadStatus" aufgerufen wird, handelt es sich deshalb um eine Wiederherstellungstransaktion. Nach dem Aufruf der Methode "checkPreloadStatus" ist die Fehlerbehebung durch Funktionsübernahme abgeschlossen.

Verfahren für die Datenbanksynchronisation

Wenn WebSphere eXtreme Scale als Cache verwendet wird, müssen Anwendungen so geschrieben werden, dass veraltete Daten toleriert werden, wenn die Datenbank unabhängig von einer eXtreme-Scale-Transaktion aktualisiert werden kann. Für den Einsatz als Verarbeitungsbereich für die synchronisierte speicherinterne Datenbank stellt eXtreme Scale mehrere Methoden für die konstante Aktualisierung des Caches bereit.

Verfahren für die Datenbanksynchronisation

Regelmäßige Aktualisierung

Der Cache kann mit Hilfe der zeitbasierten JPA-Datenbankaktualisierungskomponente (Java Persistence API) automatisch ungültig gemacht oder regelmäßig aktualisiert werden. Die Aktualisierungskomponente fragt die Datenbank in regelmäßigen Abständen über einen JPA-Provider nach Aktualisierungen oder Einfügungen ab, die seit der vorherigen Aktualisierung vorgenommen wurden. Alle gefundenen Änderungen werden automatisch ungültig gemacht oder aktualisiert, wenn ein Teilcache verwendet wird. Wenn ein vollständiger Cache verwendet wird, können die Einträge erkannt und in den Cache eingefügt werden. Es werden keine Einträge aus dem Cache entfernt.

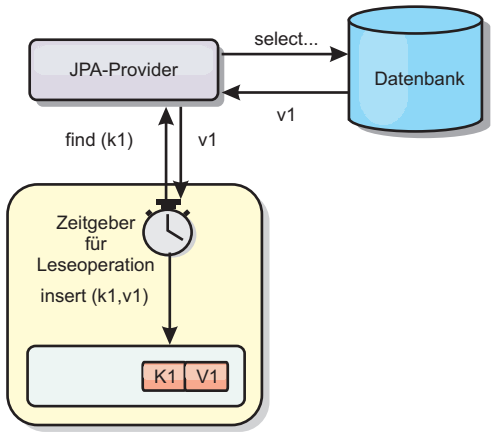


Abbildung 28. Regelmäßige Aktualisierung

Bereinigung

Teilcaches können Bereinigungsrichtlinien verwenden, um Daten ohne Beeinträchtigung der Datenbank automatisch aus dem Cache zu entfernen. Mit eXtreme Scale werden drei integrierte Richtlinien bereitgestellt: Lebensdauer (TTL, Time-to-Live), LRU (least recently used) und LFU (last frequently used). Alle drei Richtlinien kön-

nen Daten aggressiver entfernen, wenn Speicherengpässe auftreten, indem die Option für speicherbasierte Bereinigung aktiviert wird.

Ereignisbasiertes Ungültigmachen

Teilcaches und vollständige Caches können mit Hilfe eines Ereignisgenerators wie Java Message Service (JMS) ungültig gemacht oder aktualisiert werden. Das Ungültigmachen mit JMS kann manuell an jeden Prozess gebunden werden, der das Back-End über einen Datenbankauslöser aktualisiert. Es wird ein JMS-ObjectGridEventListener-Plug-in in eXtreme Scale bereitgestellt, das Clients benachrichtigen kann, wenn Änderungen im Servercache vorgenommen wurden. Auf diese Weise kann das Zeitfenster, in dem der Client veraltete Daten sieht, verringert werden.

Programmgesteuertes Ungültigmachen

Die APIs von eXtreme Scale unterstützen die manuelle Interaktion zwischen nahem Cache und Servercache über die API-Methoden "Session.beginNoWriteThrough()", "ObjectMap.invalidate()" und "EntityManager.invalidate()". Wenn ein Client- oder Serverprozess einen Teil der Daten nicht mehr benötigt, können Sie mit den Methoden zum Ungültigmachen Daten aus dem nahen Cache bzw. Servercache entfernen. Die Methode "beginNoWriteThrough" gilt für alle ObjectMap- und EntityManager-Operationen im lokalen Cache ohne Aufruf des Loaders. Wenn die Methode von einem Client aufgerufen wird, gilt die Operation nur für den nahen Cache (der ferne Loader wird nicht aufgerufen). Wird die Methode im Server aufgerufen, gilt die Operation nur für den Serverbasiscache ohne Aufruf des Loaders.

Veraltete Cachedaten ungültig machen

Um das Zeitfenster zu verkleinern, in dem Clients veraltete Daten sehen, können Sie einen ereignisgesteuerten oder programmgesteuerten Mechanismus für das Ungültigmachen von Daten verwenden.

Ereignisgesteuertes Ungültigmachen

Teilcaches und vollständige Caches können mit Hilfe eines Ereignisgenerators wie Java Message Service (JMS) ungültig gemacht oder aktualisiert werden. Das Ungültigmachen mit JMS kann manuell an jeden Prozess gebunden werden, der das Back-End über einen Datenbankauslöser aktualisiert. Es wird ein JMS-ObjectGridEventListener-Plug-in in eXtreme Scale bereitgestellt, das Clients benachrichtigen kann, wenn Änderungen im Servercache vorgenommen wurden. Dieser Typ von Benachrichtigung verkleinert das Zeitfenster, in dem der Client veraltete Daten sieht.

Der ereignisgesteuerte Mechanismus für Ungültigmachen setzt sich gewöhnlich aus den folgenden drei Komponenten zusammen:

- **Ereigniswarteschlange:** In einer Ereigniswarteschlange werden die Datenänderungsereignisse gespeichert. Die Ereigniswarteschlange kann eine JMS-Warteschlange, eine Datenbank, eine speicherinterne FIFO-Warteschlange oder ein beliebiges Manifest sein, das Datenänderungsereignisse verwalten kann.
- **Ereignis-Publisher:** Ein Ereignis-Publisher veröffentlicht die Datenänderungsereignisse in der Ereigniswarteschlange. Ein Ereignis-Publisher ist gewöhnlich eine Anwendung, die Sie erstellen, oder eine Implementierung eines eXtreme-Scale-Plug-ins. Der Ereignis-Publisher weiß, wann die Daten geändert werden, oder ändert die Daten selbst. Wenn eine Transaktion festgeschrieben wird, werden Ereignisse für die geänderten Daten generiert, und der Ereignis-Publisher veröffentlicht diese Ereignisse in der Ereigniswarteschlange.

- **Ereigniskonsument:** Ein Ereigniskonsument konsumiert Datenänderungsereignisse. Der Ereigniskonsument ist gewöhnlich eine Anwendung, die sicherstellt, das die Daten im Ziel-Grid mit den neuesten Änderungen aus anderen Grids aktualisiert werden. Dieser Ereigniskonsument interagiert mit der Ereigniswarteschlange, um die neuesten Datenänderungen abzurufen, und wendet die Datenänderungen auf das Ziel-Grid an. Die Ereigniskonsumenten können APIs von eXtreme Scale verwenden, um veraltete Daten ungültig zu machen oder um das Grid mit den neuesten Daten zu aktualisieren.

JMSObjectGridEventListener hat beispielsweise eine Option für ein Client/Server-Modell, bei der die Ereigniswarteschlange eine festgelegte JMS-Destination ist. Alle Serverprozesse sind Ereignis-Publisher. Wenn eine Transaktion festgeschrieben wird, ruft der Server die Datenänderungen ab und veröffentlicht sie in der festgelegten JMS-Destination. Alle Clientprozesse sind Ereigniskonsumenten. Sie empfangen Datenänderungen von der festgelegten JMS-Destination und wenden die Änderungen auf den nahen Cache des Clients an.

Weitere Informationen finden Sie im Abschnitt zum Aktivieren des Mechanismus für das Ungültigmachen von Clients im *Administratorhandbuch*.

Programmgesteuertes Ungültigmachen

Die APIs von WebSphere eXtreme Scale unterstützen die manuelle Interaktion zwischen nahem Cache und Servercache über die API-Methoden "Session.beginNoWriteThrough()", "ObjectMap.invalidate()" und "EntityManager.invalidate()". Wenn ein Client- oder Serverprozess einen Teil der Daten nicht mehr benötigt, können Sie mit den Methoden zum Ungültigmachen Daten aus dem nahen Cache bzw. Servercache entfernen. Die Methode "beginNoWriteThrough" gilt für alle ObjectMap- und EntityManager-Operationen im lokalen Cache ohne Aufruf des Loaders. Wenn die Methode von einem Client aufgerufen wird, gilt die Operation nur für den nahen Cache (der ferne Loader wird nicht aufgerufen). Wird die Methode im Server aufgerufen, gilt die Operation nur für den Serverbasiscache ohne Aufruf des Loaders.

Sie können den Mechanismus für programmgesteuertes Ungültigmachen zusammen mit anderen Techniken verwenden, um festzustellen, wann die Daten ungültig gemacht werden müssen. Diese Methode für Ungültigmachen verwendet beispielsweise ereignisgesteuerte Mechanismen für das Ungültigmachen, um die Datenänderungsereignisse zu empfangen, und anschließend APIs, um die veralteten Daten ungültig zu machen.

Indexierung

Verwenden Sie das Plug-in "MapIndexPlugin", um einen Index oder mehrere Indizes in einer BackingMap für die Unterstützung von Datenzugriffen ohne Schlüssel zu erstellen.

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 bie-

tet 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 Anwendungsindexschnittstelle, 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.

Die Klasse HashIndex ist die integrierte Index-Plug-in-Implementierung, die beide integrierten Anwendungsindexschnittstellen, MapIndex und MapRangeIndex, unterstützen kann. Sie können auch eigene Indizes erstellen. Sie können HashIndex

als statischen oder dynamischen Index in der BackingMap hinzufügen, ein MapIndex- oder MapRangeIndex-Index-Proxy-Objekt abrufen und das Index-Proxy-Objekt zum Suchen zwischengespeicherter Objekte verwenden.

Informationen zum Konfigurieren des HashIndex-Plug-ins finden Sie unter HashIndex konfigurieren.

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 unter Zusammengesetzter HashIndex.

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 Schüsselobjekts 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 beginne 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 Leseoperationen 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.

Konzepte für das Zwischenspeichern von Java-Objekten

WebSphere eXtreme Scale wird primär als Daten-Grid und Cache für Java-Objekte verwendet. Es gibt mehrere APIs, die Sie für die Interaktion mit dem eXtreme-Scale-Grid verwenden könen, um auf diese Objekte zuzugreifen und sie zu speichern.

In diesem Abschnitt werden einige der gebräuchlichen APIs und verschiedene Konzepte beschrieben, die Sie kennen müssen, wenn Sie eine API und eine Implementierungstopologie auswählen. Eine Beschreibung der verschiedenen Services und Topologien, die von eXtreme Scale unterstützt werden, finden Sie im Abschnitt „Caching-Architektur: Maps, Container, Clients und Kataloge" auf Seite 11.

Die zentrale Komponente von WebSphere eXtreme Scale ist das ObjectGrid. Das ObjectGrid ist der Namespace, in dem zugehörige Daten gespeichert werden und der Gruppen von Hash-Maps enthält, die Schlüssel/Wert-Paare enthalten. Diese Maps können gruppiert und partitioniert sowie hoch verfügbar und skalierbar gemacht werden.

Da das Grid naturgemäß Java-Objekte enthält, sind beim Design einer Anwendung verschiedene wichtige Faktoren zu beachten, so dass das Grid die Daten effizient speichern und aufrufen kann. Einige der Faktoren, die sich auf die Skalierbarkeit, Leistung und Speicherauslastung auswirken können, sind im Folgenden beschrieben.

Hinweise zu Klassenladeprogrammen und Klassenpfaden

Da eXtreme Scale Java-Objekte standardmäßig im Cache speichert, müssen Sie Klassen im Klassenpfad definieren, wenn auf die Daten zugegriffen wird.

Insbesondere Client- und Containerprozesse von eXtreme Scale müssen die Klassen bzw. JAR-Dateien im Klassenpfad enthalten, wenn der jeweilige Prozess gestartet wird. Trennen Sie beim Design einer Anwendung für eXtreme Scale jegliche Geschäftslogik von den persistenten Datenobjekten.

Weitere Informationen finden Sie im Artikel "Klassen laden" im Information Center von WebSphere Application Server Network Deployment.

Überlegungen, die in einer Spring-Framework-Umgebung angestellt werden müssen, finden Sie in den Informationen zum Packen im Abschnitt zur Integration von Spring Framework im *Programmierhandbuch*.

Einstellungen, die sich auf die Verwendung des Instrumentierungsagenten von WebSphere eXtreme Scale beziehen, sind im Abschnitt zum Instrumentierungsagenten im *Programmierhandbuch* beschrieben.

Verwaltung von Beziehungen

Objektorientierte Sprachen wie Java und relationale Datenbanken unterstützen Beziehungen oder Assoziationen. Beziehungen verringern den Speicherbedarf durch Die Verwendung von Objektreferenzen und Fremdschlüsseln.

Wenn Sie Beziehungen in einem Grid verwenden, müssen die Daten in einer Baumstruktur mit Integritätsbedingungen organisiert werden. Es muss einen einzigen Stammtyp in der Baumstruktur geben, und alle untergeordneten Typen dürfen nur einem einzigen Stammtyp zugeordnet sein. Beispiel: Eine Abteilung kann viele Mitarbeiter und ein Mitarbeiter viele Projekte haben. Aber ein Projekt kann keine Mitarbeiter haben, die zu verschiedenen Abteilungen gehören. Nach der Definition

eines Stammobjekts werden alle Zugriff auf dieses Stammobjekt und seine untergeordneten Objekte über das Stammobjekt verwaltet. WebSphere eXtreme Scale verwendet den Hash-Code des Stammobjektschlüssels, um eine Partition auszuwählen.

Beispiel: Partition = (Hash-Code MOD Anzahl_Partitionen)

Wenn alle Daten für eine Beziehung an eine einzige Objektinstanz gebunden sind, kann die gesamte Baumstruktur in einer einzigen Partition zusammengefasst werden, und der Zugriff auf diese Instanz kann sehr effizient über eine einzige Transaktion erfolgen. Wenn sich die Daten auf mehrere Beziehungen verteilen, müssen mehrere Partitionen beteiligt werden. Dies impliziert zusätzliche Fernaufrufe, was zu Leistungsengpässen führen kann.

Referenzdaten

Einige Beziehungen enthalten Such- oder Referenzdaten, wie z. B. "CountryName". Dies ist ein Sonderfall, in dem die Daten in jeder Partition vorhanden sein müssen. Hier kann der Zugriff auf die Daten über einen beliebigen Stammschlüssel erfolgen, und es wird immer dasselbe Ergebnis zurückgegeben. Referenzdaten wie diese sollten nur verwendet werden, wenn die Daten relativ statisch sind, da die Aktualisierung der Daten kostenintensiv sein kann, weil sie in jeder Partition durchgeführt werden muss. Die API "DataGrid" ist eine gängige Technik, mit der Referenzdaten auf dem aktuellen Stand gehalten werden können.

Kosten und Vorteile der Normalisierung

Durch die Normalisierung der Daten über Beziehungen kann der Speicherbedarf des Grids verringert werden, weil sich die Duplizierung der Daten verringert. Im Allgemeinen gilt jedoch, dass die horizontale Skalierung mit zunehmendem Volumen relationaler Daten abnimmt. Wenn Daten gruppiert werden, nimmt der Aufwand für die Verwaltung der Beziehungen und deren Größe zu. Da die Daten von Grid-Partitionen auf dem Schlüssel des Stammobjekts der Baumstruktur basieren, wird die Größe der Baumstruktur nicht berücksichtigt. Wenn Sie sehr viele Beziehungen für eine einzige Instanz der Baumstruktur haben, kann die Datenverteilung im Grid deshalb ungleichmäßig sein, d. h., eine Partition enthält mehr Daten als die anderen.

Wenn die Daten normalisiert oder reduziert werden, werden die Daten, die normalerweise von zwei Objekten gemeinsam genutzt werden, stattdessen dupliziert, und jede Tabelle kann gesondert partitioniert werden, wodurch eine gleichmäßigere Verteilung der Daten im Grid möglich ist. Dies erhöht zwar den Speicherbedarf, aber die Anwendung kann skaliert werden, da auf eine einzige Datenzeile zugegriffen werden kann, die alle erforderlichen Daten enthält. Dies ist ideal für die Grid, in denen hauptsächlich Leseoperationen durchgeführt werden, da die Verwaltung der Daten kostenintensiver wird.

Weitere Informationen finden Sie auf der Webseite "Classifying XTP systems and scaling".

Beziehungen über die Datenzugriffs-APIs verwalten

Die API "ObjectMap" ist die schnellste, flexibelste und differenzierteste der Datenzugriffs-APIs und unterstützt einen transaktionsorientierten, sitzungsbasierten Ansatz für den Zugriff auf Daten im Map-Grid. Die API "ObjectMap" ermöglicht Clients die Verwendung allgemeiner CRUD-Operationen (Create, Read, Update and

Delete, Erstellen, Lesen, Aktualisieren und Löschen) für die Verwaltung von Schlüssel/Wert-Paaren für die Objekte im verteilten Grid.

Wenn Sie die API "ObjectMap" verwenden, müssen Objektbeziehungen durch Integration des Fremdschlüssels für alle Beziehungen im übergeordneten Objekt ausgedrückt werden.

Es folgt ein Beispiel:

```
public class Department {
 Collection<String> employeeIds;
}
```

Die API "EntityManager" vereinfacht die Verwaltung von Beziehungen, indem sie persistente Daten aus den Objekten extrahiert, einschließlich der Fremdschlüssel. Wenn das Objekt später aus dem Grid abgerufen wird, wird der Beziehungsgraph erneut erstellt, wie im folgenden Beispiel gezeigt wird:

```
@Entity
public class Department {
 Collection<String> employees;
}
```

Die API "EntityManager" ist anderen Java-Objektpersistenztechnologien wie JPA und Hibernate insofern sehr ähnlich, als sie einen Graph verwalteter Java-Objektinstanzen mit dem persistenten Speicher synchronisiert. In diesem Fall ist der persistente Speicher ein eXtreme-Scale-Grid, in dem jede Entität als Map dargestellt wird, die die Entitätsdaten und nicht die Objektinstanzen enthält.

Wichtige Hinweise zu Caches

WebSphere eXtreme Scale verwendet Hash-Maps, um Daten im Grid zu speichern, wobei ein Java-Objekt als Schlüssel verwendet wird.

Richtlinien

Beachten Sie bei der Auswahl eines Schlüssels die folgenden Anforderungen.

- Schlüssel können sich nicht ändern. Wenn ein Teil des Schlüssels geändert werden muss, muss der Cacheeintrag entfernt und anschließend erneut eingefügt werden.
- Schlüssel sollten klein sein. Da Schlüssel in allen Datenzugriffsoperationen verwendet werden, empfiehlt es sich, die Schlüssel klein zu halten, so dass sie effizient serialisiert werden können und weniger Speicher belegen.
- Implementierung eines effizienten Hash- und Gleichheitsalgorithmus. Die Methoden "hashCode" und "equals(Object o)" müssen stets für jedes Schlüsselobjekt überschrieben werden.
- Zwischenspeicherung des Hash-Codes des Schlüssels. Sie sollten den Hash-Code, sofern möglich, in der Schlüsselobjektinstanz zwischenspeichern, um die Berechnungen der Methode "hashCode()" zu beschleunigen. Da der Schlüssel unveränderlich ist, muss der Hash-Code zwischenspeicherbar sein.
- Duplizierung des Schlüssels im Wert vermeiden. Wenn Sie die API "ObjectMap" verwenden, ist es praktisch, den Schlüssel im Wertobjekt zu speichern. In diesem Fall werden die Schlüssel im Speicher dupliziert.

Serialisierungsleistung

WebSphere eXtreme Scale verwendet mehrere Java-Prozesse, in denen Daten gespeichert werden. Diese Prozesse serialisieren die Daten, d. h., sie konvertieren die Daten (die das Format von Java-Objektinstanzen haben) in Bytes und bei Bedarf

zurück in Objekte, um die Daten zwischen Client- und Serverprozessen zu verschieben. 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 -Kopierroutinen 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.

Weitere Informationen finden Sie unter Plug-ins für das Serialisieren und Kopieren zwischengespeicherter Objekte und unter Bewährte Verfahren für die Schnittstelle "ObjectTransformer".

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 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 Schüssel 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 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.

Weitere Informationen finden Sie in der Beschreibung der bewährten Verfahren für die Methode "CopyMode" im *Programmierhandbuch*.

Daten für verschiedene Zeitzonen einfügen

Wenn Sie Daten mit calender-, java.util.Date- und timestamp-Attributen in ein ObjectGrid einfügen, müssen Sie sicherstellen, dass diese Datums-/Zeitattribute auf der Basis derselben Zeitzone erstellt werden, insbesondere wenn sie in mehreren Servern in verschiedenen Zeitzonen implementiert werden. Durch die Verwendung von Datums-/Zeitobjekten, die auf derselben Zeitzone basieren, können Sie sicherstellen, dass die Anwendung zeitzonensicher ist und Daten mit Hilfe von calendar-, java.util.Date- und timestamp-Prädikaten abgefragt werden können.

Ohne explizite Angabe einer Zeitzone beim Erstellen von Datums-Zeitobjekten verwendet Java die lokale Zeitzone, was zu inkonsistenten Datums-/Zeitwerten in Clients und Servern führen kann.

Stellen Sie sich beispielsweise eine verteilte Implementierung vor, in der sich client1 in der Zeitzone [GMT-0] und client2 in der Zeitzone [GMT-6] befindet und beide Clients ein java.util.Date-Objekt mit dem Wert '1999-12-31 06:00:00' erstellen möchten. client1 erstellt das java.util.Date-Objekt mit dem Wert '1999-12-31 06:00:00 [GMT-0]' und client2 das java.util.Date-Objekt mit dem Wert '1999-12-31 06:00:00 [GMT-6]'. Die beiden java.util.Date-Objekte sind nicht gleich, weil die Zeitzonen verschieden sind. Ein ähnliches Problem tritt auf, wenn Daten vorab in Partitionen geladen werden, die sich in Servern in unterschiedlichen Zeitzonen befinden, wenn die lokale Zeitzone zum Erstellen von Datums-Zeitobjekten verwendet wird.

Zur Vermeidung des beschriebenen Problems kann die Anwendung eine Zeitzone wie [GMT-0] als Basiszeitzone für die Erstellung von calendar-, java.util.Date- und timestamp-Objekten auswählen.

Weitere Informationen finden Sie im Abschnitt zum Abfragen von Daten in mehreren Zeitzonen im *Programmierhandbuch*.

Kapitel 3. Übersicht über die Cacheintegration: JPA, Sitzungen und dynamisches Caching

Das entscheidende Element, das WebSphere eXtreme Scale eine solche Vielseitigkeit und Zuverlässigkeit ermöglicht, ist die Anwendung von Caching-Konzepten für die Optimierung der Persistenz und erneuten Erfassung von Daten in praktisch jeder Implementierungsumgebung.

JPA-Loader

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.

Sie können eine JPA-Loader-Plug-in-Implementierung mit eXtreme Scale verwenden, um mit jeder vom ausgewählten Loader unterstützten Datenbank zu interagieren. 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.

Das JPALoader-Plug-in "com.ibm.websphere.objectgrid.jpa.JPALoader" und das JPAEntityLoader-Plug-in "com.ibm.websphere.objectgrid.jpa.JPAEntityLoader" sind zwei integrierte JPA-Loader-Plug-ins, die verwendet werden, um die ObjectGrid-Maps mit einer Datenbank zu synchronisieren. Sie müssen eine JPA-Implementierung wie Hibernate oder OpenJPA haben, um dieses Feature verwenden zu können. Als Datenbank kann jedes Back-End verwendet werden, das vom ausgewählten JPA-Provider unterstützt wird.

Sie können das JPALoader-Plug-in verwenden, wenn Sie Daten über die API "ObjectMap" speichern. Verwenden Sie das JPAEntityLoader-Plug-in, wenn Sie Daten über die API "EntityManager" speichern.

Architektur der JPA-Loader

Der JPA-Loader wird für eXtreme-Scale-Maps verwendet, in denen POJOs (Plain Old Java Objects) gespeichert werden.

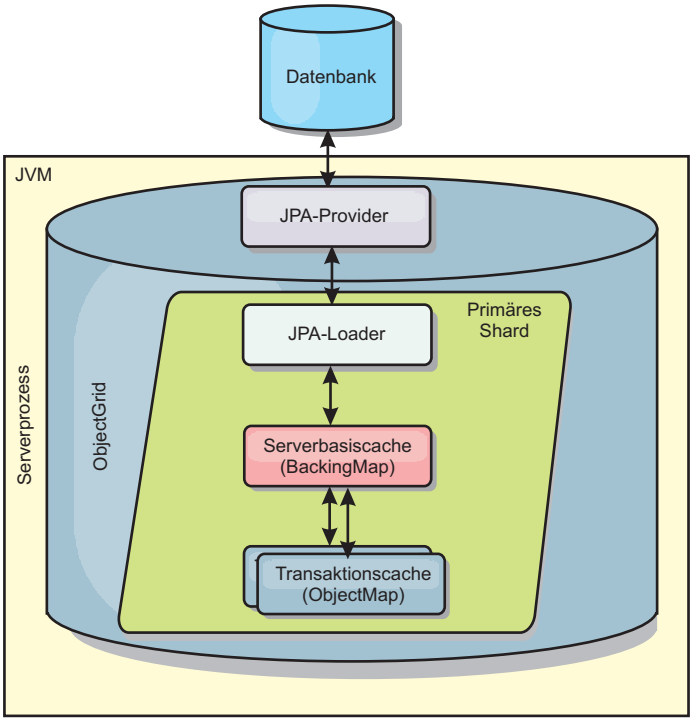


Abbildung 29. Architektur der JPA-Loader

Wenn eine Methode ObjectMap.get(Object key) aufgerufen wird, prüft die Laufzeitumgebung von eXtreme Scale zunächst, ob der Eintrag auf ObjectMap-Ebene vorhanden ist. Wenn nicht, delegiert die Laufzeitumgebung die Anforderung an den JPA-Loader. Auf die Anforderung hin, den Schlüssel zu laden, ruft der JPALoader die JPA-Methode EntityManager.find(Object key) auf, um die Daten auf JPA-Ebene zu suchen. Sind die Daten im JPA-EntityManager vorhanden, werden sie zurückgegeben, wenn nicht, interagiert der JPA-Provider mit der Datenbank, um den Wert abzurufen.

Bei einer Aktualisierung der ObjectMap, z. B. über die Methode "ObjectMap.update(Object key, Object value)", erstellt die Laufzeitumgebung von eXtreme Scale ein LogElement-Objekt für diese Aktualisierung und sendet es an den JPALoader. Der JPALoader ruft die JPA-Methode "EntityManager.merge(Object value)" auf, um den Wert in der Datenbank zu aktualisieren.

Beim JPAEntityLoader sind dieselben vier Ebenen beteiligt. Da das JPAEntityLoader-Plug-in jedoch für Maps verwendet wird, in denen eXtreme-Scale-Entitäten gespeichert werden, können Relationen zwischen den einzelnen Entitäten das Einsatzszenario komplizierter machen. Eine eXtreme-Scale-Entität unterscheidet sich von einer JPA-Entität. Weitere Informationen finden Sie in der Dokumentation zum JPAEntityLoader-Plug-in im *Programmierhandbuch*.

Methoden

Loader (Ladeprogramme) stellen drei Hauptmethoden bereit:

1. get: Gibt eine Liste mit Werten zurück, die der Liste der Schlüssel entspricht, die durch Abruf der Daten über JPA übergeben werden. Die Methode verwendet JPA, um die Entitäten in der Datenbank zu suchen. Für das JPALoader-Plug-in enthält die zurückgegebene Liste eine Liste der JPA-Entitäten, die direkt von der Suchoperation zurückgegeben werden. Für das JPAEntityLoader-Plug-

in enthält die zurückgegebene Liste Tupel für die eXtreme-Scale-Entitätswerte, die aus den JPA-Entitäten konvertiert wurden.

2. batchUpdate: Schreibt die Daten aus den ObjectGrid-Maps in die Datenbank. Ja nach Operationstyp (Einfügen, Aktualisieren oder Löschen) verwendet der Loader die JPA-Operationen "persist", "merge" und "remove", um die Daten in der Datenbank zu aktualisieren. Für JPALoader werden die Objekte in der Map direkt als JPA-Entitäten verwendet. Für JPAEntityLoader werden die Entitätstupel in der Map in Objekte konvertiert, die als JPA-Entitäten verwendet werden.
3. preloadMap: Lädt die Daten über die Methode "ClientLoader.load" des Clientladeprogramms vorab in die Map. Für partitionierte Maps wird die Methode "preloadMap" nur in einer einzigen Partition aufgerufen. Die Partition wird mit der Eigenschaft "preloadPartition" der Klasse "JPALoader" bzw. "JPAEntityLoader" angegeben. Wenn die Eigenschaft "preloadPartition" auf einen Wert kleiner als null oder größer als (*Gesamtanzahl_der_Partitionen* - 1) gesetzt wird, wird das vorherige Laden inaktiviert.

JPALoader- und JPAEntityLoader-Plug-ins arbeiten mit JPATxCallback, um die eXtreme-Scale-Transaktionen und JPA-Transaktionen zu koordinieren. JPATxCallback muss in der ObjectGrid-Instanz für die Verwendung dieser beiden Loader konfiguriert werden.

Konfiguration und Programmierung

Weitere Informationen zum Konfigurieren von JPA-Loadern finden Sie in den Informationen zu JPA-Loadern im *Administratorhandbuch*. Weitere Informationen zur Programmierung von JPA-Loadern finden Sie im *Programmierhandbuch*.

Cache-Plug-in für JPA

WebSphere eXtreme Scale enthält Cache-Plug-ins der Stufe 2 (L2) für die JPA-Provider OpenJPA und Hibernate.

Die Verwendung von eXtreme Scale als L2-Cacheprovider erhöht die Leistung beim Lesen und Abfragen von Daten und reduziert die Last der Datenbank. WebSphere eXtreme Scale bietet im Vergleich mit integrierten Cacheimplementierungen verschiedene Vorteile, weil der Cache automatisch in allen Prozessen repliziert wird. Wenn ein Client einen Wert zwischenspeichert, können alle anderen Clients den zwischengespeicherten Wert, der sich lokal im Speicher befindet, verwenden.

Mit den ObjectGrid-Cache-Plug-ins für OpenJPA und Hibernate können Sie drei Topologietypen erstellen: integriert, integriert-partitioniert und fern.

Integrierte Topologie

Eine integrierte Topologie erstellt einen eXtreme-Scale-Server in dem Prozessbereich jeder Anwendung. OpenJPA und Hibernate lesen die Speicherkopie des Caches direkt und schreiben in alle anderen Kopien. Sie können die Schreibleistung durch den Einsatz asynchroner Replikation verbessern. Diese Standardtopologie liefert die beste Leistung, wenn die zwischengespeicherte Datenmenge in einen einzigen Prozess passt.

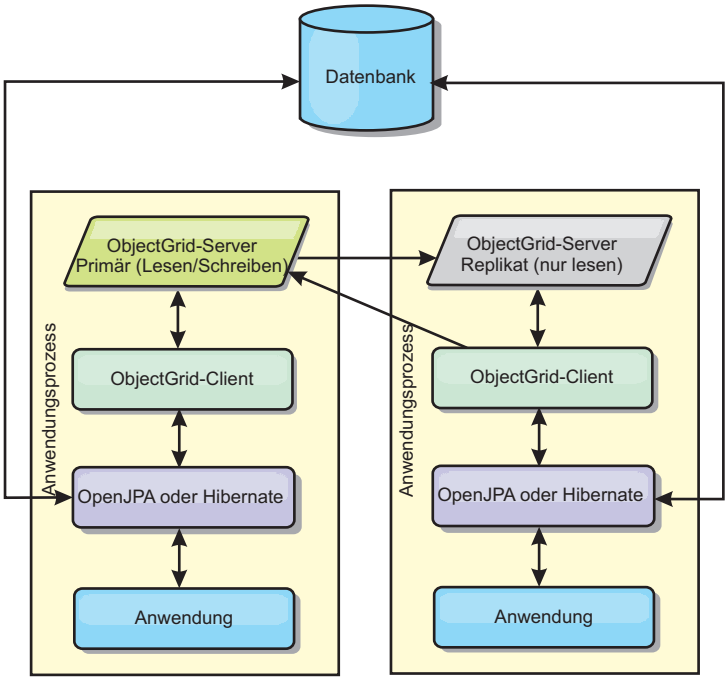


Abbildung 30. Integrierte JPA-Topologie

Vorteile:

- Alle Leseoperationen im Cache sind sehr schnelle lokale Zugriffe.
- Die Konfiguration ist einfach.

Einschränkungen:

- Das Datenvolumen ist auf die Größe des Prozesses beschränkt.
- Alle Cacheaktualisierungen werden an einen einzigen Prozess gesendet.

Integrierte, partitionierte Topologie

Wenn die zwischengespeicherten Daten nicht in einen einzigen Prozess passen, verwendet die integrierte, partitionierte Topologie ObjectGrid-Partitionen, um die Daten auf mehrere Prozesse zu verteilen. Die Leistung ist nicht so hoch wie bei der integrierten Topologie, weil die meisten Leseoperationen im Cache über Fernzugriff durchgeführt werden. Sie können diese Option trotzdem verwenden, wenn die Latenzzeit der Datenbank hoch ist.

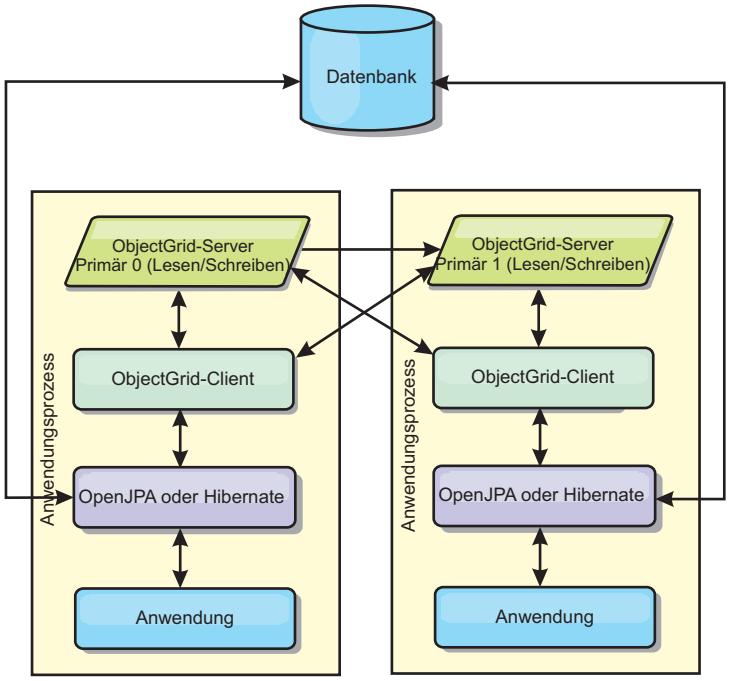


Abbildung 31. Integrierte, partitionierte JPA-Topologie

Vorteile:

- Es können große Datenvolumen gespeichert werden.
- Die Konfiguration ist einfach.
- Cacheaktualisierungen werden auf mehrere Prozesse verteilt.

Einschränkungen:

- Die meisten Lese- und Aktualisierungsoperationen im Cache werden über Fernzugriff durchgeführt.

Um beispielsweise 10 GB Daten mit maximal 1 GB pro JVM zu speichern, sind zehn Java Virtual Machines erforderlich. Die Anzahl der Partitionen muss daher auf mindestens 10 gesetzt werden. Im Idealfall wird die Anzahl der Partitionen auf eine Primzahl gesetzt, so dass in jedem Shard eine angemessene Speichermenge zugeteilt wird. Gewöhnlich entspricht der Wert der Einstellung "numberOfPartitions" der Anzahl der Java Virtual Machines. Bei dieser Einstellung enthält jede JVM eine Partition. Wenn Sie die Replikation aktivieren, müssen Sie die Anzahl der Java Virtual Machines im System erhöhen. Andernfalls wird in jeder JVM zusätzlich eine Replikatpartition gespeichert, die genauso viel Speicher belegt wie eine primäre Partition.

Lesen Sie die Informationen zur Berechnung der Speicherkapazität und der Partitionsanzahl im *Administratorhandbuch*, um die Leistung der von Ihnen ausgewählten Konfiguration zu maximieren.

In einem System mit vier Java Virtual Machines und einem numberOfPartitions-Wert von 4 beispielsweise enthält jede JVM eine primäre Partition. Bei einer Leseoperation besteht eine Chance von 25 %, dass die Daten aus einer lokal verfügbaren Partition abgerufen werden, was im Vergleich mit dem Abruf der Daten aus einer fernen JVM wesentlich schneller ist. Wenn eine Leseoperation, z. B. eine Abfrage, eine Sammlung von Daten abrufen muss, die gleichmäßig auf vier Partitionen verteilt sind, sind 75 % der Aufrufe ferne und 25 % der Aufrufe lokale Aufru-

fe. Wenn die Einstellung "ReplicaMode" auf `SYNC` oder `ASYNC` und die Einstellung "ReplicaReadEnabled" auf `true` gesetzt wird, werden vier Relikatpartitionen erstellt und auf vier Java Virtual Machines verteilt. Jede JVM enthält eine primäre Partition und eine Replikatpartition. Die Chance, dass die Leseoperation lokal ausgeführt wird, erhöht sich auf 50 %. Die Leseoperation, die eine Sammlung von Daten abgerufen muss, die gleichmäßig auf vier Partitionen verteilt sind, hat 50 % ferne Aufrufe und 50% lokale Aufrufe. Lokale Aufrufe sind wesentlich schneller als ferne Aufrufe. Mit jedem fernen Aufruf nimmt die Leistung ab.

Ferne Topologie

In einer fernen Topologie werden alle zwischengespeicherten Daten in einem oder mehreren gesonderten Prozessen gespeichert, was die Speicherbelegung der Anwendungsprozesse verringert. Sie können Ihre Daten auf unterschiedliche Prozesse verteilen, indem Sie ein partitioniertes, repliziertes eXtreme-Scale-Grid implementieren. Im Gegensatz zu den integrierten und integrierten partitionierten Konfigurationen, die in den vorherigen Abschnitten beschrieben wurden, müssen Sie ein fernes Grid unabhängig von der Anwendung und vom JPA-Provider verwalten. Weitere Einzelheiten zur Verwaltung einer Implementierung eines eXtreme-Scale-Grids finden Sie in den Informationen zur Überwachung von Implementierungsumgebungen.

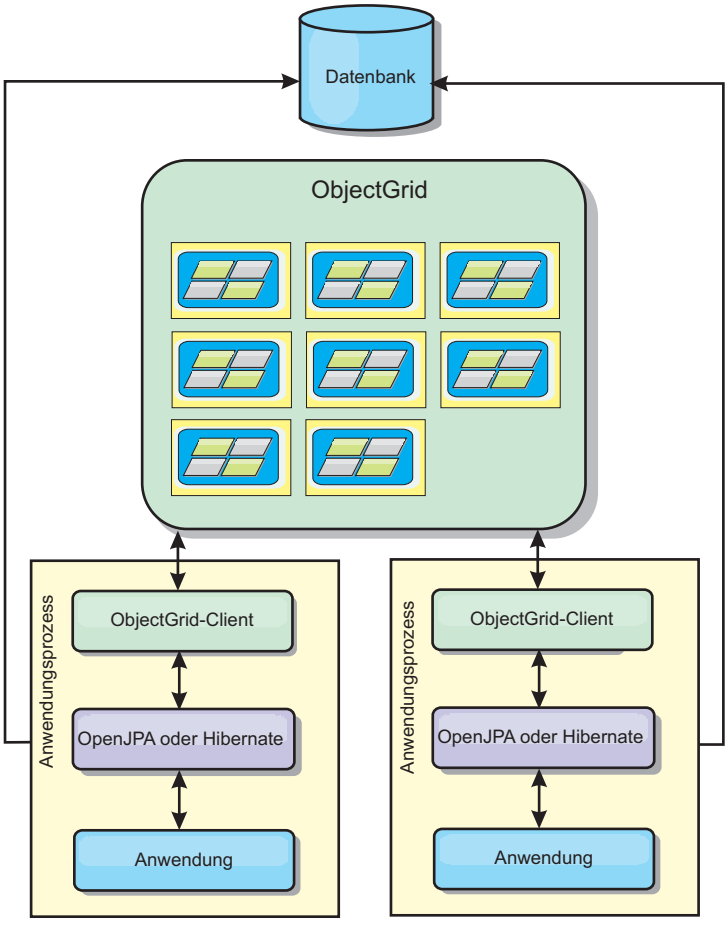


Abbildung 32. Ferne JPA-Topologie

Vorteile:

- Es können große Datenvolumen gespeichert werden.

- Der Anwendungsprozess ist frei von zwischengespeicherten Daten.
- Cacheaktualisierungen werden auf mehrere Prozesse verteilt.
- Sehr flexible Konfigurationsoptionen.

Einschränkungen:

- Alle Lese- und Aktualisierungsoperationen im Cache werden über Fernzugriff durchgeführt.

Konfiguration

Weitere Informationen zur Konfiguration von Cache-Plug-ins für JPA finden Sie im Abschnitt zu Plug-ins im *Programmierhandbuch*.

Verwaltung von HTTP-Sitzungen

Der mit WebSphere eXtreme Scale bereitgestellte Sitzungsreplikationsmanager kann mit dem Standardsitzungsmanager im Anwendungsserver zusammenarbeiten, um Sitzungsdaten eines Prozesses in einem anderen zu replizieren, damit Benutzersitzungsdaten hoch verfügbar sind.

Features

Der Sitzungsmanager wurde so konzipiert, dass er in jedem Container der Java Platform, Enterprise Edition Version 1.4 ausgeführt werden kann. Da der Sitzungsmanager keine Abhängigkeiten von WebSphere-APIs aufweist, kann er verschiedene Versionen von WebSphere Application Server sowie Anwendungsserverumgebungen anderer Anbieter unterstützen.

Der HTTP-Sitzungsmanager stellt Sitzungsreplikationsfunktionen für eine zugeordnete Anwendung bereit. Der Sitzungsreplikationsmanager arbeitet mit dem Sitzungsmanager des Webcontainers zusammen, um HTTP-Sitzungen zu erstellen und die Lebenszyklen von HTTP-Sitzungen zu verwalten, die der Anwendung zugeordnet sind. Zu diesen Verwaltungsaktivitäten für den Lebenszyklus gehören das Ungültigmachen von Sitzungen auf der Basis eines Zeitlimits oder eines expliziten Servlet- oder JSP-Aufrufs (JavaServer Pages) und der Aufruf von Sitzungs-Listenern, die der Sitzung bzw. der Webanwendung zugeordnet sind. Der Sitzungsmanager definiert seine Sitzung in einer ObjectGrid-Instanz als persistent. Diese Instanz kann eine lokale speicherinterne Instanz oder eine vollständig replizierte und partitionierte Clusterinstanz sein. Die Verwendung der letzteren Topologie ermöglicht dem Sitzungsmanager, die Unterstützung für HTTP-Sitzungs-Failover bereitzustellen, wenn Anwendungsserver heruntergefahren oder unerwartet beendet werden. Der Sitzungsmanager kann auch in Umgebungen eingesetzt werden, die keine Affinität unterstützen, wenn die Affinität nicht durch eine Lastausgleichsfunktionsschicht erzwungen wird, die Anforderungen auf die Anwendungsserver verteilt.

Einsatzszenarien

Der Sitzungsmanager kann in den folgenden Szenarien verwendet werden:

- In Umgebungen, die Anwendungsserver verschiedener Versionen von WebSphere Application Server verwenden, z. B. in einem klassischen Migrationsszenario.
- In Implementierungen, die Anwendungsserver verschiedener Anbieter verwenden. Ein Beispiel hierfür sind Anwendungen, die unter Open-Source-Anwendungsservern entwickelt werden und dann in WebSphere Application Server

eingesetzt werden. Ein weiteres Beispiel sind Anwendungen, die von der Bereitstellung auf die Produktion hochgestuft werden. Eine nahtlose Migration dieser Anwendungsserverversionen ist möglich, während alle HTTP-Sitzungen aktiv und bedient werden.

- In Umgebungen, die erfordern, dass der Benutzer Sitzungen mit höheren Servicequalitätsstufen und besseren Garantien für die Sitzungsverfügbarkeit beim Server-Failover als den Standardservicequalitätsstufen von WebSphere Application Server als persistent definieren.
- In einer Umgebung, in der die Sitzungsaffinität nicht garantiert werden kann, oder in Umgebungen, in denen die Affinität von einer anbieterspezifischen Lastausgleichsfunktion verwaltet wird und der Affinitätsmechanismus an diese Lastausgleichsfunktion angepasst werden muss.
- In einer Umgebung, in der das Management und Speichern von Sitzungen in einen externen Java-Prozess ausgelagert werden muss.
- In mehreren Zellen, in denen das Sitzungs-Failover zwischen den Zellen aktiviert werden muss.
- In mehreren Rechenzentren oder mehreren Zonen.

Funktionsweise des Sitzungsmanagers

Der Sitzungsreplikationsmanager verwendet den Standardsitzungs-Listener, um auf Änderungen der Sitzungsdaten zu warten, und schreibt die Sitzungsdaten persistent in eine ObjectGrid-Instanz (lokal oder über Fernzugriff). Die Sitzungsdaten werden über das Standard-Servlet erneut aus der ObjectGrid-Instanz (lokal oder über Fernzugriff) geladen. Mit Hilfe der in WebSphere eXtreme Scale bereitgestellten Tools können Sie den Sitzungs-Listener und den Servlet-Filter jedem Webmodul in Ihrer Anwendung hinzufügen. Sie können diese Listener und Filter auch dem Webimplementierungsdeskriptor Ihrer Anwendung hinzufügen.

Dieser Sitzungsreplikationsmanager arbeitet mit dem Webcontainersitzungsmanager jedes Anbieters zusammen, um Sitzungsdaten in Java Virtual Machines zu replizieren. Wenn der ursprüngliche Server abstürzt, können Benutzer die Sitzungsdaten von anderen Servern abrufen.

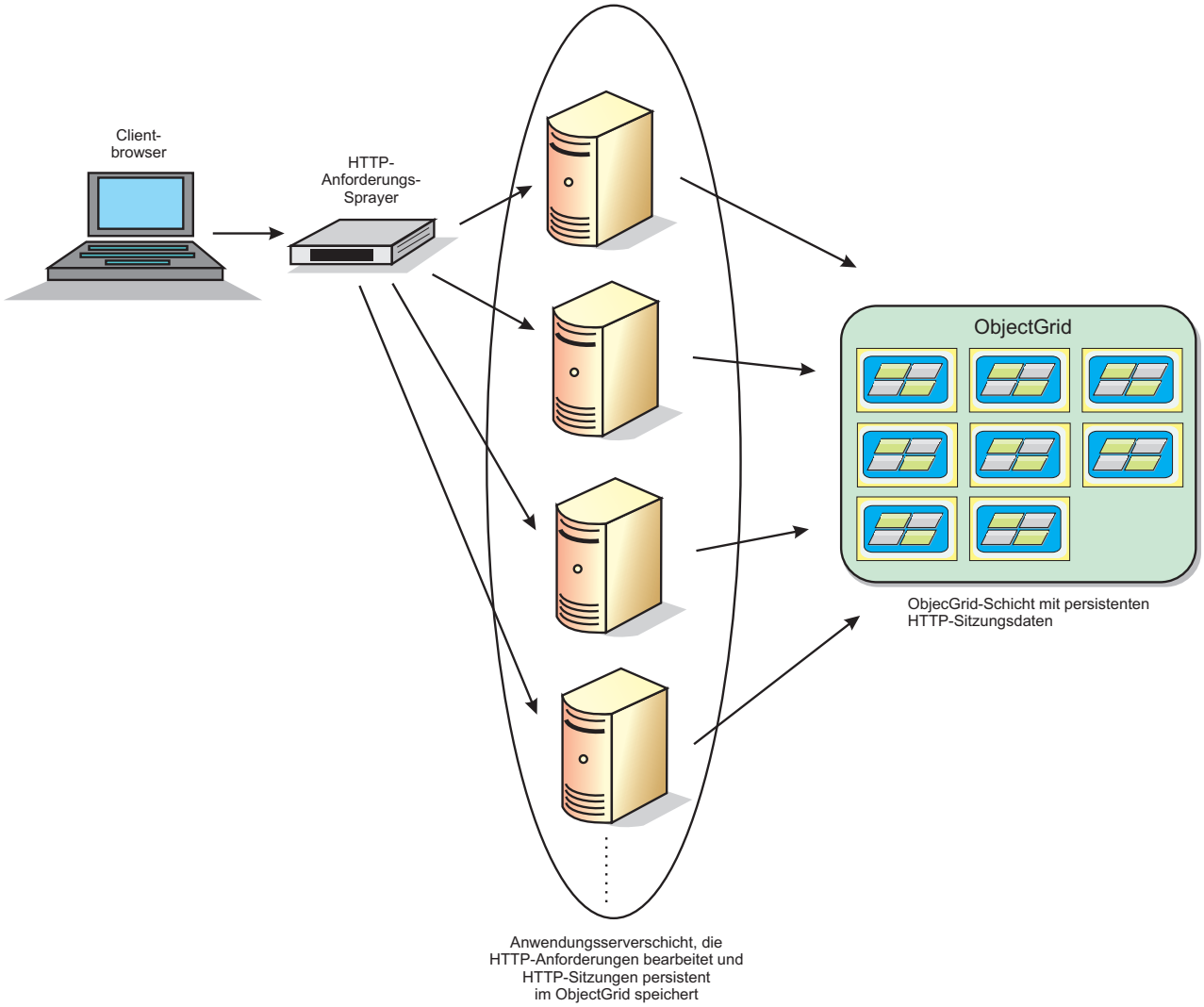


Abbildung 33. Topologie für das HTTP-Sitzungsmanagement mit einer fernen Containerkonfiguration

Implementierungstopologien

Es gibt zwei dynamische Implementierungsszenarien für die Konfiguration des Sitzungsmanagers:

- **Integrierte eXtreme-Scale-Container mit Netzanschluss**

  In diesem Szenario werden die Server von eXtreme Scale in denselben Prozessen wie die Servlets zusammengefasst. Der Sitzungsmanager kann direkt mit der lokalen ObjectGrid-Instanz kommunizieren, wodurch teure Verzögerungen bei der Netzübertragung vermieden werden. Dieses Szenario ist bevorzugt zu verwenden, wenn mit Affinität gearbeitet wird und die Leistung ein kritischer Faktor ist.

- **Ferne eXtreme-Scale-Container mit Netzanschluss**

  In diesem Szenario werden die Server von eXtreme Scale in Prozessen ausgeführt, die von dem Prozess abgesondert sind, in dem die Servlets ausgeführt werden. Der Sitzungsmanager kommuniziert mit einem fernen eXtreme-Scale-Server-Grid. Dieses Szenario ist bevorzugt zu verwenden, wenn die Webcontainerschicht nicht den erforderlichen Speicher zum Speichern der Sitzungsdaten besitzt. Die Sitzungsdaten werden auf eine separate Schicht ausgelagert, was zu

einer geringeren Speicherbelegung auf der Webcontainerschicht, aber aufgrund der fernen Position der Daten zu höheren Latenzzeiten führt.

Start generischer integrierter Container

eXtreme Scale startet einen integrierten ObjectGrid-Container automatisch in einem Anwendungsserverprozess, wenn der Webcontainer den Sitzungs-Listener oder den Servlet-Filter initialisiert, falls die Eigenschaft "objectGridType" den Wert EMBEDDED hat. Weitere Einzelheiten finden Sie unter "Initialisierungsparameter für Servlet-Kontexte".

In eXtreme Scale Version 7.1 müssen Sie eine Datei "ObjectGrid.xml" und eine Datei "objectGridDeployment.xml" nicht in die WAR-Datei der Webanwendung oder EAR-Datei packen. Die Standarddateien "ObjectGrid.xml" und "objectGridDeployment.xml" sind in der Produkt-JAR-Datei enthalten. Für verschiedene Webanwendungskontexte werden standardmäßig dynamische Maps erstellt. Statische eXtreme-Scale-Maps werden weiterhin unterstützt.

Dieser Ansatz für das Starten integrierter ObjectGrid-Container ist für jeden Typ von Anwendungsserver gültig. Die Ansätze, an denen eine Komponente von WebSphere Application Server oder WebSphere Application Server Community Edition GBean beteiligt ist, sind veraltet.

Listener-basierter Sitzungsreplikationsmanager

Der mit WebSphere eXtreme Scale bereitgestellte Sitzungsreplikationsmanager kann mit dem Standardsitzungsmanager im Anwendungsserver zusammenarbeiten, um Sitzungsdaten eines Prozesses in einem anderen zu replizieren, damit Benutzersitzungsdaten hoch verfügbar sind.

Der Sitzungsmanager wurde so konzipiert, dass er in jedem Container der Java™ Platform, Enterprise Edition Version 1.4 ausgeführt werden kann. Der Sitzungsmanager ist nicht von den WebSphere-APIs abhängig und somit in der Lage, verschiedene Versionen von WebSphere Application Server und auch Anwendungsserverumgebungen anderer Anbieter zu unterstützen.

Der HTTP-Sitzungsmanager stellt Sitzungsreplikationsfunktionen für eine zugeordnete Anwendung bereit. Der Sitzungsreplikationsmanager arbeitet mit dem Sitzungsmanager des Webcontainers zusammen, um HTTP-Sitzungen zu erstellen und die Lebenszyklen von HTTP-Sitzungen zu verwalten, die der Anwendung zugeordnet sind. Zu diesen Verwaltungsaktivitäten für den Lebenszyklus gehören das Ungültigmachen von Sitzungen auf der Basis eines Zeitlimits oder eines expliziten Servlet- oder JSP-Aufrufs (JavaServer Pages) und der Aufruf von Sitzungs-Listenern, die der Sitzung bzw. der Webanwendung zugeordnet sind. Der Sitzungsmanager definiert seine Sitzung in einer ObjectGrid-Instanz als persistent. Diese Instanz kann eine lokale speicherinterne Instanz oder eine vollständig replizierte und partitionierte Clusterinstanz sein. Die Verwendung der letzteren Topologie ermöglicht dem Sitzungsmanager, die Unterstützung für HTTP-Sitzungs-Failover bereitzustellen, wenn Anwendungsserver heruntergefahren oder unerwartet beendet werden. Der Sitzungsmanager kann auch in Umgebungen eingesetzt werden, die keine Affinität unterstützen, wenn die Affinität nicht durch eine Lastausgleichsfunktionsschicht erzwungen wird, die Anforderungen auf die Anwendungsserver verteilt.

Einsatzszenarien

Der Sitzungsmanager kann in den folgenden Szenarien verwendet werden:

- In Umgebungen, die Anwendungsserver verschiedener Versionen von WebSphere Application Server verwenden, z. B. in einem klassischen Migrationsszenario.
- In Implementierungen, die Anwendungsserver verschiedener Anbieter verwenden. Ein Beispiel hierfür sind Anwendungen, die unter Open-Source-Anwendungsservern entwickelt werden und dann in WebSphere Application Server eingesetzt werden. Ein weiteres Beispiel sind Anwendungen, die von der Bereitstellung auf die Produktion hochgestuft werden. Eine nahtlose Migration dieser Anwendungsserverversionen ist möglich, während alle HTTP-Sitzungen aktiv und bedient werden.
- In Umgebungen, die erfordern, dass der Benutzer Sitzungen mit höheren Servicequalitätsstufen und besseren Garantien für die Sitzungsverfügbarkeit beim Server-Failover als den Standardservicequalitätsstufen von WebSphere Application Server als persistent definieren.
- In einer Umgebung, in der die Sitzungsaffinität nicht garantiert werden kann, oder in Umgebungen, in denen die Affinität von einer anbieterspezifischen Lastausgleichsfunktion verwaltet wird und der Affinitätsmechanismus an diese Lastausgleichsfunktion angepasst werden muss.
- In einer Umgebung, in der das Management und Speichern von Sitzungen in einen externen Java-Prozess ausgelagert werden muss.
- In mehreren Zellen, in denen das Sitzungs-Failover zwischen den Zellen aktiviert werden muss.
- In mehreren Rechenzentren oder mehreren Zonen.

Details zum Sitzungsmanager

Der Sitzungsreplikationsmanager verwendet den Standardsitzungs-Listener, um auf Änderungen der Sitzungsdaten zu warten, und schreibt die Sitzungsdaten persistent in eine ObjectGrid-Instanz (lokal oder über Fernzugriff). Die Sitzungsdaten werden über das Standard-Servlet erneut aus der ObjectGrid-Instanz (lokal oder über Fernzugriff) geladen. Mit Hilfe der in WebSphere eXtreme Scale bereitgestellten Tools können Sie den Sitzungs-Listener und den Servlet-Filter jedem Webmodul in Ihrer Anwendung hinzufügen. Sie können diese Listener und Filter auch dem Webimplementierungsdeskriptor Ihrer Anwendung hinzufügen.

Der Sitzungsreplikationsmanager arbeitet mit dem Basissitzungsmanager jedes Anbieters zusammen, um Anwendungssitzungsdaten zu replizieren. Beachten Sie die folgenden Hinweise.

- Wählen Sie je nach Leistungsanforderungen und Datenvolumen integrierte ObjectGrid-Container oder ferne ObjectGrid-Container aus. Das integrierte Szenario bietet die einfachste Konfiguration und die bessere Leistung.
- Wählen Sie die Anzahl der fernen ObjectGrid-Container pro Webcontainer auf der Basis Ihrer Benutzerdatenvolumen aus.
- Legen Sie auf der Basis der Anzahl der Attribute, dem Volumen der Benutzerdaten und der Änderungsfrequenz fest, ob alle Sitzungsdaten gemeinsam gespeichert werden sollen oder ob jedes Attribut gesondert gespeichert werden solle.
- Wählen Sie das Replikationsintervall aus. Eine bessere Leistung kann mit einem weniger aggressiven Replikationsintervall erreicht werden.

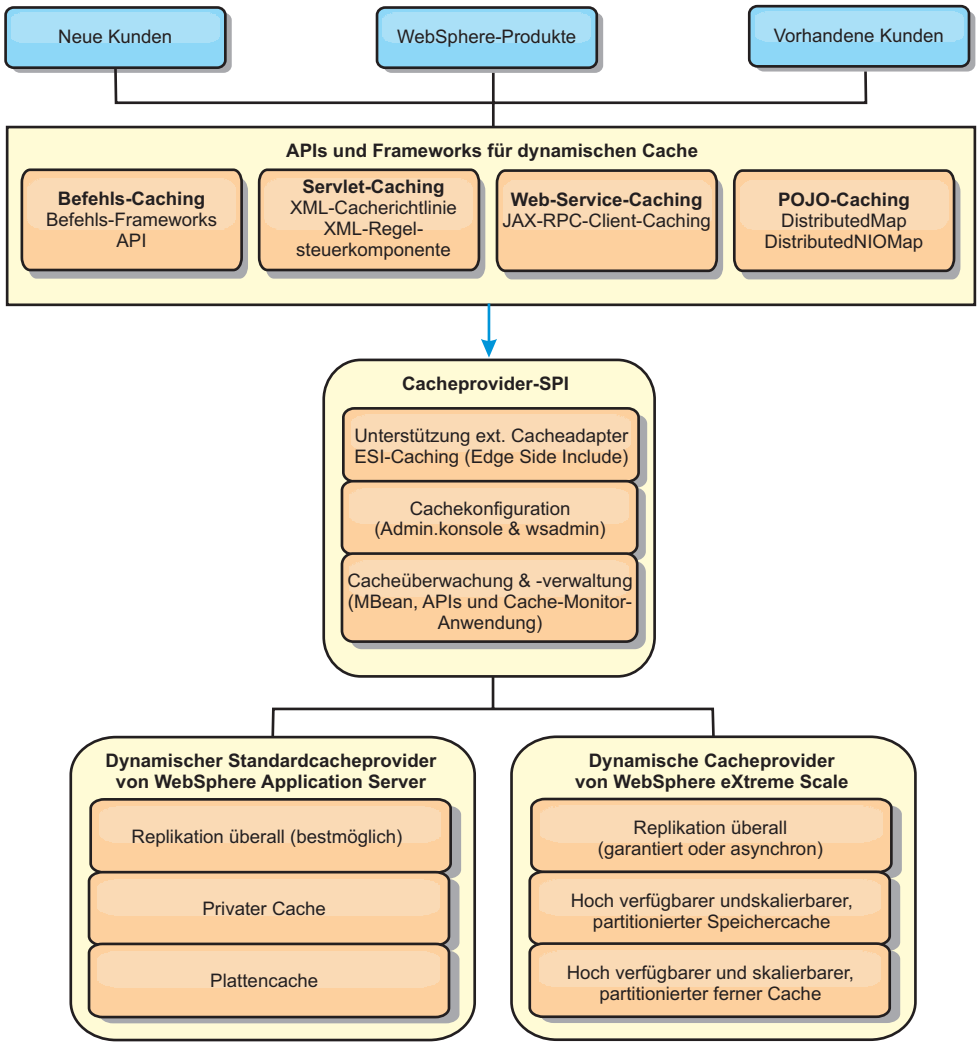
- Wählen Sie die Größe der Sitzungstabelle im Hinblick auf ein ausgewogenes Verhältnis zwischen lokaler Speicherkapazität und Leistung. Das Produkt lagert die Sitzungsbenutzerdaten aus, wenn die maximale Größe des lokalen Sitzungscaches erreicht ist.
- Das Produkt unterstützt HTTP-Sitzungen im Anwendungskontext entsprechend der Servlet-Spezifikation, um Sicherheitsprobleme und Probleme aufgrund von Attributnamenskonflikten zu vermeiden. Sie können Sitzungen in Anwendungskontexten gemeinsam nutzen, indem Sie ihre Singleton-Klasse verwenden.
- Der Sitzungsreplikationsmanager repliziert Sitzungsdaten für hohe Verfügbarkeit, indem er auf Änderungen der Sitzungsdaten wartet und die gespeicherten Sitzungsdaten bedarfsgerecht erneut lädt. Dies wird durch Wiederverwendung des anbieterspezifischen Basissitzungsmanagers für den Webcontainer erreicht. Die Sitzung wird über verschiedene native Sitzungs-IDs verknüpft. Sitzungsdaten aus einem Webcontainer werden von einem anderen Webcontainer übernommen, indem die Sitzungsdaten aus der ObjectGrid-Instanz erneut geladen werden. Sitzungs-ID und Erstellungszeit vor und nach dem Failover können unterschiedlich sein.

Dynamischer Cacheprovider

Die Anwendungsprogrammierschnittstelle (API, Application Programming Interface) für dynamischen Cache steht Java-EE-Anwendungen zur Verfügung, die in WebSphere Application Server implementiert sind. Der dynamische Cacheprovider kann genutzt werden, um Geschäftsdaten und generierte HTML zwischenzuspeichern oder um die zwischengespeicherten Daten in der Zelle über den Datenreplikationsservice (DRS) zu synchronisieren.

Übersicht

Früher war der einzige Serviceprovider für die Anwendungsprogrammierschnittstelle "Dynamic Cache" die in WebSphere Application Server integrierte Standardsteuerkomponente für dynamische Cache. Kunden können die Serviceproviderschnittstelle für dynamischen Cache in WebSphere Application Server verwenden, um eXtreme Scale in den dynamischen Cache zu integrieren. Indem Sie die Funktionalität konfigurieren, ermöglichen Sie Anwendungen, die mit der API für dynamischen Cache geschrieben wurden, und Anwendungen, die Caching auf Containerebene verwenden (z. B. Servlets), die Nutzung der Features und Leistungsfunktionen von WebSphere eXtreme Scale.



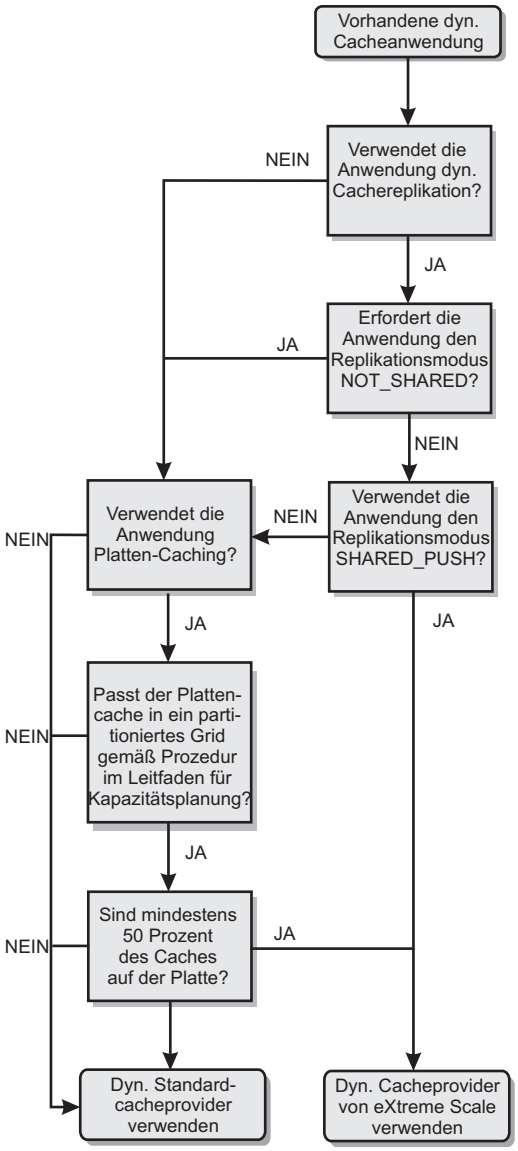
Sie können den dynamischen Cacheprovider, wie im Abschnitt Dynamischen Cacheprovider für WebSphere eXtreme Scale konfigurieren beschrieben, installieren und konfigurieren.

Einsatz von WebSphere eXtreme Scale festlegen

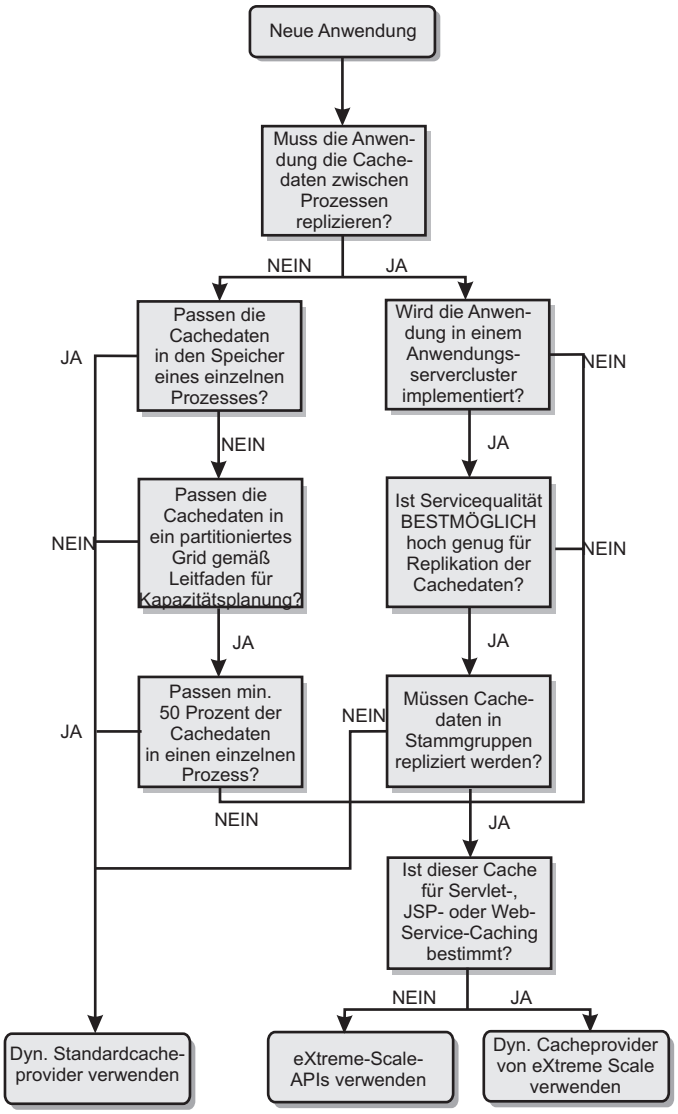
Die verfügbaren Features in WebSphere eXtreme Scale erweitern die verteilten Funktionen der API "Dynamic Cache" weit über das hinaus, was die Standardsteuerkomponente für dynamischen Cache und der Datenreplikationsservice bieten. Mit eXtreme Scale können Sie Caches erstellen, die wirklich auf mehrere Server verteilt, anstatt nur repliziert und unter den Servern synchronisiert werden. Außerdem sind die Caches von eXtreme Scale transaktionsorientiert und hoch verfügbar, wodurch sichergestellt wird, dass jeder Server denselben Inhalt für den dynamischen Cacheservice sieht. WebSphere eXtreme Scale bietet eine höhere Servicequalität für die Cachereplikation als der Datenreplikationsservice.

Diese Vorteile bedeuten jedoch nicht, dass der dynamische Cacheprovider von eXtreme Scale die richtige Wahl für jede Anwendung ist. Verwenden Sie die folgenden Entscheidungsbäume und Featurevergleichsmatrizes, um festzustellen, welche Technologie sich am besten für Ihre Anwendung eignet.

Entscheidungsbaum für die Migration vorhandener Anwendungen mit dynamischem Cache



Entscheidungsbaum für die Auswahl eines Cacheproviders für neue Anwendungen



Featurevergleich

Tabelle 7. Featurevergleich

Cachefeatures	Standardprovider	eXtreme-Scale-Provider	eXtreme-Scale-API
Lokales Speicher-Caching	x	x	x
Verteiltes Caching	Integriert	Integriert, integriert-partitioniert und fern-partitioniert	Mehrere
Linear skalierbar		x	x
Zuverlässige Replikation (synchron)		ORB	ORB
Plattenüberlauf	x		

Tabelle 7. Featurevergleich (Forts.)

Cachefeatures	Standardprovider	eXtreme-Scale-Provider	eXtreme-Scale-API
Bereinigung	LRU-/TTL-/Heap-Speicher-basiert	LRU/TTL (pro Partition)	Mehrere
Ungültigmachen	x	x	x
Beziehungen	Abhängigkeits-IDs, Schablonen	Abhängigkeits-IDs, Schablonen	x
Suchoperationen ohne Schlüssel			Abfrage und Index
Back-End-Integration			Loader
Transaktionsorientiert		Implizit	x
Schlüsselbasierter Speicher	x	x	x
Ereignisse und Listener	x	x	x
Integration von WebSphere Application Server	Nur einzelne Zelle	Mehrere Zellen	Zellenunabhängig
Unterstützung von Java Standard Edition		x	x
Überwachung und Statistiken	x	x	x
Sicherheit	x	x	x

Tabelle 8. Nahtlose Technologieintegration

Cachefeatures	Standardprovider	eXtreme-Scale-Provider	eXtreme-Scale-API
Caching von Servlet-/JSP-Ergebnissen in WebSphere Application Server	Version 5.1+	Version 6.1.0.25+	
Caching von Web-Service-Ergebnissen (JAX-RPC) in WebSphere Application Server	Version 5.1+	Version 6.1.0.25+	
Caching von HTTP-Sitzungen			x
Cacheprovider für OpenJPA und Hibernate			x
Datenbank-synchronisation mit OpenJPA und Hibernate			x

Tabelle 9. Programmierschnittstellen

Cachefeatures	Standardprovider	eXtreme-Scale-Provider	eXtreme-Scale-API
Befehlsbasierte API	Befehls-Framework-API	Befehls-Framework-API	DataGrid-API
Map-basierte API	DistributedMap-API	DistributedMap-API	ObjectMap-API
EntityManager-API			x

Eine ausführlichere Beschreibung zur Funktionsweise der verteilten Caches in eXtreme Scale finden Sie in den Informationen zu Implementierungskonfigurationen im *Administratorhandbuch*.

Anmerkung: In einem verteilten eXtreme-Scale-Cache können nur Einträge gespeichert werden, bei denen sowohl der Schlüssel als auch der Wert die Schnittstelle "java.io.Serializable" implementieren.

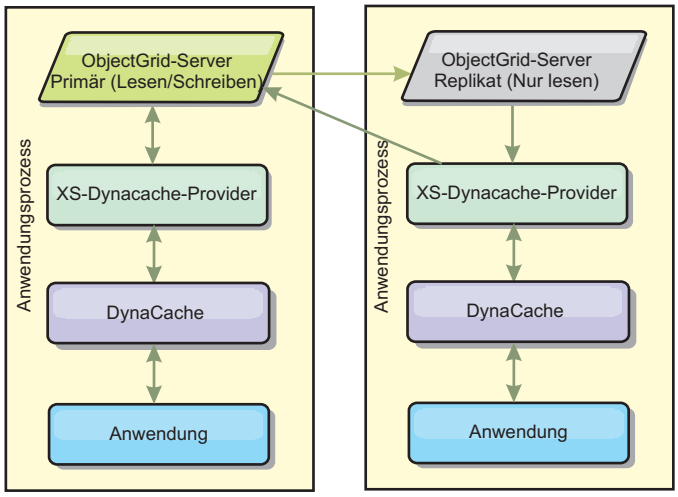
Topologietypen

Ein dynamischer Cacheservice, der mit dem eXtreme-Scale-Provider erstellt wurde, kann in jeder der drei verfügbaren Topologien implementiert werden und ermöglicht Ihnen, den Cache speziell an Ihren Leistungs-, Ressourcen-und Verwaltungsbedarf anzupassen. Diese Topologien sind die integrierte, die integriert, partitionierte Topologie und die ferne Topologie.

Integrierte Topologie

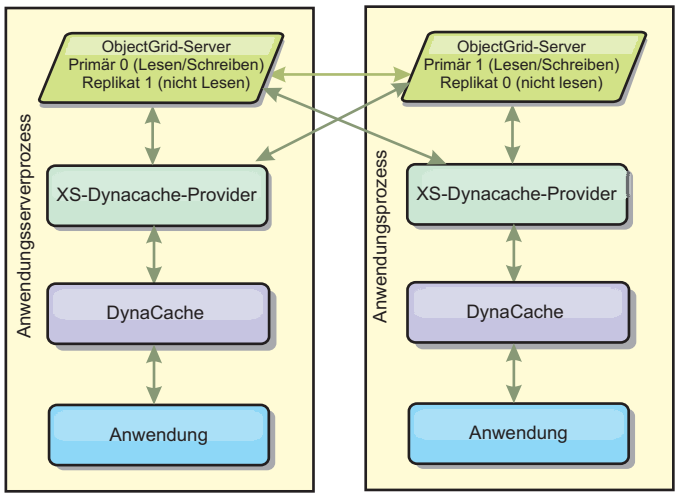
Die integrierte Topologie ist dem dynamischen Standardcache- und DRS-Provider sehr ähnlich. Verteilte Cacheinstanzen, die mit der integrierten Topologie erstellt werden, enthalten eine vollständige Kopie des Caches in jedem Prozess von eXtreme Scale, der auf den dynamischen Cacheservice zugreift und ermöglichen damit die lokale Durchführung aller Leseoperationen. Alle Schreiboperationen erfolgen über einen Einzelserverprozess, in dem die Transaktionssperren verwaltet werden, bevor sie auf den restlichen Servern repliziert werden. Deshalb eignet sich diese Topologie besser für Workloads, bei denen Anzahl der Leseoperationen im Cache im Vergleich mit den Schreiboperationen im Cache wesentlich höher ist.

Mit der integrierten Topologie werden neue oder aktualisierte Cacheeinträge nicht sofort in jedem einzelnen Serverprozess sichtbar. Ein Cacheeintrag wird erst dann sichtbar (selbst für den Server, der ihn generiert hat), wenn er über die asynchronen Replikationsservices von WebSphere eXtreme Scale weitergegeben wird. Diese Services arbeiten so schnell, wie es die Hardware zulässt, aber es kommt trotzdem zu einer kleinen Verzögerung. Die integrierte Topologie wird in der folgenden Abbildung veranschaulicht:



Integrierte, partitionierte Topologie

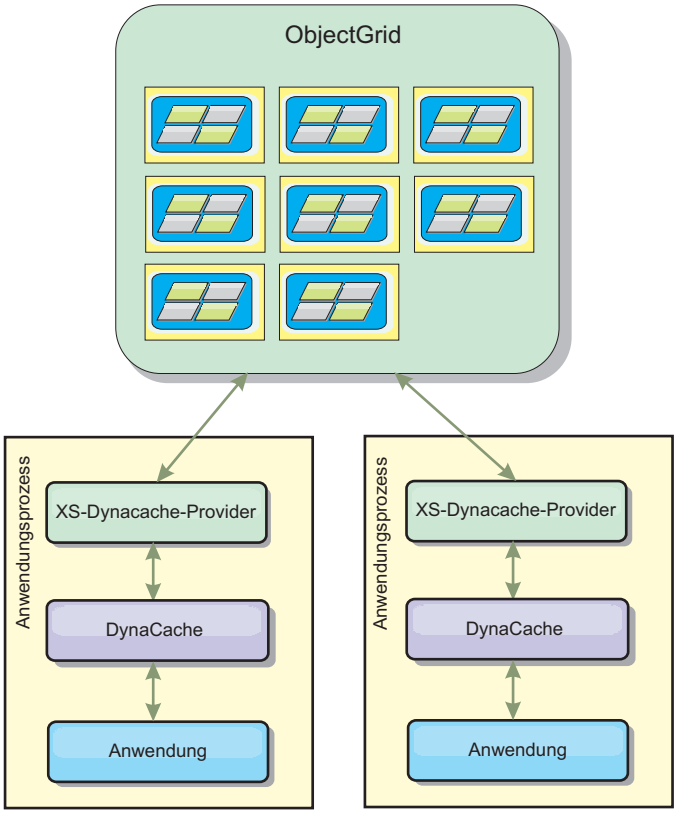
Für Workloads, bei denen die Anzahl der Schreiboperationen größer-gleich der Anzahl der Leseoperationen ist, wird die integrierte, partitionierte Topologie oder die ferne Topologie empfohlen. Bei der integrierten, partitionierten Topologie sind alle Cachedaten in den Prozessen von WebSphere Application Server enthalten, die auf den Cache zugreifen. In jedem einzelnen Prozess wird jedoch nur ein Teil der Cachedaten gespeichert. Alle Lese- und Schreiboperationen für die Daten in dieser "Partition" erfolgen über den Prozess, d. h., dass die meisten Anforderungen an den Cache über einen Prozedurfernaufruf verarbeitet werden. Dies führt zu einer höheren Latenzzeit für Leseoperationen als bei der integrierten Topologie, aber die Kapazität des verteilten Caches für die Verarbeitung von Lese- und Schreiboperationen steigt linear zur Anzahl der Prozesse von WebSphere Application Server, die auf den Cache zugreifen. Außerdem wird die maximale Größe des Caches bei dieser Topologie nicht durch die Größe eines einzelnen WebSphere-Prozesses beschränkt. Da jeder Prozess nur einen Teil des Caches enthält, entspricht die maximale Cachegröße der summierten Größe alle Prozesse abzüglich der Kosten für den Prozess. Die integrierte, partitionierte Topologie wird in der folgenden Abbildung veranschaulicht:



Angenommen, Sie haben ein Grid von Serverprozessen mit jeweils 256 MB freiem Heap-Speicher für einen dynamischen Cacheservice. Der dynamische Standardcacheprovider und der Provider von eXtreme Scale, der die integrierte Topologie verwendet, sind beide auf eine Speichercachegröße von 256 MB abzüglich Prozesskosten beschränkt. Lesen Sie hierzu den Abschnitt "Kapazitätsplanung und hohe Verfügbarkeit" weiter hinten in diesem Dokument. Der eXtreme-Scale-Provider, der die integrierte, partitionierte Topologie verwendet, ist auf eine Cachegröße von 1 GB abzüglich Prozesskosten beschränkt. Der Provider von WebSphere eXtreme Scale unterstützt somit speicherinterne dynamische Cacheservices, die die Größe eines einzelnen Serverprozesses überschreiten. Der dynamische Standardcacheprovider stützt sich auf die Verwendung eines Plattencaches, damit Cacheinstanzen über die Größe eines Einzelprozesses hinweg anwachsen können. In vielen Situationen können Sie sich durch den Provider von WebSphere eXtreme Scale den Plattencache und die kostenintensiven Plattenspeichersysteme, die für eine angemessene Leistung des Plattencaches erforderlich sind, sparen.

Ferne Topologie

Die ferne Topologie kann ebenfalls verwendet werden, um den Bedarf an einem Plattencache zu umgehen. Der einzige Unterschied zwischen der fernen Topologie und der integrierten, partitionierten Topologie besteht darin, dass bei einer fernen Topologie alle Cachedaten außerhalb der Prozesse von WebSphere Application Server gespeichert werden. WebSphere eXtreme Scale unterstützt eigenständige Containerprozesse für Cachedaten. Diese Containerprozesse haben geringere Kosten als ein Prozess von WebSphere Application Server und sind auch nicht auf die Verwendung einer bestimmten Java Virtual Machine (JVM) beschränkt. Die Daten für einen dynamischen Cacheservice, auf den mit einem 32-Bit-Prozess von WebSphere Application Server zugegriffen wird, könnte sich beispielsweise in einem eXtreme-Scale-Containerprozess befinden, der in einer 64-Bit-JVM ausgeführt wird. Dies ermöglicht Benutzern, die höhere Speicherkapazität von 64-Bit-Prozessen für das Caching zu nutzen, ohne das zusätzliche Kosten für den 64-Bit-Betrieb bei den Anwendungsserverprozessen anfallen. Die ferne Topologie wird in der folgenden Abbildung veranschaulicht:



Datenkomprimierung

Ein weiteres Leistungsfeature, das der dynamische Cacheprovider von WebSphere eXtreme Scale Benutzern für die Verwaltung der Cachekosten bietet, ist Komprimierung. Der dynamische Standardcacheprovider unterstützt keine Komprimierung zwischengespeicherter Daten im Speicher. Mit dem Provider von eXtreme Scale ist dies möglich. Die Cachekomprimierung mit dem Komprimierungsalgorithmus (deflate) kann in jeder der drei verteilten Topologien aktiviert werden. Die Aktivierung der Komprimierung erhöht die Kosten für Lese- und Schreiboperationen, steigert aber drastisch die Cachedichte für Anwendungen wie das Caching von Servlets und JSP-Dateien.

Lokaler Speichercache

Der dynamische Cacheprovider von WebSphere eXtreme Scale kann auch zur Unterstützung dynamischer Cacheinstanzen verwendet werden, in denen die **Replikation inaktiviert** ist. Wie beim dynamischen Standardcacheprovider können in diesen Caches nicht serialisierbare Daten gespeichert werden. Außerdem bieten Sie in großen Mehrprozessorunternehmensservern häufig eine bessere Leistung als der dynamische Standardcacheprovider, weil der Codepfad von eXtreme Scale auf einem möglichst hohe Anzahl gemeinsamer Zugriff auf den Speichercache ausgelegt ist.

Funktionale Unterschiede zwischen der dynamischen Cachesteuerkomponente und eXtreme Scale

Bei lokalen Speichercaches, in denen die Replikation inaktiviert ist, sind keine wesentlichen funktionalen Unterschiede zwischen Caches, die vom dynamischen Standardcacheprovider unterstützt werden, und WebSphere eXtreme Scale bemerkbar.

Dem Benutzer sollten eigentlich keine funktionalen Unterschiede zwischen den beiden Caches auffallen, abgesehen davon, dass von WebSphere eXtreme Scale unterstützte Caches keine Auslagerung auf die Platte oder Statistiken und Operationen unterstützen, die sich auf die Größe des Caches im Speicher beziehen.

Bei Caches, in denen die Replikation aktiviert ist, unterscheiden sich die von den meisten Aufrufen der API "Dynamic Cache" zurückgegebenen Ergebnisse kaum, unabhängig davon, ob der Kunde den dynamischen Standardcacheprovider oder den dynamischen Cacheprovider von eXtreme Scale verwendet. Für einige Operationen kann das Verhalten der dynamischen Cachesteuerkomponente nicht mit eXtreme Scale emuliert werden.

Statistiken zum dynamischen Cache

Statistiken zum dynamischen Cache werden über die Anwendung "CacheMonitor" oder die MBean der API "dynamic cache" berichtet. Wenn Sie den dynamischen Cacheprovider von eXtreme Scale verwenden, werden Statistiken weiterhin über diese Schnittstellen berichtet, aber der Kontext der Statistikwerte ist anders.

Wenn eine dynamische Cacheinstanz von drei Servern (A, B und C) gemeinsam genutzt wird, gibt das Statistikobjekt des dynamischen Caches nur Statistiken für die Kopie des Caches auf dem Server zurück, auf dem der Aufruf abgesetzt wurde. Wenn die Statistiken auf Server A abgerufen werden, spiegeln sie nur die Aktivitäten auf Server A wider.

Mit eXtreme Scale gibt es nur einen einzigen verteilten Cache, der von allen Servern gemeinsam genutzt wird, so dass es nicht möglich ist, die meisten Statistiken auf Serverbasis zu verfolgen, wie es der dynamische Standardcacheprovider tut. Im Folgenden finden Sie eine Liste der Statistiken, die von der API für Cachestatistiken berichtet werden, einschließlich einer Beschreibung ihrer Funktionalität, wenn Sie den dynamischen Cacheprovider von WebSphere eXtreme Scale verwenden. Wie der Standardprovider werden diese Statistiken nicht synchronisiert und können deshalb bei gleichzeitigen Workloads bis zu 10 % variieren.

- **Cachetreffer**: Cachetreffer werden auf Serverbasis verfolgt. Wenn der Datenverkehr auf Server A 10 Cachetreffer generiert und der Datenverkehr auf Server B 20 Cachetreffer, werden in den Cachestatistiken 10 Cachetreffer auf Server A und 20 Cachetreffer auf Server B berichtet.
- **Cachefehler**: Cachefehler werden wie Cachetreffer auf Serverbasis berichtet.
- **Speichercacheeinträge**: In dieser Statistik wird die Anzahl der Cacheeinträge im verteilten Cache berichtet. Jeder Server, der auf den Cache zugreift, berichtet denselben Wert für diese Statistik, und der Wert gibt die Gesamtanzahl der Cacheeinträge im Speicher über alle Server wider.
- **Speichercachegröße in MB**: Diese Metrik wird nur für Caches unterstützt, die eine ferne, integrierte oder integrierte partitionierte Topologie verwenden. Sie berichtet die Anzahl der Megabyte des Java-Heap-Speichers, der vom Cache im gesamten Grid belegt wird. Diese Statistik berichtet die Heap-Speicherbeleigung nur für primäre Partitionen. Replikate müssen gesondert berücksichtigt werden. Da die Standardeinstellung für die ferne und die integrierte partitionierte Topologie ein asynchrones Replikat ist, müssen Sie diese Zahl verdoppeln, um die echte Speicherbelegung des Caches zu erhalten.
- **Entfernte Cacheeinträge**: Diese Statistik berichtet die Gesamtanzahl der Einträge, die über beliebige Methoden aus dem Cache entfernt wurden, und ist ein kumulierter Wert für den gesamten verteilten Cache. Wenn der Datenverkehr auf Server A 10 ungültig gemachte Einträge und der Datenverkehr auf Server B 20 ungültig gemachte Einträge generiert, ist der Wert auf beiden Servern 30.

- **Entfernte LRU-Cacheeinträge**: Diese Statistik ist wie die entfernten Cacheeinträge ein kumulierter Wert. Sie verfolgt die Anzahl der Einträge, die entfernt wurden, um den Cache unter seiner maximalen Größe zu halten.
- **Ungültig gemachte Einträge wegen Zeitlimitüberschreitung**: Diese Statistik ist ebenfalls eine kumulierte Statistik und verfolgt die Anzahl der Einträge, die entfernt wurden, weil das zulässige Zeitlimit überschritten wurde.
- **Explizit ungültig gemachte Einträge**: Diese Statistik ist ebenfalls eine kumulierte Statistik und verfolgt die Anzahl der Einträge, die durch direkte Ungültigmachung nach Schlüssel, Abhängigkeits-ID oder Schablone entfernt wurden.
- **Erweiterte Statistiken**: Der dynamische Cacheprovider von eXtreme Scale exportiert die folgenden Schlüsselzeichenfolgen für erweiterte Statistiken.
  - **com.ibm.websphere.xs.dynacache.remote_hits**: Die Gesamtanzahl der im eXtreme-Scale-Container verfolgten Cachetreffer. Diese Statistik ist eine kumulierte Statistik, und der Wert in der erweiterten Statistik-Map ist ein Wert vom Typ `long`.
  - **com.ibm.websphere.xs.dynacache.remote_misses**: Die Gesamtanzahl der im eXtreme-Scale-Container verfolgten Cachefehler. Diese Statistik ist eine kumulierte Statistik, und der Wert in der erweiterten Statistik-Map ist ein Wert vom Typ `long`.

Zurückgesetzte Statistiken berichten

Der dynamische Cacheprovider ermöglicht Ihnen, Cachestatistiken zurückzusetzen. Mit dem Standardprovider löscht die Rücksetzoperation nur die Statistiken des betroffenen Servers. Der dynamische Cacheprovider von eXtreme Scale verfolgt die meisten seiner Statistikdaten in fernen Cachecontainern. Diese Daten werden weder gelöscht noch geändert, wenn die Statistiken zurückgesetzt werden. Stattdessen wird das Verhalten des dynamischen Standardcaches im Client simuliert, indem die Differenz zwischen dem aktuellen Wert einer bestimmten Statistik und dem Wert dieser Statistik beim letzten Aufruf der Rücksetzoperation auf diesem Server berichtet wird.

Wenn der Datenverkehr auf Server A beispielsweise 10 entfernte Cacheeinträge generiert, werden in den Statistiken für Server A und Server B 10 entfernte Einträge berichtet. Werden die Statistiken auf Server B jetzt zurückgesetzt und generiert der Datenverkehr auf Server A weitere 10 entfernte Einträge, werden in den Statistiken für Server A 20 entfernte Einträge berichtet und in den Statistiken für Server B 10.

Dynamische Cacheereignisse

Die API "Dynamic Cache" ermöglicht Benutzern die Registrierung von Ereignis-Listenern. Wenn Sie eXtreme Scale als dynamischen Cacheprovider verwenden, arbeiten die Ereignis-Listener für lokale Speichercaches erwartungsgemäß.

Bei verteilten Caches richtet sich das Ereignisverhalten nach der verwendeten Topologie. Für Caches, die die integrierte Topologie verwenden, werden Ereignisse auf dem Server generiert, der die Schreiboperationen verarbeitet, dem so genannten primären Shard. Das bedeutet, dass nur ein einziger Server Ereignisbenachrichtigungen empfängt, aber dieser Server erhält alle Ereignisbenachrichtigungen, die normalerweise vom dynamischen Cacheprovider erwartet werden. Da WebSphere eXtreme Scale das primäre Shard zur Laufzeit auswählt, ist es nicht möglich sicherzustellen, dass immer ein bestimmter Serverprozess diese Ereignisse empfängt.

Integrierte partitionierte Caches generieren Ereignisse auf jedem Server, der eine Partition des Caches enthält. Wenn ein Cache also 11 Partitionen hat und jeder Ser-

ver in einem Grid von WebSphere Application Server Network Deployment mit 11 Servern eine der Partitionen enthält, empfängt jeder Server die dynamischen Cacheereignisse für die Cacheeinträge, die er enthält. Es gibt keinen einzigen Prozess, der alle Ereignisse sieht, es sei denn, alle 11 Partitionen befinden sich in diesem einen Serverprozess. Wie bei der integrierten Topologie ist es nicht möglich sicherzustellen, dass immer ein bestimmter Serverprozess einen bestimmten Ereignissatz empfängt.

Caches, die die ferne Topologie verwenden, unterstützen keine dynamischen Cacheereignisse.

MBean-Aufrufe

Der dynamische Cacheprovider von WebSphere eXtreme Scale unterstützt kein Platten-Caching. Alle MBean-Aufrufe, die sich auf Platten-Caching beziehen, funktionieren nicht.

Zuordnung einer Replikationsrichtlinie für den dynamischen Cache

Der in WebSphere Application Server integrierte dynamische Cacheprovider unterstützt mehrere Richtlinien für die Cachereplikation. Diese Richtlinien können global oder für jeden Cacheeintrag konfiguriert werden. In der Dokumentation zum dynamischen Cache finden Sie eine Beschreibung dieser Replikationsrichtlinien.

Der dynamische Cacheprovider von eXtreme Scale berücksichtigt diese Richtlinien nicht direkt. Die Replikationsmerkmale eines Caches richten sich nach dem konfigurierten Typ für die verteilte eXtreme-Scale-Topologie und gelten für alle Werte in diesem Cache, unabhängig von der vom dynamischen Cacheservice definierten Replikationsrichtlinie für den Eintrag. Die folgende Liste enthält alle Replikationsrichtlinien, die vom dynamischen Cacheservice unterstützt werden, und veranschaulicht, welche eXtreme-Scale-Topologie ähnliche Replikationsmerkmale bietet.

Beachten Sie, dass der dynamische Cacheprovider von eXtreme Scale die Richtlinieneinstellungen für die DRS-Replikation für einen Cache bzw. Cacheeintrag ignoriert. Benutzer müssen die Topologie auswählen, die ihren Replikationsanforderungen am besten entspricht.

- NOT_SHARED: Derzeit kommt keine der vom dynamischen Cacheprovider von eXtreme Scale bereitgestellten Topologien dieser Richtlinie nahe. Das bedeutet, dass alle Daten, die im Cache gespeichert werden, Schlüssel und Werte haben müssen, die java.io.Serializable implementieren.
- SHARED_PUSH: Die integrierte Topologie kommt dieser Replikationsrichtlinie nahe. Wenn ein Cacheeintrag erstellt wird, wird er in allen Servern repliziert. Server suchen nur lokal nach Cacheeinträgen. Wird ein Eintrag lokal nicht gefunden, wird davon ausgegangen, dass der Eintrag nicht vorhanden ist, und es werden keine anderen Server nach dem Eintrag abgefragt.
- SHARED_PULL und SHARED_PUSH_PULL: Die integrierte, partitionierte Topologie und die ferne Topologie kommen dieser Replikationsrichtlinie nahe. Der verteilte Status des Caches ist in allen Servern vollständig konsistent.

Diese Informationen werden hauptsächlich bereitgestellt, damit Sie sicherstellen können, dass die Topologie Ihre Anforderungen bezüglich verteilter Konsistenz erfüllt. Wenn die integrierte Topologie beispielsweise die bessere Wahl für Ihre Implementierungs- und Leistungsanforderungen ist, Sie aber die von SHARED_PUSH_PULL unterstützte Cachekonsistenzstufe benötigen, sollten Sie die integ-

rierte, partitionierte Topologie verwenden, selbst wenn die Leistung geringfügig schwächer ist.

Sicherheit

Sie können dynamische Cacheinstanzen, die in einer integrierten oder einer integrierten, partitionierten Topologie ausgeführt werden, mit den in WebSphere Application Server integrierten Sicherheitsfunktionen schützen. Weitere Informationen hierzu finden Sie in der Dokumentation zum Sichern von Anwendungsservern im Information Center von WebSphere Application Server.

Wenn ein Cache in einer fernen Topologie ausgeführt wird, kann ein eigenständiger extreme-Scale-Client eine Verbindung zum Cache herstellen und den Inhalt der dynamischen Cacheinstanz beeinflussen. Der dynamische Cacheprovider von eXtreme Scale besitzt ein Verschlüsselungsfeature, das nur geringe Kosten produziert, aber verhindern kann, dass Cachedaten von Clients, die keine Clients von WebSphere Application Server sind, gelesen oder geändert werden. Zum Aktivieren dieses Features setzen Sie den optionalen Parameter **com.ibm.websphere.xs.dynacache.encryption_password** in jeder Instanz von WebSphere Application Server, die auf den dynamischen Cacheprovider zugreift, auf denselben Wert. Daraufhin werden der Wert und die Benutzermetadaten für den Cacheeintrag mit 128-Bit-AES-Verschlüsselung verschlüsselt. Es ist sehr wichtig, dass für alle Server derselbe Wert definiert wird. Servers können keine Daten lesen, die von Servern mit einem anderen Wert für diesen Parameter in den Cache gestellt werden.

Wenn der Provider von eXtreme Scale erkennt, dass unterschiedliche Werte für diese Variable in demselben Cache gesetzt sind, generiert er eine Warnung im Protokoll des Containerprozesses von eXtreme Scale.

Lesen Sie in der Dokumentation zu eXtreme Scale die Informationen zur Sicherheit von WebSphere eXtreme Scale, wenn SSL- oder Clientauthentifizierung erforderlich ist.

Weitere Informationen

- Redbook zum dynamischen Cache
- Dokumentation zum dynamischen Cache
  - WebSphere Application Server 7.0
  - WebSphere Application Server 6.1
- Dokumentation zum Datenreplikationsservice
  - WebSphere Application Server 7.0
  - WebSphere Application Server 6.1

Kapazitätsplanung und hohe Verfügbarkeit (dynamisches Caching)

Die Anwendungsprogrammierschnittstelle (API, Application Programming Interface) für dynamischen Cache steht Java-EE-Anwendungen zur Verfügung, die in WebSphere Application Server implementiert sind. Der dynamische Cache kann genutzt werden, um Geschäftsdaten und generierte HTML zwischenzuspeichern oder um die zwischengespeicherten Daten in der Zelle über den Datenreplikationsservice (DRS) zu synchronisieren.

Übersicht

Alle dynamischen Cacheinstanzen, die mit dem dynamischen Cacheprovider von WebSphere eXtreme Scale erstellt werden, sind standardmäßig hoch verfügbar. Die Stufe und die Speicherkosten der hohen Verfügbarkeit sind von der verwendeten Topologie abhängig.

Wenn Sie die integrierte Topologie verwenden, ist die Cachegröße auf den freien Speicher in einem einzelnen Serverprozess beschränkt, und in jedem Serverprozess wird eine vollständige Kopie des Caches gespeichert. Solange auch nur ein einziger Serverprozess aktiv ist, bleibt auch der Cache aktiv. Die Cachedaten gehen nur dann verloren, wenn alle Server, die auf den Cache zugreifen, beendet werden.

Beim Caching mit der integrierten partitionierten Topologie ist die Cachegröße auf den summierten freien Speicher aller Serverprozesse beschränkt. Standardmäßig verwendet der dynamische Cacheprovider von eXtreme Scale ein Replikat für jedes primäre Shard, d. h., jedes einzelne zwischengespeicherte Datenelement wird zweimal gespeichert.

Verwenden Sie die folgende Formel A, um die Kapazität eines integrierten partitionierten Caches zu bestimmen:

Formel A

F * C / (1 + R) = M

Für diese Formel gilt Folgendes:

- F = Freier Speicher pro Containerprozess
- C = Anzahl der Container
- R = Anzahl der Replikate
- M = Gesamtgröße des Caches

Für ein Grid von WebSphere Network Deployment mit 256 MB verfügbarem Speicher in jedem Prozess und 4 Serverprozessen insgesamt können in einer Cacheinstanz über alle diese Server verteilt 512 Megabyte Daten gespeichert werden. In diesem Modus kann der Cache einen Serverabsturz ohne Datenverlust "überleben". Es könnten sogar nacheinander zwei Server beendet werden, ohne dass irgendwelche Daten verloren gehen. Für das vorherige Beispiel lautet die Formel deshalb wie folgt:

256 MB * 4 Container/ (1 primäres Shard + 1 Replikat) = 512 MB

Caches, die die ferne Topologie verwenden, haben ähnliche Größenmerkmale wie Caches, die die integrierte partitionierte Topologie verwenden, sind jedoch auf den summierten verfügbaren Speicher aller Containerprozesse von eXtreme Scale beschränkt.

In fernen Topologien kann die Anzahl der Replikate erhöht werden, um eine höhere Stufe der Verfügbarkeit zu erreichen, wodurch sich jedoch der Speicheraufwand erhöht. In den meisten dynamischen Cacheanwendungen ist dies in der Regel nicht erforderlich, aber Sie können die Datei "dynacache-remote-deployment.xml" bearbeiten, um die Anzahl der Replikate zu erhöhen.

Verwenden Sie die folgenden Formeln (B und C), um zu berechnen, welche Auswirkungen das Hinzufügen weiterer Replikate auf die hohe Verfügbarkeit des Caches hat.

Formel B

N = Minimum(T -1, R)

Für diese Formel gilt Folgendes:

- N = Anzahl der Prozesse, die gleichzeitig abstürzen
- T = Gesamtanzahl der Container
- R = Gesamtanzahl der Replikate

Formel C

Obere Grenze(T/ (1+N)) = m

Für diese Formel gilt Folgendes:

- T = Gesamtanzahl der Container
- N = Gesamtanzahl der Replikate
- m = Erforderliche Mindestanzahl an Containern für die Unterstützung der Cachedaten

Informationen zur Leistungsoptimierung mit dem dynamischen Cacheprovider finden Sie im Abschnitt Dynamischen Cacheprovider optimieren.

Cachegröße festlegen

Bevor eine Anwendung, die den dynamischen Cacheprovider von WebSphere eXtreme Scale verwendet, implementiert werden kann, müssen die allgemeinen Principals, die im vorherigen Abschnitt beschrieben wurden, mit den Umgebungsdaten für die Produktionssysteme kombiniert werden. Der erste zu ermittelnde Wert ist die Gesamtanzahl der Containerprozesse und der verfügbare Hauptspeicher jedes einzelnen Prozesses zum Speichern der Cachedaten. Wenn Sie die integrierte Topologie verwenden, befinden sich die Cachecontainer in den Prozessen von WebSphere Application Server. d. h., es gibt einen Container pro Server, der den Cache verwendet. Die Bestimmung der Speicherkosten der Anwendung ohne aktiviertes Caching und ohne WebSphere Application Server ist die beste Methode zu ermitteln, wie viel Speicher im Prozess verfügbar ist. Zur Ermittlung dieses Werts bietet sich die Analyse der Daten einer ausführlichen Garbage-Collection an. Wenn Sie die ferne Topologie verwenden, können diese Informationen anhand der Ausgabe der ausführlichen Garbage-Collection für einen neu gestarteten eigenständigen Container ermittelt werden, der noch nicht mit Cachedaten gefüllt wurde. Ein letzter Aspekt, der bei der Ermittlung des für Cachedaten verfügbaren Speichers pro Prozess berücksichtigt werden muss, ist die Reservierung einer gewissen Menge des Heap-Speichers für die Garbage-Collection. Die Summe aus Containerkosten (Umgebung mit WebSphere Application Server oder eigenständige Umgebung) und reservierter Größe für den Cache sollte nicht mehr als 70 % der Gesamtgröße des Heap-Speichers betragen.

Nachdem Sie diese Informationen zusammengestellt haben, können Sie die Werte in die zuvor beschriebene Formel A eintragen, um die maximale Größe für den partitionierten Cache zu bestimmen. Sobald die maximale Größe bekannt ist, müssen Sie im nächsten Schritt die Gesamtanzahl der Cacheeinträge bestimmen, die

unterstützt werden können. Hierfür muss zunächst die durchschnittliche Größe pro Cacheeintrag bestimmt werden. Eine einfache Methode zur Bestimmung dieses Werts ist, 10 % auf die Größe des Kundenobjekts aufzuschlagen. Ausführlichere Informationen zur Festlegung der Größe von Cacheeinträgen bei der Verwendung des dynamischen Caches finden Sie in der Veröffentlichung "Tuning guide for dynamic cache and data replication service".

Wenn die Komprimierung aktiviert ist, wirkt diese sich auf die Größe des Kundenobjekts aus, aber nicht auf die Kosten des Caching-Systems. Verwenden Sie die folgende Formel, um die Größe eines zwischengespeicherten Objekts zu bestimmen, wenn Sie mit Komprimierung arbeiten:

S = O * C + O * 0.10

Für diese Formel gilt Folgendes:

- S = Durchschnittliche Größe eines zwischengespeicherten Objekts
- O = Durchschnittliche Größe eines nicht komprimierten Kundenobjekts
- C = Komprimierungsfaktor als Bruchzahl

Komprimierung: Ein Komprimierungsfaktor von 2 zu 1 ist 1/2 = 0,50. Je kleiner der Faktor ist, desto besser. Wenn das zu speichernde Objekt ein normales POJO mit überwiegend primitiven Datentypen ist, können Sie von einem Komprimierungsfaktor von 0,60 bis 0,70 ausgehen. Wenn das zwischengespeicherte Objekte ein Servlet, eine JSP-Datei oder ein Web-Service-Objekt ist, ist die optimale Methode für die Bestimmung des Komprimierungsfaktors, ein repräsentatives Beispielobjekt mit einem Komprimierungsdienstprogramm zu komprimieren. Sollte dies nicht möglich sein, können Sie im Allgemeinen von einem Komprimierungsfaktor von 0,2 bis 0,35 für diesen Typ von Daten ausgehen.

Anschließend verwenden Sie diese Informationen, um die Gesamtanzahl der Cacheeinträge zu bestimmen, die unterstützt werden können. Verwenden Sie die folgende Formel D:

Formel D

T = S / A

Für diese Formel gilt Folgendes:

- T = Gesamtanzahl der Cacheeinträge
- S = Verfügbare Gesamtkapazität für Cachedaten laut Berechnung mit Formel A
- A = Durchschnittliche Größe eines Cacheeintrags

Abschließend müssen Sie die Cachegröße in der dynamischen Cacheinstanz festlegen, damit dieser Grenzwert wirksam wird. Der dynamische Cacheprovider von WebSphere eXtreme Scale unterscheidet sich in dieser Beziehung vom dynamischen Standardcacheprovider. Verwenden Sie die folgende Formel E, um den Wert zu bestimmen, der für die Cachegröße in der dynamischen Cacheinstanz festzulegen ist:

Formel E

Cs = Ts / Np

Für diese Formel gilt Folgendes:

- Ts = Gesamtzielgröße für den Cache

- Cs = In der dynamischen Cacheinstanz festzulegende Cachegröße
- Np = Anzahl der Partitionen (der Standardwert ist 47)

Setzen Sie die Größe der dynamischen Cacheinstanz in allen Servern, die die Cacheinstanz gemeinsam nutzen, auf den mit der Formel E berechneten Wert.

Kapitel 4. Übersicht über Skalierbarkeitskonzepte

Skalierbarkeit ermöglicht in einer Implementierung von WebSphere eXtreme Scale je nach ausgewählter Konfiguration die Verteilung von Daten auf eine Gruppe von Servern (Containern).

Skalierbarkeit

WebSphere eXtreme Scale ist über die Verwendung partitionierter Daten skalierbar und kann bei Bedarf auf Tausende von Containern skaliert werden, weil jeder Container von den anderen Containern unabhängig ist.

WebSphere eXtreme Scale teilt Datenbestände in verschiedene Partitionen auf, die zur Laufzeit zwischen Prozessen oder sogar zwischen Maschinen verschoben werden können. Sie können beispielsweise mit einer Implementierung von vier Servern beginnen und diese dann auf eine Implementierung mit zehn Servern erweitern, wenn der Cachebedarf steigt. So, wie Sie weitere physische Maschinen und Verarbeitungseinheiten für die vertikale Skalierbarkeit hinzufügen können, können Sie die elastischen Horizontalskalierungsfähigkeiten von eXtreme Scale durch Partitionierung erweitern. Dies ist ein weiterer wichtiger Unterschied zwischen speicherinternen Datenbanken und eXtreme Scale (Daten-Grid), da speicherinterne Datenbanken nur vertikal skalierbar sind.

Mit WebSphere eXtreme Scale können Sie außerdem eine Gruppe von APIs verwenden, um transaktionsorientiert auf diese partitionierten und optional verteilten Daten zuzugreifen. In Bezug auf die Leistung sind die Optionen, die Sie für die Interaktion mit dem Cache auswählen, genauso signifikant wie die Funktionen für die Verwaltung des Caches für Verfügbarkeit.

Anmerkung: Skalierbarkeit ist nicht verfügbar, wenn Container miteinander kommunizieren. Das Verfügbarkeitsmanagement- oder Stammgruppenprotokoll ist ein $O(N^2)$-Wartungsalgorithmus für Überwachungssignale und Sichten, wird aber abgeschwächt, indem die Anzahl der Stammgruppen-Member bei ungefähr 20 gehalten wird. Replikation findet nur im Peer-to-Peer-Modus zwischen Shards statt.

Verteilte Clients

Das Clientprotokoll von WebSphere eXtreme Scale unterstützt eine hohe Anzahl an Clients. Die Partitionierungsstrategie bietet Unterstützung, indem davon ausgegangen wird, dass nicht immer alle Clients an allen Partitionen interessiert sind, weil Verbindungen auf mehrere Container verteilt sein können. Clients werden direkt mit den Partitionen verbunden, so dass die Latenzzeit auf eine übertragene Verbindung beschränkt ist.

Grids, Partitionen und Shards

Ein verteiltes eXtreme-Scale-Grid ist in Partitionen unterteilt. Eine Partition enthält einen exklusiven Teil der Daten. Eine Partition setzt sich aus einem oder mehreren so genannten Shards (engl. Shard = Scherbe) zusammen: einem primären Shard und Replikat-Shards. Sie müssen keine Replikat-Shards in einer Partition verwenden, aber Replikat-Shards bieten eine hohe Verfügbarkeit. Unabhängig davon, ob Ihre Implementierung ein unabhängiges speicherinternes Daten-Grid oder ein spei-

cherinterner Datenbankverarbeitungsbereich ist, ist der Datenzugriff in eXtreme Scale im Wesentlichen von den Shard-Konzepten abhängig.

Die Daten für eine Partition werden zur Laufzeit in einer Gruppe von Shards gespeichert. Diese Gruppe von Shards setzt sich aus einem primären Shard und unter Umständen einem oder mehreren Replikat-Shards zusammen. Ein Shard ist die kleinste Einheit, die eXtreme Scale in einer Java Virtual Machine hinzufügen oder entfernen kann.

Es sind zwei Verteilungsstrategien vorhanden: FIXED_PARTITIONS (Standardeinstellung) und PER_CONTAINER. Die folgende Beschreibung konzentriert sich auf die Verwendung der Strategie FIXED_PARTITIONS.

Anzahl der Shards

Wenn Ihre Umgebung beispielsweise zehn Partitionen mit einer Million Objekten ohne Replikate enthält, sind zehn Shards vorhanden, in denen jeweils 100.000 Objekte gespeichert sind. Wenn Sie diesem Szenario ein Replikat hinzufügen, ist in jeder Partition ein zusätzliches Shard vorhanden. In diesem Fall sind dann 20 Shards vorhanden, zehn primäre Shards und zehn Replikat-Shards. Und wieder sind in jedem dieser Shards 100.000 Objekte gespeichert. Jede Partition setzt sich aus einem primären Shard und einem oder mehreren (N) Replikat-Shards zusammen. Die Festlegung der optimalen Shard-Anzahl ist ein kritischer Faktor. Wenn Sie eine geringe Anzahl an Shards konfigurieren, werden die Daten nicht gleichmäßig auf die Shards verteilt, was zu Fehlern wegen Speicherengpässen und Problemen durch Prozessüberlastung führt. Sie müssen bei der Skalierung mindestens zehn Shards für jede JVM festlegen. Wenn Sie das Grid implementieren, verwenden Sie möglicherweise eine höhere Anzahl an Partitionen.

Anzahl der Shards pro JVM

Szenario mit einer geringen Anzahl an Shards für jede JVM

Daten werden in einer JVM in Shard-Einheiten hinzugefügt und entfernt. Shards werden nie unterteilt. Wenn 10 GB Daten und 20 Shards zum Speichern dieser Daten vorhanden sind, enthält jedes Shard durchschnittlich 500 MB Daten. Wenn das Grid auf neun JVMs verteilt ist, hat jede JVM durchschnittlich zwei Shards. Da 20 nicht durch 9 teilbar ist, haben einige JVMs drei Shards. Die Verteilung ist wie folgt:

- 7 JVMs mit 2 Shards
- 2 JVMs mit 3 Shards

Da jedes Shard 500 MB Daten enthält, ist die Verteilung der Daten unsymmetrisch. Die sieben JVMs mit zwei Shards enthalten jeweils 1 GB Daten. Die beiden JVMs mit drei Shards enthalten jeweils 50 % mehr Daten (oder 1,5 GB), was eine sehr viel höhere Speicherlast darstellt. Da diese beiden JVMs drei Shards haben, empfangen sie auch 50 % mehr Anforderungen für ihre Daten. Deshalb führt eine geringe Anzahl an Shards für jede JVM zu einem Ungleichgewicht. Zur Verbesserung der Leistung erhöhen Sie die Anzahl der Shards für jede JVM.

Szenario mit einer höheren Anzahl an Shards pro JVM

In diesem Szenario wird eine sehr viel höhere Anzahl an Shards verwendet. Es gibt 101 Shards mit 9 JVMs für 10 GB Daten. In diesem Fall werden in jedem Shard 99 MB Daten gespeichert. Die Shards sind wie folgt auf die JVMs verteilt:

- 7 JVMs mit 11 Shards
- 2 JVMs mit 12 Shards

Die beiden JVMs mit jeweils 12 Shards enthalten jetzt nur 99 MB mehr Daten als die anderen Shards. Dies ergibt eine Differenz von 9 %. In diesem Szenario ist die Last gleichmäßiger verteilt als in dem Szenario mit einer geringeren Anzahl an Shards, wo die Differenz bei 50 % liegt. Vom Standpunkt der Prozessorauslastung betrachtet, fällt auf die beiden JVMs mit den jeweils 12 Shards im Vergleich mit den sieben JVMs mit jeweils 11 Shards nur 9 % mehr Arbeit. Durch die Erhöhung der Shard-Anzahl in jeder JVM wird die Daten- und Prozessorauslastung fair und gleichmäßig verteilt.

Verwenden Sie 10 Shards für jede JVM, wenn Sie Ihr System in maximaler Größe bzw. die Ausführung der maximalen Anzahl an JVMs in Ihrem System planen.

Zusätzliche Verteilungsfaktoren

Die Anzahl der Partitionen, die Verteilungsstrategie und der Typ der Replikate wird in der Implementierungsrichtlinie definiert. Die Anzahl der verteilten Shards richtet sich nach der definierten Implementierungsrichtlinie. Die Eigenschaften "numInitialContainers", "minSyncReplicas", "developmentMode", "maxSyncReplicas" und "maxAsyncReplicas" haben Einfluss darauf, wo und wann Partitionen und Replikate verteilt werden können. Wenn die Initialisierung des Servers die mit maxSyncReplicas und maxAsyncReplicas angegebene Anzahl zu verteilender Replikate nicht unterstützt, müssen Sie möglicherweise weitere Replikate verteilen, wenn Sie später weitere Server starten. Bei der Planung der Shard-Anzahl pro JVM ist die die maximale Anzahl der Shards, einschließlich der Replikate, davon abhängig, dass genügend JVMs für die Unterstützung der angeforderten maximalen Anzahl an Replikaten gestartet sind. Ein Replikat wird niemals in demselben Prozess wie das primäre Shard ausgeführt, da dies beim Verlust des Prozesses zu einem Verlust des primären Shards und des Replikats führen würde. Wenn "developmentMode" auf "false" gesetzt ist, werden die primären Shards und die Replikat-Shards nicht an dieselbe Maschine verteilt.

Partitionierung

Sie können die Partitionierung verwenden, um hohe Datenvolumen in der Java Virtual Machine (JVM) zu speichern. Zur Partitionierung von Daten verwenden Sie ein anwendungsdefiniertes Schema, um die Daten aufzuteilen. Mit WebSphere eXtreme Scale kann durch Partitionierung sowohl die Skalierbarkeit als auch die Verfügbarkeit verbessert werden.

Partitionierung ist der Mechanismus, den WebSphere eXtreme Scale verwendet, um eine Anwendung horizontal zu skalieren. Partitionierung ist die Aufteilung des Anwendungszustands in Komponenten, die jeweils eine Teilmenge der vollständigen Instanzdaten enthalten. Partitionierung ist nicht dasselbe wie RAID-Striping (Redundant Array of Independent Disks), bei dem Teile jeder Instanz auf alle Stripes verteilt werden. Jede Partition enthält die vollständigen Daten für einzelne Einträge. Partitionierung ist ein effektives Mittel für Skalierung, aber nicht auf alle Anwendungen anwendbar. Anwendungen, die Transaktionsgarantien über große Datengruppen hinweg erfordern, können nicht skaliert und nicht effizient partitioniert werden, und deshalb unterstützt eXtreme Scale derzeit keine partitionsübergreifende zweiphasige Festschreibung.

Wichtig: Wählen Sie die Anzahl der Partitionen sorgfältig aus. Die in der Implementierungsrichtlinie definierte Partitionsanzahl wirkt sich direkt auf die Anzahl

der Container aus, auf die eine Anwendung skaliert werden kann. Jede Partition setzt sich aus einem primären Shard und der konfigurierten Anzahl an Replikat-Shards zusammen. Mit der Formel `(Anzahl_Partitionen*(1 + Anzahl_Replikate))` wird die Anzahl der Container berechnet, die verwendet werden kann, um eine einzelne Anwendung horizontal zu skalieren.

Partitionen verwenden

Ein Grid kann viele, bei Bedarf sogar Tausende Partitionen haben. Ein Grid kann vertikal bis auf das Produkt aus Partitionsanzahl und Shard-Anzahl pro Partition skaliert werden. Wenn Sie beispielsweise 16 Partitionen haben und jede Partition ein einziges primäres Shard oder zwei Shards hat, können Sie potenziell eine Skalierung auf bis 32 Java Virtual Machines erreichen. In diesem Fall wird für jede JVM ein einziges Shard definiert. Sie müssen, basierend auf der erwarteten Anzahl an Java Virtual Machines, die wahrscheinlich verwendet werden, eine angemessene Anzahl an Partitionen auswählen. Mit jedem Shard erhöht sich die Prozessor- und Speicherbelegung für das System. Das System ist so konzipiert, dass es horizontal skaliert werden kann, um diesen Aufwand entsprechend der Anzahl verfügbarer Java Virtual Machines des Servers handhaben zu können.

Anwendungen sollten nicht Tausende von Partitionen verwenden, wenn die Anwendung in einem Grid von vier Container-JVMs ausgeführt wird. In der Konfiguration der Anwendung sollte eine angemessene Anzahl an Shards für jede Container-JVM angegeben werden. Eine unverhältnismäßige Konfiguration sind beispielsweise 2000 Partitionen mit zwei Shards, die in vier Container-JVMs ausgeführt werden. Diese Konfiguration würde bedeuten, dass 4000 Shards auf vier Container-JVMs bzw. 1000 Shards pro Container-JVM verteilt werden.

Eine bessere Konfiguration wären 10 Shards für jede erwartetet Container-JVM. Diese Konfiguration bietet Ihnen trotzdem die Möglichkeit einer elastischen Skalierung bis hin zum Zehnfachen der Erstkonfiguration bei einer gleichzeitig angemessenen Anzahl an Shards pro Container-JVM.

Stellen Sie sich das folgende Skalierungsbeispiel vor: Sie haben derzeit sechs Computer mit jeweils zwei Container-JVMs. Sie erwarten ein Wachstum auf 20 Computer in einem Zeitraum von drei Jahren. Mit 20 Computern haben Sie 40 Container-JVMs, und Sie wählen 60 aus, um pessimistisch zu sein. Sie möchten 4 Shards pro Container-JVM haben. Sie haben 60 potenzielle Container bzw. insgesamt 240 Shards. Wenn Sie pro Partition ein primäres Shard und ein Replikat-Shard haben, möchten Sie 120 Partitionen haben. Diese Beispielrechnung ergibt 240, geteilt durch 12 Container-JVMs, bzw. 20 Shards pro Container-JVM für die Erstimplementierung mit der Möglichkeit einer späteren Skalierung auf 20 Computer.

ObjectMap und Partitionierung

Bei der Verteilungsstrategie FIXED_PARTITION werden Maps auf Partitionen und die Hash-Werte von Schlüsseln auf verschiedene Partitionen verteilt. Der Client muss nicht wissen, zu welcher Partition die Schlüssel gehören. Wenn ein MapSet mehrere Maps hat, müssen die Maps in separaten Transaktionen festgeschrieben werden.

Entitäten und Partitionierung

EntityManager-Entitäten besitzen eine Optimierung, die Clients hilft, die mit Entitäten in einem Server arbeiten. Das Entitätsschema im Server für das MapSet kann eine einzige Stammentität spezifizieren. Der Client muss auf alle Entitäten über

diese Stammentität zugreifen. Der EntityManager findet dann die zugehörigen Entitäten über diese Stammentität, ohne dass die zugehörigen Maps einen gemeinsamen Schlüssel haben müssen. Die Stammentität stellt die Affinität zu einer einzelnen Partition her. Diese Partition wird nach der Herstellung der Affinität für all Entitätsabrufe innerhalb der Transaktion verwendet. Diese Affinität kann zu Speichereinsparungen führen, weil die zugehörigen Maps keinen gemeinsamen Schlüssel benötigen. Die Stammentität muss mit einer modifizierten Entitätsannotation angegeben werden, wie im folgenden Beispiel gezeigt wird:

```
@Entity(schemaRoot=true)
```

Verwenden Sie die Entität, um das Stammelement des Objektgraphen zu ermitteln. Der Objektgraph definiert die Beziehungen zwischen Entitäten. Jede verlinkte Entität muss in dieselbe Partition aufgelöst werden. Es wird davon ausgegangen, dass sich alle untergeordneten Elemente in derselben Partition wie das Stammelement befinden. Die untergeordneten Entitäten im Objektgraphen sind von einem Client aus nur über die Stammentität zugänglich. Stammentitäten sind in partitionierten Umgebungen immer erforderlich, wenn ein Client von eXtreme Scale für die Kommunikation mit dem Server verwendet wird. Es kann nur ein einziger Stammentitätstyp pro Client definiert werden. Stammentitäten sind nicht erforderlich, wenn XTP-ObjectGrids (Extreme Transaction Processing) verwendet werden, da die gesamte Kommunikation mit der Partition über direkten lokalen Zugriff und nicht über den Client/Server-Mechanismus erfolgt.

Verteilung und Partitionen

Sie können zwischen zwei Verteilungsstrategien für WebSphere eXtreme Scale wählen: feste Partitionsverteilung und containerbezogene Verteilung. Die Auswahl der Verteilungsstrategie hat Einfluss darauf, wie die Implementierungskonfiguration Partitionen im fernen Grid verteilt.

Feste Partitionsverteilung

Sie können die Verteilungsstrategie in der XML-Datei für Implementierungsrichtlinien festlegen. Die Standardverteilungsstrategie ist die feste Partitionsverteilung, die mit der Einstellung `FIXED_PARTITION` aktiviert wird. Die Anzahl primärer Shards, die auf die verfügbaren Container verteilt werden, entspricht der Anzahl der Partitionen, die Sie mit dem Element "numberOfPartitions" konfiguriert haben. Wenn Sie Replikate konfiguriert haben, wird die minimale Gesamtanzahl der verteilten Shards mit der folgenden Formel definiert: `((1 primäres Shard + Mindestanzahl synchroner Shards) * Anzahl definierter Partitionen)`. Die maximale Gesamtanzahl verteilter Shards wird mit der folgenden Formel definiert: `((1 primäres Shard + maximale Anzahl synchroner Shards + maximale Anzahl asynchroner Shards) * Partitionen)`. Ihre Implementierung von WebSphere eXtreme Scale verteilt die Shards auf die verfügbaren Container. Die Schlüssel jeder Map werden, basierend auf der definierten Gesamtanzahl der Partitionen, in den zugeordneten Partitionen verschlüsselt. Die Schlüssel werden auch dann in derselben Partition verschlüsselt, wenn die Partition aufgrund eines Failovers oder aufgrund von Serveränderungen verschoben wird.

Wenn numberPartitions beispielsweise den Wert 6 und minSync den Wert 1 für MapSet1 hat, ist die Gesamtanzahl der Shards für dieses MapSet 12, weil jede der 6 Partitionen ein synchrones Replikat erfordert. Wenn drei Container gestartet sind, verteilt WebSphere eXtreme Scale vier Shards pro Container für MapSet1.

Containerbezogene Verteilung

Die alternative Verteilungsstrategie ist die containerbezogene Verteilung, die mit der Einstellung `PER_CONTAINER` für placementStrategy im MapSet-Element in der XML-Implementierungsdatei aktiviert wird. Bei dieser Strategie entspricht die Anzahl primärer Shards, die an jeden neuen Container verteilt wird, der Anzahl der Partitionen, *P*, die Sie konfiguriert haben. Die Implementierungsumgebung von WebSphere eXtreme Scale verteilt *P* Replikate jeder Partition für jeden verbleibenden Container. Die Einstellung von numInitialContainers wird ignoriert, wenn Sie die containerbezogene Verteilung verwenden. Die Partitionen werden größer, wenn die Container zunehmen. Die Schlüssel für Maps sind bei dieser Strategie nicht auf eine bestimmte Partition festgelegt. Der Client leitet Anforderungen an eine Partition weiter und verwendet eine wahlfreie primäre Partition. Wenn in Client die Verbindung zu derselben Sitzung wiederherstellen möchte, die er für die erneute Suche eines Schlüssels verwendet hat, müssen Sie ein Sitzungs-Handle verwenden.

Weitere Informationen finden Sie im Abschnitt zur Verwendung eines Sitzungs-Handles für das Routing im *Programmierhandbuch*.

Wenn ein Failover stattfindet oder Server gestoppt werden, verschiebt die Umgebung von WebSphere eXtreme Scale die primären Shards in der containerbezogenen Verteilungsstrategie, wenn diese noch Daten enthalten. Wenn die Shards leer sind, werden sie verworfen. Bei der containerbezogenen Strategie werden alte primäre Shards nicht beibehalten, weil neue primäre Shards für jeden Container verteilt werden.

WebSphere eXtreme Scale lässt die containerbezogene Verteilung als Alternative zur so genannten "typischen" Verteilung zu, einer Lösung mit festgelegten Partitionen, bei der der Schlüssel einer Map verschlüsselt auf einer dieser Partitionen gespeichert wird. Im containerbezogenen Fall (den Sie mit PER_CONTAINER festlegen) verteilt die Implementierung die Partitionen auf die Gruppe der online befindlichen Containerserver und skaliert diese automatisch, wenn Container dem Server-Grid hinzugefügt bzw. aus diesem entfernt werden. Ein Grid mit festen Partitionen eignet sich für schlüsselbasierte Grids, bei denen die Anwendung ein Schlüsselobjekt verwendet, um die Daten im Grid zu suchen. Die Alternative wird im Folgenden beschrieben.

Beispiel für ein containerbezogenes Grid

Grids mit der Verteilungsstrategie PER_CONTAINER sind anders. Sie legen die Verteilungsstrategie PER_CONTAINER für das Grid mit dem Attribut "placementPolicy" in der XML-Implementierungsdatei fest. Anstatt die gewünschte Gesamtpartitionsanzahl im Grid zu konfigurieren, geben Sie an, wie viele Partitionen Sie pro gestartetem Container wünschen.

Wenn Sie beispielsweise die Anzahl der Partitionen pro Container auf 5 setzen, erstellt eXtreme Scale beim Starten eines Containers 5 neue anonyme primäre Partitions-Shards in diesem Container und alle erforderlichen Replikat-Shards in den anderen bereits implementierten Containern.

Im Folgenden sehen Sie eine mögliche Folge in einer containerbezogenen Umgebung beim Anwachsen des Grids.

1. Start von Container C0 mit 5 primären Shards (P0-P4)
   - C0 enthält P0, P1, P2, P3, P4

2. Start von Container C1 mit 5 weiteren primären Shards (P5-P9). Die Replikat-Shards werden gleichmäßig auf die Container verteilt.
   - C0 enthält: P0, P1, P2, P3, P4, R5, R6, R7, R8, R9
   - C1 enthält: P5, P6, P7, P8, P9, R0, R1, R2, R3, R4
3. Start von Container C2 mit 5 weiteren primären Shards (P10-P14). Die Replikat-Shards werden gleichmäßig neu verteilt.
   - C0 enthält: P0, P1, P2, P3, P4, R7, R8, R9, R10, R11, R12
   - C1 enthält: P5, P6, P7, P8, P9, R2, R3, R4, R13, R14
   - C2 enthält: P10, P11, P12, P13, P14, R5, R6, R0, R1

Das Muster wird fortgesetzt, wenn weitere Container gestartet werden. Bei jedem Start eines weiteren Containers werden 5 neue primäre Partitionen erstellt, und die Replkate werden gleichmäßig neu auf die verfügbaren Container im Grid verteilt.

Anmerkung: WebSphere eXtreme Scale verschiebt primäre Shards bei der Strategie PER_CONTAINER nicht, es werden nur Replikate verschoben.

Denken Sie daran, dass die Partitionsnummern beliebig sind, d. h. keinen Bezug zu Schlüsseln haben, und deshalb schlüsselbasiertes Routing nicht verwendet werden kann. Wenn ein Container gestoppt wird, werden die erstellten Partitions-IDs für diesen Container nicht mehr verwendet, und es entsteht eine Lücke in den Partitions-IDs. Wenn Container C2 in dem Beispiel ausfällt, sind die Partitionen P5-P9 nicht mehr verfügbar. Es bleiben nur die Partitionen P0-P4 und P10-P14 übrig. Somit ist ein schlüsselbasiertes Hashing nicht möglich.

Die Verwendung von Zahlen wie 5 oder 10 (was wahrscheinlicher ist) für die Anzahl der Partitionen pro Container empfiehlt sich, wenn man an die Auswirkungen eines Containerausfalls denkt. Damit die Last der Host-Shards gleichmäßig im Grid verteilt wird, benötigen Sie mehr als nur eine Partition pro Container. Wenn Sie nur eine einzige Partition pro Container verwenden und ein Container ausfällt, muss ein einziger Container (der mit dem entsprechenden Replikat-Shard) die volle Last des ausgefallenen primären Shards tragen. In diesem Fall verdoppelt sich sofort die Last für den Container. Wenn Sie jedoch 5 Partitionen pro Container haben, übernehmen 5 Container die Last des ausgefallenen Containers, was die Auswirkungen auf jeden Container um 80 Prozent reduziert. Die Verwendung mehrerer Partitionen pro Container verringert die potenziellen Auswirkungen auf jeden Container im Allgemeinen erheblich. Stellen Sie sich einen Fall vor, in dem die Last eines Containers unerwartet stark ansteigt. Die Replikationslast dieses Containers wird auf 5 Container und nicht nur auf einen einzigen verteilt.

Containerbezogene Richtlinie verwenden

Es gibt verschiedene Szenarien, in denen die containerbezogene Strategie eine ideale Konfiguration ist, z. B. bei der Replikation von HTTP-Sitzungen oder beim Status von Anwendungssitzungen. In einem solchen Fall ordnet ein HTTP-Router eine Sitzung einem Servlet-Container zu. Der Servlet-Container muss eine HTTP-Sitzung erstellen und wählt eines der fünf lokalen primären Partitions-Shards für die Sitzung auswählen. Die "ID" der ausgewählten Partition wird anschließend in einem Cookie gespeichert. Der Servlet-Container hat jetzt lokalen Zugriff auf den Sitzungsstatus, was einen Zugriff ohne Latenzzeit auf die Daten für diese Anforderung bedeutet, solange die Sitzungsaffinität aufrecht erhalten bleibt. eXtreme Scale repliziert alle Änderungen an der Partition.

Stellen sich die Auswirkungen eines Falls in der Praxis vor, in dem Sie mehrere Partitionen pro Container (sagen wir erneut 5) haben. Natürlich haben Sie mit jedem neuen Container, der gestartet wird, 5 weitere primäre Partitions-Shards und 5 weitere Replikat-Shards. Mit der Zeit müssen weitere Partitionen erstellt werden, und diese dürfen weder verschoben noch entfernt werden. So verhalten sich die Container aber in Wirklichkeit nicht. Wenn ein Container gestartet wird, enthält er 5 primäre Shards, die so genannten primären "Ausgangs-Shards". Wenn der Container ausfällt, werden die Replikate zu primären Shards,und eXtreme Scale erstellt 5 weitere Replikate, um die hohe Verfügbarkeit aufrecht zu erhalten (sofern Sie die automatische Reparatur nicht inaktivieren). Die neuen primären Shards befinden sich in einem anderen Container als dem, von dem sie erstellt wurden, und werden deshalb als "fremde" primäre Shards bezeichnet. Die Anwendung sollte neue Statusinformationen oder Sitzungen nie in einem fremden primären Shards speichern. Das fremde primäre Shard hat irgendwann keine Einträge mehr, und eXtreme Scale löscht dann dieses fremde Shard und die zugehörigen Replikate. Die fremden primären Shards unterstützen die weitere Verfügbarkeit vorhandener Sitzungen (aber keine neuen Sitzungen).

Ein Client kann weiterhin mit einem Grid interagieren, das nicht schlüsselbasiert ist. Der Client startet eine Transaktion und speichert Daten im Grid unabhängig von Schlüsseln. Er fordert bei der Sitzung ein SessionHandle-Objekt an, ein serialisierbares Handle, das dem Client bei Bedarf die Interaktion mit derselben Partition ermöglicht. Weitere Informationen finden Sie im Abschnitt zur Verwendung eines SessionHandle für das Routing im *Programmierhandbuch*. WebSphere eXtreme Scale wählt eine Partition für den Client aus der Liste der primären Ausgangspartitions-Shards aus. Er gibt keine fremde primäre Partition zurück. Das SessionHandle kann beispielsweise in ein HTTP-Cookie serialisiert werden, das später wieder in ein Cookie konvertiert wird. Anschließend können die APIs von WebSphere eXtreme Scale eine Sitzung abrufen, die über das SessionHandle wieder an dieselbe Partition gebunden wird.

Anmerkung: Für die Interaktion mit einem PER_CONTAINER-Grid können keine Agenten verwendet werden.

Vorteile

Die vorherige Beschreibung weicht von einem normalen FIXED_PARTITION- oder Hash-Grid ab, weil der containerbezogene Client Daten an einer Stelle im Grid speichert, ein Handle für die Daten abruft und das Handle verwendet, um erneut auf die Daten zuzugreifen. Es gibt keinen von der Anwendung bereitgestellten Schlüssel wie im Fall mit festgelegten Partitionen.

Ihre Implementierung erstellt keine neue Partition für jede Sitzung. In einer containerbezogenen Implementierung müssen die Schlüssel, die zum Speichern von Daten in der Partition verwendet werden, deshalb innerhalb dieser Partition eindeutig sein. Sie lassen von Ihrem Client beispielsweise eine eindeutige Sitzungs-ID erstellen und verwenden diese als Schlüssel, um Informationen in den Maps dieser Partition zu suchen. Anschließend interagieren mehrere Clientsitzungen mit derselben Partition, so dass die Anwendung eindeutige Schlüssel verwenden muss, um Sitzungsdaten in jeder angegebenen Partition zu speichern.

In den vorherigen Beispielen wurden 5 Partitionen verwendet, aber der Parameter "numberOfPartitions" in der ObjectGrid-XML-Datei kann verwendet werden, um die gewünschte Anzahl an Partitionen anzugeben. Die Einstellung gilt nicht pro Grid, sondern pro Container. (Die Anzahl der Replikate wird wie bei der Richtlinie für festgelegte Partitionen angegeben.)

Die containerbezogene Richtlinie kann auch für mehrere Zonen verwendet werden. Wenn möglich, gibt eXtreme Scale ein SessionHandle für eine Partition zurück, deren primäres Shard sich in derselben Zone wie dieser Client befindet. Der Client kann die Zone als Parameter für den Container oder über eine API angeben. Die Clientzonen-ID kann mit `serverproperties` oder `clientproperties` festgelegt werden.

Die PER_CONTAINER-Strategie für ein Grid eignet sich für Anwendungen, die dialogbasierte Statusinformationen anstelle von datenbankorientierten Daten speichern. Der Schlüssel für den Zugriff auf die Daten ist eine Dialog-ID und bezieht sich nicht auf einen bestimmten Datenbanksatz. Diese Strategie bietet eine höhere Leistung (weil die primären Partitions-Shards beispielsweise mit den Servlets zusammengefasst werden können) und eine einfachere Konfiguration (ohne Partitionen und Container berechnen zu müssen).

Einzelpartitionstransaktionen und Grid-übergreifende Partitionstransaktionen

Der Hauptunterschied zwischen WebSphere eXtreme Scale und traditionellen Datenspeicherlösungen wie relationalen oder speicherinternen Datenbanken ist die Verwendung der Partitionierung, die eine lineare Skalierung des Caches ermöglicht. Die wichtigen Transaktionstypen, die berücksichtigt werden müssen, sind Einzelpartitionstransaktionen und Grid-übergreifende Partitionstransaktionen.

Im Allgemeinen können Interaktionen mit dem Cache, wie im Folgenden beschrieben, in die Kategorien "Einzelpartitionstransaktionen" und "Grid-übergreifende Partitionstransaktionen" eingeteilt werden.

Einzelpartitionstransaktionen

Einzelpartitionstransaktionen sind die vorzuziehende Methode für die Interaktion mit Caches in WebSphere eXtreme Scale. Wenn eine Transaktion auf eine Einzelpartition beschränkt ist, ist sie standardmäßig auf eine einzelne Java Virtual Machine und damit auf einen einzelnen Servercomputer beschränkt. Ein Server kann *M* dieser Transaktionen pro Sekunde ausführen, und wenn Sie *N* Computer haben, sind `M*N` Transaktionen pro Sekunde möglich. Wenn sich Ihr Geschäft erweitert und Sie doppelt so viele dieser Transaktionen pro Sekunde ausführen müssen, können Sie *N* verdoppeln, indem Sie weitere Computer kaufen. Auf diese Weise können Sie Kapazitätsanforderungen erfüllen, ohne die Anwendung zu ändern, Hardware zu aktualisieren oder die Anwendung außer Betrieb zu nehmen.

Zusätzlich zu der Möglichkeit, den Cache so signifikant skalieren zu können, maximieren Einzelpartitionstransaktionen auch die Verfügbarkeit des Caches. Jede Transaktion ist nur von einem einzigen Computer abhängig. Jeder der anderen `(N-1)` Computer kann ausfallen, ohne den Erfolgt oder die Antwortzeit der Transaktion zu beeinflussen. Wenn Sie also mit 100 Computern arbeiten und einer dieser Computer ausfällt, wird nur 1 Prozent der Transaktionen, die zum Zeitpunkt des Ausfalls dieses Servers unvollständig sind, rückgängig gemacht. Nach dem Ausfall des Servers verlagert WebSphere eXtreme Scale die Partitionen des ausgefallenen Servers auf die anderen 99 Computer. In diesem kurzen Zeitraum vor der Durchführung der Operation können die anderen 99 Computer weiterhin Transaktionen ausführen. Nur die Transaktionen, an denen die Partitionen beteiligt sind, die umgelagert werden, sind blockiert. Nach Abschluss des Failover-Prozesses ist der Cache mit 99 Prozent seiner ursprünglichen Durchsatzkapazität wieder vollständig betriebsbereit. Nachdem der ausgefallene Server ersetzt und der Ersatzserver dem

Grid hinzugefügt wurde, kehrt der Cache zu einer Durchsatzkapazität von 100 Prozent zurück.

Grid-übergreifende Transaktionen

Was Leistung, Verfügbarkeit und Skalierbarkeit betrifft, sind Grid-übergreifende Transaktionen das Gegenteil von Einzelpartitionstransaktionen. Grid-übergreifende Transaktionen greifen auf jede Partition und damit auf jeden Computer in der Konfiguration zu. Jeder Computer im Grid wird aufgefordert, einige Daten zu suchen und anschließend das Ergebnis zurückzugeben. Die Transaktion kann erst abgeschlossen werden, nachdem jeder Computer geantwortet hat, und deshalb wird der Durchsatz des gesamten Grids durch den langsamsten Computer beschränkt. Das Hinzufügen von Computern macht den langsamsten Computer nicht schneller und verbessert damit auch nicht den Durchsatz des Caches.

Grid-übergreifende Transaktionen haben einen ähnlichen Effekt auf die Verfügbarkeit. Wenn Sie mit 100 Servern arbeiten und ein Server ausfällt, werden 100 Prozent der Transaktionen, die zum Zeitpunkt des Serverausfalls in Bearbeitung sind, rückgängig gemacht. Nach dem Ausfall des Servers beginnt WebSphere eXtreme Scale mit der Verlagerung der Partitionen des ausgefallenen Servers auf die anderen 99 Computer. In dieser Zeit, d. h. bis zum Abschluss des Failover-Prozesses, kann das Grid keine dieser Transaktionen verarbeiten. Nach Abschluss des Failover-Prozesses ist der Cache wieder betriebsbereit, aber mit verringerter Kapazität. Wenn jeder Computer im Grid 10 Partitionen bereitstellt, erhalten 10 der verbleibenden 99 Computer während des Failover-Prozesses mindestens eine zusätzliche Partition. Eine zusätzliche Partition erhöht die Arbeitslast dieses Computers um mindestens 10 Prozent. Da der Durchsatz des Grids in einer Grid-übergreifenden Transaktion auf den Durchsatz des langsamsten Computers beschränkt ist, reduziert sich der Durchsatz durchschnittlich um 10 Prozent.

Einzelpartitionstransaktionen sind im Hinblick auf die horizontale Skalierung mit einem verteilten, hoch verfügbaren Objektcache wie WebSphere eXtreme Scale den Grid-übergreifenden Transaktionen vorzuziehen. Die Maximierung der Leistung solcher Systemtypen erfordert die Verwendung von Techniken, die sich von den traditionellen relationalen Verfahren unterscheiden, aber Sie Grid-übergreifende Transaktionen in skalierbare Einzelpartitionstransaktionen konvertieren.

Bewährte Verfahren für die Erstellung skalierbarer Datenmodelle

Die bewährten Verfahren für die Erstellung skalierbarer Anwendungen mit Produkten wie WebSphere eXtreme Scale sind in zwei Kategorien einteilbar: Grundsätze und Implementierungstipps. Grundsätze sind Kernideen, die im Design der Daten selbst erfasst werden müssen. Es ist sehr unwahrscheinlich, dass sich eine Anwendung, die diese Grundsätze nicht einhält, problemlos skalieren lässt, selbst für ihre Haupttransaktionen. Implementierungstipps werden für problematische Transaktionen in einer ansonsten gut entworfenen Anwendung angewendet, die sich an die allgemeinen Grundsätze für skalierbare Datenmodelle hält.

Grundsätze

Einige wichtige Hilfsmittel für die Optimierung der Skalierbarkeit sind Basiskonzepte oder Grundsätze, die beachtet werden müssen.

Duplizieren an Stelle von Normalisieren

Der wichtigste Punkt, der bei Produkten wie WebSphere eXtreme Scale zu beachten ist, ist der, dass sie für die Verteilung von Daten auf sehr viele

Computer konzipiert sind. Wenn das Ziel darin besteht, die meisten oder sogar alle Transaktionen auf einer einzelnen Partition auszuführen, muss das Datenmodelldesign sicherstellen, dass sich alle Daten, die die Transaktion unter Umständen benötigt, auf der Partition befinden. In den meisten Fällen kann dies nur durch Duplizierung der Daten erreicht werden.

Stellen Sie sich beispielsweise eine Anwendung wie ein Nachrichtenbrett vor. Zwei sehr wichtige Transaktionen für ein Nachrichtenbrett zeigen alle Veröffentlichungen eines bestimmten Benutzers und alle Veröffentlichungen unter einem bestimmten Topic an. Stellen Sie sich zunächst vor, wie diese Transaktionen mit einem normalisierten Datenmodell arbeiten, das einen Benutzerdatensatz, einen Topic-Datensatz und einen Veröffentlichungsdatensatz mit dem eigentlichen Text enthält. Wenn Veröffentlichungen mit Benutzerdatensätzen partitioniert werden, wird aus der Anzeige des Topics eine Grid-übergreifende Transaktion und umgekehrt. Topics und Benutzer können nicht gemeinsam partitioniert werden, da sie eine Viele-zu-viele-Beziehung haben.

Die beste Methode für die Skalierung dieses Nachrichtenbretts ist die Duplizierung der Veröffentlichungen, wobei eine Kopie mit dem Topic-Datensatz und eine Kopie mit dem Benutzerdatensatz gespeichert wird. Die anschließende Anzeige der Veröffentlichungen eines Benutzers ist eine Einzelpartitionstransaktion, die Anzeige der Veröffentlichungen unter einem Topic ist eine Einzelpartitionstransaktion, und die Aktualisierung oder das Löschen einer Veröffentlichung ist eine Transaktion, an der zwei Partitionen beteiligt sind. Alle drei Transaktionen können linear skaliert werden, wenn die Anzahl der Computer im Grid zunimmt.

Skalierbarkeit an Stelle von Ressourcen

Die größte Hindernis, das beim Einsatz denormalisierter Datenmodell überwunden werden muss, sind die Auswirkungen, die diese Modell auf Ressourcen haben. Die Verwaltung von zwei, drei oder mehr Kopien derselben Daten kann den Anschein erwecken, dass zu viele Ressourcen benötigt werden, als dass dieser Ansatz praktikabel ist. Wenn Sie mit diesem Szenario konfrontiert werden, berücksichtigen Sie die folgenden Fakten: Hardwareressourcen werden von Jahr zu Jahr billiger. Zweitens, und noch wichtiger, mit WebSphere eXtreme Scale fallen die meisten verborgenen Kosten weg, die bei der Implementierung weiterer Ressourcen anfallen.

Messen Sie Ressourcen anhand der Kosten und nicht anhand von Computerbegriffen wie Megabyte oder Prozessoren. Datenspeicher, die mit normalisierten relationalen Daten abreiten, müssen sich im Allgemeinen auf demselben Computer befinden. Diese erforderliche Co-Location bedeutet, dass ein einzelner größerer Unternehmenscomputer an Stelle mehrerer kleinerer Computer erworben werden muss. Bei Unternehmenshardware ist es nicht unüblich, dass ein einziger Computer, der in der Lage ist, eine Million Transaktionen pro Sekunde zu verarbeiten, mehr kostet als 10 Computer zusammen, die in der Lage sind, jeweils 100.000 Transaktionen pro Sekunden auszuführen.

Außerdem fallen Geschäftskosten für die Implementierung der Ressourcen an. Irgendwann reicht die Kapazität in einem expandierenden Unternehmen einfach nicht mehr aus. In diesem Fall setzen Sie den Betrieb entweder aus, während Sie die Umstellung auf einen größeren und schnelleren Computer durchführen, oder Sie erstellen eine zweite Produktionsumgebung, auf die Sie den Betrieb dann umstellen können. In beiden Fällen fallen zusätzliche Kosten durch das ausgefallene Geschäft oder durch die Verwaltung der doppelten Kapazität in der Übergangsphase an.

Mit WebSphere eXtreme Scale muss die Anwendung nicht heruntergefahren werden, um Kapazität hinzuzufügen. Wenn die Prognose für Ihr Geschäft lautet, dass Sie 10 Prozent mehr Kapazität für das kommende Jahr benötigen, erhöhen Sie die Anzahl der Computer im Grid um 10 Prozent. Sie können diese Erweiterung ohne Anwendungsausfallzeit und ohne den Einkauf von Kapazitäten durchführen, die Sie hinterher nicht mehr benötigen.

Datenkonvertierungen vermeiden

Wenn Sie WebSphere eXtreme Scale verwenden, müssen Daten in einem Format gespeichert werden, das von der Geschäftslogik direkt konsumiert werden kann. Die Aufteilung der Daten in ein primitiveres Format ist kostenintensiv. Die Konvertierung muss durchgeführt werden, wenn die Daten geschrieben und wenn die Daten gelesen werden. Mit relationalen Datenbanken ist diese Konvertierung unumgänglich, weil die Daten letztendlich relativ häufig auf der Platte gespeichert werden, aber mit WebSphere eXtreme Scale fallen diese Konvertierungen weg. Der größte Teil der Daten wird im Hauptspeicher gespeichert und kann deshalb in genau dem Format gespeichert werden, das die Anwendung erfordert.

Durch die Einhaltung dieser einfachen Regel können Sie Ihre Daten dem ersten Grundsatz entsprechend denormalisieren. Der gängigste Konvertierungstyp für Geschäftsdaten ist die JOIN-Operation, die erforderlich ist, um normalisierte Daten in eine Ergebnismenge zu konvertieren, die den Anforderungen der Anwendung entspricht. Durch die implizite Speicherung der Daten im richtigen Format werden diese JOIN-Operationen vermieden, und es entsteht ein denormalisiertes Datenmodell.

Unbegrenzte Abfragen vermeiden

Unbegrenzte Abfragen lassen sich nicht gut skalieren, egal, wie gut Sie Ihre Daten auch strukturieren. Verwenden Sie beispielsweise keine Transaktion, die eine Liste aller Einträge nach Wert sortiert abfragt. Diese Transaktion funktioniert möglicherweise, wenn die Gesamtanzahl der Einträge bei 1000 liegt, aber wenn die die Gesamtanzahl der Einträge 10 Million erreicht, gibt die Transaktion alle 10 Millionen Einträge zurück. Wenn Sie diese Transaktion ausführen, sind zwei Ergebnisse am wahrscheinlichsten: Die Transaktion überschreitet das zulässige Zeitlimit, oder im Client tritt eine abnormale Speicherbedingung auf.

Die beste Option ist, die Geschäftslogik so zu ändern, dass nur die Top 10 oder 20 Einträge zurückgegeben werden können. Durch diese Änderung der Logik bleibt die Größe der Transaktion verwaltbar, unabhängig davon, wie viele Einträge im Cache enthalten sind.

Schema definieren

Der Hauptvorteil der Normalisierung von Daten ist der, dass sich das Datenbanksystem im Hintergrund um die Datenkonsistenz kümmern kann. Wenn Daten für Skalierbarkeit denormalisiert werden, ist diese automatische Verwaltung der Datenkonsistenz nicht mehr möglich. Sie müssen ein Datenmodell implementieren, das auf der Anwendungsebene oder als Plug-in für das verteilte Grid arbeiten kann, um die Datenkonsistenz zu gewährleisten.

Stellen Sie sich das Beispiel mit dem Nachrichtenbrett vor. Wenn eine Transaktion eine Veröffentlichung aus einem Topic entfernt, muss das Veröffentlichungsduplikat im Benutzerdatensatz entfernt werden. Ohne ein Datenmodell ist es möglich, dass ein Entwickler den Anwendungscode

zum Entfernen der Veröffentlichung aus dem Topic schreibt und vergisst, die Veröffentlichung aus dem Benutzerdatensatz zu entfernen. Wenn der Entwickler jedoch ein Datenmodell verwendet, anstatt direkt mit dem Cache zu interagieren, kann die Methode "removePost" im Datenmodell die Benutzer-ID aus der Veröffentlichung extrahieren, den Benutzerdatensatz suchen und das Veröffentlichungsduplikat im Hintergrund entfernen.

Alternativ können Sie einen Listener implementieren, der auf der tatsächlichen Partition ausgeführt wird, das Topic überwacht und bei einer Änderung des Topics Benutzerdatensatz automatisch anpasst. Ein Listener kann von Vorteil sein, weil die Anpassung am Benutzerdatensatz lokal vorgenommen werden kann, wenn die Partition den Benutzerdatensatz enthält. Selbst wenn sich der Benutzerdatensatz auf einer anderen Partition befindet, findet die Transaktion zwischen Servern und nicht zwischen dem Client und dem Server statt. Die Netzverbindung zwischen Servern ist wahrscheinlich schneller als die Netzverbindung zwischen dem Client und dem Server.

Konkurrenzsituationen vermeiden

Szenarien wie die Verwendung eines globalen Zählers vermeiden. Das Grid kann nicht skaliert werden, wenn ein einzelner Datensatz im Vergleich mit den restlichen Datensätzen unverhältnismäßig oft verwendet wird. Die Leistung des Grids wird durch die Leistung des Computers beschränkt, der diesen Datensatz enthält.

Versuchen Sie in solchen Situationen, den Datensatz aufzuteilen, so dass er pro Partition verwaltet wird. Stellen Sie sich beispielsweise eine Transaktion vor, die die Gesamtanzahl der Einträge im verteilten Cache zurückgibt. Anstatt jede Einfüge- und Entfernungsoperation auf einen einzelnen Datensatz zugreifen zu lassen, dessen Zähler sich erhöht, können Sie einen Listener auf jeder Partition einsetzen, der die Einfüge- und Entfernungsoperation verfolgt. Mit dieser Listener-Verfolgung können aus Einfüge- und Entfernungsoperationen Einzelpartitionsoperationen werden.

Das Lesen des Zählers wird zu einer Grid-übergreifenden Operation, aber die Leseoperation war bereits vorher genauso ineffizient wie eine Grid-übergreifende Operation, weil ihre Leistung an die Leistung des Computers gebunden war, auf dem sich der Datensatz befindet.

Implementierungstipps

Zum Erreichen der besten Skalierbarkeit können Sie außerdem die folgenden Tipps beachten.

Umgekehrte Suchindizes verwenden

Stellen Sie sich ein ordnungsgemäß denormalisiertes Datenmodell vor, in dem Kundendatensätze auf der Basis der Kunden-ID partitioniert werden. Diese Partitionierungsmethode ist die logische Option, weil nahezu jede Geschäftsoperation, die mit dem Kundendatensatz ausgeführt wird, die Kunden-ID verwendet. Eine wichtige Transaktion, in der die Kunden-ID jedoch nicht verwendet wird, ist die Anmeldetransaktion. Es ist üblich, dass Benutzernamen oder E-Mail-Adressen für die Anmeldung verwendet werden, und keine Kunden-IDs.

Der einfache Ansatz für das Anmeldeszenario ist die Verwendung einer Grid-übergreifenden Transaktion, um den Kundendatensatz zu suchen. Wie zuvor erläutert, ist dieser Ansatz nicht skalierbar.

Die nächste Option ist die Partitionierung nach Benutzernamen oder E-Mail-Adressen. Diese Option ist nicht praktikabel, da alle Operationen, die auf der Kunden-ID basieren, zu Grid-übergreifenden Transaktionen werden. Außerdem möchten die Kunden auf Ihrer Site möglicherweise ihren Benutzernamen oder ihre E-Mail-Adresse ändern. Produkte wie WebSphere eXtreme Scale benötigen den Wert, der für die Partitionierung der Daten verwendet wird, um konstant zu bleiben.

Die richtige Lösung ist die Verwendung eines umgekehrten Suchindex. Mit WebSphere eXtreme Scale kann ein Cache in demselben verteilten Grid wie der Cache erstellt werden, der alle Benutzerdatensätze enthält. Dieser Cache ist hoch verfügbar, partitioniert und skalierbar. Dieser Cache kann verwendet werden, um einen Benutzernamen oder eine E-Mail-Adresse einer Kunden-ID zuzuordnen. Dieser Cache verwandelt die Anmeldung in eine Operation, an der zwei Partitionen beteiligt sind, und nicht in eine Grid-übergreifende Transaktion. Dieses Szenario ist zwar nicht so effektiv wie eine Einzelpartitionstransaktion, aber der Durchsatz nimmt linear mit steigender Anzahl an Computern zu.

Berechnung beim Schreiben

Die Generierung häufig berechneter Werte wie Durchschnittswerte oder Summen kann kostenintensiv sein, weil bei diesen Operationen gewöhnlich sehr viele Einträge gelesen werden müssen. Da in den meisten Anwendungen mehr Leseoperationen als Schreiboperationen ausgeführt werden, ist es effizient, diese Werte beim Schreiben zu berechnen und das Ergebnis anschließend im Cache zu speichern. Durch dieses Verfahren werden Leseoperationen schneller und skalierbarer.

Optionale Felder

Stellen Sie sich einen Benutzerdatensatz vor, der eine geschäftliche Telefonnummer, eine private Telefonnummer und eine Handy-Nummer enthält. Ein Benutzer kann alle, keine oder eine beliebige Kombination dieser Nummern haben. Wenn die Daten normalisiert sind, sind eine Benutzertabelle und eine Telefonnummerntabelle vorhanden. Die Telefonnummern für einen bestimmten Benutzer können über eine JOIN-Operation zwischen den beiden Tabellen ermittelt werden.

Die Denormalisierung dieses Datensatzes erfordert keine Datenduplizierung, weil die meisten Benutzer nicht dieselben Telefonnummern haben. Stattdessen müssen freie Bereiche im Benutzerdatensatz zulässig sein. Anstatt eine Telefonnummerntabelle zu verwenden, können Sie jedem Benutzerdatensatz drei Attribute hinzufügen, eines für jeden Telefonnummerntyp. Durch das Hinzufügen dieser Attribut wird die JOIN-Operation vermieden, und die Suche der Telefonnummern für einen Benutzer wird zu einer Einzelpartitionsoperation.

Verteilung von Viele-zu-viele-Beziehungen

Stellen Sie sich eine Anwendung, die Produkte und die Länden verfolgt, in denen die Produkte verkauft werden. Ein Produkt wird in vielen Läden verkauft, und ein Laden verkauft viele Produkte. Angenommen, diese Anwendung verfolgt 50 große Einzelhändler. Jedes Produkt wird in maximal 50 Länden verkauft, wobei jeder Laden Tausende von Produkten verkauft.

Verwalten Sie eine Liste der Läden in der Produktentität (Anordnung A), anstatt eine Liste von Produkten in jeder Ladenentität zu verwalten (An-

ordnung B). Wenn Sie sich einige der Transaktionen ansehen, die diese Anwendung ausführen muss, ist leicht zu erkennen, warum Anordnung A skalierbarer ist.

Sehen Sie sich zuerst die Aktualisierungen an. Wenn bei Anordnung A ein Produkt aus dem Bestand eines Ladens entfernt wird, wird die Produktentität gesperrt. Enthält das Grid 10000 Produkte, muss nur 1/10000 des Grids gesperrt werden, um die Aktualisierung durchzuführen. Bei Anordnung B enthält das Grid nur 50 Länden, so dass 1/50 des Grids gesperrt werden muss, um die Aktualisierung durchzuführen. Obwohl beide Fälle als Einzelpartitionsoperationen eingestuft werden können, lässt sich Anordnung A effizienter skalieren.

Sehen Sie sich jetzt die Leseoperationen für Anordnung A an. Das Durchsuchen eines Ladens, in dem ein Produkt verkauft wird, ist eine Einzelpartitionsoperation, die skalierbar und schnell ist, weil die Transaktion nur einen kleine Datenmenge überträgt. Bei Anordnung B wird aus dieser Transaktion eine Grid-übergreifende Transaktion, weil auf jede Ladenentität zugegriffen werden muss, um festzustellen, ob das Produkt in diesem Laden verkauft wird. Daraus ergibt sich ein enormer Leistungsvorteil für Anordnung A.

Skalierung mit normalisierten Daten

Eine zulässige Verwendung von Grid-übergreifenden Transaktionen ist die Skalierung der Datenverarbeitung. Wenn ein Grid 5 Computer enthält und eine Grid-übergreifende Transaktion zugeteilt wird, die 100.000 Datensätze auf jedem Computer durchsucht, durchsucht diese Transaktion insgesamt 500.000 Datensätze. Wenn der langsamste Computer im Grid 10 dieser Transaktionen pro Sekunde ausführen kann, ist das Grid in der Lage, 5.000.000 Datensätze pro Sekunde zu durchsuchen. Wenn sich die Daten im Grid verdoppeln, muss jeder Computer 200.000 Datensätze durchsuchen, und jede Transaktion durchsucht insgesamt 1.000.000 Datensätze. Diese Datenzunahme verringert den Durchsatz des langsamsten Computers auf 5 Transaktionen pro Sekunde und damit den Durchsatz des Grids auf 5 Transaktionen pro Sekunde. Das Grid durchsucht weiterhin 5.000.000 Datensätze pro Sekunde.

In diesem Szenario kann jeder Computer durch die Verdopplung der Computeranzahl zu seiner vorherigen Last von 100.000 Datensätzen zurückkehren, und der langsamste Computer kann wieder 10 dieser Transaktionen pro Sekunde verarbeiten. Der Durchsatz des Grids bleibt bei 10 Anforderungen pro Sekunde, aber jetzt verarbeitet jede Transaktion 1.000.000 Datensätze, so dass das Grid seine Kapazität in Bezug auf die Verarbeitung von Datensätzen auf 10.000.000 pro Sekunde verdoppelt hat.

Für Anwendungen wie Suchmaschinen, die sowohl in Bezug auf die Datenverarbeitung (angesichts der zunehmenden Größe des Internets) als auch in Bezug auf den Durchsatz (angesichts der zunehmenden Anzahl an Benutzern) skalierbar sein müssen, müssen Sie mehrere Grids mit einem Umlaufverfahren für die Anforderungen zwischen den Grids erstellen. Wenn Sie den Durchsatz erhöhen müssen, fügen Sie Computer und ein weiteres Grid für die Bearbeitung der Anforderungen hinzu. Wenn die Datenverarbeitung erhöht werden muss, fügen Sie weitere Computer hinzu, und halten Sie die Anzahl der Grids konstant.

Skalierung in Einheiten oder Pods

In diesem Abschnitt wird das Thema Skalierbarkeit behandelt, aber nicht auf die herkömmliche Weise wie beim Skalieren des Grids auf X JVMs. Vielmehr wird die Skalierbarkeit hier aus der Perspektive von Operationen, Planung und Risikomanagement beschrieben. Obwohl diese Faktoren gewöhnlich wichtiger sind als die traditionellen Überlegungen bei der Produktskalierbarkeit, werden sie häufig ignoriert. Beim Aufbau eines hoch verfügbaren Systems für die Implementierung von eXtreme Scale in einem zuverlässigen Implementierungsprozess müssen beide Typen Sklalierungsüberlegungen berücksichtigt werden.

Implementierungen eines einzigen großen Grids

Tests haben ergeben, dass eXtreme Scale auf über 1000 JVMs skaliert werden kann. Diese Tests animieren zum Erstellen von Anwendungen für die Implementierung eines einzigen Grids auf sehr vielen Maschinen. Obwohl diese Vorgehensweise möglich ist, wird sie aus verschiedenen Gründen, die im Folgenden beschrieben werden, nicht empfohlen.

1. **Budgetbedenken:** Realistisch betrachtet ist Ihre Umgebung nicht in der Lage, ein Grid mit 1000 Servern zu testen. Tests mit einem sehr viel kleineren Grid sind unter Berücksichtigung von Budgetgründen jedoch möglich, so dass Hardware nicht doppelt gekauft werden muss, insbesondere bei einer solch hohen Anzahl an Servern.
2. **Verschiedene Anwendungsversionen:** Für jeden einzelnen Test sehr viele Maschinen zu benötigen, ist nicht sehr praktisch. Es besteht das Risiko, dass nicht dieselben Faktoren getestet werden, wie es in einer Produktionsumgebung der Fall ist.
3. **Datenverlust:** Die Ausführung einer Datenbank auf einem einzigen Festplattenlaufwerk ist unzuverlässig. Jedes Problem mit dem Festplattenlaufwerk führt zum Verlust von Daten. Die Ausführung einer ausbaufähigen Anwendung in einem einzigen Grid ist ähnlich. Es ist wahrscheinlich, dass Sie Programmfehler in Ihrer Umgebung und in Ihren Anwendungen haben. Die Speicherung aller Daten auf einem einzigen großen System führt deshalb häufig zum Verlust großer Datenmengen.

Aufteilung des Grids

Die Aufteilung der Anwendung in Pods (Einheiten) ist zuverlässiger. Ein Pod ist eine Gruppe von Servern, auf denen ein homogener Anwendungs-Stack ausgeführt wird. Pods können beliebig groß sein, sollten sich idealerweise aber aus ca. 20 Maschinen zusammensetzen. Anstatt 500 Maschinen in einem einzigen Grid zu verwenden, können Sie 25 Pods mit jeweils 20 Maschinen verwenden. Es sollte jeweils nur eine einzige Version eines Anwendungs-Stacks in einem bestimmten Pod ausgeführt werden, aber die verschiedenen Pods können jeweils eigene Versionen eines Anwendungs-Stacks haben.

Im Allgemeinen werden in einem Anwendungs-Stack die Versionsstände der folgenden Komponenten berücksichtigt:

- Betriebssystem
- Hardware
- JVM
- Version von eXtreme Scale
- Anwendung
- Weitere erforderliche Komponenten

Ein Pod ist eine Verteilungseinheit mit einer für Tests geeigneten Größe. Anstatt Hunderte von Servern für Tests zu verwenden, empfiehlt es sich, nur 20 Server zu verwenden. In diesem Fall testen Sie dieselbe Konfiguration wie in einer Produktionsumgebung. In Produktionsumgebungen werden Grids mit einer Größe von maximal 20 Servern, die einen Pod bilden, verwendet. Sie können Belastungstests für einen einzigen Pod ausführen und dabei die Kapazität, die Anzahl der Benutzer, das Datenvolumen und den Transaktionsdurchsatz dieses Pods bestimmen. Dies vereinfacht die Planung und entspricht dem Standard einer vorhersehbaren Skalierung zu vorhersehbaren Kosten.

Einrichtung einer Pod-basierten Umgebung

In manchen Fällen muss der Pod nicht unbedingt 20 Server haben. Die Pod-Größe dient ausschließlich dem Zweck praxistauglicher Tests. Ein Pod sollte klein genug sein, dass der Teil der beim Auftreten von Pod-Problemen in einer Produktionsumgebung betroffenen Transaktionen tolerierbar bleibt.

Im Idealfall wirkt sich ein Programmfehler nur auf einen einzigen Pod aus. Im vorherigen Beispiel würde sich ein Programmfehler nur auf vier Prozent der Anwendungstransaktionen und nicht auf 100 Prozent auswirken. Außerdem sind Upgrades einfacher, weil sie nacheinander in jeweils einem Pod implementiert werden können. Dies vereinfacht das Szenario, so dass der Benutzer beim Auftreten eines Upgradeproblems in einem Pod auf die vorherige Version für diesen Pod zurückgreifen kann. Upgrades umfassen alle Änderungen, die an der Anwendung bzw. am Anwendungs-Stack vorgenommen werden, sowie Systemaktualisierungen. Bei Upgrades sollte möglichst jeweils nur ein einziges Element des Stacks aktualisiert werden, um die Problemdiagnose zu vereinfachen.

Zum Implementieren einer Umgebung mit Pods benötigen Sie eine Routing-Schicht oberhalb der Pods, die auf- und abwärtskompatibel ist, wenn Software-Upgrades auf Pods angewendet werden. Außerdem sollten Sie ein Verzeichnis erstellen, das Informationen dazu enthält, welcher Pod welche Daten enthält. (Hierfür können Sie ein weiteres datenbankgestütztes eXtreme-Scale-Grid verwenden, vorzugsweise mit Verwendung des Write-behind-Szenarios.) Dies ergibt eine 2-Schichten-Lösung. Schicht 1 ist das Verzeichnis und wird verwendet, um den Pod zu ermitteln, der eine bestimmte Transaktion bearbeitet. Schicht 2 setzt sich aus den Pods selbst zusammen. Wenn Schicht 1 einen Pod identifiziert, leitet das Setup jede Transaktion an den richtigen Server im Pod weiter, der gewöhnlich der Server ist, der die Partition für die von der Transaktion verwendeten Daten enthält. Optional können Sie einen nahen Cache auf Schicht 1 verwenden, um die Auswirkungen zu mindern, die mit der Ermittlung des richtigen Pods verbunden sind.

Die Verwendung von Pods ist geringfügig komplexer als die Verwendung eines einzigen Grids, aber aufgrund der Verbesserungen in Bezug auf den Betrieb, die Tests und die Zuverlässigkeit sind Pods ein entscheidender Faktor bei Skalierbarkeitstests.

Kapitel 5. Übersicht über Verfügbarkeit

Hohe Verfügbarkeit

Mit hoher Verfügbarkeit unterstützt WebSphere eXtreme Scale zuverlässige Datenredundanz und Fehlererkennung.

WebSphere eXtreme Scale organisiert JVM-Grids eigenständig in einem losen baumähnlichen Verbund mit dem Katalogservice als Stamm und den Stammgruppen, die Container enthalten, als Blättern. Weitere Informationen finden Sie im Abschnitt „Caching-Architektur: Maps, Container, Clients und Kataloge" auf Seite 11.

Jede Stammgruppe wird vom Katalogservice automatisch in Gruppen von ungefähr 20 Servern unterteilt. Die Stammgruppen-Member überwachen die anderen Member der Gruppe auf ihre Vitalität. Außerdem wählt jede Stammgruppe ein Member als führendes Member aus, das für die Kommunikation der Gruppeninformation an den Katalogservice zuständig ist. Eine Begrenzung der Stammgruppengröße sorgt für eine effektive Vitalitätsüberwachung und eine hoch skalierbare Umgebung.

Anmerkung: In einer Umgebung mit WebSphere Application Server, in der die Stammgruppengröße geändert werden kann, unterstützt eXtreme Scale maximal 50 Member pro Stammgruppe.

Fehler

Ein Prozess kann auf verschiedene Arten fehlschlagen. Ein Prozess kann fehlschlagen, weil eine Ressourcengrenze erreicht wurde, wie z. B. die maximale Größe des Heap-Speichers, oder weil eine Prozesssteuerungslogik den Prozess beendet hat. Das Betriebssystem kann ausfallen, was dazu führt, dass alle auf dem System ausgeführten Prozesse verloren gehen. Die Hardware, wie z. B. die Netzschnittstellenkarte, kann (wenn auch weniger häufig) ausfallen, was zur Trennung des Betriebssystems vom Netz führen kann. Es gibt viele weitere Fehlerquellen, die die Nichtverfügbarkeit des Prozesses zur Folge haben können. In diesem Kontext können all diese Fehler in zwei Kategorien eingeteilt werden: Prozessfehler und Konnektivitätsverlust.

Prozessfehler

WebSphere eXtreme Scale reagiert sehr schnell auf Prozessfehler. Wenn ein Prozess fehlschlägt, ist das Betriebssystem für die Bereinigung aller übrig gebliebenen Ressourcen verantwortlich, die vom Prozess verwendet wurden. Diese Bereinigung umfasst die Portzuordnung und die Konnektivität. Wenn ein Prozess fehlschlägt, wird ein Signal über die von diesem Prozess verwendeten Verbindungen gesendet, um jede einzelne Verbindung zu schließen. Mit Hilfe dieser Signale kann ein Prozessfehler in kürzester Zeit von einem anderen Prozess, der mit dem fehlgeschlagenen verbunden ist, erkannt werden.

Konnektivitätsverlust

Ein Konnektivitätsverlust tritt auf, wenn die Verbindung des Betriebssystems zum Netz getrennt wird. Infolgedessen kann das Betriebssystem keine Signale an andere

Prozesse senden. Es gibt mehrere Gründe für einen Konnektivitätsverlust, die jedoch in zwei Kategorien eingeteilt werden können: Hostausfall und Isolierung.

Hostausfall

Wenn der Netzstecker der Maschine gezogen wird, geht die Konnektivität sofort verloren.

Isolierung

Dieses Szenario stellt die komplizierteste Fehlerbedingung für Software dar. Die Behebung einer solchen Fehlerbedingung stellt sich so schwierig dar, weil davon ausgegangen wird, dass der Prozess nicht verfügbar ist, obwohl dies gar nicht der Fall ist. Für das System scheint ein Server oder ein anderer Prozess ausgefallen zu sein, obwohl er in Wirklichkeit ordnungsgemäß ausgeführt wird.

Ausfall der eXtreme-Scale-Container

Containerausfälle werden im Allgemeinen von Peer-Containern über den Stammgruppenmechanismus erkannt. Wenn ein Container oder eine Gruppe von Containern ausfällt, migriert der Katalogservice die Shards aus den betroffenen Containern. Der Katalogservice sucht zuerst nach einem synchronen Replikat, bevor er die Migration auf ein asynchrones Replikat durchführt. Nach der Migration der primären Shards in die neuen Hostcontainer durchsucht der Katalogservice die neuen Hostcontainer nach den Replikaten, die jetzt fehlen.

Anmerkung: Containerisolierung - Der Katalogservice migriert Shards aus Containern, wenn der Container als nicht verfügbar erkannt wird. Wenn diese Container wieder verfügbar werden, berücksichtigt der Katalogservice sie bei der Verteilung wie beim normalen Startablauf.

Latenzzeit für Erkennung von Containerausfällen

Ausfälle können in die folgenden beiden Kategorien eingeteilt werden: temporäre Ausfälle und permanente Ausfälle. Temporäre Ausfälle werden gewöhnlich durch einen Prozessfehler verursacht. Solche Ausfälle werden vom Betriebssystem erkannt, das genutzte Ressourcen, wie z. B. Netz-Sockets, sehr schnell wiederherstellen kann. Gewöhnlich werden temporäre Ausfälle in weniger als einer Sekunde erkannt. Die Erkennung permanenter Ausfälle kann unter Verwendung der Standardoptimierung für Überwachungssignale bis zu 200 Sekunden dauern. Zu solchen Ausfällen gehören physische Ausfälle von Maschinen, das Ziehen von Netzkabeln und Betriebssystemausfälle. Somit muss eXtreme Scale darauf vertrauen, dass permanente Ausfälle durch den Austausch von Überwachungssignalen erkannt werden, der konfiguriert werden kann. Im Abschnitt „Fehlererkennungstypen" auf Seite 119 können Sie nachlesen, wie Sie die Zeit für die Erkennung von permanenten Ausfällen verringern können.

Ausfall des Katalogservice

Da das Katalogservice-Grid ein eXtreme-Scale-Grid ist, wird der Stammgruppenmechanismus auch hier auf dieselbe Weise wie beim Containerausfallprozess verwendet. Der Hauptunterschied besteht darin, dass die Katalogservicedomäne einen Peer-Auswahlprozess für die Definition des primären Shards an Stelle des Katalogservicealgorithmus verwendet, der für die Container verwendet wird.

Beachten Sie, dass der Verteilungsservice und der Stammgruppierungsservice Services vom Typ "1 von N" sind, der Positionsservice und die Verwaltung hingegen überall ausgeführt werden. Ein Service vom Typ "1 von N" wird nur in einem einzigen Member der HA-Gruppe ausgeführt. Der Positionsservice und der Verwaltungsservice werden in allen Membern der HA-Gruppe ausgeführt. Der Verteilungsservice und der Stammgruppierungsservice sind Singletons, weil sie für das Layout des Systems verantwortlich sind. Der Positionsservice und die Verwaltung sind Services, die ausschließlich im Lesezugriff arbeiten und zur Unterstützung der Skalierbarkeit überall vorhanden sind.

Der Katalogservice verwendet die Replikation für seine eigene Fehlertoleranz. Wenn ein Katalogserviceprozess fehlschlägt, muss der Service erneut gestartet werden, um das System in einem Zustand wiederherzustellen, der die gewünschte Stufe der Verfügbarkeit bietet. Schlagen alle Prozesse, in denen der Katalogservice ausgeführt wird, fehl, verliert eXtreme Scale kritische Daten. Nach diesem Fehler müssen alle Container erneut gestartet werden. Da der Katalogservice in vielen Prozessen ausgeführt werden kann, ist das Auftreten dieses Fehlers eher unwahrscheinlich. Wenn Sie jedoch alle Prozesse auf einer einzigen Maschine, in einem einzigen Blade-Gehäuse oder über einen einzigen Netz-Switch ausführen, ist die Wahrscheinlich eines Fehlers hoch. Versuchen Sie, bekannte Fehlermodi auf Maschinen, auf denen der Katalogservice ausgeführt wird, zu beseitigen, um die Fehlerwahrscheinlichkeit zu reduzieren.

Mehrere Containerausfälle

Ein Replikat wird niemals in demselben Prozess wie das primäre Shard ausgeführt, da dies beim Verlust des Prozesses zu einem Verlust des primären Shards und des Replikats führen würde. Die Implementierungsrichtlinie definiert ein boolesches Attribut für den Entwicklungsmodus, das der Katalogservice verwendet, um festzustellen, ob ein Replikat auf derselben Maschine wie das primäre Shard platziert werden kann. In einer Entwicklungsumgebung auf einer einzigen Maschine können Sie zwei Container verwenden und die Daten des einen Containers im jeweils anderen replizieren. In Produktionsumgebungen ist die Verwendung einer einzigen Maschine unzureichend, weil der Verlust dieses Hosts zum Verlust beider Container führt. Zum Wechsel vom Entwicklungsmodus auf einer einzigen Maschine in den Produktionsmodus mit mehreren Maschinen müssen Sie den Entwicklungsmodus in der Konfigurationsdatei der Implementierungsrichtlinie inaktivieren.

Replikation für Verfügbarkeit

Die Replikation unterstützt Fehlertoleranz und erhöht die Leitung für eine verteilte eXtreme-Scale-Topologie.

Die Replikation wird aktiviert, indem einem MapSet BackingMaps zugeordnet werden.

Ein MapSet ist eine Sammlung von Maps, die nach Partitionsschlüssel klassifiziert sind. Dieser Partitionsschlüssel wird wie folgt aus dem Schlüssel der jeweiligen Map abgeleitet: Hash-Wert Modulo Partitionsanzahl. Wenn eine Map-Gruppe im MapSet also den Partitionsschlüssel X hat, werden diese Maps in einer entsprechenden Partition X im Grid gespeichert. Wenn eine andere Gruppe den Partitionsschlüssel Y hat, werden alle Maps in Partition Y gespeichert usw. Außerdem werden die Daten in den Maps auf der Basis der Richtlinie repliziert, die im MapSet definiert ist. Dies wird nur für verteilte eXtreme-Scale-Topologien verwendet und ist für lokale Instanzen nicht erforderlich.

Weitere Einzelheiten finden Sie im Abschnitt „Partitionierung” auf Seite 95.

Den MapSets wird die Anzahl vorhandener Partitionen und eine Replikationsrichtlinie zugeordnet. In der Replikationskonfiguration des MapSets wird einfach die Anzahl synchroner und asynchroner Replikat-Shards angegeben, die ein MapSet zusätzlich zum primären Shard haben sollte. Wenn beispielsweise ein synchrones und ein asynchrones Replikat verwendet werden sollen, wird für alle BackingMaps, die dem MapSet zugeordnet werden, automatisch jeweils ein Replikat-Shard auf die Gruppe verfügbarer Container für eXtreme Scale verteilt. Die Replikationskonfiguration kann Clients auch das Lesen von Daten aus synchron replizierten Servern ermöglichen. Auf diese Weise kann die Last der Leseanforderungen auf zusätzliche Server in eXtreme Scale verteilt werden. Die Replikation hat nur beim vorherigen Laden der BackingMaps Auswirkungen auf das Programmiermodell.

Einzelheiten zu den verschiedenen Konfigurationsoptionen finden Sie im Folgenden:

Vorheriges Laden von Maps

Maps können so genannte Loader (Ladeprogramme) zugeordnet werden. Ein Loader wird verwendet, um Objekte abzurufen, wenn diese nicht in der Map gefunden werden (Cachefehler), und um Änderungen in ein Back-End zu schreiben, wenn eine Transaktion festgeschrieben wird. Loader können auch für das vorherige Laden von Daten (Preload) in eine Map verwendet werden. Die Methode "preloadMap" der Schnittstelle "Loader" wird für jede Map aufgerufen, wenn die zugehörige Partition im MapSet zu einem primären Shard wird. Die Methode "preloadMap" wird nicht für Replikate aufgerufen. Sie versucht, alle geplanten referenzierten Daten über die bereitgestellte Sitzung aus dem Back-End in die Map zu laden. Die jeweilige Map wird mit dem Argument "BackingMap" angegeben, das an die Methode "preloadMap" übergeben wird.

```
void preloadMap(Session session, BackingMap backingMap) throws LoaderException;
```

Vorheriges Laden in einem partitionierten MapSet

Maps können in N Partitionen partitioniert werden. Deshalb können Maps auf mehrere Server verteilt werden, wobei jeder Eintrag mit einem Schlüssel gekennzeichnet wird, der nur in einem einzigen dieser Server gespeichert wird. Sehr große Maps können in eXtreme Scale verwaltet werden, weil die Anwendung nicht mehr durch die Heap-Speichergröße einer einzigen JVM beschränkt ist, die alle Einträge einer Map enthält. Anwendungen, die den Preload-Prozess mit der Methode "preloadMap" der Schnittstelle "Loader" ausführen möchten, müssen den Teil der Daten angeben, die vorher geladen werden sollen. Es ist immer eine feste Anzahl an Partitionen vorhanden. Sie können diese Zahl anhand des folgenden Codebeispiels bestimmen:

```
int numPartitions = backingMap.getPartitionManager().getNumOfPartitions();
int myPartition = backingMap.getPartitionId();
```

Dieses Codebeispiel zeigt, wie eine Anwendung den Teil der Daten angeben kann, der vorher aus der Datenbank geladen werden soll. Anwendungen müssen diese Methoden auch dann verwenden, wenn die Map zunächst nicht partitioniert ist. Diese Methoden bieten Flexibilität: Wenn die Map später von den Administratoren partitioniert wird, funktioniert der Loader weiterhin ordnungsgemäß.

Die Anwendung muss Abfragen absetzen, um den Teil *myPartition* aus dem Back-End abzurufen. Wenn eine Datenbank verwendet wird, kann es unter Umständen einfacher sein, eine Spalte mit der Partitionskennung für einen bestimmten Daten-

satz zu haben, sofern es keine natürliche Abfrage gibt, mit der die Daten in der Tabelle einfach partitioniert werden können.

Ein Beispiel für die Implementierung eines Loaders für eine replizierte eXtreme-Scale-Umgebung finden Sie im Abschnitt zum Schreiben eines Loaders mit einem Preload-Controller für Replikate im *Programmierhandbuch*.

Leistung

Die Preload-Implementierung kopiert Daten aus dem Back-End in die Map, indem sie mehrere Objekte in der Map in einer einzigen Transaktion speichert. Die optimale Anzahl der pro Transaktion zu speichernden Datensätze richtet sich nach mehreren Faktoren, einschließlich der Komplexität und der Größe. Wenn die Transaktion beispielsweise Blöcke mit mehr als 100 Einträgen enthält, nehmen die Leistungsgewinne ab, wenn Sie die Anzahl der Einträge erhöhen. Zur Bestimmung der optimalen Anzahl beginnen Sie mit 100 Einträgen, und erhöhen Sie dann die Anzahl, bis keine Leistungsgewinne mehr zu verzeichnen sind. Mit größeren Transaktionen kann eine bessere Replikationsleistung erzielt werden. Denken Sie daran, dass der Preload-Code nur im primären Shard ausgeführt wird. Die vorher geladenen Daten werden über das primäre Shard in allen Replikaten repliziert, die online sind.

MapSets vorher laden

Wenn die Anwendung ein MapSet mit mehreren Maps verwendet, hat jede Map einen eigenen Loader. Jeder Loader besitzt eine Preload-Methode. Alle Maps werden nacheinander von eXtreme Scale geladen. Es kann effizienter sein, den Preload-Prozess für die Maps so zu gestalten, dass eine einzige Map als Map bestimmt wird, in die die Daten vorher geladen werden. Dieser Prozess ist eine Anwendungskonvention. Beispiel: Die beiden Maps "department" (Abteilung) und "employee" (Mitarbeiter) könnten beide den Loader der Map "department" verwenden. Bei dieser Prozedur wird über Transaktionen gewährleistet, dass in dem Fall, dass eine Anwendung eine Abteilung abrufen möchte, sich die Mitarbeiter für diese Abteilung im Cache befinden. Wenn der Loader der Map "department" eine Abteilung vorher aus dem Back-End lädt, ruft er auch die Mitarbeiter für diese Abteilung ab. Das department-Objekt und die zugehörigen employee-Objekte werden dann der Map in einer einzigen Transaktion hinzugefügt.

Wiederherstellbares vorheriges Laden

Einige Kunden haben sehr große Datenmengen, die zwischengespeichert werden müssen. Das vorherige Laden dieser Daten kann sehr zeitaufwendig sein. Manchmal muss das vorherige Laden abgeschlossen sein, bevor die Anwendung online gehen kann. In diesem Fall können Sie von einem wiederherstellbaren vorherigen Laden profitieren. Angenommen, es gibt Millionen Datensätze, die vorher geladen werden müssen. Die Daten werden vorab in das primäre Shard geladen, und der Prozess scheitert bei Datensatz 800.000. Normalerweise löscht das als neue primäre Shard ausgewählte Replikat den Replikationsstatus und beginnt von vorne. eXtreme Scale kann eine Schnittstelle "ReplicaPreloadController" verwenden. Der Loader für die Anwendung muss auch die Schnittstelle "ReplicaPreloadController" implementieren. Das folgende Beispiel fügt dem Loader eine einzige Methode hinzu: `Status checkPreloadStatus(Session session, BackingMap bmap);`. Diese Methode wird von der Laufzeitumgebung von eXtreme Scale aufgerufen, bevor die Methode "preload" der Schnittstelle "Loader" aufgerufen wird. eXtreme Scale prüft das Ergebnis dieser Methode (Status), um das Verhalten festzulegen, wenn ein Replikat in ein primäres Shard hochgestuft werden muss.

Tabelle 10. Statuswert und Antwort

Zurückgegebener Statuswert	Antwort von eXtreme Scale
Status.PRELOADED_ALREADY	eXtreme Scale ruft die Methode "preload" gar nicht auf, weil dieser Statuswert anzeigt, dass der Preload-Prozess für die Map bereits vollständig abgeschlossen ist.
Status.FULL_PRELOAD_NEEDED	eXtreme Scale löscht die Map und ruft ganz regulär die Methode "preload" auf.
Status.PARTIAL_PRELOAD_NEEDED	eXtreme Scale belässt die Map im aktuellen Zustand und ruft die Methode "preload" auf. Diese Strategie ermöglicht dem Loader der Anwendung, den Preload-Prozess an der Stelle fortzusetzen, an der er zuvor abgebrochen wurde.

Es ist logisch, dass ein primäres Shard beim vorherigen Laden von Daten in die Map einen Status in einer Map im MapSet hinterlassen muss, das repliziert wird, so dass das Replikat den zurückzugebenden Status bestimmen kann. Sie können dazu eine zusätzliche Map verwenden, die z. B. den Namen RecoveryMap hat. Diese RecoveryMap muss zu demselben MapSet gehören, das vorher geladen wird, um sicherzustellen, dass die replizierte Map mit den vorher geladenen Daten konsistent ist. Eine empfohlene Implementierung folgt.

Während der Preload-Prozess die einzelnen Datensatzblöcke festschreibt, aktualisiert er im Rahmen dieser Transaktion auch einen Zähler oder Wert in der RecoveryMap. Die vorher geladenen Daten und die RecoveryMap-Daten werden automatisch in den Replikaten repliziert. Wenn das Replikat in ein primäres Shard hochgestuft wird, kann es anhand der RecoveryMap prüfen, was passiert ist.

Die RecoveryMap kann einen einzigen Eintrag mit dem Statusschlüssel enthalten. Wenn kein Objekt für diesen Schlüssel vorhanden ist, müssen Sie einen vollständigen Preload-Prozess durchführen (checkPreloadStatus gibt FULL_PRELOAD_NEEDED zurück). Wenn ein Objekt für diesen Statusschlüssel vorhanden ist und der Wert COMPLETE lautet, ist der Preload-Prozess abgeschlossen, und die Methode "checkPreloadStatus" gibt PRELOADED_ALREADY zurück. Andernfalls zeigt das Wertobjekt an, wo der Preload-Prozess erneut gestartet werden muss, und die Methode "checkPreloadStatus" gibt PARTIAL_PRELOAD_NEEDED zurück. Der Loader kann den Wiederherstellungspunkt in einer Instanzvariablen speichern, so dass der Loader beim Aufruf der Methode "preload" diesen Ausgangspunkt kennt. Die RecoveryMap kann auch einen Eintrag pro Map enthalten, wenn jede Map gesondert vorher geladen wird.

Handhabung der Wiederherstellung im synchronen Replikationsmodus mit einem Loader

Die Laufzeitumgebung von eXtreme Scale ist so konzipiert, dass festgeschriebene Daten beim Ausfall des primären Shards nicht verloren gehen. Im folgenden Abschnitt werden der verwendeten Algorithmen beschrieben. Diese Algorithmen gelten nur, wenn eine Replikationsgruppe die synchrone Replikation verwendet. Ein Loader ist optional.

Die Laufzeitumgebung von eXtreme Scale kann so konfiguriert werden, dass alle Änderungen in einem primären Shard synchron in den Replikaten repliziert werden. Wenn ein synchrones Replikat verteilt wird, erhält es eine Kopie der vorhandenen Daten im primären Shard. In dieser Zeit empfängt das primäre Shard weiterhin Transaktionen und kopiert sie asynchron in das Replikat. Das Replikat wird in dieser Zeit nicht als online eingestuft.

Wenn das Replikat denselben Stand wie das primäre Shard hat, wechselt das Replikat in den Peer-Modus, und die synchrone Replikation beginnt. Jede im primären Shard festgeschriebene Transaktion wird an die synchronen Replikate gesendet, und das primäre Shard wartet auf eine Antwort jedes Replikats. Eine synchrone Festschreibungsfolge mit einem Loader im primären Shard setzt sich aus den folgenden Schritten zusammen:

Tabelle 11. Festschreibungsfolge im primären Shard

Schritt mit Loader	**Schritt ohne Loader**
Sperren für Einträge abrufen	Identisch
Änderung mit Flush an den Loader übertragen	Nulloperation
Änderungen im Cache speichern	Identisch
Änderungen an Replikate senden und auf Bestätigung warten	Identisch
Festschreibung im Loader über das TransactionCallback-Plug-in	Methode "commit" des Plug-ins wird zwar aufgerufen, führt aber keine Aktion aus
Sperren für Einträge freigeben	Identisch

Beachten Sie, dass die Änderungen an das Replikat gesendet werden, bevor sie im Loader festgeschrieben werden. Um zu bestimmen, wann die Änderungen im Replikat festgeschrieben werden, ändern Sie diese Folge. Initialisieren Sie während der Initialisierung die Transaktionslisten im primären Shard wie folgt:

```
CommitedTx = {}, RolledBackTx = {}
```

Für eine synchrone Commit-Verarbeitung verwenden Sie die folgende Folge:

Tabelle 12. Synchrone Commit-Verarbeitung

Schritt mit Loader	**Schritt ohne Loader**
Sperren für Einträge abrufen	Identisch
Änderung mit Flush an den Loader übertragen	Nulloperation
Änderungen im Cache speichern	Identisch
Änderungen mit einer festgeschriebenen Transaktion senden, Rollback der Transaktion an das Replikat durchführen und auf Bestätigung warten	Identisch
Liste festgeschriebener und rückgängig gemachter Transaktionen löschen	Identisch
Festschreibung im Loader über das TransactionCallBack-Plug-in	Methode des TransactionCallBack-Plug-ins wird weiterhin aufgerufen, führt aber gewöhnlich keine Aktion aus
Bei erfolgreicher Festschreibung Transaktion der Liste festgeschriebener Transaktionen hinzufügen, andernfalls der Liste rückgängig gemachter Transaktionen hinzufügen	Nulloperation
Sperren für Einträge freigeben	Identisch

Für die Replikatverarbeitung verwenden Sie die folgende Folge:

1. Änderungen empfangen

2. Alle empfangenen Transaktionen in der Liste festgeschriebener Transaktionen festschreiben
3. Alle empfangenen Transaktionen in der Liste rückgängig gemachter Transaktionen rückgängig machen
4. Transaktion oder Sitzung starten
5. Änderung auf die Transaktion oder Sitzung anwenden
6. Transaktion oder Sitzung in der Liste offener Transaktionen speichern
7. Antwort zurücksenden

Beachten Sie, dass im Replikat keine Loader-Interaktionen stattfinden, während sich das Replikat im Replikatmodus befindet. Das primäre Shard muss alle Änderungen mit Push über den Loader übertragen. Das Replikat nimmt keine Änderungen vor. Dieser Algorithmus hat den Nebeneffekt, dass das Replikat immer die Transaktionen hat, diese aber erst festgeschrieben werden, wenn die nächste primäre Transaktion den Festschreibungsstatus dieser Transaktionen sendet. Erst dann werden die Transaktionen im Replikat festgeschrieben oder rückgängig gemacht. Bis dahin sind die Transaktionen nicht festgeschrieben. Sie können einen Zeitgeber für das primäre Shard hinzufügen, der das Transaktionsergebnis nach kurzer Zeit (ein paar Sekunden) sendet. Dieser Zeitgeber verringert, schließt aber das Risiko veralteter Daten in diesem Zeitfenster nicht ganz aus. Veraltete Daten sind nur dann ein Problem, wenn der Lesemodus für Replikate verwendet wird. Andernfalls haben die veralteten Daten keine Auswirkungen auf die Anwendung.

Wenn das primäre Shard ausfällt, ist es wahrscheinlich, dass einige Transaktionen zwar im primären Shard festgeschrieben oder rückgängig gemacht wurden, aber die Nachricht mit diesen Ergebnissen nicht mehr an das Replikat gesendet werden konnte. Wenn ein Replikat als neues primäres Shard hochgestuft wird, ist eine der ersten Aktionen die Behandlung dieser Bedingung. Jede offene Transaktion wird erneut für die Map-Gruppe des neuen primären Shards ausgeführt. Wenn ein Loader vorhanden ist, wird die erste Transaktion an den Loader übergeben. Diese Transaktionen werden in strikter First In/First Out-Reihenfolge angewendet. Wenn eine Transaktion scheitert, wird sie ignoriert. Sind drei Transaktionen, A, B und C, offen, kann A festgeschrieben, B rückgängig gemacht und C ebenfalls festgeschrieben werden. Keine der Transaktionen hat Auswirkung auf die anderen. Gehen Sie davon aus, dass sie voneinander unabhängig sind.

Ein Loader kann eine geringfügig andere Logik verwenden, wenn er sich im Modus für Fehlerbehebung durch Funktionsübernahme und nicht im normalen Modus befindet. Der Loader kann problemlos feststellen, wann er sich im Modus für Fehlerbehebung durch Funktionsübernahme befindet, indem er die Schnittstelle "ReplicaPreloadController" implementiert. Die Methode "checkPreloadStatus" wird erst aufgerufen, wenn der Loader wieder aus dem Modus für Fehlerbehebung durch Funktionsübernahme in den normalen Modus wechselt. Wenn die Methode "apply" der Schnittstelle "Loader" vor der Methode "checkPreloadStatus" aufgerufen wird, handelt es sich deshalb um eine Wiederherstellungstransaktion. Nach dem Aufruf der Methode "checkPreloadStatus" ist die Fehlerbehebung durch Funktionsübernahme abgeschlossen.

Lastausgleich mit Replikaten

eXtreme Scale sendet, sofern nicht anders konfiguriert, alle Lese- und Schreibanforderungen an den primären Server für eine bestimmte Replikationsgruppe. Der primäre Server muss alle Anforderungen von Clients bearbeiten. Sie können konfigurieren, dass Leseanforderungen auch an Replikate des primären Servers gesendet werden. Durch das Senden von Leseanforderungen an die Replikate kann die Last

der Leseanforderungen auf mehrere Java Virtual Machines (JVM) verteilt werden. Die Verwendung von Replikaten für Leseanforderungen kann jedoch zu inkonsistenten Antworten führen.

Der Lastausgleich mit Replikaten wird gewöhnlich nur verwendet, wenn Clients Daten zwischenspeichern, die sich ständig ändern oder wenn die Clients pessimistisches Sperren verwenden.

Wenn sich die Daten ständig ändern und anschließend im nahen Cache des Clients ungültig gemacht werden, sollte der primäre Server daraufhin eine relativ hohe Rate von get-Anforderungen von Clients sehen. Im pessimistischen Sperrmodus ist kein lokaler Cache vorhanden, also werden alle Anforderungen an den primären Server gesendet.

Wenn die Daten relativ statisch sind oder der pessimistische Modus nicht verwendet wird, hat das Senden von Leseanforderungen an das Replikat keine erheblichen Auswirkungen auf die Leistung. Die Häufigkeit der get-Anforderungen von Clients mit Caches, die voll von Daten sind, ist nicht sehr hoch.

Wenn ein Client zum ersten Mal gestartet wird, ist sein naher Cache leer. Cacheanforderungen an den leeren Cache werden an den primären Server weitergeleitet. Nach und nach werden Daten in den Clientcache gestellt, so dass die Anforderungslast abnimmt. Wenn sehr viele Clients gleichzeitig gestartet werden, kann die Last erheblich und das Senden von Leseanforderungen an das Replikat eine angemessene Leistungsoption sein.

Clientseitige Replikation

Mit eXtreme Scale können Sie eine Server-Map in einem oder mehreren Clients durch asynchrone Replikation replizieren. Ein Client kann über die Methode ClientReplicableMap.enableClientReplication eine lokale schreibgeschützte Kopie einer serverseitigen Map anfordern.

```
void enableClientReplication(Mode mode, int[] partitions,
ReplicationMapListener listener) throws ObjectGridException;
```

Der erste Parameter ist der Replikationsmodus. Dieser Modus kann eine fortlaufende Replikation oder eine Momentaufnahmenreplikation sein. Der zweite Parameter ist ein Bereich von Partitions-IDs, die die Partitionen darstellen, von denen Daten repliziert werden sollen. Wenn der Wert null oder ein leerer Bereich ist, werden die Daten von allen Partitionen repliziert. Der letzte Parameter ist ein Listener für den Empfang von Clientreplikationsereignissen. Weitere Einzelheiten hierzu finden Sie in den Beschreibungen der Schnittstellen "ClientReplicableMap" und "ReplicationMapListener" in der API-Dokumentation.

Nach dem Aktivieren der Replikation wird der Server gestartet, um die Map im Client zu replizieren. Irgendwann einmal ist der Client im Vergleich mit dem Server nur ein paar Transaktionen im Rückstand.

Fehlererkennungstypen

WebSphere eXtreme Scale kann zuverlässig Fehler erkennen.

Austausch von Überwachungssignalen

1. Sockets werden zwischen Java Virtual Machines bleiben offen, und wenn ein Socket unerwartet geschlossen wird, wird dieses unerwartete Schließen als Fehler von der Peer-JVM erkannt. Bei dieser Erkennung werden Fehlerfälle wie

beispielsweise das sehr schnelle Beenden der Java Virtual Machine abgefangen. Außerdem unterstützt dieser Erkennungstyp die Wiederherstellung nach solchen Fehlern in gewöhnlich weniger als einer Sekunde.

2. Weitere Typen von Fehlern sind Betriebssystempanik, physischer Ausfall eines Servers und Netzfehler. Diese Fehler werden über den Austausch von Überwachungssignalen erkannt.

Überwachungssignale werden in regelmäßigen Abständen von zwei Prozessen ausgetauscht. Wenn eine festgelegte Anzahl an Überwachungssignalen verpasst wird, wird von einem Fehler ausgegangen. Mit diesem Ansatz werden Fehler in N*M Sekunden erkannt, wobei N für die Anzahl verpasster Überwachungssignal und M für das Intervall steht, in dem die Überwachungssignale gesendet werden. Die direkte Festlegung von M und N wird nicht unterstützt. Stattdessen wird ein so genannter Slider-Mechanismus verwendet, der die Verwendung eines Bereichs getesteter Kombinationen von M und N ermöglicht.

Fehler

Ein Prozess kann auf verschiedene Arten fehlschlagen. Ein Prozess kann fehlschlagen, weil eine Ressourcengrenze erreicht wurde, wie z. B. die maximale Größe des Heap-Speichers, oder weil eine Prozesssteuerungslogik den Prozess beendet hat. Das Betriebssystem kann ausfallen, was dazu führt, dass alle auf dem System ausgeführten Prozesse verloren gehen. Die Hardware, wie z. B. die Netzschnittstellenkarte, kann (wenn auch weniger häufig) ausfallen, was zur Trennung des Betriebssystems vom Netz führen kann. In diesem Kontext können all diese Fehler in zwei Kategorien eingeteilt werden: Prozessfehler und Konnektivitätsverlust.

Prozessfehler

WebSphere eXtreme Scale reagiert sehr schnell auf Prozessfehler. Wenn ein Prozess fehlschlägt, ist das Betriebssystem für die Bereinigung aller übrig gebliebenen Ressourcen verantwortlich, die vom Prozess verwendet wurden. Diese Bereinigung umfasst die Portzuordnung und die Konnektivität. Wenn ein Prozess fehlschlägt, wird sofort ein Signal über die von diesem Prozess verwendeten Verbindungen gesendet, um jede einzelne Verbindung zu schließen.

Konnektivitätsverlust

Ein Konnektivitätsverlust tritt auf, wenn die Verbindung des Betriebssystems zum Netz getrennt wird. Infolgedessen kann das Betriebssystem keine Signale an andere Prozesse senden. Es gibt mehrere Gründe für einen Konnektivitätsverlust, die jedoch in zwei Kategorien eingeteilt werden können: Hostausfall und Isolierung.

Hostausfall

Wenn die Stromversorgung einer Hostmachine unterbrochen wird, fällt der Host sofort aus.

Isolierung

Dieses Szenario stellt die komplizierteste Fehlerbedingung für Software dar. Die Behebung einer solchen Fehlerbedingung stellt sich so schwierig dar, weil davon ausgegangen wird, dass der Prozess nicht verfügbar ist, obwohl dies gar nicht der Fall ist.

Containerausfall

Containerausfälle werden im Allgemeinen von Peer-Containern über den Stammgruppenmechanismus erkannt. Wenn ein Container oder eine Gruppe von Containern ausfällt, migriert der Katalogservice die Shards aus den betroffenen Containern. Der Katalogservice sucht zuerst nach einem synchronen Replikat, bevor er die Migration auf ein asynchrones Replikat durchführt. Nach der Migration der primären Shards in die neuen Hostcontainer durchsucht der Katalogservice die neuen Hostcontainer nach den Replikaten, die jetzt fehlen.

Anmerkung: Containerisolierung - Der Katalogservice migriert Shards aus Containern, wenn der Container als nicht verfügbar erkannt wird. Wenn diese Container wieder verfügbar werden, berücksichtigt der Katalogservice sie bei der Verteilung wie beim normalen Startablauf.

Latenzzeit für Erkennung von Containerausfällen

Ausfälle können in die folgenden beiden Kategorien eingeteilt werden: temporäre Ausfälle und permanente Ausfälle. Temporäre Ausfälle werden gewöhnlich durch einen Prozessfehler verursacht. Solche Ausfälle werden vom Betriebssystem erkannt, das genutzte Ressourcen, wie z. B. Netz-Sockets, sehr schnell wiederherstellen kann. Gewöhnlich werden temporäre Ausfälle in weniger als einer Sekunde erkannt. Die Erkennung permanenter Ausfälle kann unter Verwendung der Standardoptimierung für Überwachungssignale bis zu 200 Sekunden dauern. Zu solchen Ausfällen gehören physische Ausfälle von Maschinen, das Ziehen von Netzkabeln und Betriebssystemausfälle. Somit muss eXtreme Scale darauf vertrauen, dass permanente Ausfälle durch den Austausch von Überwachungssignalen erkannt werden, der konfiguriert werden kann.

Mehrere Containerausfälle

Ein Replikat wird niemals in demselben Prozess wie das primäre Shard ausgeführt, da dies beim Verlust des Prozesses zu einem Verlust des primären Shards und des Replikats führen würde. Die Implementierungsrichtlinie definiert ein Attribut, das der Katalogservice verwendet, um festzustellen, ob ein Replikat auf derselben Maschine wie das primäre Shard platziert werden kann. In einer Entwicklungsumgebung auf einer einzigen Maschine können Sie zwei Container verwenden und die Daten des einen Containers im jeweils anderen replizieren. In Produktionsumgebungen ist die Verwendung einer einzigen Maschine unzureichend, weil der Verlust dieses Hosts zum Verlust beider Container führt. Zum Wechsel vom Entwicklungsmodus auf einer einzigen Maschine in den Produktionsmodus mit mehreren Maschinen müssen Sie den Entwicklungsmodus in der Konfigurationsdatei der Implementierungsrichtlinie inaktivieren.

Ausfall des Katalogservice

Da das Katalogservice-Grid ein eXtreme-Scale-Grid ist, wird der Stammgruppenmechanismus auch hier auf dieselbe Weise wie beim Containerausfallprozess verwendet. Der Hauptunterschied besteht darin, dass die Katalogservicedomäne einen Peer-Auswahlprozess für die Definition des primären Shards an Stelle des Katalogservicealgorithmus verwendet, der für die Container verwendet wird.

Beachten Sie, dass der Verteilungsservice und der Stammgruppierungsservice Services vom Typ "1 von N" sind, der Positionsservice und die Verwaltung hingegen überall ausgeführt werden. Ein Service vom Typ "1 von N" wird nur in einem einzigen Member der HA-Gruppe ausgeführt. Der Positionsservice und der Verwal-

tungsservice werden in allen Membern der HA-Gruppe ausgeführt. Der Verteilungsservice und der Stammgruppierungsservice sind Singletons, weil sie für das Layout des Systems verantwortlich sind. Der Positionsservice und die Verwaltung sind Services, die ausschließlich im Lesezugriff arbeiten und zur Unterstützung der Skalierbarkeit überall vorhanden sind.

Der Katalogservice verwendet die Replikation für seine eigene Fehlertoleranz. Wenn ein Katalogserviceprozess fehlschlägt, muss der Service erneut gestartet werden, um das System in einem Zustand wiederherzustellen, der die gewünschte Stufe der Verfügbarkeit bietet. Schlagen alle Prozesse, in denen der Katalogservice ausgeführt wird, fehl, verliert eXtreme Scale kritische Daten. Nach diesem Fehler müssen alle Container erneut gestartet werden. Da der Katalogservice in vielen Prozessen ausgeführt werden kann, ist das Auftreten dieses Fehlers eher unwahrscheinlich. Wenn Sie jedoch alle Prozesse auf einer einzigen Maschine, in einem einzigen Blade-Gehäuse oder über einen einzigen Netz-Switch ausführen, ist die Wahrscheinlich eines Fehlers hoch. Versuchen Sie, bekannte Fehlermodi auf Maschinen, auf denen der Katalogservice ausgeführt wird, zu beseitigen, um die Fehlerwahrscheinlichkeit zu reduzieren.

Tabelle 13. Zusammenfassung der Fehlererkennung und Wiederherstellung

Verlusttyp	**Erkennungsmechanismus**	**Wiederherstellungsmethode**
Prozessverlust	Ein-/Ausgabe	Neustart
Serververlust	Überwachungssignal	Neustart
Netzausfall	Überwachungssignal	Netz und Verbindung wiederherstellen
Serverseitige Blockierung	Überwachungssignal	Server stoppen und erneut starten
Server ausgelastet	Überwachungssignal	Warten, bis Server wieder verfügbar ist

Katalogservice mit hoher Verfügbarkeit

Eine Katalogservicedomäne ist das Grid der verwendeten Katalogserver, die Topologieinformationen für alle Container in Ihrer eXtreme-Scale-Umgebung enthalten. Der Katalogservice steuert den Lastausgleich und das Routing für alle Clients. Wenn Sie eXtreme Scale als speicherinternen Verarbeitungsbereich implementieren möchten, müssen Sie den Katalogservice für die hohe Verfügbarkeit zu einer Katalogservicedomäne zusammenfassen.

Komponenten der Katalogservicedomäne

Wenn mehrere Katalogserver gestartet werden, wird einer der Server als Masterkatalogserver ausgewählt. Dieser Masterserver akzeptiert IIOP-Überwachungssignale (Internet Inter-ORB Protocol) und bearbeitet Systemdatenänderungen, die sich aufgrund von Änderungen im Katalogservice oder in den Containern ergeben.

Wenn Clients einen Kontakt zu einem der Katalogserver herstellen, wird die Routing-Tabelle für die Katalogservicedomäne über den CORBA-Servicekontext (Common Object Request Broker Architecture) an die Clients weitergegeben.

Konfigurieren Sie mindestens drei Katalogserver. Wenn Ihre Konfiguration Zonen enthält, können Sie einen Katalogserver pro Zone konfigurieren.

Wenn ein Server und ein Container von eXtreme Scale den Kontakt zu einem der Katalogserver herstellen, wird die Routing-Tabelle für die Katalogservicedomäne ebenfalls über den CORBA-Servicekontext an den Server und Container von eXtreme Scale weitergegeben. Ist der kontaktierte Katalogserver derzeit nicht der Masterkatalogserver, wird die Anforderung automatisch an den aktuellen Masterkatalogserver umgeleitet und die Routing-Tabelle für den Katalogserver aktualisiert.

Anmerkung: Ein Katalogserver-Grid und ein Containerserver-Grid unterscheiden sich in vielerlei Hinsicht. Die Katalogservicedomäne ist für die hohe Verfügbarkeit Ihrer Systemdaten verantwortlich. Das Container-Grid ist für die hohe Verfügbarkeit, die Skalierbarkeit und das Workload-Management Ihrer Daten verantwortlich. Deshalb sind zwei verschiedene Routing-Tabellen vorhanden: die Routing-Tabelle für die Katalogservicedomäne und die Routing-Tabelle für die Server-Grid-Shards.

Die Zuständigkeiten des Katalogs sind in eine Reihe von Services unterteilt. Der Stammgruppenmanager führt die Peer-Gruppierung für die Vitalitätsüberwachung durch, der Verteilungsservice führt die Zuordnung durch, der Verwaltungsservice ermöglicht den Zugriff auf die Verwaltung, und der Positionsservice verwaltet die Positionen.

Implementierung der Katalogservicedomäne

Stammgruppenmanager

Der Katalogservice verwendet den High Availability Manager (kurz HA-Manager), um Prozesse für die Überwachung der Verfügbarkeit zu gruppieren. Jede Prozessgruppierung ist eine Stammgruppe. Mit eXtreme Scale gruppiert der Stammgruppenmanager die Prozesse dynamisch. Diese Prozesse werden für die Skalierbarkeit klein gehalten. Jede Stammgruppe wählt ein führendes Member aus, das zusätzlich dafür verantwortlich ist, Statusnachrichten an den Stammgruppenmanager zu senden, wenn einzelne Member ausfallen. Über denselben Statusmechanismus wird erkannt, wenn alle Member einer Gruppe ausfallen und daraufhin die Kommunikation mit dem führenden Member verloren geht.

Der Stammgruppenmanager ist ein vollständig automatischer Service, der für die Organisation der Container in kleine Servergruppen zuständig ist, die dann automatisch lose miteinander zu einem ObjectGrid verbunden werden. Wenn ein Container den ersten Kontakt zum Katalogservice herstellt, wartet er auf die Zuteilung einer neuen oder vorhandenen Gruppe. Eine Implementierung von eXtreme Scale setzt sich aus vielen solcher Gruppen zusammen, und diese Gruppierung ist ein Enabler für die Skalierbarkeit von Schlüsseln. Jede Gruppe ist eine Gruppe von Java Virtual Maschines, die den Austausch von Überwachungssignalen verwenden, um die Verfügbarkeit der jeweils anderen Gruppen zu überwachen. Eines dieser Gruppen-Member wird als führendes Member ausgewählt und ist zusätzlich dafür verantwortlich, die Verfügbarkeitsinformationen an den Katalogservice zu übermitteln, um durch Neuzuordnung und Routenweiterleitung auf Fehler reagieren zu können.

Verteilungsservice

Der Katalogservice verwaltet die Verteilung von Shards auf die verfügbaren Container. Der Verteilungsservice ist dafür verantwortlich, dass die Ressourcen gleichmäßig auf die physischen Ressourcen verteilt werden. Der Verteilungsservice ordnet die einzelnen Shards ihren Hostcontainern zu. Der Verteilungsservice wird als einer von N ausgewählten Services im Daten-Grid ausgeführt, so dass genau eine Instanz des Service aktiv ist. Wenn diese Instanz ausfällt, wird ein anderer Prozess

ausgewählt, der die Arbeit dieser Instanz übernimmt. Aus Redundanzgründen wird der Status des Katalogservice in allen Servern, in denen der Katalogservice ausgeführt, repliziert.

Verwaltung

Der Katalogservice ist auch der logische Einstiegspunkt für die Systemverwaltung. Der Katalogservice enthält eine Managed Bean (MBean) und stellt JMX-URLs (Java Management Extensions) für alle vom Service verwalteten Server bereit.

Positionsservice

Der Positionsservice tritt als Touchpoint für Clients, die die Container suchen, in denen die gewünschte Anwendung ausgeführt wird, sowie für die Container auf, die in ihnen ausgeführte Anwendungen beim Verteilungsservice registrieren möchten. Der Positionsservice wird zur horizontalen Skalierung dieser Funktion in allen Grid-Membern ausgeführt.

Der Katalogservice enthält Logik, die in stabilen Zuständen gewöhnlich inaktiv ist. Deshalb wirkt sich der Katalogservice nur geringfügig auf die Skalierbarkeit aus. Der Service ist so konzipiert, dass er Hunderte von Containern bedienen kann, die gleichzeitig verfügbar werden. Für die Verfügbarkeit konfigurieren Sie den Katalogservice in einem Grid.

Planung

Nach dem Starten einer Katalogservicedomäne werden die Member des Grids miteinander verbunden. Planen Sie die Topologie Ihrer Katalogservicedomäne sorgfältig, weil die Konfiguration der Katalogservicedomäne zur Laufzeit nicht geändert werden kann. Verteilen Sie das Grid so breit wie möglich, um Fehler zu verhindern.

Katalogservicedomäne starten

Weitere Informationen zum Erstellen einer Katalogservicedomäne finden Sie in den Details zum Starten einer Katalogservicedomäne im *Administratorhandbuch*.

Container von eXtreme Scale, die in WebSphere Application Server integriert sind, zu einer eigenständigen Katalogservicedomäne verbinden

Sie können Container von eXtreme Scale, die in eine Umgebung von WebSphere Application Server integriert sind, zu einer eigenständigen Katalogservicedomäne verbinden.

- (Veraltet) In früheren Releases haben Sie die Katalogservices über eine angepasste Eigenschaft mit einer Katalogservicedomäne verbunden. Sie können diese Eigenschaft zwar weiterhin verwenden, aber sie ist veraltet. Weitere Einzelheiten zu dieser angepassten Eigenschaft finden Sie in den Informationen zum Starten des Katalogserviceprozesses in einem WebSphere Application Server im *Administratorhandbuch*.

  Anmerkung: Servernamenskollision: Da diese Eigenschaft sowohl zum Starten des Katalogservers von eXtreme Scale als auch zum Herstellen einer Verbindung zum Katalogserver verwendet wird, dürfen die Katalogserver keinen Namen haben, der mit einem der Server von WebSphere Application Server übereinstimmt.

Weitere Informationen finden Sie in „Katalogserver-Quorum”.

Katalogserver-Quorum

Das Quorum ist die minimale Anzahl an Katalogservern, die erforderlich ist, um Verteilungsoperationen für das Daten-Grid durchzuführen. Die minimale Anzahl ist die vollständige Gruppe der Katalogserver, sofern Sie das Quorum nicht überschreiben.

Wichtige Begriffe

Im Folgenden finden Sie eine Liste der Begriffe, die sich auf Quorumaspekte für WebSphere eXtreme Scale beziehen.

- **Brownout**: Ein Brownout ist der temporäre Verlust der Konnektivität zwischen Servern.
- **Blackout:** Ein Blackout ist der permanente Verlust der Konnektivität zwischen Servern.
- **Rechenzentrum:** Ein Rechenzentrum ist eine sich an einem bestimmten Standort befindliche Gruppe von Servern, die im Allgemeinen über ein lokales Netz (LAN, Local Area Netzwork) miteinander verbunden sind.
- **Zone:** Eine Zone ist eine Konfigurationsoption, die verwendet wird, um Server zu gruppieren, die ein gemeinsames physisches Merkmal haben. Zonenbeispiele für eine Gruppe von Servern sind ein Rechenzentrum, wie im vorherigen Listenpunkt beschrieben, ein Bereichsnetz, ein Gebäude, ein Stockwerk eines Gebäudes usw.
- **Überwachungssignal**: Der Austausch von Überwachungssignalen ist ein Mechanismus, der verwendet wird, um festzustellen, ob eine bestimmte Java Virtual Machine (JVM) aktiv ist.

Topologie

In diesem Abschnitt wird erläutert, wie WebSphere eXtreme Scale in einem Netz arbeitet, das unzuverlässige Komponenten enthält. Ein Beispiel für ein solches Netz ist ein Netz, das sich über mehrere Rechenzentren erstreckt.

IP-Adressraum

WebSphere eXtreme Scale erfordert ein Netz, in dem alle adressierbaren Elemente im Netz ungehindert eine Verbindung zu jedem anderen adressierbaren Element im Netz herstellen können. Dies bedeutet, dass WebSphere eXtreme Scale einen flachen IP-Adressraum erfordert und Firewalls für den gesamten Datenverkehr voraussetzt, der zwischen den IP-Adressen und Ports übertragen wird, die von den Java Virtual Machines (JVM) verwendet werden, die Elemente von WebSphere eXtreme Scale enthalten.

Verbundene LANs

Jedem lokalen Netz (LAN, Local Area Network) wird eine Zonenkennung für die Anforderungen von WebSphere eXtreme Scale zugeordnet. WebSphere eXtreme Scale fragt JVMs in einer einzigen Zone aggressiv mit Überwachungssignalen ab. Ein verpasstes Überwachungssignal löst ein Failover-Ereignis aus, wenn der Katalogservice das Quorum hat.

Katalogservicedomäne und Containerserver

Eine Katalogservicedomäne ist eine Sammlung ähnlicher JVMs. Eine Katalogservicedomäne ist ein Grid, das sich aus Katalogservern zusammensetzt und eine feste Größe hat. Die Anzahl der Containerserver ist jedoch dynamisch. Containerserver können nach Bedarf hinzugefügt und entfernt werden. In einer Konfiguration mit drei Rechenzentren erfordert WebSphere eXtreme Scale eine Katalogservice-JVM pro Rechenzentrum.

Die Katalogservicedomäne verwendet einen Mechanismus mit vollständigem Quorum. Das bedeutet, dass alle Member des Daten-Grids einer Aktion zustimmen müssen.

Containerserver-JVMs werden mit einer Zonenkennung gekennzeichnet. Das Grid der Container-JVMs wird automatisch in kleinere JVM-Stammgruppen aufgeteilt. Eine Stammgruppe enthält nur die JVMs aus einer einzigen Zone. JVMs unterschiedlicher Zonen werden nie in dieselbe Stammgruppe aufgenommen.

Eine Stammgruppe versucht aggressiv, Ausfälle ihrer Member-JVMs zu erkennen. Die Container-JVMs einer Stammgruppe dürfen sich nicht über mehrere LANs erstrecken, die über Verbindungen wie ein Weitverkehrsnetz (WAN, Wide Area Network) miteinander verbunden sind. Dies bedeutet, dass eine Stammgruppe keine Container aus derselben Zone enthalten kann, die in anderen Rechenzentren ausgeführt werden.

Serverlebenszyklus

Katalogserverstart

Die Katalogserver werden mit dem Befehl "startOgServer" gestartet. Der Quorum-Mechanismus ist standardmäßig inaktiviert. Zum Aktivieren des Quorums können Sie das Flag "`-quorum enabled`" an den Befehl "startOgServer" übergeben oder die Eigenschaft "`enableQuorum=true`" in der Eigenschaftendatei hinzufügen. Für alle Katalogserver muss dieselbe Quorum-Einstellung festgelegt werden.

```
# bin/startOgServer cat0 –serverProps objectGridServer.properties
```

Datei "objectGridServer.properties"

```
catalogClusterEndPoints=cat0:cat0.domain.com:6600:6601,
cat1:cat1.domain.com:6600:6601
catalogServiceEndPoints= cat0.domain.com:2809, cat1.domain.com:2809
enableQuorum=true
```

Containerserverstart

Die Containerserver werden mit dem Befehl "startOgServer" gestartet. Wenn ein Daten-Grid über mehrere Rechenzentren verteilt ausgeführt wird, müssen die Server die Zonenkennung verwenden, um das Rechenzentrum zu identifizieren, in dem sie sich befinden. Die Einstellung der Zone in den Grid-Servern ermöglicht WebSphere eXtreme Scale, den Zustand der Server zu überwachen, die dem Rechenzentrum zugeordnet sind, und somit den rechenzentrumsübergreifenden Datenverkehr zu minimieren.

```
# bin/startOgServer gridA0 –serverProps objectGridServer.properties –
objectgridfile xml/objectgrid.xml –deploymentpolicyfile xml/
deploymentpolicy.xml
```

Datei "objectGridServer.properties"

```
catalogServiceEndPoints= cat0.domain.com:2809, cat1.domain.com:2809
zoneName=ZoneA
```

Grid-Server herunterfahren

Die Grid-Server werden mit dem Befehl "stopOgServer" gestoppt. Wenn Sie ein gesamtes Rechenzentrum für Wartungsarbeiten herunterfahren möchten, übergeben Sie die Liste aller Server, die zu dieser Zone gehören. Die Übergabe dieser Liste ermöglicht einen sauberen Statusübergang von der Zone, die umgerüstet wird, auf die noch aktiven Zonen.

```
# bin/stopOgServer gridA0,gridA1,gridA2 –catalogServiceEndPoints
cat0.domain.com:2809,cat1.domain.com:2809
```

Ausfallerkennung

WebSphere eXtreme Scale erkennt Prozessabstürze anhand abnormaler Beendigungsereignisse für Sockets. Der Katalogservice wird sofort darüber benachrichtigt, wenn ein Prozess beendet wird. Ein Blackout wird anhand von nicht beantworteten Überwachungssignalen erkannt. WebSphere eXtreme Scale schützt sich durch die Verwendung einer Quorum-Implementierung selbst vor Brownout-Bedingungen in Rechenzentren.

Implementierung des Austauschs von Überwachungssignalen

In diesem Abschnitt wird beschrieben, wie die Aktivitätsprüfung in WebSphere eXtreme Scale implementiert wird.

Austausch von Überwachungssignalen zwischen Stammgruppen-Membern

Der Katalogservice verteilt Container-JVMs auf Stammgruppen begrenzter Größe. Eine Stammgruppe verwendet zwei Methoden, um den Ausfall ihrer Member zu erkennen. Wenn der Socket einer JVM geschlossen wird, wird diese JVM als inaktiv eingestuft. Außerdem sendet jedes Member in einem in der Konfiguration festgelegten Intervall Überwachungssignale über diese Sockets. Wenn eine JVM nicht innerhalb der konfigurierten maximalen Zeitspanne auf diese Überwachungssignale antwortet, wird die JVM als inaktiv eingestuft.

Ein einziges Member jeder Stammgruppe wird als führendes Member bestimmt. Das führende Stammgruppen-Member ist dafür verantwortlich, dem Katalogservice in regelmäßigen Abständen den Aktivitätsstatus der Stammgruppe und alle Änderungen der Stammgruppenzugehörigkeiten zu melden. Eine Zugehörigkeitsänderung kann beispielsweise der Ausfall einer JVM oder das Hinzufügen einer neuen JVM sein, die in die Stammgruppe aufgenommen wird.

Wenn das führende Stammgruppen-Member zu keinem der Member der Katalogservicedomäne eine Verbindung herstellen kann, setzt es seine Verbindungsversuche fort.

Austausch von Überwachungssignalen in der Katalogservicedomäne

Die Katalogservicedomäne gleicht einer privaten Stammgruppe mit einer statischen Zugehörigkeit und einem Quorum-Mechanismus. Sie erkennt Ausfälle auf dieselbe Weise wie eine normale Stammgruppe. Das Verhalten ist jedoch anders und umfasst eine Quorum-Logik. Außerdem verwendet der Katalogservice eine weniger aggressive Konfiguration für den Austausch von Überwachungssignalen.

Austausch von Überwachungssignalen in einer Stammgruppe

Der Katalogservice muss über den Ausfall von Containerservern informiert werden. Jede Stammgruppe ist für die Erkennung des Ausfalls der Container-JVM und die Meldung dieses Ausfalls an den Katalogservice durch das führende Stammgruppen-Member verantwortlich. Der vollständige Ausfall aller Member einer Stammgruppe ist auch möglich. Wenn die vollständige Stammgruppe ausfällt, muss der Katalogservice diesen Verlust erkennen.

Wenn der Katalogservice eine Container-JVM als ausgefallen kennzeichnet und der Container später wieder als aktiv gemeldet wird, wird die Container-JVM angewiesen, die Containerserver von WebSphere eXtreme Scale herunterzufahren. Eine JVM in diesem Status ist in Abfragen mit dem Befehl "xsadmin" nicht sichtbar. Die Nachrichten in den Protokollen der Container-JVM weisen darauf hin, dass die Container-JVM ausgefallen ist. Sie müssen diese JVM manuell erneut starten.

Bei einem Verlust des Quorums wird der Austausch von Überwachungssignalen so lange ausgesetzt, bis das Quorum wiederhergestellt ist.

Quorum-Verhalten des Katalogservice

Normalerweise haben die Member des Katalogservice vollständige Konnektivität. Die Katalogservicedomäne ist eine statische Gruppe von JVMs. WebSphere eXtreme Scale erwartet, dass stets alle Member des Katalogservice online sind. Der Katalogservice reagiert nur auf Containerereignisse, wenn der Katalogservice das Quorum hat.

Verliert der Katalogservice das Quorum, wartet er, bis das Quorum wiederhergestellt ist. In der Zeit, in der der Katalogservice kein Quorum hat, ignoriert er alle Ereignisse der Containerserver. Containerserver wiederholen alle Anforderungen, die vom Katalogserver in der Zeit zurückgewiesen werden, da WebSphere eXtreme Scale von der Wiederherstellung des Quorums ausgeht.

Die folgende Nachricht zeigt einen Quorum-Verlust an. Suchen Sie nach dieser Nachricht in Ihren Katalogserviceprotokollen.

```
CWOBJ1254W: Der Katalogservice wartet auf das Quorum.
```

WebSphere eXtreme Scale erwartet in den folgenden Fällen einen Quorum-Verlust:

- Das Member mit der Katalogservice-JVM fällt aus.
- Es tritt ein Netz-Brownout ein.
- Es tritt ein Ausfall des Rechenzentrums ein.

Das Stoppen einer Katalogserverinstanz mit dem Befehl `stopOgServer` führt nicht zum Verlust des Quorums, weil das System weiß, dass die Serverinstanz gestoppt wurde. Dies ist etwas anderes als ein JVM-Ausfall oder ein Brownout.

Quorum-Verlust nach JVM-Ausfall

Ein Katalogserver, der ausfällt, führt zu einem Quorum-Verlust. In diesem Fall muss das Quorum so schnell wie möglich wiederhergestellt werden. Ein ausgefallener Katalogservice kann dem Grid erst dann wieder beitreten, wenn das Quorum wiederhergestellt ist.

Quorum-Verlust nach einem Netz-Brownout

WebSphere eXtreme Scale ist auf die Möglichkeit von Brownouts vorbereitet. Ein Brownout ist ein temporärer Verlust der Konnektivität zwischen Rechenzentren. Brownout-Bedingungen sind gewöhnlich transient und sollten innerhalb von Sekunden bzw. Minuten behoben sein. WebSphere eXtreme Scale versucht während eines Brownouts den normalen Betrieb aufrecht zu erhalten. Ein Brownout wird als einzelnes Fehlereignis betrachtet. Es wird davon ausgegangen, dass der Fehler behoben und der normale Betrieb anschließend fortgesetzt wird, ohne dass Aktionen von WebSphere eXtreme Scale erforderlich sind.

Ein langer Brownout kann nur durch Benutzereingriff als Blackout klassifiziert werden. Das Quorum muss auf einer Seite des Brownouts wiederhergestellt werden, damit das Ereignis als als Blackout klassifiziert werden kann.

JVM des Katalogservice stoppen und erneut starten

Wenn ein Katalogserver mit dem Befehl "stopOgServer" gestoppt wird, sinkt das Quorum um einen Server. Das bedeutet, dass die verbleibenden Server weiterhin das Quorum haben. Bei einem Neustart des Katalogservers steigt das Quorum wieder auf die verherige Zahl.

Konsequenzen eines Quorum-Verlusts

Wenn eine Container-JVM ausfällt und das Quorum verloren gegangen ist, findet erst dann eine Wiederherstellung statt, wenn die Brownout-Bedingung behoben ist bzw. wenn der Kunde im Fall eines Blackouts einen Befehl zum Wiederherstellen des Quorums absetzt. WebSphere eXtreme Scale betrachtet einen Quorum-Verlust und einen Containerausfall als doppelten Fehler, aber ein solches Ereignis tritt nur selten ein. Das bedeutet, dass Anwendungen den Schreibzugriff auf Daten, die in der ausgefallenen JVM gespeichert wurden, so lange verlieren können, bis das Quorum wiederhergestellt ist und die normale Wiederherstellung stattfindet.

Wenn Sie versuchen, einen Container zu starten, während das Quorum verloren ist, wird der Container nicht gestartet.

Während eines Quorum-Verlusts ist eine vollständige Clientkonnektivität zulässig. Wenn während des Quorum-Verlusts keine Containerausfälle oder Konnektivitätsprobleme auftreten, können die Clients weiterhin vollständig mit den Containerservern interagieren.

Wenn ein Brownout auftritt, haben einige Clients möglicherweise erst dann wieder Zugriff auf die primären Kopieren oder Replikatkopien der Daten, wenn die Brownout-Bedingung behoben ist.

Neue Clients können gestartet werden, da in jedem Rechenzentrum eine Katalogservice-JVM vorhanden sein sollte, so dass der Client selbst bei einem Brownout-Ereignis mindestens eine Katalogservice-JVM erreichen kann.

Wiederherstellung des Quorums

Wenn das Quorum aus irgendeinem Grund verloren geht, wird zur Wiederherstellung des Quorums ein Wiederherstellungsprotokoll ausgeführt. Beim Verlust des Quorums wird die Aktivitätsprüfung für die Stammgruppen ausgesetzt, und alle Ausfallberichte werden ignoriert. Sobald das Quorum wiederhergestellt ist, nimmt der Katalogservice die Aktivitätsprüfung aller Stammgruppen wieder auf, um ihre Zugehörigkeit unverzüglich zu bestimmen. Alle zuvor in den als ausgefallen gemeldeten Container-JVMs befindichen Shards werden wiederhergestellt. Wenn pri-

märe Shards verloren gegangen sind, werden die noch aktiven Replikate zu primären Shards hochgestuft. Wenn Replikat-Shards verloren gegangen sind, werden zusätzliche Replikate in den noch aktiven JVMs erstellt.

Quorum überschreiben

Diese Option sollte nur verwendet werden, wenn ein Rechenzentrum ausfällt. Der Quorum-Verlust aufgrund des Ausfalls einer Containerservice-JVM oder eines Netz-Brownouts wird gewöhnlich automatisch behoben, sobald die Katalogservice-JVM erneut gestartet bzw. das Netz-Brownout behoben ist.

Nur Administratoren haben Kenntnis von einem Ausfall eines Rechenzentrums. WebSphere eXtreme Scale behandelt Brownouts und Blackouts ähnlich. Sie müssen die eXtreme-Scale-Umgebung über solche Ausfälle mit dem Befehl "xsadmin" in Kenntnis setzen, um das Quorum zu überschreiben. Auf diese Weise wird dem Katalogservice mitgeteilt, davon auszugehen, dass das Quorum mit den aktuellen Zugehörigkeiten erreicht ist und eine vollständige Wiederherstellung stattfindet. Wenn Sie einen Befehl zum Überschreiben des Quorums absetzen, bestätigen Sie, dass die JVMs im ausgefallenen Rechenzentrum wirklich ausgefallen sind und nicht wiederhergestellt werden.

In der folgenden Liste sind einige Szenarien für das Überschreiben des Quorums beschrieben. Angenommen, Sie haben drei Katalogserver: A, B, und C.

- Brownout: Angenommen, es tritt ein Brownout auf, bei dem C vorübergehend isoliert wird. Der Katalogservice verliert das Quorum und wartet auf die Behebung des Brownouts, nach der C der Katalogservicedomäne wieder beitritt und das Quorum wiederhergestellt ist. Ihre Anwendung stellt während dieser Zeit keine Probleme fest.
- Vorübergehender Ausfall: Hier fällt C aus, und der Katalogservice verliert das Quorum. In diesem Fall müssen Sie das Quorum überschreiben. Sobald das Quorum wiederhergestellt ist, kann C erneut gestartet werden. C tritt der Katalogservicedomäne nach dem Neustart wieder bei. Die Anwendung stellt während dieser Zeit keine Probleme fest.
- Ausfall eines Rechenzentrums: Sie verifizieren, dass das Rechenzentrum wirklich ausgefallen und vom Netz isoliert ist. Anschließend setzen Sie den Befehl "xsadmin" zum Überschreiben des Quorums ab. Die anderen beiden noch aktiven Rechenzentren führen eine vollständige Wiederherstellung durch, indem sie Shards, die sich im ausgefallenen Rechenzentrum befinden, ersetzen. Der Katalogservice wird jetzt mit einem vollständigen Quorum von A und B ausgeführt. Die Anwendung kann zwischen dem Beginn des Blackouts und dem Überschreiben des Quorums Verzögerungen oder Ausnahmen feststellen. Sobald das Quorum überschrieben ist, wird das Grid wiederhergestellt und der normale Betrieb fortgesetzt.
- Wiederherstellung des Rechenzentrums: Die noch aktiven Rechenzentren werden bereits mit überschriebenem Quorum ausgeführt. Wenn das Rechenzentrum, in dem C enthalten ist, erneut gestartet wird, müssen alle JVMs im Rechenzentrum erneut gestartet werden. Anschließend tritt C der vorhandenen Katalogservicedomäne wieder bei, und das Quorum wird ohne Benutzereingriff wiederhergestellt.
- Ausfall eines Rechenzentrums und Brownout: Das Rechenzentrum, in dem C enthalten ist, fällt aus. Das Quorum wird überschrieben und in den verbleibenden Rechenzentren wiederhergestellt. Wenn ein Brownout zwischen A und B auftritt, kommen die herkömmlichen Regeln für eine Wiederherstellung nach ei-

nem Brownout zur Anwendung. Nach der Behebung des Brownouts wird das Quorum wiederhergestellt, und die erforderliche Wiederherstellung nach einem Quorum-Verlust findet statt.

Containerverhalten

In diesem Abschnitt wird beschrieben, wie sich die Containerserver-JVMs verhalten, wenn das Quorum verloren geht und anschließend wiederhergestellt wird.

Container enthalten einen oder mehrere Shards. Shards sind primäre Kopien oder Replikatkopien für eine bestimmte Partition. Der Katalogservice ordnet Shards einem Container zu, und der Container berücksichtigt diese Zuordnung so lange, bis er neue Anweisungen vom Katalogservice erhält. Wenn ein primäres Shard in einem Container nicht mit einem Replikat-Shard kommunizieren kann, weil ein Brownout eingetreten ist, setzt es seine Kommunikationsversuche so lange fort, bis es neue Anweisungen vom Katalogservice erhält.

Wenn ein Netz-Brownout auftritt und ein primäres Shard nicht mehr mit dem Replikat kommunizieren kann, wiederholt es die Verbindung so lange, bis der Katalogservice neue Anweisungen bereitstellt.

Verhalten synchroner Replikate

Wenn eine Verbindung unterbrochen ist, kann das primäre Shard neue Transaktionen akzeptieren, solange mindestens so viele Replikate online sind, wie mit der Eigenschaft "minsync" für das Mapset festgelegt wurde. Wenn neue Transaktionen im primären Shard verarbeitet werden, während die Verbindung zum synchronen Replikat unterbrochen ist, wird der Replikatinhalt gelöscht und nach Wiederherstellung der Verbindung mit dem aktuellen Status des primären Shards synchronisiert.

Von der synchronen Replikation zwischen Rechenzentren und der synchronen Replikation über WAN-Verbindungen wird dringend abgeraten.

Verhalten asynchroner Replikate

Wenn eine Verbindung unterbrochen ist, kann das primäre Shard neue Transaktionen akzeptieren. Das primäre Shard puffert die Änderungen bis zu einem bestimmten Grenzwert. Wenn die Verbindung mit dem Replikat wiederhergestellt wird, bevor der Grenzwert erreicht ist, wird das Replikat mit den gepufferten Änderungen aktualisiert. Beim Erreichen des Grenzwerts löscht das primäre Shard die gepufferte Liste, und bei der Wiederherstellung der Verbindung zum Replikat wird der Replikatinhalt gelöscht und neu mit dem primären Shard synchronisiert.

Clientverhalten

Unabhängig davon, ob die Katalogservicedomäne das Quorum hat oder nicht, können Clients für Bootstrapping immer eine Verbindung zum Katalogserver herstellen. Der Client versucht, eine Verbindung zu einer Katalogserverinstanz herzustellen, um eine Routentabelle anzufordern und anschließend mit dem Grid zu interagieren. Die Netzkonnektivität kann die Interaktion des Clients mit einigen Partitionen aufgrund der Netzkonfiguration verhindern. Der Client kann für den Abruf ferner Daten eine Verbindung zu lokalen Replikaten herstellen, sofern er entsprechend konfiguriert ist. Clients können keine Daten aktualisieren, wenn die primäre Partition für diese Daten nicht verfügbar ist.

Quorum-Befehle mit xsadmin

In diesem Abschnitt werden für Quorum-Situationen hilfreiche xsadmin-Befehle beschrieben.

Quorum-Status abfragen

Der Quorum-Status einer Katalogserverinstanz kann mit dem Befehl "xsadmin" abgefragt werden.

```
xsadmin –ch cathost –p 1099 –quorumstatus
```

Es gibt fünf mögliche Ergebnisse.

- Quorum ist inaktiviert: Die Katalogserver werden in einem Modus ausgeführt, in dem das Quorum inaktiviert ist. Dieser Modus ist für Entwicklungsumgebungen und Umgebungen mit einem einzigen Rechenzentrum bestimmt. Er wird für Konfigurationen mit mehreren Rechenzentren nicht empfohlen.
- Quorum ist aktiviert, und der Katalogserver hat das Quorum: Das Quorum ist aktiviert, und das System arbeitet normal.
- Quorum ist aktiviert, aber der Katalogserver wartet auf das Quorum: Das Quorum ist aktiviert, und das Quorum ist verloren gegangen.
- Quorum ist aktiviert, und das Quorum wurde überschrieben: Das Quorum ist aktiviert, und das Quorum wurde überschrieben.
- Quorum-Status ist unzulässig: Wenn ein Brownout auftritt, wird der Katalogservice in zwei Partitionen, A und B, aufgeteilt. Der Katalogserver A hat das Quorum überschrieben. Die Netzpartition wird aufgelöst. Der Status des Servers in der Partition B ist unzulässig, und es ist ein Neustart der JVM erforderlich. Dieser Fall tritt auch ein, wenn die Katalog-JVM in Partition B wegen des Brownouts erneut gestartet wird und die Brownout-Bedingung anschließend behoben wird.

Quorum überschreiben

Der Befehl "xsadmin" kann zum Überschreiben des Quorums verwendet werden. Alle noch aktiven Katalogserverinstanzen können verwendet werden. Alle noch aktiven Instanzen werden benachrichtigt, wenn eine Instanz angewiesen wird, das Quorum zu überschreiben. Die Syntax ist wie folgt:

```
xsadmin –ch cathost –p 1099 –overridequorum
```

Diagnosebefehle

- Quorum-Status: Siehe die Beschreibung im vorherigen Abschnitt.
- Stammgruppen auflisten: Zeigt eine Liste aller Stammgruppen an. Die Member und führenden Member der Stammgruppen werden angezeigt.

  ```
  xsadmin –ch cathost –p 1099 –coregroups
  ```
- Server entfernen: Dieser Befehl entfernt einen Server manuell aus dem Grid. Dies ist normalerweise nicht erforderlich, da Server automatisch entfernt werden, wenn sie als ausgefallen erkannt werden, aber der Befehl wird für die Verwendung unter Anleitung der IBM Unterstützungsfunktion bereitgestellt.

  ```
  xsadmin –ch cathost –p 1099 –g Grid –teardown server1,server2,server3
  ```
- Routentabelle anzeigen: Dieser Befehl zeigt die aktuelle Routentabelle an, indem eine neue Clientverbindung zum Grid simuliert wird. Außerdem validiert der

Befehl die Routentabelle, indem er prüft, ob alle Containerserver ihre Rolle in der Routentabelle kennen, z. B. welcher Typ von Shard für welche Partition bestimmt ist.

```
xsadmin –ch cathost –p 1099 –g myGrid -routetable
```

- Nicht zugeordnete Shards anzeigen: Wenn einige Shards nicht im Grid verteilt werden können, können diese Shards mit diesem Befehl aufgelistet werden. Dies geschieht nur, wenn der Verteilungsservice eine Einschränkung hat, die die Verteilung verhindert. Wenn Sie beispielsweise JVMs auf einem einzelnen physischen System starten, das im Produktionsmodus arbeitet, können nur primäre Shards verteilt werden. Replikate werden nicht zugordnet, solange JVMs auf dem zweiten System gestartet werden. Der Verteilungsservice verteilt Replikate nur an JVMs, die andere Adressen haben als die JVMs, in denen sich die primären Shards befinden. Wenn eine Zone keine JVMs enthält, kann dies auch dazu führen, dass Shards nicht zugeordnet werden.

  ```
  xsadmin –ch cathost –p 1099 –g myGrid –unassigned
  ```

- Trace-Einstellungen festlegen: Dieser Befehl legt die Trace-Einstellungen für alle JVMs fest, die dem für den Befehl "xsadmin" angegebenen Filter entsprechen. Diese Einstellung ändert die Trace-Einstellungen nur so lange, bis ein anderer Befehl verwendet wird bzw. die geänderten JVMs ausfallen oder gestoppt werden.

  ```
  xsadmin –ch cathost –p 1099 –g myGrid –fh host1 –settracespec
  ObjectGrid*=event=enabled
  ```

  Dieser Befehl aktiviert die Trace-Erstellung für alle JVMs auf dem System mit dem angegebenen Hostnamen, in diesem Fall host1.

- Map-Größen prüfen: Der Befehl "mapsizes" ist hilfreich, um sicherzustellen, dass die Schlüsselverteilung auf die Shards im Schlüssel einheitlich erfolgt. Wenn einige Container erheblich mehr Schlüssel als andere haben, verwendet die Hash-Funktion für die Schlüsselobjekte wahrscheinlich eine schlechte Verteilung.

  ```
  xsadmin -ch cathost -p 1099 -g myGrid -m myMapSet -mapsizes myMap
  ```

Hinweise zur Transportsicherheit

Da Rechenzentren gewöhnlich auf verschiedene geografische Standorte verteilt sind, können Benutzer aus Sicherheitsgründen die Transportsicherheit zwischen den Rechenzentren aktivieren.

Lesen Sie die Informationen zur Sicherheit auf Transportschicht im *Administratorhandbuch*.

Replikate und Shards

Mit eXtreme Scale kann eine speicherinterne Datenbank oder ein Shard von einer Java Virtual Machine (JVM) in einer anderen repliziert werden. Ein Shard stellt eine Partition dar, die in einen Container gestellt wird. Mehrere Shards, die unterschiedliche Partitionen darstellen, können in einem einzigen Container enthalten sein. Jede Partition hat eine Instanz, die ein primäres Shard ist, und eine konfigurierbare Anzahl an Replikat-Shards. Die Replikat-Shards sind synchron der asynchron. Die Typen und die Verteilung der Replikat-Shards werden von eXtreme Scale über eine Implementierungsrichtlinie bestimmt, die die Mindest- und Maximalanzahl synchroner und asynchroner Shards festlegt.

Shard-Typen

Für die Replikation werden drei Typen von Shards verwendet:

- primäre Shards,
- synchrone Replikate,
- asynchrone Replikate.

Das primäre Shard empfängt alle Einfüge-, Aktualisierungs- und Entfernungsoperationen. Das primäre Shard fügt Replikate hinzu und entfernt Replikate, repliziert Daten in den Replikaten und verwaltet Commit- und Rollback-Operationen für Transaktionen.

Synchrone Replikate haben denselben Status wie das primäre Shard. Wenn ein primäres Shard Daten in einem synchronen Replikat repliziert, wird die Transaktion erst festgeschrieben, wenn sie im synchronen Replikat festgeschrieben wurde.

Asynchrone Replikate können denselben Status haben wie das primäre Shard. Wenn ein primäres Shard Daten in einem asynchronen Replikat repliziert, wartet das primäre Shard nicht auf die Festschreibung durch das asynchrone Replikat.

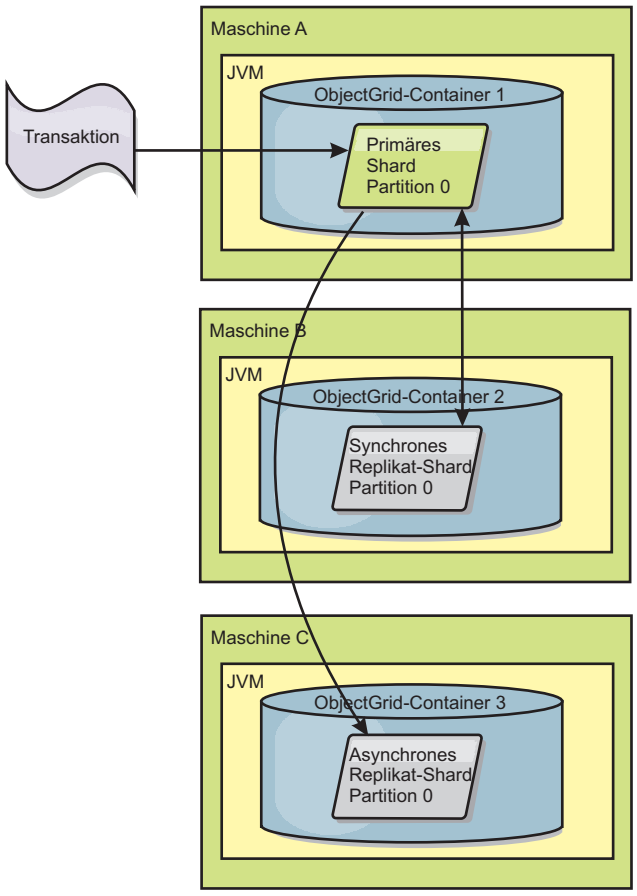


Abbildung 34. Kommunikationspfad zwischen einem primären Shard und einem Replikat-Shard

Mindestanzahl synchroner Replikat-Shards

Wenn ein primäres Shard die Festschreibung von Daten vorbereitet, prüft es, wie viele synchrone Replikat-Shards für die Festschreibung der Transaktion gestimmt

haben. Wenn die Transaktion im Replikat normal verarbeitet wird, stimmt das Replikat der Festschreibung zu. Wenn im synchronen Replikat etwas schiefgelaufen ist, stimmt das Replikat der Festschreibung nicht zu. Bevor ein primäres Shard Daten festschreibt, muss die Anzahl synchroner Replikat-Shards, die für die Festschreibung gestimmt haben, der minSyncReplica-Einstellung in der Implementierungsrichtlinie entsprechen. Wenn die Anzahl synchroner Replikat-Shards, die der Festschreibung zustimmen, zu niedrig ist, schreibt das primäre Shard die Transaktion nicht fest, und es tritt ein Fehler auf. Diese Aktion stellt sicher, dass die erforderliche Anzahl synchroner Replikate mit den korrekten Daten verfügbar ist. Synchrone Replikate, in denen Fehler aufgetreten sind, nehmen ihre Registrierung zurück, um ihren Zustand zu korrigieren. Weitere Informationen zum Zurücknehmen der Registrierung finden Sie im Abschnitt Wiederherstellung von Replikat-Shards.

Das primäre Shard löst eine Fehler des Typs "ReplicationVotedToRollbackTransactionException" aus, wenn die Anzahl synchroner Replikate, die der Festschreibung zugestimmt haben, zu niedrig ist.

Replikation und Loader

Normalerweise schreibt ein primäres Shard synchron über den Loader in eine Datenbank. Der Loader und die Datenbank sind immer synchronisiert. Wenn das primäre Shard von einem Replikat-Shard übernommen wird, sind die Datenbank und der Loader möglicherweise nicht mehr synchronisiert. Beispiel:

- Das primäre Shard kann die Transaktion an das Replikat senden und dann vor der Festschreibung der Daten in der Datenbank ausfallen.
- Das primäre Shard kann die Daten in der Datenbank festschreiben und dann vor dem Senden der Daten an das Replikat ausfallen.

Beide Fälle führen dazu, dass das Replikat mit einer Transaktion gegenüber der Datenbank im Vorlauf bzw. im Rückstand ist. Diese Situation ist nicht akzeptabel. eXtreme Scale verwendet ein spezielles Protokoll und einen Vertrag mit der Loader-Implementierung, um dieses Problem ohne zweiphasige Festschreibung zu lösen. Das Protokoll folgt:

Seite des primären Shard

- Transaktion zusammen mit den vorherigen Transaktionsergebnissen senden.
- In die Datenbank schreiben und versuchen, die Transaktion festzuschreiben.
- Wenn die Datenbank festschreibt, dann in eXtreme Scale festschreiben. Wenn die Datenbank nicht festschreibt, Transaktion rückgängig machen.
- Ergebnis aufzeichnen.

Replikatseite

- Transaktion empfangen und puffern.
- Für alle Ergebnisse, die mit der Transaktion geändert werden, die gepufferten Transaktionen festschreiben und alle rückgängig gemachten Transaktionen verwerfen.

Replikatseite bei Failover

- Für alle gepufferten Transaktionen die Transaktionen dem Loader bereitstellen, und der Loader versucht, die Transaktionen festzuschreiben.
- Der Loader muss so geschrieben sein, dass jede Transaktion idempotent ist.

- Wenn die Transaktion bereits in der Datenbank vorhanden ist, führt der Loader keine Operation durch.
- Wenn die Transaktion noch nicht in der Datenbank vorhanden ist, wendet der der Loader die Transaktion an.
- Nachdem alle Transaktionen verarbeitet wurden, kann das neue primäre Shard mit der Bearbeitung der Anforderungen beginnen.

Dieses Protokoll stellt sicher, dass die Datenbank denselben Stand wie das neue primäre Shard hat.

Shard-Zuordnung: primäre Shards und Replikate

Der Katalogservice ist für die Verteilung von Shards verantwortlich. Jedes ObjectGrid hat eine Reihe von Partitionen, die jeweils ein primäres Shard und einen optionalen Satz an Replikat-Shards haben. Replikat-Shards und primäre Shards für eine Partition werden vom Katalogservice nicht in demselben Container platziert. Der Katalogservice platziert Replikat- und primäre Shards nicht in Containern, die dieselbe IP-Adresse haben (sofern sich die Konfiguration nicht im Entwicklungsmodus befindet). Er verteilt die Shards gleichmäßig auf die verfügbaren Container.

Wenn ein neuer Container gestartet wird, ruft eXtreme Scale Shards aus relativ überladenen Containern in den neuen leeren Container ab. Mit diesem Verhalten kann eXtreme Scale seine wesentliche Elastizität aufbauen und aufrecht erhalten. Die Elastizität manifestiert sich in der leistungsfähigen Horizonalskalierungsfähigkeit - vorwärts und rückwärts.

Horizontale Vorwärtsskalierung

Horizontale Vorwärtsskalierung bedeutet Folgendes: Wenn einem eXtreme-Scale-Grid zusätzliche Java Virtual Machines oder Container hinzugefügt werden, versucht eXtreme Scale, vorhandene Shards (primäre Shards oder Replikate) aus dem alten JVM-Satz in den neuen JVM-Satz zu verschieben. Durch diese Verschiebung kann das Grid erweitert werden, um den Prozessor, das Netz und den Speicher der neu hinzugefügten JVMs zu nutzen. Bei dieser Verschiebung wird auch versucht, ein Gleichgewicht im Grid herzustellen und sicherzustellen, dass das Datenvolumen gleichmäßig auf alle JVMs im Grid verteilt wird. Je größer das Grid wird, desto keiner ist der Teil des Gesamt-Grids, der auf jeden einzelnen Server fällt. eXtreme Scale geht davon aus, dass die Daten gleichmäßig auf die Partitionen verteilt sind. Diese Erweiterung ermöglicht eine horizontale Vorwärtsskalierung.

Horizontale Rückskalierung

Horizontale Rückskalierung bedeutet Folgendes: Wenn eine JVM ausfällt, versucht eXtreme Scale, den Schaden zu beheben. Besitzt die ausgefallene JVM ein Replikat, ersetzt eXtreme Scale das verloren gegangene Replikat durch Erstellung eines neuen Replikats in einer noch aktiven JVM. Besitzt die ausgefallene JVM ein primäres Shard, sucht eXtreme Scale das am besten geeignete Replikat in den noch aktiven JVMs und stuft dieses Replikat als neues primäres Shard hoch. Anschließend ersetzt eXtreme Scale das hochgestufte Replikat durch ein neues Replikat, das in den verbleibenden Servern erstellt wird. Zur Gewährleistung der Skalierbarkeit, behält eXtreme Scale die Replikatanzahl für Partitionen beim Ausfall von Servern bei.

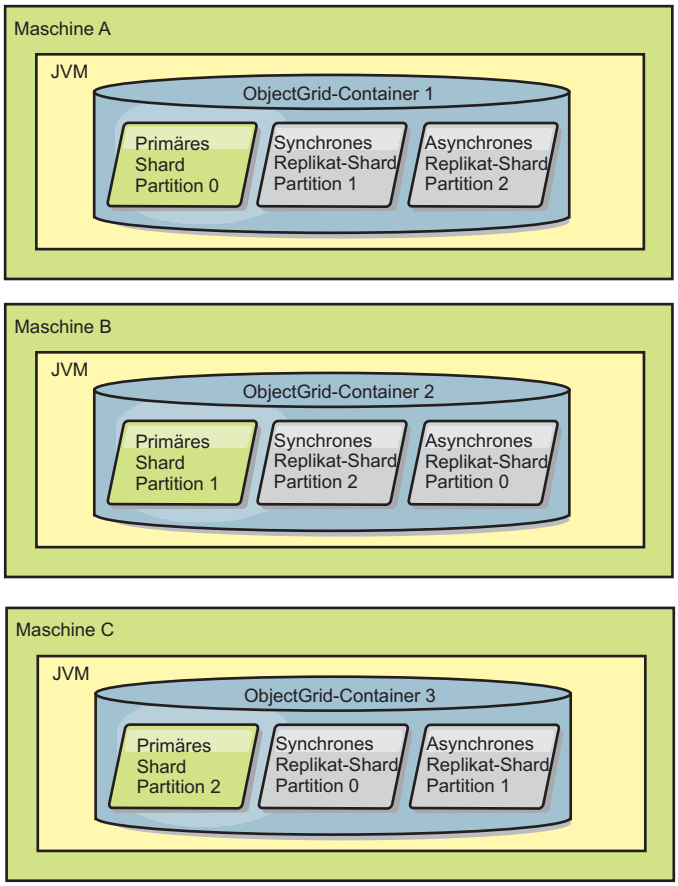


Abbildung 35. Verteilung eines ObjectGrid-MapSets über eine Implementierungsrichtlinie mit 3 Partitionen mit einem minSyncReplicas-Wert von 1, einem maxSyncReplicas-Wert von 1 und einem maxAsyncReplicas-Wert von 1

Aus Replikaten lesen

Sie können MapSets so konfigurieren, dass ein Client aus einem Replikat lesen kann und nicht nur auf die primären Shards beschränkt ist.

Häufig kann es von Vorteil sein, Replikate nicht nur einfach als potenzielle primäre Shards für Fehlerfälle einzusetzen. MapSets können beispielsweise so konfiguriert werden, dass Leseoperationen an Replikate weitergeleitet werden können, indem die Option "replicaReadEnabled" im MapSet auf "true" gesetzt wird. Die Standardeinstellung ist "false".

Weitere Informationen zum Element "MapSet" finden Sie im Abschnitt zur XML-Deskriptordatei für Implementierungsrichtlinien im *Administratorhandbuch*.

Durch die Aktivierung des Lesens von Replikaten kann die Leistung verbessert werden, indem Leseanforderungen an mehrere Java Virtual Machines verteilt werden. Wenn Sie diese Option nicht aktivieren, werden alle Leseanforderungen, wie z. B. die Methoden "ObjectMap.get" und "Query.getResultIterator", an das primäre Shard weitergeleitet. Wenn replicaReadEnabled auf "true" gesetzt ist, können einige Get-Anforderungen veraltete Daten zurückgeben, so dass eine Anwendung, die diese Option verwendet, in der Lage sein muss, diese Möglichkeit zu tolerieren. Cachefehler treten jedoch nicht auf. Wenn die Daten nicht im Replikat enthalten sind, wird die Get-Anforderung an das primäre Shard umgeleitet und wiederholt.

Die Option "replicaReadEnabled" kann mit synchroner und asynchroner Replikation verwendet werden.

Lastausgleich mit Replikaten

Der Lastausgleich mit Replikaten wird gewöhnlich nur verwendet, wenn Clients Daten zwischenspeichern, die sich ständig ändern oder wenn die Clients pessimistisches Sperren verwenden.

eXtreme Scale sendet, sofern nicht anders konfiguriert, alle Lese- und Schreibanforderungen an den primären Server für eine bestimmte Replikationsgruppe. Der primäre Server muss alle Anforderungen von Clients bearbeiten. Sie können konfigurieren, dass Leseanforderungen auch an Replikate des primären Servers gesendet werden. Durch das Senden von Leseanforderungen an die Replikate kann die Last der Leseanforderungen auf mehrere Java Virtual Machines (JVM) verteilt werden. Die Verwendung von Replikaten für Leseanforderungen kann jedoch zu inkonsistenten Antworten führen.

Der Lastausgleich mit Replikaten wird gewöhnlich nur verwendet, wenn Clients Daten zwischenspeichern, die sich ständig ändern oder wenn die Clients pessimistisches Sperren verwenden.

Wenn sich die Daten ständig ändern und anschließend im nahen Cache des Clients ungültig gemacht werden, sollte der primäre Server daraufhin eine relativ hohe Rate von get-Anforderungen von Clients sehen. Im pessimistischen Sperrmodus ist kein lokaler Cache vorhanden, also werden alle Anforderungen an den primären Server gesendet.

Wenn die Daten relativ statisch sind oder der pessimistische Modus nicht verwendet wird, hat das Senden von Leseanforderungen an das Replikat keine erheblichen Auswirkungen auf die Leistung. Die Häufigkeit der get-Anforderungen von Clients mit Caches, die voll von Daten sind, ist nicht sehr hoch.

Wenn ein Client zum ersten Mal gestartet wird, ist sein naher Cache leer. Cacheanforderungen an den leeren Cache werden an den primären Server weitergeleitet. Nach und nach werden Daten in den Clientcache gestellt, so dass die Anforderungslast abnimmt. Wenn sehr viele Clients gleichzeitig gestartet werden, kann die Last erheblich und das Senden von Leseanforderungen an das Replikat eine angemessene Leistungsoption sein.

Lebenszyklus-, Wiederherstellungs- und Fehlerereignisse

Für die Unterstützung der Replikation setzt sich der Lebenszyklus von Shards aus verschiedenen Stadien und Ereignissen zusammen. Der Lebenszyklus eines Shards setzt sich aus dem Onlinebringen, der Laufzeit, dem Beenden, potenziellen Failover und der Fehlerbehandlung zusammen. Shards können als Reaktion auf einen geänderten Serverstatus von einem Replikat-Shard auf ein primäres Shard hochgestuft werden.

Lebenszyklusereignisse

Wenn primäre Shards und Replikat-Shards verteilt und gestartet werden, finden verschiedene Ereignisse statt, um die Shards online zu bringen und in den Empfangsmodus zu versetzen.

Primäres Shard

Der Katalogservice verteilt ein primäres Shard für eine Partition. Außerdem verteilt der Katalogservice die Positionen der primären Shards gleichmäßig und leitet das Failover für die primären Shards ein.

Wenn ein Shard zu einem primären Shard wird, erhält es eine Liste mit Replikaten vom Katalogservice. Das neue primäre Shard erstellt eine Replikatgruppe und registriert alle Replikate.

Wenn das primäre Shard bereit ist, wird eine Nachricht in der Datei `SystemOut.log` für den Container, in dem das Shard ausgeführt wird, protokolliert, in der darauf hingewiesen wird, dass das Shard für das Geschäft bereit ist. In der Nachricht für die Geschäftsbereitschaft (oder die Nachricht CWOBJ1511I) sind der Map-Name, der Name des MapSets und die Partitionsnummer des gestarteten primären Shards aufgelistet.

```
CWOBJ1511I: Map-Name:MapSet-Name:Partitionsnummer (primär) ist für Business bereit.
```

Weitere Informationen zur Verteilung der Shards durch den Katalogservice finden Sie im Abschnitt „Shard-Zuordnung: primäre Shards und Replikate" auf Seite 136.

Replikat-Shard

Replikat-Shards werden hauptsächlich vom primären Shard gesteuert, sofern das Replikat-Shard keine Probleme feststellt. Während eines normalen Lebenszyklus ist das primäre Shard für das Verteilen, Registrieren und Zurücknehmen der Registrierung eines Replikat-Shards zuständig.

Wenn ein primäres Shard ein Replikat-Shard initialisiert, wird eine Nachricht im Protokoll angezeigt, die beschreibt, wo das Replikat ausgeführt wird, um so anzuzeigen, dass das Replikat-Shard verfügbar ist. In der Nachricht für die Geschäftsbereitschaft (oder die Nachricht CWOBJ1511I) sind der Map-Name, der Name des MapSets und die Partitionsnummer des Replikat-Shards aufgelistet. Diese Nachricht sehen Sie im Folgenden:

```
CWOBJ1511I: Map-Name:MapSet-Name:Partitionsnummer (synchrones Replikat) ist für Business bereit.
```

oder

```
CWOBJ1511I: Map-Name:MapSet-Name:Partitionsnummer (asynchrones Replikat) ist für Business bereit.
```

Asynchrones Replikat-Shard: Ein asynchrones Replikat-Shard fragt Daten vom primären Shard ab. Das Replikat passt die Abfragetaktung automatisch an, wenn es keine Daten vom primären Shard empängt, was darauf hinweist, das die Verbindung zum primären Shard blockiert ist. Die Taktung wird auch angepasst, wenn das Replikat einen Fehler empfängt, der auf einen Ausfall des primären Shards oder auf ein Netzproblem hinweist.

Wenn das asynchrone Replikat mit der Replikation beginnt, gibt es die folgende Nachricht in seiner Datei "SystemOut.log" aus. Die Nachricht kann pro CWOBJ1511-Nachricht mehrfach ausgegeben werden. Sie wird erneut ausgegeben, wenn das Replikat eine Verbindung zu einem anderen primären Shard herstellt oder wenn Schablonen-Maps hinzugefügt werden.

```
CWOBJ1543I: Das asynchrone Replikat objectGrid-Name:MapSet-Name:Partitionsnummer wurde gestarter
bzw. setzt die Replikation über das primäre Shard fort. Replikation für die Maps: [Map-Namen]
```

Synchrones Replikat-Shard: Wenn das synchrone Replikat-Shard gestartet wird, ist es noch nicht im Peer-Modus. Wenn sich ein Replikat-Shard im Peer-Modus befindet, empfängt es Daten vom primären Shard, sobald Daten beim primären Shard

eintreffen. Vor dem Wechsel in den Peer-Modus benötigt das Replikat-Shard eine Kopie aller vorhandenen Daten im primären Shard.

Das synchrone Replikat kopiert ähnlich wie ein asynchrones Replikat Daten vom primären Shard, indem es Daten abfragt. Wenn es die vorhandenen Daten vom primären Shard kopiert, wechselt es in den Peer-Modus und empfängt die Daten, sobald das primäre Shard die Daten empfängt.

Nach dem Wechsel eines Replikat-Shards in den Peer-Modus gibt das Replikat eine Nachricht in seiner Datei "SystemOut.log" aus. Die angegebene Zeit bezieht sich auf die Zeit, die das Replikat-Shard benötigt hat, um seine Anfangsdaten vom primären Shard zu erhalten. Die angezeigte Zeit kann null oder sehr gering sein, wenn das primäre Shard keine zu replizierenden Daten enthält. Die Nachricht kann pro CWOBJ1511-Nachricht mehrfach ausgegeben werden. Sie wird erneut ausgegeben, wenn das Replikat eine Verbindung zu einem anderen primären Shard herstellt oder wenn Schablonen-Maps hinzugefügt werden.

```
CWOBJ1526I: Replikat ObjectGrid-Name:Mapset-Name:Partitionsnummer:Map-Name wechselt
in X Sekunden in den Peer-Modus.
```

Wenn das synchrone Replikat-Shards im Peer-Modus arbeitet, muss das primäre Shards Transaktionen auf alle im Peer-Modus arbeitenden synchronen Replikate kopieren. Das synchrone Replikat-Shard hat dieselbe Version der Daten wie das primäre Shard. Wenn in der Implementierungsrichtlinie eine minimale Anzahl synchroner Replikate oder minSync definiert ist, muss diese Anzahl synchroner Replikate der Festschreibung zustimmen, bevor die Transaktion erfolgreich auf dem primären Shard festgeschrieben werden kann.

Wiederherstellungsereignisse

Die Replikation ist für die Wiederherstellung nach Ausfällen und Fehlereignissen bestimmt. Wenn ein primäres Shard ausfällt, wird seine Arbeit von einem anderen Replikat übernommen. Treten Fehler bei den Replikat-Shards auf, versucht das Replikat-Shard, eine Recovery durchzuführen. Der Katalogservice steuert die Verteilung und Transaktionen der neuen primären Shards bzw. neuen Replikat-Shards.

Replikat-Shards wird zu einem primären Shard

Ein Replikat-Shard kann aus zwei Gründen zu einem primären Shard werden. Das primäre Shard wird gestoppt oder fällt aus, oder es wurde eine Entscheidung im Sinne der Lastverteilung getroffen, die ein Verschieben des vorherigen primären Shards an eine neue Position erfordert.

Der Katalogservice wählt ein neues primäres Shards unter den vorhandenen synchronen Replikat-Shards aus. Wenn ein primäres Shard verschoben werden muss und keine Replikate vorhanden sind, wird ein temporäres Replikat für den Übergang verwendet. Das neue primäre Shard registriert alle vorhandenen Replikate und akzeptiert anschließend Transaktionen als das neue primäre Shard. Wenn die vorhandenen Replikat-Shards die richtige Datenversion haben, werden die aktuellen Daten beibehalten, wenn sich die Replikat-Shards beim neuen primären Shard registrieren. Asynchrone Replikate fragen das neue primäre Shard ab.

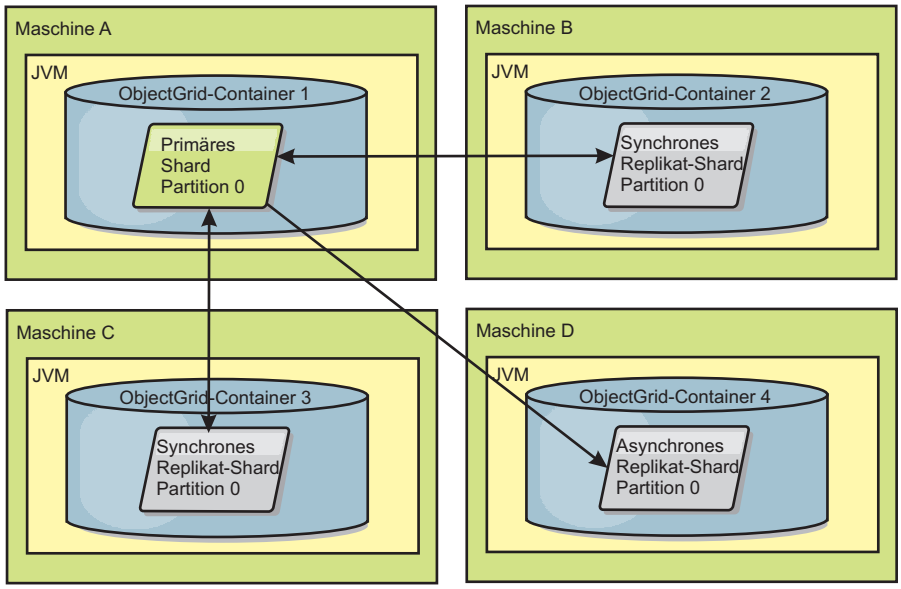


Abbildung 36. Beispielverteilung eines ObjectGrid-MapSets für die Partition "partition0". Die Implementierungsrichtlinie enthält den minSyncReplicas-Wert 1, den maxSyncReplicas-Wert 2 und den maxAsyncReplicas-Wert 1.

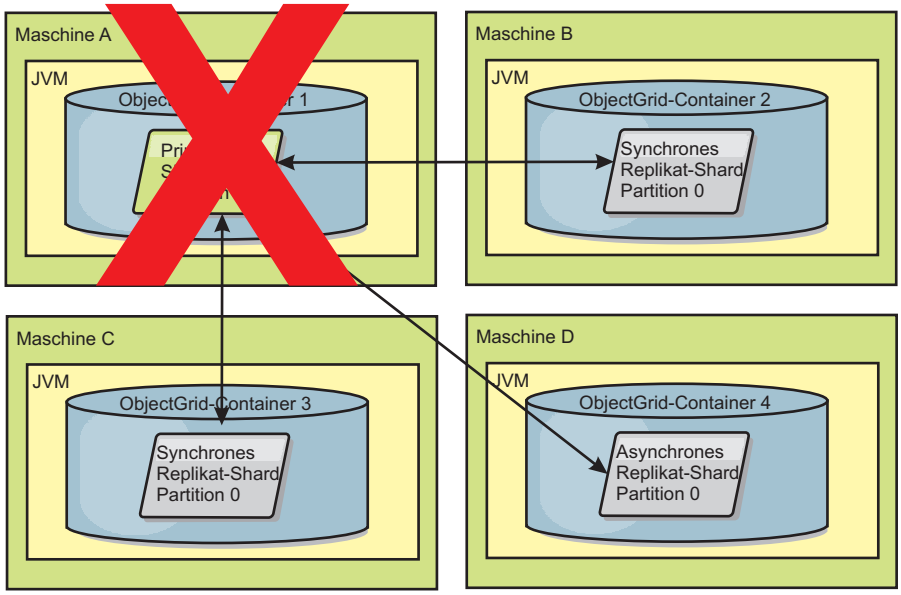


Abbildung 37. Container für das primäre Shard fällt aus

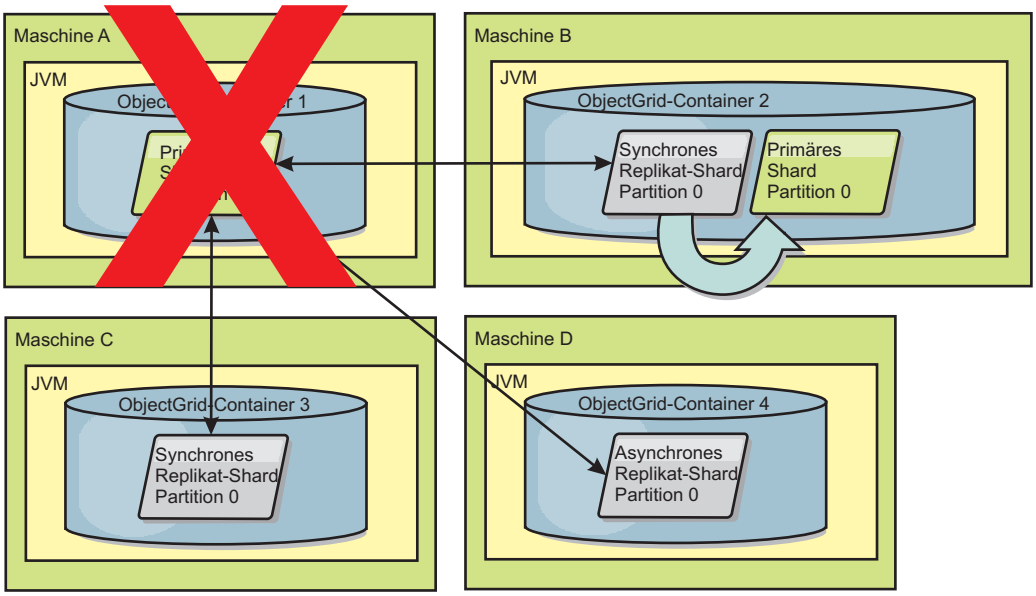


Abbildung 38. Synchrones Replikat-Shard in ObjectGrid-Container 2 wird zum primären Shard

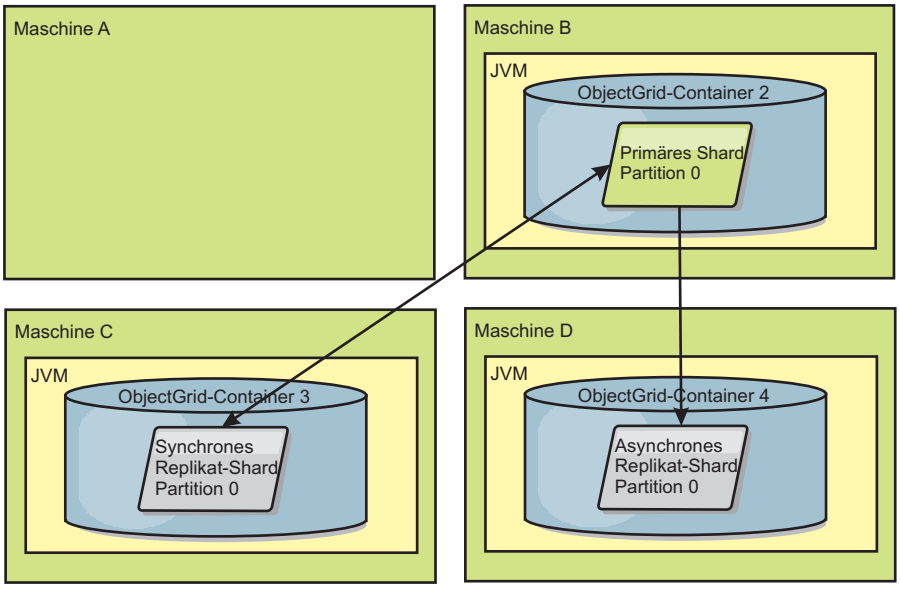


Abbildung 39. Maschine B enthält das primäre Shard. In Abhängigkeit von der Einstellung des Modus für automatische Reparatur und der Verfügbarkeit der Container wird ein neues synchrones Replikat-Shard auf einer Maschine erstellt oder nicht.

Wiederherstellung von Replikat-Shards

Ein synchrones Replikat-Shard wird vom primären Shard gesteuert. Wenn ein Replikat-Shard jedoch ein Problem feststellt, kann es ein Ereignis zur Zurücknahme der Registrierung auslösen, um den Status der Daten zu korrigieren. Das Replikat-Shard löscht die aktuellen Daten und ruft eine aktuelle Kopie vom primären Shard ab.

Wenn ein Replikat-Shard ein Ereignis zur Zurücknahme der Registrierung einleitet, gibt das Replikat eine Protokollnachricht aus:

```
CWOBJ1524I: Replikat-Listener
ObjectGrid-Name:MapSet-Name:Partition muss sich beim primären Shard registrieren.
Ursache: Aufgelistete Ausnahme
```

Wenn eine Transaktion während der Verarbeitung einen Fehler in einem Replikat-Shard verursacht, hat das Replikat-Shard einen unbekannten Status. Die Transaktion wird erfolgreich im primären Shard ausgeführt, aber im Replikat ist irgendein Fehler aufgetreten. Zur Behebung des Problems leitet das Replikat ein Ereignis zum Zurücknehmen der Registrierung ein. Mit einer neuen Kopie der Daten vom primären Shard kann das Replikat-Shard seine Verarbeitung fortsetzen. Tritt dasselbe Problem erneut auf, setzt das Replikat-Shard seine Versuche, die Registrierung zurückzunehmen, nicht fort. Weitere Einzelheiten finden Sie im Abschnitt „Fehlerereignisse".

Fehlerereignisse

Ein Replikat kann die Replikation von Daten einstellen, wenn es Fehlersituationen feststellt, die nicht behoben werden können.

Zu viele Registrierungsversuche

Wenn ein Replikat sich mehrfach erneut registriert, ohne die Daten erfolgreich festzuschreiben, wird das Replikat gestoppt. Das Stoppen verhindert, dass ein Replikat in eine Endlosschleife für die Registrierung eintritt. Standardmäßig wiederholt ein Replikat-Shard seinen Registrierungsversuch dreimal nacheinander, bevor es gestoppt wird.

Wenn sich ein Replikat-Shard zu oft neu registriert, gibt es die folgende Nachricht im Protokoll aus:

```
CWOBJ1537E: ObjectGrid-Name:MapSet-Name:Partition hat die zulässige Anzahl
erneuter Registrierungen (timesAllowed) ohne erfolgreiche Transaktionen überschritten.
```

Wenn die Wiederherstellung des Replikats durch erneute Registrierung nicht möglich ist, kann ein tiefgreifendes Problem bei den Transaktionen, die sich auf das Replikat-Shard beziehen, vorliegen. Ein mögliches Problem sind fehlende Ressourcen im Klassenpfad, wenn ein Fehler bei der Dekomprimierung der Schlüssel oder Werte aus der Transaktion auftritt.

Fehler beim Wechsel in den Peer-Modus

Wenn ein Replikat versucht, in den Peer-Modus zu wechseln und ein Fehler bei der Verarbeitung des vorhandenen Datenvolumens vom primären Shard (Prüfpunktdaten) auftritt, wird das Replikat beendet. Das Beenden verhindert, dass ein Replikat mit ungültigen Anfangsdaten gestartet wird. Da das Replikat dieselben Daten vom primären Shard empfängt, wenn es sich erneut registriert, findet keine Wiederholung statt.

Wenn ein Replikat-Shard nicht in den Peer-Modus wechseln kann, gibt es die folgende Nachricht im Protokoll aus:

```
CWOBJ1527W Replikat ObjectGrid-Name:MapSet-Name:Partition:Map-Name konnte nach numSeconds Sekunden nicht in den Peer-Modus wechseln.
```

Es wird eine weitere Nachricht im Protokoll angezeigt, die erläutert, warum das Replikat nicht in den Peer-Modus wechseln konnte.

Fehler bei Wiederherstellung nach Neuregistrierung oder beim Wechsel in Peer-Modus

Wenn ein Replikat die erneute Registrierung nicht durchführen oder nicht in den Peer-Modus wechseln kann, bleibt das Replikat so lange inaktiviert, bis ein neues Verteilungsereignis stattfindet. Ein neues Verteilungsereignis kann das Starten oder Stoppen eines neuen Servers sein. Sie können ein Verteilungsereignis auch mit der Methode "triggerPlacement" in der MBean "PlacementServiceMBean" einleiten.

Routing an bevorzugte Zonen

Mit dem Routing an bevorzugte Zonen kann eXtreme Scale Transaktionen auf der Basis Ihrer Spezifikationen an Zonen weiterleiten.

WebSphere eXtreme Scale bietet Ihnen sehr viele Steuerungsmöglichkeiten in Bezug auf die Verteilung der Shards eines ObjectGrids.

Das Routing an bevorzugte Zonen ermöglicht eXtreme-Scale-Clients, eine Präferenz für eine bestimmte Zone oder eine Gruppe von Zonen anzugeben. eXtreme Scale versucht, Clientransaktionen an die bevorzugten Zonen weiterzuleiten, bevor andere Zonen ausgewählt werden.

Voraussetzungen für das Routing an bevorzugte Zonen

Vor der Entscheidung für das Routing an bevorzugte Zonen müssen verschiedene Faktoren berücksichtigt werden. Stellen Sie sicher, dass die Anwendung die Voraussetzungen Ihres Szenarios zu erfüllen.

Es ist eine containerbezogene Partitionsverteilung erforderlich, um das Routing an bevorzugte Zonen nutzen zu können. Diese Verteilungsstrategie eignet sich gut für Anwendungen, die Sitzungsdaten im ObjectGrid speichern. Die Standardstrategie für die Verteilung von Partitionen in WebSphere eXtreme Scale ist "fixed-partition". Bei der Festschreibung einer Transaktion wird ein Hash-basierter Algorithmus für die Schlüssel verwendet, um zu bestimmen, welche Partition das Schlüssel/Wert-Paar der Map aufnimmt, wenn die Verteilung mit festen Partitionen verwendet wird.

Bei der containerbezogenen Verteilung werden Ihre Daten beim Festschreiben einer Transaktion über das SessionHandle-Objekt einer zufällig ausgewählten Partition zugeordnet. Sie müssen das SessionHandle-Objekt wiederherstellen können, um Ihre Daten aus dem ObjectGrid abrufen zu können.

Da Sie Zonen verwenden können, um eine bessere Kontrolle darüber zu haben, wo sich primäre Shards und ihre Replikate in der Domäne befinden, ist eine Implementierung mit mehreren Zonen vorteilhaft, wenn Ihre Daten auf mehrere physische Positionen verteilt sind. Die geographische Trennung von primären Shards und Replikaten ist eine Methode sicherzustellen, dass der katastrophale Verlust eines Rechenzentrums keine Auswirkung auf die Verfügbarkeit der Daten hat.

Wenn die Daten in einer Topologie mit mehreren Zonen verteilt werden, ist es wahrscheinlich, dass auch die Clients in der Topologie verteilt werden. Das Routing von Clients an ihre lokalen Zonen oder Rechenzentren hat den offensichtlichen Vorteil einer kürzeren Netzlatenzzeit. Nutzen Sie die Vorteile dieses Szenarios, wenn dies möglich ist.

Topologie für das Routing an bevorzugte Zonen konfigurieren

Stellen Sie sich das folgende Szenario vor: Sie haben zwei Rechenzentren: Chicago und London. Zur Minimierung der Clientantwortzeiten möchten Sie, dass Clients Daten aus ihrem lokalen Rechenzentrum lesen und ihre Daten auch dorthin schreiben.

Primäre ObjectGrid-Shards müssen in jedem Rechenzentrum verteilt werden, so dass Transaktionen lokal von jeder Position aus geschrieben werden können. Außerdem müssen die Clients die Zonen kennen, um Daten an die lokale Zone zu senden.

Bei der containerbezogenen Verteilung werden neue primäre Shards auf jeden gestarteten Container verteilt. Replikate werden entsprechend den Zonen- und Verteilungsregeln in der Implementierungsrichtlinie verteilt. Standardmäßig wird ein Replikat einer anderen Zone als das primäre Shard zugeteilt. Sehen Sie sich die folgende Implementierungsrichtlinie für dieses Szenario an:

```
<?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="universe">
  <mapSet name="mapSet1" placementStrategy="PER_CONTAINER"
   numberOfPartitions="3" maxAsyncReplicas="1">
   <map ref="planet" />
  </mapSet>
 </objectgridDeployment>
</deploymentPolicy>
```

Jeder Container, der mit der Implementierungsrichtlinie gestartet wird, erhält 3 neue primäre Shards. Jedes primäre Shard hat 1 asynchrones Replikat. Starten Sie jeden Container mit dem entsprechenden Zonennamen. Verwenden Sie den Parameter "-zone", wenn Sie Ihre Container mit dem Script "startOgServer" starten.

Für einen Containerserver in Chicago:

- UNIX Linux

```
startOgServer.sh s1 -objectGridFile ../xml/universeGrid.xml
-deploymentPolicyFile ../xml/universeDp.xml
-catalogServiceEndpoints MyServer1.company.com:2809
-zone Chicago
```

- Windows

```
startOgServer.bat s1 -objectGridFile ../xml/universeGrid.xml
-deploymentPolicyFile ../xml/universeDp.xml
-catalogServiceEndpoints MyServer1.company.com:2809
-zone Chicago
```

Wenn Ihre Container in WebSphere Application Server ausgeführt werden, müssen Sie eine Knotengruppe erstellen und diese mit dem Präfix "ReplicationZone" benennen. Server, die auf Knoten in solchen Knotengruppen ausgeführt werden, werden an die entsprechende Zone verteilt. Server, die auf einem Knoten in Chicago ausgeführt werden, könnten beispielsweise in einer Knotengruppe mit dem Namen "ReplicationZoneChicago" enthalten sein.

Primäre Shards für die Zone "Chicago" haben Replikate in der Zone "London". Primäre Shards in der Zone "London" haben Replikate in der Zone "Chicago".

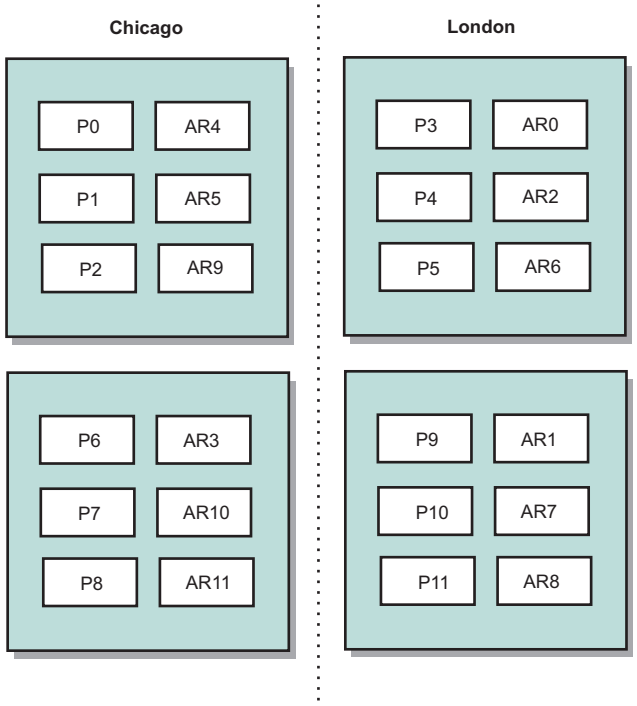


Abbildung 40. Primäre Shards und Replikate in Zonen

Legen Sie die bevorzugten Zonen für die Clients fest. Diese Informationen können auf verschiedene Weise bereitgestellt werden. Die direkteste Methode ist die Bereitstellung einer Clienteigenschaftendatei für Ihre Client-JVM. Erstellen Sie eine Datei mit dem Namen "objectGridClient.properties", und stellen Sie sicher, dass sie im Klassenpfad enthalten ist. Weitere Informationen finden Sie Abschnitt Artikel zur Clienteigenschaftendatei im *Administratorhandbuch*.

Fügen Sie die Eigenschaft "preferZones" in die Datei ein. Setzen Sie den Eigenschaftswert auf die entsprechende Zone. Clients in Chicago sollten folgende Definition in der Datei "objectGridClient.properties" enthalten:

```
preferZones=Chicago
```

Die Eigenschaftendatei für Clients in London sollte Folgendes enthalten:

```
preferZones=London
```

Diese Eigenschaft weist jeden einzelnen Client an, Transaktionen an seine lokale Zone weiterzuleiten, sofern dies möglich ist. Daten, die in das primäre Shard in der lokalen Zone eingefügt werden, werden asynchron in der fremden Zone repliziert.

SessionHandle für das Routing an die lokale Zone verwenden

Die containerbezogene Verteilungsstrategie verwendet keinen Hash-basierten Algorithmus, um die Position der Schlüssel/Wert-Paare im ObjectGrid zu bestimmen. Ihr ObjectGrid muss SessionHandle-Objekte verwenden, um sicherzustellen, dass Transaktionen an die richtige Position weitergeleitet werden, wenn diese Verteilungsstrategie verwendet wird. Wenn eine Transaktion festgeschrieben wird, wird ein SessionHandle-Objekt an das Session-Objekt gebunden, sofern noch keines definiert wurde. Alternativ kann das SessionHandle-Objekt über den Aufruf von "Ses-

sion.getSessionHandle" vor der Festschreibung der Transaktion an das Session-Objekt gebunden werden. Das folgende Code-Snippet zeigt, wie ein SessionHandle-Objekt vor der Festschreibung der Transaktion an das Session-Objekt gebunden wird:

```
Session ogSession = objectGrid.getSession();

// SessionHandle-Objekt binden
SessionHandle sessionHandle = ogSession.getSessionHandle();

ogSession.begin();
ObjectMap map = ogSession.getMap("planet");
map.insert("planet1", "mercury");

// Transaktion wird an die vom SessionHandle-Objekt angegebene Partition weitergeleitet
ogSession.commit();
```

Angenommen, der vorherige Code wird in einem Client im Rechenzentrum Chicago ausgeführt. Da das Attribut "preferZones" für diesen Client auf "Chicago" gesetzt wurde, wird diese Transaktion an eine der primären Partitionen in der Zone "Chicago" weitergeleitet (Partition 0, 1, 2, 6, 7 oder 8).

Dieses SessionHandle-Objekt ist der Pfad zurück zu der Partition, in der diese festgeschriebenen Daten gespeichert sind. Das SessionHandle-Objekt muss wiederverwendet oder wiederhergestellt und im Session-Objekt gesetzt werden, um zu der Partition mit den festgeschriebenen Daten zurückzugelangen.

```
ogSession.setSessionHandle(sessionHandle);
ogSession.begin();

// Zurückgegebener Wert ist "mercury "
String value = map.get("planet1");
ogSession.commit();
```

Da diese Transaktion das SessionHandle-Objekt wiederverwendet, das während der insert-Transaktion erstellt wurde, wird die get-Transaktion an die Partition weitergeleitet, die die eingefügten Daten enthält. Diese Transaktion kann die zuvor eingefügten Daten nicht abrufen, wenn das SessionHandle-Objekt nicht gesetzt ist.

Auswirkung von Container- und Zonenfehlern auf das Routing an bevorzugte Zonen

Alle Transaktionen eines Clients mit der gesetzten Eigenschaft "preferZones" werden unter normalen Umständen an die angegebenen Zonen weitergeleitet. Der Verlust eines Containers führt jedoch dazu, dass ein Replikat in einer fremden Zone in ein primäres Shard hochgestuft wird. Ein Client, der zuvor Daten an Partitionen in der lokalen Zone weitergeleitet hat, kann jetzt zum Routing an die fremde Zone gezwungen sein, um zuvor eingefügte Daten abzurufen.

Stellen Sie sich das folgende Szenario vor: Ein Container in der Zone "Chicago" geht verloren. Dieser Container enthält die primären Shards für die Partitionen 0, 1 und 2. Die neuen primären Shards für diese Partitionen befinden sich nun in der Zone "London", da diese Zone die Replikate für diese Partitionen enthält.

Jeder Chicago-Client, der ein SessionHandle-Objekt verwendet, das auf eine der übernommenen Partitionen zeigt, wird jetzt nach London weitergeleitet. Chicago-Clients, die die neuen SessionHandle-Objekte verwenden, werden an Chicago-basierte primäre Shards weitergeleitet.

Wenn die gesamte Zone "Chicago" verloren geht, werden alle Replikate in der Zone "London" zu primären Shards. In diesem Fall werden die Transaktionen aller Chicago-Clients nach London weitergeleitet.

Multmaster-Replikationstopologie (AP)

Wenn Sie das asynchrone Multimaster-Replikationsfeature verwenden, können zwei oder mehr Grids exakte Spiegel voneinander werden. Diese Spiegelung wird durch asynchrone Replikation zwischen Links erreicht werden, die die Grids miteinander verbinden. Jedes Grid ist in einer vollständig unabhängigen "Domäne" enthalten, hat einen eigenen Katalogservice, eigene Containerserver und einen eindeutigen Domänennamen. Mit dem asynchronen Multimaster-Replikationsfeature können Sie Links verwenden, um eine Sammlung dieser Domänen miteinander zu verbinden, und diese Domänen anschließend durch Replikation über die Links synchronisieren. eXtreme Scale ermöglicht Ihnen, nahezu jede Topologie zu erstellen, weil die Definition der Links zwischen den Domänen Ihnen überlassen bleibt.

Domänen: Grids mit speziellen Merkmalen

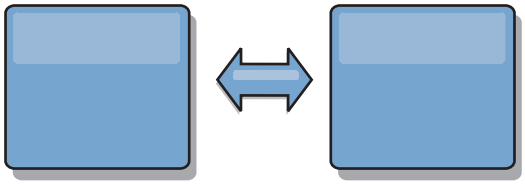
In Multimaster-Replikationstopologien verwendete Grids werden als *Domänen* bezeichnet. Jede Domäne muss die folgenden Merkmale haben:

- Hat einen privaten Katalogservice mit einem eindeutigen Domänennamen
- Hat denselben Grid-Namen wie andere Grids in der Domäne
- Hat dieselbe Anzahl an Partitionen wie andere Grids in der Domäne
- Ist ein Grid vom Typ FIXED_PARTITION (Grids vom Typ PER_CONTAINER können nicht repliziert werden)
- Hat dieselbe Anzahl an Partitionen (kann, muss aber nicht dieselbe Anzahl und dieselben Typen von Replikaten haben)
- Hat dieselben Replikationsdatentypen wie andere Grids in der Domäne
- Hat dieselben MapSet-Namen, Map-Namen und dynamischen Map-Schablonen wie andere Grids in der Domäne

Nachdem die Domänen in der Topologie gestartet wurden, werden alle Grids mit den zuvor beschriebenen Merkmalen repliziert. Beachten Sie, dass die Replikationsrichtlinien der Grids ignoriert werden.

Links, die Domänen verbinden

Eine Grid-Replikationsinfrastruktur ist ein verbundener Graph von Domänen mit bidirektionalen Links zwischen den Domänen. Über einen Link können zwei Domänen Daten miteinander austauschen. Die einfachste Topologie ist beispielsweise ein Domänenpaar mit einem einzigen Link zwischen ihnen. Die Domänen werden von links aus gesehen beginnend mit "A", dann "B" usw. benannt. Der Link kann ein Weitverkehrsnetz (WAN) durchqueren und große Distanzen überwinden. Wenn der Link unterbrochen wird, können trotzdem Änderungen an den Daten in den beiden Domänen vorgenommen werden. Die Änderungen werden später abgeglichen, wenn der Link die Domänen wieder verbindet. Links versuchen nach der Unterbrechung der Netzverbindung automatisch, die Verbindung wiederherzustellen.

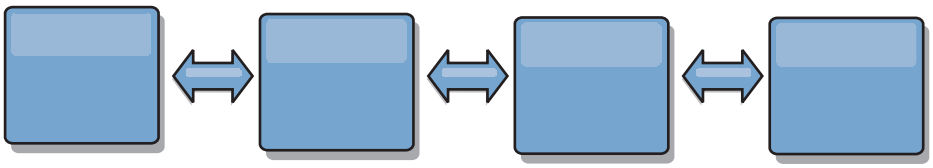


Nach dem Einrichten der Links versucht eXtreme Scale zuerst, alle Domänen abzugleichen, und dann, diese identischen Bedingungen beizubehalten, wenn Änderun-

gen in den Domänen stattfinden. eXtreme Scale hat das Ziel, dass jede Domäne eine exakte Spiegelung jeder anderen Domäne ist, die über die Links verbunden ist. Die Replikationslinks zwischen den Domänen stellen sicher, dass alle Änderungen, die in einer Domäne vorgenommen werden, in die anderen Domänen kopiert werden.

Reihentopologien

Obwohl die Reihentopologie zu den einfachsten Topologien gehört, veranschaulicht sie einige Qualitäten der Links. Zunächst ist es nicht erforderlich, dass eine Domäne direkte mit jeder anderen Domäne verbunden ist, damit sie Änderungen empfängt. Domäne B übernimmt Änderungen von Domäne A. Domäne C empfängt Änderungen von Domäne A über Domäne B, die die Domänen A und B verbindet. Domäne D empfängt Änderungen von den anderen Domänen über Domäne C. Auf diese Weise kann die Quelle der Änderungen von der Verteilung der Änderungen entlastet werden.



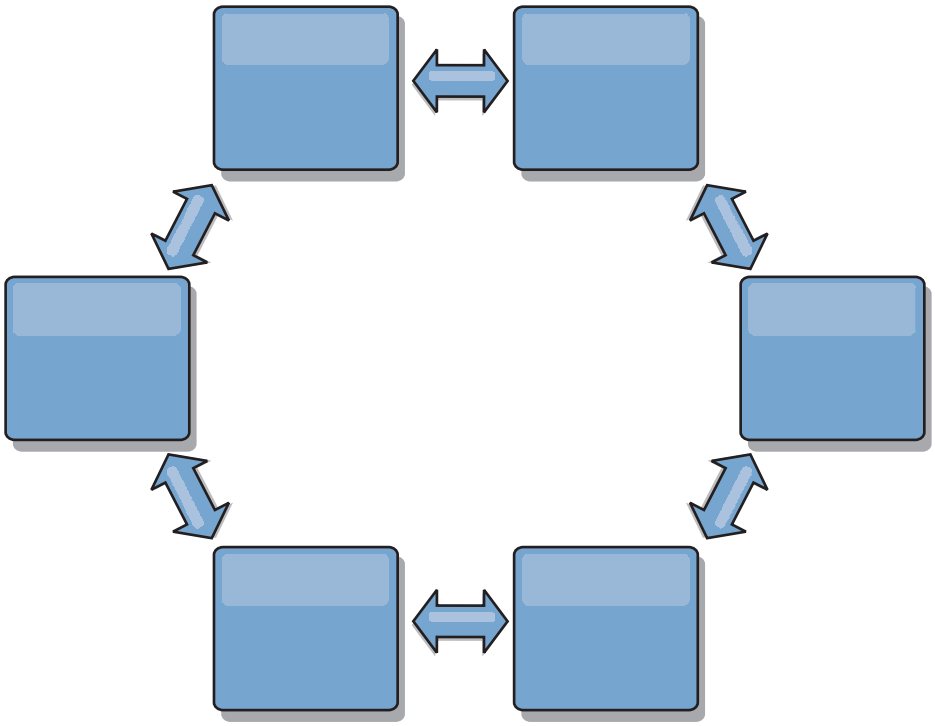
Wenn Domäne C ausfällt, treten die folgenden Ereignisse ein:

1. Domäne D ist verwaist, bis Domäne C erneut gestartet wird.
2. Domäne C synchronisiert sich selbst mit Domäne B, die eine Kopie von Domäne A ist.
3. Domäne D verwendet Domäne C, um sich mit den Änderungen in den Domänen A und B zu synchronisieren, die vorgenommen wurden, während Domäne D verwaist war (aufgrund des Ausfalls von Domäne C).

Am Ende sind die Domänen A, B, C und D wieder identisch.

Ringtopologien

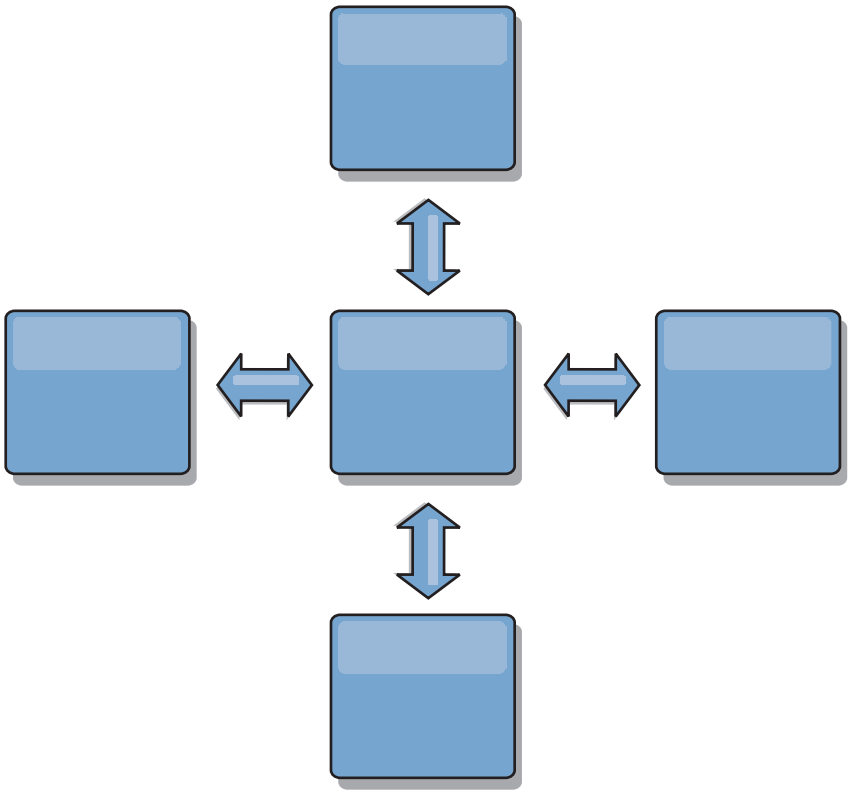
Ringtopologien sind ein Beispiel für eine Topologie mit erhöhter Ausfallsicherheit. Eine Domäne oder ein einzelner Link kann ausfallen, aber die verbleibenden Domänen können trotzdem Änderungen empfangen, weil die Änderungen im Ring in die andere Richtung (von der ausgefallenen Domäne bzw. vom ausgefallenen Link weg) weitergegeben werden. Jede Domäne hat zwei Links zu den anderen Domänen. Jede Domäne hat maximal zwei Links, unabhängig davon, wie groß eine Ringtopologie ist. Die Latenzzeit für die Weitergabe der Änderung kann hoch sein, weil Änderungen einer bestimmten Domäne unter Umständen mehrere Domänen durchqueren müssen, bevor sie von allen Domänen gesehen werden. Eine Reihentopologie hat dasselbe Problem.



Dies ist eine Abbildung einer fortgeschrittenen Ringtopologie mit einer Stammdomäne in der Mitte des Rings. Die Stammdomäne dient als zentrale Clearing-Stelle, während die anderen Domänen als ferne Clearing-Stellen für Änderungen dienen, die in der Stammdomäne vorgenommen werden. Die Stammdomäne kann Änderungen zwischen den Domänen arbitrieren. Wenn eine Ringtopologie mehrere Ringe um die Stammdomäne herum enthält, kann die Stammdomäne Änderungen nur zwischen den Domänen im innersten Ring arbitrieren. Die Ergebnisse der Arbitrierung werden jedoch an die Domänen in den anderen Ringen verteilt.

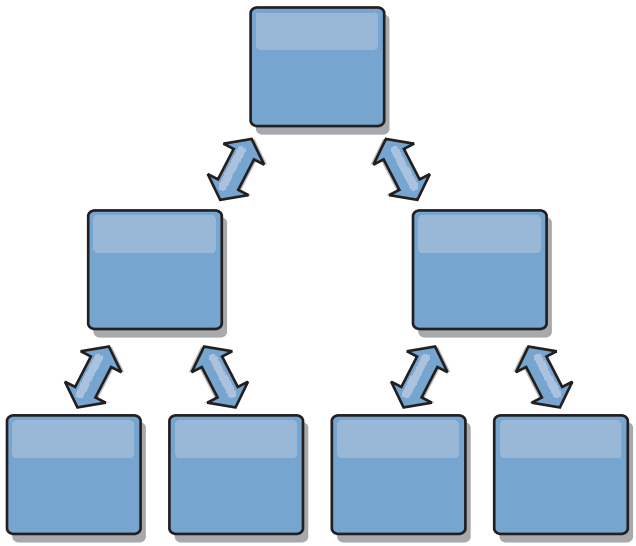
Hub- und Peripherietopologien

Eine Hub- und Peripherietopologie hat bessere Latenzzeiten, d. h., Änderungen durchqueren maximale eine zwischengeordnete Domäne (den Hub), bringt aber andere Probleme mit sich. Eine solche Topologie hat eine zentrale Domäne, die als Hub dient. Diese "Hub-Domäne" ist über einen Link mit jeder "Peripheriedomäne" verbunden. Die Last der Verteilung von Änderungen an die Domänen liegt klar auf dem Hub. Der Hub dient als Clearing-Stelle für Kollisionen. Deshalb kann dieses Setup für bestimmte Szenarien von entscheidender Bedeutung sein. In einer Umgebung mit einer hohen Aktualisierungsrate muss der Hub unter Umständen auf mehr Hardwarekomponenten als die Peripheriedomänen ausgeführt werden, damit er Schritt halten kann. eXtreme Scale ist für eine lineare Skalierung konzipiert, d. h., Sie können den Hub bei Bedarf ohne Schwierigkeit vergrößern. Wenn der Hub jedoch ausfällt, werden die Änderungen erst nach einem Neustart des Hubs wieder verteilt. Alle Änderungen in den Peripheriedomänen werden verteilt, nachdem die Hub-Verbindung wiederhergestellt ist.



Baumtopologien

Ein letztes Topologiebeispiel ist eine azyklische gerichtete Baumstruktur. Azyklisch bedeutet, dass es keine Zyklen und keine Schleifen gibt. Gerichtet bedeutet, dass Links nur zwischen übergeordneten und untergeordneten Elementen existieren. Diese Konfiguration kann für Topologien hilfreich sein, die so viele Domänen haben, dass die Verwendung eines zentralen Hubs, der mit jeder möglichen Peripheriedomäne verbunden ist, nicht praktisch ist, oder in Domänen, in denen Sie untergeordnete Domänen hinzufügen können, ohne die Stammdomäne zu aktualisieren.



Diese Topologie kann trotzdem eine zentrale Clearing-Stelle in der Stammdomäne haben, aber die zweite Ebene kann als ferne Clearing-Stelle für Änderungen die-

nen, die in den Domänen unterhalb dieser Ebene vorgenommen werden. Die Stammdomäne kann Änderungen zwischen den Domänen nur auf der zweiten Ebene arbitrieren. N-gliedrige Baumstrukturen sind ebenfalls möglich. Eine N-gliedrige Baumstruktur hat auf jeder Ebene N untergeordnete Elemente. Jede Domäne hat ein Fanout (Ausgabefächerung) von N.

Arbitrierungshinweise für das Topologiedesign

Änderungskollisionen können auftreten, wenn dieselben Datensätze gleichzeitig an zwei Stellen geändert werden können. Konfigurieren Sie alle Domänen mit denselben Werten für CPU-, Hauptspeicher und Netzressourcen. Sie können beobachten, dass Domänen, die für das Kollisions-Handling (Arbitrierung) zuständig sind, mehr Ressourcen als andere Domänen verbrauchen. Kollisionen werden automatisch erkannt. Sie werden mit einem der folgenden beiden Mechanismen behoben:

- **Standardkollisions-Arbiter**: Standardmäßig werden die Änderungen aus der Domäne verwendet, deren Name in der lexikalischer Reihenfolge am niedrigsten steht. Wenn beispielsweise Domäne A und Domäne B einen Konflikt in Bezug auf einen Datensatz verursachen, wird die Änderung aus Domäne B ignoriert. Domäne A behält ihre Version, und der Datensatz in Domäne B wird geändert, so dass er dem Datensatz aus Domäne A entspricht. Dies gilt auch für Anwendungen, in denen Benutzer oder Sitzungen normalerweise gebunden sind oder eine Affinität zu einem der Grids haben.
- **Angepasster Kollisions-Arbiter:** Anwendungen können einen angepassten Arbiter bereitstellen. Wenn eine Domäne eine Kollision erkennt, ruft sie den Arbiter auf. Informationen zum Entwickeln eines 'guten' angepassten Arbiters finden Sie unter Angepasste Arbiter für Multimaster-Replikation.

Für Topologien, in denen Kollisionen möglich sind, sollten Sie die Verwendung einer Hub- und Peripherietopologie oder einer Baumtopologie in Erwägung ziehen. Diese beiden Topologien sind dienlich, um Endloskollisionen zu vermeiden, die auftreten können, wenn

1. in mehreren Domänen eine Kollision auftritt,
2. jede Domäne die Kollision lokal behebt, was zu Überarbeitungen führt,
3. die Überarbeitungen kollidieren, was zu Überarbeitungen von Überarbeitungen führt,
4. da die Überarbeitungen zwischen den verschiedenen Domänen weitergeleitet werden, um eine Synchronizität zu erreichen.

Zur Vermeidung von Endloskollisionen wählen Sie eine bestimmte Domäne, eine *Arbitrierungsdomäne*, als Kollisions-Handler für einen Teil der Domänen aus. In einer Hub- und Peripherietopologie kann der Hub beispielsweise als Kollisions-Handler verwendet werden. Der Peripheriekollisions-Handler ignoriert alle von den Peripheriedomänen erkannten Kollisionen. Die Hub-Domäne erstellt Überarbeitungen, wodurch Kollisionsüberarbeitungen, die außer Kontrolle geraten, verhindert werden. Die für die Behandlung von Kollisionen zugeordnete Domäne muss einen Link zu allen Domänen haben, für die sie Kollisionen beheben soll. In einer Baumtopologie beheben alle internen übergeordneten Domänen Kollisionen für die ihnen unmittelbar untergeordneten Domänen. Wenn Sie eine Ringtopologie verwendet, ist es nicht möglich, eine einzige Domäne im Ring zu bestimmen, die die Kollisionen beheben soll.

In der folgenden Tabelle sind die kompatiblen Arbitrierungsansätze für die verschiedenen Topologien zusammengefasst.

Tabelle 14. Arbitrierungsansätze. Der folgenden Tabelle können Sie entnehmen, ob Anwendungsarbitrierung mit den verschiedenen Technologien kompatibel ist.

Topologie	Anwendungs-arbitrierung?	Anmerkungen
Eine Reihe von zwei Domänen	Ja	Wählen Sie eine Domäne als Arbiter aus.
Eine Reihe von drei Domänen	Ja	Die mittlere Domäne muss der Arbiter sein. Stellen Sie sich die mittlere Domäne als Hub in einer einfachen Hub- und Peripherietopologie vor.
Eine Reihe von mehr als drei Domänen	Nein	Anwendungsarbitrierung wird nicht unterstützt.
Ein Hub mit N Peripheriedomänen	Ja	Der Hub mit den Links zu allen Peripheriedomänen muss die Arbitrierungsdomäne sein.
Ein Ring mit N Domänen	Nein	Anwendungsarbitrierung wird nicht unterstützt.
Eine azyklische gerichtete Baumstruktur (N-gliedriger Baumstruktur)	Ja	Alle Stammknoten müssen nur die ihnen direkt untergeordneten Knoten arbitrieren.

Linkhinweise für das Topologiedesign

Im Idealfall enthält eine Topologie die Mindestanzahl an Links und optimiert gleichzeitig Kompromisse zwischen Latenzzeit für Änderungen, Fehlertoleranz und Leistungsmerkmale.

- **Latenzzeit bei Änderungen**

  Die Latenzzeit bei Änderungen wird durch die Anzahl zwischengeschalteter Domänen bestimmt, die eine Änderung durchlaufen muss, bevor sie in einer bestimmten Domäne ankommt.

  Eine Topologie weist die beste Latenzzeit bei Änderungen auf, wenn zwischengeschaltete Domänen wegfallen, weil jede Domäne mit jeder anderen Domäne verlinkt wird. Eine Domäne muss jedoch proportional zur Anzahl ihrer Links Replikationsarbeiten ausführen. In großen Topologien kann die reine Anzahl zu definierender Links einen hohen Verwaltungsaufwand darstellen.

  Die Geschwindigkeit, mit der eine Änderung in andere Domänen kopiert wird, richtet sich nach weiteren Faktoren, wie z. B.:

  – CPU und Netzbandbreite in der Quellendomäne,
  – Anzahl zwischengeschalteter Domänen und Links zwischen der Quellen- und der Zieldomäne,
  – CPU- und Netzressourcen, die der Quellendomäne, der Zieldomäne und den zwischengeschalteten Domänen zur Verfügung stehen.

- **Fehlertoleranz**

  Die Fehlertoleranz wird durch die Anzahl der existierenden Pfade zwischen zwei Domänen für die Änderungsreplikation bestimmt.

  Wenn nur ein einziger Link zwischen Domänen existiert und dieser Link ausfällt, werden Änderungen nicht weitergegeben. Wenn der einzige Link von einer Domäne zu einer anderen Domäne über zwischengeschaltete Domänen verläuft, werden die Änderungen nicht weitergegeben, wenn eine der zwischengeschalteten Domänen inaktiv ist.

  Stellen Sie sich eine Reihentopologie mit drei Domänen, A, B und C, vor:

```
A <-> B <-> C
```

Wenn eine der folgenden Bedingungen zutrifft, sieht die Domäne C Änderungen von Domäne A nicht:

- Domäne A ist aktiv, und Domäne B ist inaktiv.
- Der Links zwischen A und B ist inaktiv.
- Der Link zwischen B und C ist inaktiv.

Im Gegensatz dazu kann in einer Ringtopologie jede Domäne Änderungen aus beiden Richtungen übernehmen.

```
A <-> B <-> C <-> zurück zu A
```

Wenn beispielsweise Domäne B inaktiv ist, kann Domäne C trotzdem Änderungen direkt von Domäne A übernehmen.

Ein Hub- und Peripheriedesign ist anfällig, wenn der Hub ausfällt, weil alle Änderungen über den Hub verteilt werden. Es ist jedoch zu bedenken, dass eine einzige Domäne trotzdem ein vollständig fehlertolerantes Grid ist, in dem Fehler wie WAN-Probleme oder Probleme im physischen Rechenzentrum selten auftreten.

- **Leistung**

  Die Anzahl der in einer Domäne definierten Links wirkt sich auf die Leistung aus. Je mehr Links definiert werden, desto mehr Ressourcen werden benötigt, und deshalb kann die Replikationsleistung abnehmen. Die Möglichkeit, Änderungen für Domäne A über andere Domänen zu beziehen, entlastet die Domäne A effektiv von der Replikation ihrer Transaktionen in allen Domänen. *Die Last für die Verteilung der Änderungen in einer Domäne ist auf die Anzahl der von dieser Domäne verwendeten Links beschränkt. Sie hat nicht mit der Anzahl der Domänen in der Topologie zu tun.* Diese Eigenschaft bietet Skalierbarkeit und die Möglichkeit, die Last für die Verteilung von Änderungen auf die Domänen in der Topologie zu verteilen, anstatt diese auf eine einzige Domäne zu konzentrieren.

  Eine Domäne kann Änderungen indirekt über andere Domänen beziehen. Stellen Sie sich eine Reihentopologie mit fünf Domänen vor.

  ```
  A <=> B <=> C <=> D <=> E
  ```

  - A bezieht Änderungen von B, C, D und E über B.
  - B bezieht Änderungen von A und C direkt und Änderungen von D und E über C.
  - C bezieht Änderungen von B und D direkt und Änderungen von A über B und Änderungen von E über D.
  - D bezieht Änderungen von C und E direkt und Änderungen von A und B über C.
  - E bezieht Änderungen von D direkt und Änderungen von A, B und C über D.

  Die Verteilungslast ist in den Domänen A und E am geringsten, weil sie jeweils nur einen Link zu einer einzigen Domäne haben. Die Verteilungslast in den Domänen B, C und D ist doppelt so hoch wie die Laste in den Domänen A und E, weil die Domänen B, C und D jeweils einen Links zu zwei Domänen haben. Diese Verteilung der Last bliebe auch bei 1000 Domänen in der Reihe konstant, weil sich die Last nach der Anzahl der Links jeder Domäne richtet und nicht nach der Gesamtanzahl der Domänen in der Topologie.

Leistungsaspekte

Berücksichtigen Sie bei der Verwendung von Multimaster-Replikationstopologien die folgenden Einschränkungen:

- **Optimierung der Verteilung von Änderungen** (zuvor beschrieben)
- **Latenzzeit bei der Replikation** (zuvor beschrieben)
- **Leistung von Replikationslinks**: eXtreme Scaleerstellt einen einzigen TCP/IP-Socket zwischen jedem JVM-Paar. Der gesamte Datenverkehr zwischen diesen JVMs findet über diesen Socket statt, einschließlich der Multimaster-Replikation. Da sich Domänen in mindestens N Container-JVMs mit N TCP-Links zu Peer-Domänen befinden, weisen die Domänen mit einer höheren Anzahl an Containern höhere Replikationsleistungsraten auf. Mehr Container bedeuten mehr CPU- und Netzressourcen.
- **Optimierung des TCP-Sliding-Window und RFC 1323**: Wenn Sie die Unterstützung für RFC 1323 auf beiden Seiten eines Links aktivieren, werden mehr Daten in einer Austauschoperation zugelassen, was zu einem höheren Durchsatz führt. Die Technik erweitert die Fensterkapazität um einen Faktor von ca. 16.000.

  TCP-Sockets verwenden einen Sliding-Window-Mechanismus, um den Fluss von Massendaten zu steuern, der den Socket gewöhnlich auf 64 KB für ein Umlaufintervall beschränkt. Wenn ein Umlaufintervall 100 ms lang ist, ist die Bandbreite ohne Optimierung auf 640 KB/Sekunde beschränkt. Um die verfügbare Bandbreite eines Links vollständig nutzen zu können, müssen unter Umständen spezielle Optimierungs-Tasks für ein Betriebssystem ausgeführt werden. Die meisten Betriebssysteme haben Optimierungsparameter, einschließlich Optionen für RFC 1323, um den Durchsatz über Links mit hoher Latenzzeit zu verbessern.

  Es gibt mehrere Faktoren, die sich auf die Replikationsleistung auswirken können:

  – Geschwindigkeit, mit der eXtreme Scale Änderungen beziehen kann,
  – Geschwindigkeit, mit der eXtreme Scale Replikationsanforderungen bearbeiten kann,
  – Kapazität des Sliding Window,
  – Optimierung des Netzpuffers auf beiden Seiten eines Links, damit eXtreme Scale Änderungen über den Socket so schnell wie möglich beziehen kann.
- **Objektserialisierung**: Alle Daten müssen serialisierbar sein. Wenn eine Domäne COPY_TO_BYTES nicht verwendet, muss die Domäne Java-Serialisierung oder ObjectTransformer verwenden, um die Serialisierungsleistung zu optimieren.
- **Komprimierung**: eXtreme Scale komprimiert standardmäßig alle Daten, die zwischen Domänen gesendet werden. Es gibt keine Option für die Inaktivierung der Komprimierung im aktuellen Release.
- **Hauptspeicheroptimierung**: *Die Speicherbelegung für eine Multimaster-Replikationstopologie ist weitgehend unabhängig von der Anzahl der Domänen in der Topologie.*

  Die Aktivierung der Multimaster-Replikation bedeutet feste Gemeinkosten pro Map-Eintrag für die Versionssteuerung. Außerdem überwacht jeder Container ein festes Datenvolumen für jede Domäne in der Topologie. Eine Topologie mit zwei Domänen belegt ungefähr denselben Speicher wie eine Topologie mit 50 Domänen. eXtreme Scale verwendet keine Wiedergabeprotokolle oder ähnlichen Warteschlangen in der Implementierung, d. h., wenn ein Replikationslink für längere Zeit nicht verfügbar ist, gibt Datenstruktur mit zunehmender Größe, die darauf wartet, die Replikation fortzusetzen, wenn der Link erneut gestartet wird.

Mehrere Rechenzentren mit FIXED_PARTITION

Sie können jetzt ein FIXED_PARTITION-Grid zwischen zwei oder mehr Rechenzentren einsetzen. Jedes Rechenzentrum benötigt im Hinblick auf die Multimaster-Replikation eine eigene Domäne. Jedes Rechenzentrum kann Daten aus der lokalen Domäne lesen und in diese schreiben. Diese Änderungen werden an die anderen Rechenzentren über die von Ihnen definierten Links weitergegeben.

Vollständig replizierte Clients

Diese Topologievariante umfasst ein eXtreme-Scale-Serverpaar, das als Hub ausgeführt wird. Jeder Client erstellt ein eigenständiges Einzelcontainer-Grid mit einem Katalog in der Client-JVM. Ein Client verwendet sein Grid, um die Verbindung zum Hub-Katalog herzustellen, was bewirkt, dass sich der Client mit dem Hub synchronisiert, sobald die Verbindung zum Hub hergestellt ist.

Alle vom Client vorgenommenen Änderungen sind lokal und werden asynchron im Hub repliziert. Der Hub dient als Arbitrierungsdomäne und verteilt Änderungen an alle verbundenen Clients. Die Topologie mit vollständig replizierten Clients ist ein guter L2-Cache für einen objektbezogenen Mapper wie OpenJPA. Änderungen werden über den Hub schnell an die Client-JVMs verteilt. Solange die Cachegröße vom verfügbaren Heap-Speicher der Clients untergebracht werden kann, ist diese Topologie eine geeignete Architektur für diesen L2-Stil.

Verwenden Sie bei Bedarf mehrere Partitionen für die Skalierung der Hub-Domäne in mehreren JVMs. Da alle Daten weiterhin in eine einzige Client-JVM passen müssen, erhöht die Verwendung mehrerer Partitionen die Kapazität des Hubs für die Verteilung und Arbitrierung von Änderungen, aber nicht die Kapazität einer einzelnen Domäne.

Einschränkungen

Berücksichtigen Sie bei der Entscheidung, ob und wie Multimaster-Replikationstopologien zu verwenden sind, die folgenden Einschränkungen:

- **Konfiguration von Loadern mit mehreren Domänen**

  Domänen müssen Zugriff auf alle Klassen haben, die als Schlüssel und Werte verwendet werden. Alle Abhängigkeiten müssen sich in allen Klassenpfaden für Grid-Container-JVMs für alle Domänen widerspiegeln. Wenn ein CollisionArbiter-Plug-in den Wert für einen Cacheeintrag abruft, müssen die Klassen für die Werte für die Domäne vorhanden sein, die den Arbiter aufruft.
- **Verwendung von Loadern wird nicht empfohlen**

  Loader können verwendet werden, um Änderungen zwischen einem Grid und einer Datenbank zu übertragen. Es ist unwahrscheinlich, dass alle Grids (Domänen) in einer Topologie geografisch mit derselben Datenbank kollokiert sind. Aufgrund der Latenzzeit im WAN und anderen Faktoren kann dieser Anwendungsfall unerwünscht sein.

  Das vorherige Laden (Preload) des Grids ist ein weiterer Punkt, der eines sorgfältigen Designs bedarf. Beim Neustart eines Grids wird der Preload gewöhnlich erneut durchgeführt. Ein Preload ist bei der Verwendung einer Multimaster-Replikation nicht erforderlich oder sogar nicht erwünscht. Sobald eine Domäne online ist, lädt sie automatisch den Inhalt der Domänen, mit denen sie verlinkt ist. Deshalb ist es nicht erforderlich, einen manuellen Preload für ein Grid einzuleiten, das eine Domäne in einer Multimaster-Replikationstopologie ist.

Loader halten gewöhnlich Einfüge- und Aktualisierungsregeln ein. Bei der Multimaster-Replikation müssen Einfügungen als Zusammenführungen behandelt werden. Wenn die Daten nach einem Domänenneustart über Fernzugriff abgerufen werden, werden vorhandene Daten in die lokale Domäne "eingefügt". Da diese Daten unter Umständen bereits in der lokalen Datenbank enthalten sind, schlägt eine typische Einfügung mit einer Ausnahme wegen doppelt vorhandener Schlüssel in der Datenbank fehl. Stattdessen muss die Semantik für Zusammenführung (merge) verwendet werden.

eXtreme Scale kann für die Durchführung eines Shard-basierten Preloads unter Verwendung der Preload-Methoden in Loader-Plug-ins konfiguriert werden. Verwenden Sie diese Technik nicht in einer Multimaster-Replikationstopologie. Verwenden Sie stattdessen einen clientbasierte Preload, wenn die Topologie (anfänglich) gestartet wird. Lassen Sie zu, dass die Multimaster-Topologie alle neu gestarteten Domänen mit einer aktuellen Kopie der in anderen Domänen der Topologie gespeicherten Daten aktualisiert. Nach dem Start von Domänen ist die Multimaster-Topologie für die Synchronisation der Domänen zuständig.

- **Keine Unterstützung für EntityManager**

  Ein MapSet, das eine Entitäts-Map enthält, wird in Domänen nicht repliziert.
- **Keine Unterstützung für Bytefeldgruppen-Maps**

  Ein MapSet, das eine Map enthält, die mit COPY_TO_BYTES konfiguriert ist, wird in Domänen nicht repliziert.
- **Keine Unterstützung für Write-behind-Operationen**

  Ein MapSet, das eine Map enthält, die mit Write-behind-Unterstützung konfiguriert ist, wird in Domänen nicht repliziert.

JMS für Verteilung von Transaktionsänderungen

Verwenden Sie Java Message Service (JMS) für die Verteilung von Transaktionsänderungen zwischen verschiedenen Schichten und in Umgebungen auf heterogenen Plattformen.

JMS ist ein ideales Protokoll für die Verteilung von Änderungen zwischen verschiedenen Schichten und in Umgebungen auf heterogenen Plattformen. Einige Anwendungen, die eXtreme Scale verwenden, können beispielsweise in IBM WebSphere Application Server Community Edition, Apache Geronimo oder Apache Tomcat implementiert sein, wohingegen andere Anwendungen in WebSphere Application Server Version 6.x ausgeführt werden. JMS eignet sich optimal für die Verteilung von Änderungen zwischen eXtreme-Scale-Peers in diesen unterschiedlichen Umgebungen. Der Nachrichtentransport des High Availability Manager ist sehr schnell, kann aber nur Änderungen an Java Virtual Machines verteilen, die sich in derselben Stammgruppe befinden. JMS ist zwar langsamer, unterstützt aber die gemeinsame Nutzung eines ObjectGrids durch größere und unterschiedlichere Gruppen von Anwendungsclients. JMS ist ideal, wenn Daten in einem ObjectGrid von einem Swing-Fat-Client und einer in WebSphere Extended Deployment implementierten Anwendung gemeinsam genutzt werden.

Der integrierte Mechanismus für die Inaktivierung von Clients und die Peer-to-Peer-Replikation sind Beispiele für JMS-basierte Verteilung von Transaktionsänderungen. Weitere Informationen finden Sie in der Beschreibung der Peer-to-Peer-Replikation mit JMS im *Administratorhandbuch*.

JMS implementieren

JMS wird für die Verteilung von Transaktionsänderungen über ein Java-Objekt implementiert, das sich wie ein ObjectGridEventListener verhält. Dieses Objekt kann den Status auf die folgenden vier Arten weitergeben:

1. Invalidate: (Ungültigmachen) Jeder Eintrag, der entfernt, aktualisiert oder gelöscht wird, wird in allen Peer-JVMs entfernt, wenn diese die Nachricht empfangen.
2. Invalidate conditional: (Bedingtes Ungültigmachen) Der Eintrag wird nur entfernt, wenn die lokale Version kleiner-gleich der Version im Veröffentlichungskomponente (Publisher) ist.
3. Push: (Übertragung mit Push) Jeder Eintrag, der entfernt, aktualisiert, gelöscht oder eingefügt wurde, wird in allen Peer-JVMs hinzugefügt bzw. überschrieben, wenn diese die JMS-Nachricht empfangen.
4. Push conditional: (Bedingte Übertragung mit Push) Der Eintrag wird auf Empfangsseite nur dann aktualisiert bzw. hinzugefügt, wenn der lokale Eintrag älter ist als die Version, die veröffentlicht wird.

Auf zu veröffentlichende Änderungen warten

Das Plug-in implementiert die Schnittstelle "ObjectGridEventListener", um das Ereignis "transactionEnd" abzufangen. Wenn eXtreme Scale diese Methode aufruft, versucht das Plug-in die LogSequence-Liste für jede Map, die von der Transaktion angerührt wurde, in eine JMS-Nachricht zu konvertieren und anschließend zu veröffentlichen. Das Plug-in kann so konfiguriert werden, dass Änderungen für alle Maps oder einen Teil der Maps veröffentlicht werden. LogSequence-Objekte werden für die Maps verarbeitet, für die die Veröffentlichung aktiviert ist. Die ObjectGrid-Klasse "LogSequenceTransformer" serialisiert eine gefilterte LogSequence für jede Map in einen Datenstrom. Nachdem alle LogSequences in den Datenstrom serialisiert wurden, wird eine JMS-ObjectMessage erstellt und unter einem bekannten Topic veröffentlicht.

Auf JMS-Nachrichten warten und sie auf das lokale ObjectGrid anwenden

Dasselbe Plug-in startet auch einen Thread, der in einer Schleife ausgeführt wird und alle Nachrichten empfängt, die unter dem bekannten Topic veröffentlicht werden. Wenn eine Nachricht ankommt, wird der Nachrichteninhalt an die Klasse "LogSequenceTransformer" übergeben, wo sie in eine Gruppe von LogSequence-Objekten konvertiert wird. Anschließend wird eine Transaktion ohne Durchschreiben (no-write-through) gestartet. Jedes LogSequence-Objekt wird an die Methode "Session.processLogSequence" übergeben, die die lokalen Maps mit den Änderungen aktualisiert. Die Methode "processLogSequence" erkennt den Verteilungsmodus. Die Transaktion wird festgeschrieben, und der lokale Cache enthält die Änderungen. Weitere Informationen zur Verwendung von JMS für die Verteilung von Transaktionsänderungen finden Sie in den Informationen zum Verteilen von Änderungen zwischen Peer-Java-Virtual-Machines im *Administratorhandbuch*.

MapSets für die Replikation

Die Replikation wird durch Zuordnung von BackingMaps zu einem MapSet aktiviert.

Ein MapSet ist eine Sammlung von Maps, die nach Partitionsschlüssel klassifiziert sind. Dieser Partitionsschlüssel wird wie folgt aus dem Schlüssel der jeweiligen

Map abgeleitet: Hash-Wert Modulo Partitionsanzahl. Wenn eine Map-Gruppe im MapSet also den Partitionsschlüssel X hat, werden diese Maps in einer entsprechenden Partition X im Grid gespeichert. Wenn eine andere Gruppe den Partitionsschlüssel Y hat, werden alle Maps in Partition Y gespeichert usw. Außerdem werden die Daten in den Maps auf der Basis der Richtlinie repliziert, die im MapSet definiert ist. Dies wird nur für verteilte eXtreme-Scale-Topologien verwendet und ist für lokale Instanzen nicht erforderlich.

Weitere Einzelheiten finden Sie im Abschnitt „Partitionierung” auf Seite 95.

Den MapSets wird die Anzahl vorhandener Partitionen und eine Replikationsrichtlinie zugeordnet. In der Replikationskonfiguration des MapSets wird einfach die Anzahl synchroner und asynchroner Replikat-Shards angegeben, die ein MapSet zusätzlich zum primären Shard haben sollte. Wenn beispielsweise ein synchrones und ein asynchrones Replikat verwendet werden sollen, wird für alle BackingMaps, die dem MapSet zugeordnet werden, automatisch jeweils ein Replikat-Shard auf die Gruppe verfügbarer Container für eXtreme Scale verteilt. Die Replikationskonfiguration kann Clients auch das Lesen von Daten aus synchron replizierten Servern ermöglichen. Auf diese Weise kann die Last der Leseanforderungen auf zusätzliche Server in eXtreme Scale verteilt werden. Die Replikation hat nur beim vorherigen Laden der BackingMaps Auswirkungen auf das Programmiermodell.

Einzelheiten zu den verschiedenen Konfigurationsoptionen finden Sie im Folgenden:

Kapitel 6. Übersicht über die Transaktionsverarbeitung

Sitzungen und Transaktionsverarbeitung

WebSphere eXtreme Scale verwendet Transaktionen als Mechanismus für die Interaktion mit Daten.

Für die Interaktion mit Daten benötigt der Thread in Ihrer Anwendung ein eigenes Session-Objekt. Wenn die Anwendung das ObjectGrid in einem Thread verwenden möchte, rufen Sie eine der Methoden "ObjectGrid.getSession" auf, um einen Thread anzufordern. Über das Session-Objekt kann die Anwendung die in den ObjectGrid-Maps gespeicherten Daten bearbeiten.

Wenn eine Anwendung ein Session-Objekt verwendet, muss dieses im Kontext einer Transaktion enthalten sein. Eine Transaktion wird über die Methoden "begin", "commit" und "rollback" des Session-Objekts gestartet und festgeschrieben bzw. rückgängig gemacht. Anwendungen können auch im Modus für automatische Festschreibung arbeiten, in dem das Session-Objekt eine Transaktion automatisch startet und festschreibt, wenn eine Operation in der Map durchgeführt wird. Der Modus für automatische Festschreibung ist nicht in der Lage, mehrere Operationen zu einer einzigen Transaktion zu gruppieren, und damit die langsamere Option, wenn Sie einen Stapel mit mehreren Operationen in einer einzigen Transaktion erstellen. Für Transaktionen, die nur eine einzige Operation enthalten, ist die automatische Festschreibung jedoch die schnellere Option.

Transaktionen

Transaktionen haben viele Vorteile in Bezug auf die Speicherung und Bearbeitung von Daten. Sie können Transaktionen verwenden, um das Grid vor gleichzeitigen Änderungen zu schützen, mehrere Änderungen als Einheit anzuwenden, Daten zu replizieren und einen Lebenszyklus für Sperren bei Änderungen zu implementieren.

Wenn eine Transaktion gestartet wird, ordnet WebSphere eXtreme Scale eine spezielle Differenz-Map zu, in der die aktuellen Änderungen bzw. Kopien von Schlüssel/Wert-Paaren gespeichert werden, die von der Transaktion verwendet werden. Normalerweise wird beim Zugriff auf ein Schlüssel/Wert-Paar der Wert kopiert, bevor die Anwendung den Wert erhält. Die Differenz-Map verfolgt alle Änderungen, wenn Operationen wie Einfüge-, Aktualisierungs-, Abruf-, Entfernungsoperationen usw. durchgeführt werden. Schlüssel werden nicht kopiert, weil sie als unveränderlich gelten. Wenn ein ObjectTransformer-Objekt angegeben ist, wird dieses Objekt zum Kopieren des Werts verwendet. Wenn die Transaktion optimistisches Sperren verwendet, werden auch Vorher-Kopien der Werte verfolgt, um sie als Vergleichswerte beim Festschreiben der Transaktion zu verwenden.

Wenn eine Transaktion rückgängig gemacht wird, werden die Informationen in der Differenz-Map verworfen und Sperren für die Einträge freigegeben. Wenn eine Transaktion festgeschrieben wird, werden die Änderungen auf die Maps angewendet und die Sperren freigegeben. Bei der Verwendung optimistischen Sperrens vergleicht eXtreme Scale die Vorher-Versionen der Werte mit den Werten, die in der Map enthalten sind. Diese Werte müssen übereinstimmen, damit die Transaktion festgeschrieben werden kann. Dieser Vergleich ermöglicht ein Schema für das Sperren mehrerer Versionen. Hierbei entstehen jedoch zusätzliche Kosten, weil zwei

Kopien erstellt werden, wenn die Transaktion auf den Eintrag zugreift. Alle Werte werden erneut kopiert, und die neue Kopie wird in der Map gespeichert. WebSphere eXtreme Scale führt diesen Kopiervorgang durch, um sich selbst davor zu schützen, dass die Anwendung die Anwendungsreferenz in den Wert nach Festschreibung ändert.

Sie können dies vermeiden, indem Sie mehrere Kopien der Informationen erstellen. Die Anwendung kann eine Kopie auch über pessimistisches Sperren anstatt über optimistisches Sperren speichern. Dies schränkt jedoch den gemeinsamen Zugriff ein. Die Kopie des Wertes zur Festschreibungszeit kann ebenfalls vermieden werden, wenn die Anwendung zustimmt, einen Wert nach der Festschreibung nicht mehr zu ändern.

Vorteile von Transaktionen

Verwenden Sie Transaktionen für die folgenden Zwecke:

Wenn Sie Transaktionen verwenden, ist Folgendes möglich:

- Änderungen können rückgängig gemacht werden, wenn eine Ausnahme eintritt oder wenn die Geschäftslogik Statusänderungen widerrufen muss.
- zum Anwenden mehrerer Änderungen als atomare Einheit beim Festschreiben,
- Sperren für Daten können gehalten und freigegeben werden, um mehrere Änderungen als atomare Einheit zur Festschreibungszeit anzuwenden.
- Threads werden vor gleichzeitigen Änderungen geschützt.
- Es kann ein Lebenszyklus für Sperren bei Änderungen implementiert werden.
- Es kann eine atomare Replikationseinheit erzeugt werden.

Transaktionsgröße

Größere Transaktionen sind effizienter, insbesondere für die Replikation. Größere Transaktionen können sich jedoch nachteilig auf den gemeinsamen Zugriff auswirken, weil die Sperren für Einträge länger gehalten werden. Wenn Sie größere Transaktionen verwenden, kann dies die Replikationsleistung steigern. Dieser Leistungsanstieg ist wichtig, wenn Sie eine Map vorher laden. Experimentieren Sie mit verschiedenen Stapelgrößen, um festzustellen, welche sich für Ihr Szenario am besten eignet.

Größere Transaktionen sind auch bei Loadern hilfreich. Wenn ein Loader verwendet wird, der SQL-Stapeloperationen durchführen kann, sind je nach Transaktion erhebliche Leistungssteigerungen und auf der Datenbankseite erheblich Lastreduktionen möglich. Diese Leistungssteigerung ist von der Loader-Implementierung abhängig.

Modus für automatische Festschreibung

Wenn keine Transaktion aktiv gestartet wird und eine Anwendung mit einem ObjectMap-Objekt interagiert, wird eine automatische Start- und Festschreibungsoperation für die Anwendung durchgeführt. Diese automatische Start- und Festschreibungsoperation funktioniert, verhindert aber, dass Rollback und Sperren effektiv funktionieren. Die Geschwindigkeit der synchronen Replikation nimmt aufgrund der sehr geringen Transaktionsgröße ab. Wenn Sie eine EntityManager-Anwendung verwenden, sollten Sie den Modus für automatische Festschreibung nicht verwenden, weil Objekte, die mit der Methode EntityManager.find gesucht werden, un-

mittelbar nach der Rückkehr der Methode nicht mehr verwaltet werden und somit unbrauchbar sind.

Externe Transaktionskoordinatoren

Gewöhnlich beginnt eine Transaktion mit der Methode "session.begin" und endet mit der Methode "session.commit". Wenn eXtreme Scale integriert ist, können die Transaktionen jedoch auch von externen Transaktionskoordinatoren gestartet und beendet werden. Wenn Sie einen externen Transaktionskoordinator verwenden, müssen Sie die Methoden "session.begin" und "session.commit" nicht aufrufen. Weitere Informationen zu eXtreme Scale und externer Transaktionsinteraktion finden Sie im *Programmierhandbuch*. Wenn Sie WebSphere Application Server verwenden, können Sie das WebSphereTranscationCallback-Plug-in einsetzen. Weitere Informationen zu den in WebSphere eXtreme Scale verfügbaren Plug-ins finden Sie im *Programmierhandbuch*.

Attribut "CopyMode"

Sie können die Anzahl der Kopien optimieren, indem Sie das Attribut "CopyMode" der BackingMap- bzw. ObjectMap-Objekte definieren.

Sie können die Anzahl der Kopien optimieren, indem Sie das Attribut "CopyMode" der BackingMap- bzw. ObjectMap-Objekte definieren. Das Attribut "CopyMode" hat die folgenden gültigen Werte:

- COPY_ON_READ_AND_COMMIT
- COPY_ON_READ
- NO_COPY
- COPY_ON_WRITE
- COPY_TO_BYTES

Der Wert COPY_ON_READ_AND_COMMIT ist der Standardwert. Wenn Sie den Wert COPY_ON_READ definieren, wird eine Kopie der ersten empfangenen Daten erstellt, aber es wird keine Kopie zur Festschreibungszeit erstellt. Dieser Modus ist sicher, wenn die Anwendung einen Wert nach dem Festschreiben einer Transaktion nicht mehr ändert. Wenn Sie den Wert NO_COPY definieren, werden die Daten nicht kopiert, was nur bei schreibgeschützten Daten sicher ist. Wenn sich die Daten nicht ändern, ist es nicht erforderlich, sie aus Isolationsgründen zu kopieren.

Gehen Sie bei der Verwendung des Attributwerts NO_COPY für Maps, die aktualisiert werden können, vorsichtig vor. WebSphere eXtreme Scale verwendet die beim ersten Berühren der Daten erstellt Kopie für das Transaktions-Rollback. Da die Änderung nur die Kopie geändert hat, verwirft eXtreme Scale die Kopie. Wenn der Attributwert NO_COPY verwendet wird und die Anwendung den festgeschriebenen Wert ändert, ist ein Rollback nicht möglich. Das Ändern der festgeschriebenen Werte führt zu Problemen bei Indizes, Replikation usw, weil die Indizes und Replikate bei der Festschreibung der Transaktion aktualisiert werden. Wenn Sie festgeschriebene Daten ändern und dann ein Rollback für die Transaktion durchführen (wobei in diesem Fall gar kein Rollback durchgeführt wird), werden die Indizes nicht aktualisiert, und es findet keine Replikation statt. Andere Threads können die nicht festgeschriebenen Änderungen sofort sehen, selbst wenn Sperren gesetzt sind. Verwenden Sie den Attributwert NO_COPY nur für schreibgeschützte Maps oder für Anwendungen, die den entsprechenden Kopiervorgang durchführen, bevor der Wert geändert wird. Wenn Sie den Attributwert NO_COPY verwenden und sich

mit einem Datenintegritätsproblem an die IBM Unterstützungsfunktion wenden, werden Sie aufgefordert, das Problem im Kopiermodus COPY_ON_READ_AND-_COMMIT zu reproduzieren.

Wenn Sie den Wert COPY_TO_BYTES verwenden, werden Werte in der Map in serialisierter Form gespeichert. Beim Lesen dekomprimiert eXtreme Scale den Wert aus der serialisierten Form und speichert beim Festschreiben den Wert in serialisierter Form. Wenn Sie diese Methode verwenden, wird beim Lesen und beim Festschreiben ein Kopiervorgang durchgeführt.

Der Standardkopiermodus für eine Map kann im BackingMap-Objekt konfiguriert werden. Mit der Methode "ObjectMap.setCopyMode" können Sie den Kopiermodus für Maps auch vor dem Starten einer Transaktion ändern.

Im Folgenden sehen Sie ein Beispiel für ein BackingMap-Snippet aus einer Datei `objectgrid.xml`, das veranschaulicht, wie der Kopiermodus für eine bestimmte BackingMap gesetzt wird. In diesem Beispiel wird davon ausgegangen, dass Sie `cc` als Namespace für `objectgrid/config` verwenden.

```
<cc:backingMap name="RuntimeLifespan" copyMode="NO_COPY"/>
```

Weitere Informationen finden Sie in der Dokumentation zu den bewährten Verfahren für copyMode im *Programmierhandbuch*.

Sperren von Map-Einträgen

Eine ObjectGrid-BackingMap unterstützt mehrere Sperrstrategien für Maps für die Verwaltung der Konsistenz von Cacheeinträgen.

Jede BackingMap kann für die Verwendung einer der folgenden Sperrstrategien konfiguriert werden:

1. Optimistischer Sperrmodus
2. Pessimistischer Sperrmodus
3. Ohne

Die Standardsperrstrategie ist OPTIMISTIC (optimistisch). Verwenden Sie optimistisches Sperren, wenn die Daten nur selten geändert werden. Sperren werden nur für kurze Dauer gehalten, während die Daten aus dem Cache gelesen und in die Transaktion kopiert werden. Wenn der Transaktionscache mit dem Hauptcache synchronisiert wird, werden alle zwischengespeicherten Objekte, die aktualisiert wurden, mit der Originalversion verglichen. Wenn die Prüfung negativ ausfällt, wird eine Rollback-Operation für die Transaktion durchgeführt, und es wird eine Ausnahme des Typs "OptimisticCollisionException" ausgegeben.

Bei der Sperrstrategie PESSIMISTIC (pessimistisch) werden Sperren für Cacheeinträge angefordert. Diese Strategie sollte verwendet werden, wenn die Daten häufig geändert werden. Jedesmal, wenn ein Cacheeintrag gelesen wird, wird eine Sperre angefordert und so lange gehalten, bis die Transaktion abgeschlossen wird. Die Dauer einiger Sperren kann über die verfügbaren Isolationsstufen für die Sitzung optimiert werden.

Wenn keine Sperren erforderlich sind, weil die Daten nie oder nur in ruhigen Phasen aktualisiert werden, können Sie die Sperren inaktivieren, indem Sie die Sperrstrategie NONE (Ohne) konfigurieren. Diese Strategie ist sehr schnell, weil kein Sperrenmanager erforderlich ist. Die Sperrstrategie NONE eignet sich ideal für Suchtabellen und schreibgeschützte Maps.

Weitere Informationen zu Sperrstrategien finden Sie in der Dokumentation zu Sperrstrategien in der *Produktübersicht*.

Sperrstrategie festlegen

Das folgende Beispiel demonstriert, wie für die BackingMaps "map1", "map2" und "map3" jeweils eine andere Sperrstrategie festgelegt wird. Das erste Snippet veranschaulicht die Verwendung von XML für die Konfiguration der Sperrstrategie und das zweite Snippet einen programmgesteuerten Ansatz.

XML-Ansatz

BackingMap-Konfiguration - XML-Beispiel

```
<?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="test">
            <backingMap name="map1"
                lockStrategy="PESSIMISTIC" numberOfLockBuckets="31"/>
            <backingMap name="map2"
                lockStrategy="OPTIMISTIC" numberOfLockBuckets="409"/>
            <backingMap name="map3"
                lockStrategy="NONE"/>
        </objectGrid>
    </objectGrids>
</objectGridConfig>
```

Programmgesteuerter Ansatz

BackingMap-Konfiguration - Programmgesteuertes Beispiel

```
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.setNumberOfLockBuckets(31);
bm = og.defineMap("map2");
bm.setNumberOfLockBuckets(409);
bm.setLockStrategy( LockStrategy.OPTIMISTIC );
bm = og.defineMap("map3");
bm.setLockStrategy( LockStrategy.NONE );
```

Um zu verhindern, dass eine Ausnahme des Typs "java.lang.IllegalStateException" ausgelöst wird, muss die Methode "setLockStrategy" vor dem Aufruf der Methode "initialize" bzw. "getSession" in der lokalen ObjectGrid-Instanz aufgerufen werden.

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

Konfiguration des Sperrenmanagers

Wenn Sie die Sperrstrategie PESSIMISTIC oder OPTIMISTIC verwenden, wird ein Sperrenmanager für die BackingMap erstellt. Der Sperrenmanager verwendet eine Hash-Tabelle, um die Einträge zu verfolgen, die von einer oder mehreren Transaktionen gesperrt werden. Wenn die Hash-Tabelle viele Map-Einträge enthält, kann durch die Verwendung weiterer Sperr-Buckets eine bessere Leistung erzielt wer-

den. Das Risiko von Java-Synchronisationskollisionen sinkt mit zunehmender Anzahl an Buckets. Werden zusätzliche Buckets verwendet, sind auch mehr gemeinsame Zugriffe möglich. Die vorherigen Beispiele haben veranschaulicht, wie eine Anwendung die Anzahl der Sperr-Buckets für eine bestimmte BackingMap-Instanz festlegen kann.

Um zu verhindern, dass eine Ausnahme des Typs "java.lang.IllegalStateException" ausgelöst wird, muss die Methode "setNumberOfLockBuckets" vor dem Aufruf der Methode "initialize" bzw. "getSession" in der ObjectGrid-Instanz aufgerufen werden. Der Methodenparameter "setNumberOfLockBuckets" ist ein primitiver Java-Integer, der die Anzahl der zu verwendenden Sperr-Buckets angibt. Die Verwendung einer Primzahl gewährleistet eine gleichmäßige Verteilung der Map-Einträge auf die Sperr-Buckets. Ein guter Ausgangspunkt für das Erzielen der besten Leistung ist, die Anzahl der Sperr-Buckets auf ungefähr zehn Prozent der erwarteten Anzahl an BackingMap-Einträgen zu setzen.

LockDeadlockException

Im Folgenden sehen Sie ein Codebeispiel für das Abfangen der Ausnahme und die daraufhin angezeigte Nachricht.

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

Das Ergebnis ist wie folgt:

```
com.ibm.websphere.objectgrid.plugins.LockDeadlockException: _Message
```

Diese Nachricht stellt die Zeichenfolge dar, die als Parameter übergeben wird, wenn die Ausnahme erstellt und ausgelöst wird.

Ursache für die Ausnahme

Die meisten Deadlock-Ausnahmen treten bei der Verwendung der pessimistischen Sperrstrategie ein, wenn zwei separate Client jeweils eine gemeinsame Sperre für ein bestimmtes Objekt erhalten. Beide Clients versuchen, eine exklusive Sperre für dieses Objekt zu erhalten. In der folgenden Abbildung wird eine solche Situation, einschließlich der Transaktionsblöcke gezeigt, die für das Auslösen der Ausnahme verantwortlich sind.

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

Dies ist eine abstrakte Sicht der Vorgänge in Ihrem Programm beim Eintreten der Ausnahme. Diese Situation kann in einer Anwendung auftreten, in der viele Threads dieselbe ObjectMap aktualisieren. Im Folgenden sehen Sie ein Beispiel mit zwei Clients, die die Transaktionsblöcke ausführen, die in der vorherigen Abbildung veranschaulicht wurden.

Mögliche Lösungen

Die in Abbildung 1 dargestellte Situation kann eintreten, wenn mehrere Threads Transaktionen für eine bestimmte Map starten. In diesem Fall wird die Ausnahme ausgelöst, um eine Blockierung Ihres Programms zu verhindern. Sie können sich selbst benachrichtige und dem Catch-Block Code hinzufügen, um weitere Details

zur Ursache bereitzustellen. Da diese Ausnahme nur bei einer pessimistischen Sperrstrategie ausgelöst wird, ist eine einfache Lösung die Umstellung auf eine optimistische Sperrstrategie. Wenn Sie jedoch eine pessimistische Sperrstrategie benötigen, können Sie anstelle der Methode "get" die Methode "getForUpdate" verwenden. Auf diese Weise werden die Ausnahmen für die zuvor beschriebene Situation verhindert.

Sperrstrategien

Die folgenden Sperrstrategien sind verfügbar: PESSIMISTIC (pessimistisch), OPTIMISTIC (optimistisch) und NONE (Ohne). Bei der Auswahl einer Sperrstrategie müssen Sie Aspekte wie den Prozentsatz der einzelnen Typen von Operationen, die potenzielle Verwendung eines Loaders usw. berücksichtigen.

Sperren werden über Transaktionen gebunden. Sie können die folgenden Sperreinstellungen angeben:

- **Keine Sperren**: Die Ausführung ohne Sperren ist die schnellste. Wenn Sie schreibgeschützte Daten verwenden, benötigen Sie möglicherweise keine Sperren.
- **Pessimistisches Sperren**: Es werden Sperren für Einträge angefordert, die so lange gehalten werden, bis die Transaktion festgeschrieben wird. Diese Sperrstrategie bietet eine gute Konsistenz, geht aber zu Lasten des Durchsatzes.
- **Optimistisches Sperren**: Erstellt eine Vorher-Kopie jedes Datensatzes, den die Transaktion berührt, und vergleicht diese mit den aktuellen Eintragswerten, wenn die Transaktion festgeschrieben wird. Wenn die Vorher-Kopie und der Eintragswert nicht übereinstimmen, wird die Transaktion rückgängig gemacht. Es werden keine Sperren bis zur Festschreibungszeit gehalten. Diese Sperrstrategie unterstützt einen besseren gemeinsamen Zugriff als die pessimistische Strategie, birgt aber das Risiko von Transaktions-Rollbacks und Speicherkosten für die zusätzliche Kopie des Eintrags.

Legen Sie die Sperrstrategie in der BackingMap fest. Es ist nicht möglich, die Sperrstrategie für jede einzelne Transaktion zu ändern. Im Folgenden sehen Sie ein Beispiel-XML-Snippet, das veranschaulicht, wie der Sperrmodus in einer Map über die XML-Datei festgelegt wird, und in dem davon ausgegangen wird, dass `cc` der Namespace für `objectgrid/config` ist:

```
<cc:backingMap name="RuntimeLifespan" lockStrategy="PESSIMISTIC" />
```

Pessimistisches Sperren

Verwenden Sie die pessimistische Sperrstrategie für Maps mit Lese-/Schreibzugriff, wenn keine anderen Sperrstrategien möglich sind. Wenn eine ObjectGrid-Map für die Verwendung der pessimistischen Sperrstrategie konfiguriert ist, wird eine pessimistische Transaktionssperre für einen Map-Eintrag angefordert, wenn eine Transaktion den Eintrag zum ersten Mal aus der BackingMap abruft. Die pessimistische Sperre wird so lange gehalten, bis die Anwendung die Transaktion abschließt. Gewöhnlich wird die pessimistische Sperrstrategie in den folgenden Situationen verwendet:

- Die BackingMap ist mit oder ohne Loader (Ladeprogramm) konfiguriert, und es sind keine Versionsinformationen verfügbar.
- Die BackingMap wird direkt von einer Anwendung verwendet, die Hilfe von eXtreme Scale für die Steuerung des gemeinsamen Zugriffs benötigt.
- Es sind Versionsinformationen verfügbar, aber Aktualisierungstransaktionen für die Einträge in der BackingMap lösen häufig Kollisionen aus, was zu Fehlern bei der optimistische Aktualisierung führt.

Da die pessimistische Sperrstrategie die größten Auswirkungen auf Leistung und Skalierbarkeit hat, sollte diese Strategie nur für Maps mit Lese-/Schreibzugriff verwendet werden, wenn keine anderen Sperrstrategien angewendet werden können, z. B., wenn häufig Fehler bei der optimistischen Aktualisierung auftreten oder wenn die Wiederherstellung nach einem Fehler bei einer optimistischen Aktualisierung nur schwer für eine Anwendung durchzuführen ist.

Optimistisches Sperren

Bei der optimistischen Sperrstrategie wird davon ausgegangen, dass zwei gleichzeitig ausgeführte Transaktionen niemals versuchen, denselben Map-Eintrag zu aktualisieren. Aufgrund dieser Annahme müssen die Sperren nicht für den gesamte Lebenszyklus der Transaktion gehalten werden, da es unwahrscheinlich ist, dass mehrere Transaktionen einen Map-Eintrag gleichzeitig aktualisieren. Die optimistische Sperrstrategie wird gewöhnlich in den folgenden Situationen verwendet:

- Eine BackingMap ist mit oder ohne Loader (Ladeprogramm) konfiguriert, und es sind Versionsinformationen verfügbar.
- Es werden hauptsächlich nur Transaktionen für eine BackingMap ausgeführt, die Leseoperationen durchführen. Einfüge-, Aktualisierungs- und Entfernungsoperationen für Map-Einträge finden in der BackingMap nur selten statt.
- Eine BackingMap wird häufiger eingefügt, aktualisiert oder entfernt, als sie gelesen wird, aber es finden nur selten Kollisionen zwischen den Transaktionen statt, die sich auf denselben Map-Eintrag beziehen.

Wie bei der pessimistischen Sperrstrategie bestimmen die Methoden in der Schnittstelle "ObjectMap", wie eXtreme Scale automatisch versucht, eine Sperre für den Map-Eintrag anzufordern, auf den zugegriffen wird. Es bestehen jedoch die folgenden Unterschiede zwischen der pessimistischen und der optimistischen Sperrstrategie:

- Wie bei der pessimistischen Sperrstrategie wird bei der optimistischen Sperrstrategie beim Aufruf der Methoden "get" und "getAll" eine S-Sperre angefordert. Beim optimistischen Sperren wird die S-Sperre jedoch nicht bis zum Abschluss der Transaktion gehalten. Stattdessen wird die S-Sperre freigegeben, bevor die Methode zur Anwendung zurückkehrt. Die Anforderung der Sperre hat den Zweck, dass eXtreme Scale sicherstellen kann, dass nur festgeschriebene Daten von anderen Transaktionen für die aktuelle Transaktion sichtbar sind. Nachdem eXtreme Scale sichergestellt hat, dass die Daten festgeschrieben wurden, wird die S-Sperre freigegeben. Während der Festschreibung wird eine optimistische Versionsprüfung durchgeführt, um sicherzustellen, dass der Map-Eintrag nicht von einer anderen Transaktion geändert wurde, nachdem die aktuelle Transaktion die S-Sperre freigegeben hat. Wenn ein Eintrag nicht aus der Map abgerufen wird, bevor er aktualisiert, ungültig gemacht oder gelöscht wird, ruft die Laufzeitumgebung von eXtreme Scale den Eintrag implizit aus der Map ab. Diese implizite get-Operation wird durchgeführt, um den aktuellen Wert abzurufen, den der Eintrag zum Zeitpunkt der Änderungsanforderung hatte.
- Anders als bei der pessimistischen Sperrstrategie werden die Methoden "getForUpdate" und "getAllForUpdate" bei der optimistischen Sperrstrategie genauso wie die Methoden "get" und "getAll" behandelt, d. h., beim Starten der Methode wird eine S-Sperre angefordert, die dann freigegeben wird, bevor die Methode zur Anwendung zurückkehrt.

Alle anderen ObjectMap-Methoden werden genauso behandelt, wie es bei der pessimistischen Sperrstrategie der Fall ist, d. h., wenn die Methode "commit" aufgerufen wird, wird eine X-Sperre für jeden Map-Eintrag angefordert, der eingefügt, ak-

tualisiert, entfernt, angerührt oder ungültig gemacht wurde, und diese X-Sperre wird so lange gehalten, bis die Commit-Verarbeitung der Transaktion abgeschlossen ist.

Bei der optimistischen Sperrstrategie wird davon ausgegangen, dass gleichzeitig ausgeführte Transaktionen nicht versuchen, denselben Map-Eintrag zu aktualisieren. Aufgrund dieser Annahme müssen die Sperren nicht für die gesamte Lebensdauer der Transaktion gehalten werden, da es unwahrscheinlich ist, dass mehrere Transaktionen einen Map-Eintrag gleichzeitig aktualisieren. Da eine Sperre jedoch nicht gehalten wird, ist es potenziell möglich, dass eine andere gleichzeitig ausgeführte Transaktion den Map-Eintrag ändert, nachdem die aktuelle Transaktion ihre S-Sperre freigegeben hat.

Zur Behandlung dieser Möglichkeit ruft eXtreme Scale während der Festschreibung eine X-Sperre ab und führt eine optimistische Versionsprüfung durch, um sicherzustellen, dass der Map-Eintrag nicht von einer anderen Transaktion geändert wurde, nachdem die aktuelle Transaktion den Map-Eintrag aus der BackingMap gelesen hat. Wenn der Map-Eintrag von einer anderen Transaktion geändert wurde, fällt die Versionsprüfung negativ aus, und es wird eine Ausnahme des Typs "OptimisticCollisionException" ausgegeben. Diese Ausnahme erzwingt ein Rollback der aktuellen Transaktion, und die Anwendung muss die vollständige Transaktion wiederholen. Die optimistische Sperrstrategie ist sehr hilfreich, wenn eine Map hauptsächlich nur gelesen wird und es unwahrscheinlich ist, dass gleichzeitige Aktualisierungen an demselben Map-Eintrag vorgenommen werden.

Ohne Sperren

Wenn eine BackingMap ohne Sperrstrategie konfiguriert ist, werden keine Transaktionssperren für einen Map-Eintrag angefordert.

Dies ist hilfreich, wenn eine Anwendung ein Persistenzmanager ist, wie z. B. ein EJB-Container, oder wenn eine Anwendung Hibernate für das Abrufen persistenter Daten verwendet. In diesem Szenario ist die BackingMap ohne Loader konfiguriert, und der Persistenzmanager verwendet die BackingMap als Datencache. Der Persistenzmanager übernimmt die Steuerung des gemeinsamen Zugriffs für Transaktionen, die auf dieselben Map-Einträge zugreifen.

Für die Steuerung des gemeinsamen Zugriffs muss WebSphere eXtreme Scale keine Transaktionssperren anfordern. In dieser Situation wird davon ausgegangen, dass der Persistenzmanager seine Transaktionssperren nicht freigibt, bevor die ObjectGrid-Map mit festgeschriebenen Änderungen aktualisiert wurde. Wenn der Persistenzmanager seine Sperren freigibt, muss eine pessimistische oder optimistische Sperrstrategie verwendet werden. Angenommen, der Persistenzmanager eines EJB-Containers aktualisiert eine ObjectGrid-Map mit Daten, die in der vom EJB-Container verwalteten Transaktion festgeschrieben wurden. Wenn die Aktualisierung der ObjectGrid-Map stattfindet, bevor der Persistenzmanager seine Transaktionssperren freigibt, können Sie die Strategie ohne Sperren verwenden. Wenn die Aktualisierung der ObjectGrid-Map stattfindet, nachdem der Persistenzmanager seine Transaktionssperren freigibt, müssen Sie die optimistische oder die pessimistische Sperrstrategie verwenden.

Ein weiteres Szenario, in dem die Strategie ohne Sperren verwendet werden kann, ist das, wenn die Anwendung eine BackingMap direkt verwendet und ein Loader für die Map konfiguriert ist. In diesem Szenario verwendet der Loader die Unterstützung für die Steuerung des gemeinsamen Zugriffs, die von einem Verwaltungssystem für relationale Datenbanken bereitgestellt wird, indem er entweder Java

Database Connectivity (JDBC) oder Hibernate für den Zugriff auf die Daten in einer relationalen Datenbank verwendet. Die Loader-Implementierung kann einen optimistischen oder pessimistischen Ansatz verwenden. Mit einem Loader, der einen optimistischen Sperr- oder Versionssteuerungsansatz verwendet, kann die höchste Anzahl gemeinsamer Zugriffe und die höchste Leistung erzielt werden. Weitere Informationen zur Implementierung eines optimistischen Sperransatzes finden Sie im Abschnitt "OptimisticCallback" in den Hinweisen zu Loadern im *Administratorhandbuch*. Wenn Sie einen Loader verwenden, der die Unterstützung für pessimistisches Sperren eines zugrunde liegenden Back-Ends verwendet, können Sie den Parameter "forUpdate" verwenden, der an die Methode "get" der Schnittstelle "Loader" übergeben wird. Setzen Sie diesen Parameter auf "true", wenn die Methode "getForUpdate der Schnittstelle "ObjectMap von der Anwendung zum Abrufen der Daten verwendet wird. Der Loader kann diesen Parameter verwenden, um festzustellen, ob eine aktualisierbare Sperre für die Zeile, die gelesen wird, angefordert werden muss. DB2 fordert beispielsweise eine aktualisierbare Sperre an, wenn eine SQL-Anweisung SELECT eine Klausel `FOR UPDATE` enthält. Dieser Ansatz bietet dieselben Präventionsmechanismen für Deadlocks, die im Abschnitt „Pessimistisches Sperren” auf Seite 167 beschrieben sind.

Weitere Informationen finden Sie im Abschnitt zur Behandlung von Sperren im *Programmierhandbuch* bzw. zum Sperren von Map-Einträgen im *Administratorhandbuch*.

JMS für Verteilung von Transaktionsänderungen

Verwenden Sie Java Message Service (JMS) für die Verteilung von Transaktionsänderungen zwischen verschiedenen Schichten und in Umgebungen auf heterogenen Plattformen.

JMS ist ein ideales Protokoll für die Verteilung von Änderungen zwischen verschiedenen Schichten und in Umgebungen auf heterogenen Plattformen. Einige Anwendungen, die eXtreme Scale verwenden, können beispielsweise in IBM WebSphere Application Server Community Edition, Apache Geronimo oder Apache Tomcat implementiert sein, wohingegen andere Anwendungen in WebSphere Application Server Version 6.x ausgeführt werden. JMS eignet sich optimal für die Verteilung von Änderungen zwischen eXtreme-Scale-Peers in diesen unterschiedlichen Umgebungen. Der Nachrichtentransport des High Availability Manager ist sehr schnell, kann aber nur Änderungen an Java Virtual Machines verteilen, die sich in derselben Stammgruppe befinden. JMS ist zwar langsamer, unterstützt aber die gemeinsame Nutzung eines ObjectGrids durch größere und unterschiedlichere Gruppen von Anwendungsclients. JMS ist ideal, wenn Daten in einem ObjectGrid von einem Swing-Fat-Client und einer in WebSphere Extended Deployment implementierten Anwendung gemeinsam genutzt werden.

Der integrierte Mechanismus für die Inaktivierung von Clients und die Peer-to-Peer-Replikation sind Beispiele für JMS-basierte Verteilung von Transaktionsänderungen. Weitere Informationen finden Sie in der Beschreibung der Peer-to-Peer-Replikation mit JMS im *Administratorhandbuch*.

JMS implementieren

JMS wird für die Verteilung von Transaktionsänderungen über ein Java-Objekt implementiert, das sich wie ein ObjectGridEventListener verhält. Dieses Objekt kann den Status auf die folgenden vier Arten weitergeben:

1. Invalidate: (Ungültigmachen) Jeder Eintrag, der entfernt, aktualisiert oder gelöscht wird, wird in allen Peer-JVMs entfernt, wenn diese die Nachricht empfangen.
2. Invalidate conditional: (Bedingtes Ungültigmachen) Der Eintrag wird nur entfernt, wenn die lokale Version kleiner-gleich der Version im Veröffentlichungskomponente (Publisher) ist.
3. Push: (Übertragung mit Push) Jeder Eintrag, der entfernt, aktualisiert, gelöscht oder eingefügt wurde, wird in allen Peer-JVMs hinzugefügt bzw. überschrieben, wenn diese die JMS-Nachricht empfangen.
4. Push conditional: (Bedingte Übertragung mit Push) Der Eintrag wird auf Empfangsseite nur dann aktualisiert bzw. hinzugefügt, wenn der lokale Eintrag älter ist als die Version, die veröffentlicht wird.

Auf zu veröffentlichende Änderungen warten

Das Plug-in implementiert die Schnittstelle "ObjectGridEventListener", um das Ereignis "transactionEnd" abzufangen. Wenn eXtreme Scale diese Methode aufruft, versucht das Plug-in die LogSequence-Liste für jede Map, die von der Transaktion angerührt wurde, in eine JMS-Nachricht zu konvertieren und anschließend zu veröffentlichen. Das Plug-in kann so konfiguriert werden, dass Änderungen für alle Maps oder einen Teil der Maps veröffentlicht werden. LogSequence-Objekte werden für die Maps verarbeitet, für die die Veröffentlichung aktiviert ist. Die ObjectGrid-Klasse "LogSequenceTransformer" serialisiert eine gefilterte LogSequence für jede Map in einen Datenstrom. Nachdem alle LogSequences in den Datenstrom serialisiert wurden, wird eine JMS-ObjectMessage erstellt und unter einem bekannten Topic veröffentlicht.

Auf JMS-Nachrichten warten und sie auf das lokale ObjectGrid anwenden

Dasselbe Plug-in startet auch einen Thread, der in einer Schleife ausgeführt wird und alle Nachrichten empfängt, die unter dem bekannten Topic veröffentlicht werden. Wenn eine Nachricht ankommt, wird der Nachrichteninhalt an die Klasse "LogSequenceTransformer" übergeben, wo sie in eine Gruppe von LogSequence-Objekten konvertiert wird. Anschließend wird eine Transaktion ohne Durchschreiben (no-write-through) gestartet. Jedes LogSequence-Objekt wird an die Methode "Session.processLogSequence" übergeben, die die lokalen Maps mit den Änderungen aktualisiert. Die Methode "processLogSequence" erkennt den Verteilungsmodus. Die Transaktion wird festgeschrieben, und der lokale Cache enthält die Änderungen. Weitere Informationen zur Verwendung von JMS für die Verteilung von Transaktionsänderungen finden Sie in den Informationen zum Verteilen von Änderungen zwischen Peer-Java-Virtual-Machines im *Administratorhandbuch*.

Einzelpartitionstransaktionen und Grid-übergreifende Partitionstransaktionen

Der Hauptunterschied zwischen WebSphere eXtreme Scale und traditionellen Datenspeicherlösungen wie relationalen oder speicherinternen Datenbanken ist die Verwendung der Partitionierung, die eine lineare Skalierung des Caches ermöglicht. Die wichtigen Transaktionstypen, die berücksichtigt werden müssen, sind Einzelpartitionstransaktionen und Grid-übergreifende Partitionstransaktionen.

Im Allgemeinen können Interaktionen mit dem Cache, wie im Folgenden beschrieben, in die Kategorien "Einzelpartitionstransaktionen" und "Grid-übergreifende Partitionstransaktionen" eingeteilt werden.

Einzelpartitionstransaktionen

Einzelpartitionstransaktionen sind die vorzuziehende Methode für die Interaktion mit Caches in WebSphere eXtreme Scale. Wenn eine Transaktion auf eine Einzelpartition beschränkt ist, ist sie standardmäßig auf eine einzelne Java Virtual Machine und damit auf einen einzelnen Servercomputer beschränkt. Ein Server kann *M* dieser Transaktionen pro Sekunde ausführen, und wenn Sie *N* Computer haben, sind `M*N` Transaktionen pro Sekunde möglich. Wenn sich Ihr Geschäft erweitert und Sie doppelt so viele dieser Transaktionen pro Sekunde ausführen müssen, können Sie *N* verdoppeln, indem Sie weitere Computer kaufen. Auf diese Weise können Sie Kapazitätsanforderungen erfüllen, ohne die Anwendung zu ändern, Hardware zu aktualisieren oder die Anwendung außer Betrieb zu nehmen.

Zusätzlich zu der Möglichkeit, den Cache so signifikant skalieren zu können, maximieren Einzelpartitionstransaktionen auch die Verfügbarkeit des Caches. Jede Transaktion ist nur von einem einzigen Computer abhängig. Jeder der anderen `(N-1)` Computer kann ausfallen, ohne den Erfolgt oder die Antwortzeit der Transaktion zu beeinflussen. Wenn Sie also mit 100 Computern arbeiten und einer dieser Computer ausfällt, wird nur 1 Prozent der Transaktionen, die zum Zeitpunkt des Ausfalls dieses Servers unvollständig sind, rückgängig gemacht. Nach dem Ausfall des Servers verlagert WebSphere eXtreme Scale die Partitionen des ausgefallenen Servers auf die anderen 99 Computer. In diesem kurzen Zeitraum vor der Durchführung der Operation können die anderen 99 Computer weiterhin Transaktionen ausführen. Nur die Transaktionen, an denen die Partitionen beteiligt sind, die umgelagert werden, sind blockiert. Nach Abschluss des Failover-Prozesses ist der Cache mit 99 Prozent seiner ursprünglichen Durchsatzkapazität wieder vollständig betriebsbereit. Nachdem der ausgefallene Server ersetzt und der Ersatzserver dem Grid hinzugefügt wurde, kehrt der Cache zu einer Durchsatzkapazität von 100 Prozent zurück.

Grid-übergreifende Transaktionen

Was Leistung, Verfügbarkeit und Skalierbarkeit betrifft, sind Grid-übergreifende Transaktionen das Gegenteil von Einzelpartitionstransaktionen. Grid-übergreifende Transaktionen greifen auf jede Partition und damit auf jeden Computer in der Konfiguration zu. Jeder Computer im Grid wird aufgefordert, einige Daten zu suchen und anschließend das Ergebnis zurückzugeben. Die Transaktion kann erst abgeschlossen werden, nachdem jeder Computer geantwortet hat, und deshalb wird der Durchsatz des gesamten Grids durch den langsamsten Computer beschränkt. Das Hinzufügen von Computern macht den langsamsten Computer nicht schneller und verbessert damit auch nicht den Durchsatz des Caches.

Grid-übergreifende Transaktionen haben einen ähnlichen Effekt auf die Verfügbarkeit. Wenn Sie mit 100 Servern arbeiten und ein Server ausfällt, werden 100 Prozent der Transaktionen, die zum Zeitpunkt des Serverausfalls in Bearbeitung sind, rückgängig gemacht. Nach dem Ausfall des Servers beginnt WebSphere eXtreme Scale mit der Verlagerung der Partitionen des ausgefallenen Servers auf die anderen 99 Computer. In dieser Zeit, d. h. bis zum Abschluss des Failover-Prozesses, kann das Grid keine dieser Transaktionen verarbeiten. Nach Abschluss des Failover-Prozesses ist der Cache wieder betriebsbereit, aber mit verringerter Kapazität. Wenn jeder Computer im Grid 10 Partitionen bereitstellt, erhalten 10 der verbleibenden 99 Computer während des Failover-Prozesses mindestens eine zusätzliche Partition. Eine zusätzliche Partition erhöht die Arbeitslast dieses Computers um mindestens 10 Prozent. Da der Durchsatz des Grids in einer Grid-übergreifenden Transaktion auf den Durchsatz des langsamsten Computers beschränkt ist, reduziert sich der Durchsatz durchschnittlich um 10 Prozent.

Einzelpartitionstransaktionen sind im Hinblick auf die horizontale Skalierung mit einem verteilten, hoch verfügbaren Objektcache wie WebSphere eXtreme Scale den Grid-übergreifenden Transaktionen vorzuziehen. Die Maximierung der Leistung solcher Systemtypen erfordert die Verwendung von Techniken, die sich von den traditionellen relationalen Verfahren unterscheiden, aber Sie Grid-übergreifende Transaktionen in skalierbare Einzelpartitionstransaktionen konvertieren.

Bewährte Verfahren für die Erstellung skalierbarer Datenmodelle

Die bewährten Verfahren für die Erstellung skalierbarer Anwendungen mit Produkten wie WebSphere eXtreme Scale sind in zwei Kategorien einteilbar: Grundsätze und Implementierungstipps. Grundsätze sind Kernideen, die im Design der Daten selbst erfasst werden müssen. Es ist sehr unwahrscheinlich, dass sich eine Anwendung, die diese Grundsätze nicht einhält, problemlos skalieren lässt, selbst für ihre Haupttransaktionen. Implementierungstipps werden für problematische Transaktionen in einer ansonsten gut entworfenen Anwendung angewendet, die sich an die allgemeinen Grundsätze für skalierbare Datenmodelle hält.

Grundsätze

Einige wichtige Hilfsmittel für die Optimierung der Skalierbarkeit sind Basiskonzepte oder Grundsätze, die beachtet werden müssen.

Duplizieren an Stelle von Normalisieren

Der wichtigste Punkt, der bei Produkten wie WebSphere eXtreme Scale zu beachten ist, ist der, dass sie für die Verteilung von Daten auf sehr viele Computer konzipiert sind. Wenn das Ziel darin besteht, die meisten oder sogar alle Transaktionen auf einer einzelnen Partition auszuführen, muss das Datenmodelldesign sicherstellen, dass sich alle Daten, die die Transaktion unter Umständen benötigt, auf der Partition befinden. In den meisten Fällen kann dies nur durch Duplizierung der Daten erreicht werden.

Stellen Sie sich beispielsweise eine Anwendung wie ein Nachrichtenbrett vor. Zwei sehr wichtige Transaktionen für ein Nachrichtenbrett zeigen alle Veröffentlichungen eines bestimmten Benutzers und alle Veröffentlichungen unter einem bestimmten Topic an. Stellen Sie sich zunächst vor, wie diese Transaktionen mit einem normalisierten Datenmodell arbeiten, das einen Benutzerdatensatz, einen Topic-Datensatz und einen Veröffentlichungsdatensatz mit dem eigentlichen Text enthält. Wenn Veröffentlichungen mit Benutzerdatensätzen partitioniert werden, wird aus der Anzeige des Topics eine Grid-übergreifende Transaktion und umgekehrt. Topics und Benutzer können nicht gemeinsam partitioniert werden, da sie eine Viele-zu-viele-Beziehung haben.

Die beste Methode für die Skalierung dieses Nachrichtenbretts ist die Duplizierung der Veröffentlichungen, wobei eine Kopie mit dem Topic-Datensatz und eine Kopie mit dem Benutzerdatensatz gespeichert wird. Die anschließende Anzeige der Veröffentlichungen eines Benutzers ist eine Einzelpartitionstransaktion, die Anzeige der Veröffentlichungen unter einem Topic ist eine Einzelpartitionstransaktion, und die Aktualisierung oder das Löschen einer Veröffentlichung ist eine Transaktion, an der zwei Partitionen beteiligt sind. Alle drei Transaktionen können linear skaliert werden, wenn die Anzahl der Computer im Grid zunimmt.

Skalierbarkeit an Stelle von Ressourcen

Die größte Hindernis, das beim Einsatz denormalisierter Datenmodell überwunden werden muss, sind die Auswirkungen, die diese Modell auf

Ressourcen haben. Die Verwaltung von zwei, drei oder mehr Kopien derselben Daten kann den Anschein erwecken, dass zu viele Ressourcen benötigt werden, als dass dieser Ansatz praktikabel ist. Wenn Sie mit diesem Szenario konfrontiert werden, berücksichtigen Sie die folgenden Fakten: Hardwareressourcen werden von Jahr zu Jahr billiger. Zweitens, und noch wichtiger, mit WebSphere eXtreme Scale fallen die meisten verborgenen Kosten weg, die bei der Implementierung weiterer Ressourcen anfallen.

Messen Sie Ressourcen anhand der Kosten und nicht anhand von Computerbegriffen wie Megabyte oder Prozessoren. Datenspeicher, die mit normalisierten relationalen Daten abreiten, müssen sich im Allgemeinen auf demselben Computer befinden. Diese erforderliche Co-Location bedeutet, dass ein einzelner größerer Unternehmenscomputer an Stelle mehrerer kleinerer Computer erworben werden muss. Bei Unternehmenshardware ist es nicht unüblich, dass ein einziger Computer, der in der Lage ist, eine Million Transaktionen pro Sekunde zu verarbeiten, mehr kostet als 10 Computer zusammen, die in der Lage sind, jeweils 100.000 Transaktionen pro Sekunden auszuführen.

Außerdem fallen Geschäftskosten für die Implementierung der Ressourcen an. Irgendwann reicht die Kapazität in einem expandierenden Unternehmen einfach nicht mehr aus. In diesem Fall setzen Sie den Betrieb entweder aus, während Sie die Umstellung auf einen größeren und schnelleren Computer durchführen, oder Sie erstellen eine zweite Produktionsumgebung, auf die Sie den Betrieb dann umstellen können. In beiden Fällen fallen zusätzliche Kosten durch das ausgefallene Geschäft oder durch die Verwaltung der doppelten Kapazität in der Übergangsphase an.

Mit WebSphere eXtreme Scale muss die Anwendung nicht heruntergefahren werden, um Kapazität hinzuzufügen. Wenn die Prognose für Ihr Geschäft lautet, dass Sie 10 Prozent mehr Kapazität für das kommende Jahr benötigen, erhöhen Sie die Anzahl der Computer im Grid um 10 Prozent. Sie können diese Erweiterung ohne Anwendungsausfallzeit und ohne den Einkauf von Kapazitäten durchführen, die Sie hinterher nicht mehr benötigen.

Datenkonvertierungen vermeiden

Wenn Sie WebSphere eXtreme Scale verwenden, müssen Daten in einem Format gespeichert werden, das von der Geschäftslogik direkt konsumiert werden kann. Die Aufteilung der Daten in ein primitiveres Format ist kostenintensiv. Die Konvertierung muss durchgeführt werden, wenn die Daten geschrieben und wenn die Daten gelesen werden. Mit relationalen Datenbanken ist diese Konvertierung unumgänglich, weil die Daten letztendlich relativ häufig auf der Platte gespeichert werden, aber mit WebSphere eXtreme Scale fallen diese Konvertierungen weg. Der größte Teil der Daten wird im Hauptspeicher gespeichert und kann deshalb in genau dem Format gespeichert werden, das die Anwendung erfordert.

Durch die Einhaltung dieser einfachen Regel können Sie Ihre Daten dem ersten Grundsatz entsprechend denormalisieren. Der gängigste Konvertierungstyp für Geschäftsdaten ist die JOIN-Operation, die erforderlich ist, um normalisierte Daten in eine Ergebnismenge zu konvertieren, die den Anforderungen der Anwendung entspricht. Durch die implizite Speicherung der Daten im richtigen Format werden diese JOIN-Operationen vermieden, und es entsteht ein denormalisiertes Datenmodell.

Unbegrenzte Abfragen vermeiden

Unbegrenzte Abfragen lassen sich nicht gut skalieren, egal, wie gut Sie Ihre Daten auch strukturieren. Verwenden Sie beispielsweise keine Transaktion, die eine Liste aller Einträge nach Wert sortiert abfragt. Diese Transaktion funktioniert möglicherweise, wenn die Gesamtanzahl der Einträge bei 1000 liegt, aber wenn die die Gesamtanzahl der Einträge 10 Million erreicht, gibt die Transaktion alle 10 Millionen Einträge zurück. Wenn Sie diese Transaktion ausführen, sind zwei Ergebnisse am wahrscheinlichsten: Die Transaktion überschreitet das zulässige Zeitlimit, oder im Client tritt eine abnormale Speicherbedingung auf.

Die beste Option ist, die Geschäftslogik so zu ändern, dass nur die Top 10 oder 20 Einträge zurückgegeben werden können. Durch diese Änderung der Logik bleibt die Größe der Transaktion verwaltbar, unabhängig davon, wie viele Einträge im Cache enthalten sind.

Schema definieren

Der Hauptvorteil der Normalisierung von Daten ist der, dass sich das Datenbanksystem im Hintergrund um die Datenkonsistenz kümmern kann. Wenn Daten für Skalierbarkeit denormalisiert werden, ist diese automatische Verwaltung der Datenkonsistenz nicht mehr möglich. Sie müssen ein Datenmodell implementieren, das auf der Anwendungsebene oder als Plug-in für das verteilte Grid arbeiten kann, um die Datenkonsistenz zu gewährleisten.

Stellen Sie sich das Beispiel mit dem Nachrichtenbrett vor. Wenn eine Transaktion eine Veröffentlichung aus einem Topic entfernt, muss das Veröffentlichungsduplikat im Benutzerdatensatz entfernt werden. Ohne ein Datenmodell ist es möglich, dass ein Entwickler den Anwendungscode zum Entfernen der Veröffentlichung aus dem Topic schreibt und vergisst, die Veröffentlichung aus dem Benutzerdatensatz zu entfernen. Wenn der Entwickler jedoch ein Datenmodell verwendet, anstatt direkt mit dem Cache zu interagieren, kann die Methode "removePost" im Datenmodell die Benutzer-ID aus der Veröffentlichung extrahieren, den Benutzerdatensatz suchen und das Veröffentlichungsduplikat im Hintergrund entfernen.

Alternativ können Sie einen Listener implementieren, der auf der tatsächlichen Partition ausgeführt wird, das Topic überwacht und bei einer Änderung des Topics Benutzerdatensatz automatisch anpasst. Ein Listener kann von Vorteil sein, weil die Anpassung am Benutzerdatensatz lokal vorgenommen werden kann, wenn die Partition den Benutzerdatensatz enthält. Selbst wenn sich der Benutzerdatensatz auf einer anderen Partition befindet, findet die Transaktion zwischen Servern und nicht zwischen dem Client und dem Server statt. Die Netzverbindung zwischen Servern ist wahrscheinlich schneller als die Netzverbindung zwischen dem Client und dem Server.

Konkurrenzsituationen vermeiden

Szenarien wie die Verwendung eines globalen Zählers vermeiden. Das Grid kann nicht skaliert werden, wenn ein einzelner Datensatz im Vergleich mit den restlichen Datensätzen unverhältnismäßig oft verwendet wird. Die Leistung des Grids wird durch die Leistung des Computers beschränkt, der diesen Datensatz enthält.

Versuchen Sie in solchen Situationen, den Datensatz aufzuteilen, so dass er pro Partition verwaltet wird. Stellen Sie sich beispielsweise eine Transaktion vor, die die Gesamtanzahl der Einträge im verteilten Cache zurückgibt. Anstatt jede Einfüge- und Entfernungsoperation auf einen einzelnen Datensatz zugreifen zu lassen, dessen Zähler sich erhöht, können Sie einen Liste-

ner auf jeder Partition einsetzen, der die Einfüge- und Entfernungsoperation verfolgt. Mit dieser Listener-Verfolgung können aus Einfüge- und Entfernungsoperationen Einzelpartitionsoperationen werden.

Das Lesen des Zählers wird zu einer Grid-übergreifenden Operation, aber die Leseoperation war bereits vorher genauso ineffizient wie eine Grid-übergreifende Operation, weil ihre Leistung an die Leistung des Computers gebunden war, auf dem sich der Datensatz befindet.

Implementierungstipps

Zum Erreichen der besten Skalierbarkeit können Sie außerdem die folgenden Tipps beachten.

Umgekehrte Suchindizes verwenden

Stellen Sie sich ein ordnungsgemäß denormalisiertes Datenmodell vor, in dem Kundendatensätze auf der Basis der Kunden-ID partitioniert werden. Diese Partitionierungsmethode ist die logische Option, weil nahezu jede Geschäftsoperation, die mit dem Kundendatensatz ausgeführt wird, die Kunden-ID verwendet. Eine wichtige Transaktion, in der die Kunden-ID jedoch nicht verwendet wird, ist die Anmeldetransaktion. Es ist üblich, dass Benutzernamen oder E-Mail-Adressen für die Anmeldung verwendet werden, und keine Kunden-IDs.

Der einfache Ansatz für das Anmeldeszenario ist die Verwendung einer Grid-übergreifenden Transaktion, um den Kundendatensatz zu suchen. Wie zuvor erläutert, ist dieser Ansatz nicht skalierbar.

Die nächste Option ist die Partitionierung nach Benutzernamen oder E-Mail-Adressen. Diese Option ist nicht praktikabel, da alle Operationen, die auf der Kunden-ID basieren, zu Grid-übergreifenden Transaktionen werden. Außerdem möchten die Kunden auf Ihrer Site möglicherweise ihren Benutzernamen oder ihre E-Mail-Adresse ändern. Produkte wie WebSphere eXtreme Scale benötigen den Wert, der für die Partitionierung der Daten verwendet wird, um konstant zu bleiben.

Die richtige Lösung ist die Verwendung eines umgekehrten Suchindex. Mit WebSphere eXtreme Scale kann ein Cache in demselben verteilten Grid wie der Cache erstellt werden, der alle Benutzerdatensätze enthält. Dieser Cache ist hoch verfügbar, partitioniert und skalierbar. Dieser Cache kann verwendet werden, um einen Benutzernamen oder eine E-Mail-Adresse einer Kunden-ID zuzuordnen. Dieser Cache verwandelt die Anmeldung in eine Operation, an der zwei Partitionen beteiligt sind, und nicht in eine Grid-übergreifende Transaktion. Dieses Szenario ist zwar nicht so effektiv wie eine Einzelpartitionstransaktion, aber der Durchsatz nimmt linear mit steigender Anzahl an Computern zu.

Berechnung beim Schreiben

Die Generierung häufig berechneter Werte wie Durchschnittswerte oder Summen kann kostenintensiv sein, weil bei diesen Operationen gewöhnlich sehr viele Einträge gelesen werden müssen. Da in den meisten Anwendungen mehr Leseoperationen als Schreiboperationen ausgeführt werden, ist es effizient, diese Werte beim Schreiben zu berechnen und das Ergebnis anschließend im Cache zu speichern. Durch dieses Verfahren werden Leseoperationen schneller und skalierbarer.

Optionale Felder

Stellen Sie sich einen Benutzerdatensatz vor, der eine geschäftliche Telefonnummer, eine private Telefonnummer und eine Handy-Nummer enthält. Ein Benutzer kann alle, keine oder eine beliebige Kombination dieser Nummern haben. Wenn die Daten normalisiert sind, sind eine Benutzertabelle und eine Telefonnummerntabelle vorhanden. Die Telefonnummern für einen bestimmten Benutzer können über eine JOIN-Operation zwischen den beiden Tabellen ermittelt werden.

Die Denormalisierung dieses Datensatzes erfordert keine Datenduplizierung, weil die meisten Benutzer nicht dieselben Telefonnummern haben. Stattdessen müssen freie Bereiche im Benutzerdatensatz zulässig sein. Anstatt eine Telefonnummerntabelle zu verwenden, können Sie jedem Benutzerdatensatz drei Attribute hinzufügen, eines für jeden Telefonnummerntyp. Durch das Hinzufügen dieser Attribut wird die JOIN-Operation vermieden, und die Suche der Telefonnummern für einen Benutzer wird zu einer Einzelpartitionsoperation.

Verteilung von Viele-zu-viele-Beziehungen

Stellen Sie sich eine Anwendung, die Produkte und die Länden verfolgt, in denen die Produkte verkauft werden. Ein Produkt wird in vielen Läden verkauft, und ein Laden verkauft viele Produkte. Angenommen, diese Anwendung verfolgt 50 große Einzelhändler. Jedes Produkt wird in maximal 50 Länden verkauft, wobei jeder Laden Tausende von Produkten verkauft.

Verwalten Sie eine Liste der Läden in der Produktentität (Anordnung A), anstatt eine Liste von Produkten in jeder Ladenentität zu verwalten (Anordnung B). Wenn Sie sich einige der Transaktionen ansehen, die diese Anwendung ausführen muss, ist leicht zu erkennen, warum Anordnung A skalierbarer ist.

Sehen Sie sich zuerst die Aktualisierungen an. Wenn bei Anordnung A ein Produkt aus dem Bestand eines Ladens entfernt wird, wird die Produktentität gesperrt. Enthält das Grid 10000 Produkte, muss nur 1/10000 des Grids gesperrt werden, um die Aktualisierung durchzuführen. Bei Anordnung B enthält das Grid nur 50 Länden, so dass 1/50 des Grids gesperrt werden muss, um die Aktualisierung durchzuführen. Obwohl beide Fälle als Einzelpartitionsoperationen eingestuft werden können, lässt sich Anordnung A effizienter skalieren.

Sehen Sie sich jetzt die Leseoperationen für Anordnung A an. Das Durchsuchen eines Ladens, in dem ein Produkt verkauft wird, ist eine Einzelpartitionsoperation, die skalierbar und schnell ist, weil die Transaktion nur einen kleine Datenmenge überträgt. Bei Anordnung B wird aus dieser Transaktion eine Grid-übergreifende Transaktion, weil auf jede Ladenentität zugegriffen werden muss, um festzustellen, ob das Produkt in diesem Laden verkauft wird. Daraus ergibt sich ein enormer Leistungsvorteil für Anordnung A.

Skalierung mit normalisierten Daten

Eine zulässige Verwendung von Grid-übergreifenden Transaktionen ist die Skalierung der Datenverarbeitung. Wenn ein Grid 5 Computer enthält und eine Grid-übergreifende Transaktion zugeteilt wird, die 100.000 Datensätze auf jedem Computer durchsucht, durchsucht diese Transaktion insgesamt 500.000 Datensätze. Wenn der langsamste Computer im Grid 10 dieser Transaktionen pro Sekunde ausführen kann, ist das Grid in der Lage, 5.000.000 Datensätze pro Sekunde zu durchsuchen. Wenn sich die Daten im Grid verdoppeln, muss jeder Computer 200.000 Datensätze durchsuchen, und jede Transaktion durchsucht insgesamt 1.000.000 Datensätze. Diese Da-

tenzunahme verringert den Durchsatz des langsamsten Computers auf 5 Transaktionen pro Sekunde und damit den Durchsatz des Grids auf 5 Transaktionen pro Sekunde. Das Grid durchsucht weiterhin 5.000.000 Datensätze pro Sekunde.

In diesem Szenario kann jeder Computer durch die Verdopplung der Computeranzahl zu seiner vorherigen Last von 100.000 Datensätzen zurückkehren, und der langsamste Computer kann wieder 10 dieser Transaktionen pro Sekunde verarbeiten. Der Durchsatz des Grids bleibt bei 10 Anforderungen pro Sekunde, aber jetzt verarbeitet jede Transaktion 1.000.000 Datensätze, so dass das Grid seine Kapazität in Bezug auf die Verarbeitung von Datensätzen auf 10.000.000 pro Sekunde verdoppelt hat.

Für Anwendungen wie Suchmaschinen, die sowohl in Bezug auf die Datenverarbeitung (angesichts der zunehmenden Größe des Internets) als auch in Bezug auf den Durchsatz (angesichts der zunehmenden Anzahl an Benutzern) skalierbar sein müssen, müssen Sie mehrere Grids mit einem Umlaufverfahren für die Anforderungen zwischen den Grids erstellen. Wenn Sie den Durchsatz erhöhen müssen, fügen Sie Computer und ein weiteres Grid für die Bearbeitung der Anforderungen hinzu. Wenn die Datenverarbeitung erhöht werden muss, fügen Sie weitere Computer hinzu, und halten Sie die Anzahl der Grids konstant.

Kapitel 7. Übersicht über die Sicherheit

WebSphere eXtreme Scale kann den Datenzugriff sichern, unter anderem durch Integration mit externen Sicherheitsprovidern.

Anmerkung: In einem vorhandenen nicht zwischengespeicherten Datenspeicher, z. B. einer Datenbank, haben Sie wahrscheinlich integrierte Sicherheitsfeatures, die Sie nicht aktiv konfigurieren oder aktivieren müssen. Nachdem Sie Ihre Daten jedoch mit eXtreme Scale zwischengespeichert haben, müssen Sie die daraus resultierende wichtige Tatsache berücksichtigen, dass die Sicherheitsfeatures Ihres Back-Ends nicht mehr wirksam sind. Sie können die Sicherheit von eXtreme Scale auf den erforderlichen Stufen konfigurieren, so dass Ihre neue zwischengespeicherte Datenarchitektur ebenfalls sicher ist.
Es folgt eine kurze Zusammenfassung der Sicherheitsfeatures von eXtreme Scale. Ausführlichere Informationen zur Konfiguration der Sicherheit finden Sie im *Administratorhandbuch* und im *Programmierhandbuch*.

Grundlegende Informationen zur verteilten Sicherheit

Die verteilte Sicherheit von eXtreme Scale basiert auf drei Schlüsselkonzepten:

Vertrauenswürdige Authentifizierung
: Die Möglichkeit, die Identität des Anforderers zu bestimmen. WebSphere eXtreme Scale unterstützt Client/Server- und Server/Server-Authentifizierung.

Berechtigung
: Die Möglichkeit, dem Anforderer Zugriffsberechtigungen zu erteilen. WebSphere eXtreme Scale unterstützt verschiedene Berechtigungen für verschiedene Operationen.

Sicherer Transport
: Die sichere Übertragung von Daten über ein Netz. WebSphere eXtreme Scale unterstützt die Protokolle Layer Security/Secure Sockets Layer (TLS/SSL).

Authentifizierung

WebSphere eXtreme Scale unterstützt ein verteiltes Client/Server-Framework. Eine Client/Server-Sicherheitsinfrastruktur ist verfügbar, um den Zugriff auf Server von eXtreme Scale zu sichern. Wenn der Server von eXtreme Scale beispielsweise eine Authentifizierung erfordert, muss ein Client von eXtreme Scale Berechtigungsnachweise für die Authentifizierung beim Server vorlegen. Diese Berechtigungsnachweise können eine Kombination von Benutzername und Kennwort, ein Clientzertifikat, ein Kerberos-Ticket oder Daten sein, die in einem zwischen Client und Server vereinbarten Format präsentiert werden.

Berechtigung

Berechtigungen von WebSphere eXtreme Scale basieren auf Subject-Objekten und Berechtigungen. Sie können Java Authentication and Authorization Services (JAAS) für die Berechtigung des Zugriffs verwenden, oder Sie können eine angepasste Lö-

sung wie Tivoli Access Manager (TAM) für die Behandlung der Berechtigungen integrieren. Die folgenden Berechtigungen können einem Client oder einer Gruppe erteilt werden:

Map-Berechtigung
Berechtigung zum Durchführen von Einfüge-, Lese-, Aktualisierungs-, Bereinigungs- oder Löschoperationen in Maps.

ObjectGrid-Berechtigung
Berechtigung zum Ausführen von Objekt- oder Entitätsabfragen und Datenstromabfragen für ObjectGrid-Objekte.

DataGrid-Agentenberechtigung
Berechtigung für die Implementierung von DataGrid-Agenten in einem ObjectGrid.

Serverseitige Map-Berechtigung
Berechtigung zum Replizieren einer Server-Map auf der Clientseite oder zum Erstellen eines dynamischen Index für die Server-Map.

Verwaltungsberechtigung
Berechtigung für die Ausführung von Verwaltungs-Tasks.

Transportsicherheit

Zum Sichern der Client/Server-Kommunikation unterstützt WebSphere eXtreme Scale TLS/SSL. Diese Protokolle bieten Sicherheit auf Transportebene mit Authentizität, Integrität und Vertraulichkeit für eine sichere Verbindung zwischen einem Client und einem Server von eXtreme Scale.

Grid-Sicherheit

In einer sicheren Umgebung muss ein Server in der Lage sein, die Authentizität eines anderen Servers zu prüfen. WebSphere eXtreme Scale verwendet für diesen Zweck einen Mechanismus mit Shared-Secret-Schlüsselzeichenfolgen. Dieser Shared-Secret-Schlüsselmechanismus gleicht einem gemeinsam genutzten Kennwort. Alle Server von eXtreme Scale stimmen der Verwendung einer gemeinsamen Shared-Secret-Zeichenfolge zu. Wenn ein Server dem Grid beitritt, wird er aufgefordert, diese Shared-Secret-Zeichenfolge vorzulegen. Wenn die Shared-Secret-Zeichenfolge des beitretenden Servers der Zeichenfolge im Masterserver entspricht, kann der Server dem Grid beitreten. Andernfalls wird die Beitrittsanforderung zurückgewiesen.

Das Senden einer Shared-Secret-Zeichenfolge als Klartext ist nicht sicher. Die Sicherheitsinfrastruktur von eXtreme Scale stellt ein SecureTokenManager-Plug-in bereit, über das der Server den geheimen Schlüssel vor dem Senden sichern kann. Sie können festlegen, wie die Sicherungsoperation implementiert wird. WebSphere eXtreme Scale stellt eine Implementierung bereit, in der die Sicherungsoperation so implementiert ist, dass das Shared Secret verschlüsselt und signiert wird.

JMX-Sicherheit (Java Management Extensions) in einer dynamischen Implementierungstopologie

Die JMX-MBean-Sicherheit wird in allen Versionen von eXtreme Scale unterstützt. Clients der Katalogserver-MBeans und Containerserver-MBeans können authentifiziert werden und auf die MBean-Operationen zugreifen.

Lokale Sicherheit von eXtreme Scale

Die lokale Sicherheit von eXtreme Scale unterscheidet sich vom verteilten eXtreme-Scale-Modell, weil die Anwendung direkt instanziiert wird und eine ObjectGrid-Instanz verwendet. Ihre Anwendung und eXtreme-Scale-Instanzen befinden sich in derselben Java Virtual Machine (JVM). Da es in diesem Modell kein Client/Server-Konzept gibt, wird die Authentifizierung nicht unterstützt. Ihre Anwendungen müssen ihre Authentifizierung selbst verwalten und anschließend das authentifizierte Subject-Objekt an eXtreme Scale übergeben. Der Berechtigungsmechanismus, der für das lokale Programmiermodell von eXtreme Scale verwendet wird, ist jedoch dasselbe wie beim Client/Server-Modell.

Konfiguration und Programmierung

Weitere Informationen zum Konfigurieren und Programmieren der Sicherheit finden Sie im *Administratorhandbuch* und im *Programmierhandbuch*.

Kapitel 8. Übersicht über REST-Datenservices

Der REST-Datenservice von WebSphere eXtreme Scale ist ein Java-HTTP-Service, der mit Microsoft WCF Data Services (offiziell ADO.NET Data Services) kompatibel ist und Open Data Protocol (OData) implementiert. Microsoft WCF Data Services ist mit dieser Spezifikation kompatibel, wenn Visual Studio 2008 SP1 und .NET Framework 3.5 SP1 verwendet werden.

Kompatibilitätsanforderungen

Der REST-Datenzugriff ermöglicht jedem HTTP-Client den Zugriff auf ein Daten-Grid. Der REST-Datenservice ist mit der Unterstützung der WCF Data Services kompatibel, die mit Microsoft .NET Framework 3.5 SP1 bereitgestellt wird. Anwendungen, die REST unterstützen, können mit den zahlreichen Tools, die im Lieferumfang von Microsoft Visual Studio 2008 SP1 enthalten sind, entwickelt werden. Die Abbildung enthält eine Übersicht über die Interaktion von WCF Data Services mit Clients und Datenbanken.

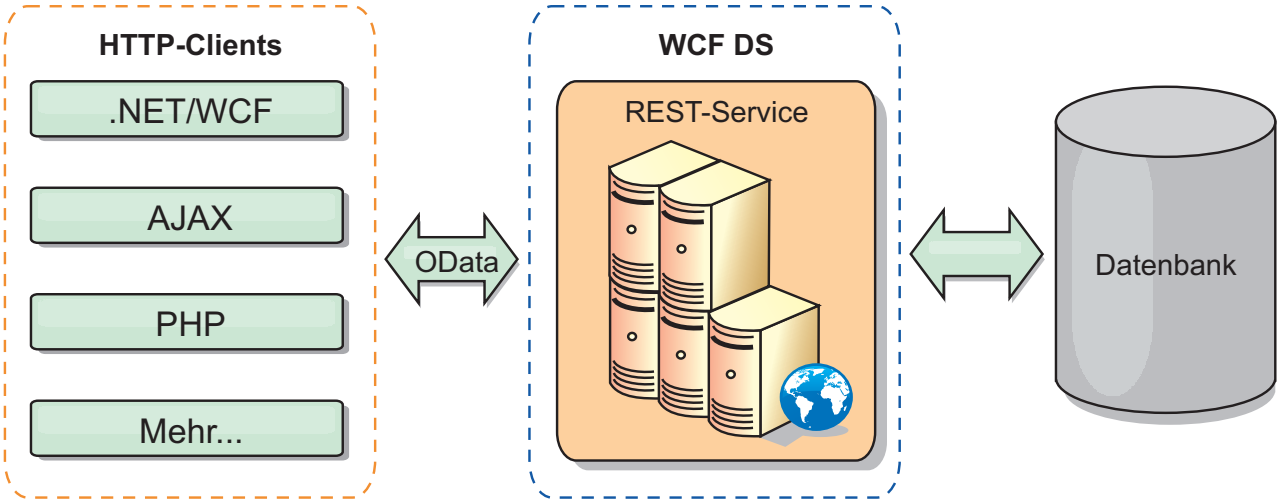


Abbildung 41. Microsoft WCF Data Services

WebSphere eXtreme Scale enthält einen umfassend ausgestatteten API-Satz für Java-Clients. Wie in der folgenden Abbildung gezeigt, ist der REST-Datenservice ein Gateway zwischen HTTP-Clients und dem eXtreme-Scale-Grid, das mit dem Grid über einen eXtreme-Scale-Client kommuniziert. Der REST-Datenservice ist ein Java-Servlet, das flexible Implementierungen für gängige JEE-Plattformen (Java Platform, Enterprise Edition) wie WebSphere Application Server unterstützt. Der REST-Datenservice kommuniziert mit dem eXtreme-Scale-Grid über die Java-APIs von WebSphere eXtreme Scale. Er unterstützt WCF-Data-Services-Clients und alle anderen Clients, die mit HTTP und XML kommunizieren können.

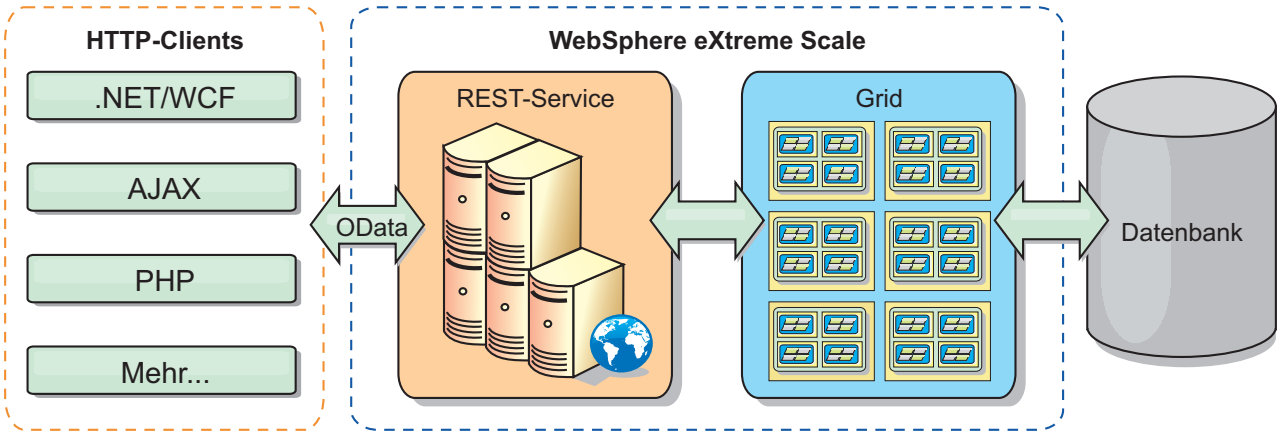


Abbildung 42. REST-Datenservice von WebSphere eXtreme Scale

Lesen Sie den Abschnitt „Muster und Lernprogramm zum REST-Datenservice” auf Seite 224, oder verwenden Sie die folgenden Links, um mehr über WCF Data Services zu erfahren.

- Microsoft WCF Data Services Developer Center
- ADO.NET Data Services overview on MSDN
- Whitepaper: Using ADO.NET Data Services
- Atom Publish Protocol: Data Services URI and Payload Extensions
- Conceptual Schema Definition File Format
- Entity Data Model for Data Services Packaging Format
- Open Data Protocol
- Open Data Protocol FAQ

Features

Diese Version des REST-Datenservice von eXtreme Scale unterstützt die folgenden Features:

- Automatische Modellierung von Entitäten der eXtreme-Scale-API "EntityManager" als Entitäten von WCF Data Services, die die folgende Unterstützung umfasst:
  - Konvertierung von Java-Datentypen in Typen des Entitätsdatenmodells
  - Unterstützung der Entitätszuordnung
  - Unterstützung der Zuordnung von Schemastammelementen und Schlüsseln, die für partitionierte Daten-Grids erforderlich ist

  Weitere Informationen finden Sie im Abschnitt Entitätsmodell.
- XML von Atom Publish Protocol (AtomPub oder APP) und Nutzdatenformat von JavaScript™ Object Notation (JSON)
- CRUD-Operationen (Create, Read, Update and Delete, Erstellen, Lesen, Aktualisieren und Löschen), die die entsprechenden HTTP-Anforderungsmethoden, POST, GET, PUT und DELETE, verwenden. Außerdem wird die Microsoft-Erweiterung MERGE unterstützt.
- Einfache Abfragen unter Verwendung von Filtern
- Stapelabruf- und Änderungssetanforderungen
- Unterstützung partitionierter Grids für hohe Verfügbarkeit
- Interoperabilität mit Clients der eXtreme-Scale-API "EntityManager"

- Unterstützung für Standard-JEE-Webserver
- Optimistisches Sperren bei gemeinsamen Zugriffen
- Benutzerberechtigung und -authentifizierung zwischen dem REST-Datenservice und dem eXtreme-Scale-Daten-Grid

Bekannte Probleme und Einschränkungen

- Getunnelte Anforderungen werden nicht unterstützt.

Kapitel 9. Übersicht über die Integration des Spring-Frameworks

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 Unterstützung von Spring-Erweiterungs-Beans und -Namespaces.

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 Bean com.ibm.ws.objectgrid.jpa.plugins.JPAPropFactoryImpl 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 das Sitzungsmanagement 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.

Kapitel 10. Lernprogramme, Beispiele und Mustercodes

Es sind mehrere Lernprogramme, Beispiele und Muster für WebSphere eXtreme Scale verfügbar.

Lernprogramme

Die folgenden Lernprogramme sind derzeit verfügbar:

- Lernprogramm zu ObjectMap
- „Lernprogramm zum EntityManager: Übersicht"

Beispiele

In den folgenden Abschnitten werden die Schlüsselfunktionen von WebSphere eXtreme Scale veranschaulicht.

- Sehen Sie sich das Beispiel zur Daten-Grid-API im *Programmierhandbuch* an.
- Sehen Sie sich die Einzelheiten zur Konfiguration lokaler Implementierungen im *Administratorhandbuch* an.

Muster

Mustercodes, die veranschaulichen, wie die ObjectGrid-APIs in verschiedenen Umgebungen verwendet werden, sind im Lieferumfang des Produkts WebSphere eXtreme Scale enthalten.

Abschnitt mit Lernprogrammen und Beispielen

Tabelle 15. Verfügbare Artikel nach Feature

Artikel	Features
Grid-fähige Anwendungen erstellen	API "ObjectMap", API "EntityManager", Abfrage, Agenten Java SE und Java EE, Statistiken, Partitionierung, Verwaltung/ Operationen, Eclipse
Skalierbare Datenverarbeitung mit Grids	EntityManager-API, Agenten
Skalierbare, flexible und leistungsfähige Datenbankalternative erstellen	ObjectMap-API, Replikation, Partitionierung, Verwaltung/Operationen, Eclipse
xsadmin für WebSphere eXtreme Scale erweitern	Verwaltung
Redbook: User's Guide	Alle Abschnitte

Lernprogramm zum EntityManager: Übersicht

Das Lernprogramm zum EntityManager zeigt Ihnen, wie Sie WebSphere eXtreme Scale verwenden, um Auftragsinformationen auf einer Website zu speichern. Sie können eine einfache Java-SE-5-Anwendung erstellen, die eine speicherinterne lokale eXtreme-Scale-Implementierung verwendet. Die Entitäten verwenden Annotationen und allgemeine Features von Java SE 5.

Vorbereitende Schritte

Stellen Sie sicher, dass die folgenden Voraussetzungen erfüllt sind, bevor Sie mit diesem Lernprogramm beginnen:

- Sie müssen Java SE 5 haben.
- Die Datei `objectgrid.jar` muss in Ihrem Klassenpfad enthalten sein.

Lernprogramm zum EntityManager: Entitätsklasse erstellen

Der erste Schritt des EntityManager-Lernprogramms zeigt Ihnen, wie Sie ein lokales ObjectGrid mit einer einzigen Entität erstellen, indem Sie eine Entity-Klasse erstellen, den Entitätstyp bei eXtreme Scale registrieren und eine Entitätsinstanz im Cache speichern.

Informationen zu diesem Vorgang

Vorgehensweise

1. Erstellen Sie das Order-Objekt. Zum Identifizieren des Objekts als ObjectGrid-Entität fügen Sie die Annotation "@Entity" hinzu. Wenn Sie diese Annotation hinzufügen, werden alle serialisierbaren Attribute im Objekt automatisch persistent in eXtreme Scale definiert, sofern Sie keine Annotationen für die Attributen verwenden, um die Attribute zu überschreiben. Das Attribut "orderNumber" wird mit "@Id" annotiert, um anzuzeigen, dass es sich bei diesem Attribut um den Primärschlüssel handelt. Es folgt ein Beispiel für ein Order-Objekt:

   Order.java

   ```
   @Entity
   public class Order
   {
       @Id String orderNumber;
       Date date;
       String customerName;
       String itemName;
       int quantity;
       double price;
   }
   ```

2. Führen Sie die eXtreme-Scale-Anwendung "Hello World" aus, um die Entitätsoperationen zu demonstrieren. Das folgende Beispielprogramm kann im eigenständigen Modus ausgeführt werden, um die Entitätsoperationen zu demonstrieren. Verwenden Sie dieses Programm in einem Eclipse-Java-Projekt, im dem die Datei `objectgrid.jar` dem Klassenpfad hinzugefügt wurde. Es folgt ein Beispiel für eine einfache Anwendung "Hello world", die eXtreme Scale verwendet:

   Application.java

   ```
   package emtutorial.basic.step1;

   import com.ibm.websphere.objectgrid.ObjectGrid;
   import com.ibm.websphere.objectgrid.ObjectGridManagerFactory;
   import com.ibm.websphere.objectgrid.Session;
   import com.ibm.websphere.objectgrid.em.EntityManager;

   public class Application
   {

       static public void main(String [] args)
           throws Exception
       {
           ObjectGrid og =
       ObjectGridManagerFactory.getObjectGridManager().createObjectGrid();
           og.registerEntities(new Class[] {Order.class});

           Session s = og.getSession();
           EntityManager em = s.getEntityManager();

           em.getTransaction().begin();
   ```

```
        Order o = new Order();
        o.customerName = "John Smith";
        o.date = new java.util.Date(System.currentTimeMillis());
        o.itemName = "Widget";
        o.orderNumber = "1";
        o.price = 99.99;
        o.quantity = 1;

        em.persist(o);
        em.getTransaction().commit();
        em.getTransaction().begin();
        o = (Order)em.find(Order.class, "1");
        System.out.println("Found order for customer: " + o.customerName);
        em.getTransaction().commit();    }
}
```

Diese Beispielanwendung führt die folgenden Operationen aus:

a. Sie initialisiert eine lokale eXtreme-Scale-Implementierung mit einem automatisch generierten Namen.
b. Sie registriert die Entitätsklassen über die API "registerEntities" bei der Anwendung, obwohl die Verwendung der API "registerEntities" nicht immer erforderlich ist.
c. Sie ruft ein Session-Objekt und eine Referenz auf den EntityManager für das Session-Objekt ab.
d. Sie ordnet jedes eXtreme-Scale-Session-Objekt einem einzigen EntityManager und einer EntityTransaction zu. Jetzt wird der EntityManager verwendet.
e. Die Methode "registerEntities" erstellt ein BackingMap-Objekt mit dem Namen "Order" und ordnet die Metadaten für das Order-Objekt dem BackingMap-Objekt zu. Zu diesen Metadaten gehören der Schlüssel und Attribute ohne Schlüsselfunktion sowie die Attributtypen und -namen.
f. Es wird eine Transaktion gestartet und eine Order-Instanz erstellt. Die Transaktion wird mit einigen Werten erfüllt und mit der Methode "EntityManager.persist" als persistent definiert, was die Entität als Entität identifiziert, die auf den Einschluss in die zugeordnete ObjectGrid-Map wartet.
g. Anschließend wird die Transaktion festgeschrieben, und die Entität wird in die ObjectMap eingeschlossen.
h. Es wird eine weitere Transaktion erstellt, und das Order-Objekt wird mit dem Schlüssel 1 abgerufen. Die Datentypänderung in der Methode "EntityManager.find" ist erforderlich, weil keine generischen Funktionen von Java SE 5 verwendet werden, um sicherzustellen, dass die Datei `objectgrid.jar` in einer Java Virtual Machine der Java SE Version 1.4 und höher funktioniert.

Lernprogramm zum EntityManager: Entitätsbeziehungen erstellen

Erstellen Sie eine einfache Beziehung zwischen Entitäten, indem Sie zwei Entitätsklassen mit einer Beziehung erstellen, die Entitäten beim ObjectGrid registrieren und die Entitätsinstanzen im Cache speichern.

Vorgehensweise

1. Erstellen Sie die Entität customer, die zum Speichern von Kundendetails, unabhängig vom Order-Objekt, verwendet wird. Es folgt ein Beispiel für die Entität customer:

```
Customer.java
@Entity
public class Customer
{
    @Id String id;
    String firstName;
```

```
    String surname;
    String address;
    String phoneNumber;
}
```

Diese Klasse enthält Informationen zum Kunden, wie z. B. den Namen, die Adresse und die Telefonnummer.

2. Erstellen Sie das Order-Objekt, das dem Order-Objekt im Abschnitt „Lernprogramm zum EntityManager: Entitätsklasse erstellen" auf Seite 190 gleicht. Es folgt ein Beispiel für das Order-Objekt:

Order.java

```
@Entity
public class Order
{
    @Id String orderNumber;
    Date date;
    @ManyToOne(cascade=CascadeType.PERSIST) Customer customer;
    String itemName;
    int quantity;
    double price;
}
```

In diesem Beispiel ersetzt eine Referenz auf ein Customer-Objekt das Attribut "customerName". Die Referenz hat eine Annotation, die eine Viele-zu-eins-Beziehung anzeigt. Eine Viele-zu-eins-Beziehung zeigt an, dass jeder Auftrag genau einen Kunden hat, aber mehrere Aufträge auf denselben Kunden verweisen können. Der Annotationsmodifikator "cascade" zeigt an, dass bei der persistenten Definition des Order-Objekts durch den EntityManager auch das Customer-Objekt als persistent definiert werden muss. Wenn Sie die Option "cascade persist" (die Standardoption) nicht setzen, müssen Sie das Customer-Objekt mit dem Order-Objekt manuell als persistent definieren.

3. Definieren Sie mit den Entitäten die Maps für die ObjectGrid-Instanz. Jede Map wird für eine bestimmte Entität definiert, und eine Entität erhält den Namen "Order" und die andere den Namen "Customer". Die folgende Beispielanwendung veranschaulicht, wie ein Kundenauftrag gespeichert und abgerufen wird:

Application.java

```
public class Application
{
    static public void main(String [] args)
        throws Exception
    {
        ObjectGrid og =
     ObjectGridManagerFactory.getObjectGridManager().createObjectGrid();
        og.registerEntities(new Class[] {Order.class});

        Session s = og.getSession();
        EntityManager em = s.getEntityManager();

        em.getTransaction().begin();

        Customer cust = new Customer();
        cust.address = "Main Street";
        cust.firstName = "John";
        cust.surname = "Smith";
        cust.id = "C001";
        cust.phoneNumber = "5555551212";

        Order o = new Order();
        o.customer = cust;
        o.date = new java.util.Date();
        o.itemName = "Widget";
        o.orderNumber = "1";
        o.price = 99.99;
        o.quantity = 1;

        em.persist(o);
```

```
        em.getTransaction().commit();
        em.getTransaction().begin();
        o = (Order)em.find(Order.class, "1");
        System.out.println("Found order for customer: "
    + o.customer.firstName + " " + o.customer.surname);
        em.getTransaction().commit();    }
}
```

Diese Anwendung gleicht der Beispielanwendung im vorherigen Schritt. Im vorherigen Beispiel wird nur eine einzige Klasse "Order" registriert. WebSphere eXtreme Scale erkennt und schließt die Referenz auf die Entität "Customer" automatisch ein. Es wird eine Customer-Instanz für `John Smith` erstellt und vom neuen Order-Objekt referenziert. Daraufhin wird der neue Kunde automatisch als persistent definiert, weil die Beziehung zwischen zwei Aufträgen den Modifikator "cascade" enthält, der erfordert, dass jedes Objekt als persistent definiert wird. Wenn das Order-Objekt gefunden wird, sucht der EntityManager automatisch das zugehörige Customer-Objekt und fügt eine Referenz auf das Objekt ein.

Lernprogramm zum EntityManager: Schema für die Entität "Order"

Erstellen Sie vier Entitätsklassen mit unidirektionalen und bidirektionalen Beziehungen, sortierten Listen und Fremdschlüsselbeziehungen. Die EntityManager-APIs werden verwendet, um die Entitäten zu suchen und persistent zu speichern. Aufbauend auf den Entitäten"Order" und "Customer", die in den verschiedenen Teilen des Lernprogramms verwendet werden, werden in diesem Schritt des Lernprogramms zwei weitere Entitäten hinzugefügt: Item und OrderLine.

Informationen zu diesem Vorgang

Abbildung 43. Schema für die Entität "Order". Eine Entität "Order" (Auftrag) hat eine Referenz auf einen Kunden (Customer) und keiner oder mehreren Auftragspositionen (OrderLine). Jede Entität "OrderLine" hat eine Referenz auf einen einzelnen Artikel (Item) und enthält die bestellte Menge.

Vorgehensweise

1. Erstellen Sie die Entität "Customer", ähnlich wie in den vorherigen Beispielen.

   Customer.java
   ```
   @Entity
   public class Customer
   {
       @Id String id;
       String firstName;
       String surname;
       String address;
       String phoneNumber;
   }
   ```

2. Erstellen Sie die Entität "Item", die Informationen zu einem Produkt enthält, das im Lagerbestand enthalten ist, z. B. Produktbeschreibung, Menge oder Preis.

   Item.java
   ```
   @Entity
   public class Item
   {
       @Id String id;
       String description;
       long quantityOnHand;
       double price;
   }
   ```

3. Erstellen Sie die Entität "OrderLine". Jeder Auftrag hat keine oder mehrere Auftragspositionen, die die Menge jedes Artikels im Auftrag angeben. Der Schlüssel für die Auftragspositionen ist ein Verbundschlüssel, der sich aus dem Auftrag zusammensetzt, der Eigner der Auftragsposition ist, und eine ganze Zahl, die der Auftragsposition eine Nummer zuweist. Fügen Sie den Modifikator "cascade persist" jeder Beziehung in Ihren Entitäten hinzu.

   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;
   }
   ```

4. Erstellen Sie das endgültige Auftragsobjekt (Order), das eine Referenz auf den Kunden (Customer) für den Auftrag und eine Sammlung von Auftragspositionsobjekten (OrderLine) enthält.

   Order.java
   ```
   @Entity
   public class Order
   {
       @Id String orderNumber;
       java.util.Date date;
       @ManyToOne(cascade=CascadeType.PERSIST) Customer customer;
       @OneToMany(cascade=CascadeType.ALL, mappedBy="order")
       @OrderBy("lineNumber") List<OrderLine> lines;  }
   ```

   Der Modifikator "cascade ALL" wird als Modifikator für Positionen verwendet. Dieser Modifikator signalisiert dem EntityManager, sowohl die Operation to PERSIST als auch die Operation REMOVE zu kaskadieren. Wenn beispielsweise die Entität "Order" persistent gespeichert oder entfernt wird, werden auch alle OrderLine-Entitäten persistent gespeichert bzw. entfernt.

   Wenn eine OrderLine-Entität aus der Positionsliste im Order-Objekt entfernt wird, ist die Referenz ungültig. Die OrderLine-Entität wird jedoch nicht aus dem Cache entfernt. Sie müssen die EntityManager-API "remove" verwenden, um Entitäten aus dem Cache zu entfernen. Die Operation REMOVE wird nicht für die Customer-Entität und die Item-Entität aus dem OrderLine-Objekt verwendet. Deshalb bleibt die Customer-Entität erhalten, obwohl der Auftrag bzw. der Artikel entfernt wird, wenn die Auftragsposition entfernt wird.

   Der Modifikator "mappedBy" gibt eine Umkehrbeziehung mit der Zielentität an. Der Modifikator gibt an, welches Attribut in der Zielentität auf die Quellenentität und die Eignerseite einer 1:1- oder N:N-Beziehung verweist. Gewöhnlich können Sie den Modifikator weglassen. Es wird jedoch ein Fehler angezeigt, in dem Sie darauf hingewiesen werden, dass der Modifikator angegeben werden muss, wenn WebSphere eXtreme Scale den Modifikator nicht automatisch erkennen kann. Eine OrderLine-Entität, die zwei Attribute vom Typ "Order" in einer N:1-Beziehung enthält, ist gewöhnlich für diesen Fehler verantwortlich.

   Die Annotation "@OrderBy" gibt die Reihenfolge an, in der die OrderLine-Entitäten in der Positionsliste aufgeführt werden sollen. Wenn die Annotation nicht angegeben wird, werden die Positionen in beliebiger Reihenfolge angezeigt. Obwohl die Positionen der Order-Entität über das Absetzen von ArrayList hinzugefügt werden, bei dem die Reihenfolge eingehalten wird, erkennt der EntityManager die Reihenfolge nicht zwingenderweise. Wenn Sie die Methode "find" ausführen, um das Order-Objekt aus dem Cache abzurufen, ist das Listenobjekt kein ArrayList-Objekt.

5. Erstellen Sie die Anwendung. Im folgenden Beispiel wird das endgültige Auftragsobjekt (Order) veranschaulicht, das eine Referenz auf den Kunden (Customer) für den Auftrag und eine Sammlung von Auftragspositionsobjekten (OrderLine) enthält.
   a. Suchen Sie die zu sortierenden Artikel, die dann zu verwalteten Entitäten werden.
   b. Erstellen Sie die Auftragsposition, und ordnen Sie sie jedem Artikel zu.
   c. Erstellen Sie den Auftrag, und ordnen Sie ihn jeder Auftragsposition und dem Kunde zu.
   d. Speichern Sie den Auftrag persistent, woraufhin automatisch alle Auftragspositionen persistent gespeichert werden.
   e. Schreiben Sie die Transaktion fest, woraufhin alle Entitäten freigegeben werden und der Status der Entitäten mit dem Cache synchronisiert wird.
   f. Geben Sie die Auftragsdaten aus. Die OrderLine-Entitäten werden automatisch nach OrderLine-ID sortiert.

```
Application.java

static public void main(String [] args)
        throws Exception
    {
        ...

        // Dem Bestand einige Artikel hinzufügen.
        em.getTransaction().begin();
        createItems(em);
        em.getTransaction().commit();
        // Neuen kunden mit den Artikeln im Einkaufskorb erstellen.
        em.getTransaction().begin();
        Customer cust = createCustomer();
        em.persist(cust);

        // Neuen Auftrag erstellen und für jeden Artikel eine Auftragsposition hinzufügen.
        // Jede Position wird automatisch persistent gespeichert, da die Option
        // Cascade=ALL definiert ist.
        Order order = createOrderFromItems(em, cust, "ORDER_1",
     new String[]{"1", "2"}, new int[]{1,3});
        em.persist(order);
        em.getTransaction().commit();
        // Auftragszusammenfassung ausgeben
        em.getTransaction().begin();
        order = (Order)em.find(Order.class, "ORDER_1");
        System.out.println(printOrderSummary(order));
        em.getTransaction().commit();    }

    public static Customer createCustomer() {
        Customer cust = new Customer();
        cust.address = "Main Street";
        cust.firstName = "John";
        cust.surname = "Smith";
        cust.id = "C001";
        cust.phoneNumber = "5555551212";
        return cust;
    }

    public static void createItems(EntityManager em) {
        Item item1 = new Item();
        item1.id = "1";
        item1.price = 9.99;
        item1.description = "Widget 1";
        item1.quantityOnHand = 4000;
        em.persist(item1);
```

```
        Item item2 = new Item();
        item2.id = "2";
        item2.price = 15.99;
        item2.description = "Widget 2";
        item2.quantityOnHand = 225;
        em.persist(item2);

    }

    public static Order createOrderFromItems(EntityManager em,
   Customer cust, String orderId, String[] itemIds, int[] qty) {

        Item[] items = getItems(em, itemIds);

        Order order = new Order();
        order.customer = cust;
        order.date = new java.util.Date();
        order.orderNumber = orderId;
        order.lines = new ArrayList<OrderLine>(items.length);
    for(int i=0;i<items.length;i++){
      OrderLine line = new OrderLine();
            line.lineNumber = i+1;
            line.item = items[i];
            line.price = line.item.price;
            line.quantity = qty[i];
            line.order = order;
            order.lines.add(line);
        }
        return order;
    }

    public static Item[] getItems(EntityManager em, String[] itemIds) {
        Item[] items = new Item[itemIds.length];
        for(int i=0;i<items.length;i++){
    items[i] = (Item) em.find(Item.class, itemIds[i]);
        }
        return items;
    }
```

Der nächste Schritt ist das Löschen einer Entität. Die Schnittstelle "EntityManager" enthält eine Methode "remove", die ein Objekt als gelöscht markiert. Die Anwendung muss die Entität aus allen Beziehungssammlungen entfernen, bevor sie die Methode "remove" aufruft. Als letzten Schritt bearbeiten Sie die Referenzen und führen die Methode "remove" oder "em.remove(object)" aus.

Lernprogramm zum EntityManager: Einträge aktualisieren

Wenn Sie eine Entität ändern möchten, können Sie die Instanz suchen, die Instanz und alle referenzierten Entitäten aktualisieren und anschließend die Transaktion festschreiben.

Vorgehensweise

Aktualisieren Sie Einträge. Das folgende Beispiel veranschaulicht, wie Sie die Order-Instanz suchen, die Instanz und alle referenzierten Entitäten ändern und die Transaktion festschreiben.

```
public static void updateCustomerOrder(EntityManager em) {
    em.getTransaction().begin();
    Order order = (Order) em.find(Order.class, "ORDER_1");
    processDiscount(order, 10);
    Customer cust = order.customer;
    cust.phoneNumber = "5075551234";
    em.getTransaction().commit();}

public static void processDiscount(Order order, double discountPct) {
```

```
    for(OrderLine line : order.lines) {
        line.price = line.price * ((100-discountPct)/100);
    }
}
```

Bei der Ausführung der Flush-Operation für die Transaktion werden alle verwalteten Entitäten mit dem Cache synchronisiert. Beim Festschreiben einer Transaktion wird automatische eine Flush-Operation ausgeführt. In diesem Fall wird die Order-Instanz zu einer verwalteten Entität. Alle von Order, Customer und OrderLine referenzierten Entitäten werden ebenfalls zu verwalteten Entitäten. Beim Ausführen der Flush-Operation für die Transaktion werden alle Entitäten dahingehend geprüft, ob sie geändert wurden. Die geänderten Entitäten werden im Cache aktualisiert. Nach Abschluss der Transaktion durch Festschreibung oder Rollback werden die Entitäten freigegeben, und alle Änderungen, die an den Entitäten vorgenommen wurden, werden nicht in den Cache übernommen.

Lernprogramm zum EntityManager: Einträge über einen Index aktualisieren und entfernen

Sie können einen Index verwenden, um Entitäten zu suchen, zu aktualisieren und zu entfernen.

Vorgehensweise

Aktualisieren und entfernen Sie Entitäten über einen Index. Verwenden Sie einen Index, um Entitäten zu suchen, zu aktualisieren und zu entfernen. In den folgenden Beispielen wird die Entitätsklasse "Order" für die Verwendung der Annotation "@Index" aktualisiert. Die Annotation "@Index" signalisiert WebSphere eXtreme Scale, einen Bereichsindex für ein Attribut zu erstellen. Der Name des Index entspricht dem Namen des Attributs und hat immer den Indextyp "MapRangeIndex".

Order.java

```
@Entity
public class Order
{
    @Id String orderNumber;
    @Index java.util.Date date;
    @OneToOne(cascade=CascadeType.PERSIST) Customer customer;
    @OneToMany(cascade=CascadeType.ALL, mappedBy="order")
    @OrderBy("lineNumber") List<OrderLine> lines;  }
```

Das folgende Beispiel veranschaulicht, wie alle in der letzten Minute übergebenen Aufträge (Order) storniert werden. Suchen Sie den Auftrag über einen Index, fügen Sie die Artikel aus dem Auftrag dem Bestand wieder hinzu, und entfernen Sie den Auftrag und die zugeordneten Positionen aus dem System.

```
public static void cancelOrdersUsingIndex(Session s)
  throws ObjectGridException {
    // Alle Aufträge stornieren, die in der letzten Minute übergeben wurden
    java.util.Date cancelTime = new
   java.util.Date(System.currentTimeMillis() - 60000);
    EntityManager em = s.getEntityManager();
    em.getTransaction().begin();
    MapRangeIndex dateIndex = (MapRangeIndex)
   s.getMap("Order").getIndex("date");
    Iterator<Tuple> orderKeys = dateIndex.findGreaterEqual(cancelTime);
  while(orderKeys.hasNext()) {
   Tuple orderKey = orderKeys.next();
   // Auftrag suchen, damit er entfernt werden kann
   Order curOrder = (Order) em.find(Order.class, orderKey);
   // Sicherstellen, dass der Auftrag nicht von einer anderen Person geändert wurde
   if(curOrder != null && curOrder.date.getTime() >= cancelTime.getTime()) {
    for(OrderLine line : curOrder.lines) {
     // Artikel wieder dem Bestand hinzufügen.
     line.item.quantityOnHand += line.quantity;
     line.quantity = 0;
```

```
    }
   em.remove(curOrder);
   }
  }
em.getTransaction().commit();}
```

Lernprogramm zum EntityManager: Einträge über eine Abfrage aktualisieren und entfernen

Sie können Entitäten über eine Abfrage aktualisieren und entfernen.

Vorgehensweise

Aktualisieren und entfernen Sie Entitäten über eine Abfrage.

Order.java

```
@Entity
public class Order
{
    @Id String orderNumber;
    @Index java.util.Date date;
    @OneToOne(cascade=CascadeType.PERSIST) Customer customer;
    @OneToMany(cascade=CascadeType.ALL, mappedBy="order")
    @OrderBy("lineNumber") List<OrderLine> lines;  }
```

Die Entitätsklasse "Order" ist dieselbe wie im vorherigen Beispiel. Die Klasse stellt die Annotation "@Index" weiterhin bereit, weil die Abfragezeichenfolge das Datum verwendet, um die Entität zu finden. Die Abfragesteuerkomponente verwendet Indizes, wenn sie verwendet werden können.

```
public static void cancelOrdersUsingQuery(Session s) {
        // Alle Aufträge stornieren, die in der letzten Minute übergeben wurden
        java.util.Date cancelTime =
     new java.util.Date(System.currentTimeMillis() - 60000);
        EntityManager em = s.getEntityManager();
        em.getTransaction().begin();

        // Abfrage erstellen, die den Auftrag nach Datum sucht. Da
        // ein Index für das Auftragsdatum definiert ist, wird er
        // von der Abfrage automatisch verwendet.
Query query = em.createQuery("SELECT order FROM Order order
     WHERE order.date >= ?1");
        query.setParameter(1, cancelTime);
        Iterator<Order> orderIterator = query.getResultIterator();
    while(orderIterator.hasNext()) {
     Order order = orderIterator.next();
     // Sicherstellen, dass der Auftrag nicht von einer anderen Person geändert wurde
     // Da die Abfrage einen Index verwendet, ist keine Sperre für die Zeile gesetzt.
     if(order != null && order.date.getTime() >= cancelTime.getTime()) {
       for(OrderLine line : order.lines) {
        // Artikel wieder dem Bestand hinzufügen.
        line.item.quantityOnHand += line.quantity;
        line.quantity = 0;
       }
       em.remove(order);
     }
   }
em.getTransaction().commit();}
```

Wie im vorherigen Beispiel beabsichtigt die Methode "cancelOrdersUsingQuery", alle Aufträge zu stornieren, die in der letzten Minute übergeben wurden. Zum Stornieren des Auftrags, suchen Sie den Auftrag über eine Abfrage, fügen Sie die Artikel aus dem Auftrag dem Bestand wieder hinzu, und entfernen Sie den Auftrag und die zugeordneten Positionen aus dem System.

Lernprogramm zu ObjectQuery

Mit den folgenden Schritten können Sie ein lokales, speicherinternes ObjectGrid entwickeln, in dem Auftragsinformationen für eine Website gespeichert werden und wie ObjectQuery für die Abfragen der Daten im Grid verwendet wird.

Vorbereitende Schritte

Stellen Sie sicher, dass die Datei `objectgrid.jar` im Klassenpfad enthalten ist.

Informationen zu diesem Vorgang

Jeder Schritt im Lernprogramm baut auf dem vorherigen Schritt auf. Führen Sie jeden Schritt aus, um ein einfache Anwendung der Java Platform, Standard Edition Version 1.4 (oder höher) zu erstellen, die ein lokales, speicherinternes ObjectGrid verwendet.

Vorgehensweise

1. „Lernprogramm zu ObjectQuery - Schritt 1”
   - Vorgehensweise zum Erstellen eines lokalen ObjectGrids
   - Vorgehensweise zum Definieren eines Schemas für ein einzelnes Objekt über Feldzugriff
   - Vorgehensweise zum Speichern des Objekts
   - Vorgehensweise zum Abfragen des Objekts mit ObjectQuery
2. „Lernprogramm zu ObjectQuery - Schritt 2” auf Seite 201
   - Vorgehensweise zum Erstellen eines Index, den die Abfrage verwenden kann
3. „Lernprogramm zu ObjectQuery - Schritt 3” auf Seite 202
   - Vorgehensweise zum Erstellen eines Schemas mit zwei zusammengehörigen Entitäten
   - Vorgehensweise zum Speichern von Objekten mit einer Fremdschlüsselreferenz zwischen den Objekten
   - Vorgehensweise zum Abfragen der Objekte über eine einzelne Abfrage mit JOIN
4. „Lernprogramm zu ObjectQuery - Schritt 4” auf Seite 204
   - Vorgehensweise zum Erstellen eines Schemas mit mehreren zusammengehörigen Entitäten
   - Vorgehensweise zum Verwenden des Methoden- bzw. Eigenschaftszugriffs anstelle des Feldzugriffs

Lernprogramm zu ObjectQuery - Schritt 1

Mit den folgenden Schritten können Sie die Entwicklung eines lokalen, speicherinternen ObjectGrids fortsetzen, in dem Auftragsinformationen für ein Onlineeinzelhandelsunternehmen über die ObjectMap-APIs gespeichert werden. Sie definieren ein Schema für eine Map und führen eine Abfrage der Map aus.

Vorgehensweise

1. Erstellen Sie ein ObjectGrid mit einem Map-Schema.

   Erstellen Sie ein ObjectGrid mit einem einzigen Map-Schema für die Map, fügen Sie ein Objekt in den Cache ein, und rufen Sie das Objekt später über eine einfache Abfrage ab.

OrderBean.java

```
public class OrderBean implements Serializable {
    String orderNumber;
    java.util.Date date;
    String customerName;
    String itemName;
    int quantity;
    double price;
}
```

2. Definieren Sie den Primärschlüssel.

   Der vorherige Code zeigt ein OrderBean-Objekt. Dieses Objekt implementiert die Schnittstelle "java.io.Serializable", weil alle Objekt im Cache (standardmäßig) den Typ "Serializable" haben müssen.

   Das Attribut "orderNumber" ist der Primärschlüssel des Objekts. Das folgende Beispielprogramm kann im eigenständigen Modus ausgeführt werden. Sie müssen dieses Lernprogramm in einem Eclipse-Java-Projekt ausführen, in dem die Datei `objectgrid.jar` dem Klassenpfad hinzugefügt wurde.

   Application.java

```
package querytutorial.basic.step1;

import java.util.Iterator;

import com.ibm.websphere.objectgrid.ObjectGrid;
import com.ibm.websphere.objectgrid.ObjectGridManagerFactory;
import com.ibm.websphere.objectgrid.ObjectMap;
import com.ibm.websphere.objectgrid.Session;
import com.ibm.websphere.objectgrid.config.QueryConfig;
import com.ibm.websphere.objectgrid.config.QueryMapping;
import com.ibm.websphere.objectgrid.query.ObjectQuery;

public class Application
{

    static public void main(String [] args) throws Exception
    {
        ObjectGrid og = ObjectGridManagerFactory.getObjectGridManager().createObjectGrid();
        og.defineMap("Order");

        // Schema definieren
        QueryConfig queryCfg = new QueryConfig();
        queryCfg.addQueryMapping(new QueryMapping("Order", OrderBean.class.getName(),
     "orderNumber", QueryMapping.FIELD_ACCESS));
        og.setQueryConfig(queryCfg);

        Session s = og.getSession();
        ObjectMap orderMap = s.getMap("Order");

        s.begin();
        OrderBean o = new OrderBean();
        o.customerName = "John Smith";
        o.date = new java.util.Date(System.currentTimeMillis());
        o.itemName = "Widget";
        o.orderNumber = "1";
        o.price = 99.99;
        o.quantity = 1;
        orderMap.put(o.orderNumber, o);
        s.commit();
        s.begin();
        ObjectQuery query = s.createObjectQuery("SELECT o FROM Order o WHERE o.itemName='Widget'");
        Iterator result = query.getResultIterator();
        o = (OrderBean) result.next();
        System.out.println("Found order for customer: " + o.customerName);
        s.commit();    }
}
```

   Diese eXtreme-Scale-Anwendung initialisiert zuerst ein lokales ObjectGrid mit einem automatisch generierten Namen. Anschließend erstellt die Anwendung ein BackingMap- und ein QueryConfig-Objekt, das Folgendes definiert: Java-Typ, der der Map zugeordnet wird, Namen des Felds, das der Primärschlüssel für die Map ist, und Zugriff auf die Daten im Objekt. Anschließend wird ein Session-Objekt angefordert, um die ObjectMap-Instanz abzurufen und ein OrderBean-Objekt in einer Transaktion in die Map einzufügen.

Nach dem Festschreiben der Daten im Cache können Sie das ObjectQuery-Objekt verwenden, um das OrderBean-Objekt über eines der persistenten Felder in Klasse zu suchen. Persistente Felder sind Felder, die den Modifikator "transient" nicht haben. Da Sie keine Indizes für die BackingMap definiert haben, muss das ObjectQuery-Objekt jedes Objekt in der Map durch Java-Reflexion scannen.

Nächste Schritte

Im Abschnitt „Lernprogramm zu ObjectQuery - Schritt 2" wird demonstriert, wie die Abfrage mit einem Index optimiert werden kann.

Lernprogramm zu ObjectQuery - Schritt 2

Mit den folgenden Schritten erstellen Sie ein ObjectGrid mit einer einzigen Map und einem Index sowie ein Schema für die Map. Anschließend fügen Sie ein Objekt in den Cache ein und rufen dieses später über eine einfache Abfrage ab.

Vorbereitende Schritte

Stellen Sie sicher, dass Sie die Anweisungen im Abschnitt „Lernprogramm zu ObjectQuery - Schritt 1" auf Seite 199 ausgeführt haben, bevor Sie mit diesem Schritt des Lernprogramms fortfahren.

Vorgehensweise

Schema und Index

Application.java

```
// Index erstellen
    HashIndex idx= new HashIndex();
    idx.setName("theItemName");
    idx.setAttributeName("itemName");
    idx.setRangeIndex(true);
    idx.setFieldAccessAttribute(true);
    orderBMap.addMapIndexPlugin(idx);
}
```

Der Index muss eine Instanz von "com.ibm.websphere.objectgrid.plugins.index-.HashIndex" mit den folgenden Einstellungen sein:

- Der Wert für das Attribut "name" kann frei gewählt werden, muss aber für eine bestimmte BackingMap eindeutig sein.
- Das Attribut "AttributeName" gibt den Namen des Felds bzw. der Bean-Eigenschaft an, das bzw. die von der Indexierungssteuerkomponente verwendet wird, um die Klasse selbst zu überwachen. In diesem Fall ist es der Name des Felds, für das Sie einen Index erstellen.
- Das Attribut "RangeIndex" muss immer "true" sein.
- Der Wert des Attributs "FieldAccessAttribute" muss mit dem Wert übereinstimmen, der im QueryMapping-Objekt bei der Erstellung des Abfrageschemas festgelegt wurde. In diesem Fall erfolgt der Zugriff auf das Java-Objekt direkt über die Felder.

Wenn eine Abfrage ausgeführt wird, die die Ergebnisse nach dem Feld "itemName" filtert, verwendet die Abfragesteuerkomponente automatisch den definierten Index. Durch die Verwendung des Index kann die Abfrage schneller ausgeführt werden,

und das Durchsuchen der Map ist nicht erforderlich. Im nächsten Schritt wird demonstriert, wie eine Abfrage mit einem Index optimiert werden kann.
Nächster Schritt

Lernprogramm zu ObjectQuery - Schritt 3

Mit dem folgenden Schritt erstellen Sie ein ObjectGrid mit zwei Maps und ein Schema für die Maps mit einer Beziehung, fügen Objekte in den Cache ein und rufen diese später über eine einfache Abfrage ab.

Vorbereitende Schritte

Stellen Sie sicher, dass Sie die Anweisungen im Abschnitt „Lernprogramm zu ObjectQuery - Schritt 2" auf Seite 201 ausgeführt haben, bevor Sie mit diesem Schritt fortfahren.

Informationen zu diesem Vorgang

In diesem Beispiel werden zwei Maps verwendet, denen jeweils ein einzelner Java-Typ zugeordnet ist. Die Map "Order" enthält OrderBean-Objekte, und die Map "Customer" enthält CustomerBean-Objekte.

Vorgehensweise

Definieren Sie Maps mit einer Beziehung.

OrderBean.java

```
public class OrderBean implements Serializable {
    String orderNumber;
    java.util.Date date;
    String customerId;
    String itemName;
    int quantity;
    double price;
}
```

Die OrderBean enthält keinen customerName mehr. Stattdessen enthält sie die customerId, die der Primärschlüssel für das CustomerBean-Objekt und die Map "Customer" ist.

CustomerBean.java

```
public class CustomerBean implements Serializable{
    private static final long serialVersionUID = 1L;
     String id;
    String firstName;
    String surname;
    String address;
    String phoneNumber;
}
```

Im Folgenden sehen Sie die Beziehung zwischen den beiden Typen bzw. Maps:

Application.java

```
public class Application
{

    static public void main(String [] args)
        throws Exception
    {
        ObjectGrid og = ObjectGridManagerFactory.getObjectGridManager().createObjectGrid();
        og.defineMap("Order");
```

```
        og.defineMap("Customer");

        // Schema definieren
        QueryConfig queryCfg = new QueryConfig();
        queryCfg.addQueryMapping(new QueryMapping(
            "Order", OrderBean.class.getName(), "orderNumber", QueryMapping.FIELD_ACCESS));
        queryCfg.addQueryMapping(new QueryMapping(
            "Customer", CustomerBean.class.getName(), "id", QueryMapping.FIELD_ACCESS));
        queryCfg.addQueryRelationship(new QueryRelationship(
             OrderBean.class.getName(), CustomerBean.class.getName(), "customerId", null));
        og.setQueryConfig(queryCfg);

        Session s = og.getSession();
        ObjectMap orderMap = s.getMap("Order");
        ObjectMap custMap = s.getMap("Customer");

        s.begin();
        CustomerBean cust = new CustomerBean();
        cust.address = "Main Street";
        cust.firstName = "John";
        cust.surname = "Smith";
        cust.id = "C001";
        cust.phoneNumber = "5555551212";
        custMap.insert(cust.id, cust);

        OrderBean o = new OrderBean();
        o.customerId = cust.id;
        o.date = new java.util.Date();
        o.itemName = "Widget";
        o.orderNumber = "1";
        o.price = 99.99;
        o.quantity = 1;
        orderMap.insert(o.orderNumber, o);
        s.commit();
        s.begin();
        ObjectQuery query = s.createObjectQuery(
            "SELECT c FROM Order o JOIN o.customerId as c WHERE o.itemName='Widget'");
        Iterator result = query.getResultIterator();
        cust = (CustomerBean) result.next();
        System.out.println("Found order for customer: " + cust.firstName + " " + cust.surname);
        s.commit();    }
}
```

Im Folgenden sehen Sie die funktional entsprechende XML im ObjectGrid-Implementierungsdeskriptor:

```
<?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">
      <backingMap name="Order"/>
      <backingMap name="Customer"/>


      <querySchema>
         <mapSchemas>
           <mapSchema
             mapName="Order"
             valueClass="com.mycompany.OrderBean"
             primaryKeyField="orderNumber"
             accessType="FIELD"/>
           <mapSchema
             mapName="Customer"
             valueClass="com.mycompany.CustomerBean"
             primaryKeyField="id"
             accessType="FIELD"/>
         </mapSchemas>
         <relationships>
           <relationship
             source="com.mycompany.OrderBean"
             target="com.mycompany.CustomerBean"
             relationField="customerId"/>
         </relationships>
```

```
      </querySchema>
    </objectGrid>
  </objectGrids>
</objectGridConfig>
```

Nächste Schritte

Im Abschnitt „Lernprogramm zu ObjectQuery - Schritt 4" wird der aktuelle Schritt erweitert, indem Objekt mit Feld- und Eigenschaftszugriff und weitere Beziehungen hinzugefügt werden.

Lernprogramm zu ObjectQuery - Schritt 4

Im folgenden Schritt wird demonstriert, wie Sie ein ObjectGrid mit vier Maps und ein Schema für diese Maps mit mehreren unidirektionalen und bidirektionalen Beziehungen erstellen. Anschließend fügen Sie Objekte in den Cache ein und rufen sie später über mehrere Abfragen ab.

Vorbereitende Schritte

Stellen Sie sicher, dass Sie die Anweisungen im Abschnitt „Lernprogramm zu ObjectQuery - Schritt 3" auf Seite 202 ausgeführt haben, bevor Sie mit diesem Schritt fortfahren.

Vorgehensweise

Mehrere Map-Beziehungen

OrderBean.java

```
public class OrderBean implements Serializable {
    String orderNumber;
    java.util.Date date;
    String customerId;
    String itemName;
    int quantity;
    double price;
}
```

Wie im vorherigen Schritt enthält OrderBean keinen customerName mehr. Stattdessen enthält sie die customerId, die der Primärschlüssel für das CustomerBean-Objekt und die Map "Customer" ist.

CustomerBean.java

```
public class CustomerBean implements Serializable{
    private static final long serialVersionUID = 1L;
    String id;
    String firstName;
    String surname;
    String address;
    String phoneNumber;
}
```

Nachdem die angegebenen Klassen erstellt wurden, können Sie die folgende Anwendung ausführen:

Application.java

```
public class Application
{

    static public void main(String [] args)
```

```
        throws Exception
    {
        ObjectGrid og = ObjectGridManagerFactory.getObjectGridManager().createObjectGrid();
        og.defineMap("Order");
        og.defineMap("Customer");

        // Schema definieren
        QueryConfig queryCfg = new QueryConfig();
        queryCfg.addQueryMapping(new QueryMapping(
            "Order", OrderBean.class.getName(), "orderNumber", QueryMapping.FIELD_ACCESS));
        queryCfg.addQueryMapping(new QueryMapping(
            "Customer", CustomerBean.class.getName(), "id", QueryMapping.FIELD_ACCESS));
        queryCfg.addQueryRelationship(new QueryRelationship(
             OrderBean.class.getName(), CustomerBean.class.getName(), "customerId", null));
        og.setQueryConfig(queryCfg);

        Session s = og.getSession();
        ObjectMap orderMap = s.getMap("Order");
        ObjectMap custMap = s.getMap("Customer");

        s.begin();
        CustomerBean cust = new CustomerBean();
        cust.address = "Main Street";
        cust.firstName = "John";
        cust.surname = "Smith";
        cust.id = "C001";
        cust.phoneNumber = "5555551212";
        custMap.insert(cust.id, cust);

        OrderBean o = new OrderBean();
        o.customerId = cust.id;
        o.date = new java.util.Date();
        o.itemName = "Widget";
        o.orderNumber = "1";
        o.price = 99.99;
        o.quantity = 1;
        orderMap.insert(o.orderNumber, o);
        s.commit();
        s.begin();
        ObjectQuery query = s.createObjectQuery(
            "SELECT c FROM Order o JOIN o.customerId as c WHERE o.itemName='Widget'");
        Iterator result = query.getResultIterator();
        cust = (CustomerBean) result.next();
        System.out.println("Found order for customer: " + cust.firstName + " " + cust.surname);
        s.commit();    }
}
```

Die folgende XML-Konfiguration (im ObjectGrid-Implementierungsdeskriptor) entspricht funktional dem zuvor beschriebenen programmgesteuerten Ansatz.

```
<?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="og1">
      <backingMap name="Order"/>
      <backingMap name="Customer"/>


      <querySchema>
         <mapSchemas>
           <mapSchema
             mapName="Order"
             valueClass="com.mycompany.OrderBean"
             primaryKeyField="orderNumber"
             accessType="FIELD"/>
           <mapSchema
             mapName="Customer"
             valueClass="com.mycompany.CustomerBean"
             primaryKeyField="id"
             accessType="FIELD"/>
         </mapSchemas>
         <relationships>
           <relationship
             source="com.mycompany.OrderBean"
             target="com.mycompany.CustomerBean"
             relationField="customerId"/>
         </relationships>
```

```
      </querySchema>
    </objectGrid>
  </objectGrids>
</objectGridConfig>
```

Lernprogramm zur Java-SE-Sicherheit: Übersicht

Mit dem folgenden Lernprogramm können Sie eine verteilte eXtreme-Scale-Umgebung in einer Java-SE-Umgebung erstellen.

Vorbereitende Schritte

Stellen Sie sicher, dass Sie mit den Grundlagen einer verteilten eXtreme-Scale-Konfiguration vertraut sind.

Informationen zu diesem Vorgang

In diesem Lernprogramm werden der Katalogserver, der Containerserver und der Client alle in einer Java-SE-Umgebung ausgeführt. Jeder Schritt im Lernprogramm baut auf dem vorherigen Schritt auf. Führen Sie jeden der Schritte aus, um eine verteilte eXtreme-Scale-Konfiguration zu sichern und eine einfache Java-SE-Anwendung für den Zugriff auf eine gesicherte eXtreme-Scale-Konfiguration zu entwickeln.

Lernprogramm starten

Vorgehensweise

1. „Lernprogramm zur Java-SE-Sicherheit - Schritt 1" auf Seite 207
   - Nicht gesicherten Katalogserver starten
   - Nicht gesicherten Containerserver starten
   - Client für den Zugriff auf die Daten starten
   - xsadmin zum Anzeigen der Map-Größe verwenden
   - Server stoppen
2. „Lernprogramm zur Java-SE-Sicherheit - Schritt 2" auf Seite 210
   - CredentialGenerator verwenden
   - Authentifikator verwenden
   - Sicheren Katalogserver starten
   - Sicheren Containerserver starten
   - Client für den Zugriff auf ein gesichertes ObjectGrid starten
   - xsadmin zum Anzeigen der Map-Größe verwenden
   - Sicheren Server stoppen
3. „Lernprogramm zur Java-SE-Sicherheit - Schritt 3" auf Seite 216
   - JAAS-Berechtigungsrichtlinie verwenden
4. „Lernprogramm zur Java-SE-Sicherheit - Schritt 4" auf Seite 220
   - Keystore und Truststore erstellen
   - SSL-Eigenschaften für den Server konfigurieren
   - SSL-Eigenschaften für den Client konfigurieren
   - xsadmin zum Anzeigen der Map-Größe verwenden
   - Sicheren Server stoppen

Lernprogramm zur Java-SE-Sicherheit - Schritt 1

In diesem Abschnitt wird ein *einfaches nicht gesichertes Muster* beschrieben. In den nachfolgenden Schritten des Lernprogramms werden nach und nach weitere Sicherheitsfeatures hinzugefügt, um die verfügbare integrierte Sicherheit zu erhöhen.

Vorbereitende Schritte

Anmerkung: Alle für diesen Schritt des Lernprogramms erforderlichen Dateien sind im folgenden Abschnitt beschrieben.

Vorgehensweise

Muster ausführen
Starten Sie den Katalogservice mit den folgenden Scripts. Weitere Informationen zum Starten des Katalogservice finden Sie in den Informationen zum Starten des Katalogservice im *Administratorhandbuch*.

1. Navigieren Sie wie folgt zum Verzeichnis "bin": `cd ObjectGrid-Stammverzeichnis/bin`
2. Starten Sie einen Katalogserver mit dem Namen "catalogServer":
   - UNIX Linux `startOgServer.sh catalogServer`
   - Windows `startOgServer.bat catalogServer`
3. Navigieren Sie wie folgt zum Verzeichnis "bin": `cd ObjectGrid-Stammverzeichnis/bin`
4. Starten Sie anschließend mit dem folgenden Script einen Containerserver mit dem Namen "c0":
   - UNIX Linux
     ```
     startOgServer.sh c0 -objectGridFile ../xml/SimpleApp.xml -deploymentPolicyFile ../xml/SimpleDP.xml -catalogServiceEndpoints localhost:2809
     ```
   - Windows
     ```
     startOgServer.bat c0 -objectGridFile ../xml/SimpleApp.xml -deploymentPolicyFile ../xml/SimpleDP.xml -catalogServiceEndpoints localhost:2809
     ```

Beispiel

Weitere Informationen zum Starten von Containerservern finden Sie in den Informationen zum Starten der Containerprozesse im *Administratorhandbuch*.

Nach dem Starten des Katalogservers und des Containerservers starten Sie den Client wie folgt:

1. Navigieren Sie wieder zum Verzeichnis "bin".
2.
   ```
   java -classpath ../lib/objectgrid.jar;../applib/secsample.jar com.ibm.websphere.objectgrid.security.sample.guide.SimpleApp
   ```

Die Datei `secsample.jar` enthält die Klasse "SimpleApp".

Die Ausgabe dieses Programms ist wie folgt:

```
Der Kundenname für ID 0001 ist fName lName
```

Sie können auch xsadmin verwenden, um die Map-Größen des Grids "accounting" anzuzeigen.

- Navigieren Sie zum Verzeichnis `ObjectGrid-Stammverzeichnis/bin`.

- Verwenden Sie den Befehl xsadmin mit der Option "-mapSizes" wie folgt:
  - UNIX Linux `xsadmin.sh -g accounting -m mapSet1 -mapSizes`
  - Windows `xsadmin.bat -g accounting -m mapSet1 -mapSizes`

  Sie sehen die folgende Ausgabe.

```
This administrative utility is provided as a sample only and is not to be
considered a fully supported component of the WebSphere eXtreme Scale
product.
Connecting to Catalog service at localhost:1099
********** Displaying Results for Grid - accounting, MapSet - mapSet1
**********
*** Listing Maps for c0 ***
Map Name: customer Partition #: 0 Map Size: 1 Shard Type: Primary
Server Total: 1
Total Domain Count: 1
```

Stopping servers

Container server

Use the following command to stop the container server c0.

UNIX Linux
```
stopOgServer.sh c0 -catalogServiceEndPoints
localhost:2809
```

Windows
```
stopOgServer.bat c0 -catalogServiceEndPoints localhost:2809
```

You will see the following message.

```
CWOBJ2512I: ObjectGrid server c0 stopped.
```

Katalogserver

Sie können einen Katalogserver mit dem folgenden Befehl stoppen.

UNIX Linux
```
stopOgServer.sh catalogServer -catalogServiceEndPoints
localhost:2809
```

Windows
```
stopOgServer.bat catalogServer -catalogServiceEndPoints
localhost:2809
```

Wenn Sie den Katalogserver beenden, wird die folgende Nachricht angezeigt.

```
CWOBJ2512I: ObjectGrid-Server catalogServer wurde gestoppt.
```

Erforderliche Dateien

Die folgende Datei ist die Java-Klasse für die Anwendung "SimpleApp".

SimpleApp.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 (C) COPYRIGHT International Business Machines Corp. 2007-2009
package com.ibm.websphere.objectgrid.security.sample.guide;

import com.ibm.websphere.objectgrid.ClientClusterContext;
import com.ibm.websphere.objectgrid.ObjectGrid;
import com.ibm.websphere.objectgrid.ObjectGridManager;
import com.ibm.websphere.objectgrid.ObjectGridManagerFactory;
import com.ibm.websphere.objectgrid.ObjectMap;
import com.ibm.websphere.objectgrid.Session;

public class SimpleApp {

    public static void main(String[] args) throws Exception {

        SimpleApp app = new SimpleApp();
        app.run(args);
    }

    /**
     * read and write the map
     * @throws Exception
     */
    protected void run(String[] args) throws Exception {
        ObjectGrid og = getObjectGrid(args);

        Session session = og.getSession();

        ObjectMap customerMap = session.getMap("customer");

        String customer = (String) customerMap.get("0001");

        if (customer == null) {
            customerMap.insert("0001", "fName lName");
        } else {
            customerMap.update("0001", "fName lName");
        }
        customer = (String) customerMap.get("0001");

        System.out.println("The customer name for ID 0001 is " + customer);
    }

    /**
     * Get the ObjectGrid
     * @return Eine ObjectGrid-Instanz.
     * @throws Exception
     */
    protected ObjectGrid getObjectGrid(String[] args) throws Exception {
        ObjectGridManager ogManager = ObjectGridManagerFactory.getObjectGridManager();

        // Create an ObjectGrid
        ClientClusterContext ccContext = ogManager.connect("localhost:2809", null, null);
        ObjectGrid og = ogManager.getObjectGrid(ccContext, "accounting");

        return og;

    }

}
```

Die Methode "getObjectGrid" in dieser Klasse ruft ein ObjectGrid ab, und die Methode "run" liest einen Datensatz aus der Map "Customer" und aktualisiert den Wert.

Wenn Sie diesen Mustercode in einer verteilten Umgebung ausführen möchten, müssen Sie eine ObjectGrid-XML-Deskriptordatei `SimpleApp.xml` und eine XML-Implementierungsdatei `SimpleDP.xml` erstellen. Diese Dateien werden im folgenden Beispiel gezeigt:

SimpleApp.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="accounting">
```

```
            <backingMap name="customer" readOnly="false" copyKey="true"/>
        </objectGrid>
    </objectGrids>
</objectGridConfig>
```

Die folgende XML-Datei konfiguriert die Implementierungsumgebung.

SimpleDP.xml

```
<?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="accounting">
  <mapSet name="mapSet1" numberOfPartitions="1" minSyncReplicas="0" maxSyncReplicas="2" maxAsyncReplicas="1">
   <map ref="customer"/>
  </mapSet>
 </objectgridDeployment>
</deploymentPolicy>
```

Dies ist eine einfache ObjectGrid-Konfiguration mit einer einzigen ObjectGrid-Instanz mit dem Namen "accounting" und einer einzigen Map mit dem Namen "customer" (im MapSet "mapSet1"). Die Datei `SimpleDP.xml` enthält ein einziges MapSet mit einer Partition und einer erforderlichen Mindestanzahl von 0 Replikaten.

Nächster Schritt des Lernprogramms

Lernprogramm zur Java-SE-Sicherheit - Schritt 2

Aufbauend auf dem vorherigen Schritt, zeigt der folgende Abschnitt, wie die Clientauthentifizierung in einer verteilten eXtreme-Scale-Umgebung implementiert wird.

Vorbereitende Schritte

Stellen Sie sicher, dass Sie die Anweisungen im Abschnitt „Lernprogramm zur Java-SE-Sicherheit - Schritt 1” auf Seite 207 ausgeführt haben.

Informationen zu diesem Vorgang

Wenn die Clientauthentifizierung aktiviert ist, wird ein Client authentifiziert, bevor eine Verbindung zum eXtreme-Scale-Server hergestellt wird. In diesem Abschnitt wird anhand von Mustercode und Scripts veranschaulicht, wie die Clientauthentifizierung in einer eXtreme-Scale-Serverumgebung durchgeführt werden kann.

Wie jedes andere Authentifizierungsverfahren setzt sich diese minimale Authentifizierung aus den folgenden Schritten zusammen:

1. Der Administrator ändert Konfigurationen, um die Authentifizierung als Voraussetzung festzulegen.
2. Der Client übergibt einen Berechtigungsnachweis an den Server.
3. Der Server authentifiziert den Berechtigungsnachweis anhand der Registry.

Vorgehensweise

1. **Clientberechtigungsnachweis**

   Ein Clientberechtigungsnachweis wird durch eine Schnittstelle des Typs "com.ibm.websphere.objectgrid.security.plugins.Credential" dargestellt. Gültige Clientberechtigungsnachweise 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 der Dokumentation der API "Credential".

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.

eXtreme Scale stellt auch ein Plug-in für die Generierung eines Berechtigungsnachweises bereit. Dieses Plug-in wird durch die Schnittstelle "com.ibm.websphere.objectgrid.security.plugins.CredentialGenerator" dargestellt und wird verwendet, um einen Clientberechtigungsnachweis zu generieren. Dies ist hilfreich, wenn der Berechtigungsnachweis eine Verfallszeit hat. In diesem Fall wird die Methode "getCredential()" aufgerufen, um einen Berechtigungsnachweis zu erneuern. Weitere Einzelheiten finden Sie in der Dokumentation zur API "CredentialGenerator".

Sie können diese beiden Schnittstellen für die eXtreme-Scale-Clientumgebung implementieren, um Clientberechtigungsnachweise abzurufen.

In diesem Mustercode werden die folgenden beiden Muster-Plug-in-Implementierungen verwendet, die von eXtreme Scale bereitgestellt werden.

```
com.ibm.websphere.objectgrid.security.plugins.builtins.UserPasswordCredential
```

```
com.ibm.websphere.objectgrid.security.plugins.builtins.UserPasswordCredentialGenerator
```

Weitere Informationen zu diesen Plug-ins finden Sie im Abschnitt zur Programmierung der Clientauthentifizierung im *Programmierhandbuch*.

2. **Serverauthentifikator** 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 eXtreme-Scale-Server authentifiziert das Credential-Objekt, bevor er die Anforderung verarbeitet. Bei erfolgreicher Authentifizierung des Credential-Objekts wird ein Subject-Objekt zurückgegeben, dass 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 der Dokumentation zur API "Authenticator".

   In diesem Beispiel wird eine integrierte eXtreme-Scale-Implementierung verwendet, die Implementierung "KeyStoreLoginAuthenticator", die für Test- und Beispielzwecke bestimmt ist (ein Keystore ist eine einfache Benutzer-Registry und sollte nicht für eine Produktionsumgebung verwendet werden). "Programmierung der Clientauthentifizierung" im *Programmierungshandbuch*.

   Dieser KeyStoreLoginAuthenticator verwendet ein KeyStoreLoginModule, um den Benutzer über das JAAS-Anmeldemodul "KeyStoreLogin" für den Keystore zu authentifizieren. Der Keystore kann als Option für die Klasse "KeyStoreLoginModule" konfiguriert werden. Im folgenden Beispiel sehen Sie den keyStoreLogin-Alias, der in der JAAS-Konfigurationsdatei "og_jaas.config" konfiguriert ist:

   ```
   KeyStoreLogin{
   com.ibm.websphere.objectgrid.security.plugins.builtins.KeyStoreLoginModule required
        keyStoreFile="../security/sampleKS.jks" debug = true;
   };
   ```

   Die folgenden Befehle erstellen einen Keystore "sampleKS.jks" im Verzeichnis "%OBJECTGRID_HOME%/security" mit dem Kennwort "sampleKS1". Außer-

dem werden drei Benutzerzertifikate mit eigenen Kennwörtern erstellt, die den Benutzer "administrator", den Benutzer "manager" und den Benutzer "cashier" darstellen.

a. Navigieren Sie zum Stammverzeichnis von eXtreme Scale:

```
cd objectgridRoot
```

b. Erstellen Sie ein Verzeichnis mit dem Namen "security":

```
mkdir security
```

c. Navigieren Sie zum neu erstellten Verzeichnis "security":

```
cd security
```

d. Verwenden Sie keytool (im Verzeichnis `javaHOME/bin`), um einen Benutzer "administrator" mit dem Kennwort "administrator1" im Keystore "sampleKS-.jks" zu erstellen:

```
keytool -genkey -v -keystore ./sampleKS.jks -storepass sampleKS1
-alias administrator -keypass administrator1
-dname CN=administrator,O=acme,OU=OGSample -validity 10000
```

e. Verwenden Sie keytool (im Verzeichnis `javaHOME/bin`), um einen Benutzer "manager" mit dem Kennwort "manager1" im Keystore "sampleKS.jks" zu erstellen:

```
keytool -genkey -v -keystore ./sampleKS.jks -storepass sampleKS1
-alias manager -keypass manager1
-dname CN=manager,O=acme,OU=OGSample -validity 10000
```

f. Verwenden Sie keytool (im Verzeichnis `javaHOME/bin`), um einen Benutzer "cashier" mit dem Kennwort "cashier1" im Keystore "sampleKS.jks" zu erstellen:

```
keytool -genkey -v -keystore ./sampleKS.jks -storepass sampleKS1
-alias cashier -keypass cashier1 -dname CN=cashier,O=acme,OU=OGSample
-validity 10000
```

Die Clientsicherheitskonfiguration wird in der Clienteigenschaftendatei konfiguriert. Verwenden Sie die folgenden Befehle, um eine Kopie im Verzeichnis `%OBJECTGRID_HOME%/security` zu erstellen:

a. Wechseln Sie in das Verzeichnis "security":

```
cd objectgridRoot/security
```

b. Kopieren Sie die Datei "sampleClient.properties" in die Datei "client.properties":

```
cp ../properties/sampleClient.properties client.properties
```

Die folgenden Eigenschaften sind in der Datei "client.properties" im Verzeichnis "security" hervorgehoben:

a. **securityEnabled:** Wenn Sie "securityEnabled" auf "true" (Standardwert) setzen, wird die Clientsicherheit aktiviert, die die Authentifizierung umfasst.

b. **credentialAuthentication:** Setzen Sie "credentialAuthentication" auf "Supported" (Standardwert), d. h., der Client unterstützt die Authentifizierung von Berechtigungsnachweisen.

c. **transportType:** Setzen Sie "transportType" auf "TCP/IP", d. h., es wird kein SSL verwendet.

d. **singleSignOnEnabled:** Setzen Sie diese Eigenschaft auf "false" (Standardwert). Single Sign-on (SSO) ist nicht verfügbar.

3. **Serversicherheitskonfiguration**

Die Serversicherheitskonfiguration wird in der XML-Sicherheitsdeskriptordatei und in der Servereigenschaftendatei angegeben.Die XML-Sicherheitsdeskriptordatei beschreibt die Sicherheitseigenschaften, die für alle Server (einschließlich Katalogservern und Containerservern) gelten. Ein Beispiel für eine solche Ei-

genschaft ist die Authentifikatorkonfiguration, die die Benutzer-Registry und das Authentifizierungsverfahren darstellt.

Im Folgenden sehen Sie die Datei `security.xml`, die für diesen Mustercode verwendet wird:

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

 <security securityEnabled="true" loginSessionExpirationTime="300" >

        <authenticator className ="com.ibm.websphere.objectgrid.security.plugins.builtins.KeyStoreLoginAuthenticator"
</authenticator>
    </security>

</securityConfig>
```

a. **securityEnabled:** Wenn diese Einstellung den Wert "true" hat, wird die Serversichrheit, einschließlich Authentifizierung aktiviert.

b. **loginSessionExpirationTime:** Setzen Sie diese Eigenschaft auf 300 (Standardwert).

c. **authenticator:** Fügen Sie die Authentifikatorklasse "KeyStoreLoginAuthenticator" der XML-Custerdatei wie folgt hinzu:

```
<authenticator className ="com.ibm.websphere.objectgrid.security.plugins.builtins.KeyStoreLoginAuthenticator">
        </authenticator>
```

d. **credentialAuthentication:** Setzen Sie das Attribut "credentialAuthentication" auf "Required", damit der Server eine Authentifizierung erfordert.

Eine ausführlichere Erläuterung der Datei `security.xml` finden Sie in den Informationen zur XML-Sicherheitsdeskriptordatei im *Administratorhandbuch*.

Kopieren Sie die Servereigenschaftendatei in das Verzeichnis "security". Dieses Mal müssen Sie keine Änderungen in der Datei vornehmen.

a. Navigieren Sie zum Verzeichnis "security":

```
cd objectgridRoot/security
```

b. Kopieren Sie die Muster-ObjectGrid-Datei `sampleServer.properties` aus dem Verzeichnis "properties" in die neue Datei `server.properties`:

```
cp ../properties/containerServer.properties server.properties
```

Nehmen Sie die folgenden Änderungen in der Datei `server.properties` vor:

a. **securityEnabled:** Setzen Sie das Attribut **securityEnabled** auf "true".

b. **transportType:** Setzen Sie das Attribut **transportType** auf "TCP/IP", d. h., es wird kein SSL verwendet.

c. **secureTokenManagerType:** Setzen Sie das Attribut **secureTokenManagerType** auf "none", damit der Manager für sichere Token nicht konfiguriert wird.

4. **Sicherer Client** Verbinden Sie die Clientanwendung, wie im folgenden Beispiel gezeigt, sicher mit dem Server:

```
// 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 (C) COPYRIGHT International Business Machines Corp. 2007-2009
package com.ibm.websphere.objectgrid.security.sample.guide;

import com.ibm.websphere.objectgrid.ClientClusterContext;
import com.ibm.websphere.objectgrid.ObjectGrid;
import com.ibm.websphere.objectgrid.ObjectGridManager;
import com.ibm.websphere.objectgrid.ObjectGridManagerFactory;
import com.ibm.websphere.objectgrid.security.config.ClientSecurityConfiguration;
import com.ibm.websphere.objectgrid.security.config.ClientSecurityConfigurationFactory;
import com.ibm.websphere.objectgrid.security.plugins.CredentialGenerator;
```

```
import com.ibm.websphere.objectgrid.security.plugins.builtins.UserPasswordCredentialGenerator;

public class SecureSimpleApp extends SimpleApp {

    public static void main(String[] args) throws Exception {

        SecureSimpleApp app = new SecureSimpleApp();
        app.run(args);
    }

    /**
     * Get the ObjectGrid
     * @return Eine ObjectGrid-Instanz.
     * @throws Exception
     */
    protected ObjectGrid getObjectGrid(String[] args) throws Exception {
        ObjectGridManager ogManager = ObjectGridManagerFactory.getObjectGridManager();
        ogManager.setTraceFileName("logs/client.log");
        ogManager.setTraceSpecification("ObjectGrid*=all=enabled:ORBRas=all=enabled");

        // Creates a ClientSecurityConfiguration object using the specified file
        ClientSecurityConfiguration clientSC = ClientSecurityConfigurationFactory
                .getClientSecurityConfiguration(args[0]);

        // Creates a CredentialGenerator using the passed-in user and password.
        CredentialGenerator credGen = new UserPasswordCredentialGenerator(args[1], args[2]);
        clientSC.setCredentialGenerator(credGen);

        // Create an ObjectGrid by connecting to the catalog server
        ClientClusterContext ccContext = ogManager.connect("localhost:2809", clientSC, null);
        ObjectGrid og = ogManager.getObjectGrid(ccContext, "accounting");

        return og;

    }

}
```

Es gibt drei Punkte, in der sich die sichere Anwendung von der nicht gesicherten Anwendung unterscheidet:

a. Es wurde ein ClientSecurityConfiguration-Objekt durch Übergabe der konfigurierten Datei `client.properties` erstellt.
b. Es wurde ein UserPasswordCredentialGenerator-Objekt durch Verwendung der übergebenen Benutzer-ID/Kennwort-Kombination erstellt.
c. Es wurde eine Verbindung zum Katalogserver hergestellt, um ein ObjectGrid vom ClientClusterContext durch Übergabe eines ClientSecurityConfiguration-Objekts abzurufen.

5. **Führen Sie die Anwendung aus.**

   Zum Ausführen der Anwendung starten Sie den Katalogserver. Setzen Sie die Befehlszeilenoptionen "-clusterFile" und -serverProps" zur Übergabe der Sicherheitseigenschaften ab:

   a. Navigieren Sie wie folgt zum Verzeichnis "bin":

      ```
      cd ObjectGrid-Stammverzeichnis/bin
      ```

   b. Starten Sie den Katalogserver:

      - UNIX Linux

        ```
        startOgServer.sh catalogServer -clusterSecurityFile ../security/security.xml
        -serverProps ../security/server.properties -jvmArgs
        -Djava.security.auth.login.config="../security/og_jaas.config"
        ```

      - Windows

        ```
        startOgServer.bat catalogServer -clusterSecurityFile ../security/security.xml
        -serverProps ../security/server.properties -jvmArgs
        -Djava.security.auth.login.config="../security/og_jaas.config"
        ```

   Starten Sie anschließend einen sicheren Containerserver mit dem folgenden Script:

   a. Navigieren Sie erneut zum Verzeichnis "bin":

      ```
      cd ObjectGrid-Stammverzeichnis/bin
      ```

   b. Starten Sie einen sicheren Containerserver:

- Linux UNIX

  ```
  startOgServer.sh c0 -objectgridFile ../xml/SimpleApp.xml
  -deploymentPolicyFile ../xml/SimpleDP.xml
  -catalogServiceEndpoints localhost:2809
  -serverProps ../security/server.properties
  -jvmArgs -Djava.security.auth.login.config="../security/og_jaas.config"
  ```

- Windows

  ```
  startOgServer.bat c0 -objectgridFile ../xml/SimpleApp.xml
  -deploymentPolicyFile ../xml/SimpleDP.xml
  -catalogServiceEndpoints localhost:2809
  -serverProps ../security/server.properties
  -jvmArgs -Djava.security.auth.login.config="../security/og_jaas.config"
  ```

Die Servereigenschaftendatei wird mit der Option "-serverProps" übergeben.

Nachdem Sie den Server gestartet haben, starten Sie den Client mit dem folgenden Befehl:

a. `cd ObjectGrid-Stammverzeichnis/bin`

b.
```
java -classpath ../lib/objectgrid.jar;../applib/secsample.jar
  com.ibm.websphere.objectgrid.security.sample.guide.SecureSimpleApp
../security/client.properties manager manager1
```

Die Datei `secsample.jar` enthält die Klasse "SimpleApp".

SecureSimpleApp verwendet drei Parameter, die in der folgenden Liste aufgeführt sind:

a. Die Datei `../security/client.properties` ist die Datei mit den Clientsicherheitseigenschaften.

b. `manager` ist die Benutzer-ID.

c. `manager1` ist das Kennwort.

Nachdem Sie die Klasse angegeben haben, werden die folgenden Ergebnisse ausgegeben:

Der Kundenname für `ID 0001` ist `fName lName`.

Sie können auch xsadmin verwenden, um die Map-Größen des Grids "accounting" anzuzeigen.

- Navigieren Sie zum Verzeichnis `ObjectGrid-Stammverzeichnis/bin`.
- Verwenden Sie den Befehl `xsadmin` mit der Option "-mapSizes" wie folgt:
  - UNIX Linux `xsadmin.sh -g accounting -m mapSet1 -username manager -password manager1 -mapSizes`
  - Windows `xsadmin.bat -g accounting -m mapSet1 -username manager -password manager1 -mapSizes`

  Sie sehen die folgende Ausgabe.

  ```
  This administrative utility is provided as a sample only and is not to
  be considered a fully supported component of the WebSphere eXtreme
  Scale product.

  Connecting to Catalog service at localhost:1099

  *********** Displaying Results for Grid - accounting, MapSet - mapSet1
  ***********

  *** Listing Maps for c0 ***

  Map Name: customer Partition #: 0 Map Size: 1 Shard Type: Primary

  Server Total: 1

  Total Domain Count: 1
  ```

Jetzt können Sie den Befehl "stopOgServer" verwenden, um den Containerserver- oder Katalogserverprozess zu stoppen. Sie müssen jedoch eine Sicherheits-

konfigurationsdatei angeben. In der Musterclienteigenschaftendatei werden die folgenden beiden Eigenschaften für die Generierung eines Benutzer-ID/Kennwort-Berechtigungsnachweises (manager/manager1) definiert.

```
credentialGeneratorClass=com.ibm.websphere.objectgrid.security.plugins.builtins.UserPasswordCredentialGenerator
credentialGeneratorProps=manager manager1
```

Stoppen Sie den Container "c0" mit dem folgenden Befehl:

- UNIX Linux `stopOgServer.sh c0 -catalogServiceEndPoints localhost:2809 -clientSecurityFile ..\security\client.properties`
- Windows `stopOgServer.bat c0 -catalogServiceEndPoints localhost:2809 -clientSecurityFile ..\security\client.properties`

Wenn Sie die Option "-clientSecurityFile" nicht angeben, wird eine Ausnahme mit der folgenden Nachricht angezeigt:

```
>> SERVER (id=39132c79, host=9.10.86.47) TRACE START:
>> org.omg.CORBA.NO_PERMISSION: Server requires credential
authentication but there is no security context from the client. This
usually happens when the client does not pass a credential the server.
vmcid: 0x0
minor code: 0
completed: No
```

Sie können den Katalogserver auch mit dem folgenden Befehl beenden. Wenn Sie jedoch den nächsten Schritt des Lernprogramms ausführen möchten, können Sie den Katalogserver aktiviert lassen.

- UNIX Linux `stopOgServer.sh catalogServer -catalogServiceEndPoints localhost:2809 -clientSecurityFile ..\security\client.properties`
- Windows `stopOgServer.bat catalogServer -catalogServiceEndPoints localhost:2809 -clientSecurityFile ..\security\client.properties`

Wenn Sie den Katalogserver beenden, wird die folgende Nachricht angezeigt.

```
CWOBJ2512I: ObjectGrid-Server catalogServer wurde gestoppt
```

Sie haben Ihr System jetzt erfolgreich teilweise gesichert, indem Sie die Authentifizierung aktiviert haben. Die haben den Server für die Integration der Benutzer-Registry konfiguriert, den Client für die Bereitstellung von Clientberechtigungsnachweisen konfiguriert und die Clienteigenschaftendatei und die XML-Clusterdatei für die Aktivierung der Authentifizierung geändert.

Wenn Sie ein ungültiges Kennwort angeben, wird eine Ausnahme angezeigt, in der Sie darauf hingewiesen werden, dass der Benutzername oder das Kennwort nicht korrekt ist.

Weitere Einzelheiten zur Clientauthentifizierung finden Sie in den Informationen zur Anwendungsclientauthentifizierung im *Administratorhandbuch*.

Nächster Schritt des Lernprogramms

Lernprogramm zur Java-SE-Sicherheit - Schritt 3

Nach der Authentifizierung eines Clients (wie im vorherigen Schritt) können Sie über die Berechtigungsmechanismen von eXtreme Scale Sicherheitsberechtigungen erteilen.

Vorbereitende Schritte

Stellen Sie sicher, dass Sie die Anweisungen im Abschnitt „Lernprogramm zur Java-SE-Sicherheit - Schritt 2” auf Seite 210 ausgeführt haben, bevor Sie mit dieser Task fortfahren.

Informationen zu diesem Vorgang

Im vorherigen Schritt dieses Lernprogramms wurde veranschaulicht, wie die Authentifizierung in einem eXtreme-Scale-Grid aktiviert wird. Aufgrund der Aktivierung kann kein nicht authentifizierter Client eine Verbindung zu Ihrem Server mehr herstellen und Anforderungen an Ihr System übergeben. Jeder authentifizierte Client hat jedoch dieselben Berechtigungen oder Privilegien beim Server, z. B. Lesen, Schreiben oder Löschen von Daten, die in ObjectGrid-Maps gespeichert sind. Clients können auch jeden Typ von Abfrage absetzen. In diesem Abschnitt wird gezeigt, wie Sie über eXtreme-Scale-Berechtigungen verschiedenen authentifizierten Benutzern unterschiedliche Privilegien erteilen.

Ähnlich wie viele andere Systeme verwendet eXtreme Scale einen rechtebasierten Berechtigungsmechanismus. WebSphere eXtreme Scale hat verschiedene Berechtigungskategorien, die von verschiedenen Berechtigungsklassen dargestellt werden. Hier wird die Berechtigungsklasse "MapPermission" verwendet. Vollständige Informationen zu den Berechtigungskategorien finden Sie in den Referenzinformationen zu Clientberechtigungen im *Programmierhandbuch*..

In WebSphere eXtreme Scale stellt die Klasse "com.ibm.websphere.objectgrid.security.MapPermission" Berechtigungen für die eXtreme-Scale-Ressourcen dar, insbesondere die Methoden der Schnittstellen "ObjectMap" und "JavaMap". WebSphere eXtreme Scale definiert die folgenden Berechtigungszeichenfolgen für den Zugriff auf die Methoden der Schnittstellen "ObjectMap" und "JavaMap":

- read: Erteilt die Berechtigung zum Lesen der Daten aus der Map.
- write: Erteilt die Berechtigung zum Aktualisieren der Daten in der Map.
- insert: Erteilt die Berechtigung zum Einfügen der Daten in die Map.
- remove: Erteilt die Berechtigung zum Entfernen der Daten aus der Map.
- invalidate: Erteilt die Berechtigung zum Ungültigmachen der Daten in der Map.
- all: Erteilt alle zuvor beschriebenen Berechtigungen: read, write, insert, remote und invalidate.

Die Berechtigung findet statt, wenn ein Client eine Methode von ObjectMap oder JavaMap aufruft. Der Laufzeitumgebung von eXtreme Scale prüft verschiedene Map-Berechtigungen für verschiedene Methoden. Wenn dem Client die erforderlichen Berechtigungen nicht erteilt wurden, wird eine Ausnahme des Typs "AccessControlException" ausgegeben.

Dieses Lernprogramm veranschaulicht, wie über JAAS-Berechtigung verschiedenen Benutzern Berechtigungen für Map-Zugriffe erteilt werden.

Vorgehensweise

1. **Aktivieren Sie die eXtreme-Scale-Berechtigung**. Zum Aktivieren der Berechtigung im ObjectGrid müssen Sie das Attribut "securityEnabled" für das gewünschte ObjectGrid in der XML-Datei auf `true` setzen. Die Sicherheit im ObjectGrid zu aktivieren, bedeutet, dass Sie die Berechtigung aktivieren. Verwenden Sie die folgenden Befehle, um eine neue ObjectGrid-XML-Dadtei mit aktivierter Sicherheit zu erstellen.

a. Navigieren Sie zum Verzeichnis bin.

```
cd ObjectGrid-Stammverzeichnis/bin
```

b. Kopieren Sie die Datei SimpleApp.xml in die Datei SecureSimpleApp.xml.

```
cp SimpleApp.xml SecureSimpleApp.xml
```

c. Öffnen Sie die Datei SecureSimpleApp.xml, und fügen Sie, wie in der folgenden XML gezeigt, securityEnabled="true" auf ObjectGrid-Ebene hinzu.

```
<?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" securityEnabled="true">
            <backingMap name="customer" readOnly="false" copyKey="true"/>
        </objectGrid>
    </objectGrids>
</objectGridConfig>
```

2. **Definieren Sie die Berechtigungsrichtlinie.** Im vorherigen Schritt haben Sie zur Vorbereitung der Clientauthentifizierung drei Benutzer im Keystore erstellt: cashier, manager und administrator. In diesem Beispiel wird gezeigt, dass der Benutzer "cashier" nur Leseberechtigungen für alle Maps und der Benutzer "manager" alle Berechtigungen besitzt. In diesem Beispiel wird die JAAS-Berechtigung verwendet. Die JAAS-Berechtigung verwendet eine Berechtigungsrichtliniendatei, um Principals Berechtigungen zu erteilen. Die folgende datei wird im Verzeichnis "security" definiert:

```
grant codebase "http://www.ibm.com/com/ibm/ws/objectgrid/security/PrivilegedAction"
    principal javax.security.auth.x500.X500Principal "CN=cashier,O=acme,OU=OGSample" {
    permission com.ibm.websphere.objectgrid.security.MapPermission "accounting.*", "read ";
};

grant codebase "http://www.ibm.com/com/ibm/ws/objectgrid/security/PrivilegedAction"
    principal javax.security.auth.x500.X500Principal "CN=manager,O=acme,OU=OGSample" {
    permission com.ibm.websphere.objectgrid.security.MapPermission "accounting.*", "all";
};
```

Anmerkung:

- codebase "http://www.ibm.com/com/ibm/ws/objectgrid/security/PrivilegedAction" ist ein speziell reservierter URL für ObjectGrid. Alle ObjectGrid-Berechtigungen, die Principals erteilt werden, müssen diese spezielle Codebasis verwenden.
- Die erste grant-Anweisung erteilt dem Principal "CN=cashier,O=acme,OU=OGSample" die Map-Berechtigung "read", so dass der Benutzer "cashier" ausschließlich Leseberechtigung für alle Maps im ObjectGrid "accounting" hat.
- Die zweite grant-Anweisung erteilt dem Principal "CN=manager,O=acme,OU=OGSample" die Map-Berechtigung "all", so dass der Benutzer "manager" alle Berechtigungen für die Maps im ObjectGrid "accounting" hat.

Jetzt können Sie einen Server mit einer Berechtigungsrichtlinie starten. Die JAAS-Berechtigungsrichtliniendatei kann mit der Standardeigenschaft "-D" definiert werden: -Djava.security.auth.policy=../security/ogAuth.policy

3. **Führen Sie die Anwendung aus.**

Nachdem Sie die zuvor beschriebenen Dateien erstellt haben, können Sie die Anwendung ausführen.

Verwenden Sie die folgenden Befehle, um den Katalogserver zu starten. Weitere Informationen zum Starten des Katalogservice finden Sie in den Informationen zum Starten eines Katalogservice im *Administratorhandbuch*.

a. Navigieren Sie wie folgt zum Verzeichnis "bin": cd ObjectGrid-Stammverzeichnis/bin

b. Starten Sie den Katalogserver:

- UNIX Linux `startOgServer.sh catalogServer -clusterSecurityFile ../security/security.xml -serverProps ../security/server.properties -jvmArgs -Djava.security.auth.login.config="../security/og_jaas.config"`
- Windows `startOgServer.bat catalogServer -clusterSecurityFile ../security/security.xml -serverProps ../security/server.properties -jvmArgs -Djava.security.auth.login.config="../security/og_jaas.config"`

Die Dateien `security.xml` und `server.properties` wurden im vorherigen Schritt dieses Lernprogramms erstellt.

T

c. Anschließend können Sie einen sicheren Containerserver mit dem folgenden Script starten. Führen Sie das Script im Verzeichnis "bin" aus:

- UNIX Linux `# startOgServer.sh c0 -objectGridFile ../xml/SecureSimpleApp.xml -deploymentPolicyFile ../xml/SimpleDP.xml -catalogServiceEndpoints localhost:2809 -serverProps ../security/server.properties -jvmArgs -Djava.security.auth.login.config="../security/og_jaas.config" -Djava.security.auth.policy="../security/og_auth.policy"`
- Windows `startOgServer.bat c0 -objectGridFile ../xml/SecureSimpleApp.xml -deploymentPolicyFile ../xml/SimpleDP.xml -catalogServiceEndpoints localhost:2809 -serverProps ../security/server.properties -jvmArgs -Djava.security.auth.login.config="../security/og_jaas.config" -Djava.security.auth.policy="../security/og_auth.policy"`

Beachten Sie die folgenden Unterschiede zum vorherigen Startbefehl für den Katalogserver:

- Verwenden Sie die Datei `SecureSimpleApp.xml` anstelle der Datei `SimpleAp-p.xml`.
- Es wird eine weitere Eigenschaft `-Djava.security.auth.policy` hinzugefügt, um die JAAS-Berechtigungsrichtliniendatei für den Containerserverprozess zu definieren.

Verwenden Sie denselben Befehl wie im vorherigen Schritt des Lernprogramms.

a. Navigieren Sie zum Verzeichnis "bin".

b. `java -classpath ../lib/objectgrid.jar;../applib/secsample.jar com.ibm.websphere.objectgrid.security.sample.guide.SecureSimpleApp ../security/client.properties manager manager1`

Da der Benutzer "manager" alle Berechtigungen für die Maps im ObjectGrid "accounting" hat, wird die Anwendung ordnungsgemäß ausgeführt.

Jetzt verwenden Sie an Stelle des Benutzers "manager" den Benutzer "cashier", um die Clientanwendung zu starten.

c. Navigieren Sie zum Verzeichnis "bin".

d. `java -classpath ../lib/objectgrid.jar;../applib/secsample.jar com.ibm.ws.objectgrid.security.sample.guide.SecureSimpleApp ../security/client.properties cashier cashier1`

Die folgende Ausnahme wird ausgegeben:

```
Exception in thread "P=387313:O=0:CT" com.ibm.websphere.objectgrid.TransactionException:
rolling back transaction, see caused by exception
 at com.ibm.ws.objectgrid.SessionImpl.rollbackPMapChanges(SessionImpl.java:1422)
  at com.ibm.ws.objectgrid.SessionImpl.commit(SessionImpl.java:1149)
```

```
  at com.ibm.ws.objectgrid.SessionImpl.mapPostInvoke(SessionImpl.java:2260)
  at com.ibm.ws.objectgrid.ObjectMapImpl.update(ObjectMapImpl.java:1062)
  at com.ibm.ws.objectgrid.security.sample.guide.SimpleApp.run(SimpleApp.java:42)
 at com.ibm.ws.objectgrid.security.sample.guide.SecureSimpleApp.main(SecureSimpleApp.java:27)
Caused by: com.ibm.websphere.objectgrid.ClientServerTransactionCallbackException:
   Client Services - received exception from remote server:
     com.ibm.websphere.objectgrid.TransactionException: transaction rolled back,
   see caused by Throwable
        at com.ibm.ws.objectgrid.client.RemoteTransactionCallbackImpl.processReadWriteResponse(
            RemoteTransactionCallbackImpl.java:1399)
        at com.ibm.ws.objectgrid.client.RemoteTransactionCallbackImpl.processReadWriteRequestAndResponse(
            RemoteTransactionCallbackImpl.java:2333)
        at com.ibm.ws.objectgrid.client.RemoteTransactionCallbackImpl.commit(RemoteTransactionCallbackImpl.java:557)
        at com.ibm.ws.objectgrid.SessionImpl.commit(SessionImpl.java:1079)
        ... 4 more
Caused by: com.ibm.websphere.objectgrid.TransactionException: transaction rolled back, see caused by Throwable
        at com.ibm.ws.objectgrid.ServerCoreEventProcessor.processLogSequence(ServerCoreEventProcessor.java:1133)
        at com.ibm.ws.objectgrid.ServerCoreEventProcessor.processReadWriteTransactionRequest
     (ServerCoreEventProcessor.java:910)
        at com.ibm.ws.objectgrid.ServerCoreEventProcessor.processClientServerRequest(ServerCoreEventProcessor.java:1285)

        at com.ibm.ws.objectgrid.ShardImpl.processMessage(ShardImpl.java:515)
        at com.ibm.ws.objectgrid.partition.IDLShardPOA._invoke(IDLShardPOA.java:154)
        at com.ibm.CORBA.poa.POAServerDelegate.dispatchToServant(POAServerDelegate.java:396)
        at com.ibm.CORBA.poa.POAServerDelegate.internalDispatch(POAServerDelegate.java:331)
        at com.ibm.CORBA.poa.POAServerDelegate.dispatch(POAServerDelegate.java:253)
        at com.ibm.rmi.iiop.ORB.process(ORB.java:503)
        at com.ibm.CORBA.iiop.ORB.process(ORB.java:1553)
        at com.ibm.rmi.iiop.Connection.respondTo(Connection.java:2680)
        at com.ibm.rmi.iiop.Connection.doWork(Connection.java:2554)
        at com.ibm.rmi.iiop.WorkUnitImpl.doWork(WorkUnitImpl.java:62)
        at com.ibm.rmi.iiop.WorkerThread.run(ThreadPoolImpl.java:202)
        at java.lang.Thread.run(Thread.java:803)
Caused by: java.security.AccessControlException: Access denied (
   com.ibm.websphere.objectgrid.security.MapPermission accounting.customer write)
        at java.security.AccessControlContext.checkPermission(AccessControlContext.java:155)
        at com.ibm.ws.objectgrid.security.MapPermissionCheckAction.run(MapPermissionCheckAction.java:141)
        at java.security.AccessController.doPrivileged(AccessController.java:275)
        at javax.security.auth.Subject.doAsPrivileged(Subject.java:727)
        at com.ibm.ws.objectgrid.security.MapAuthorizer$1.run(MapAuthorizer.java:76)
        at java.security.AccessController.doPrivileged(AccessController.java:242)
        at com.ibm.ws.objectgrid.security.MapAuthorizer.check(MapAuthorizer.java:66)
        at com.ibm.ws.objectgrid.security.SecuredObjectMapImpl.checkMapAuthorization(SecuredObjectMapImpl.java:429)
        at com.ibm.ws.objectgrid.security.SecuredObjectMapImpl.update(SecuredObjectMapImpl.java:490)
        at com.ibm.ws.objectgrid.SessionImpl.processLogSequence(SessionImpl.java:1913)
        at com.ibm.ws.objectgrid.SessionImpl.processLogSequence(SessionImpl.java:1805)
        at com.ibm.ws.objectgrid.ServerCoreEventProcessor.processLogSequence(ServerCoreEventProcessor.java:1011)
        ... 14 more
```

Diese Ausnahme tritt ein, weil der Benutzer "cashier" keine Schreibberechtigung hat und deshalb die Map "Customer" nicht aktualisieren kann.

Jetzt unterstützt Ihr System Berechtigung. Sie können Berechtigungsrichtlinien definieren, um unterschiedlichen Benutzern unterschiedliche Berechtigungen zu erteilen. Weitere Informationen zur Berechtigung finden Sie in den Informationen zur Anwendungsclientberechtigung im *Programmierhandbuch*.

Nächste Schritte

Führen Sie den nächsten Schritt des Lernprogramms aus (siehe „Lernprogramm zur Java-SE-Sicherheit - Schritt 4").

Lernprogramm zur Java-SE-Sicherheit - Schritt 4

Im folgenden Schritt wird erläutert, wie Sie eine Sicherheitsschicht für die Kommunikation zwischen den Endpunkten Ihrer Umgebung aktivieren.

Vorbereitende Schritte

Stellen Sie sicher, dass Sie die Anweisungen im Abschnitt „Lernprogramm zur Java-SE-Sicherheit - Schritt 3" auf Seite 216 ausgeführt haben, bevor Sie mit dieser Task fortfahren.

Informationen zu diesem Vorgang

Die eXtreme-Scale-Topologie unterstützt Transport Layer Security/Secure Sockets Layer (TLS/SSL) für die sichere Kommunikation zwischen ObjectGrid-Endpunkten (Client, Containerserver und Katalogserver). Dieser Schritt des Lernprogramms zum Aktivieren der Transportsicherheit baut auf den vorherigen Schritten auf.

Vorgehensweise

1. **Erstellen Sie TLS/SSL-Schlüssel und Keystores.**

   Zum Akivieren der Transportsicherheit müssen Sie einen Keystore und einen Truststore erstellen. In dieser Übung wird nur ein einziges Keystore/Truststore-Paar erstellt. Diese Speicher werden für ObjectGrid-Clients, Containerserver und Katalogserver verwendet und mit keytool von JDK erstellt.

   - *Privaten Schlüssel im Keystore erstellen*

     ```
     keytool -genkey -alias ogsample -keystore key.jks -storetype JKS
     -keyalg rsa -dname "CN=ogsample, OU=Your Organizational Unit, O=Your
     Organization, L=Your City, S=Your State, C=Your Country" -storepass
     ogpass -keypass ogpass -validity 3650
     ```

     Mit diesem Befehl wird ein Keystore "key.jks" erstellt, in dem ein Schlüssel "ogsample" gespeichert ist. Dieser Keystore "key.jks" wird als SSL-Keystore verwendet.

   - *Öffentliches Zertifikat exportieren*

     ```
     keytool -export -alias ogsample -keystore key.jks -file temp.key
     -storepass ogpass
     ```

     Mit diesem Befehl wird das öffentliche Zertifikat des Schlüssels "ogsample" extrahiert und in der Datei "temp.key" gespeichert.

   - *Öffentliches Zertifikat des Clients in den Truststore importieren*

     ```
     keytool -import -noprompt -alias ogsamplepublic -keystore trust.jks
     -file temp.key -storepass ogpass
     ```

     Mit diesem Befehl wird das öffentliche Zertifikat dem Keystore "trust.jks" hinzugefügt. Dieser Keystore "trust.jks" wird als SSL-Truststore verwendet.

2. **ObjectGrid-Eigenschaftendateien konfigurieren**

   In diesem Schritt müssen Sie die ObjectGrid-Eigenschaftendateien für die Aktivierung der Transportsicherheit konfigurieren.

   Kopieren Sie zuerst die Dateien "key.jks" und "trust.jks" in das Verzeichnis "Objectgrid-Stammverzeichnis/security".

   Die folgenden Eigenschaften werden in den Dateien "client.properties" und "server.properties" definiert.

   ```
   transportType=SSL-Required

   alias=ogsample
   contextProvider=IBMJSSE2
   protocol=SSL
   keyStoreType=JKS
   keyStore=../security/key.jks
   keyStorePassword=ogpass
   trustStoreType=JKS
   trustStore=../security/trust.jks
   trustStorePassword=ogpass
   ```

   transportType: Die Eigenschaft "transportType" wird auf "SSL-Required" gesetzt, d. h., für den Transport ist SSL erforderlich. Deshalb muss für alle ObjectGrid-Endpunkte (Clients, Katalogserver und Containerserver) SSL konfiguriert werden, und die gesamte Transportkommunikation wird verschlüsselt.

Die anderen Eigenschaften werden zum Definieren der SSL-Konfigurationen verwendet. Eine ausführliche Beschreibung finden Sie in den Informationen zur Sicherheit auf Transportebene und zu Secure Sockets Layer im *Administratorhandbuch*. Stellen Sie sicher, dass Sie die Anweisungen in diesem Abschnitt befolgen, um Ihre Datei "orb.properties" zu aktualisieren.

Stellen Sie sicher, dass Sie den folgenden Anweisungen folgen, um Ihre Datei `orb.properties` zu aktualisieren.

In der Datei `server.properties` müssen Sie eine zusätzliche Eigenschaft "clientAuthentication" hinzufügen und diese auf "false" setzen. Auf der Serverseite müssen Sie den Client nicht anerkennen.

```
clientAuthentication=false
```

3. **Anwendung ausführen**

   Die Befehle sind dieselben wie im Abschnitt „Lernprogramm zur Java-SE-Sicherheit - Schritt 3" auf Seite 216.

   Verwenden Sie die folgenden Befehle, um einen Katalogserver zu starten:

   a. Navigieren Sie wie folgt zum Verzeichnis "bin": `cd ObjectGrid-Stammverzeichnis/bin`

   b. Starten Sie den Katalogserver:

      - Linux UNIX

        ```
        startOgServer.sh catalogServer -clusterSecurityFile ../security/security.xml
        -serverProps ../security/server.properties -JMXServicePort 11001
        -jvmArgs -Djava.security.auth.login.config="../security/og_jaas.config"
        ```

      - Windows

        ```
        startOgServer.bat catalogServer -clusterSecurityFile ../security/security.xml
        -serverProps ../security/server.properties -JMXServicePort 11001 -jvmArgs
        -Djava.security.auth.login.config="../security/og_jaas.config"
        ```

      Die Dateien `security.xml` und `server.properties` wurden im Abschnitt „Lernprogramm zur Java-SE-Sicherheit - Schritt 2" auf Seite 210 erstellt.

      Verwenden Sie die Option "-JMXServicePort", um den JMX-Port für den Server explizit anzugeben. Diese Option ist erforderlich, um den Befehl "xsadmin" verwenden zu können.

      Führen Sie einen sicheren ObjectGrid-Containerserver aus:

   c. Navigieren Sie erneut zum Verzeichnis "bin": `cd ObjectGrid-Stammverzeichnis/bin`

   d.

      - Linux UNIX

        ```
        startOgServer.sh c0 -objectGridFile ../xml/SecureSimpleApp.xml
        -deploymentPolicyFile ../xml/SimpleDP.xml -catalogServiceEndpoints
        localhost:2809 -serverProps ../security/server.properties
        -JMXServicePort 11002 -jvmArgs
        -Djava.security.auth.login.config="../security/og_jaas.config"
        -Djava.security.auth.policy="../security/og_auth.policy"
        ```

      - Windows

        ```
        startOgServer.bat c0 -objectGridFile ../xml/SecureSimpleApp.xml
        -deploymentPolicyFile ../xml/SimpleDP.xml -catalogServiceEndpoints localhost:2809
        -serverProps ../security/server.properties -JMXServicePort 11002
        -jvmArgs -Djava.security.auth.login.config="../security/og_jaas.config"
        -Djava.security.auth.policy="../security/og_auth.policy"
        ```

      Beachten Sie die folgenden Unterschiede zum vorherigen Startbefehl für den Katalogserver:

      - Es wird die Datei "SecureSimpleApp.xml" an Stelle der Datei "SimpleApp.xml" verwendet.
      - Es wird eine weitere Eigenschaft "-Djava.security.auth.policy" hinzugefügt, um die JAAS-Berechtigungsrichtliniendatei für den Containerserverprozess zu definieren.

Führen Sie den folgenden Befehl für die Clientauthentifizierung aus:

a. `cd ObjectGrid-Stammverzeichnis/bin`

b.

```
javaHome/java -classpath ../lib/objectgrid.jar;../applib/secsample.jar
com.ibm.websphere.objectgrid.security.sample.guide.SecureSimpleApp
../security/client.properties manager manager1
```

Da der Benutzer "manager" die Berechtigung für alle Maps im ObjectGrid "accounting" hat, wird die Anwendung erfolgreich ausgeführt.

Sie können auch xsadmin verwenden, um die Map-Größen des Grids "accounting" anzuzeigen.

- Navigieren Sie zum Verzeichnis `ObjectGrid-Stammverzeichnis/bin`.
- Verwenden Sie den Befehl `xsadmin` mit der Option "-mapSizes" wie folgt:
  - UNIX Linux

    ```
    xsadmin.sh -g accounting -m mapSet1 -mapsizes -p 11001 -ssl
    -trustpath ..\security\trust.jks -trustpass ogpass -trusttype jks
    -username manager -password manager1
    ```

  - Windows

    ```
    xsadmin.bat -g accounting -m mapSet1 -mapsizes -p 11001 -ssl
    -trustpath ..\security\trust.jks -trustpass ogpass -trusttype jks
    -username manager -password manager1
    ```

  Hier geben Sie den JMX-Port des Katalogservice mit "-p 11001" an.

  Sie sehen die folgende Ausgabe.

```
This administrative utility is provided as a sample only and is not to
be considered a fully supported component of the WebSphere eXtreme Scale product.
Connecting to Catalog service at localhost:1099
********** Displaying Results for Grid - accounting, MapSet - mapSet1 **********
*** Listing Maps for c0 ***
Map Name: customer Partition #: 0 Map Size: 1 Shard Type: Primary
Server Total: 1
Total Domain Count: 1
```

Anwendung mit einem ungültigen Keystore ausführen

Wenn Ihr Truststore das öffentliche Zertifikat zum privaten Schlüssel im Keystore nicht enthält, wird eine Ausnahme angezeigt, in der erläutert wird, dass der Schlüssel nicht anerkannt werden kann.

Zur Demonstration erstellen Sie einen weiteren Keystore mit dem Namen "key2.jks".

```
keytool -genkey -alias ogsample -keystore key2.jks -storetype JKS
-keyalg rsa -dname "CN=ogsample, OU=Your Organizational Unit, O=Your
Organization, L=Your City, S=Your State, C=Your Country" -storepass
ogpass -keypass ogpass -validity 3650
```

Ändern Sie anschließend die Datei "server.properties" so, dass "keyStore" auf diesen neuen Keystore "key2.jks" zeigt:

```
keyStore=../security/key2.jks
```

Führen Sie den folgenden Befehl aus, um den Katalogserver zu starten:

a. Navigieren Sie wie folgt zum Verzeichnis "bin": `cd ObjectGrid-Stammverzeichnis/bin`

b. Starten Sie den Katalogserver:

   Linux UNIX

```
startOgServer.sh c0 -objectGridFile ../xml/SecureSimpleApp.xml
-deploymentPolicyFile ../xml/SimpleDP.xml -catalogServiceEndpoints localhost:2809
-serverProps ../security/server.properties -jvmArgs
-Djava.security.auth.login.config="../security/og_jaas.config"
-Djava.security.auth.policy="../security/og_auth.policy"
```

   Windows

```
startOgServer.bat c0 -objectGridFile ../xml/SecureSimpleApp.xml
-deploymentPolicyFile ../xml/SimpleDP.xml -catalogServiceEndpoints localhost:2809
 -serverProps ../security/server.properties -jvmArgs
-Djava.security.auth.login.config="../security/og_jaas.config"
-Djava.security.auth.policy="../security/og_auth.policy"
```

Sie sehen die folgende Ausgabe:

```
Caused by: com.ibm.websphere.objectgrid.ObjectGridRPCException:
    com.ibm.websphere.objectgrid.ObjectGridRuntimeException:
        SSL connection fails and plain socket cannot be used.
```

Ändern Sie abschließend die Datei `server.properties` wieder so zurück, dass sie die Datei `key.jks` verwendet.

Muster und Lernprogramm zum REST-Datenservice

In diesem Abschnitt wird beschrieben, wie Sie schnell mit der Verwendung des REST-Datenservice von WebSphere eXtreme Scale beginnen können. Anweisungen werden für WebSphere Application Server Version 7.0, WebSphere Application Server Community Edition und Apache Tomcat bereitgestellt.

Informationen zu diesem Vorgang

Das enthaltene Muster setzt sich aus Quellcode und kompilierten Binärdateien für die Ausführung eines partitionierten eXtreme-Scale-Grids zusammen. Dieses Muster veranschaulicht, wie ein einfaches Grid, das Modell und die Daten mit eXtreme-Scale-Entitäten erstellt werden und stellt zwei Befehlszeilenclientanwendungen bereit, mit denen Entitäten unter Verwendung von Java oder C# (siehe Abbildung 1) hinzugefügt und abgefragt werden können.

Der Java-Musterclient verwendet die Java-API "EntityManager" von eXtreme Scale, um Daten im Grid persistent zu speichern und Daten aus dem Grid abzufragen. Dieser Client kann in Eclipse oder über ein Befehlszeilenscript ausgeführt werden. Der Java-Musterclient demonstriert den REST-Datenservice zwar nicht, ermöglicht aber die Aktualisierung der Daten im Grid, so dass ein Webbrowser oder andere Clients die Daten lesen können. Der Java-Musterclient und der Webbrowser, die in Abbildung 1 gezeigt werden veranschaulichen HTTP-Clients, die den REST-Datenservice verwenden, und Java-Clients von eXtreme Scale, die dasselbe eXtreme-Scale-Grid und die darin enthaltenen Daten verwenden.

Der in C# geschriebene Microsoft-WCF-Data-Services-Musterclient kommuniziert mit dem eXtreme-Scale-Grid über den REST-Datenservice unter Verwendung des .NET-Frameworks. Der WCF-Data-Services-Client kann zum Aktualisieren und Abfragen des Grids verwendet werden.

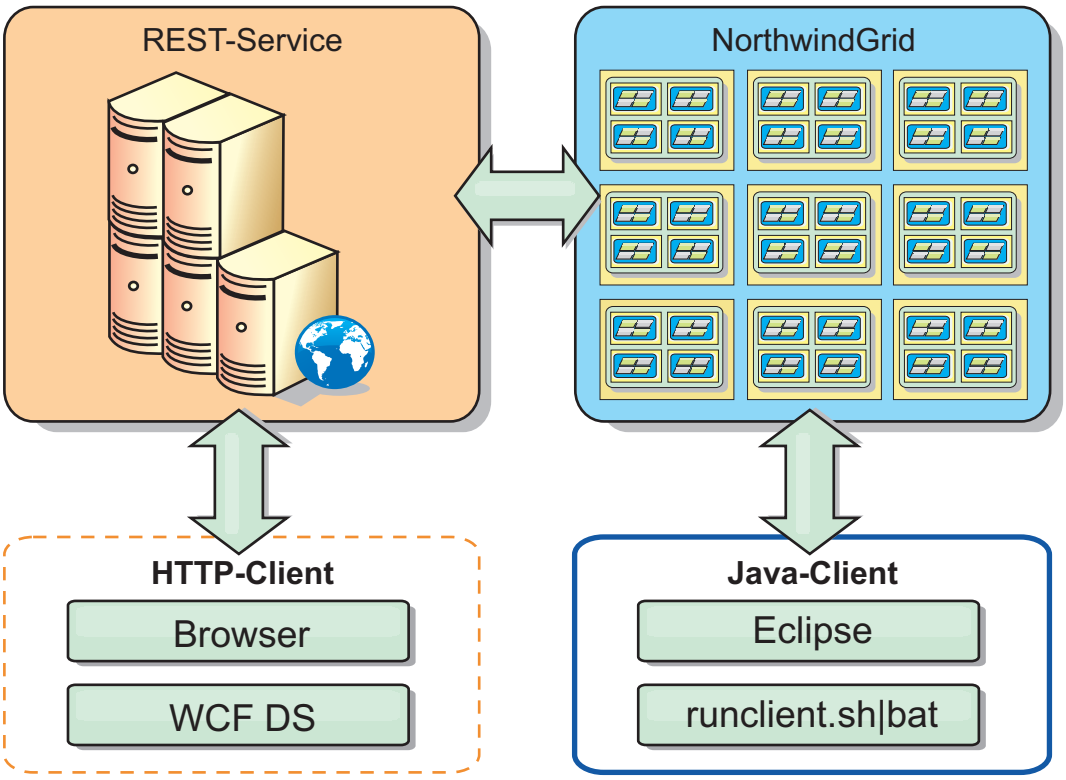


Abbildung 44. Mustertopologie zur Einführung

Vorgehensweise

1. Konfigurieren und starten Sie das eXtreme-Scale-Grid. Weitere Informationen finden Sie unter „REST-Datenservice aktivieren" auf Seite 227.
2. Konfigurieren und starten Sie den REST-Datenservice in einem Webserver. Weitere Informationen finden Sie unter „Anwendungsserver für den REST-Datenservice konfigurieren" auf Seite 236.
3. Führen Sie einen Client aus, um mit dem REST-Datenservice zu interagieren. Es sind zwei Optionen verfügbar:
   a. Führen Sie den Java-Musterclient aus, um das Grid unter Verwendung der API "EntityManager" mit Daten zu füllen und die Daten im Grid unter Verwendung eines Webbrowsers und des REST-Datenservice von eXtreme Scale abzufragen. Lesen Sie hierzu den Artikel „Java-Client mit REST-Datenservices verwenden" auf Seite 246.
   b. Führen Sie den in C# geschriebenen WCF-Data-Services-Musterclient aus. Lesen Sie hierzu den Artikel „WCF-Client von Visual Studio 2008 mit dem REST-Datenservice" auf Seite 248.

Verzeichniskonventionen

In diesem Abschnitt werden zahlreiche Beispiele und die Befehlszeilensyntax beschrieben, in denen spezielle Verzeichnisse wie das *WXS-Installationsstammverzeichnis* und *WXS-Ausgangsverzeichnis* referenziert werden müssen. Diese Verzeichnisse sind wie folgt definiert.

WXS-Installationsstammverzeichnis
Das Verzeichnis *WXS-Installationsstammverzeichnis* ist das Stammverzeichnis, in dem die Produktdateien von WebSphere eXtreme Scale installiert sind. Dies

kann das Verzeichnis sein, in dem die ZIP-Datei mit der Testversion entpackt wurde, oder das Verzeichnis, in dem das vollständige Produkt eXtreme Scale installiert ist.

- Beispiel für die entpackte Testversion:

  `/opt/IBM/WebSphere/eXtremeScale`
- Beispiel für eine Installation von eXtreme Scale in einem eigenständigen Verzeichnis:

  `/opt/IBM/eXtremeScale`
- Beispiel für die Integration von eXtreme Scale mit WebSphere Application Server:

  `/opt/IBM/WebSphere/AppServer`

WXS-Ausgangsverzeichnis
Das Verzeichnis *WXS-Ausgangsverzeichnis* ist das Stammverzeichnis der Produktbibliotheken, Muster und Komponenten von WebSphere eXtreme Scale. Dieses Verzeichnis ist identisch mit dem Verzeichnis *WXS-Installationsstammverzeichnis*, wenn die Testversion entpackt wird. Bei eigenständigen Installationen ist dies das ObjectGrid-Unterverzeichnis im *WXS-Installationsstammverzeichnis*. Bei Installationen, die mit WebSphere Application Server integriert sind, ist dieses Verzeichnis das Verzeichnis `optionalLibraries/ObjectGrid` im *WXS-Installationsstammverzeichnis*.

- Beispiel für die entpackte Testversion:

  `/opt/IBM/WebSphere/eXtremeScale`
- Beispiel für eine Installation von eXtreme Scale in einem eigenständigen Verzeichnis:

  `/opt/IBM/eXtremeScale/ObjectGrid`
- Beispiel für die Integration von eXtreme Scale mit WebSphere Application Server:

  `/opt/IBM/WebSphere/AppServer/optionalLibraries/ObjectGrid`

WAS-Stammverzeichnis
Das Verzeichnis *WAS-Stammverzeichnis* ist das Stammverzeichnis einer Installation von WebSphere Application Server:

`/opt/IBM/WebSphere/AppServer`

Ausgangsverzeichnis_des_REST-Service
Das Verzeichnis *Ausgangsverzeichnis_des_REST-Service* ist das Verzeichnis, in dem sich die Bibliotheken und Muster des REST-Datenservice von eXtreme Scale befinden. Das Verzeichnis hat den Namen *restservice* und ist ein Unterverzeichnis des Verzeichnisses *WXS-Ausgangsverzeichnis*.

- Beispiel für eigenständige Implementierungen:

  `/opt/IBM/WebSphere/eXtremeScale/ObjectGrid/restservice`
- Beispiel für integrierte Implementierungen mit WebSphere Application Server:

  `/opt/IBM/WebSphere/AppServer/optionalLibraries/ObjectGrid/restservice`

Tomcat-Stammverzeichnis
Das *Tomcat-Stammverzeichnis* ist das Stammverzeichnis der Apache-Tomcat-Installation.

`/opt/tomcat5.5`

WASCE-Stammverzeichnis
Das *WASCE-Stammverzeichnis* ist das Stammverzeichnis der Installation von WebSphere Application Server Community Edition.

```
/opt/IBM/WebSphere/AppServerCE
```

Java-Ausgangsverzeichnis
Das *Java-Ausgangsverzeichnis* ist das Stammverzeichnis der Installation von Java Runtime Environment (JRE).

```
/opt/IBM/WebSphere/eXtremeScale/java
```

REST-Datenservice aktivieren

Der REST-Datenservice kann Entitätsmetadaten von WebSphere eXtreme Scale für die Darstellung jeder Entität als EntitySet darstellen.

eXtreme-Scale-Muster-Grid starten

Im Allgemeinen starten Sie das eXtreme-Scale-Grid, bevor Sie den REST-Datenservice starten. Mit den folgenden Schritten werden ein einziger Katalogserviceprozess von eXtreme Scale und zwei Containerserverprozesse gestartet.

Für die Installation von WebSphere eXtreme Scale stehen drei verschiedene Methoden zur Verfügung:

- Testinstallation
- Eigenständige Implementierung
- Integrierte Implementierung von WebSphere Application Server

Skalierbares Datenmodell in eXtreme Scale

Im Muster "Microsoft Northwind" wird die Tabelle "Order Detail" verwendet, um eine N:N-Assoziation zwischen "Orders" und "Products" herzustellen.

Mit Hilfe von ORM-Spezifikationen (Object to Relational Mapping) wie ADO.NET Entity Framework und Java Persistence API (JPA) können die Tabellen und Beziehungen über Entitäten zugeordnet werden. Diese Architektur ist jedoch nicht skalierbar. Alle Komponenten müssen sich auf derselben Maschine oder in einem kostenintensiven Maschinencluster finden, um eine angemessene Leistung zu erzielen.

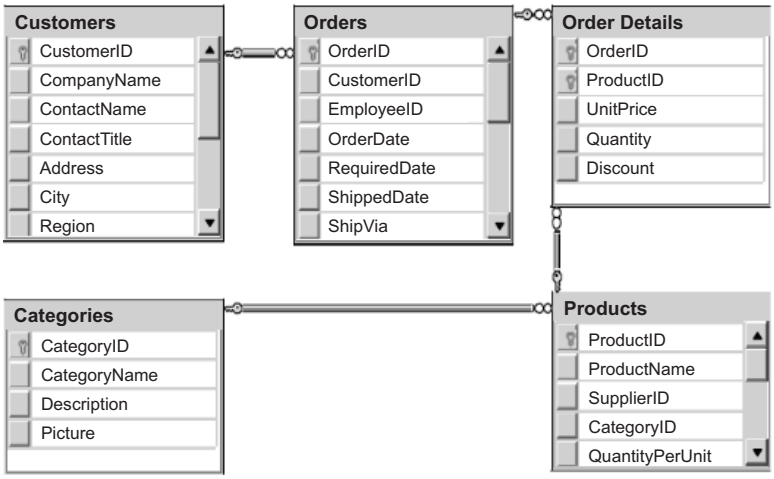


Abbildung 45. Schemadiagramm zum Muster "Microsoft SQL Server Northwind"

Zum Erstellen einer skalierbaren Version des Musters müssen die Entitäten so modelliert werden, dass jede Entität oder Gruppe zusammengehöriger Entitäten auf der Basis eines einzigen Schlüssels partitioniert werden können. Durch die Erstellung von Partitionen auf der Basis eines einzigen Schlüssels können Anforderungen auf mehrere unabhängige Server verteilt werden. Für diese Konfiguration wurden die Entitäten in zwei Baumstrukturen aufgeteilt, in die Baumstruktur "Customer und Order" und in die Baumstruktur "Product und Category". In diesem Modell können beide Baumstrukturen unabhängig voneinander partitioniert werden und deshalb unterschiedlich anwachsen, was die Skalierbarkeit erhöht.

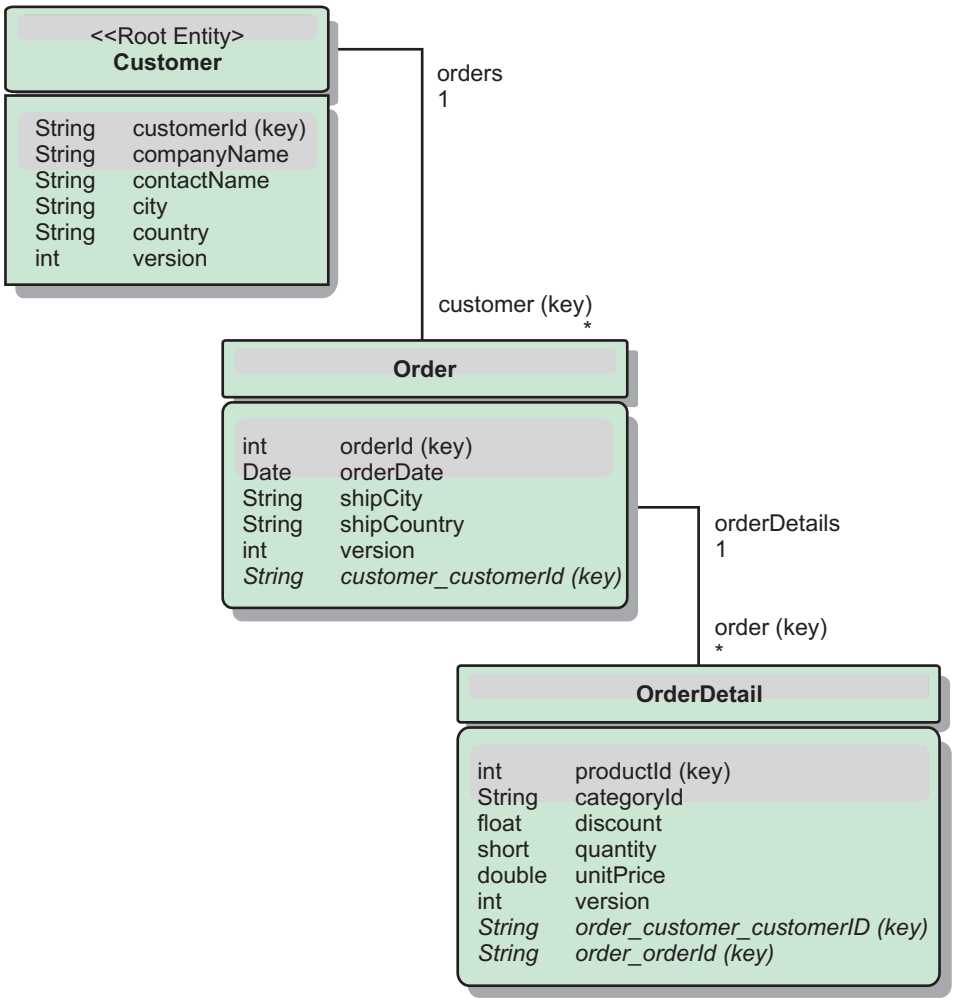


Abbildung 46. Entitätsschemadiagramm "Customer und Order"

Order und Product haben eindeutige, separate ganze Zahlen als Schlüssel. Die Tabellen "Order" und "Product" sind also voneinander unabhängig. Stellen Sie sich beispielsweise die Auswirkungen der Größe eines Katalogs (Anzahl der von Ihnen vertriebenen Produkte) mit der Gesamtanzahl an Bestellungen vor. Intuitiv geht man davon aus, dass das Vorhandensein vieler Produkte bedeutet, dass auch viele Bestellungen vorhanden sind, aber das muss nicht unbedingt der Fall sein. Würde diese Annahme stimmen, könnten Sie Ihre Verkaufszahlen problemlos steigern, indem Sie Ihrem Katalog einfach weitere Produkte hinzufügen. Bestellungen und Produkte haben eigene voneinander unabhängige Tabellen. Sie können dieses Konzept noch erweitern und für Bestellungen und Produkte jeweils eigene separate Daten-Grids verwenden. Mit unabhängigen Grids können Sie die Anzahl der Partitionen und Servern sowie die Größe jedes Grids gesondert steuern, so dass Ihre Anwendung skalieren kann. Wenn Sie die Größe Ihres Katalogs verdoppeln, müssen Sie das Produkt-Grid verdoppeln, aber das Grid für die Bestellungen bleibt unverändert. Der umgekehrte Fall gilt für eine Bestellspitze oder eine erwartete Bestellspitze.

In dem Schema hat ein Kunde (Customer) null oder mehr Bestellungen (Order), und eine Bestellung Bestellpositionen (OrderDetail), jeweils mit einem bestimmten Produkt. Ein Produkt (Product) wird in jedem OrderDetail anhand seiner ID (dem Produktschlüssel) identifiziert. Customer, Order und OrderDetails werden in einzigen Grid gespeichert, mit Customer als Stammentität des Grids. Sie können Kun-

den nach ID abrufen, müssen Bestellungen aber anhand der Kunden-ID abrufen. Deshalb wird die Kunden-ID der Bestellung im Schlüssel hinzugefügt. Analog dazu sind Kunden-ID und Bestell-ID Teil der OrderDetail-ID.

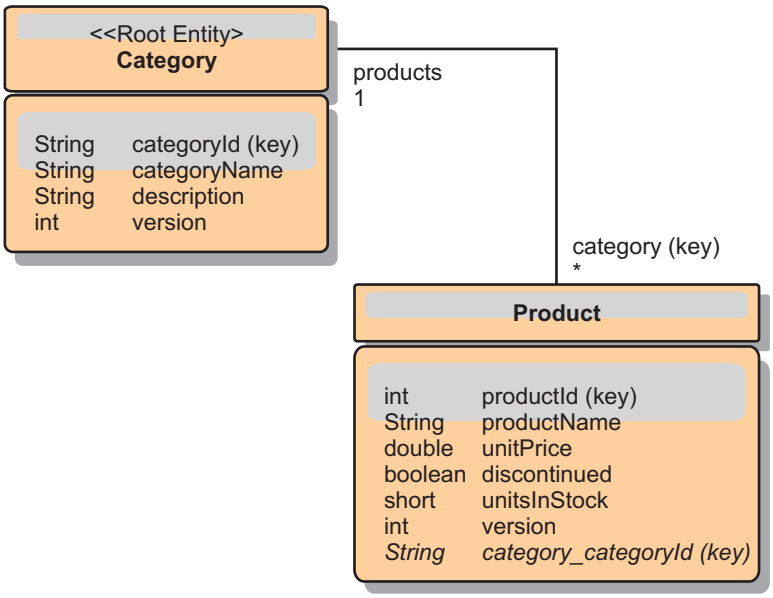


Abbildung 47. Entitätsschemadiagramm "Category und Product"

Im Schema "Category and Product" ist "Category" der Schemastamm. Mit diesem Schema können Kunden Produkte nach Kategorie abfragen. Weitere Einzelheiten zu Schlüsselassoziationen und deren Bedeutung finden Sie unter „Daten mit REST abrufen und aktualisieren".

Daten mit REST abrufen und aktualisieren

Das Protokoll "OData" erfordert, dass alle Entitäten über ihre kanonische Form adressiert werden können. Das bedeutet, dass jede Entität den Schlüssel der partitionierten Stammentität (den Schemastamm) enthalten muss.

Im Folgenden sehen Sie ein Beispiel für die Verwendung der Assoziation von einer Stammentität für die Adressierung einer untergeordneten Entität in:

```
/Customer('ACME')/order(100)
```

In WCF Data Services muss die untergeordnete Entität direkt adressierbar sein, d. h., der Schlüssel im Schemastamm muss Teil des Schlüssels der untergeordneten Entität sein: /Order(customer_customerId='ACME', orderId=100). Dies wird erreicht, indem eine Assoziation zur Stammentität erstellt wird, wobei die 1:1- bzw. N:1-Assoziation zur Stammentität auch als Schlüssel bezeichnet wird. Wenn Entitäten in den Schlüssel eingeschlossen werden, werden die Attribute der übergeordneten Entität als Schlüsseleigenschaften bereitgestellt.

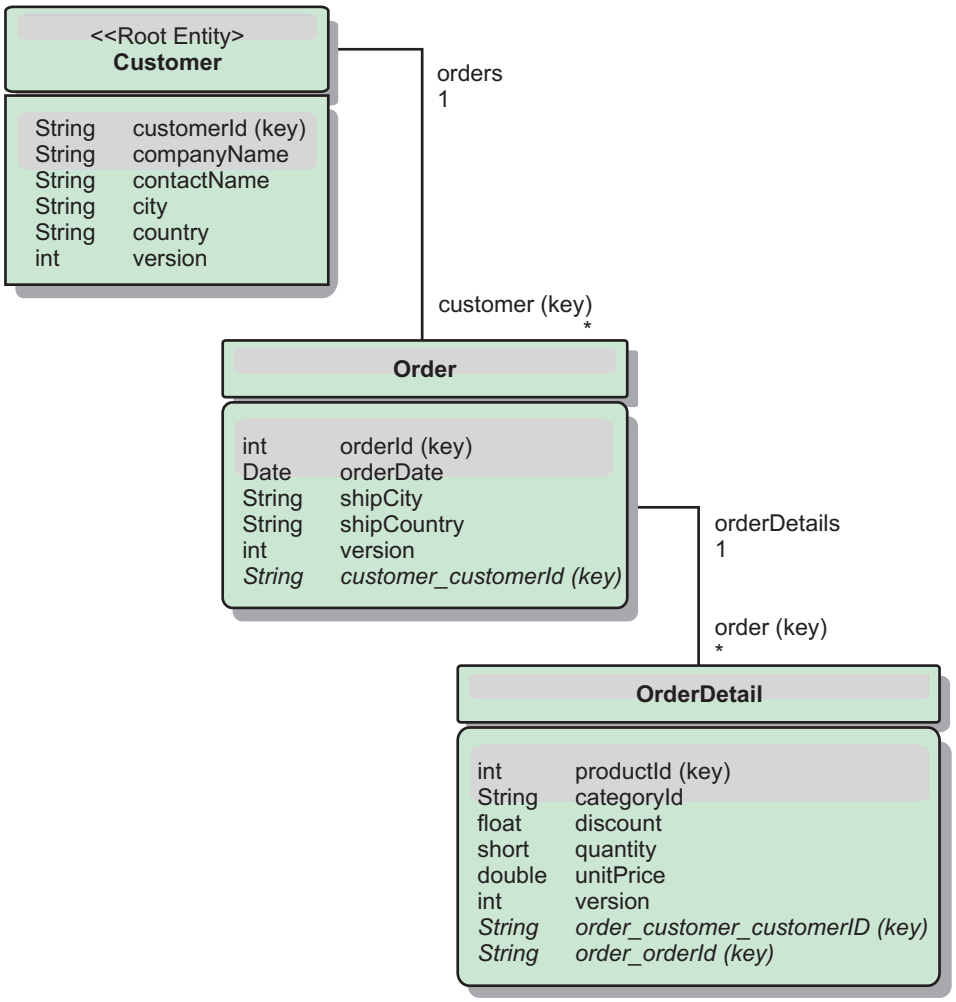


Abbildung 48. Entitätsschemadiagramm "Customer und Order"

Das Customer/Order-Entitätsschemadiagramm veranschaulicht, wie jede Entität unter Verwendung des Customer partitioniert wird. Die Entität "Order" enthält den Customer als Teil ihres Schlüssels und ist deshalb direkt adressierbar. Der REST-Datenservice stellt alle Schlüsselassoziationen als einzelne Eigenschaften bereit: Order hat customer_customerId, und OrderDetail hat order_customer_customerId und order_orderId.

Mit der API "EntityManager" können Sie die Order-Entität anhand der Customer- und Order-ID suchen:

```
transaction.begin();
// Order mit Customer suchen. Die ID wird erst dann in die Klasse
// Customer eingeschlossen, wenn die OrderId-Schlüsselinstanz erstellt wird.
Order order = (Order) em.find(Order.class,
    new OrderId(100, new Customer('ACME')));
...
transaction.commit();
```

Mit dem REST-Datenservice kann die Order-Entität über beide der folgenden URLs abgerufen werden:

- `/Order(orderId=100, customer_customerId='ACME')`
- `/Customer('ACME')/orders?$filter=orderId eq 100`

Auf den Kundenschlüssel wird über den Attributnamen der Customer-Entität, ein Unterstreichungszeichen und den Attributnamen der Customer-ID, customer_customerId, zugegriffen.

Eine Entität kann auch eine Entität, die keine Stammentität ist, in ihrem Schlüssel enthalten, wenn alle Vorgänger dieser Entität Assoziationen zur Stammentität haben. In diesem Beispiel hat OrderDetail eine Schlüsselassoziation zu Order und Order eine Schlüsselassoziation zur Customer-Stammentität. Verwendung der API "EntityManager":

```
transaction.begin();
// OrderDetailId-Schlüsselinstanz erstellen. Sie
// enthält Order und Customer, bei denen nur
// der Schlüssel definiert ist.
Customer customerACME = new Customer("ACME");
Order order100 = new Order(100, customerACME);
OrderDetailId orderDetailKey =
    new OrderDetailId(order100, "COMP");
OrderDetail orderDetail = (OrderDetail)
    em.find(OrderDetail.class, orderDetailKey);
...
```

Der REST-Datenservice lässt die direkte Adressierung von OrderDetail zu:

```
/OrderDetail(productId=500, order_customer_customerId='ACME', order_orderId =100)
```

Die Assoziation von der Entität "OrderDetail" zur Entität "Product" wurde unterbrochen, um die unabhängige Partitionierung von Order- und Product-Inventar zuzulassen. Die Entität "OrderDetail" speichert die Kategorie und die Produkt-ID anstelle einer festen Beziehung. Durch die Entkopplung der beiden Entitätsschemas wird jeweils nur auf eine einzige Partition zugegriffen.

Das im Diagramm dargestellte Category/Product-Schema zeigt, dass "Category" die Stammentität ist und dass jedes "Product" eine Assoziation zu einer Entität "Category" hat. Die Entität "Category" ist in der Product-ID enthalten. Der REST-Datenservice stellt eine Schlüsseleigenschaft bereit, category_categoryId, die über die das Product direkt adressiert werden kann.

Da Category die Stammentität ist, muss die Category in einer partitionierten Umgebung bekannt sein, damit das Product gefunden wird. Wenn die API "EntityManager" verwendet wird, muss die Transaktion vor der Suche des Product an die Entität "Category" gebunden werden.

Verwendung der API "EntityManager":

```
transaction.begin();
// Stammentität Category nur mit dem Schlüssel erstellen. Auf diese
// Weise kann eine ProductId erstellt werden, ohne zuerst die
// Category suchen zu müssen. Die Transaktion ist jetzt an die
// Partition gebunden, in der die Category "COMP" gespeichert ist.
Category cat = new Category("COMP");
Product product = (Product) em.find(Product.class,
    new ProductId(500, cat));
...
```

Der REST-Datenservice ermöglicht die direkte Adressierung des Product:

```
/Product(productId=500, category_categoryId='COMP')
```

Eigenständiges Grid für REST-Datenservices starten

Führen Sie die folgenden Schritte aus, um das Muster-Grid für den REST-Service von WebSphere eXtreme Scale für eine eigenständige Implementierung von eXtreme Scale zu starten.

Vorbereitende Schritte

Installieren Sie die Testversion von WebSphere eXtreme Scale oder das vollständige Produkt:

- Installieren Sie die eigenständige Version des Produkts WebSphere eXtreme Scale 7.1, und wenden Sie alle nachfolgenden Fixes an.
- Laden Sie die Testversion von WebSphere eXtreme Scale Version 7.1 herunter, und entpacken Sie diese. Diese Version enthält den REST-Datenservice von WebSphere eXtreme Scale.

Informationen zu diesem Vorgang

Starten Sie das Muster-Grid von WebSphere eXtreme Scale.

Vorgehensweise

1. Starten Sie den Katalogserviceprozess. Öffnen Sie eine Befehlszeile oder ein Terminalfenster, und setzen Sie die Umgebungsvariable JAVA_HOME:
   - Linux UNIX `export JAVA_HOME=`*`Java-Ausgangsverzeichnis`*
   - Windows `set JAVA_HOME=`*`Java-Ausgangsverzeichnis`*
2. `cd Ausgangsverzeichnis_des_REST-Service/gettingstarted`
3. Starten Sie den Katalogserviceprozess. Wenn Sie den Service *ohne* eXtreme-Scale-Sicherheit starten möchten, verwenden Sie die folgenden Befehle:
   - Linux UNIX `./runcat.sh`
   - Windows `runcat.bat`

   Wenn Sie den Service mit eXtreme-Scale-Sicherheit starten möchten, verwenden Sie die folgenden Befehle:
   - Linux UNIX `./runcat_secure.sh`
   - Windows `runcat_secure.bat`
4. Starten Sie zwei Containerserverprozesse. Öffnen Sie eine weitere Befehlszeile oder ein Terminalfenster, und setzen Sie die Umgebungsvariable JAVA_HOME:
   - Linux UNIX `export JAVA_HOME=`*`Java-Ausgangsverzeichnis`*
   - Windows `set JAVA_HOME=`*`Java-Ausgangsverzeichnis`*
5. `cd Ausgangsverzeichnis_des_REST-Service/gettingstarted`
6. Starten Sie einen Containerserverprozess:

   Wenn Sie den Server ohne eXtreme-Scale-Sicherheit starten möchten, verwenden Sie die folgenden Befehle:
   - Linux UNIX `./runcontainer.sh container0`
   - Windows `runcontainer.bat container0`

   Wenn Sie den Server mit eXtreme-Scale-Sicherheit starten möchten, verwenden Sie die folgenden Befehle:
   - Linux UNIX `./runcontainer_secure.sh container0`
   - Windows `runcontainer_secure.bat container0`

7. Öffnen Sie eine weitere Befehlszeile oder ein Terminalfenster, und setzen Sie die Umgebungsvariable JAVA_HOME:
   - Linux UNIX `export JAVA_HOME=Java-Ausgangsverzeichnis`
   - Windows `set JAVA_HOME=Java-Ausgangsverzeichnis`
8. `cd Ausgangsverzeichnis_des_REST-Service/gettingstarted`
9. Starten Sie einen zweiten Containerserverprozess.

   Wenn Sie den Server ohne eXtreme-Scale-Sicherheit starten möchten, verwenden Sie die folgenden Befehle.
   - Linux UNIX `./runcontainer.sh container1`
   - Windows `runcontainer.bat container1`

   Wenn Sie den Server mit eXtreme-Scale-Sicherheit starten möchten, verwenden Sie die folgenden Befehle:
   - Linux UNIX `./runcontainer_secure.sh container1`
   - Windows `runcontainer_secure.bat container1`

Ergebnisse

Warten Sie, bis die eXtreme-Scale-Container bereit sind, bevor Sie mit den nächsten Schritten fortfahren. Die Containerserver sind bereit, wenn die folgende Nachricht im Terminalfenster angezeigt wird:

```
CWOBJ1001I: Der ObjectGrid-Server Containername ist für die Verarbeitung
von Anforderungen bereit.
```

Containername steht für den Namen des gestarteten Containers.

Grid für REST-Datenservices in WebSphere Application Server starten

Führen Sie die folgenden Schritte aus, um ein eigenständiges Musterdaten-Grid für den REST-Service von WebSphere eXtreme Scale für eine in WebSphere Application Server integrierte Implementierung von WebSphere eXtreme Scale zu starten. Obwohl WebSphere eXtreme Scale mit WebSphere Application Server integriert ist, starten Sie mit diesen Schritten einen eigenständigen Katalogserviceprozess und -container von WebSphere eXtreme Scale.

Vorbereitende Schritte

Installieren Sie das Produkt WebSphere eXtreme Scale Version 7.1 in einem Installationsverzeichnis der WebSphere Application Server Version 7.0.0.5 oder höher (mit aktivierter Sicherheit), und erweitern Sie mindestens ein Application-Server-Profil.

Informationen zu diesem Vorgang

Starten Sie das Muster-Grid von WebSphere eXtreme Scale.

Vorgehensweise

1. Starten Sie den Katalogserviceprozess. Öffnen Sie eine Befehlszeile oder ein Terminalfenster, und setzen Sie die Umgebungsvariable JAVA_HOME:
   - Linux UNIX `export JAVA_HOME=Java-Ausgangsverzeichnis`
   - Windows `set JAVA_HOME=Java-Ausgangsverzeichnis`

```
cd Ausgangsverzeichnis_des_REST-Service/gettingstarted
```

2. Starten Sie den Katalogserviceprozess.

   Wenn Sie den Server ohne eXtreme-Scale-Sicherheit starten möchten, verwenden Sie die folgenden Befehle.

   - Linux UNIX `./runcat.sh`
   - Windows `runcat.bat`

   Wenn Sie den Server mit eXtreme-Scale-Sicherheit starten möchten, verwenden Sie die folgenden Befehle:

   - Linux UNIX `./runcat_secure.sh`
   - Windows `runcat_secure.bat`

3. Starten Sie zwei Containerserverprozesse. Öffnen Sie eine weitere Befehlszeile oder ein Terminalfenster, und setzen Sie die Umgebungsvariable JAVA_HOME:

   - Linux UNIX `export JAVA_HOME=`*`Java-Ausgangsverzeichnis`*
   - Windows `set JAVA_HOME=`*`Java-Ausgangsverzeichnis`*

4. Starten Sie einen Containerserverprozess.

   Wenn Sie den Server ohne eXtreme-Scale-Sicherheit starten möchten, verwenden Sie die folgenden Befehle.

   a. Öffnen Sie ein Befehlszeilenfenster.
   b. `cd Ausgangsverzeichnis_des_REST-Service/gettingstarted`
   c. Zum Starten des Servers *ohne* eXtreme-Scale-Sicherheit verwenden Sie die folgenden Befehle:
      - Linux UNIX `./runcontainer.sh container0`
      - Windows `runcontainer.bat container0`
   d. Wenn Sie den Server mit eXtreme-Scale-Sicherheit starten möchten, verwenden Sie die folgenden Befehle:
      - Linux UNIX `./runcontainer_secure.sh container0`
      - Windows `runcontainer_secure.bat container0`

5. Starten Sie einen zweiten Containerserverprozess.

   a. Öffnen Sie ein Befehlszeilenfenster.
   b. `cd Ausgangsverzeichnis_des_REST-Service/gettingstarted`
   c. Zum Starten des Servers *ohne* eXtreme-Scale-Sicherheit verwenden Sie die folgenden Befehle:
      - Linux UNIX `./runcontainer.sh container1`
      - Windows `runcontainer.bat container1`
   d. Zum Starten des Servers *mit* eXtreme-Scale-Sicherheit verwenden Sie die folgenden Befehle:
      - Linux UNIX `./runcontainer_secure.sh container1`
      - Windows `runcontainer_secure.bat container1`

Ergebnisse

Warten Sie, bis die Containerserver bereit sind, bevor Sie mit den nächsten Schritten fortfahren. Die Containerserver sind bereit, wenn die folgende Nachricht angezeigt wird:

CWOBJ1001I: Der ObjectGrid-Server *Containername* ist für die Verarbeitung von Anforderungen bereit.

Containername steht für den Namen des im vorherigen Schritt gestarteten Containers.

Anwendungsserver für den REST-Datenservice konfigurieren

REST-Datenservices in WebSphere Application Server Version 7.0 starten

In diesem Abschnitt wird beschrieben, wie der REST-Datenservice von eXtreme Scale mit WebSphere Application Server Version 7.0 gestartet wird.

Vorbereitende Schritte

Vergewissern Sie sich, dass das eXtreme-Scale-Muster-Grid gestartet ist. Einzelheiten zum Starten des Grids finden Sie unter „REST-Datenservice aktivieren" auf Seite 227.

Vorgehensweise

1. Laden Sie WebSphere Application Server Version 7.0 for Developers herunter, und installieren Sie das Produkt.

   Einschränkung: Aktivieren Sie die Sicherheit nicht.
2. Laden Sie Fixpack 5 oder höher für WebSphere Application Server Version 7.0 herunter, und installieren Sie es.
3. Fügen Sie die JAR-Datei für die eXtreme-Scale-Clientlaufzeitumgebung, die Datei `wsogclient.jar` und die Konfigurations-JAR-Datei bzw. das Konfigurationsverzeichnis für den REST-Datenservice dem Klassenpfad des Anwendungsservers hinzu:
   a. Öffnen Sie die Administrationskonsole von WebSphere Application Server.
   b. Navigieren Sie zu **Umgebung** → **Gemeinsam genutzte Bibliotheken**.
   c. Klicken Sie auf **Neu**.
   d. Fügen Sie den Feldern die folgenden Einträge hinzu:
      1) Name: extremescale_client_v71
      2) Klassenpfad: `WXS-Ausgangsverzeichnis/lib/wsogclient.jar`
   e. Klicken Sie auf **OK**.
   f. Klicken Sie auf **Neu**.
   g. Fügen Sie den entsprechenden Feldern die folgenden Einträge hinzu:
      1) Name: `extremescale_gettingstarted_config`
      2) Klassenpfad:
         - `Ausgangsverzeichnis_des_REST-Service/gettingstarted/restclient/bin`
         - `Ausgangsverzeichnis_des_REST-Service//gettingstarted/common/bin`

      Hinweis: Fügen Sie jeden Pfad in einer gesonderten Zeile hinzu.
   h. Klicken Sie auf **OK**.
   i. Speichern Sie die Änderungen in der Masterkonfiguration.
4. Installieren Sie die EAR-Datei des REST-Datenservice `wxsrestservice.ear` über die Administrationskonsole in WebSphere Application Server:
   a. Öffnen Sie die Administrationskonsole von WebSphere Application Server.

b. Navigieren Sie zu **Anwendungen → Neue Anwendung**.
c. Navigieren Sie zur Datei `Ausgangsverzeichnis_des_REST-Service/lib/wxsrestservice.ear`, wählen Sie die Datei aus, und klicken Sie auf **Weiter**.
d. Wählen Sie die detaillierten Installationsoptionen aus, und klicken Sie auf **Weiter**.
e. Klicken Sie in der Anzeige mit Anwendungssicherheitswarnungen auf **Weiter**.
f. Wählen Sie die Standardinstallationsoptionen aus, und klicken Sie auf **Weiter**.
g. Wählen Sie einen Server aus, dem Sie die Anwendung zuordnen möchten, und klicken Sie auf **Weiter**.
h. Verwenden Sie auf der Seite für das erneute Laden von JSP-Dateien die Standardeinstellungen, und klicken Sie auf **Weiter**.
i. Ordnen Sie auf der Seite "Gemeinsam genutzte Bibliotheken" das Modul `wxsrestservice.war` den folgenden definierten gemeinsam genutzten Bibliotheken zu:
   - extremescale_client_v71
   - extremescale_ gettingstarted _config
j. Verwenden Sie auf der Seite für die Zuordnung von Beziehungen zu gemeinsam genutzten Bibliotheken die Standardeinstellungen, und klicken Sie auf **Weiter**.
k. Verwenden Sie auf der Seite für die Zuordnung virtueller Hosts die Standardeinstellungen, und klicken Sie auf **Weiter**.
l. Setzen Sie auf der Seite für die Zuordnung der Kontextstammelemente das Kontextstammelement auf `wxsrestservice`.
m. Klicken Sie in der Anzeige "Zusammenfassung" auf **Fertig stellen**, um die Installation durchzuführen.
n. Speichern Sie die Änderungen in der Masterkonfiguration.

5. Starten Sie den Anwendungsserver und die REST-Datenserviceanwendung "wxsrestservice" von eXtreme Scale. Suchen Sie nach dem Start der Anwendung in der Protokolldatei `SystemOut.log` des Anwendungsservers die folgende Nachricht: `CWOBJ4000I: Der REST-Datenservice von WebSphere eXtreme Scale wurde gestartet.`
6. Vergewissern Sie sich, dass der REST-Datenservice funktioniert.
   a. Öffnen Sie einen Browser, und navigieren Sie zu `http://localhost:9080/wxsrestservice/restservice/NorthwindGrid`. Das Servicedokument für das Grid NorthwindGrid wird angezeigt.
   b. Navigieren Sie zu `http://localhost:9080/wxsrestservice/restservice/NorthwindGrid/$metadata`. Das EDMX-Dokument (Entity Model Data Extensions) wird angezeigt.
7. Zum Stoppen der Grid-Prozesse verwenden Sie die Tastenkombination `STRG+C` im entsprechenden Befehlsfenster.

REST-Datenservices mit WebSphere eXtreme Scale integrated in WebSphere Application Server 7.0 starten

In diesem Abschnitt wird beschrieben, wie der REST-Datenservice von eXtreme Scale mit dem Produkt WebSphere Application Server Version 7.0, das mit WebSphere eXtreme Scale integriert und erweitert wurde, konfiguriert und gestartet wird.

Vorbereitende Schritte

Vergewissern Sie sich, dass das eigenständige eXtreme-Scale-Muster-Grid gestartet ist. Einzelheiten zum Starten des Grids finden Sie unter „REST-Datenservice aktivieren” auf Seite 227.

Informationen zu diesem Vorgang

Führen Sie zum Einstieg in die Verwendung des REST-Datenservice von WebSphere eXtreme Scale mit WebSphere Application Server die folgenden Schritte aus:

Vorgehensweise

1. Fügen Sie die Konfigurations-JAR-Datei des Musters für den REST-Datenservice von WebSphere eXtreme Scale dem Klassenpfad hinzu:
   a. Öffnen Sie die WebSphere-Administrationskonsole.
   b. Navigieren Sie zu "Umgebung -> Gemeinsam genutzte Bibliotheken".
   c. Klicken Sie auf "Neu".
   d. Fügen Sie den entsprechenden Feldern die folgenden Einträge hinzu:
      1) Name: extremescale_gettingstarted_config
      2) Klassenpfad
         - `Ausgangsverzeichnis_des_REST-Service/gettingstarted/restclient/bin`
         - `Ausgangsverzeichnis_des_REST-Service//gettingstarted/common/bin`

         Hinweis: Jeder Pfad muss in einer gesonderten Zeile angegeben werden.
   e. Klicken Sie auf **OK**.
   f. Speichern Sie die Änderungen in der Masterkonfiguration.
2. Installieren Sie die EAR-Datei des REST-Datenservice "wxsrestservice.ear" über die WebSphere-Administrationskonsole in WebSphere Application Server:
   a. Öffnen Sie die WebSphere-Administrationskonsole.
   b. Navigieren Sie zu "Anwendungen -> Neue Anwendung".
   c. Navigieren Sie zur Datei `Ausgangsverzeichnis_des_REST-Service/lib/wxsrestservice.ear` im Dateisystem. Wählen Sie die Datei aus, und klicken Sie auf **Weiter**.
   d. Wählen Sie die detaillierten Installationsoptionen aus, und klicken Sie auf **Weiter**.
   e. Klicken Sie in der Anzeige mit Anwendungssicherheitswarnungen auf **Weiter**.
   f. Wählen Sie die Standardinstallationsoptionen aus, und klicken Sie auf **Weiter**.
   g. Wählen Sie einen Server aus, dem Sie das Modul "wxsrestservice.war" zuordnen möchten, und klicken Sie auf **Weiter**.
   h. Verwenden Sie auf der Seite für das erneute Laden von JSP-Dateien die Standardeinstellungen, und klicken Sie auf **Weiter**.
   i. Ordnen Sie auf der Seite "Gemeinsam genutzte Bibliotheken" das Modul "wxsrestservice.war" den folgenden gemeinsam genutzten Bibliotheken zu, die in Schritt 1 definiert wurden: `extremescale_ gettingstarted _config`.

j. Verwenden Sie auf der Seite für die Zuordnung von Beziehungen zu gemeinsam genutzten Bibliotheken die Standardeinstellungen, und klicken Sie auf **Weiter**.

k. Verwenden Sie auf der Seite für die Zuordnung virtueller Hosts die Standardeinstellungen, und klicken Sie auf **Weiter**.

l. Setzen Sie auf der Seite für die Zuordnung der Kontextstammelemente das Kontextstammelement auf "wxsrestservice".

m. Klicken Sie in der Anzeige "Zusammenfassung" auf **Fertig stellen**, um die Installation durchzuführen.

n. Speichern Sie die Änderungen in der Masterkonfiguration.

3. Wenn das eXtreme-Scale-Grid mit aktivierter eXtreme-Scale-Sicherheit gestartet wurde, setzen Sie die folgende Eigenschaft in der Datei `Ausgangsverzeichnis_des_REST-Service/gettingstarted/restclient/bin/wxsRestService.properties`.

```
ogClientPropertyFile=Ausgangsverzeichnis_des_REST-Service/gettingstarted/security/security.ogclient.properties
```

4. Starten Sie den Anwendungsserver und die REST-Datenserviceanwendung "wxsrestservice" von eXtreme Scale.

   Suchen Sie nach dem Start der Anwendung in der Protokolldatei "SystemOut.log" des Anwendungsservers die folgende Nachricht: `CWOBJ4000I: Der REST-Datenservice von WebSphere eXtreme Scale wurde gestartet.`

5. Vergewissern Sie sich, dass der REST-Datenservice funktioniert:

   a. Öffnen Sie einen Browser, und navigieren Sie zur folgenden Adresse:

      http://localhost:9080/wxsrestservice/restservice/NorthwindGrid

      Das Servicedokument für das Grid NorthwindGrid wird angezeigt.

   b. Navigieren Sie zur folgenden Adresse:

      http://localhost:9080/wxsrestservice/restservice/NorthwindGrid/$metadata

      Das EDMX-Dokument (Entity Model Data Extensions) wird angezeigt.

6. Zum Stoppen der Grid-Prozesse verwenden Sie die Tastenkombination STRG+C im entsprechenden Befehlsfenster.

REST-Datenservice in WebSphere Application Server Community Edition starten

In diesem Abschnitt wird beschrieben, wie der REST-Datenservice von eXtreme Scale mit WebSphere Application Server Community Edition konfiguriert und gestartet wird.

Vorbereitende Schritte

Vergewissern Sie sich, dass das Musterdaten-Grid gestartet ist. Einzelheiten zum Starten des Grids finden Sie unter „REST-Datenservice aktivieren” auf Seite 227.

Vorgehensweise

1. Laden Sie WebSphere Application Server Community Edition Version 2.1.1.3 oder höher herunter, und installieren Sie das Produkt im `WASCE-Stammverzeichnis`, z. B. `/opt/IBM/wasce`.
2. Starten Sie den Server von WebSphere Application Server Community Edition mit dem folgenden Befehl:
   - Linux UNIX `WASCE-Stammverzeichnis/bin/startup.sh`
   - Windows `WASCE-Stammverzeichnis/bin/startup.bat`

3. Wenn das eXtreme-Scale-Grid mit aktivierter eXtreme-Scale-Sicherheit gestartet wurde, setzen Sie die folgenden Eigenschaften in der Datei `Ausgangsverzeichnis_des_REST-Service/gettingstarted/restclient/bin/wxsRestService.properties`.

```
ogClientPropertyFile=Ausgangsverzeichnis_des_REST-Service/gettingstarted/security/security.ogclient.properties
loginType=none
```

4. Installieren Sie den REST-Datenservice von eXtreme Scale und das bereitgestellte Muster im Server von WebSphere Application Server Community Edition:
   a. Fügen Sie die JAR-Datei für die ObjectGrid-Clientlaufzeitumgebung dem Repository von WebSphere Application Server Community Edition hinzu:
      1) Öffnen Sie die Administrationskonsole von WebSphere Application Server Community Edition, und melden Sie sich an.

         Tipp: Der Standard-URL ist `http://localhost:8080/console`. Die Standardbenutzer-ID ist `system`, und das Kennwort ist `manager`.
      2) Klicken Sie im Ordner "Services" auf **Repository**.
      3) Tragen Sie im Abschnitt **Archiv dem Repository hinzufügen** Folgendes in die Eingabetextfelder ein:

Tabelle 16. Archivierung im Repository

Textfeld	Wert
Datei	WXS-Ausgangsverzeichnis/lib/ogclient.jar
Gruppe	com.ibm.websphere.xs
Artefakt	ogclient
Version	7.0
Typ	jar

      4) Klicken Sie auf die Schaltfläche "Installieren".

         Tipp: Suchen Sie im folgenden technischen Hinweis nach Einzelheiten zu den verschiedenen Methoden von Konfigurationsklassen- und Bibliotheksabhängigkeiten: Specifying external dependencies to applications running on WebSphere Application Server Community Edition.
   b. Implementieren Sie das Modul des REST-Datenservice, das in der Datei `wxsrestservice.war` enthalten ist, im Server von WebSphere Application Server Community Edition.
      1) Bearbeiten Sie die XML-Musterimplementierungsdatei `Ausgangsverzeichnis_des_REST-Service/gettingstarted/wasce/geronimo-web.xml`, und fügen Sie den Klassenpfadverzeichnissen für das Einführungsmuster Pfadabhängigkeiten hinzu.

         Ändern Sie die classesDirs-Pfade für die beiden GBeans des GettingStarted-Clients:
         - Der classesDirs-Pfad für die GBean "GettingStarted_Client_SharedLib" muss auf `Ausgangsverzeichnis_des_REST-Service/Ggettingstarted/restclient/bin` gesetzt werden.
         - Der classesDirs-Pfad für die GBean "GettingStarted_Common_SharedLib" muss auf `Ausgangsverzeichnis_des_REST-Service/gettingstarted/common/bin` gesetzt werden.
      2) Öffnen Sie die Administrationskonsole von WebSphere Application Server Community Edition, und melden Sie sich an.

Tipp: Der Standard-URL ist `http://localhost:8080/console`. Die Standardbenutzer-ID ist `system`, und das Kennwort ist `manager`.

3) Klicken Sie auf **Neu implementieren**.
4) Geben Sie auf der Seite **Neue Anwendungen installieren** die folgenden Werte in die Textfelder ein:

Tabelle 17. Installationswerte

Textfeld	Wert
Archiv	`Ausgangsverzeichnis_des_REST-Service/lib/wxsrestservice.war`
Plan	`Ausgangsverzeichnis_des_REST-Service/gettingstarted/wasce/geronimo-web.xml`

5) Klicken Sie auf die Schaltfläche "Installieren".

   Auf der Konsolseite sollte angezeigt werden, dass die Anwendung erfolgreich installiert und gestartet werden.
6) Überprüfen Sie anhand des Systemausgabeprotokolls von WebSphere Application Server Community Edition oder der Konsole, ob der REST-Datenservice erfolgreich gestartet wurde, indem Sie nach der folgenden Nachricht suchen:

   ```
   CWOBJ4000I: Der REST-Datenservice von WebSphere eXtreme Scale
   wurde gestartet.
   ```

5. Vergewissern Sie sich, dass der REST-Datenservice funktioniert:
   a. Öffnen Sie den folgenden Link in einem Browserfenster: `http://localhost:8080/wxsrestservice/restservice/NorthwindGrid`. Das Servicedokument für das Grid "NorthwindGrid" wird angezeigt.
   b. Öffnen Sie den folgenden Link in einem Browserfenster: `http://localhost:8080/wxsrestservice/restservice/NorthwindGrid/$metadata`. Das EDMX-Dokument (Entity Model Data Extensions) wird angezeigt.
6. Zum Stoppen der Grid-Prozesse verwenden Sie die Tastenkombination `STRG+C` im entsprechenden Befehlsfenster.
7. Verwenden Sie zum Stoppen von WebSphere Application Server Community Edition den folgenden Befehl:
   - UNIX Linux `WASCE-Stammverzeichnis/bin/shutdown.sh`
   - Windows `WASCE-Stammverzeichnis\bin\shutdown.bat`

   Tipp: Die Standardbenutzer-ID ist `system`, und das Kennwort ist `manager`. Wenn Sie einen angepassten Port verwenden, verwenden Sie die Option `-port`.

REST-Datenservices in Apache Tomcat starten

In diesem Abschnitt wird beschrieben, wie der REST-Datenservice von eXtreme Scale mit Apache Tomcat Version 5.5 oder höher konfiguriert und gestartet wird.

Vorbereitende Schritte

Vergewissern Sie sich, dass das eXtreme-Scale-Muster-Grid gestartet ist. Einzelheiten zum Starten des Grids finden Sie unter „REST-Datenservice aktivieren” auf Seite 227.

Vorgehensweise

1. Laden Sie Apache Tomcat Version 5.5 oder höher in das Tomcat-Installationsverzeichnis herunter, und installieren Sie diese Version. Beispiel: `/opt/tomcat`

2. Installieren Sie den REST-Datenservice von eXtreme Scale und das bereitgestellte Muster wie folgt im Tomcat-Server:
   a. Wenn Sie eine Sun JRE oder ein Sun JDK verwenden, müssen Sie den IBM ORB in Tomcat installieren:
      - Für Tomcat Version 5.5

        Kopieren Sie alle JAR-Dateien von

        `WXS-Ausgangsverzeichnis/lib/endorsed`

        nach

        `Tomcat-Stammverzeichnis/common/endorsed`
      - Für Tomcat Version 6.0
        1) Erstellen Sie ein Verzeichnis "endorsed".
           - UNIX Linux `mkdir Tomcat-Stammverzeichnis/endorsed`
           - Windows `md Tomcat-Stammverzeichnis/endorsed`
        2) Kopieren Sie alle JAR-Dateien von

           `WXS-Ausgangsverzeichnis/lib/endorsed`

           nach

           `Tomcat-Stammverzeichnis/endorsed`
   b. Implementieren Sie das REST-Datenservicemodul "wxsrestservice.war" im Tomcat-Server.

      Kopieren Sie die Datei "wxsrestservice.war" von

      `Ausgangsverzeichnis_des_REST-Service/lib`

      nach

      `Tomcat-Stammverzeichnis/webapps`
   c. Fügen Sie die JAR-Datei für die ObjectGrid-Clientlaufzeitumgebung und die Anwendungs-JAR-Datei dem gemeinsam genutzten Klassenpfad in Tomcat hinzu:
      1) Bearbeiten Sie die Datei `Tomcat-Stammverzeichnis/conf/catalina.properties`.
      2) Fügen Sie die folgenden Pfadnamen am Ende der Eigenschaft "shared.loader" in Form einer durch Kommas begrenzten Liste hinzu:
         - WXS-Ausgangsverzeichnis/lib/ogclient.jar
         - Ausgangsverzeichnis_des_REST-Service/gettingstarted/restclient/bin
         - Ausgangsverzeichnis_des_REST-Service/gettingstarted/common/bin

         Wichtig: Das Pfadtrennzeichen muss ein **Schrägstrich** sein.
3. Wenn das eXtreme-Scale-Grid mit aktivierter eXtreme-Scale-Sicherheit gestartet wurde, setzen Sie die folgenden Eigenschaften in der Datei `Ausgangsverzeichnis_des_REST-Service/gettingstarted/restclient/bin/wxsRestService.properties`.

```
ogClientPropertyFile=Ausgangsverzeichnis_des_REST-Service/gettingstarted/security/security.ogclient.properties
loginType=none
```

4. Starten Sie den Tomcat-Server mit dem REST-Datenservice:
   - Wenn Sie Tomcat 5.5 unter UNIX® or Windows® oder Tomcat 6.0 unter UNIX verwenden, gehen Sie wie folgt vor:
     a. `cd Tomcat-Stammverzeichnis/bin`
     b. Starten Sie den Server:
        - UNIX Linux `./catalina.sh run`

– Windows `catalina.bat run`

c. In der Konsole werden daraufhin die Apache-Tomcat-Protokolle angezeigt. Wenn der REST-Datenservice erfolgreich gestartet wurde, wird die folgende Nachricht in der Administrationskonsole angezeigt:

`CWOBJ4000I: Der REST-Datenservice von WebSphere eXtreme Scale wurde gestartet.`

- Wenn Sie Tomcat 6.0 unter Windows verwenden, gehen Sie wie folgt vor:
  a. `cd Tomcat-Stammverzeichnis/bin`
  b. Starten Sie das Konfigurationstool von Apache Tomcat 6 mit dem folgenden Befehl: `tomcat6w.exe`
  c. Klicken Sie im Eigenschaftsfenster von Apache Tomcat 6 auf die Startschaltfläche, um den Tomcat-Server zu starten.
  d. Sehen Sie sich die folgenden Protokolle an, um sich zu vergewissern, dass der Tomcat-Server erfolgreich gestartet wurde:
     – `Tomcat-Stammverzeichnis/bin/catalina.log`

       Zeigt den Status der Tomcat-Server-Engine an.
     – `Tomcat-Stammverzeichnis/bin/stdout.log`

       Zeigt das Systemausgabeprotokoll an.
  e. Wenn der REST-Datenservice erfolgreich gestartet wurde, wird die folgende Nachricht im Systemausgabeprotokoll angezeigt: `CWOBJ4000I: Der REST-Service von WebSphere eXtreme Scale wurde gestartet.`

5. Vergewissern Sie sich, dass der REST-Datenservice funktioniert:
   a. Öffnen Sie einen Browser, und navigieren Sie zur folgenden Adresse:

      http://localhost:8080/wxsrestservice/restservice/NorthwindGrid

      Das Servicedokument für das Grid NorthwindGrid wird angezeigt.
   b. Navigieren Sie zur folgenden Adresse:

      http://localhost:8080/wxsrestservice/restservice/NorthwindGrid/$metadata

      Das EDMX-Dokument (Entity Model Data Extensions) wird angezeigt.
6. Zum Stoppen der Grid-Prozesse verwenden Sie die Tastenkombination STRG+C im entsprechenden Befehlsfenster.
7. Zum Stoppen von Tomcat verwenden Sie die Tastenkombination STRG+C in dem Fenster, in dem Sie Tomcat gestartet haben.

Browser mit REST-Datenservices verwenden

Der REST-Datenservice von eXtreme Scale erstellt bei der Verwendung eines Webbrowsers standardmäßig ATOM-Feeds. Das ATOM-Feed-Format ist möglicherweise mit älteren Browsern nicht kompatibel, oder es kann so interpretiert werden, dass die Daten nicht als XML angezeigt werden können. Die folgenden Abschnitte enthalten Details zur Konfiguration von Internet Explorer Version 8 und Firefox Version 3 für die Anzeige von ATOM-Feeds und XML im Browser.

Informationen zu diesem Vorgang

Der REST-Datenservice von eXtreme Scale erstellt bei der Verwendung eines Webbrowsers standardmäßig ATOM-Feeds. Das ATOM-Feed-Format ist möglicherweise mit älteren Browsern nicht kompatibel, oder es kann so interpretiert werden, dass die Daten nicht als XML angezeigt werden können. Bei älteren Browsern werden Sie aufgefordert, die Dateien auf der Platte zu speichern. Verwenden Sie nach dem Download der Dateien den von Ihnen bevorzugten XML-Reader, um die Dateien

anzuzeigen. Die generierte XML wird für die Anzeige nicht formatiert, und deshalb wird alles in einer einzigen Zeile ausgegeben. Die meisten Programme zum Lesen von XML, wie z. B. Eclipse, unterstützten eine Neuformatierung der XML in ein lesbares Format.

Bei modernen Browsern wie Microsoft Internet Explorer Version 8 und Firefox Version 3 können die ATOM-XML-Dateien nativ im Browser angezeigt werden. Die folgenden Abschnitte enthalten Details zur Konfiguration von Internet Explorer Version 8 und Firefox Version 3 für die Anzeige von ATOM-Feeds und XML im Browser.

Vorgehensweise

Internet Explorer Version 8 konfigurieren

- Verwenden Sie die folgenden Schritte, um Internet Explorer für das Lesen der vom REST-Datenservice generierten ATOM-Feeds zu aktivieren:
  1. Klicken Sie auf **Extras** → **Internetoptionen**-
  2. Wählen Sie das Register **Inhalte** aus.
  3. Klicken Sie im Abschnitt **Feeds und Web Slices** auf die Schaltfläche **Einstellungen**.
  4. Wählen Sie das Feld "Feedleseanzeige einschalten" ab.
  5. Klicken Sie auf **OK**, um zum Browser zurückzukehren.
  6. Starten Sie Internet Explorer erneut.

Firefox Version 3 konfigurieren

- Firefox zeigt Seiten mit dem Inhaltstyp "application/atom+xml" nicht automatisch an. Wenn eine solche Seite zum ersten Mal angezeigt wird, fordert Firefox Sie zum Speichern der Datei auf. Zum Anzeigen der Seite öffnen Sie die Datei in Firefox wie folgt:
  1. Wählen Sie im Dialogfenster für die Anwendungsauswahl das Optionsfeld "Öffnen mit" aus, und klicken Sie auf die Schaltfläche **Durchsuchen**.
  2. Navigieren Sie zum Firefox-Installationsverzeichnis. Beispiel: `C:\Program Files\Mozilla Firefox`
  3. Wählen Sie `firefox.exe` aus, und klicken Sie auf die Schaltfläche **OK**.
  4. Wählen Sie das Markierungsfeld "Für Dateien dieses Typs immer diese Aktion ausführen" aus.
  5. Klicken Sie auf die Schaltfläche **OK**.
  6. Anschließend zeigt Firefox die ATOM-XML-Seite in einem neuen Browserfenster oder auf einer neuen Registerkarte an.
- Firefox gibt ATOM-Feeds automatisch in einem lesbaren Format wieder. Die Feeds, die vom REST-Datenservice erstellt werden, enthalten jedoch XML. Firefox kann die XML nur anzeigen, wenn Sie den Feed-Renderer inaktivieren. Anders als in Internet Explorer muss in Firefox das Plug-in für die Wiedergabe von ATOM-Feeds explizit geändert werden. Zum Konfigurieren von Firefox für das Lesen von ATOM-Feeds als XML-Dateien gehen Sie wie folgt vor:
  1. Öffnen Sie die folgende Datei in einem Texteditor: `<Firefox-Installationsstammverzeichnis>\components\FeedConverter.js`. In dem Pfad steht `<Firefox-Installationsstammverzeichnis>` für das Stammverzeichnis, in dem Firefox installiert ist.

     Bei Windows-Betriebssystemen ist das Standardverzeichnis `C:\Program Files\Mozilla Firefox`.
  2. Suchen Sie das Snippet, das wie folgt aussieht:

```
// show the feed page if it wasn't sniffed and we have a document,
// or we have a document, title, and link or id
if (result.doc && (!this._sniffed ||
    (result.doc.title && (result.doc.link || result.doc.id)))) {
```

3. Setzen Sie die beiden Zeilen, die mit `if` und `result` beginnen, auf Kommentar, indem Sie `//` (zwei Schrägstriche) an den Anfang der Zeilen setzen.
4. Fügen Sie dem Snippet die folgende Anweisung an: `if(0) {`.
5. Der Text sollte anschließend wie folgt aussehen:

```
// show the feed page if it wasn't sniffed and we have a document,
// or we have a document, title, and link or id
//if (result.doc && (!this._sniffed ||
//    (result.doc.title && (result.doc.link || result.doc.id)))) {
if(0) {
```

6. Speichern Sie die Datei.
7. Starten Sie Firefox erneut.
8. Jetzt kann Firefox automatisch alle Feeds im Browser anzeigen.

- Testen Sie Ihr Setup, indem Sie verschiedene URLs ausprobieren.

Beispiel

In diesem Abschnitt werden einige Beispiel-URLs beschrieben, die Sie verwenden können, um die Daten anzuzeigen, die von dem Einführungsmuster hinzugefügt wurden, das mit dem REST-Datenservice von eXtreme Scale bereitgestellt wird. Fügen Sie vor der Verwendung der folgenden URLs mit dem Java-Musterclient oder dem Musterclient für Visual Studio WCF Data Services den Standarddatensatz zum eXtreme-Scale-Muster-Grid hinzu.

In den folgenden Beispielen wird der Port 8080 angenommen, der aber variieren kann. Lesen Sie den Abschnitt zum Konfigurieren des REST-Datenservice in verschiedenen Anwendungsservern.

- Einen einzigen Kunden (customer) mit der ID "ACME" anzeigen:

  `http://localhost:8080/wxsrestservice/restservice/NorthwindGrid/Customer('ACME')`
- Alle Bestellungen (order) für den Kunden "ACME" anzeigen:

  `http://localhost:8080/wxsrestservice/restservice/NorthwindGrid/Customer('ACME')/orders`
- Kunden "ACME" und die zugehörigen Bestellungen anzeigen:

  `http://localhost:8080/wxsrestservice/restservice/NorthwindGrid/Customer('ACME')?$expand=orders`
- Bestellung 1000 für den Kunden "ACME" anzeigen:

  `http://localhost:8080/wxsrestservice/restservice/NorthwindGrid/Order(orderId=1000,customer_customerId='ACME')`
- Bestellung 1000 für den Kunden "ACME" und den zugeordnten Kunden anzeigen:

  `http://localhost:8080/wxsrestservice/restservice/NorthwindGrid/Order(orderId=1000,customer_customerId='ACME')?$expand=customer`
- Bestellung 1000 für den Kunden "ACME" sowie den zughörigen Kunden und Bestelldetails (OrderDetails) anzeigen:

  `http://localhost:8080/wxsrestservice/restservice/NorthwindGrid/Order(orderId=1000,customer_customerId='ACME')?$expand=customer,orderD`
- Alle Bestellungen für den Kunden "ACME" für den Monat Oktober 2009 (GMT) anzeigen:

```
http://localhost:8080/wxsrestservice/restservice/NorthwindGrid/Customer(customerId='ACME')/orde
ge datetime'2009-10-01T00:00:00'
and orderDate lt datetime'2009-11-01T00:00:00'
```

- Die ersten drei Bestellungen und die zugehörigen Bestelldetails für den Kunden "ACME" für den Monat Oktober 2009 (GMT) anzeigen:

```
http://localhost:8080/wxsrestservice/restservice/NorthwindGrid/Customer(customerId='ACME')/orders?
ge datetime'2009-10-01T00:00:00'
and orderDate lt datetime'2009-11-01T00:00:00'
&$orderby=orderDate&$top=3&$expand=orderDetails
```

Java-Client mit REST-Datenservices verwenden

Die Java-Clientanwendung verwendet die eXtreme-Scale-API "EntityManager", um Daten in das Grid einzufügen.

Informationen zu diesem Vorgang

In den vorherigen Abschnitten wurde beschrieben, wie ein eXtreme-Scale-Grid erstellt und mit dem REST-Datenservice von eXtreme Scale konfiguriert und gestartet wird. Die Java-Clientanwendung verwendet die eXtreme-Scale-API "EntityManager", um Daten in das Grid einzufügen. Die demonstriert nicht, wie die REST-Schnittstellen verwendet werden. Dieser Client soll veranschaulichen, wie die API "EntityManager" verwendet wird, um mit dem eXtreme-Scale-Grid zu interagieren und somit Daten im Grid zu ändern. Zum Anzeigen der Daten im Grid über den REST-Datenservice verwenden Sie einen Webbrowser oder Clientanwendung von Visual Studio 2008.

Vorgehensweise

Führen Sie den folgenden Befehl aus, um dem eXtreme-Scale-Grid schnell Inhalt hinzuzufügen:

1. Öffnen Sie eine Befehlszeile oder ein Terminalfenster, und setzen Sie die Umgebungsvariable JAVA_HOME:
   - Linux UNIX `export JAVA_HOME=`*`Java-Ausgangsverzeichnis`*
   - Windows `set JAVA_HOME=`*`Java-Ausgangsverzeichnis`*
2. `cd Ausgangsverzeichnis_des_REST-Service/gettingstarted`
3. Fügen Sie einige Daten in das Grid ein. Die eingefügten Daten werden später mit einem Webbrowser und dem REST-Datenservice abgerufen.

   Wenn das Grid *ohne* eXtreme-Scale-Sicherheit gestartet wurde, verwenden Sie die folgenden Befehle:

   - UNIX Linux `./runclient.sh load default`
   - Windows `runclient.bat load default`

   Wenn das Grid *mit* eXtreme-Scale-Sicherheit gestartet wurde, verwenden Sie die folgenden Befehle:

   - UNIX Linux `./runclient_secure.sh load default`
   - Windows `runclient_secure.bat load default`

   Für einen Java-Client verwenden Sie die folgende Befehlssyntax:

   - UNIX Linux `runclient.sh` *`Befehl`*
   - Windows `runclient.bat` *`Befehl`*

   Die folgenden Befehle sind verfügbar:

   - `load default`

     Lädt einen vordefinierten Satz von Customer-, Category- und Product-Entitäten in das Grid und erstellt einen zufälligen Satz von Bestellungen (Order) für jeden Kunden (Customer).

- `load category` *`Kategorie-ID Kategoriename erste_Produkt-ID Anzahl_Produkte`*

  Erstellt eine Produktkategorie und eine festgelegte Anzahl an Product-Entitäten im Grid. Der Parameter "erste_Produkt-ID" gibt die ID-Nummer des ersten Produkts an, und jedem nachfolgenden Produkt wird die jeweils nächste ID zugeordnet, bis die angegebene Anzahl an Produkten erstellt wurde.
- `load customer` *`Firmencode KontaktnameFirmenname Anzahl_Bestellungen erste_Bestellungs-IDLieferstadt max_Artikel Skonto`*

  Lädt einen neuen Kunden (Customer) in das Grid und erstellt einen festen Satz an Bestellungen (Order-Entitäten) für ein zufälliges Produkt, das derzeit im Grid geladen ist. Die Anzahl der Bestellungen wird mit dem Parameter <Anzahl_Bestellungen> bestimmt. Jede Bestellung hat eine zufällige Anzahl an OrderDetail-Entitäten (bis <max-Artikel>).
- `display customer` *`Firmencode`*

  Zeigt eine Customer-Entität und die zugehörigen Order- und OrderDetail-Entitäten an.
- `display category` *`Kategorie-ID`*

  Zeigt eine Category-Entität eines Produkts und die zugehörigen Product-Entitäten an.

Ergebnisse

- `runclient.bat load default`
- `runclient.bat load customer IBM "John Doe" "IBM Corporation" 5 5000 Rochester 5 0.05`
- `runclient.bat load category 5 "Household Items" 100 5`
- `runclient.bat display customer IBM`
- `runclient.bat display category 5`

Muster-Grid und Java-Client mit Eclipse ausführen und erstellen

Das Einführungsmuster zum REST-Datenservice kann mit Eclipse aktualisiert und erweitert werden. Einzelheiten zum Konfigurieren Ihrer Eclipse-Umgebung finden Sie im Textdokument `Ausgangsverzeichnis_des_REST-Service/gettingstarted/ECLIPSE_README.txt`.

Nach dem Importieren des Projekts "WXSRestGettingStarted" in Eclipse und nach der erfolgreichen Erstellung des Musters wird das Muster automatisch neu kompiliert, und die zum Starten des Containerservers und -clients verwendeten Scriptdateien verwenden automatisch die Klassendateien und XML-Dateien. Der REST-Datenservice erkennt automatisch alle Änderungen, die nach der Konfiguration des Webservers vorgenommen wurden, damit die Eclipse-Build-Verzeichnisse automatisch gelesen werden.

Wichtig: Wenn Quellen- oder Konfigurationsdateien geändert werden, müssen der eXtreme-Scale-Container und die REST-Datenserviceanwendung erneut gestartet werden. Der eXtreme-Scale-Containerserver muss vor der Webanwendung des REST-Datenservice gestartet werden.

WCF-Client von Visual Studio 2008 mit dem REST-Datenservice

Das Einführungsmuster zum REST-Datenservice von eXtreme Scale enthält einen WCF-Data-Services-Client, der mit dem REST-Datenservice von eXtreme Scale interagieren kann. Das Muster ist als Befehlszeilenanwendung in C# geschrieben.

Softwarevoraussetzungen

Der in C# geschriebene Musterclient von WCF Data Services setzt Folgendes voraus:

- Betriebssystem
  - Microsoft Windows XP
  - Microsoft Windows Server 2003
  - Microsoft Windows Server 2008
  - Microsoft Windows Vista
- Microsoft Visual Studio 2008 mit Service-Pack 1

  Tipp: Zusätzliche Hardware- und Softwarevoraussetzungen finden Sie unter dem vorherigen Link.
- Microsoft .NET Framework 3.5 Service Pack 1
- Microsoft Support: An update for the .NET Framework 3.5 Service Pack 1 is available

Einführungslink erstellen und ausführen

Der Musterclient von WCF Data Services enthält ein Visual-Studio-2008-Projekt sowie eine Lösung und den Quellcode für die Ausführung des Musters. das Muster muss in Visual Studio 2008 geladen und in ein ausführbares Windows-Programm kompiliert werden, bevor es ausgeführt werden kann. Informationen zum Erstellen und Ausführen des Musters finden Sie im folgenden Textdokument: `Ausgangsverzeichnis_des_REST-Service/gettingstarted/VS2008_README.txt`.

Befehlssyntax für den in C# geschriebenen WCF-Data-Service-Client

Windows `WXSRESTGettingStarted.exe <Service-URL> <Befehl>`

<Service-URL> steht für den URL des zuvor konfigurierten REST-Datenservice von eXtreme Scale.

Die folgenden Befehle sind verfügbar:

- `load default`

  Lädt einen vordefinierten Satz von Customer-, Category- und Product-Entitäten in das Grid und erstellt einen zufälligen Satz von Bestellungen (Order) für jeden Kunden (Customer).
- `load category <Kategorie-ID> <Kategoriename> <erste_Produkt-ID> <Anzahl_Produkte>`

  Erstellt eine Produktkategorie und eine festgelegte Anzahl an Product-Entitäten im Grid. Der Parameter "erste_Produkt-ID" gibt die ID-Nummer des ersten Produkts an, und jedem nachfolgenden Produkt wird die jeweils nächste ID zugeordnet, bis die angegebene Anzahl an Produkten erstellt wurde.

- `load customer <Firmencode> <Kontaktname> <Firmenname> <Anzahl_Bestellungen> <erste_Bestellungs-ID> <Lieferstadt> <max_Artikel> <Skonto>`

  Lädt einen neuen Kunden (Customer) in das Grid und erstellt einen festen Satz an Bestellungen (Order-Entitäten) für ein zufälliges Produkt, das derzeit im Grid geladen ist. Die Anzahl der Bestellungen wird mit dem Parameter <Anzahl_Bestellungen> bestimmt. Jede Bestellung hat eine zufällige Anzahl an OrderDetail-Entitäten (bis <max-Artikel>).
- `display customer <Firmencode>`

  Zeigt eine Customer-Entität und die zugehörigen Order- und OrderDetail-Entitäten an.
- `display category <Kategorie-ID>`

  Zeigt eine Category-Entität eines Produkts und die zugehörigen Product-Entitäten an.
- `unload`

  Entfernt alle Entitäten, die mit dem Befehl "default load" geladen wurden.

Die folgenden Beispiele veranschaulichen verschiedene Befehle.

- `WXSRestGettingStarted.exe http://localhost:8080/wxsrestservice/restservice/NorthwindGrid load default`
- `WXSRestGettingStarted.exe http://localhost:8080/wxsrestservice/restservice/NorthwindGrid load customer`
- `IBM "John Doe" "IBM Corporation" 5 5000 Rochester 5 0.05`
- `WXSRestGettingStarted.exe http://localhost:8080/wxsrestservice/restservice/NorthwindGrid load category 5 "Household Items" 100 5`
- `WXSRestGettingStarted.exe http://localhost:8080/wxsrestservice/restservice/NorthwindGrid display customer IBM`
- `WXSRestGettingStarted.exe http://localhost:8080/wxsrestservice/restservice/NorthwindGrid display category 5`

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. Anstelle der IBM Produkte, Programme oder Services können auch andere, ihnen äquivalente Produkte, Programme oder Services verwendet werden, solange diese keine gewerblichen oder anderen Schutzrechte von IBM verletzen. Die Verantwortung für den Betrieb der Produkte, Programme oder Fremdservices in Verbindung mit Fremdprodukten und Fremdservices liegt beim Kunden, soweit solche Verbindungen nicht ausdrücklich von IBM bestätigt sind.

Für in diesem Handbuch 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

T

U

V

W

Antwort

eXtreme Scale Version 7.1
Produktübersicht
WebSphere eXtreme Scale
Produktübersicht

Anregungen zur Verbesserung und Ergänzung dieser Veröffentlichung nehmen wir gerne entgegen. Bitte informieren Sie uns über Fehler, ungenaue Darstellungen oder andere Mängel.

Zur Klärung technischer Fragen sowie zu Liefermöglichkeiten und Preisen wenden Sie sich bitte entweder an Ihre IBM Geschäftsstelle, Ihren IBM Geschäftspartner oder Ihren Händler.

Unsere Telefonauskunft "HALLO IBM" (Telefonnr.: 0180 3 313233) steht Ihnen ebenfalls zur Klärung allgemeiner Fragen zur Verfügung.

Kommentare:

Danke für Ihre Bemühungen.

Sie können ihre Kommentare betr. dieser Veröffentlichung wie folgt senden:
- Als Brief an die Postanschrift auf der Rückseite dieses Formulars
- Als E-Mail an die folgende Adresse: ibmterm@de.ibm.com

Name	Adresse
Firma oder Organisation	
Rufnummer	E-Mail-Adresse

Antwort

IBM

IBM Deutschland GmbH
SW TSC Germany

71083 Herrenberg