WebSphere® eXtreme Scale Version 7.0
Produktübersicht

WebSphere eXtreme Scale Produktübersicht

WebSphere® eXtreme Scale Version 7.0
Produktübersicht

WebSphere eXtreme Scale Produktübersicht

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

Inhaltsverzeichnis

Abbildungsverzeichnis

Tabellen

Informationen zu *Produktübersicht*

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

Bibliothek von WebSphere eXtreme Scale

Die Bibliothek von WebSphere eXtreme Scale enthält die folgenden Bücher:

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

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

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

Zielgruppe

Dieses Buch ist für alle Benutzer bestimmt, die sich über das Produkt WebSphere eXtreme Scale informieren möchten.

Aufbau des Handbuchs

Das Buch enthält Informationen zu den folgenden wichtigen Themen:

- **Kapitel 1** enthält eine Übersicht über WebSphere eXtreme Scale.
- **Kapitel 2** enthält Informationen zu den Caching-Konzepten des Produkts.
- **Kapitel 3** enthält Informationen zur Cacheintegration.
- **Kapitel 4** enthält Informationen zur Skalierbarkeit.
- **Kapitel 5** enthält Informationen zur Verfügbarkeit.
- **Kapitel 6** enthält Informationen zur Sicherheit.
- **Kapitel 7** enthält Informationen zur Transaktionsverarbeitung.
- **Kapitel 8** enthält Lernprogramme zu den Basiskonzepten des Produkts.
- **Kapitel 9** enthält das Produktglossar.

Aktualisierungen für dieses Handbuch

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

Hinweise zu Rückmeldungen

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

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 starker Transaktionsaufkommen mit hoher Effizienz und linearer Skalierbarkeit und bietet Servicequalitäten wie Transaktionsintegrität, hohe Verfügbarkeit und voraussagbare Antwortzeiten.

Die elastische Skalierbarkeit von WebSphere eXtreme Scale wird durch verteiltes Objekt-Caching ermöglicht. Elastisch bedeutet, dass das Grid sich selbst überwacht und verwaltet, horizontale und vertikale Skalierbarkeit unterstützt und selbst-heilend ist, weil eine automatische Wiederherstellung nach Fehlern erfolgt. Die horizontale Skalierbarkeit ermöglicht das Hinzufügen von Speicherkapazität, während das Grid aktiv ist, ohne dass ein Neustart erforderlich ist. Im Gegensatz dazu ermöglicht die vertikale Skalierbarkeit das sofortige Entfernen von Speicherkapazität während des Betriebs.

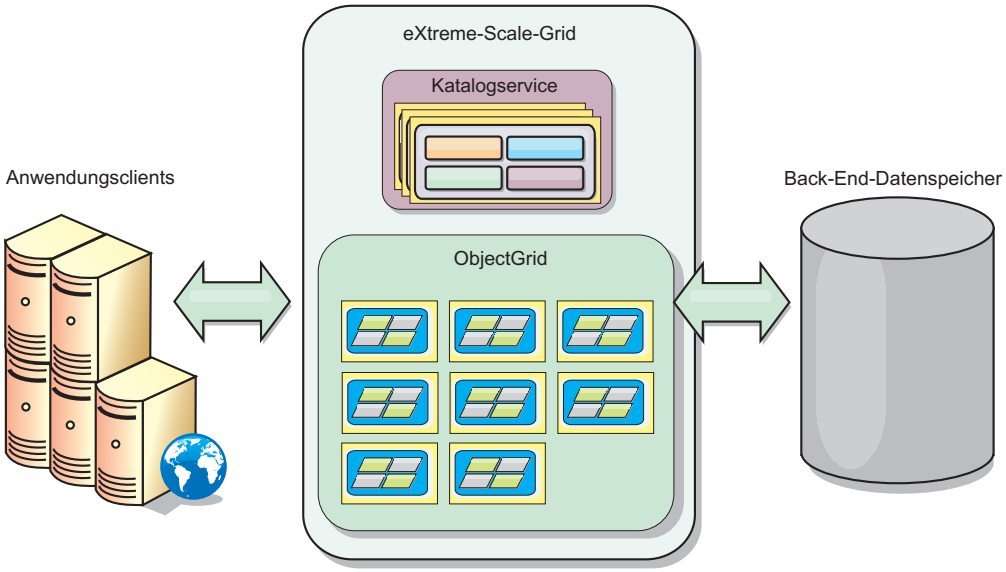
WebSphere eXtreme Scale kann auf verschiedene Arten eingesetzt werden. Das Produkt kann als extrem leistungsfähiger Cache oder als speicherinterner Datenbankverarbeitungsbereich für die Verwaltung des Anwendungsstatus oder als Plattform für die Erstellung leistungsfähiger XTP-Anwendungen (Extreme Transaction 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 Komplexitäten handhaben zu können, die eXtreme Scale unterstützt. Sie haben in beiden Szenarios 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.

Um solchen Fehlern entgegenzutreten, teilt eXtreme Scale die Datenmenge in Partitionen ein (äquivalent zu beschränkten Baumschemas), die jeweils aus einer primären Kopie (Shard) und Replikat-Shards zur Sicherung der Daten bestehen. Eine speicherinterne Datenbank kann diese Art von Funktionalität nicht bereitstellen, weil sie nicht auf diese Weise strukturiert und dynamisch ist. Sie sind dafür verantwortlich, die Operationen, die eXtreme Scale automatisch ausführt, manuell auszuführen. Da eine speicherinterne Datenbank natürlich eine Datenbank ist, kann sie auch SQL-Operationen zulassen und bietet im Vergleich mit Datenbanken, die nicht speicherintern sind, eine wesentlich Verarbeitungsgeschwindigkeit. WebSphere eXtreme Scale verwendet an Stelle der SQL-Unterstützung eine eigene Abfragesprache, ist aber wesentlich elastischer, unterstützt die Partitionierung von Daten und bietet eine zuverlässige Wiederherstellung nach Fehlern.

Mit seinem Write-Behind-Cachefeature kann WebSphere eXtreme Scale als Front-End-Cache für eine Datenbank eingesetzt werden, um den Durchsatz zu erhöhen und gleichzeitig Datenbanklast und Konkurrenzsituationen zu verringern. WebSphere eXtreme Scale ermöglicht eine voraussagbare horizontale Skalierung und vertikale Skalierung zu voraussagbaren Kosten.

Die folgende Abbildung zeigt eine verteilte kohärente Cacheumgebung, in der die eXtreme-Scale-Grid-Clients Daten an das Grid senden und Daten aus dem 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 Grid hinzugefügt werden. Die Skalierung des Caches erfolgt linear zum Wachstum des Grids. Die Daten können für zusätzliche Fehlertoleranz auch repliziert werden.



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 Maschine 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 weiterhin durch die 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 Grid-Paradigma ist ein Schlüssel/Wert-Paar, in dem das Grid Werte (Java-Objekte) mit einem zugehörigen Schlüssel (einem weiteren Java-Objekt) speicher, unter dem der Wert nachfolgend abgerufen wird. In eXtreme Scale ist eine Map ein Container für diese Schlüssel/Wert-Paare.

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

Es können nicht nur einfache Java-Objekte, sondern auch Objekte mit Beziehungen und eine SQL-ähnliche Abfragesprache (SELECT, FROM, WHERE) gespeichert werden, um diese abzurufen. Ein Kundenobjekt (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 die Programmierschnittstelle "EntityManager", um Entitäten ähnlich wie Java-Enterprise-Edition-Entitäten im Cache zu speichern. Entitätsbeziehungen können automatisch über eine XML-Entitätsdeskriptordatei oder über Annotationen in Java-Klassen erkannt werden. Somit kann eine Entität anhand des Primärschlüssels mit der Methode "find" der Schnittstelle "EntityManager" abgerufen werden. Entitäten können innerhalb der Transaktionsgrenzen als persistent im Grid definiert oder aus dem Grid 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 Antwortzeigen. Weitere Informationen finden Sie im Abschnitt zur Unterstützung von Real Time im Handbuch *Administratorhandbuch*.

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 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, falls Server oder ihre übergeordneten physischen Maschinen 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: Mit map.put(Schlüssel,Wert) wird ein Wert in den Cache gestellt, und mit and map.get(Schlüssel) wird der Wert anschließend abgerufen.

Es können nicht nur einfache Java-Objekte, sondern auch Objekte mit Beziehungen und eine SQL-ähnliche Abfragesprache (SELECT, FROM, WHERE) zwischengespeichert werden, um diese abzurufen. Ein Kundenobjekt (Order) kann beispielsweise ein Kundenobjekt (Customer) und mehrere zugehörige Artikelobjekte (Item) haben. WebSphere eXtreme Scale unterstützt 1:1-, 1:n- und n:n-Beziehungen. Außerdem wird in eXtreme die Programmierschnittstelle ″EntityManager″ unterstützt, über die Entitäten wie Java-Enterprise-Edition-Entitäten im Cache gespeichert werden können. Entitätsbeziehungen können automatisch über eine XML-Entitätsdeskriptordatei oder über Annotationen in Java-Klassen erkannt werden. Somit kann eine Entität anhand des Primärschlüssels mit der Methode ″find″ der Schnittstelle ″EntityManager″ abgerufen werden. Entitäten können innerhalb der Transaktionsgrenzen als persistent im Grid definiert oder aus dem Grid entfernt werden.

Neue und veraltete Features in diesem Release

WebSphere eXtreme Scale enthält zahlreiche neue Features in Version 7.0, unter anderem die Integration des dynamischen Caches, Maps für Bytefeldgruppen, und vieles mehr.

Neues in WebSphere eXtreme Scale Version 7.0

Tabelle 1. Neue Features in WebSphere eXtreme Scale Version 7.0

Feature	Beschreibung
Integration des dynamische Caches	Der dynamische Cacheprovider ermöglicht Anwendungen, die den dynamischen Cache von WebSphere Application Server verwenden, die Nutzung der hoch entwickelten Features und Leistungsverbesserungen von WebSphere eXtreme Scale. Mit diesem Feature erreichen Sie eine höhere Servicequalität, eine lineare Skalierbarkeit und eine hohe Verfügbarkeit zahlreicher Geschäftsanwendungen mit minimalen invasiven Änderungen. Lesen Sie die Informationen zum dynamischen Cache im Handbuch *Produktübersicht*.
Maps für Bytefeldgruppen	Mit Maps für Bytefeldgruppen kann die Anwendung den Wert eines Schlüssel/Wert-Paars in einer Bytefeldgruppe anstatt in Objektform speichern, was den Speicherbedarf großer Objektgraphen reduziert. Lesen Sie die Informationen zu Maps für Bytefeldgruppen im Handbuch *Programmierhandbuch*.
WebSphere Real Time	Mit der Unterstützung für WebSphere Real Time, dem branchenführenden Java-Angebot für die Echtzeitverarbeitung, ermöglicht WebSphere eXtreme Scale XTP-Anwendungen konsistentere und voraussagbare Antwortzeigen. Weitere Informationen finden Sie im Abschnitt zur Unterstützung von Real Time im Handbuch *Administratorhandbuch*.
Aktivierung von Metriken	WebSphere eXtreme Scale enthält Implementierungen von Metrikzugriffsadapters für eine verbesserte Integration von IBM® Tivoli Monitoring (ITM) und Hyperic HQ, die Ihnen einen umfassenden Einblick in das operative Verhalten von Geschäftslösungen bieten. Lesen Sie die Informationen zu den Überwachungstools anderer Anbieter im *Administratorhandbuch*..
Dynamische Maps	Die Erstellung von Multi-Tenant-Anwendungen wurde erheblich vereinfacht. Mit Hilfe von Map-Schablonen können Anwendungen neue Maps bei Bedarf erstellen und damit die Verwendung von Anwendungsdiskriminatoren in den Schlüsseln oder die Erstellung zusätzlicher Maps, die nie verwendet werden, vermeiden. Lesen Sie die Informationen zu dynamischen Maps im *Programmierhandbuch*..
Anforderungszeitlimit	WebSphere eXtreme Scale wurde erweitert und verarbeitet jetzt viele allgemeine Wiederholungs- und Ausnahmelogik-Tasks in der Grid-Middleware. Das Anforderungszeitlimit für Clients entlastet Entwickler, weil sie keine Standardwiederholungslogik für die meisten Operationen mit Map-Interaktion mehr schreiben müssen. Die meisten wiederholungsfähigen Bedingungen werden jetzt automatisch behandelt, so dass Sie sich auf die Geschäftslogikaspekte der Anwendungsentwicklung konzentrieren können. Lesen Sie die Informationen zum Anforderungszeitlimit im *Administratorhandbuch*
Zusammengesetzter Index	Dieses Feature kann die Verwendung von Indizes vereinfachen, wenn mehrere Attribute abgefragt werden, und den Aufwand für die Definition mehrerer Indizes einsparen. Auch die Schnittstelle "Query" wurde optimiert, so dass Sie jetzt zusammengesetzte Indizes verwenden können. Lesen Sie die Informationen zu zusammengesetzten Indizes im *Programmierhandbuch*..

Veraltete Features

Tabelle 2. Veraltete Features

Veraltet	Empfohlene Migrationsaktion
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.
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*.

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 Website Download eXtreme Scale trial 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.

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 eines verteilten Grids werden so auf die verschiedenen übergeordneten eXtreme-Scale-Server verteilt, dass jeder Server einige 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 beschränkt das Hauptspeicherkapazitätslimit des Servers die Größe einer Partition. 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 auf alle Server verteilt, so dass die Last auf die einzelnen Server so gleichmäßig wie möglich verteilt ist. Dies wird als horizontale Skalierung bezeichnet. Für eine vertikale Skalierung 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.

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.

Anwendungsimplementierung planen

Vor der Implementierung von WebSphere eXtreme Scale in einer Produktionsumgebung sind verschiedene Optionen zu berücksichtigen.

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 in jedem Server gespeichert?

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.

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 freigeben, 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 die Sitzungsverwaltung 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*.

Kapitel 2. Caching-Konzepte

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 eXtreme Scale verwendet. In diesem Abschnitt werden verschiedene Caching-Konzepte und -Szenarios und die verfügbaren Topologien für die Implementierung eines eXtreme-Scale-Grids beschrieben.

Architektur und Topologie

Mit WebSphere eXtreme Scale können Sie lokales speicherinternes Daten-Caching und verteiltes Client/Server-Daten-Caching konfigurieren.

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. Zwischengespeicherte Daten werden im eXtreme-Scale-Server gespeichert, und Clients stellen die Verbindung zum Server über Fernzugriff her.

Verteilte Caches bieten eine bessere Leistung, Verfügbarkeit und Skalierbarkeit und können über dynamische 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.

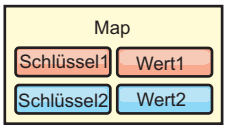


Abbildung 1. Map

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.

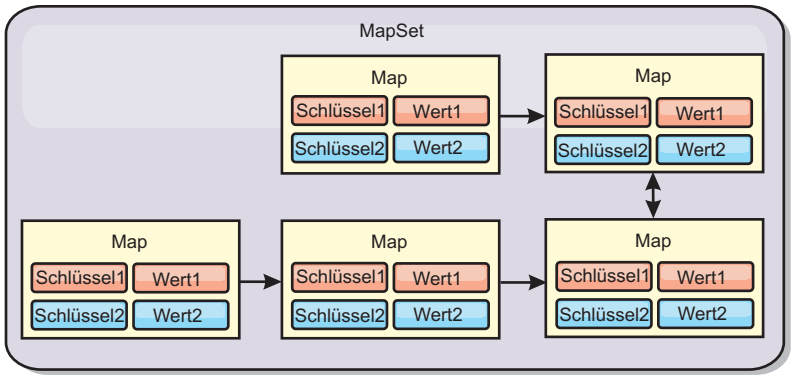


Abbildung 2. Mapsets

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.

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 eXtreme Scale können mehrere Map-Schemas definiert werden. Weitere Einzelheiten finden Sie in den Informationen zur API "ObjectMap" im *Programmierhandbuch*.

Außerdem kann 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. 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.

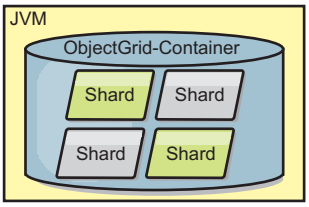


Abbildung 3. Container

Partitionen enthalten einen Teil der Daten des Grids.eXtreme Scale stellt automatisch mehrere Partitionen in einen einzelnen Container und verteilt die Partitionen dann, je mehr 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 ObjectGrid hat nach der Implementierung der Daten keine Möglichkeit mehr, ein erneutes Hashing für die gesamten Daten durchzuführen. Gehen Sie bei der Schätzung der Partitionsanzahl großzügig vor.

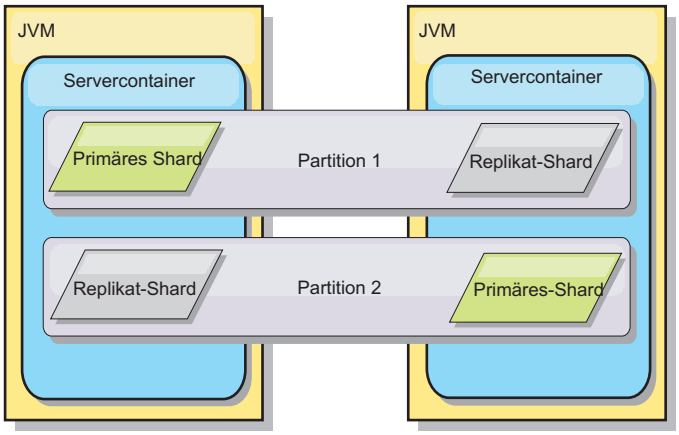


Abbildung 4. Partition

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.

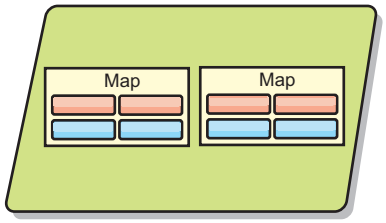


Abbildung 5. Shard

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 liegen kann.

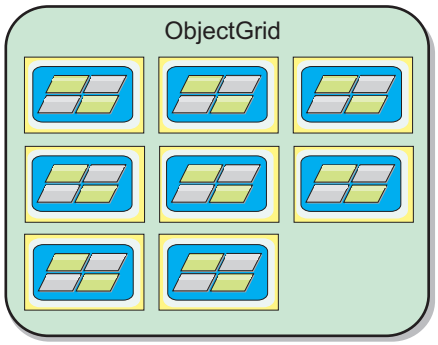


Abbildung 6. ObjectGrid

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 ausgefallen sind, wird der Client automatisch an den entsprechenden Server weitergeleitet, 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 einzelner Katalogservice kann für die Verwaltung mehrerer Instanzen von eXtreme Scale verwendet werden.
- 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.

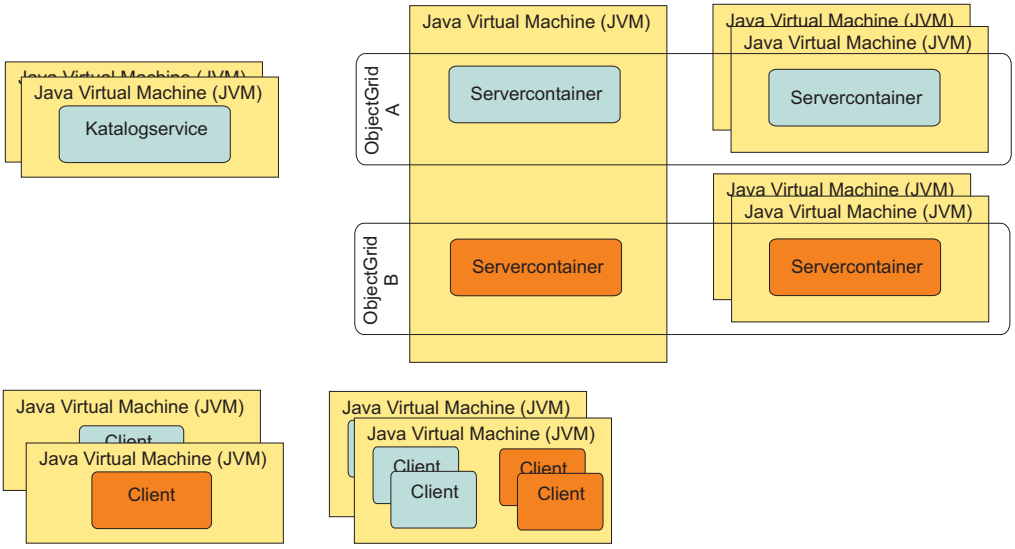


Abbildung 7. Mögliche Topologien

Katalogservice

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.

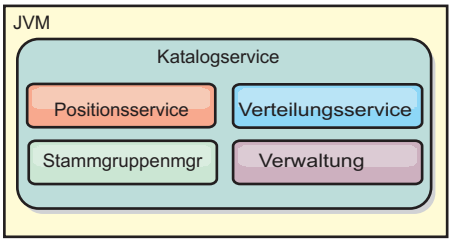


Abbildung 8. Katalogservice

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

Positionsservice
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 ein über eine 1:N-Beziehung ausgewählter Service im Cluster ausgeführt, so dass stets eine einzige Instanz des Verteilungsservice aktiv ist. 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 ein Katalogservice-Grid. Ein Katalogservice-Grid setzt sich aus mehreren Java Virtual Machines, einschließlich einer Master-JVM und einer Reihe von Ausweich-JVMs zusammen.

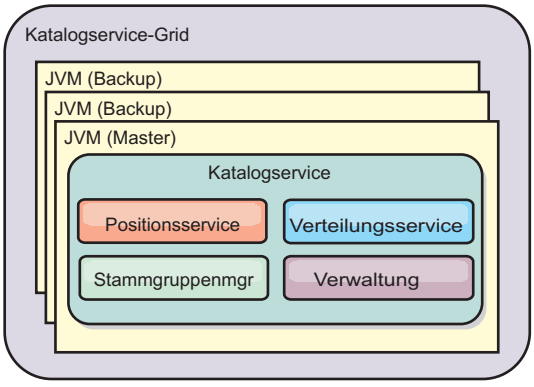


Abbildung 9. Katalogservice-Grid

Lokaler Speichercache

Im einfachsten Fall kann eXtreme Scale als lokaler 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 unterstützen, 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.

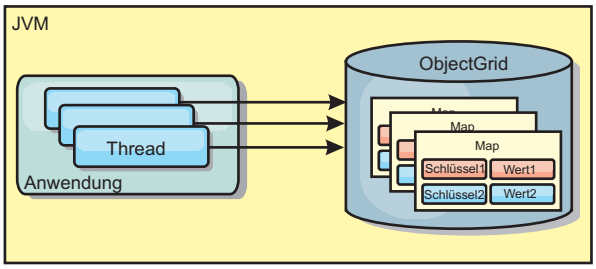


Abbildung 10. Szenario mit lokalem speicherinternen Speichercache

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 und für das Caching häufig aufgerufener Daten.
- 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 Bereinigungsprogrammen (Evictor), 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

Eine der Einschränkungen eines lokalen eXtreme-Scale-Caches ist die, dass es beim Vorhandensein mehrerer Prozesse mit unabhängigen Cacheinstanzen schwierig ist, den Cache synchron zu halten.

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

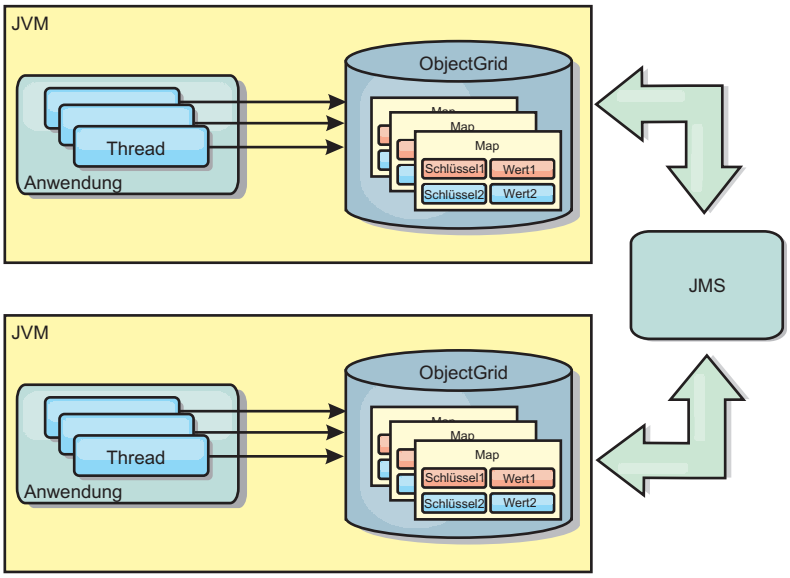


Abbildung 11. Auf Peers replizierter Cache mit Änderungen, die über JMS weitergegeben werden

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.

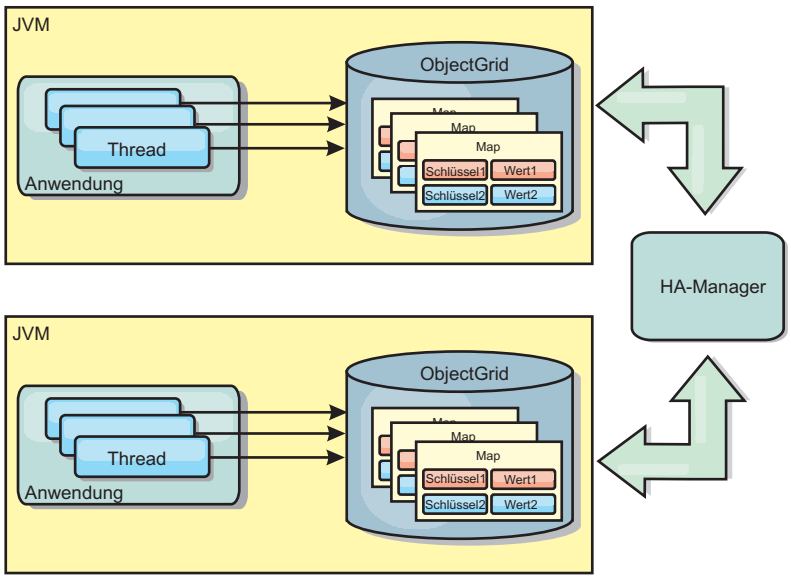


Abbildung 12. Auf Peers replizierter Cache mit Änderungen, die über den High Availability Manager weitergegeben werden

Vorteile

- Die Daten sind gültiger, 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 und für das Caching häufig aufgerufener Daten.
- 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 klein (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. Dies kann die Entwicklung und Implementierung der Anwendung vereinfachen, weil keine Datenbank konfiguriert werden muss.

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-Cache-instanzen weiterzugeben.

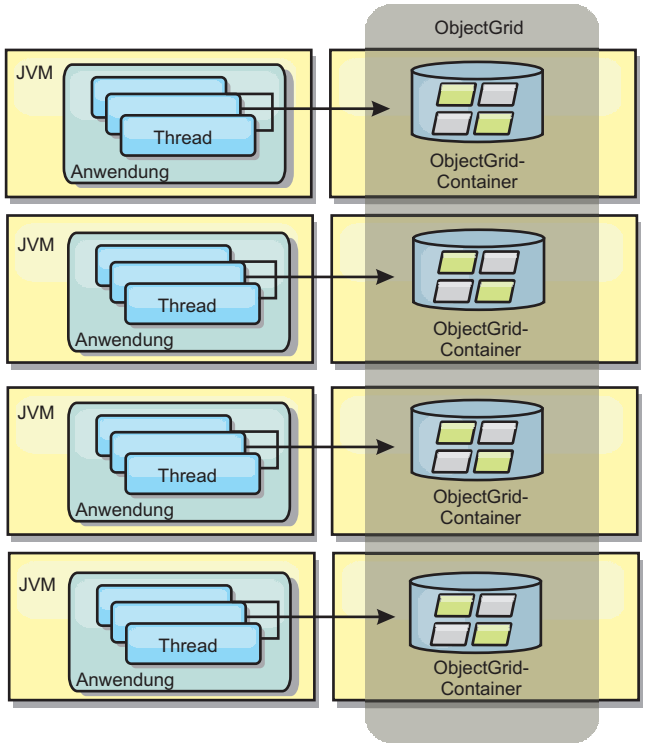


Abbildung 13. Verteilter Cache

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.

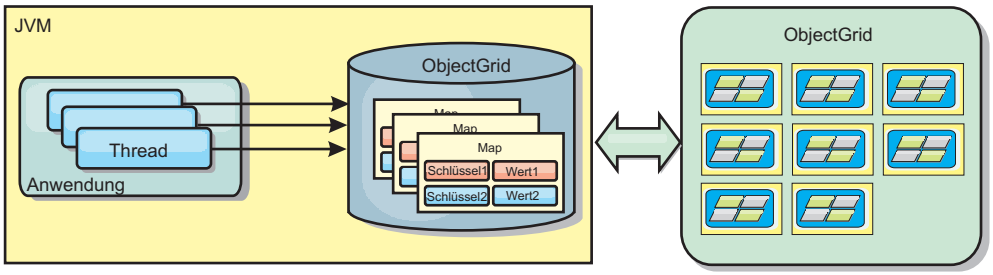


Abbildung 14. Naher Cache

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 Ladeprogramm-Plug-in haben. Wenn das Grid die angeforderten Daten nicht enthält, wird das Ladeprogramm 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 die Korrekturwerte des Clients auf ″0″. Einzelheiten zu den Sperrstrategien von eXtreme Scale finden Sie unter ″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 Korrekturwerte des Clients verwenden.

Vorteil

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

Nachteile

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

Einsatz

Verwenden Sie diese Technik, wenn die Antwortzeiten wichtig sind 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.

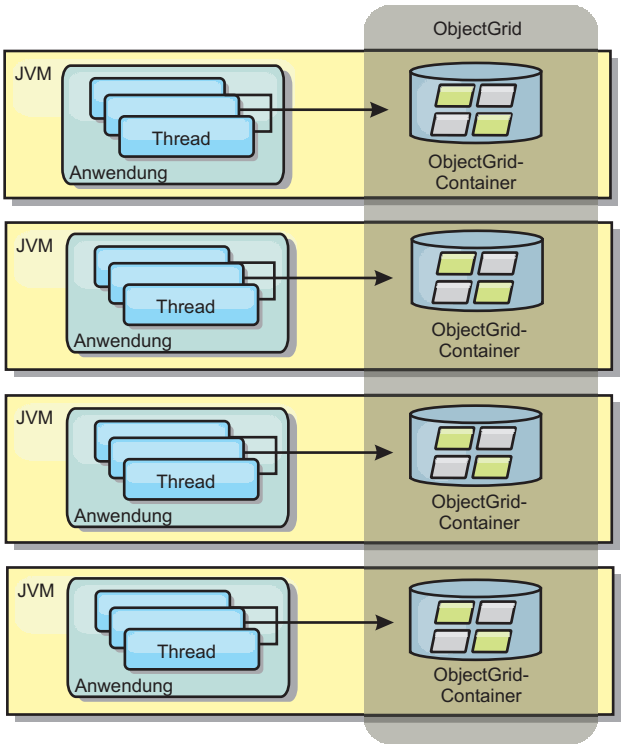


Abbildung 15. Integrierter Cache

Vorteile

- Vereinfachte Verwaltung, da weniger Prozesse zu verwalten sind
- Vereinfachte Anwendungsimplementierung, weil das Grid das Klassenladeprogramm 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 Anwendungs-Upgrades, 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*.

Datenbankintegration

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.

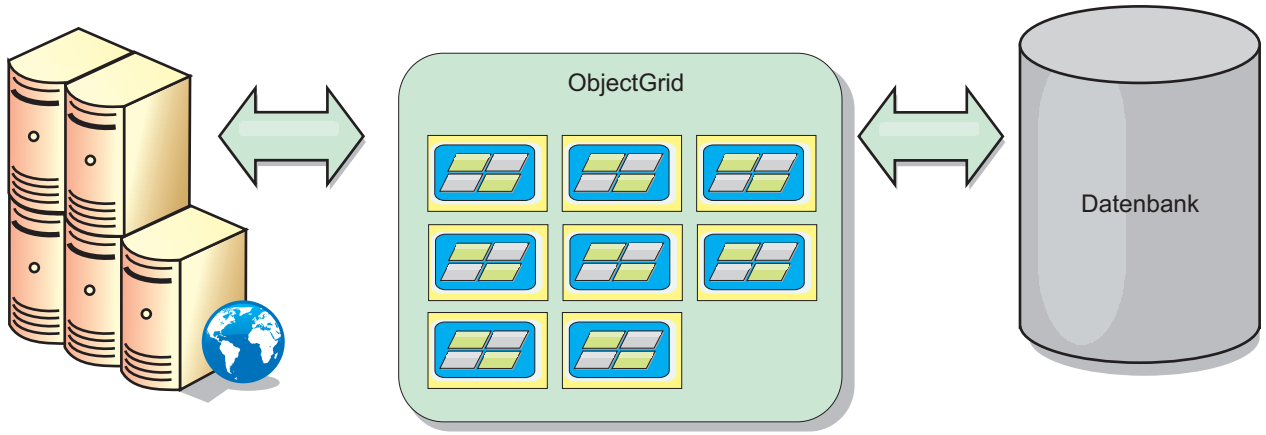


Abbildung 16. ObjectGrid als Datenbankpuffer

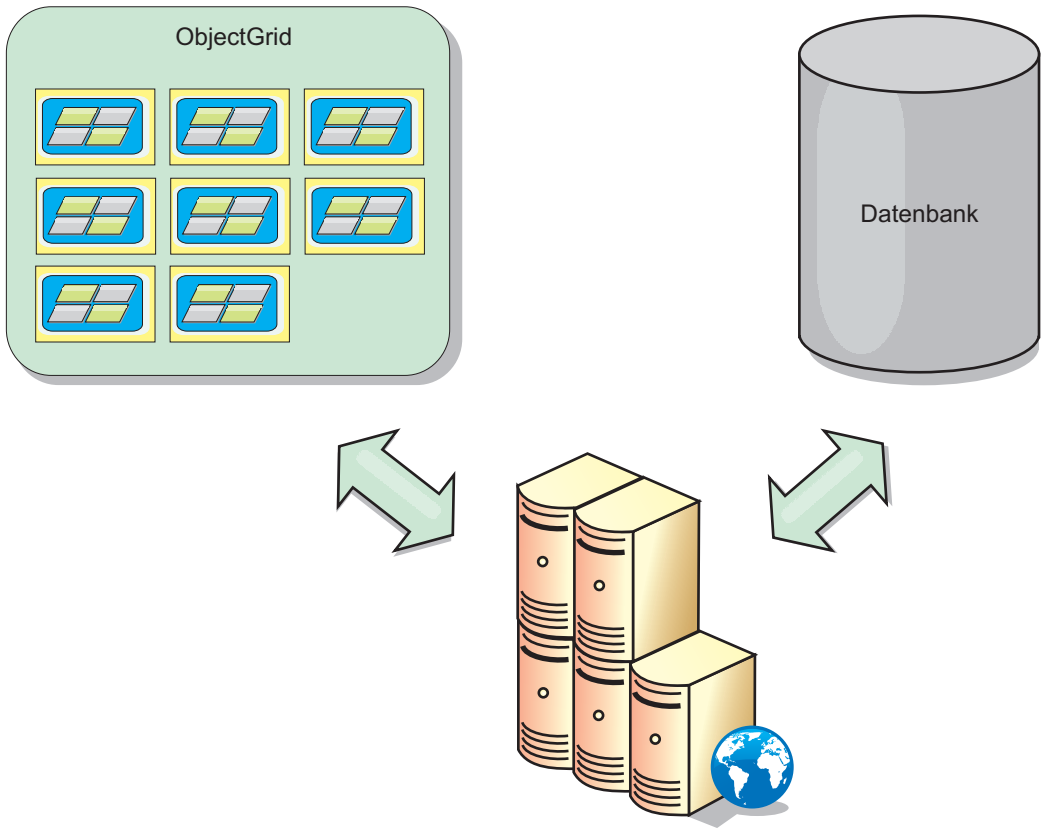


Abbildung 17. ObjectGrid als Nebencache

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, vorübergehend 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

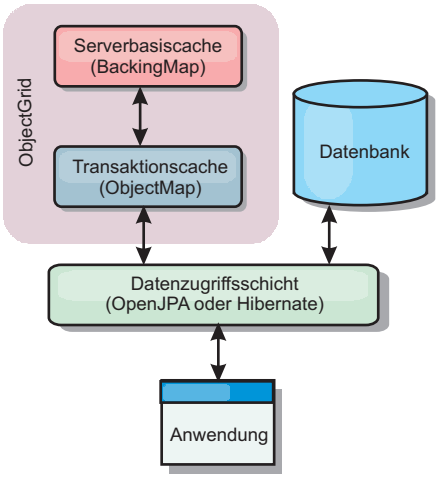


Abbildung 18. Nebencache

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

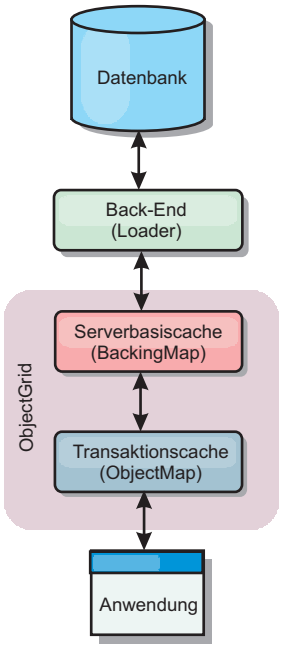


Abbildung 19. Integrierter Cache

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.

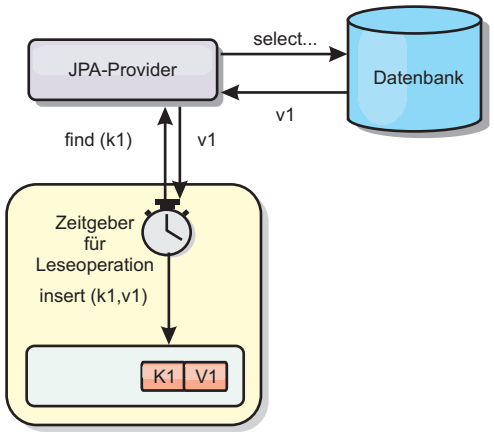


Abbildung 20. Regelmäßige Aktualisierung

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 können Daten aggressiver entfernen, wenn Speicherengpässe auftreten, indem die Option für speicherbasierte Bereinigung aktiviert wird. Weitere Einzelheiten finden Sie im Abschnitt „Bereinigung" auf Seite 27.

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.

Regelmäßige Aktualisierung

Wenn der speicherinterne Datenbankverarbeitungsbereich von 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. Die Verwendung einer regelmäßigen Aktualisierung ist eine Methode, die eXtreme Scale verwenden kann, um den Cache zu aktualisieren.

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.

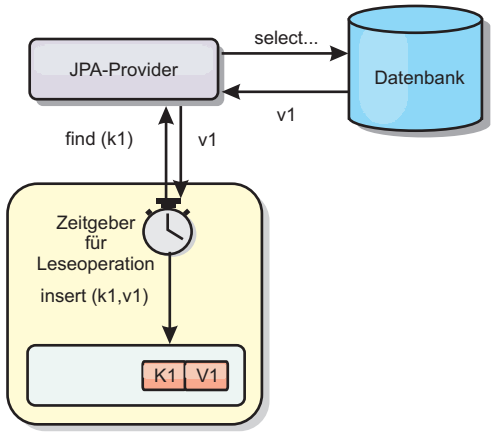


Abbildung 21. Regelmäßige Aktualisierung

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 können Daten aggressiver entfernen, wenn Speicherengpässe auftreten, indem die Option für speicherbasierte Bereinigung aktiviert wird. Weitere Informationen zu Bereinigungsprogrammen (Evictor) finden Sie im Abschnitt „Bereinigung".

Bereinigung

WebSphere eXtreme Scale stellt einen Standardmechanismus für das Entfernen von Cacheeinträgen und ein Plug-in für das Erstellen angepasster Bereinigungsprogramme (Evictor) bereit. Ein Bereinigungsprogramm steuert die Zugehörigkeit von Einträgen in jeder BackingMap. Das Standardbereinigungsprogramm verwendet eine Bereinigungsrichtlinie für jede BackingMap, die auf der Lebensdauer (TTL, Time-to-Live) basiert. Wenn Sie ein Plug-in-fähiges Bereinigungsprogramm bereitstellen, verwendet dieses gewöhnlich eine Bereinigungsrichtlinie, die auf der Anzahl der Einträge und nicht auf der Lebensdauer basiert.

Standard-TTL-Bereinigungsprogramm

WebSphere eXtreme Scale stellt ein TTL-Bereinigungsprogramm für jede BackingMap bereit. Das TTL-Bereinigungsprogramm verwaltet eine Verfallszeit für jeden erstellten Eintrag. Nach dem Ablauf der Verfallszeit eines Eintrags entfernt das Bereinigungsprogramm den Eintrag aus der BackingMap. Um den Leistungseinfluss zu minimieren, den das Entfernen eines Eintrags hat, kann das TTL-

Bereinigungsprogramm das Entfernen eines Eintrags über den Ablauf der Verfallszeit hinweg hinauszögern. Das TTL-Bereinigungsprogramm entfernt Einträge nie vor ihrem Verfallsdatum.

Die BackingMap hat Attribute, mit denen gesteuert werden kann, wie das TTL-Bereinigungsprogramm die Verfallszeit für jeden Eintrag berechnet. Anwendungen setzen das Attribut "ttlType", um anzugeben, wie das TTL-Bereinigungsprogramm die Verfallszeit berechnet. Das Attribut "ttlType" kann auf einen der folgenden Werte gesetzt werden:

1. None: Gibt an, dass Einträge in der BackingMap nicht verfallen. Das TTL-Bereinigungsprogramm entfernt diese Einträge nicht.
2. CreationTime: Gibt an, dass die Erstellungszeit eines Eintrags für die Berechnung der Verfallszeit verwendet wird.
3. LastAccessTime: Gibt an, dass die Zeit des letzten Zugriffs auf einen Eintrag für die Berechnung der Verfallszeit verwendet wird.

Wenn das Attribut "ttlType" für eine BackingMap nicht gesetzt ist, wird der Standardtyp "None" verwendet, so dass das TTL-Bereinigungsprogramm Einträge nie entfernt. Wenn das Attribut "ttlType" auf "CreationTime" oder "LastAccessTime" gesetzt ist, wird der Wert des TTL-Attributs in der BackingMap der Erstellungszeit bzw. der letzten Zugriffszeit hinzugefügt, um die Verfallszeit zu berechnen. Die Zeit für das TTL-Attribut einer Map wird in Sekunden angegeben. Der Wert 0 für das TTL-Attribut ist ein Sonderwert, der verwendet wird, um anzuzeigen, dass der Map-Eintrag eine unbegrenzte Lebensdauer hat, d. h., dass der Eintrag so lange in der Map bleibt, bis die Anwendung den Map-Eintrag explizit entfernt oder ungültig macht.

Optionale Bereinigungsprogramme

Das TTL-Standardbereinigungsprogramm verwendet eine Bereinigungsrichtlinie, die auf Zeit basiert, und die Anzahl der Einträge in der BackingMap hat keine Auswirkung auf die Verfallszeit eines Eintrags. Sie können ein optionales Plug-in-Bereinigungsprogramm verwenden, um Einträge auf der Basis der Anzahl vorhandener Einträge an Stelle der Zeit zu entfernen.

Die folgenden optionalen Plug-in-Bereinigungsprogramme stellen einige gängige Algorithmen bereit, mit denen entschieden werden kann, welche Einträge entfernt werden sollen, wenn eine BackingMap ihr Größenlimit erreicht. *

- Das Bereinigungsprogramm "LRUEvictor" verwendet einen LRU-Algorithmus (Least Recently Used), um zu entscheiden, welche Einträge entfernt werden sollen, wenn die BackingMap eine maximale Anzahl an Einträgen überschreitet.
- Das Bereinigungsprogramm "LFUEvictor" verwendet einen LFU-Algorithmus (Least Frequently Used), um zu entscheiden, welche Einträge entfernt werden sollen, wenn die BackingMap eine maximale Anzahl an Einträgen überschreitet.

Die BackingMap informiert ein Bereinigungsprogramm, wenn Einträge in einer Transaktion erstellt, geändert oder entfernt werden. Die BackingMap verfolgt diese Einträge und entscheidet, wann Einträge aus der BackingMap entfernt werden müssen.

Eine BackingMap hat keine Konfigurationsdaten für eine maximale Größe. Stattdessen werden Eigenschaften des Bereinigungsprogramms definiert, um das Verhalten des Bereinigungsprogramms zu steuern. Die Bereinigungsprogramme "LRUEvictor" und "LFUEvictor" haben beide eine Eigenschaft für die maximale Größe, die das Bereinigungsprogramm anweist, mit dem Entfernen von Einträgen zu

beginnen, wenn die maximale Größe überschritten wird. Wie das TTL-Bereinigungsprogramm entfernen auch die Bereinigungsprogramme "LRUEvictor" und "LFUEvictor" einen Eintrag möglicherweise nicht sofort, wenn die maximale Anzahl an Einträgen erreicht ist, um die Auswirkungen auf die Leistung zu minimieren.

Wenn der LRU- bzw. LFU-Bereinigungsalgorithmus für eine bestimmte Anwendung nicht angemessen ist, können Sie eigene Bereinigungsprogramme schreiben, um eine eigene Bereinigungsstrategie zu erstellen.

Speicherbasierte Bereinigung

Wichtig: Die speicherbasierte Bereinigung wird nur in Java Platform, Enterprise Edition Version 5 und höher unterstützt.

Alle integrierten Bereinigungsprogramme unterstützten die speicherbasierte Bereinigung, die in der Schnittstelle "BackingMap" aktiviert werden kann, indem das Attribut "evictionTriggers" der BackingMap auf `MEMORY_USAGE_THRESHOLD` gesetzt wird. Weitere Informationen zum Definieren des Attributs "evictionTriggers" in der BackingMap finden Sie in den Beschreibungen der Schnittstelle "BackingMap" und der ObjectGrid-XML-Deskriptordatei im *Administratorhandbuch*.

Die speicherbasierte Bereinigung basiert auf einem Schwellenwert für die Auslastung des Heap-Speichers. Wenn die speicherbasierte Bereinigung in der BackingMap aktiviert ist und die BackingMap ein integriertes Bereinigungsprogramm hat, wird der Schwellenwert für die Auslastung auf einen Standardprozentsatz des Gesamtspeichers gesetzt, wenn noch kein Schwellenwert definiert wurde.

Wenn Sie die speicherbasierte Bereinigung verwenden, müssen Sie den Schwellenwert für die Garbage-Collection auf denselben Wert wie die Zielauslastung des Heap-Speichers setzen. Beispiel: Wenn der Schwellenwert für die speicherbasierte Bereinigung auf 50 Prozent und der Schwellenwert für die Garbage-Collection auf den Standardwert von 70 Prozent gesetzt wird, kann die Auslastung des Heap-Speichers auf maximal 70 Prozent ansteigen. Diese Erhöhung der Heap-Speicherauslastung findet statt, weil die speicherbasierte Bereinigung erst nach einem Garbage-Collection-Zyklus ausgelöst wird.

Der von WebSphere eXtreme Scale verwendete Algorithmus für die speicherbasierte Bereinigung reagiert auf das Verhalten des verwendeten Algorithmus für die Garbage-Collection. Der beste Algorithmus für die speicherbasierte Bereinigung ist der IBM Standarddurchsatz-Collector. Algorithmen für eine Garbage-Collection nach Objektalter können ein unerwünschtes Verhalten bewirken, und deshalb sollten Sie diese Algorithmen nicht für die speicherbasierte Bereinigung verwenden.

Wenn Sie den Prozentsatz für den Auslastungsschwellenwert ändern möchten, setzen Sie die Eigenschaft "memoryThresholdPercentage" in den Container- und Servereigenschaftendateien für eXtreme-Scale-Serverprozesse.

Wenn die Speicherbelegung zur Laufzeit den Schwellenwert für die Zielauslastung überschreitet, beginnen speicherbasierte Bereinigungsprogramme mit dem Entfernen von Einträgen und versuchen, die Speicherbelegung unterhalb des Schwellenwerts für die Zielauslastung zu halten. Es gibt jedoch keine Garantien, dass die Bereinigung schnell genug ist, um potenzielle abnormale Speicherbedingungen zu verhindern, wenn die Laufzeitumgebung des Systems weiterhin so schnell Speicher belegt.

Bewährte Verfahren für Standardbereinigungsprogramme:

Sie können das Verhalten des TTL-Standardbereinigungsprogramms (Time to Live, Lebensdauer) ändern, indem Sie Attribute und Eigenschaften definieren.

Zusätzlich zu den Plug-in-Bereinigungsprogrammen, die im Abschnitt ″Bewährte Verfahren für Plug-in-Bereinigungsprogramme″ beschrieben werden, wird mit jeder BackingMap ein TTL-Standardbereinigungsprogramm erstellt. Das Standardbereinigungsprogramm entfernt Einträge, die auf einem TTL-Konzept basieren. Dieses Verhalten wird mit dem Attribut ″ttlType″ definiert. Es sind drei ttlType-Attribute vorhanden:

- Ohne: Gibt an, dass Einträge nicht verfallen und deshalb nicht aus der Map entfernt werden.
- Erstellungszeit: Gibt an, dass Einträge auf der Basis der Erstellungszeit entfernt werden.
- Letzte Zugriffszeit: Gibt an, dass Einträge auf der Basis der letzten Zugriffszeit entfernt werden.

Eigenschaft ″TimeToLive″

Diese Eigenschaft ist zusammen mit der Eigenschaft ″ttlType″ aus Leistungsperspektive das entscheidendste. Wenn Sie das ttlType-Attribut CREATION_TIME (Erstellungszeit) verwenden, entfernt das Bereinigungsprogramm einen Eintrag, wenn die seit der Erstellung des Eintrags vergangene Zeit dem Wert des Attributs ″TimeToLive″ entspricht. Wenn Sie das Attribut ″TimeToLive″ auf 10 Sekunden setzen, werden alle Einträge in der Map nach 10 Sekunden entfernt. Sie müssen beim Festlegen des Werts für das ttlType-Attribut CREATION_TIME vorsichtig vorgehen. Die Verwendung dieses Bereinigungsprogramms empfiehlt sich, wenn dem Cache sehr viele Einträge hinzugefügt werden, die nur für eine bestimmte Zeit verwendet werden. Mit dieser Strategie werden alle erstellten Einträge nach der festgelegten Zeit entfernt.

Im Folgenden finden Sie ein Beispiel für einen sinnvollen Einsatz des TTL-Typs CREATION_TIME: Sie verwenden eine Webanwendung, die Börsennotierungen abruft. Die Topaktualität der Notierungen ist jedoch nicht kritisch. In diesem Fall werden die Börsennotierungen 20 Minuten lang in einem ObjectGrid zwischengespeichert. Nach 20 Minuten verfallen die Einträge in der ObjectGrid-Map und werden daraufhin entfernt. In einem Intervall von ungefähr 20 Minuten verwendet die ObjectGrid-Map das Ladeprogramm-Plug-in, um die Map-Daten mit aktuellen Daten aus der Datenbank zu aktualisieren. Die Datenbank wird alle 20 Minuten mit den topaktuellen Börsennotierungen aktualisiert. Somit ist die Verwendung eines TimeToLive-Werts von 20 Minuten ideal.

Wenn Sie das ttlType-Attribut LAST_ACCESSED_TIME verwenden, legen Sie für ″TimeToLive″ einen niedrigeren Wert als beim ttlType-Attribut CREATION_TIME fest, weil das TimeToLive-Attribut eines Eintrags jedes Mal zurückgesetzt wird, wenn auf den Eintrag zugegriffen wird. Anders ausgedrückt, wenn das TimeToLive-Attribut den Wert 15 hat und ein Eintrag seit 14 Sekunden existiert, aber dann aufgerufen wird, wird seine Verfallszeit um weitere 15 Sekunden verlängert. Wenn Sie ″TimeToLive″ auf einen relativ hohen Wert setzen, werden viele Einträge möglicherweise nie entfernt. Setzen Sie ″TimeToLive″ jedoch auf einen Wert von ungefähr 15 Sekunden, können Einträge entfernt werden, wenn nicht sehr häufig auf sie zugegriffen wird.

Im Folgenden finden Sie ein Beispiel für einen sinnvollen Einsatz des TTL-Typs LAST_ACCESSED_TIME: Eine ObjectGrid-Map wird verwendet, um Sitzungsdaten eines Clients zu speichern. Sitzungsdaten müssen gelöscht werden, wenn der Client die Sitzungsdaten innerhalb eines bestimmten Zeitraums nicht verwendet. Die Sitzungsdaten verfallen beispielsweise, wenn innerhalb von 30 Minuten keine Aktivität des Clients erfolgt. In diesem Fall der TTL-Typ LAST_ACCESSED_TIME mit einem TimeToLive-Attributwert von 30 Minuten genau die Einstellung, die für diese Anwendung benötigt wird.

Im folgenden Beispiel wird eine BackingMap erstellt, das ttlType-Attribut für das Standardbereinigungsprogramm festgelegt und die Eigenschaft "TimeToLive" des Bereinigungsprogramms definiert:

```
ObjectGrid objGrid = new ObjectGrid;
BackingMap bMap = objGrid.defineMap("SomeMap");
bMap.setTtlEvictorType(TTLType.LAST_ACCESSED_TIME);
bMap.setTimeToLive(1800);
```

Die meisten Einstellungen des Bereinigungsprogramms müssen vor der Initialisierung des ObjectGrids gesetzt werden.

Sie können auch eigene Bereinigungsprogramme schreiben. Weitere Informationen hierzu finden Sie in den Informationen zum Schreiben angepasster Bereinigungsprogramme im *Programmierhandbuch*.

Szenarios mit integriertem 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 Ladeprogramm-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 Szenarios 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, der die Daten aus der Back-End-Datenbank lädt und den Cache einfügt. Nachfolgende Anforderungen für denselben Datenschlüssel werden im Cache gefunden, bis der Eintrag gelöscht, ungültig gemacht oder durch Bereinigung entfernt wird.

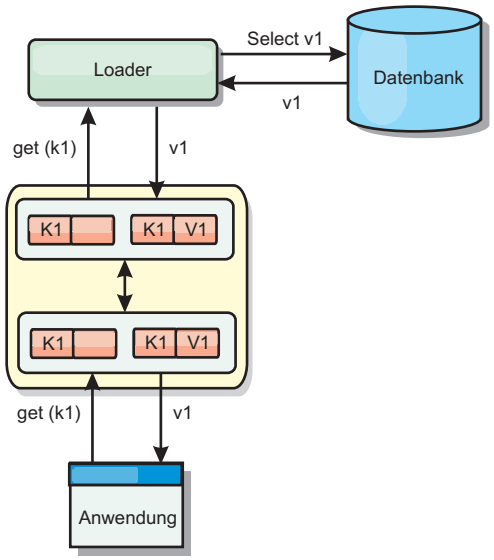


Abbildung 22. Read-Through-Caching

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.

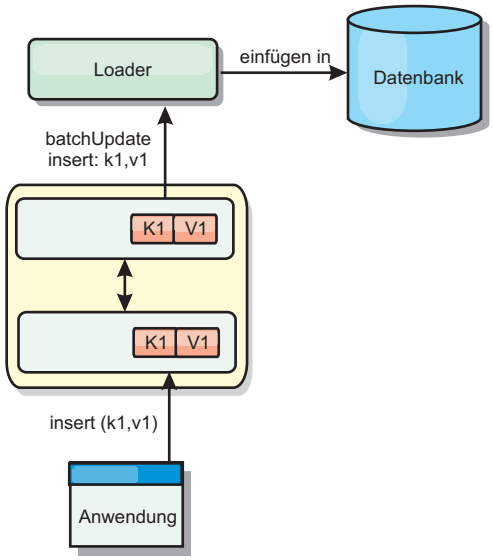


Abbildung 23. Write-Through-Caching

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 34.

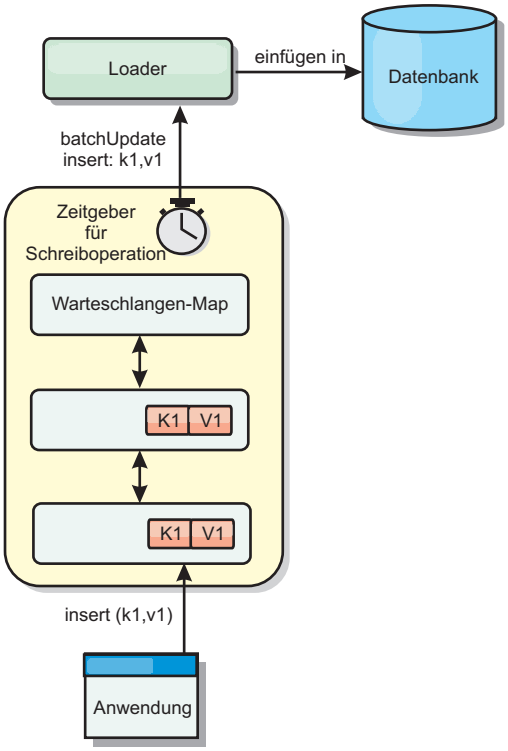


Abbildung 24. Write-Behind-Caching

Weitere Informationen finden Sie im Abschnitt „Write-Behind-Caching" auf Seite 34.

Loader

Ein Loader (Ladeprogramm) ist ein Plug-in, das als Verbindung zwischen einer BackingMap und einem Back-End, wie z. B. einer Datenbank, arbeitet.

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.

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.

Weitere Informationen zu Loadern finden Sie in den Informationen zu Loadern im *Administratorhandbuch*.

Write-Behind-Caching

Sie können Write-Behind-Caching verwenden, um die Kosten bei der Aktualisierung einer Back-End-Datenbank zu reduzieren.

Einführung

Beim Write-Behind-Caching werden Aktualisierungen für das Ladeprogramm-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.

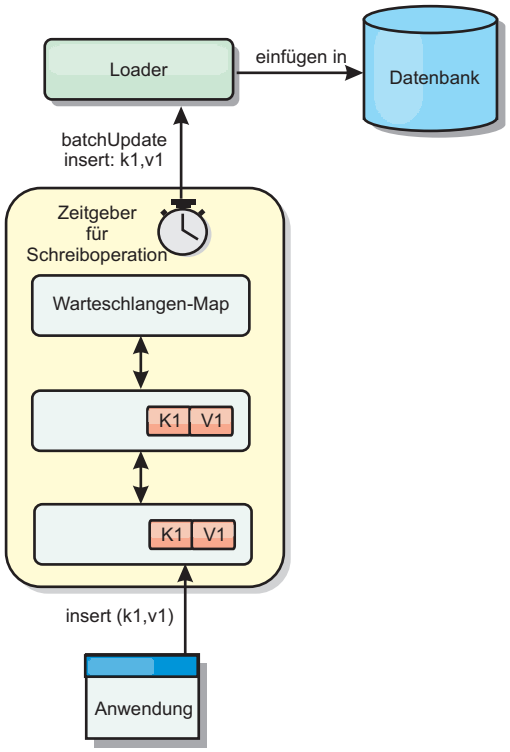


Abbildung 25. Write-Behind-Caching

Bei der Write-Behind-Konfiguration in einer BackingMap wird ein Thread zwischen dem Ladeprogramm und der Map erstellt. Anschließend delegiert das Ladeprogramm 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 das Write-Behind-Ladeprogramm 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 das echte Back-End-Ladeprogramm.

Das Write-Behind-Ladeprogramm sendet nur LogElement-Objekte der Typen ″insert″ (Einfügen), ″update″ (Aktualisieren) und ″delete″ (Löschen) an das echte Ladeprogramm. 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:** Das Write-Behind-Ladeprogramm fasst die Aktualisierungen auf Schlüsselbasis so zusammen, dass nur eine einzige zusammenfasste Aktualisierung 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 Ladeprogramm

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.

Das Write-Behind-Ladeprogramm 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 das Ladeprogramm idempotent ist, d. h., wenn Loader.batchUpdate(TxId, LogSequence) zweimal mit demselben Wert aufgerufen wird, liefen diese Aufrufe dasselbe Ergebnis wie ein einmaliger Aufruf. Ladeprogrammimplementierungen müssen zum

Aktivieren dieses Features die Schnittstelle ″RetryableLoader″ implementieren. Weitere Einzelheiten finden Sie in der API-Dokumentation.

Ausfall des Ladeprogramms

Das Ladeprogramm-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. Das Write-Behind-Ladeprogramm reiht die Aktualisierungen in eine Warteschlange ein und versucht anschließend in regelmäßigen Abständen, die Datenänderungen mit Push an das Ladeprogramm zu übertragen. Das Ladeprogramm muss die ObjectGrid-Laufzeitumgebung darüber benachrichtigen, dass ein Problem mit der Datenbankkonnektivität vorliegt, indem es eine Ausnahme vom Typ ″LoaderNotAvailableException″ auslöst.

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

- Wenn das Write-Behind-Ladeprogramm eine Ausnahme vom Typ ″LoaderException″ abfängt, geht es von einem Datenfehler aus, z. B. von einem doppelten Schlüssel. Das Write-Behind-Ladeprogramm 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 das Write-Behind-Ladeprogramm eine Ausnahme vom Typ ″LoaderNotAvailableException″ abfängt, geht es von einem Ausfall aus, weil es keine Verbindung zum Datenbank-Back-End herstellen kann, z. B., weil das Datenbank-Back-End inaktiv ist, keine Datenbankverbindung verfügbar ist oder das Netz inaktiv ist. Das Write-Behind-Ladeprogramm 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 das Write-Behind-Ladeprogramm 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 Ladeprogrammaktualisierung 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 das Ladeprogramm 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

Vorheriges Laden von Daten und Vorbereitung

In vielen Szenarios können Sie davon profitieren, wenn Sie Daten vorher in das Grid laden.

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 unmittelbaren 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 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.

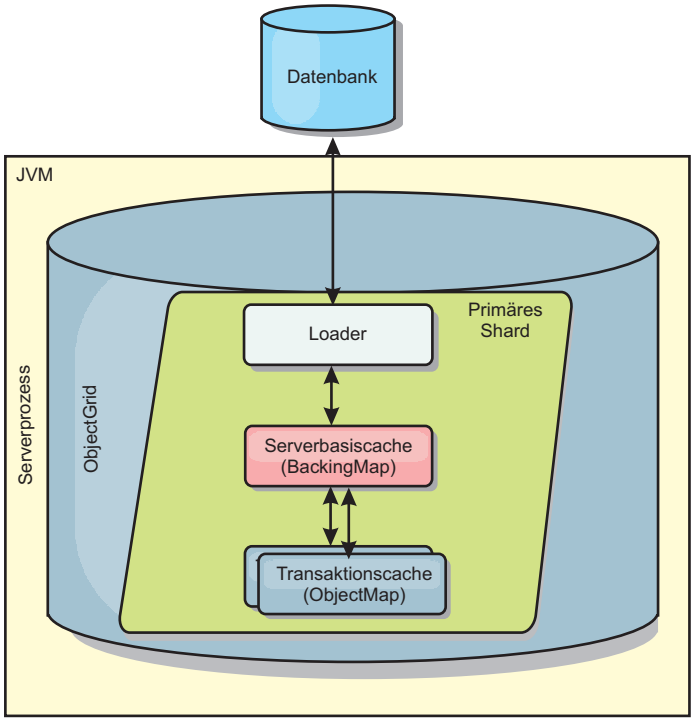


Abbildung 26. Loader-Plug-in

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.

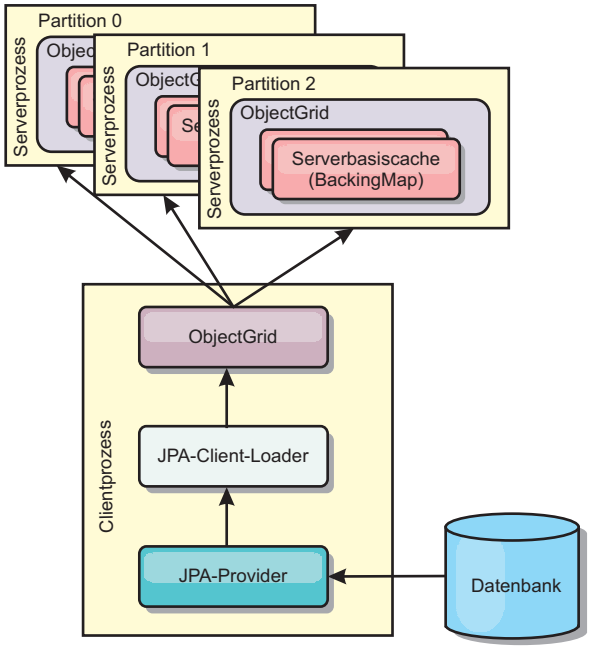


Abbildung 27. Client-Loader

Cachekonzepte für Java-Objekte

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 „Architektur und Topologie” auf Seite 9.

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 die Klassendefinitionen im Klassenpfad enthalten sein, 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 Abschnitt 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 eingeschränkten Baumstruktur 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 gebräuchliche 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 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. Die Daten, die das Format von Java-Objektinstanzen haben, werden in Bytes und bei Bedarf wieder zurück in Objekte konvertiert, um die Daten zwischen den Client- und Serverprozessen hin- und herzuschieben. Das Marshaling der Daten ist die kostenintensivste Operation und muss vom Anwendungsentwickler beim Design des Schemas, bei der Konfiguration des Grids und bei der Interaktion mit den Datenzugriffs-APIs berücksichtigt werden.

Die Java-Standardserialisierungs- und -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.

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.

Kapitel 3. Cacheintegration

WebSphere eXtreme Scale kann in andere Caching-Produkte integriert werden. JPA kann zwischen WebSphere eXtreme Scale und der Datenbank verwendet werden, um Änderungen als Ladeprogramm zu integrieren. Sie können auch den dynamischen Cacheprovider von WebSphere eXtreme Scale verwenden, um WebSphere eXtreme Scale als Plug-in in der dynamischen Cachekomponente von WebSphere Application Server zu verwenden. Eine andere Erweiterung von WebSphere Application Server ist der HTTP-Sitzungsmanager von WebSphere eXtreme Scale, der als Unterstützung für die Zwischenspeicherung von HTTP-Sitzungen eingesetzt werden kann.

eXtreme Scale mit JPA verwenden

Java Persistence API (JPA) ist eine Spezifikation, die die Zuordnung von Java-Objekten zu relationalen Datenbank ermöglicht. JPA enthält eine vollständige ORM-Spezifikation (Object-Relational Mapping, objektbezogene Zuordnung) mit Metadatenannotationen für die Sprache Java und XML-Deskriptoren für die Definition der Zuordnung von Java-Objekten zu einer relationalen Datenbank und umgekehrt. Es gibt eine Reihe von Open-Source- und kostenpflichtigen Implementierungen.

Zur Verwendung von JPA müssen Sie einen unterstützten JPA-Provider wie OpenJPA oder Hibernate, JAR-Dateien und eine Datei `META-INF/persistence.xml` in Ihrem Klassenpfad haben.

Übersicht über JPA-Loader

Ein JPA-Loader (Java Persistence API (JPA)) ist eine Ladeprogramm-Plug-in-Implementierung, die JPA für die Interaktion mit der Datenbank verwendet.

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.

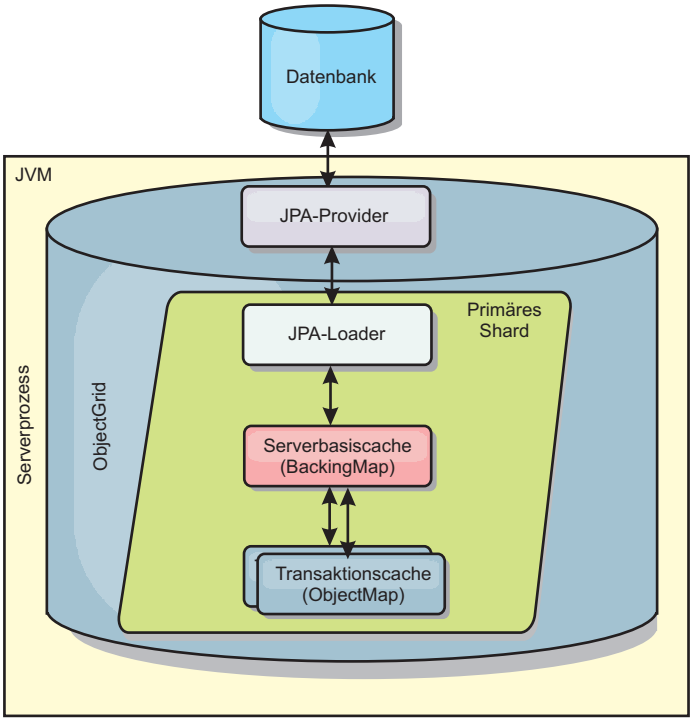


Abbildung 28. Architektur der JPA-Loader

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 JPA-Loader 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 Einzelheiten finden Sie im Abschnitt JPAEntityLoader-Plug-in.

Methoden

Ladeprogramme (oder Loader) 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 das Ladeprogramm 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 Ladeprogramme konfiguriert werden.

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 sich 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.

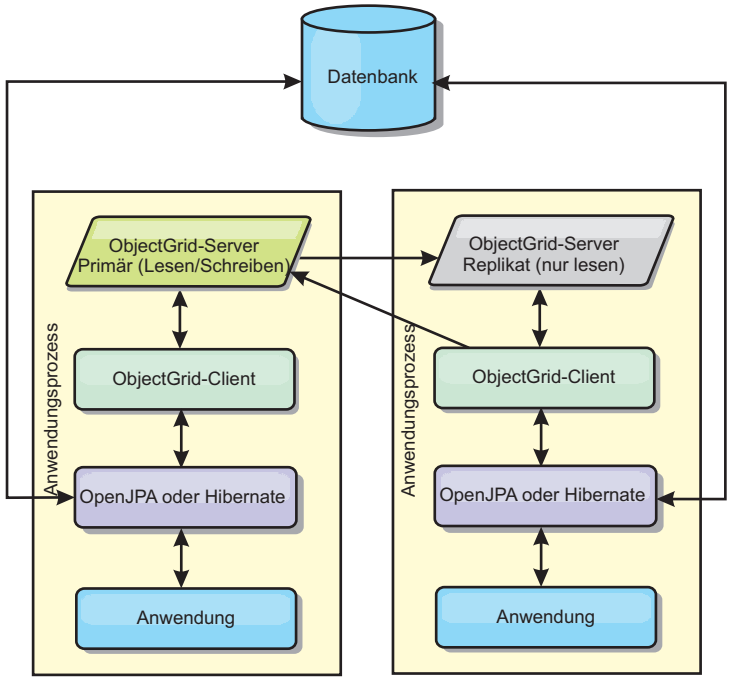


Abbildung 29. Integrierte JPA-Topologie

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.

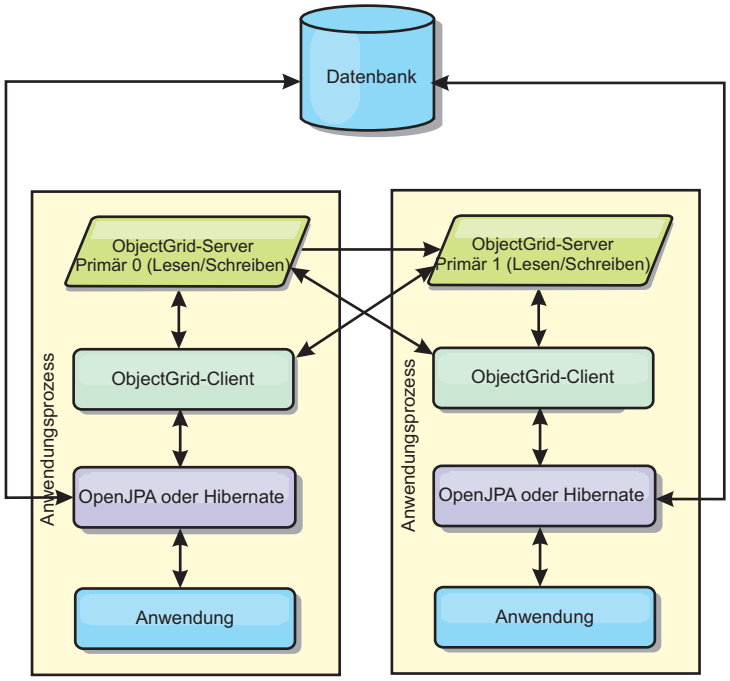


Abbildung 30. Integrierte, partitionierte JPA-Topologie

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.

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 Aufrufe. 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 konfigurieren, dass eXtreme Scale partitioniert und repliziert wird. Eine ferne Konfiguration wird unabhängig von der Anwendung und vom JPA-Provider verwaltet.

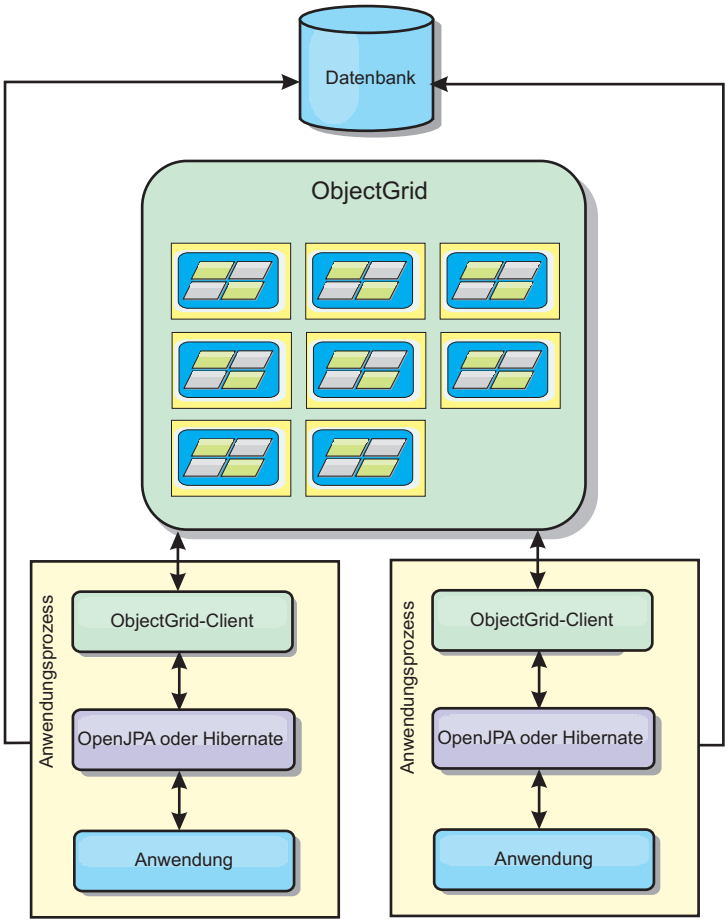


Abbildung 31. Ferne JPA-Topologie

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.

Verwaltung von HTTP-Sitzungen

In einer Umgebung mit WebSphere Application Server Version 5 oder höher kann der mit WebSphere eXtreme Scale bereitgestellte HTTP-Sitzungsmanager den Standardsitzungsmanager im Anwendungsserver außer Kraft setzen.

Der HTTP-Sitzungsmanager von WebSphere eXtreme Scale kann auch in WebSphere Application Server Version 6.0.2 oder höher oder auch in einer Umgebung ohne WebSphere Application Server, wie z. B. WebSphere Application Server Community Edition oder Apache Tomcat ausgeführt werden.

Features

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 Sitzungsverwaltungsfunktionen für eine zugeordnete Anwendung zur Verfügung. Der Sitzungsmanager erstellt HTTP-Sitzungen und verwaltet die Lebenszyklen von HTTP-Sitzungen, 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.

Einsatzszenarios

Der Sitzungsmanager kann in den folgenden Szenarios 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 Last-

ausgleichsfunktion verwaltet wird und der Affinitätsmechanismus an diese Lastausgleichsfunktion angepasst werden muss.
- In einer Umgebung, in der die Sitzungsverwaltung und -speicherung in einen externen Java-Prozess ausgelagert werden muss.
- In mehreren Zellen, in denen das Sitzungs-Failover zwischen den Zellen aktiviert werden muss.

Funktionsweise des Sitzungsmanagers

Der Sitzungsmanager integriert sich in den Anforderungspfad in Form eines Servlet-Filters. Mit Hilfe von Tools, die mit WebSphere eXtreme Scale bereitgestellt werden, können Sie diesen Servlet-Filter jedem Webmodul hinzufügen. Sie können den Filter auch manuell dem Webimplementierungsdeskriptor Ihrer Anwendung hinzufügen. Dieser Filter empfängt die Anforderung vor dem Servlet bzw. den JSP-Dateien in der Zielanwendung. Der Filter fängt die HTTPRequest- und HTTPResponse-Objekte ab und erstellt ein Wrapper-Objekt in seiner Implementierung.

Diese Filterimplementierung fängt alle Aufrufe für die HTTPRequest- und HTTPResponse-Objekte ab, die sich auf die HTTP-Sitzung beziehen. Diese Aufrufe werden vom Sitzungsmanager bearbeitet und nicht an den Basissitzungsmanager in der zugrunde liegenden Java-EE-Serverimplementierung übergeben. Der Sitzungsmanager erstellt und verwaltet Sitzungen und ihre Lebenszyklen. Dazu gehört, dass er Sitzungen nach Ablauf eines festgelegten Zeitlimits ungültig macht und Listener startet, wenn Sitzungen ungültig gemacht werden oder andere Lebenszyklusereignisse eintreten.

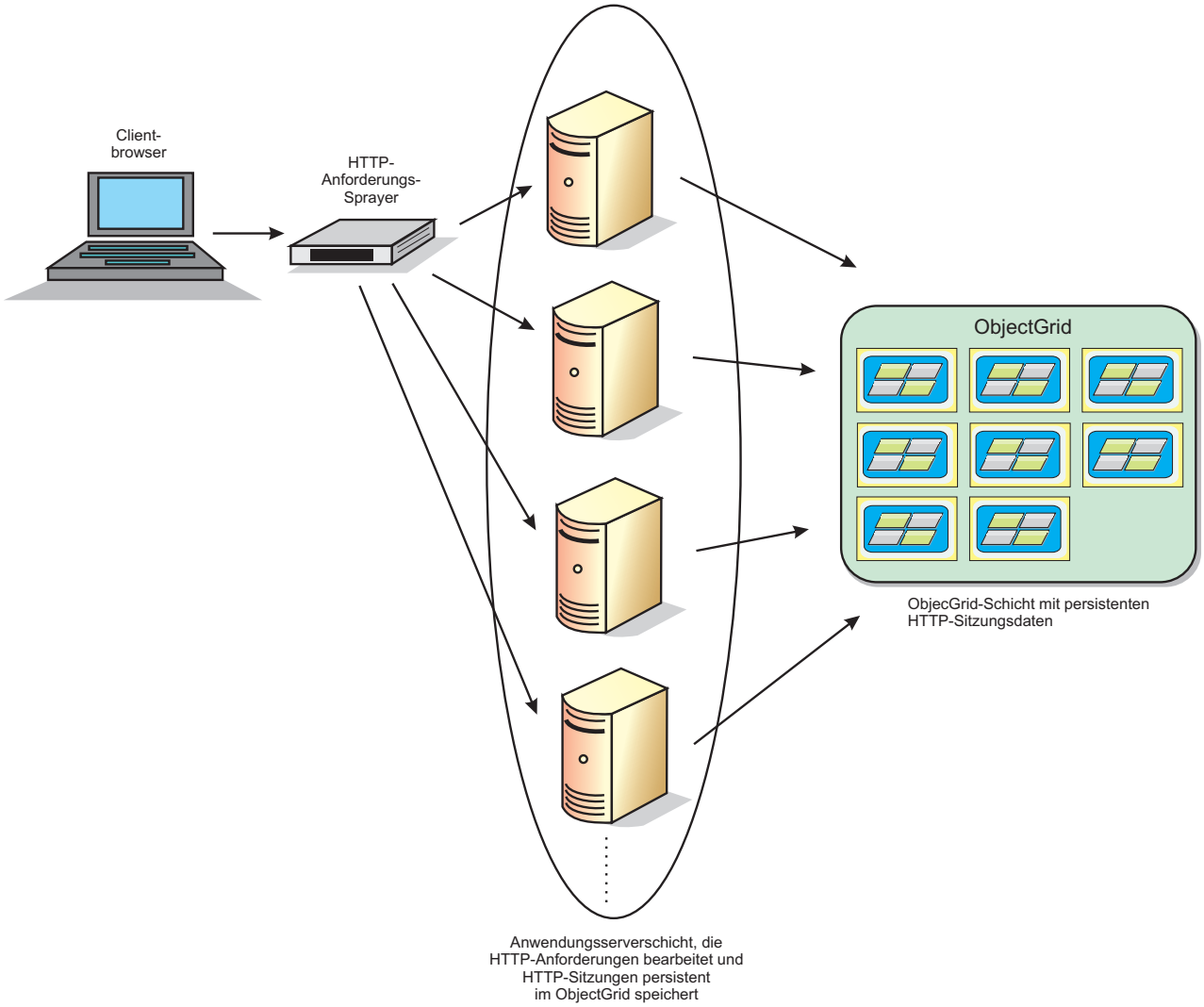


Abbildung 32. Topologie für die HTTP-Sitzungsverwaltung mit einer fernen Containerkonfiguration

Implementierungstopologien

Es gibt zwei dynamische Implementierungsszenarios 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.
- **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 einer fernen Instanz von eXtreme Scale.

eXtreme-Scale-Sitzung in einer Sitzungsmanageranwendung abrufen

```
public void doGet(HttpServletRequest req, HttpServletResponse resp)
    throws ServletException, IOException {

        HttpSession session = req.getSession(true);

        System.out.println("calling getSession");
        // call getAttribute("com.ibm.websphere.objectgrid.session")
    // to get the ObjectGrid session instance
        Session ogSession = (Session)session.getAttribute
      ("com.ibm.websphere.objectgrid.session");

        System.out.println("ogSession = "+ogSession);
    }
```

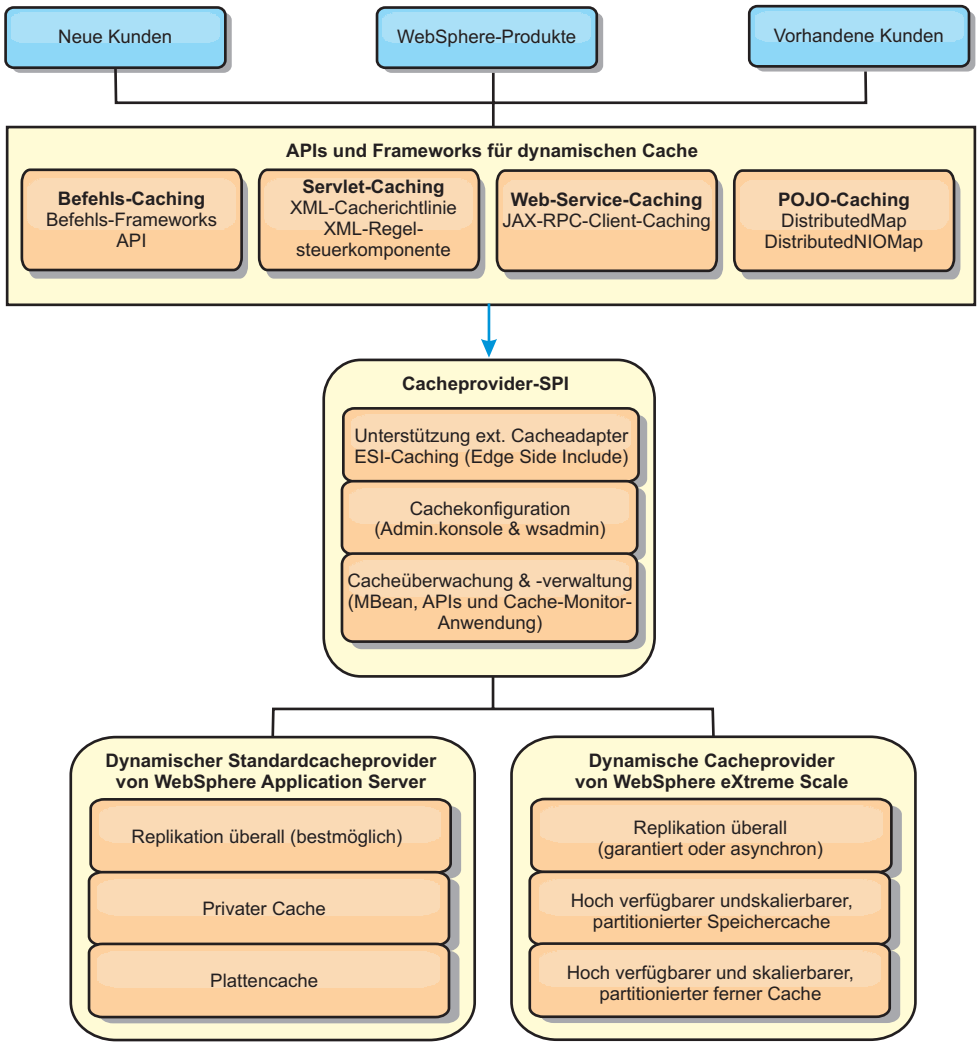
Alle Änderungen, die über die vom Methodenaufruf ″getAttribute″ zurückgegebenen Sitzung vorgenommen werden, werden festgeschrieben, wenn die zugrunde liegende Sitzung festgeschrieben wird.

Dynamischer Cacheprovider von WebSphere eXtreme Scale

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

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.



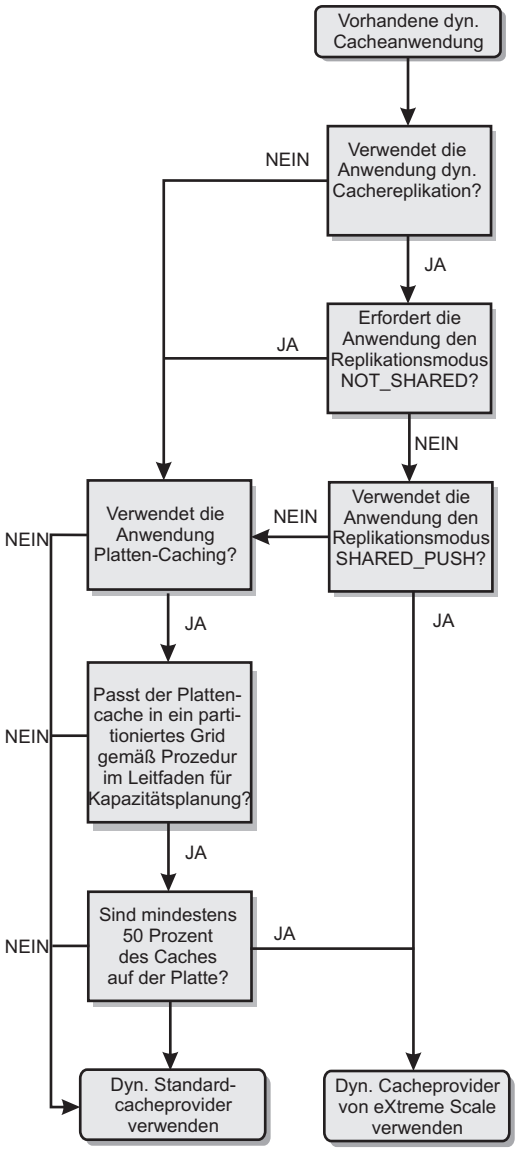
Sie können den dynamischen Cacheprovider gemäß der Beschreibung im Abschnitt „Dynamischen Cacheprovider für WebSphere eXtreme Scale konfigurieren" auf Seite 66 installieren und konfigurieren.

Einsatz von WebSphere eXtreme Scale festlegen

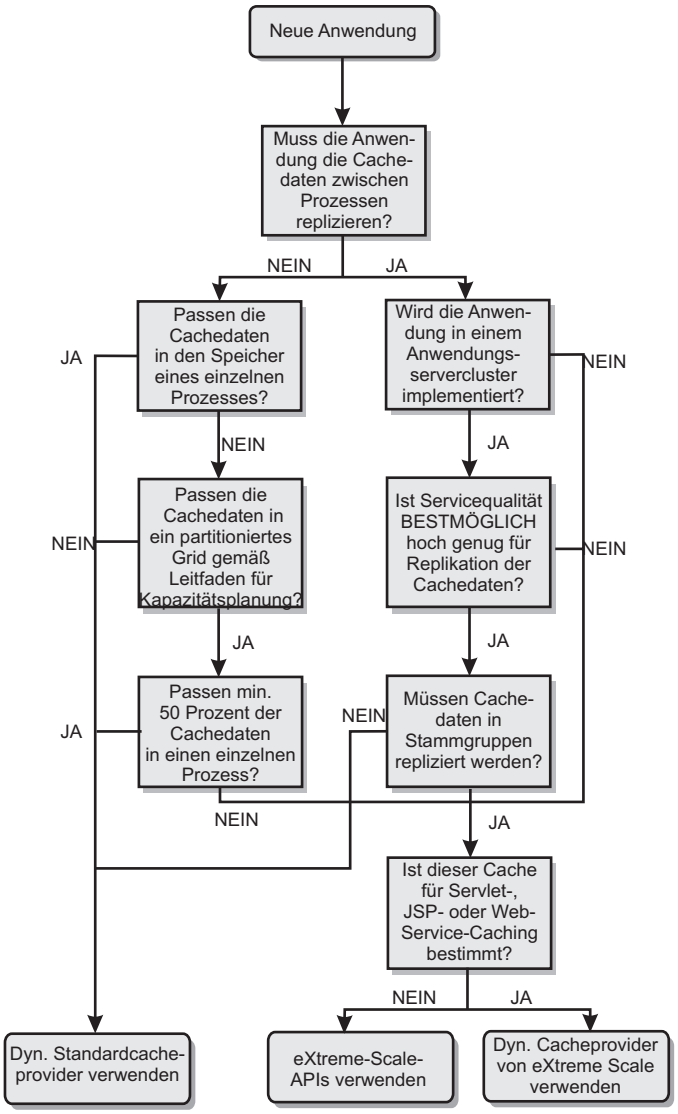
Die verfügbaren Features in WebSphere eXtreme Scale erweitern die verteilten Funktionen der API "Dynamic Cache" weit über das hinaus, was die Standardsteuerkomponente für dynamischen Cache und der Datenreplikationsservice bieten. Mit eXtreme Scale können Sie Caches erstellen, die wirklich auf mehrere 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 3. 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 3. 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			Ladeprogramme
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 4. 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 5. 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 der Funktionsweise von verteilten eXtreme-Scale-Caches finden Sie im Abschnitt ″Implementierungskonfigurationen für eXtreme Scale″ im *Programmierhandbuch*.

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.

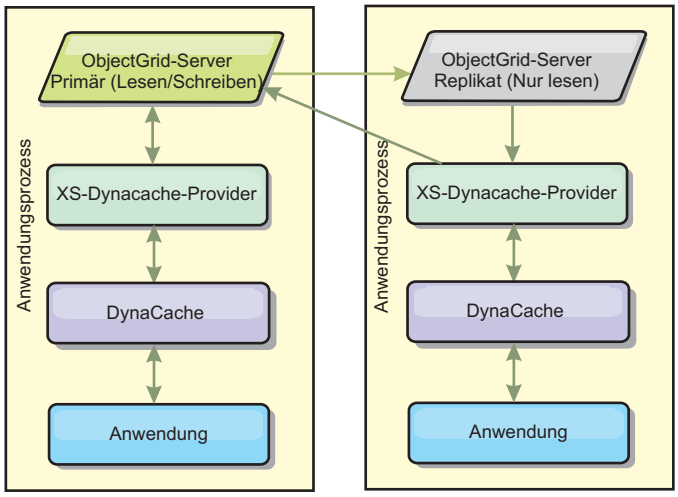
Topologietypen

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 spezielle an Ihren Leistungs-, Ressourcen-und Verwaltungsbedarf anzupassen. Diese Topologien sind die integrierte, die integriert, partitionierte Topologie und die ferne Topologie.

Integrierte Topologie

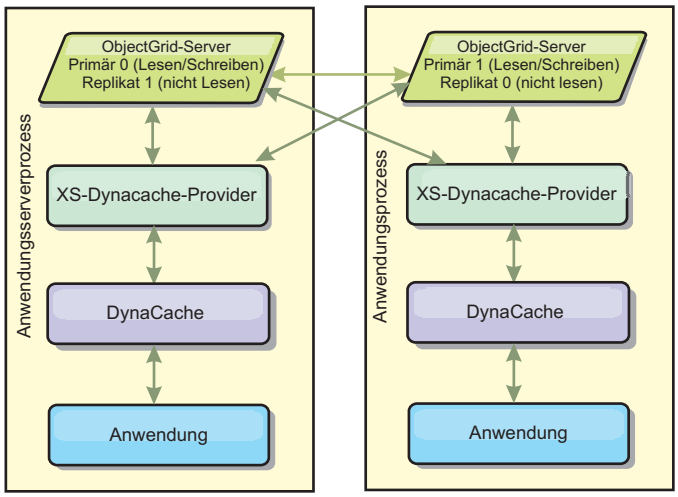
Die integrierte Topologie ist dem dynamischen Standardcache- und DRS-Provider sehr ähnlich. Verteilte Cacheinstanzen, die mit der integrierten Topologie erstellt werden, enthalten eine vollständige Kopie des Caches in jedem Prozess von eXtreme Scale, der auf den dynamischen Cacheservice zugreift und 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

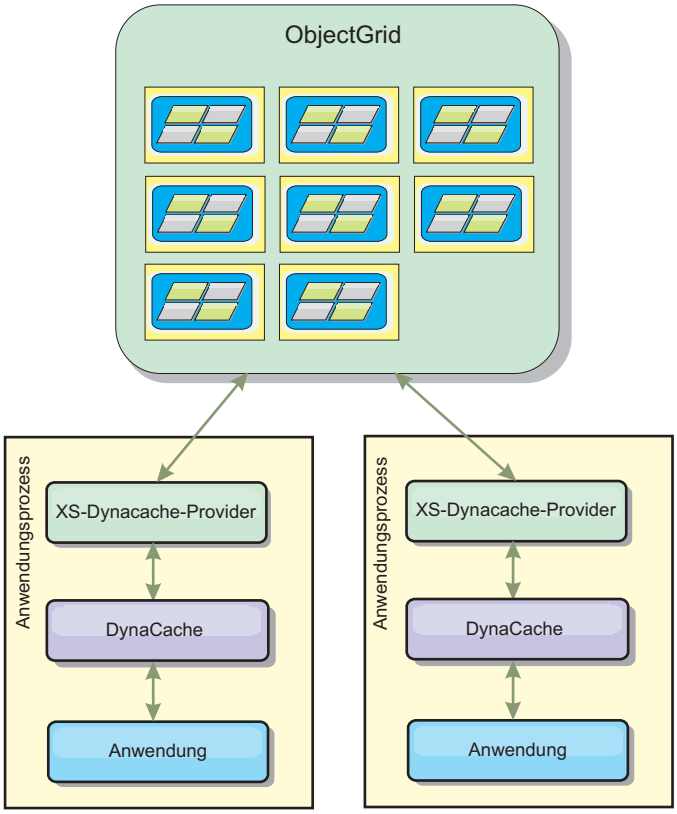
Für Workloads, bei denen die Anzahl der Schreiboperationen größer-gleich der Anzahl der Leseoperationen ist, wird die integrierte, partitionierte Topologie oder die ferne Topologie empfohlen. Bei der integrierten, partitionierten Topologie sind alle Cachedaten in den Prozessen von WebSphere Application 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 bemerk-

bar. 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 ″Cache Statistics″ 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 derzeit nicht unterstützt und gibt immer -1 zurück.
- **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 Server in einem Grid von WebSphere 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 integrierte, 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.

Sehen Sie sich die eXtreme-Scale-Dokumentation zur Sicherheit an, 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

Dynamischen Cacheprovider für WebSphere eXtreme Scale konfigurieren

Die Installation und Konfiguration des dynamischen Cacheproviders für eXtreme Scale richtet sich nach Ihren Anforderungen und Ihrer konfigurierten Umgebung.

Vorbereitungen

Installieren Sie den dynamischen Cacheprovider.

Zur Verwendung des dynamischen Cacheproviders muss WebSphere eXtreme Scale über die Knotenimplementierungen von WebSphere Application Server (einschließlich des Deployment-Manager-Knotens) installiert werden. Der dynamische Cacheprovider von eXtreme Scale wird in den folgenden Versionen von WebSphere Application Server unterstützt.

- WebSphere Application Server 6.1.0.25 + PK85622 und höher

- WebSphere Application Server 7.0.0.3 + PK85622 und höher

Installationsanweisungen finden Sie im Abschnitt Installation für 6.1 bzw. Installation für 7.0.

Warum und wann dieser Vorgang ausgeführt wird

Führen Sie die folgenden Schritte zum Konfigurieren des dynamischen Cacheproviders von eXtreme Scale aus:

1. Aktivieren Sie den dynamischen Cacheprovider von eXtreme Scale.

   In WebSphere Application Server 7.0 können Sie den dynamischen Cacheservice für die Verwendung des dynamischen Cacheproviders von eXtreme Scale über die Administrationskonsole oder mit einer angepassten Eigenschaft konfigurieren.

   Nach der Installation von eXtreme Scale ist der dynamische Cacheprovider von eXtreme Scale sofort als Option unter "Cacheprovider" in der Administrationskonsole verfügbar. Weitere Informationen finden Sie im Abschnitt Cacheserviceprovider auswählen.

   Sie können den dynamischen Cacheprovider von eXtreme Scale auch für eine Cacheinstanz konfigurieren, indem Sie die folgenden angepassten Eigenschaft/Wert-Paare in der Instanz festlegen. Diese angepassten Eigenschaften sind die einzige Möglichkeit, den Provider von eXtreme Scale in WebSphere Application Server 6.1 wie folgt zu aktivieren.

   ```
   com.ibm.ws.cache.CacheConfig.cacheProviderName =
      "com.ibm.ws.objectgrid.dynacache.CacheProviderImpl"
   ```

   Wenn Sie den dynamischen Cacheprovider von eXtreme Scale für das Caching von JSPs, Web-Services oder Befehlen verwenden möchten, müssen Sie die baseCache-Instanz für die Verwendung des dynamischen Cacheproviders von eXtreme Scale setzen. Dieselben Konfigurationseigenschaften werden zum Konfigurieren der baseCache-Instanz verwendet. Denken Sie auch daran, dass diese Konfigurationseigenschaften als angepasste JVM-Eigenschaften definiert werden müssen. Diese Voraussetzung gilt für alle Cachekonfigurationseigenschaften, die in diesem Abschnitt beschrieben werden, mit Ausnahme des Servlet-Cachings. Wenn Sie eXtreme Scale mit dem dynamischen Cacheprovider für das Servlet-Caching verwenden möchten, müssen Sie die Unterstützung in den Systemeigenschaften und nicht in den angepassten Eigenschaften konfigurieren.

   Wenn die baseCache-Instanz für die Verwendung des dynamischen Cacheproviders von eXtreme Scale konfiguriert ist, verwenden alle anderen Cacheinstanzen im Server den Provider von eXtreme Scale standardmäßig. Damit eine Cacheinstanz den dynamischen Standardcacheprovider verwendet, müssen Sie die Eigenschaft des Cacheproviders wie folgt setzen: com.ibm.ws.cache.CacheConfig.cacheProviderName = "default". Alle Konfigurationseigenschaften des dynamischen Cacheproviders von eXtreme Scale, die für die baseCache-Instanz definiert werden, sind die Standardkonfigurationseigenschaften für alle Cacheinstanzen, die von eXtreme Scale gestützt werden. Kunden müssen sehr vorsichtig sein, wenn sie sich für diese Eigenschaften auf die Standardwerte stützen.

2. Konfigurieren Sie die Replikationseinstellung für den Cache.

   Mit dem dynamischen Cacheprovider von eXtreme Scale ist es möglich, lokale Cacheinstanzen und replizierte Cacheinstanzen zu verwenden. Bei lokalen Cacheinstanzen ist keine weitere Konfiguration erforderlich, und der Rest dieses Abschnitts kann übersprungen werden.

   Es gibt zwei Methoden für die Erstellung replizierter Caches. Die erste Methode ist die Aktivierung der Cachereplikation über die Administrations-

konsole. Diese Aktivierung kann in WebSphere Application Server Version 7.0 jederzeit vorgenommen werden, setzt in WebSphere Application Server Version 6.1 aber die Erstellung einer DRS-Replikationsdomäne voraus.

Die zweite Methode ist die Verwendung des folgenden Eigenschaft/Wert-Paars, mit dem der Cache gezwungen wird zu melden, dass er ein replizierter Cache ist, auch wenn ihm keine DRS-Replikationsdomäne zugeordnet ist.

```
com.ibm.ws.cache.CacheConfig.enableCacheReplication = "true"
```

3. Konfigurieren Sie die Topologie für den dynamischen Cacheservice.

   Der einzige erforderliche Konfigurationsparameter für den dynamischen Cacheprovider von eXtreme Scale ist die Cachetopologie. Die folgende Variable muss als angepasste Eigenschaft im dynamischen Cacheservice definiert werden:

   ```
   com.ibm.websphere.xs.dynacache.topology
   ```

   Im Folgenden sehen Sie die drei gültigen Werte für diese Eigenschaft:

   - embedded
   - embedded_partitioned
   - remote

   Sie müssen einen der zulässigen Werte verwenden.

4. Konfigurieren Sie die Anzahl der Anfangscontainer für den dynamischen Cacheservice.

   Sie können die Leistung von Caches, die die integrierte partitionierte Topologie verwenden, maximieren, indem Sie die Anzahl der Anfangscontainer konfigurieren. Die folgende Variable muss als Systemeigenschaft in der Java Virtual Machine von WebSphere Application Server definiert werden:

   ```
   com.ibm.websphere.xs.dynacache.num_initial_containers
   ```

   Der empfohlene Wert für diese Konfigurationseigenschaft ist eine ganze Zahl, die der Gesamtanzahl der Instanzen von WebSphere Application Server entspricht, die auf diese verteilte Cacheinstanz zugreifen, bzw. geringfügig darunter liegt. Wenn ein dynamischer Cacheservice von Grid-Membern gemeinsam genutzt wird, muss die Variable auf die Anzahl der Grid-Member gesetzt werden.

5. Konfigurieren Sie das Katalogservice-Grid von eXtreme Scale.

   Wenn Sie eXtreme Scale als dynamischen Cacheprovider für eine verteilte Cacheinstanz verwenden, müssen Sie ein Katalogservice-Grid von eXtreme Scale konfigurieren.

   Ein einzelnes Katalogservice-Grid kann mehrere dynamische Cacheserviceprovider bedienen, die von eXtreme Scale gestützt werden.

   Ein Katalogservice kann innerhalb und außerhalb von Prozessen von WebSphere Application Server ausgeführt werden.

   Wenn Sie ein Katalogservice-Grid ausführen, müssen Sie die angepasste Eigenschaft **catalog.services.cluster** für die Katalogserviceendpunkte definieren.

6. Konfigurieren Sie angepasste Schlüsselobjekte.

   Wenn Sie angepasste Objekte als Schlüssel verwenden, müssen die Objekte die Schnittstelle ″Serializable″ oder ″Externalizable″ implementieren. Wenn Sie die integrierte oder integrierte partitionierte Topologie verwenden, müssen Sie die Objekte in den gemeinsam genutzten Bibliothekspfad von WebSphere stellen, wie es auch der Fall ist, wenn sie mit dem dynamischen Standardcacheprovider verwendet werden. Weitere Einzelheiten finden Sie im Abschnitt Schnittstellen ″DistributedMap″ und ″DistributedObjectCache″ für den dynamischen Cache verwenden im Information Center von WebSphere Application Server Network Deployment.

Wenn Sie die ferne Topologie verwenden, müssen Sie die angepassten Schlüsselobjekte für die eigenständigen Container von eXtreme Scale in den Klassenpfad stellen.

7. Konfigurieren Sie eXtreme-Scale-Containerserver.

   Die zwischengespeicherten Daten werden in eXtreme-Scale-Containern gespeichert. Container können innerhalb und außerhalb von Prozessen von WebSphere Application Server ausgeführt werden. Der Provider von eXtreme Scale erstellt automatisch Container innerhalb des WebSphere-Prozesses, wenn Sie integrierte oder integriert partitionierte Topologien für eine Cacheinstanz verwenden. Für diese Topologien ist keine weitere Konfiguration erforderlich.

   Wenn Sie die ferne Topologie verwenden, müssen Sie eigenständige eXtreme-Scale-Container vor den Instanzen von WebSphere Application Server starten, die auf die Cacheinstanz zugreifen. Stellen Sie sicher, dass alle Container für einen bestimmten dynamischen Cacheservice auf dieselben Katalogserviceendpunkte zeigen.

   Die XML-Konfigurationsdateien für den eigenständigen Container des dynamischen Cacheproviders von eXtreme Scale befinden sich im Verzeichnis `<Installationsstammverzeichnis>/customLibraries/ObjectGrid/dynacache/etc` für Installationen in WebSphere Application Server bzw. im Verzeichnis `<Installationsstammverzeichnis>/ObjectGrid/dynacache/etc` für eigenständige Installationen. Die Dateien haben die Namen `dynacache-remote-objectgrid.xml` und `dynacache-remote-definition.xml`. Erstellen Sie Kopien dieser Dateien, die Sie bearbeiten und verwenden, wenn Sie eigenständige Container für den dynamischen Cacheprovider von eXtreme Scale starten. Der Parameter **numIntitialContainers** in der Datei **dynacache-remote-deployment.xml** muss entsprechend der Anzahl der ausgeführten Containerprozesse gesetzt werden.

   Anmerkung: Die Gruppe der Containerprozesse muss genügend freien Speicher haben, um alle dynamischen Cacheinstanzen zu bedienen, die für die ferne Topologie konfiguriert wurden. Alle Prozesse von WebSphere Application Server, die dieselben oder ähnliche Werte für **catalog.services.cluster** verwenden, müssen dieselbe Gruppe eigenständiger Container verwenden. Die Anzahl der Container und die Anzahl der Maschinen, auf denen sich die Container befinden, müssen entsprechend dimensioniert werden. Weitere Einzelheiten finden Sie im Abschnitt „Kapazitätsplanung und hohe Verfügbarkeit".

   Der folgende Code enthält einen Beispiel-UNIX®-Befehlszeileneintrag, der einen eigenständigen Container für den dynamischen Cacheprovider von eXtreme Scale startet:

   `startOgServer.sh` *container1* `-objectGridFile` *../dynacache/etc/dynacache-remote-objectgrid.xml* `-deploymentPolicyFile` *../dynacache/etc/dynacache-remote-deployment.xml* `-catalogServiceEndpoints` *MyServer1.company.com:2809*

Kapazitätsplanung und hohe Verfügbarkeit

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 ist von der verwendeten Topologie abhängig.

Wenn Sie die integrierte Topologie verwenden, ist die Cachegröße auf die Menge freien Speichers 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 = Feier Speicher pro Containerprozess
- C = Anzahl der Container
- R = Anzahl der Replikate
- M = Gesamtgröße des Caches

Bei einem Grid von WebSphere Network Deployment mit 256 MB verfügbarem Speicher in jedem Prozess und 4 Serverprozessen insgesamt könnten in einer Cacheinstanz über alle diese Server verteilt 512 Megabyte an 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 Containers/ (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 durch 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" auf Seite 73.

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 verfügbarer Hauptspeicher zum Speichern der Cachedaten für jeden einzelnen Prozess. 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, wird 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 bestimmten, 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 eine 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 ist es. 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 Gesamtgröße 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.

Dynamischen Cacheprovider optimieren

Der dynamische Cacheprovider von WebSphere eXtreme Scale unterstützt die folgenden Konfigurationsparameter für die Leistungsoptimierung.

Warum und wann dieser Vorgang ausgeführt wird

- **com.ibm.websphere.xs.dynacache.ignore_value_in_change_event**: Wenn Sie einen Listener für Änderungsereignisse beim dynamischen Cacheprovider registrieren und eine ChangeEvent-Instanz generieren, entstehen Kosten für die Entserialisierung des Cacheeintrags, so dass der Wert in das ChangeEvent gestellt werden kann. Die Entserialisierung des Cacheeintrags wird beim Generieren von CacheEvents übersprungen, wenn Sie diesen optionalen Parameter in der Cacheinstanz auf `true` setzen. Der zurückgegebene Wert ist null, wenn eine Entfernungsoperation durchgeführt wird, bzw. eine Bytefeldgruppe, die die serialisierte Form des Objekts enthält. InvalidationEvent-Instanzen bringen ähnliche Leistungseinbußen mit sich, die Sie verhindern können, indem Sie ″com.ibm.ws-.cache.CacheConfig.ignoreValueInInvalidationEvent″ auf `true` setzen.
- **com.ibm.websphere.xs.dynacache.disable_recursive_invalidate**: Das Ungültigmachen des dynamischen Caches ist standardmäßig rekursiv. Wenn der dynamische Cacheprovider von WebSphere eXtreme Scale verwendet wird, fallen zusätzliche Kosten für das rekursive Ungültigmachen von Einträgen in integrierten partitionierten und fernen Topologien an. Wenn ein Cache kein rekursive Ungültigmachen erfordert, setzen Sie diesen Parameter auf `true`, um die Kosten zu vermeiden.
- **com.ibm.websphere.xs.dynacache.enable_compression**: Der dynamische Cacheprovider von eXtreme Scale kann die Cacheeinträge im Speicher komprimieren, um die Cachedichte zu erhöhen. Diese Komprimierung bringt zwar zusätzliche Kosten für Lese- und Schreiboperationen mit sich, kann aber den Speicherbedarf für Anwendung wie Servlet-Caching erheblich verringern.

Kapitel 4. Skalierbarkeit

WebSphere eXtreme Scale ist über die Verwendung partitionierter Daten skalierbar und kann 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 Skalierungsfähigkeiten von eXtreme Scale horizontal 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.

Partitionierung

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.

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.

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.

Partitionen verwenden

Ein Grid kann viele, bei Bedarf sogar Tausende Partitionen haben. Ein Grid kann vertikal bis zum 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 potenzielle eine vertikale Skalierung von 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.

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. Es wird davon ausgegangen, dass sich alle untergeordneten Elemente in derselben Partition wie das Stammelement befinden. Die untergeordneten Entitäten in diesem Graphen 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

Es sind zwei Verteilungsstrategien in WebSphere eXtreme Scale verfügbar: feste Partitionsverteilung und containerbezogene Verteilung. Die Verteilungsstrategie hat Auswirkungen auf die Verteilung der Partitionen und Daten.

Feste Partitionsverteilung

Sie können die Verteilungsstrategie in der XML-Implementierungsrichtliniendatei 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 *Programming Guide*.

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.

Schnittstelle "PartitionableKey"

Wenn eine eXtreme-Scale-Konfiguration eine festgelegte Partitionsverteilungsstrategie verwendet, ist sie vom Hashing des Schlüssels in einer Partition abhängig, um den Wert einzufügen, abzurufen, zu aktualisieren oder zu entfernen. Die Methode "hashCode" wird für den Schlüssel aufgerufen und muss klar definiert sein, wenn ein angepasster Schlüssel erstellt wird. Für die Verwendung der Schnittstelle "PartitionableKey" wird jedoch eine andere Option verwendet. Mit der Schnittstelle "PartitionableKey" können Sie ein anderes Obejkt als den Schlüssel für das Hashing in einer Partition verwenden.

Sie können die Schnittstelle "PartitionableKey" verwenden, wenn mehrere Maps vorhanden sind und die Daten, die Sie festschreiben, zusammengehören und deshalb in derselben Partition gespeichert werden sollten. WebSphere eXtreme Scale unterstützt keine zweiphasige Festschreibung. Deshalb sollten nicht mehrere Map-Transaktionen festgeschrieben werden, wenn sie sich auf mehrere Partitionen erstrecken. Wenn PartitionableKey das Hashing für Schlüssel in verschiedenen Maps in demselben MapSet in derselben Partition durchführt, können sie zusammen festgeschrieben werden.

Sie können die Schnittstelle "PartitionableKey" auch verwenden, wenn Gruppen von Schlüsseln in derselben Partition gespeichert werden sollten, aber nicht unbedingt in einer einzigen Transaktion. Wenn das Hashing von Schlüssel auf der Basis von Position, Abteilung, Domänentyp oder einem anderen Typ von Kennung durchgeführt werden soll, können Sie untergeordneten Schlüsseln einen übergeordneten PartitionableKey zuordnen.

Das Hashing für Mitarbeiter sollte beispielsweise in derselben Partition wie bei der zugehörigen Abteilung durchgeführt werden. Jeder Mitarbeiter hat ein PartitionableKey-Objekt, das zur Map "department" gehört. In diesem Fall wird für das Hashing des Mitarbeiters und der Abteilung dieselbe Partition verwendet.

Die Schnittstelle "PartitionableKey" stellt eine Methode bereit, die Methode "ibmGetPartition". Das von dieser Methode zurückgegebene Objekt muss die Methode

″hashCode″ implementieren. Das von der alternativen Methode ″hashCode″ zurückgegebene Objekt wird verwendet, um den Schlüssel an eine Partition weiterzuleiten.

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 Grid-übergreifenden Transaktionen für die horizontale Skalierung mit einem verteilten, hoch verfügbaren Objektcache wie WebSphere eXtreme Scale 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 Thema an. Stellen Sie sich zunächst vor, wie diese Transaktionen mit einem normalisierten Datenmodell arbeiten, das einen Benutzerdatensatz, einen Themendatensatz und einen Veröffentlichungs-

datensatz mit dem eigentlichen Text enthält. Wenn Veröffentlichungen mit Benutzerdatensätzen partitioniert werden, wird aus der Anzeige des Themas eine Grid-übergreifende Transaktion und umgekehrt. Themen 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 Themendatensatz und eine Kopie mit dem Benutzerdatensatz gespeichert wird. Die anschließende Anzeige der Veröffentlichungen eines Benutzers ist eine Einzelpartitionstransaktion, die Anzeige der Veröffentlichungen unter einem Thema 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 Thema 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 Thema 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, die Änderung am Thema erkennt und den 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

Szenarios 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 (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 Einzel-

partitionsoperation, 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.

Kapitel 5. Verfügbarkeit

Mit hoher Verfügbarkeit unterstützt WebSphere eXtreme Scale Redundanz 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 „Architektur und Topologie" auf Seite 9.

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 116 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 das Katalogservice-Grid 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 1:N-Services sind, der Positionsservice und die Verwaltung hingegen überall ausgeführt werden. 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 76.

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

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 von 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 in primäre Shard geladen, und der Prozess scheitert bei Datensatz 800.000. Normalerweise löscht das als neues 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 6. 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.

Tabelle 6. Statuswert und Antwort (Forts.)

Zurückgegebener Statuswert	Antwort von eXtreme Scale
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 Replikation 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 7. 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 8. 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, bis die nächste primäre Transaktion den Festschreibungsstatus dieser Transaktion sendet. Erst dann werden die Transaktionen im Replikat festgeschrieben oder rückgängig gemacht. Bis dahin sind die Transaktionen nicht festgeschrieben. Sie können einen Zeitgeber in dem primären Shard hinzufügen, das das Transaktionsergebnis nach kurzer Zeit (ein paar Sekunden) sendet. Dieser Zeitgeber verringert, schließt aber veraltete 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 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.

Replikationsarchitektur

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. Zum Durchführen der Replikation sind ein primäres Shard und Replikat-Shards für jede Partition vorhanden. 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. Ein asynchrones Replikat hält jedoch die Reihenfolge der Transaktionen ein, in der sie vom primären Shard gesendet wurden.

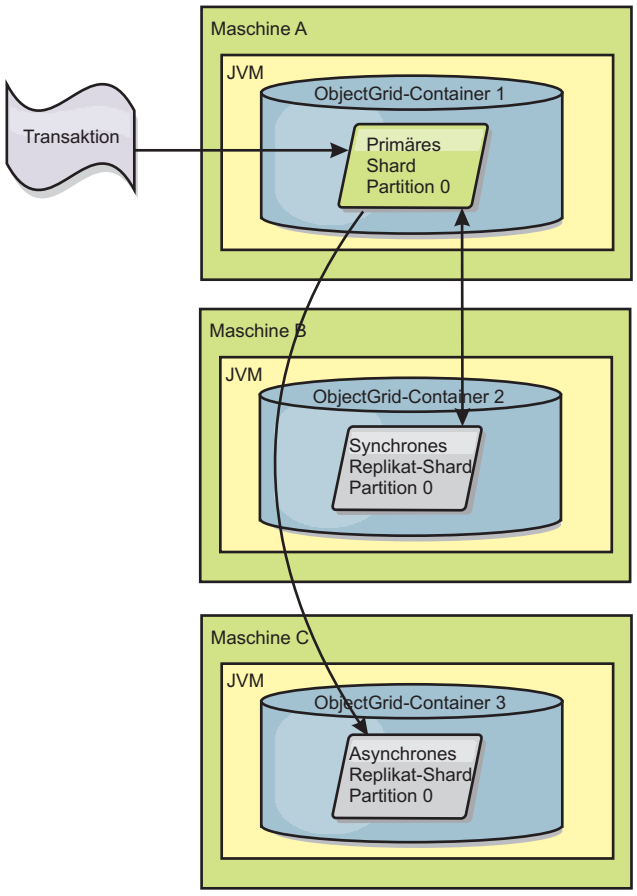


Abbildung 33. Kommunikationspfad zwischen einem primären Shard und einem Replikat-Shard

Speicherkosten der Replikation

Um ein Replikat online zu bringen, muss eine Prüfpunkt-Map erstellt werden, um vorhandene Daten vom primären Shard zu kopieren. Die Hauptspeicherkosten auf dem primären Shard entstehen jedoch durch den Speicher für jeden einzelnen Eintrag, der nach dem Prüfpunkt geändert wird. Wenn die Daten nach einem Prüfpunkt häufig geändert werden, sind die Speicherkosten für diese Operation proportional zur Anzahl der geänderten Einträge. Nach dem Kopieren des Prüfpunkts in das Replikat werden die Änderungen zurück in den Prüfpunkt eingefügt. Die geänderten Einträge werden erst dann freigegeben, wenn sie an an alle erforderlichen Replikate gesendet wurden.

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 Platzierung von Shards verantwortlich. Jedes ObjectGrid hat eine Reihe von Partitionen. Jede Partition hat ein primäres Shard und einen optionalen Satz an Replikat-Shards. Replikat-Shards und primäre Shards für eine Partition werden vom Katalogservice nicht in demselben Container platziert. Außerdem platziert der Katalogservice Replikat-Shards und primäre Shards nur dann in Containern, die dieselbe IP-Adresse haben, wenn sich die Konfiguration im Entwicklungsmodus befindet. Der Katalogservice verteilt die Shard-Typen 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 Skalierungsfunktionalität - horizontal wie vertikal.

Horizontale Skalierung

Horizontale Skalierung 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 Skalierung.

Vertikale Skalierung

Vertikale Skalierung 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 das 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.

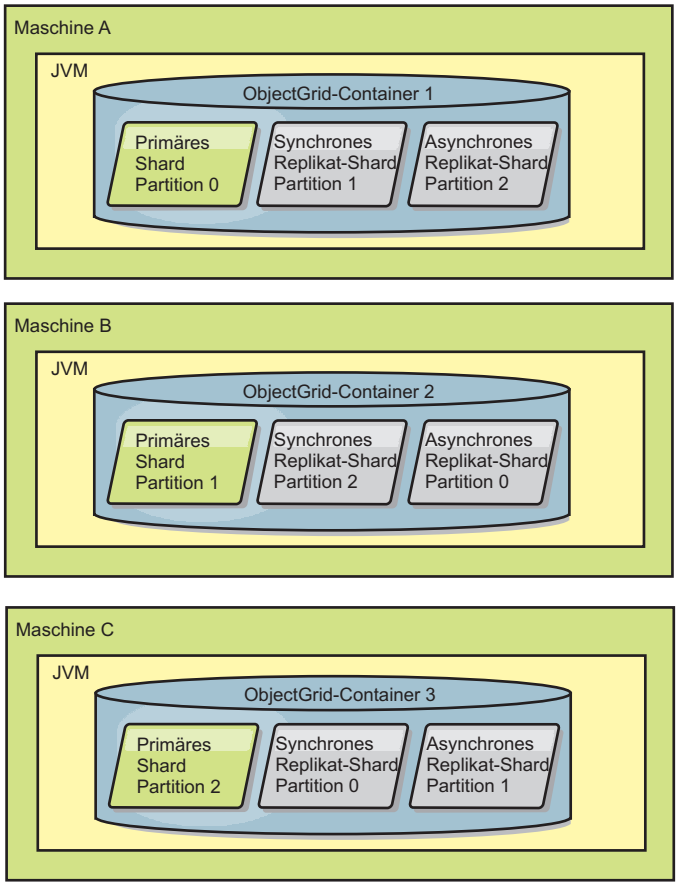


Abbildung 34. Verteilung eines ObjectGrid-MapSets über eine Implementierungsrichtlinie mit 3 Partitionen mit einem minSyncReplicas-Wert von 1, einem maxSyncReplicas-Wert von 1 und einem maxAsyncReplicas-Wert von 1

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 98.

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.
```

Wenn das 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.

Um eine Kopie der Daten im primären Shard zu erhalten, durchlaufen synchrone und asynchrone Replikat-Shards dieselbe Startreihenfolge. Während der Registrierung des Replikat-Shards erstellt das primäre Shard einen Prüfpunkt der aktuellen Daten. Die Daten im primären Shard haben einen so genannten Copy-on-Write-Status (Erstellen einer Kopie beim Schreiben). Die aktuellen Daten, die an das Repli-

kat-Shard übertragen werden, werden nie geändert, aber sobald eine neue Transaktion Datensätze im primären Shard ändert, wird eine Kopie beim Schreiben erstellt. Das primäre Shard kann die Verarbeitung fortsetzen, ohne die Daten, die an das Replikat-Shard übertragen werden, zu ändern. Das primäre Shard verwaltet eine Liste der Änderungen, da seit dem Prüfpunkt vorgenommen wurden.

Die Prüfpunktdaten werden mit Push an das Replikat übertragen. Wenn die Prüfpunktdaten beim Replikat ankommen, wird der Speicher für den Prüfpunkt freigegeben. Die seit dem Prüfpunkt vorgenommenen Änderungen werden zusammengeführt. Auch die Liste der Änderungen wird mit Push an das Replikat-Shard übertragen. Nach der Übertragung der Änderungen mit Push wird der Speicher für die Änderungen freigegeben.

Nach Abschluss seiner Prüfpunktphase wechselt das Replikat-Shard in den Peer-Modus und empfängt Daten, sobald das primäre Shard die Daten empfängt. Von diesem Zeitpunkt an verhält sich das Replikat-Shard entweder als synchrones oder asynchrones Replikat-Shard.

Wenn ein Replikat-Shard in den Peer-Modus wechselt wird eine Nachricht für das Replikat in der Datei `SystemOut.log` ausgegeben.

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

Die angegebene Zeit bezieht sich auf die Zeit, die das Replikat-Shard benötigt hat, um seine Anfangsdaten vom primären Shard zu erhalten, einschließlich der Prüfpunktdaten und aller zusätzlichen Änderungen, die während der Erstellung der Prüfpunktkopie vorgenommen wurden. Die angezeigte Zeit kann null oder sehr gering sein, wenn das primäre Shard keine zu replizierenden Daten enthält.

Synchrones Replikat-Shard

Wenn das neue Replikat ein synchrones Replikat-Shard ist, startet das neue Replikat eine Anforderung oder Antwort, sobald eine Transaktion im primären Shard festgeschrieben wird. Das primäre Shard wartet, bis das Replikat-Shard antwortet, dass es die Daten erhalten hat. Das synchrone Replikat-Shard hat dieselbe Version der Daten wie das primäre Shard.

Asynchrones Replikat-Shard

Wenn das neue Replikat ein asynchrones Replikat-Shard ist, sendet das primäre Shard die Daten an das Replikat, wartet aber nicht auf eine Antwort. Das asynchrone Replikat sortiert und wendet die vom primären Shard gesendeten Daten an. Es ist nicht gewährleistet, dass die Datenversion des asynchronen Shards mit der des primären Shards übereinstimmt.

Peer-Modus für alle Replikat-Shards

Wenn Sie im Peer-Modus arbeiten und alle Replikat-Shards eine Transaktionsänderung empfangen, wird der Speicher für die Transaktion im primären Shard freigegeben. Die Replikat-Shards empfangen nur bei Transaktionen, die Daten ändern, wie z. B. Einfüge-, Aktualisierungs- oder Entfernungsoperationen, Daten vom primären Shard. Es wird keine Verbindung zum Lesen der Daten im primären Shard zu den Replikat-Shards hergestellt.

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. 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. Wenn ein asynchrones Replikat-Shard im Rückstand ist, erhält es eine aktuelle Kopie der Daten und Register.

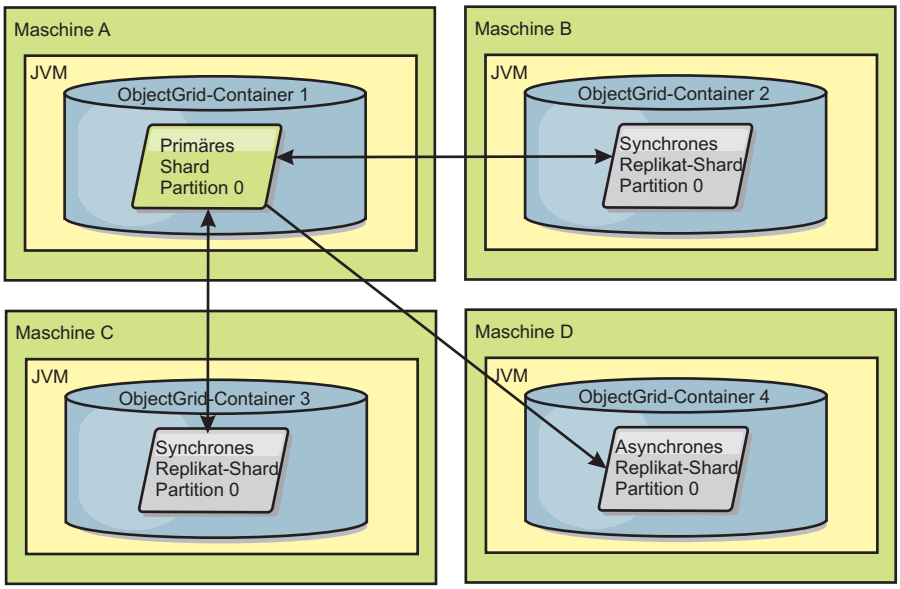


Abbildung 35. Beispielverteilung eines ObjectGrid-MapSets für die Partition "partition0". Die Implementierungsrichtlinie enthält den minSyncReplicas-Wert 1, den maxSyncReplicas-Wert 2 und den maxAsyncReplicas-Wert 1.

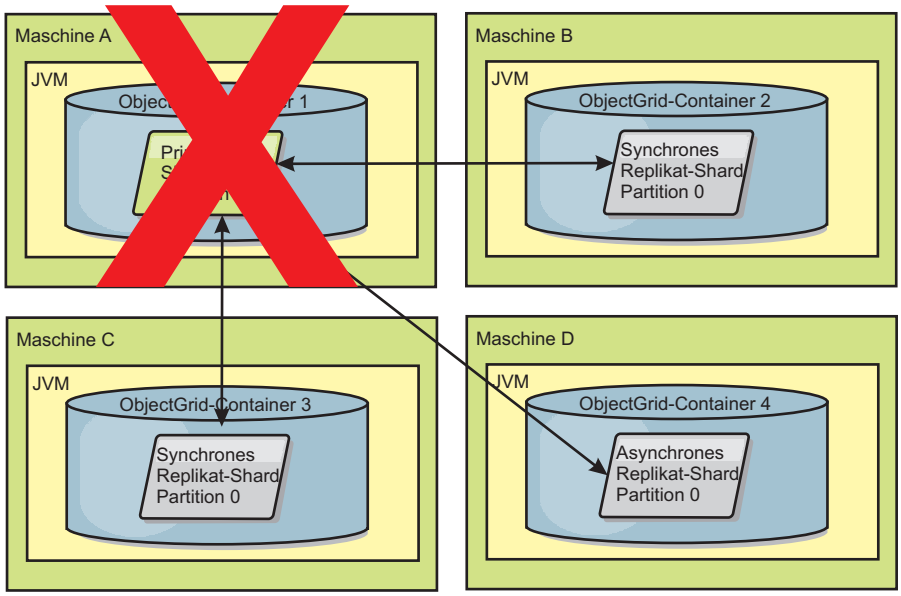


Abbildung 36. Container für das primäre Shard fällt aus

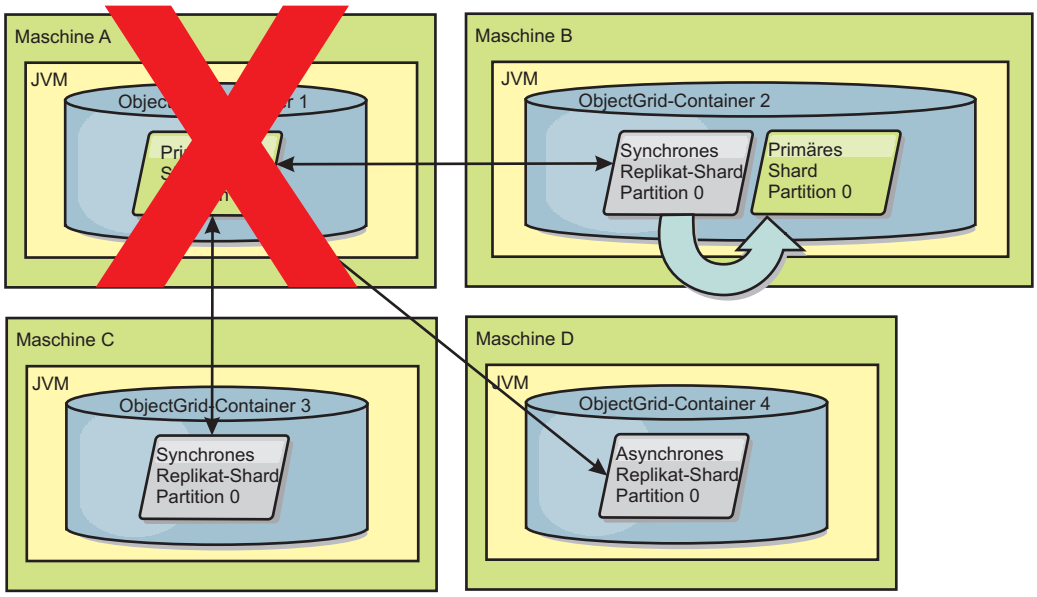


Abbildung 37. Synchrones Replikat-Shard in ObjectGrid-Container 2 wird zum primären Shard

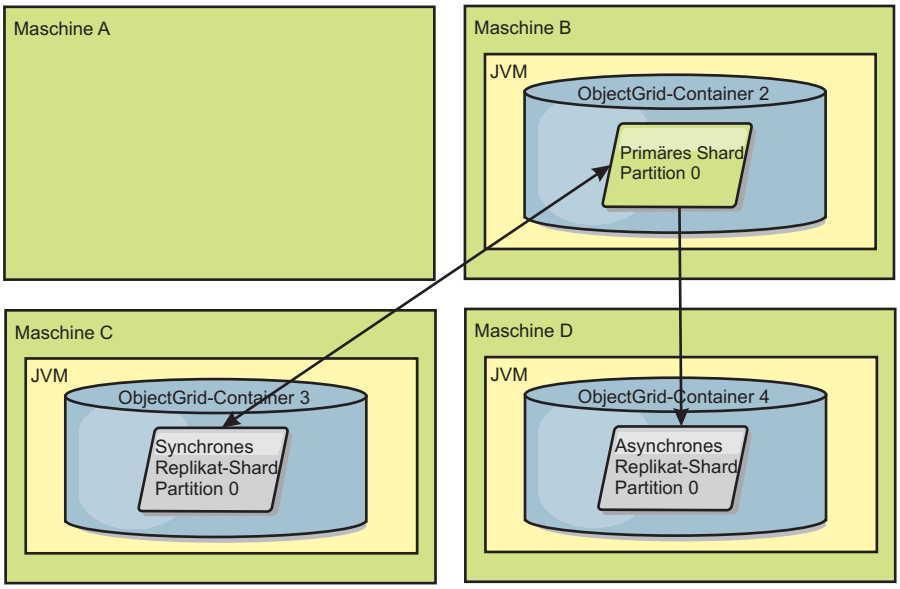


Abbildung 38. Maschine B enthält das primäre Shard. In Abhängigkeit von der Einstellung des Modus für automatische Reparatur und der Verfügbarkeit der Container wird ein neues synchrones Replikat-Shard auf einer Maschine erstellt oder nicht.

Wiederherstellung von Replikat-Shards

Ein 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, wenne in 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.

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″ in der Map-Set auf ″true″ gesetzt wird. Die Standardeinstellung ist ″false″.

Weitere Informationen zum Element ″MapSet″ finden Sie im Abschnitt zur XML-Implementierungsrichtliniendeskriptordatei 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.

Zonen für die Verteilung von Replikaten verwenden

Die Zonenunterstützung ermöglicht fortgeschrittene Konfigurationen für die Verteilung von Shards auf Rechenzentren. Mit dieser Funktionalität können Grid mit Tausenden von Partitionen ohne großen Aufwand mit einer Handvoll optionaler Verteilungsregeln verwaltet werden. Ein Rechenzentrum kann auf verschiedene Stockwerke eines Gebäudes, verschiedene Gebäude oder selbst verschiedene Städte verteilt sein. Die Unterscheidungsmerkmale werden über Zonenregeln konfiguriert.

Flexibilität von Zonen

Sie können Shards auf Zonen verteilen. Diese Funktion gibt Ihnen mehr Kontrolle darüber, wie eXtreme Scale Shards in einem Grid verteilt. Java Virtual Machines, die einen eXtreme-Scale-Server enthalten, können mit einer Zonen-ID gekennzeichnet werden. Die Implementierungsdatei kann jetzt eine oder mehrere Zonenregeln enthalten, und diese Zonenregeln werden einem Shard-Typ zugeordnet. Am besten lässt sich dies anhand einer Reihe von Beispielen mit anschließenden Zusatzinformation erklären.

Verteilungszonen steuern, wie eXtreme Scale primäre Shards und Replikate zuordnet, um erweiterte Topologie zu konfigurieren.

Eine Java Virtual Machine kann mehrere aktive eXtreme-Scale-Server haben. Ein Container kann mehrere Shards einer einzigen eXtreme-Scale-Instanz enthalten.

Diese Funktionalität ist hilfreich, um sicherzustellen, dass Replikate und primäre Shards für eine höhere Verfügbarkeit auf unterschiedliche Positionen oder Zonen verteilt werden. Normalerweise verteilt eXtreme Scale ein primäres Shard und ein Replikat-Shard nicht an Java Virtual Machines mit derselben IP-Adresse. Diese einfache Regel verhindert gewöhnlich, dass zwei eXtreme-Scale-Server auf demselben physischen Computer platziert werden. Möglicherweise benötigen Sie jedoch einen flexibleren Mechanismus. Sie verwenden beispielsweise zwei Blade-Gehäuse und möchten, das die primären Shards *einheitenübergreifend* auf beide Gehäuse verteilt werden und dass das Replikat jedes primären Shards nicht in demselben Gehäuse platziert wird wie das zugehörige primäre Shard.

Einheitenübergreifend verteilen bedeutet, dass die primären Shards in jeweils einer Zone und die zugehörigen Replikate in der jeweils anderen Zone platziert werden. Das primäre Shard 0 wird beispielsweise in Zone A und das synchrone Replikat 0 in Zone B platziert. Das primäre Shard 1 wird in Zone B und das synchrone Replikat 1 wird in Zone A platziert.

Der Gehäusename ist in diesem Fall der Zonenname. Alternativ können Sie Zonen nach den Stockwerken in einem Gebäude benennen und Zonen verwenden, um sicherzustellen, dass primäre Shards und Replikate derselben Daten auf unterschiedlichen Stockwerken gespeichert werden. Gebäude und Rechenzentren kön-

nen ebenfalls verwendet werden. Es wurden Tests mit Zonen in Rechenzentren durchgeführt, um sicherzustellen, dass die Daten ordnungsgemäß zwischen den Rechenzentren repliziert werden. Wenn Sie den HTTP-Sitzungsmanager für eXtreme Scale verwenden, können Sie ebenfalls Zonen verwenden. Mit diesem Feature können Sie eine einzelne Webanwendung auf drei Rechenzentren verteilen und sicherstellen, dass HTTP-Sitzungen für Benutzer in den Rechenzentren repliziert werden, so dass die Sitzungen wiederhergestellt werden können, falls ein komplettes Rechenzentrum ausfällt.

WebSphere eXtreme Scale kennt den Bedarf, ein großes Grid über mehrere Datenzentren hinweg zu verwalten. Das Produkt kann sicherstellen, dass Sicherungen und primäre Shards für dieselbe Partition bei Bedarf auf unterschiedliche Rechenzentren verteilt werden. Es kann alle primären Shards im Rechenzentrum 1 platzieren und alle Replikate im Rechenzentrum 2, oder es kann die primären Shards und Replikate im Umlaufverfahren auf beide Datenzentren verteilen. Die Regeln sind so flexibel, dass zahlreiche Szenarios möglich sind. eXtreme Scale kann auch Tausende von Servern verwalten. Diese Funktionalität, kombiniert mit der vollständig automatischen Verteilung unter Berücksichtigung von Rechenzentren macht solch große Grids aus Verwaltungssicht kosteneffizient. Administratoren können ohne großen Aufwand und effizient festlegen, was sie möchten.

Als Administrator verwenden Sie Verteilungszonen, um zu steuern, wo primäre Shards und Replikat-Shards platziert werden. Auf diese Weise können fortgeschrittene Topologien mit hoher Leistung und hoher Verfügbarkeit konfiguriert werden. Wie bereits erwähnt, können Sie eine Zone für jede logische Gruppierung von eXtreme-Scale-Prozessen definieren. Diese Zonen können physischen Workstation-Standorten wie Rechenzentren, Stockwerken eines Rechenzentrums oder Blade-Gehäusen entsprechen. Sie können Daten einheitenübergreifend auf Zonen verteilen, was Ihnen eine höhere Verfügbarkeit bietet, oder Sie können die primären Shards und Replikate auf verschiedene Zonen aufteilen, wenn ein fehlertoleranter Modus erforderlich ist.

eXtreme-Scale-Server einer Zone zuordnen, die nicht WebSphere Extended Deployment verwendet

Wenn eXtreme Scale mit Java Standard Edition oder einem Anwendungsserver verwendet wird, der nicht auf WebSphere Extended Deployment Version 6.1 basiert, kann eine JVM, die ein Shard-Container ist, mit den folgenden Verfahren einer Zone zugeordnet werden.

Anwendungen, die das Script ″startOgServer″ verwenden

Das Script ″startOgServer″ wird verwendet, um eine eXtreme-Scale-Anwendung zu starten, wenn diese nicht in einen vorhandenen Server integriert ist. Mit dem Parameter **-zone** wird die Zone angegeben, die für alle Container im Server verwendet werden soll.

Zone beim Starten eines Containers über APIs angeben

Knoten von WebSphere Extended Deployment Zonen zuordnen

Wenn Sie eXtreme Scale mit Java-EE-Anwendungen (Java Platform, Enterprise Edition) von WebSphere Extended Deployment verwenden, können Sie das Knotengruppenfeature von WebSphere Extended Deployment verwenden, um die JVMs der Cluster-Member automatisch an bestimmte Zonen zu verteilen. Eine JVM kann nur zu einer einzigen Zone gehören. Server, die auf einem Knoten ausgeführt wer-

den, der zu einer solchen Knotengruppe gehört, werden in die Zone eingeschlossen, die über den Namen der Knotengruppe angegeben wird. Sie müssen sicherstellen, dass diese Cluster-Member nicht zu zwei oder mehr solcher Knotengruppen gehören. Die JVM eines Cluster-Members prüft die Zonenzugehörigkeit nur beim Start. Das Hinzufügen einer neuen Knotengruppe und das Ändern der Zugehörigkeit haben nur Auswirkungen auf neu gestartete bzw. erneut gestartete Java Virtual Machines.

Zonenregeln

Eine eXtreme-Scale-Partition hat ein einziges primäres Shard und kein oder mehrere Replikat-Shards. In diesem Beispiel wird die folgende Namenskonvention für diese Shards verwendet: `P` ist das primäre Shard, `S` ist ein synchrones Replikat und `A` ein asynchrones Replikat. Eine Zonenregeln besteht aus drei Komponenten:

- Regelname,
- Zonenliste,
- Attribut inclusive oder exclusive.

Eine Zonenregel gibt die gültige Gruppe von Zonen an, an die ein Shard verteilt werden kann. Das Attribut ″inclusive″ zeigt an, dass nach der Verteilung eines Shards an eine Zone aus der Liste alle anderen Shards ebenfalls an diese Zone verteilt werden. Die Einstellung ″exclusive″ zeigt an, dass jedes Shard für eine Partition an eine jeweils andere Zone aus der Zonenliste verteilt wird. Die Verwendung der Einstellung ″exclusive″ bedeutet beispielsweise, dass die Zonenliste bei drei Shards (primäres Shard und zwei synchrone Replikate) drei Zonen enthalten muss.

Jedem Shard kann eine einzige Zonenregel zugeordnet werden. Eine Zonenregel kann von zwei Shards gemeinsam verwendet werden. Wenn eine Regel gemeinsam genutzt wird, gilt das Attribut ″inclusive″ bzw. ″exclusive″ für die Shards aller Typen, die eine Regel gemeinsam nutzen.

Beispiele

Es folgen diverse Beispiele, die verschiedene Szenarios und die entsprechende Konfiguration für die Implementierung des jeweiligen Szenarios veranschaulichen.

Primäre Shards und Replikate einheitenübergreifend auf Zonen verteilen

Sie haben drei Blade-Gehäuse und möchten, dass die primären Shards auf alle drei Gehäuse verteilt werden und dass jeweils ein einziges Replikat auf einem jeweils anderen Gehäuse als das zugehörige primäre Shard platziert wird. Definieren Sie jedes Gehäuse als Zone mit den Gehäusenamen `ALPHA`, `BETA` und `GAMMA`. Im Folgenden sehen Sie die zugehörige Implementierungs-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="library">
   <mapSet name="ms1" numberOfPartitions="37" minSyncReplicas="1"
maxSyncReplicas="1" maxAsyncReplicas="0">
   <map ref="book" />
   <zoneMetadata>
    <shardMapping shard="P" zoneRuleRef="stripeZone"/>
    <shardMapping shard="S" zoneRuleRef="stripeZone"/>
    <zoneRule name ="stripeZone" exclusivePlacement="true" >
     <zone name="ALPHA" />
     <zone name="BETA" />
     <zone name="GAMMA" />
    </zoneRule>
   </zoneMetadata>
  </mapSet>
 </objectgridDeployment>
</deploymentPolicy>
```

Diese Implementierungs-XML enthält ein Grid mit dem Namen ″library″ mit einer einzigen Map mit dem Namen ″book″. Es werden vier Partitionen mit einem einzigen synchronen Replikat verwendet. Die Klausel für die Zonenmetadaten zeigt die Definition einer einzigen Zonenregel und die Zuordnung von Zonenregeln zu Shards. Die primären und synchronen Shards werden der Zonenregel ″stripeZone″ zugeordnet. Die Zonenregel umfasst alle drei Zonen und verwendet eine exklusive Verteilung. Diese Regel bedeutet Folgendes: Wenn das primäre Shard für die Partition 0 in Zone `ALPHA` platziert wird, dann wird das Replikat für die Partition 0 in Zone `BETA` oder `GAMMA` platziert. Die primären Shards für andere Partitionen werden in anderen Zonen platziert und die zugehörigen Replikate dann in einer jeweils anderen als das entsprechende primäre Shard.

Asynchrones Replikat in einer anderen Zone als das primäre Shard und das synchrone Replikat

In diesem Beispiel gibt es zwei Gebäude, die über eine Verbindung mit hoher Latenzzeit miteinander verbunden sind. In allen Szenarios möchten Sie eine hohe Verfügbarkeit, um einen Datenverlust zu vermeiden. Der Einfluss der synchronen Replikation zwischen den Gebäuden auf die Leistung drängt Sie jedoch zu einem Kompromiss. Sie möchten ein primäres Shard mit einem synchronen Replikat in einem Gebäude und ein asynchrones Replikat in einem anderen Gebäude. Normalerweise sind Ausfälle auf JVM-Abstürze oder Computerausfälle und nicht auf Probleme mit der Größe zurückzuführen. Mit dieser Topologie können Sie normale Ausfälle ohne Datenverlust bewältigen. Der Verlust eines Gebäudes tritt so selten auf, dass in diesem Fall ein gewisser Datenverlust akzeptabel ist. Sie können zwei Zonen erstellen, eine für jedes Gebäude. Im Folgenden sehen Sie die zugehörige XML-Implementierungsdatei:

```
<?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="library">
  <mapSet name="ms1" numberOfPartitions="13" minSyncReplicas="1"
   maxSyncReplicas="1" maxAsyncReplicas="1">
   <map ref="book" />
   <zoneMetadata>
    <shardMapping shard="P" zoneRuleRef="primarySync"/>
    <shardMapping shard="S" zoneRuleRef="primarySync"/>
    <shardMapping shard="A" zoneRuleRef="aysnc"/>
    <zoneRule name ="primarySync" exclusivePlacement="false" >
      <zone name="BldA" />
      <zone name="BldB" />
    </zoneRule>
    <zoneRule name="aysnc" exclusivePlacement="true">
      <zone name="BldA" />
      <zone name="BldB" />
    </zoneRule>
   </zoneMetadata>
  </mapSet>
 </objectgridDeployment>
</deploymentPolicy>
```

Das primäre Shard und das synchrone Replikat-Shard verwenden beide die Zonenregel ″primaySync″ mit der Einstellung ″false″ für das Attribut ″exclusive″. Wenn das primäre Shard oder das synchrone Replikat in einer Zone platziert wird, wird deshalb das jeweils andere Shard in derselben Zone platziert. Das asynchrone Replikat verwendet eine zweite Zonenregel mit denselben Zonen wie die Zonenregel ″primarySync″, verwendet aber die Einstellung ″true″ für das Attribut **exclusivePlacement**. Dieses Attribut gibt an, dass ein Shard nicht zusammen mit einem anderen Shard aus derselben Partition in einer Zone platziert werden kann. Deshalb wird das asynchrone Replikat nicht in derselben Zone platziert wie das primäre Shard bzw. das synchrone Replikat.

Alle primären Shards an eine Zone und alle Replikate an eine andere Zone verteilen

In diesem Szenario befinden sich alle primären Shards in einer bestimmten Zone und alle Replikate in einer anderen Zone. Es gibt ein primäres Shard und ein einziges asynchrones Replikat. Alle Replikate befinden sich in Zone A und alle primären Shards in Zone B.

```
<?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="library">
   <mapSet name="ms1" numberOfPartitions="13" minSyncReplicas="0"
    maxSyncReplicas="0" maxAsyncReplicas="1">
    <map ref="book" />
    <zoneMetadata>
     <shardMapping shard="P" zoneRuleRef="primaryRule"/>
     <shardMapping shard="A" zoneRuleRef="replicaRule"/>
     <zoneRule name ="primaryRule">
      <zone name="A" />
     </zoneRule>
     <zoneRule name="replicaRule">
      <zone name="B" />
       </zoneRule>
      </zoneMetadata>
     </mapSet>
   </objectgridDeployment>
 </deploymentPolicy>
```

In diesem Beispiel sehen Sie zwei Regeln, eine für die primären Shards (P) und eine andere für die Replikate (A).

Zonen über Weitverkehrsnetze (WANs)

Sie können eine einzige Instanz von eXtreme Scale über mehrere Gebäude oder Rechenzentren hinweg mit langsameren Netzverbindungen untereinander implementieren. Langsamere Netzverbindungen führen zu einer geringeren Bandbreite und zu einer höheren Latenzzeit. Das Risiko von Netzpartitionen ist in diesem Modus aufgrund der Netzüberlastung und anderen Faktoren ebenfalls erhöht. eXtreme Scale nähert sich diesen ungeeigneten Umgebungsbedingungen auf die folgenden Arten.

Begrenzter Austausch von Überwachungssignalen zwischen Zonen

Java Virtual Machines, die in Stammgruppen zusammengefasst sind, tauschen Überwachungssignale miteinander aus. Wenn der Katalogservice die Java Virtual Machines in Gruppen organisiert, können sich diese Gruppen nicht über mehrere Zonen erstrecken. Ein führendes Member dieser Gruppe überträgt die Zugehörigkeitsdaten mit Push an den Katalogservice. Der Katalogservice prüft alle gemeldeten Fehler, bevor er Maßnahmen ergreift. Dazu versucht er, eine Verbindung zu den fehlerverdächtigen Java Virtual Machines herzustellen. Wenn der Katalogservice eine falsche Fehlererkennung feststellt, ergreift er keine Maßnahmen, das die Stammgruppenpartition in kurzer Zeit wieder repariert ist.

Außerdem sendet der Katalogservice in regelmäßigen Abständen Überwachungssignale an die führenden Member jeder Stammgruppe, um Fälle von Stammgruppenisolation zu behandeln.

Katalogservice als Tiebreaker im Grid

Der Katalogservice ist der Tiebreaker für ein eXtreme-Scale-Grid. Es ist von entscheidender Bedeutung, dass der Katalogservice mit einer einzigen Stimme auftritt,

damit das Grid ausgeführt werden kann. Der Katalogservice wird in einer festen Gruppe von Java Virtual Machines ausgeführt und repliziert die Daten vom ausgewählten primären Shard in allen anderen Java Virtual Machines dieser Gruppe. Der Katalogservice muss auf die physischen Zonen oder Rechenzentren verteilt werden, um das Risiko zu verringern, dass er vom Grid isoliert wird, und dafür zu sorgen, dass er bei erwarteten Fehlerszenarios aktiv bleibt.

Der Katalogservice kommuniziert mit den Container-JVMs im Grid über idempotente oder wiederherstellbare Operationen. Für die Kommunikation wird IIOP verwendet. Alle Statusänderungen im Katalogservice werden in den aktuellen Membern, in denen der Katalogservice ausgeführt wird, repliziert. Diese Replikation ist nur erfolgreich, wenn die Mehrheit der Java Virtual Machines die Änderung akzeptieren. Das bedeutet, dass bei einer Partitionierung des Katalogservice nur die Partition mit der Mehrzahl der Stimmen Änderungen festschreiben kann. Das primäre Shard des Service sendet nur dann Befehle an die Container-JVMs, wenn die Statusänderung, die diesen Befehl erstellt, festgeschrieben wird. Das bedeutet, dass eine Minderheitspartition ihren Status nicht aktualisieren und keine Befehle an Container absetzen kann.

Ein partitionierter Katalogservice stoppt das Grid nicht. Das Grid akzeptiert weiterhin Clientanforderungen und führt Operationen durch. Wenn es keine Katalogservicepartition gibt, die eine Stimmenmehrheit hat, werden Ausfälle im Grid erst dann behoben, wenn der Katalogservice wieder die Mehrheit besitzt. Verzögert sich die Wiederherstellung nach Ausfällen übermäßig lange, gehen bestimmte Partitionen offline, bis der Katalogservice wieder die Mehrheit hat.

Die führenden Member-JVMs der Stammgruppe berichten Zugehörigkeitsänderungen an den Katalogservice. Wenn der Katalogservice partitioniert wird, überträgt der Service eine aktualisierte Routentabelle für den Katalogservice mit Push. Eine solche Routentabelle einer Minderheitspartition enthält nicht die Position einer primären Partition. Die führende Member-JVM muss über alle möglichen Katalogservice-JVMs iterieren, um die primäre Partition zu finden. Sie muss dies in regelmäßigen Abständen tun, während Sie auf die Aufhebung der Partitionierung wartet. Nach dem Empfang einer Routentabelle mit einer primären Partition werden die offenen Wiederherstellungsaktionen von der primären Mehrheitspartition des Katalogservers weitergeleitet.

Wenn die Stammgruppe eine bestimmte Zeit lang keine Verbindung zum primären Katalogservice herstellen kann, wird sie physisch vom restlichen Grid (unter Umständen mit einer Minderheitspartition des Katalogservice) getrennt, oder der Katalogservice bleibt in Minderheitspartitionen blockiert. Der Unterschied ist nicht zu erkennen. Wenn es eine Mehrheitspartition des Katalogservice gibt, wird möglicherweise gerade eine Wiederherstellung nach einem Verlust der getrennten Stammgruppe durchgeführt. Dies kann zu zwei primären Exemplaren derselben Partition führen, der alten vorhandenen primären Partition und der neuen primären Partition im restlichen Netz. Die Mehrheitspartition des Katalogservice hat keine Möglichkeit, die alten primären Partitionen zu beenden, weil sie über das Netz von den alten primären Partitionen getrennt ist. Denn der Katalogservice wiederhergestellt wird und die getrennte Stammgruppe die neuen primären Partitionen erkennt, stellt der Katalogservice fest, dass es zwei primäre Partitionen gibt. Er weist die zuvor getrennten Stammgruppen an, alle Shards zu löschen, und es findet eine Neuverteilung statt.

Wenn der Katalogservice in zwei Minderheitspartitionen oder eine einzige aktive Minderheitspartition partitioniert wird, muss der Benutzer in die Wiederherstellung eingreifen. Damit Maßnahmen ergriffen werden können, muss ein JMX-Befehl

(Java Management Extensions) abgesetzt werden, um anzugeben, dass eine einzige Minderheitspartition zulässig ist. Sie müssen sicherstellen, dass die anderen Minderheitspartitionen gestoppt werden.

Routing an bevorzugte Zonen

Mit dem Routing an bevorzugte Zonen kann eXtreme Scale Transaktionen auf der Basis Ihrer Präferenzangaben an Zonen weiterleiten.

WebSphere eXtreme Scale bietet Ihnen sehr viele Steuerungsmöglichkeiten in Bezug auf die Verteilung der Shards eines ObjectGrids. Informationen zu einigen grundlegenden Szenarios und zum Konfigurieren der entsprechenden Implementierungsrichtlinien finden Sie im Abschnitt „Zonen für die Verteilung von Replikaten verwenden" auf Seite 106.

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. WebSphere eXtreme Scale arbeitet bei der Partitionsverteilung standardmäßig mit festen Partitionen. 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:

-
  ```
  startOgServer.sh s1 -objectGridFile ../xml/universeGrid.xml
  -deploymentPolicyFile ../xml/universeDp.xml
  -catalogServiceEndpoints MyServer1.company.com:2809
  -zone Chicago
  ```
-
  ```
  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. Weitere Informationen finden Sie unter ″Knoten von WebSphere Extended Deployment Zonen zuordnen″ im Abschnitt „Zonen für die Verteilung von Replikaten verwenden” auf Seite 106.

Primäre Shards für die Zone ″Chicago″ haben Replikate in der Zone ″London″. Primäre Shards in der Zone ″London″ haben Replikate in der Zone ″Chicago″.

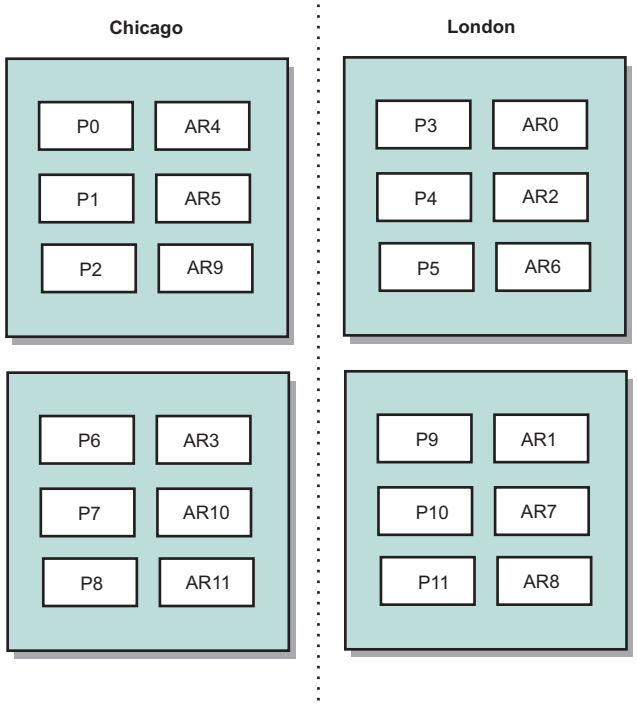


Abbildung 39. Primäre Shards und Replikate in Zonen

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 im Abschnitt 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 ″Session.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.

Fehlererkennungstypen

WebSphere eXtreme Scale verwendet zwei Methoden für die Fehlererkennung.

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 behandelt.

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. 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.

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. Im Abschnitt „Fehlererkennungstypen” auf Seite 116 können Sie nachlesen, wie Sie die Zeit für die Erkennung von permanenten Ausfällen verringern können.

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 das Katalogservice-Grid 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 1:N-Services sind, der Positionsservice und die Verwaltung hingegen überall ausgeführt werden. 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.

Katalogservice mit hoher Verfügbarkeit

Ein Katalogservice ist das Grid von Katalogservern, die Sie verwenden und die Topologieinformationen zu allen Containern in Ihrer Umgebung von eXtreme Scale bewahren. Der Katalogservice steuert den Lastausgleich und das Routing für alle Clients. Zum Implementieren von eXtreme Scale als speicherinternen Verarbeitungsbereich müssen Sie den Katalogservice für die hohe Verfügbarkeit als Cluster in ein Grid einfügen.

Komponenten des Katalogservice

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 das Katalogserver-Grid über den CORBA-Servicekontext (Common Object Request Broker Architecture) an die Clients weitergegeben.

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

Wenn ein Server und ein Container von eXtreme Scale einen Kontakt zu einem der Katalogserver herstellen, wird die Routing-Tabelle für das Katalogserver-Grid ebenfalls über den CORBA-Servicekontext an den Server und Container von eXtreme Scale weitergegeben. Ist der kontaktierte Katalogserver derzeit nicht der Master-

katalogserver, 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. Das Katalogserver-Grid 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 das Katalogserver-Grid 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 des Katalog-Grids

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, was dazu führt, dass 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 Servergruppe. 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. Er wird als ein über eine 1:N-Beziehung ausgewählter Service im Grid ausgeführt, so dass stets eine einzige 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 eines Katalog-Grids werden die Member des Grids miteinander verbunden. Planen Sie die Topologie Ihres Katalog-Grids sorgfältig, weil die Katalogkonfiguration zur Laufzeit nicht geändert werden kann. Verteilen Sie das Grid so divers wie möglich, um Fehler zu verhindern.

Katalogserver-Grid starten

Container von eXtreme Scale, die in WebSphere Application Server integriert sind, zu einem eigenständigen Katalog-Grid verbinden

Sie können Container von eXtreme Scale, die in eine Umgebung von WebSphere Application Server integriert sind, zu einem eigenständigen Katalog-Grid verbinden. Verwenden Sie dieselbe Eigenschaft, um den Anwendungsserver mit dem Katalog-Grid zu verbinden. Die Eigenschaft verwaltet jedoch nicht den Lebenszyklus des Katalogs mit den Servern. Stattdessen ermöglicht die Eigenschaft den Containern, das ferne Katalog-Grid zu finden. Weitere Informationen finden Sie in .

Anmerkung: Kollision beim Servernamen: 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.

Katalogserver-Quorum

Das Quorum ist die minimale Anzahl an Katalogservern, die erforderlich sind, um Verteilungsoperationen für das Grid durchzuführen. Die minimale Anzahl ist die vollständige Gruppe der Katalogserver, sofern das Quorum nicht außer Kraft gesetzt wurde.

Wichtige Begriffe

Im Folgenden finden Sie eine Liste der Begriffe, die sich auf Quorumaspekte für 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 Ort 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. Server in einem Rechenzentrum können beispielsweise alle als zu einr Zone gehörig gekennzeichnet werden.
- Überwachungssignal: Ein Überwachungssignal ist ein mit Ping an eine Java Virtual Machine (JVM) abgesetztes Signal, um dessen Aktivitätsstatus zu bestimmen.

Topologie

In diesem Abschnitt wird erläutert, wie IBM 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 sendet aggressiv Überwachungssignale an die JVMs in einer einzelnen Zone, und ein nicht beantwortetes Überwachungssignal führt zu einem Failover-Ereignis, solange der Katalogsservice das Quorum hat.

Katalogservice-Grid und Containerserver

Ein Grid ist eine Sammlung ähnlicher JVMs. Ein Katalogservice ist ein Grid, das sich aus Katalogservern zusammensetzt und eine festgelegte 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.

Das Katalogservice-Grid verwendet einen Mechanismus mit vollständigem Quorum. Das bedeutet, dass alle Member des 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

Catalogserverstart

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 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 heruntergefahren 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/startOgServer 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 vewrendet zwei Methoden, um den Ausfall ihrer Member zu erkennen. Wenn der Socket einer JVM geschlossen wird, wird diese JVM als inaktiv betrachtet. 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 betrachtet.

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 Stammgruppenzugehörigekeiten zu melden. Eine Zugehörigkeitsänderung kann beispielsweise der Ausfall JVM oder eine neu hinzugefügte JVM sein, die in die Stammgruppe aufgenommen wird.

Wenn das führende Stammgruppen-Member zu einem der Member des Ansprechpartners keine Verbindung herstellen kann, setzt er seine Verbindungsversuche fort.

Austausch von Überwachungssignalen im Katalogservice-Grid

Der Katalogservice gleicht einer privaten Stammgruppe mit einer statischen Zugehörigkeit und einem Quorum-Mechanismus. Er 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 für xsadmin-Abfragen nicht sichtbar. Es werden Nachrichten in den Protokollen der Container-JVM aufgezeichnet, die Informationen zu diesen Vorkommnissen enthalten. Diese JVMs müssen manuell erneut gestartet werden.

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. Das Katalogservice-Grid 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 einen Quorum-Verlust aus den folgenden Gründen:

- das JVM-Member des Katalogservice fällt aus,
- Netz-Brownout,
- Ausfall des Rechenzentrums.

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 wird 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, fällt 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 so lange keinen Zugriff auf die primären oder Replikatkopien der Daten, bis 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 primäre 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 Szenarios 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 Brownout, nach der C dem Katalogservice-Grid 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 dem Katalogservice-Grid 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 gestrartet werden. Anschließend tritt C dem vorhandenen Katalogservice-Grid wieder bei, und das Quorum wird ohne Benutzereingriff auf die normale Situation zurückgesetzt.
- 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 einem 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 für 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 erneut mit dem primären Shard synchronisiert.

Clientverhalten

Unabhängig von, ob das Katalogservice-Grid 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 einer 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.

SSL inaktiviert

Es wird dringend empfohlen, in der ObjectGrid-XML-Deskriptordatei ein Transaktionszeitlimit anzugeben. Der Client wiederholt eine einzelne Aktion so lange, bis dieses Zeitlimit überschritten wird. Wenn Sie kein Zeitlimit angeben, setzt der Client seine Versuche unbegrenzt fort. Dies wird nicht empfohlen, da der Client-Thread in diesem Fall für eine unbestimmte Zeit blockiert wird. Das Transaktionszeitlimit muss auf ein Vielfaches der maximal erwarteten Transaktionsdauer gesetzt werden.

Außerdem wird dringend empfohlen, in der Datei "orb.properties" das ORB-Anforderungszeitlimit festzulegen. Der Client wird an einem Socket, an dem eine Brownout-Bedingung aufgetreten ist, so lange blockiert, bis diese maximale Zeit abgelaufen ist. Wenn die seit dem Senden der Anforderung abgelaufene Zeit immer noch kleiner ist als das Transaktionszeitlimit, wiederholt WebSphere eXtreme Scale die Anforderung. Der Client setzt seine Versuche so lange fort, bis

die insgesamt abgelaufene Zeit das Transaktionszeitlimit überschreitet. Deshalb ist die maximale Blockierungsdauer des Clients die Summe aus Transaktionszeitlimit und ORB-Anforderungszeitlimit.

Der ORB wird so lange blockiert, bis der TCP-Stack den Socket, bei dem die Brownout-Bedingung aufgetreten ist, schließt, sofern kein ORB-Anforderungszeitlimit angegeben ist. Diese Zeit richtet sich nach der Optimierung des TCP-Stacks, und je nach Einstellung des Parameters TCP FIN_WAIT und anderer zugehöriger Parameter kann es sich hierbei um Stunden handeln.

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 bestätigt, dass 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, hat die Hash-Funktion in den Schlüsselobjekten wahrscheinlich eine schlechte Verteilung.

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

JMS für die Verteilung von Transaktionsänderungen verwenden

Verwenden Sie Java Message Service (JMS) für die Verteilung von Änderungen 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 dersel-

ben 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

Administratorhandbuch

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 Änderun-

gen 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 Beschreibung der Verteilung von Änderungen zwischen Peer-JVMs im *Administratorhandbuch*.

Kapitel 6. Sicherheit

WebSphere eXtreme Scale ist ein verteiltes Caching-System. Wenn Sie den Zugriff zum Schutz Ihrer Daten sichern möchten, können Sie die Sicherheit aktivieren und externe Sicherheitsprovider integrieren.

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 basierten auf Subject-Objekten und Berechtigungen. Sie können Java Authentication and Authorization Services (JAAS) für die Berechtigung des Zugriffs verwenden, oder Sie können eine angepasste

Lösung 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 instanziert 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 7. Transaktionsverarbeitung

WebSphere eXtreme Scale verwendet Transaktionen als Mechanismus für die Interaktion mit Daten.

Sitzungen und Transaktionen

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.

Vorteile von Transaktionen

Wenn Sie Transaktionen verwenden, ist Folgendes möglich:

- Änderungen können rückgängig gemacht werden, wenn eine Ausnahme eintritt oder wenn die Geschäftslogik Statusänderungen widerrufen muss.
- 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.

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.

Transaktionsübersicht

Verwenden Sie Transaktionen für die folgenden Zwecke:

- zum Schutz eines Threads vor gleichzeitigen Änderungen,
- zum Anwenden mehrerer Änderungen als atomare Einheit beim Festschreiben,
- zum Implementieren eines Lebenzyklus für Sperren bei Änderungen,

- als Replikationseinheit.

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.

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.

Attribut "CopyMode"

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*.

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, unmittelbar nach der Rückkehr der Methode nicht mehr verwaltet werden und somit unbrauchbar sind.

Sperrmodus und Transaktionen

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" />
```

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*.

Sperrstrategien

Die folgenden Sperrstrategien sind verfügbar: PESSIMISTIC (pessimistisch), OPTIMISTIC (optimistisch) und NONE (Ohne). Zur Auswahl einer Sperrstrategie müssen Sie wissen, wie viel Prozent auf jeden Typ von Operation fällt, ob ein Loader verwendet wird usw.

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, aktualisiert, 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 141 beschrieben sind.

Kapitel 8. Lernprogramme

Sie können diese Lernprogramme als Einführung in bestimmte Funktionen von WebSphere eXtreme Scale verwenden.

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.

Vorbereitungen

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.

Warum und wann dieser Vorgang ausgeführt wird

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.

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 145 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.

Warum und wann dieser Vorgang ausgeführt wird

Abbildung 40. 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.

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.

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.

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.

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.

Vorbereitungen

Stellen Sie sicher, dass die Datei `objectgrid.jar` im Klassenpfad enthalten ist.

Warum und wann dieser Vorgang ausgeführt wird

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.

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 156
   - Vorgehensweise zum Erstellen eines Index, den die Abfrage verwenden kann
3. „Lernprogramm zu ObjectQuery - Schritt 3" auf Seite 156
   - 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 159
   - 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.

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 Maßnahme

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.

Vorbereitungen

Stellen Sie sicher, dass Sie die Anweisungen im Abschnitt „Lernprogramm zu ObjectQuery - Schritt 1" auf Seite 154 ausgeführt haben, bevor Sie mit diesem Schritt des Lernprogramms fortfahren.

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 nach dem Feld "itemName" filtert, verwendet die Abfragesteuerkomponente automatisch den Index. Auf diese Weise 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.

Vorbereitungen

Stellen Sie sicher, dass Sie die Anweisungen im Abschnitt „Lernprogramm zu ObjectQuery - Schritt 2″ auf Seite 156 ausgeführt haben, bevor Sie mit diesem Schritt fortfahren.

Warum und wann dieser Vorgang ausgeführt wird

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.

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 Maßnahme

Im Abschnitt „Lernprogramm zu ObjectQuery - Schritt 4" auf Seite 159 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.

Vorbereitungen

Stellen Sie sicher, dass Sie die Anweisungen im Abschnitt „Lernprogramm zu ObjectQuery - Schritt 3″ auf Seite 156 ausgeführt haben, bevor Sie mit diesem Schritt fortfahren.

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 - Hauptseite

Mit dem folgenden Lernprogramm können Sie eine verteilte eXtreme-Scale-Umgebung in einer Java-SE-Umgebung erstellen.

Vorbereitungen

Stellen Sie sicher, dass Sie mit den Grundlagen einer verteilten eXtreme-Scale-Konfiguration vertraut sind.

Warum und wann dieser Vorgang ausgeführt wird

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

1. „Lernprogramm zur Java-SE-Sicherheit - Schritt 1"
   - 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 164
   - 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 171
   - JAAS-Berechtigungsrichtlinie verwenden
4. „Lernprogramm zur Java-SE-Sicherheit - Schritt 4" auf Seite 175
   - 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.

Vorbereitungen

Anmerkung: Alle für diesen Schritt des Lernprogramms erforderlichen Dateien sind im folgenden Abschnitt beschrieben.

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":
   - startOgServer.sh catalogServer
   - 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":
   - startOgServer.sh c0 -objectGridFile ../xml/SimpleApp.xml -deploymentPolicyFile ../xml/SimpleDP.xml -catalogServiceEndpoints localhost:2809
   - 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:
  - xsadmin.sh -g accounting -m mapSet1 -mapSizes
  - 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.

```
stopOgServer.sh c0 -catalogServiceEndPoints localhost:2809
```

```
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.

```
stopOgServer.sh catalogServer -catalogServiceEndPoints localhost:2809
```

```
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.

Vorbereitungen

Stellen Sie sicher, dass Sie die Anweisungen im Abschnitt „Lernprogramm zur Java-SE-Sicherheit - Schritt 1" auf Seite 161 ausgeführt haben.

Warum und wann dieser Vorgang ausgeführt wird

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.

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″. Außerdem 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 Eigenschaft 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:

      -

        ```
        startOgServer.sh catalogServer -clusterSecurityFile ../security/security.xml
        -serverProps ../security/server.properties -jvmArgs
        -Djava.security.auth.login.config="../security/og_jaas.config"
        ```

      -

        ```
        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:

      -

        ```
        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"
        ```

      -

        ```
        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:
  - `xsadmin.sh -g accounting -m mapSet1 -username manager -password manager1 -mapSizes`
  - `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 Sicherheitskonfigurationsdatei 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:

- `stopOgServer.sh c0 -catalogServiceEndPoints localhost:2809 -clientSecurityFile ..\security\client.properties`
- `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.

- `stopOgServer.sh catalogServer -catalogServiceEndPoints localhost:2809 -clientSecurityFile ..\security\client.properties`

- `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.

Vorbereitungen

Stellen Sie sicher, dass Sie die Anweisungen im Abschnitt „Lernprogramm zur Java-SE-Sicherheit - Schritt 2" auf Seite 164 ausgeführt haben, bevor Sie mit dieser Task fortfahren.

Warum und wann dieser Vorgang ausgeführt wird

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 durch eXtreme-Scale-Berechtigung verschiedenen authentifizierten Benutzern 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. Eine vollständige Beschreibung der Berechtigungskategorien finden Sie in den Referenzinformationen zur Clientberechtigung.

In eXtreme Scale stellt die Klasse ″com.ibm.websphere.objectgrid.security.MapPermission″ Berechtigungen für die eXtreme-Scale-Ressourcen, insbesondere die Methoden der Schnittstellen ″ObjectMap″ und ″JavaMap″, dar. WebSphere eXtreme Scale definiert die folgenden Berechtigungszeichenfolgen für den Zugriff auf die Methoden der Schnittstellen ″ObjectMap″ und ″JavaMap″:

- 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. Die Laufzeitumgebung von eXtreme Scale überprüft die verschiedenen Map-Berechtigungen für die verschiedenen 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.

1. **eXtreme-Scale-Berechtigung aktivieren** Zum Aktivieren der Berechtigung in ObjectGrid müssen Sie ″securityEnabled″ für das jeweilige ObjectGrid in der XML-Datei auf ″true″ setzen. Sicherheit im ObjectGrid bedeutet Berechtigung. Verwenden Sie die folgenden Befehle, um eine neue ObjectGrid-XML mit aktivierter Sicherheit zu erstellen.
   a. cd ObjectGrid-Stammverzeichnis/bin
   b. cp SimpleApp.xml SecureSimpleApp.xml

   Fügen Sie ″securityEnabled=″true″″ auf ObjectGrid-Ebene hinzu, wie in der folgenden XML gezeigt:

```
<?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.** Wenn Sie sich erinnern, 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 ″og_auth.policy″ 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:

   - Die Codebasis ″http://www.ibm.com/com/ibm/ws/objectgrid/security/PrivilegedAction″ ist ein speziell für ObjectGrid reservierter URL. 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:

      - `startOgServer.sh catalogServer -clusterSecurityFile ../security/security.xml -serverProps ../security/server.properties -jvmArgs -Djava.security.auth.login.config="../security/og_jaas.config"`
      - `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.

      Anschließend können Sie einen sicheren Containerserver mit dem folgenden Script starten:

   c. Navigieren Sie erneut zum Verzeichnis ″bin″.

   d.

      - `# 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"`
      - `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:

   - 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.

Jetzt können Sie denselben Befehl wie im vorherigen Schritt des Lernprogramms verwenden:

a. Navigieren Sie wie zuvor 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 wieder 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 wird ausgegeben, 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*.

Nächster Schritt

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.

Vorbereitungen

Stellen Sie sicher, dass Sie die Anweisungen im Abschnitt „Lernprogramm zur Java-SE-Sicherheit - Schritt 3" auf Seite 171 ausgeführt haben, bevor Sie mit dieser Task fortfahren.

Warum und wann dieser Vorgang ausgeführt wird

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.

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 171.

   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:

      -
        ```
        startOgServer.sh catalogServer -clusterSecurityFile ../security/security.xml
        -serverProps ../security/server.properties -JMXServicePort 11001
        -jvmArgs -Djava.security.auth.login.config="../security/og_jaas.config"
        ```
      -
        ```
        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 164 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.

      -

```
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"
```

-
```
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 ″Simple-App.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:
  -
```
xsadmin.sh -g accounting -m mapSet1 -mapsizes -p 11001 -ssl
-trustpath ..\security\trust.jks -trustpass ogpass -trusttype jks
-username manager -password manager1
```
  -
```
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:

```
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"
```

```
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.

Kapitel 9. Glossar

Dieses Glossar enthält Begriffe und Definitionen für WebSphere eXtreme Scale.

In diesem Glossar werden die folgenden Querverweise verwendet:

1. 'Siehe' verweist von einem Terminus zu einem bevorzugt zu verwendenden Synonym oder von einem Akronym oder einer Abkürzung zu der Definition der vollständigen Langform des Terminus.
2. 'Siehe auch' verweist auf einen zugehörigen Terminus oder auf ein Antonym.

Wenn Sie Glossare zu anderen IBM Produkten anzeigen möchten, dann rufen Sie die Website www.ibm.com/software/globalization/terminology auf.

Abfrage.

1. Eine Anforderung von Informationen aus einer Datenbank, die auf bestimmten Bedingungen basiert, z. B. eine Anforderung einer Liste aller Kunden in einer Kundentabelle, deren Kontostand höher als 1000 Euro ist.

2. Eine wiederverwendbare Informationsanforderung über ein oder mehr Modellelemente.

Ableitung. In der objektorientierten Programmierung die Verbesserung oder Erweiterung einer Klasse auf der Basis einer anderen Klasse.

Administrator. Eine Person, die für die Ausführung von Verwaltungstasks wie beispielsweise der Vergabe von Zugriffsberechtigungen und für das Content-Management verantwortlich ist. Administratoren sind auch für die Vergabe von Berechtigungsstufen an Benutzer zuständig.

Agent. Ein Programm, das eine Aktion für einen Benutzer oder ein anderes Programm ohne Benutzereingriff oder nach einem regelmäßigen Zeitplan ausführt und die Ergebnisse zurück an den Benutzer oder das Programm meldet.

Aktualisierbare Sperre. Eine Sperre, die die Absicht aufzeigt, einen Cacheeintrag zu aktualisieren, wenn pessimistisches Sperren verwendet wird.

Aktualisierungspaket. Eine kumulative Sammlung von Programmkorrekturen mit neuen Funktionen. Siehe auch Fixpack, Vorläufiger Fix.

Allgemein. Dieser Begriff bezeichnet das Anzeigen einer Gruppe von Objekten auf abstrakter oder höherer Ebene.

Angepasstes Installationspaket. Ein angepasstes Installationsimage, das mindestens ein Wartungspaket, eine Konfigurationsarchivierungsdatei aus einem eigenständigen Serverprofil, mindestens eine Unternehmensarchivdatei, Scripts und sonstige Dateien enthalten kann, die Unterstützung bei der Anpassung der daraus resultierenden Installation bieten.

Anwendung. Computerprogramme oder Softwarekomponenten, die eine Funktion für die direkte Unterstützung eines oder mehrerer Geschäftsprozesse bereitstellen.

Anwendungsprogrammierschnittstelle (API). Ein Schnittstelle, die einem in einer höheren Programmiersprache geschriebenen Anwendungsprogramm die Verwendung bestimmter Daten oder Funktionen des Betriebssystem oder eines anderen Programms ermöglicht.

Anwendungsserver. Ein Serverprogramm in einem verteilten Netz, das die Ausführungsumgebung für ein Anwendungsprogramm bereitstellt.

APAR. Siehe Authorized Program Analysis Report.

API. Siehe Anwendungsprogrammierschnittstelle (Application Programming Interface).

Arbeitsbereich.

1. Ein Verzeichnis auf Platte, das alle Projektdateien sowie Informationen wie Benutzervorgaben enthält.

2. Ein temporäres Repository für Konfigurationsdaten, das von Clients mit Verwaltungsfunktionen verwendet wird.

3. In Eclipse die Sammlung der Projekte und anderen Ressourcen, die der Benutzer momentan in der Workbench entwickelt. Metadaten zu diesen Ressourcen befinden sich in einem Verzeichnis im Dateisystem. Die Ressourcen können sich in demselben Verzeichnis befinden.

Asynchron. Dieser Begriff bezeichnet Ereignisse, die zeitlich nicht synchronisiert sind oder nicht in regelmäßigen oder voraussagbaren Zeitabständen auftreten.

Asynchrones Messaging. Eine Methode der Kommunikation zwischen Programmen, bei der das Programm eine Nachricht in eine Nachrichtenwarteschlange stellt und dann mit seinen eigenen Anforderungen fortfährt, ohne auf eine Antwort auf die Nachricht zu warten.

Asynchrones Replikat. Ein Shard, das Aktualisierungen nach der Festschreibung der Transaktion empfängt. Diese Methode ist schneller als ein synchrones Replikat, birgt aber das Risiko eines Datenverlusts, weil das asynchrone Replikat aus mehreren Transaktionen hinter dem primären Shard bestehen kann.

Aufruf. Die Aktivierung eines Programms oder einer Prozedur.

Ausdruck. Ein SQL- oder XQuery-Operand oder eine Sammlung von SQL- oder XQuery-Operatoren und -Operanden, die einen einzigen Wert ergibt.

Ausführungs-Trace. Eine Kette von Ereignissen, die in hierarchischem Format auf der Ereignisseite des Integrationstestclients aufgezeichnet und angezeigt wird.

Ausgeschriebener Name. Die Eigenschaft, die den logischen Namen für den Server auf der z/OS-Plattform angibt.

Auslösen. Bei der objektorientierten Programmierung das Veranlassen eines Statusübergangs.

Ausnahme. Eine Bedingung oder ein Ereignis, die bzw. das von einem normalen Prozess nicht verarbeitet werden kann.

Ausnahmebehandlungsroutine. Eine Gruppe von Routinen, die auf eine abnormale Bedingung reagieren. Eine Ausnahmebehandlungsroutine kann die normale Prozessausführung unterbrechen und auch wieder aufnehmen.

Ausstellerzertifikat. Der anerkannte Zertifikateintrag, der in der Regel in einer Truststore-Datei enthalten ist.

Authentifizierter Benutzer. Ein Portalbenutzer, der für die Anmeldung beim Portal einen gültigen Benutzereintrag (Benutzer-ID und Kennwort) verwendet hat. Authentifizierte Benutzer haben Zugriff auf alle öffentlichen Bereiche.

Authentifizierung. Ein Sicherheitsservice, der sicherstellt, dass ein Benutzer eines Computersystems wirklich die Person ist, die er zu sein vorgibt. Allgemeine Mechanismen für die Implementierung dieses Service sind Kennwörter und digitale Signaturen. Die Authentifizierung unterscheidet sich von der Berechtigung. Die Authentifizierung dient nicht zur Erteilung oder Verweigerung von Zugriff auf Systemressourcen.

Authentifizierungsalias. Ein Alias, mit dem die Berechtigung für den Zugriff auf Ressourcenadapter und Datenquellen gewährt wird. Ein Authentifizierungsalias enthält Authentifizierungsdaten einschließlich einer Benutzer-ID und eines Kennworts.

Authorized Program Analysis Report. Eine Anforderung zur Behebung eines Fehlers in einem unterstützten Release eines von IBM gelieferten Programms.

Automatische Erkennung. Die Erkennung von Serviceartefakten in einem Dateisystem, einer externen Registry oder in einer anderen Quelle.

Autonomic Manager. Eine Gruppe von Software- oder Hardwarekomponenten, die über Richtlinien konfiguriert werden und das Verhalten anderer Software- oder Hardwarekomponenten so verwalten, wie es auch ein Benutzer tun würde. Zu einem Autonomic Manager gehört ein Regelkreis, der aus Überwachungs-, Analyse-, Plan- und Ausführungskomponenten besteht.

Autorisierung/Berechtigung. Der Prozess, mit dem einem Benutzer, System oder Prozess entweder uneingeschränkter oder eingeschränkter Zugriff auf ein Objekt, eine Ressource oder eine Funktion gewährt wird.

Bean. Eine Definition oder Instanz einer JavaBeans-Komponente. Siehe auch JavaBeans, Enterprise-Bean.

Bean-Klasse. Bei der EJB-Programmierung (Enterprise JavaBeans) eine Java-Klasse, die eine javax.ejb.EntityBean- oder eine javax.ejb.SessionBean-Klasse implementiert.

Bean-managed Messaging. Über JavaBeans realisiertes Messaging. Eine Funktion des asynchronen Messaging, durch die eine Enterprise-Bean die vollständige Steuerung der Messaging-Infrastruktur erhält.

Bean-Managed Persistence (BMP) . Der Mechanismus, durch den die Datenübertragung zwischen den Variablen einer Entity-Bean und einem Ressourcenmanager von der Entity-Bean verwaltet wird. (Sun)

Bean-Managed Transaction (BMT) . Über JavaBeans realisierte Transaktion. Die Fähigkeit einer Session-Bean, eines Servlet oder einer Anwendungsclientkomponente, eigene Transaktionen direkt ohne Unterstützung eines Containers zu verwalten.

Bean Scripting Framework. Eine Architektur, mit der die Funktionen von Scripting-Sprachen in Java-Anwendungen eingebunden werden können.

Befehlszeile. Eine leere Zeile in der Anzeige, in der Befehle, Optionsnummern oder ausgewählte Optionen eingegeben werden können.

Berechtigung. Die Berechtigung zur Ausführung von Aktivitäten wie Lesen und Schreiben von lokalen Dateien, Erstellen von Netzverbindungen und Laden des nativen Codes.

Berechtigungsnachweis. In der JAAS-Architektur (Java Authentication and Authorization Service) eine Subjektklasse mit sicherheitsrelevanten Attributen. Diese Attribute können Informationen enthalten, die für die Authentifizierung des Subjekts für neue Services verwendet wird.

Berechtigungsrichtlinie. Eine Richtlinie, deren Richtlinienziel ein Geschäftsservice ist und deren Vertrag mindestens eine Zusicherung enthält, auf deren Basis die Berechtigung zur Ausführung einer Kanalaktion erteilt wird.

Berechtigungstabelle. Eine Tabelle, die die Informationen für die Zuordnung einer Rolle zu einem Benutzer oder zu einer Gruppe enthält, die den zulässigen Zugriff eines Clients auf eine bestimmte Ressource definieren.

Bereinigungsprogramm. Eine Komponente, die die Zugehörigkeit von Einträgen in jeder BackingMap-Instanz steuert. Teilcaches können Bereinigungsprogramme verwenden, um Daten ohne Beeinträchtigung der Datenbank automatisch aus dem Cache zu entfernen.

Bibliothek.

1. Eine Gruppe von Modellelementen einschließlich der zugehörigen Geschäftselemente, Prozesse, Tasks, Ressourcen und Organisationen.

2. Ein Projekt, das für Entwicklung, Versionsmanagement und Organisation gemeinsam genutzter Ressourcen verwendet wird. In einer Bibliothek kann nur ein Teil der Artefakttypen erstellt und gespeichert werden, z. B. Geschäftsobjekte und Schnittstellen.

Binärformat. Die Darstellung eines Dezimalwerts, bei der jedes Feld zwei oder vier Byte lang sein muss. Das Vorzeichen (+ oder -) wird im Feld in dem Bit ganz links angegeben, die restlichen Bit des Felds geben den Zahlenwert an. Positive Zahlen haben im Bit für das Vorzeichen eine 0 und werden im Standardformat (″True Format″) angegeben. Negative Zahlen haben im Bit für das Vorzeichen eine 1 und werden im Zweierkomplementformat (″Two's Complement Format″) angegeben.

Bindung im Zellenbereich. Ein Bindungsbereich, in dem die Bindung nicht spezifisch und nicht einem Knoten oder Server zugeordnet ist. Diese Art Namensbindung wird unter dem persistenten Ausgangskontext einer Zelle erstellt.

BMP. Siehe Bean-managed Persistence.

BMT. Siehe Bean-managed Transaction.

Boot-Programm. Ein kleines Programm, mit dem umfangreichere Programme während der Systeminitialisierung geladen werden können.

Bootstrapping. Der Prozess, bei dem eine Ausgangsreferenz des Namensservice abgerufen wird. Die Bootstrap-Einstellung und der Hostname bilden den Ausgangskontext für JNDI-Referenzen (Java Naming and Directory Interface).

Bottom-up-Entwicklung. Bei Web-Services der Prozess der Entwicklung eines Service auf der Basis eines vorhandenen Artefakts wie z. B. einer Java-Bean oder einer Enterprise-Bean und nicht auf der Basis einer WSDL-Datei (WSDL = Web Services Description Language).

Bytecode. Ein systemunabhängiger Code, der vom Java-Compiler generiert und vom Java-Interpreter ausgeführt wird. (Sun)

Cacheinstanzressource. Eine Position, unter der jede Java EE-Anwendung (Java EE = Java Platform Enterprise Edition) Daten speichern, verteilen und gemeinsam nutzen kann.

Cachereplikation. Die gemeinsame Nutzung von Cache-IDs, Cacheeinträgen und Cacheinvalidierungen mit anderen Servern innerhalb einer Replikationsdomäne.

CIP. Siehe Angepasstes Installationspaket (Customized Installation Package).

Client. Ein Softwareprogramm oder ein Computer, der Services von einem Server anfordert. Siehe auch Host.

Clientanwendung. Eine Anwendung, die auf einer Workstation ausgeführt wird und über eine Verbindung zu einem Client verfügt, über die die Anwendung auf die Warteschlangensteuerungsservices auf einem Server zugreifen kann.

Client/Server. Bezeichnung des Interaktionsmodells bei der verteilten Datenverarbeitung, in dem ein Programm auf einem Computer eine Anforderung an ein Programm auf einem anderen Computer sendet und die Antwort abwartet. Das anfordernde Programm ist ein Client und das antwortende Programm ein Server.

Cloudscape. Ein integrierbares, vollständig Java-gestütztes, objektrelationales Datenbankverwaltungssystem (ORDBMS, Object-Relational Database Management System).

Cluster. Eine Gruppe von Anwendungsservern, die zusammenarbeitet, um Funktionen wie Lastausgleich und Ausweichbetrieb bereitzustellen.

Command-Bean. Ein Proxy, der eine einzelne Operation mithilfe der Methode execute() aufrufen kann.

Containerserver. Eine Serverinstanz, die mehrere Shards haben kann. Eine Java Virtual Machine (JVM) kann mehrere Containerserver haben.

Converter. Bei der EJB-Programmierung (EJB = Enterprise JavaBeans) eine Klasse, die eine Datenbankdarstellung in einen Objekttyp umsetzt (und umgekehrt).

Create-Methode. In Enterprise-Beans eine Methode, die in der Home-Schnittstelle definiert ist und von einem Client aufgerufen wird, um eine Enterprise-Bean zu erstellen. (Sun)

CSV-Datei. Eine Datei, deren Datensätze durch Kommas voneinander getrennte Felder enthalten.

Dämon. Ein Programm, das unbeaufsichtigt ausgeführt wird und fortlaufend oder in regelmäßigen Abständen Funktionen wie die Netzsteuerung ausführt.

Dashboard. Eine Webseite, die einen oder auch mehrere Viewer umfassen kann, mit deren Hilfe Geschäftsdaten grafisch dargestellt werden können.

Daten-Grid. Ein System für den Zugriff auf Terabytes oder Petrabytes von Daten.

DB2. Eine Familie von IBM Lizenzprogrammen für die Verwaltung relationaler Datenbanken.

Deadlock. Eine Bedingung, bei der zwei unabhängige Steuerungsthreads blockiert werden, wobei jeder vom anderen eine Aktion erwartet. Deadlocks treten häufig dann auf, wenn Synchronisationsmechanismen zur Vermeidung von Konkurrenzsituationen hinzugefügt werden.

Demilitarized Zone (DMZ). Eine Konfiguration, die mehrere Firewalls enthält, um zwischen ein Unternehmens-Intranet und ein öffentliches Netz wie das Internet mehrere Sicherheitsebenen hinzuzufügen.

Deployment Manager. Ein Server, der den Betrieb einer logischen Gruppe oder einer Zelle anderer Server verwaltet.

Destination. Ein Exitpunkt, der verwendet wird, um Dokumente an ein Back-End-System oder einen Handelspartner zu liefern.

Differenziert. Dieser Begriff bezeichnet das Anzeigen der Details zu einem einzelnen Objekt.

Digitales Zertifikat. Ein elektronisches Dokument, das zur Identifikation einer Person, eines Systems, eines Servers, eines Unternehmens oder einer anderen Entität sowie zur Zuordnung eines öffentlichen Schlüssels zu der Entität dient. Ein digitales Zertifikat wird von einer Zertifizierungsstelle ausgestellt und von dieser Instanz digital signiert.

DMZ . Siehe Demilitarized Zone.

DNS. Siehe Domain Name System.

Document Type Definition (DTD). Die Regeln, die die Struktur einer bestimmten Klasse von SGML- oder XML-Dokumenten festlegen. Die DTD definiert die Struktur mit Elementen, Attributen und Notationen und richtet Integritätsbedingungen dafür ein, wie die einzelnen Elemente, Attribute und Notationen in der jeweiligen Klasse von Dokumenten verwendet werden können.

Domain Name System (DNS). Das verteilte Datenbanksystem, das Domänennamen IP-Adressen zuordnet.

Domäne. Ein Objekt, ein Symbol oder ein Container, der andere Objekte enthält, die die Ressourcen einer Domäne darstellen. Die Objektdomäne kann für die Verwaltung dieser Ressourcen verwendet werden.

Do-while-Schleife. Eine Schleife, die die gleiche Aktivitätenfolge so lange wiederholt, bis eine bestimmte Bedingung erfüllt ist. Anders als eine While-Schleife testet eine Do-while-Schleife ihre Bedingung am Ende der Schleife. Dies bedeutet, dass ihre Aktivitätenfolge immer mindestens einmal ausgeführt wird.

Dropdown. Siehe Pulldown.

DTD. Siehe Dokumenttypdefinition.

DTD-Dokumentdefinition. Eine Beschreibung oder ein Layout eines XML-Dokuments, die bzw. das auf einer XML-Dokumenttypdefinition basiert.

Durchsatz. Die Kennzahl für das Arbeitsvolumen, das von einem Gerät, wie einem Computer oder Drucker, im Verlauf eines bestimmten Zeitraums ausgeführt wird, beispielsweise die Anzahl Jobs pro Tag.

Dynamischer Cache. Eine Konsolidierung mehrerer Caching-Aktivitäten einschließlich der entsprechenden Servlets, Web-Services und WebSphere-Befehle in einem Service, in dem diese Aktivitäten Konfigurationsparameter gemeinsam nutzen und zusammenwirken, um die Leistung zu verbessern.

Dynamischer Cluster. Ein Servercluster, der basierend auf den von den Cluster-Membern erfassten Leistungsdaten anhand von Wertigkeiten die Arbeitslast dynamisch auf seine Cluster-Member verteilt.

EAR . Siehe Unternehmensarchiv.

EAR-Projekt. Siehe Unternehmensanwendungsprojekt.

Eclipse. Eine Open-Source-Initiative, die unabhängigen Softwareanbietern (ISVs) und anderen Toolentwicklern eine Standardplattform für die Entwicklung von plug-kompatiblen Anwendungsentwicklungstools zur Verfügung stellt.

Edition. Eine Nachfolgegeneration der Implementierung einer bestimmten Gruppe versionsgesteuerter Artefakte.

Editorbereich. In Eclipse und bei Eclipse-basierten Produkten der Bereich des Workbench-Fensters, in dem Dateien zum Bearbeiten geöffnet werden.

Eigenschaft. Ein Merkmal eines Objekts, durch das das Objekt beschrieben wird. Eine Eigenschaft kann geändert werden. Eigenschaften können unter anderem Namen, Typ, Wert oder Verhalten eines Objekts beschreiben.

Eigenständig. Unabhängig von allen anderen Einheiten, Programmen oder Systemen. In einer Netzumgebung greift ein eigenständiges System auf alle erforderlichen Ressourcen lokal zu.

Eigenständiger Server. Ein Katalogservice oder Containerserver, der vom Betriebssystem verwaltet wird, das den Serverprozess startet und stoppt.

Eingabeaufforderung. Eine Komponente einer Aktion, mit der angegeben wird, dass eine Benutzereingabe für ein Feld erforderlich ist, bevor ein Übergang zur Ausgabeanzeige erfolgt.

Eingangsunterbrechungspunkt. Ein Unterbrechungspunkt, der für ein Komponentenelement gesetzt ist und erreicht wird, bevor das Komponentenelement aufgerufen wird.

EJB. Siehe Enterprise JavaBeans.

EJB-Abfrage. In der EJB-Abfragesprache eine Zeichenfolge, die Folgendes enthält: eine optionale SELECT-Klausel, die die EJB-Objekte angibt, die zurückgegeben werden sollen; eine FROM-Klausel, die die Bean-Collections benennt; eine optionale WHERE-Klausel, die Suchvergleichselemente für die Collections enthält; eine optionale ORDER BY-Klausel, die die Sortierung der Ergebniscollection angibt; und Eingabeparameter, die den Argumenten der Finder-Methode entsprechen.

EJB-Container. Ein Container, der den EJB-Komponentenvertrag der Java EE-Architektur implementiert. In diesem Vertrag wird eine Laufzeitumgebung für Enterprise-Beans angegeben, die Services für die Sicherheit, den gemeinsamen Zugriff, das Lebenszyklusmanagement, Transaktionen, die Implementierung und anderes enthält. (Sun)

EJB-Factory. Eine Access-Bean, die die Erstellung einer Enterprise-Bean-Instanz oder die Suche nach einer solchen Instanz vereinfacht.

EJB-Home-Objekt. Bei der EJB-Programmierung (Enterprise JavaBeans) ein Objekt, das die Lebenszyklusoperationen (Erstellen, Entfernen, Suchen) für eine Enterprise-Bean bereitstellt. (Sun)

EJB-JAR-Datei. Ein Java-Archiv, das ein EJB-Modul enthält. (Sun)

EJB-Kontext. In Enterprise-Beans ein Objekt, das einer Enterprise-Bean ermöglicht, vom Container bereitgestellte Services aufzurufen und Informationen über den Aufrufenden einer Clientmethode abzurufen. (Sun)

EJB-Modul. Eine Softwareeinheit, die sich aus einer oder mehreren Enterprise-Beans und einem EJB-Deployment-Deskriptor besteht. (Sun)

EJB-Objekt. Bei Enterprise-Beans ein Objekt, dessen Klasse die ferne Schnittstelle der Enterprise-Bean implementiert (Sun).

EJB-Projekt. Ein Projekt, das die Ressourcen enthält, die für EJB-Anwendungen benötigt werden. Hierzu zählen Enterprise-Beans, Home-Schnittstellen, Local- und Remote-Schnittstellen, JSP-Dateien, Servlets und Implementierungsdeskriptoren.

EJB-Referenz. Ein logischer Name, der von einer Anwendung zur Lokalisierung der Home-Schnittstelle einer Enterprise-Bean in der Zielbetriebsumgebung verwendet wird.

EJB-Server. Software, die Services für einen EJB-Container bereitstellt. Ein EJB-Server kann einen oder mehrere EJB-Container aufnehmen. (Sun)

EJB-Vererbung. Eine Form der Vererbung, bei der eine Enterprise-Bean die Eigenschaften, Methoden und Deskriptorattribute für die Steuerung auf Methodenebene von einer anderen Enterprise-Bean in derselben Gruppe erbt.

Endpunkt.

1. Eine JCA-Anwendung oder ein anderer Clientkonsument eines Ereignisses aus dem unternehmensweiten Informationssystem.

2. Das System, das den Ursprung oder das Ziel einer Sitzung darstellt.

Endpunkt-Listener. Der Punkt oder die Adresse, an dem bzw. der eingehende Nachrichten für einen Web-Service von einem Service Integration Bus empfangen werden.

Engpass. Eine Situation im System, die eintritt, wenn die Konkurrenz um eine Ressource sich auf die Leistung auswirkt.

Enterprise-Bean. Eine Komponente, die eine Geschäfts-Task oder Geschäftsentität implementiert und in einem EJB-Container enthalten ist. Entity-Beans, Session-Beans und Message-Driven Beans sind verschiedene Typen von Enterprise-Beans. (Sun) Siehe auch Bean.

Enterprise JavaBeans (EJB). Eine von Sun Microsystems definierte Komponentenarchitektur für die Entwicklung und Implementierung objektorientierter, verteilter Unternehmensanwendungen (Java EE).

Enterprise Service Bus (ESB). Eine flexible Konnektivitätsinfrastruktur für die Integration von Anwendungen und Services, die eine flexible und einfach zu verwaltende Strategie für die SOA-Implementierung (SOA = service-orientierte Architektur) bietet.

Entität.

1. Eine einfache Java-Klasse, die eine Zeile in einer Datenbanktabelle oder einen Eintrag in einer Zuordnung darstellt.

2. In Markup-Sprachen wie XML eine Sammlung von Zeichen, die als Einheit referenziert werden können, beispielsweise für die Einbindung von häufig wiederholtem Text oder Sonderzeichen in einem Dokument.

Entity-Bean. In der EJB-Programmierung eine Enterprise-Bean, die persistente Daten darstellt, die in einer Datenbank verwaltet werden. Jede Entity-Bean hat eine eigene Entität. (Sun)

Entserialisierung. Eine Methode für die Konvertierung einer serialisierten Variablen in Objektdaten.

Ereignis.

1. Die Änderung eines Status wie beispielsweise der Abschluss oder das Fehlschlagen einer Operation, eines Geschäftsprozesses oder einer Benutzertask, die eine nachfolgende Aktion wie beispielsweise das Speichern der Ereignisdaten in einem Datenrepository oder das Aufrufen eines anderen Geschäftsprozesses auslösen kann.

2. Eine Änderung an den in einem EIS (Enterprise Information System) gespeicherten Daten, die vom Adapter verarbeitet und zur Lieferung von Geschäftsobjekten vom EIS zu den Endpunkten (Anwendungen) verwendet wird, die über die Änderung benachrichtigt werden müssen.

Erstellungsdefinitionsdatei. Eine XML-Datei, die Komponenten und Merkmale für ein angepasstes Installationspaket (CIP, Customized Installation Package) angibt.

Erstellungspfad. Der Pfad, der während der Kompilierung des Java-Quellcodes verwendet wird, um referenzierte Klassen zu finden, die sich in anderen Projekten befinden.

Erstellungsplan. Eine XML-Datei, die die Verarbeitung definiert, die zum Erstellen der Generierungsausgabe erforderlich ist und die das System angibt, auf dem die Verarbeitung stattfindet.

Erstellungszeitdaten. Objekte, die vom Umsetzungsprogramm nicht verwendet werden, wie beispielsweise EDI-Standards, ROD-Dokumenttypen (ROD = Record Oriented Data; satzorientierte Daten) und Zuordnungen.

ESB. Siehe Enterprise Service Bus (ESB).

Exklusive Sperre. Eine Sperre, die verhindert, dass Anwendungsprozesse ausgeführt werden, die gleichzeitig auf Datenbankdaten zugreifen. Siehe auch Gemeinsame Sperre.

Export. Eine offene Schnittstelle eines SCA-Moduls (SCA = Service Component Architecture), die einen Geschäftsservice für externe Systeme und Benutzer bereitstellt. Ein Export verfügt über eine Bindung, die definiert, wie auf den Service von Serviceanforderern zugegriffen werden kann, z. B. als Web-Service.

Exportdatei.

1. Eine Datei, die während des Entwicklungsprozesses für eingehende Operationen erstellt wurde und die Konfigurationseinstellungen für die Eingangsverarbeitung enthält.

2. Eine Datei, die exportierte Daten enthält.

Extensible Markup Language (XML). Eine Standardmetasprache für die Definition von Markup-Sprachen, die auf Standard Generalized Markup Language (SGML) basiert.

eXtreme-Scale-Grid. Ein Datenmuster, das für die Interaktion mit eXtreme Scale verwendet wird, wenn sich alle Daten und Clients in einem einzigen Prozess befinden.

Factory. In der objektorientierten Programmierung eine Klasse, die zum Erstellen von Instanzen einer anderen Klasse verwendet wird. Eine Factory wird verwendet, um die Erstellung von Objekten einer bestimmten Klasse auf einen Bereich zu isolieren, damit neue Funktionen ohne weitreichende Codeänderungen bereitgestellt werden können.

Failover. Eine automatische Operation, mit der auf ein redundantes oder Bereitschaftssystem umgeschaltet werden kann, wenn eine Software-, Hardware- oder Netzunterbrechung eintritt.

Fehler. Eine Abweichung zwischen einem berechneten, festgestellten oder gemessenen Wert oder einer solchen Bedingung und dem wahren, definierten oder theoretisch richtigen Wert bzw. der entsprechenden Bedingung.

Fehlerhafte Leseoperation. Eine Leseanforderung, die keinen Sperrmechanismus besitzt. Das bedeutet, dass Daten gelesen werden können, die später zurückgesetzt werden, was zu einer Inkonsistenz zwischen den gelesenen Daten und den Daten in der Datenbank führt.

Fehlerprotokollstrom. Ein fortlaufender Fluss von Fehlerinformationen, die in einem vordefinierten Format übertragen werden.

Firewall . Eine Netzkonfiguration, die in der Regel Hardware und Software umfasst und verhindert, dass nicht autorisierter Datenverkehr in ein sicheres Netz eingeht bzw. dieses verlässt.

Fixpack. Eine kumulative Gruppe von Fixes (Programmkorrekturen), die zwischen den geplanten Refresh-Packs, Produktaktualisierungen oder Releases bereitgestellt wird. Fixpacks ermöglichen es den Kunden, ihre Systeme auf eine bestimmte Wartungsstufe umzustellen. Siehe auch Vorläufiger Fix.

For-Schleife. Eine Schleife, die eine angegebene Anzahl von Wiederholungen für eine bestimmte Aktivitätenfolge ausführt.

Garbage-Collection. Eine Routine, die den Hauptspeicher durchsucht, um Speicher aus Programmsegmenten oder inaktiven Daten zurückzufordern.

Gehäuse. Ein Metallrahmen, in den verschiedene elektronische Komponenten eingebaut werden.

Geltungsbereich.

1. Eine Spezifikation der Begrenzung, innerhalb deren Systemressourcen verwendet werden können.

2. Bei Web-Services eine Eigenschaft, die die Lebensdauer des Objekts angibt, das die Aufrufanforderung bearbeitet.

Gemeinsame Sperre. Eine Sperre, die gleichzeitig aktive Anwendungsprozesse auf Leseoperationen für Datenbankdaten beschränkt.

General Inter-ORB Protocol (GIOP). Ein Protokoll, das von der Common Object Request Broker Architecture (CORBA) zum Festlegen des Nachrichtenformats verwendet wird.

Generisches Objekt. Ein Objekt, mit dem in API-Aufrufen und XPATH-Ausdrücken auf Konzepte, angepasste Entitäten oder Sammlungen verwiesen wird. Beispiel: Mit dem XPATH-Ausdruck '/WSRR/GenericObject' werden alle Konzepte aus WebSphere Service Registry and Repository abgerufen.

Gerüst. Der Entwurf für eine Implementierungsklasse.

Getter-Methode. Eine Methode, die den Zweck hat, den Wert einer Instanz oder einer Klassenvariablen abzurufen. Dadurch kann ein anderes Objekt den Wert einer seiner Variablen herausfinden.

GIOP . Siehe General Inter-ORB Protocol.

Global.

1. Dieser Begriff bezeichnet die Eigenschaft eines Elements, das für alle Prozesse innerhalb eines Arbeitsbereichs bereitgestellt wird. Ein globales Element wird im Projektbaum aufgeführt und kann in mehreren Prozessen verwendet werden. Tasks, Prozesse, Repositorys und Services können entweder global oder lokal sein. Global bedeutet hier, dass von allen Prozessen eines Projekts auf sie verwiesen werden kann, lokal bedeutet hingegen, dass sie einem bestimmten Prozess zugeordnet sind.

2. Dieser Begriff bezeichnet Informationen, die für mehr als ein Programm oder eine Unterroutine verfügbar sind.

Globale Instanz-ID. Eine global eindeutige ID, die entweder von der Anwendung oder vom Emitter generiert und als Primärschlüssel für die Ereignisidentifikation verwendet wird.

Globales Attribut. In XML ein Attribut, das als untergeordnetes Schemaelement und nicht als Teil einer komplexen Typdefinition deklariert ist. Auf globale Attribute kann unter Verwendung des Attributs ref in einem oder auch in mehreren Inhaltsmodellen verwiesen werden.

Globales Element. In XML ein Element, das als untergeordnetes Schemaelement und nicht als Teil einer komplexen Typdefinition deklariert ist. Auf globale Elemente kann unter Verwendung des Attributs ref in einem oder auch in mehreren Inhaltsmodellen verwiesen werden.

Globale Sicherheit. Die globale Sicherheit bezieht sich auf alle Anwendungen, die in der Umgebung ausgeführt werden, und legt fest, ob Sicherheitseinrichtungen zum Einsatz kommen. Ferner legt sie den Typ des für die Authentifizierung verwendeten Registry fest sowie andere Werte, von denen viele Standardeinstellungen sind.

Globale Transaktion. Eine wiederherstellbare Arbeitseinheit, die durch einen oder mehrere Ressourcenmanager in einer Umgebung für verteilte Transaktionen ausgeführt und von einem externen Transaktionsmanager koordiniert wird.

Globale Variable. Eine Variable, die verwendet wird, um die ihr während der Umsetzung zugeordneten Werte aufzunehmen und zu bearbeiten. Sie wird von verschiedenen Zuordnungen und Dokumentumsetzungen gemeinsam verwendet. Einer von drei Variablentypen, die von der Zuordnungsbefehlssprache von Data Interchange Services unterstützt werden.

Gruppe.

1. Ein Benutzerverbund, der Zugriffsberechtigungen für geschützte Ressourcen gemeinsam benutzen kann.

2. Ein Satz zusammengehöriger Dokumente in einem Austausch. Ein Austausch muss keine Gruppen enthalten, kann aber auch viele Gruppen enthalten.

3. Zwei oder mehrere Personen in einem Bereich, die für die Zugehörigkeit zu einem Bereich in einer Gruppe zusammengefasst werden.

HA. Siehe Hohe Verfügbarkeit (HA, High Availability).

HA-Gruppe. Eine Sammlung mit mindestens einem Mitglied, die zur Bereitstellung einer hohen Verfügbarkeit für einen Prozess genutzt wird.

HA-Richtlinie. Eine Gruppe von Regeln, die für eine HA-Gruppe definiert sind. Diese Regelgruppe gibt vor, ob null (0) oder mehr Mitglieder aktiviert sind. Die Richtlinie wird einer bestimmten HA-Gruppe zugeordnet, indem die Übereinstimmungskriterien der Richtlinie mit dem Gruppennamen abgeglichen werden.

High Availability Manager. Ein Gerüst, in dem die Stammgruppenzugehörigkeit bestimmt und Statusinformationen zwischen den Stammgruppen-Membern übertragen wird.

Hohe Verfügbarkeit (HA, High Availability). Bezeichnung für ein Clustersystem, das neu konfiguriert wird, wenn ein Knoten- oder Dämonfehler auftritt, damit die Workloads auf die verbleibenden Knoten im Cluster umverteilt werden können.

Host.

1. Ein Computer, der mit einem Netz verbunden ist und einen Zugriffspunkt auf dieses Netz bereitstellt. Der Host kann ein Client, ein Server oder Client und Server gleichzeitig sein.

2. Bei der Leistungsprofilerstellung ein System, das über Prozesse verfügt, für die ein Profil erstellt werden soll. Siehe auch Server.

Hostname.

1. Bei der Internetkommunikation der Name eines Computers. Der Hostname kann ein vollständig qualifizierter Domänenname wie mycomputer.city.company.com oder ein bestimmter Teilname wie mycomputer sein.

2. Der Netzname für einen Netzadapter auf einer physischen Maschine, auf der der Knoten installiert ist.

Hostsystem. Ein Großrechnersystem innerhalb eines Unternehmens, das als Host für 3270-Anwendungen eingesetzt wird. Bei Entwicklungstools für 3270-Terminal-Services verwendet der Entwickler den 3270-Terminal-Service-Recorder, um eine Verbindung zum Hostsystem herzustellen.

HTTP over SSL (HTTPS). Ein Web-Protokoll für sichere Transaktionen, das Seitenanforderungen von Benutzern und vom Web-Server zurückgegebene Seiten verschlüsselt und entschlüsselt.

HTTPS.

1. Siehe HTTP over SSL.

2. Siehe Hypertext Transfer Protocol Secure.

Hypertext Transfer Protocol Secure (HTTPS). Ein Internet-Protokoll, das von Webservern und Webbrowsern für die sichere Übertragung und Anzeige von Hypermedia-Dokumenten im Internet verwendet wird.

IDE . Siehe Integrierte Entwicklungsumgebung (Integrated Development Environment).

If-then-Regel. Eine Regel, in der die Aktion ('then'-Teil) nur dann ausgeführt wird, wenn die Bedingung ('if'-Teil) zutrifft.

IIOP. Siehe Internet Inter-ORB Protocol.

Implementieren. Dateien oder Software in einer Betriebsumgebung installieren. In Java EE (Java Platform Enterprise Edition) umfasst das Implementieren die Erstellung eines Implementierungsdeskriptors, der für den zu implementierenden Anwendungstyp geeignet ist.

Implementierphase. Siehe Implementierungsphase.

Implementierungscode. Zusätzlicher Code, der den Bean-Implementierungscode aktiviert, der von einem Anwendungsentwickler zur Verwendung in einer bestimmten EJB-Laufzeitumgebung geschrieben wurde. Implementierungscode kann mithilfe von Tools generiert werden, die der Anbieter des Anwendungsservers liefert.

Implementierungsdeskriptor. Eine XML-Datei (Extensible Markup Language), die den Einsatz eines Moduls oder einer Anwendung beschreibt, indem Sie Konfigurations- und Containeroptionen festlegt. Ein EJB-Deployment-Deskriptor übergibt beispielsweise Informationen zur Verwaltung und Steuerung einer Enterprise-Bean an einen EJB-Container.

Implementierungsphase. Eine Phase, in der Operationen zur Erstellung der Hosting-Umgebung für Anwendungen und zur Implementierung dieser Anwendungen ausgeführt werden. In dieser Phase werden auch die Ressourcenabhängigkeiten der Anwendung sowie Betriebsbedingungen, Kapazitätsanforderungen, Integritätsbedingungen und Zugriffsbeschränkungen aufgelöst.

Implementierungsrichtlinie. Eine optionale Methode für die Konfiguration einer eXtreme-Scale-Umgebung auf der Basis verschiedener Elemente, wie z. B. Anzahl der Systeme, Server, Partitionen, Replikate (einschließlich Replikattyp) und Heap-Speichergrößen für jeden Server.

Implementierungstopologie. Die Konfiguration von Servern und Clustern in einer Implementierungsumgebung und die physischen und logischen Beziehungen zwischen diesen Einheiten.

Implementierungsumgebung. Eine Sammlung konfigurierter Cluster, Server und Middlewarekomponenten, die zusammenarbeiten, um eine Umgebung bereitzustellen, in der Softwaremodule unterstützt werden können. Eine Implementierungsumgebung kann beispielsweise einen Host für Nachrichtenziele, einen Prozessor oder eine Sortierkomponente für Geschäftsereignisse und Verwaltungsprogramme umfassen.

Implementierungsverzeichnis. Das Verzeichnis, in dem sich die veröffentlichte Serverkonfiguration und die Webanwendung auf dem System, auf dem der Anwendungsserver installiert ist, befinden.

Import.

1. Das Entwicklungsartefakt, das zum Importieren eines Service verwendet wird, der nicht in einem Modul integriert ist.

2. Der Punkt, über den ein SCA-Modul auf einen externen Service (d. h. einen Service außerhalb des SCA-Moduls) in derselben Weise zugreift, wie dies bei einem lokalen Service möglich ist. Ein Import definiert die Interaktion zwischen dem SCA-Modul und dem Serviceanbieter. Ein Import verfügt über eine Bindung sowie mindestens eine Schnittstelle.

Index. Eine Gruppe von Zeigern, die logisch nach den Werten eines Schlüssels sortiert sind. Indizes ermöglichen den schnellen Zugriff auf Daten und können die Eindeutigkeit der SChlüsselwerte für die Zeilen in der Tabelle erzwingen.

Information Center. Eine Sammlung von Informationen, die Benutzern eines oder mehrerer Produkte zur Unterstützung bereitgestellt wird, die vom Produkt gesondert gestartet werden kann und die eine Liste der Abschnitte zur Navigation sowie eine Suchmaschine enthält.

Installationspaket. Eine installierbare Einheit eines Softwareprodukts. Softwareproduktpakete sind separat installierbare Einheiten, die unabhängig von anderen Paketen dieses Softwareprodukts eingesetzt werden können.

Installationsziel. Das System, auf dem ausgewählte Installationspakete installiert werden.

Instanz. Ein spezielles Vorkommen eines Objekts, das einer Klasse angehört.

Instanzieren. Das Darstellen einer Abstraktion durch eine konkrete Instanz.

Integrated Development Environment (IDE) . Eine Gruppe von Softwareentwicklungstools wie Quelleneditoren, Compiler und Debugger, auf die über eine gemeinsame Benutzerschnittstelle zugegriffen werden kann.

Integrierter Server. Ein Katalogservice oder Containerserver, der sich in einem vorhandenen Prozess befindet und innerhalb des Prozesses gestartet und gestoppt wird.

Internet Inter-ORB Protocol (IIOP). Ein Protokoll, das für die Kommunikation zwischen CORBA-Object-Request-Brokern (CORBA = Common Object Request Broker Architecture) eingesetzt wird.

Internet Protocol (IP). Ein Protokoll, das Daten über ein Netz oder über miteinander verbundene Netze leitet. Dieses Protokoll agiert als Mittler zwischen den höheren Protokollschichten und dem physischen Netz.

IP. Siehe Internet Protocol.

IP-Sprayer. Eine Einheit, die sich zwischen den von den Benutzern eingehenden Anforderungen und den Anwendungsserverknoten befindet und Anforderungen an Knoten weiterleitet.

Iteration. Siehe Schleife.

Iterator. Eine Klasse oder Anweisung, die verwendet wird, um die Objekte einer Objektgruppe nacheinander zu durchlaufen.

JAAS. Siehe Java Authentication and Authorization Service.

JAF . Siehe JavaBeans Activation Framework.

JAR-Datei. Eine Java-Archivdatei. Siehe auch Webarchiv, Unternehmensarchiv.

Java. Eine objektorientierte Programmiersprache für portierbaren, interpretierenden Code, die die Interaktion zwischen fernen Objekten unterstützt. Java wurde von Sun Microsystems, Incorporated entwickelt und spezifiziert.

Java API für XML (JAX). Eine Gruppe Java-basierter Anwendungsprogrammierschnittstellen für die Verarbeitung verschiedener Operationen, an denen über Extensible Markup Language (XML) definierte Daten beteiligt sind.

Java-Archiv. Ein Format für komprimierte Dateien, mit dem alle Ressourcen, die zur Installation und Ausführung eines Java-Programms erforderlich sind, in einer einzigen Datei gespeichert werden können. Siehe auch Webarchiv, Unternehmensarchiv.

Java Authentication and Authorization Service (JAAS). In Java EE technology, a standard API for performing security-based operations. Über JAAS können Services die Authentifizierung und Berechtigung von Benutzern ausführen. Gleichzeitig bleiben die zugehörigen Anwendungen unabhängig von den zugrunde liegenden Technologien.

JavaBeans . Ein für Java von Sun Microsystems definiertes, portierbares und plattformunabhängiges Komponentenmodell, das wiederverwendbar ist. Siehe auch Bean.

JavaBeans Activation Framework (JAF) . Eine Standarderweiterung für die Java-Plattform, die beliebige Datentypen und verfügbare Operationen bestimmt und eine Bean so instanzieren kann, dass sie relevante Services ausführt.

Java Command Language. Eine Scripting-Sprache für die Java-Umgebung, die zur Erstellung von Webinhalten und zur Steuerung von Java-Anwendungen benutzt werden kann.

Java Connector Security. Eine Architektur, die zur Erweiterung des End-to-End-Sicherheitsmodells für Java EE-basierte Anwendungen entworfen wurde, so dass dieses auch unternehmensweite Informationssysteme (EIS = Enterprise Information Systems) umfassen kann.

Java Database Connectivity (JDBC) . Ein Industriestandard für datenbankunabhängige Konnektivität zwischen der Java-Plattform und verschiedenen Datenbanken. Die JDBC-Schnittstelle stellt eine Schnittstelle auf Aufrufebene für den SQL-basierten und den XQuery-basierten Datenbankzugriff bereit.

Java-Datei. Eine Quellendatei (mit der Erweiterung .java), die bearbeitet und in Bytecode (eine Datei mit der Erweiterung .class) kompiliert werden kann.

Javadoc.

1. Ein Tool, das die Deklarationen und Dokumentationskommentare in einer Gruppe von Quellendateien syntaktisch analysiert und eine Reihe von HTML-Seiten erstellt, die die Klassen, untergeordneten Klassen, Schnittstellen, Konstruktoren, Methoden und Felder beschreiben. (Sun)

2. Dieser Begriff bezeichnet ein Tool, das die Deklarationen und Dokumentationskommentare in einer Gruppe von Quellendateien syntaktisch analysiert und eine Reihe von HTML-Seiten erstellt, die die Klassen, untergeordneten Klassen, Schnittstellen, Konstruktoren, Methoden und Felder beschreiben.

Java EE. Siehe Java Platform Enterprise Edition.

Java EE-Anwendung. Eine beliebige implementierbare Einheit mit Java EE-Funktionalität. Bei dieser Einheit kann es sich um ein einzelnes Modul oder um eine Gruppe von Modulen handeln, die in einer EAR-Datei (EAR = Enterprise Archive) mit einem Java EE-Anwendungsimplementierungsdeskriptor gepackt sind. (Sun)

Java EE Connector Architecture (JCA). Eine Standardarchitektur für die Verbindung der Java EE-Plattform mit heterogenen unternehmensweiten Informationssystemen (EIS = Enterprise Information Systems).

Java-EE-Server. Eine Laufzeitumgebung, die EJB- oder Web-Container bereitstellt.

Java-Klasse. Eine Klasse, die in der Programmiersprache Java geschrieben wurde.

JavaMail API. Ein plattform- und protokollunabhängiges Gerüst für die Erstellung Java-basierter Mail-Clientanwendungen.

Java Message Service (JMS). Eine Anwendungsprogrammierschnittstelle, die Java-Sprachfunktionen für die Verarbeitung von Nachrichten zur Verfügung stellt.

Java Naming and Directory Interface (JNDI). Eine Erweiterung der Java-Plattform, die eine Standardschnittstelle für heterogene Namens- und Verzeichnisservices bereitstellt.

Java Platform, Enterprise Edition (Java EE). Eine Umgebung zum Entwickeln und Implementieren von Unternehmensanwendungen, die von Sun Microsystems Inc. definiert wurde. Die Java EE-Plattform besteht aus einer Reihe von Services, Anwendungsprogrammierschnittstellen (APIs) und Protokollen, die die Funktionalität für die Entwicklung mehrschichtiger, webbasierter Anwendungen zur Verfügung stellen. (Sun)

Java Platform, Standard Edition (Java SE). Die zentrale Plattform der Java-Technologie. (Sun)

Java-Projekt. Bei Eclipse ein Projekt, das kompilierbaren Java-Quellcode enthält und einen Container für Quellenordner oder Pakete darstellt.

Java Runtime Environment (JRE). Eine Untergruppe des Java Developer Kit, die die zentralen ausführbaren Programme und Dateien enthält, auf denen die Java-Standardplattform basiert. Die Java Runtime Environment (JRE) umfasst die Java Virtual Machine (JVM) sowie die wichtigsten Klassen und Unterstützungsdateien.

JavaScript. Eine Web-Scripting-Sprache, die sowohl von Browsern als auch von Web-Servern verwendet wird. (Sun)

JavaScript Object Notation. Ein einfaches Datenaustauschformat, das auf der Objekt-Literal-Notation von JavaScript basiert. JSON ist programmiersprachenneutral, verwendet allerdings Konventionen aus Sprachen, wie C, C++, C#, Java, JavaScript, Perl und Python.

Java SE. Siehe Java Platform Standard Edition.

Java Secure Socket Extension (JSSE) . Ein Java-Paket, das zur Bereitstellung der sicheren Internetkommunikation dient. Es implementiert eine Java-Version der Protokolle SSL (Secure Sockets Layer) und TSL (Transport Layer Security) und unterstützt die Datenverschlüsselung, die Serverauthentifizierung sowie die Überprüfung der Nachrichtenintegrität und optional die Clientauthentifizierung.

Java SE Development Kit (JDK). Der Name des Software Development Kit, das von Sun Microsystems für die Java-Plattform bereitgestellt wird.

JavaServer Pages (JSP) . Eine serverseitige Scripting-Technologie, mit deren Hilfe Java-Code dynamisch in Webseiten (HTML-Dateien) eingebettet und beim Bereitstellen der Seite ausgeführt werden kann, um dynamische Inhalte an den Client zurückzugeben.

Java Specification Request (JSR). Eine formal eingereichter Spezifikationsvorschlag für die Java-Plattform.

Java Virtual Machine (JVM). Die Softwareimplementierung eines Prozessors, die kompilierten Java-Code (Applets und Anwendungen) ausführt.

Java Virtual Machine Profiler Interface (JVMPI). Ein Profilerstellungstool, das die Erfassung von Informationen wie z. B. von Daten zur Garbage-Collection sowie Angaben zu der Java Virtual Machine-Anwendungsprogrammierschnittstelle (JVM-API) unterstützt, die zur Ausführung des Anwendungsservers eingesetzt wird.

JAX . Siehe Java API for XML.

JCA. Siehe Java EE Connector Architecture.

JDBC. Siehe Java Database Connectivity.

JDK. Siehe Java SE Development Kit.

JMS. Siehe Java Message Service.

JMS-Datenbindung. Eine Datenbindung, die eine Zuordnung zwischen dem von einer externen JMS-Nachricht verwendeten Format und der SDO-Darstellung (SDO = Service Data Object) bereitstellt, die von einem SCA-Modul (SCA = Service Component Architecture) eingesetzt wird.

JMX . Siehe Java Management Extensions.

JMX (Java Management Extensions). Eine Methode für die Verwaltung mit Java-Technologie. JMX ist eine universelle, offene Erweiterung der Programmiersprache Java für die Verwaltung, die in jedem Unternehmen eingesetzt werden kann, in dem eine Verwaltung erforderlich ist.

JNDI. Siehe Java Naming and Directory Interface.

JSP. Siehe JavaServer Pages.

JSP-Datei. Eine scriptgesteuerte HTML-Datei mit der Dateierweiterung ".jsp", die den Einschluss dynamischer Inhalte in Webseiten ermöglicht. Eien JSP-Datei kann über einen URL direkt angefordert, von einem Servlet oder in einer HTML-Seite aufgerufen werden.

JSP-Seite. Ein textbasiertes Dokument, das feste Schablonendaten und JSP-Elemente verwendet, die beschreiben, wie eine Anforderung verarbeitet werden soll, um eine Antwort zu erstellen. (Sun)

JSR. Siehe Java Specification Request.

JSSE. Siehe Java Secure Socket Extension.

JVM. Siehe Java Virtual Machine.

JVMPI. Siehe Java Virtual Machine Profiler Interface.

Jython. Eine Implementierung der Programmiersprache Python, die mit der Java-Plattform integriert ist.

Katalog. Ein Container, dessen Projektbaum abhängig vom jeweiligen Containertyp Prozesse, Daten, Ressourcen, Organisationen oder Berichte enthält.

Katalogservice. Ein Service, der die Positionierung von Shards steuert und den Status der Container erkennt und überwacht.

Kategorie. Ein Container, der in einem Strukturdiagramm zum Gruppieren von Elementen auf der Basis eines gemeinsamen Attributs oder einer gemeinsamen Qualität verwendet wird.

Klasse. In der objektorientierten Programmierung ein Modell oder eine Schablone, die verwendet werden kann, um Objekte mit einer gemeinsamen Definition und gemeinsamen Merkmalen, Operationen und Verhaltensmustern zu erstellen. Ein Objekt ist eine Instanz einer Klasse.

Klassendatei. Eine kompilierte Java-Quellendatei.

Klassenhierarchie. Die Beziehungen zwischen Klassen, die einen gemeinsamen Vorfahren (Basisklasse) haben.

Klassenlader. Eine Komponente der Java Virtual Machine (JVM), die für das Suchen und Laden von Klassendateien verwendet wird. Ein Klassenladeprogramm wirkt sich auf das Packen von Anwendungen und das Laufzeitverhalten von gepackten Anwendungen aus, die auf Anwendungsservern implementiert sind.

Klassenpfad. Eine Liste mit Verzeichnissen und JAR-Dateien, die Ressourcendateien oder Java-Klassen enthalten, die ein Programm zur Laufzeit dynamisch laden kann.

Klassifikationsmerkmal. Ein spezielles Attribut, das zur Gruppierung und Farbcodierung von Prozesselementen dient.

Knoten.

1. Eine logische Gruppierung verwalteter Server.

2. Ein Element in einer Baumstruktursteuerung, wie beispielsweise ein einfaches Element, ein Verbundelement, ein Zuordnungsbefehl, ein Kommentar oder ein Gruppenknoten.

3. In XML die kleinste Einheit der gültigen und vollständigen Struktur in einem Dokument.

4. Die grundlegenden Formen, aus denen sich ein Diagramm zusammensetzt.

Kohärenter Cache. Ein Cache, der die Integrität so verwaltet, dass alle Clients dieselben Daten sehen.

Kompiliereinheit. Ein Teil eines Computerprogramms, das ausreichend vollständig ist, um ordnungsgemäß kompiliert werden zu können.

Kompilierzeit. Der Zeitraum, während dessen ein Computerprogramm zu einem ausführbaren Programm kompiliert wird.

Komponente.

1. Ein wiederverwendbares Objekt oder Programm, das eine bestimmte Funktion ausführt und mit anderen Komponenten und Anwendungen arbeitet.

2. Bei Eclipse ein bestimmtes Plug-in oder mehrere Plug-ins, die zusammenarbeiten, um eine eigenständige Funktionsgruppe bereitzustellen.

Komponentenelement. Eine Entität in einer Komponente, für die ein Unterbrechungspunkt gesetzt werden kann, wie beispielsweise eine Aktivität oder ein Java-Snippet in einem Geschäftsprozess bzw. ein Mediationsbasiselement oder ein Knoten in einem Mediationsablauf.

Komponenteninstanz. Eine aktive Komponente, die parallel zu anderen Instanzen derselben Komponente ausgeführt werden kann.

Komponententest. Ein automatisierter Test einer oder mehrerer Komponenten einer Unternehmensanwendung, die Java-Klassen, EJB-Beans oder Web-Services beinhalten kann.

Lastausgleich. Die Überwachung von Anwendungsservern und die Verwaltung der Workload auf Servern. Wenn einer der Server überlastet ist, werden die Anforderungen an einen anderen Server mit mehr Kapazität weitergeleitet.

Laufzeit. Der Zeitraum, während dessen ein Computerprogramm ausgeführt wird.

Laufzeittopologie. Eine Abbildung des momentanen Zustands der Umgebung.

LDAP. Siehe Lightweight Directory Access Protocol.

LDAP-Verzeichnis. Ein Repository-Typ, in dem Informationen zu Personen, Organisationen und anderen Ressourcen gespeichert werden und auf den über das LDAP-Protokoll zugegriffen wird. Die Einträge im Repository sind in einer hierarchischen Struktur organisiert. In manchen Fällen gibt diese hierarchische Struktur die Struktur oder Geographie einer Organisation wieder.

Lebensdauer. (TTL, Time-to-Live) Das Zeitintervall in Sekunden, für das ein Eintrag vor dem Löschen im Cache enthalten sein kann.

Lebenszyklus. Ein vollständiger Durchlauf durch die vier Softwareentwicklungsphasen Konzeption, Ausarbeitung, Konstruktion und Ablösung.

Lightweight Directory Access Protocol (LDAP). Ein offenes Protokoll, das TCP/IP verwendet, um den Zugriff auf Verzeichnisse bereitzustellen, die ein X.500-Modell unterstützen und bei dem weniger aufwendige Ressourcenanforderungen gelten als bei dem komplexeren X.500 Directory Access Protocol (DAP). LDAP kann beispielsweise eingesetzt werden, um Personen, Organisationen und andere Ressourcen in einem Internet- oder Intranetverzeichnis zu lokalisieren.

Listener. Ein Programm, das ankommende Anforderungen erkennt und den zugehörigen Channel startet.

Listener-Port. Ein Objekt, das die Zuordnung zwischen einer Verbindungs-Factory, einer Zieladresse und einer implementierten Message-Driven-Bean (MDB) definiert. Listener-Ports vereinfachen die Verwaltung der Zuordnungen zwischen diesen Ressourcen.

Loader. (Ladeprogramm) Eine Komponente, die Daten aus einem persistenten Speicher liest bzw. in diesen schreibt.

Lokal.

1. Bezeichnet eine Einheit, eine Datei oder ein System, auf die bzw. das direkt über das System eines Benutzers zugegriffen wird, ohne dass hierbei eine Übertragungsleitung verwendet werden muss.

2. Bezeichnet ein Element, das nur innerhalb seines eigenen Prozesses zur Verfügung steht.

Lokale Datenbank. Eine Datenbank, die sich auf der momentan verwendeten Workstation befindet.

LTPA. Siehe Lightweight Third Party Authentication.

LTPA (Lightweight Third Party Authentication). Ein Protokoll, das die Sicherheit in einer verteilten Umgebung durch Verschlüsselung unterstützt.

Managed Bean (MBean). In der Spezifikation Java Management Extensions (JMX) die Java-Objekte, die Ressourcen und die zugehörige Instrumentierung implementieren.

Map.

1. Eine Datenstruktur, die Schlüssel Werten zuordnet.

2. Eine Datei, die die Umsetzung zwischen Quellen und Zielen definiert.

3. In der EJB-Entwicklungsumgebung die Spezifikation, die angibt, wie die CMP-Felder (CMP = Container-managed Persistence) einer Enterprise-Bean den Spalten in einer Tabelle einer relationalen Datenbank oder einem anderen persistenten Speicher entsprechen.

MBean. Siehe Managed Bean.

MBean-Provider. Eine Bibliothek, die eine Implementierung einer JMX-MBean und die zugehörige XML-Deskriptordatei enthält.

Messgröße. Ein Behältnis für Informationen in einem Überwachungskontext, normalerweise für einen Geschäftsleistungsmesswert.

Methode. Bei der objektorientierten Programmierung eine Operation, die von einem Objekt ausgeführt werden kann. Ein Objekt kann viele Methoden aufweisen.

Nachgeordnet. Dieser Begriff bezeichnet die Richtung eines Verarbeitungsablaufs. Dieser verläuft vom ersten Knoten innerhalb des Prozesses (d. h. vom vorgeordneten Knoten) zum letzten Knoten des Prozesses (d. h. zum nachgeordneten Knoten).

Namespace. Ein logischer Container, in dem alle Namen eindeutig sind. Die eindeutige Kennung für ein Artefakt setzt sich aus dem Namespace und dem lokalen Namen des Artefakts zusammen.

Node Agent. Ein Verwaltungsagent, der alle Anwendungsserver auf einem Knoten verwaltet und den Knoten in der Verwaltungszelle repräsentiert.

ObjectGrid. Eine Grid-fähige Speicherdatenbank für Anwendungen, die in Java geschrieben werden. ObjectGrid kann als speicherinterne Datenbank oder zum Verteilen von Daten in einem Netz verwendet werden.

Object Request Broker. In der objektorientierten Programmierung eine Software, die als Vermittler fungiert, indem Sie den Austausch von Anforderungen und Antworten zwischen Objekten transparent erlaubt.

Objekt. Im objektorientierten Design oder in der objektorientierten Programmierung eine konkrete Ausführung (Instanz) einer Klasse, die aus Daten und den zugehörigen Operationen besteht. Ein Objekt enthält die Instanzdaten, die von der Klasse definiert werden. Eigner der Operationen, die den Daten zugeordnet sind, ist aber die Klasse.

Objektorientierte Programmierung. Eine Programmierungsmethode auf der Basis der Konzepte zur Datenabstraktion und Vererbung. Im Gegensatz zu Verfahren der prozeduralen Programmierung liegt der Schwerpunkt der objektorientierten Programmierung nicht darauf, wie etwas erreicht wird, sondern darauf, welche Datenobjekte das Problem umfasst und wie diese bearbeitet werden.

ODBC. Siehe Open Database Connectivity.

Open Database Connectivity (ODBC). Eine standardisierte Anwendungsprogrammierschnittstelle (API = Application Programming Interface) für den Zugriff auf Daten in relationalen und nicht relationalen Datenbankmanagementsystemen. Unter Verwendung dieser API können Datenbankanwendungen auf Daten zugreifen, die in Datenbankmanagementsystemen auf verschiedenen Computern gespeichert sind, auch wenn die einzelnen Datenbankmanagementsysteme unterschiedliche Formate für die Datenspeicherung und unterschiedliche Programmierschnittstellen verwenden.

Open Source. Software, deren Quellcode für die Verwendung oder für Änderungen allgemein zur Verfügung steht. Open-Source-Software wird normalerweise in Form einer allgemeinen Zusammenarbeit entwickelt und wird frei verfügbar gemacht, ihre Verwendung und Neuverteilung kann jedoch Lizenzbeschränkungen unterliegen. Linux ist ein sehr bekanntes Beispiel für Open-Source-Software.

Operation. Eine Implementierung von Funktionen oder Abfragen, zu deren Ausführung ein Objekt aufgerufen werden kann.

ORB. Siehe Object Request Broker.

Ordner. Ein Container, der zum Verwalten von Objekten verwendet wird.

Organisation. Eine Entität, in der Personen zusammenarbeiten, um angegebene Ziele zu erreichen, wie ein Unternehmen, eine Firma oder eine Fabrik.

Paket.

1. In der Java-Programmierung ein Gruppe von Typen. Pakete werden mit dem Schlüsselwort package deklariert. (Sun)

2. Der Wrapper um den Dokumentinhalt, der das zum Übertragen eines Dokuments im Internet zu verwendende Format (z. B. RNIF, AS1 oder AS2) definiert.

3. In Module zusammengefasste Komponenten und in Unternehmensanwendungen zusammengefasste Module.

Partitionierungsfeature (WPF). Ein Programmierungsframework und eine Systemmanagementinfrastruktur, die das Konzept der Partitionierung für Enterprise-Beans, HTTP-Datenverkehr und Datenbankzugriff unterstützt.

Performance Monitoring Infrastructure (PMI). Eine Gruppe von Paketen und Bibliotheken für die Erfassung, Zustellung, Verarbeitung und Anzeige von Leistungsdaten.

Persistenter Datenspeicher. Ein nicht flüchtiger Speicher für Ereignisdaten (wie beispielsweise ein Datenbanksystem), der über Sitzungsgrenzen hinweg existiert und nach der Ausführung des erstellenden Programms oder Prozesses erhalten bleibt.

Persistent speichern. Elemente über Sitzungsgrenzen hinweg verwalten, in der Regel in einem nicht flüchtigen Speicher wie einem Datenbanksystem oder einem Verzeichnis.

Persistenz.

1. Ein Merkmal von Daten, die über Sitzungsgrenzen hinweg beibehalten werden, oder eines Objekts, das auch nach der Ausführung des erstellenden Programms oder Prozesses erhalten bleibt (normalerweise in einem nicht flüchtigen Speicher, wie einem Datenbanksystem).

2. In Java EE das Protokoll, mit dem der Status einer Entity-Bean zwischen den jeweiligen Instanzvariablen und einer zugrunde liegenden Datenbank übertragen wird. (Sun)

Pessimistisches Sperren. Eine Sperrstrategie, bei der bei Auswahl einer Zeile eine Sperre gesetzt wird, die so lange gehalten wird, bis versucht wird, eine Aktualisierungs- oder Löschoperation mit Suche für diese Zeile durchzuführen.

Plattform Java. Ein Sammelbegriff für die Sprache Java zum Schreiben von Programmen. Sie umfasst eine Gruppe von APIs, Klassenbibliotheken und anderen Programmen, die bei der Entwicklung, Kompilierung und Prüfung von Programmen auf Fehler verwendet werden, sowie eine Java Virtual Machine (JVM), die die Klassendateien lädt und ausführt. (Sun)

Plug-in. Ein separat installierbares Softwaremodul, das einem vorhandenen Programm, einer vorhandenen Anwendung oder einer vorhandenen Schnittstelle eine Funktion hinzufügt.

PMI . Siehe Performance Monitoring Infrastructure.

Port. Gemäß Definition in einem WSDL-Dokument (Web Services Description Language) ein Endpoint, der als Kombination von Bindung und Netzadresse definiert wird.

Portnummer. Bei der Internetkommunikation die Kennung für einen logischen Connector zwischen einer Anwendungsentität und dem Transportservice.

Primärschlüssel.

1. Ein Objekt, das eine Entity-Bean eines bestimmten Typs eindeutig kennzeichnet.

2. In einer relationalen Datenbank ein Schlüssel, der eine einzige Zeile in einer Datenbanktabelle eindeutig kennzeichnet.

Primitiver Datentyp. In Java eine Datentypkategorie, die eine Variable mit einem Einzelwert beschreibt, dessen Größe und Format dem Typ entsprechen: eine Zahl, ein Zeichen oder ein boolescher Wert. Beispiele für primitive Datentypen sind byte, short, int, long, float, double, char, boolean.

Profil. Daten, die die Merkmale eines Benutzers, einer Gruppe, einer Ressource, eines Programms, einer Einheit oder einer fernen Position beschreiben.

Protokollbindung. Eine Bindung, mit der der Enterprise Service Bus Nachrichten unabhängig vom Übertragungsprotokoll verarbeiten kann.

Protokollierung. Die Aufzeichnung von Daten zu bestimmten Ereignissen auf dem System, wie z. B. Fehler.

Proxy. Ein Anwendungsgateway zwischen zwei Netzen für eine bestimmte Netzanwendung, wie Telnet oder FTP, wenn beispielsweise der Proxy-Telnet-Server einer Firewall die Authentifizierung des Benutzers ausführt und dann den Datenverkehr über den Proxy laufen lässt, als ob dieser nicht vorhanden wäre. Die Funktion wird in der Firewall ausgeführt, nicht auf der Client-Workstation. Dadurch vergrößert sich die Arbeitslast in der Firewall.

Proxy-Cluster. Eine Gruppe von Proxyservern, die HTTP Anforderungen über den Cluster verteilt.

Proxy-Peer-Zugriffspunkt. Ein Mittel, mit dem die Kommunikationseinstellungen für einen Peer-Zugriffspunkt ermittelt werden können, auf den kein direkter Zugriff möglich ist.

Proxy-Server.

1. Ein Server, der als Zwischenstation für HTTP-Webanforderungen von einer Anwendung oder einem Webserver auftritt. Ein Proxy-Server tritt stellvertretend für die Content-Server im Unternehmen auf.

2. Ein Server, der Anforderungen für einen anderen Server empfängt und für den Client agiert (als Proxy des Clients), um den angeforderten Service abzurufen. Ein Proxy-Server wird häufig verwendet, wenn der Client und der

Server für eine Direktverbindung inkompatibel sind. Beispiel: Der Client kann die Anforderungen zur Sicherheitsauthentifizierung des Servers nicht erfüllen, soll aber für einige Services berechtigt werden.

Prozess.

1. Eine immer weiter voranschreitende Prozedur, die aus einer Reihe gesteuerter Aktivitäten besteht, die systematisch auf ein bestimmtes Ergebnis oder Ziel gerichtet sind.

2. Die Reihenfolge der Dokumente oder Nachrichten, die zwischen den Community-Managern und den Teilnehmern ausgetauscht werden sollen, um eine Geschäftstransaktion auszuführen.

PTF. Siehe Vorläufige Programmkorrektur (Program Temporary Fix).

public.

1. Bei der objektorientierten Programmierung bezieht sich dieser Begriff auf einen Klasseneintrag, auf den alle Klassen zugreifen können.

2. In der Programmiersprache Java bezieht sich dieser Begriff auf eine Methode oder Variable, auf die Elemente aus anderen Klassen zugreifen können. (Sun)

Punkt-zu-Punkt. Dieser Begriff bezeichnet einen Stil einer Messaging-Anwendung, bei der die sendende Anwendung das Ziel der Nachricht kennt.

QoS. Siehe Servicequalität (Quality of Service).

Qualifikationsmerkmal. Ein einfaches Element, das einem anderen generischen Verbundelement oder einfachen Element eine bestimmte Bedeutung gibt. Qualifikationsmerkmale werden bei der Zuordnung einmaliger oder mehrfacher Vorkommen verwendet. Ein Qualifikationsmerkmal kann auch zur Bezeichnung des Namensbereichs verwendet werden, mit dem der zweite Teil des Namens interpretiert wird. Dieser Teil wird im Allgemeinen als ID bezeichnet.

Read-Through-Cache. Ein Teilcache, der Dateneinträge nach Schlüssel lädt, wenn sie angefordert werden. Wenn Daten nicht im Cache gefunden werden, werden die fehlenden Daten mit dem Ladeprogramm abgerufen, das die Daten aus dem Back-End-Repository lädt und in den Cache einfügt.

Region . Ein zusammenhängender Bereich des virtuellen Speichers, der allgemeine Merkmale aufweist und von Prozessen gemeinsam genutzt werden kann.

Rekursion. Eine Programmiertechnik, bei der ein Programm oder eine Routine sich selbst aufruft, um aufeinander folgende Schritte in einer Operation auszuführen, wobei jeder Schritt die Ausgabe des vorhergehenden Schrittes verwendet.

Replikat. Ein Server, der eine Kopie des Verzeichnisses bzw. der Verzeichnisse eines anderen Servers enthält. Replikate sichern Server, um die Leistung bzw. Antwortzeiten zu verbessern und die Datenintegrität zu gewährleisten.

Replikation. Der Prozess, mit dem ein definierter Satz von Daten an mehr als einer Position verwaltet wird. Zur Replikation gehören das Kopieren festgelegter Änderungen für eine Position (eine Quelle) an eine andere (ein Ziel) und die Synchronisation der Daten an beiden Positionen.

Ressource.

1. Eine diskrete Ressource. Beispiele: Anwendungssuiten, Anwendungen, Geschäftsservices, Schnittstellen, Endpunkte und Geschäftsereignisse.

2. Eine Einrichtung eines Computer- oder Betriebssystems, die für einen Job, eine Task oder ein aktives Programm erforderlich ist. Ressourcen können unter anderem Hauptspeicher, Ein-/Ausgabeeinheiten, die Verarbeitungseinheit, Datensätze, Dateien, Bibliotheken, Ordner, Anwendungsserver und Steuer- oder Verarbeitungsprogramme sein.

3. Eine Person, ein Bauteil oder Material, die bzw. das für die Ausführung einer Task oder eines Projekts verwendet wird. Jede Ressource ist ein bestimmtes Vorkommen oder Beispiel für eine Ressourcendefinition.

Richtlinie. Eine Gruppe von Aspekten, die das Verhalten einer verwalteten Ressource oder eines Benutzers beeinflussen.

Rolle.

1. Eine Beschreibung einer Funktion, die von einer Einzelperson oder einer Massenressource ausgeführt werden soll, sowie der zu ihrer Ausführung erforderlichen Qualifikationen. Bei der Simulation und Analyse wird der Begriff Rolle auch für die qualifizierten Ressourcen verwendet.

2. Eine Jobfunktion, die die Tasks, die ein Benutzer ausführen kann, und die Ressourcen angibt, auf die ein Benutzer Zugriff hat. Einem Benutzer können eine oder mehrere Rollen zugeordnet werden.

3. Eine logische Gruppe von Principals, die eine Gruppe von Berechtigungen bereitstellt. Der Zugriff auf Operationen wird über die Zugriffsberechtigungen für eine Rolle gesteuert.

4. In einer Beziehung legt eine Rolle die Funktion und Teilnahme von Entitäten fest. Rollen erfassen Anforderungen bezüglich Struktur und Integritätsbedingungen für teilnehmende Entitäten und ihre Art der Teilnahme. Beispielsweise lauten die Rollen in einer Beschäftigungsbeziehung ″Arbeitgeber″ und ″Mitarbeiter″.

Root. Der Benutzername für den Systembenutzer mit der höchsten Berechtigungsstufe.

Schleife. Eine Instruktionsfolge, die wiederholt ausgeführt wird.

Schlüssel.

1. Ein verschlüsselter mathematischer Wert, der zum digitalen Signieren, Überprüfen, Verschlüsseln oder Entschlüsseln einer Nachricht verwendet wird.

2. Die Informationen, die eine reale Entität, die von einem Überwachungskontext protokolliert wird, beschreiben und eindeutig identifizieren.

Schlüsselwort. Eines der vordefinierten Wörter einer Programmiersprache, einer Kunstsprache, Anwendung oder eines Befehls.

Schnittstelle. Eine Gruppe von Operationen, die verwendet werden, um den Service einer Klasse oder einer Komponente anzugeben.

Schwellenwert. Eine Einstellung, die für einen Interrupt in einer Simulation gilt, der definiert, wann eine Prozesssimulation auf der Basis einer Bedingung angehalten werden soll, die für einen angegebenen Anteil von Vorkommen eines bestimmten Ereignisses existiert.

Script. Eine Reihe von Befehlen, die in einer Datei zusammengefasst sind und durch die beim Ausführen der Datei eine bestimmte Funktion ausgeführt wird. Scripts werden während ihrer Ausführung interpretiert.

Scripting. Eine Art der Programmierung, bei der vorhandene Komponenten als Basis zum Erstellen von Anwendung wiederverwendet werden.

SDK. Siehe Software Development Kit.

Secure Socket Layer (SSL). Ein Sicherheitsprotokoll für den Schutz personenbezogener Daten bei der Datenübertragung. Mit SSL können Client/Server-Anwendungen auf eine Weise kommunizieren, die das Ausspionieren, die Manipulation von Daten während der Übertragung und die Nachrichtenfälschung verhindern soll.

Serialisierung. Bei der objektorientierten Programmierung das sequenzielle Schreiben von Daten aus dem Programmspeicher an ein Kommunikationsmedium.

Serialisierungsmethode. Eine Methode zur Konvertierung von Objektdaten in ein anderes Format, wie z. B. Binärformat oder XML.

Servant-Region . Ein zusammenhängender Bereich des virtuellen Speichers, der dynamisch gestartet wird, wenn die Arbeitslast größer wird, und automatisch gestoppt wird, wenn die Arbeitslast geringer wird.

Server. Ein Softwareprogramm oder ein Computer, das bzw. der Services für andere Softwareprogramme oder Computer bereitstellt. Siehe auch Host.

Servercluster. Eine Gruppe von Servern, die in der Regel auf unterschiedlichen physischen Maschinen installiert sind und mit denselben Anwendungen konfiguriert sind, aber als ein logischer Server zusammenarbeiten.

Service-Level-Agreement (SLA). Ein Vertrag zwischen einem Kunden und einem Serviceanbieter, der die Erwartungen hinsichtlich des Service-Levels, die Verfügbarkeit, Leistungswerte und andere messbare Zielsetzungen betreffend, angibt.

Servicequalität (Quality of Service). Eine Gruppe von Kommunikationsmerkmalen, die eine Anwendung erfordert. Die Servicequalität definiert bestimmte Werte für die Übertragungspriorität, die Stufe der Weiterleitungszuverlässigkeit und die Sicherheitsstufe.

Servlet. Ein Java-Programm, das in einem Webserver ausgeführt wird und die Funktionen des Servers durch Generierung dynamischen Inhalts als Reaktion auf Webclientanforderungen erweitert. Servlets werden häufig verwendet, um Datenbanken mit dem Web zu verbinden.

Setter-Methode. Eine Methode, mit der ein Wert einer Instanz- oder Klassenvariablen manipuliert werden kann. Mit dieser Methode kann ein anderes Objekt den Wert einer seiner Variablen ermitteln.

Shard. (Engl., Scherbe) Eine Instanz einer Partition. Ein Shard kann eine primäre Instanz oder ein Replikat sein.

Shell-Script. Ein Programm oder Script, das von der Shell eines Betriebssystems interpretiert wird.

Sicherheitsadministrator. Die Person, die den Zugriff auf Geschäftsdaten und Programmfunktionen steuert.

Sicherheitstoken. Die Darstellung von Anforderungen, die ein Client stellt. Beispiele für derartige Anforderungen sind Name, Kennwort, Identität, Schlüssel, Zertifikat, Gruppe, Berechtigung usw.

Sitzung.

1. Eine logische oder virtuelle Verbindung zwischen zwei Stationen, Softwareprogrammen oder Einheiten in einem Netz, die die Kommunikation und den Datenaustausch zwischen diesen beiden Elementen ermöglicht.

2. Eine Reihe von Anforderungen an ein Servlet, die von demselben Benutzer und demselben Browser stammen.

3. In Java EE ein Objekt, mit dem ein Servlet die Interaktion eines Benutzers mit einer Webanwendung über mehrere HTTP-Anforderungen hinweg protokollieren kann.

Sitzungsaffinität. Eine Methode für die Konfiguration von Anwendungen, in denen ein Client immer mit demselben Server verbunden ist. Diese Konfigurationen inaktivieren das Workload Management nach der ersten Verbindung, weil eine Clientanforderung immer an denselben Server weitergeleitet werden muss.

Skalierbarkeit. Die Erweiterungsmöglichkeit eines Systems, wenn Ressourcen, wie Prozessoren, Hauptspeicher oder Speicher, hinzugefügt werden.

SLA. Siehe Service-Level-Agreement.

Software Development Kit (SDK). Eine Sammlung von Tools, APIs und Dokumentation, die den Entwickler bei der Entwicklung von Software in einer bestimmten Maschinensprache oder für eine bestimmte Betriebsumgebung unterstützt.

Speicherverlust. Die Auswirkung eines Programms, das Referenzen auf Objekte verwaltet, die nicht mehr benötigt werden und deshalb freigegeben werden müssen.

Sperre. Eine Methode, mit der verhindert werden kann, dass von einem Anwendungsprozess vorgenommene nicht festgeschriebene Änderungen von einem anderen Anwendungsprozess wahrgenommen werden oder dass ein Anwendungsprozess Daten aktualisiert, auf die ein anderer Prozess gerade zugreift. Eine Sperre gewährleistet die Integrität der Daten, indem sie verhindert, dass gleichzeitig angemeldete Benutzer auf inkonsistente Daten zugreifen.

SQL. Siehe Structured Query Language.

SQL-Abfrage. Eine Komponente bestimmter SQL-Anweisungen, von der eine Ergebnistabelle angegeben wird.

SSL. Siehe Secure Sockets Layer.

SSL-Channel. Ein Typ von Channel in einer Transportkette, der der Transportkette ein SSL-Konfigurationsrepertoire zuordnet.

Stapelspeicher. Ein Bereich im Hauptspeicher, in dem gewöhnlich Informationen, wie temporäre Registerdaten, Werte von Parametern und Rückkehradressen von Unterroutinen, gespeichert werden und der auf dem Prinzip LIFO (Last in, First out) basiert.

Statisch. Ein Schlüsselwort der Programmiersprache Java, mit dem eine Variable als Klassenvariable definiert wird.

Structured Query Language (SQL) . Eine standardisierte Sprache für die Definition und Bearbeitung von Daten in einer relationalen Datenbank.

Synchroner Prozess. Ein Prozess, der durch den Aufruf einer Anforderungs-/Antwortoperation gestartet wird. Das Ergebnis des Prozesses wird von derselben Operation zurückgegeben.

Synchrones Replikat. Ein Shard, das Aktualisierungen im Rahmen der Transaktion für das primäre Shard empfängt, um die Datenkonsistenz zu gewährleisten, wodurch die Antwortzeit im Vergleich mit einem asynchronen Replikat verbessert werden kann.

Synchronisieren. Das Hinzufügen, Entfernen oder Ändern einer Funktion oder eines Artefakts, damit sie/es einer anderen Funktion bzw. einem anderen Artefakt entspricht.

Syntax. Die Regeln zum Erstellen eines Befehls oder einer Anweisung.

Systemanalytiker. Ein Spezialist, der für die Umsetzung von Geschäftsanforderungen in Systemdefinitionen und Lösungen verantwortlich ist.

TCP. Siehe Transmission Control Protocol.

TCP-Channel . Ein Typ von Channel in einer Transportkette, der Clientanwendungen ermöglicht, persistente Verbindungen in einem lokalen Netz (LAN) zu verwenden.

TCP/IP. Siehe Transmission Control Protocol/Internet Protocol.

TCP/IP-Überwachungsserver. Eine Laufzeitumgebung, die alle Anforderungen und Antworten zwischen einem Webbbrowser und einem Anwendungsserver sowie TCP/IP-Aktivitäten überwacht.

Thin Application Client. Eine einfache, für den Download verfügbare Java-Anwendungslaufzeit, die mit Enterprise-Beans interagieren kann.

Thin Client. Ein Client, der über wenig oder keine installierte Software, aber über Zugriff auf Software verfügt, die von zugeordneten Netzservern verwaltet und bereitgestellt wird. Ein Thin Client ist eine Alternative zu einem Client mit vollem Funktionsumfang, wie beispielsweise einer Workstation.

Thread. Ein Datenstrom von Maschineninstruktionen, der von einem Prozess gesteuert wird. Bei manchen Betriebssystemen ist ein Thread die kleinste Operationseinheit in einem Prozess. Mehrere Threads können gleichzeitig laufen und dabei unterschiedliche Jobs ausführen.

Thread-Konflikt. Eine Bedingung, bei der ein Thread auf eine Sperre oder ein Objekt wartet, die bzw. das von einem anderen Thread gehalten wird.

Tivoli Performance Viewer. Ein Java-Client, der die PMI-Daten (Performance Monitoring Infrastructure) von einem Anwendungsserver abruft und sie in verschiedenen Formaten anzeigt.

Token.

1. Eine Markierung, mit der der aktuelle Status einer Prozessinstanz während eines Simulationslaufs protokolliert wird.

2. Eine bestimmte Nachricht oder ein bestimmtes Bitmuster, die bzw. das die Berechtigung oder temporäre Kontrolle für die Übertragung von Daten in einem Netz angibt.

Topologie. Die physische oder logische Zuordnung der Position der Netzbetriebskomponenten oder Knoten in einem Netz. Bus, Ring, Stern und Baumstruktur sind gängige Beispiele für Netztopologien.

Transaktion. Ein Prozess, bei dem alle Datenänderungen, die während einer Transaktion vorgenommen werden, zusammen als Einheit festgeschrieben oder als Einheit rückgängig gemacht werden.

Transmission Control Protocol/Internet Protocol (TCP/IP). Eine Gruppe von standardisierten, nicht proprietären Übertragungsprotokollen, die über verschiedene Typen miteinander verbundener Netze zuverlässige durchgängige Verbindungen zwischen Anwendungen zur Verfügung stellt.

Transmission Control Protocol (TCP). Ein Übertragungsprotokoll, das im Internet sowie in allen Netzen verwendet wird, die die IETF-Standards (IETF = Internet Engineering Task Force) für netzübergreifende Protokolle verwenden. TCP bietet ein zuverlässiges Host-to-Host-Protokoll in DFV-Netzen mit Paketvermittlung und in miteinander verbundenen Systemen solcher Netze.

Trennen von Webservern. Eine Topologie, in der der Webserver physisch vom Anwendungsserver getrennt ist.

Truststore-Datei. Eine Schlüsseldatenbankdatei, die die öffentlichen Schlüssel für eine vertrauenswürdige Entität enthält.

Typ.

1. In der Java-Programmierung eine Klasse oder Schnittstelle.

2. In einem WSDL-Dokument ein Element, das Datentypdefinitionen enthält und hierfür ein Typsystem verwendet (z. B. XSD).

Überwachungssignal. Ein Signal, das eine Entität an eine andere sendet, um anzugeben, dass sie noch immer aktiv ist.

UDDI. Siehe Universal Description, Discovery, and Integration.

Umgebung. Eine benannte Gruppe von logischen und physischen Ressourcen, die verwendet wird, um die Leistung einer Funktion zu verbessern.

Umgebungsvariable. Eine Variable, die angibt, wie ein Betriebssystem oder ein anderes Programm ausgeführt wird, oder die die vom Betriebssystem erkannten Einheiten festlegt.

Unbeaufsichtigte Installation. Eine Installation, die keine Nachrichten an die Konsole sendet, sondern Nachrichten und Fehler in Protokolldateien speichert. Bei einer unbeaufsichtigten Installation können Antwortdateien für die Dateneingabe verwendet werden.

Unbeaufsichtigter Modus. Eine Methode für die Installation oder Deinstallation einer Produktkomponente über die Befehlszeile, ohne dass eine grafische Benutzerschnittstelle angezeigt wird. Bei Verwendung des Befehlszeilenmodus gibt der Benutzer die für das Installations- oder Deinstallationsprogramm erforderlichen Daten direkt in der Befehlszeile oder in einer Datei (mit dem Namen Options- oder Antwortdatei) an.

Uniform Resource Identifier (URI).

1. Eine kompakte Zeichenfolge zum Identifizieren einer abstrakten oder physischen Ressource.

2. Eine eindeutige Adresse, die zum Identifizieren von Inhalten im Web (z. B. einer Textseite, einem Video- oder Tonclip, einem Standbild oder animierten Bild bzw. einem Programm) verwendet wird. Die häufigste Form von URI ist die Adresse einer Webseite, die eine Sonderform oder eine Untergruppe der URI darstellt und Uniform Resource Locator (URL) genannt wird. Eine URI beschreibt in der Regel, wie auf die Ressource, den Computer mit der Ressource und den Namen der Ressource (einen Dateinamen) auf dem Computer zugegriffen wird.

Uniform Resource Locator (URL). Die eindeutige Adresse einer Informationsressource, die in einem Netz wie dem Internet zugänglich ist. Der URL enthält das Namenskürzel für das Protokoll, mit dem auf die Informationsressource zugegriffen wird, und die Informationen, mit deren Hilfe das Protokoll die Informationsressource lokalisiert.

Uniform Resource Name (URN). Ein Name, der einen Web-Service eindeutig für einen Client identifiziert.

Universal Description, Discovery, and Integration (UDDI). Eine Reihe standardisierter Spezifikationen, mit denen Unternehmen und Anwendungen auf schnelle und einfache Weise Web-Services über das Internet finden und verwenden können.

Universally Unique Identifier (UUID). Die numerische 128-Bit-Kennung, mit der sichergestellt wird, dass zwei Komponenten nicht dieselbe Kennung aufweisen.

UNIX System Services. Ein Element von z/OS, das eine UNIX-Umgebung erstellt, die den Spezifikationen XPG4 UNIX 1995 entspricht und zwei Schnittstellen offener Systeme für das Betriebssystem z/OS bereitstellt: eine Anwendungsprogrammierschnittstelle (API = Application Programming Interface) und eine interaktive Shell-Schnittstelle.

Unterabfrage. In SQL ein Subselect in einem Prädikat. Beispiel: Eine Anweisung SELECT in der WHERE- oder HAVING-Klausel einer anderen SQL-Anweisung.

Unterbrechungspunkt. Ein markierter Punkt innerhalb eines Prozesses oder eines Programmablaufs, bei dessen Erreichen der Ablauf angehalten wird, um beispielsweise Debug- oder Überwachungsoperationen auszuführen.

Untergeordneter Knoten. Ein Knoten, der sich im Geltungsbereich eines anderen Knotens befindet.

Unterklasse. In Java eine Klasse, die über Vererbung aus einer bestimmten Klasse ableitet wird.

Unternehmensanwendungsprojekt. Eine Struktur und Hierarchie von Ordnern und Dateien, die einen Implementierungsdeskriptor, ein IBM Erweiterungsdokument sowie Dateien enthalten, die für alle im Implementierungsdeskriptor definierten Java EE-Module einheitlich sind.

Unternehmensarchiv (EAR). Ein besonderer Typ einer JAR-Datei, der durch den Java EE-Standard definiert ist und der zur Implementierung von Java EE-Anwendungen auf Java EE-Anwendungsservern verwendet wird. Eine EAR-Datei enthält EJB-Komponenten, einen Implementierungsdeskriptor sowie Webarchivdateien (WAR-Dateien) für einzelne Webanwendungen. Siehe auch Webarchiv.

URI. Siehe Uniform Resource Identifier.

URL. Siehe Uniform Resource Locator.

URL-Schema. Ein Format, das eine andere Objektreferenz enthält.

URN. Siehe Uniform Resource Name.

UUID . Siehe Universally Unique Identifier.

Variable. Eine Darstellung eines änderbaren Werts.

Veraltet. Dieser Begriff bezeichnet eine Entität, z. B. ein Programmierungselement oder eine Programmierungsfunktion, die zwar noch unterstützt wird, deren Verwendung aber nicht mehr empfohlen wird und die möglicherweise sehr bald nicht mehr aktuell sein wird.

Vererbung. Eine Technik in der objektorientierten Programmierung, bei der vorhandene Klassen als Basis für die Erstellung anderer Klassen verwendet werden. Durch die Vererbung umfassen die spezifischeren Elemente die Struktur und das Verhalten der allgemeineren Elemente.

Verfügbarkeit.

1. Die Bedingung, die Benutzern den Zugriff auf und die Verwendung ihrer Anwendungen und Daten ermöglich.

2. Die Zeiträume, während derer auf eine Ressource zugegriffen werden kann. Die Services eines Vertragsnehmers können beispielsweise werktags von 09.00 Uhr bis 17.00 Uhr sowie samstags von 09.00 Uhr bis 15.00 Uhr verfügbar sein.

Verknüpfung.

1. (Join) Ein Prozesselement zur Rekombination und Synchronisation von Parallelverarbeitungspfaden nach einer Entscheidung oder Verzweigung. Eine Verknüpfung wartet an allen Eingangszweigen auf Eingabedaten, bevor die Fortführung des Prozesses zugelassen wird.

2. Eine relationale SQL-Operation, in der Daten aus zwei Tabellen, gewöhnlich über eine Verknüpfungsbedingung, die die zu verknüpfenden Spalten angibt, abgerufen werden können.

3. Die Konfiguration einer Eingangsverbindung, die das Verhalten dieser Verbindung festlegt.

Version. Ein gesondert lizenziertes Programm, das gewöhnlich neuen wichtigen Code oder neue wichtige Funktionen enthält.

Verteiltes eXtreme Scale. Ein Verwendungsmuster für die Interaktion mit eXtreme Scale, wenn Server und Clients in mehreren Prozessen existieren.

Verzweigung. (Fork) Ein Prozesselement, das Kopien seiner Eingabe erstellt und diese gleichzeitig über mehrere Verarbeitungspfade weiterleitet.

Virtualisierung. Ein Verfahren, das die Merkmale von Ressourcen auf der Basis der Art und Weise kapselt, in der andere Systeme mit diesen Ressourcen interagieren.

Virtuelle Maschine. Eine abstrakte Spezifikation einer Datenverarbeitungseinheit, die auf verschiedene Arten in der Software und Hardware implementiert sein kann.

Virtueller Host. Eine Konfiguration, mit der ein einziges Hostsystem mehreren Hostsystemen ähneln kann. Ressourcen, die einem virtuellen Host zugeordnet wurden, können nicht Daten mit Ressourcen gemeinsam nutzen, die einem anderen virtuellen Host zugeordnet wurden, auch wenn sich die virtuellen Hosts auf demselben physischen System befinden.

Vitalität. Der allgemeine Zustand bzw. Status der Datenbankumgebung.

Vorgeordnet. Dieser Begriff bezeichnet die Richtung des Verarbeitungsablaufs. Dieser verläuft vom Start des Prozesses (vorgeordnet) zum Ende des Prozesses (nachgeordnet).

Vorläufige Programmkorrektur (Program Temporary Fix). Bei den Produktreihen IBM System i, IBM System p und IBM System z eine Programmkorrektur, die von IBM getestet wurde und allen Kunden zur Verfügung gestellt wird. Siehe auch Fixpack.

Vorläufiger Fix. Ein zertifizierter Fix, der für alle Kunden zwischen regulär geplanten Fixpacks, Refresh-Packs oder Releases zur Verfügung gestellt wird. Siehe auch Fixpack.

Waiter. Ein Thread, der auf eine Verbindung wartet.

WAR . Siehe Webarchiv.

Wartungsmodus. Ein Status eines Knotens oder Servers, den Administratoren für dessen Diagnose, Wartung oder Optimierung verwenden können, ohne dass eingehender Datenverkehr in einer Produktionsumgebung unterbrochen wird.

WCCM . Siehe WebSphere Common Configuration Model.

Webarchiv (WAR). Ein komprimiertes Dateiformat, das vom Java EE-Standard definiert wird und mit dem alle Ressourcen, die zur Installation und Ausführung einer Webanwendung erforderlich sind, in einer einzigen Datei gespeichert werden können. Siehe auch Unternehmensarchiv.

Webbrowser. Ein Clientprogramm, das Anforderungen an einen Webserver einleitet und die Informationen anzeigt, die der Server zurückgibt.

Webcontainer. Ein Container, der den Webkomponentenvertrag der Java EE-Architektur implementiert. (Sun)

Webcontainer-Channel. Ein Typ von Channel in einer Transportkette, der eine Brücke in der Transportkette zwischen einem eingehenden HTTP-Channel und einer Servlet- oder JSP-Steuerkomponente erstellt.

Web-Crawler. Ein Crawler-Typ, der das Web durch Abrufen eines Webdokuments und Verfolgen der Links in diesem Dokument erkundet.

Webkomponente. Ein Servlet, eine JSP-Datei oder eine HTML-Datei. Eine oder mehrere Webkomponenten bilden ein Webmodul.

Webserver. Ein Softwareprogramm, das HTTP-Anforderungen verarbeiten kann.

Webserver-Plug-in. Ein Softwaremodul, das den Webserver bei der Weiterleitung von Anforderungen für dynamischen Inhalt, wie z. B. Servlets, an den Anwendungsserver unterstützt.

Website. Eine zusammengehörige Sammlung von im Web verfügbaren Dateien, die von einer einzigen Entität (einer Organisation oder einer Einzelperson) verwaltet wird und Hypertextinformationen für die Benutzer enthält. Eine Website enthält häufig Hypertext-Links zu anderen Websites.

WebSphere. Eine IBM Marke, die Tools für die Entwicklung von e-business-Anwendungen und Middleware für die Ausführung von Webanwendungen enthält.

WebSphere Common Configuration Model (WCCM) . Ein Modell, das programmgestützten Zugriff auf Konfigurationsdaten bietet.

What You See Is What You Get (WYSIWYG). Eine Funktionalität eines Editors, mit der Seiten ständig genau so angezeigt werden, wie sie gedruckt bzw. auf andere Weise wiedergegeben werden.

While-Schleife. Eine Schleife, die die gleiche Aktivitätenfolge so lange wiederholt, bis eine bestimmte Bedingung erfüllt ist. Die While-Schleife testet die zugehörige Bedingung zu Beginn jeder Schleife. Wenn die Bedingung von Beginn an nicht zutrifft, wird die Aktivitätenfolge in der Schleife nie ausgeführt.

WLM. Siehe Workload Manager.

Workload-Management. Die Optimierung der Verteilung eingehender Verarbeitungsaufträge an die Anwendungsserver, Enterprise-Beans, Servlets und anderen Objekte, die den jeweiligen Auftrag effektiv verarbeiten können.

Workload Manager (WLM). Eine Komponente von z/OS, mit der mehrere Workloads gleichzeitig in einem einzigen z/OS-Image oder über mehrere Images hinweg ausgeführt werden können.

Write-Behind-Cache. Ein Cache, der jede Schreiboperation asynchron über ein Ladeprogramm in die Datenbank schreibt.

Write-Through-Cache. (Durchschreibcache) Ein Cache, der jede Schreiboperation synchron über ein Ladeprogramm in die Datenbank schreibt.

WYSIWYG. Siehe What You See Is What You Get.

XA. Eine bidirektionale Schnittstelle zwischen einem oder mehreren Ressourcenmanagern, die den Zugriff auf gemeinsam genutzte Ressourcen ermöglichen, und einem Transaktionsmanager, der Transaktionen überwacht und auflöst.

XML. Siehe Extensible Markup Language.

X/Open XA. Die Schnittstelle X/Open Distributed Transaction Processing XA. Ein empfohlener Standard für die Kommunikation bei verteilten Transaktionen. Der Standard definiert eine bidirektionale Schnittstelle zwischen Ressourcenmanager, die den Zugriff auf gemeinsam genutzte Ressourcen in Transaktionen ermöglichen, und einem Transaktionsservice, der Transaktionen überwacht und auflöst.

Zeichenfolge. In Programmiersprachen das Datenformat, das für das Speichern und Bearbeiten von Text verwendet wird.

Zeitbindung. Eine angepasste Validierungsaktion, die zum Messen der Dauer eines Methodenaufrufs oder einer Folge von Methodenaufrufen verwendet wird.

Zeitgeber. Eine Task, die zu bestimmten Zeitpunkten eine Ausgabe generiert.

Zeitlimit. Ein Zeitintervall, das zugeordnet wird, damit ein Ereignis stattfinden oder beendet werden kann, bevor die Operation unterbrochen wird.

Zelle.

1. Eine Gruppe verwalteter Prozesse, die in denselben Deployment Manager eingebunden sind und Stammgruppen mit hoher Verfügbarkeit umfassen können.

2. Mindestens ein Prozess, der als Host für Laufzeitkomponenten eingesetzt wird. Jede Zelle hat mindestens eine benannte Stammgruppe.

Zertifikatssammelspeicher. Eine Gruppe vorläufiger Zertifikate oder Zertifikatwiderruflisten (CRL, Certificate Revocation List), die von einem Zertifikatpfad verwendet wird, um eine Zertifizierungskette für die Validierung zu erstellen.

Zonenbasierte Unterstützung. Eine Funktion, die die regelbasierte Positionierung von Shards ermöglicht, um die Grid-Verfügbarkeit durch Positionierung der Shards in unterschiedlichen Rechenzentren (auf verschiedenen Stockwerken oder sogar in verschiedenen Gebäuden oder Regionen) zu verbessern.

z/OS. Ein IBM Betriebssystem für Großrechner, bei dem 64-Bit-Realspeicher verwendet wird.

Bemerkungen

Hinweise auf IBM Produkte, Programme und Services in dieser Veröffentlichung bedeuten nicht, dass IBM diese in allen Ländern, in denen IBM vertreten ist, anbietet. Hinweise auf IBM Lizenzprogramme oder andere IBM Produkte bedeuten nicht, dass nur Programme, Produkte oder Services von IBM verwendet werden können. An Stelle der IBM Produkte, Programme oder Services können auch andere ihnen äquivalente Produkte, Programme oder Services verwendet werden, solange diese keine gewerblichen oder anderen Schutzrechte der IBM verletzen. Die Verantwortung für den Betrieb der Produkte, Programme oder Services in Verbindung mit Fremdprodukten und Fremdservices liegt beim Kunden, soweit solche Verbindungen nicht ausdrücklich von IBM bestätigt sind.

Für in diesem Dokument beschriebene Erzeugnisse und Verfahren kann es IBM Patente oder Patentanmeldungen geben. Mit der Auslieferung dieses Handbuchs ist keine Lizenzierung dieser Patente verbunden. Lizenzanforderungen sind schriftlich an folgende Adresse zu richten (Anfragen an diese Adresse müssen auf Englisch formuliert werden):

IBM Director of Licensing
IBM Europe, Middle East & Africa
Tour Descartes
2, avenue Gambetta
92066 Paris La Defense
France

Lizenznehmer des Programms, die Informationen zu diesem Produkt wünschen mit der Zielsetzung: (i) den Austausch von Informationen zwischen unabhängigen, erstellten Programmen und anderen Programmen (einschließlich des vorliegenden Programms) sowie (ii) die gemeinsame Nutzung der ausgetauschten Informationen zu ermöglichen, wenden sich an folgende Adresse:

IBM Corporation
Mail Station P300
522 South Road
Poughkeepsie, NY 12601-5400
USA
Attention: Information Requests

Die Bereitstellung dieser Informationen kann unter Umständen von bestimmten Bedingungen - in einigen Fällen auch von der Zahlung einer Gebühr - abhängig sein.

Marken

Folgende Namen sind Marken der IBM Corporation in den USA und/oder anderen Ländern:

- AIX
- CICS
- Cloudscape
- DB2
- Domino
- IBM
- Lotus
- RACF
- Redbooks
- Tivoli
- WebSphere
- z/OS

Java und alle auf Java basierenden Marken und Logos sind Marken von Sun Microsystems, Inc. in den USA und/oder anderen Ländern.

LINUX ist eine Marke von Linus Torvalds in den USA und/oder anderen Ländern.

Microsoft, Windows, Windows NT und das Windows-Logo sind Marken der Microsoft Corporation in den USA und/oder anderen Ländern.

UNIX ist eine eingetragene Marke von The Open Group in den USA und anderen Ländern.

Weitere Unternehmens-, Produkt- oder Servicenamen können Marken anderer Hersteller sein.

Index

T

U

V

W

Z

Gedruckt in Deutschland