IBM WebSphere eXtreme Scale バージョン 7.1.1

バージョン 7 リリース 1

製品概要

2011 年 11 月 21 日

本書は、WebSphere eXtreme Scale バージョン 7 リリース 1 モディフィケーション 1、および新しい版で明記されていない限り、以降のすべてのリリースおよびモディフィケーションに適用されます。

お客様の環境によっては、資料中の円記号がバックスラッシュと表示されたり、バックスラッシュが円記号と表示されたりする場合があります。

原典： WebSphere® eXtreme Scale Version7.1.1
Version 7 Release 1
Product Overview
November 21, 2011

発行： 日本アイ・ビー・エム株式会社

担当： トランスレーション・サービス・センター

第1刷 2011.12

目次

図

表

製品概要 情報

WebSphere® eXtreme Scale の資料セットには、WebSphere eXtreme Scale 製品の使用、プログラミング、および管理に必要な情報を提供する 3 つのボリュームがあります。

WebSphere eXtreme Scale ライブラリー

WebSphere eXtreme Scale ライブラリーには、以下の資料が含まれます。

- *製品概要* には、ユース・ケース・シナリオ、およびチュートリアルなど、WebSphere eXtreme Scale 概念の高水準の観点が含まれます。
- 「*インストール・ガイド*」では、WebSphere eXtreme Scale の一般的なトポロジーをインストールする方法について説明しています。
- *管理ガイド* には、アプリケーション・デプロイメント計画の作成方法、容量計画の作成方法、製品のインストールと構成方法、サーバーの始動と停止方法、環境のモニター方法、環境の保護方法など、システム管理者に必要な情報が含まれます。
- *プログラミング・ガイド* には、掲載されている API 情報を使用して WebSphere eXtreme Scale 用のアプリケーションを開発する方法に関する、アプリケーション開発者のための情報が含まれます。

これらの資料をダウンロードするには、WebSphere eXtreme Scale ライブラリー・ページにアクセスしてください。

このライブラリーと同じ情報は、WebSphere eXtreme Scaleバージョン 7.1.1 インフォメーション・センターからも入手することができます。

オフラインでのブックの使用

WebSphere eXtreme Scale ライブラリー内のすべてのブックには、インフォメーション・センターへのリンクが含まれており、ルート URL は `http://publib.boulder.ibm.com/infocenter/wxsinfo/v7r1m1` です。これらのリンクを使用して、関連情報に直接アクセスできます。ただし、オフラインで作業していてこれらのリンクのいずれかを見つけた場合は、ライブラリー内の他のブックでそのリンクのタイトルを検索できます。 API 資料、用語集、およびメッセージ解説書は、PDF ブックでは用意されていません。

本書の対象者

本書は、WebSphere eXtreme Scale の学習に関心をお持ちの方々を対象にしています。

本書の更新の取得

本書の更新は、WebSphere eXtreme Scale ライブラリー・ページから最新のバージョンをダウンロードすることで取得できます。

第 1 章 製品概要

WebSphere eXtreme Scale は、柔軟性があってスケーラブルな、メモリー内データ・グリッドです。データ・グリッドは、複数のサーバーにまたがって、アプリケーション・データおよびビジネス・ロジックを動的にキャッシュに入れ、区画化し、複製し、管理します。WebSphere eXtreme Scale は、高い効率性と直線的に伸びるスケーラビリティーを備えた、大量のトランザクション処理を実現します。WebSphere eXtreme Scale を使用すると、トランザクションの整合性、高可用性、予測可能な応答時間などの高いサービス品質も手に入れることができます。

WebSphere eXtreme Scale概要

WebSphere eXtreme Scale は、柔軟性があってスケーラブルな、メモリー内データ・グリッドです。データ・グリッドは、複数のサーバーにまたがって、アプリケーション・データおよびビジネス・ロジックを動的にキャッシュに入れ、区画化し、複製し、管理します。WebSphere eXtreme Scale は、高い効率性と直線的に伸びるスケーラビリティーを備えた、大量のトランザクション処理を実現します。 WebSphere eXtreme Scale を使用すると、トランザクションの整合性、高可用性、予測可能な応答時間などの高いサービス品質も手に入れることができます。

WebSphere eXtreme Scale の使用方法はいくつかあります。非常に強力なキャッシュとしても、アプリケーション状態を管理するメモリー内データベース処理スペースとしても、Extreme Transaction Processing (XTP) アプリケーションを構築するためにも、この製品を使用することができます。これらの XTP 機能には、最もビジネス上重要で要求が厳しいアプリケーションをサポートするためのアプリケーション・インフラストラクチャーが組み込まれています。

柔軟性のあるスケーラビリティー

分散オブジェクト・キャッシュを使用することによって、柔軟性のあるスケーラビリティーが可能になります。 柔軟性のあるスケーラビリティーによって、データ・グリッドはそれ自体をモニターし、管理します。 データ・グリッドは、トポロジーへのサーバーの追加または削除を行うことができます。これにより、必要に応じてメモリー、ネットワーク・スループット、および処理能力の増加や減少を行います。 スケールアウト処理が開始すると、データ・グリッドの実行中に、再始動を必要とせずに、キャパシティーが追加されます。 逆に、スケールイン処理は即時にキャパシティーを除去します。 データ・グリッドは、障害から自動的にリカバリーすることで自己修復も行います。

WebSphere eXtreme Scale とメモリー内データベースとの比較

WebSphere eXtreme Scale を実際のメモリー内データベースとみなすことはできません。メモリー内データベースは、単純すぎて WebSphere eXtreme Scale が管理できる複雑さを取り扱えない場合があります。メモリー内データベースは、サーバーに障害が起きても、その問題を修正できません。環境全体がその 1 台のサーバー上にある場合、障害が悲惨な結果を招く場合があります。

この種の障害に関する問題に対処するため、eXtreme Scale は所与のデータ・セットを、コンストレインド・ツリー・スキーマに相当する区画に分割します。 コンストレインド・ツリー・スキーマは、エンティティー間のリレーションシップを記述します。 区画を使用している場合、エンティティー・リレーションシップはツリー・データ構造をモデル化します。この構造では、ツリーの先頭がルート・エンティティーで、区画化される唯一のエンティティーです。ルート・エンティティーの他の子はすべて、ルート・エンティティーと同一の区画に保管されています。それぞれの区画は、プライマリー・コピーまたは断片として存在します。区画には、データのバックアップ用レプリカ断片も含まれます。メモリー内データベースは、このように構造化されておらず、動的でもないため、この種の機能を提供できません。メモリー内データベースを使用している場合は、WebSphere eXtreme Scale が自動的に行う操作を実装する必要があります。メモリー内データベースでは SQL 操作を実行でき、メモリー内でないデータベースと比較して処理速度が上がります。
WebSphere eXtreme Scale には、SQL サポートの代わりに独自の照会言語があります。この照会言語は、より柔軟性があり、データの区画化を可能にし、信頼性の高い障害リカバリーを提供します。

データベースを使用した WebSphere eXtreme Scale

後書きキャッシュ機能により、WebSphere eXtreme Scale はデータベースのフロントエンド・キャッシュとして働くことができます。 このフロントエンド・キャッシュを使用して、データベースの負荷と競合を減らしながら、スループットが増加します。 WebSphere eXtreme Scale は、予測可能な処理コストで予測可能なスケールインおよびスケールアウトを提供します。

以下の図は、コヒーレントな分散キャッシュ環境で、eXtreme Scale クライアントとデータ・グリッドとの間でデータがやり取りされることを示しています。データ・グリッドは、バックエンド・データ・ストアと自動的に同期できます。すべてのクライアントがキャッシュ内の同じデータを見るので、キャッシュはコヒーレントです。データの各部分は、キャッシュ内の 1 つだけの書き込み可能サーバーに保管されます。データの各部分の 1 つのコピーがあることで、さまざまなバージョンのデータを保管することになりかねない、レコードの無駄なコピーを防止します。コヒーレントなキャッシュは、より多くのサーバーがデータ・グリッドに追加されるにつれて、より多くのデータを保持し、データ・グリッドのサイズが増えるのにつれて直線的に増加します。耐障害性を強化するため、オプションでデータを複製することもできます。

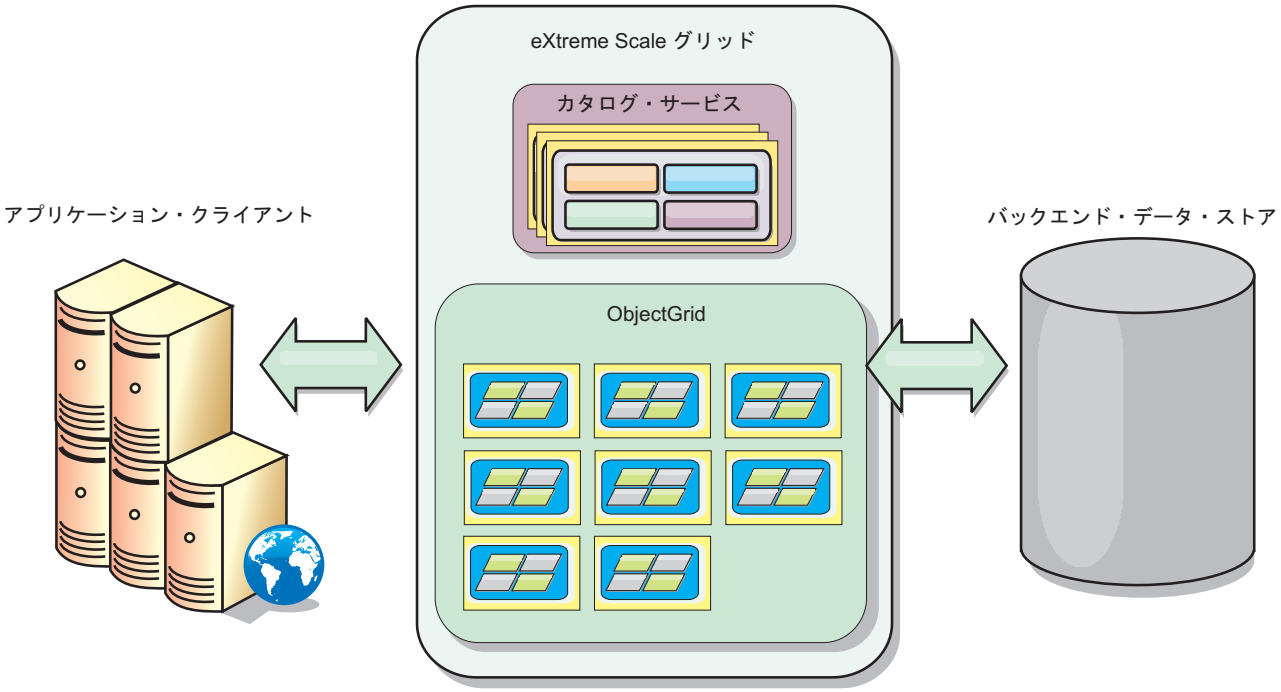


図 1. ハイレベル・トポロジー

WebSphere eXtreme Scale には、メモリー内データ・グリッドを提供するコンテナー・サーバー と呼ばれるサーバーがあります。このサーバーは、WebSphere Application Server 内部でも、単純な Java Standard Edition (J2SE) Java 仮想マシン上でも実行できます。単一の物理サーバー上で複数のコンテナー・サーバーを実行することが可能です。 結果として、メモリー内データ・グリッドは大きくなる場合があります。データ・グリッドは、アプリケーションやアプリケーション・サーバーのメモリーまたはアドレス・スペースによる制限はなく、それらに影響を与えることはありません。多数の物理サーバー上で稼働する何千、何万の Java 仮想マシンのメモリーを合計したメモリーを使用できます。

WebSphere eXtreme Scale を、メモリー内データベース処理スペースとして、ディスク、データベース、またはその両方でバックアップすることができます。

eXtreme Scale にはいくつかの Java API が用意されていますが、多くの使用状況ではユーザー・プログラミングは必要なく、ユーザーの WebSphere インフラストラクチャーでの構成とデプロイを行えばいいだけです。

データ・グリッドの概説

最も単純な eXtreme Scale プログラミング・インターフェースは、ObjectMap インターフェースです。これは単純なマップ・インターフェースであり、キャッシュに値を入れる map.put(key,value) メソッドと、後で値を取り出す map.get(key) メソッドを含んでいます。

データ・グリッドの基礎となるパラダイムは、キーと値のペアです。データ・グリッドは値 (Java オブジェクト) を、関連付けられたキー (別の Java オブジェクト) と一緒に保管します。その後、キーを使用して値が取り出されます。eXtreme Scale では、マップはこのようなキーと値のペアのエントリーで構成されます。

WebSphere eXtreme Scale は、単一の 1 つだけのローカル・キャッシュから、複数の Java 仮想マシンまたはサーバーを使用する大容量の分散キャッシュにいたるまで、多くのデータ・グリッド構成を提供します。

単純な Java オブジェクトを保管することに加えて、リレーションシップを持つオブジェクトを保管することもできます。 `SELECT ... FROM ... WHERE` ステートメントを使用する SQL に似た照会言語を使用して、これらのオブジェクトを取り出すことができます。 例えば、1 つの注文オブジェクトが 1 つの顧客オブジェクトを持ち、複数の品目オブジェクトをその注文オブジェクトに関連付けるという例が考えられます。 WebSphere eXtreme Scale は、1 対 1、1 対多、多対 1、および多対多のリレーションシップをサポートします。

WebSphere eXtreme Scale1 は、キャッシュにエンティティーを保管するための、EntityManager プログラミング・インターフェースもサポートします。 このプログラミング・インターフェースは、Java Enterprise Edition のエンティティーによく似ています。エンティティー・リレーションシップは、エンティティー記述子 XML ファイルから、または Java クラス内のアノテーションから、自動的に検出できます。EntityManager インターフェース上で find メソッドを使用して、1 次キーによってキャッシュからエンティティーを取り出すことができます。エンティティーをデータ・グリッドにパーシストすること、またはデータ・グリッドから除去することが 1 トランザクション境界内で実行できます。

単純な英字名がキーになっている分散例を考えてみましょう。キー別にキャッシュを 4 つの区画に分割できます。区画 1 は A から E で始まるキー、区画 2 は F から L で始まるキーに対応し、以下同様にします。可用性のため、1 つの区画が 1 つのプライマリー断片と 1 つのレプリカ断片を持つことにします。キャッシュ・データに対する変更は、プライマリー断片に対して行われ、レプリカ断片に複製されます。データ・グリッド・データを収容するサーバーの数を構成します。eXtreme Scale は、これらのサーバー・インスタンス全部を対象にして断片にデータを分配します。可用性のため、レプリカ断片はプライマリー断片とは別の物理サーバーに置かれます。

WebSphere eXtreme Scale は、カタログ・サービスを使用して、各キーのプライマリー断片を配置します。物理サーバーに障害が起こり、後でリカバリーするときに、eXtreme Scale サーバー間での断片の移動を扱います。例えば、レプリカ断片を含んでいるサーバーに障害が起こった場合、eXtreme Scale は新しいレプリカ断片を割り振ります。 プライマリー断片を含んでいるサーバーに障害が起こった場合は、レプリカ断片がプロモートされてプライマリー断片になります。上記と同じように、新しいレプリカ断片が作成されます。

バージョン 7.1.1 の新機能

WebSphere eXtreme Scale は、バージョン 7.1.1 で多くの新機能を組み込みました。このトピックでは、最新の製品アップデートについて説明しています。

DataSerializer プラグイン

クライアントとサーバーが情報を交換するとき、またはサーバーがあるサーバーから別のサーバーにデータを複製するときは、ネットワークを介してデータを送信できるようにするために、データを変換、すなわちシリアライズする必要がありま

す。これまでのリリースでは、データをシリアライズするために、デフォルトの Java シリアライゼーションまたは ObjectTransformer プラグインのどちらかを使用しました。このリリースでは、製品が特定のオブジェクト形式を必要とせずにバイト配列と対話できるように、WebSphere eXtreme Scale に対して、DataSerializer プラグインを使用して、シリアライゼーション形式つまりバイト配列を効率良く記述することができます。 詳細...

OSGi フレームワーク

OSGi フレームワークを使用して、プラグインを OSGi サービスとして公開できます。これにより、eXtreme Scale ランタイムがプラグインを使用できるようになります。さらに、eXtreme Scale サーバーおよびクライアントを OSGi コンテナー内で始動できます。これにより、ランタイム環境への eXtreme Scale プラグインの追加や eXtreme Scale プラグインの更新を動的に行うことができます。 詳細...

動的キャッシュ・プロバイダーのパフォーマンスの向上

WebSphere eXtreme Scale 動的キャッシュ・プロバイダー内の無効化処理が改善されました。invalidate(key, wait) メソッドの **wait** パラメーターが `false` 値に設定されると、無効化要求は、非同期にバッチで処理されます。この機能拡張により、パフォーマンスが著しく向上します。詳細...

デフォルトの配置の振る舞いの変更

これまでのリリースでは、新しいコンテナー・サーバーがデータ・グリッドで始動されると、そのコンテナー・サーバーでの断片の配置は即座に開始されました。この即時配置のために、新しいコンテナー・サーバーを含む、サーバーのプロセッサー使用率が高くなっていました。デフォルトの振る舞いは、配置の開始を 15000 ミリ秒 (15 秒) 遅らせるという設定に変更されました。配置の時間間隔は、**placementDeferralInterval** サーバー・プロパティーで変更できます。詳細...

Java Persistence API (JPA) レベル 2 (L2) キャッシュ・プラグインのイントラドメイン・トポロジーの構成

JPA L2 キャッシュにイントラドメイン・トポロジーを構成することにより、プライマリー断片は、構成のすべてのコンテナー・サーバーに配置されます。それぞれのプライマリー断片は、区画の全体の内容を含みます。この構成を使用することで、クライアントはローカルでデータにアクセスでき、かつどのプライマリー断片についてもデータ・グリッドへの書き込みが可能であるため、パフォーマンスを向上させることができます。詳細...

xscmd ユーティリティー

xscmd ユーティリティーは、**xsadmin** ユーティリティーの新しくサポートされたバージョンです。 **xsadmin** ユーティリティーは、これまでのリリースで、サポートされないサンプルとして含まれていました。詳細...

ログ分析レポートを生成するためのツール

新しい **xsloganalyzer** ツールを使用すると、ログ・ファイルからレポートを生成できます。生成されたレポートは、使用している環境のパフォーマンスの分析や問題

のトラブルシューティングに役立ちます。 詳細...

IBM eXtremeIO および IBM eXtremeMemory

eXtremeMemory を構成することにより、オブジェクトを Java ヒープでなくネイティブ・メモリーに保管できます。eXtremeMemory を構成すると、新しいトランスポート・メカニズムである eXtremeIO が使用可能になります。オブジェクトを Java ヒープから移動することで、ガーベッジ・コレクションに伴う一時停止を回避でき、より安定したパフォーマンスを得られるうえ、応答時間も予測可能になります。 詳細...

WebSphere Application Server バージョン 8 サポート

WebSphere eXtreme Scale バージョン 7.1.1 は、WebSphere Application Server および WebSphere Application Server Network Deployment バージョン 8 にインストールできるようになりました。詳細...

リリース情報

製品のサポート Web サイト、製品資料、および製品の最新の更新、制限、および既知の問題へのリンクが提供されています。

- 『最新の更新、制限、および既知の問題へのアクセス』
- 『システムのアクセスおよびソフトウェア要件』
- 7 ページの『製品資料へのアクセス』
- 7 ページの『製品のサポート Web サイトへのアクセス』
- 7 ページの『IBM ソフトウェア・サポートへの連絡』

最新の更新、制限、および既知の問題へのアクセス

リリース情報は、製品のサポート・サイトの技術情報から入手できます。WebSphere eXtreme Scale のすべての技術情報のリストを参照するには、サポート Web ページにアクセスしてください。ここで提供されているリンクをクリックすると、サポート Web ページで関連リリース・ノートが検索されます。関連リリース・ノートはリストとして返されます。

- 7.1.1+ バージョン 7.1.1 のリリース情報のリストを参照するには、サポート Web ページにアクセスしてください。
- バージョン 7.1 のリリース情報のリストを参照するには、サポート Web ページにアクセスしてください。
- バージョン 7.0 のリリース情報のリストを参照するには、サポート Web ページにアクセスしてください。
- バージョン 6.1 のリリース情報のリストを参照するには、リリース情報ウィキ・ページにアクセスしてください。

システムのアクセスおよびソフトウェア要件

ハードウェアおよびソフトウェア要件は以下のページに記載されています。

- システム要件の詳細

製品資料へのアクセス

情報セット全体に関しては、ライブラリー・ページにアクセスしてください。

製品のサポート Web サイトへのアクセス

最新の技術情報、ダウンロード、フィックス、およびその他のサポート関連情報を検索するには、サポート・ポータルにアクセスしてください。

IBM® ソフトウェア・サポートへの連絡

この製品で問題が発生した場合には、最初に以下のアクションを試行してください。

- 製品資料に記載されているステップを実行します。
- 関連資料をオンライン・ヘルプで検索します。
- エラー・メッセージをメッセージ解説書で検索します。

上記の方法で問題を解決できない場合、IBM テクニカル・サポートに連絡してください。

ハードウェアおよびソフトウェアの要件

ハードウェア要件およびオペレーティング・システム要件の概要をご覧ください。WebSphere eXtreme Scale に対して使用するハードウェアまたはオペレーティング・システムのレベルについて、特定のレベルの要件はありませんが、公式にサポートされるハードウェアおよびソフトウェアのオプションは、製品サポート・サイトの「システム要件」ページから入手できます。インフォメーション・センターの情報と「システム要件」ページの情報に違いがある場合は、Web サイトの情報を優先してください。インフォメーション・センターの前提条件の情報は、便宜上提供されているだけです。

ハードウェアおよびソフトウェア要件の正式なセットについては、システム要件ページを参照してください。

eXtreme Scale のインストールおよびデプロイ先として、オペレーティング・システムの特定レベルの要件はありません。 Java Platform, Standard Edition (Java SE) および Java Platform, Enterprise Edition (Java EE) のそれぞれのインストールでは、異なるオペレーティング・システムのレベルまたはフィックスが必要です。

この製品は、Java EE および Java SE 環境にインストールしてデプロイできます。また、クライアント・コンポーネントを WebSphere Application Server に統合せずに、直接 Java EE アプリケーションにバンドルすることができます。 WebSphere eXtreme Scale は、Java SE 5 以降、および WebSphere Application Server バージョン 6.1 以降をサポートしています。

ハードウェア要件

WebSphere eXtreme Scale では、ハードウェアの具体的なレベルの要件はありません。ハードウェア要件は、WebSphere eXtreme Scale を実行するのに使用される Java Platform, Standard Edition のインストール済み環境でサポートされるハードウェアによって異なります。 eXtreme Scale を WebSphere Application Server または

別の Java Platform, Enterprise Edition 実装環境で使用する場合、これらのプラットフォームのハードウェア要件は WebSphere eXtreme Scale にとって十分です。

オペレーティング・システム要件

- **Web コンソールを使用しない場合**

  eXtreme Scale では、オペレーティング・システムの具体的なレベルの要件はありません。各 Java SE および Java EE 実装は、それぞれ異なるオペレーティング・システム・レベル、または、Java 実装のテスト中に発見された問題に対するフィックスを必要とします。これらの実装に必要なレベルは、eXtreme Scale にとって十分です。

- **Web コンソールを使用する場合**

  コンソールを使用する場合、それぞれのオペレーティング・システムについて以下の要件が適用されます。

  – Linux: 32 ビットまたは 64 ビット JVM
  – Linux PPC: 32 ビット JVM のみ
  – Windows: 32 ビット JVM のみ
  – AIX®: 32 ビット JVM のみ

Web ブラウザー要件

Web コンソールは、以下の Web ブラウザーをサポートしています。

- Mozilla Firefox、バージョン 3.5.x 以降
- Mozilla Firefox、バージョン 3.6.x 以降
- Microsoft Internet Explorer バージョン 7 または 8

WebSphere Application Server 要件

- WebSphere Application Server バージョン 6.1.0.39 以降
- WebSphere Application Server バージョン 7.0.0.19 以降
- WebSphere Application Server バージョン 8.0.0.1 以降

詳しくは、WebSphere Application Server の推奨フィックスを参照してください。

その他のアプリケーション・サーバー要件

その他の Java EE 実装は、ローカル・インスタンスとして、または、eXtreme Scale サーバーへのクライアントとして、eXtreme Scale ランタイムを使用できます。Java SE を実装する場合は、バージョン 5 以降を使用する必要があります。

ディレクトリー規則

wxs_install_root や *wxs_home* など、参照が必要な特別のディレクトリーに対して、資料全体で、次のディレクトリー規則が使用されます。インストール中、およびコマンド行ツールの使用時も含めて、さまざまなシナリオで、これらのディレクトリーにアクセスします。

wxs_install_root
wxs_install_root ディレクトリーは、WebSphere eXtreme Scale 製品ファイル

がインストールされているルート・ディレクトリーです。*wxs_install_root* ディレクトリーは、試用版のアーカイブが解凍されたディレクトリー、または WebSphere eXtreme Scale 製品がインストールされているディレクトリーの可能性があります。

- 試用版を解凍した場合の例:

  例: /opt/IBM/WebSphere/eXtremeScale

- WebSphere eXtreme Scale がスタンドアロン・ディレクトリーにインストールされている場合の例:

  例: /opt/IBM/eXtremeScale

- WebSphere eXtreme Scale が WebSphere Application Server に統合されている場合の例:

  例: /opt/IBM/WebSphere/AppServer

wxs_home

wxs_home ディレクトリーは、WebSphere eXtreme Scale 製品ライブラリー、サンプル、およびコンポーネントのルート・ディレクトリーです。このディレクトリーは、試用版を解凍した場合は、*wxs_install_root* ディレクトリーと同じです。スタンドアロンのインストール済み環境の場合、*wxs_home* ディレクトリーは、*wxs_install_root* ディレクトリー内の ObjectGrid サブディレクトリーです。WebSphere Application Server に統合されているインストール済み環境の場合、このディレクトリーは、*wxs_install_root* ディレクトリー内の optionalLibraries/ObjectGrid ディレクトリーです。

- 試用版を解凍した場合の例:

  例: /opt/IBM/WebSphere/eXtremeScale

- WebSphere eXtreme Scale がスタンドアロン・ディレクトリーにインストールされている場合の例:

  例: /opt/IBM/eXtremeScale/ObjectGrid

- WebSphere eXtreme Scale が WebSphere Application Server に統合されている場合の例:

  例: /opt/IBM/WebSphere/AppServer/optionalLibraries/ObjectGrid

was_root

was_root ディレクトリーは、WebSphere Application Server インストール済み環境のルート・ディレクトリーです。

例: /opt/IBM/WebSphere/AppServer

restservice_home

restservice_home ディレクトリーは、WebSphere eXtreme Scale REST データ・サービスのライブラリーおよびサンプルが配置されるディレクトリーです。このディレクトリーは restservice という名前で、*wxs_home* ディレクトリー内のサブディレクトリーです。

- スタンドアロン・デプロイメントの場合の例:

  例: /opt/IBM/WebSphere/eXtremeScale/ObjectGrid/restservice

- WebSphere Application Server 統合デプロイメントの場合の例:

  例: /opt/IBM/WebSphere/AppServer/optionalLibraries/ObjectGrid/restservice

tomcat_root
tomcat_root は、Apache Tomcat インストール済み環境のルート・ディレクトリーです。

例: /opt/tomcat5.5

wasce_root
wasce_root は、WebSphere Application Server Community Edition インストール済み環境のルート・ディレクトリーです。

例: /opt/IBM/WebSphere/AppServerCE

java_home
java_home は、Java Runtime Environment (JRE) インストール済み環境のルート・ディレクトリーです。

例: /opt/IBM/WebSphere/eXtremeScale/java

samples_home
samples_home は、チュートリアルに使用するサンプル・ファイルを解凍したディレクトリーです。

例: /wxs-samples/

dvd_root
dvd_root ディレクトリーは、製品が含まれた DVD のルート・ディレクトリーです。

例: dvd_root/docs/

equinox_root
equinox_root ディレクトリーは、Eclipse Equinox OSGi フレームワークのインストール済み環境のルート・ディレクトリーです。

例: /opt/equinox

user_home
user_home ディレクトリーは、ユーザー・ファイル (セキュリティー・プロファイルなど) が保管されている場所です。

Windows c:¥Documents and Settings¥*user_name*

UNIX /home/*user_name*

WebSphere eXtreme Scale の技術概要

WebSphere eXtreme Scale は、柔軟性があってスケーラブルな、メモリー内データ・グリッドです。これは、複数のサーバーにまたがって、アプリケーション・データおよびビジネス・ロジックを動的にキャッシュに入れ、区画化し、複製し、管理します。

WebSphere eXtreme Scale はメモリー内データベースではないため、固有の構成要件を考慮する必要があります。データ・グリッドをデプロイするための最初のステッ

プは、コア・グループおよびカタログ・サービスを開始することです。これらは、データ・グリッドに参加している他のすべての Java 仮想マシンの調整役を果たし、構成情報を管理します。WebSphere eXtreme Scale プロセスは、コマンド行からの単純なコマンド・スクリプト呼び出しで開始されます。

次のステップは、データ・グリッドがデータを保管および取得するための WebSphere eXtreme Scale サーバー・プロセスを開始することです。開始されたサーバーは、自動的にコア・グループおよびカタログ・サービスに登録され、連携してデータ・グリッド・サービスを提供できるようになります。サーバーが多いほど、データ・グリッドの容量も信頼性も高まります。

ローカル・データ・グリッドは、単一の、単一インスタンスのグリッドであり、1 つのグリッド内にすべてのデータがあります。WebSphere eXtreme Scale をメモリー内データベース処理スペースとして効果的に使用するように、分散データ・グリッドを構成し、デプロイすることができます。分散グリッドのデータは、これを含む各種 eXtreme Scale サーバーに分散されます。つまり、データは、各サーバーが区画と呼ばれるデータの一部のみを含むように分散されます。

分散データ・グリッド構成パラメーターのうち最も重要なパラメーターが、グリッド内の区画の数です。グリッド・データは、この数のサブセットに分割されます (それぞれのサブセットを区画と呼びます)。 カタログ・サービスは、データの区画を、区画のキーに基づいて配置します。区画数は、データ・グリッドの容量とスケーラビリティーに直接影響します。1 つのサーバーが 1 つ以上のデータ・グリッド区画を含むことができます。したがって、区画のサイズはサーバーのメモリー・スペースによって制限されます。逆に、区画の数を増やすと、データ・グリッドの容量は増加します。データ・グリッドの最大容量は、区画数に、各サーバーで使用可能なメモリー・サイズを掛けたものです。サーバーは JVM でもかまいませんが、デプロイメント環境に合わせて eXtreme Scale サーバーを定義できます。

1 つの区画のデータは、1 つの断片に保管されます。可用性のため、レプリカ (同期または非同期のどちらでも可) を持つようにデータ・グリッドを構成できます。グリッド・データに対する変更は、プライマリー断片に対して行われ、レプリカ断片に複製されます。したがって、データ・グリッドで使用される/必要とされるメモリー合計は、データ・グリッドのサイズに (1 (プライマリー) + レプリカの数) を掛けたものになります。

WebSphere eXtreme Scale は、データ・グリッドの断片を、そのデータ・グリッドを構成している多数のサーバー間で分配します。これらのサーバーは、同じ物理サーバー上にある場合も、異なる物理サーバー上にある場合もあります。可用性のため、レプリカ断片はプライマリー断片とは別の物理サーバーに置かれます。

WebSphere eXtreme Scale は、サーバーの状態をモニターし、断片または物理サーバーの障害および復旧時に断片を移動します。 例えば、レプリカ断片を含んでいるサーバーに障害が起こった場合、WebSphere eXtreme Scale は新しいレプリカ断片を割り振り、その新規レプリカにプライマリーからデータを複製します。 プライマリー断片を含んでいるサーバーに障害が起こった場合は、レプリカ断片がプロモートされてプライマリー断片になり、新しいレプリカ断片が作成されます。データ・グリッド用に追加サーバーを開始した場合、すべてのサーバー間で断片のバランスが取られます。この再度バランスを取ることを「スケールアウト」と呼びます。同様

に、サーバーの 1 つを停止して、データ・グリッドが使用するリソースを削減する場合があり、それを「スケールイン」と呼びます。この結果として、残りのサーバー間で断片のバランスが取られます。

キャッシュの概要

WebSphere eXtreme Scale は、データベース・バックエンドにインライン・キャッシングを提供するために使用するか、サイド・キャッシュとして使用できるメモリー内のデータベース処理スペースとして機能できます。インライン・キャッシングは、データと対話するための基本手段として eXtreme Scale を使用します。eXtreme Scale をサイド・キャッシュとして使用する場合は、データ・グリッドと連動してバックエンドが使用されます。このセクションでは、さまざまなキャッシュ概念やシナリオを説明し、データ・グリッドをデプロイするために使用可能なトポロジーについても解説します。

キャッシング・アーキテクチャー: マップ、コンテナー、クライアント、およびカタログ

WebSphere eXtreme Scale を使用して、アーキテクチャーはローカルのメモリー内でのデータ・キャッシング、または分散クライアント/サーバーでのデータ・キャッシングを使用できます。

WebSphere eXtreme Scale を作動させるには、最低限の追加インフラストラクチャーが必要です。インフラストラクチャーは、サーバー上で Java Platform, Enterprise Edition アプリケーションをインストール、開始、および停止するためのスクリプトで構成されます。キャッシュ・データは eXtreme Scale サーバー内に保管され、クライアントはリモート側でサーバーに接続します。

分散キャッシュは、より高いパフォーマンス、可用性、およびスケーラビリティーをもたらすもので、動的トポロジーを使用して構成できます。こうした構成では、サーバーのバランスが自動的に取られます。また、既存の eXtreme Scale サーバーを再始動せずに、別のサーバーを追加することもできます。単純なデプロイメントを作成することも、数千ものサーバーが必要になる大規模なテラバイト・サイズのデプロイメントを作成することもできます。

カタログ・サービス

カタログ・サービスは、定常状態ではアイドルになるロジックをホストし、スケーラビリティーにはほとんど影響しません。カタログ・サービスが構築されている目的は、同時に使用可能になる数百ものコンテナーにサービスを提供し、それらのコンテナーを管理するサービスを実行することです。

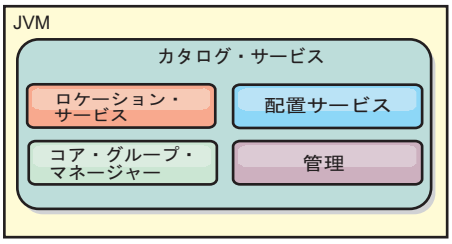


図 2. カタログ・サービス

カタログの役割分担には以下のサービスが含まれます。

ロケーション・サービス
ロケーション・サービスは、アプリケーションをホスティングするコンテナーを探しているクライアントと、ホスティングされるアプリケーションを配置サービスに登録しようとしているコンテナーの局所性を提供します。ロケーション・サービスは、この機能をスケールアウトするために、すべてのグリッド・メンバーで実行されます。

配置サービス
配置サービスは、グリッドの中枢神経的な存在であり、個々の断片をホスト・コンテナーに割り振る責任を担います。配置サービスは、クラスター内で N 個の中から 1 つ選ばれたサービスとして実行されます。 N 個の中の 1 つのポリシーが使用されるため、配置サービスの実行中のインスタンスは常に 1 つのみです。 そのインスタンスが停止する必要がある場合は、別のプロセスが引き継ぎます。カタログ・サービスのあらゆる状態は、予備のために、カタログ・サービスをホスティングするすべてのサーバーに複製されます。

コア・グループ・マネージャー
コア・グループ・マネージャーは、ヘルス・モニタリングのためにピアのグループ化を管理し、コンテナーを少数のサーバーからなるグループに編成し、サーバーのグループを自動的に統合します。初めてカタログ・サービスにコンタクトしたコンテナーは、いくつかの Java 仮想マシン (JVM) からなる新規または既存のグループのいずれかに割り当てられるのを待機します。Java 仮想マシンの各グループは、ハートビートを通してそれらの各メンバーの可用性をモニターします。再割り振りと経路転送によって障害に対処できるように、グループ・メンバーの 1 つが可用性情報をカタログ・サービスに中継します。

管理 WebSphere eXtreme Scale 環境の管理には、計画、デプロイ、管理、およびモニターの 4 つのステージがあります。

可用性のために、カタログ・サービス・ドメインを構成します。カタログ・サービス・ドメインは、複数の Java 仮想マシン (マスター JVM が 1 つと、多数のバックアップ Java 仮想マシン) から構成されます。

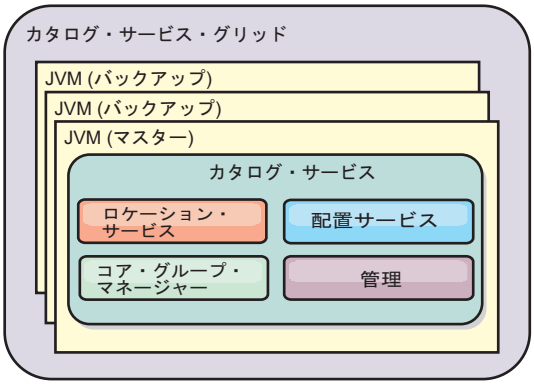


図 3. カタログ・サービス・ドメイン

コンテナー・サーバー、区画、および断片

コンテナー・サーバーは、データ・グリッドのアプリケーション・データを保管します。通常、このデータは区画と呼ばれるパーツに分割され、複数のコンテナー・サーバーでホストされます。これを受けて各コンテナー・サーバーは、完全なデータのサブセットをホストします。JVM は 1 つ以上のコンテナー・サーバーをホストすることができ、各コンテナー・サーバーは複数の断片をホストできます。

要確認: すべてのデータをホストするコンテナー・サーバーのヒープ・サイズを計画してください。それにあわせて、ヒープ設定を適宜構成してください。

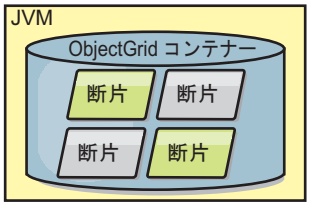


図 4. コンテナー・サーバー

区画は、グリッド内のデータのサブセットをホストします。WebSphere eXtreme Scale は、自動的に複数の区画を単一コンテナー・サーバーに配置し、追加のコンテナー・サーバーが使用可能になるとそれらに区画を分散させます。

重要: 区画の数は動的に変更できないため、最終的なデプロイメントの前に、区画の数を慎重に選択してください。ネットワーク内での区画の配置にはハッシュ・メカニズムが使用され、いったんデプロイされた後でデータ・セット全体を eXtreme Scale が再ハッシュすることはできません。 一般に、区画の数は多めに見積もって構いません。

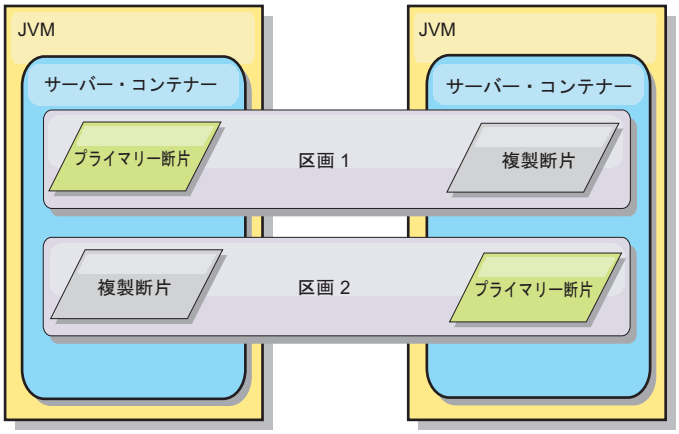


図 5. 区画

断片とは区画のインスタンスであり、プライマリーまたはレプリカの 2 つのロールのいずれか 1 つを持ちます。プライマリー断片とそのレプリカによって、区画の物理的な実体が構成されます。各区画はいくつかの断片を持ち、それぞれの断片が、その区画に含まれるデータ全体をホストします。 1 つの断片がプライマリー断片であり、他はレプリカ断片です。レプリカ断片は、プライマリー断片に含まれているデータの冗長コピーです。プライマリー断片は、トランザクションからキャッシュへの書き込みが可能な唯一の区画インスタンスです。レプリカ断片は、「ミラーリングされた」区画インスタンスです。レプリカ断片は、同期または非同期にプライマリー断片から更新内容を受信します。レプリカ断片の場合、トランザクションはキャッシュからの読み取りのみが可能です。レプリカは、プライマリーと同じコンテナー・サーバーではホストされません。また、通常はプライマリーと同じマシン上ではホストされません。

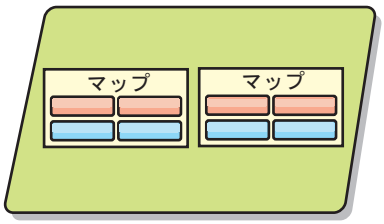


図 6. 断片

データの可用性を向上させる、または永続性の保証を高めるには、データを複製する必要があります。ただし、レプリカ生成はトランザクションのコストを増加させるため、可用性と引き換えにパフォーマンスが犠牲になります。eXtreme Scale では、同期レプリカ生成と非同期レプリカ生成のサポートに加え、同期と非同期の両方のレプリカ生成モードを使用するハイブリッド・レプリカ生成モデルがサポートされるため、このコストをコントロールできます。同期レプリカ断片は、データ整合性を保証するため、プライマリー断片のトランザクションの一部として更新内容を受信します。トランザクション完了の前に、プライマリーと同期レプリカの両方でトランザクションをコミットする必要があるため、同期レプリカでは応答時間が倍の長さになることがあります。非同期レプリカ断片は、パフォーマンスへの影響を制限するために、トランザクションがコミットされた後に更新内容を受信しま

す。しかし、非同期レプリカでは、プライマリーよりトランザクションがいくつか遅れることがあるため、非同期レプリカ断片でデータ損失の可能性が生じます。

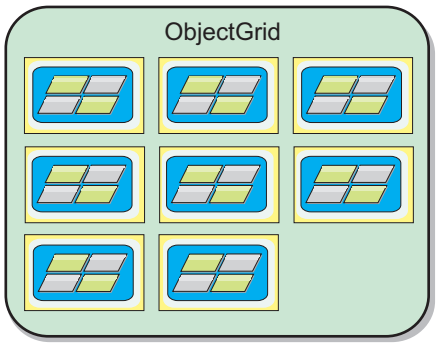


図 7. ObjectGrid

マップ

マップはキーと値のペアを格納するコンテナーであり、アプリケーションはマップを使用して、キーで索引付けされた値を保管できます。マップでは、キーまたは値の索引属性に追加できる索引がサポートされます。こうした索引が照会ランタイムによって自動的に使用され、照会を実行するのに最も効率的な方法が決定されます。

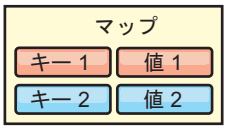


図 8. マップ

マップ・セットは、共通の区画化アルゴリズムを持つマップの集合です。マップ内のデータは、マップ・セットに定義されたポリシーに基づいて複製されます。マップ・セットは分散トポロジーでのみ使用され、ローカル・トポロジーでは必要ありません。

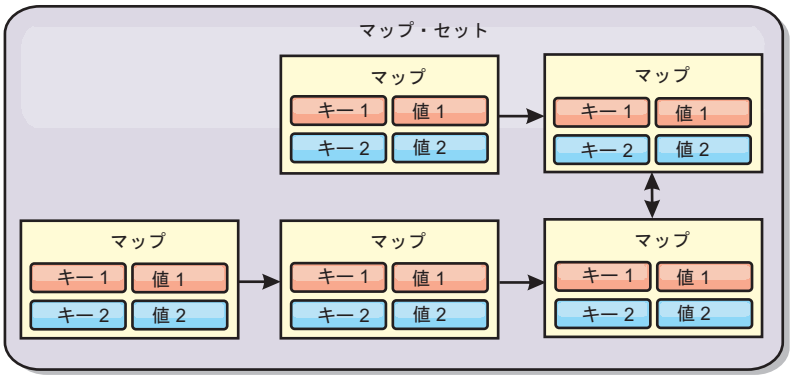


図 9. マップ・セット

マップ・セットには、関連のあるスキーマを含めることができます。スキーマとは、同種のオブジェクト・タイプまたはエンティティーを使用している場合に各マップ間の関係を記述したメタデータです。

WebSphere eXtreme Scale は、ObjectMap API を使用して、シリアライズ可能な Java オブジェクトを各マップに保管できます。スキーマはマップ全体に定義することができ、それぞれのマップが単一のタイプのオブジェクトを保持している場合に、それらのマップ内のオブジェクト間のリレーションシップを表します。マップ・オブジェクトの内容を照会するには、マップにスキーマを定義しておく必要があります。WebSphere eXtreme Scale では、複数のマップ・スキーマを定義できます。

WebSphere eXtreme Scale は、EntityManager API を使用して、エンティティーを保管することもできます。各エンティティーは、マップに関連付けられています。エンティティー・マップ・セットのスキーマは、エンティティー記述子 XML ファイルまたはアノテーション付き Java クラスのどちらかを使用して自動的に検出されます。各エンティティーは、キー属性のセット、および非キー属性のセットを持ちます。また、他のエンティティーへのリレーションシップも持つことができます。WebSphere eXtreme Scale では、1 対 1、1 対多、多対 1、および多対多のリレーションシップがサポートされています。各エンティティーは、マップ・セット内の単一のマップに物理的にマップされます。エンティティーにより、複数のマップにまたがる複雑なオブジェクト・グラフを簡単にアプリケーションに装備できます。分散トポロジーは、複数のエンティティー・スキーマを持つことができます。

詳しくは、オブジェクトおよびそのリレーションシップのキャッシング (EntityManager API)を参照してください。

クライアント

クライアントは、カタログ・サービスに接続し、サーバー・トポロジーの記述を取得し、必要に応じて各サーバーと直接通信します。新規サーバーの追加または既存サーバーの障害などのためにサーバー・トポロジーが変更されると、動的カタログ・サービスは、データをホスティングする適切なサーバーにクライアントを経路指定します。 クライアントは、アプリケーション・データのキーを調べて、要求をどの区画に送付するのかを決定しなければなりません。クライアントは、単一のトランザクションで複数の区画からデータを読み取ることができます。ただし、クライアントが更新できるのは、1 つのトランザクションで単一区画のみです。クライ

アントが複数のエントリーを更新した場合、クライアント・トランザクションはその区画を更新に使用する必要があります。

可能なデプロイメントの組み合わせが、次のリストに示されています。

- カタログ・サービスは、Java 仮想マシン内で自身のグリッド内に存在します。単一のカタログ・サービスを使用して、複数の eXtreme Scale クライアントまたはサーバーを管理することができます。
- コンテナーは、JVM 内で単独で開始することも、別の ObjectGrid インスタンスの他のコンテナーと一緒に任意の JVM にロードすることもできます。
- クライアントは任意の JVM 内に存在でき、1 つ以上の ObjectGrid インスタンスと通信できます。また、クライアントはコンテナーと同じ JVM 内に存在することも可能です。

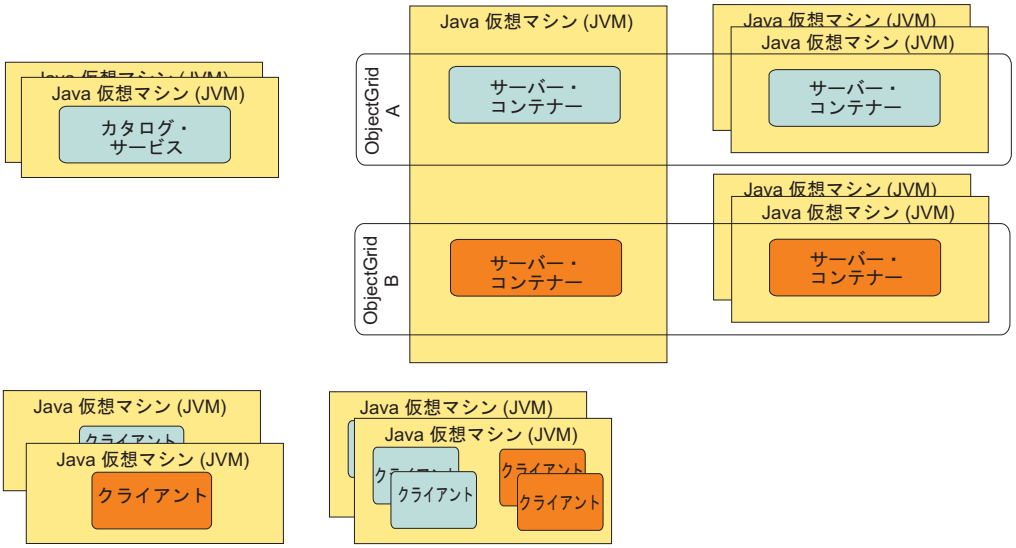


図 10. 可能なトポロジー

ゾーン

ゾーンを使用して、断片の配置を制御できます。ゾーンは、ユーザー定義の物理サーバーの論理グループです。各種ブレード・サーバー、ブレード・サーバーのシャーシ、ビルのフロア、マルチ・データ・センター環境での異なる地理的ロケーションなど、さまざまなタイプのゾーンの例を以下で示します。また、それぞれ固有の IP アドレスを持っている複数のサーバー・インスタンスが同じ物理サーバー上で実行されている仮想化環境での使用状況も取り上げます。

データ・センター間で定義済みのゾーン

ゾーンの典型的な例および使用状況は、2 カ所以上に地理的に分散したデータ・センターがある場合です。分散データ・センターにより、データ・センターでの障害からのリカバリーのためにデータ・グリッドが異なるロケーションに広がります。例えば、リモート・データ・センターにデータ・グリッドの非同期レプリカ断片の完全な集合を配置するようにすることをお勧めします。このストラテジーによっ

て、ローカル・データ・センターで障害が発生しても、透過的にデータを失うことなくリカバリーできます。データ・センター自体は、高速で待ち時間の短いネットワークを使用しています。ただし、あるデータ・センターと別のデータ・センターとの間の通信では、待ち時間が長くなります。同期レプリカは各データ・センター内で使用され、待ち時間が短いことにより、レプリカ生成が応答時間に与える影響が最小に抑えられます。非同期レプリカ生成を使用すると、応答時間への影響が緩和されます。ローカル・データ・センターで障害が発生した場合に、地理的距離があると、可用性が提供されます。

以下の例では、Chicago ゾーンにプライマリー断片があるものについては、そのレプリカが London ゾーンにあります。London ゾーンにプライマリー断片があるものについては、そのレプリカが Chicago ゾーンにあります。

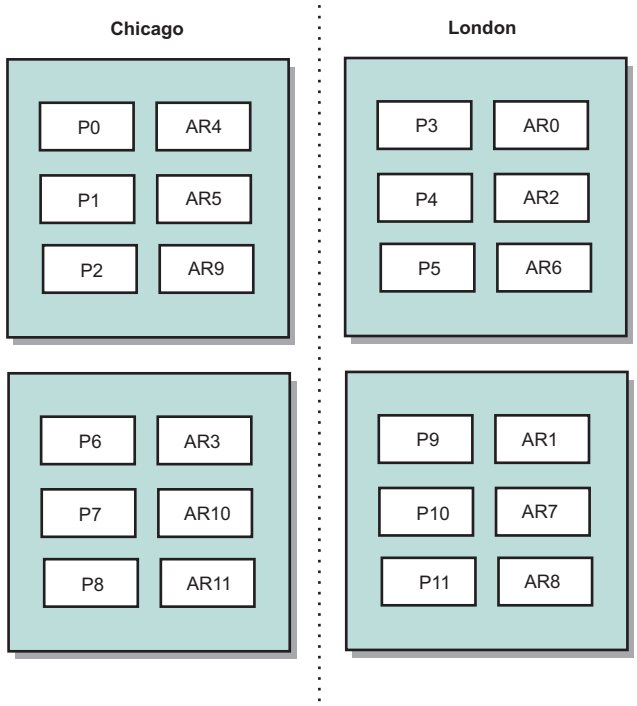


図 11. ゾーン内のプライマリーとレプリカ

eXtreme Scale での以下の 3 つの構成設定により、断片配置を制御します。

- デプロイメント・ファイルの設定
- グループ・コンテナー
- ルールの指定

以下のセクションでは、各種オプションについて説明します。大まかに複雑さの低いものから高いものまで提示します。

開発モードを無効にする

デプロイメント XML ファイルで、`developmentMode="false"` と設定します。

この単純なステップにより、1 番目の eXtreme Scale 断片配置ポリシーがアクティブになります。

この XML ファイルについて詳しくは、デプロイメント・ポリシー記述子 XML ファイルを参照してください。

ポリシー 1: 同じ区画の断片は、別個の物理サーバーに配置する。

1 つのレプリカ断片を使用したデータ・グリッドの単純な例を考えてみましょう。このポリシーでは、各区画のプライマリー断片とレプリカ断片は異なる物理サーバー上にあります。1 つの物理サーバーに障害が起こった場合でも、データは失われません。各区画のプライマリー断片またはレプリカ断片が障害の発生しなかった異なる物理サーバー上にあるか、その両方が障害の発生しなかった他のいくつかの物理サーバー上にあることになります。

このポリシーの高可用性および単純性のために、すべての実稼働環境で最も効率的な設定となります。多くの場合、環境内の断片配置を効果的に制御するために必要な唯一のステップは、このポリシーを適用することです。

このポリシーを適用した場合、1 台の物理サーバーは 1 つの IP アドレスによって定義されます。断片は、コンテナー・サーバー内に配置されます。コンテナー・サーバーは、例えば、**startOgServer** スクリプトに指定する **-listenerHost** パラメーターなど、1 つの IP アドレスを持っています。複数のコンテナー・サーバーが同じ IP アドレスを持っている場合があります。

1 台の物理サーバーが複数の IP アドレスを持っているため、環境をさらに制御するには、次のステップを検討してください。

コンテナー・サーバーをグループ化するためのゾーンの定義

コンテナー・サーバーは、startOgServer スクリプトに **-zone** パラメーターを指定して、ゾーンに割り当てます。WebSphere Application Server 環境では、ゾーンは特定の名前形式 `ReplicationZone<Zone>` でノード・グループを通じて定義されます。このようにして、ゾーンの名前とメンバーシップを選択します。詳しくは、コンテナー・サーバーのゾーンの定義を参照してください。

ポリシー 2: 同じ区画の断片は、別個のゾーンに配置する。

1 つのレプリカ断片を使用したデータ・グリッドの例を拡張して、今度はそれを 2 つのデータ・センターにわたってデプロイしたと考えてみましょう。各データ・センターは独立ゾーンとして定義します。1 番目のデータ・センター内のコンテナー・サーバーにゾーン名 DC1 を使用し、2 番目のデータ・センター内のコンテナー・サーバーに対してゾーン名 DC2 を使用します。このポリシーでは、各区画のプライマリー断片とレプリカ断片は、異なるデータ・センターにあります。1 つのデータ・センターに障害が起こった場合でも、データは失われません。それぞれの区画について、プライマリー断片かレプリカ断片のいずれかがもう一方のデータ・センターにあります。

このポリシーでは、ゾーンを定義することによって断片配置を制御できます。興味のある物理的境界、論理的境界、またはグループを選択します。次に、それぞれのグループごとに固有のゾーン名を選択し、各ゾーン内のコンテナー・サーバーを当該ゾーンの名前で始動します。このようにして、eXtreme Scale は、同じ区画の断片が別個のゾーンに配置されるように断片を配置します。

ゾーン・ルールの指定

断片配置に対する最高詳細レベルの制御は、ゾーン・ルールの使用によって達成されます。ゾーン・ルールは、eXtreme Scale デプロイメント・ポリシー記述子 XML の zoneMetadata エレメントで指定します。ゾーン・ルールは、断片が配置される一連のゾーンを定義します。shardMapping エレメントは、断片をゾーン・ルールに割り当てます。以下のように、shardMapping エレメントの断片属性は断片タイプを指定します。

- P はプライマリー断片を指定します。
- S は同期レプリカ断片を指定します。
- A は非同期レプリカ断片を指定します。

複数の同期または非同期レプリカが存在する場合は、適切な断片タイプの shardMapping エレメントを提供する必要があります。zoneRule エレメントの exclusivePlacement 属性は、ゾーン内での同じ区画の断片の配置を決定します。exclusivePlacement 属性値は、次のとおりです。

- `true` (同じ区画の別の断片と同じゾーン内に断片を配置することができません。)

  要確認: 「true」の場合、少なくともゾーンを使用する断片と同じ数のゾーンをルールで使用しなければなりません。そうすることで、確実に各断片をそれ自身のゾーンに置くことができます。
- `false` (同じ区画の断片は、同じゾーン内に配置することができます。)

デフォルト設定は true です。

詳しくは、例: デプロイメント・ポリシー記述子 XML ファイルを使用したゾーン定義を参照してください。

拡張使用状況

断片配置ストラテジーのさまざまな使用状況を以下に示します。

ローリング・アップグレード

デプロイメントの再開を要する保守など、ローリング・アップグレードを物理サーバーに適用するシナリオを考えてみましょう。この例では、データ・グリッドが 20 台の物理サーバーにわたって広がっており、1 つの同期レプリカを使用して定義されていると想定します。保守のために、10 台の物理サーバーを一度にシャットダウンするとします。

10 台の物理サーバーのグループをシャットダウンとき、シャットダウンするサーバー上にプライマリー断片とレプリカ断片の両方がある区画はありません。そうでないと、その区画のデータが失われます。

最も簡単な解決策は、3 番目のゾーンを定義することです。2 つのゾーンにそれぞれ 10 台ずつ物理サーバーを入れる代わりに、3 つのゾーンを使用し、2 つのゾーンには 7 台の物理サーバーを、1 つのゾーンには 6 台の物理サーバーを入れます。データがより多くのゾーンに広がると、可用性のためのフェイルオーバーが向上します。

別の手法として、ゾーンをもう 1 つ定義するのではなく、レプリカを追加するというのがあります。

eXtreme Scale のアップグレード

eXtreme Scale ソフトウェアをライブ・データを含むデータ・グリッドでローリング・アップグレードする場合は、次の問題を考慮してください。カタログ・サービス・ソフトウェアのバージョンは、コンテナー・サーバー・ソフトウェアのバージョンより大きいか等しくなければなりません。まずローリング・ストラテジーを使用してすべてのカタログ・サーバーをアップグレードします。デプロイメントのアップグレードについて詳しくは、トピックeXtreme Scale サーバーの更新を参照してください。

データ・モデルの変更

関連問題として、ダウン時間を生じさせることなく、どのようにデータ・グリッド内に保管されたオブジェクトのデータ・モデルまたはスキーマを変更するかということがあります。データ・モデルを変更するのに、データ・グリッドを停止し、コンテナー・サーバー・クラスパスの更新されたデータ・モデル・クラスを使用して再開し、データ・グリッドを再ロードすると、破壊的な影響を与える可能性があります。代わりに、新規データ・グリッドを新しいスキーマを使用して開始し、古いデータ・グリッドのデータを新規データ・グリッドにコピーし、最後に古いデータ・グリッドをシャットダウンします。

これらの各プロセスは、破壊的な影響を与え、ダウン時間が生じます。ダウン時間を生じさせることなくデータ・モデルを変更するには、以下のいずれかのフォーマットでオブジェクトを保管します。

- XML を値として使用する
- Google protobuf を使用して作成されたブロブを使用する
- JavaScript Object Notation (JSON) を使用する

クライアント・サイドで Plain Old Java Object (POJO) からこれらのフォーマットのいずれかに簡単に進むには、シリアライザーを使用します。スキーマの変更がより簡単になります。

仮想化

クラウド・コンピューティングおよび仮想化は、人気のある新たなテクノロジーです。デフォルトでは、eXtreme Scale は、「ポリシー 1」に述べられているように、確実に同じ区画の 2 つの断片が同じ IP アドレス上に置かれないようにします。VMware などの仮想イメージ上にデプロイする場合は、それぞれ固有の IP アドレスを持つ複数のサーバー・インスタンスを同じ物理サーバー上で実行できます。レプリカを別個の物理サーバー上にのみ配置可能であるようにするには、ゾーンを使用すると、この問題を解決できます。物理サーバーをゾーンにグループ化し、プライマリー断片とレプリカ断片を別個のゾーンに保持するゾーン配置ルールを使用します。

広域ネットワークでのゾーン

低速のネットワーク接続を使用する複数のビルまたはデータ・センターにまたがって、1 つの eXtreme Scale データ・グリッドをデプロイしたい場合があります。ネ

ットワーク接続が低速になれば、それだけ処理能力が低下し、接続待ち時間が長くなります。このモードでは、ネットワーク輻輳やその他の要因のために、ネットワーク分割の可能性も増します。

これらのリスクに対処するために、eXtreme Scale カタログ・サービスは、コンテナー・サーバーの障害を検出するためにハートビートをやりとりするコア・グループにコンテナー・サーバーを編成します。これらのコア・グループは複数のゾーンに広がりません。各コア・グループ内のリーダーがメンバーシップ情報をカタログ・サービスにプッシュします。カタログ・サービスは、報告された障害があれば、問題のコンテナー・サーバーにハートビートを行うことによってそれを検証してから、メンバーシップ情報に応答します。カタログ・サービスは、誤障害検出を認めた場合、何のアクションも実行しません。コア・グループ区画が短時間で回復するためです。またカタログ・サービスは、定期的にコア・グループ・リーダーに低速でハートビートを行い、コア・グループ分離の症状を処理します。

Evictor

Evictor は、データ・グリッドからデータを削除します。時間ベースの Evictor を設定できます。あるいは、Evictor は BackingMap に関連付けられているため、BackingMap インターフェースを使用してプラグ可能 Evictor を指定することもできます。

デフォルトの Evictor

すべてのバックアップ・マップでデフォルトの TTL Evictor が作成されます。 デフォルトの Evictor は、存続時間 の概念に基づいてエントリーを除去します。この振る舞いは、次のタイプを持つ ttlType 属性で定義されます。

なし

エントリーの期限切れがないように、それによってマップからエントリーが除去されることがないように指定します。

作成時間

作成された時に応じてエントリーが除去されるように指定します。

CREATION_TIME ttlType を使用している場合、Evictor は、作成からの時間がその TimeToLive 属性値と等しいときにエントリーを除去します。TimeToLive 属性値は、アプリケーション構成でミリ秒単位で設定されます。 TimeToLive 属性値を 10 秒に設定すると、エントリーは挿入の 10 秒後に自動的に除去されます。

この値を CREATION_TIME ttlType に設定する場合は注意が必要です。この Evictor は、一定時間にのみ使用される、キャッシュへの妥当な追加量がある場合に、最も有効に使用されます。このストラテジーによって、作成されたものはすべて、一定時間後に除去されます。

CREATION_TIME ttlType は、20 分以下の間隔で株価情報を最新表示するようなシナリオで役立ちます。例えば、ある Web アプリケーションは株価情報の取得をしますが、最新情報を取得することは重要でないとします。この場合、株価情報は 20 分間 ObjectGrid にキャッシュされます。 20 分後、ObjectGrid マップの有効期限が切れ、除去されます。ほぼ 20 分ごと

に、ObjectGrid マップは Loader プラグインを使用してマップ・データをデータベースの新しいデータで更新します。データベースは 20 分ごとに最新の株価情報によって更新されます。

最終アクセス時間

(読み取りか更新かに関わらず) 最後にアクセスされた時に応じてエントリーが除去されるように指定します。

最終更新時間

最後に更新された時に応じてエントリーが除去されるように指定します。

LAST_ACCESS_TIME または LAST_UPDATE_TIME ttlType 属性を使用している場合は、CREATION_TIME ttlType を使用している場合よりも TimeToLive をより低い数に設定します。エントリーの TimeToLive 属性は、アクセスされるたびにリセットされるからです。言い換えれば、TimeToLive 属性が 15 で、エントリーが 14 秒間存在し、それからアクセスされた場合、このエントリーはあと 15 秒間有効期限が切れることはありません。TimeToLive を比較的高い数値に設定した場合は、多くのエントリーがまったく除去されなくなる可能性があります。ただし、この値を 15 秒程度に設定すると、エントリーは頻繁にアクセスされない場合に除去されることになります。

The LAST_ACCESS_TIME または LAST_UPDATE_TIME ttlType は、ObjectGrid マップを使用してクライアントからのセッション・データを保持するようなシナリオで役立ちます。セッション・データは、クライアントがそのセッション・データを一定時間使用しない場合は破棄する必要があります。例えば、セッション・データは、クライアントによるアクティビティーが 30 分間なかった後にタイムアウトになるとします。この場合、LAST_ACCESS_TIME または LAST_UPDATE_TIME の TTL タイプを使用し、TimeToLive 属性を 30 分に設定するのが、このアプリケーションにおいて適切です。

独自の Evictor を作成することもできます。詳しくは、カスタム Evictor の作成を参照してください。

プラグ可能 Evictor

デフォルトの TTL Evictor は、時刻ベースの除去ポリシーを使用し、 BackingMap 内のエントリーの数は、エントリーの有効期限の時間には影響を及ぼしません。オプションのプラグ可能 Evictor を使用して、時刻ではなく、存在するエントリー数に基づいてエントリーを除去することができます。

以下のオプションのプラグ可能 Evictor は、 BackingMap が一定のサイズの限界を超えたときに除去するエントリーを決定するために、一般に使用されるアルゴリズムをいくつか提供します。

- LRUEvictor Evictor は、BackingMap が最大エントリー数を超えたときに除去するエントリーを決定する際、最長未使用時間 (LRU) アルゴリズムを使用します。
- LFUEvictor Evictor は、BackingMap が最大エントリー数を超えたときに除去するエントリーを決定する際、最少使用頻度 (LFU) アルゴリズムを使用します。

BackingMap は、トランザクション内でエントリーが作成、変更、または除去されると Evictor に通知します。 BackingMap は、これらのエントリーをトラッキングし、BackingMap インスタンスから 1 つ以上のエントリーをいつ除去するかを選択します。

BackingMap インスタンスには、最大サイズについての構成情報はありません。代わりに、Evictor の振る舞いを制御する Evictor プロパティーが設定されます。LRUEvictor と LFUEvictor の両方の最大サイズ・プロパティーを使用して、最大サイズを超えた後、Evictor がエントリーを除去開始するようにします。 TTL Evictor と同様に、LRU Evictor と LFU Evictor では、最大エントリー数に達した場合、パフォーマンスへの影響を最小化するためにエントリーを直ちに除去することはありません。

特定のアプリケーションに LRU または LFU 除去アルゴリズムが適していない場合、独自の Evictor を作成して、除去ストラテジーを作成できます。

メモリー・ベースの除去

重要: メモリー・ベースの除去は、Java Platform, Enterprise Edition バージョン 5 以降でのみサポートされます。

組み込み Evictor はすべて、メモリー・ベースの除去をサポートし、これは、BackingMap の evictionTriggers 属性を「MEMORY_USAGE_THRESHOLD」に設定することにより使用可能にできます。 BackingMap での evictionTriggers 属性の設定方法について詳しくは、BackingMap インターフェースおよびObjectGrid 記述子 XML ファイルを参照してください。

メモリー・ベースの除去は、ヒープ使用量のしきい値に基づいています。BackingMap でメモリー・ベースの除去が使用可能になっていて、BackingMap に組み込み Evictor がある場合、使用量のしきい値は、まだ設定されていなければ、合計メモリーのデフォルトのパーセンテージに設定されます。

メモリー・ベースの除去を使用している場合、ガーベッジ・コレクションしきい値を、ターゲット・ヒープ使用率と同じ値に構成する必要があります。例えば、メモリー・ベースの除去のしきい値が 50 パーセントに設定されていて、ガーベッジ・コレクションのしきい値がデフォルトの 70 パーセント・レベルであると、ヒープ使用率は 70 パーセントまで上がる可能性があります。このヒープ使用率の増加が起きるのは、メモリー・ベースの除去が 1 ガーベッジ・コレクション・サイクルの後にのみトリガーされるためです。

デフォルトの使用量しきい値のパーセンテージを変更するには、eXtreme Scale サーバー・プロセスのコンテナーおよびサーバーのプロパティー・ファイルで memoryThresholdPercentage プロパティーを設定します。クライアント・プロセスでターゲットの使用量しきい値を設定する場合は、MemoryPoolMXBean を使用できます。

WebSphere eXtreme Scale が使用するメモリー・ベースの除去アルゴリズムは、使用中のガーベッジ・コレクションのアルゴリズムの動作に影響を受けやすいのです。メモリー・ベースの除去の最善のアルゴリズムは、IBM デフォルト・スループット・コレクターです。世代ガーベッジ・コレクション・アルゴリズムは、好ましく

ない動作を引き起こす可能性があるため、メモリー・ベースの除去と一緒に、このアルゴリズムを使用すべきではありません。

使用量しきい値のパーセンテージを変更するには、eXtreme Scale サーバー・プロセスのコンテナーおよびサーバーのプロパティー・ファイルで memoryThresholdPercentage プロパティーを設定します。

実行時に、メモリー使用量がターゲットの使用量しきい値を超えると、メモリー・ベースの Evictor はエントリーの除去を開始して、メモリー使用量がターゲットの使用量しきい値を下回るようにします。ただし、継続してシステム・ランタイムによるメモリー消費が迅速に進むと、除去速度が十分速くても、メモリー不足エラーが起こる可能性がなくなるという保証はありません。

OSGi フレームワークの概要

OSGi は、Java に対して動的モジュール・システムを定義します。OSGi サービス・プラットフォームは、階層化アーキテクチャーを持ち、さまざまな標準 Java プロファイルで実行されるように設計されています。OSGi コンテナー内の WebSphere eXtreme Scale サーバーおよびクライアントを始動できます。

OSGi コンテナー内でアプリケーションを実行する利点

WebSphere eXtreme Scale OSGi サポートにより、Eclipse Equinox OSGi フレームワークに製品をデプロイできます。これまで、eXtreme Scale で使用するプラグインを更新する場合は、Java 仮想マシン (JVM) を再始動して、プラグインの新規バージョンを適用する必要がありました。現在は、OSGi フレームワークが提供する動的更新機能を使用して、JVM を再始動せずにプラグイン・クラスを更新できます。これらのプラグインは、ユーザー・バンドルによってサービスとしてエクスポートされます。WebSphere eXtreme Scale は、OSGi レジストリーでルックアップして、サービス (複数可) にアクセスします。

eXtreme Scale コンテナーは、OSGi Configuration Admin サービスまたは OSGi Blueprint を使用して容易かつ動的に始動するように構成できます。新規データ・グリッドをその配置ストラテジーを使用してデプロイする場合は、OSGi 構成を作成するか、eXtreme Scale 記述子 XML ファイルを使用してバンドルをデプロイすることによって、これを行うことができます。OSGi サポートを使用すると、eXtreme Scale 構成データを含むアプリケーション・バンドルを、システム全体を再始動することなく、インストール、開始、停止、更新、およびアンインストールできます。この機能を使用して、データ・グリッドを中断することなくアプリケーションをアップグレードできます。

プラグイン Bean およびプラグイン・サービスをカスタム断片有効範囲で構成することができます。これにより、データ・グリッド内で実行中の他のサービスに対して高度な統合オプションを使用することができます。各プラグインは OSGi Blueprint ランキングを使用して、プラグインの各インスタンスが正しいバージョンでアクティブ化されていることを確認できます。OSGi Managed Bean (MBean) と **xscmd** ユーティリティーが提供され、これにより、eXtreme Scale プラグイン OSGi サービスとそのランキングを照会することができます。

この機能によって、管理者は、構成および管理に関する潜在的なエラーを迅速に認識することができ、eXtreme Scale によって使用中のプラグイン・サービス・ランキ

ングをアップグレードすることができます。

OSGi バンドル

OSGi フレームワーク内のプラグインと対話し、それをデプロイするには、バンドルを使用する必要があります。 OSGi サービス・プラットフォームにおけるバンドルとは、Java アーカイブ (JAR) ファイルです。このファイルには Java コード、リソース、およびマニフェスト (バンドルとその依存関係についての記述) が含まれます。バンドルはアプリケーションのデプロイメントの単位です。 eXtreme Scale 製品は、以下のバンドル・タイプをサポートします。

サーバー・バンドル

サーバー・バンドルは `objectgrid.jar` ファイルであり、eXtreme Scale スタンドアロン・サーバーのインストールでインストールされます。これは、eXtreme Scale サーバーの稼働に必要で、eXtreme Scale クライアントまたはローカルのメモリー内のキャッシュの実行にも使用できます。`objectgrid.jar` ファイルのバンドル ID は com.ibm.websphere.xs.server_`<version>` で、バージョンのフォーマットは `<Version>.<Release>.<Modification>` です。例えば、eXtreme Scale バージョン 7.1.1 のサーバー・バンドルは、com.ibm.websphere.xs.server_7.1.1 です。

クライアント・バンドル

クライアント・バンドルは `ogclient.jar` ファイルであり、eXtreme Scale スタンドアロンおよびクライアントのインストールでインストールされます。これは、eXtreme Scale クライアントまたはローカルのメモリー内のキャッシュの実行に使用されます。`ogclient.jar` ファイルのバンドル ID は、com.ibm.websphere.xs.client_`version` です。ここで、version は、`<Version>.<Release>.<Modification>` の形式です。例えば、eXtreme Scale バージョン 7.1.1 のクライアント・バンドルは、com.ibm.websphere.xs.client_7.1.1 です。

制限

オブジェクト・リクエスト・ブローカー (ORB) を再始動できないため、eXtreme Scale バンドルを再始動できません。eXtreme Scale サーバーを再始動するには、OSGi フレームワークを再開する必要があります。

キャッシュ統合の概要

多用途性や信頼性などを持ちながら実行する能力を WebSphere eXtreme Scale に付与する重大なエレメントは、キャッシング概念のアプリケーションです。それは、ほとんどすべてのデプロイメント環境で、データのパーシスタンスや再収集を最適化します。

JPA レベル 2 (L2) キャッシュ・プラグイン

WebSphere eXtreme Scale には、OpenJPA と Hibernate Java Persistence API (JPA) プロバイダーの両方に対するレベル 2 (L2) のキャッシュ・プラグインが組み込まれ

ています。これらのプラグインのいずれかを使用すると、アプリケーションは JPA API を使用します。データ・グリッドがアプリケーションとデータベースとの間に導入されて、応答時間が短縮されます。

eXtreme Scale を L2 キャッシュ・プロバイダーとして使用することにより、データ読み取りおよび照会時のパフォーマンスが向上し、データベースに対する負荷が軽減します。WebSphere eXtreme Scale ではキャッシュがすべてのプロセスで自動的に複製されるので、組み込みキャッシュ実装をしのぐ利点があります。あるクライアントが値をキャッシュすると、他のすべてのクライアントが、そのキャッシュされた値をローカルのメモリー内で使用できるようになります。

L2 キャッシュ・プロバイダーのトポロジーおよびプロパティーは、`persistence.xml` ファイルで構成できます。これらのプロパティーの構成について詳しくは、JPA キャッシュ構成プロパティーを参照してください。

ヒント: JPA L2 キャッシュ・プラグインは、JPA API を使用するアプリケーションを必要とします。WebSphere eXtreme Scale API を使用して JPA データ・ソースにアクセスする場合は、JPA ローダーを使用します。詳しくは、70 ページの『JPA ローダー』を参照してください。

JPA L2 キャッシュ・トポロジーに関する考慮事項

次の要因が、構成するトポロジーのタイプに影響を与えます。

1. **キャッシュに入れられる予想データ量**
   - データが単一 JVM ヒープに収まる場合は、30 ページの『組み込みトポロジー』または 29 ページの『イントラドメイン・トポロジー』を使用します。
   - データが単一 JVM ヒープに収まらない場合は、31 ページの『組み込みの区画化トポロジー』、または 33 ページの『リモート・トポロジー』を使用します。
2. **予想される読み取り/書き込み率**

   読み取り/書き込み率は、L2 キャッシュのパフォーマンスに影響します。トポロジーごとに、読み取り操作および書き込み操作の処理は異なります。
   - 30 ページの『組み込みトポロジー』: ローカル読み取り、リモート書き込み
   - 29 ページの『イントラドメイン・トポロジー』: ローカル読み取り、ローカル書き込み
   - 31 ページの『組み込みの区画化トポロジー』: 区画化: リモート読み取り、リモート書き込み
   - 33 ページの『リモート・トポロジー』: リモート読み取り、リモート書き込み

   ほとんどが読み取りのみのアプリケーションの場合、可能であれば、組み込みトポロジーおよびイントラドメイン・トポロジーを使用します。書き込みの方が多いアプリケーションの場合、イントラドメイン・トポロジーを使用します。
3. **データのキーによる検索に対する照会のパーセンテージ**

   JPA 照会キャッシュが使用可能に設定されている場合、照会操作は JPA 照会キャッシュを使用します。読み取り/書き込み率が高いアプリケーションに対してのみ JPA 照会キャッシュを使用可能に設定します。例えば、読み取り操作が 99%

に迫る場合です。JPA 照会キャッシュを使用する場合、 30 ページの『組み込みトポロジー』または『イントラドメイン・トポロジー』を使用する必要があります。

キーによる検索操作では、ターゲット・エンティティーにリレーションシップがなければ、ターゲット・エンティティーが取り出されます。ターゲット・エンティティーに EAGER フェッチ・タイプとのリレーションシップがあれば、これらのリレーションシップがターゲット・エンティティーと一緒に取り出されます。JPA データ・キャッシュの中でこれらのリレーションシップを取り出すと、少数のキャッシュのヒットですべてのリレーションシップ・データを取得することになります。

4. **容認されるデータの失効性レベル**

   JVM がほとんどないシステムでは、書き込み操作でデータのレプリカ生成の待ち時間が発生します。キャッシュの目的は、同期化された最終データ・ビューをすべての JVM にわたって保守することです。イントラドメイン・トポロジーを使用している場合、書き込み操作ではデータのレプリカ生成で遅延が発生します。このトポロジーを使用しているアプリケーションの場合、データを上書きする可能性のある失効読み取りと同時書き込みを容認できる必要があります。

7.1.1+
イントラドメイン・トポロジー

イントラドメイン・トポロジーを使用すると、プライマリー断片がトポロジー内のすべてのコンテナー・サーバーに置かれます。これらのプライマリー断片には、区画のデータの全セットが含まれています。これらのプライマリー断片はすべて、キャッシュ書き込み操作も実行できます。この構成を使用すると、すべてのキャッシュ書き込み操作が単一のプライマリー断片を経由しなければならない組み込みトポロジーのボトルネックが解決します。

イントラドメイン・トポロジーでは、構成ファイルでレプリカを定義していたとしても、レプリカ断片は作成されません。それぞれの冗長プライマリー断片にはデータの全コピーが含まれているため、それぞれのプライマリー断片をレプリカ断片とみなすこともできます。この構成は、単一区画を使用し、組み込みトポロジーと似ています。

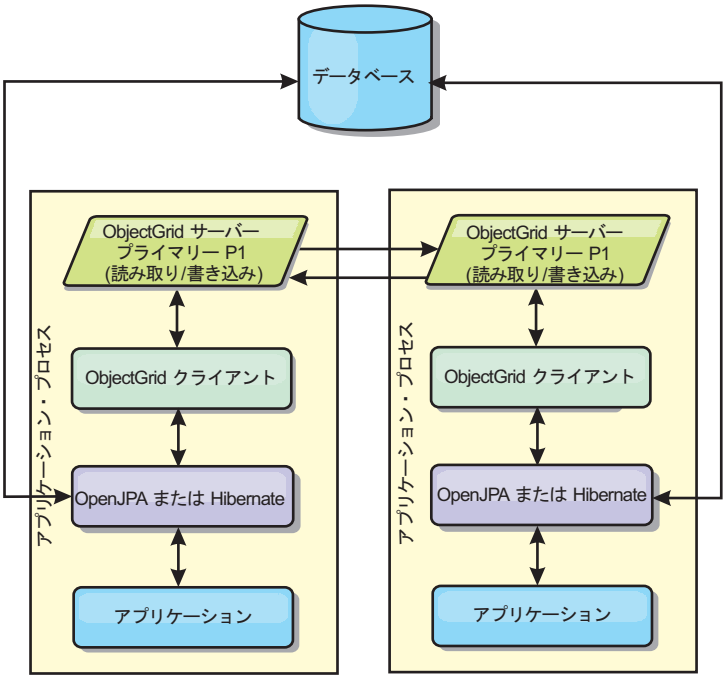


図 12. JPA イントラドメイン・トポロジー

イントラドメイン・トポロジーでの関連 JPA キャッシュ構成プロパティー:

```
ObjectGridName=objectgrid_name,ObjectGridType=EMBEDDED,PlacementScope=CONTAINER_SCOPE,PlacementScopeTopology=HUB | RING
```

利点:

- キャッシュ読み取りおよび更新がローカルで行われる。
- 構成が簡単である。

制約:

- このトポロジーは、コンテナー・サーバーに区画データの全セットを含めることができる場合に最適です。
- すべてのコンテナー・サーバーはプライマリー断片をホストするため、レプリカ断片は、構成済みであったとしても配置されることはありません。ただし、すべてのプライマリー断片は、他のプライマリー断片を使用して複製されることになるため、これらのプライマリー断片は互いのレプリカとなります。

組み込みトポロジー

ヒント: 最高のパフォーマンスのためにイントラドメイン・トポロジーを使用することを検討してください。

組み込みトポロジーでは、各アプリケーションのプロセス・スペース内に コンテナー・サーバーを作成します。OpenJPA および Hibernate が、キャッシュのメモリー内コピーで直接読み取りを行い、他のすべてのコピーに書き込みを行います。非同期レプリカ生成を使用することによって、書き込みのパフォーマンスを向上させることができます。このデフォルト・トポロジーは、キャッシュ・データの量が少なく、1 つのプロセスに十分収まる場合に最も良く機能します。組み込みトポロジーを使用して、データの単一区画を作成します。

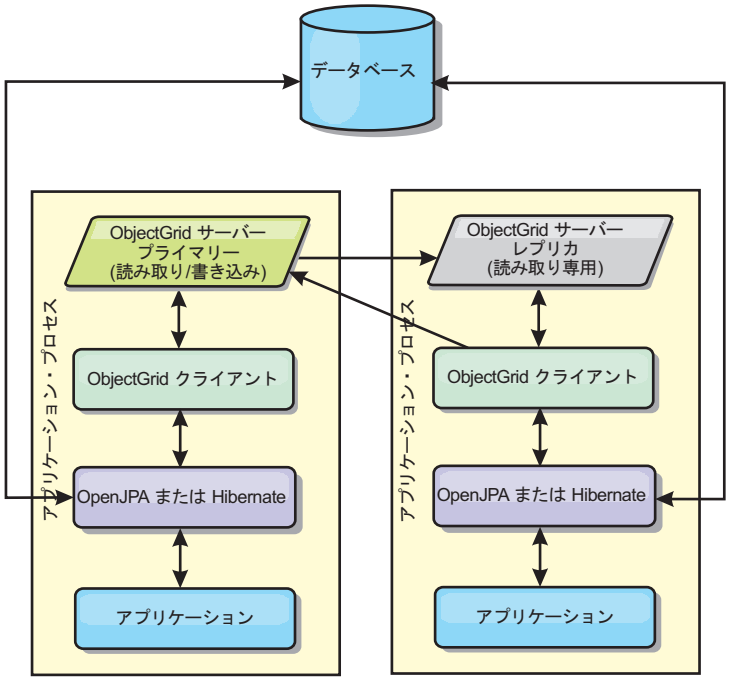


図 13. JPA 組み込みトポロジー

組み込みトポロジーでの関連 JPA キャッシュ構成プロパティー:

```
ObjectGridName=objectgrid_name,ObjectGridType=EMBEDDED,MaxNumberOfReplicas=num_replicas,ReplicaMode=SYNC | ASYNC | NONE
```

利点:

- すべてのキャッシュ読み取りが高速のローカル・アクセスである。
- 構成が簡単である。

制約:

- データ量が、プロセスのサイズに限られる。
- すべてのキャッシュ更新は、1 つのプライマリー断片を通じて送信される。これにより、ボトルネックが発生する。

組み込みの区画化トポロジー

ヒント: 最高のパフォーマンスのためにイントラドメイン・トポロジーを使用することを検討してください。

注意:
組み込み区画化トポロジーで JPA 照会キャッシュを使用しないでください。照会キャッシュは、エンティティー・キーのコレクションである照会結果を保管します。照会キャッシュは、データ・キャッシュからエンティティー・データを取り出します。データ・キャッシュは複数のプロセス間で分割されるため、これらの追加呼び出しによって L2 キャッシュの利点が失われる可能性があります。

キャッシュ・データの量が多すぎて 1 つのプロセスに収まらないときは、組み込み区画化トポロジーを使用できます。このトポロジーでは、データを複数のプロセス間で分割します。データはプライマリー断片間で分割されるため、それぞれのプライマリー断片にデータのサブセットが含まれます。データベース待ち時間が大きい

場合でも、このオプションを使用できます。

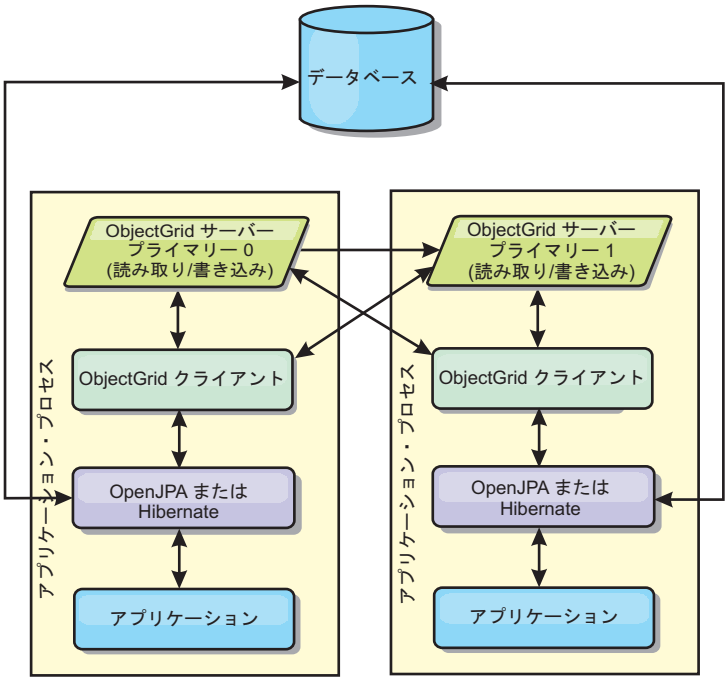


図 14. JPA 組み込み区画化トポロジー

組み込みの区画化トポロジーでの関連 JPA キャッシュ構成プロパティー:

```
ObjectGridName=objectgrid_name,ObjectGridType=EMBEDDED_PARTITION,ReplicaMode=SYNC | ASYNC | NONE,
NumberOfPartitions=num_partitions,ReplicaReadEnabled=TRUE | FALSE
```

利点:

- 大容量のデータを保管できる。
- 構成が簡単である。
- キャッシュ更新が複数のプロセスに分散される。

制約:

- ほとんどのキャッシュ読み取りおよび更新がリモートで行われる。

例えば、JVM あたり最大 1 GB で 10 GB のデータをキャッシュに入れる場合、Java 仮想マシンが 10 必要になります。したがって、区画数は 10 以上に設定されます。理想的には、区画数を素数に設定しなければなりません。そうすると、各断片が適当な量のメモリーを保管するようになります。通常、numberOfPartitions は、Java 仮想マシン の数に等しくなるようにします。このように設定すれば、各 JVM に 1 つの区画が格納されます。レプリカ生成を使用可能にする場合、システム内の Java 仮想マシンの数を増やす必要があります。 そうでないと、各 JVM ごとに 1 つのレプリカ区画も格納することになり、この区画はプライマリー区画と同量のメモリーを消費します。

選択された構成のパフォーマンスを最大化するには、「*管理ガイド*」の『メモリー・サイズ設定および区画数の計算』を参照してください。

例えば、4 つの Java 仮想マシン があるシステムで numberOfPartitions 設定値が 4 の場合、各 JVM は 1 つのプライマリー区画をホストします。読み取り操作では、25 パーセントの確率でローカルで使用可能な区画からデータを取り出すことができ、これは、リモート JVM からデータを取得する場合と比較してはるかに速くなります。照会の実行などの読み取り操作で 4 つの区画が均等に関わるデータ・コレクションを取り出す必要がある場合、呼び出しの 75 パーセントがリモートで、呼び出しの 25 パーセントがローカルです。ReplicaMode が SYNC または ASYNC に設定され、ReplicaReadEnabled が true に設定されている場合、4 つのレプリカ区画が作成され、これが 4 つの Java 仮想マシン に分散されます。各 JVM は、1 つのプライマリー区画と 1 つのレプリカ区画をホストします。読み取り操作をローカルで実行する確率は、50 パーセントに増えます。4 つの区画が均等に関わるデータ・コレクションを取り出す読み取り操作では、50 パーセントのリモート呼び出しと 50 パーセントのローカル呼び出しがあります。ローカル呼び出しは、リモート呼び出しよりはるかに高速です。リモート呼び出しが行われると、パフォーマンスは落ちます。

リモート・トポロジー

注意:
リモート・トポロジーで JPA 照会キャッシュを使用しないでください。照会キャッシュは、エンティティー・キーのコレクションである照会結果を保管します。照会キャッシュは、データ・キャッシュからエンティティー・データを取り出します。データ・キャッシュはリモートであるため、これらの追加呼び出しによって L2 キャッシュの利点が失われる可能性があります。

ヒント: 最高のパフォーマンスのためにイントラドメイン・トポロジーを使用することを検討してください。

リモート・トポロジーでは、すべてのキャッシュ・データを 1 つ以上の個別のプロセスに保管し、アプリケーション・プロセスのメモリー使用を減らします。区画化され、複製された eXtreme Scale データ・グリッドをデプロイすることによって、個別のプロセスへデータ配布するという利点が生かせます。 前のセクションで説明した組み込み構成および組み込み区画化構成とは対照的に、リモート・データ・グリッドを管理する場合は、アプリケーションおよび JPA プロバイダーから独立して管理する必要があります。

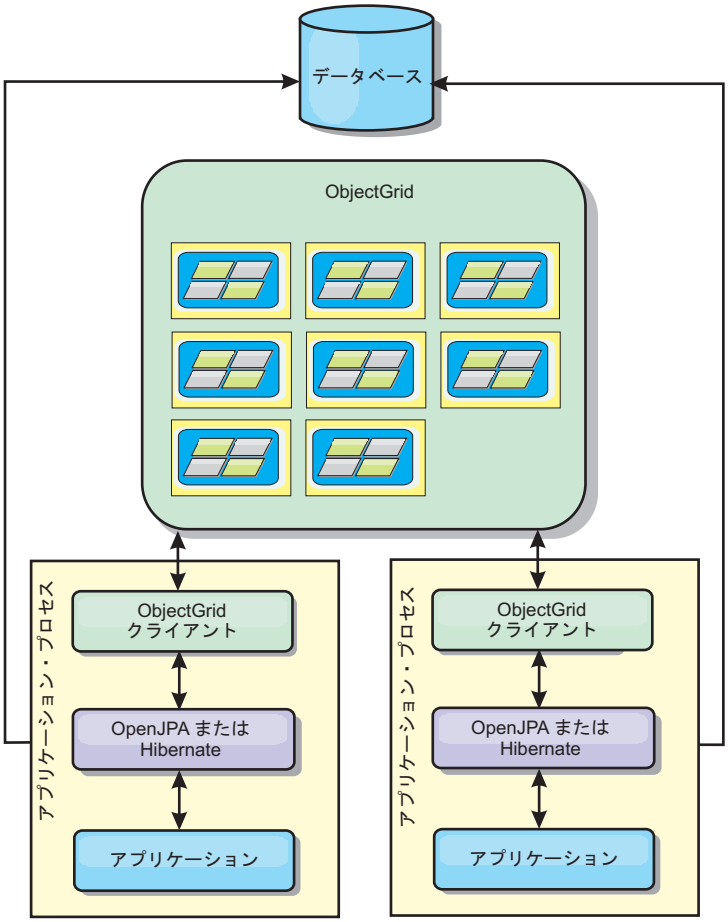


図 15. JPA リモート・トポロジー

リモート・トポロジーでの関連 JPA キャッシュ構成プロパティー:

`ObjectGridName=objectgrid_name,ObjectGridType=REMOTE`

REMOTE ObjectGrid タイプでは、ObjectGrid およびデプロイメント・ポリシーが JPA アプリケーションとは別に定義されるため、プロパティー設定は必要ありません。JPA キャッシュ・プラグインは、リモートで、既存のリモート ObjectGrid に接続します。

ObjectGrid とのすべての対話がリモートで行われるため、このトポロジーは、すべての ObjectGrid タイプの中で、パフォーマンスが最も遅くなります。

利点:

- 大容量のデータを保管できる。
- アプリケーション・プロセスがキャッシュ・データから解放される。
- キャッシュ更新が複数のプロセスに分散される。
- 柔軟な構成オプションがある。

制約:

- すべてのキャッシュ読み取りおよび更新がリモートで行われる。

HTTP セッション管理

WebSphere eXtreme Scale に付属のセッション・レプリカ生成マネージャーは、アプリケーション・サーバーのデフォルト・セッション・マネージャーと連動することができます。ユーザー・セッション・データの高可用性をサポートするために、セッション・データは、あるプロセスから別のプロセスへ複製されます。

フィーチャー

セッション・マネージャーは、Java Platform, Enterprise Edition バージョン 5 以降の任意のコンテナーで実行できるように設計されています。セッション・マネージャーは、WebSphere API にはまったく依存していないため、ベンダーのアプリケーション・サーバー環境をサポートするのと同様に、さまざまなバージョンの WebSphere Application Server をサポートできます。

HTTP セッション・マネージャーは、関連するアプリケーションのセッション・レプリカ生成機能を提供します。 セッション・レプリカ生成マネージャーは、Web コンテナーのセッション・マネージャーと連動します。セッション・マネージャーと Web コンテナーは協力して HTTP セッションを作成し、アプリケーションに関連付けられた HTTP セッションのライフサイクルを管理します。 このライフサイクル管理には、タイムアウト、明示的サーブレット、または JavaServer Pages (JSP) 呼び出しを基にしたセッションの無効化、およびそのセッションまたは Web アプリケーションと関連付けられているセッション・リスナーの起動などが含まれます。セッション・マネージャーは、そのセッションを、完全に複製され、クラスター化され、区画化されたデータ・グリッド内にパーシストします。WebSphere eXtreme Scale セッション・マネージャーを使用すると、アプリケーション・サーバーがシャットダウンされるか予期せずに終了した場合に、セッション・マネージャーが HTTP セッション・フェイルオーバー・サポートを提供することができます。セッション・マネージャーは、要求をアプリケーション・サーバー層に分散するロード・バランサー層によってアフィニティーが強制されない、アフィニティーをサポートしていない環境でも機能します。

使用に関するシナリオ

セッション・マネージャーは、以下のシナリオで使用できます。

- マイグレーション・シナリオの場合など、さまざまなバージョンの WebSphere Application Server でアプリケーション・サーバーを使用する環境。
- さまざまなベンダーのアプリケーション・サーバーを使用するデプロイメント。例えば、オープン・ソース・アプリケーション・サーバーで開発され、WebSphere Application Server でホストされているアプリケーションが挙げられます。別の例としては、ステージングから実動にプロモートされるアプリケーションがあります。すべての HTTP セッションがライブで、サービスされている間は、これらのアプリケーション・サーバー・バージョンのシームレスなマイグレーションが可能です。
- より高いレベルのサービスの品質 (QoS) でセッションを保持するようにユーザーに要求する環境。サーバーのフェイルオーバー中、セッションの可用性は、デフォルトの WebSphere Application Server QoS レベル以上に保証されます。
- セッション・アフィニティーが保証されない環境、または、アフィニティーがベンダーのロード・バランサーによって保守される環境。ベンダーのロード・バラ

ンサーを使用する場合は、そのロード・バランサー向けにアフィニティー・メカニズムをカスタマイズする必要があります。

- セッション管理および外部 Java プロセスへの保管に必要な処理による負荷を軽減する環境。
- セル間のセッション・フェイルオーバーを使用可能にする複数のセル。
- 複数のデータ・センターまたは複数のゾーン。

セッション・マネージャーの動作

セッション・レプリカ生成マネージャーは、セッション・リスナーを使用して、セッション・データの変更を listen します。セッション・レプリカ生成マネージャーは、セッション・データを、ローカルまたはリモートのいずれかで ObjectGrid インスタンス内に永続化します。 WebSphere eXtreme Scale に付属のツールを使用して、セッション・リスナーおよびサーブレット・フィルターをアプリケーション内の各 Web モジュールに追加できます。また、これらのリスナーおよびフィルターを、アプリケーションの Web デプロイメント記述子に手動で追加することも可能です。

このセッション・レプリカ生成マネージャーは、各ベンダーの Web コンテナーのセッション・マネージャーと連動し、Java 仮想マシン間でセッション・データを複製します。元のサーバーが停止したとき、ユーザーはセッション・データを他のサーバーから取得することができます。

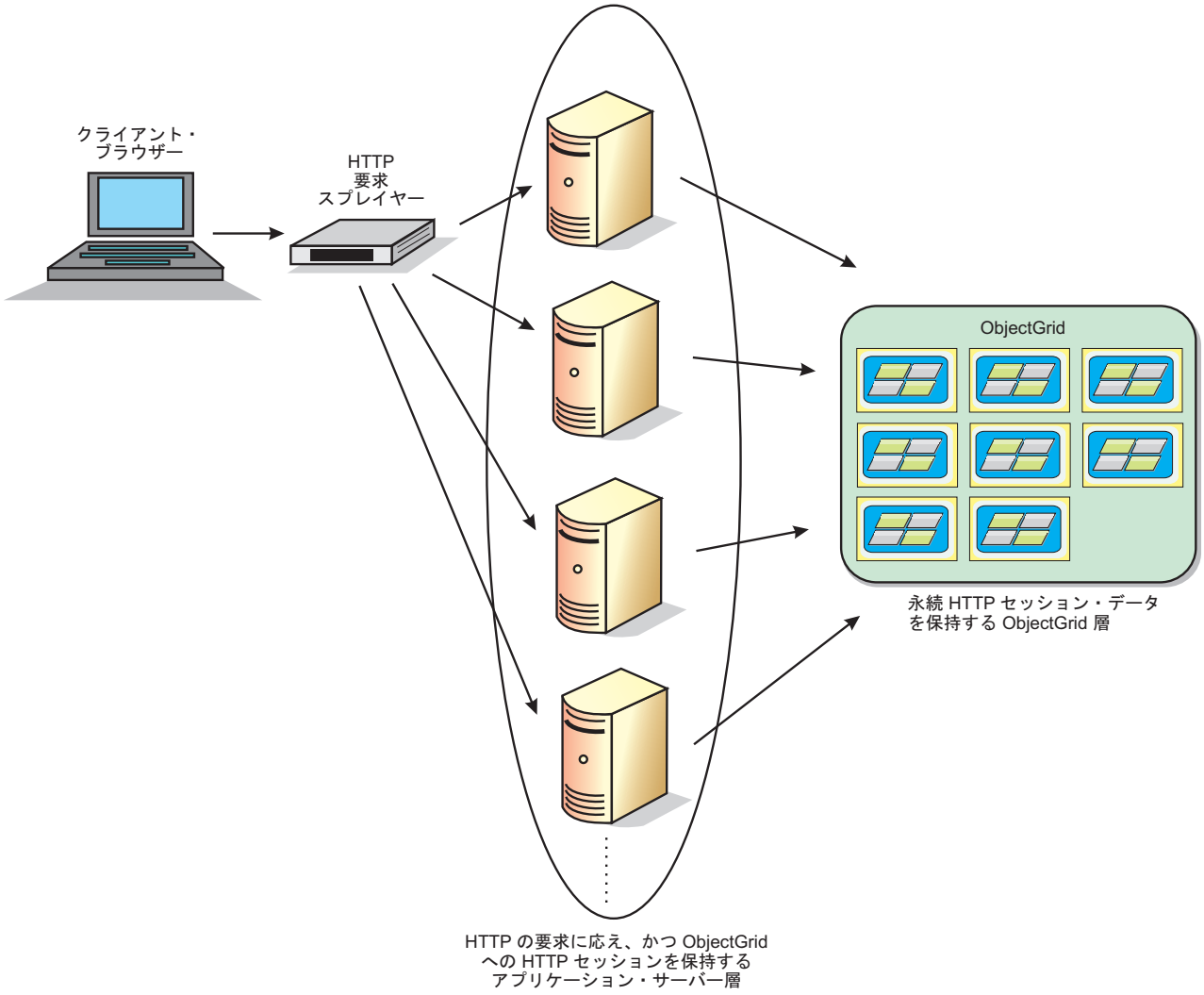


図 16. リモート・コンテナー構成を含む HTTP セッション管理トポロジー

デプロイメント・トポロジー

セッション・マネージャーは、以下の 2 つの異なる動的デプロイメント・シナリオを使用して構成することができます。

組み込みの、ネットワーク接続 eXtreme Scale コンテナー・サーバー

このシナリオでは、eXtreme Scale サーバーは、サーブレットと同じプロセス内に連結されます。セッション・マネージャーはローカル ObjectGrid インスタンスに直接通信できるため、コストのかかるネットワーク遅延を回避することができます。このシナリオは、アフィニティーを持って実行され、パフォーマンスが重要な場合に適しています。

リモートの、ネットワーク接続 eXtreme Scale コンテナー・サーバー

このシナリオでは、eXtreme Scale サーバーは、サーブレットが実行されるプロセスの外部プロセスで実行されます。セッション・マネージャーはリモートの eXtreme Scale サーバー・グリッドと通信します。このシナリオは、Web コンテナー層にセッション・データを保管するメモリーがない場合に適しています。 セッション・データは別の層へ負荷を軽減され、その結果

Web コンテナー層のメモリー使用量が下がります。データがリモート・ロケーションにあるため、待ち時間は長くなります。

汎用組み込みコンテナーの開始

objectGridType プロパティーが EMBEDDED に設定されている場合、eXtreme Scale は、Web コンテナーがセッション・リスナーまたはサーブレット・フィルターを初期化する際に、任意のアプリケーション・サーバー・プロセス内で自動的に組み込み ObjectGrid コンテナーを開始します。詳しくは、サーブレット・コンテキスト初期化パラメーターを参照してください。

ObjectGrid.xml ファイルおよび objectGridDeployment.xml ファイルを Web アプリケーションの WAR ファイルまたは EAR ファイルへパッケージする必要はありません。デフォルトの ObjectGrid.xml ファイルおよび objectGridDeployment.xml ファイルは製品の JAR ファイルへパッケージされています。動的マップは、さまざまな Web アプリケーション・コンテキストに対してデフォルトで作成されます。静的な eXtreme Scale マップは引き続きサポートされます。

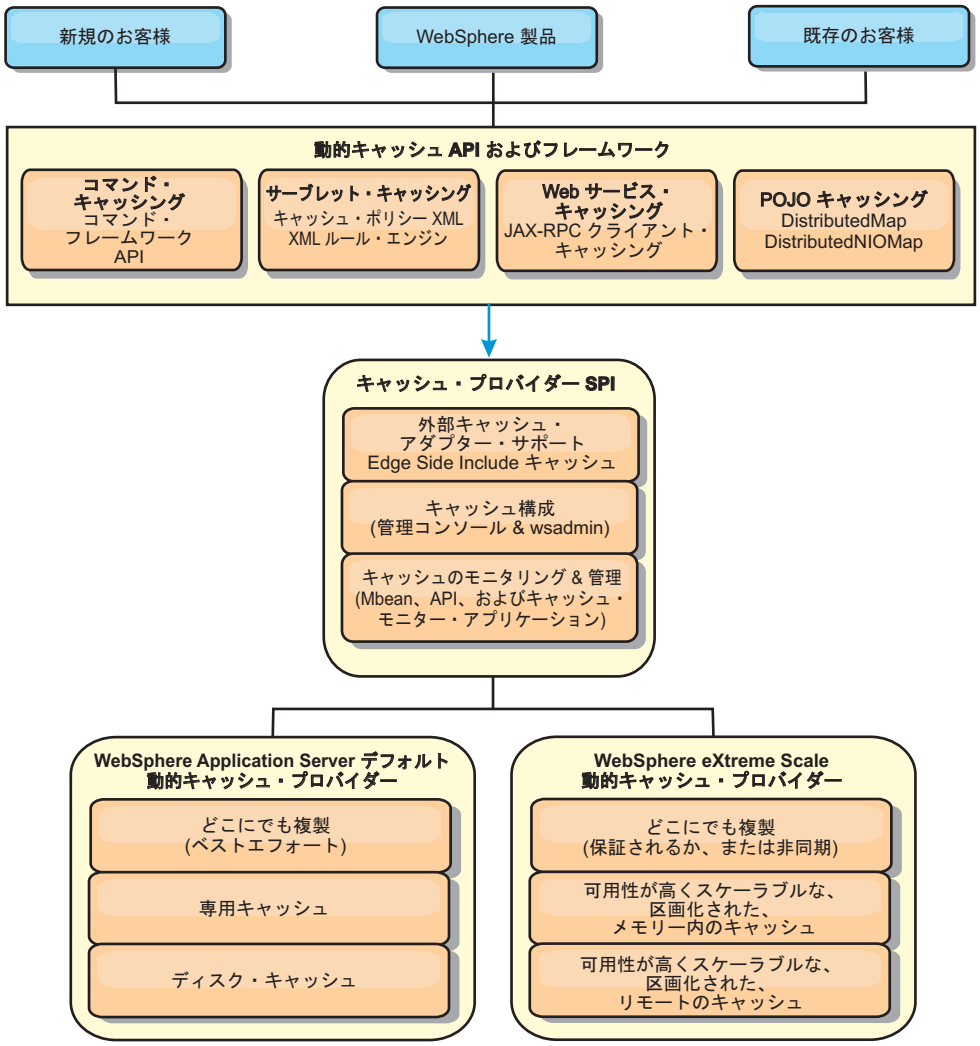
組み込み ObjectGrid コンテナーを開始するためのこのアプローチは、どのタイプのアプリケーション・サーバーにも適用されます。WebSphere Application Server コンポーネントまたは WebSphere Application Server Community Edition GBean を組み込むアプローチは推奨されません。

動的キャッシュ・プロバイダー

WebSphere Application Server にデプロイされた Java EE アプリケーションでは、動的キャッシュ API を使用できます。動的キャッシュ・プロバイダーは、ビジネス・データや生成された HTML をキャッシュに入れるために、または、データ・レプリカ生成サービス (DRS) を使用してセル内のキャッシュ・データを同期化するために利用できます。

概説

以前は、動的キャッシュ API の唯一のサービス・プロバイダーは、 WebSphere Application Server に組み込まれた、デフォルトの動的キャッシュ・エンジンでした。ユーザーは、 WebSphere Application Server 内の動的キャッシュ・サービス・プロバイダー・インターフェースを使用して、eXtreme Scale を動的キャッシュに接続できます。この機能をセットアップすると、動的キャッシュ API を使用して書かれたアプリケーションまたはコンテナー・レベル・キャッシュを使用するアプリケーション (サーブレットなど) が WebSphere eXtreme Scaleの機能およびパフォーマンス能力を利用できるようになります。



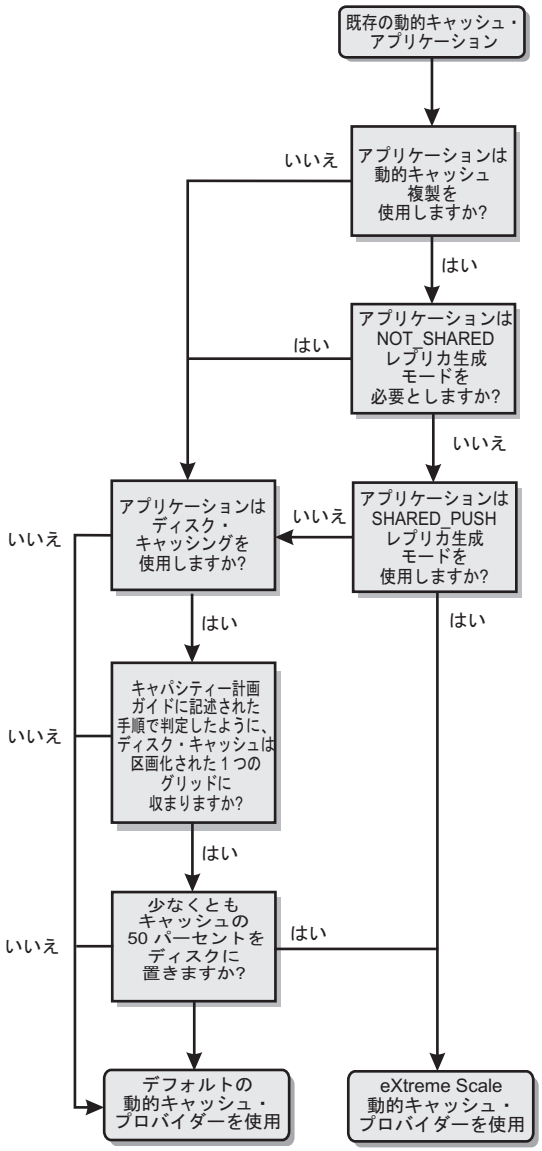
動的キャッシュ・プロバイダーのインストールと構成の手順については、WebSphere eXtreme Scale の動的キャッシュ・プロバイダーの構成を参照してください。

WebSphere eXtreme Scale の利用方法の決定

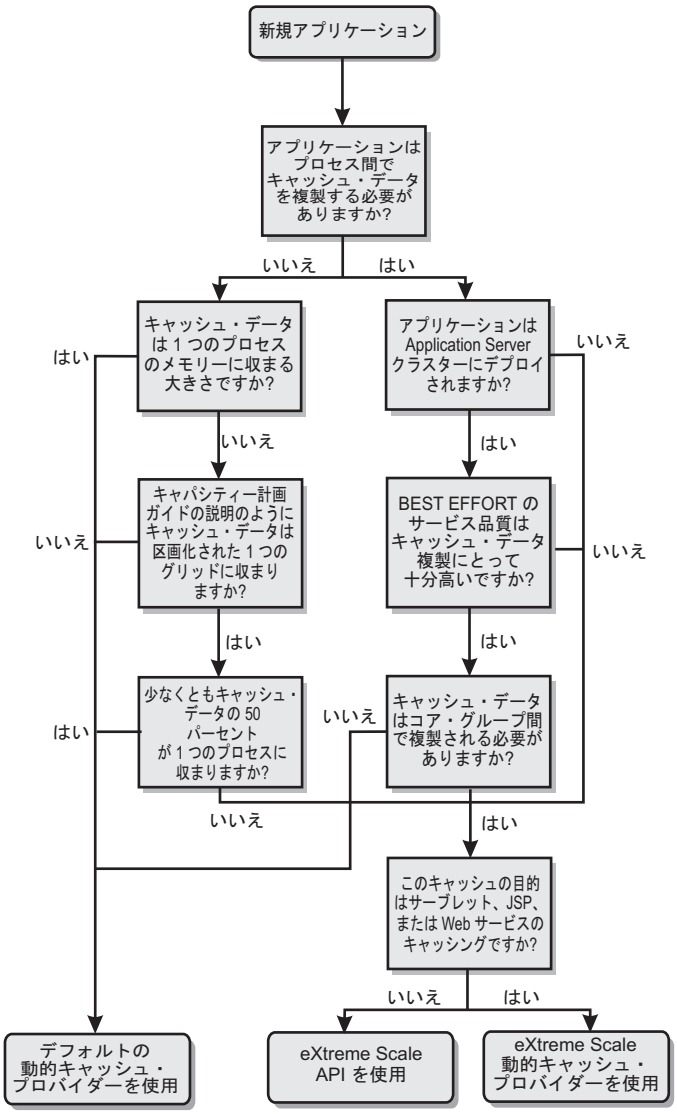
WebSphere eXtreme Scale で使用可能なフィーチャーによって動的キャッシュ API の分散機能は大幅に強化され、デフォルト動的キャッシュ・エンジンおよびデータ・レプリカ生成サービスで提供される機能を超えるものになっています。eXtreme Scale を使用することにより、複数のサーバー間で単に複製し、同期化するだけでなく、サーバー間で本当に分散したキャッシュを作成できます。さらに、eXtreme Scale キャッシュは、トランザクション・ベースであり、可用性がとても高く、動的キャッシュ・サービスに関して各サーバーが同じ内容を参照することを保証します。WebSphere eXtreme Scale は、キャッシュ・レプリカ生成に関して、DRS よりも高いサービス品質を提供します。

ただし、これらの利点は、どのようなアプリケーションでも eXtreme Scale 動的キャッシュ・プロバイダーが正しい選択であることを意味するわけではありません。以下のデシジョン・ツリーおよび機能比較マトリックスを使用して、ご使用のアプリケーションに最適のテクノロジーを決定してください。

既存の動的キャッシュ・アプリケーションをマイグレーションする際のデシジョン・ツリー



新規アプリケーションのキャッシュ・プロバイダーを選択する際のデシジョン・ツリー



機能比較

表 1. 機能比較

キャッシュ機能	デフォルト・プロバイダー	eXtreme Scale プロバイダー	eXtreme ScaleAPI
ローカルのメモリー内のキャッシング	x	x	x
分散キャッシング	組み込み	組み込み、組み込み区画化、およびリモート区画化	Multiple
直線的にスケーラブル		x	x
信頼できるレプリカ生成 (同期)		ORB	ORB

表 1. 機能比較 (続き)

キャッシュ機能	デフォルト・プロバイダー	eXtreme Scale プロバイダー	eXtreme ScaleAPI
ディスク・オーバーフロー	x		
除去	LRU/TTL/ヒープ・ベース	LRU/TTL (区画ごと)	Multiple
無効化	x	x	x
リレーションシップ	依存関係 ID、テンプレート	依存関係 ID、テンプレート	x
非キー検索			照会および索引
バックエンド統合			ローダー
トランザクションの		暗黙	x
キー・ベースの保管	x	x	x
イベントおよびリスナー	x	x	x
WebSphere Application Server 統合	単一セルのみ	複数セル	セルに無関係
Java Standard Edition サポート		x	x
モニタリングおよび統計	x	x	x
セキュリティー	x	x	x

表 2. シームレスなテクノロジー統合

キャッシュ機能	デフォルト・プロバイダー	eXtreme Scale プロバイダー	eXtreme ScaleAPI
WebSphere Application Server サーブレット/JSP 結果キャッシング	V5.1+	V6.1.0.25+	
WebSphere Application Server Web Services (JAX-RPC) 結果キャッシング	V5.1+	V6.1.0.25+	
HTTP セッション・キャッシング			x
OpenJPA および Hibernate 用のキャッシュ・プロバイダー			x
OpenJPA および Hibernate を使用したデータベース同期			x

表 3. プログラミング・インターフェース

キャッシュ機能	デフォルト・プロバイダー	eXtreme Scale プロバイダー	eXtreme ScaleAPI
コマンド・ベース API	コマンド・フレームワーク API	コマンド・フレームワーク API	DataGrid API
マップ・ベース API	DistributedMap API	DistributedMap API	ObjectMap API
EntityManager API			x

eXtreme Scale 分散キャッシュがどのように機能するかについて詳しくは、 「*管理ガイド*を参照してください。

注: eXtreme Scale 分散キャッシュは、キーと値が両方とも java.io.Serializable インターフェースを実装するエントリーのみを保管できます。

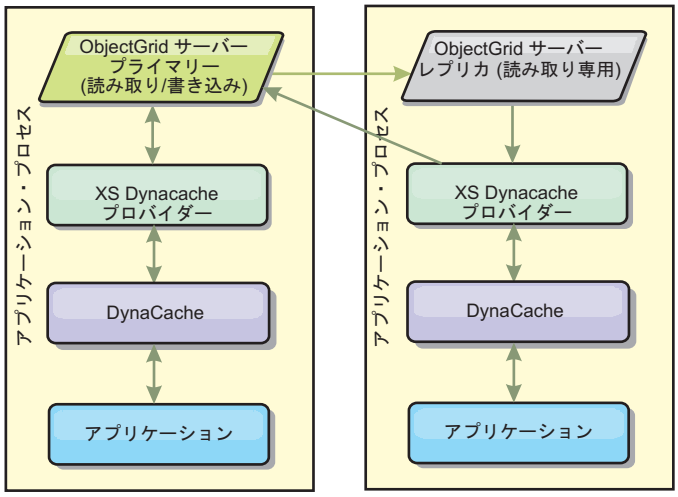
トポロジーのタイプ

eXtreme Scale プロバイダーを使用して作成された動的キャッシュ・サービスは、パフォーマンス、リソース、および管理上の必要に合わせて、3 つのトポロジーのいずれかにデプロイできます。これらのトポロジーは、組み込み、組み込み区画化、およびリモートです。

組み込みトポロジー

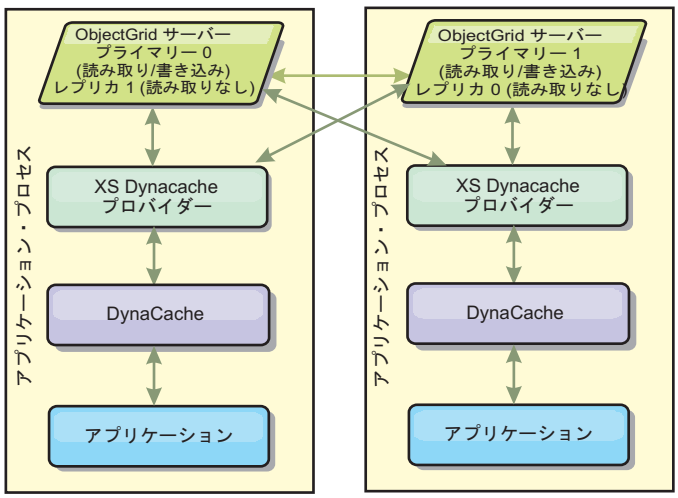
組み込みトポロジーは、デフォルトの動的キャッシュおよび DRS プロバイダーに似ています。組み込みトポロジーで作成された分散キャッシュ・インスタンスは、動的キャッシュ・サービスにアクセスする各 eXtreme Scale プロセスの内部でキャッシュの完全コピーを保持するので、すべての読み取り操作をローカルで実行できます。すべての書き込み操作は、シングル・サーバー・プロセスを完了し、そのプロセスでトランザクションのロックが管理された後で残りのサーバーに複製されます。したがって、このトポロジーは、キャッシュ読み取り操作がキャッシュ書き込み操作よりもずっと多いワークロードに向いています。

組み込みトポロジーでは、新規および更新されたキャッシュ・エントリーが、すべてのサーバー・プロセスで即時に可視になるわけではありません。キャッシュ・エントリーは、WebSphere eXtreme Scale の非同期レプリカ生成サービスによって伝搬されるまでは、そのエントリーを生成したサーバーに対してさえ可視になりません。これらのサービスは、ハードウェアで可能な限り速く作動しますが、それでも少しは遅延が発生します。次の図に、組み込みトポロジーを示します。



組み込み区画化トポロジー

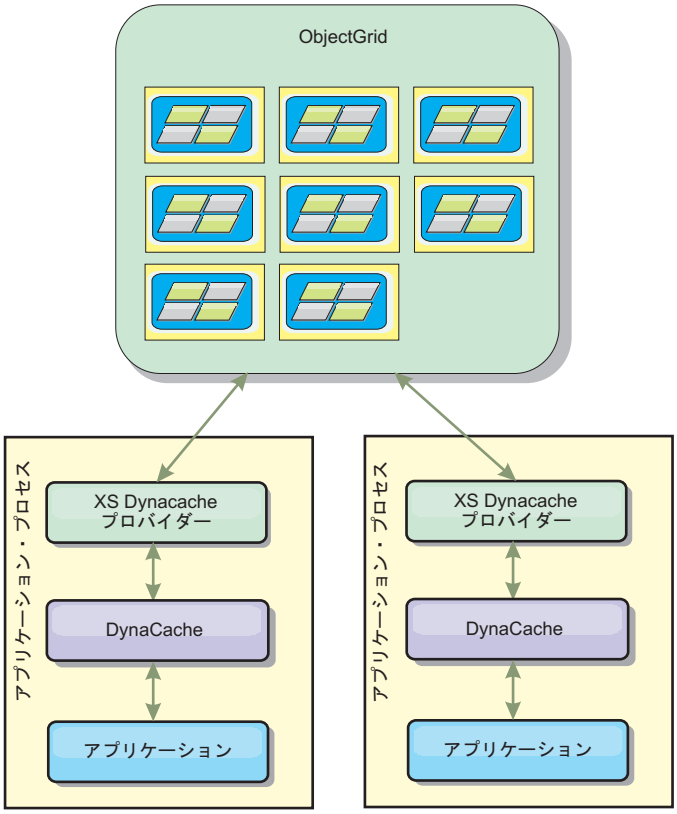
キャッシュ書き込みが読み取りと同じくらいか、より頻繁に発生するようなワークロードの場合、組み込み区画化トポロジーまたはリモート・トポロジーが推奨されます。組み込み区画化トポロジーは、キャッシュにアクセスする WebSphere Application Server プロセスの内部でキャッシュ・データのすべてを保持します。しかし、各プロセスが保管するのは、キャッシュ・データの一部分のみです。この「区画」に置かれたデータに対するすべての読み取りおよび書き込みは、そのプロセスを実行します。したがって、キャッシュに対するほとんどの要求はリモート・プロシージャー・コールで達成されることになります。その結果、読み取り操作の待ち時間は組み込みトポロジーよりも大きくなりますが、読み取り操作および書き込み操作を処理するための分散キャッシュの容量は、キャッシュにアクセスする WebSphere Application Server プロセスの数に応じて直線的に変化します。また、このトポロジーでは、キャッシュの最大サイズが 1 つの WebSphere プロセスのサイズに制約されることはありません。各プロセスはキャッシュの一部分だけを保持するので、キャッシュの最大サイズは、すべてのプロセスの総計サイズからプロセスのオーバーヘッドを引いたものになります。次の図に、組み込み区画化トポロジーを示します。



例えば、あるグリッドのサーバー・プロセスのそれぞれに、動的キャッシュ・サービスをホストするための 256 メガバイトの空きヒープがあるとします。組み込みトポロジーを使用する場合、デフォルトの動的キャッシュ・プロバイダーと eXtreme Scale プロバイダーの両方とも、メモリー内キャッシュのサイズは、256 メガバイトからオーバーヘッドを引いた値に制限されます。この資料のもう少し後にある『キャパシティー・プランニングおよび高可用性』セクションを参照してください。組み込み区画化トポロジーを使用する eXtreme Scale プロバイダーの場合、キャッシュ・サイズは 1 ギガバイトからオーバーヘッドを引いた値に制限されます。このようにして、WebSphere eXtreme Scale プロバイダーは、単一のサーバー・プロセスのサイズよりも大きいメモリー内の動的キャッシュのサービスを可能にします。デフォルトの動的キャッシュ・プロバイダーは、ディスク・キャッシュの使用に頼ることで、キャッシュ・インスタンスが大きくなって単一プロセスのサイズを超えることを可能にしています。多くのシチュエーションで、WebSphere eXtreme Scale プロバイダーを使用すると、実行に必要なディスク・キャッシュやコストの高いディスク・ストレージ・システムの必要性をなくすことができます。

リモート・トポロジー

リモート・トポロジーを使用しても、ディスク・キャッシュの必要性をなくすことができます。 リモート・トポロジーと組み込み区画化トポロジーとの唯一の相違点は、リモート・トポロジーを使用しているときは、キャッシュ・データのすべてが WebSphere Application Server プロセスの外側に保管されることです。WebSphere eXtreme Scale は、キャッシュ・データ用にスタンドアロンのコンテナー・プロセスをサポートします。これらのコンテナー・プロセスのオーバーヘッドは WebSphere Application Server プロセスよりも小さく、特定の Java 仮想マシン (JVM) を使用しなければならないという制限もありません。例えば、 32 ビット WebSphere Application Server プロセスによってアクセスされる動的キャッシュ・サービスのデータを、64 ビット JVM 上で実行している eXtreme Scale コンテナー・プロセス内に置くことが可能です。これによって、ユーザーは、64 ビット・プロセスの大きなメモリー容量をキャッシング用に利用できると同時に、アプリケーション・サーバー・プロセス用には 64 ビットの追加オーバーヘッドを負わなくてもすみます。次の図に、リモート・トポロジーを示します。



データ圧縮

WebSphere eXtreme Scale 動的キャッシュ・プロバイダーによって提供される、キャッシュ・オーバーヘッド管理についてユーザーを支援するもう 1 つのパフォーマンス機能が、圧縮です。デフォルトの動的キャッシュ・プロバイダーは、メモリー内のキャッシュ・データの圧縮を許可していません。eXtreme Scale プロバイダーを使用すると、これが可能になります。3 種類の分散トポロジーのどれでも、デフレート・アルゴリズムを使用するキャッシュ圧縮を使用可能にできます。圧縮を使用可能にすると、読み取りおよび書き込み操作のオーバーヘッドは増えますが、サーブレットおよび JSP キャッシングのようなアプリケーション用のキャッシュ密度は大幅に増加します。

ローカルのメモリー内のキャッシュ

WebSphere eXtreme Scale 動的キャッシュ・プロバイダーは、**レプリカ生成使用不可**の動的キャッシュ・インスタンスをバックアップするためにも使用できます。デフォルトの動的キャッシュ・プロバイダーと同じように、これらのキャッシュは非シリアライズ可能データを保管できます。また、eXtreme Scale コード・パスはメモリー内キャッシュの並行性を最大化するよう設計されているため、大容量のマルチプロセッサー・エンタープライズ・サーバー上ではデフォルト動的キャッシュ・プロバイダーよりも優れたパフォーマンスを提供します。

動的キャッシュ・エンジンおよび eXtreme Scale の機能の相違点

ローカルのメモリー内のキャッシュで、レプリカ生成が使用不可にされている場合は、デフォルトの動的キャッシュ・プロバイダーによってバックアップされたキャ

ッシュと WebSphere eXtreme Scale によってバックアップされたキャッシュとの間には、それほどの機能的な違いはありません。WebSphere eXtreme Scale がバックアップしたキャッシュは、メモリー内キャッシュのサイズに関係する統計および操作、またはディスク・オフロードをサポートしない点は除いて、ユーザーはこれら 2 つのキャッシュの機能的な違いに気付かないはずです。

レプリカ生成が使用可能にされているキャッシュの場合は、デフォルトの動的キャッシュ・プロバイダーを使用しているのか、それとも eXtreme Scale 動的キャッシュ・プロバイダーを使用しているのかに関わらず、ほとんどの動的キャッシュ API 呼び出しで戻される結果にはそれほどの相違はありません。一部の操作については、 eXtreme Scale を使用して動的キャッシュ・エンジンの動作をエミュレートできません。

動的キャッシュ統計

動的キャッシュ統計は、CacheMonitor アプリケーションまたは動的キャッシュ MBean を介して報告されます。eXtreme Scale 動的キャッシュ・プロバイダーを使用している場合でも、統計はこれらのインターフェースを通して報告されますが、統計値のコンテキストは異なります。

A、B、C という名前の 3 つのサーバー間で 1 つの動的キャッシュ・インスタンスが共有されている場合、動的キャッシュ統計オブジェクトは、その呼び出しが実行されたサーバーにあるキャッシュのコピーに関する統計だけを戻します。サーバー A で統計が取得された場合、その統計はサーバー A でのアクティビティーのみを反映します。

eXtreme Scale を使用している場合、すべてのサーバー間で共有されている分散キャッシュは 1 つしかありません。したがって、デフォルトの動的キャッシュ・プロバイダーのようにほとんどの統計値をサーバーごとにトラッキングするというのは不可能です。WebSphere eXtreme Scale 動的キャッシュ・プロバイダーを使用している場合に、キャッシュ統計 API によって報告される統計のリストと、それぞれの統計が何を表すのかを以下に示します。デフォルト・プロバイダーのように、これらの統計は同期化されていないため、並行ワークロードのために最高 10% は変わる可能性があります。

- **キャッシュ・ヒット** : キャッシュ・ヒットはサーバーごとにトラッキングされます。サーバー A 上のトラフィックが 10 キャッシュ・ヒットを生成し、サーバー B 上のトラフィックが 20 キャッシュ・ヒットを生成する場合、キャッシュ統計は、サーバー A での 10 キャッシュ・ヒットとサーバー B での 20 キャッシュ・ヒットを報告します。
- **キャッシュ・ミス**: キャッシュ・ミスは、キャッシュ・ヒットと同様、サーバーごとにトラッキングされます。
- **メモリー・キャッシュ・エントリー数**: この統計値は、分散キャッシュ内のキャッシュ・エントリーの数を報告します。この統計値に関しては、キャッシュにアクセスするすべてのサーバーが同じ値を報告し、その値は、サーバーすべてのメモリー内のキャッシュ・エントリーの総数です。
- **メモリー・キャッシュ・サイズ (MB)**: このメトリックは、リモート・トポロジー、組み込みトポロジー、または組み込み区画化トポロジーを使用するキャッシュに対してのみサポートされます。 これは、グリッド全体でキャッシュが消費する Java ヒープ・スペースのメガバイト数を報告します。 この統計は、プライマ

リー区画に対するヒープ使用量のみを報告します。レプリカを考慮してください。 リモート・トポロジーおよび組み込み区画化トポロジーのデフォルト設定は 1 つの非同期レプリカであるため、キャッシュのメモリー消費量を正確に知るには、この数値を 2 倍してください。

- **キャッシュ除去**: この統計値は、任意の方法でキャッシュから除去されたエントリーの総数を報告し、分散キャッシュ全体の集約値です。サーバー A 上のトラフィックが 10 の無効化を生成し、サーバー B 上のトラフィックが 20 の無効化を生成する場合、両サーバーの値は 30 になります。
- **キャッシュ最長未使用時間 (LRU) 除去**: この統計値は、キャッシュ除去と同様、集約値です。キャッシュを最大サイズより小さく保っておくために除去されたエントリーの数がトラッキングされます。
- **タイムアウト無効化**: これも集約の統計値であり、タイムアウトになったために除去されたエントリーの数をトラッキングします。
- **明示的無効化**: これも集約の統計値であり、キー、依存関係 ID、またはテンプレートによる直接的な無効化で除去されたエントリーの数をトラッキングします。
- **拡張統計**: eXtreme Scale 動的キャッシュ・プロバイダーは、以下の拡張統計キー・ストリングをエクスポートします。
  - **com.ibm.websphere.xs.dynacache.remote_hits**: eXtreme Scale コンテナーでトラッキングされたキャッシュ・ヒットの総数。これは、集約統計値であり、拡張統計マップ内ではこの値は `long` です。
  - **com.ibm.websphere.xs.dynacache.remote_misses**: eXtreme Scale コンテナーでトラッキングされたキャッシュ・ミスの総数。集約統計値であり、拡張統計マップ内ではこの値は `long` です。

統計リセットの報告

動的キャッシュ・プロバイダーを使用して、キャッシュ統計をリセットすることができます。デフォルト・プロバイダーでは、リセット操作でクリアされるのは、影響を受けるサーバーの統計のみです。 eXtreme Scale 動的キャッシュ・プロバイダーは、リモート・キャッシュ・コンテナーの統計データの大部分をトラッキングします。このデータは、統計がリセットされたときに、クリアされることも、変更されることもありません。代わりに、デフォルト動的キャッシュ動作がクライアント上でシミュレートされ、ある統計の現行値と、そのサーバーで最後にリセットが呼び出されたときのその統計の値との差が報告されます。

例えば、サーバー A 上のトラフィックが 10 のキャッシュ除去を生成する場合、サーバー A とサーバー B の統計は 10 の除去を報告します。サーバー B の統計がリセットされ、サーバー A 上のトラフィックが追加で 10 の除去を生成したとすると、サーバー A の統計は 20 の除去を報告し、サーバー B の統計は 10 の除去を報告します。

動的キャッシュ・イベント

動的キャッシュ API を使用して、ユーザーはイベント・リスナーを登録できます。動的キャッシュ・プロバイダーとして eXtreme Scale を使用している場合、イベント・リスナーはローカルのメモリー内キャッシュに対して、予期されるように機能します。

分散キャッシュに対するイベント動作は、使用されるトポロジーに依存します。組み込みトポロジーを使用しているキャッシュの場合、イベントは書き込み操作を処理するサーバー (つまり、プライマリー断片) で生成されます。これは、1 つのサーバーのみがイベント通知を受け取ることを意味しますが、動的キャッシュ・プロバイダーで一般的に予期されるイベント通知はすべてそのサーバーが受け取ります。WebSphere eXtreme Scale は実行時にプライマリー断片を選択するため、ある特定のサーバー・プロセスが常にこれらのイベントを受け取るように保証することはできません。

組み込み区画化キャッシュは、キャッシュのいずれかの区画をホストするどのサーバー上でも、イベントを生成します。 したがって、11 の区画に分割されたキャッシュがあり、WebSphere Application Server Network Deployment グリッドの 11 のサーバーそれぞれが 1 つの区画をホストしている場合、各サーバーは、そのサーバーがホストしているキャッシュ・エントリーの動的キャッシュ・イベントを受け取ります。11 すべての区画が 1 つのサーバー・プロセスでホストされているのでない限り、1 つのサーバー・プロセスだけがイベントのすべてを認識するということはありません。組み込みトポロジーの場合と同様、ある特定のサーバー・プロセスが、ある特定のイベント・セットまたはイベントを受け取るように保証することはできません。

リモート・トポロジーを使用するキャッシュは、動的キャッシュ・イベントをサポートしません。

MBean 呼び出し

WebSphere eXtreme Scale 動的キャッシュ・プロバイダーはディスク・キャッシングをサポートしません。ディスク・キャッシングに関係する MBean 呼び出しはすべて機能しません。

動的キャッシュ・レプリカ生成ポリシーのマッピング

WebSphere Application Server 組み込み動的キャッシュ・プロバイダーは、複数のキャッシュ・レプリカ生成ポリシーをサポートします。これらのポリシーは、グローバルに構成するか、各キャッシュ・エントリーに対して構成することができます。動的キャッシュ資料で、これらのポリシーに関する説明を参照してください。

eXtreme Scale 動的キャッシュ・プロバイダーは、直接これらのポリシーに従うわけではありません。キャッシュのレプリカ生成に関する特性は、動的キャッシュ・サービスによってエントリーに設定されるレプリカ生成ポリシーに関わらず、構成された eXtreme Scale 分散トポロジー・タイプによって決まり、そのキャッシュ内に置かれるすべての値に適用されます。以下に、動的キャッシュ・サービスでサポートされるレプリカ生成ポリシーのすべてをリストし、どの eXtreme Scale トポロジーが類似のレプリカ生成特性を提供するのかを示します。

eXtreme Scale 動的キャッシュ・プロバイダーは、キャッシュまたはキャッシュ・エントリーに対する DRS レプリカ生成ポリシー設定を無視することに注意してください。ユーザーは、レプリカ生成のニーズに適したトポロジーを選択する必要があります。

- NOT_SHARED – 現在は、eXtreme Scale 動的キャッシュ・プロバイダーによって提供されるトポロジーのうち、このポリシーに近似するものはありません。これ

は、キャッシュに保管されるすべてのデータは、java.io.Serializable を実装するキーおよび値を持っていなければならないことを意味します。

- SHARED_PUSH – 組み込みトポロジーが、このレプリカ生成ポリシーに近似しています。キャッシュ・エントリーが作成されると、そのエントリーはすべてのサーバーに複製されます。サーバーは、キャッシュ・エントリーをローカルで検索するだけです。エントリーがローカルで検出されなかった場合は存在しないと見なされ、そのエントリーを探すために他のサーバーが照会されることはありません。
- SHARED_PULL および SHARED_PUSH_PULL – 組み込み区画化トポロジーおよびリモート・トポロジーが、このレプリカ生成ポリシーに近似しています。キャッシュの分散状態は、すべてのサーバー間で完全に一貫しています。

この情報が主として提供されるので、トポロジーをユーザーの分散整合性ニーズに確実に合わせることができます。例えば、ユーザーのデプロイメントおよびパフォーマンスのニーズには組み込みトポロジーが適しているが、SHARED_PUSH_PULL で提供されるレベルのキャッシュ整合性を必要とする場合、パフォーマンスが少し劣ることになっても、組み込み区画化トポロジーを使用することを検討してください。

セキュリティー

組み込みトポロジーまたは組み込み区画化トポロジーで実行している動的キャッシュ・インスタンスを、WebSphere Application Server に構築されたセキュリティー機能を使用して保護できます。WebSphere Application Server インフォメーション・センターでアプリケーション・サーバーの保護を参照してください。

リモート・トポロジーでキャッシュが実行している場合、スタンドアロン eXtreme Scale クライアントはそのキャッシュに接続して、動的キャッシュ・インスタンスの内容に影響を与えることが可能です。eXtreme Scale 動的キャッシュ・プロバイダーに備わっている低オーバーヘッドの暗号化機能は、非 WebSphere Application Server クライアントによるキャッシュ・データの読み取りまたは変更を防ぐことができます。この機能を使用可能にするには、オプション・パラメーター **com.ibm.websphere.xs.dynacache.encryption_password** を、動的キャッシュ・プロバイダーにアクセスするすべての WebSphere Application Server インスタンスで同じ値に設定します。これによって、128 ビット AES 暗号化を使用して CacheEntry の値およびユーザー・メタデータが暗号化されます。 すべてのサーバーで同じ値に設定されていることが重要です。サーバーは、このパラメーターに異なる値が設定されているサーバーによってキャッシュに入れられたデータを読み取ることはできません。

eXtreme Scale プロバイダーは、同じキャッシュでこの変数に異なる値が設定されていることを検出すると、警告を生成して eXtreme Scale コンテナー・プロセスのログに入れます。

SSL またはクライアント認証が必要な場合は、143 ページの『セキュリティーの概要』に関する eXtreme Scale 資料を参照してください。

追加情報

- 動的キャッシュに関するレッドブック
- 動的キャッシュ文書
  - WebSphere Application Server 7.0
  - WebSphere Application Server 6.1
- DRS 資料
  - WebSphere Application Server 7.0
  - WebSphere Application Server 6.1

データベース統合: 後書き、インライン、およびサイド・キャッシング

WebSphere eXtreme Scale が使用される目的は、従来のデータベースをその背後に置くことで、通常はデータベースにプッシュされる読み取りアクティビティーをなくすことです。コヒーレント・キャッシュは、オブジェクト関連マッパーを直接または間接に使用することにより、アプリケーションで使用できます。コヒーレント・キャッシュは、データベースまたは読み取りからの下流工程の負荷を軽減します。シナリオがもう少し複雑で、一部のデータのみが従来のパーシスタンス保証を必要とするデータ・セットへのトランザクション・アクセスなどの場合は、フィルター操作を使用して書き込みトランザクションの負荷を軽減します。

WebSphere eXtreme Scale は、高度にフレキシブルなメモリー内のデータベース処理スペースとして機能するように構成できます。ただし、WebSphere eXtreme Scale は、オブジェクト・リレーショナル・マッパー (ORM) ではありません。データ・グリッドに含まれているデータがどこから取得されたのかを認識しません。アプリケーションまたは ORM は、データを eXtreme Scale サーバーに配置できます。データの発生元であるデータベースとの一貫性を保つのは、データのソースの責任です。これは、データベースから取り出されたデータを eXtreme Scale は自動的に無効化できないことを意味します。アプリケーションまたはマッパーは、この機能を提供して、eXtreme Scale に保管されているデータを管理する必要があります。

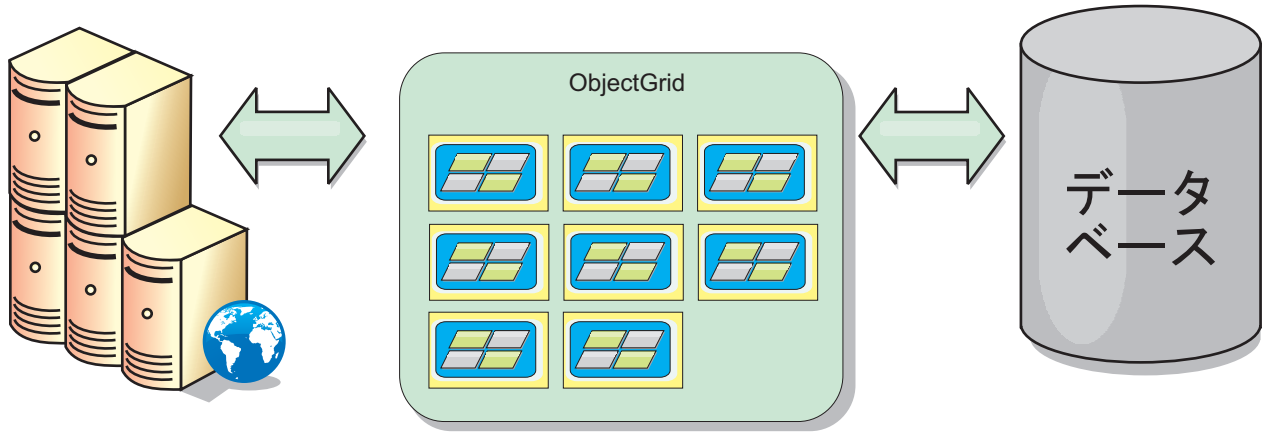


図 17. データベース・バッファーとしての ObjectGrid

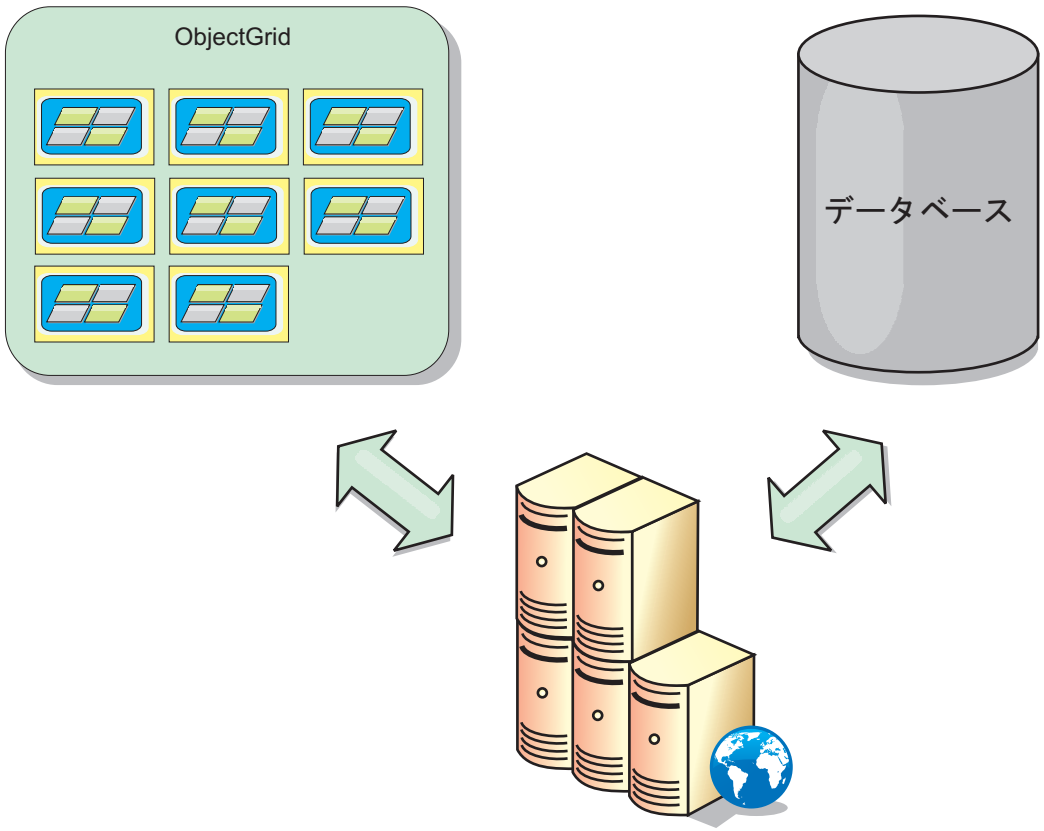


図 18. サイド・キャッシュとしての ObjectGrid

スパース・キャッシュおよび完全キャッシュ

WebSphere eXtreme Scale は、スパース・キャッシュまたは完全キャッシュとして使用できます。完全キャッシュがデータすべてを保持する一方で、スパース・キャッシュはデータ全体のサブセットしか保持しません。必要時には、データをゆっくりと取り込むことができます。通常、スパース・キャッシュは、データが部分的にしか使用可能でないため、キーを使用して (索引や照会を使用せず) アクセスされます。

スパース・キャッシュ

キーがスパース・キャッシュに存在しない場合、またはデータが使用できず、キャッシュ・ミスが発生している場合は、次の層が呼び出されます。データは、例えば、データベースからフェッチされ、データ・グリッド・キャッシュ層に挿入されます。照会または索引を使用する場合、現在ロードされている値のみがアクセスされ、要求は他の層に転送されません。

完全キャッシュ

完全キャッシュには必要なすべてのデータが含まれ、索引または照会により非キー属性を使用してアクセスできます。データベースから完全キャッシュにデータがプリロードされた後、アプリケーションはデータへのアクセスを試みます。データがロードされた後は、完全キャッシュをデータベースの代わりとして使用できます。

すべてのデータがあるので、照会および索引を使用して、データの検出と集約を行うことができます。

サイド・キャッシュ

WebSphere eXtreme Scale をサイド・キャッシュとして使用する場合は、データ・グリッドと一緒にバックエンドが使用されます。

サイド・キャッシュ

アプリケーションのデータ・アクセス層のサイド・キャッシュとしてこの製品を構成できます。このシナリオの場合、WebSphere eXtreme Scale は、通常であればバックエンド・データベースから取得されるオブジェクトを一時的に保管するために使用されます。アプリケーションは、データがデータ・グリッドに含まれているかどうかチェックします。データがデータ・グリッドにあった場合、そのデータが呼び出し元に返されます。データがない場合、データがバックエンド・データベースから取得されます。そして、次の要求がキャッシュ・コピーを使用できるように、データがデータ・グリッドに挿入されます。次の図は、OpenJPA や Hibernate などの任意のデータ・アクセス層で WebSphere eXtreme Scale をサイド・キャッシュとして使用する方法を示しています。

Hibernate および OpenJPA 向けサイド・キャッシュ・プラグイン

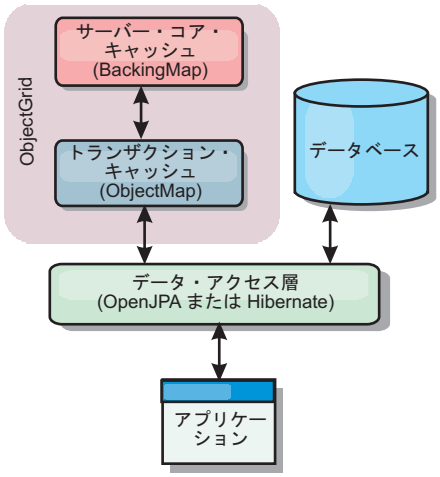


図 19. サイド・キャッシュ

WebSphere eXtreme Scale には、この製品を自動サイド・キャッシュとして使用できるようにする、OpenJPA 用と Hibernate 用のどちらのキャッシュ・プラグインも組み込まれています。WebSphere eXtreme Scale をキャッシュ・プロバイダーとして使用すると、データの読み取りおよび照会時のパフォーマンスが高まり、データベースへの負荷が軽減されます。WebSphere eXtreme Scale ではキャッシュが自動的にすべてのプロセス間で複製されるので、組み込みキャッシュ実装をしのぐ利点があります。あるクライアントが値をキャッシュに入れると、他のすべてのクライアントがキャッシュに入れられた値を使用できるようになります。

インライン・キャッシュ

インライン・キャッシングは、データベース・バックエンドに構成することも、データベースのサイド・キャッシュとして構成することもできます。インライン・キャッシングは、データと対話するための基本手段として eXtreme Scale を使用します。eXtreme Scale がインライン・キャッシュとして使用される場合、アプリケーションは、Loader プラグインを使用してバックエンドと対話します。

インライン・キャッシュ

インライン・キャッシュとして使用される場合、WebSphere eXtreme Scale は Loader プラグインを使用してバックエンドと対話します。このシナリオでは、アプリケーションが直接 eXtreme Scale API にアクセスできるため、データ・アクセスが単純化されます。キャッシュ内のデータとバックエンドのデータが確実に同期されるようにするための数種類のキャッシング・シナリオが、eXtreme Scale においてポートされています。次の図は、インライン・キャッシュがアプリケーションおよびバックエンドと対話する方法を示しています。

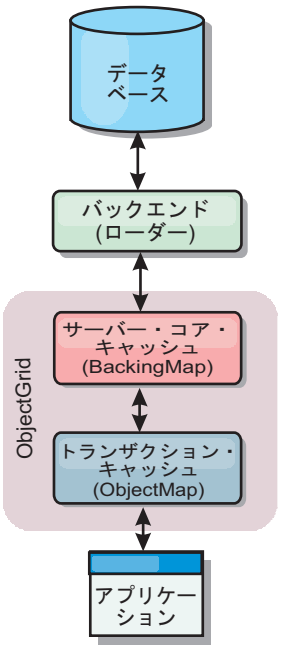


図 20. インライン・キャッシュ

インライン・キャッシング・オプションにより、アプリケーションが eXtreme Scale API に直接アクセスできるようになるため、データ・アクセスが単純化されます。WebSphere eXtreme Scale は、以下のような複数のインライン・キャッシング・シナリオをサポートします。

- リードスルー
- ライトスルー
- 後書き

リードスルー・キャッシングのシナリオ

リードスルー・キャッシュは、データ・エントリーの要求時にキーによるそのロードが暫時的に行われるスパース・キャッシュです。これが行われる場合、呼び出し元は、エントリーがどのように取り込まれるかを知る必要はありません。データが eXtreme Scale キャッシュに見つからない場合、eXtreme Scale は、その欠落データを Loader プラグインから取得します。このプラグインは、バックエンド・データベースからデータをロードして、そのデータをキャッシュに挿入します。同じデータ・キーに対する後続の要求は、削除、無効化、または除去されるまでキャッシュに存在します。

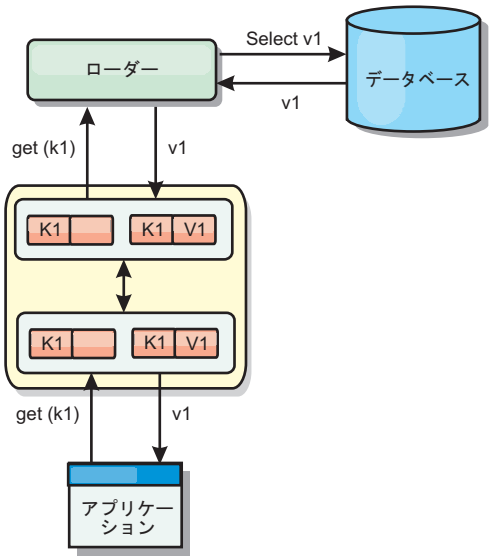


図 21. リードスルー・キャッシング

ライトスルー・キャッシングのシナリオ

ライトスルー・キャッシュでは、キャッシュへの書き込みが行われるたびに、ローダーを使用してデータベースへの書き込みが同期的に行われます。このメソッドでは、バックエンドとの整合性はありますが、データベース操作が同期されるため、書き込みパフォーマンスは低下します。キャッシュとデータベースがともに更新されるため、同じデータに対する後続の読み取りはキャッシュに残り、データベース呼び出しが回避されます。ライトスルー・キャッシュは、多くの場合、リードスルー・キャッシュと一緒に使用されます。

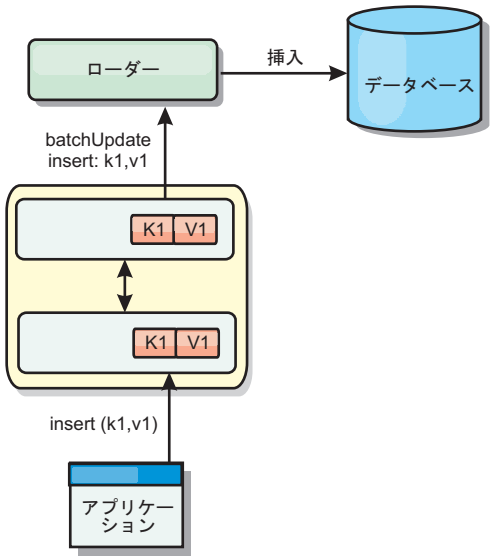


図 22. ライトスルー・キャッシング

後書きキャッシングのシナリオ

変更を非同期的に書き込むことにより、データベースの同期性が改善されます。後書きキャッシュまたはライト・バック・キャッシュとも呼ばれます。通常はローダーに対して同期的に書き込まれる変更は、eXtreme Scale 内でバッファー化されてから、バックグラウンド・スレッドを使用してデータベースに書き込まれます。データベース操作をクライアント・トランザクションから除去し、データベース書き込みを圧縮できるため、書き込みパフォーマンスが著しく向上します。

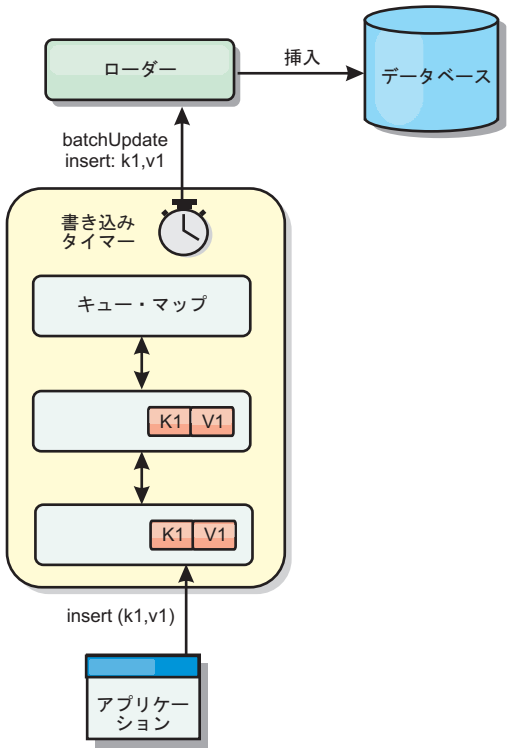


図 23. 後書きキャッシング

後書きキャッシング

後書きキャッシングを使用して、バックエンドとして使用しているデータベースを更新する際に発生するオーバーヘッドを減らすことができます。

後書きキャッシングの概要

後書きキャッシングでは、Loader プラグインの更新が非同期にキューに入れられます。eXtreme Scale トランザクションをデータベース・トランザクションから分離することにより、マップの更新、挿入、および除去の、パフォーマンスを改善できます。非同期更新は、時間ベースの遅延 (例えば 5 分) またはエントリー・ベースの遅延 (例えば 1000 エントリー) 後に実行されます。

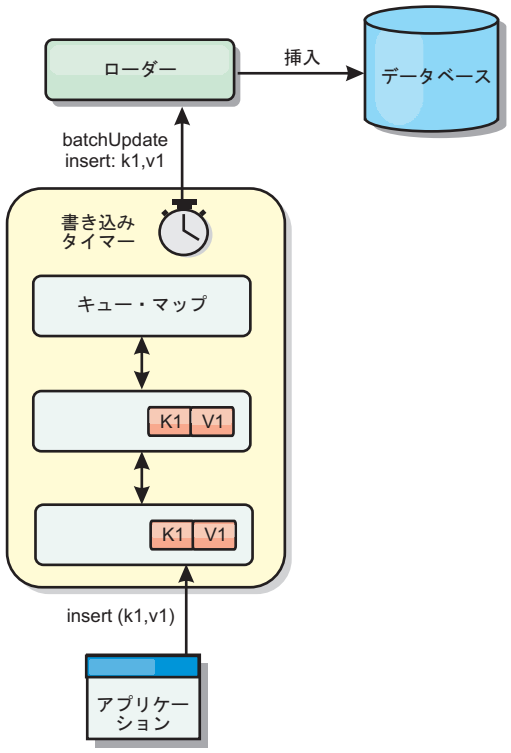


図 24. 後書きキャッシング

BackingMap の後書き構成により、ローダーとマップとの間にスレッドが作成されます。次に、ローダーは、BackingMap.setWriteBehind メソッド内の構成設定に従って、そのスレッドを通してデータ要求を委任します。eXtreme Scale トランザクションが、マップのエントリーを挿入、更新、または削除すると、これらの各レコードごとに 1 つずつ LogElement オブジェクトが作成されます。これらのエレメントは後書きローダーに送信され、キュー・マップと呼ばれる特別な ObjectMap 内でキューに入れられます。後書き設定が有効になっているバックアップ・マップは、それぞれ独自のキュー・マップを持っています。後書きスレッドは、キューに入れられたデータをキュー・マップから定期的に除去して、実際のバックエンド・ローダーにプッシュします。

後書きローダーは、挿入、更新、および削除タイプの LogElement オブジェクトのみを実際のローダーに送信します。それ以外のタイプの LogElement オブジェクト (例えば、EVICT タイプ) はすべて無視されます。

後書きサポートは、eXtreme Scale をデータベースに組み込む際に使用する Loader プラグインの *拡張機能* です。例えば、JPA ローダーの構成については JPA ローダーの構成 の情報を参照してください。

利点

後書きサポートを使用可能にすると、以下のような利点があります。

- **バックエンド障害の分離:** 後書きキャッシングは、バックエンド障害からの分離層を提供します。バックエンドのデータベースで障害が発生すると、更新はキュー・マップ内でキューに入れられます。アプリケーションは、トランザクション

を eXtreme Scale に送り続けることができます。バックエンドが復旧すると、キュー・マップ内のデータはバックエンドにプッシュされます。

- **バックエンドの負荷の削減:** 後書きローダーは更新をキー単位でマージします。その結果、キュー・マップ内には、キーごとにマージされた更新が 1 つのみ存在します。このマージにより、バックエンド・データベースに対する更新の数が減ります。
- トランザクション・パフォーマンスの改善: データがバックエンドと同期されるのをトランザクションが待機する必要がないので、個別の eXtreme Scale トランザクション時間が削減されます。

アプリケーション設計に関する考慮事項

後書きサポートを使用可能にすることは簡単ですが、後書きサポートを扱うアプリケーションを設計する際には、注意すべき考慮事項があります。後書きサポートがない場合、ObjectGrid トランザクションにバックエンド・トランザクションが包含されます。 ObjectGrid トランザクションはバックエンド・トランザクションの開始前に開始し、バックエンド・トランザクションの終了後に終了します。

後書きサポートが有効な場合、ObjectGrid トランザクションは、バックエンド・トランザクションが開始する前に終了します。ObjectGrid トランザクションとバックエンド・トランザクションは切り離されます。

参照保全性の制約

後書きサポートで構成されているそれぞれのバックアップ・マップは、データをバックエンドにプッシュするための独自の後書きスレッドを持ちます。したがって、1 つの ObjectGrid トランザクションにさまざまなマップを更新するデータが含まれていても、バックエンドでは、それぞれ異なるバックエンド・トランザクションでデータの更新が行われます。例えば、トランザクション T1 はマップ Map1 のキー key1 とマップ Map2 のキー key2 を更新するとします。マップ Map1 に対する key1 更新は、1 つのバックエンド・トランザクションでバックエンドに対して更新され、マップ Map2 に対する key2 更新は、異なる後書きスレッドにより別のバックエンド・トランザクションでバックエンドに対して更新されます。Map1 に保管されたデータと Map2 に保管されたデータがバックエンドでの外部キー制約などの関係を持つ場合、更新が失敗する可能性があります。

バックエンド・データベースの参照保全性制約を設計するときは、順不同の更新に必ず対応できるようにしてください。

キュー・マップのロックの振る舞い

トランザクションの動作で他に大きく異なる点は、ロックの振る舞いです。ObjectGrid は、PESSIMISTIC、OPTIMISITIC、および NONE の 3 つの異なるロック・ストラテジーをサポートします。後書きキュー・マップは、 *バックアップ・マップに構成されているロック・ストラテジーに関係なく、ペシミスティック・ロック・ストラテジーを使用します。キュー・マップのロックを取得する操作には 2 つの異なるタイプがあります。

- ObjectGrid トランザクションのコミット時、またはフラッシュ (マップ・フラッシュまたはセッション・フラッシュ) の発生時、トランザクションはキュー・マップ内のキーを読み取り、キーに S ロックをかけます。

- ObjectGrid トランザクションのコミット時、トランザクションは、キーの S ロックを X ロックにアップグレードしようとします。

キュー・マップのこの余分な動作のため、ロックの動作に少々違いがあります。

- ユーザー・マップがペシミスティック・ロック・ストラテジーで構成されている場合、ロックの動作にほとんど違いはありません。フラッシュまたはコミットが呼び出されるたび、キュー・マップ内の同じキーに S ロックがかけられます。コミット時間中、ユーザー・マップ内のキーに X ロックが取得されるだけでなく、キュー・マップ内のキーに対しても X ロックが取得されます。
- ユーザー・マップが OPTIMISTIC または NONE ロック・ストラテジーで構成されている場合、ユーザー・トランザクションは PESSIMISTIC ロック・ストラテジーのパターンに従います。フラッシュまたはコミットが呼び出されるたびに、キュー・マップ内の同じキーに対して S ロックが取得されます。コミット時間の間、同じトランザクションを使用するキュー・マップ内のキーに対して X ロックが設定されます。

ローダー・トランザクションの再試行

ObjectGrid は、2 フェーズ・トランザクションまたは XA トランザクションをサポートしません。後書きスレッドは、キュー・マップからレコードを除去して、バックエンドに対してそのレコードを更新します。トランザクションの最中にサーバーに障害が起こると、一部のバックエンドの更新が失われる可能性があります。

後書きローダーは、失敗したトランザクションの書き込みを自動的に再試行し、データ損失を防ぐために未確定 LogSequence をバックエンドに送信します。このアクションを行うには、ローダーがべき等である必要があります。この意味は、Loader.batchUpdate(TxId, LogSequence) が同じ値で 2 回呼び出されたとき、それは適用された回数があたかも 1 回だったかのように、同じ結果を返すということです。ローダー実装は、この機能を使用可能にするため、RetryableLoader インターフェースを実装しなければなりません。詳しくは、API 資料を参照してください。

ローダーの障害

Loader プラグインは、バックエンド・データベースと通信できない場合、失敗することがあります。これは、データベース・サーバーまたはネットワーク接続がダウンしている場合に発生することがあります。後書きローダーは、更新をキューに入れ、データ変更を定期的にローダーにプッシュしようと試みます。ローダーは、LoaderNotAvailableException 例外をスローして、データベース接続の問題があることを ObjectGrid ランタイムに通知しなければなりません。

したがって、ローダー実装で、データ障害または物理的ローダー障害を識別できるようになっている必要があります。データ障害は LoaderException または OptimisticCollisionException としてスローまたは再スローされる必要がありますが、物理的なローダーの障害は LoaderNotAvailableException としてスローまたは再スローされる必要があります。 ObjectGrid は、これら 2 つの例外を異なる方法で処理します。

- LoaderException が後書きローダーによってキャッチされると、重複キー障害などのある種のデータ障害のため、後書きローダーはそれを障害とみなします。後書きローダーは、更新のバッチ処理を解除し、データ障害を分離するため、1 度に

1 レコードずつ更新しようとします。1 レコードの更新時に再度 {{LoaderException}} がキャッチされると、失敗した更新レコードが作成され、失敗した更新マップのログに記録されます。

- LoaderNotAvailableException が後書きローダーによってキャッチされると、データベース・エンドに接続できない (例えば、データベース・バックエンドがダウンしている、データベース接続が使用可能でない、ネットワークがダウンしているなど) ため、後書きローダーはそれを障害とみなします。後書きローダーは 15 秒待ってから、データベースへのバッチ更新を再試行します。

一般的な間違いは、LoaderNotAvailableException がスローされるべきなのに、LoaderException がスローされることです。後書きローダーでキューに入れられたすべてのレコードは、失敗更新レコードとなります。このような場合、バックエンド障害分離の目的が果たせなくなります。

パフォーマンスの考慮事項

後書きキャッシング・サポートの場合、ローダー更新をトランザクションから除去することで、応答時間が増加します。また、データベース更新が結合されるため、データベース・スループットも増加します。データをキュー・マップからプルし、ローダーにプッシュされる後書きスレッドの導入によって生じるオーバーヘッドを理解しておく必要があります。

予想される使用パターンおよび環境に基づいて、最大更新数または最大更新時間を調整する必要があります。最大更新カウントまたは最大更新時間の値が小さすぎると、後書きスレッドのオーバーヘッドが、その利点を帳消しにするおそれがあります。これら 2 つのパラメーターに大きな値を設定する場合も、データのキューイングに必要なメモリー使用が増え、データベース・レコードが不整合になる時間が増加するおそれがあります。

最善のパフォーマンスを得るために、後書き関係のパラメーターは、以下の要因を考慮に入れて調整してください。

- 読み取りトランザクションと書き込みトランザクションの比率
- 同一レコード更新の頻度
- データベース更新の待ち時間

ローダー

Loader プラグインを使用すると、通常は、同一システムあるいは別システムの永続ストアに保持されるデータのメモリー・キャッシュとしてデータ・グリッド・マップを動作させることができます。通常、データベースまたはファイル・システムは永続ストアとして使用されます。リモート Java 仮想マシン (JVM) もデータのソースとして使用でき、eXtreme Scale を使用してハブ・ベースのキャッシュを構築できます。ローダーには、永続ストアとの間でデータの読み取りおよび書き込みを行うロジックが備わっています。

概説

ローダーは、変更がバックアップ・マップに対して行われた場合、または、バックアップ・マップがデータ要求を満足できない (キャッシュ・ミス) 場合に呼び出されるバックアップ・マップ・プラグインです。ローダーは、キーに関する要求をキャ

ッシュが満足できなくなったときに起動され、リードスルー機能や、キャッシュにデータをゆっくり設定する機能を提供します。また、ローダーによって、キャッシュ値が変わったときのデータベース更新が可能になります。1 つのトランザクション内のすべての変更は、データベースとの対話の数を最小化できるよう、まとめてグループ化されます。ローダーと共に TransactionCallback プラグインが、バックエンド・トランザクションの境界をトリガーするために使用されます。このプラグインの使用は、複数のマップが 1 つのトランザクションに含まれている場合、または、トランザクション・データがコミットなしでキャッシュに書き込まれる場合に重要です。

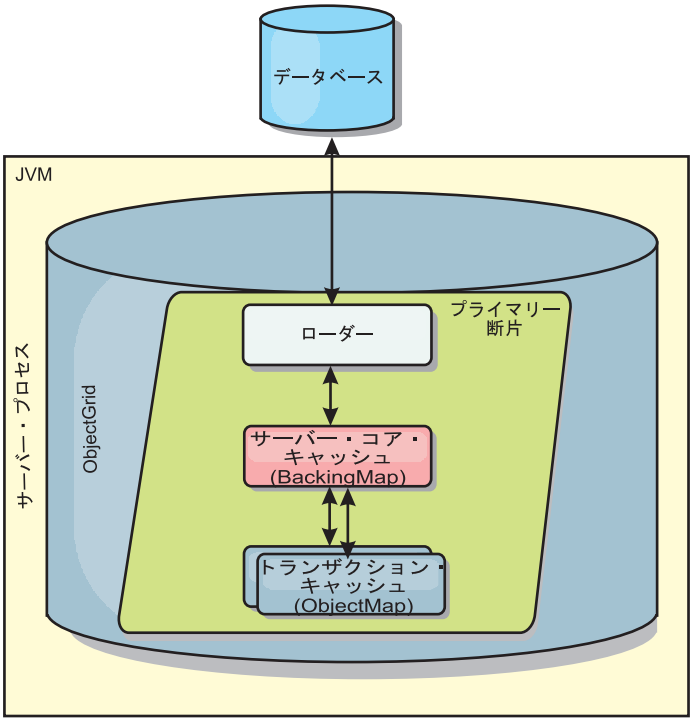


図 25. ローダー

ローダーは、データベース・ロックの保持を回避するために、資格過剰の更新を使用することもできます。バージョン属性をキャッシュ値の中に入れることによって、値がキャッシュ内で更新されるときにローダーは値の前と後のイメージを見ることができます。その後、データベースまたはバックエンドを更新する際にこの値を使用して、データが更新されていないことを検証できます。 ローダーは、開始時にデータ・グリッドをプリロードするよう構成することもできます。区画に分割されている場合、各区画ごとに 1 つのローダー・インスタンスが関連付けられます。例えば、「Company」マップに 10 個の区画がある場合、プライマリー区画ごとに 1 つずつ、10 個のローダー・インスタンスがあります。このマップのプライマリー断片がアクティブにされると、ローダーに対して preloadMap メソッドが同期または非同期で呼び出され、マップ区画にバックエンドからのデータが自動的にロードされます。非同期で呼び出される場合、すべてのクライアント・トランザクションはブロックされ、データ・グリッドへの矛盾するアクセスを防止します。代わりに、クライアント・プリローダーを使用してデータ・グリッド全体にデータをロードできます。

2 つの組み込みローダーにより、リレーショナル・データベース・バックエンドとの統合が非常に単純化されます。JPA ローダーは、Java Persistence API (JPA) 仕様の OpenJPA および Hibernate 実装の両方のオブジェクト関係マッピング (ORM) 機能を使用します。詳しくは、 70 ページの『JPA ローダー』を参照してください。

複数データ・センター構成でローダーを使用する場合は、どのようにして改訂データとキャッシュの整合性をデータ・グリッド間で維持するかを検討する必要があります。詳しくは、 182 ページの『マルチマスター・トポロジーでのローダーについての考慮事項』を参照してください。

ローダーの構成

ローダーを BackingMap 構成に追加するには、プログラマチック構成または XML 構成を使用します。ローダーには、バックアップ・マップとの間で以下のような関係があります。

- 1 つのバックアップ・マップは 1 つのローダーしか持てない。
- クライアント・バックアップ・マップ (ニア・キャッシュ) はローダーを持てない。
- 1 つのローダー定義を複数のバックアップ・マップに適用できるが、各バックアップ・マップは独自のローダー・インスタンスを持つ。

データのプリロードおよびウォームアップ

ローダーのユーザーを組み込む多くのシナリオで、データ・グリッドをデータと一緒にプリロードして準備しておくことができます。

データ・グリッドは、完全キャッシュとして使用される場合、データのすべてを保持しなければならず、いずれかのクライアントが接続する前にデータがロードされている必要があります。スパース・キャッシュを使用する場合は、クライアントが接続時にデータにすぐにアクセスできるよう、キャッシュをデータでウォームアップしておくことができます。

以下のセクションで説明するように、データをデータ・グリッドにプリロードする方法は 2 つあります。1 つは Loader プラグインを使用する方法で、もう 1 つはクライアント・ローダーを使用する方法です。

Loader プラグイン

Loader プラグインは、各マップに関連付けられ、1 つのプライマリー区画断片をデータベースと同期化させる役割を担います。断片がアクティブになると、Loader プラグインの preloadMap メソッドが自動的に呼び出されます。 例えば、100 の区画がある場合、ローダーのインスタンスは 100 存在し、それぞれが、各自の区画のためにデータをロードします。同期的に実行された場合、プリロードが完了するまですべてのクライアントがブロックされます。

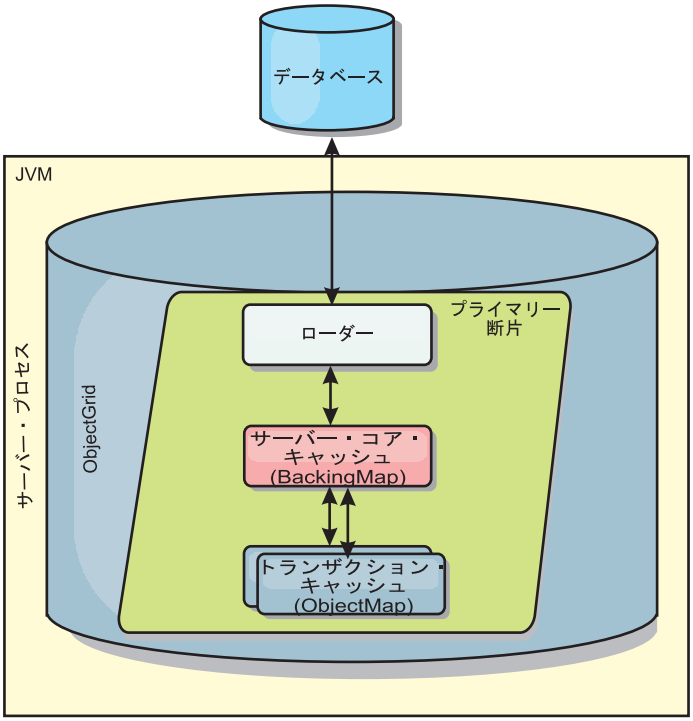


図 26. Loader プラグイン

クライアント・ローダー

クライアント・ローダーは、1 つ以上のクライアントを使用してグリッドにデータをロードするパターンです。複数のクライアントを使用してグリッドにデータをロードすることは、区画スキーマがデータベースに保管されない場合は効率的です。クライアント・ローダーは手動で呼び出すか、データ・グリッドの開始時に自動的に呼び出すことができます。データ・グリッドにデータをプリロードしている間はクライアントがデータ・グリッドにアクセスできないように、クライアント・ローダーは、オプションで、StateManager を使用してデータ・グリッドの状態をプリロード・モードに設定できます。WebSphere eXtreme Scale には Java Persistence API (JPA) ベースのローダーが組み込まれていて、OpenJPA または Hibernate JPA プロバイダーのどちらかでデータ・グリッドに自動的にロードするために使用できます。キャッシュ・プロバイダーについて詳しくは、27 ページの『JPA レベル 2 (L2) キャッシュ・プラグイン』を参照してください。

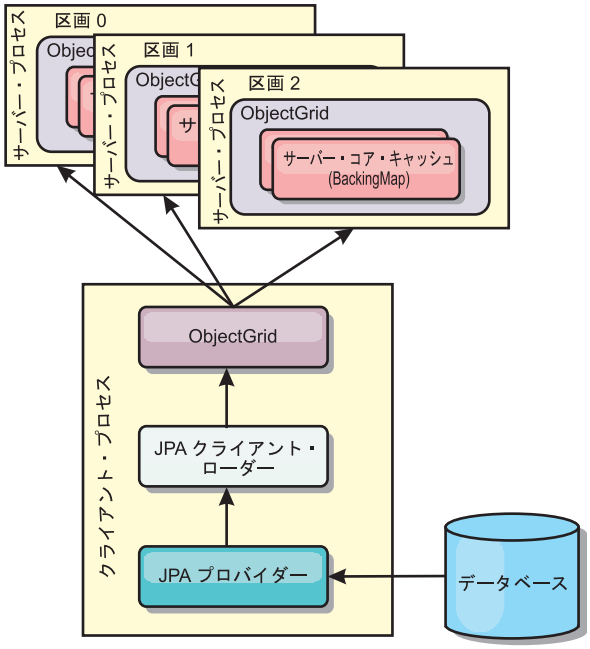


図 27. クライアント・ローダー

データベースの同期手法

WebSphere eXtreme Scale をキャッシュとして使用する際、データベースを eXtreme Scale トランザクションとは独立して更新できる場合、失効データを許容するようにアプリケーションを作成する必要があります。同期されたメモリー内データベース処理スペースとして機能するため、eXtreme Scale はキャッシュを常に最新の状態に保つ方法をいくつか備えています。

データベースの同期手法

定期的リフレッシュ

時間ベースの Java Persistence API (JPA) データベース・アップデーターを使用して、定期的なキャッシュの無効化または更新を自動的に実行できます。このアップデーターは、JPA プロバイダーを使用してデータベースを定期的に照会することによって、前回の更新以降に発生した更新または挿入があるかどうかを調べます。示された変更は、スパース・キャッシュで使用された場合、自動的に無効にされるか、更新されます。完全キャッシュで使用された場合、エントリーをディスカバーして、キャッシュに挿入することができます。エントリーがキャッシュから除去されることはありません。

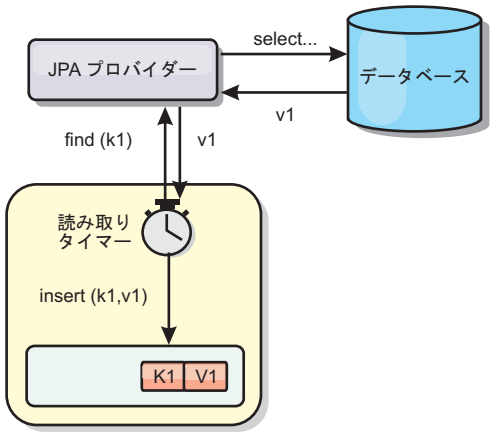


図 28. 定期的リフレッシュ

除去

スパース・キャッシュでは、除去ポリシーを使用して、データベースに影響を及ぼすことなく、キャッシュからデータを自動的に除去できます。eXtreme Scale には、Time-To-Live (存続時間)、Least-Recently-Used (最長未使用時間)、および Least-Frequently-Used (最も使用頻度の少ない) という 3 つの組み込みポリシーがあります。メモリー・ベースの除去オプションを使用可能にすると、メモリーが制約状態になるので、3 つのポリシーではすべて、必要であればデータをより積極的に除去することができます。

イベント・ベースの無効化

スパース・キャッシュおよび完全キャッシュは、Java Message Service (JMS) などのイベント生成プログラムを使用して無効化または更新することができます。JMS を使用した無効化は、データベース・トリガーを使用してバックエンドを更新するどのプロセスにも手動で関連付けることができます。サーバー・キャッシュで変更があった場合にクライアントに通知できる JMS ObjectGridEventListener プラグインが eXtreme Scale で提供されています。これにより、クライアントが失効データを表示する時間を短縮できます。

プログラマチックな無効化

eXtreme Scale API により、 Session.beginNoWriteThrough()、ObjectMap.invalidate()、および EntityManager.invalidate() API メソッドを使用したニア・キャッシュおよびサーバー・キャッシュの手動対話が可能になります。クライアントまたはサーバーのプロセスでデータの一部がもう必要ない場合、無効化メソッドを使用して、ニア・キャッシュまたはサーバー・キャッシュからデータを除去できます。beginNoWriteThrough メソッドは、ローダーを呼び出すことなく、ObjectMap または EntityManager 操作をローカル・キャッシュに適用します。クライアントから呼び出された場合のこの操作は、ニア・キャッシュのみに適用されます (リモート・ローダーは呼び出されません)。サーバーで呼び出された場合のこの操作は、ローダーを呼び出すことなく、サーバー・コア・キャッシュのみに適用されます。

データの無効化

Scale キャッシュ・データを削除するには、イベント・ベースの無効化メカニズムまたはプログラマチックな無効化メカニズムが使用できます。

イベント・ベースの無効化

スパース・キャッシュおよび完全キャッシュは、Java Message Service (JMS) などのイベント生成プログラムを使用して無効化または更新することができます。JMS を使用した無効化は、データベース・トリガーを使用してバックエンドを更新するどのプロセスにも手動で関連付けることができます。サーバー・キャッシュが変更した場合にクライアントに通知できる JMS ObjectGridEventListener プラグインが eXtreme Scale で提供されています。 この通知タイプによって、クライアントが失効データを表示する時間を短縮します。

イベント・ベースの無効化は、一般的には以下の 3 つのコンポーネントで構成されます。

- **イベント・キュー:** イベント・キューには、データ変更イベントが保管されます。 データ変更イベントを管理できるのであれば、イベント・キューは JMS キュー、データベース、メモリー内の FIFO キュー、またはすべての種類のマニフェストの可能性があります。
- **イベント・パブリッシャー:** イベント・パブリッシャーは、データ変更イベントをイベント・キューにパブリッシュします。 イベント・パブリッシャーは、通常、作成されたアプリケーションまたは eXtreme Scale プラグインの実装です。イベント・パブリッシャーは、いつデータが変更されたかを知っています。あるいはイベント・パブリッシャーがデータ自体を変更します。トランザクションがコミットすると、変更されたデータに対してイベントが生成され、イベント・パブリッシャーはこれらのイベントをイベント・キューにパブリッシュします。
- **イベント・コンシューマー:** イベント・コンシューマーは、データ変更イベントをコンシュームします。 イベント・コンシューマーは、通常アプリケーションで、ターゲット・グリッド・データが他のグリッドからの最新の変更を使用して更新されることを確認します。 このイベント・コンシューマーは、イベント・キューと対話をして最新のデータ変更を取得し、ターゲット・グリッドのデータ変更を適用します。 イベント・コンシューマーは eXtreme Scale API を使用して、失効データを無効にしたり、グリッドを最新データで更新することができます。

例えば、JMSObjectGridEventListener にはクライアント/サーバー・モデルのオプションがあり、そのイベント・キューは指定された JMS 宛先です。 すべてのサーバー・プロセスがイベント・パブリッシャーです。 トランザクションがコミットすると、サーバーはデータ変更を取得し、それを指定された JMS 宛先にパブリッシュします。 すべてのクライアント・プロセスがイベント・コンシューマーです。 指定された JMS 宛先からデータ変更を受信し、その変更をクライアントのニア・キャッシュに適用します。

詳しくは、「*管理ガイド*」でクライアント無効化メカニズムの使用可能化に関するトピックを参照してください。

プログラマチックな無効化

WebSphere eXtreme Scale API により、 Session.beginNoWriteThrough() 、ObjectMap.invalidate()、および EntityManager.invalidate() API メソッドを使用したニア・キャッシュおよびサーバー・キャッシュの手動対話が可能になります。クライアントまたはサーバーのプロセスでデータの一部がもう必要ない場合、無効化メソッドを使用して、ニア・キャッシュまたはサーバー・キャッシュからデータを除去できます。beginNoWriteThrough メソッドは、ローダーを呼び出すことなく、ObjectMap または EntityManager 操作をローカル・キャッシュに適用します。クライアントから呼び出された場合のこの操作は、ニア・キャッシュのみに適用されます (リモート・ローダーは呼び出されません)。サーバーで呼び出された場合のこの操作は、ローダーを呼び出すことなく、サーバー・コア・キャッシュのみに適用されます。

他の手法と一緒にプログラマチックな無効化を使用して、データをいつ無効にするかを決定します。 例えば、この無効化メソッドは、イベント・ベースの無効化メカニズムを使用してデータ変更イベントを受信し、API を使用して失効データを無効にします。

索引付け

MapIndexPlugin プラグインは、BackingMap 上にいくつかの索引を作成して、非キー・データ・アクセスをサポートするために使用します。

索引のタイプおよび構成

索引付けフィーチャーは、MapIndexPlugin プラグインと表されるか、または略して Index で表されます。Index は BackingMap プラグインです。BackingMap では、各索引プラグインが索引構成規則に従っている限り、複数の索引プラグインを構成できます。

索引付けフィーチャーは、1 つ以上の索引を BackingMap に作成する場合に使用できます。1 つの索引は、BackingMap 内の 1 つのオブジェクトの 1 つの属性または属性のリストから作成されます。このフィーチャーにより、アプリケーションはより迅速に特定のオブジェクトを見つけることができます。索引付けフィーチャーを使用すると、アプリケーションは特定の値を持つオブジェクトや、ある範囲の索引属性値内にあるオブジェクトを見つけることができます。

可能な索引付けには、静的および動的という 2 つのタイプがあります。静的索引付けの場合、ObjectGrid インスタンスを初期化する前に、BackingMap に索引プラグインを構成する必要があります。この構成を行うには、BackingMap を XML で構成するか、またはプログラマチックに構成します。静的索引付けでは、まず最初に、ObjectGrid の初期化中に索引を作成します。索引は常に BackingMap に同期しており、いつでも使用できる準備ができています。静的索引付けプロセスが既に開始している場合、索引は、eXtreme Scale トランザクション管理プロセスの一環として保守されます。トランザクションが変更をコミットすると、それらの変更は静的索引も更新し、トランザクションがロールバックされれば索引の変更もロールバックされます。

動的索引付けの場合は、索引を含む ObjectGrid インスタンスの初期化の前または後に、BackingMap に索引を作成することができます。動的索引付けプロセスのライフ

サイクルはアプリケーションによって制御されるので、不要になったら動的索引を削除することができます。アプリケーションが動的索引を作成する場合は、索引作成プロセスを完了するまでに時間がかかるために、その索引をすぐに使用できないことがあります。この時間は索引付けされるデータの量に依存するので、特定の索引付けイベントが発生したときにそのことを通知してもらいたいアプリケーションのために、DynamicIndexCallback インターフェースが提供されています。これらのイベントには、準備完了、エラー、および破棄があります。アプリケーションは、このコールバック・インターフェースを実装し、動的索引付けプロセスに登録できます。

BackingMap に索引プラグインが構成されている場合、対応する ObjectMap からアプリケーション索引プロキシー・オブジェクトを取得することができます。ObjectMap の getIndex メソッドを呼び出し、索引プラグインの名前を渡すと、索引プロキシー・オブジェクトが戻されます。索引プロキシー・オブジェクトを適切なアプリケーション索引インターフェース (MapIndex、MapRangeIndex、またはカスタマイズされた索引インターフェースなど) にキャストする必要があります。索引プロキシー・オブジェクトを取得したら、アプリケーション索引インターフェースで定義されたメソッドを使用して、キャッシュされたオブジェクトを検出することができます。

次のリストに、索引付けの使用手順をまとめます。

- 静的または動的索引プラグインを BackingMap に追加します。
- ObjectMap の getIndex メソッドを発行して、アプリケーション索引プロキシー・オブジェクトを取得します。
- MapIndex、MapRangeIndex またはカスタマイズされた索引インターフェースなどの適切なアプリケーション索引インターフェースに、索引プロキシー・オブジェクトをキャストします。
- アプリケーション索引インターフェースで定義されたメソッドを使用して、キャッシュされたオブジェクトを検出します。

HashIndex クラスは、組み込みアプリケーション索引インターフェースである MapIndex と MapRangeIndex の両方をサポートすることのできる組み込み索引プラグイン実装です。ユーザー独自の索引を作成することもできます。HashIndex を静的索引または動的索引として BackingMap に追加して、MapIndex または MapRangeIndex の索引プロキシー・オブジェクトを取得し、その索引プロキシー・オブジェクトを使用してキャッシュ・オブジェクトを検索することができます。

デフォルトの索引

ローカル・マップ内のキーを反復処理する場合は、デフォルトの索引を使用できます。この索引はまったく構成を必要としませんが、エージェントを使用するか ShardEvents.shardActivated(ObjectGrid shard) メソッドから取得した ObjectGrid インスタンスを使用して、断片に対して使用しなければなりません。

データ品質に関する考慮事項

索引照会メソッドの結果が表わすのは、特定の時刻におけるデータのスナップショットのみです。結果がアプリケーションに戻された後には、データ・エントリーに対するロックは取得されません。アプリケーションは、戻されたデータ・セットに

対してデータ更新が発生する可能性があることに注意する必要があります。例えば、アプリケーションは MapIndex の findAll メソッドを実行して、キャッシュされたオブジェクトのキーを取得します。戻されたこのキー・オブジェクトは、キャッシュ内のデータ項目に関連付けられています。アプリケーションは、キー・オブジェクトを提供することにより、ObjectMap に対して get メソッドを実行して、オブジェクトを検出できるようになっている必要があります。get メソッドが呼び出される直前に、別のトランザクションがキャッシュからそのデータ・オブジェクトを削除した場合、戻される結果はヌルです。

索引付けのパフォーマンスに関する考慮事項

索引付けフィーチャーの主な目的の 1 つは、BackingMap の全体的なパフォーマンスを改善することです。索引付けの使い方が不適切な場合は、アプリケーションのパフォーマンスが低下する可能性があります。このフィーチャーを使用する前に、次の要因について検討します。

- **並行書き込みトランザクションの数:** 索引処理は、トランザクションが BackingMap にデータを書き込むたびに起こりえます。アプリケーションが索引照会操作を試行しているときに、多くのトランザクションがデータをマップに書き込んでいると、パフォーマンスが低下します。
- **照会操作で戻される結果セットのサイズ:** 結果セットのサイズが大きくなるにつれて、照会のパフォーマンスは低下します。結果セットのサイズが BackingMap の 15% 以上になるとパフォーマンスは低下する傾向にあります。
- **同じ BackingMap に作成される索引の数:** 各索引がシステム・リソースを消費します。BackingMap に作成される索引の数が増えると、パフォーマンスは低下します。

索引付け機能は、BackingMap パフォーマンスを大幅に改善できることがあります。理想的なケースは、BackingMap の大部分の操作が読み取りであり、照会の結果セットが BackingMap エントリーのわずかな割合に過ぎず、ごく少数の索引が BackingMap に対して作成される場合です。

JPA ローダー

Java Persistence API (JPA) は、 Java オブジェクトをリレーショナル・データベースにマップするための仕様です。JPA には、Java 言語メタデータ・アノテーション、XML 記述子、またはその両方を使用して、Java オブジェクトとリレーショナル・データベースとの間のマッピングを定義するための、完全なオブジェクト・リレーショナル・マッピング (ORM) 仕様が含まれています。オープン・ソースおよび商用の実装がいくつか使用できます。

eXtreme Scale と一緒に Java Persistence API (JPA) Loader プラグイン実装を使用すると、選択されたローダーがサポートする任意のデータベースと対話することができます。JPA を使用するには、サポートされる JPA プロバイダー (OpenJPA や Hibernate など)、JAR ファイル、および `META-INF/persistence.xml` ファイルがクラスパスになければなりません。

JPALoader の com.ibm.websphere.objectgrid.jpa.JPALoader および JPAEntityLoader com.ibm.websphere.objectgrid.jpa.JPAEntityLoader プラグインは、ObjectGrid マップとデータベースを同期するために使用される 2 つの組み込み JPA Loader プラグイン

です。この機能を使用するには、Hibernate または OpenJPA などの JPA 実装がなくてはなりません。データベースは、選択された JPA プロバイダーがサポートする任意のバックエンドを使用できます。

ObjectMap API を使用してデータを保管する場合、JPALoader プラグインを使用することができます。EntityManager API を使用してデータを保管する場合、JPAEntityLoader プラグインを使用します。

JPA ローダー・アーキテクチャー

JPA ローダー は、Plain Old Java Object (POJO) を保管する eXtreme Scale マップに使用されます。

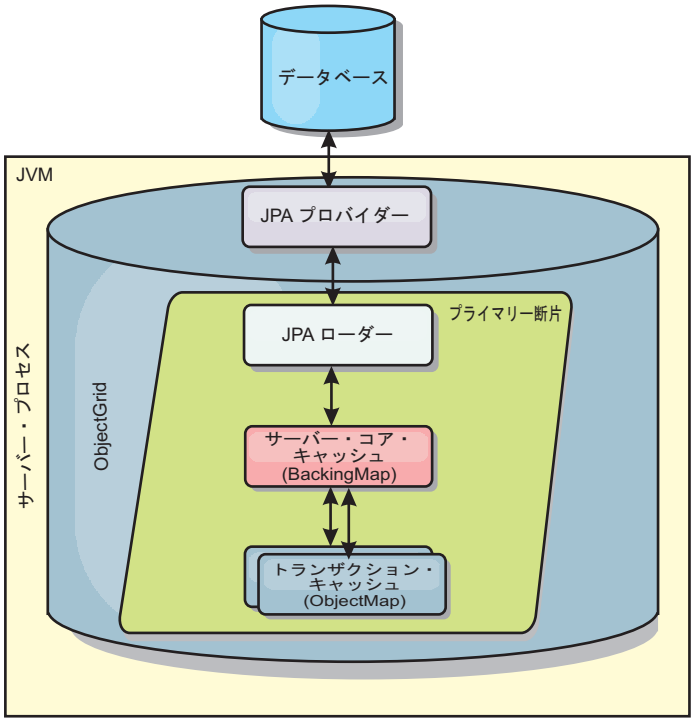


図 29. JPA ローダー・アーキテクチャー

ObjectMap.get(Object key) メソッドが呼び出されると、eXtreme Scale ランタイムが、まず ObjectMap 層にエントリーがあるかどうかをチェックします。ない場合、ランタイムは、要求を JPA Loader に委任します。キーのロード要求時に、JPALoader は JPA EntityManager.find(Object key) メソッドを呼び出して、JPA 層からのデータを検索します。データが JPA エンティティー・マネージャーに含まれている場合、そのデータが返されます。含まれていない場合は、JPA プロバイダーがデータベースと対話して値を取得します。

例えば、ObjectMap.update(Object key, Object value) メソッドを使用して ObjectMap に対する更新が行われると、eXtreme Scale ランタイムは、この更新に対する LogElement を作成し、これを JPALoader に送ります。JPALoader は、JPA EntityManager.merge(Object value) メソッドを呼び出して、データベースに対する値を更新します。

JPAEntityLoader の場合も、同じ 4 つの層が含まれます。ただし、JPAEntityLoader プラグインは、eXtreme Scale エンティティーを保管するマップに使用されるため、エンティティー間の関係が使用シナリオを複雑にする可能性があります。eXtreme Scale エンティティーは、JPA エンティティーとは区別されます。詳しくは、*プログラミング・ガイド*の JPAEntityLoader プラグインに関する説明を参照してください。

メソッド

ローダーでは、3 つの主要なメソッドを提供しています。

1. get: JPA を使用してデータを取得することにより、渡されたキーのリストに対応する値のリストを返します。このメソッドは、JPA を使用して、データベース内のエンティティーを検出します。JPALoader プラグインの場合、返されるリストには、find 操作から直接得られた JPA エンティティーのリストが含まれます。JPAEntityLoader プラグインの場合、返されるリストには、JPA エンティティーから変換された eXtreme Scale エンティティー値タプルが含まれます。
2. batchUpdate: ObjectGrid マップのデータをデータベースに書き込みます。異なる操作タイプ (挿入、更新、削除) に応じて、ローダーは、JPA パーシスト、マージ、および除去操作を使用してデータベースに対するデータを更新します。JPALoader の場合、マップ内のオブジェクトが JPA エンティティーとして直接使用されます。JPAEntityLoader の場合、マップ内のエンティティー・タプルが、JPA エンティティーとして使用されるオブジェクトに変換されます。
3. preloadMap: ClientLoader.load クライアント・ローダー・メソッドを使用してマップをプリロードします。区画化マップの場合、preloadMap メソッドは 1 つの区画でのみ呼び出されます。区画は、JPALoader または JPAEntityLoader クラスの preloadPartition プロパティーに指定します。 preloadPartition 値がゼロより小さく設定されているか、*total_number_of_partitions* - 1) より大きく設定されている場合、プリロードは使用不可になります。

JPALoader と JPAEntityLoader のいずれのプラグインも、JPATxCallback クラスで動作し、eXtreme Scale トランザクションと JPA トランザクションを調整します。これら 2 つのローダーを使用するには、JPATxCallback を ObjectGrid インスタンス内に構成する必要があります。

構成およびプログラミング

JPA ローダーをマルチマスター環境で使用する場合は、182 ページの『マルチマスター・トポロジーでのローダーについての考慮事項』を参照してください。JPA ローダーの構成について詳しくは、「*管理ガイド*」で JPA ローダーに関する説明を参照してください。JPA ローダーのプログラミングについて詳しくは、*プログラミング・ガイド*を参照してください。

シリアライゼーションの概要

データは、データ・グリッドで Java オブジェクトとして常に表されていますが、必ずしも保管されているとは限りません。WebSphere eXtreme Scale は、クライアント・プロセスとサーバー・プロセスの間でのデータ移動のために、複数の Java プロセスを使用して、Java オブジェクト・インスタンスをバイトに変換し、必要に応じて再度オブジェクトに戻すことによって、データをシリアライズします。

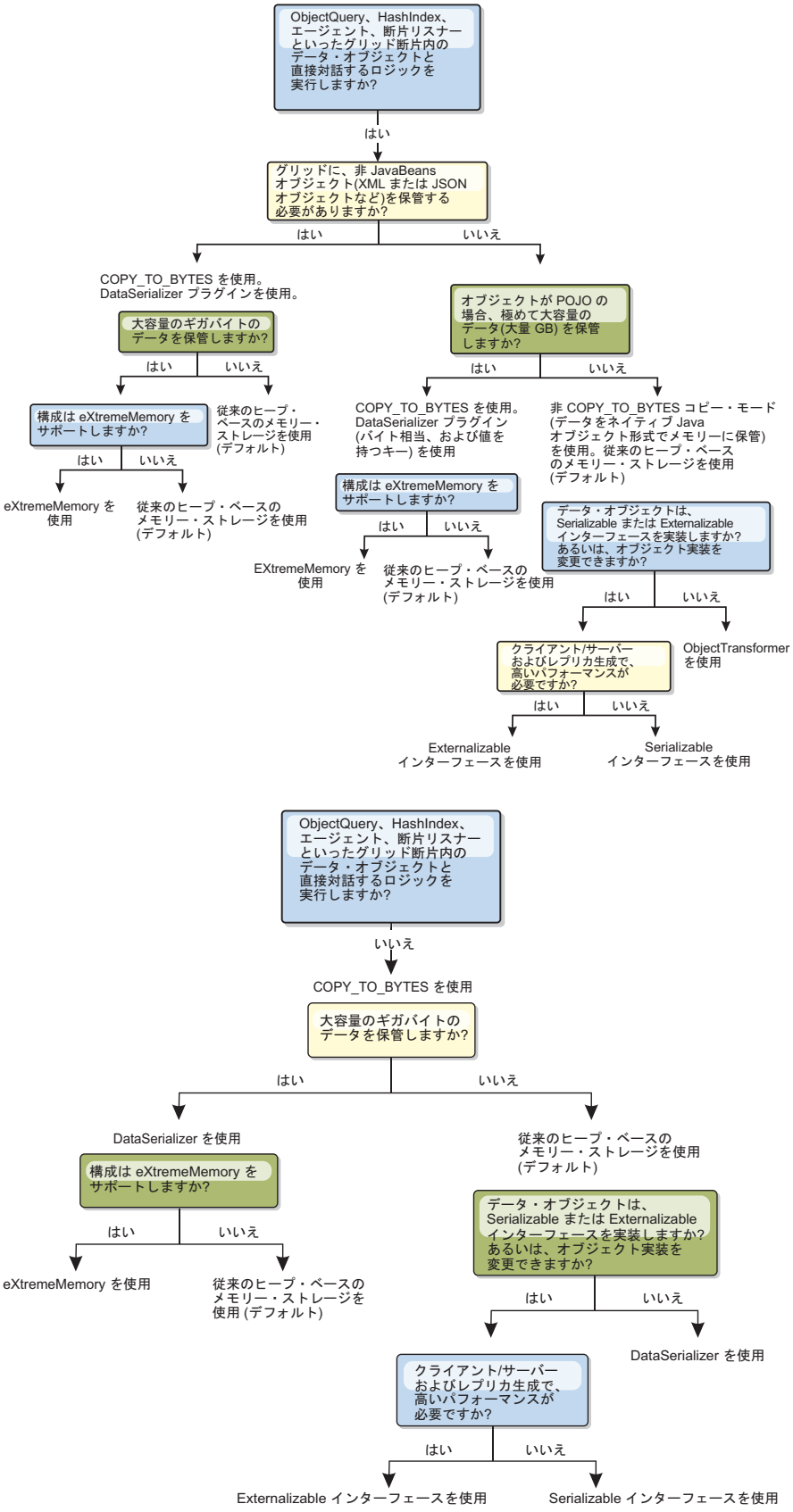
データは、シリアライズされます。すなわち、以下のシチュエーションでネットワークを利用して伝送するために、データはデータ・ストリームに変換されます。

- クライアントがサーバーと通信し、それらのサーバーがクライアントに情報を送り返す場合
- あるサーバーから別のサーバーにサーバーがデータを複製する場合

代わりに、WebSphere eXtreme Scale を介したシリアライゼーション・プロセスはやめて、生データをバイト配列として保管することにしてもかまいません。バイト配列を使用すれば、メモリーへの保管にかかるコストはずっと少なくてすみます。これは、Java Virtual Machine (JVM) がガーベッジ・コレクション中に検索するオブジェクトが少なく、必要なときだけデシリアライズできるためです。照会または索引を使用してオブジェクトにアクセスする必要がない場合にのみ、バイト配列を使用します。データはバイトとして保管されるため、eXtreme Scale には照会のための属性を記述するメタデータはありません。

eXtreme Scale でデータをシリアライズするために、Java シリアライゼーション、ObjectTransformer プラグイン、または DataSerializer プラグインを使用できます。これらのオプションのいずれかを使用してシリアライゼーションを最適化するには、COPY_TO_BYTES モードを使用します。すると、トランザクションのコミット時にデータがシリアライズされ、シリアライゼーションが 1 回のみ行われるようになるため、パフォーマンスを最大 70 % 高めることができます。シリアライズされたデータは、クライアントからサーバーに変更なしで送信されるか、サーバーから複製されたサーバーに送信されます。COPY_TO_BYTES モードを使用すると、大きなオブジェクト・グラフが消費するメモリー占有スペースを削減できます。

次の図を参考にして、開発ニーズに最適なシリアライゼーション・メソッドの種類を決定してください。最初の図は、グリッド断片の中で直接データ・オブジェクトと対話するロジックを実行している場合に使用可能なシリアライゼーション・メソッドを説明しています。最後の図は、グリッド断片と直接に対話していない場合に使用可能なオプションを表示しています。



eXtreme Scale 製品でサポートされているシリアライゼーションの形式について詳しくは、以下のトピックを参照してください。

Java を使用したシリアライゼーション

Java シリアライゼーションとは、Serializable インターフェースを使用するデフォルトのシリアライゼーションか、または、Serializable インターフェースと Externalizable インターフェースの両方を使用するカスタム・シリアライゼーションのいずれかを指します。

デフォルトのシリアライゼーション

デフォルトのシリアライゼーションを使用するには、java.io.Serializable インターフェースを実装します。このインターフェースには、オブジェクトをバイトに変換する API が含まれています。バイトに変換されたオブジェクトは後でデシリアライズされます。 java.io.ObjectOutputStream クラスを使用して、オブジェクトを永続化します。次に、ObjectOutputStream.writeObject() メソッドを呼び出して、シリアライゼーションを開始し、Java オブジェクトをフラット化します。

カスタムのシリアライゼーション

一部のケースでは、オブジェクトを変更して、カスタム・シリアライゼーションを使用するようにする必要があります (例えば、java.io.Externalizable インターフェースを実装する、または java.io.Serializable インターフェースを実装しているクラスの writeObject および readObject メソッドを実装するなど)。 ObjectGrid API または EntityManager API のメソッド以外のメカニズムを使用してオブジェクトをシリアライゼーションするときは、カスタムのシリアライズした技法を採用する必要があります。

例えば、オブジェクトまたはエンティティーがインスタンス・データとして DataGrid API エージェント内に保管されるとき、またはエージェントがオブジェクトやエンティティーを返すとき、それらのオブジェクトは ObjectTransformer を使用して変換されません。ただし、`EntityMixin` インターフェースが使用されている場合、エージェントは、自動的に ObjectTransformer を使用します。詳しくは、『DataGrid エージェントとエンティティー・ベースのマップ』を参照してください。

ObjectTransformer プラグイン

ObjectTransformer プラグインを使用すると、パフォーマンス向上のために、キャッシュ内のオブジェクトをシリアライズ、デシリアライズ、およびコピーすることができます。

 ObjectTransformer インターフェースは、DataSerializer プラグインで置換されました。これを使用して、既存の製品 API がデータと効率的に対話できるように WebSphere eXtreme Scale 内の任意のデータを効率的に格納できます。

プロセッサーの使用に関するパフォーマンス上の問題がある場合は、各マップに ObjectTransformer プラグインを追加します。 ObjectTransformer プラグインを使用しない場合、合計プロセッサー時間の 60 から 70 パーセントまではエントリーのシリアライズとコピーに費やされます。

目的

ObjectTransformer プラグインがあれば、アプリケーションで以下の操作に対するカスタム・メソッドを提供できます。

- エントリーに対するキーのシリアライズまたはデシリアライズ
- エントリーに対する値のシリアライズまたはデシリアライズ
- エントリーに対するキーまたは値のコピー

ObjectTransformer プラグインが提供されない場合、 ObjectGrid はシリアライズおよびデシリアライズのシーケンスを使用してオブジェクトをコピーするので、ユーザーがキーと値のシリアライズを行う必要があります。この方法には費用がかかるので、パフォーマンスが重大である場合には ObjectTransformer プラグインを使用してください。アプリケーションが、トランザクションのオブジェクトを最初に検索する際に、コピーが行われます。このコピーは、マップのコピー・モードを NO_COPY に設定すると行われません。あるいは、コピー・モードを COPY_ON_READ に設定すると、コピー数を軽減できます。アプリケーションの必要に応じて、このプラグインにカスタム・コピー・メソッドを提供することによって、コピー操作を最適化します。このようなプラグインにより、コピー・オーバーヘッドを合計プロセッサー時間の 65-70 パラメーターから 2/3 パーセントに軽減できます。

デフォルトの copyKey および copyValue メソッド実装では、最初に clone メソッド (このメソッドが提供されている場合) を使用しようとします。 clone メソッド実装が提供されていない場合は、実装のデフォルトはシリアライゼーションになります。

eXtreme Scale が分散モードで実行されているときは、オブジェクト・シリアライゼーションも直接使用されます。 LogSequence は、変更内容を ObjectGrid のピアに送信する前に、ObjectTransformer プラグインを使用して、キーおよび値のシリアライズを支援します。組み込み Java Developer Kit シリアライゼーションを使用するのではなく、シリアライゼーションのカスタム・メソッドを提供するときは、注意が必要です。オブジェクトのバージョン管理は複雑な問題であり、カスタム・メソッドがバージョン管理用に設計されていることが確認できない場合、バージョンの互換性に問題が発生することがあります。

以下のリストでは、eXtreme Scale がキーと値の両方のシリアライズを試みる方法を説明しています。

- カスタム ObjectTransformer プラグインが作成され、プラグインされている場合、eXtreme Scale は ObjectTransformer インターフェース内のメソッドを呼び出して、キーと値をシリアライズし、オブジェクトのキーおよび値のコピーを取得します。
- カスタム ObjectTransformer プラグインが使用されていない場合、eXtreme Scale はデフォルトに従って値のシリアライズとデシリアライズを行います。デフォルト・プラグインが使用されている場合、各オブジェクトは、外部化可能またはシリアライズ可能として実装されます。
  - オブジェクトが Externalizable インターフェースをサポートする場合、writeExternal メソッドが呼び出されます。外部化可能として実装されたオブジェクトは、パフォーマンスを向上させます。

– Externalizablee インターフェースをサポートせず、Serializable インターフェースを実装しないオブジェクトは、ObjectOutputStream メソッドを使用して保存されます。

ObjectTransformer インターフェースの使用

ObjectTransformer は、ObjectTransformer インターフェースを実装し、共通 ObjectGrid プラグイン規則に準拠している必要があります。

ObjectTransformer オブジェクトを BackingMap 構成に追加する場合、以下のように、プログラマチック構成と XML 構成の 2 つの方法が使用されます。

ObjectTransformer オブジェクトのプログラマチックなプラグイン

以下のコード・スニペットは、カスタム ObjectTransformer オブジェクトを作成し、それを BackingMap に追加します。

```
ObjectGridManager objectGridManager = ObjectGridManagerFactory.getObjectGridManager();
ObjectGrid myGrid = objectGridManager.createObjectGrid("myGrid", false);
BackingMap backingMap = myGrid.getMap("myMap");
MyObjectTransformer myObjectTransformer = new MyObjectTransformer();
backingMap.setObjectTransformer(myObjectTransformer);
```

ObjectTransformer をプラグインするための XML 構成方法

ObjectTransformer 実装のクラス名が、com.company.org.MyObjectTransformer クラスであると仮定します。このクラスは、ObjectTransformer インターフェースを実装します。ObjectTransformer 実装は、以下の XML を使用して構成することができます。

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

    <backingMapPluginCollections>
 <backingMapPluginCollection id="myMap">
      <bean id="ObjectTransformer" className="com.company.org.MyObjectTransformer" />
 </backingMapPluginCollection>
    </backingMapPluginCollections>
</objectGridConfig>
```

ObjectTransformer の使用に関するシナリオ

ObjectTransformer プラグインは、以下の状態で使用できます。

- シリアライズ不能オブジェクト
- シリアライズ可能オブジェクトであるが、シリアライゼーション・パフォーマンスを改善する
- キーまたは値のコピー

以下の例で、ObjectGrid は Stock クラスのストアに使用されます。

```
/**
* Stock object for ObjectGrid demo
*
*
*/
public class Stock implements Cloneable {
```

```
String ticket;
double price;
String company;
String description;
int serialNumber;
long lastTransactionTime;
/**
* @return Returns the description.
*/
public String getDescription() {
    return description;
}
/**
* @param description The description to set.
*/
public void setDescription(String description) {
    this.description = description;
}
/**
* @return Returns the lastTransactionTime.
*/
public long getLastTransactionTime() {
    return lastTransactionTime;
}
/**
* @param lastTransactionTime The lastTransactionTime to set.
*/
public void setLastTransactionTime(long lastTransactionTime) {
    this.lastTransactionTime = lastTransactionTime;
}
/**
* @return Returns the price.
*/
public double getPrice() {
    return price;
}
/**
* @param price The price to set.
*/
public void setPrice(double price) {
    this.price = price;
}
/**
* @return Returns the serialNumber.
*/
public int getSerialNumber() {
    return serialNumber;
}
/**
* @param serialNumber The serialNumber to set.
*/
public void setSerialNumber(int serialNumber) {
    this.serialNumber = serialNumber;
}
/**
* @return Returns the ticket.
*/
public String getTicket() {
    return ticket;
}
/**
* @param ticket The ticket to set.
*/
public void setTicket(String ticket) {
    this.ticket = ticket;
}
/**
* @return Returns the company.
*/
public String getCompany() {
    return company;
}
/**
* @param company The company to set.
*/
public void setCompany(String company) {
    this.company = company;
}
//clone
public Object clone() throws CloneNotSupportedException
```

```
    {
        return super.clone();
    }
}
```

Stock クラス用に、カスタム・オブジェクト変換プログラム・クラスを作成できます。

```
/**
* Custom implementation of ObjectGrid ObjectTransformer for stock object
*
*/
public class MyStockObjectTransformer implements ObjectTransformer {
/* (non-Javadoc)
* @see
* com.ibm.websphere.objectgrid.plugins.ObjectTransformer#serializeKey
* (java.lang.Object,
* java.io.ObjectOutputStream)
*/
public void serializeKey(Object key, ObjectOutputStream stream) throws IOException {
    String ticket= (String) key;
    stream.writeUTF(ticket);
}

/* (non-Javadoc)
* @see com.ibm.websphere.objectgrid.plugins.
ObjectTransformer#serializeValue(java.lang.Object,
java.io.ObjectOutputStream)
*/
public void serializeValue(Object value, ObjectOutputStream stream) throws IOException {
    Stock stock= (Stock) value;
    stream.writeUTF(stock.getTicket());
    stream.writeUTF(stock.getCompany());
    stream.writeUTF(stock.getDescription());
    stream.writeDouble(stock.getPrice());
    stream.writeLong(stock.getLastTransactionTime());
    stream.writeInt(stock.getSerialNumber());
}

/* (non-Javadoc)
* @see com.ibm.websphere.objectgrid.plugins.
ObjectTransformer#inflateKey(java.io.ObjectInputStream)
*/
public Object inflateKey(ObjectInputStream stream) throws IOException, ClassNotFoundException {
    String ticket=stream.readUTF();
    return ticket;
}

/* (non-Javadoc)
* @see com.ibm.websphere.objectgrid.plugins.
ObjectTransformer#inflateValue(java.io.ObjectInputStream)
*/

public Object inflateValue(ObjectInputStream stream) throws IOException, ClassNotFoundException {
    Stock stock=new Stock();
    stock.setTicket(stream.readUTF());
    stock.setCompany(stream.readUTF());
    stock.setDescription(stream.readUTF());
    stock.setPrice(stream.readDouble());
    stock.setLastTransactionTime(stream.readLong());
    stock.setSerialNumber(stream.readInt());
    return stock;
}

/* (non-Javadoc)
* @see com.ibm.websphere.objectgrid.plugins.
ObjectTransformer#copyValue(java.lang.Object)
*/
public Object copyValue(Object value) {
    Stock stock = (Stock) value;
    try {
        return stock.clone();
    }
    catch (CloneNotSupportedException e)
    {
        // display exception message    }
}

/* (non-Javadoc)
* @see com.ibm.websphere.objectgrid.plugins.
ObjectTransformer#copyKey(java.lang.Object)
*/
public Object copyKey(Object key) {
    String ticket=(String) key;
    String ticketCopy= new String (ticket);
    return ticketCopy;
}
}
```

次に、このカスタム MyStockObjectTransformer クラスを BackingMap にプラグインします。

```
ObjectGridManager ogf=ObjectGridManagerFactory.getObjectGridManager();
ObjectGrid og = ogf.getObjectGrid("NYSE");
BackingMap bm = og.defineMap("NYSEStocks");
MyStockObjectTransformer ot = new MyStockObjectTransformer();
bm.setObjectTransformer(ot);
```

DataSerializer プラグインを使用したシリアライゼーション

既存の製品 API がデータと効率的に対話できるように、DataSerializer プラグインを使用して WebSphere eXtreme Scale 内の任意のデータを効率的に格納します。

Java シリアライゼーションや ObjectTransformer プラグインなどのシリアライゼーション・メソッドを使用して、ネットワーク上でデータをマーシャルできます。また、COPY_TO_BYTES コピー・モードでこれらのシリアライゼーション・オプションを使用すると、クライアントとサーバーとの間でのデータの移動コストが削減され、パフォーマンスが向上します。ただし、これらのオプションは、生じる可能性のある以下の問題を解決しません。

- キーがバイトで保管されておらず、Java オブジェクトのままである。
- サーバー・サイド・コードがまだオブジェクトをインフレートする必要がある。例えば、照会および索引がまだリフレクションを使用しており、オブジェクトをインフレートする必要がある。さらに、エージェント、リスナー、およびプラグインは、まだオブジェクト形式が必要である。
- クラスがまだサーバー・クラスパスにある必要がある。
- データがまだ Java シリアライゼーション形式 (ObjectOutputStream) である。

DataSerializer プラグインを使用すると、これらの問題を解決する効率的な方法が導入されます。とりわけ、DataSerializer プラグインは、WebSphere eXtreme Scale が特定のオブジェクト形式なしでバイト配列を調べることができるように、シリアライゼーション形式、つまり、バイト配列を記述する手段をこの製品に提供します。public DataSerializer プラグイン・クラスおよびインターフェースは、パッケージ com.ibm.websphere.objectgrid.plugins.io 内にあります。詳しくは、を参照してください。

重要: EntityManager API を使用する場合、エンティティー Java オブジェクトは、直接、BackingMaps に保管されません。EntityManager API はエンティティー・オブジェクトを Tuple オブジェクトに変換します。エンティティー・マップは、高度に最適化された ObjectTransformer と自動的に関連付けられます。ObjectMap API または EntityManager API を使用してエンティティー・マップと対話する際、必ず ObjectTransformer エンティティーが起動されます。したがって、エンティティーを使用している場合、製品が自動的にこのプロセスを完了するため、シリアライゼーションの作業が必要ありません。

スケーラビリティーの概要

WebSphere eXtreme Scale は、区画に分割されたデータの使用を通じて拡張可能です。そして各コンテナーは互いに独立しているため、必要ならば何千というコンテナーに拡張できます。

WebSphere eXtreme Scale は、データ・セットを、プロセス間で (実行時には物理サーバー間でさえ) 移動できる異なる区画に分割します。例えば、4 つのサーバーのデプロイメントから始め、その後キャッシュに対する要求が増えるに従って 10 個のサーバーのデプロイメントに拡張することができます。ちょうど、垂直スケーラビリティーに備えて物理サーバーや処理装置をさらに追加できるように、柔軟性のあるスケーリング能力を区画化によって水平方向に拡張することができます。水平スケーリングは、メモリー内のデータベースに対して WebSphere eXtreme Scale を使用することの大きな利点です。メモリー内のデータベースは、垂直方向にしか拡張することができません。

さらに、WebSphere eXtreme Scale により、一連の API を使用して、区画化され、分散されたデータにトランザクション・アクセスをすることもできます。パフォーマンスの観点から、キャッシュとの対話のために行う選択は、キャッシュを可用性の面で管理する機能と同じくらい重要です。

注: スケーラビリティーは、コンテナー同士が互いに通信しているときは使用不可です。可用性管理、つまりコア・グループ化のプロトコルは、$O(N^2)$ ハートビートおよびビュー保守アルゴリズムですが、コア・グループ・メンバーの数を 20 より下に維持することにより、このプロトコルの負担は軽減されます。レプリカ生成は断片間でのみ対等です。

分散クライアント

WebSphere eXtreme Scale クライアント・プロトコルは、非常に多くのクライアントをサポートします。接続を複数のコンテナー間に広げることができるため、すべてのクライアントは常にすべての区画の対象となるわけではないと仮定すれば、区画化ストラテジーは役に立ちます。クライアントは区画に直接接続されるため、待ち時間は 1 つの転送接続に制限されます。

データ・グリッド、区画、および断片

データ・グリッドは、複数の区画に分割されます。各区画は、データの排他的なサブセットを保持します。1 つの区画は、1 つ以上の断片 (プライマリー断片とレプリカ断片) を含んでいます。レプリカ断片は区画で必須ではありませんが、使用すると、高可用性を提供できます。デプロイメントが独立したメモリー内のデータ・グリッドであれ、メモリー内のデータベース処理スペースであれ、eXtreme Scale でのデータ・アクセスは、大きく断片に依存しています。

1 つの区画のデータは、実行時、断片の集合に保管されます。この断片の集合には、プライマリー断片と、可能性として 1 つ以上のレプリカ断片が含まれます。断片は、eXtreme Scale が Java 仮想マシン に対して追加または除去できる最小の単位です。

配置ストラテジーには、固定区画配置 (デフォルト) とコンテナーごとの配置の 2 つがあります。以下では、固定区画配置ストラテジーの使用に焦点を当てて説明します。

断片の総数

環境にレプリカなしの 1,000,000 のオブジェクトを保持する 10 の区画があるとすると、それぞれ 100,000 のオブジェクトを保管する 10 の断片が存在することにな

ります。このシナリオにレプリカを 1 つ追加すると、各区画には追加の断片が存在することになります。この場合、20 の断片、すなわち、10 のプライマリー断片と 10 のレプリカ断片が存在することになります。これらの断片の 1 つ 1 つには、100,000 のオブジェクトが保管されます。各区画は、1 つのプライマリー断片と 1 つ以上 (N) のレプリカ断片で構成されます。最適な断片数を決定することは、非常に重要です。少数の断片を構成すると、データが断片間に均等に配分されず、メモリー不足エラーやプロセッサーの過負荷問題が生じることになります。各 JVM あたり、少なくとも 10 の断片になるように見積もってください。初めにデータ・グリッドをデプロイするときは、可能性として多数の区画を使用することになります。

JVM あたりの断片数のシナリオ

シナリオ: 少数の JVM あたりの断片数

データは、断片単位を使用して JVM に対して追加および除去されます。断片がさらに小さく分割されることはありません。10 GB のデータがあり、このデータを保持するために 20 の断片が存在しているとすると、各断片には、平均 500 MB のデータが保持されていることになります。9 つの Java 仮想マシンがデータ・グリッドをホストしている場合、各 JVM には、平均して 2 つの断片を持ちます。20 は 9 で割りきれませんから、いくつかの Java 仮想マシン は次の配分のように 3 つの断片を持つことになります。

- 2 つの断片を持つ Java 仮想マシン が 7 つ
- 3 つの断片を持つ Java 仮想マシン が 2 つ

各断片には 500 MB のデータが保持されていますから、データ配分は均等ではありません。2 つの断片を持つ 7 つの Java 仮想マシン は、それぞれ 1 GB のデータをホストすることになります。3 つの断片を持つ 2 つの Java 仮想マシン には、50% 多い 1.5 GB のデータが設定され、メモリー負担がずっと大きくなります。2 つの Java 仮想マシン は、3 つの断片をホスティングしているため、そのデータに対しても 50% 多い要求を受信することになります。したがって、各 JVM あたりに少数の断片数が設定される場合は、不均衡が生じます。パフォーマンスを改善するためには、各 JVM あたりの断片数を大きくします。

シナリオ: JVM あたりの断片数を大きくする

このシナリオでは、断片数をより大きくすることを考えます。このシナリオでは、10 GB のデータをホスティングする 9 つの Java 仮想マシンに、101 の断片があるとします。この場合、各断片には 99 MB のデータが保持されます。Java 仮想マシン には、次のように断片が配分されます。

- 11 の断片を持つ Java 仮想マシン が 7 つ
- 12 の断片を持つ Java 仮想マシン が 2 つ

12 の断片を持つ 2 つの Java 仮想マシン は、99 MB のデータだけ他の断片より多くなりますが、これは 9% の違いに相当します。このシナリオの方が、少数の断片によるシナリオの 50% の違いに比べてはるかに均等に配分されています。プロセッサー使用の観点から見れば、12 の断片を持つ 2 つの Java 仮想マシン には、11 の断片を持つ 7 つの Java 仮想マシン に比べてわずか 9% 多くの作業が割り当

てられることになります。各 JVM 中の断片数を大きくすることにより、データおよびプロセッサー使用が、平等、均等に配分されることになります。

システムを作成する場合、あるいは、システムが計画期間で最大数の Java 仮想マシンを実行している場合、最大サイズのシナリオとして各 JVM あたり 10 の断片を使用するようにしてください。

追加の配置要因

区画数、配置ストラテジー、およびレプリカの数とタイプは、デプロイメント・ポリシーで設定されます。 配置される断片数は、ユーザーが定義するデプロイメント・ポリシーによって決まります。
minSyncReplicas、developmentMode、maxSyncReplicas、および maxAsyncReplicas 属性は、区画とレプリカが配置される場所に影響します。

次の要因が断片の配置可能な時間に影響します。

- **xscmd -c suspendBalancing** コマンドおよび **xscmd -c resumeBalancing** コマンド。
- 7.1.1+ 断片がコンテナー・サーバーに配置される前のミリ秒数を定義する **placementDeferralInterval** プロパティーが含まれた、サーバー・プロパティー・ファイル。
- デプロイメント・ポリシー内の **numInitialContainers** 属性。

初期始動時に最大数のレプリカが配置されない場合、後で追加のサーバーを始動したときに追加のレプリカが配置される場合があります。JVM ごとの断片数を計画する場合、プライマリー断片とレプリカ断片の最大数は、構成されたレプリカの最大数をサポートするのに十分な JVM が開始されているかどうかによって異なります。 レプリカはプライマリーと同じプロセス内に配置するべきではありません。あるプロセスが失われると、プライマリーもレプリカも共に失われます。
developmentMode 属性が `false` に設定されている場合、プライマリーとレプリカは同じ物理サーバーに配置されません。

区画化

区画化を使用して、アプリケーションをスケールアウトします。区画の数は、デプロイメント・ポリシーで定義できます。

区画化について

区画化は、新磁気ディスク制御機構 (RAID) ストライピングと同様のものではありませんが、各インスタンスをすべてのストライプ間でスライスします。各区画により、個別エントリーの完全なデータがホスティングされます。区画化は、非常に有効なスケーリング手段ですが、すべてのアプリケーションに適用できるわけではありません。複数の大規模なデータ集合にわたってトランザクションの保証を必要とするアプリケーションをスケールアウトすることはできず、効果的に区画化することもできません。WebSphere eXtreme Scale は、区画をまたがる 2 フェーズ・コミットを現在のところサポートしていません。

重要: 区画数を選択する際は慎重に行ってください。デプロイメント・ポリシーで定義される区画の数は、アプリケーションが拡張できるコンテナー・サーバーの数

に直接影響を与えます。各区画は、プライマリー断片および構成済みの数のレプリカ断片から構成されます。公式 (Number_Partitions*(1 + Number_Replicas)) は、単一のアプリケーションを拡張するのに使用できるコンテナーの数を表します。

区画の使用

1 つのデータ・グリッドに、最大数千個の区画を作成することができます。データ・グリッドは、区画の数に区画ごとの断片の数を掛けた結果の大きさまで拡張できます。例えば、16 個の区画があり、各区画にはプライマリーとレプリカがそれぞれ 1 つずつ、つまり 2 つの断片があるとすれば、潜在的には 32 個の Java 仮想マシンまで拡張できることになります。この場合、JVM ごとに 1 つの断片が定義されます。使用する可能性が高い Java 仮想マシンの予定数に基づいて、適切な区画数を選択する必要があります。断片が 1 つ増すごとにシステムのためのプロセッサーおよびメモリーの使用量が増加します。システムは、使用可能な Java 仮想マシンの数に応じてこのオーバーヘッドを処理するように拡張される設計になっています。

アプリケーションが 4 つのコンテナー・サーバー Java 仮想マシンのデータ・グリッドで実行される場合は、アプリケーションが数千もの区画を使用しないようにしてください。アプリケーションは、それぞれのコンテナー・サーバー JVM について、適切な数の断片を持つよう構成する必要があります。例えば、2 つの断片を持つ 2000 の区画が 4 つのコンテナー Java 仮想マシンで稼働するという構成は適切ではありません。このような構成では、4 つのコンテナー Java 仮想マシンに 4000 個の断片が配置されるか、またはコンテナー JVM ごとに 1000 個の断片が配置されることになります。

予定される各コンテナー JVM の断片を 10 未満に構成することをお勧めします。それでもなお、この構成では、コンテナー JVM ごとの断片の数を適切に保ちながら、初期構成の 10 倍の柔軟性のある拡張を行える可能性があります。

次のような拡張例を検討してください。現在、6 台の物理サーバーがあり、物理サーバーごとに 2 つのコンテナー Java 仮想マシンがあるとします。今後 3 年間で、物理サーバーを 20 台まで増やす予定です。物理サーバーが 20 台ある場合、コンテナー・サーバー Java 仮想マシンは 40 になりますが、余裕を持って 60 を選択することにします。コンテナー JVM ごとに 4 つの断片が必要です。60 のコンテナーでは、断片は合計 240 になります。区画ごとにプライマリーとレプリカがあるとすると、120 の区画が必要になります。この例は、240 を 12 のコンテナー Java 仮想マシンで除算すること、つまり後でコンピューターを 20 台まで拡張できるようにするため、初期デプロイメント時にコンテナー JVM ごとに 20 の断片を想定することを示しています。

ObjectMap および区画化

デフォルトの FIXED_PARTITION 配置ストラテジーでは、マップは区画間で分割され、キーは異なる区画にハッシュされます。 クライアントは、どの区画にキーが属しているのかを知る必要はありません。 mapSet に複数のマップがある場合、マップは別々のトランザクションでコミットされるはずです。

エンティティーおよび区画化

エンティティー・マネージャー・エンティティーは、サーバーのエンティティーを処理するクライアント向けに最適化されています。マップ・セットに対するサーバ

ーのエンティティー・スキーマで、単一のルート・エンティティーを指定できます。クライアントは、このルート・エンティティーを介してすべてのエンティティーにアクセスする必要があります。それで、エンティティー・マネージャーは、同一区画のそのルートから関連エンティティーを検出することができ、関連マップには共通キーは必要ありません。ルート・エンティティーは単一区画とのアフィニティーを確立します。この区画は、アフィニティーの確立後、トランザクション内のすべてのエンティティーの取り出しに使用されます。このアフィニティーは、関連マップが共通キーを必要としないため、メモリーを節約できます。ルート・エンティティーは、以下の例に示すような変更したエンティティー・アノテーションで指定する必要があります。

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

このエンティティーを使用してオブジェクト・グラフのルートを検出することができます。オブジェクト・グラフは、1 つ以上のエンティティー間のリレーションシップを定義します。 リンクされた各エンティティーは同じ区画に解決される必要があります。 すべての子エンティティーはルートと同一の区画にあると仮定されます。オブジェクト・グラフ内の子エンティティーへのアクセスは、ルート・エンティティーのクライアントからのみ可能です。 区画化環境では、eXtreme Scale クライアントを使用してサーバーと通信するときに、常にルート・エンティティーが必要です。クライアントごとに定義できるルート・エンティティー・タイプは、1 つのみです。Extreme Transaction Processing (XTP) スタイルの ObjectGrid を使用している場合は、区画とのすべての通信が、クライアントおよびサーバー機構ではなく、直接のローカル・アクセスによって実行されるため、ルート・エンティティーは必要ありません。

配置と区画

WebSphere eXtreme Scale には、固定区画とコンテナーごとの、2 つの配置ストラテジーがあります。 配置ストラテジーの選択は、デプロイメント構成が区画をどのようにリモート・データ・グリッド内に配置するかに影響します。

固定区画配置

配置ストラテジーはデプロイメント・ポリシー XML ファイルで設定することができます。デフォルトの配置ストラテジーは固定区画配置で、これは FIXED_PARTITION 設定で使用可能になります。使用可能なコンテナーに配置されるプライマリー断片の数と numberOfPartitions 属性で構成した区画の数が等しくなります。レプリカを構成した場合は、配置される断片の最小合計数は次の式によって定義されます。((1 プライマリー断片 + 同期断片の最小数) * 定義されている区画の数)。配置される断片の最大合計数は次の式によって定義されます。 ((1 プライマリー断片 + 同期断片の最大数 + 非同期断片の最大数) * 区画数)。 WebSphere eXtreme Scale のデプロイメントにより、これらの断片は使用可能なコンテナーに拡散されます。各マップのキーは、定義した合計区画数に基づいて、割り当てられた区画にハッシュされます。フェイルオーバーやサーバー変更のために区画が移動された場合でも、これらのキーは同じ区画にハッシュされます。

例えば、numberPartitions 値が 6 で、MapSet1 の minSync 値が 1 である場合は、6 個の区画のそれぞれが同期レプリカを必要とするため、そのマップ・セットの合計断片数は 12 となります。コンテナーが 3 つ開始されると、WebSphere eXtreme Scale は MapSet1 用にコンテナーごとに 4 個の断片を配置します。

コンテナーごとの配置

代替配置ストラテジーはコンテナーごとの配置です。これは、デプロイメント XML ファイルのマップ・セット・エレメントにある placementStrategy 属性に対する PER_CONTAINER 設定で使用可能になります。このストラテジーでは、各新規コンテナーに配置されたプライマリー断片の数と構成した区画の数 (*P*) が等しくなります。 WebSphere eXtreme Scale デプロイメント環境では、残っているコンテナーごとに各区画の *P* 個のレプリカが配置されます。コンテナーごとの配置を使用していると、numInitialContainers 設定は無視されます。区画はコンテナーの増大につれて大きくなります。このストラテジーでは、マップのキーは特定の区画に固定されません。クライアントはある区画に経路指定して、ランダム・プライマリーを使用します。再度キーの検出に使用される同じセッションに再接続したいクライアントがあると、そのクライアントはセッション・ハンドルを使用しなければなりません。

詳しくは、「*プログラミング・ガイド*」に記載されている経路指定のための SessionHandle の使用に関するトピックを参照してください。

フェイルオーバーまたはサーバーが停止された場合、WebSphere eXtreme Scale 環境はプライマリー断片を (まだそこにデータが入っていれば) コンテナーごとの配置ストラテジーに従って移動します。空の断片は廃棄されます。コンテナーごとのストラテジーでは、すべてのコンテナーについて新しいプライマリー断片が配置されるため、古いプライマリー断片は保存されません。

WebSphere eXtreme Scale では、「標準的」配置ストラテジーと称される、区画の 1 つにマップのキーをハッシュする固定区画を使用した方法の代替として、コンテナーごとの配置が可能です。 コンテナーごとの場合 (PER_CONTAINER で設定)、デプロイメントは区画を一連のオンライン・コンテナー・サーバーに配置し、コンテナーがサーバー・データ・グリッドに追加またはサーバー・データ・グリッドから削除されるのに合わせて、自動的にスケールアウトまたはスケールインします。 固定区画を使用した方法のデータ・グリッドは、キー・ベースのグリッドに使用すると効果があり、アプリケーションはキー・オブジェクトを使用してグリッドのデータを検索します。 次に、代替方法について説明します。

コンテナーごとのデータ・グリッドの例

PER_CONTAINER データ・グリッドはさまざまです。 デプロイメント XML ファイルの placementStrategy 属性で、データ・グリッドが PER_CONTAINER 配置ストラテジーを使用するように指定します。 データ・グリッド内の区画が合計でいくつ必要なのかを構成する代わりに、開始するコンテナーごとに区画がいくつ必要なのかを指定します。

例えば、コンテナーごとに 5 つの区画を設定する場合、そのコンテナー・サーバーを始動すると、5 つの新しい匿名プライマリー区画が作成され、必要なレプリカはデプロイ済みの他のコンテナー・サーバー上に作成されます。

以下は、データ・グリッドが拡張していくに従い、コンテナーごとの環境で可能性のあるシーケンスです。

1. 5 つのプライマリー (P0 から P4) をホスティングしているコンテナー C0 を開始します。
   - C0 ホスト: P0、P1、P2、P3、P4。

2. さらに 5 つのプライマリー (P5 から P9) をホスティングしているコンテナー C1 を開始します。 コンテナー上でレプリカのバランスが取られます。
   - C0 ホスト: P0、P1、P2、P3、P4、R5、R6、R7、R8、R9。
   - C1 ホスト: P5、P6、P7、P8、P9、R0、R1、R2、R3、R4。
3. さらに 5 つのプライマリー (P10 から P14) をホスティングしているコンテナー C2 を開始します。 さらにレプリカのバランスが取られます。
   - C0 ホスト: P0、P1、P2、P3、P4、R7、R8、R9、R10、R11、R12。
   - C1 ホスト: P5、P6、P7、P8、P9、R2、R3、R4、R13、R14。
   - C2 ホスト: P10、P11、P12、P13、P14、R5、R6、R0、R1。

さらに多くのコンテナーが開始される間このパターンが続き、5 つの新しいプライマリー区画が毎回作成されて、データ・グリッド内の使用可能なコンテナー上で再度レプリカのバランスが取られます。

注: WebSphere eXtreme Scale は PER_CONTAINER ストラテジーを使用する場合、プライマリーは移動せず、レプリカのみを移動します。

区画番号は任意でキーと無関係なため、キー・ベースのルーティングは使用できないことに注意してください。 コンテナーが停止するとそのコンテナー用に作成された区画 ID は使用されなくなるので、区画 ID の間に抜けができます。 例では、コンテナー C2 に障害が起こると区画 P5 から P9 がなくなり、P0 から P4 と P10 から P14 のみが残るため、キー・ベースのハッシュは不可能です。

コンテナーの障害の影響を考慮すれば、5 あるいはさらに適当な 10 などの数字をコンテナーごとの区画数に使用するのが最も適切に機能します。 データ・グリッド中に断片を均等にホスティングするという負荷を分散するには、各コンテナーに対して複数の区画が必要です。 コンテナーごとの区画が単一だった場合、コンテナーに障害が起こると 1 つのコンテナー (対応するレプリカ断片をホスティングするコンテナー) だけで失われたプライマリーの負荷をすべて引き受けなければなりません。 この場合、負荷はコンテナーに対して直ちに 2 倍になります。 ただし、コンテナーごとに 5 つの区画があれば、5 つのコンテナーが失われたコンテナーの負荷を受け取り、各コンテナーへの影響は 80 パーセント減少します。 コンテナーごとに複数の区画を使用すると、各コンテナーへの実質的な影響の可能性は一般的に低くなります。 さらに直接的に、コンテナーが予想外に急増するケースを考えてください。コンテナーのレプリカ生成の負荷は、1 つだけではなく 5 つのコンテナーに分散されます。

コンテナーごとのポリシーの使用

いくつかのシナリオでは、HTTP セッション・レプリカ生成またはアプリケーション・セッション状態などの場合、コンテナーごとのストラテジーは理想的な構成だと考えられています。 そのような場合、HTTP ルーターはセッションをサーブレット・コンテナーに割り当てます。 サーブレット・コンテナーは HTTP セッションを作成する必要があり、5 つのローカル・プライマリー区画のうちの 1 つをそのセッション用に選択します。 その後、選択された区画の「ID」は Cookie に保管されます。 これでサーブレット・コンテナーは、セッション状態へのローカル・アクセス権限を持ち、セッション・アフィニティーが保持されている限り、この要求に対

するデータへのアクセス待ち時間はゼロになることを意味します。 さらに、eXtreme Scale は、すべての変更を区画に複製します。

実際には、コンテナーごとに複数区画を持つ場合の悪影響に注意してください (再度 5 つの例を使用します)。 当然、新たに開始した各コンテナーには、さらに 5 つのプライマリー区画と、さらに 5 つのレプリカがあります。 時間とともに、さらに区画が作成され、移動または破棄されないはずです。しかし、これは実際にコンテナーが振る舞う様子ではありません。 コンテナーは、開始すると 5 つのプライマリー断片をホストします。これは「ホーム」プライマリーと呼ばれ、それらを作成したそれぞれのコンテナー上に存在します。 コンテナーに障害が起こると、レプリカがプライマリーになり、eXtreme Scale はさらに 5 つのレプリカを作成し、高可用性を保持します (自動修復を使用不可にしていない場合)。新規プライマリーはそれを作成したコンテナーとは別のコンテナーにあり、「外部」プライマリーと呼ばれます。 アプリケーションは、新規状態または新規セッションを外部プライマリーには決して配置しません。 最終的に、外部プライマリーはエントリーを持たず、eXtreme Scale は外部プライマリーとその関連レプリカを自動的に削除します。外部プライマリーの目的は、既存のセッション (新しいセッションではなく) を引き続き使用可能にしておくことです。

クライアントは、キーを利用しないデータ・グリッドとも対話することができます。 クライアントは、ただトランザクションを開始し、データをどのキーとも無関係のデータ・グリッドに保管します。クライアントは、必要なときに同じ区画と対話できるようにするシリアライズ可能ハンドル、SessionHandle オブジェクトをセッションに要求します。 詳しくは、「*プログラミング・ガイド*」に記載されている経路指定のための SessionHandle の使用に関するトピックを参照してください。 WebSphere eXtreme Scale は、ホーム・プライマリー区画のリストからクライアント用の区画を選択します。外部プライマリー区画は戻されません。 SessionHandle は、例えば HTTP Cookie でシリアライズされ、後で Cookie を 元の SessionHandle に変換することができます。 次に WebSphere eXtreme Scale API は SessionHandle を使用して、再度同じ区画にバインドされたセッションを取得します。

注: エージェントを使用して PER_CONTAINER データ・グリッドと対話することはできません。

利点

コンテナーごとのクライアントは、データをグリッド内の場所に保管し、データへのハンドルを取得し、そのハンドルを使用してデータに再びアクセスするため、上記の説明は通常の FIXED_PARTITION またはハッシュ・データ・グリッドと異なります。 固定区画の場合のような、アプリケーション提供のキーはありません。

デプロイメントは、各セッションごとに新規区画を作りません。 そのため、コンテナーごとのデプロイメントでは、データを区画に保管するために使用されるキーは、その区画内で固有でなければなりません。 例えば、クライアントが固有の SessionID を生成し、それをキーとして使用してその区画内のマップの情報を検索するという例が考えられます。 複数のクライアント・セッションが同じ区画と対話するため、アプリケーションは固有のキーを使用してセッション・データを特定の各区画に保管する必要があります。

上記の例では 5 つの区画を使用しましたが、objectgrid XML ファイルの numberOfPartitions パラメーターを使用すると、必要に応じて区画を指定することができます。データ・グリッドごとではなく、設定はコンテナーごとです。 (レプリカの数は、固定区画ポリシーの場合と同じ方法で指定されます。)

コンテナーごとのポリシーは、複数ゾーンでも使用できます。 可能であれば、eXtreme Scale は、プライマリーがそのクライアントと同じゾーンにある区画に SessionHandle を返します。クライアントは、コンテナーのパラメーターとして、または API を使用してゾーンを指定できます。 クライアント・ゾーン ID は serverproperties または clientproperties を使用して設定することができます。

データ・グリッドの PER_CONTAINER ストラテジーは、データベース指向データではなく、会話型状態を保管するアプリケーションに適しています。データにアクセスするキーは会話 ID で、特定のデータベース・レコードには関係しません。 このストラテジーにより、パフォーマンスはさらに向上し (例えばプライマリー区画をサーブレットと連結できるので)、構成はさらに容易になります (区画およびコンテナーを計算する必要はありません)。

単一区画トランザクションおよびクロスデータ・グリッド・トランザクション

WebSphere eXtreme Scale とリレーショナル・データベースやメモリー内データベースなどの従来のデータ・ストレージ・ソリューションとの間の主な相違は、キャッシュの直線的な増加を可能にする区画化を使用することにあります。考慮すべき重要なトランザクションのタイプに、単一区画トランザクションと各区画 (クロスデータ・グリッド) トランザクションがあります。

一般的に、以下のセクションで説明するようにキャッシュとの対話は、単一区画トランザクションまたはクロスデータ・グリッド・トランザクションとして分類できます。

単一区画トランザクション

単一区画トランザクションは、WebSphere eXtreme Scale によってホストされるキャッシュと対話する場合に適した方法です。単一区画に制限されている場合のトランザクションは、デフォルトで単一の Java 仮想マシン、すなわち単一のサーバー・コンピューターに制限されます。サーバーは、こうしたトランザクションを毎秒 *M* 個実行することができるので、*N* 台のコンピューターがある場合は、毎秒 M*N 個のトランザクションを実行できます。ビジネスが拡大し、毎秒こうしたトランザクションを 2 倍の数実行する必要性が出てきた場合、さらにコンピューターを購入して *N* を 2 倍にすることができます。これにより、アプリケーションを変更したり、ハードウェアをアップグレードしたり、さらにはアプリケーションをオフラインにしたりすることさえなく、容量ニーズを満たすことができます。

単一区画トランザクションは、キャッシュの拡大をかなり大幅に行えるようになっているほか、キャッシュの可用性を最大限に引き出します。各トランザクションは、1 台のコンピューターのみに依存します。他の (N-1) 台のコンピューターのいずれかに障害が起こっても、このトランザクションの成否および応答時間には影響しません。したがって、100 台のコンピューター (サーバー) を稼働していて、そのうち 1 台に障害が生じても、そのサーバーに障害が生じた時点で進行中であった 1

パーセントのトランザクションしかロールバックされません。サーバーの障害後、WebSphere eXtreme Scale は、障害を起こしたサーバーによってホストされる区画を他の 99 台のコンピューターに再配置します。これは短時間の処理であり、この操作の完了前であれば、この時間内に他の 99 台のコンピューターはトランザクションを完了できます。再配置される区画に関係するトランザクションしか、ブロックされません。フェイルオーバー・プロセスが完了すると、キャッシュは、元のスループット量の 99 パーセントで完全に操作可能状態で引き続き稼働できるようになります。障害のあるサーバーが交換されて、データ・グリッドに戻されると、キャッシュは 100 パーセントのスループット量に戻ります。

クロスデータ・グリッド・トランザクション

パフォーマンス、可用性、およびスケーラビリティーの面では、クロスデータ・グリッド・トランザクションは、単一区画トランザクションの対極にあります。クロスデータ・グリッド・トランザクションは、すべての区画、つまり構成内のすべてのコンピューターにアクセスします。データ・グリッド内の各コンピューターは、ある種のデータを検索して、その結果を戻すように求められます。トランザクションは、すべてのコンピューターが応答するまで完了できません。したがってデータ・グリッド全体のスループットは、最低速のコンピューターによって制限されます。コンピューターを追加しても、最低速のコンピューターの処理速度が増すわけではなく、キャッシュのスループットは改善しません。

クロスデータ・グリッド・トランザクションは、可用性についても同じ影響を及ぼします。先の例を拡大すると、100 台のサーバーが稼働していて、そのうち 1 台に障害が生じたとすると、そのサーバーに障害が生じた時点で進行中であったトランザクションの 100 パーセントがロールバックされます。サーバーの障害後、WebSphere eXtreme Scale は、このサーバーによってホストされる区画を他の 99 台のコンピューターに再配置する処理を開始します。この時間の間、フェイルオーバー・プロセスが完了するまでは、データ・グリッドは、該当するトランザクションをどれも処理できなくなります。フェイルオーバー・プロセスが完了すると、キャッシュは、続行できるようになりますが、容量は減少します。データ・グリッド内の各コンピューターが 10 個の区画をサービスしていた場合、残りの 99 台のコンピューターのうち 10 台は、フェイルオーバー・プロセスの一部として少なくとも 1 つの余分の区画を受け取ることになります。余分の区画を 1 つ追加すると、該当コンピューターのワークロードは 10 パーセント以上増えます。データ・グリッドのスループットは、クロスデータ・グリッド・トランザクション内の最低速のコンピューターのスループットに制限されるので、平均して、スループットは 10 パーセント減少します。

WebSphere eXtreme Scale のような高可用性の分散オブジェクト・キャッシュでのスケールアウトの場合は、単一区画トランザクションのほうがクロスデータ・グリッド・トランザクションよりも適しています。こうした種類のシステムのパフォーマンスを最大限にするには、従来のリレーショナルの方法論とは異なる手法を使用する必要がありますが、クロスデータ・グリッド・トランザクションをスケーラブルな単一区画トランザクションに変えることができます。

スケーラブル・データ・モデルのビルドのベスト・プラクティス

WebSphere eXtreme Scale のような製品でのスケーラブル・アプリケーションをビルドする際のベスト・プラクティスには、基本原則と実装ヒントという 2 つのカテゴ

リーがあります。基本原則は、データ自体の設計に取り込む必要がある中心的なアイデアです。こうした原則を守らないアプリケーションは、たとえそのメインライン・トランザクションに対しても、適切に拡大できる可能性が低くなります。実装ヒントは、スケーラブル・データ・モデルの本来は一般的な原則に従って適切に設計されたアプリケーション内の問題のあるトランザクションに適用されます。

基本原則

スケーラビリティーを最適化する重要な手段の一部として、基本的な概念または原則を考慮する必要があります。

正規化に代わる重複

WebSphere eXtreme Scale のような製品の場合、その製品が多数のコンピューター間でデータを展開できるように設計されているということを念頭に入れておくことが重要です。ほとんどまたはすべてのトランザクションを単一区画で完全なものとすることが目標である場合は、データ・モデル設計で、トランザクションが必要とする可能性のあるすべてのデータがその区画に存在するようにする必要があります。ほとんどの場合、データを複製することによってのみ、この目標を実現できます。

例えば、メッセージ・ボードのようなアプリケーションを考えてみます。メッセージ・ボードの 2 つの極めて重要なトランザクションとして、一定のユーザーからのすべてのポスト・メッセージを表示するものと、特定のトピックに関するすべてのポスト・メッセージを表示するものがあります。まずこうしたトランザクションがユーザー・レコード、トピック・レコード、さらに実際のテキストが含まれるポスト・レコードを含む正規化されたデータ・モデルをどのように扱うかを考えてみます。ポスト・メッセージがユーザー・レコードによって区画に分割されている場合、トピックを表示することは、クロスグリッド・トランザクションとなります。またその逆もいえます。トピックおよびユーザーは、多対多の関係を持っているので一緒に区画に分割することはできません。

このメッセージ・ボードの拡大を行う最善の策は、ポスト・メッセージを複製して、トピック・レコードを持つコピーを 1 つ、ユーザー・レコードを持つコピーを 1 つ保存することです。この結果、ユーザーからのポスト・メッセージを表示することは単一区画トランザクションとなり、トピックに関するポスト・メッセージを表示することは単一区画トランザクションとなり、ポスト・メッセージを更新または削除することは、2 区画トランザクションとなります。データ・グリッド内のコンピューターの数が増えるにつれ、これら 3 つのトランザクションがすべて直線的に拡大します。

リソースに代わるスケーラビリティー

非正規化されたデータ・モデルを考慮する場合に克服すべき最大の障害は、こうしたモデルがリソースに与える影響です。ある種のデータのコピーを 2 つ、3 つ、またはそれ以上保持すると、利用される資源が多すぎるように見える場合があります。こうしたシナリオに直面したら、ハードウェア・リソースが年々低価格になっているという事実を思い出してください。第 2 に (さらに重要)、WebSphere eXtreme Scaleは、追加資源のデプロイに関連した隠れコストを削減します。

メガバイトやプロセッサーといったコンピューター関連ではなく、コスト関連でリソースを測定してください。正規化された関係データを扱うデータ・ストアは、一般的に同じコンピューターに存在する必要があります。こうしたコロケーションの必要性から、いくつか小型コンピューターを購入するのではなく、1 台の大型の企業向けコンピューターを購入したほうがよいという結果が導かれます。ただし企業向けハードウェアの場合、通常では、毎秒 100 万のトランザクションの実行が可能な 1 台のコンピューターを使用するほうが、それぞれ毎秒 10 万のトランザクションの実行が可能な 10 台のコンピューターを結合した場合よりコストがかなりかかることは珍しいことではありません。

リソースを追加する際のビジネス・コストも存在します。ビジネスが成長していくと、結果的に容量不足となります。容量不足となると、より大型の高速コンピューターに移行する際にシャットダウンが必要になるか、切り替え可能な第 2 の実稼働環境の作成が必要になります。いずれにせよ、ビジネス損失が発生するか、遷移期間にほぼ 2 倍の容量の維持が必要になるという形で追加コストが発生します。

WebSphere eXtreme Scale を使用すると、容量追加のためにアプリケーションをシャットダウンする必要がなくなります。ビジネスで翌年に 10 パーセントの追加容量が必要になることが見込まれた場合、データ・グリッド内のコンピューターの数を 10 パーセント増加します。このパーセンテージ分の増加の際に、アプリケーション・ダウン時間もなく、超過容量の購入の必要もありません。

データ形式変更の防止

WebSphere eXtreme Scale を使用している場合、データは、ビジネス・ロジックで直接消費可能な形式で保管されます。データをよりプリミティブな形式に分解することには、コストがかかります。データの書き込みおよび読み取り時に、変換を実行する必要があります。リレーショナル・データベースを使用する場合、データが最終的にディスクにパーシストされることがごく頻繁に行われるため、この変換は必要に応じて実行されますが、WebSphere eXtreme Scale を使用すると、こうした変換を実行する必要がなくなります。データは大部分メモリーに保管されるため、アプリケーションが必要とするそのままの形式で保管することができます。

この単純な規則に従うと、最初の原則に従ってデータを非正規化するのに役立ちます。ビジネス・データ用の最も一般的なタイプの変換は、正規化されたデータをアプリケーションのニーズに合う結果セットに変えるために必要な JOIN 演算です。データを正しい形式で保管すると、暗黙的にこうした JOIN 演算の実行が避けられ、非正規化されたデータ・モデルが作成されます。

未結合照会の除去

いくらデータを適切に構成しても、未結合照会は正しく拡張されません。例えば、値でソートされたすべての項目のリストを要求するようなトランザクションは使用しないでください。こうしたトランザクションは、はじめのうち合計項目数が 1000 であると、機能するかもしれませんが、合計項目数が 1000 万に達すると、トランザクションは 1000 万すべての項目を戻します。このトランザクションを実行した場合、最も考えられる 2 つの結果

は、トランザクションのタイムアウトになるか、クライアントにメモリー不足エラーが発生するかのいずれかです。

最善のオプションは、上位 10 または 20 の項目だけが戻されるように、ビジネス・ロジックを変更することです。このロジック変更によって、キャッシュ内の項目数に関係なく、トランザクションのサイズが管理可能な程度に保たれます。

スキーマの定義

データの正規化の主な利点は、データベース・システムが状況の背後にあるデータの整合性を考慮できることです。データがスケーラビリティーのために非正規化されると、この自動データ整合性管理は存在しなくなります。データの整合性を保証するために、アプリケーション層で機能できるか、分散データ・グリッドに対するプラグインとして機能できるデータ・モデルを実装する必要があります。

メッセージ・ボードの例を考えてみます。トランザクションがトピックからポスト・メッセージを除去した場合、ユーザー・レコード上の重複するポスト・メッセージを除去する必要があります。データ・モデルがなくても、開発者は、トピックからポスト・メッセージを除去し、さらに確実にユーザー・レコードからそのポスト・メッセージを除去するアプリケーション・コードを作成することができます。ただし、仮に開発者がキャッシュと直接に対話する代わりにデータ・モデルを使用していたとしても、データ・モデル上の removePost メソッドによって、ポスト・メッセージからユーザー ID を抜き出して、ユーザー・レコードを検索し、この状況の背後にある重複ポスト・メッセージを除去することができます。

あるいは、実際の区画で実行し、トピックの変更を検出して、ユーザー・レコードを自動的に調整するリスナーを実装することができます。リスナーは、役に立ちます。区画がユーザー・レコードを持つようになった場合に、ユーザー・レコードの調整がローカルで可能になるか、ユーザー・レコードが異なる区画にあっても、トランザクションがクライアントとサーバーの間ではなく、サーバー間で実行されるためです。サーバー間のネットワーク接続のほうが、クライアントとサーバーの間のネットワーク接続よりも高速である可能性があります。

競合の防止

グローバル・カウンターを持つようなシナリオは避けてください。1 つのレコードが残りのレコードと比べて極端に多く使用されている場合は、データ・グリッドは拡張されません。データ・グリッドのパフォーマンスは、この特定のレコードを保持するコンピューターのパフォーマンスによって制限されています。

このような状態では、そのレコードを区画単位で管理できるように分割してみてください。例えば、分散キャッシュ内の合計エントリー数を戻すトランザクションを考えます。すべての挿入および除去操作で増大する単一のレコードにアクセスする代わりに、各区画のリスナーに挿入および除去操作を追跡させます。このリスナーによるトラッキングを使用すると、挿入および除去を単一区画操作とすることができます。

カウンターの読み取りはクロスデータ・グリッド操作となりますが、ほとんどの場合、それは元々クロスデータ・グリッド操作と同じく非効率的です。そのパフォーマンスがレコードをホストするコンピューターのパフォーマンスと関係しているためです。

実装ヒント

最善のスケーラビリティーを達成するには、以下のヒントも考慮してください。

逆引き索引の使用

顧客レコードが顧客 ID 番号に基づいて区画化されるような適切に非正規化されたデータ・モデルを考えます。この区画化方法は論理的な選択といえます。顧客レコードによって実行されるほぼすべてのビジネス・オペレーションは、顧客 ID 番号を使用するからです。ただし、顧客 ID 番号を使用しない重要なトランザクションに、ログイン・トランザクションがあります。ログインには顧客 ID 番号よりもユーザー名や電子メール・アドレスが使用されるほうが一般的です。

ログイン・シナリオの簡単な方法は、顧客レコードを見つけるためにクロスデータ・グリッド・トランザクションを使用することです。先に説明したように、この方法は拡張されません。

次のオプションとして、ユーザー名または電子メールに基づいて区画化することがあります。このオプションは、顧客 ID に基づくすべての操作がクロスデータ・グリッド・トランザクションとなるので、実用的でありません。またサイトのユーザーがユーザー名や電子メール・アドレスを変更したい場合もあります。WebSphere eXtreme Scale のような製品は、データをその不変性の維持のために区画化するのに使用される値を必要とします。

適切な解決方法として、逆引き索引を使用することができます。WebSphere eXtreme Scale を使用すると、すべてのユーザー・レコードを保持するキャッシュと同じ分散グリッドにキャッシュを作成できます。このキャッシュは、高可用性で、区画化され、しかもスケーラブルです。このキャッシュは、ユーザー名または電子メール・アドレスを顧客 ID にマップするために使用できます。このキャッシュでは、ログインは、クロスグリッド操作ではなく 2 区画操作となります。このシナリオは単一区画トランザクションほどよくはありませんが、コンピューターの数が増えるにつれ、スループットが直線的に増加します。

書き込み時の計算

平均や合計などの一般的な計算値は、作成にコストがかかることがあります。こうした操作には、通常膨大な数のエントリーを読み取る必要があるためです。ほとんどのアプリケーションでは、読み取りのほうが書き込みよりも一般的であるため、こうした値を書き込み時に計算し、結果をキャッシュに保管するほうが効率的です。これにより、読み取り操作は高速になり、よりスケーラブルになります。

オプション・フィールド

業務内容、自宅住所、および電話番号を保持するユーザー・レコードを考えます。これらすべてが定義されているユーザーもいれば、まったく定義されていないユーザーもいれば、一部が定義されているユーザーもいます。デー

タが正規化されていると、ユーザー・テーブルおよび電話番号テーブルが存在することになります。一定ユーザーの電話番号は、この 2 つのテーブル間の JOIN 操作を使用して検出できます。

このレコードを非正規化する場合、データの重複は必要ありません。ほとんどのユーザーが電話番号を共有しないためです。代わりに、ユーザー・レコードで空スロットを使用できるようになっている必要があります。電話番号テーブルを使用する代わりに、各ユーザー・レコードに電話番号タイプごとに 1 つずつ 3 つの属性を追加します。この属性の追加により、JOIN 操作がなくなり、ユーザーの電話番号検索が単一区画操作となります。

多対多関係の配置

製品とその販売店を追跡するアプリケーションを考えてみます。1 つの製品が多くの店舗で販売され、1 つの店舗で多くの製品が販売されます。このアプリケーションが 50 の大規模小売業者を追跡するものとします。各製品が最大 50 の店舗で販売され、それぞれの店舗で何千もの製品が販売されます。

各店舗エンティティー内に製品リストを保持する (配置 B) 代わりに、製品エンティティーの内部に店舗リストを保持します (配置 A)。このアプリケーションが実行する必要があるトランザクションの一部を見ると、配置 A がよりスケーラブルである理由が明らかになります。

まず更新に注目します。配置 A では、店舗の在庫から製品を除去する場合、製品エンティティーがロックされます。データ・グリッドに 10000 の製品が保持されている場合、グリッドの 1/10000 しか更新の実行をロックする必要がありません。配置 B では、データ・グリッドには 50 の店舗しか含まれていないので、更新を完了するには、グリッドの 1/50 をロックする必要があります。これらは両方とも単一区画操作と考えることができますが、配置 A のほうがより効率よくスケールアウトされます。

現在、配置 A による読み取りを考えていますから、トランザクションで少量のデータのみが転送されるため、製品の販売店舗の検索は拡張され、高速な単一区画トランザクションとなります。配置 B では、製品が店舗で販売されているかどうかを確認するために、各店舗エンティティーにアクセスする必要があるため、このトランザクションはクロスデータ・グリッド・トランザクションになります。これは、配置 A の大きなパフォーマンス上の利点を明らかにします。

正規化されたデータによる拡張

クロスデータ・グリッド・トランザクションの正当な使用法の 1 つにデータ処理の拡張があります。データ・グリッドに 5 台のコンピューターがあり、各コンピューターについて約 100,000 のレコード全部をソートするクロスグリッド・トランザクションがディスパッチされると、そのトランザクションは全体で 500,000 個のレコードをソートします。データ・グリッド内の最低速のコンピューターが毎秒これらのトランザクションのうちの 10 個を実行できる場合、データ・グリッドは全体で毎秒 5,000,000 レコードをソートできます。グリッド内のデータが 2 倍になると、各コンピューターは全体で 200,000 個のレコードをソートする必要があり、各トランザクションは全体で 1,000,000 個のレコードをソートします。このデータの増加は、最低速のコンピューターのスループットを毎秒 5 トランザクションに

減少させるので、データ・グリッドのスループットは毎秒 5 トランザクションに減少します。それでもデータ・グリッドは全体で毎秒 5,000,000 レコードをソートします。

このシナリオでは、コンピューターの数を 2 倍にすると、各コンピューターは元の 100,000 レコードのソートという負荷状態に戻るので、最低速のコンピューターは、これらのトランザクションを毎秒 10 個で処理できるようになります。データ・グリッドのスループットは、毎秒 10 要求という同じ状態ですが、現在では各トランザクションは 1,000,000 レコードを処理するので、処理するレコードに関してはグリッドの容量は毎秒 10,000,000 レコードと 2 倍になります。

ユーザー数の増加に合わせてインターネットとスループットの規模を拡大するため、データ処理に関して両方を拡張する必要のある検索エンジンなどのアプリケーションでは、グリッド間の要求のラウンドロビンを備えた複数のデータ・グリッドを作成する必要があります。スループットを拡大する必要がある場合、要求をサービスするために、コンピューターを追加し、別のデータ・グリッドを追加します。データ処理を拡大する必要がある場合、コンピューターを追加して、データ・グリッド数を一定に保ちます。

ユニットまたはポッドでの拡張

データ・グリッドを何千という Java 仮想マシンにわたってデプロイすることは可能ですが、構成テストの信頼性を高め、その実行を簡単にするには、データ・グリッドをユニット、つまり、ポッドに分割することを検討することをお勧めします。ポッドとは、同じアプリケーションのセットを実行しているサーバーの一群です。

大きな単一データ・グリッドのデプロイ

テストによって、eXtreme Scale が 1000 を超す JVM にスケールアウトできることが検証されています。 このようなテストでは、単一データ・グリッドを多数のコンピューターにデプロイするようなアプリケーションの構築を勧められます。そのようなアプリケーションの構築は可能ですが、以下のようないくつかの理由により推奨されません。

1. **予算の問題:** ご使用の環境では現実的に 1000 のサーバー・データ・グリッドをテストできません。 ただし、予算の理由を考慮して、より小さなデータ・グリッドをテストすることはできますので、特にそのような多数のサーバーの場合、2 倍のハードウェアを購入する必要はありません。
2. **異なるアプリケーションのバージョン:** スレッドをそれぞれテストするために多数のコンピューターを必要とするのは実用的ではありません。 実稼働環境で行うのと同じ要因をテストしないことがリスクになります。
3. **データ損失:** データベースを単一ハード・ディスクで実行することは信用性が高くありません。 ハード・ディスクに問題が発生すると、データ損失の原因になります。成長するアプリケーションを単一データ・グリッドで実行するのも同様です。 ご使用の環境およびご使用のアプリケーションにバグがある可能性があります。 したがって、すべてのデータを大きな単一システムに配置することによって、大量のデータの損失につながることがあります。

データ・グリッドの分割

アプリケーション・データ・グリッドをポッド (ユニット) に分割することは、より信頼性の高い選択肢です。ポッドとは、同種のアプリケーション・スタックを実行するサーバーの一群です。 ポッドのサイズは任意ですが、約 20 台の物理サーバーで構成されるのが理想的です。 単一データ・グリッドに 500 台の物理コンピューターがあるよりも、20 台の物理コンピューターの 25 ポッドにしてください。 単一の種類のアプリケーション・スタックは、指定されたポッドで実行する必要がありますが、異なるポッドが独自の種類のアプリケーション・スタックを持つことは構いません。

一般的に、アプリケーション・スタックは以下のコンポーネント・レベルを考慮します。

- オペレーティング・システム
- ハードウェア
- JVM
- WebSphere eXtreme Scale のバージョン
- アプリケーション
- その他の必要なコンポーネント

ポッドは、テストに都合のいいようにサイズ変更したデプロイメント・ユニットです。数百台のサーバーでテストを行う代わりに、20 台のサーバーで行うのはより実用的です。 この場合、実動環境と同じ構成を引き続きテストしていきます。 実動環境では、1 つのポッドを構成する、20 台のサーバーの最大サイズでグリッドが使用されます。 1 つのポッドにストレス・テストをかけ、そのキャパシティー、ユーザー数、データ量、およびトランザクション・スループットを判別できます。 これによって、より簡単に計画が立てやすく、予測可能なコストで予測可能な拡張を行うという規格に従うことができます。

ポッド・ベースの環境の設定

別のケースでは、ポッドには必ずしも 20 台のサーバーが必要というわけではありません。 ポッドのサイズの目的は、実用的なテストのためです。実動環境でポッドに問題が発生しても、影響を受ける一部のトランザクションが耐えられるように、ポッドのサイズは十分に小さくしてください。

バグが発生しても、単一のポッドにのみ影響するのが理想的です。バグは 100 パーセントではなく、アプリケーション・トランザクションの 4 パーセントにしか影響を与えません。 さらに、一度に 1 つのポッドをロールアウトできるので、アップグレードはより簡単です。 結果として、ポッドへのアップグレードで問題が生じた場合、ユーザーはそのポッドを切り替えて前のレベルに戻すことができます。 アップグレードには、アプリケーションに対する変更、アプリケーション・スタックに対する変更、またはシステム更新を含みます。問題診断をより正確に行うために、アップグレードではできる限り、スタックのエレメントを一度に 1 つだけ変更するようにしてください。

ポッドを使用する環境を実装するには、ポッドにソフトウェアのアップグレードがあった場合に上下に互換性のあるルーティング層がポッドの上に必要です。 また、どのポッドが何のデータを持っているかについての情報を含むディレクトリーを作

成する必要もあります。 (このために、できれば後書きシナリオを使って、別の eXtreme Scale データ・グリッドをその後ろのデータベースと一緒に使用してください。) これは、2 層の解決策を生み出します。 層 1 はディレクトリーで、特定のトランザクションを処理するポッドを検索するために使用されます。 層 2 はポッド自体で構成されます。 層 1 がポッドを識別すると、セットアップは各トランザクションをポッド内の適切なサーバーに送付します。このサーバーは、通常、トランザクションが使用するデータ用区画を保持するサーバーです。 さらに、必要であれば層 1 でニア・キャッシュを使用して、適切なポッドの検索に関連する影響を小さくすることができます。

ポッドの使用は、単一データ・グリッドを持つよりも少し複雑ですが、操作、テスト、および信頼性の面で改善されるため、スケーラビリティーのテストの重要な一部になっています。

可用性の概要

高可用性

高可用性を備えている WebSphere eXtreme Scale は、信頼できるデータの冗長性を備え、障害も検出できます。

WebSphere eXtreme Scale は、 Java 仮想マシン のデータ・グリッドを自己編成して、ゆるやかに連合する 1 つのツリーにします。そのツリーのルートにはカタログ・サービスが置かれ、ツリーのリーフ部分にはコンテナーを保持するコア・グループが置かれます。詳しくは、 12 ページの『キャッシング・アーキテクチャー: マップ、コンテナー、クライアント、およびカタログ』を参照してください。

各コア・グループはカタログ・サービスによって自動的に作成され、約 20 個のサーバーからなるグループに入れられます。コア・グループ・メンバーは、グループ内の他メンバーのヘルス・モニタリングを提供します。また、各コア・グループは、カタログ・サービスにグループ情報を伝達するためのリーダーとして 1 つのメンバーを選びます。コア・グループのサイズを制限することにより、良好なヘルス・モニタリングおよび高度にスケーラブルな環境を維持できます。

注: コア・グループ・サイズを変更できる WebSphere Application Server 環境では、eXtreme Scale は、コア・グループあたり 50 を超えるメンバー数はサポートしません。

ハートビート処理

1. Java 仮想マシン間ではソケットがオープン状態のままなので、ソケットが予想外に閉じると、この予想外のクローズはピア Java 仮想マシンの障害として検出されます。この検出機能は、Java 仮想マシンが極端に早く終了したなどの障害事例をキャッチします。また、この検出機能により、通常 1 秒足らずで、こうしたタイプの障害から回復できます。
2. その他のタイプの障害には、オペレーティング・システム・パニック、物理サーバー障害、ネットワーク障害などがあります。こうした障害は、ハートビート処理を使用して検出します。

プロセスのペア間では定期的にハートビートが送信されます。一定数のハートビートが欠落すると、障害とみなされます。この方法は、N*M 秒で障害を検出します。N は欠落ハートビートの数で、M はハートビート間隔です。M と N を直接指定することはサポートされていません。スライダー・メカニズムを使用してテストされる一定範囲の M と N の組み合わせを指定できるようになっています。

障害

プロセスに障害が起きるのには、いくつかの場合があります。何らかのリソース限界に達したり (例えば、最大ヒープ・サイズ)、プロセス制御ロジックがプロセスを強制終了したといった理由で、プロセスに障害が起こることがあります。オペレーティング・システムに障害が起きると、システム上で実行中のすべてのプロセスが失われます。ネットワーク・インターフェース・カード (NIC) などの頻繁には障害が起きないハードウェアに障害が起きると、オペレーティング・システムがネットワークから切断されます。さらに多くのポイントで障害が起きると、プロセスが使用不可になります。このような状況において、これらすべての障害は、プロセス障害または接続不良の 2 つの障害タイプのうちのいずれかに分類されます。

プロセス障害

WebSphere eXtreme Scale は、プロセス障害に迅速に対応します。プロセスに障害が起きると、オペレーティング・システムは、プロセスが使用していたリソースの残りすべてをクリーンアップする必要が生じます。このクリーンアップには、ポートの割り当ておよび接続が含まれています。プロセスに障害が起きると、信号はそのプロセスによって使用されていた接続を通して送信され、各接続がクローズされます。これらの信号を使用し、障害の起きたプロセスに接続されている他のプロセスによって即時にプロセス障害が検出されます。

接続不良

オペレーティング・システムが切断されると、接続不良が発生します。その結果として、オペレーティング・システムは信号を他のプロセスに送信することができなくなります。接続不良が発生する理由はいくつかありますが、それらの理由は、ホスト障害と孤立化の 2 つのカテゴリーに分割されます。

ホスト障害

マシンの電源コンセントのプラグが抜かれると、即座に動作しなくなります。

孤立化

このシナリオは、使用不可であると見なされたプロセスが実際には使用不可ではないため、ソフトウェアが正しく対処することが最も難しい障害の状態です。基本的に、システムにはサーバーまたは他のプロセスが失敗したように見えるが、実際は正常に実行しているという状況です。

コンテナー障害

コンテナー障害は、通常コア・グループ・メカニズムを通してピア・コンテナーによって発見されます。コンテナーまたはコンテナーのセットに障害が起きると、カタログ・サービスにより、そのコンテナーにホスティングされていた断片が移行さ

れます。カタログ・サービスにより、非同期のレプリカに移行する前に最初に同期レプリカが検索されます。プライマリー断片が新規ホスト・コンテナーに移行された後で、カタログ・サービスにより、欠落しているレプリカの新規ホスト・コンテナーが検索されます。

注: コンテナー孤立化 - コンテナーが使用不可であることが検出されると、カタログ・サービスによりコンテナーの断片がコンテナーから移行されます。これらのコンテナーが使用可能になると、カタログ・サービスは、通常の開始フローの場合と同じように、これらのコンテナーを配置可能とみなします。

コンテナー障害検出までの待ち時間

障害は、ソフトとハードの障害に分類されます。ソフト障害の原因は、一般にプロセスの障害です。そのような障害はオペレーティング・システムによって検出されます。オペレーティング・システムでは、ネットワーク・ソケットなどの使用されたリソースを迅速にリカバリーできます。ソフト障害の場合、標準的な障害検出までの時間は、1 秒未満です。ハード障害の場合、デフォルトのハートビート・チューニングを使用すると、検出まで最長 200 秒かかることもあります。そのような障害は、物理的なマシンの破損、ネットワーク・ケーブル切断、オペレーティング・システム障害などです。ハード障害を検出するために、ランタイムは構成可能なハートビートに依存します。

カタログ・サービス障害

カタログ・サービス・グリッドは、1 つの eXtreme Scale グリッドなので、コンテナー障害プロセスと同じようにコア・グループ・メカニズムも使用します。主な相違点は、カタログ・サービス・ドメインでは、コンテナーに使用するカタログ・サービス・アルゴリズムの代わりに、プライマリー断片の定義にピア選択プロセスを使用する点です。

配置サービスおよびコア・グループ化サービスは、N 個の中の 1 つのサービスです。N 個の中の 1 つのサービスは、高可用性グループのメンバーの 1 つで実行されます。 ロケーション・サービスおよび管理は、高可用性グループのすべてのメンバーで実行されています。配置サービスおよびコア・グループ化サービスは、システムをレイアウトする必要があるため別のものです。ロケーション・サービスおよび管理は読み取り専用サービスであり、スケーラビリティーを提供するためにあらゆる場所に存在します。

カタログ・サービスはレプリカ生成を使用して、独自の障害限界を設定します。カタログ・サービス・プロセスに障害が起きると、サービスが再開し、システムを必要なレベルの可用性に復元します。カタログ・サービスをホスティングしているすべてのプロセスで障害が起こると、データ・グリッドから重要なデータが失われます。この障害が発生した場合は、すべてのコンテナー・サーバーを再始動する必要があります。カタログ・サービスは多くのプロセスで実行されているため、この障害は起きる可能性のないイベントです。ただし、単一のボックスですべてのプロセスを実行している場合は、単一のブレード・シャーシ内、または単一のネットワーク・スイッチで、障害が起きる可能性があります。カタログ・サービスをホスティングしているボックスから共通の障害モードを除去して、障害が起きる可能性を減らします。

複数のコンテナー障害

プロセスが失われるとプライマリーおよびレプリカの両方が失われるため、レプリカはプライマリーとして同じプロセスに配置するべきではありません。単一マシンの開発環境においては、2 つのコンテナーを所有でき、それらの間で複製できます。デプロイメント・ポリシーで開発モード属性を定義して、レプリカをプライマリーと同じマシンに配置するよう構成できます。しかし、実動環境では、そのホストを失うことにより両方のコンテナー・サーバーを失うことになるため、単一マシンの使用では不十分です。単一マシンでの開発モードと複数のマシンを使用する実動モード間でモードを変更するには、デプロイメント・ポリシー構成ファイルで開発モードを無効にします。

表 4. 障害のディスカバリーおよび復旧の要約

ロス・タイプ	ディスカバリー (検出) メカニズム	復旧メソッド
プロセス・ロス	入出力	再始動
サーバー・ロス	ハートビート	再始動
ネットワーク障害	ハートビート	ネットワークおよび接続の再確立
サーバー・サイド・ハング	ハートビート	サーバーの停止および再始動
サーバー・ビジー	ハートビート	サーバーが使用可能になるまで待機

可用性向上のためのレプリカ生成

レプリカ生成は、耐障害性を強化し、分散 eXtreme Scale トポロジーのパフォーマンスを向上させます。レプリカ生成は、バックアップ・マップをマップ・セットと関連付けることによって使用可能になります。

マップ・セットについて

マップ・セットは、区画キーによってカテゴリー化されるマップの集まりです。この区画キーは、個別マップのキーから、そのハッシュ・モジュロを取って区画数とすることで派生します。マップ・セット内の 1 つのマップ・グループが区画キー X を持つとすると、それらのマップはデータ・グリッド内の対応する区画 X に保管されます。別のグループが区画キー Y を持つとすると、それらのマップはすべて区画 Y に保管されます。以下同様です。マップ内のデータは、マップ・セットに定義されたポリシーに基づいて複製されます。レプリカ生成は、分散トポロジーで発生します。

マップ・セットには、区画数とレプリカ生成ポリシーが割り当てられます。マップ・セット・レプリカ生成構成は、マップ・セットがプライマリー断片に加えて持つ必要がある同期および非同期のレプリカ断片の数を示します。例えば、1 つの同期レプリカと 1 つの非同期レプリカが存在する場合、マップ・セットに割り当てられたすべての BackingMap は、それぞれ、データ・グリッドの使用可能なコンテナー・サーバー・セット内に自動的に配布されるレプリカ断片を持ちます。また MapSet レプリカ生成構成により、クライアントは同期複製されたサーバーからデータを読み取れるようになります。これにより、読み取り要求の負荷を eXtreme Scale 内のその他のサーバーにも分散することができます。レプリカ生成は、バックアッ

プ・マップのプリロード時にプログラミング・モデルに影響するだけです。

マップのプリロード

マップはローダーに関連付けることができます。ローダーは、オブジェクトがマップに見つからない場合 (キャッシュ・ミスの場合) に、そのオブジェクトをフェッチするためにも、またトランザクションのコミット時に変更をバックエンドに書き込むためにも使用されます。ローダーは、マップへのデータのプリロードに使用することもできます。Loader インターフェースの preloadMap メソッドは、マップ・セット内のその対応する区画がプライマリーとなると、各マップで呼び出されます。preloadMap メソッドは、レプリカでは呼び出されません。このメソッドは、提供されたセッションを使用して、対象となる参照データのすべてをバックエンドからマップにロードしようとします。関係するマップは、preloadMap メソッドに渡される BackingMap 引数によって識別されます。

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

区画に分割されたマップ・セットでのプリロード

マップは、N 個の区画に分割することができます。したがってマップは、複数のサーバーに渡ってストライプすることができます。この場合、各エントリーは、これらのサーバーのうちの 1 つにのみ保管されているキーによって識別されます。アプリケーションは、マップのすべてのエントリーを保持する場合に単一 JVMのヒープ・サイズによる制限を受けなくなるため、非常に大きいマップを eXtreme Scale に保持できるようになります。Loader インターフェースの preloadMap メソッドがプリロードされるアプリケーションは、それがプリロードするデータのサブセットを識別する必要があります。常に、固定数の区画が存在します。この数を判別するには、以下のコード例を使用してください。

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

このコード例は、アプリケーションが、データベースからプリロードするデータのサブセットを識別できることを示しています。アプリケーションは、マップが最初に区画に分割されていない場合でも、これらのメソッドを常に使用しなければなりません。これらのメソッドによって柔軟性が実現されます。管理者が後でマップを区画に分割した場合でも、ローダーは正常に機能し続けます。

アプリケーションは、バックエンドから *myPartition* サブセットを検索する照会を発行する必要があります。テーブル内のデータを簡単に区画に分割できるなんらかの自然な照会がある場合を除き、データベースが使用される場合は、所定レコードの区画 ID の列を持つ方が、処理が容易である可能性があります。

パフォーマンス

プリロードの実装では、複数のオブジェクトを単一トランザクションでマップに保管して、データをバックエンドからマップにコピーします。トランザクションごとに保管されるレコードの最適数は、複雑さやサイズなど、いくつかの要因によって決まります。例えば、トランザクションに 100 エントリーを超えるブロックが含まれると、以後は、エントリーの数を増やすに従ってパフォーマンス利益が減少していきます。 最適数を知るためには、まず 100 エントリーから始めて、徐々に数を増やしていきます。これをパフォーマンス利益がゼロに減少するまで続けます。 トランザクションが大きいほど、レプリカ生成パフォーマンスが向上します。ただ

し、プライマリーのみがプリロード・コードを実行することに注意してください。プリロードされたデータは、プライマリーから、オンラインになっているすべてのレプリカに複製されます。

マップ・セットのプリロード

アプリケーションが複数のマップを持つマップ・セットを使用する場合、各マップはそれぞれ独自のローダーを持ちます。各ローダーに、プリロード・メソッドがあります。各マップは、eXtreme Scale によって順次にロードされます。1 つのマップをプリロード・マップに指定して全マップをプリロードすると、より効率的になる可能性があります。このプロセスは、アプリケーション規則です。例えば、部門と従業員という 2 つのマップが、部門マップと従業員マップの両方をプリロードするために、部門 Loader を使用するとします。このプロシージャーにより、トランザクション上、アプリケーションで部門が必要な場合、その部門の従業員がキャッシュされます。部門 Loader が部門をバックエンドからプリロードするときに、その部門の従業員もフェッチします。その後で、部門オブジェクトとそれに関連する従業員オブジェクトが、単一のトランザクションを使用して、マップに追加されます。

リカバリー可能なプリロード

非常に大きいデータ・セットをキャッシュする必要がある場合があります。このデータのプリロードは、非常に時間がかかる可能性があります。アプリケーションがオンラインになる前に、プリロードを完了しなければならない場合もあります。プリロードをリカバリー可能にすると、便利です。100 万個のレコードをプリロードする必要があるとします。プライマリーがこれらのレコードをプリロードし、800,000 件目のレコードの時点でプライマリーが失敗するとします。通常、新規プライマリーとして選択されたレプリカは、複製状態をクリアして、最初からプリロードを開始します。eXtreme Scale では、ReplicaPreloadController インターフェースを使用できます。アプリケーションのローダーで、ReplicaPreloadController インターフェースを実装する必要が生じることもあります。この例では、単一メソッド `Status checkPreloadStatus(Session session, BackingMap bmap);` をローダーに追加します。Loader インターフェースのプリロード・メソッドが正常に呼び出されるためには、このメソッドが eXtreme Scale ランタイムによって呼び出されます。レプリカがプライマリーにプロモートされると、常に eXtreme Scale がこのメソッド (Status) の結果をテストして、その振る舞いを決定します。

表 5. 状況値および応答

返される状況値	eXtreme Scale の応答
Status.PRELOADED_ALREADY	この状況値は、マップが完全にプリロードされていることを示しているため、eXtreme Scale はプリロード・メソッドをまったく呼び出しません。
Status.FULL_PRELOAD_NEEDED	eXtreme Scale はマップをクリアし、プリロード・メソッドを正常に呼び出します。
Status.PARTIAL_PRELOAD_NEEDED	eXtreme Scale は、マップを現状のままにして、プリロードを呼び出します。このストラテジーによって、アプリケーション・ローダーは、この時点以降プリロードを継続することができます。

プライマリーは、マップのプリロード中、返す必要のある状況をレプリカ側で判別できるように、複製中のマップ・セット内のマップに必ず何らかの状態を残す必要があります。RecoveryMap などと呼ばれる追加のマップを使用することができま

す。マップがプリロード中のデータで一貫して複製されるようにするため、この RecoveryMap は、プリロード中の同じマップ・セットの一部である必要があります。推奨の実装は、以下のとおりです。

プリロードがレコードの各ブロックをコミットすると、プロセスも、RecoveryMap 内のカウンターまたは値をそのトランザクションの一部として更新します。プリロードされたデータと RecoveryMap データは、レプリカにアトミックに複製されます。レプリカがプライマリーに格上げされると、RecoveryMap をチェックして何が起こったかを確認できるようになります。

RecoveryMap は、状態キーを持つ単一エントリーを保持できます。このキーに対するオブジェクトが存在しない場合には、完全な preload (checkPreloadStatus returns FULL_PRELOAD_NEEDED) が必要です。この状態キーに対するオブジェクトが存在し、値が COMPLETE の場合は、プリロードが完了し、checkPreloadStatus メソッドで PRELOADED_ALREADY が返されます。これ以外の場合、値オブジェクトは、プリロードを再開する場所を示し、checkPreloadStatus メソッドは PARTIAL_PRELOAD_NEEDED を返します。ローダーは、プリロードが呼び出されたときにローダーに開始点がわかるように、ローダーのインスタンス変数にリカバリー・ポイントを保管できます。また、各マップが個別にプリロードされる場合、RecoveryMap もマップごとにエントリーを保持できます。

Loader での同期レプリカ生成モードにおけるリカバリーの処理

eXtreme Scale ランタイムは、プライマリーが失敗したときにコミット済みデータを失わないよう設計されています。次のセクションでは、使用されるアルゴリズムについて説明します。これらのアルゴリズムは、レプリカ生成グループが同期レプリカ生成を使用する場合にのみ適用されます。ローダーはオプションです。

eXtreme Scale ランタイムは、すべての変更がプライマリーからレプリカに同期複製されるように構成することができます。同期レプリカが配置されると、その同期レプリカは、プライマリー断片にある既存データのコピーを受け取ります。この間もプライマリーはトランザクションを受け取り続け、受け取ったトランザクションを非同期にレプリカにコピーします。レプリカはこの時点ではオンラインであるとは見なされません。

レプリカがプライマリーに追いついた後、レプリカはピア・モードに入り、同期レプリカ生成が始まります。プライマリーでコミットされたトランザクションはすべて同期レプリカに送信され、プライマリーは各レプリカからの応答を待ちます。ローダーを使用する、プライマリーでの同期コミット・シーケンスは、以下の一連のステップのようになります。

表 6. プライマリーでのコミット・シーケンス

Loader を使用する場合のステップ	**Loader を使用しない場合のステップ**
エントリーのロックを取得します。	同じ
変更をローダーにフラッシュします。	操作しない
キャッシュに変更を保存します。	同じ
変更をレプリカに送信し、確認応答を待機します。	同じ

表 6. プライマリーでのコミット・シーケンス (続き)

Loader を使用する場合のステップ	**Loader を使用しない場合のステップ**
TransactionCallback プラグインでローダーをコミットします。	プラグイン・コミットが呼び出されますが、何も実行しません。
エントリーのロックを解除します。	同じ

変更がレプリカに送信された後、ローダーにコミットされることに注意してください。変更がレプリカでコミットされる条件を判別するには、このシーケンスを訂正します。初期化時に、以下のようにプライマリーで tx リストを初期化します。

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

同期コミットの処理中に、以下のシーケンスを使用します。

表 7. 同期コミット処理

Loader を使用する場合のステップ	**Loader を使用しない場合のステップ**
エントリーのロックを取得します。	同じ
変更をローダーにフラッシュします。	操作しない
キャッシュに変更を保存します。	同じ
コミット済みトランザクションで変更を送信し、トランザクションをレプリカにロールバックし、確認応答を待機します。	同じ
コミット済みトランザクションおよびロールバック済みトランザクションのリストをクリアします。	同じ
TransactionCallBack プラグインでローダーをコミットします。	TransactionCallBack プラグイン・コミットがやはり呼び出されますが、通常、何も行われません。
コミットが成功した場合、トランザクションがコミット済みトランザクションに追加され、成功しなかった場合はロールバック済みトランザクションに追加されます。	操作しない
エントリーのロックを解除します。	同じ

レプリカ処理の場合、以下のシーケンスを使用します。

1. レプリカが変更されます。
2. コミット済みトランザクション・リスト内のすべての受信済みトランザクションをコミットします。
3. ロールバック済みトランザクション・リスト内のすべての受信済みトランザクションをロールバックします。
4. トランザクションまたはセッションを開始します。
5. トランザクションまたはセッションに変更を適用します。
6. 保留リストにトランザクションまたはセッションを保存します。
7. 応答を返信します。

レプリカがレプリカ・モードである間は、レプリカ上でローダーによる相互作用が行われないことに注意してください。プライマリーは、すべての変更を Loader を介してプッシュする必要があります。レプリカは変更をプッシュしません。このアルゴリズムの副次作用は、レプリカに常にトランザクションがあるが、次のプライマリー・トランザクションによってこれらのトランザクションのコミット状況が送信されるまで、コミットされないことです。その場合には、トランザクションはレプリカ上でコミットまたはロールバックされます。このようになるまでは、トランザクションはコミットされません。短い時間 (数秒) 後にトランザクションの結果が送信されるようなタイマーをプライマリーに追加することができます。このタイマーは、その時刻ウィンドウに対する失効性を制限しますが、除去はしません。こうした失効性は、レプリカ読み取りモードを使用する場合のみの問題です。それ以外の点では、失効性は、アプリケーションに影響を与えません。

プライマリーが失敗した場合、プライマリーでコミットまたはロールバックされたトランザクションがいくつかある可能性がありますが、これらの結果が含まれるメッセージがレプリカに到達しませんでした。レプリカが新規プライマリーにプロモートされる際の最初のアクションの 1 つは、この状態に対処することです。保留中の各トランザクションは、新規プライマリーのマップ・セットに対して再処理されます。ローダーがある場合は、そのローダーに各トランザクションが送られます。これらのトランザクションには、厳密な先入れ先出し法 (FIFO) 順序が適用されます。失敗したトランザクションは無視されます。例えば、3 つのトランザクション A、B、および C が保留中の場合、A はコミットし、B はロールバックし、C もコミットする可能性があります。1 つのトランザクションが他のトランザクションに影響を与えることはありません。これらのトランザクションは独立したものと見なされます。

ローダーで使用されるロジックは、フェイルオーバー・リカバリー・モードと通常モードの場合では若干異なることがあります。ローダーがフェイルオーバー・リカバリー・モードであるときは、ReplicaPreloadController インターフェースを実装することで容易に識別できます。checkPreloadStatus メソッドは、フェイルオーバー・リカバリーが完了した場合にのみ呼び出されます。このため、Loader インターフェースの apply メソッドが checkPreloadStatus メソッドより前に呼び出される場合は、リカバリー・トランザクションになります。checkPreloadStatus メソッドが呼び出されると、フェイルオーバー・リカバリーが完了します。

レプリカ間のロード・バランシング

特に構成されていない限り、eXtreme Scale は、すべての読み取り要求と書き込み要求を指定されたレプリカ生成グループのプライマリー・サーバーに送信します。プライマリーは、クライアントからのすべての要求にサービスを提供する必要があります。読み取り要求をプライマリーのレプリカに送信できるようにするとよいでしょう。読み取り要求をレプリカに送信することにより、読み取り要求の負荷を複数の Java 仮想マシン (JVM) で共有できるようになります。ただし、読み取り要求のためにレプリカを使用すると、応答が不整合になる可能性があります。

レプリカ間のロード・バランシングは、通常、クライアントが常時変更されるデータをキャッシュしているか、またはクライアントがペシミスティック・ロックを使用している場合にのみ使用されます。

データが絶えず変更され、そのためクライアントのニア・キャッシュで無効化された場合は、結果としてクライアントからプライマリーへの get 要求率が比較的高くなります。同様に、ペシミスティック・ロック・モードでは、ローカル・キャッシュが存在しないため、すべての要求がプライマリーに送信されます。

データが比較的静的であるか、またはペシミスティック・モードが使用されていない場合には、読み取り要求をレプリカへ送信しても、パフォーマンスにそれほど大きな影響を与えません。データで満杯のキャッシュを持つクライアントからの get 要求の頻度は、高くありません。

クライアントが始動されたばかりのときには、ニア・キャッシュは空です。空のキャッシュに対するキャッシュ要求は、プライマリーに転送されます。時間が経過してクライアント・キャッシュにデータが入れられると、要求ロードは除去されます。多数のクライアントが同時に始動する場合は、ロード (負荷) が重大な影響を及ぼす可能性があります。そのため、レプリカ読み取りがパフォーマンス上適切な選択になることがあります。

クライアント・サイドのレプリカ生成

eXtreme Scale により、非同期レプリカ生成を使用して、サーバー・マップを 1 つ以上のクライアントに複製することができます。クライアントは ClientReplicableMap.enableClientReplication メソッドを使用して、サーバー・サイド・マップのローカルの読み取り専用コピーを要求できます。

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

最初のパラメーターはレプリカ生成モードです。このモードには、連続レプリカ生成またはスナップショット・レプリカ生成を指定できます。2 番目のパラメーターは、データの複製元の区画を表す区画 ID の配列です。この値がヌルの場合、または空の配列の場合、データはすべての区画から複製されます。最後のパラメーターは、クライアント・レプリカ生成イベントを受信するためのリスナーです。詳しくは、API 資料の ClientReplicableMap および ReplicationMapListener を参照してください。

レプリカ生成が有効になると、サーバーはクライアントへのマップの複製を開始します。結局のところ、クライアントは、どの時点においてもわずか数トランザクションでサーバーに到達します。

高可用性カタログ・サービス

カタログ・サービス・ドメインは、使用しているカタログ・サーバーのデータ・グリッドであり、eXtreme Scale 環境内のすべてのコンテナーのトポロジー情報を保持します。カタログ・サービスは、すべてのクライアントの平衡化とルーティングを制御します。 eXtreme Scale をメモリー内のデータベース処理スペースとしてデプロイするには、高可用性を実現するためにカタログ・サービスをカタログ・サービス・ドメインにクラスター化する必要があります。

カタログ・サービス・ドメインのコンポーネント

複数のカタログ・サーバーが始動すると、サーバーのいずれか 1 つがマスター・カタログ・サーバーとして選択されます。マスター・カタログ・サーバーは、Internet

Inter-ORB Protocol (IIOP) ハートビートを受信し、カタログ・サービスまたはコンテナーの変更に応じて、システム・データの変更を処理します。

クライアントがいずれかのカタログ・サーバーにアクセスすると、カタログ・サービス・ドメインのルーティング・テーブルは、共通オブジェクト・リクエスト・ブローカー・アーキテクチャー (CORBA) サービス・コンテキストを通してクライアントに伝搬されます。

少なくとも 3 つのカタログ・サーバーを構成します。後日、サーバーをシームレスにアップグレードできるようにするには、カタログ・サーバーは、コンテナー・サーバーとは別個のノードまたは別個のインストール・イメージ上にインストールする必要があります。構成がゾーンに分かれている場合、ゾーンごとに 1 つずつカタログ・サーバーを構成することができます。

eXtreme Scale サーバーおよびコンテナーが、カタログ・サーバーの 1 つにアクセスすると、カタログ・サービス・ドメインのルーティング・テーブルが、CORBA サービス・コンテキストを通して eXtreme Scale サーバーおよびコンテナーにも伝搬されます。また、アクセスされたカタログ・サーバーがその時点でマスター・カタログ・サーバーでなかった場合、要求は現行マスター・カタログ・サーバーに自動的に転送され、カタログ・サーバーのルーティング・テーブルも更新されます。

注: カタログ・サービス・ドメインとコンテナー・サーバー・データ・グリッドは、まったく別のものです。カタログ・サービス・ドメインは、システム・データの高可用性のためのものです。コンテナー・サーバーのデータ・グリッドは、ユーザーのデータの高可用性、スケーラビリティー、およびワークロード管理を目的としています。したがって、カタログ・サービス・ドメインのルーティング・テーブルとコンテナー・サーバー・データ・グリッド断片のルーティング・テーブルという 2 つの異なるルーティング・テーブルが存在します。

カタログ・サービス・ドメインの責務は、以下の一連のサービスに分割されます。

コア・グループ・マネージャー

カタログ・サービスは、高可用性マネージャー (HA マネージャー) を使用して、可用性モニタリングのためにプロセスをグループ化します。各プロセス・グループが、コア・グループです。eXtreme Scale では、コア・グループ・マネージャーが動的にプロセスをグループ化します。これらのプロセスは、スケーラビリティーのために小さく維持されます。各コア・グループはそれぞれリーダーを選出します。リーダーには、個々のメンバーに障害が発生したときに状況をコア・グループ・マネージャーに送信するという責任が追加されます。同じ状況のメカニズムは、グループのすべてのメンバーで障害が起こったときに (これにより、リーダーとの通信に障害が発生します)、それを検出するために使用されます。

コア・グループ・マネージャーは完全に自動化されたサービスです。コンテナーを少数のサーバーからなるグループに編成する責務を持ち、そのグループは自動で緩やかに統合して ObjectGrid を形成します。コンテナーは、カタログ・サービスへの初回接続時、新規または既存のグループに割り当てられるまで待機します。eXtreme Scale デプロイメントはそうした多くのグループから構成されており、このグループ化は重要なスケーラビリティー・イネーブラーです。 各グループは Java 仮想マシンで構成されるグループで

あり、ハートビートを使用して、他のグループの可用性をモニターします。これらのグループ・メンバーの 1 つがリーダーに選出され、リーダーには可用性情報をカタログ・サービスにリレーする責務が追加され、再割り振りとルート転送により障害に対処できるようにします。

配置サービス
カタログ・サービスは、使用可能なすべてのコンテナー・サーバーでの断片の配置を管理します。物理リソース間でのバランスの維持は、配置サービスの担当です。配置サービスは、個々の断片をホスト・コンテナーに割り振る責任を担います。配置サービスは、データ・グリッド内で N 個の中から 1 つ選ばれたサービスとして実行されるため、サービスのインスタンスは 1 つだけが実行されます。 そのインスタンスに障害が起こると、別のプロセスが選出され、それが引き継ぎます。予備のために、カタログ・サービスの状態は、カタログ・サービスをホスティングするすべてのサーバーに複製されます。

管理 カタログ・サービスは、システム管理のための論理的なエントリー・ポイントでもあります。カタログ・サービスは、Managed Bean (MBean) をホストし、カタログ・サービスが管理しているすべてのサーバーの Java Management Extensions (JMX) URL を提供します。

ロケーション・サービス
ロケーション・サービスは、探しているアプリケーションをホストするコンテナーを検索しているクライアント、およびホストされるアプリケーションを配置サービスに登録しようとしているコンテナー・サーバーを検索しているクライアントの両方に対し、タッチ・ポイントとしての役割を果たします。ロケーション・サービスは、この機能をスケールアウトするために、すべてのデータ・グリッド・メンバーで実行されます。

カタログ・サービス・ドメインのデプロイメント

カタログ・サービスは、一般的に定常状態でアイドルになるロジックをホストします。その結果として、カタログ・サービスがスケーラビリティーに与える影響はごくわずかです。サービスは、同時に使用可能になる多数のコンテナーにサービスを提供するために作成されています。可用性のために、カタログ・サービスをデータ・グリッドに構成します。

計画

カタログ・サービス・ドメインが開始されると、データ・グリッドのメンバーは相互にバインドされます。カタログ・サービス・ドメイン構成を実行時に変更することはできないので、カタログ・サービス・ドメインのトポロジーは慎重に計画してください。エラー防止のため、データ・グリッドはできるだけ広範囲に分散させてください。

カタログ・サービス・ドメインの開始

カタログ・サービス・ドメインの作成について詳しくは、*管理ガイド*にあるカタログ・サービス・ドメインの開始についての説明を参照してください。

WebSphere Application Server に組み込まれている eXtreme Scale コンテナーのスタンドアロン・カタログ・サービス・ドメインへの接続

WebSphere Application Server 環境に組み込まれている eXtreme Scale コンテナーを構成して、スタンドアロン・カタログ・サービス・ドメインに接続できます。

- カタログ・サービス・ドメインは WebSphere Application Server 管理コンソールで作成できます。詳しくは、WebSphere Application Server でのカタログ・サービス・ドメインの作成「*管理ガイド*」でカタログ・サービス・ドメインの作成についての説明を参照してください。

-  (非推奨) 過去のリリースでは、カスタム・プロパティーを作成することによって、カタログ・サービスをカタログ・サービス・ドメインに接続しました。 このプロパティーをそのまま使用することはできますが、推奨されません。 このカスタム・プロパティーについて詳しくは、WebSphere Application Server 環境でのサーバーの開始と停止 「*管理ガイド*」にある WebSphere Application Server環境でのカタログ・サービス・プロセスの開始に関する情報を参照してください。

  注: サーバー名の競合: このプロパティーは、eXtreme Scale カタログ・サーバーの開始だけでなく、そこに接続する目的でも使用されるため、カタログ・サーバーの名前をどの WebSphere Application Server とも同じ名前にすることはできません。

詳しくは、『カタログ・サーバー・クォーラム』を参照してください。

カタログ・サーバー・クォーラム

クォーラム・メカニズムが有効である場合、クォーラム内のすべてのカタログ・サーバーは、データ・グリッドで配置操作を行うために使用可能でなければなりません。

- 『重要な用語』
- 111 ページの『ハートビートおよび障害検出』
- 112 ページの『クォーラムの振る舞い』
  - 115 ページの『クォーラム損失時のコンテナーの振る舞い』
- 115 ページの『クォーラム損失時のクライアントの振る舞い』

重要な用語

- **ハートビート**: サーバーが実行中であることを伝えるために、サーバー間で送信されるシグナル。
- **クォーラム**: データ・グリッドで配置操作をやりとりし、実行するカタログ・サーバーのグループ。手動でクォーラム・メカニズムを管理アクションでオーバーライドしていない限り、このグループは、データ・グリッド内のすべてのカタログ・サーバーで構成されます。
- **ブラウン・アウト:** 1 つ以上のサーバー間における一時的な接続喪失。
- **ブラック・アウト:** 1 つ以上のサーバー間における完全な接続喪失。
- **データ・センター:** 地理的に配置されたサーバーの一群のことで、通常はサーバー同士がローカル・エリア・ネットワーク (LAN) で接続されています。

- **ゾーン:** ゾーンとは、何らかの物理的特性を共有するサーバーを 1 つのグループにまとめるために使用される構成オプションのことです。サーバーのグループに対するゾーンの例として、データ・センター、エリア・ネットワーク、ビル、またはビルの 1 フロアなどがあります。

ハートビートおよび障害検出

コンテナー・サーバーおよびコア・グループ

カタログ・サービスはコンテナー・サーバーを限られたサイズのコア・グループに配置します。コア・グループは、そのメンバーの障害を検出しようと試みます。コア・グループのメンバーの中から 1 つだけがコア・グループ・リーダーに選ばれます。コア・グループ・リーダーは、コア・グループが活動中であることをカタログ・サービスに定期的に知らせるとともに、メンバーシップの変更をカタログ・サービスに報告します。メンバーシップの変更としては、JVM の障害や、コア・グループに参加する、新しく追加された JVM などが考えられます。

JVM ソケットが閉じられていると、その JVM はもはや使用できないと見なされます。さらに、各コア・グループ・メンバーは構成によって決定されたペースでこれらのソケットをハートビートします。 JVM が構成された最大時間内にこれらのハートビートに応答しないと、その JVM はもはや使用できないと見なされ、障害検出がトリガーされます。

カタログ・サービスがあるコンテナー JVM を障害と判断した後で、そのコンテナー・サーバーが使用可能であると報告された場合は、このコンテナー JVM に対して WebSphere eXtreme Scale コンテナー・サーバーをシャットダウンするよう指示が出されます。この状態の JVM は **xscmd** ユーティリティー・コマンド照会では不可視です。コンテナー JVM のログのメッセージは、コンテナー JVM が失敗したことを示します。これらの JVM は、手動で再始動する必要があります。

コア・グループ・リーダーがいずれかのメンバーと連絡できないと、コア・グループ・リーダーはこのメンバーへの連絡を再試行し続けます。

コア・グループの全メンバーが完全に障害を起こす可能性もあります。コア・グループ全体で障害が起こった場合は、カタログ・サービスがその障害を検出しなければなりません。

カタログ・サービス・ドメインのハートビート

カタログ・サービス・ドメインは静的メンバーシップおよびクォーラム・メカニズムを持つ専用コア・グループに似ています。そして、通常のコア・グループと同じ方法で障害検出を行います。ただし、その振る舞いはクォーラム・ロジックを含むように変更されています。さらに、カタログ・サービスではそれほど活発でないハートビートの構成が使用されます。

障害検出

WebSphere eXtreme Scale は、異常ソケット閉鎖イベントを通じていつプロセスが終了したか検出します。プロセスが終了すると、そのことが直ちにカタログ・サービスに知らされます。

ハートビートの構成について詳しくは、フェイルオーバー検出のためのハートビート間隔設定のチューニング「*管理ガイド*」のフェイルオーバー検出の構成に関する情報を参照してください。

クォーラムの振る舞い

通常、カタログ・サービスのメンバーは完全な接続性を備えています。カタログ・サービス・ドメインは JVM の静的集合です。WebSphere eXtreme Scale は、カタログ・サービスのすべてのメンバーがオンラインであることを想定しています。すべてのメンバーがオンラインであるとき、そのカタログ・サービスはクォーラムを持っています。カタログ・サービスは、カタログ・サービスがクォーラムを持っている間だけコンテナー・イベントに応答します。

クォーラム損失の理由

WebSphere eXtreme Scale は、以下のシナリオによってクォーラムの損失を予想します。

- カタログ・サービス JVM メンバーの障害
- ネットワーク・ブラウン・アウトの発生
- データ・センター損失の発生

WebSphere eXtreme Scale は、以下のシナリオではクォーラムを失いません。

- **stopOgServer** コマンドや他のすべての管理アクションを使用したカタログ・サーバー・インスタンスの停止。システムはこのサーバー・インスタンスが停止したことを知っており、これは JVM 障害やブラウン・アウトとは異なります。

カタログ・サービスがクォーラムを失うと、カタログ・サービスはクォーラムが再確立されるのを待ちます。カタログ・サービスは、クォーラムを持っていない間は、コンテナー・サーバーからのイベントを無視します。コンテナー・サーバーは、このときにカタログ・サーバーが拒否したすべての要求を再試行し続けます。ハートビートは、クォーラムが再確立されるまで中断状態となります。

JVM 障害によるクォーラム損失

障害を起こしたカタログ・サーバーはクォーラム損失の原因となります。JVM に障害が発生した場合、できるだけ迅速にクォーラムをオーバーライドする必要があります。障害を起こしたカタログ・サービスは、クォーラムがオーバーライドされるまで、データ・グリッドに再参加できません。

ネットワーク・ブラウン・アウトによるクォーラム損失

WebSphere eXtreme Scale はブラウン・アウトの可能性を予想できる設計になっています。ブラウン・アウトとは、データ・センター間の接続が一時的に失われた状態をいいます。ブラウン・アウトは通常一時的なもので、数秒または数分で解消されます。ブラウン・アウト中、WebSphere eXtreme Scale は通常動作の維持を試みますが、ブラウン・アウトは 1 つの障害イベントと見なされます。この障害は修正されることが想定されており、アクションを必要とせずに通常動作が再開されます。

長時間に及ぶブラウン・アウトは、ユーザーの介入がある場合にのみブラック・アウトに分類できます。このイベントをブラック・アウトに分類するためには、ブラウン・アウトの一方の側でクォーラムをオーバーライドする必要があります。

カタログ・サービス JVM の循環

stopOgServer コマンドを使用してカタログ・サーバーが停止された場合は、クォーラムを持つサーバーが 1 つ減少します。残りのサーバーはまだクォーラムを持っています。このカタログ・サーバーを再始動すると、クォーラムは元の数に戻ります。

クォーラム損失の影響

クォーラムが失われると同時にコンテナー JVM が障害を起こした場合は、ブラウン・アウトが回復するまでリカバリーは行われません。ブラック・アウト・シナリオの場合は、お客様がクォーラム・オーバーライド・コマンドを実行するまでリカバリーは行われません。クォーラム損失とコンテナー障害で二重障害と見なされますが、これはまれにしか起こりません。二重障害のために、アプリケーションは障害を起こした JVM に保管されたデータへの書き込みアクセスを失う可能性があります。クォーラムが復元されると、通常のリカバリーが行われます。

同様に、クォーラム損失イベントの発生中にコンテナーを開始しようとしても、コンテナーは開始されません。

クォーラムの損失中に完全クライアント接続が許可されます。クォーラム損失イベント中にコンテナー障害も接続問題も起こらなければ、クライアントはコンテナー・サーバーと完全に対話することができます。

ブラウン・アウトが発生すると、クライアントによっては、ブラウン・アウトが解消されるまで、データのプライマリーまたはレプリカ・コピーにアクセスできない場合があります。

カタログ・サービス JVM は各データ・センターに存在しなければならないため、新規のクライアントを開始することができます。したがって、ブラウン・アウト・イベント中でもクライアントが少なくとも 1 つのカタログ・サービスには到達できます。

クォーラムのリカバリー

何らかの理由によってクォーラムが失われた場合は、クォーラムが再確立される際、リカバリー・プロトコルが実行されます。クォーラム損失イベントが発生すると、コア・グループに対する活性検査はすべて中断され、障害報告も無視されます。クォーラムが元に戻ると、カタログ・サービスはすべてのコア・グループをチェックして、直ちにそれらのメンバーシップを判別します。障害として報告されたコンテナー JVM でそれまでホストされていた断片は、いずれもリカバリーされます。プライマリー断片が失われると、残っているレプリカがプライマリー断片にプロモートされます。レプリカ断片が失われた場合は、残存物の上に追加のレプリカ断片が作成されます。

クォーラムのオーバーライド

クォーラムのオーバーライドは、データ・センター障害が発生した場合にのみ行ってください。カタログ・サービス JVM の障害またはネットワーク・ブラウン・アウトのためにクォーラムが失われた場合は、カタログ・サービス JVM が再始動されるか、またはネットワークのブラウン・アウトが終わると、リカバリーが自動的に行われます。

データ・センターの障害に通じているのは管理者のみです。WebSphere eXtreme Scale はブラウン・アウトとブラック・アウトを同様に扱います。このような障害の WebSphere eXtreme Scale 環境を、**xscmd -c overrideQuorum** コマンドを使用して知らせる必要があります。このコマンドは、カタログ・サービスに対して、現在のメンバーシップでクォーラムが達成されていると想定するように指示し、完全リカバリーが行われます。クォーラム・オーバーライド・コマンドを発行したときは、障害のあるデータ・センター内の JVM が実際に障害を起こしていて、しかもリカバリーする見込みがないことを保証していることになります。

以下のリストは、クォーラムのオーバーライドに関するいくつかのシナリオを考慮したものです。このシナリオでは、A、B、および C という 3 つのカタログ・サーバーがあるとします。

- **ブラウン・アウト:** C カタログ・サーバーは、一時的に分離されます。カタログ・サービスは、クォーラムを失い、ブラウン・アウトが終了するのを待機します。ブラウン・アウトが終了したら、C カタログ・サービスはカタログ・サービス・ドメインに再参加し、クォーラムが再確立されます。この間、アプリケーションはいかなる問題も検出しません。
- **一時障害:** 一時障害時に、C カタログ・サーバーが障害を起こし、カタログ・サービスがクォーラムを失います。クォーラムをオーバーライドする必要があります。クォーラムが再確立されたら、C カタログ・サーバーを再始動できます。C カタログ・サーバーは、再始動すると、カタログ・サービス・ドメインに再度参加します。この間、アプリケーションはいかなる問題も検出しません。
- **データ・センター障害:** データ・センターが障害を起こし、しかもネットワーク上で分離されていることを確認します。次に、**xscmd -c overrideQuorum** コマンドを発行します。そうすると、残存している 2 つのデータ・センターが、障害を起こしたデータ・センターでホストされていた断片を置き換えることによって完全リカバリーを実行します。カタログ・サービスは、現在、A および B カタログ・サーバーのフル・クォーラムで実行されています。アプリケーションは、ブラック・アウトが始まってからクォーラムがオーバーライドされるまでの間に、遅延や例外を検出することがあります。クォーラムがオーバーライドされると、データ・グリッドがリカバリーし、通常動作が再開されます。
- **データ・センター・リカバリー:** 残存しているデータ・センターは、オーバーライドされたクォーラムで既に実行されています。 C カタログ・サーバーを含むデータ・センターが再始動された場合、そのデータ・センターにあるすべての JVM も再始動する必要があります。そうすると、C カタログ・サーバーは既存のカタログ・サービス・ドメインに再参加し、クォーラム設定はユーザーの介入なしで通常状態に戻ります。
- **データ・センターの障害およびブラウン・アウト:** C カタログ・サーバーを含むデータ・センターで障害が起こります。残りのデータ・センターでは、クォーラムがオーバーライドされてリカバリーされます。A カタログ・サーバーと B カタログ・サーバーとの間でブラウン・アウトが生じると、通常のブラウン・アウ

ト・リカバリー・ルールが適用されます。ブラウン・アウトが解消されると、クォーラムが再確立され、クォーラム損失からの必要なリカバリーが実行されます。

クォーラム損失時のコンテナーの振る舞い

コンテナーは 1 つ以上の断片をホストします。断片は、特定の区画のプライマリーであるかレプリカであるか、そのいずれかです。カタログ・サービスが断片をコンテナーに割り当てると、カタログ・サービスから新しい指示が送られてくるまで、コンテナー・サーバーはこの割り当てを使用します。例えば、ネットワークのブラウン・アウト時にプライマリー断片は、カタログ・サービスがプライマリー断片にさらに指示を与えるまでそのレプリカ断片と通信を試み続けます。

同期レプリカの振る舞い

接続が中断されている間でも、オンラインであるレプリカの数が少なくともマップ・セットの **minsync** プロパティー値である場合、プライマリー断片は新しいトランザクションを受け入れることができます。同期レプリカへのリンクが中断されているときにプライマリー断片で新しいトランザクションが処理された場合は、リンクが再確立された時点で、レプリカはプライマリーの現在の状態と再同期されます。

データ・センター間や WAN スタイルのリンクに対しては、同期レプリカ生成を構成しないでください。

非同期レプリカの振る舞い

接続が中断されている間、プライマリー断片は新しいトランザクションを受け入れることができます。プライマリー断片は変更内容を限界までバッファーに入れます。この限界に達する前にレプリカとの接続が再確立された場合は、バッファーに入れられた変更内容を使用してレプリカが更新されます。限界に達すると、プライマリーはバッファーに入れられたリストを破棄します。そしてレプリカが再接続されると、レプリカはクリアされて再同期されます。

クォーラム損失時のクライアントの振る舞い

カタログ・サービス・ドメインがクォーラムを持っていてもいなくても、常時、クライアントはカタログ・サーバーに接続して、データ・グリッドをブートストラップすることができます。クライアントは、経路テーブルを取得した上でデータ・グリッドと対話するために、カタログ・サーバー・インスタンスへの接続を試みます。ネットワークの接続性により、ネットワーク・セットアップが原因でクライアントが一部の区画と対話できなくなる場合があります。クライアントはローカル・レプリカに接続してリモート・データを取得できますが、その場合はクライアントがそうするように構成されていなければなりません。クライアントは、該当するプライマリー区画が使用可能でない場合、データを更新できません。

レプリカおよび断片

eXtreme Scale を使用すると、メモリー内のデータベースまたは断片を、Java 仮想マシン (JVM) 相互間で複製することができます。断片は、コンテナー上に配置された区画を表します。異なる区画を表す複数の断片が、単一のコンテナー上に存在す

ることができます。各区画にはインスタンスがあり、そのインスタンスは、プライマリー断片と構成可能な複数のレプリカ断片です。 レプリカ断片は、同期または非同期のいずれかです。レプリカ断片のタイプと配置は、eXtreme Scale により、同期断片および非同期断片の最小数と最大数を指定するデプロイメント・ポリシーを使用して決定されます。

断片タイプ

レプリカ生成では、次の 3 つのタイプの断片が使用されます。

- プライマリー
- 同期レプリカ
- 非同期レプリカ

プライマリー断片は、挿入、更新、および除去の各操作をすべて受信します。プライマリー断片は、レプリカの追加と除去を行い、データをレプリカに対して複製し、トランザクションのコミットとロールバックを管理します。

同期レプリカは、プライマリーと同じ状態を保持します。プライマリーがデータを同期レプリカに対して複製する場合、トランザクションは、同期レプリカ上でコミットするまで、コミットされません。

非同期レプリカは、プライマリーと同じ状態である場合も、同じ状態でない場合もあります。プライマリーがデータを非同期レプリカに対して複製する場合、プライマリーは、非同期レプリカがコミットするのを待機しません。

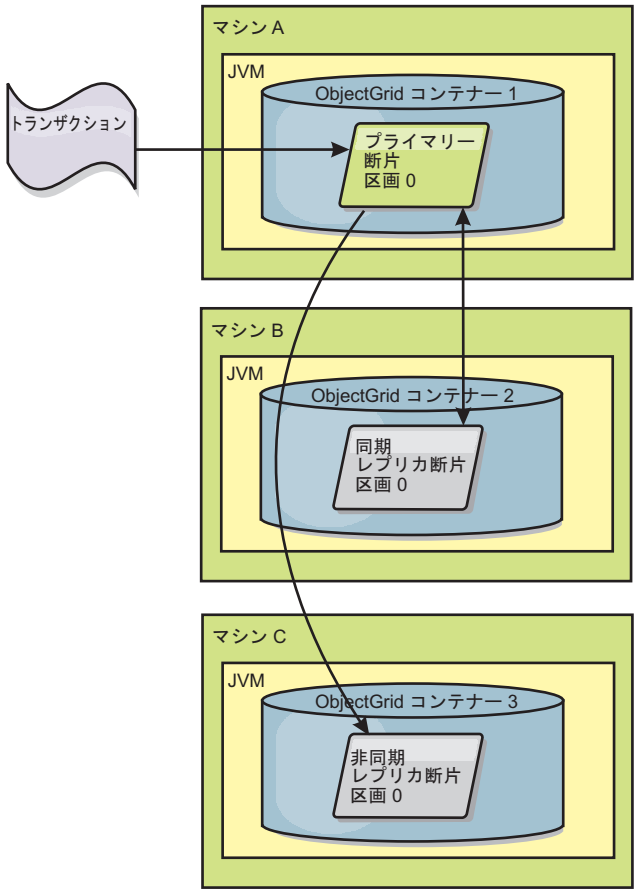


図 30. プライマリー断片とレプリカ断片との間の通信パス

最小同期レプリカ断片数

プライマリーは、データのコミットを準備するときに、トランザクションのコミットを断定した同期レプリカ断片の数を検査します。トランザクションがレプリカに対して通常に処理を行う場合は、コミットを断定します。同期レプリカに何らかの異常がある場合は、コミットしないことを断定します。プライマリーがコミットする前に、コミットを断定している同期レプリカ断片の数が、デプロイメント・ポリシーの minSyncReplica 設定に適合している必要があります。コミットを断定している同期レプリカ断片の数が小さ過ぎる場合、プライマリーはトランザクションをコミットせず、エラーとなります。このアクションにより、正しいデータには、必要な数の同期レプリカが必ず使用可能となります。エラーを検出した同期レプリカは、再登録して、その状態を修正します。再登録について詳しくは、レプリカ断片のリカバリーを参照してください。

コミットを断定した同期レプリカの数が少な過ぎる場合、プライマリーは、ReplicationVotedToRollbackTransactionException エラーをスローします。

レプリカ生成およびローダー

通常、プライマリー断片は、変更を、ローダーを介して同期的にデータベースに書き込みます。ローダーとデータベースは、常に同期しています。プライマリーがレプリカ断片にフェイルオーバーする場合、データベースとローダーは同期しない場合があります。以下に例を示します。

- プライマリーは、トランザクションをレプリカに送信してから、データベースに対してコミットする前に、失敗する場合があります。
- プライマリーは、データベースに対してコミットしてから、レプリカに送信する前に、失敗する場合があります。

どちらの場合も、レプリカが、1 トランザクションだけデータベースの前に、または 1 トランザクションだけデータベースの後ろに移動します。この状態は許容されません。eXtreme Scale は、ローダー実装に特殊なプロトコルと契約を使用して、2 フェーズ・コミットを使用せずにこの問題を解決します。プロトコルは、次のようになります。

プライマリー・サイド
- トランザクションを、前のトランザクション結果と一緒に送信します。
- データベースに書き込み、トランザクションのコミットを試行します。
- データベースがコミットする場合、eXtreme Scale 上でコミットします。データベースがコミットしない場合には、トランザクションをロールバックします。
- 結果を記録します。

レプリカ・サイド
- トランザクションを受信し、それをバッファーに入れます。
- すべての結果について、トランザクションと一緒に送信し、バッファーに入れられたすべてのトランザクションをコミットし、ロールバックされたすべてのトランザクションを廃棄します。

フェイルオーバーする場合のレプリカ・サイド
- バッファーに入れられたすべてのトランザクションについて、トランザクションをローダーに提供し、ローダーはトランザクションのコミットを試行します。
- 各トランザクションがべき等になるようにローダーを作成する必要があります。
- トランザクションが既にデータベース内にある場合は、ローダーはいかなる操作も行いません。
- トランザクションがデータベース内にない場合には、ローダーはトランザクションを適用します。
- トランザクションがすべて処理されてから、新しいプライマリーが要求のサービス提供を開始できます。

このプロトコルにより、データベースは、確実に新規プライマリー状態と同じレベルとなります。

断片配置

カタログ・サービスは断片を配置します。各 ObjectGrid には複数の区画があり、区画ごとに、プライマリー断片、およびオプションのレプリカ断片セットがあります。カタログ・サービスは、断片が使用可能なコンテナー・サーバーに均等に分散されるようにそのバランスを取って、断片を割り振ります。構成が開発モードでない限り、同じ区画のレプリカおよびプライマリー断片を同じコンテナー・サーバー上、つまり、同じ IP アドレス上に配置しません。

新規コンテナー・サーバーが始動すると、eXtreme Scale は、比較的負荷の多いコンテナー・サーバーから断片を取り出して、この新しい空のコンテナー・サーバーに入れます。この断片の移動により、水平スケーリングが可能になります。

スケールアウト

スケールアウトとは、追加のコンテナー・サーバーがデータ・グリッドに追加された場合に、eXtreme Scale が、プライマリーであれレプリカであれ、既存の断片を古いコンテナー・サーバーのセットから新しいセットに移動しようとすることです。この移動により、データ・グリッドを拡張して、新規に追加されたコンテナー・サーバーのプロセッサー、ネットワーク、およびメモリーを利用できるようになります。また、この移動は、データ・グリッドのバランスを取り、データ・グリッド内の各 JVM がホストするデータ量が等しくなるようにします。データ・グリッドの拡張にともなって、グリッド全体のうち各サーバーがホストするサブセットは小さくなります。eXtreme Scale は、区画間でデータが均等に分散していると想定します。この拡張により、スケールアウトが可能になります。

スケールイン

スケールインとは、JVM の 1 つに障害が起こった場合に、そのダメージを eXtreme Scale が修復しようとすることです。障害が発生した JVM にレプリカがあった場合、eXtreme Scale は、残っている JVM のレプリカを新規に作成して、失われたレプリカと置き換えます。障害が発生した JVM にプライマリーがあった場合、eXtreme Scale は、残りの中から最適なレプリカを見つけて、そのレプリカを新規プライマリーとしてプロモートします。次に、eXtreme Scale は、障害が起きていないサーバー上に新しいレプリカを作成して、プロモートしたレプリカと置き換えます。スケーラビリティーを保つため、サーバーに障害が起こっても eXtreme Scale は区画のレプリカ数を維持します。

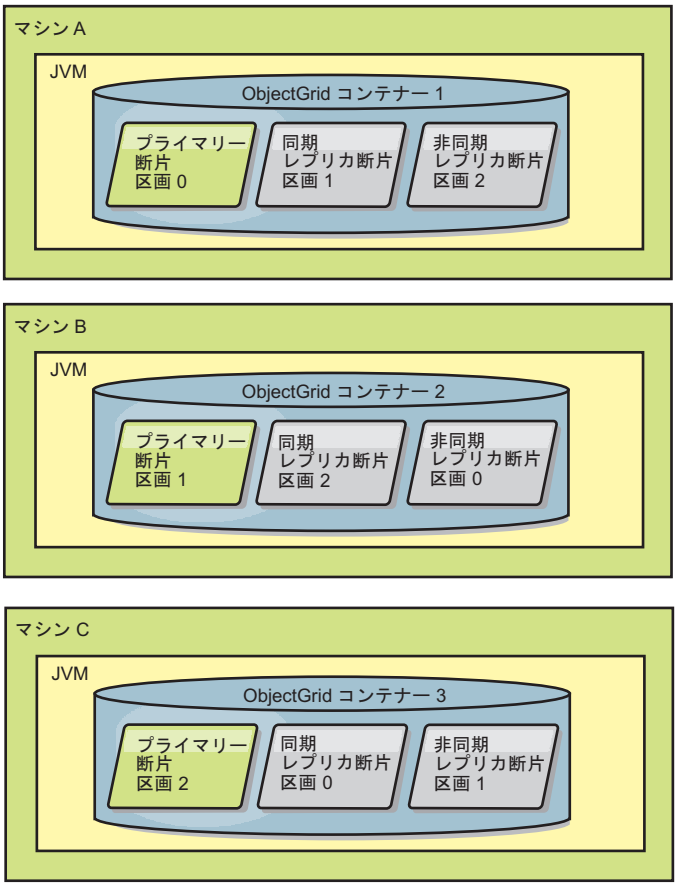


図 31. 区画が 3 つあり、minSyncReplicas 値を 1 に、maxSyncReplicas 値を 1 に、 maxAsyncReplicas 値を 1 にするというデプロイメント方針での ObjectGrid マップ・セットの配置

レプリカからの読み取り

クライアントがプライマリー断片のみに制限されずレプリカからの読み取りも許可されるようにマップ・セットを構成することができます。

障害に備えてレプリカを単に潜在的なプライマリー以上のものにできれば便利なときがよくあります。例えば、MapSet の replicaReadEnabled オプションを true に設定すれば、読み取り操作がレプリカに経路指定されるようにマップ・セットを構成することができます。デフォルト設定は false です。

MapSet エレメントについて詳しくは、「*管理ガイド*」に記載されているデプロイメント・ポリシー記述子 XML ファイルに関するトピックを参照してください。

レプリカの読み取りを使用可能にすると、読み取り要求がより多くの Java™ 仮想マシンに拡散されるので、パフォーマンスが向上します。このオプションが使用可能になっていないと、ObjectMap.get メソッドや Query.getResultIterator メソッドなどの読み取り要求はすべてプライマリーに経路指定されます。 replicaReadEnabled が true に設定されているときには、get 要求が不整合データを戻す可能性があるため、このオプションを使用しているアプリケーションはこの可能性を許容できるようにする必要があります。ただし、キャッシュ・ミスは起こりません。データがレプリカ上にないと、get 要求はプライマリーにリダイレクトされ、再試行されます。

replicaReadEnabled オプションは、同期レプリカ生成および非同期レプリカ生成の両方と一緒に使用できます。

レプリカ間のロード・バランシング

レプリカ間のロード・バランシングは、通常、クライアントが常時変更されるデータをキャッシュしているか、またはクライアントがペシミスティック・ロックを使用している場合にのみ使用されます。

特に構成されていない限り、eXtreme Scale は、すべての読み取り要求と書き込み要求を指定されたレプリカ生成グループのプライマリー・サーバーに送信します。プライマリーは、クライアントからのすべての要求にサービスを提供する必要があります。読み取り要求をプライマリーのレプリカに送信できるようにするとよいでしょう。読み取り要求をレプリカに送信することにより、読み取り要求の負荷を複数の Java 仮想マシン (JVM) で共有できるようになります。ただし、読み取り要求のためにレプリカを使用すると、応答が不整合になる可能性があります。

レプリカ間のロード・バランシングは、通常、クライアントが常時変更されるデータをキャッシュしているか、またはクライアントがペシミスティック・ロックを使用している場合にのみ使用されます。

データが絶えず変更され、そのためクライアントのニア・キャッシュで無効化された場合は、結果としてクライアントからプライマリーへの get 要求率が比較的高くなります。同様に、ペシミスティック・ロック・モードでは、ローカル・キャッシュが存在しないため、すべての要求がプライマリーに送信されます。

データが比較的静的であるか、またはペシミスティック・モードが使用されていない場合には、読み取り要求をレプリカへ送信しても、パフォーマンスにそれほど大きな影響を与えません。データで満杯のキャッシュを持つクライアントからの get 要求の頻度は、高くありません。

クライアントが始動されたばかりのときには、ニア・キャッシュは空です。空のキャッシュに対するキャッシュ要求は、プライマリーに転送されます。時間が経過してクライアント・キャッシュにデータが入れられると、要求ロードは除去されます。数多くのクライアントが同時に始動される場合には、ロードは大きくなる可能性があるため、パフォーマンス上、レプリカ読み取りを選択するほうが適切な場合があります。

断片のライフサイクル

断片は、さまざまな状態とイベントを経由してレプリカ生成をサポートします。断片のライフサイクルには、オンラインになること、ランタイム、シャットダウン、フェイルオーバー、およびエラー処理があります。サーバー状態変更を処理するため、レプリカ断片はプライマリー断片にプロモート可能です。

ライフサイクル・イベント

プライマリー断片とレプリカ断片は、配置されて開始されると、一連のイベントを経由してオンラインとなり、listen モードとなります。

プライマリー断片

カタログ・サービスは、プライマリー断片を区画に配置します。カタログ・サービスは、プライマリー断片のロケーションの平衡化、およびプライマリー断片に対するフェイルオーバーの開始の作業も行います。

ある断片がプライマリー断片になると、このプライマリー断片はカタログ・サービスからレプリカのリストを受信します。新規のプライマリー断片は、レプリカ・グループを作成し、すべてのレプリカを登録します。

プライマリーが作動可能となると、「ビジネス用にオープン」メッセージが、プライマリーの実行されているコンテナーの SystemOut.log ファイルに表示されます。オープン・メッセージ、つまり CWOBJ1511I メッセージには、開始したプライマリー断片のマップ名、マップ・セット名、および区画番号がリストされます。

```
CWOBJ1511I: mapName:mapSetName:partitionNumber (primary) is open for business.
```

カタログ・サービスが断片をどのように配置するかに関する情報については、118 ページの『断片配置』を参照してください。

レプリカ断片

レプリカ断片は、問題を検出した場合を除き、主にプライマリー断片に制御されます。通常のライフサイクルでは、プライマリー断片が、レプリカ断片を配置、登録、および登録抹消します。

プライマリー断片がレプリカ断片を初期化すると、メッセージでログが表示されます。このログには、レプリカが実行されている場所が記述され、レプリカ断片が使用可能であることが示されます。オープン・メッセージ、つまり CWOBJ1511I メッセージには、レプリカ断片のマップ名、マップ・セット名、および区画番号がリストされます。このメッセージは、次のとおりです。

```
CWOBJ1511I: mapName:mapSetName:partitionNumber (synchronous replica) is open for business.
```

または

```
CWOBJ1511I: mapName:mapSetName:partitionNumber (asynchronous replica) is open for business.
```

非同期レプリカ断片: 非同期レプリカ断片はデータを求めてプライマリーをポーリングします。 レプリカは、プライマリーからデータを受信しないと、それはレプリカがプライマリーに追いついたことを意味するため、自動的にポーリング時間を調整します。 プライマリーに障害が起こったことを示すエラーを受け取った場合、またはネットワークに問題があった場合も、レプリカは調整します。

非同期レプリカが複製を開始すると、非同期レプリカは、以下のメッセージをレプリカ用の SystemOut.log ファイルに出力します。 このメッセージは、1 回の CWOBJ1511 メッセージにつき複数回出力される可能性があります。 レプリカが別のプライマリーに接続する場合、あるいはテンプレート・マップが追加された場合、このメッセージは再度出力されます。

```
CWOBJ1543I: The asynchronous replica objectGridName:mapsetName:partitionNumber started or
continued replicating from the primary. Replicating for maps: [mapName]
```

同期レプリカ断片: 同期レプリカ断片が最初に開始するときは、まだピア・モードにはなっていません。 レプリカ断片がピア・モードになっていると、レプリカ断片

は、データがプライマリーに着信するときにプライマリーからデータを受信します。ピア・モードに入る前に、レプリカ断片には、プライマリー断片上のすべての既存データのコピーが必要となります。

同期レプリカは、非同期レプリカと同様に、データを求めてポーリングすることでプライマリー断片からデータをコピーします。 同期レプリカは、既存データをプライマリーからコピーする際、ピア・モードに切り替わり、プライマリーがデータを受信すると同時にデータの受信を開始します。

レプリカ断片がピア・モードに達すると、レプリカ断片は、メッセージをレプリカ用の SystemOut.log ファイルに出力します。所要時間は、レプリカ断片が、プライマリー断片から最初のデータをすべて取得するのに要した時間の長さを指します。プライマリー断片によって複製される既存のデータが存在しない場合、所要時間は、ゼロまたは非常に低く表示されます。このメッセージは、1 回の CWOBJ1511 メッセージにつき複数回出力される場合があります。レプリカが別のプライマリーに接続する場合、あるいはテンプレート・マップが追加された場合、このメッセージは再度出力されます。

```
CWOBJ1526I: Replica objectGridName:mapsetName:partitionNumber:mapName entering peer
mode after X seconds.
```

同期レプリカ断片がピア・モードになっていると、プライマリー断片はすべてのピア・モードの同期レプリカにトランザクションを複製しなければなりません。 同期レプリカ断片のデータは、プライマリー断片のデータと同じレベルに保たれます。同期レプリカの最小数または minSync をデプロイメント・ポリシーで設定している場合、同期レプリカの最小数がコミットに賛成してからトランザクションはプライマリーで正常にコミットできます。

リカバリー・イベント

レプリカ生成は、障害およびエラー・イベントからリカバリーするように設計されています。あるプライマリー断片が失敗すると、別のレプリカが引き継ぎます。エラーがレプリカ断片上にある場合、レプリカ断片は、リカバリーを試行します。カタログ・サービスは、新規プライマリー断片または新規レプリカ断片の配置とトランザクションを制御します。

レプリカ断片がプライマリー断片となる

レプリカ断片は、2 つの理由でプライマリー断片となります。プライマリー断片が停止または失敗した場合と、バランスを取る決定が行われて、前のプライマリー断片を新規ロケーションに移動した場合です。

カタログ・サービスは、新規プライマリー断片を、既存の同期レプリカ断片から選択します。プライマリーを移動させる必要があり、レプリカがない場合は、一時レプリカを配置して遷移を完了します。 新規プライマリー断片は、すべての既存レプリカを登録し、トランザクションを新規プライマリー断片として受け入れます。既存のレプリカ断片に正しいレベルのデータが存在する場合、現行データは、レプリカ断片が新規プライマリー断片に登録されると同時に保存されます。非同期レプリカは新規プライマリーに対してポーリングします。

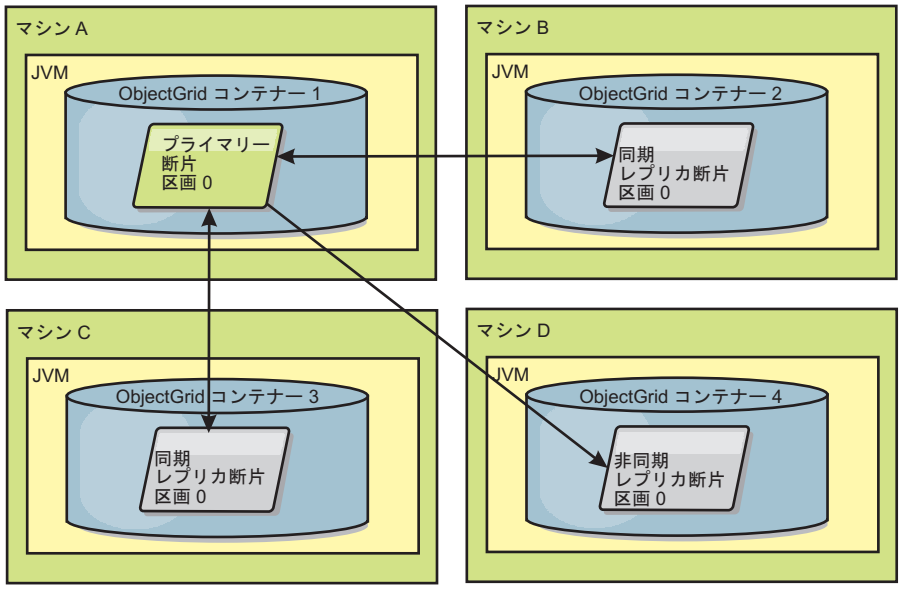


図 32. partition0 区画用の ObjectGrid マップ・セットの配置例。これは、minSyncReplicas 値を 1 に、maxSyncReplicas 値を 2 に、maxAsyncReplicas 値を 1 にするというデプロイメント方針です。

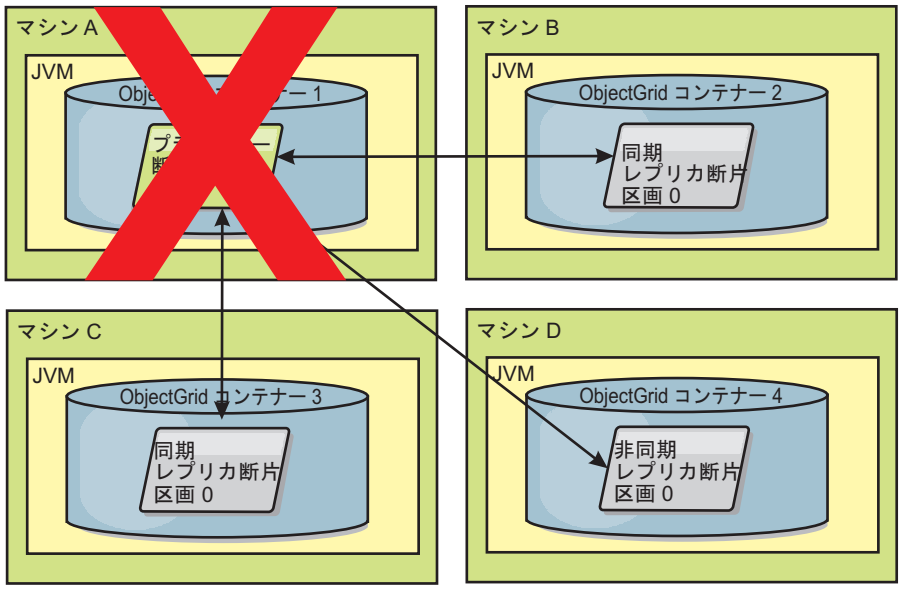


図 33. プライマリー断片のコンテナーに障害が起こる

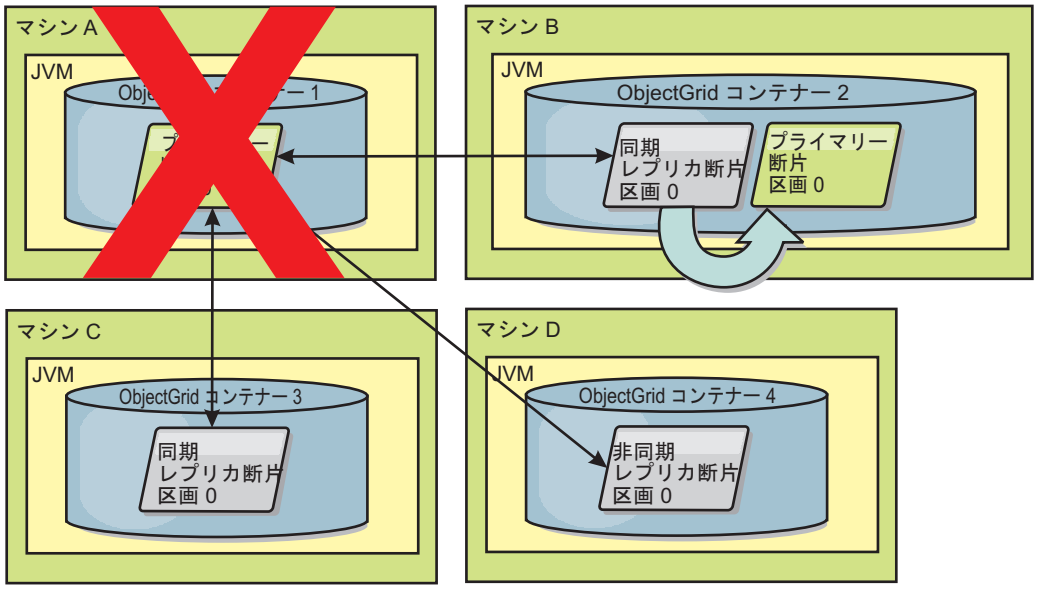


図 34. ObjectGrid コンテナー 2 にある同期レプリカ断片がプライマリー断片になる

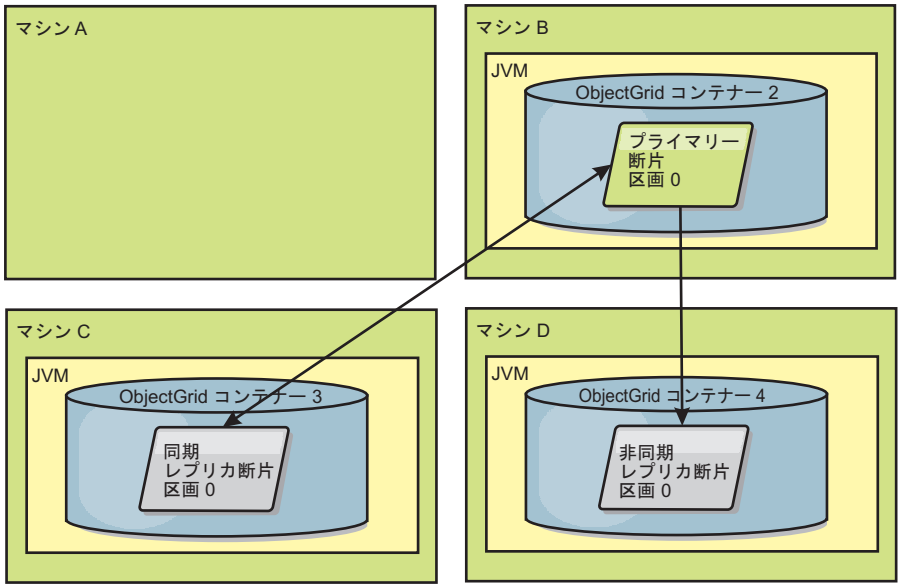


図 35. マシン B にプライマリー断片が含まれています。自動修復モードがどのように設定されているか、およびコンテナーが使用可能かどうかに基づいて、新しい同期レプリカ断片がマシンに配置されるかどうかが決まります。

レプリカ断片のリカバリー

同期レプリカ断片は、プライマリー断片によって制御されます。ただし、レプリカ断片は、問題を検出すると、登録イベントをトリガーして、データの状態を訂正することができます。レプリカは、現行データをクリアして、新しいコピーをプライマリーから取得します。

レプリカ断片が登録イベントを開始すると、そのレプリカはログ・メッセージを出力します。

```
CWOBJ1524I: Replica listener
objectGridName:mapSetName:partition must re-register with the primary.
Reason: Exception listed
```

トランザクションの処理中にレプリカ断片上でエラーが発生した場合には、そのレプリカ断片は不明な状態です。トランザクションはプライマリー断片上で正常に処理されましたが、レプリカ上で何らかの異常が発生しました。この状態を訂正するため、レプリカは再登録イベントを開始します。プライマリーからのデータの新しいコピーを使用して、レプリカ断片は続行できます。同じ問題が再発生する場合、レプリカ断片は、連続して再登録を行いません。詳しくは、『障害イベント』 を参照してください。

障害イベント

レプリカは、リカバリーできないエラー状態を検出した場合、データの複製を停止することがあります。

多すぎる登録試行

レプリカがデータを正常にコミットせずに、登録を複数回トリガーする場合、レプリカは停止します。停止により、レプリカがエンドレス登録ループに入ることが防止されます。デフォルトでは、レプリカ断片は、登録を連続して 3 回試行してから、停止します。

レプリカ断片が何度も登録しすぎる場合、レプリカは、次のメッセージをログに出力します。

```
CWOBJ1537E: objectGridName:mapSetName:partition exceeded the maximum number
of times to reregister (timesAllowed) without successful transactions..
```

レプリカが再登録によってリカバリーできない場合は、レプリカ断片に関連するトランザクションに、広範囲な問題が存在する可能性があります。トランザクションからのキーまたは値の展開中にエラーが発生する場合、クラスパス上のリソースが欠落しているという問題が考えられます。

ピア・モードに入る際の障害

プライマリー (チェックポイント・データ) からの既存のバルク・データの処理中に、レプリカがピア・モードに入ろうとしてエラーが発生した場合、レプリカはシャットダウンします。シャットダウンは、レプリカが正しくない初期データで開始するのを防止します。レプリカは、再登録すれば、プライマリーから同じデータを受信するため、再試行はしません。

レプリカ断片がピア・モードに入ることに失敗した場合、レプリカ断片は、次のメッセージをログに出力します。

```
CWOBJ1527W Replica objectGridName:mapSetName:partition:mapName failed to enter peer mode after numSeconds seconds.
```

レプリカがピア・モードに入ることに失敗した理由を説明する追加のメッセージがログに表示されます。

再登録またはピア・モード障害後のリカバリー

レプリカが再登録できないか、ピア・モードに入ることができない場合、そのレプリカは、新規配置イベントが発生するまで非アクティブ状態になります。新規配置

イベントは、新規サーバーの開始または停止である場合があります。配置イベントは、PlacementServiceMBean Mbean で triggerPlacement メソッドを使用して開始することもできます。

レプリカ生成のためのマップ・セット

レプリカ生成は、BackingMap をマップ・セットに関連付けることで使用可能になります。

マップ・セットは、区画キーによってカテゴリー化されるマップの集まりです。この区画キーは、個別マップのキーから、そのハッシュ・モジュロを取って区画数とすることで派生します。マップ・セット内の 1 つのマップ・グループが区画キー X を持つとすると、それらのマップはデータ・グリッド内の対応する区画 X に保管されます。別のグループが区画キー Y を持つとすると、それらのマップはすべて区画 Y に保管されます。以下同様です。また、マップ内のデータは、マップ・セットに定義されたポリシーに基づいて複製されます。これは、分散 eXtreme Scale トポロジーのみに使用されます (ローカル・インスタンスの場合は不要です)。

詳しくは、 83 ページの『区画化』 を参照してください。

マップ・セットは、それらが持つ区画の数およびレプリカ生成ポリシーを割り当てられます。マップ・セット・レプリカ生成構成は、マップ・セットがプライマリー断片に加えて持つことになる同期および非同期のレプリカ断片の数を示すだけです。例えば、1 つの同期レプリカと 1 つの非同期レプリカが存在することになる場合、マップ・セットに割り当てられたすべての BackingMap は、それぞれ eXtreme Scale の使用可能なコンテナー・セット内に自動的に配布されるレプリカ断片を持ちます。また MapSet レプリカ生成構成により、クライアントは同期複製されたサーバーからデータを読み取れるようになります。これにより、読み取り要求の負荷を eXtreme Scale 内のその他のサーバーにも分散することができます。レプリカ生成は、BackingMap のプリロード時にプログラミング・モデルに影響するだけです。

トランザクション処理の概要

WebSphere eXtreme Scale は、データとの相互作用のメカニズムとしてトランザクションを使用します。

データとの相互作用のために、アプリケーション内のスレッドは、独自のセッションを必要とします。アプリケーションがスレッド上で ObjectGrid を使用する必要がある場合、ObjectGrid.getSession メソッドの 1 つを呼び出してスレッドを取得します。このセッションを使用すると、アプリケーションは ObjectGrid マップに保管されているデータの処理を行うことができます。

アプリケーションが Session オブジェクトを使用する場合、そのセッションはトランザクションのコンテキスト内にある必要があります。 Session オブジェクトに対する begin メソッド、commit メソッド、および rollback メソッドにより、トランザクションは、開始してコミット、あるいは開始してロールバックを行います。また、アプリケーションは自動コミット・モードで動作することも可能で、この場合、マップに対する操作が実行されるたびに、Session は自動的にトランザクションを開始してコミットします。自動コミット・モードでは複数の操作を単一トランザクションにグループ化することはできないため、複数操作のバッチを作成して単一トランザクションにする場合は、自動コミット・モードの方が時間がかかるオプシ

ョンです。ただし、単一の操作しか含まないトランザクションの場合は、自動コミット・モードの方が速いオプションになります。

トランザクション

トランザクションには、データ保管および操作に関して多くの利点があります。トランザクションを使用すれば、同時変更からデータ・グリッドを保護したり、複数の変更を 1 つの並行ユニットとして適用したり、データを複製したり、変更に対するロックのライフサイクルを実装したりすることができます。

トランザクションが開始すると、WebSphere eXtreme Scale は別の特別なマップを割り振って、そのトランザクションが使用するキーと値のペアの現在の変更またはコピーを保持します。通常、キーと値のペアにアクセスすると、アプリケーションがその値を受け取る前に、値のコピーが作成されます。その別のマップは、挿入、更新、取得、除去などの操作についてすべての変更を追跡します。キーは不変のものと見なされているため、コピーされません。ObjectTransformer オブジェクトを指定すると、このオブジェクトが値をコピーするために使用されます。 トランザクションがオプティミスティック・ロックを使用している場合は、トランザクションのコミット時に、以前の値のイメージも比較のために追跡されます。

トランザクションがロールバックされる場合、その別のマップの情報は破棄され、エントリーに対するロックは解除されます。トランザクションをコミットすると、変更がマップに適用され、ロックが解除されます。オプティミスティック・ロックが使用されている場合、eXtreme Scale は、以前のイメージ・バージョンの値とマップ内の値を比較します。トランザクションをコミットするには、これらの値が一致している必要があります。こうした比較によって複数バージョンのロック体系が可能になりますが、トランザクションがそのエントリーにアクセスすると、代わりに 2 つのコピーが作成されます。すべての値が再度コピーされ、新しいコピーがマップに保管されます。 WebSphere eXtreme Scale は、コミット後に値へのアプリケーション参照を変更するアプリケーションから自身を保護するために、このコピーを実行します。

情報の複数のコピーを使用しないようにできます。アプリケーションは、並行性を制限する代償としてオプティミスティック・ロックの代わりにペシミスティック・ロックを使用することで、コピーを節約できます。コミット後に値を変更しないことにアプリケーションが同意すれば、コミット時の値のコピーも回避することができます。

トランザクションの利点

トランザクションを使用するのは、以下の理由からです。

トランザクションを使用して、以下の操作を行うことができます。

- 例外が発生した場合や、ビジネス・ロジックにより状態変更を元に戻す必要がある場合に、変更をロールバックします。
- コミット時に複数の変更をアトミック単位で適用する
- データに対するロックの保持および解除を行い、コミット時に複数の変更をアトミック単位で適用します。
- 同時変更からスレッドを保護します。
- 変更に対するロックのライフサイクルを実装します。

- アトミック単位のレプリカ生成をします。

トランザクション・サイズ

トランザクションは、特にレプリカ生成の場合には、大きいほど効果的です。ただし、大きなトランザクションの場合はエントリーのロックの保持時間が長くなるため、並行性に悪影響を及ぼします。大きなトランザクションを使用すると、レプリカ生成のパフォーマンスが向上する場合があります。このパフォーマンスの向上は、マップを事前にロードする場合には重要です。さまざまなバッチ・サイズで実験を行い、使用するシナリオに最適なサイズを判別してください。

大きなトランザクションはローダーにとっても好都合です。 SQL バッチを実行できるローダーを使用している場合は、トランザクションによっては著しくパフォーマンスが向上する可能性があり、データベース側ではロードを著しく削減することができます。このパフォーマンス向上は、ローダーの実装方法によって異なります。

自動コミット・モード

アクティブに始動されたトランザクションがない場合は、アプリケーションが ObjectMap オブジェクトとの対話を行うと、アプリケーションの代わりに自動的に開始およびコミット操作が行われます。この自動的な開始およびコミット操作は役に立ちますが、ロールバックおよびロックが有効に機能する妨げとなります。トランザクションのサイズが小さすぎると、同期レプリカ生成スピードに影響します。エンティティー・マネージャー・アプリケーションを使用している場合は、自動コミット・モードは使用しないでください。その理由は、EntityManager.find メソッドで検索されたオブジェクトが、そのメソッドが戻されると同時に管理不能となり、使用不可となるためです。

外部トランザクション・コーディネーター

通常、トランザクションは、session.begin メソッドで開始し、session.commit メソッドで終了します。ただし、eXtreme Scale が組み込まれていると、トランザクションは、外部トランザクション・コーディネーターによって開始および終了する場合があります。外部トランザクション・コーディネーターを使用している場合は、session.begin メソッドを呼び出す必要も、session.commit メソッドで終了する必要もありません。 WebSphere Application Server を使用している場合は、WebSphereTranscationCallback プラグインを使用できます。

CopyMode 属性

ObjectGrid 記述子 XML ファイルで BackingMap または ObjectMap オブジェクトの CopyMode 属性を定義することで、コピーの数を調整することができます。

BackingMap または ObjectMap オブジェクトの CopyMode 属性を定義することで、コピーの数を調整することができます。コピー・モードには以下の値があります。

- COPY_ON_READ_AND_COMMIT
- COPY_ON_READ
- NO_COPY
- COPY_ON_WRITE

- COPY_TO_BYTES
- COPY_TO_BYTES_RAW

COPY_ON_READ_AND_COMMIT がデフォルト値です。 COPY_ON_READ 値は、最初のデータ取得時にはコピーを行いますが、コミット時にはコピーを行いません。アプリケーションが、トランザクションのコミット後の値を変更しなければ、このモードが安全です。NO_COPY 値は、データをコピーしないため、読み取り専用データの場合のみ安全です。データが変更されない限り、分離目的でデータをコピーする必要はありません。

更新される可能性があるマップに NO_COPY 属性値を使用する場合は、注意が必要です。WebSphere eXtreme Scale は最初のタッチ時のコピーを使用して、トランザクションのロールバックを可能にします。アプリケーションはコピーを変更しただけなので、eXtreme Scale はそのコピーを破棄します。 NO_COPY 属性値が使用され、かつアプリケーションがコミットされた値を変更した場合は、ロールバックを完了することが不可能になります。 索引やレプリカはトランザクションのコミット時に更新されるため、コミット済みの値を変更すると、索引、レプリカ生成などに問題が生じます。コミット済みのデータを変更してからトランザクションをロールバックした場合は、これによって実際にはまったくロールバックされないため、索引は更新されず、レプリカ生成は行われません。他のスレッドは、コミットされていない変更を、ロックがあっても即時に参照することができます。読み取り専用マップ、または値を変更する前に適切なコピーを完了するアプリケーションの場合は、NO_COPY 属性値を使用してください。 NO_COPY 属性値を使用した場合に、データ保全性の問題で IBM サポートに連絡すると、コピー・モードを COPY_ON_READ_AND_COMMIT に設定して問題を再現するように求められます。

COPY_TO_BYTES 値は、マップ内の値をシリアライズ・フォームに保管します。eXtreme Scale は、読み取り時にシリアライズ・フォームからの値を拡張し、コミット時に値をシリアライズ・フォームに保管します。この方法によれば、読み取り時とコミット時の両方でコピーが行われます。

マップのデフォルトのコピー・モードは、BackingMap オブジェクトで構成することができます。さらに、トランザクションを開始する前に、ObjectMap.setCopyMode メソッドを使用してマップのコピー・モードを変更することができます。

objectgrid.xml ファイルにあり、指定のバックアップ・マップのコピー・モードを設定する方法を示すバックアップ・マップ・スニペットの例は以下のとおりです。この例では、objectgrid/config 名前空間として cc を使用しているものとします。

```
<cc:backingMap name="RuntimeLifespan" copyMode="NO_COPY"/>
```

ロック・マネージャー

ロック・ストラテジーを構成すると、キャッシュ・エントリーの整合性を維持するために、ロック・マネージャーがバッキング・マップに作成されます。

ロック・マネージャー構成

ロック・ストラテジーに OPTIMISTIC または PESSIMISTIC が使用されている場合は、BackingMap に対してロック・マネージャーが作成されます。ロック・マネージャーは、ハッシュ・マップを使用して、1 つ以上のトランザクションによってロッ

クされるエントリーを追跡します。 ハッシュ・マップに多くのマップ・エントリーが存在する場合、ロック・バケットが多いほど、パフォーマンスが良好になる可能性が高くなります。バケット数が増えるにつれて、Java 同期の衝突のリスクは下がります。またロック・バケットを増やすことが、並行性の増大につながります。前の例では、特定の BackingMap インスタンスに使用するロック・バケットの数をアプリケーションでどのように設定できるかを示しています。

java.lang.IllegalStateException 例外を避けるには、ObjectGrid インスタンスで initialize メソッドまたは getSession メソッドを呼び出す前に setNumberOfLockBuckets メソッドを呼び出す必要があります。 setNumberOfLockBuckets メソッド・パラメーターは、使用するロック・バケットの数を指定する Java プリミティブ整数です。素数を使用すると、ロック・バケット上のマップ・エントリーの一様分布が可能になります。最良のパフォーマンスを得るために適した開始点は、BackingMap エントリーの予想される数のおよそ 10 パーセントにロック・バケットの数を設定することです。

ロック・ストラテジー

ロック・ストラテジーには、ペシミスティック、オプティミスティック、およびロックなしがあります。 ロック・ストラテジーを選択する場合、各タイプの操作の比率、ローダーを使用するかどうかなどの問題を考慮する必要があります。

ロックはトランザクションに束縛されます。以下のロック設定を指定することができます。

- **ロックなし**: ロック設定を使用しないと、実行は最速になります。読み取り専用データを使用していれば、ロックは必要ない場合があります。
- **ペシミスティック・ロック**: エントリーに対するロックを取得し、コミット時までそのロックを保持します。このロック戦略は、スループットを低下させる代わりに、優れた一貫性を提供します。
- **オプティミスティック・ロック**: トランザクションがタッチするすべてのレコードの以前のイメージを取得して、トランザクションのコミット時に、そのイメージと現在のエントリーの値を比較します。エントリーの値が変更された場合、そのトランザクションはロールバックします。コミット時までロックは保持されません。このロック戦略は、ペシミスティック戦略よりも並行性において優れていますが、トランザクション・ロールバックのリスクがあり、エントリーのコピーを作成するためにメモリーを消費します。

BackingMap でロック戦略を設定します。各トランザクションのロック戦略を変更することはできません。XML ファイルを使用してマップに対してロック・モードを設定する方法を示す XML スニペットの例は以下のとおりです。この場合、cc は、objectgrid/config 名前空間用の名前空間であるとします。

```
<cc:backingMap name="RuntimeLifespan" lockStrategy="PESSIMISTIC" />
```

ペシミスティック・ロック

ほかのロック・ストラテジーが可能でない場合は、マップの読み書きにペシミスティック・ロック・ストラテジーを使用します。ObjectGrid マップがペシミスティック・ロック・ストラテジーを使用するように構成されている場合、トランザクションが最初に BackingMap からのエントリーを取得すると、マップ・エントリーのペシミスティック・トランザクション・ロックが取得されます。ペシミスティック・

ロックは、アプリケーションがトランザクションを完了するまでは保留されます。通常の場合、ペシミスティック・ロック・ストラテジーは、以下の状態で使用されます。

- BackingMap がローダー付き、またはローダーなしで構成され、バージョン管理情報が使用可能でない場合。
- BackingMap が、並行処理制御について eXtreme Scale からの支援を必要とするアプリケーションによって直接使用されている場合。
- バージョン管理情報は使用できるが、更新トランザクションがバッキング・エントリー上で頻繁に衝突し、その結果、オプティミスティック更新が失敗する場合。

ペシミスティック・ロック・ストラテジーは、パフォーマンスとスケーラビリティーに最大のインパクトを与えるので、このストラテジーはほかのロック・ストラテジーが実行可能でないときのマップの読み取りと書き込みにのみ使用してください。例えば、こうした状態には、オプティミスティック更新の失敗が頻繁に発生する場合や、オプティミスティック障害からのリカバリーをアプリケーションが処理するには難しい場合が含まれます。

オプティミスティック・ロック

オプティミスティック・ロック・ストラテジーでは、並行して実行中に、2 つのトランザクションが同じマップ・エントリーを更新することはないと想定します。このことから、トランザクションのライフサイクル中、ロック・モードを保留する必要はありません。これは、複数のトランザクションがマップ・エントリーを並行して更新するとは考えられないためです。オプティミスティック・ロック・ストラテジーは通常、以下の場合に使用されます。

- BackingMap がローダー付き、またはローダーなしで構成され、バージョン管理情報が使用可能である場合。
- BackingMap のほとんどのトランザクションが読み取り操作を実行するトランザクションである場合。 BackingMap に対するエントリーの挿入、更新、または除去操作は、あまり行われません。
- BackingMap は、読み取りと比べてより頻繁に挿入、更新、または除去されるが、トランザクションは同じマップ・エントリー上でほとんど衝突しない場合。

ペシミスティック・ロック・ストラテジーと同様に、 ObjectMap インターフェース上のメソッドは、eXtreme Scale が、アクセス中のマップ・エントリーのロック・モードを自動的に取得する方法を決定します。ただし、ペシミスティック・ストラテジーとオプティミスティック・ストラテジーの間には、以下のような違いがあります。

- ペシミスティック・ロック・ストラテジーと同様に、メソッドの呼び出しの際、get メソッドおよび getAll メソッドによって S ロック・モードが取得されます。しかし、オプティミスティック・ロックを使用すると、 S ロック・モードはトランザクションが完了するまで保留されません。代わりに、S ロック・モードはメソッドがアプリケーションに戻す前に保留解除されます。ロック・モードの取得の目的は、eXtreme Scale が、その他のトランザクションからのコミット済みデータのみが現行トランザクションに可視となるように保証できるようにすることです。 eXtreme Scale がそのデータがコミット済みであることを確認した後で、S ロック・モードは保留解除されます。コミット時に、オプティミスティッ

ク・バージョン管理チェックが実行され、現行トランザクションがその S ロック・モードを保留解除した後で、マップ・エントリーを変更したトランザクションが他にないことが確認されます。更新、無効化、または削除される前にマップからエントリーがフェッチされない場合、eXtreme Scale ランタイムによって、暗黙的にマップからエントリーがフェッチされます。この暗黙的な get 操作は、エントリーの変更が要求された時点における現行値を取得するために実行されます。

- ペシミスティック・ロック・ストラテジーとは異なり、 getForUpdate メソッドと getAllForUpdate メソッドは、オプティミスティック・ロック・ストラテジーが使用された場合には、 get メソッドと getAll メソッドと同様に処理されます。つまり、S ロック・モードはメソッドの開始時に取得され、S ロック・モードはアプリケーションに戻る前に保留解除されます。

その他の ObjectMap メソッドは、すべてペシミスティック・ロック・ストラテジーの場合と同様に処理されます。つまり、commit メソッドが呼び出されると、挿入、更新、除去、タッチ、または無効化されたマップ・エントリー用に X ロック・モードが獲得され、トランザクションがコミット処理を完了するまで X ロック・モードが保留されます。

オプティミスティック・ロック・ストラテジーでは、並行して実行中のトランザクションが同じマップ・エントリーを更新することはないと想定します。この想定から、トランザクションの存続期間中、ロック・モードを保留する必要はありません。これは、複数のトランザクションがマップ・エントリーを並行して更新するとは考えられないためです。しかし、ロック・モードが保留されなかったため、現行トランザクションがその S ロック・モードを保留解除した後で、別の並行トランザクションがマップ・エントリーを更新する可能性があります。

この可能性に対処するため、eXtreme Scale はコミット時に X ロックを取得し、オプティミスティック・バージョン管理チェックを行って、現行トランザクションが BackingMap からマップ・エントリーを読み取って以降、他にマップ・エントリーを変更したトランザクションがないことを確認します。別のトランザクションがマップ・エントリーを変更した場合、バージョン・チェックは失敗し、OptimisticCollisionException 例外が発生します。この例外により、現行トランザクションが強制的にロールバックされ、トランザクション全体がアプリケーションによって再試行されることになります。オプティミスティック・ロック・ストラテジーは、マップがほとんど既読で、同じマップ・エントリーに対する更新が起こる可能性が低い場合に非常に便利です。

ロックなし

BackingMap がロックなしストラテジーを使用するよう構成されている場合、マップ・エントリーのトランザクション・ロックは獲得されません。

ロックなしストラテジーは、アプリケーションが Enterprise JavaBeans (EJB) コンテナーなどのパーシスタンス・マネージャーである場合や、アプリケーションが Hibernate を使用して永続データを取得している場合に有効です。このシナリオでは、BackingMap はローダーを使用せずに構成され、パーシスタンス・マネージャーによってデータ・キャッシュとして使用されます。またこのシナリオでは、パーシスタンス・マネージャーにより、同じマップ・エントリーにアクセスするトランザクション間の並行性制御が提供されます。

WebSphere eXtreme Scale は、並行性制御のためにトランザクション・ロックを入手する必要はありません。これは、パーシスタンス・マネージャーが、コミットされた変更で ObjectGrid マップを更新する前にそのトランザクション・ロックをリリースしないことを前提としています。パーシスタンス・マネージャーがロックを解放する場合は、ペシミスティックまたはオプティミスティック・ロック・ストラテジーを使用しなければなりません。例えば、EJB コンテナーのパーシスタンス・マネージャーが、EJB コンテナー管理のトランザクション内でコミットされたデータで ObjectGrid Map を更新していると仮定します。ObjectGrid マップの更新が、パーシスタンス・マネージャーのトランザクション・ロックが解放される前に発生する場合、ロックなしストラテジーを使用することができます。パーシスタンス・マネージャーのトランザクション・ロックが解放された後で ObjectGrid マップ更新が発生する場合は、オプティミスティックまたはペシミスティックのいずれかのロック・ストラテジーを使用してください。

ロックなしストラテジーの使用が可能なもう 1 つのシナリオは、アプリケーションが BackingMap を直接使用し、ローダーがマップに対して構成されているときです。このシナリオでは、ローダーは、Java Database Connectivity (JDBC) または Hibernate のいずれかを使用してリレーショナル・データベース内のデータにアクセスすることによって、リレーショナル・データベース管理システム (RDBMS) によって提供される並行性制御サポートを使用します。ローダーの実装は、オプティミスティックまたはペシミスティックのいずれかの方法を使用できます。オプティミスティック・ロックまたはバージョン管理方法を使用するローダーは、大量の並行性およびパフォーマンスの達成を支援します。オプティミスティック・ロック手法の実装について詳しくは、「*管理ガイド*」内のローダー考慮事項に関する説明の OptimisticCallback セクションを参照してください。基礎となるバックエンドのペシミスティック・ロック・サポートを使用するローダーを使用する場合は、ローダー・インターフェースの get メソッドに渡される forUpdate パラメーターを使用することがあります。アプリケーションがデータを取得するために ObjectMap インターフェースの getForUpdate メソッドを使用した場合は、このパラメーターを true に設定します。ローダーはこのパラメーターを使用して、読み取り中の行のアップグレード可能なロックを要求するかどうかを判別できます。例えば、DB2® は、SQL の SELECT ステートメントに `FOR UPDATE` 節が含まれている場合、アップグレード可能なロックを獲得します。このアプローチは、131 ページの『ペシミスティック・ロック』で説明されているのと同じデッドロック防止を提供します。

詳しくは、「*プログラミング・ガイド*」のロックの処理に関するトピック、または「*管理ガイド*」のマップ・エントリー・ロックに関するトピックを参照してください。

トランザクションの配布

異なる層間、または混合プラットフォーム上の環境間で、トランザクションの変更を配布するために Java Message Service (JMS) を使用します。

JMS は、異なる層または混合しているプラットフォームの環境で配布された変更に理想的なプロトコルです。例えば、eXtreme Scale を使用するいくつかのアプリケーションが、IBM WebSphere Application Server Community Edition、Apache Geronimo、または Apache Tomcat にデプロイされていて、別のアプリケーションが WebSphere Application Server バージョン 6.x で実行しているとします。このような多様な環境における eXtreme Scale ピア間で配布される変更には、JMS が理想的で

す。HA マネージャーのメッセージ・トランスポートは非常に高速ですが、単一コア・グループに属する Java 仮想マシン にのみ変更を配布できます。 JMS はそれに比較すれば低速ですが、より広範囲で、多様なアプリケーション・クライアントのセットに ObjectGrid を共有させることができます。 JMS は、ファット Swing クライアントと、 WebSphere Extended Deployment にデプロイされているアプリケーションとの間で、ObjectGrid 内のデータを共有する場合に理想的です。

JMS を使用したトランザクションの変更の配布の例としては、組み込みの クライアント無効化メカニズムやピアツーピア・レプリカ生成メカニズムなどがあります。詳しくは、*管理ガイド*の JMS を使用したピアツーピア・レプリカ生成の構成に関する説明を参照してください。

JMS の実装

JMS は、ObjectGridEventListener として動作する Java オブジェクトを使用してトランザクションの変更を配布するために実装されます。このオブジェクトは、以下の 4 つの方法で状態を伝搬することができます。

1. 無効化: 除去、更新、または削除されるエントリーは、メッセージを受け取ると、すべてのピア Java 仮想マシンで除去されます。
2. 無効化の条件: ローカル・バージョンがパブリッシャーのバージョンと同じか、またはそれより古い場合のみ、エントリーが除去されます。
3. プッシュ: 除去、更新、削除または挿入されたエントリーは、JMS メッセージを受信する場合、すべてのピア Java 仮想マシンに追加または上書きされます。
4. プッシュ条件: ローカル・エントリーがパブリッシュされているバージョンより新しくない場合に、エントリーは受信サイドで更新または追加のみ行われます。

パブリッシュする変更の listen

プラグインは、ObjectGridEventListener インターフェースを実装し、transactionEnd イベントをインターセプトします。 eXtreme Scale がこのメソッドを呼び出す場合、プラグインはトランザクションによってタッチされる各マップの LogSequence リストを JMS メッセージに変換し、それをパブリッシュしようとします。プラグインは、すべてのマップまたはマップのサブセットの変更をパブリッシュするよう構成することができます。 LogSequence オブジェクトは、パブリッシュが使用可能なマップのために処理されます。 LogSequenceTransformer ObjectGrid クラスは、ストリームに対して各マップのフィルタリングされた LogSequence をシリアライズします。すべての LogSequences がストリームにシリアライズされたら、JMS ObjectMessage が作成され、既知のトピックにパブリッシュされます。

JMS メッセージの listen およびローカル ObjectGrid への適用

同じプラグインはまた、既知のトピックにパブリッシュされるすべてのメッセージを受け取りながら、ループでスピンするスレッドを開始します。メッセージを受け取ると、LogSequenceTransformer クラスにメッセージ・コンテンツを渡します。このクラスでメッセージ・コンテンツは LogSequence オブジェクトのセットに変換されます。その後、ノー・ライトスルー・トランザクションが開始されます。各 LogSequence オブジェクトは Session.processLogSequence メソッドに提供され、その変更でローカル Map を更新します。 processLogSequence メソッドは、配布モードを理解しています。トランザクションはコミットされ、ローカル・キャッシュが変

更を反映します。 JMS を使用してトランザクションの変更を配布する方法について詳しくは、「*管理ガイド*」の Java 仮想マシンのピア間での変更の配布に関する説明を参照してください。

単一区画トランザクションおよびクロスデータ・グリッド・トランザクション

WebSphere eXtreme Scale とリレーショナル・データベースやメモリー内データベースなどの従来のデータ・ストレージ・ソリューションとの間の主な相違は、キャッシュの直線的な増加を可能にする区画化を使用することにあります。考慮すべき重要なトランザクションのタイプに、単一区画トランザクションと各区画 (クロスデータ・グリッド) トランザクションがあります。

一般的に、以下のセクションで説明するようにキャッシュとの対話は、単一区画トランザクションまたはクロスデータ・グリッド・トランザクションとして分類できます。

単一区画トランザクション

単一区画トランザクションは、WebSphere eXtreme Scale によってホストされるキャッシュと対話する場合に適した方法です。単一区画に制限されている場合のトランザクションは、デフォルトで単一の Java 仮想マシン、すなわち単一のサーバー・コンピューターに制限されます。サーバーは、こうしたトランザクションを毎秒 *M* 個実行することができるので、*N* 台のコンピューターがある場合は、毎秒 `M*N` 個のトランザクションを実行できます。ビジネスが拡大し、毎秒こうしたトランザクションを 2 倍の数実行する必要性が出てきた場合、さらにコンピューターを購入して *N* を 2 倍にすることができます。これにより、アプリケーションを変更したり、ハードウェアをアップグレードしたり、さらにはアプリケーションをオフラインにしたりすることさえなく、容量ニーズを満たすことができます。

単一区画トランザクションは、キャッシュの拡大をかなり大幅に行えるようになっているほか、キャッシュの可用性を最大限に引き出します。各トランザクションは、1 台のコンピューターのみに依存します。他の `(N-1)` 台のコンピューターのいずれかに障害が起こっても、このトランザクションの成否および応答時間には影響しません。したがって、100 台のコンピューター (サーバー) を稼働していて、そのうち 1 台に障害が生じても、そのサーバーに障害が生じた時点で進行中であった 1 パーセントのトランザクションしかロールバックされません。サーバーの障害後、WebSphere eXtreme Scale は、障害を起こしたサーバーによってホストされる区画を他の 99 台のコンピューターに再配置します。これは短時間の処理であり、この操作の完了前であれば、この時間内に他の 99 台のコンピューターはトランザクションを完了できます。再配置される区画に関係するトランザクションしか、ブロックされません。フェイルオーバー・プロセスが完了すると、キャッシュは、元のスループット量の 99 パーセントで完全に操作可能状態で引き続き稼働できるようになります。障害のあるサーバーが交換されて、データ・グリッドに戻されると、キャッシュは 100 パーセントのスループット量に戻ります。

クロスデータ・グリッド・トランザクション

パフォーマンス、可用性、およびスケーラビリティーの面では、クロスデータ・グリッド・トランザクションは、単一区画トランザクションの対極にあります。クロスデータ・グリッド・トランザクションは、すべての区画、つまり構成内のすべて

のコンピューターにアクセスします。データ・グリッド内の各コンピューターは、ある種のデータを検索して、その結果を戻すように求められます。トランザクションは、すべてのコンピューターが応答するまで完了できません。したがってデータ・グリッド全体のスループットは、最低速のコンピューターによって制限されます。コンピューターを追加しても、最低速のコンピューターの処理速度が増すわけではなく、キャッシュのスループットは改善しません。

クロスデータ・グリッド・トランザクションは、可用性についても同じ影響を及ぼします。先の例を拡大すると、100 台のサーバーが稼働していて、そのうち 1 台に障害が生じたとすると、そのサーバーに障害が生じた時点で進行中であったトランザクションの 100 パーセントがロールバックされます。サーバーの障害後、WebSphere eXtreme Scale は、このサーバーによってホストされる区画を他の 99 台のコンピューターに再配置する処理を開始します。この時間の間、フェイルオーバー・プロセスが完了するまでは、データ・グリッドは、該当するトランザクションをどれも処理できなくなります。フェイルオーバー・プロセスが完了すると、キャッシュは、続行できるようになりますが、容量は減少します。データ・グリッド内の各コンピューターが 10 個の区画をサービスしていた場合、残りの 99 台のコンピューターのうち 10 台は、フェイルオーバー・プロセスの一部として少なくとも 1 つの余分の区画を受け取ることになります。余分の区画を 1 つ追加すると、該当コンピューターのワークロードは 10 パーセント以上増えます。データ・グリッドのスループットは、クロスデータ・グリッド・トランザクション内の最低速のコンピューターのスループットに制限されるので、平均して、スループットは 10 パーセント減少します。

WebSphere eXtreme Scale のような高可用性の分散オブジェクト・キャッシュでのスケールアウトの場合は、単一区画トランザクションのほうがクロスデータ・グリッド・トランザクションよりも適しています。こうした種類のシステムのパフォーマンスを最大限にするには、従来のリレーショナルの方法論とは異なる手法を使用する必要がありますが、クロスデータ・グリッド・トランザクションをスケーラブルな単一区画トランザクションに変えることができます。

スケーラブル・データ・モデルのビルドのベスト・プラクティス

WebSphere eXtreme Scale のような製品でのスケーラブル・アプリケーションをビルドする際のベスト・プラクティスには、基本原則と実装ヒントという 2 つのカテゴリーがあります。基本原則は、データ自体の設計に取り込む必要がある中心的なアイデアです。こうした原則を守らないアプリケーションは、たとえそのメインライン・トランザクションに対しても、適切に拡大できる可能性が低くなります。実装ヒントは、スケーラブル・データ・モデルの本来は一般的な原則に従って適切に設計されたアプリケーション内の問題のあるトランザクションに適用されます。

基本原則

スケーラビリティーを最適化する重要な手段の一部として、基本的な概念または原則を考慮する必要があります。

正規化に代わる重複

WebSphere eXtreme Scale のような製品の場合、その製品が多数のコンピューター間でデータを展開できるように設計されているということを念頭に入れておくことが重要です。ほとんどまたはすべてのトランザクションを単一

区画で完全なものとすることが目標である場合は、データ・モデル設計で、トランザクションが必要とする可能性のあるすべてのデータがその区画に存在するようにする必要があります。ほとんどの場合、データを複製することによってのみ、この目標を実現できます。

例えば、メッセージ・ボードのようなアプリケーションを考えてみます。メッセージ・ボードの 2 つの極めて重要なトランザクションとして、一定のユーザーからのすべてのポスト・メッセージを表示するものと、特定のトピックに関するすべてのポスト・メッセージを表示するものがあります。まずこうしたトランザクションがユーザー・レコード、トピック・レコード、さらに実際のテキストが含まれるポスト・レコードを含む正規化されたデータ・モデルをどのように扱うかを考えてみます。ポスト・メッセージがユーザー・レコードによって区画に分割されている場合、トピックを表示することは、クロスグリッド・トランザクションとなります。またその逆もいえます。トピックおよびユーザーは、多対多の関係を持っているので一緒に区画に分割することはできません。

このメッセージ・ボードの拡大を行う最善の策は、ポスト・メッセージを複製して、トピック・レコードを持つコピーを 1 つ、ユーザー・レコードを持つコピーを 1 つ保存することです。この結果、ユーザーからのポスト・メッセージを表示することは単一区画トランザクションとなり、トピックに関するポスト・メッセージを表示することは単一区画トランザクションとなり、ポスト・メッセージを更新または削除することは、2 区画トランザクションとなります。データ・グリッド内のコンピューターの数が増えるにつれ、これら 3 つのトランザクションがすべて直線的に拡大します。

リソースに代わるスケーラビリティー

非正規化されたデータ・モデルを考慮する場合に克服すべき最大の障害は、こうしたモデルがリソースに与える影響です。ある種のデータのコピーを 2 つ、3 つ、またはそれ以上保持すると、利用される資源が多すぎるように見える場合があります。こうしたシナリオに直面したら、ハードウェア・リソースが年々低価格になっているという事実を思い出してください。第 2 に(さらに重要)、WebSphere eXtreme Scaleは、追加資源のデプロイに関連した隠れコストを削減します。

メガバイトやプロセッサーといったコンピューター関連ではなく、コスト関連でリソースを測定してください。正規化された関係データを扱うデータ・ストアは、一般的に同じコンピューターに存在する必要があります。こうしたコロケーションの必要性から、いくつか小型コンピューターを購入するのではなく、1 台の大型の企業向けコンピューターを購入したほうがよいという結果が導かれます。ただし企業向けハードウェアの場合、通常では、毎秒 100 万のトランザクションの実行が可能な 1 台のコンピューターを使用するほうが、それぞれ毎秒 10 万のトランザクションの実行が可能な 10 台のコンピューターを結合した場合よりコストがかなりかかることは珍しいことではありません。

リソースを追加する際のビジネス・コストも存在します。ビジネスが成長していくと、結果的に容量不足となります。容量不足となると、より大型の高速コンピューターに移行する際にシャットダウンが必要になるか、切り替え

可能な第 2 の実稼働環境の作成が必要になります。いずれにせよ、ビジネス損失が発生するか、遷移期間にほぼ 2 倍の容量の維持が必要になるという形で追加コストが発生します。

WebSphere eXtreme Scale を使用すると、容量追加のためにアプリケーションをシャットダウンする必要がなくなります。ビジネスで翌年に 10 パーセントの追加容量が必要になることが見込まれた場合、データ・グリッド内のコンピューターの数を 10 パーセント増加します。このパーセンテージ分の増加の際に、アプリケーション・ダウン時間もなく、超過容量の購入の必要もありません。

データ形式変更の防止

WebSphere eXtreme Scale を使用している場合、データは、ビジネス・ロジックで直接消費可能な形式で保管されます。データをよりプリミティブな形式に分解することには、コストがかかります。データの書き込みおよび読み取り時に、変換を実行する必要があります。リレーショナル・データベースを使用する場合、データが最終的にディスクにパーシストされることがごく頻繁に行われるため、この変換は必要に応じて実行されますが、WebSphere eXtreme Scale を使用すると、こうした変換を実行する必要がなくなります。データは大部分メモリーに保管されるため、アプリケーションが必要とするそのままの形式で保管することができます。

この単純な規則に従うと、最初の原則に従ってデータを非正規化するのに役立ちます。ビジネス・データ用の最も一般的なタイプの変換は、正規化されたデータをアプリケーションのニーズに合う結果セットに変えるために必要な JOIN 演算です。データを正しい形式で保管すると、暗黙的にこうした JOIN 演算の実行が避けられ、非正規化されたデータ・モデルが作成されます。

未結合照会の除去

いくらデータを適切に構成しても、未結合照会は正しく拡張されません。例えば、値でソートされたすべての項目のリストを要求するようなトランザクションは使用しないでください。こうしたトランザクションは、はじめのうち合計項目数が 1000 であると、機能するかもしれませんが、合計項目数が 1000 万に達すると、トランザクションは 1000 万すべての項目を戻します。このトランザクションを実行した場合、最も考えられる 2 つの結果は、トランザクションのタイムアウトになるか、クライアントにメモリー不足エラーが発生するかのいずれかです。

最善のオプションは、上位 10 または 20 の項目だけが戻されるように、ビジネス・ロジックを変更することです。このロジック変更によって、キャッシュ内の項目数に関係なく、トランザクションのサイズが管理可能な程度に保たれます。

スキーマの定義

データの正規化の主な利点は、データベース・システムが状況の背後にあるデータの整合性を考慮できることです。データがスケーラビリティーのために非正規化されると、この自動データ整合性管理は存在しなくなります。データの整合性を保証するために、アプリケーション層で機能できるか、分散データ・グリッドに対するプラグインとして機能できるデータ・モデルを実装する必要があります。

メッセージ・ボードの例を考えてみます。トランザクションがトピックからポスト・メッセージを除去した場合、ユーザー・レコード上の重複するポスト・メッセージを除去する必要があります。データ・モデルがなくても、開発者は、トピックからポスト・メッセージを除去し、さらに確実にユーザー・レコードからそのポスト・メッセージを除去するアプリケーション・コードを作成することができます。ただし、仮に開発者がキャッシュと直接に対話する代わりにデータ・モデルを使用していたとしても、データ・モデル上の removePost メソッドによって、ポスト・メッセージからユーザー ID を抜き出して、ユーザー・レコードを検索し、この状況の背後にある重複ポスト・メッセージを除去することができます。

あるいは、実際の区画で実行し、トピックの変更を検出して、ユーザー・レコードを自動的に調整するリスナーを実装することができます。リスナーは、役に立ちます。区画がユーザー・レコードを持つようになった場合に、ユーザー・レコードの調整がローカルで可能になるか、ユーザー・レコードが異なる区画にあっても、トランザクションがクライアントとサーバーの間ではなく、サーバー間で実行されるためです。サーバー間のネットワーク接続のほうが、クライアントとサーバーの間のネットワーク接続よりも高速である可能性があります。

競合の防止

グローバル・カウンターを持つようなシナリオは避けてください。1 つのレコードが残りのレコードと比べて極端に多く使用されている場合は、データ・グリッドは拡張されません。データ・グリッドのパフォーマンスは、この特定のレコードを保持するコンピューターのパフォーマンスによって制限されています。

このような状態では、そのレコードを区画単位で管理できるように分割してみてください。例えば、分散キャッシュ内の合計エントリー数を戻すトランザクションを考えます。すべての挿入および除去操作で増大する単一のレコードにアクセスする代わりに、各区画のリスナーに挿入および除去操作を追跡させます。このリスナーによるトラッキングを使用すると、挿入および除去を単一区画操作とすることができます。

カウンターの読み取りはクロスデータ・グリッド操作となりますが、ほとんどの場合、それは元々クロスデータ・グリッド操作と同じく非効率的です。そのパフォーマンスがレコードをホストするコンピューターのパフォーマンスと関係しているためです。

実装ヒント

最善のスケーラビリティーを達成するには、以下のヒントも考慮してください。

逆引き索引の使用

顧客レコードが顧客 ID 番号に基づいて区画化されるような適切に非正規化されたデータ・モデルを考えます。この区画化方法は論理的な選択といえます。顧客レコードによって実行されるほぼすべてのビジネス・オペレーションは、顧客 ID 番号を使用するからです。ただし、顧客 ID 番号を使用しない重要なトランザクションに、ログイン・トランザクションがあります。ログインには顧客 ID 番号よりもユーザー名や電子メール・アドレスが使用されるほうが一般的です。

ログイン・シナリオの簡単な方法は、顧客レコードを見つけるためにクロスデータ・グリッド・トランザクションを使用することです。先に説明したように、この方法は拡張されません。

次のオプションとして、ユーザー名または電子メールに基づいて区画化することがあります。このオプションは、顧客 ID に基づくすべての操作がクロスデータ・グリッド・トランザクションとなるので、実用的でありません。またサイトのユーザーがユーザー名や電子メール・アドレスを変更したい場合もあります。WebSphere eXtreme Scale のような製品は、データをその不変性の維持のために区画化するのに使用される値を必要とします。

適切な解決方法として、逆引き索引を使用することができます。WebSphere eXtreme Scale を使用すると、すべてのユーザー・レコードを保持するキャッシュと同じ分散グリッドにキャッシュを作成できます。このキャッシュは、高可用性で、区画化され、しかもスケーラブルです。このキャッシュは、ユーザー名または電子メール・アドレスを顧客 ID にマップするために使用できます。このキャッシュでは、ログインは、クロスグリッド操作ではなく 2 区画操作となります。このシナリオは単一区画トランザクションほどよくはありませんが、コンピューターの数が増えるにつれ、スループットが直線的に増加します。

書き込み時の計算

平均や合計などの一般的な計算値は、作成にコストがかかることがあります。こうした操作には、通常膨大な数のエントリーを読み取る必要があるためです。ほとんどのアプリケーションでは、読み取りのほうが書き込みよりも一般的であるため、こうした値を書き込み時に計算し、結果をキャッシュに保管するほうが効率的です。これにより、読み取り操作は高速になり、よりスケーラブルになります。

オプション・フィールド

業務内容、自宅住所、および電話番号を保持するユーザー・レコードを考えます。これらすべてが定義されているユーザーもいれば、まったく定義されていないユーザーもいれば、一部が定義されているユーザーもいます。データが正規化されていると、ユーザー・テーブルおよび電話番号テーブルが存在することになります。一定ユーザーの電話番号は、この 2 つのテーブル間の JOIN 操作を使用して検出できます。

このレコードを非正規化する場合、データの重複は必要ありません。ほとんどのユーザーが電話番号を共有しないためです。代わりに、ユーザー・レコードで空スロットを使用できるようになっている必要があります。電話番号テーブルを使用する代わりに、各ユーザー・レコードに電話番号タイプごとに 1 つずつ 3 つの属性を追加します。この属性の追加により、JOIN 操作がなくなり、ユーザーの電話番号検索が単一区画操作となります。

多対多関係の配置

製品とその販売店を追跡するアプリケーションを考えてみます。1 つの製品が多くの店舗で販売され、1 つの店舗で多くの製品が販売されます。このアプリケーションが 50 の大規模小売業者を追跡するものとします。各製品が最大 50 の店舗で販売され、それぞれの店舗で何千もの製品が販売されます。

各店舗エンティティー内に製品リストを保持する (配置 B) 代わりに、製品エンティティーの内部に店舗リストを保持します (配置 A)。このアプリケーションが実行する必要があるトランザクションの一部を見ると、配置 A がよりスケーラブルである理由が明らかになります。

まず更新に注目します。配置 A では、店舗の在庫から製品を除去する場合、製品エンティティーがロックされます。データ・グリッドに 10000 の製品が保持されている場合、グリッドの 1/10000 しか更新の実行をロックする必要がありません。配置 B では、データ・グリッドには 50 の店舗しか含まれていないので、更新を完了するには、グリッドの 1/50 をロックする必要があります。これらは両方とも単一区画操作と考えることができますが、配置 A のほうがより効率よくスケールアウトされます。

現在、配置 A による読み取りを考えていますから、トランザクションで少量のデータのみが転送されるため、製品の販売店舗の検索は拡張され、高速な単一区画トランザクションとなります。配置 B では、製品が店舗で販売されているかどうかを確認するために、各店舗エンティティーにアクセスする必要があるため、このトランザクションはクロスデータ・グリッド・トランザクションになります。これは、配置 A の大きなパフォーマンス上の利点を明らかにします。

正規化されたデータによる拡張

クロスデータ・グリッド・トランザクションの正当な使用法の 1 つにデータ処理の拡張があります。データ・グリッドに 5 台のコンピューターがあり、各コンピューターについて約 100,000 のレコード全部をソートするクロスグリッド・トランザクションがディスパッチされると、そのトランザクションは全体で 500,000 個のレコードをソートします。データ・グリッド内の最低速のコンピューターが毎秒これらのトランザクションのうちの 10 個を実行できる場合、データ・グリッドは全体で毎秒 5,000,000 レコードをソートできます。グリッド内のデータが 2 倍になると、各コンピューターは全体で 200,000 個のレコードをソートする必要があり、各トランザクションは全体で 1,000,000 個のレコードをソートします。このデータの増加は、最低速のコンピューターのスループットを毎秒 5 トランザクションに減少させるので、データ・グリッドのスループットは毎秒 5 トランザクションに減少します。それでもデータ・グリッドは全体で毎秒 5,000,000 レコードをソートします。

このシナリオでは、コンピューターの数を 2 倍にすると、各コンピューターは元の 100,000 レコードのソートという負荷状態に戻るので、最低速のコンピューターは、これらのトランザクションを毎秒 10 個で処理できるようになります。データ・グリッドのスループットは、毎秒 10 要求という同じ状態ですが、現在では各トランザクションは 1,000,000 レコードを処理するので、処理するレコードに関してはグリッドの容量は毎秒 10,000,000 レコードと 2 倍になります。

ユーザー数の増加に合わせてインターネットとスループットの規模を拡大するため、データ処理に関して両方を拡張する必要のある検索エンジンなどのアプリケーションでは、グリッド間の要求のラウンドロビンを備えた複数のデータ・グリッドを作成する必要があります。スループットを拡大する必要がある場合、要求をサービスするために、コンピューターを追加し、別のデ

ータ・グリッドを追加します。データ処理を拡大する必要がある場合、コンピューターを追加して、データ・グリッド数を一定に保ちます。

セキュリティーの概要

WebSphere eXtreme Scale はデータ・アクセスを保護し、外部セキュリティー・プロバイダーと統合することができます。

注: データベースなど、既存の非キャッシュ・データ・ストアでは、積極的に構成したり、有効にしたりする必要のない組み込みセキュリティー・フィーチャーがある可能性があります。ただし、eXtreme Scale でデータをキャッシュした後では、その結果として生じる、バックエンドのセキュリティー・フィーチャーが効力を持たなくなるような重要な状況を考慮する必要があります。eXtreme Scale セキュリティーを必要なレベルで構成すると、データの新しいキャッシュ・アーキテクチャーも保護できます。
以下に、eXtreme Scale セキュリティー機能について簡単に説明します。セキュリティーの構成について詳しくは、「*管理ガイド*」および「*プログラミング・ガイド*」を参照してください。

分散セキュリティーの基礎

分散 eXtreme Scale セキュリティーは、次の 3 つの主要概念に基づいています。

信頼できる認証
: 要求側 の ID を判別する能力。WebSphere eXtreme Scale は、クライアントとサーバー間の認証も、サーバー相互間の認証もともにサポートします。

許可
: 要求側にアクセス権を付与する許可を与える能力。WebSphere eXtreme Scale は、さまざまな操作に対しさまざまな許可をサポートします。

セキュア・トランスポート
: ネットワーク上での安全なデータ伝送。WebSphere eXtreme Scale は、Transport Layer Security/Secure Sockets Layer (TLS/SSL) プロトコルをサポートします。

認証

WebSphere eXtreme Scale は、分散クライアント・サーバー・フレームワークをサポートします。クライアント・サーバー・セキュリティー・インフラストラクチャーは、eXtreme Scale サーバーへのアクセスを安全にするために配置されています。例えば、認証が eXtreme Scale サーバーによって必要とされる場合、認証のための資格情報を eXtreme Scale クライアントがサーバーに提供する必要があります。これらの資格情報は、ユーザー名とパスワードのペア、クライアント証明書、Kerberos チケット、またはクライアントとサーバーが合意した形式で示されたデータなどです。

許可

WebSphere eXtreme Scale の許可は、サブジェクトおよびアクセス権に基づいています。Java 認証・承認サービス (JAAS) を使用してアクセスを許可したり、Tivoli® Access Manager (TAM) などのカスタム・アプローチを接続して許可を処理したりできます。クライアントまたはグループに対しては、以下の許可を与えることができます。

マップ許可
マップに対して挿入、読み取り、更新、除去、または削除の操作を実行することを許可します。

ObjectGrid 許可
ObjectGrid オブジェクトに対してオブジェクト照会またはエンティティー照会およびストリーム照会を実行することを許可します。

DataGrid エージェント許可
DataGrid エージェントを ObjectGrid へデプロイすることを許可します。

サーバー・サイド・マップ許可
サーバー・マップをクライアント・サイドに複製すること、またはサーバー・マップに動的索引を作成することを許可します。

管理許可
管理タスクを実行することを許可します。

トランスポート・セキュリティー

クライアント・サーバー通信を保護するため、WebSphere eXtreme Scale は TLS/SSL をサポートします。これらのプロトコルは、eXtreme Scale クライアントとサーバー間のセキュア接続のための、認証性、保全性、および機密性を備えたトランスポート層セキュリティーを提供します。

グリッド・セキュリティー

セキュア環境では、サーバーは他のサーバーの認証性を確認できる必要があります。WebSphere eXtreme Scale は、この目的のために共有秘密ストリングのメカニズムを使用します。この秘密鍵のメカニズムは、共有パスワードと同様です。すべての eXtreme Scale サーバーは、共有秘密ストリングについて同意します。データ・グリッドに加わるサーバーは、秘密ストリングを提示するよう求められます。参加しようとするサーバーの秘密ストリングがマスター・サーバーのものと一致すると、そのサーバーはグリッドに参加できます。一致しない場合、結合要求は拒否されます。

平文の機密事項の送信は保護されません。eXtreme Scale セキュリティー・インフラストラクチャーには、サーバーがこの機密事項を送信前に保護できるようにするため、SecureTokenManager プラグインが用意されています。セキュア操作の実装方法を選択できます。WebSphere eXtreme Scale は、セキュア操作が実装され、機密事項が暗号化され署名されるような実装を提供します。

動的デプロイメント・トポロジーでの Java Management Extensions (JMX) セキュリティー

JMX MBean セキュリティーは、すべてのバージョンの eXtreme Scale でサポートされています。カタログ・サーバー MBean およびコンテナー・サーバー MBean のクライアントを認証可能にして、MBean 操作へのアクセスを実施できるようになります。

ローカル eXtreme Scale セキュリティー

ローカル eXtreme Scale セキュリティーは、アプリケーションが ObjectGrid インスタンスを直接にインスタンス化して、使用するので、分散 eXtreme Scale モデルとは異なります。アプリケーションおよび eXtreme Scale インスタンスは、同じ Java 仮想マシン (JVM) 内にあります。このモデルにはクライアント/サーバーの概念が含まれていないので、認証はサポートされません。アプリケーションがそれ自身の認証を管理し、認証済みサブジェクト・オブジェクトを eXtreme Scale に渡す必要があります。ただし、ローカル eXtreme Scale プログラミング・モデルに使用される許可メカニズムは、クライアント/サーバー・モデルに使用されるものと同じです。

構成およびプログラミング

セキュリティーに関する構成とプログラミングについて詳しくは、外部プロバイダーとのセキュリティー統合およびセキュリティー APIを参照してください。

REST データ・サービスの概要

WebSphere eXtreme Scale REST データ・サービスは、Microsoft WCF Data Services (正式には ADO.NET Data Services) と互換性があり、Open Data Protocol (OData) を実装する Java HTTP サービスです。Microsoft WCF Data Services は、Visual Studio 2008 SP1 および .NET Framework 3.5 SP1 を使用する場合、この仕様と互換性があります。

互換性の要件

REST データ・サービスは、HTTP クライアントをデータ・グリッドにアクセスできるようにします。REST データ・サービスは、Microsoft .NET Framework 3.5 SP1 で提供される WCF Data Services サポートと互換性があります。Microsoft Visual Studio 2008 SP1 で提供されるリッチ・ツールを使用して RESTful アプリケーションを開発することができます。 この図では、WCF Data Services がクライアントおよびデータベースとどのように対話をするのかについて、概要が示されています。

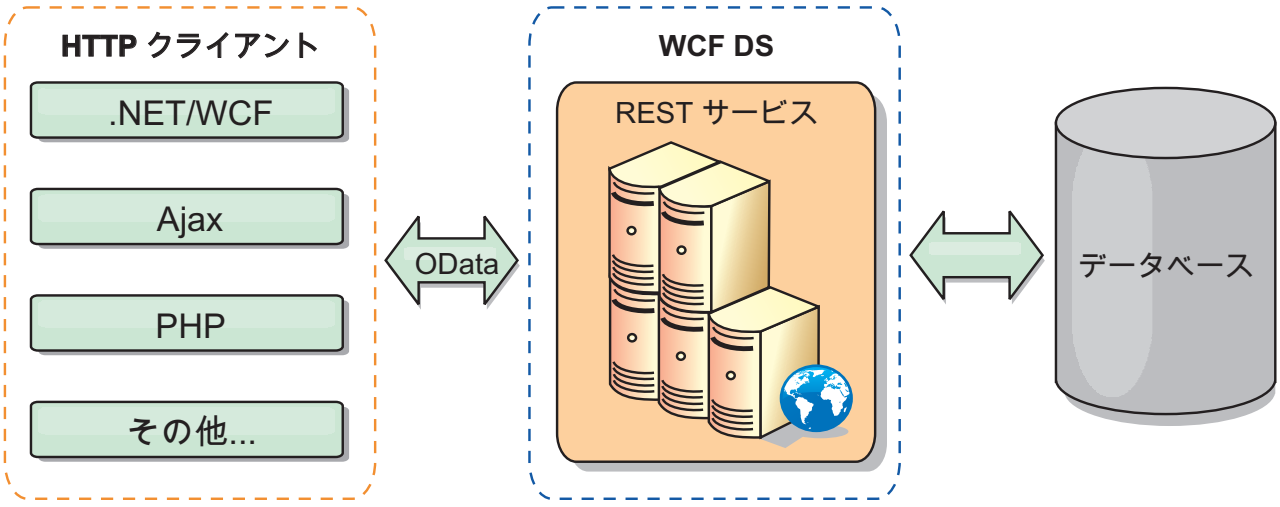


図 36. Microsoft WCF Data Services

WebSphere eXtreme Scale には Java クライアント用の機能の豊富な API セットが含まれています。 次の図で示されているように、REST データ・サービスは HTTP クライアントと WebSphere eXtreme Scale データ・グリッドの間のゲートウェイで、WebSphere eXtreme Scale クライアントを介してグリッドと通信します。REST データ・サービスは Java サーブレットで、これにより、WebSphere Application Server などの共通 Java Platform, Enterprise Edition (JEE) プラットフォームに対する柔軟なデプロイメントが可能です。REST データ・サービスは、WebSphere eXtreme Scale Java API を使用して WebSphere eXtreme Scale データ・グリッドと通信します。WCF Data Services クライアントまたはその他のクライアントが HTTP および XML と通信することができます。

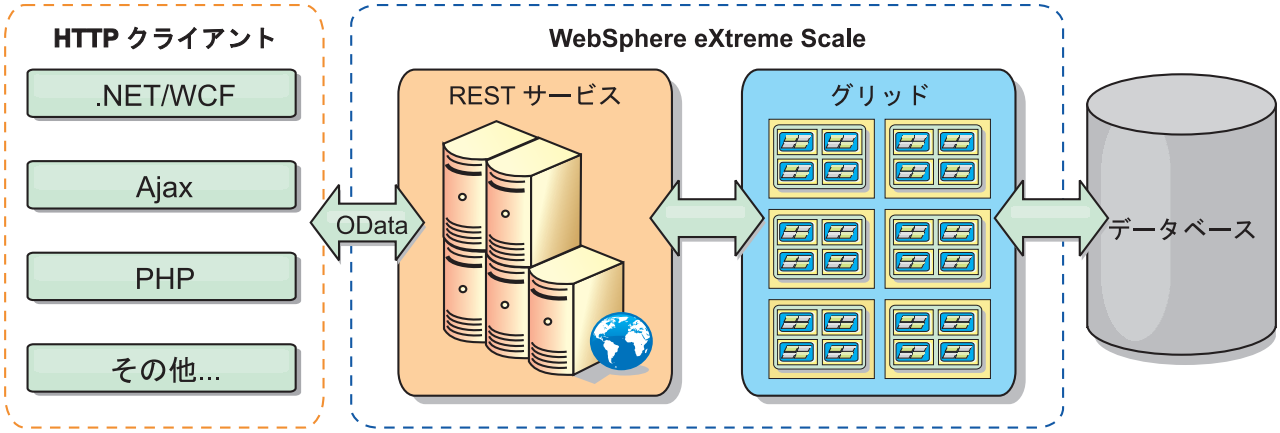


図 37. WebSphere eXtreme Scale REST データ・サービス

WCF Data Services について詳しくは、REST データ・サービスの構成 を参照するか、以下のリンクを使用してください。

- MicrosoftWCF Data Services デベロッパー・センター (Microsoft WCF Data Services Developer Center)
- MSDN の ADO.NET Data Services の概要 (ADO.NET Data Services overview on MSDN)
- 「ADO.NET Data Service を使用する」ホワイトペーパー (Whitepaper: Using ADO.NET Data Services)
- Atom Publishing Protocol: データ・サービス URI およびペイロード拡張 (Atom Publish Protocol: Data Services URI and Payload Extensions)
- 概念スキーマ定義ファイル形式 (Conceptual Schema Definition File Format)
- データ・サービス・パッケージング形式のエンティティー・データ・モデル (Entity Data Model for Data Services Packaging Format)
- OData プロトコル (Open Data Protocol)
- OData プロトコルのよくある質問 (Open Data Protocol FAQ)

フィーチャー

このバージョンの eXtreme Scale REST データ・サービスは、以下のフィーチャーをサポートします。

- WCF Data Services エンティティーとしての eXtreme Scale EntityManager API エンティティーの自動モデリングには、以下のサポートが組み込まれます。

  - Java データ型の Entity Data Model 型への変換
  - エンティティー・アソシエーションのサポート
  - 区画に分割されたデータ・グリッドに必要なスキーマ・ルートおよびキー・アソシエーションのサポート

  詳しくは、エンティティー・モデルを参照してください。
- Atom Publishing Protocol (AtomPub または APP) XML および JavaScript Object Notation (JSON) データ・ペイロード形式。
- それぞれの HTTP 要求メソッドを使用する作成、読み取り、更新、および削除 (CRUD) 操作である、POST、GET、PUT および DELETE。 さらに、Microsoft 拡張機能の MERGE がサポートされます。
- フィルターを使用した単純照会
- バッチ検索および変更設定要求
- 高可用性のための、区画に分割されたデータ・グリッドのサポート
- eXtreme Scale EntityManager API クライアントとのインターオペラビリティー
- 標準 JEE Web サーバーのサポート
- オプティミスティック並行性
- REST データ・サービスと eXtreme Scale データ・グリッドの間のユーザー許可およびユーザー認証

既知の問題と制限

- トンネル要求はサポートされません。

第 2 章 計画

WebSphere eXtreme Scale をインストールして、データ・グリッド・アプリケーションをデプロイする前に、キャッシング・トポロジーを決定し、キャパシティー・プランニングを実行し、ハードウェア要件およびソフトウェア要件、ネットワーキングとチューニングの設定などを検討する必要があります。運用チェックリストを使用して、アプリケーションをデプロイできる環境になっているかどうかを確認することもできます。

使用する WebSphere eXtreme Scale アプリケーションを設計する際に利用できるベスト・プラクティスについては、developerWorks®: ハイパフォーマンスで高い回復力を持つ WebSphere eXtreme Scale アプリケーションを作成するための原則とベスト・プラクティス (Principles and best practices for building high performing and highly resilient WebSphere eXtreme Scale application) の記事を参照してください。

トポロジーの計画

WebSphere eXtreme Scale を使用して、アーキテクチャーはローカルのメモリー内でのデータ・キャッシング、または分散クライアント/サーバーでのデータ・キャッシングを使用できます。 アーキテクチャーは、データベースとさまざまな関係を持つことができます。複数のデータ・センターに及ぶトポロジーを構成することもできます。

WebSphere eXtreme Scale を作動させるには、最低限の追加インフラストラクチャーが必要です。インフラストラクチャーは、サーバー上で Java Platform, Enterprise Edition アプリケーションをインストール、開始、および停止するためのスクリプトで構成されます。キャッシュ・データはコンテナー・サーバー内に保管され、クライアントはリモート側でサーバーに接続します。

メモリー内の環境

メモリー内のローカル環境にデプロイすると、WebSphere eXtreme Scale は、単一 Java 仮想マシン内で稼働するため、複製されません。ローカル環境を構成するには、ObjectGrid XML ファイルまたは ObjectGrid API を使用できます。

分散環境

分散環境にデプロイすると、WebSphere eXtreme Scale は Java 仮想マシンのセット内で稼働し、パフォーマンス、可用性、およびスケーラビリティーが向上します。この構成では、データのレプリカ生成および区画化の使用が可能です。また、既存の eXtreme Scale サーバーを再始動せずに、別のサーバーを追加することもできます。ローカル環境の場合と同じように、分散環境でも ObjectGrid XML ファイル、または同等のプログラマチック構成が必要です。構成詳細を持つデプロイメント・ポリシー XML ファイルも提供する必要があります。

単純なデプロイメントを作成することも、数千ものサーバーが必要になる大規模なテラバイト・サイズのデプロイメントを作成することもできます。

ローカルのメモリー内のキャッシュ

最も単純なケースでは、WebSphere eXtreme Scale は、ローカルの (非分散型の) メモリー内のデータ・グリッド・キャッシュとして使用できます。 ローカルのケースは、特に複数のスレッドにより一時データにアクセスして変更する必要がある、高い並行性を持つアプリケーションで有効になります。 ローカル・データ・グリッドに保持されるデータは、索引を付け、照会を使用して検索することができます。照会は、大規模なメモリー内データ・セットを処理するのに役立ちます。Java 仮想マシン (JVM)で提供されるサポートは、すぐに使用する準備ができているものの、データ構造に制限があります。

WebSphere eXtreme Scale でのローカルのメモリー内キャッシュ・トポロジーは、単一 Java 仮想マシン内で、一時データへの整合したトランザクション・アクセスを可能にするために使用されます。

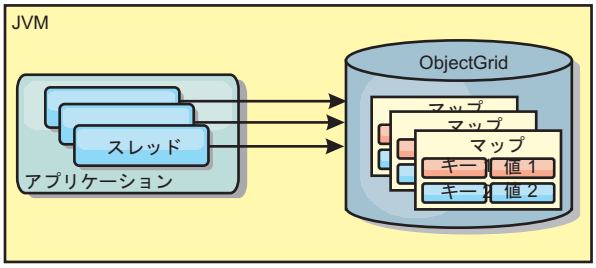


図 38. ローカルのメモリー内のキャッシュ・シナリオ

利点

- セットアップが簡単: ObjectGrid は、プログラマチックに作成することも、ObjectGrid デプロイメント記述子 XML ファイルまたは Spring などのその他のフレームワークを使用して宣言的に作成することもできます。
- 高速: 各 BackingMap は、最適のメモリー使用効率および並行性が得られるように独立して調整できます。
- 扱うデータ・セットが小さい単一 Java 仮想マシン・トポロジー、また頻繁にアクセスされるデータのキャッシングに最適。
- トランザクション型。BackingMap 更新は、単一の作業単位にまとめることができ、Java Transaction Architecture (JTA) トランザクションなどの 2 フェーズ・トランザクションの最終参加者として統合することができます。

欠点

- フォールト・トレラントでない。
- データは複製されない。メモリー内キャッシュは読み取り専用参照データに最適。
- スケーラブルでない。データベースが必要とするメモリーの量が Java 仮想マシンを圧倒するおそれがある。
- Java 仮想マシンを追加するときに、次のような問題が発生する。
  - データを簡単には区画化できない

- Java 仮想マシン間で状態を手動で複製しなければならない。そうしないと、各キャッシュ・インスタンスが同一データの別バージョンを保持するようになります
- 無効化にかかるコストが高い。
- 各キャッシュは個別にウォームアップが必要になる。ウォームアップは、有効なデータがキャッシュに設定されるようにデータをロードする期間です。

使用する場合

ローカルのメモリー内キャッシュのデプロイメント・トポロジーは、キャッシュに入れるデータ量が小さく (1 つの Java 仮想マシンに収まる場合)、比較的安定している場合に限って使用するようにしてください。このアプローチの場合、不整合データの存在を許容する必要があります。Evictor を使用して、最も使用頻度が高いデータまたは最近使用されたデータをキャッシュに保持するようにすると、キャッシュ・サイズを小さく維持し、データの関連性を高くすることができます。

ピア複製されるローカル・キャッシュ

独立したキャッシュ・インスタンスを持つプロセスが複数ある場合は、確実にキャッシュが同期されるようにする必要があります。キャッシュ・インスタンスが確実に同期されるようにするには、Java Message Service (JMS)を使用して、ピア複製されるキャッシュを有効にします。

WebSphere eXtreme Scale には、ピア ObjectGrid インスタンス間にトランザクション変更を自動的に伝搬する 2 つのプラグインがあります。
JMSObjectGridEventListener プラグインは、JMS を使用して、eXtreme Scale 変更を自動的に伝搬します。

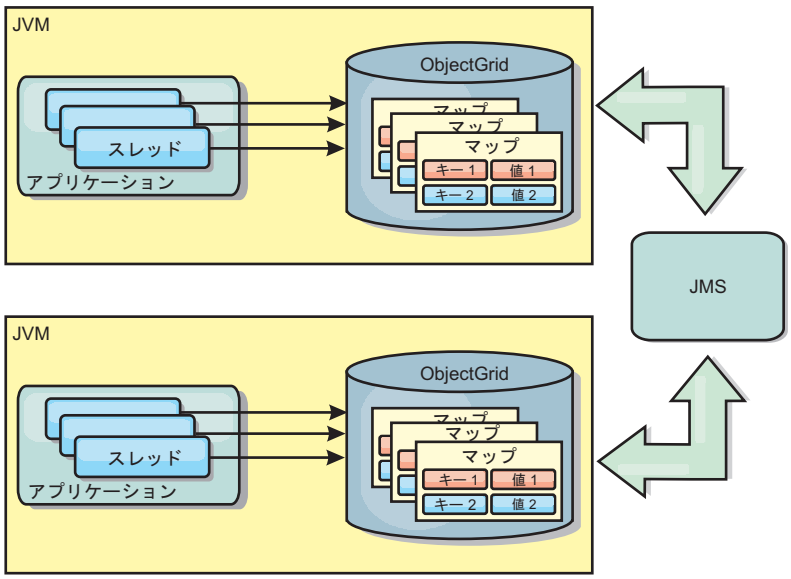


図 39. JMS によって変更が伝搬されるピア複製キャッシュ

WebSphere Application Server 環境を実行している場合は、TranPropListener プラグインも使用可能です。TranPropListener プラグインは、高可用性 (HA) マネージャー

を使用して、各ピア・キャッシュ・インスタンスに変更を伝搬します。

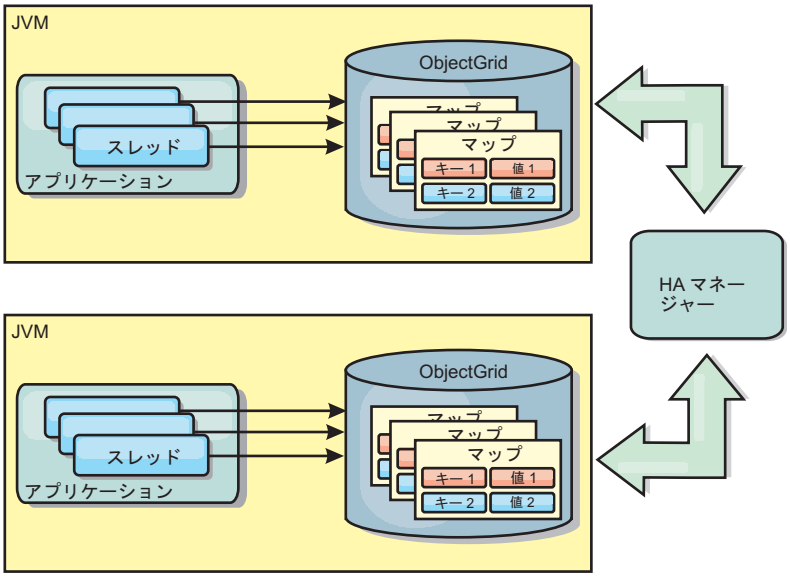


図 40. HA マネージャーによって変更が伝搬されるピア複製キャッシュ

利点

- より頻繁にデータが更新されるため、データが有効な場合が増えます。
- TranPropListener プラグインを使用すると、ローカル環境と同様、 eXtreme Scale デプロイメント記述子 XML ファイルや他のフレームワーク (Spring など) を使用して、eXtreme Scale をプログラマチックまたは宣言的に作成できます。HA マネージャーとの統合は自動的に行われます。
- 最適のメモリー使用効率および並行性が得られるように、各 BackingMap を独立して調整できます。
- BackingMap 更新は、単一の作業単位にまとめることができ、Java Transaction Architecture (JTA) トランザクションなどの 2 フェーズ・トランザクションの最終参加者として統合することができます。
- 十分小さなデータ・セットの少数 JVM トポロジー、または頻繁にアクセスされるデータのキャッシングに最適です。
- eXtreme Scale に対する変更は、すべてのピア eXtreme Scale インスタンスに複製されます。 変更は、永続サブスクリプションが使用されている限り、整合性が保たれます。

欠点

- JMSObjectGridEventListener の構成および保守は、複雑になる場合があります。eXtreme Scale は、eXtreme Scale デプロイメント記述子 XML ファイルまたは Spring などのその他のフレームワークを使用して、プログラマチックまたは宣言的に作成できます。
- スケーラブルではありません。データベースが必要とするメモリー量が、JVM の負担になる場合があります。
- Java 仮想マシンを追加する場合に不適切な機能:
  - データを簡単には区画化できない

- 無効化にコストがかかります。
- 各キャッシュは個別にウォームアップが必要になります。

使用する場合

デプロイメント・トポロジーは、キャッシュに入れるデータ量が小さく、1 つの JVM に収まり、かつ比較的安定している場合にのみ使用します。

組み込みキャッシュ

WebSphere eXtreme Scale グリッドは、組み込み eXtreme Scale サーバーとして既存のプロセス内で実行することも、外部プロセスとして管理することもできます。

組み込みグリッドは、WebSphere Application Server などのアプリケーション・サーバー内で実行する場合に便利です。組み込まれていない eXtreme Scale サーバーは、コマンド行スクリプトを使用し、Java プロセスで実行することによって開始できます。

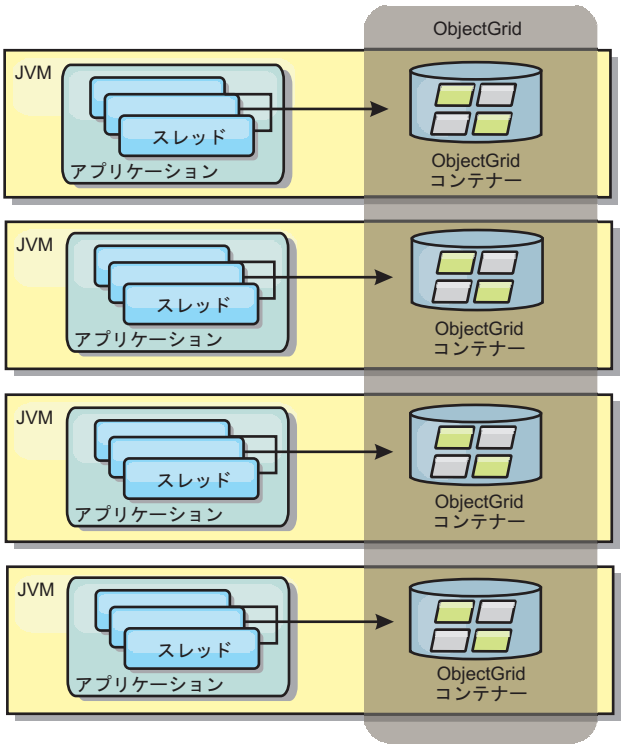


図 41. 組み込みキャッシュ

利点

- 管理するプロセスが減るため、管理が簡単になります。
- グリッドがクライアント・アプリケーションのクラス・ローダーを使用しているため、アプリケーションのデプロイメントが簡単です。
- 区画化と高可用性をサポートします。

欠点

- すべてのデータがプロセス内に連結されるため、クライアント・プロセスのメモリー占有スペースが増えます。
- クライアント要求にサービスを提供するための CPU 使用率が高くなります。
- クライアントがサーバーと同じアプリケーション Java アーカイブ・ファイルを使用しているため、アプリケーション・アップグレードの処理がさらに難しくなります。
- 柔軟性が低くなります。クライアントとグリッド・サーバーは、同じレートで拡張することができません。サーバーを外部で定義すると、プロセス数の管理の柔軟性が増します。

使用する場合

組み込みグリッドは、クライアント・プロセスにグリッド・データおよび潜在的なフェイルオーバー・データ用の空きメモリーが豊富にある場合に使用します。

詳しくは、*管理ガイド*のクライアント無効化メカニズムの使用可能化に関するトピックを参照してください。

分散キャッシュ

WebSphere eXtreme Scale は、共有キャッシュとして使用されることが最も多く、これまで使用されていたような従来のデータベースに代わり、データへのトランザクション・アクセスを複数のコンポーネントに提供します。共有キャッシュにより、データベースを構成する必要がなくなります。

キャッシュのコヒーレンス

すべてのクライアントがキャッシュ内の同じデータを見るので、キャッシュはコヒーレントです。各データはキャッシュ内の 1 つのサーバーのみに保管されるため、さまざまなバージョンのデータを保管することになりかねない、レコードの無駄なコピーが防止されます。コヒーレントなキャッシュは、より多くのサーバーがデータ・グリッドに追加されるにつれて、より多くのデータを保持することができ、グリッドのサイズが増えるのにつれて直線的に増加します。クライアントはこのデータ・グリッドからのデータに、リモート・プロシージャー・コールを使用してアクセスするので、このキャッシュはリモート・キャッシュまたは、ファー・キャッシュとも呼ばれます。データの区画化により、各プロセスは、全データ・セットの中から固有のサブセットを保持します。データ・グリッドが大きいほどより多くのデータを保持でき、そのデータに対するより多くの要求にサービスを提供できます。コヒーレントであることによって、失効データが存在しないため、データ・グリッドの周囲で無効化データをプッシュする必要がなくなります。コヒーレント・キャッシュは、各データの最新コピーのみを保持します。

WebSphere Application Server 環境を実行している場合は、TranPropListener プラグインも使用可能です。TranPropListener プラグインは、WebSphere Application Server 高可用性コンポーネント (HA マネージャー) を使用して、変更を各ピア ObjectGrid キャッシュ・インスタンスに伝搬します。

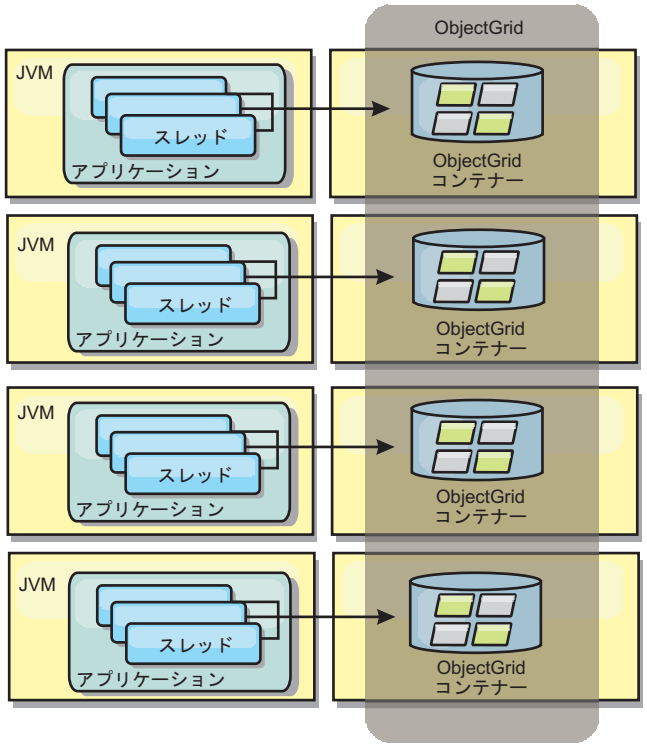


図 42. 分散キャッシュ

ニア・キャッシュ

クライアントは、eXtreme Scale が分散トポロジーで使用されている場合、オプションでローカルのインライン・キャッシュを持つことができます。オプションのこのキャッシュはニア・キャッシュと呼ばれます。これは、各クライアントにある独立した ObjectGrid であり、リモート用のキャッシュ (サーバー・サイド・キャッシュ) として機能します。ニア・キャッシュは、ロックがオプティミスティックまたはロックなしに構成されている場合、デフォルトで使用可能にされており、ロックがペシミスティックに構成されている場合は使用することができません。

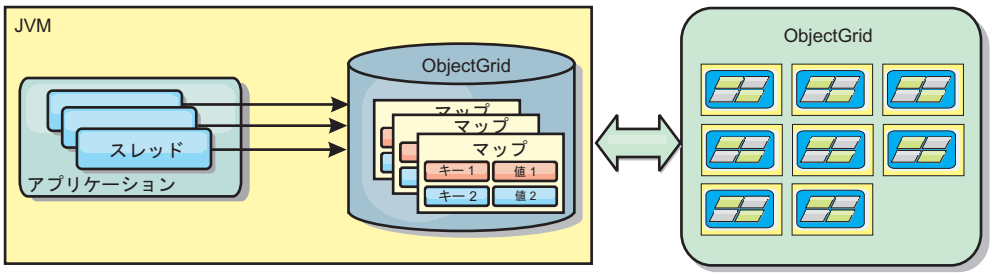


図 43. ニア・キャッシュ

ニア・キャッシュは、リモート側で eXtreme Scale サーバーに保管されているキャッシュ・データ・セット全体のサブセットへのメモリー内アクセスを可能にするため、非常に高速です。ニア・キャッシュは区画化されず、任意のリモート eXtreme Scale 区画からのデータを含みます。WebSphere eXtreme Scale は、以下のように、3 つまでのキャッシュ層を持つことができます。

1. トランザクション層キャッシュには、単一トランザクションのすべての変更が含まれます。トランザクション・キャッシュは、トランザクションがコミットされるまで、データの作業用コピーを保持します。クライアント・トランザクションが ObjectMap のデータを要求すると、最初にトランザクションがチェックされます。
2. クライアント層のニア・キャッシュは、サーバー層のデータのサブセットを保持します。トランザクション層にデータがない場合、データはクライアント層にあればクライアント層から取り出され、トランザクション・キャッシュに挿入されます。
3. サーバー層のデータ・グリッドには大半のデータが含まれ、すべてのクライアント間で共有されます。サーバー層は区画に分割できるので、大量のデータをキャッシュに入れることができます。クライアントのニア・キャッシュにデータが存在しないと、サーバー層からデータがフェッチされ、クライアント・キャッシュに挿入されます。サーバー層は、Loader プラグインを保持することもできます。グリッドに要求されたデータがない場合、Loader が呼び出され、結果のデータがバックエンドのデータ・ストアからグリッドに挿入されます。

ニア・キャッシュを使用不可にするには、クライアント・オーバーライド eXtreme Scale 記述子構成で numberOfBuckets 属性を 0 に設定します。eXtreme Scale のロック・ストラテジーについて詳しくは、マップ・エントリーのロックに関するトピックを参照してください。 ニア・キャッシュは、クライアント・オーバーライド eXtreme Scale 記述子構成を使用して、別の除去ポリシーや異なるプラグインを使用するように構成することもできます。

利点

- データへのアクセスがすべてローカルで行われるため、応答時間が速くなります。ニア・キャッシュ内でデータを探すことで、まず、サーバーのグリッドにいく手間が省け、リモート・データでさえもローカルでアクセス可能になります。

欠点

- 各層のニア・キャッシュはデータ・グリッド内の現行データと同期していない場合があるため、失効データの期間が長くなります。
- メモリー不足を回避するため、エビクターに頼り、データを無効化する必要があります。

使用する場合

応答時間が重要で、失効したデータは許容できる場合に使用します。

データベース統合: 後書き、インライン、およびサイド・キャッシング

WebSphere eXtreme Scale が使用される目的は、従来のデータベースをその背後に置くことで、通常はデータベースにプッシュされる読み取りアクティビティーをなくすことです。コヒーレント・キャッシュは、オブジェクト関連マッパーを直接または間接に使用することにより、アプリケーションで使用できます。コヒーレント・キャッシュは、データベースまたは読み取りからの下流工程の負荷を軽減します。シナリオがもう少し複雑で、一部のデータのみが従来のパーシスタンス保証を必要

とするデータ・セットへのトランザクション・アクセスなどの場合は、フィルター操作を使用して書き込みトランザクションの負荷を軽減します。

WebSphere eXtreme Scale は、高度にフレキシブルなメモリー内のデータベース処理スペースとして機能するように構成できます。ただし、WebSphere eXtreme Scale は、オブジェクト・リレーショナル・マッパー (ORM) ではありません。データ・グリッドに含まれているデータがどこから取得されたのかを認識しません。アプリケーションまたは ORM は、データを eXtreme Scale サーバーに配置できます。データの発生元であるデータベースとの一貫性を保つのは、データのソースの責任です。これは、データベースから取り出されたデータを eXtreme Scale は自動的に無効化できないことを意味します。アプリケーションまたはマッパーは、この機能を提供して、eXtreme Scale に保管されているデータを管理する必要があります。

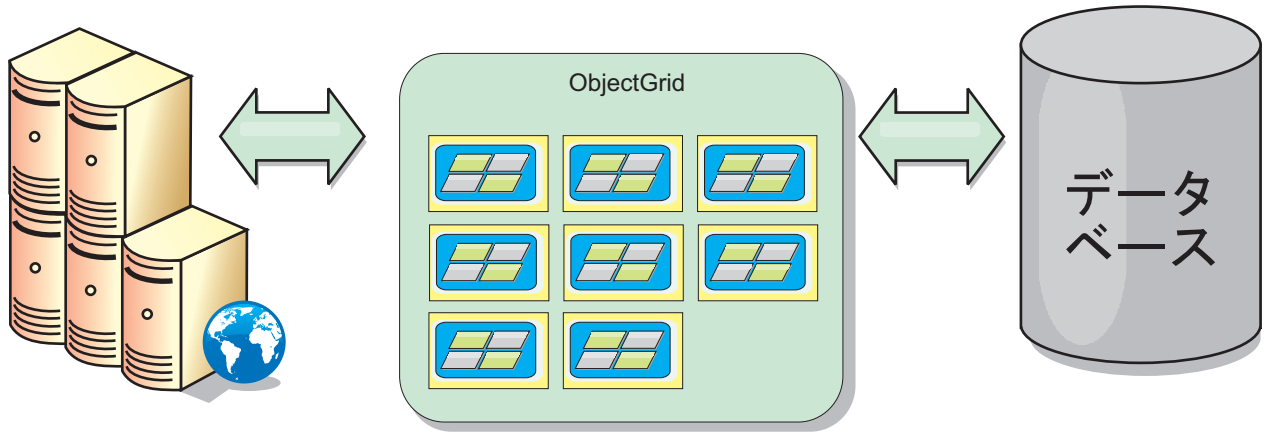


図 44. データベース・バッファーとしての ObjectGrid

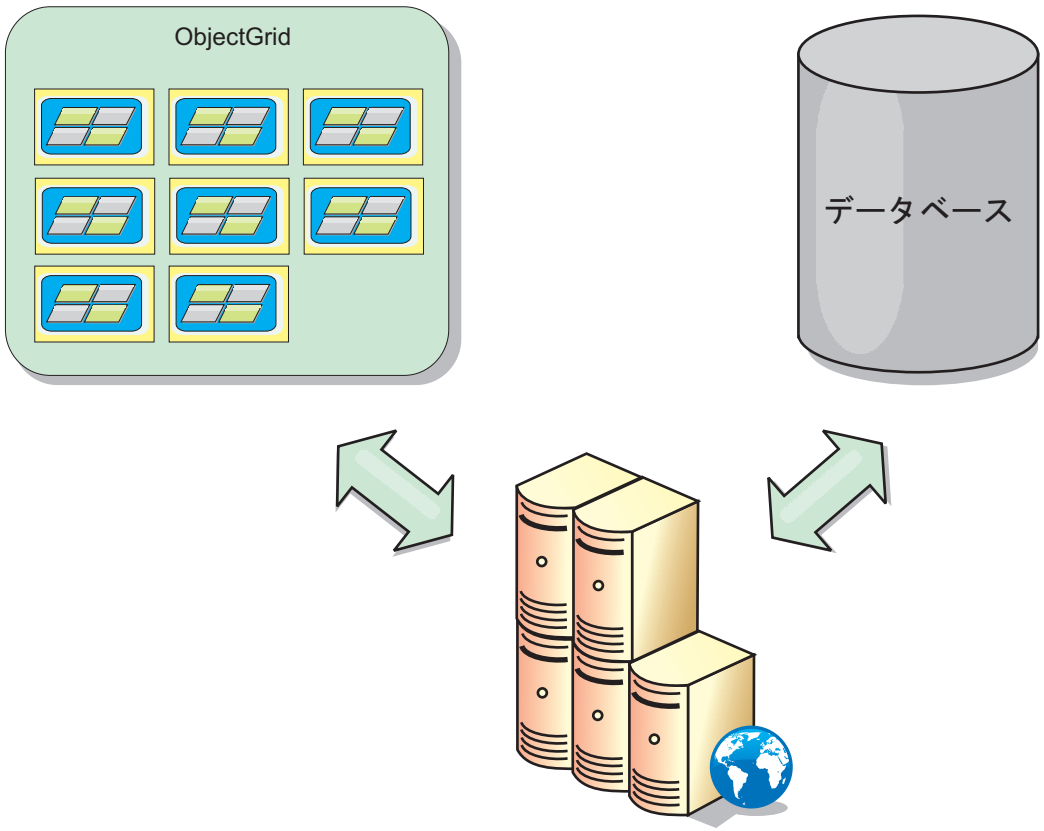


図 45. サイド・キャッシュとしての ObjectGrid

スパース・キャッシュおよび完全キャッシュ

WebSphere eXtreme Scale は、スパース・キャッシュまたは完全キャッシュとして使用できます。完全キャッシュがデータすべてを保持する一方で、スパース・キャッシュはデータ全体のサブセットしか保持しません。必要時には、データをゆっくりと取り込むことができます。通常、スパース・キャッシュは、データが部分的にしか使用可能でないため、キーを使用して (索引や照会を使用せず) アクセスされます。

スパース・キャッシュ

キーがスパース・キャッシュに存在しない場合、またはデータが使用できず、キャッシュ・ミスが発生している場合は、次の層が呼び出されます。データは、例えば、データベースからフェッチされ、データ・グリッド・キャッシュ層に挿入されます。照会または索引を使用する場合、現在ロードされている値のみがアクセスされ、要求は他の層に転送されません。

完全キャッシュ

完全キャッシュには必要なすべてのデータが含まれ、索引または照会により非キー属性を使用してアクセスできます。データベースから完全キャッシュにデータがプリロードされた後、アプリケーションはデータへのアクセスを試みます。データがロードされた後は、完全キャッシュをデータベースの代わりとして使用できます。

すべてのデータがあるので、照会および索引を使用して、データの検出と集約を行うことができます。

サイド・キャッシュ

WebSphere eXtreme Scale をサイド・キャッシュとして使用する場合は、データ・グリッドと一緒にバックエンドが使用されます。

サイド・キャッシュ

アプリケーションのデータ・アクセス層のサイド・キャッシュとしてこの製品を構成できます。このシナリオの場合、WebSphere eXtreme Scale は、通常であればバックエンド・データベースから取得されるオブジェクトを一時的に保管するために使用されます。アプリケーションは、データがデータ・グリッドに含まれているかどうかチェックします。データがデータ・グリッドにあった場合、そのデータが呼び出し元に返されます。データがない場合、データがバックエンド・データベースから取得されます。そして、次の要求がキャッシュ・コピーを使用できるように、データがデータ・グリッドに挿入されます。次の図は、OpenJPA や Hibernate などの任意のデータ・アクセス層で WebSphere eXtreme Scale をサイド・キャッシュとして使用する方法を示しています。

Hibernate および OpenJPA 向けサイド・キャッシュ・プラグイン

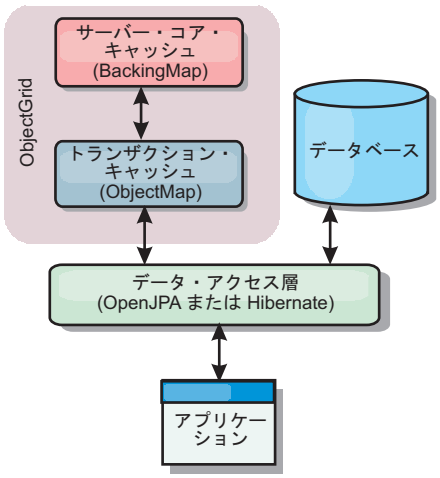


図 46. サイド・キャッシュ

WebSphere eXtreme Scale には、この製品を自動サイド・キャッシュとして使用できるようにする、OpenJPA 用と Hibernate 用のどちらのキャッシュ・プラグインも組み込まれています。WebSphere eXtreme Scale をキャッシュ・プロバイダーとして使用すると、データの読み取りおよび照会時のパフォーマンスが高まり、データベースへの負荷が軽減されます。WebSphere eXtreme Scale ではキャッシュが自動的にすべてのプロセス間で複製されるので、組み込みキャッシュ実装をしのぐ利点があります。あるクライアントが値をキャッシュに入れると、他のすべてのクライアントがキャッシュに入れられた値を使用できるようになります。

インライン・キャッシュ

インライン・キャッシングは、データベース・バックエンドに構成することも、データベースのサイド・キャッシュとして構成することもできます。インライン・キ

ャッシングは、データと対話するための基本手段として eXtreme Scale を使用します。eXtreme Scale がインライン・キャッシュとして使用される場合、アプリケーションは、Loader プラグインを使用してバックエンドと対話します。

インライン・キャッシュ

インライン・キャッシュとして使用される場合、WebSphere eXtreme Scale は Loader プラグインを使用してバックエンドと対話します。このシナリオでは、アプリケーションが直接 eXtreme Scale API にアクセスできるため、データ・アクセスが単純化されます。キャッシュ内のデータとバックエンドのデータが確実に同期されるようにするための数種類のキャッシング・シナリオが、eXtreme Scale においてポートされています。次の図は、インライン・キャッシュがアプリケーションおよびバックエンドと対話する方法を示しています。

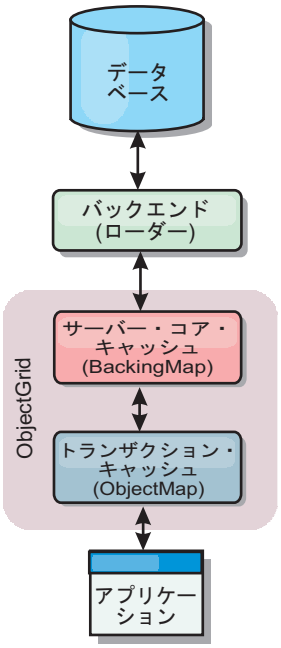


図 47. インライン・キャッシュ

インライン・キャッシング・オプションにより、アプリケーションが eXtreme Scale API に直接アクセスできるようになるため、データ・アクセスが単純化されます。WebSphere eXtreme Scale は、以下のような複数のインライン・キャッシング・シナリオをサポートします。

- リードスルー
- ライトスルー
- 後書き

リードスルー・キャッシングのシナリオ

リードスルー・キャッシュは、データ・エントリーの要求時にキーによるそのロードが暫時的に行われるスパース・キャッシュです。これが行われる場合、呼び出し元は、エントリーがどのように取り込まれるかを知る必要はありません。データが eXtreme Scale キャッシュに見つからない場合、eXtreme Scale は、その欠落データを Loader プラグインから取得します。このプラグインは、バックエンド・データ

ベースからデータをロードして、そのデータをキャッシュに挿入します。同じデータ・キーに対する後続の要求は、削除、無効化、または除去されるまでキャッシュに存在します。

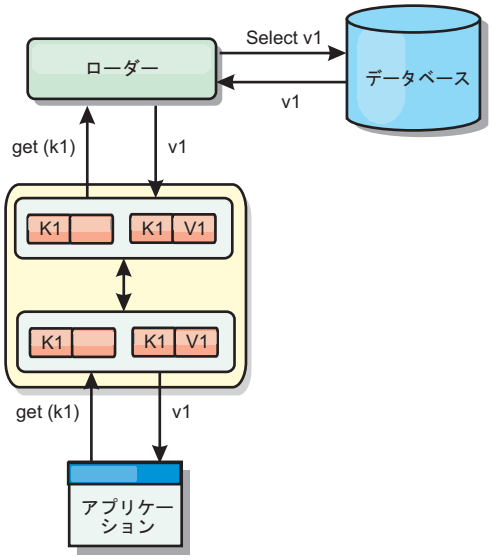


図 48. リードスルー・キャッシング

ライトスルー・キャッシングのシナリオ

ライトスルー・キャッシュでは、キャッシュへの書き込みが行われるたびに、ローダーを使用してデータベースへの書き込みが同期的に行われます。このメソッドでは、バックエンドとの整合性はありますが、データベース操作が同期されるため、書き込みパフォーマンスは低下します。キャッシュとデータベースがともに更新されるため、同じデータに対する後続の読み取りはキャッシュに残り、データベース呼び出しが回避されます。ライトスルー・キャッシュは、多くの場合、リードスルー・キャッシュと一緒に使用されます。

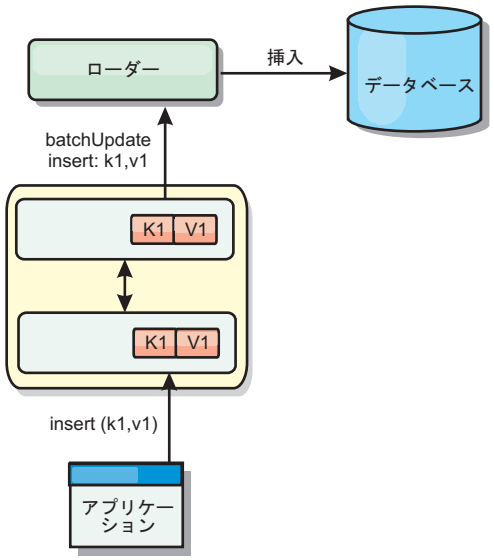


図 49. ライトスルー・キャッシング

後書きキャッシングのシナリオ

変更を非同期的に書き込むことにより、データベースの同期性が改善されます。後書きキャッシュまたはライト・バック・キャッシュとも呼ばれます。通常はローダーに対して同期的に書き込まれる変更は、eXtreme Scale 内でバッファー化されてから、バックグラウンド・スレッドを使用してデータベースに書き込まれます。データベース操作をクライアント・トランザクションから除去し、データベース書き込みを圧縮できるため、書き込みパフォーマンスが著しく向上します。

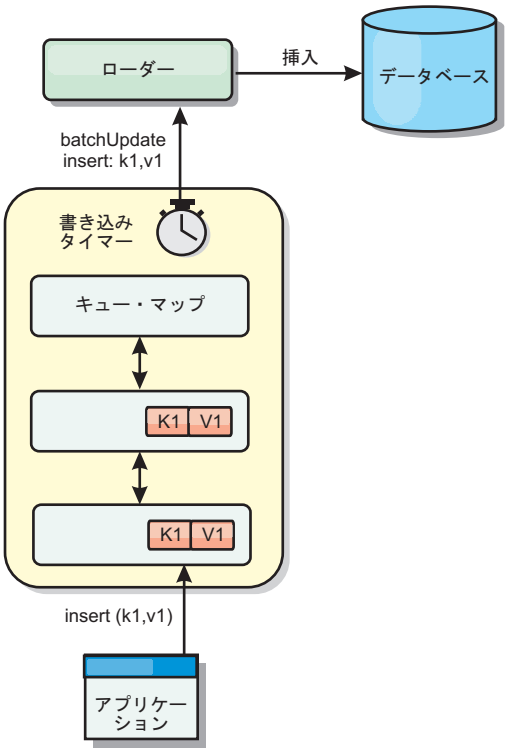


図 50. 後書きキャッシング

後書きキャッシング

後書きキャッシングを使用して、バックエンドとして使用しているデータベースを更新する際に発生するオーバーヘッドを減らすことができます。

後書きキャッシングの概要

後書きキャッシングでは、Loader プラグインの更新が非同期にキューに入れられます。eXtreme Scale トランザクションをデータベース・トランザクションから分離することにより、マップの更新、挿入、および除去の、パフォーマンスを改善できます。非同期更新は、時間ベースの遅延 (例えば 5 分) またはエントリー・ベースの遅延 (例えば 1000 エントリー) 後に実行されます。

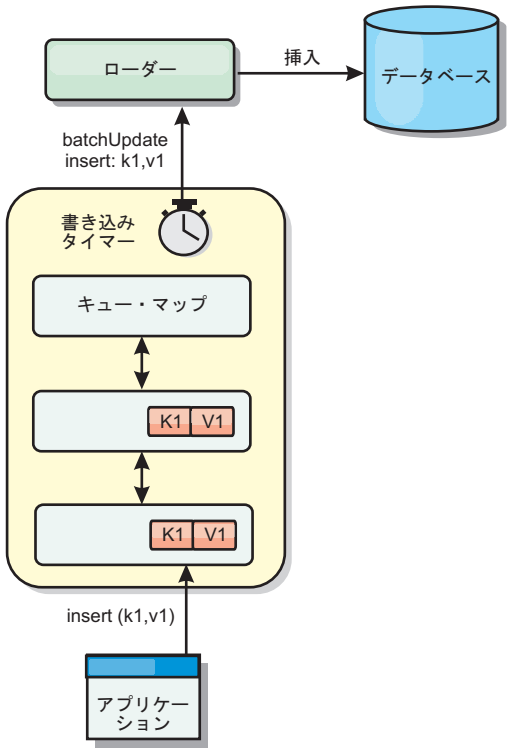


図 51. 後書きキャッシング

BackingMap の後書き構成により、ローダーとマップとの間にスレッドが作成されます。次に、ローダーは、BackingMap.setWriteBehind メソッド内の構成設定に従って、そのスレッドを通してデータ要求を委任します。eXtreme Scale トランザクションが、マップのエントリーを挿入、更新、または削除すると、これらの各レコードごとに 1 つずつ LogElement オブジェクトが作成されます。これらのエレメントは後書きローダーに送信され、キュー・マップと呼ばれる特別な ObjectMap 内でキューに入れられます。後書き設定が有効になっているバックアップ・マップは、それぞれ独自のキュー・マップを持っています。後書きスレッドは、キューに入れられたデータをキュー・マップから定期的に除去して、実際のバックエンド・ローダーにプッシュします。

後書きローダーは、挿入、更新、および削除タイプの LogElement オブジェクトのみを実際のローダーに送信します。それ以外のタイプの LogElement オブジェクト (例えば、EVICT タイプ) はすべて無視されます。

後書きサポートは、eXtreme Scale をデータベースに組み込む際に使用する Loader プラグインの *拡張機能* です。例えば、JPA ローダーの構成については JPA ローダーの構成 の情報を参照してください。

利点

後書きサポートを使用可能にすると、以下のような利点があります。

- **バックエンド障害の分離:** 後書きキャッシングは、バックエンド障害からの分離層を提供します。バックエンドのデータベースで障害が発生すると、更新はキュー・マップ内でキューに入れられます。アプリケーションは、トランザクション

を eXtreme Scale に送り続けることができます。バックエンドが復旧すると、キュー・マップ内のデータはバックエンドにプッシュされます。

- **バックエンドの負荷の削減:** 後書きローダーは更新をキー単位でマージします。その結果、キュー・マップ内には、キーごとにマージされた更新が 1 つのみ存在します。このマージにより、バックエンド・データベースに対する更新の数が減ります。
- トランザクション・パフォーマンスの改善: データがバックエンドと同期されるのをトランザクションが待機する必要がないので、個別の eXtreme Scale トランザクション時間が削減されます。

アプリケーション設計に関する考慮事項

後書きサポートを使用可能にすることは簡単ですが、後書きサポートを扱うアプリケーションを設計する際には、注意すべき考慮事項があります。後書きサポートがない場合、ObjectGrid トランザクションにバックエンド・トランザクションが包含されます。 ObjectGrid トランザクションはバックエンド・トランザクションの開始前に開始し、バックエンド・トランザクションの終了後に終了します。

後書きサポートが有効な場合、ObjectGrid トランザクションは、バックエンド・トランザクションが開始する前に終了します。ObjectGrid トランザクションとバックエンド・トランザクションは切り離されます。

参照保全性の制約

後書きサポートで構成されているそれぞれのバックアップ・マップは、データをバックエンドにプッシュするための独自の後書きスレッドを持ちます。したがって、1 つの ObjectGrid トランザクションにさまざまなマップを更新するデータが含まれていても、バックエンドでは、それぞれ異なるバックエンド・トランザクションでデータの更新が行われます。例えば、トランザクション T1 はマップ Map1 のキー key1 とマップ Map2 のキー key2 を更新するとします。マップ Map1 に対する key1 更新は、1 つのバックエンド・トランザクションでバックエンドに対して更新され、マップ Map2 に対する key2 更新は、異なる後書きスレッドにより別のバックエンド・トランザクションでバックエンドに対して更新されます。Map1 に保管されたデータと Map2 に保管されたデータがバックエンドでの外部キー制約などの関係を持つ場合、更新が失敗する可能性があります。

バックエンド・データベースの参照保全性制約を設計するときは、順不同の更新に必ず対応できるようにしてください。

キュー・マップのロックの振る舞い

トランザクションの動作で他に大きく異なる点は、ロックの振る舞いです。ObjectGrid は、PESSIMISTIC、OPTIMISITIC、および NONE の 3 つの異なるロック・ストラテジーをサポートします。後書きキュー・マップは、 *バックアップ・マップに構成されているロック・ストラテジーに関係なく、ペシミスティック・ロック・ストラテジーを使用します。キュー・マップのロックを取得する操作には 2 つの異なるタイプがあります。

- ObjectGrid トランザクションのコミット時、またはフラッシュ (マップ・フラッシュまたはセッション・フラッシュ) の発生時、トランザクションはキュー・マップ内のキーを読み取り、キーに S ロックをかけます。

- ObjectGrid トランザクションのコミット時、トランザクションは、キーの S ロックを X ロックにアップグレードしようとします。

キュー・マップのこの余分な動作のため、ロックの動作に少々違いがあります。

- ユーザー・マップがペシミスティック・ロック・ストラテジーで構成されている場合、ロックの動作にほとんど違いはありません。フラッシュまたはコミットが呼び出されるたび、キュー・マップ内の同じキーに S ロックがかけられます。コミット時間中、ユーザー・マップ内のキーに X ロックが取得されるだけでなく、キュー・マップ内のキーに対しても X ロックが取得されます。
- ユーザー・マップが OPTIMISTIC または NONE ロック・ストラテジーで構成されている場合、ユーザー・トランザクションは PESSIMISTIC ロック・ストラテジーのパターンに従います。フラッシュまたはコミットが呼び出されるたびに、キュー・マップ内の同じキーに対して S ロックが取得されます。コミット時間の間、同じトランザクションを使用するキュー・マップ内のキーに対して X ロックが設定されます。

ローダー・トランザクションの再試行

ObjectGrid は、2 フェーズ・トランザクションまたは XA トランザクションをサポートしません。後書きスレッドは、キュー・マップからレコードを除去して、バックエンドに対してそのレコードを更新します。トランザクションの最中にサーバーに障害が起こると、一部のバックエンドの更新が失われる可能性があります。

後書きローダーは、失敗したトランザクションの書き込みを自動的に再試行し、データ損失を防ぐために未確定 LogSequence をバックエンドに送信します。このアクションを行うには、ローダーがべき等である必要があります。この意味は、Loader.batchUpdate(TxId, LogSequence) が同じ値で 2 回呼び出されたとき、それは適用された回数があたかも 1 回だったかのように、同じ結果を返すということです。ローダー実装は、この機能を使用可能にするため、RetryableLoader インターフェースを実装しなければなりません。詳しくは、API 資料を参照してください。

ローダーの障害

Loader プラグインは、バックエンド・データベースと通信できない場合、失敗することがあります。これは、データベース・サーバーまたはネットワーク接続がダウンしている場合に発生することがあります。後書きローダーは、更新をキューに入れ、データ変更を定期的にローダーにプッシュしようと試みます。ローダーは、LoaderNotAvailableException 例外をスローして、データベース接続の問題があることを ObjectGrid ランタイムに通知しなければなりません。

したがって、ローダー実装で、データ障害または物理的ローダー障害を識別できるようになっている必要があります。データ障害は LoaderException または OptimisticCollisionException としてスローまたは再スローされる必要がありますが、物理的なローダーの障害は LoaderNotAvailableException としてスローまたは再スローされる必要があります。 ObjectGrid は、これら 2 つの例外を異なる方法で処理します。

- LoaderException が後書きローダーによってキャッチされると、重複キー障害などのある種のデータ障害のため、後書きローダーはそれを障害とみなします。後書きローダーは、更新のバッチ処理を解除し、データ障害を分離するため、1 度に

1 レコードずつ更新しようとします。1 レコードの更新時に再度 {{LoaderException}} がキャッチされると、失敗した更新レコードが作成され、失敗した更新マップのログに記録されます。

- LoaderNotAvailableException が後書きローダーによってキャッチされると、データベース・エンドに接続できない (例えば、データベース・バックエンドがダウンしている、データベース接続が使用可能でない、ネットワークがダウンしているなど) ため、後書きローダーはそれを障害とみなします。後書きローダーは 15 秒待ってから、データベースへのバッチ更新を再試行します。

一般的な間違いは、LoaderNotAvailableException がスローされるべきなのに、LoaderException がスローされることです。後書きローダーでキューに入れられたすべてのレコードは、失敗更新レコードとなります。このような場合、バックエンド障害分離の目的が果たせなくなります。

パフォーマンスの考慮事項

後書きキャッシング・サポートの場合、ローダー更新をトランザクションから除去することで、応答時間が増加します。また、データベース更新が結合されるため、データベース・スループットも増加します。データをキュー・マップからプルし、ローダーにプッシュされる後書きスレッドの導入によって生じるオーバーヘッドを理解しておく必要があります。

予想される使用パターンおよび環境に基づいて、最大更新数または最大更新時間を調整する必要があります。最大更新カウントまたは最大更新時間の値が小さすぎると、後書きスレッドのオーバーヘッドが、その利点を帳消しにするおそれがあります。これら 2 つのパラメーターに大きな値を設定する場合も、データのキューイングに必要なメモリー使用が増え、データベース・レコードが不整合になる時間が増加するおそれがあります。

最善のパフォーマンスを得るために、後書き関係のパラメーターは、以下の要因を考慮に入れて調整してください。

- 読み取りトランザクションと書き込みトランザクションの比率
- 同一レコード更新の頻度
- データベース更新の待ち時間

ローダー

Loader プラグインを使用すると、通常は、同一システムあるいは別システムの永続ストアに保持されるデータのメモリー・キャッシュとしてデータ・グリッド・マップを動作させることができます。通常、データベースまたはファイル・システムは永続ストアとして使用されます。リモート Java 仮想マシン (JVM) もデータのソースとして使用でき、eXtreme Scale を使用してハブ・ベースのキャッシュを構築できます。ローダーには、永続ストアとの間でデータの読み取りおよび書き込みを行うロジックが備わっています。

概説

ローダーは、変更がバックアップ・マップに対して行われた場合、または、バックアップ・マップがデータ要求を満足できない (キャッシュ・ミス) 場合に呼び出されるバックアップ・マップ・プラグインです。ローダーは、キーに関する要求をキャ

ッシュが満足できなくなったときに起動され、リードスルー機能や、キャッシュにデータをゆっくり設定する機能を提供します。また、ローダーによって、キャッシュ値が変わったときのデータベース更新が可能になります。1 つのトランザクション内のすべての変更は、データベースとの対話の数を最小化できるよう、まとめてグループ化されます。ローダーと共に TransactionCallback プラグインが、バックエンド・トランザクションの境界をトリガーするために使用されます。このプラグインの使用は、複数のマップが 1 つのトランザクションに含まれている場合、または、トランザクション・データがコミットなしでキャッシュに書き込まれる場合に重要です。

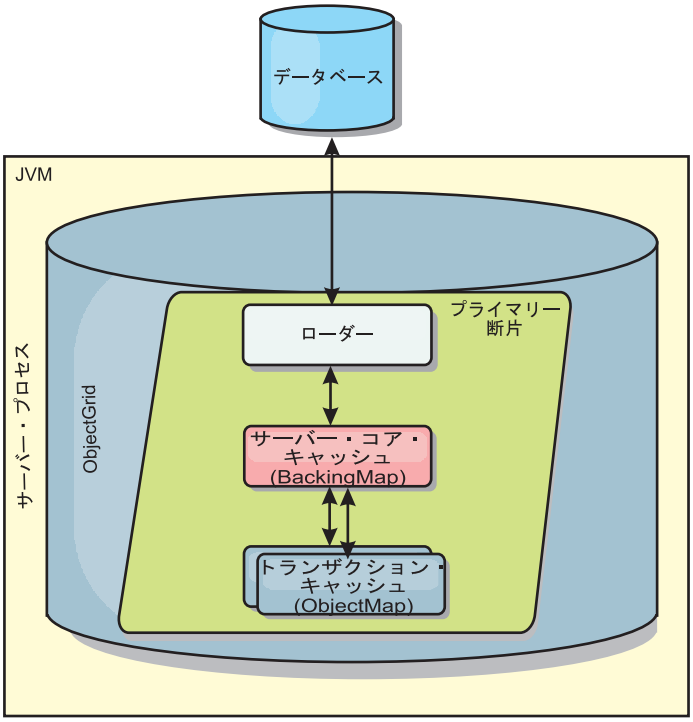


図 52. ローダー

ローダーは、データベース・ロックの保持を回避するために、資格過剰の更新を使用することもできます。バージョン属性をキャッシュ値の中に入れることによって、値がキャッシュ内で更新されるときにローダーは値の前と後のイメージを見ることができます。その後、データベースまたはバックエンドを更新する際にこの値を使用して、データが更新されていないことを検証できます。 ローダーは、開始時にデータ・グリッドをプリロードするよう構成することもできます。区画に分割されている場合、各区画ごとに 1 つのローダー・インスタンスが関連付けられます。例えば、「Company」マップに 10 個の区画がある場合、プライマリー区画ごとに 1 つずつ、10 個のローダー・インスタンスがあります。このマップのプライマリー断片がアクティブにされると、ローダーに対して preloadMap メソッドが同期または非同期で呼び出され、マップ区画にバックエンドからのデータが自動的にロードされます。非同期で呼び出される場合、すべてのクライアント・トランザクションはブロックされ、データ・グリッドへの矛盾するアクセスを防止します。代わりに、クライアント・プリローダーを使用してデータ・グリッド全体にデータをロードできます。

2 つの組み込みローダーにより、リレーショナル・データベース・バックエンドとの統合が非常に単純化されます。JPA ローダーは、Java Persistence API (JPA) 仕様の OpenJPA および Hibernate 実装の両方のオブジェクト関係マッピング (ORM) 機能を使用します。詳しくは、70 ページの『JPA ローダー』を参照してください。

複数データ・センター構成でローダーを使用する場合は、どのようにして改訂データとキャッシュの整合性をデータ・グリッド間で維持するかを検討する必要があります。詳しくは、182 ページの『マルチマスター・トポロジーでのローダーについての考慮事項』を参照してください。

ローダーの構成

ローダーを BackingMap 構成に追加するには、プログラマチック構成または XML 構成を使用します。ローダーには、バックアップ・マップとの間で以下のような関係があります。

- 1 つのバックアップ・マップは 1 つのローダーしか持てない。
- クライアント・バックアップ・マップ (ニア・キャッシュ) はローダーを持てない。
- 1 つのローダー定義を複数のバックアップ・マップに適用できるが、各バックアップ・マップは独自のローダー・インスタンスを持つ。

データのプリロードおよびウォームアップ

ローダーのユーザーを組み込む多くのシナリオで、データ・グリッドをデータと一緒にプリロードして準備しておくことができます。

データ・グリッドは、完全キャッシュとして使用される場合、データのすべてを保持しなければならず、いずれかのクライアントが接続する前にデータがロードされている必要があります。スパース・キャッシュを使用する場合は、クライアントが接続時にデータにすぐにアクセスできるよう、キャッシュをデータでウォームアップしておくことができます。

以下のセクションで説明するように、データをデータ・グリッドにプリロードする方法は 2 つあります。1 つは Loader プラグインを使用する方法で、もう 1 つはクライアント・ローダーを使用する方法です。

Loader プラグイン

Loader プラグインは、各マップに関連付けられ、1 つのプライマリー区画断片をデータベースと同期化させる役割を担います。断片がアクティブになると、Loader プラグインの preloadMap メソッドが自動的に呼び出されます。 例えば、100 の区画がある場合、ローダーのインスタンスは 100 存在し、それぞれが、各自の区画のためにデータをロードします。同期的に実行された場合、プリロードが完了するまですべてのクライアントがブロックされます。

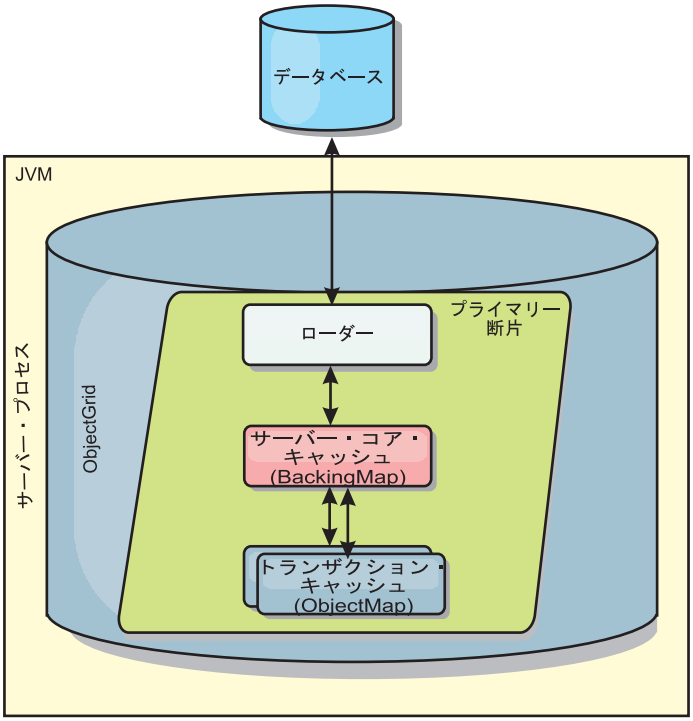


図 53. Loader プラグイン

クライアント・ローダー

クライアント・ローダーは、1 つ以上のクライアントを使用してグリッドにデータをロードするパターンです。複数のクライアントを使用してグリッドにデータをロードすることは、区画スキーマがデータベースに保管されない場合は効率的です。クライアント・ローダーは手動で呼び出すか、データ・グリッドの開始時に自動的に呼び出すことができます。データ・グリッドにデータをプリロードしている間はクライアントがデータ・グリッドにアクセスできないように、クライアント・ローダーは、オプションで、StateManager を使用してデータ・グリッドの状態をプリロード・モードに設定できます。WebSphere eXtreme Scale には Java Persistence API (JPA) ベースのローダーが組み込まれていて、OpenJPA または Hibernate JPA プロバイダーのどちらかでデータ・グリッドに自動的にロードするために使用できます。キャッシュ・プロバイダーについて詳しくは、27 ページの『JPA レベル 2 (L2) キャッシュ・プラグイン』を参照してください。

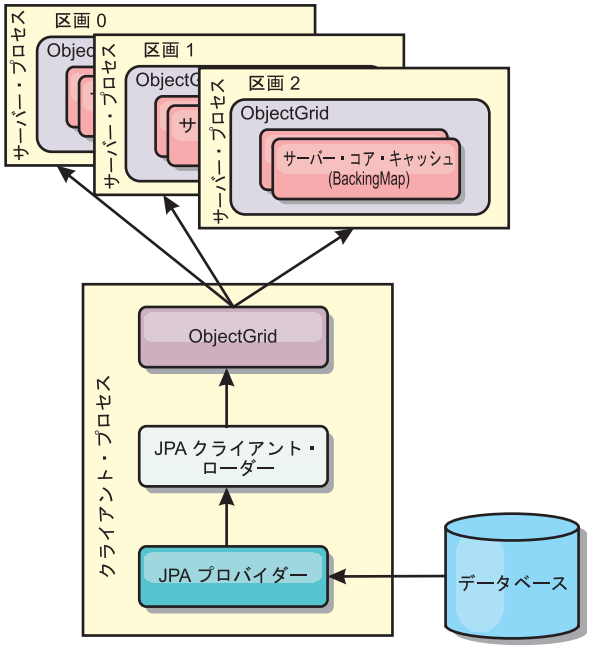


図 54. クライアント・ローダー

データベースの同期手法

WebSphere eXtreme Scale をキャッシュとして使用する際、データベースを eXtreme Scale トランザクションとは独立して更新できる場合、失効データを許容するようにアプリケーションを作成する必要があります。同期されたメモリー内データベース処理スペースとして機能するため、eXtreme Scale はキャッシュを常に最新の状態に保つ方法をいくつか備えています。

データベースの同期手法

定期的リフレッシュ

時間ベースの Java Persistence API (JPA) データベース・アップデーターを使用して、定期的なキャッシュの無効化または更新を自動的に実行できます。このアップデーターは、JPA プロバイダーを使用してデータベースを定期的に照会することによって、前回の更新以降に発生した更新または挿入があるかどうかを調べます。示された変更は、スパース・キャッシュで使用された場合、自動的に無効にされるか、更新されます。完全キャッシュで使用された場合、エントリーをディスカバーして、キャッシュに挿入することができます。エントリーがキャッシュから除去されることはありません。

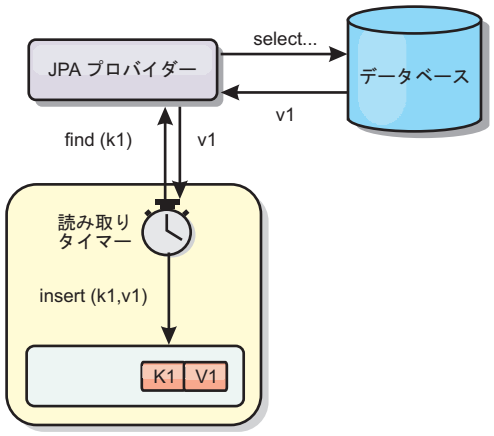


図 55. 定期的リフレッシュ

除去

スパース・キャッシュでは、除去ポリシーを使用して、データベースに影響を及ぼすことなく、キャッシュからデータを自動的に除去できます。eXtreme Scale には、Time-To-Live (存続時間)、Least-Recently-Used (最長未使用時間)、および Least-Frequently-Used (最も使用頻度の少ない) という 3 つの組み込みポリシーがあります。メモリー・ベースの除去オプションを使用可能にすると、メモリーが制約状態になるので、3 つのポリシーではすべて、必要であればデータをより積極的に除去することができます。

イベント・ベースの無効化

スパース・キャッシュおよび完全キャッシュは、Java Message Service (JMS) などのイベント生成プログラムを使用して無効化または更新することができます。JMS を使用した無効化は、データベース・トリガーを使用してバックエンドを更新するどのプロセスにも手動で関連付けることができます。サーバー・キャッシュで変更があった場合にクライアントに通知できる JMS ObjectGridEventListener プラグインが eXtreme Scale で提供されています。これにより、クライアントが失効データを表示する時間を短縮できます。

プログラマチックな無効化

eXtreme Scale API により、 Session.beginNoWriteThrough()、ObjectMap.invalidate()、および EntityManager.invalidate() API メソッドを使用したニア・キャッシュおよびサーバー・キャッシュの手動対話が可能になります。クライアントまたはサーバーのプロセスでデータの一部がもう必要ない場合、無効化メソッドを使用して、ニア・キャッシュまたはサーバー・キャッシュからデータを除去できます。beginNoWriteThrough メソッドは、ローダーを呼び出すことなく、ObjectMap または EntityManager 操作をローカル・キャッシュに適用します。クライアントから呼び出された場合のこの操作は、ニア・キャッシュのみに適用されます (リモート・ローダーは呼び出されません)。サーバーで呼び出された場合のこの操作は、ローダーを呼び出すことなく、サーバー・コア・キャッシュのみに適用されます。

データの無効化

Scale キャッシュ・データを削除するには、イベント・ベースの無効化メカニズムまたはプログラマチックな無効化メカニズムが使用できます。

イベント・ベースの無効化

スパース・キャッシュおよび完全キャッシュは、Java Message Service (JMS) などのイベント生成プログラムを使用して無効化または更新することができます。JMS を使用した無効化は、データベース・トリガーを使用してバックエンドを更新するどのプロセスにも手動で関連付けることができます。サーバー・キャッシュが変更した場合にクライアントに通知できる JMS ObjectGridEventListener プラグインが eXtreme Scale で提供されています。 この通知タイプによって、クライアントが失効データを表示する時間を短縮します。

イベント・ベースの無効化は、一般的には以下の 3 つのコンポーネントで構成されます。

- **イベント・キュー:** イベント・キューには、データ変更イベントが保管されます。 データ変更イベントを管理できるのであれば、イベント・キューは JMS キュー、データベース、メモリー内の FIFO キュー、またはすべての種類のマニフェストの可能性があります。
- **イベント・パブリッシャー:** イベント・パブリッシャーは、データ変更イベントをイベント・キューにパブリッシュします。 イベント・パブリッシャーは、通常、作成されたアプリケーションまたは eXtreme Scale プラグインの実装です。イベント・パブリッシャーは、いつデータが変更されたかを知っています。あるいはイベント・パブリッシャーがデータ自体を変更します。トランザクションがコミットすると、変更されたデータに対してイベントが生成され、イベント・パブリッシャーはこれらのイベントをイベント・キューにパブリッシュします。
- **イベント・コンシューマー:** イベント・コンシューマーは、データ変更イベントをコンシュームします。 イベント・コンシューマーは、通常アプリケーションで、ターゲット・グリッド・データが他のグリッドからの最新の変更を使用して更新されることを確認します。 このイベント・コンシューマーは、イベント・キューと対話をして最新のデータ変更を取得し、ターゲット・グリッドのデータ変更を適用します。 イベント・コンシューマーは eXtreme Scale API を使用して、失効データを無効にしたり、グリッドを最新データで更新することができます。

例えば、JMSObjectGridEventListener にはクライアント/サーバー・モデルのオプションがあり、そのイベント・キューは指定された JMS 宛先です。 すべてのサーバー・プロセスがイベント・パブリッシャーです。 トランザクションがコミットすると、サーバーはデータ変更を取得し、それを指定された JMS 宛先にパブリッシュします。 すべてのクライアント・プロセスがイベント・コンシューマーです。 指定された JMS 宛先からデータ変更を受信し、その変更をクライアントのニア・キャッシュに適用します。

詳しくは、「*管理ガイド*」でクライアント無効化メカニズムの使用可能化に関するトピックを参照してください。

プログラマチックな無効化

WebSphere eXtreme Scale API により、 Session.beginNoWriteThrough() 、ObjectMap.invalidate()、および EntityManager.invalidate() API メソッドを使用したニア・キャッシュおよびサーバー・キャッシュの手動対話が可能になります。クライアントまたはサーバーのプロセスでデータの一部がもう必要ない場合、無効化メソッドを使用して、ニア・キャッシュまたはサーバー・キャッシュからデータを除去できます。beginNoWriteThrough メソッドは、ローダーを呼び出すことなく、ObjectMap または EntityManager 操作をローカル・キャッシュに適用します。クライアントから呼び出された場合のこの操作は、ニア・キャッシュのみに適用されます (リモート・ローダーは呼び出されません)。サーバーで呼び出された場合のこの操作は、ローダーを呼び出すことなく、サーバー・コア・キャッシュのみに適用されます。

他の手法と一緒にプログラマチックな無効化を使用して、データをいつ無効にするかを決定します。 例えば、この無効化メソッドは、イベント・ベースの無効化メカニズムを使用してデータ変更イベントを受信し、API を使用して失効データを無効にします。

索引付け

MapIndexPlugin プラグインは、BackingMap 上にいくつかの索引を作成して、非キー・データ・アクセスをサポートするために使用します。

索引のタイプおよび構成

索引付けフィーチャーは、MapIndexPlugin プラグインと表されるか、または略して Index で表されます。Index は BackingMap プラグインです。BackingMap では、各索引プラグインが索引構成規則に従っている限り、複数の索引プラグインを構成できます。

索引付けフィーチャーは、1 つ以上の索引を BackingMap に作成する場合に使用できます。1 つの索引は、BackingMap 内の 1 つのオブジェクトの 1 つの属性または属性のリストから作成されます。このフィーチャーにより、アプリケーションはより迅速に特定のオブジェクトを見つけることができます。索引付けフィーチャーを使用すると、アプリケーションは特定の値を持つオブジェクトや、ある範囲の索引属性値内にあるオブジェクトを見つけることができます。

可能な索引付けには、静的および動的という 2 つのタイプがあります。静的索引付けの場合、ObjectGrid インスタンスを初期化する前に、BackingMap に索引プラグインを構成する必要があります。この構成を行うには、BackingMap を XML で構成するか、またはプログラマチックに構成します。静的索引付けでは、まず最初に、ObjectGrid の初期化中に索引を作成します。索引は常に BackingMap に同期しており、いつでも使用できる準備ができています。静的索引付けプロセスが既に開始している場合、索引は、eXtreme Scale トランザクション管理プロセスの一環として保守されます。トランザクションが変更をコミットすると、それらの変更は静的索引も更新し、トランザクションがロールバックされれば索引の変更もロールバックされます。

動的索引付けの場合は、索引を含む ObjectGrid インスタンスの初期化の前または後に、BackingMap に索引を作成することができます。動的索引付けプロセスのライフ

サイクルはアプリケーションによって制御されるので、不要になったら動的索引を削除することができます。アプリケーションが動的索引を作成する場合は、索引作成プロセスを完了するまでに時間がかかるために、その索引をすぐに使用できないことがあります。この時間は索引付けされるデータの量に依存するので、特定の索引付けイベントが発生したときにそのことを通知してもらいたいアプリケーションのために、DynamicIndexCallback インターフェースが提供されています。これらのイベントには、準備完了、エラー、および破棄があります。アプリケーションは、このコールバック・インターフェースを実装し、動的索引付けプロセスに登録できます。

BackingMap に索引プラグインが構成されている場合、対応する ObjectMap からアプリケーション索引プロキシー・オブジェクトを取得することができます。ObjectMap の getIndex メソッドを呼び出し、索引プラグインの名前を渡すと、索引プロキシー・オブジェクトが戻されます。索引プロキシー・オブジェクトを適切なアプリケーション索引インターフェース (MapIndex、MapRangeIndex、またはカスタマイズされた索引インターフェースなど) にキャストする必要があります。索引プロキシー・オブジェクトを取得したら、アプリケーション索引インターフェースで定義されたメソッドを使用して、キャッシュされたオブジェクトを検出することができます。

次のリストに、索引付けの使用手順をまとめます。

- 静的または動的索引プラグインを BackingMap に追加します。
- ObjectMap の getIndex メソッドを発行して、アプリケーション索引プロキシー・オブジェクトを取得します。
- MapIndex、MapRangeIndex またはカスタマイズされた索引インターフェースなどの適切なアプリケーション索引インターフェースに、索引プロキシー・オブジェクトをキャストします。
- アプリケーション索引インターフェースで定義されたメソッドを使用して、キャッシュされたオブジェクトを検出します。

HashIndex クラスは、組み込みアプリケーション索引インターフェースである MapIndex と MapRangeIndex の両方をサポートすることのできる組み込み索引プラグイン実装です。ユーザー独自の索引を作成することもできます。HashIndex を静的索引または動的索引として BackingMap に追加して、MapIndex または MapRangeIndex の索引プロキシー・オブジェクトを取得し、その索引プロキシー・オブジェクトを使用してキャッシュ・オブジェクトを検索することができます。

デフォルトの索引

ローカル・マップ内のキーを反復処理する場合は、デフォルトの索引を使用できます。この索引はまったく構成を必要としませんが、エージェントを使用するか ShardEvents.shardActivated(ObjectGrid shard) メソッドから取得した ObjectGrid インスタンスを使用して、断片に対して使用しなければなりません。

データ品質に関する考慮事項

索引照会メソッドの結果が表わすのは、特定の時刻におけるデータのスナップショットのみです。結果がアプリケーションに戻された後には、データ・エントリーに対するロックは取得されません。アプリケーションは、戻されたデータ・セットに

対してデータ更新が発生する可能性があることに注意する必要があります。例えば、アプリケーションは MapIndex の findAll メソッドを実行して、キャッシュされたオブジェクトのキーを取得します。戻されたこのキー・オブジェクトは、キャッシュ内のデータ項目に関連付けられています。アプリケーションは、キー・オブジェクトを提供することにより、ObjectMap に対して get メソッドを実行して、オブジェクトを検出できるようになっている必要があります。get メソッドが呼び出される直前に、別のトランザクションがキャッシュからそのデータ・オブジェクトを削除した場合、戻される結果はヌルです。

索引付けのパフォーマンスに関する考慮事項

索引付けフィーチャーの主な目的の 1 つは、BackingMap の全体的なパフォーマンスを改善することです。索引付けの使い方が不適切な場合は、アプリケーションのパフォーマンスが低下する可能性があります。このフィーチャーを使用する前に、次の要因について検討します。

- **並行書き込みトランザクションの数:** 索引処理は、トランザクションが BackingMap にデータを書き込むたびに起こりえます。アプリケーションが索引照会操作を試行しているときに、多くのトランザクションがデータをマップに書き込んでいると、パフォーマンスが低下します。
- **照会操作で戻される結果セットのサイズ:** 結果セットのサイズが大きくなるにつれて、照会のパフォーマンスは低下します。結果セットのサイズが BackingMap の 15% 以上になるとパフォーマンスは低下する傾向にあります。
- **同じ BackingMap に作成される索引の数:** 各索引がシステム・リソースを消費します。BackingMap に作成される索引の数が増えると、パフォーマンスは低下します。

索引付け機能は、BackingMap パフォーマンスを大幅に改善できることがあります。理想的なケースは、BackingMap の大部分の操作が読み取りであり、照会の結果セットが BackingMap エントリーのわずかな割合に過ぎず、ごく少数の索引が BackingMap に対して作成される場合です。

複数データ・センター・トポロジーの計画

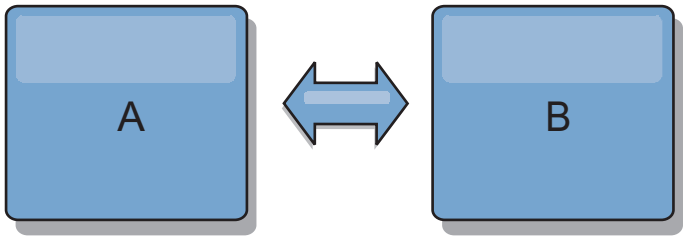
マルチマスター非同期レプリカ生成機能を使用すると、2 つ以上のデータ・グリッドを、互いの正確なミラーにすることができます。 各データ・グリッドは独立したカタログ・サービス・ドメイン内でホストされ、独自のカタログ・サービス、コンテナー・サーバー、および固有の名前を所有しています。 マルチマスター非同期レプリカ生成機能により、リンクを使用してカタログ・サービス・ドメインのコレクションを接続できます。すると、カタログ・サービス・ドメインは、リンクを介したレプリカ生成を使用して同期されます。カタログ・サービス・ドメイン間のリンクの定義を使用して、ほとんどどのトポロジーでも構成できます。

マルチマスター・レプリカ生成のためのトポロジー

マルチマスター・レプリカ生成を導入するデプロイメントのトポロジーを選択する際、いくつかの異なるオプションがあります。

カタログ・サービス・ドメインを接続するリンク

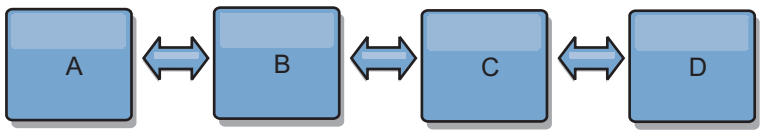
レプリカ生成データ・グリッドのインフラストラクチャーは、カタログ・サービス・ドメイン間を双方向のリンクで接続したカタログ・サービス・ドメインのグラフです。 リンクを使用して、2 つのカタログ・サービス・ドメインはデータ変更内容をやりとりできます。例えば、最も単純なトポロジーは、カタログ・サービス・ドメイン間に単一のリンクを持つ 1 対のカタログ・サービス・ドメインです。 カタログ・サービス・ドメインは、左から A、B、C というようにアルファベット順で指定されています。リンクは、遠距離にわたる広域ネットワーク (WAN) を経由する場合もあります。 リンクが遮断されたとしても、いずれかのカタログ・サービス・ドメインでまだデータを変更できます。トポロジーは、リンクがカタログ・サービス・ドメインと再接続したときに変更を調整します。ネットワーク接続が中断されると、リンクは自動的に再接続しようとします。



リンクをセットアップすると、eXtreme Scale はまず、すべてのカタログ・サービス・ドメインを同一にしようと試みます。次に、いずれかのカタログ・サービス・ドメインで変更が発生すると、eXtreme Scale は同一の状態を維持するよう試みます。目標は、各カタログ・サービス・ドメインがリンクで接続されたすべての他のカタログ・サービス・ドメインの正確なミラーになることです。 カタログ・サービス・ドメイン間のレプリカ生成リンクは、1 つのドメインで行われたすべての変更を確実に他のドメインにコピーするのに役立ちます。

ライン・トポロジー

ライン・トポロジーはこのような単純なデプロイメントですが、かなりのリンク品質を実証します。 まず、変更を受け取るために、カタログ・サービス・ドメインは直接すべての他のカタログ・サービス・ドメインに接続する必要がありません。 ドメイン B はドメイン A から変更をプルします。ドメイン C は、ドメイン A と C を接続するドメイン B を介してドメイン A から変更を受信します。同様に、ドメイン D はドメイン C を介して他のドメインから変更を受信します。この機能によって、変更のソースから変更を配布する負荷が分散されます。



ドメイン C に障害が起こった場合、以下のアクションの発生が考えられることに注意してください。

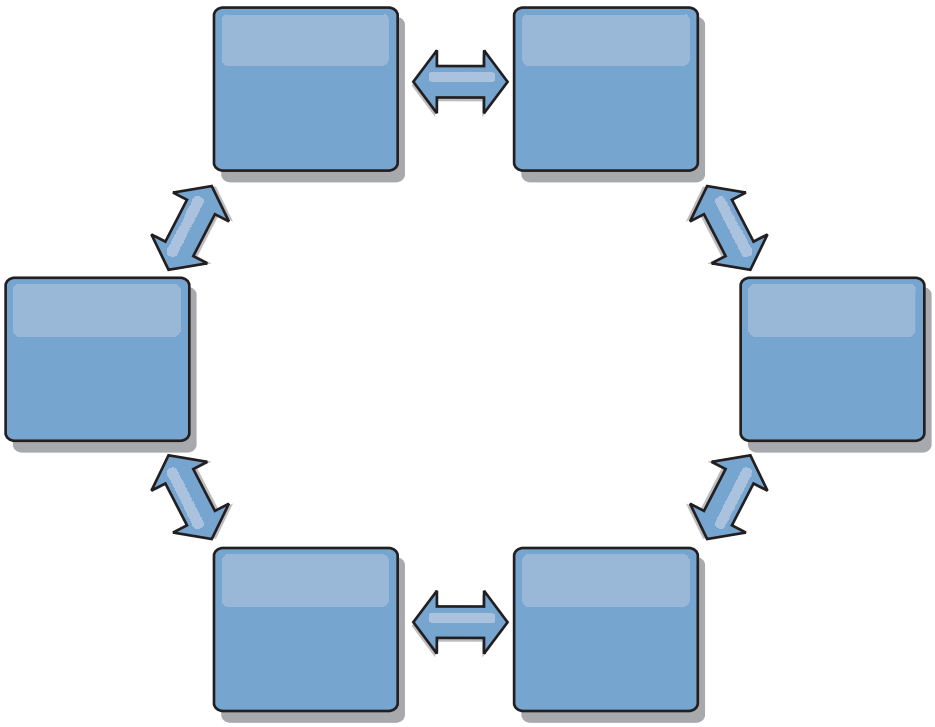
1. ドメイン D は、ドメイン C が再開されるまで孤立します。
2. ドメイン C は、ドメイン A のコピーであるドメイン B と自分自身を同期させます。

3. ドメイン D は、ドメイン C を使用して、ドメイン A と B で発生した変更と自分自身を同期させます。これらの変更は最初は、ドメイン D が孤立していた間 (ドメイン C がダウンしていた間) に発生しました。

最終的に、ドメイン A、B、C、および D はすべて、互いのドメインと再び同一になります。

リング・トポロジー

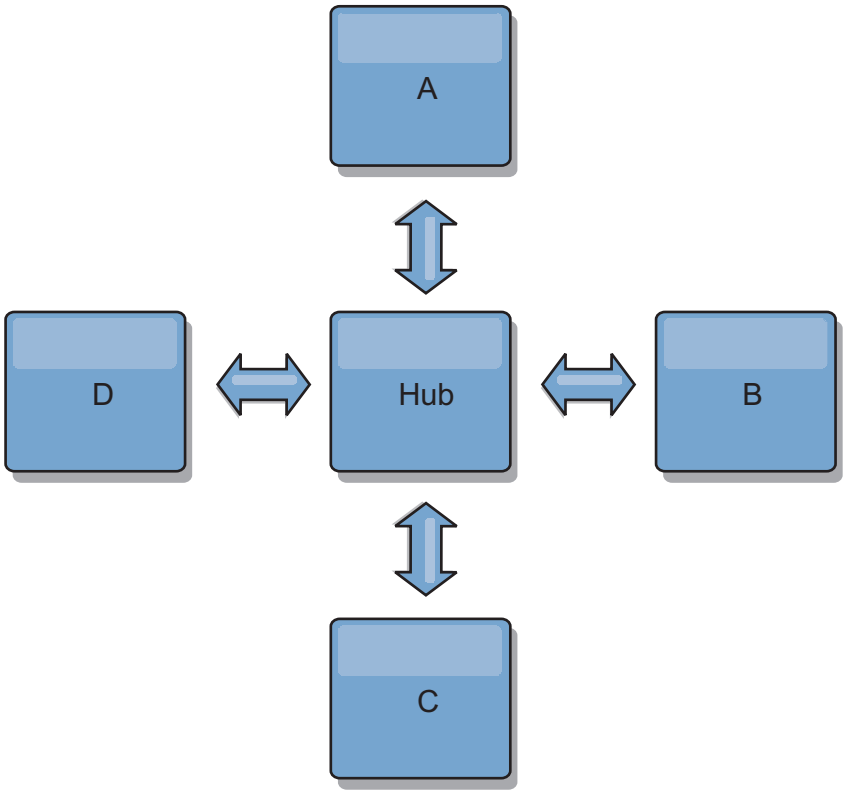
リング・トポロジーは、より回復力のあるトポロジーの例です。 カタログ・サービス・ドメインまたは単一リンクに障害が起こった場合でも、残ったカタログ・サービス・ドメインがまだ変更を取得できます。そのカタログ・サービス・ドメインは、障害から離れて、リングの周りを回ります。リング・トポロジーの大きさには関係なく、各カタログ・サービス・ドメインは他のカタログ・サービス・ドメインとのリンクを最大 2 つ持ちます。変更を伝搬するための待ち時間は長くなる場合があります。特定のカタログ・サービス・ドメインでの変更は、すべてのカタログ・サービス・ドメインにその変更が反映されるまで、複数のリンクを経由して伝搬する必要がある場合があります。ライン・トポロジーにも同じ特性があります。



リングの中心に置いたルート・カタログ・サービス・ドメインを使用した、より洗練されたリング・トポロジーをデプロイすることも可能です。ルート・カタログ・サービス・ドメインは、調整の中心点として機能します。他のカタログ・サービス・ドメインは、ルート・カタログ・サービス・ドメインで生じた変更に対する調整のリモート・ポイントとして働きます。ルート・カタログ・サービス・ドメインはカタログ・サービス・ドメイン間の変更をアービトレーションすることができます。ルート・カタログ・サービス・ドメインを囲む複数のリングがリング・トポロジーに含まれている場合、ドメインは最も内側にあるリング間の変更のみをアービトレーションすることができます。ただし、アービトレーションの結果は他のリングのカタログ・サービス・ドメインにも広がります。

ハブ・アンド・スポーク・トポロジー

ハブ・アンド・スポーク・トポロジーでは、ハブ・カタログ・サービス・ドメインを経由して変更が伝搬します。ハブは指定される唯一の中間カタログ・サービス・ドメインであるため、ハブ・アンド・スポーク・トポロジーでは待ち時間が短縮されます。ハブ・ドメインは、リンク経由ですべてのスポーク・ドメインに接続されています。ハブは、カタログ・サービス・ドメイン間で変更を配布します。ハブは、衝突に対して調整のポイントとして機能します。更新頻度の高い環境では、同期を保つために、スポークよりも多くのハードウェア上でハブを稼働する必要がある場合があります。 WebSphere eXtreme Scale は、直線的に拡大するように設計されています。つまり、問題なく、必要に応じてハブをさらに大きくすることができます。 ただし、ハブに障害が起こった場合は、変更はハブが再始動するまで配布されません。 スポーク・カタログ・サービス・ドメイン上の変更は、ハブが再接続された後に配布されます。



また、完全に複製したクライアントを使用したストラテジー、すなわち、ハブとして稼働している eXtreme Scale サーバーのペアを使用するトポロジーのバリエーションを使用することもできます。各クライアントは、クライアント JVM 内に、必要なものを完備した単一コンテナー・データ・グリッドとカタログを作成します。クライアントは、そのデータ・グリッドを使用してハブ・カタログに接続します。この接続により、クライアントはハブへの接続を取得すると、すぐにハブと同期するようになります。

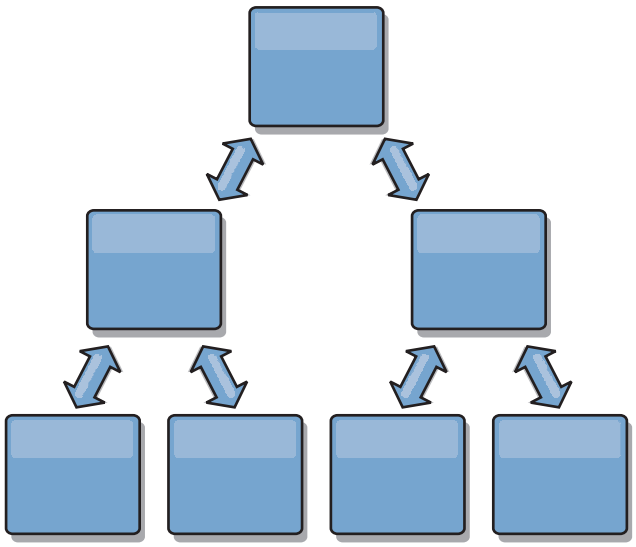
クライアントによって行われた変更は、クライアントに対してローカルで、非同期でハブに複製されます。 ハブはアービトレーション・ドメインとして機能し、すべての接続されたクライアントに変更を配布します。完全複製クライアントのトポロジーは、OpenJPA などのオブジェクト・リレーショナル・マッパーに信頼性の高い

L2 キャッシュを提供します。変更はハブを介してクライアント JVM 間に迅速に配布されます。 キャッシュ・サイズを使用可能なヒープ・スペース内に含むことができる場合、このトポロジーは L2 のこのスタイルにとって信頼できるアーキテクチャーです。

必要であれば、複数の区画を使用して、複数の JVM 上にハブ・ドメインを拡張します。 すべてのデータはまだ単一のクライアント JVM に収まらなければならないため、複数の区画を使用してハブの容量を増加させ、変更の配布とアービトレーションを行います。ただし、複数の区画を使用しても、単一ドメインの容量は変更されません。

ツリー・トポロジー

非循環有向ツリーを使用することもできます。非循環ツリーには循環やループはなく、有向セットアップにより、リンクの存在は親と子の間のみに制限されます。この構成は、多くのカタログ・サービス・ドメインを含むトポロジーで役立ち、すべての可能なスポークに接続される中央ハブを使用することは実用的ではありません。また、このタイプのトポロジーは、ルート・カタログ・サービス・ドメインを更新することなく子カタログ・サービス・ドメインを追加する必要がある場合にも便利です。



ツリー・トポロジーでもまだ、ルート・カタログ・サービス・ドメインに調整の中心点を置くことができます。第 2 レベルはまだ、それらの下のカタログ・サービス・ドメインで生じた変更に対する調整のリモート・ポイントとして機能します。ルート・カタログ・サービス・ドメインは、第 2 レベルにあるカタログ・サービス・ドメイン間の変更のみをアービトレーションすることができます。それぞれが各レベルで N 個の子を持つ、n 進ツリーを使用することもできます。それぞれのカタログ・サービス・ドメインは、*n* 個のリンクに接続します。

完全複製クライアント

このトポロジー変化には、ハブとして稼働する 1 対の eXtreme Scale サーバーが含まれます。 各クライアントは、クライアント JVM 内に、必要なものを完備した単一コンテナー・データ・グリッドとカタログを作成します。 クライアントは、その

データ・グリッドを使用してハブ・カタログに接続します。これにより、クライアントはハブへの接続を取得すると、すぐにハブと同期するようになります。

クライアントによって行われた変更は、クライアントに対してローカルで、非同期でハブに複製されます。 ハブはアービトレーション・ドメインとして機能し、すべての接続されたクライアントに変更を配布します。完全複製クライアントのトポロジーは、OpenJPA などのオブジェクト・リレーショナル・マッパーに適した L2 キャッシュを提供します。変更はハブを介してクライアント JVM 間に迅速に配布されます。 キャッシュ・サイズをクライアントの使用可能なヒープ・スペース内に含むことができる限り、このトポロジーは L2 のこのスタイルに適したアーキテクチャーです。

必要であれば、複数の区画を使用して、複数の JVM 上にハブ・ドメインを拡張します。 すべてのデータはまだ単一のクライアント JVM に収まらなければならないため、複数の区画を使用してハブの容量を増加させ、変更の配布とアービトレーションを行いますが、単一ドメインの容量は変更しません。

マルチマスター・トポロジーに関する構成の考慮事項

マルチマスター・レプリカ生成トポロジーを使用するかどうかを決定し、その使用方法について決定する際は、以下の問題を考慮してください。

- **マップ・セット要件**

  カタログ・サービス・ドメインのリンクを介して変更を複製するには、マップ・セットは以下の特性を持っている必要があります。

  – カタログ・サービス・ドメイン内の ObjectGrid 名およびマップ・セット名は、他のカタログ・サービス・ドメインの ObjectGrid 名およびマップ・セット名と一致していなければならない。例えば、ObjectGrid「og1」およびマップ・セット「ms1」がカタログ・サービス・ドメイン A とカタログ・サービス・ドメイン B で構成されていないと、それらのカタログ・サービス・ドメイン間でマップ・セット内のデータを複製できません。

  – FIXED_PARTITION データ・グリッドである。PER_CONTAINER データ・グリッドを複製できません。

  –

  – 各カタログ・サービス・ドメイン内の同じデータ・タイプが複製される

  – 各カタログ・サービス・ドメイン内に同じマップおよび動的マップ・テンプレートが含まれている。

  – エンティティー・マネージャーを使用しない。エンティティー・マップを含むマップ・セットは、カタログ・サービス・ドメインを介して複製されません。

  – 後書きキャッシング・サポートを使用しない。後書きサポートで構成されたマップを含むマップ・セットは、カタログ・サービス・ドメインを介して複製されません。

  トポロジー内のカタログ・サービス・ドメインが開始されると、前述の特性を持つすべてのマップ・セットが複製を開始します。

- **複数のカタログ・サービス・ドメインを使用するクラス・ローダー**

  カタログ・サービス・ドメインは、キーおよび値として使用されるクラスすべてへのアクセス権限を持たなければなりません。すべての依存関係は、すべてのド

メインのデータ・グリッド・コンテナー Java 仮想マシン (JVM) に対するすべてのクラスパスに反映されなければなりません。 CollisionArbiter プラグインがキャッシュ・エントリーの値を取得する場合、その値に対するクラスはアービターを開始するドメインに存在しなければなりません。

マルチマスター・トポロジーでのローダーについての考慮事項

マルチマスター・トポロジーでローダーを使用する場合は、起こり得る衝突および改訂情報の維持についての問題を考慮する必要があります。データ・グリッドはその中の各項目について改訂情報を維持しており、構成内の他のプライマリー断片がデータ・グリッドにエントリーを書き込むときに衝突を検出できるようになっています。エントリーがローダーから追加されると、この改訂情報は含められず、エントリーは新しい改訂を持つようになります。エントリーの改訂は新規挿入に見えるため、別のプライマリー断片もこの状態を変更したり、ローダーから同じ情報を引き込んだりした場合に、偽の衝突が発生する場合があります。

レプリカ生成の変更は、データ・グリッド内に今はないが、レプリカ生成トランザクション中に変更されるキーのリストを使用して、ローダーに対して get メソッドを呼び出します。レプリカ生成が行われると、これらのエントリーは衝突エントリーとなります。衝突をアービトレーションし、改訂を適用すると、バッチ更新がローダーで呼び出されて変更内容がデータベースに適用されます。改訂ウィンドウで変更されたマップはすべて、同じトランザクションで更新されます。

プリロードの問題

データ・センター A とデータ・センター B を使用した 2 つのデータ・センター・トポロジーがあるとします。2 つのデータ・センターはそれぞれ独立したデータベースを持っていますが、データ・センター A にのみ、実行中のデータ・グリッドがあります。マルチマスター構成でデータ・センター間のリンクを確立すると、データ・センター A 内のデータ・グリッドがデータ・センター B 内の新規データ・グリッドにデータをプッシュし始め、すべてのエントリーとの衝突を引き起こします。別の大きな問題は、データ・センター A 内のデータベースには存在せず、データ・センター B 内のデータベースにあるすべてのデータで発生します。これらの行にはデータが取り込まれず、アービトレーションされません。結果として、解決されない不整合が発生します。

プリロードの問題に対する解決策

データベース内にのみ存在するデータは改訂を持つことができないため、常にローカル・データベースからデータ・グリッドを完全にプリロードした後、マルチマスター・リンクを設定する必要があります。次に、両方のデータ・グリッドはデータを改訂し、アービトレーションすることができ、最終的に整合した状態に達します。

スパース・キャッシュの問題

スパース・キャッシュを使用すると、アプリケーションはまずデータ・グリッド内のデータの検索を試みます。データがデータ・グリッド内にないと、ローダーを使用してデータベースでデータが検索されます。キャッシュ・サイズを小規模に維持するために、エントリーは定期的にデータ・グリッドから除去されます。

このキャッシュ・タイプは、マルチマスター構成シナリオでは問題となる場合があります。なぜなら、データ・グリッド内のエントリーは、衝突が発生するときやどちら側が変更を行ったかを検出するのを助ける、改訂用メタデータを持っているためです。データ・センター間のリンクが機能していない場合、一方のデータ・センターがエントリーを更新し、最終的にデータ・グリッド内のデータベースを更新し、エントリーを無効化することができます。リンクが復旧すると、データ・センターは互いに改訂を同期しようとします。しかし、データベースが更新され、データ・グリッド・エントリーが無効化されているため、ダウンしていたデータ・センターの観点から見ると、変更が失われています。結果として、両側のデータ・グリッドで同期がとれず、整合性がなくなります。

スパース・キャッシュの問題に対する解決策

ハブおよびスポーク・トポロジー:

ハブおよびスポーク・トポロジーのハブでのみローダーを実行し、結果として、データの整合性を維持しながら、データ・グリッドをスケールアウトすることができます。ただし、このデプロイメントを検討している場合は、ローダーがデータ・グリッドを部分的にロードできることに注意してください。これは、Evictor が構成済みであることを意味します。構成のスポークがスパース・キャッシュだが、ローダーがない場合は、どのキャッシュ・ミスもデータベースからデータを取り出すことができません。この制約事項のため、ハブおよびスポーク構成では、完全に取り込まれたキャッシュ・トポロジーを使用する必要があります。

無効化および除去

無効化により、データ・グリッドとデータベース間の不整合が発生します。プログラマチックに、または除去機能を使用して、データ・グリッドからデータを削除できます。アプリケーションの開発時に、改訂処理では無効化された変更内容は複製されず、プライマリー断片間で不整合が発生しないよう注意する必要があります。

無効化イベントは、キャッシュ状態変更ではなく、レプリカ生成は生じません。いかなる構成済み Evictor も構成内の他の Evictor と独立して実行されます。例えば、カタログ・サービス・ドメインでのメモリーしきい値について構成済みの Evictor が 1 つあるが、リンクされている他のカタログ・サービス・ドメインに異なるタイプのあまり活動的でない Evictor がある場合があります。データ・グリッド・エントリーがメモリーしきい値ポリシーのために削除されても、他のカタログ・サービス・ドメイン内のエントリーは影響を受けません。

データベースの更新およびデータ・グリッドの無効化

問題が発生するのは、バックグラウンドで直接データベースを更新しながら、マルチマスター構成で更新済みエントリーについてデータ・グリッドに対して無効化を呼び出しているときです。この問題は、いくつかのタイプのキャッシュ・アクセスがエントリーをデータ・グリッドに移動するまで、データ・グリッドが別のプライマリー断片への変更を複製できないために発生します。

単一論理データベースへの複数の書き込みプログラム

ローダーを介して接続された複数のプライマリー断片と一緒に単一データベースを使用していると、トランザクションの競合が発生します。ローダーの実装は、特に

これらのタイプのシナリオを処理する必要があります。

マルチマスター・レプリカ生成を使用したデータのミラーリング

独立したカタログ・サービス・ドメインに接続された独立したデータベースを構成できます。この構成では、ローダーはあるデータ・センターの変更内容を別のデータ・センターにプッシュできます。

マルチマスター・レプリカ生成での設計上の考慮事項

マルチマスター・レプリカ生成を実装する場合、アービトレーション、リンク作成、およびパフォーマンスなど、設計における側面を考慮する必要があります。

トポロジー設計におけるアービトレーションの考慮事項

同じレコードが 2 個所で同時に変更される可能性がある場合には、変更の競合が生じることがあります。各カタログ・サービス・ドメインが、同程度のプロセッサー、メモリー、ネットワーク・リソースを持つようにセットアップしてください。変更の衝突処理 (アービトレーション) を実行しているカタログ・サービス・ドメインは、他のカタログ・サービス・ドメインよりも多くのリソースを使用することに気付くことがあります。 衝突は、自動的に検出されます。 衝突は、以下の 2 つのメカニズムの 1 つを使用して処理されます。

- **デフォルト衝突アービター**: デフォルトのプロトコルは、字句的に最も小さい名前の付いたカタログ・サービス・ドメインからの変更を使用します。 例えば、カタログ・サービス・ドメイン A と B によってレコードの競合が生じる場合には、カタログ・サービス・ドメイン B の変更は無視されます。カタログ・サービス・ドメイン A はそのバージョンを保持し、カタログ・サービス・ドメイン B のレコードはカタログ・サービス・ドメイン A からのレコードに一致するように変更されます。この動作は、ユーザーやセッションが正常にバインドされているアプリケーション、またはユーザーやセッションがデータ・グリッドの 1 つにアフィニティーを持つ対象となるアプリケーションにも同様に適用されます。
- **カスタム衝突アービター**: アプリケーションはカスタム・アービターを提供することができます。カタログ・サービス・ドメインは衝突を検出すると、アービターを開始します。便利なカスタム・アービターの開発について詳しくは、 マルチマスター・レプリカ生成のためのカスタム・アービターの作成を参照してください。

衝突が起こる可能性のあるトポロジーに対しては、ハブ・アンド・スポーク・トポロジーまたはツリー・トポロジーの実装を検討してください。これらの 2 つのトポロジーは、以下のシナリオで発生する可能性のある、恒常的な衝突の回避につながります。

1. 複数のカタログ・サービス・ドメインで衝突が発生します。
2. 各カタログ・サービス・ドメインが衝突をローカルで処理し、改訂を生成します。
3. 改訂が衝突し、その結果、改訂の改訂をもたらします。

衝突を回避するには、カタログ・サービス・ドメインのサブセットの衝突アービターとして、*アービトレーション・カタログ・サービス・ドメイン* と呼ばれる特定のカタログ・サービス・ドメインを選択します。例えば、ハブ・アンド・スポーク・トポロジーはハブを衝突ハンドラーとして使用する場合があります。 スポーク衝突

ハンドラーは、スポーク・カタログ・サービス・ドメインで検出されたすべての衝突を無視します。ハブ・カタログ・サービス・ドメインは、改訂を作成し、予期しない衝突の改訂を回避します。衝突を処理するように割り当てられたカタログ・サービス・ドメインは、衝突の処理に責任を持つすべてのドメインにリンクしていなければなりません。 ツリー・トポロジーでは、内部の親ドメインが自分の直接の子の衝突を処理します。 対照的に、リング・トポロジーを使用する場合、リング内の 1 つのカタログ・サービス・ドメインをアービターとして指定することはできません。

次の表に、さまざまなトポロジーと互換性のあるアービトレーション・アプローチをまとめました。

表 8. アービトレーション・アプローチ： この表は、アプリケーション・アービトレーションがさまざまなトポロジーと互換性があるかどうかについて記述します。

トポロジー	アプリケーション・アービトレーション?	注
2 つのカタログ・サービス・ドメインのライン	あり	1 つのカタログ・サービス・ドメインをアービターとして選択します。
3 つのカタログ・サービス・ドメインのライン	あり	真ん中のカタログ・サービス・ドメインがアービターでなければなりません。 真ん中のカタログ・サービス・ドメインが、単純なハブ・アンド・スポーク・トポロジーのハブだと考えてください。
3 つより多いカタログ・サービス・ドメインのライン	なし	アプリケーション・アービトレーションはサポートされません。
N 個のスポークを持つハブ	あり	すべてのスポークへのリンクを持つハブがアービトレーション・カタログ・サービス・ドメインでなければなりません。
N 個のカタログ・サービス・ドメインのリング	なし	アプリケーション・アービトレーションはサポートされません。
非循環有向ツリー (n 進ツリー)	あり	すべてのルート・ノードは、自分の直接の子孫のみを評価する必要があります。

トポロジー設計におけるリンクの考慮事項

変更待ち時間、フォールト・トレランス、およびパフォーマンス特性におけるトレードオフを最適化している間、トポロジーにはリンクの最小数が含まれているのが理想的です。

- **変更待ち時間**

  変更待ち時間は、変更が特定のカタログ・サービス・ドメインに到着する前に経由しなければならない中間カタログ・サービス・ドメインの数によって決まります。

  トポロジーが、すべてのカタログ・サービス・ドメインを他のすべてのカタログ・サービス・ドメインにリンクすることによって中間カタログ・サービス・ドメインを除去すれば、トポロジーの変更待ち時間は最善になります。 ただし、カ

タログ・サービス・ドメインはそのリンク数に比例してレプリカ生成作業を実行しなければなりません。 大規模トポロジーの場合、非常に多くのリンクが定義され、管理が負担になる場合があります。

変更が他のカタログ・サービス・ドメインにコピーされる速度は、以下の追加要因によって異なります。

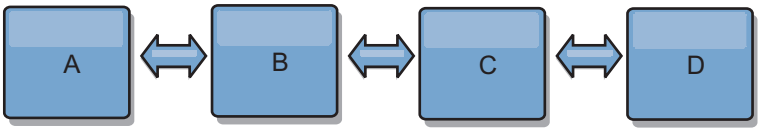
- ソース・カタログ・サービス・ドメイン上のプロセッサーとネットワーク帯域幅
- ソース・カタログ・サービス・ドメインとターゲット・カタログ・サービス・ドメインの間の中間カタログ・サービス・ドメイン数とリンク数
- ソース・カタログ・サービス・ドメイン、ターゲット・カタログ・サービス・ドメイン、および中間カタログ・サービス・ドメインで使用可能なプロセッサーとネットワーク・リソース

- **フォールト・トレランス**

フォールト・トレランスは、変更のレプリカ生成のために、2 つのカタログ・サービス・ドメイン間に存在するパス数によって決定します。

特定のカタログ・サービス・ドメインのペア間に 1 つしかリンクがないと、リンク障害が発生した場合に変更を伝搬できません。同様に、中間ドメインのいずれかでリンク障害が発生すると、カタログ・サービス・ドメイン間で変更が伝搬されません。あるカタログ・サービス・ドメインから別のカタログ・サービス・ドメインへの単一リンクが中間ドメインを経由するトポロジーを考えることができます。その場合、中間カタログ・サービス・ドメインのいずれかがダウンすると、変更が伝搬されません。

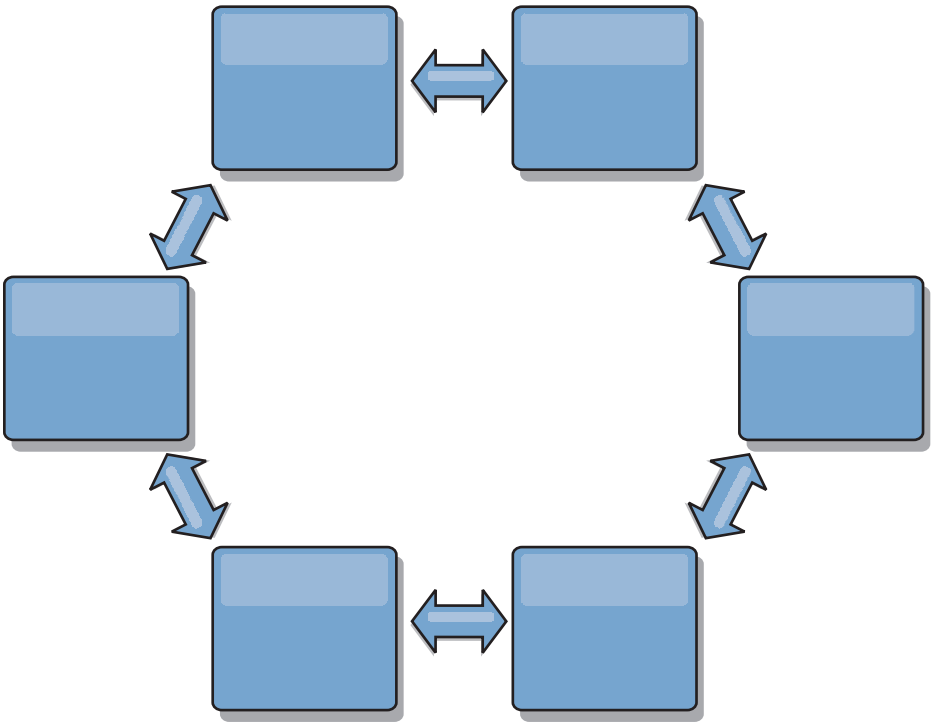
4 つのカタログ・サービス・ドメイン A、B、C、および D を持つライン・トポロジーを考えてみます。



以下のいくつかの状態のままであれば、ドメイン D は A からの変更はまったく見えません。

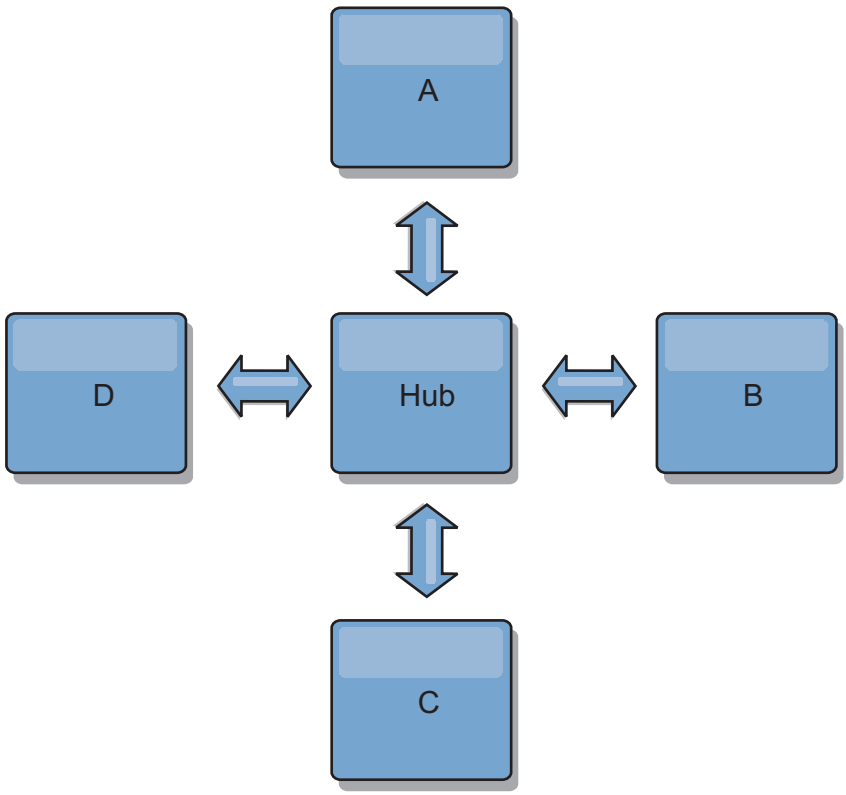
- ドメイン A が稼働中で B がダウン
- ドメイン A および B が稼働中で C がダウン
- A と B の間のリンクがダウン
- B と C の間のリンクがダウン
- C と D の間のリンクがダウン

対照的に、リング・トポロジーの場合、各カタログ・サービス・ドメインはどちらかの方向から変更を受け取ることができます。



例えば、リング・トポロジー内の特定のカタログ・サービスがダウンしている場合、2 つの隣接ドメインはまだ互いに変更を直接プルすることができます。

すべての変更はハブを経由して伝搬されます。したがって、ライン・トポロジーやリング・トポロジーとは対照的に、ハブ・アンド・スポーク設計は、ハブに障害が起きた場合に機能停止となる可能性が高いといえます。



単一カタログ・サービス・ドメインは、ある量のサービス損失に対しては回復力があります。ただし、広域ネットワーク障害や物理データ・センター間のリンク障害などのより大規模な障害が発生した場合は、いずれかのカタログ・サービス・ドメインが中断される可能性があります。

- **リンク作成およびパフォーマンス**

  カタログ・サービス・ドメイン上に定義されるリンク数は、パフォーマンスに影響します。リンクが多いと使われるリソースも多くなり、結果的にレプリカ生成パフォーマンスが落ちる場合もあります。 他のドメインを介してドメイン A の変更を取得する機能は、そのトランザクションを各場所に複製するドメイン A の負荷を効果的に軽減します。ドメイン上の変更配布の負荷は、トポロジー内のドメインの数ではなく、ドメインが使用するリンクの数によって制限されます。このロード・プロパティーは、スケーラビリティーを提供するため、トポロジー内のドメインは変更の配布に伴う負荷を分配できます。

  カタログ・サービス・ドメインは、他のカタログ・サービス・ドメインを間接的に経由して変更を取得できます。5 つのカタログ・サービス・ドメインを持つライン・トポロジーを考えてみます。

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

  - A は、B、C、D、および E から B を介して変更をプルします。
  - B は、A と C からは直接、D と E からは C を介して変更をプルします。
  - C は、B と D からは直接、A からは B を介して、E からは D を介して変更をプルします。
  - D は、C と E からは直接、A と B からは C を介して変更をプルします。

– E は、D からは直接、A、B、および C からは D を介して変更をプルします。

カタログ・サービス・ドメイン A および E は、それぞれ単一カタログ・サービス・ドメインへのリンクのみを持っているので、配布の負荷は最も低くなります。 ドメイン B、C、および D は、それぞれ 2 つのドメインへのリンクを持っています。つまり、ドメイン B、C、および D 上の配布の負荷は、ドメイン A および E 上の負荷の 2 倍になります。ワークロードは、トポロジー内のドメイン総数ではなく、各ドメインのリンク数によって決まります。つまり、記述される負荷の分散は、ラインに 1000 ドメインを含んだとしても一定のままです。

マルチマスター・レプリカ生成のパフォーマンスに関する考慮事項

マルチマスター・レプリカ生成トポロジーを使用する際は、以下の制限を考慮してください。

- **変更の配布のチューニング**は、前のセクションで説明したとおりです。
- **レプリカ生成リンクのパフォーマンス** WebSphere eXtreme Scale は、任意の一対の JVM 間で、単一の TCP/IP ソケットを作成します。JVM 間のすべてのトラフィックは、マルチマスター・レプリカ生成のトラフィックも含め、単一ソケットを経由して発生します。カタログ・サービス・ドメインは少なくとも *n* 個のコンテナー JVM でホストされ、少なくとも *n* 個の TCP リンクをピア・カタログ・サービス・ドメインに提供しています。つまり、コンテナー数をより多く持つカタログ・サービス・ドメインには、より高いレプリカ生成のパフォーマンス・レベルがあります。より多くのコンテナーがあると、より多くのプロセッサーとネットワーク・リソースが必要になります。
- **TCP スライディング・ウィンドウのチューニングおよび RFC 1323** リンクの両端の RFC 1323 サポートを使用して、より多くのデータが往復します。このサポートにより高いスループットが実現され、約 16,000 の要因でウィンドウの容量が拡張されます。

  TCP ソケットが、スライディング・ウィンドウのメカニズムを使用して大量データのフローを制御することを思い出してください。このメカニズムは、通常、往復のインターバルのソケットを 64 KB に制限します。往復のインターバルが 100 ミリ秒の場合、追加チューニングをすることなく帯域幅は 640 KB/秒に制限されます。 リンクで使用可能な帯域幅を完全に使用する場合は、オペレーティング・システムに固有のチューニングが必要になることがあります。 ほとんどのオペレーティング・システムにはチューニング・パラメーターがあり、高度な待ち時間リンクのスループットを向上させる RFC 1323 オプションも含まれます。

  以下の複数の要因がレプリカ生成のパフォーマンスに影響する可能性があります。

  – eXtreme Scale が変更を取得する速度。
  – eXtreme Scale が取得レプリカ生成要求をサービスできる速度。
  – スライディング・ウィンドウの容量。
  – リンクの両端のネットワーク・バッファーをチューニングすると、eXtreme Scale は、効率的にソケット上の変更を取得します。
- **オブジェクト・シリアライゼーション** すべてのデータはシリアライズ可能でなければなりません。 カタログ・サービス・ドメインが `COPY_TO_BYTES` を使用して

いない場合、そのカタログ・サービス・ドメインは Java シリアライゼーションまたは ObjectTransformers を使用して、シリアライゼーション・パフォーマンスを最適化する必要があります。

- **圧縮** WebSphere eXtreme Scale は、デフォルトでカタログ・サービス・ドメイン間で送信されるすべてのデータを圧縮します。 現在、圧縮を使用不可にすることはできません。
- **メモリー・チューニング** マルチマスター・レプリカ生成トポロジーのメモリー使用量は、トポロジー内のカタログ・サービス・ドメイン数とはほとんど関係ありません。

  マルチマスター・レプリカ生成を使用すると、バージョン管理を扱うマップ・エントリーごとに一定の処理量が追加されます。各コンテナーはトポロジー内の各カタログ・サービス・ドメインの一定量のデータも追跡します。 2 つのカタログ・サービス・ドメインを持つトポロジーは、50 カタログ・サービス・ドメインを持つトポロジーとほぼ同じメモリーを使用します。 WebSphere eXtreme Scale は、その実装環境のリプレイ・ログや類似のキューを使用しません。すなわち、レプリカ生成リンクがかなりの期間使用できず、後で再開する場合、リカバリー構造は準備されていません。

他の WebSphere 製品とのインターオペラビリティー

WebSphere eXtreme Scale を他のサーバー製品 (WebSphere Application Server や WebSphere Application Server Community Edition など) と統合することができます。

WebSphere Application Server

WebSphere Application Server を WebSphere eXtreme Scale 構成のさまざまな側面に統合できます。データ・グリッド・アプリケーションをデプロイし、WebSphere Application Server を使用して、コンテナー・サーバーおよびカタログ・サーバーをホストできます。WebSphere Application Server セキュリティーを WebSphere eXtreme Scale 環境で使用することもできます。

WebSphere Portal

WebSphere Portal の HTTP セッションを WebSphere eXtreme Scale のデータ・グリッドに保持できます。

WebSphere Application Server Community Edition

WebSphere Application Server Community Edition はセッション状態を共有できますが、効率的でスケーラブルな方法ではありません。 WebSphere eXtreme Scale は、状態の複製に使用できるハイパフォーマンスな分散パーシスタンス層を提供しますが、WebSphere Application Server の外部にあるアプリケーション・サーバーと容易には統合しません。この 2 つの製品を統合することで、スケーラブルなセッション管理ソリューションを提供することができます。

WebSphere Real Time

WebSphere Real Time (業界最先端のリアルタイム Java 製品) のサポートにより、WebSphere eXtreme Scale は、Extreme Transaction Processing (XTP) アプリケーションが、より安定した予測可能な応答時間を得られるようにします。

第 3 章 シナリオ

シナリオでは、ユーザーが概念を理解したり、タスクを完遂したりするのに役立つ例を紹介します。完全なイメージを作るために、シナリオでは実世界の情報を使用します。

OSGi 環境を使用した eXtreme Scale プラグインの開発および実行

OSGi 環境で一般的な作業を実行するために、以下のシナリオを使用してください。例えば、OSGi フレームワークは、OSGi コンテナー内のサーバーおよびクライアントを始動する場合に最適であり、これにより、WebSphere eXtreme Scale プラグインをランタイム環境に動的に追加および更新できます。

以下のシナリオは、プラグインを動的にインストール、開始、停止、変更、およびアンインストールすることを可能にする動的プラグインの作成および実行について述べています。動的機能がなくとも、OSGi フレームワークを使用できるようにする別の適切なシナリオを実行することも可能です。アプリケーションをバンドルとしてもパッケージできます。これは、サービスによって定義され、サービスを介して通信されます。これらのサービス・ベースのバンドルには、より効率的な開発およびデプロイメント機能など、いくつかの利点があります。

シナリオの目標

このモジュールのレッスンを完了すると、以下のタスクの実行方法がわかります。

- OSGi 環境で使用するための eXtreme Scale 動的プラグインを作成します。
- eXtreme Scale コンテナーを、動的機能なしで OSGi 環境で実行します。

前提条件

OSGi サポートおよびそれが提供可能な利点について詳しくは、26 ページの『OSGi フレームワークの概要』のトピックを参照してください。

OSGi フレームワークの概要

OSGi は、Java に対して動的モジュール・システムを定義します。OSGi サービス・プラットフォームは、階層化アーキテクチャーを持ち、さまざまな標準 Java プロファイルで実行されるように設計されています。OSGi コンテナー内の WebSphere eXtreme Scale サーバーおよびクライアントを始動できます。

OSGi コンテナー内でアプリケーションを実行する利点

WebSphere eXtreme Scale OSGi サポートにより、Eclipse Equinox OSGi フレームワークに製品をデプロイできます。これまで、eXtreme Scale で使用するプラグインを更新する場合は、Java 仮想マシン (JVM) を再始動して、プラグインの新規バージョンを適用する必要がありました。現在は、OSGi フレームワークが提供する動的更新機能を使用して、JVM を再始動せずにプラグイン・クラスを更新できます。これ

らのプラグインは、ユーザー・バンドルによってサービスとしてエクスポートされます。WebSphere eXtreme Scale は、OSGi レジストリーでルックアップして、サービス (複数可) にアクセスします。

eXtreme Scale コンテナーは、OSGi Configuration Admin サービスまたは OSGi Blueprint を使用して容易かつ動的に始動するように構成できます。新規データ・グリッドをその配置ストラテジーを使用してデプロイする場合は、OSGi 構成を作成するか、eXtreme Scale 記述子 XML ファイルを使用してバンドルをデプロイすることによって、これを行うことができます。OSGi サポートを使用すると、eXtreme Scale 構成データを含むアプリケーション・バンドルを、システム全体を再始動することなく、インストール、開始、停止、更新、およびアンインストールできます。この機能を使用して、データ・グリッドを中断することなくアプリケーションをアップグレードできます。

プラグイン Bean およびプラグイン・サービスをカスタム断片有効範囲で構成することができます。これにより、データ・グリッド内で実行中の他のサービスに対して高度な統合オプションを使用することができます。各プラグインは OSGi Blueprint ランキングを使用して、プラグインの各インスタンスが正しいバージョンでアクティブ化されていることを確認できます。OSGi Managed Bean (MBean) と **xscmd** ユーティリティーが提供され、これにより、eXtreme Scale プラグイン OSGi サービスとそのランキングを照会することができます。

この機能によって、管理者は、構成および管理に関する潜在的なエラーを迅速に認識することができ、eXtreme Scale によって使用中のプラグイン・サービス・ランキングをアップグレードすることができます。

OSGi バンドル

OSGi フレームワーク内のプラグインと対話し、それをデプロイするには、*バンドル*を使用する必要があります。 OSGi サービス・プラットフォームにおけるバンドルとは、Java アーカイブ (JAR) ファイルです。このファイルには Java コード、リソース、およびマニフェスト (バンドルとその依存関係についての記述) が含まれます。バンドルはアプリケーションのデプロイメントの単位です。 eXtreme Scale 製品は、以下のバンドル・タイプをサポートします。

サーバー・バンドル

サーバー・バンドルは `objectgrid.jar` ファイルであり、eXtreme Scale スタンドアロン・サーバーのインストールでインストールされます。これは、eXtreme Scale サーバーの稼働に必要で、eXtreme Scale クライアントまたはローカルのメモリー内のキャッシュの実行にも使用できます。`objectgrid.jar` ファイルのバンドル ID は com.ibm.websphere.xs.server_`<version>` で、バージョンのフォーマットは `<Version>.<Release>.<Modification>` です。例えば、eXtreme Scale バージョン 7.1.1 のサーバー・バンドルは、com.ibm.websphere.xs.server_7.1.1 です。

クライアント・バンドル

クライアント・バンドルは `ogclient.jar` ファイルであり、eXtreme Scale スタンドアロンおよびクライアントのインストールでインストールされます。これは、eXtreme Scale クライアントまたはローカルのメモリー内のキャッシュの実行に使用されます。`ogclient.jar` ファイルのバンドル ID

は、com.ibm.websphere.xs.client_version です。ここで、version は、<Version>.<Release>.<Modification> の形式です。例えば、eXtreme Scale バージョン 7.1.1 のクライアント・バンドルは、com.ibm.websphere.xs.client_7.1.1 です。

制限

オブジェクト・リクエスト・ブローカー (ORB) を再始動できないため、eXtreme Scale バンドルを再始動できません。eXtreme Scale サーバーを再始動するには、OSGi フレームワークを再開する必要があります。

クライアントおよびサーバーの Eclipse Gemini を持つ Eclipse Equinox OSGi フレームワークのインストール

OSGi フレームワークに WebSphere eXtreme Scale をデプロイするには、Eclipse Equinox 環境をセットアップする必要があります。

このタスクについて

このタスクを実行するには、Blueprint フレームワークをダウンロードしてインストールする必要があります。そうすれば、後で、JavaBeans を構成し、それをサービスとして公開することができます。サービスの使用が重要である理由は、プラグインを OSGi サービスとして公開すれば、そのサービスを eXtreme Scale ランタイム環境が使用できるからです。製品は、Eclipse Equinox コア OSGi フレームワークの中で、Eclipse Gemini と Apache Aries の 2 つの blueprint コンテナーをサポートします。次の手順を使用して、Eclipse Gemini コンテナーをセットアップします。

手順

1. Eclipse Web サイト から、Eclipse Equinox SDK Version 3.6.1 以降をダウンロードします。 Equinox フレームワーク用のディレクトリーを作成します。例えば、/opt/equinox です。以下の説明では、このディレクトリーを equinox_root と呼びます。圧縮ファイルを equinox_root ディレクトリーに解凍します。
2. Eclipse Web サイトから、gemini-blueprint 1.0.0 圧縮ファイルをダウンロードします。 ファイルの内容を一時ディレクトリーに解凍し、解凍された次のファイルを equinox_root/plugins ディレクトリーにコピーします。

   ```
   dist/gemini-blueprint-core-1.0.0.jar
   dist/gemini-blueprint-extender-1.0.0.jar
   dist/gemini-blueprint-io-1.0.0.jar
   ```

3. 次の SpringSource Web ページから、Spring Framework Version 3.0.5 をダウンロードします。http://www.springsource.com/download/community それを一時ディレクトリーに解凍し、解凍された次のファイルを equinox_root/plugins ディレクトリーにコピーします。

   ```
   org.springframework.aop-3.0.5.RELEASE.jar
   org.springframework.asm-3.0.5.RELEASE.jar
   org.springframework.beans-3.0.5.RELEASE.jar
   org.springframework.context-3.0.5.RELEASE.jar
   org.springframework.core-3.0.5.RELEASE.jar
   org.springframework.expression-3.0.5.RELEASE.jar
   ```

4. SpringSource Web ページから、AOP Alliance Java アーカイブ (JAR) ファイルをダウンロードします。 com.springsource.org.aopalliance-1.0.0.jar を equinox_root/plugins ディレクトリーにコピーします。
5. SpringSource Web ページから、Apache commons logging 1.1.1 JAR ファイルをダウンロードします。 com.springsource.org.apache.commons.logging-1.1.1.jar ファイルを equinox_root/plugins ディレクトリーにコピーします。
6. Luminis OSGi Configuration Admin コマンド行クライアントをダウンロードします。このバンドルを使用して、OSGi 管理構成を管理します。次の Web ページから、JAR ファイルをダウンロードできます。https://opensource.luminis.net/wiki/display/SITE/OSGi+Configuration+Admin+command+line+client net.luminis.cmc-0.2.5.jar を equinox_root/plugins ディレクトリーにコピーします。
7. 次の Web ページから、Apache Felix file installation Version 3.0.2 バンドルをダウンロードします。http://felix.apache.org/site/index.html org.apache.felix.fileinstall-3.0.2.jar ファイルを equinox_root/plugins ディレクトリーにコピーします。
8. equinox_root/plugins ディレクトリーの中に、構成ディレクトリーを作成します。例えば次のとおりです。

   ```
   mkdir equinox_root/plugins/configuration
   ```

9. 次の config.ini ファイルを、equinox_root/plugins/configuration ディレクトリーの中に作成します。このとき、*equinox_root* を、使用する equinox_root ディレクトリーの絶対パスに置き換え、各行の円記号 (¥) の後のすべての後続スペースを削除します。ファイルの最後に、ブランク行を含める必要があります。例えば次のとおりです。

   ```
   osgi.noShutdown=true
   osgi.java.profile.bootdelegation=none
   org.osgi.framework.bootdelegation=none
   eclipse.ignoreApp=true
   osgi.bundles=¥
   org.eclipse.osgi.services_3.2.100.v20100503.jar@1:start, \
   org.eclipse.osgi.util_3.2.100.v20100503.jar@1:start, \
   org.eclipse.equinox.cm_1.0.200.v20100520.jar@1:start, \
   com.springsource.org.apache.commons.logging-1.1.1.jar@1:start, ¥
   com.springsource.org.aopalliance-1.0.0.jar@1:start, ¥
   org.springframework.aop-3.0.5.RELEASE.jar@1:start, ¥
   org.springframework.asm-3.0.5.RELEASE.jar@1:start, ¥
   org.springframework.beans-3.0.5.RELEASE.jar@1:start, ¥
   org.springframework.context-3.0.5.RELEASE.jar@1:start, ¥
   org.springframework.core-3.0.5.RELEASE.jar@1:start, ¥
   org.springframework.expression-3.0.5.RELEASE.jar@1:start, ¥
   org.apache.felix.fileinstall-3.0.2.jar@1:start, ¥
   net.luminis.cmc-0.2.5.jar@1:start, ¥
   gemini-blueprint-core-1.0.0.jar@1:start, ¥
   gemini-blueprint-extender-1.0.0.jar@1:start, ¥
   gemini-blueprint-io-1.0.0.jar@1:start
   ```

   既に環境をセットアップしている場合は、次のディレクトリーを削除することで、Equinox プラグイン・リポジトリーをクリーンアップできます。

   ```
   equinox_root¥plugins¥configuration¥org.eclipse.osgi
   ```

10. 次のコマンドを実行して、Equinox コンソールを開始します。

    別のバージョンの Equinox を実行している場合、JAR ファイル名は次の例の中のものと異なります。

    ```
    java -jar plugins\org.eclipse.osgi_3.6.1.R36x_v20100806.jar -console
    ```

eXtreme Scale バンドルのインストール

WebSphere eXtreme Scale には、Eclipse Equinox OSGi フレームワークにインストールできるバンドルが組み込まれています。OSGi 内で eXtreme Scale サーバーを開始したり、eXtreme Scale クライアントを使用したりするには、これらのバンドルが必要です。

始める前に

このタスクは、次の製品のインストールが完了していることを前提としています。

- Eclipse Equinox OSGi フレームワーク
- eXtreme Scale スタンドアロン・クライアントまたはサーバー

このタスクについて

eXtreme Scale には、2 つのバンドルが組み込まれています。各 OSGi フレームワークでは、次のバンドルのいずれか 1 つのみが必要になります。

objectgrid.jar

サーバー・バンドルは `objectgrid.jar` ファイルであり、eXtreme Scale スタンドアロン・サーバーのインストールによってインストールされます。eXtreme Scale サーバーを実行するために必要なバンドルですが、eXtreme Scale クライアントまたはローカルのメモリー内キャッシュの実行にも使用できます。`objectgrid.jar` ファイルのバンドル ID は com.ibm.websphere.xs.server_<version> で、バージョンのフォーマットは <Version>.<Release>.<Modification> です。例えば、eXtreme Scale バージョン 7.1.1 のサーバー・バンドルは com.ibm.websphere.xs.server_7.1.1 です。

ogclient.jar

`ogclient.jar` バンドルは、eXtreme Scale スタンドアロンおよびクライアントのインストール済み環境にインストールされ、eXtreme Scale クライアントまたはローカルのメモリー内キャッシュを実行するために使用されます。`ogclient.jar` ファイルのバンドル ID は com.ibm.websphere.xs.client_<version> で、バージョンのフォーマットは <Version>_<Release>_<Modification で す。例えば、eXtreme Scale バージョン 7.1.1 のクライアント・バンドルは com.ibm.websphere.xs.client_7.1.1 です。

eXtreme Scale プラグインの作成法の詳細については、システム API とプラグインのトピックを参照してください。

手順

OSGi コンソールを使用して、eXtreme Scale クライアントまたはサーバー・バンドルを Eclipse Equinox OSGi フレームワークにインストールするには、以下のようにします。

1. コンソールを有効にするよう指定して Eclipse Equinox フレームワークを開始します。例えば、次のようにします。

```
java_home/bin/java -jar <equinox_root>/plugins/
org.eclipse.osgi_3.6.1.R36x_v20100806.jar -console
```

2. Equinox コンソールで、eXtreme Scale クライアントまたはサーバー・バンドルをインストールします。

   ```
   osgi> install file:///<path to bundle>
   ```

3. Equinox が、新しくインストールされたバンドルのバンドル ID を表示します。

   ```
   Bundle id is 25
   ```

4. Equinox コンソールで、次のようにバンドルを開始します。ここで、<id> は、バンドルのインストール時に割り当てられたバンドル ID です。

   ```
   osgi>  start <id>
   ```

5. Equinox コンソールで、サービス状況を取得して、バンドルが開始したことを確認します。例えば、次のようにします。

   ```
   osgi> ss
   ```

   バンドルが正常に開始した場合、バンドルは ACTIVE 状態を表示します。例えば、次のとおりです。

   ```
   25      ACTIVE      com.ibm.websphere.xs.server_7.1.1
   ```

config.ini ファイルを使用して、eXtreme Scale クライアントまたはサーバー・バンドルを Eclipse Equinox OSGi フレームワークにインストールするには、以下のようにします。

6. eXtreme Scale クライアントまたはサーバー (objectgrid.jar または ogclient.jar) バンドルを <wxs_install_root>/ObjectGrid/lib から、次の例のような Eclipse Equinox プラグイン・ディレクトリーにコピーします。 <equinox_root>/plugins
7. Eclipse Equinox config.ini 構成ファイルを編集し、バンドルを osgi.bundles プロパティーに追加します。例えば、次のとおりです。

   ```
   osgi.bundles=¥
   org.eclipse.osgi.services_3.2.100.v20100503.jar@1:start, \
   org.eclipse.osgi.util_3.2.100.v20100503.jar@1:start, \
   org.eclipse.equinox.cm_1.0.200.v20100520.jar@1:start, \
   objectgrid.jar@1:start
   ```

   重要: 最後のバンドル名の後にブランク行があることを確認してください。各バンドルはコンマで区切ります。

8. コンソールを有効にするよう指定して Eclipse Equinox フレームワークを開始します。例えば、次のようにします。

   ```
   java_home/bin/java -jar <equinox_root>/plugins/
   org.eclipse.osgi_3.6.1.R36x_v20100806.jar -console
   ```

9. Equinox コンソールで、サービス状況を取得して、バンドルが開始したことを確認します。

   ```
   osgi> ss
   ```

   バンドルが正常に開始した場合、バンドルは ACTIVE 状態を表示します。例えば、次のとおりです。

   ```
   25      ACTIVE      com.ibm.websphere.xs.server_7.1.1
   ```

タスクの結果

Eclipse Equinox OSGi フレームワークに eXtreme Scale サーバーまたはクライアント・バンドルがインストールされ、開始されました。

OSGi 環境で使用する eXtreme Scale 動的プラグインのビルドと実行

すべての eXtreme Scale プラグインを OSGi 環境用に構成できます。動的プラグインの主なメリットは、グリッドをシャットダウンしなくともアップグレードが可能なことです。これにより、グリッド・コンテナー・プロセスを再始動せずにアプリケーションの移行が可能になります。

このタスクについて

WebSphere eXtreme Scale OSGi サポートにより、Eclipse Equinox などの OSGi フレームワークに製品をデプロイできます。これまで、eXtreme Scale で使用するプラグインを更新する場合は、Java 仮想マシン (JVM) を再始動して、プラグインの新規バージョンを適用する必要がありました。eXtreme Scale が提供する動的プラグイン・サポートと OSGi フレームワークが提供するバンドル更新機能により、現在では JVM を再始動せずにプラグイン・クラスを更新できるようになりました。これらのプラグインは、*バンドル* によりサービスとしてエクスポートされます。WebSphere eXtreme Scale は、OSGi レジストリーをルックアップすることでサービスにアクセスします。OSGi サービス・プラットフォームにおけるバンドルとは、Java アーカイブ (JAR) ファイルです。このファイルには Java コード、リソース、およびマニフェスト (バンドルとその依存関係についての記述) が含まれます。バンドルはアプリケーションのデプロイメントの単位です。

手順

1. eXtreme Scale 動的プラグインをビルドします。
2. OSGi Blueprint を使用して eXtreme Scale プラグインを構成します。
3. OSGi 対応プラグインをインストールして開始します。

eXtreme Scale 動的プラグインのビルド

WebSphere eXtreme Scale には、ObjectGrid および BackingMap プラグインが含まれます。 これらのプラグインは Java で実装され、ObjectGrid 記述子 XML ファイルを使用して構成されます。動的にアップグレードできる動的プラグインを作成する場合、動的プラグインは更新時に何らかのアクションを完了する必要がある可能性があるため、ObjectGrid および BackingMap ライフサイクル・イベントを認識する必要があります。ライフサイクルのコールバック・メソッド、イベント・リスナー、あるいはその両方でプラグイン・バンドルを拡張すると、プラグインが適切なタイミングでそれらのアクションを完了できるようになります。

始める前に

このトピックは、適切なプラグインのビルドが完了していることを前提とします。eXtreme Scale プラグインの作成法の詳細については、システム API とプラグインのトピックを参照してください。

このタスクについて

すべての eXtreme Scale プラグインは、BackingMap または ObjectGrid インスタンスに適用されます。多くのプラグインは他のプラグインと対話もします。例えば、Loader および TransactionCallback プラグインは連携して、データベース・トランザクションやさまざまなデータベース JDBC 呼び出しと適切に対話します。プラグイ

ンの中には、パフォーマンスを改善するために、他のプラグインの構成データをキャッシュに入れる必要があるものもあります。

BackingMapLifecycleListener および ObjectGridLifecycleListener プラグインは、個別の BackingMap および ObjectGrid インスタンスのライフサイクル操作が可能です。このプロセスにより、プラグインは親の BackingMap または ObjectGrid とそれぞれのプラグインに変更があると、通知を受けることができます。 BackingMap プラグインは BackingMapLifecyleListener インターフェースを実装し、ObjectGrid プラグインは ObjectGridLifecycleListener インターフェースを実装します。親の BackingMap または ObjectGrid のライフサイクルに変化があると、これらのプラグインが自動的に呼び出されます。ライフサイクル・プラグインの詳細については、プラグイン・ライフサイクルの管理のトピックを参照してください。

ライフサイクル・メソッドまたはイベント・リスナーを使用したバンドルの拡張は、次の共通タスクの中で必要になる可能性があります。

- リソース (スレッド、メッセージング・サブスクライバーなど) の開始と停止
- ピア・プラグインが更新された際の通知指定、プラグインへの直接アクセスの許可、変更の検出

別のプラグインに直接アクセスするときは、必ず OSGi コンテナー経由でそのプラグインにアクセスして、システムのすべてのパーツが正しいプラグインを参照できるようにしてください。例えば、アプリケーション内のあるコンポーネントがプラグインのインスタンスを直接参照するかキャッシュに入れると、そのプラグインが動的に更新された後も、コンポーネントはそのバージョンのプラグインへの参照を維持します。この振る舞いは、メモリー・リークのほかにアプリケーション関連の問題の原因にもなります。したがって、コードを作成するときは、OSGi の getService() セマンティクスを使用して参照を獲得する動的プラグインを使用してください。アプリケーションが 1 つ以上のプラグインをキャッシュに入れる必要がある場合は、ObjectGridLifecycleListener および BackingMapLifecycleListener インターフェースを使用してライフサイクル・イベントを listen します。また、アプリケーションは、スレッド・セーフな方法で、必要なときにキャッシュをリフレッシュできなければなりません。

OSGi で使用するすべての eXtreme Scale プラグインは、BackingMapPlugin または ObjectGridPlugin インターフェースもそれぞれ実装する必要があります。MapSerializerPlugin インターフェースなどの新しいプラグインでは、この実装が実施されます。これらのインターフェースは、状態をプラグインに注入したり、プラグインのライフサイクルを制御したりするための一貫性のあるインターフェースを eXtreme Scale ランタイム環境と OSGi に提供します。

このタスクを使用して、ピア・プラグインが更新されたときに通知するよう指定します。リスナー・インスタンスを生成するリスナー・ファクトリーを作成してもかまいません。

手順

- ObjectGrid プラグイン・クラスを更新して、ObjectGridPlugin インターフェースを実装します。 このインターフェースは、eXtreme Scale がプラグインを初期化したり、ObjectGrid インスタンスを設定したり、プラグインを破棄したりできるようにするメソッドを組み込みます。次のサンプル・コードを参照してください。

```
package com.mycompany;
import com.ibm.websphere.objectgrid.plugins.ObjectGridPlugin;
...

public class MyTranCallback implements TransactionCallback, ObjectGridPlugin {

    private ObjectGrid og = null;

    private enum State {
        NEW, INITIALIZED, DESTROYED
    }

    private State state = State.NEW;

    public void setObjectGrid(ObjectGrid grid) {
        this.og = grid;
    }

    public ObjectGrid getObjectGrid() {
        return this.og;
    }
    void initialize() {
        // Handle any plug-in initialization here.  This is called by
        // eXtreme Scale, and not the OSGi bean manager.
        state = State.INITIALIZED;
    }
    boolean isInitialized() {
        return state == State.INITIALIZED;
    }

    public void destroy() {
        // Destroy the plug-in and release any resources.  This
        // can be callsed by the OSGi Bean Manager or by eXtreme Scale.
        state = State.DESTROYED;
    }

    public boolean isDestroyed() {
        return state == State.DESTROYED;
    }
}
```

- ObjectGrid プラグイン・クラスを更新して、ObjectGridLifecycleListener インターフェースを実装します。次のサンプル・コードを参照してください。

```
package com.mycompany;
import com.ibm.websphere.objectgrid.plugins.ObjectGridLifecycleListener;
      import com.ibm.websphere.objectgrid.plugins.ObjectGridLifecycleListener.LifecycleEvent;
      ...

      public class MyTranCallback implements TransactionCallback, ObjectGridPlugin, ObjectGridLifecycleListener{
          public void objectGridStateChanged(LifecycleEvent event) {
              switch(event.getState()) {
              case NEW:
              case DESTROYED:
              case DESTROYING:
              case INITIALIZING:
                  break;
              case INITIALIZED:
                  // Lookup a Loader or MapSerializerPlugin using
                  // OSGi or directly from the ObjectGrid instance.
                  lookupOtherPlugins()
                  break;
              case STARTING:
              case PRELOAD:
                  break;
              case ONLINE:
                  if (event.isWritable()) {
                      startupProcessingForPrimary();
                  } else {
                      startupProcessingForReplica();
                  }
                  break;
              case QUIESCE:
                  if (event.isWritable()) {
                      quiesceProcessingForPrimary();
                  } else {
                      quiesceProcessingForReplica();
                  }
                  break;
              case OFFLINE:
                  shutdownShardComponents();
                  break;
              }
          }
          ...
      }
```

- BackingMap プラグインを更新します。 BackingMap プラグイン・クラスを更新して、BackingMap インターフェースを実装します。このインターフェースは、

eXtreme Scale がプラグインを初期化したり、BackingMap インスタンスを設定したり、プラグインを破棄したりできるようにするメソッドを組み込みます。次のサンプル・コードを参照してください。

```
package com.mycompany;
import com.ibm.websphere.objectgrid.plugins.BackingMapPlugin;
...

public class MyLoader implements Loader, BackingMapPlugin {

    private BackingMap bmap = null;

    private enum State {
        NEW, INITIALIZED, DESTROYED
    }

    private State state = State.NEW;

    public void setBackingMap(BackingMap map) {
        this.bmap = map;
    }

    public BackingMap getBackingMap() {
        return this.bmap;
    }
    void initialize() {
        // Handle any plug-in initialization here.  This is called by
        // eXtreme Scale, and not the OSGi bean manager.
        state = State.INITIALIZED;
    }
    boolean isInitialized() {
        return state == State.INITIALIZED;
    }

    public void destroy() {
        // Destroy the plug-in and release any resources.  This
        // can be callsed by the OSGi Bean Manager or by eXtreme Scale.
        state = State.DESTROYED;
    }

    public boolean isDestroyed() {
        return state == State.DESTROYED;
    }
}
```

- BackingMap プラグイン・クラスを更新して、BackingMapLifecycleListener インターフェースを実装します。 次のサンプル・コードを参照してください。

```
package com.mycompany;

      import com.ibm.websphere.objectgrid.plugins.BackingMapLifecycleListener;
      import com.ibm.websphere.objectgrid.plugins.BackingMapLifecycleListener.LifecycleEvent;
      ...

      public class MyLoader implements Loader, ObjectGridPlugin, ObjectGridLifecycleListener{
          ...
          public void backingMapStateChanged(LifecycleEvent event) {
              switch(event.getState()) {
              case NEW:
              case DESTROYED:
              case DESTROYING:
              case INITIALIZING:
                  break;
              case INITIALIZED:
                  // Lookup a MapSerializerPlugin using
                  // OSGi or directly from the ObjectGrid instance.
                  lookupOtherPlugins()
                  break;
              case STARTING:
              case PRELOAD:
                  break;
              case ONLINE:
                  if (event.isWritable()) {
                      startupProcessingForPrimary();
                  } else {
                      startupProcessingForReplica();
                  }
                  break;
              case QUIESCE:
                  if (event.isWritable()) {
                      quiesceProcessingForPrimary();
                  } else {
                      quiesceProcessingForReplica();
                  }
                  break;
              case OFFLINE:
                  shutdownShardComponents();
                  break;
```

```
        }
    }
    ...
}
```

タスクの結果

ObjectGridPlugin または BackingMapPlugin インターフェースを実装することで、eXtreme Scale はプラグインのライフサイクルを正しいタイミングで制御できます。

ObjectGridLifecycleListener または BackingMapLifecycleListener インターフェースを実装すると、プラグインは、関連付けられた ObjectGrid または BackingMap ライフサイクル・イベントのリスナーとして自動的に登録されます。すべての ObjectGrid および BackingMap プラグインの初期化が完了し、検索および使用が可能になったことをシグナル通知するときは INITIALIZING イベントが使用されます。ObjectGrid がオンラインになり、イベントの処理を開始する準備ができたことをシグナル通知するときは ONLINE イベントが使用されます。

OSGi Blueprint での eXtreme Scale プラグインの構成

eXtreme Scale ObjectGrid および BackingMap プラグインはすべて、Eclipse Gemini または Apache Aries で使用可能な OSGi Blueprint サービスを使用して OSGi Bean およびサービスとして定義できます。

始める前に

プラグインを OSGi サービスとして構成するには、プラグインを OSGi バンドルにパッケージ化し、必要なプラグインの基本原則を理解する必要があります。バンドルは、WebSphere eXtreme Scale サーバー・パッケージまたはクライアント・パッケージに加えてプラグインが必要とするその他の従属パッケージをインポートするか、eXtreme Scale サーバー・バンドルまたはクライアント・バンドルへのバンドル依存関係を作成しなければなりません。このトピックでは、Blueprint XML を構成して、プラグイン Bean を作成し、それらを eXtreme Scale で使用できるように OSGi サービスとして公開する方法を説明します。

このタスクについて

Bean とサービスは Blueprint XML ファイル内に定義します。そうすると、Blueprint コンテナーによって Bean が検出および作成され、Bean 同士がワイヤリングされ、サービスとして公開されます。このプロセスにより、eXtreme Scale サーバー・バンドルとクライアント・バンドルを含め、その他の OSGi バンドルで Bean が使用可能になります。

eXtreme Scale で使用するカスタム・プラグイン・サービスを作成する場合、プラグインをホスティングするバンドルは、Blueprint を使用するように構成しなければなりません。さらに、Blueprint XML ファイルを作成し、そのファイルをバンドル内に保管しなければなりません。Blueprint Container 仕様の全般的な知識を得るには、Blueprint Container 仕様による OSGi アプリケーションの構築を参照してください。

手順

1. Blueprint XML ファイルを作成します。 ファイルには任意の名前を付けることができます。ただし、次のように blueprint 名前空間を含める必要があります。

```
<?xml version="1.0" encoding="UTF-8"?>
<blueprint xmlns="http://www.osgi.org/xmlns/blueprint/v1.0.0">
...
</blueprint>
```

2. eXtreme Scale プラグインごとに Bean 定義を Blueprint XML ファイル内に作成します。

   Bean は <bean> エレメントを使用して定義し、他の Bean 参照にワイヤリングでき、初期化パラメーターを組み込むことができます。

   重要: Bean の定義時は、正しいスコープを使用する必要があります。Blueprint は singleton スコープとプロトタイプ・スコープをサポートします。eXtreme Scale はカスタム断片スコープもサポートします。

   すべての Bean は、関連付けられる各 ObjectGrid 断片または BackingMap インスタンスで固有でなければならないため、ほとんどの eXtreme Scale プラグインはプロトタイプ・スコープまたは断片スコープの Bean として定義します。正しいインスタンスの取得を可能にするために Bean を他のコンテキストで使用する場合、断片スコープの Bean が便利です。

   プロトタイプ・スコープの Bean を定義するには、Bean の scope="prototype" 属性を使用します。

   ```
   <bean id="myPluginBean" class="com.mycompany.MyBean" scope="prototype">
   ...
   </bean>
   ```

   断片スコープの Bean を定義するには、objectgrid 名前空間を XML スキーマに追加し、Bean の scope="objectgrid:shard" 属性を使用してください。

   ```
   <?xml version="1.0" encoding="UTF-8"?>

   <blueprint xmlns="http://www.osgi.org/xmlns/blueprint/v1.0.0"
              xmlns:objectgrid="http://www.ibm.com/schema/objectgrid"

       xsi:schemaLocation="http://www.ibm.com/schema/objectgrid
                   http://www.ibm.com/schema/objectgrid/objectgrid.xsd">

       <bean id="myPluginBean" class="com.mycompany.MyBean"
      scope="objectgrid:shard">
       ...
       </bean>

   ...
   ```

3. 各プラグイン Bean の PluginServiceFactory Bean 定義を作成します。 正しい Bean スコープを適用できるように、すべての eXtreme Scale Bean に PluginServiceFactory Bean を定義する必要があります。eXtreme Scale には、ユーザーが使用できる BlueprintServiceFactory が組み込まれています。それには設定が必要な 2 つのプロパティーがあります。blueprintContainer プロパティーには blueprintContainer 参照を設定し、beanId プロパティーには Bean ID 名を設定する必要があります。eXtreme Scale が適切な Bean のインスタンスを生成するためにサービスを検索すると、サーバーは Blueprint コンテナーを使用して Bean コンポーネント・インスタンスを検索します。

```
bean id="myPluginBeanFactory"
    class="com.ibm.websphere.objectgrid.plugins.osgi.BluePrintServiceFactory">
    <property name="blueprintContainer" ref="blueprintContainer" />
    <property name="beanId" value="myPluginBean" />
</bean>
```

4. 各 PluginServiceFactory Bean のサービス・マネージャーを作成します。 各サービス・マネージャーは、<service> エレメントを使用して PluginServiceFactory Bean を公開します。サービス・エレメントは、OSGi に公開する名前、PluginServiceFactory Bean への参照、公開するインターフェース、およびサービスのランキングを識別します。eXtreme Scale はサービス・マネージャー・ランキングを使用して、eXtreme Scale グリッドがアクティブなときにサービス・アップグレードを実行します。 ランキングが指定されない場合、OSGi フレームワークはランキング 0 を想定します。詳細については、サービス・ランキングの更新を参照してください。

   Blueprint には、サービス・マネージャーを構成するためのオプションがいくつかあります。PluginServiceFactory Bean の単純なサービス・マネージャーを定義するには、PluginServiceFactory Bean ごとに <service> エレメントを作成します。

   ```
   <service ref="myPluginBeanFactory"
       interface="com.ibm.websphere.objectgrid.plugins.osgi.PluginServiceFactory"
       ranking="1">
   </service>
   ```

5. Blueprint XML ファイルをプラグイン・バンドル内に保管します。 Blueprint XML ファイルは OSGI-INF/blueprint ディレクトリー内に保管し、Blueprint コンテナーが検出されるようにしなければなりません。

   Blueprint XML ファイルを他のディレクトリーに保管するには、次の Bundle-Blueprint マニフェスト・ヘッダーを指定する必要があります。

   ```
   Bundle-Blueprint: OSGI-INF/blueprint.xml
   ```

タスクの結果

これで、OSGi Blueprint コンテナー内に公開される eXtreme Scale プラグインが構成されました。さらに、OSGi Blueprint サービスを使用してプラグインを参照するように ObjectGrid 記述子 XML ファイルも構成されました。

OSGi 対応プラグインのインストールと開始

このタスクでは、動的プラグイン・バンドルを OSGi フレームワークにインストールします。その後、そのプラグインを開始します。

始める前に

このトピックは、以下のタスクが完了していることを前提としています。

- eXtreme Scale サーバーまたはクライアント・バンドルを Eclipse Equinox OSGi フレームワークにインストール済みである。 197 ページの『eXtreme Scale バンドルのインストール』を参照してください。
- 1 つ以上の動的 BackingMap または ObjectGrid プラグインを実装済みである。 199 ページの『eXtreme Scale 動的プラグインのビルド』を参照してください。
- 動的プラグインを OSGi バンドル内に OSGi サービスとしてパッケージ化済みである。

このタスクについて

このタスクでは、Eclipse Equinox コンソールを使用してバンドルをインストールする方法を説明します。バンドルはいくつかの異なる方式 (config.ini 構成ファイルを変更するなど) を使用してインストールできます。Eclipse Equinox を組み込む製品には、バンドルを管理するための代わりの方式があります。Eclipse Equinox で config.ini ファイルにバンドルを追加する方法について詳しくは、「Eclipse runtime options」を参照してください。

OSGi では、重複サービスを持つバンドルの開始が許可されます。WebSphere eXtreme Scale は最新のサービス・ランキングを使用します。1 つの eXtreme Scale データ・グリッド内で複数の OSGi フレームワークを開始する場合は、各サーバーで正しいサービス・ランキングが開始されるようにしなければなりません。そうしないと、いろいろなバージョンが混在したグリッドが開始されます。

データ・グリッドで使用中のバージョンを確認するには、xscmd ユーティリティーを使用して、現在のランキングと使用可能なランキングを確認します。使用可能なサービス・ランキングの詳細については、**xscmd** による eXtreme Scale プラグインの OSGi サービスの更新を参照してください。

手順

OSGi コンソールを使用してプラグイン・バンドルを Eclipse Equinox OSGi フレームワークにインストールします。

1. コンソールを有効にするよう指定して Eclipse Equinox フレームワークを開始します。例えば、次のようにします。

   ```
   <java_home>/bin/java -jar <equinox_root>/plugins/org.eclipse.osgi_3.6.1.R36x_v20100806.jar -console
   ```

2. Equinox コンソールで、プラグイン・バンドルをインストールします。

   ```
   osgi> install file:///<path to bundle>
   ```

   Equinox が、新しくインストールされたバンドルのバンドル ID を表示します。

   ```
   Bundle id is 17
   ```

3. Equinox コンソールで、次の行を入力してバンドルを開始します。ここで、<id> は、バンドルのインストール時に割り当てられたバンドル ID です。

   ```
   osgi>  install <id>
   ```

4. Equinox コンソールで、サービス状況を取得して、バンドルが開始したことを確認します。

   ```
   osgi> ss
   ```

   バンドルが正常に開始した場合、バンドルは ACTIVE 状態を表示します。例えば、次のとおりです。

   ```
   17      ACTIVE      com.mycompany.plugin.bundle_VRM
   ```

config.ini ファイルを使用して、プラグイン・バンドルを Eclipse Equinox OSGi フレームワークにインストールします。

5. プラグイン・バンドルを次の例のような Eclipse Equinox プラグイン・ディレクトリーにコピーします。

   ```
   <equinox_root>/plugins
   ```

6. Eclipse Equinox config.ini 構成ファイルを編集し、バンドルを osgi.bundles プロパティーに追加します。例えば、次のとおりです。

```
osgi.bundles=¥
org.eclipse.osgi.services_3.2.100.v20100503.jar@1:start, \
org.eclipse.osgi.util_3.2.100.v20100503.jar@1:start, \
org.eclipse.equinox.cm_1.0.200.v20100520.jar@1:start, \
com.mycompany.plugin.bundle_VRM.jar@1:start
```

重要: 最後のバンドル名の後にブランク行が存在することを確認してください。各バンドルはコンマで区切ります。

7. コンソールを有効にするよう指定して Eclipse Equinox フレームワークを開始します。例えば、次のようにします。

```
<java_home>/bin/java -jar <equinox_root>/plugins/org.eclipse.osgi_3.6.1.R36x_v20100806.jar -console
```

8. Equinox コンソールで、サービス状況を取得して、バンドルが開始したことを確認します。例えば、次のようにします。

```
osgi> ss
```

バンドルが正常に開始した場合、バンドルは ACTIVE 状態を表示します。例えば、次のとおりです。

```
17      ACTIVE      com.mycompany.plugin.bundle_VRM
```

タスクの結果

これでプラグイン・バンドルがインストールされ、開始されました。今度は、eXtreme Scale コンテナーまたはクライアントを開始できます。eXtreme Scale プラグインの作成法の詳細については、システム API とプラグインのトピックを参照してください。

OSGi 環境での動的プラグインを持つ eXtreme Scale コンテナーの実行

Eclipse Gemini または Apache Aries を使用する Eclipse Equinox OSGi フレームワーク内でアプリケーションがホスティングされる場合は、このタスクに従って OSGi に WebSphere eXtreme Scale アプリケーションをインストールし、構成できます。

始める前に

このタスクを開始する前に、必ず次のタスクを完了してください。

- Eclipse Gemini を使用する Eclipse Equinox OSGi フレームワークのインストール
- OSGi 環境で使用する eXtreme Scale 動的プラグインのビルドと実行

このタスクについて

動的プラグインを使用すると、グリッドがアクティブなままでもプラグインを動的にアップグレードできます。これにより、グリッド・コンテナー・プロセスを再始動せずにアプリケーションの更新が可能になります。eXtreme Scale プラグインの作成法の詳細については、システム API とプラグインを参照してください。

手順

1. ObjectGrid 記述子 XML ファイルを使用して OSGi 対応プラグインを構成します。

2. Eclipse Equinox OSGi フレームワークを使用して eXtreme Scale コンテナー・サーバーを開始します。
3. xscmd ユーティリティーを使用して eXtreme Scale プラグインの OSGi サービスを管理します。
4. OSGi Blueprint を使用してサーバーを構成します。

ObjectGrid 記述子 XML ファイルを使用した OSGi 対応プラグインの構成

このタスクでは、既存の OSGi サービスを記述子 XML ファイルに追加して、WebSphere eXtreme Scale コンテナーが OSGi 対応プラグインを正しく認識し、ロードできるようにします。

始める前に

プラグインを構成するときは、必ず以下を実行してください。

- パッケージを作成し、OSGi デプロイメントのために動的プラグインを使用可能にする。
- プラグインを表す OSGi サービスの名前を用意しておく。

このタスクについて

プラグインをラップする OSGi サービスの作成は完了しています。次は、これらのサービスを objectgrid.xml ファイル内に定義して、eXtreme Scale コンテナーがプラグインを正常にロードおよび構成できるようにする必要があります。

手順

1. グリッド固有のプラグイン (TransactionCallback など) は、objectGrid エレメントの下に指定しなければなりません。 objectgrid.xml ファイルの次の例を参照してください。

```
<?xml version="1.0" encoding="UTF-8"?>

<objectGridConfig xmlns:xsi="http://www.w3.org/2001/XMLSchema-instance"
    xsi:schemaLocation="http://ibm.com/ws/objectgrid/config ../objectGrid.xsd"
    xmlns="http://ibm.com/ws/objectgrid/config">

    <objectGrids>
        <objectGrid name="MyGrid" txTimeout="60">
            <bean id="myTranCallback" osgiService="myTranCallbackFactory"/>
            ...
        </objectGrid>
        ...
    </objectGrids>
    ...
/objectGridConfig>
```

   重要: osgiService 属性値は、myTranCallback PluginServiceFactory でサービスが定義された blueprint XML ファイルに指定されている ref 属性値と一致しなければなりません。

2. マップ固有のプラグイン (例えば、ローダー、シリアライザーなど) は、backingMapPluginCollections エレメント内に指定し、backingMap エレメントから参照されなければなりません。 objectgrid.xml ファイルの次の例を参照してください。

```
<?xml version="1.0" encoding="UTF-8"?>

objectGridConfig xmlns:xsi="http://www.w3.org/2001/XMLSchema-instance"
    xsi:schemaLocation="http://ibm.com/ws/objectgrid/config ../objectGrid.xsd"
    xmlns="http://ibm.com/ws/objectgrid/config">
    <objectGrids>
        <objectGrid name="MyGrid" txTimeout="60">
            <backingMap name="MyMap1" lockStrategy="PESSIMISTIC"
                copyMode="COPY_TO_BYTES" nullValuesSupported="false"
                pluginCollectionRef="myPluginCollectionRef1"/>
            <backingMap name="MyMap2" lockStrategy="PESSIMISTIC"
                copyMode="COPY_TO_BYTES" nullValuesSupported="false"
                pluginCollectionRef="myPluginCollectionRef2"/>
            ...
        </objectGrid>
        ...
    </objectGrids>
    ...
    <backingMapPluginCollections>
        <backingMapPluginCollection id="myPluginCollectionRef1">
            <bean id="MapSerializerPlugin" osgiService="mySerializerFactory"/>
        </backingMapPluginCollection>
        <backingMapPluginCollection id="myPluginCollectionRef2">
            <bean id="MapSerializerPlugin" osgiService="myOtherSerializerFactory"/>
            <bean id="Loader" osgiService="myLoader"/>
        </backingMapPluginCollection>
        ...
    </backingMapPluginCollections>
    ...
</objectGridConfig>
```

タスクの結果

この例の objectgrid.xml ファイルは、MyMap1 と MyMap2 の 2 つのマップを持つ MyGrid というグリッドを作成するよう eXtreme Scale に指示します。 MyMap1 マップは、OSGi サービス mySerializerFactory によってラップされるシリアライザーを使用します。MyMap2 マップは、OSGi サービス myOtherSerializerFactory のシリアライザーと、OSGi サービス myLoader のローダーを使用します。

Eclipse Equinox OSGi フレームワークを使用した eXtreme Scale サーバーの始動

WebSphere eXtreme Scale コンテナー・サーバーは、いくつかの方法を使用して、Eclipse Equinox OSGi フレームワークの中で始動することができます。

始める前に

eXtreme Scale コンテナーを開始する前に、次のタスクを完了していなければなりません。

1. WebSphere eXtreme Scale サーバー・バンドルが Eclipse Equinox にインストールされていなければなりません。
2. アプリケーションは OSGi バンドルとしてパッケージされていなければなりません。
3. WebSphere eXtreme Scale プラグインがある場合は、OSGi バンドルとしてパッケージされていなければなりません。これらのプラグインは、アプリケーションと同じバンドルにバンドルすることも、別々のバンドルとしてバンドルすることもできます。

このタスクについて

このタスクでは、Eclipse Equinox OSGi フレームワークの中で eXtreme Scale コンテナー・サーバーを始動する方法を説明します。Eclipse Equinox 実装を使用してコンテナー・サーバーを始動するには、次のいずれかの方法を使用することができます。

- OSGi Blueprint サービス

  OSGi バンドルの中に、すべての構成およびメタデータを含めることができます。次の図を参考にして、この方法の Eclipse Equinox プロセスを理解してください。

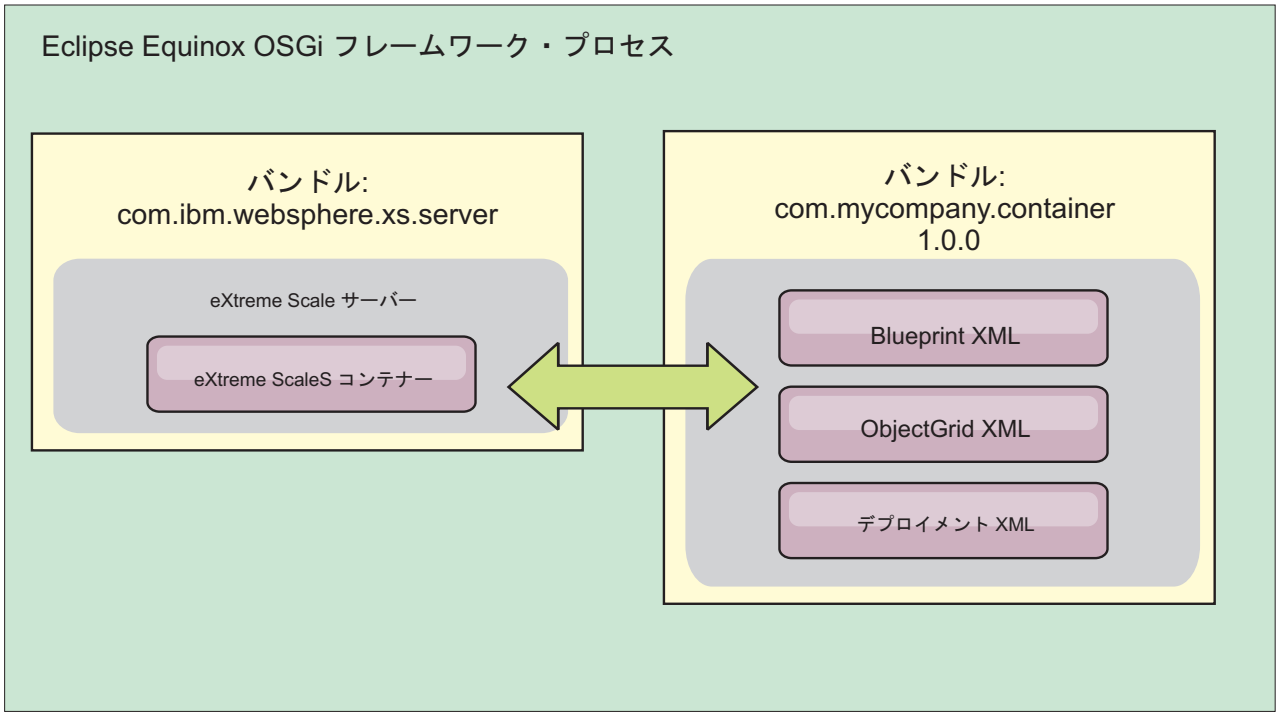


図 56. OSGi バンドルにすべての構成およびメタデータを含めるための Eclipse Equinox プロセス

- OSGi Configuration Admin サービス

  OSGi バンドルの外部で構成およびメタデータを指定できます。次の図を参考にして、この方法の Eclipse Equinox プロセスを理解してください。

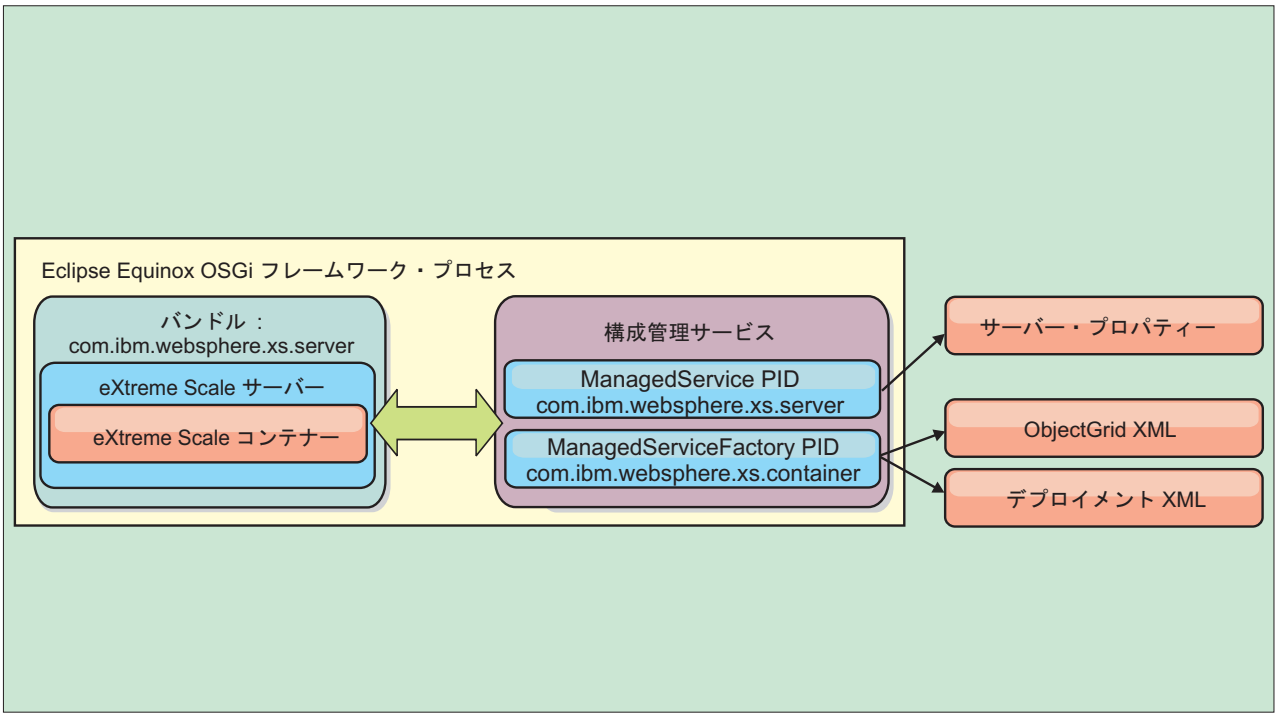


図 57. OSGi バンドルの外部で構成およびメタデータを指定するための Eclipse Equinox プロセス

- プログラムによる構成

  カスタマイズされた構成ソリューションをサポートします。

いずれの場合にも、eXtreme Scale サーバーの singleton が構成され、1 つ以上のコンテナーが構成されます。

eXtreme Scale サーバー・バンドル `objectgrid.jar` には、OSGi フレームワークの中で eXtreme Scale グリッド・コンテナーを開始して実行するのに必要なすべてのライブラリーが含まれます。サーバー・ランタイム環境は、OSGi サービス・マネージャーを使用して、ユーザー提供のプラグインおよびデータ・オブジェクトと対話します。

重要: eXtreme Scale サーバー・バンドルが始動され、eXtreme Scale サーバーが初期化された後に、eXtreme Scale サーバーを再始動することはできません。 eXtreme Scale サーバーを再始動するには、Eclipse Equinox プロセスを再開する必要があります。

Spring 名前空間に対する eXtreme Scale サポートを使用して、Blueprint XML ファイルで eXtreme Scale コンテナー・サーバーを構成できます。サーバーおよびコンテナーの XML エレメントが Blueprint XML ファイルに追加されると、eXtreme Scale 名前空間ハンドラーが、バンドルの始動時に Blueprint XML ファイルで定義されるパラメーターを使用して、コンテナー・サーバーを自動的に始動します。バンドルが停止されると、ハンドルはコンテナーを停止します。

Blueprint XML で eXtreme Scale コンテナー・サーバーを構成するには、次のステップを実行します。

手順

- OSGi Blueprint を使用して、eXtreme Scale コンテナー・サーバーを始動します。
  1. コンテナー・バンドルを作成します。
  2. コンテナー・バンドルを Eclipse Equinox OSGi フレームワークにインストールします。 205 ページの『OSGi 対応プラグインのインストールと開始』を参照してください。
  3. コンテナー・バンドルを開始します。
- OSGi Configuration Admin を使用して、eXtreme Scale コンテナー・サーバーを始動します。
  1. Config Admin を使用して、サーバーおよびコンテナーを構成します。
  2. eXtreme Scale サーバー・バンドルが開始されるか、Config Admin によって永続 ID が作成されると、サーバーおよびコンテナーは自動的に始動します。
- ServerFactory API を使用して、eXtreme Scale コンテナー・サーバーを始動します。サーバー API 資料を参照してください。
  1. OSGi バンドル・アクティベーター・クラスを作成し、eXtreme Scale ServerFactory API を使用してサーバーを始動します。

xscmd ユーティリティーによる OSGi 対応サービスの管理

xscmd ユーティリティーを使用して、各コンテナーが使用しているサービスとサービス・ランキングを表示したり、バンドルの新しいバージョンを使用するようランタイム環境を更新したりするなど、管理者用タスクを実行できます。

このタスクについて

Eclipse Equinox OSGi フレームワークでは、同一バンドルの複数バージョンをインストールでき、それらのバンドルを実行時に更新できます。WebSphere eXtreme Scale は、多数の OSGi フレームワーク・インスタンス内でコンテナー・サーバーを実行する分散環境です。

手動で OSGi フレームワークにバンドルをコピーしたり、インストールしたり、それらのバンドルを開始したりする作業は管理者の担当です。eXtreme Scale には、ObjectGrid 記述子 XML ファイル内で eXtreme Scale プラグインとして識別されたサービスを追跡する OSGi ServiceTrackerCustomizer が組み込まれています。 **xscmd** ユーティリティーを使用すると、プラグインのどのバージョンが使用されているか、どのようなバージョンが使用可能かを確認でき、バンドル・アップグレードも実行できます。

eXtreme Scale はサービス・ランキング番号を使用して、各サービスのバージョンを識別します。参照先が同じサービスが 2 つ以上ロードされると、eXtreme Scale は、ランキングが最も高いサービスを自動的に使用します。

手順

- **osgiCurrent** コマンドを実行して、各 eXtreme Scale サーバーが正しいサービス・ランキングのプラグインを使用していることを確認します。

  eXtreme Scale はランキングが最も高いサービス参照を自動的に選択するため、複数ランキングのプラグイン・サービスが設定されたデータ・グリッドが開始される可能性があります。

コマンドがランキングの不一致を検出するか、サービスを検出できない場合は、ゼロ以外のエラー・レベルが設定されます。コマンドが正常に完了した場合、エラー・レベルは 0 に設定されます。

次の例は、4 つのサーバーの同一グリッドに 2 つのプラグインがインストールされている場合の **osgiCurrent** コマンドの出力を示します。 loaderPlugin プラグインはランキング 1 を使用し、txCallbackPlugin プラグインはランキング 2 を使用しています。

```
OSGi Service Name Current Ranking ObjectGrid Name MapSet Name Server Name
----------------- --------------- --------------- ----------- -----------
loaderPlugin      1               MyGrid          MapSetA     server1
loaderPlugin      1               MyGrid          MapSetA     server2
loaderPlugin      1               MyGrid          MapSetA     server3
loaderPlugin      1               MyGrid          MapSetA     server4
txCallbackPlugin  2               MyGrid          MapSetA     server1
txCallbackPlugin  2               MyGrid          MapSetA     server2
txCallbackPlugin  2               MyGrid          MapSetA     server3
txCallbackPlugin  2               MyGrid          MapSetA     server4
```

次の例は、新しいランキングの loaderPlugin が設定された server2 を開始した場合の **osgiCurrent** コマンドの出力を示します。

```
OSGi Service Name Current Ranking ObjectGrid Name MapSet Name Server Name
----------------- --------------- --------------- ----------- -----------
loaderPlugin      1               MyGrid          MapSetA     server1
loaderPlugin      2               MyGrid          MapSetA     server2
loaderPlugin      1               MyGrid          MapSetA     server3
loaderPlugin      1               MyGrid          MapSetA     server4
txCallbackPlugin  2               MyGrid          MapSetA     server1
txCallbackPlugin  2               MyGrid          MapSetA     server2
txCallbackPlugin  2               MyGrid          MapSetA     server3
txCallbackPlugin  2               MyGrid          MapSetA     server4
```

- **osgiAll** コマンドを実行して、各 eXtreme Scale コンテナー・サーバーで正しいプラグイン・サービスが開始されたことを確認します。

  ObjectGrid 構成が参照しているサービスを含んでいるバンドルが開始すると、eXtreme Scale ランタイム環境はプラグインを自動的に追跡します。ただし、すぐには使用しません。**osgiAll** コマンドは、各サーバーで使用可能なプラグインを表示します。

  パラメーターを指定せずに実行すると、すべてのグリッドおよびサーバーのすべてのサービスが表示されます。追加のフィルター (**-serviceName** <service_name> フィルターなど) を指定して、単一サービスやデータ・グリッドのサブセットなどに出力を制限できます。

  次の例は、2 つのサーバーで 2 つのプラグインが開始された場合の **osgiAll** コマンドの出力を示します。loaderPlugin はランキング 1 と 2 の両方が開始済みで、txCallbackPlugin はランキング 1 が開始済みです。出力の最後にあるサマリー・メッセージから、両方のサーバーが同じサービス・ランキングを認識していることを確認できます。

```
Server: server1
   OSGi Service Name   Available Rankings
   -----------------   ------------------
   loaderPlugin        1, 2
   txCallbackPlugin    1

Server: server2
```

```
   OSGi Service Name   Available Rankings
   -----------------   ------------------
   loaderPlugin        1, 2
   txCallbackPlugin    1

Summary - All servers have the same service rankings.
```

次の例は、server1 でランキング 1 の loaderPlugin を含んでいるバンドルが停止した場合の **osgiAll** コマンドの出力を示します。出力の下部にあるサマリー・メッセージから、現在 server1 にはランキング 1 の loaderPlugin がないことを確認できます。

```
Server: server1
   OSGi Service Name   Available Rankings
   -----------------   ------------------
   loaderPlugin        2
   txCallbackPlugin    1

Server: server2
   OSGi Service Name   Available Rankings
   -----------------   ------------------
   loaderPlugin        1, 2
   txCallbackPlugin    1

Summary - The following servers are missing service rankings:
   Server  OSGi Service Name Missing Rankings
   ------  ----------------- ----------------
   server1 loaderPlugin      1
```

次の例は、**-sn** 引数を使用してサービス名が指定されたときに、そのサービスが存在しない場合の出力を示します。

```
Server: server2
   OSGi Service Name Available Rankings
   ----------------- ------------------
   invalidPlugin     No service found

Server: server1
   OSGi Service Name Available Rankings
   ----------------- ------------------
   invalidPlugin     No service found

Summary - All servers have the same service rankings.
```

- **osgiCheck** コマンドを実行して、プラグイン・サービスとランキングのセットをチェックし、それらが使用可能かどうか確認します。

  osgiCheck コマンドは、-serviceRankings <service name>;<ranking>[,<serviceName>;<ranking>] の形式で、サービス・ランキングのセットを 1 つ以上受け入れます。

  ランキングがすべて使用可能な場合、メソッドはエラー・レベル 0 を返します。1 つ以上のランキングが使用不可の場合は、ゼロ以外のエラー・レベルと指定されたサービス・ランキングを含んでいないすべてのサーバーの表が設定されます。追加フィルターを使用して、eXtreme Scale ドメイン内の使用可能なサーバーのサブセットにサービス・チェックを制限できます。

  例えば、指定されたランキングまたはサービスがない場合は、次のメッセージが表示されます。

```
Server  OSGi Service Unavailable Rankings
------  ------------ --------------------
server1 loaderPlugin 3
server2 loaderPlugin 3
```

- **osgiUpdate** コマンドを実行して、単一 ObjectGrid および MapSet 内のすべてのサーバーを対象に 1 つ以上のプラグインのランキングを 1 回の操作で更新します。

  コマンドは、-serviceRankings <service name>;<ranking>[,<serviceName>;<ranking>] -g <grid name> -ms <mapset name> の形式で、サービス・ランキングのセットを 1 つ以上受け入れます。

  このコマンドでは、以下の操作を実行できます。
  - 指定されたサービスが各サーバーで更新のために使用可能なことを確認する。
  - StateManager インターフェースを使用してグリッドの状態をオフラインに変更する。詳しくは、ObjectGrid の可用性の管理を参照してください。このプロセスはグリッドを静止し、実行中のトランザクションがすべて完了するまで待機し、新規トランザクションを開始できないようにします。またこのプロセスは、トランザクション・アクティビティーを停止するように ObjectGridLifecycleListener プラグインと BackingMapLifecycleListener プラグインにシグナル通知します。イベント・リスナー・プラグインの詳細については、イベント・リスナーの指定のためのプラグインを参照してください。
  - 新しいサービス・バージョンを使用するように OSGi フレームワーク内で実行中の各 eXtreme Scale コンテナーを更新する。
  - グリッドの状態をオンラインに変更し、トランザクションを継続できるようにする。

  更新処理はべき等のプロセスであるため、クライアントがいずれかのタスクを完了できない場合は、操作がロールバックされることになります。クライアントがロールバックを実行できない場合またはクライアントが更新処理中に中断された場合は、同じコマンドを再実行でき、クライアントは適切なステップから続行します。

  クライアントがプロセスを続行できず、別のクライアントからプロセスが再始動された場合、-force オプションを使用すると、クライアントが更新を実行できるようになります。**osgiUpdate** コマンドは、複数のクライアントが同一マップ・セットを同時に更新しないようにします。**osgiUpdate** コマンドの詳細については、**xscmd** による eXtreme Scale プラグインの OSGi サービスの更新を参照してください。

OSGi Blueprint でのサーバーの構成

OSGi Blueprint XML ファイルを使用して WebSphere eXtreme Scale コンテナー・サーバーを構成できます。この方法によりパッケージ化が簡単になるほか、自己完結型サーバー・バンドルの作成が可能になります。

始める前に

このトピックは、以下のタスクが完了していることを前提としています。

- Eclipse Gemini または Apache Aries の Blueprint コンテナーを使用する Eclipse Equinox OSGi フレームワークをインストールし、開始していること。
- eXtreme Scale サーバー・バンドルをインストールし、開始していること。
- eXtreme Scale 動的プラグイン・バンドルの作成が完了していること。
- eXtreme Scale ObjectGrid 記述子 XML ファイルとデプロイメント・ポリシー XML ファイルの作成が完了していること。

このタスクについて

このタスクでは、Blueprint XML ファイルを使用して eXtreme Scale サーバーとコンテナーを構成する方法を説明します。この手順の結果として、コンテナー・バンドルが作成されます。コンテナー・バンドルが開始されると、eXtreme Scale サーバー・バンドルはそのバンドルを追跡し、サーバー XML を解析し、サーバーとコンテナーを開始します。

コンテナー・バンドルは、動的プラグイン更新が必要でない場合またはプラグインが動的更新をサポートしない場合に、オプションでアプリケーションおよび eXtreme Scale プラグインと結合できます。

手順

1. objectgrid 名前空間が組み込まれた Blueprint XML ファイルを作成します。ファイルには任意の名前を付けることができます。ただし、blueprint 名前空間を含める必要があります。

```
<?xml version="1.0" encoding="UTF-8"?>

<blueprint xmlns="http://www.osgi.org/xmlns/blueprint/v1.0.0"
           xmlns:objectgrid="http://www.ibm.com/schema/objectgrid"
           xsi:schemaLocation="http://www.ibm.com/schema/objectgrid
                http://www.ibm.com/schema/objectgrid/objectgrid.xsd">
...
</blueprint>
```

2. 適切なサーバー・プロパティーを使用して eXtreme Scale サーバーの XML 定義を追加します。 すべての使用可能な構成プロパティーの詳細については、Spring 記述子 XML ファイルを参照してください。次の XML 定義の例を参照してください。

```
objectgrid:server
    id="xsServer"
   tracespec="ObjectGridOSGi=all=enabled"
   tracefile="logs/osgi/wxsserver/trace.log"
    jmxport="1199"
   listenerPort="2909">
    <objectgrid:catalog host="catserver1.mycompany.com" port="2809" />
    <objectgrid:catalog host="catserver2.mycompany.com" port="2809" />
</objectgrid:server>
```

3. サーバー定義への参照と、バンドルに組み込まれている ObjectGrid 記述子 XML ファイルと ObjectGrid デプロイメント XML ファイルを使用して eXtreme Scale コンテナーの XML 定義を追加します。例えば、次のようにします。

```
<objectgrid:container id="container"
    objectgridxml="/META-INF/objectGrid.xml"
    deploymentxml="/META-INF/objectGridDeployment.xml"
    server="xsServer" />
```

4. Blueprint XML ファイルをコンテナー・バンドル内に保管します。 Blueprint XML は OSGI-INF/blueprint ディレクトリー内に保管し、Blueprint コンテナーが検出されるようにしなければなりません。

   Blueprint XML を他のディレクトリーに保管するには、Bundle-Blueprint マニフェスト・ヘッダーを指定する必要があります。例えば、次のようにします。

   ```
   Bundle-Blueprint: OSGI-INF/blueprint.xml
   ```

5. ファイルを単一バンドル JAR ファイルにパッケージ化します。 次のバンドル・ディレクトリー階層の例を参照してください。

   ```
   MyBundle.jar
       /META-INF/manifest.mf
       /META-INF/objectGrid.xml
       /META-INF/objectGridDeployment.xml
       /OSGI-INF/blueprint/blueprint.xml
   ```

タスクの結果

これで eXtreme Scale コンテナー・バンドルが作成されたので、Eclipse Equinox にインストールできます。コンテナー・バンドルが開始されると、eXtreme Scale サーバー・バンドル内の eXtreme Scale サーバー・ランタイム環境が、バンドルに定義されているパラメーターを使用して singleton eXtreme Scale サーバーを自動的に開始し、コンテナー・サーバーも開始します。バンドルは停止したり開始したりでき、それを受けてコンテナーも停止または開始されます。サーバーは singleton であり、バンドルがはじめて開始されたときは停止しません。

第 4 章 サンプル

WebSphere eXtreme Scale のさまざまなチュートリアル、例、およびサンプルが使用できます。

例

以下のトピックでは、WebSphere eXtreme Scale の主要なフィーチャーについて説明しています。

- DataGrid API の例
- ローカル・デプロイメントの構成

コミュニティー・サンプル

以下のサンプルは、WebSphere eXtreme Scale の各種フィーチャーを示すために、さまざまな環境でのこの製品の使用法を説明しています。

- **非同期サービス・フレームワーク** - 非同期サービス・フレームワークは、メッセージの非同期処理のためのスケーラブルなフォールト・トレラント処理ファブリックを提供します。サンプルのダウンロード方法も含め、詳しくは、サンプル・ギャラリー: 非同期サービス・フレームワークのサンプル (英語のみ) を参照してください。
- **クライアント認証セキュリティー** - このサンプルは、サーバーがいずれかのグリッド・アクセスを提供するまで有効な資格情報を提供するように、クライアントを必要とする認証を構成する方法について説明します。サンプルのダウンロード方法も含め、詳しくは、サンプル・ギャラリー: クライアント認証セキュリティー (英語のみ) を参照してください。
- **動的マップの作成** - このサンプルは、グリッドが既に初期化された後にマップを作成する方法を示します。eXtreme Scale 7.0 以降の場合、テンプレートを使用してマップを取得できます。サンプルのダウンロード方法も含め、詳しくは、サンプル・ギャラリー: グリッド初期化後の動的マップの作成 (英語のみ) を参照してください。
- **マルチマスター・レプリカ生成** - マルチマスター・レプリカ生成入門サンプルは、マルチマスター (AP) レプリカ生成のクイック入門として提供されています。サンプルのダウンロード方法も含め、詳しくは、サンプル・ギャラリー: マルチマスター・レプリカ生成サンプル (英語のみ) を参照してください。
- **Entity Manager API を使用した照会** - このサンプルは、分散され、区画に分割されたマップで EntityManager API を使用して照会を使用する方法を示します。サンプルのダウンロード方法も含め、詳しくは、サンプル・ギャラリー: Entity Manager API を使用した、区画に分割されたグリッド内での照会の実行 (英語のみ) を参照してください。
- **ReduceGridAgent 実装を使用した並行照会** - データ・グリッド API を使用して、グリッド内のすべての区画に対して照会を実行する方法を示します。サンプ

ルのダウンロード方法も含め、詳しくは、サンプル・ギャラリー: ReduceGridAgent を使用した照会の並行実行 (英語のみ) を参照してください。

チュートリアルおよび例を伴う項目

表 9. フィーチャーごとの使用可能項目

項目	フィーチャー
グリッド対応アプリケーションの作成	ObjectMap API、EntityManager API、照会、エージェント、Java SE および EE、統計、区画化、管理/操作、Eclipse
スケーラブルなグリッド・スタイルのコンピューティングおよびデータ処理	EntityManager API、エージェント
スケーラブルで、回復力のある代替の高性能データベースの作成	ObjectMap API、レプリカ生成、区画化、管理/操作、Eclipse
WebSphereeXtreme Scale 用の xsadmin の強化	管理
Redbook: User's Guide	すべてのトピック

無料試用

WebSphere eXtreme Scale を使用し始める前に、無料試用版をダウンロードしてください。拡張機能を使用してデータ・キャッシング概念を拡張することにより、革新的で高性能なアプリケーションを開発することができます。

試用版のダウンロード

eXtreme Scale 試用版のダウンロードから、WebSphere eXtreme Scale の無料試用版をダウンロードすることができます。

試用版の eXtreme Scale のダウンロードと unzip が完了したら、`gettingstarted` ディレクトリーに移動して `GETTINGSTARTED_README.txt` をお読みください。 このチュートリアルには eXtreme Scale の使用を開始するための概要、複数のサーバーにデータ・グリッドを作成する方法、グリッド内のデータを保管または取得する簡単なアプリケーションの実行方法などが説明されています。実稼働環境に eXtreme Scale をデプロイする前に、使用するサーバーの数、各サーバーのストレージ容量、同期または非同期レプリカ生成など、いくつかのオプションについて検討する必要があります。

サンプル・プロパティー・ファイル

サーバー・プロパティー・ファイルには、カタログ・サーバーとコンテナー・サーバーを実行するための設定が含まれています。サーバー・プロパティー・ファイルは、スタンドアロンに対して、あるいは WebSphere Application Server 構成に対して 1 つ指定することができます。クライアント・プロパティー・ファイルには、クライアントの設定が含まれます。

wxs_install_root¥`properties` ディレクトリーにある以下のサンプル・プロパティー・ファイルを使用して、プロパティー・ファイルを作成できます。

- `sampleServer.properties`

- sampleClient.properties

サンプル: xsadmin ユーティリティー

xsadmin ユーティリティーを使用すれば、WebSphere eXtreme Scale トポロジーに関するテキスト情報をフォーマットして表示することができます。このサンプル・ユーティリティーは現在のデプロイメント・データの解析とディスカバリーの方法を提供するもので、カスタム・ユーティリティーの作成基盤として使用することができます。

始める前に

- 7.1.1+ **xsadmin** ユーティリティーは、デプロイメントのカスタム・ユーティリティーをどのように作成できるかを示すサンプルとして提供されています。 **xscmd** ユーティリティーは、環境のモニターおよび管理のための、サポートされるユーティリティーとして提供されています。詳しくは、**xscmd** ユーティリティーによる管理を参照してください。
- **xsadmin** ユーティリティーを使用して結果を表示するには、データ・グリッド・トポロジーを作成しておく必要があります。カタログ・サーバーおよびコンテナー・サーバーは、始動済みでなければなりません。詳しくは、スタンドアロン・サーバーの始動と停止を参照してください。
- 製品と一緒にインストールされたランタイム環境を使用するように *JAVA_HOME* 環境変数が設定されていることを確認してください。製品の試用版を使用している場合は、*JAVA_HOME* 環境変数を設定する必要があります。

このタスクについて

xsadmin サンプル・ユーティリティーは、Managed Bean (MBeans) の実装を使用します。このサンプルのモニター・アプリケーションは、統合されたモニター機能をすぐに使用可能にします。このモニター機能は、com.ibm.websphere.objectgrid.management パッケージ内のインターフェースを使用して拡張できます。 **xsadmin** サンプル・アプリケーションのソース・コードは、スタンドアロン・インストールの場合は *wxs_home*/samples/xsadmin.jar ファイルで確認でき、WebSphere Application Server インストールの場合は *wxs_home*/xsadmin.jar ファイルで確認できます。

xsadmin サンプル・ユーティリティーを使用して、マップの内容など、データ・グリッドの現在のレイアウトおよび特定の状態を表示できます。この例では、このタスクで使用されるデータ・グリッドのレイアウトは、*MapSetA* マップ・セットに属する 1 つの *MapA* マップを含む単一の *ObjectGridA* データ・グリッドで構成されています。この例では、データ・グリッド内のすべてのアクティブ・コンテナーを表示したり、*MapA* マップのマップ・サイズに関するフィルタリング済み指標を印刷したりする方法について説明します。使用できるコマンド・オプションをすべて知りたい場合は、引数なしか、または **-help** オプションを付けて **xsadmin** ユーティリティーを実行してください。

手順

1. bin ディレクトリーに移動します。

   ```
   cd wxs_home/bin
   ```

2. **xsadmin** ユーティリティーを実行します。
   - オンライン・ヘルプを表示するには、次のコマンドを実行します。

     UNIX

     ```
     xsadmin.sh
     ```

     Windows

     ```
     xsadmin.bat
     ```

     このユーティリティーが機能するためには、リストされているオプションの中の 1 つだけを渡す必要があります。**-g**、**-m** いずれのオプションも指定されていない場合は、**xsadmin** ユーティリティーはトポロジー内のすべてのグリッドについて情報を印刷します。
   - すべてのサーバーの統計を使用可能にするには、以下のコマンドを実行します。

     UNIX

     ```
     xsadmin.sh -g ObjectGridA -setstatsspec ALL=enabled
     ```

     Windows

     ```
     xsadmin.bat -g ObjectGridA -setstatsspec ALL=enabled
     ```
   - ある特定のグリッドについてすべてのオンライン・コンテナーを表示するには、次のコマンドを実行します。

     UNIX

     ```
     xsadmin.sh -g ObjectGridA -m MapSetA -containers
     ```

     Windows

     ```
     xsadmin.bat -g ObjectGridA -m MapSetA -containers
     ```

     すべてのコンテナー情報が表示されます。出力の例を次に示します。

     ```
     Connecting to Catalog service at localhost:1099


     *** Show all online containers for grid - ObjectGridA & mapset - MapSetA

     Host: 192.168.0.186
     Container: server1_C-0, Server:server1, Zone:DefaultZone
     Partition Shard Type
             0 Primary

     Num containers matching = 1
     Total known containers = 1
     Total known hosts = 1
     ```

重要: Transport Layer Security/Secure Sockets Layer (TLS/SSL) が使用可能であるときにこの情報を取得するには、JMX サービス・ポートを設定してカタログ・サーバーおよびコンテナー・サーバーを始動する必要があります。JMX サービス・ポートを設定するには、**startOgServer** スクリプトで **-JMXServicePort** オプションを使用するか、ServerProperties インターフェースで setJMXServicePort メソッドを呼び出すことができます。

- カタログ・サービスに接続して MapA に関する情報を表示するには、次のコマンドを実行します。

  UNIX

  ```
  xsadmin.sh -g ObjectGridA -m MapSetA -mapsizes -fm MapA
  ```

  Windows

  ```
  xsadmin.bat -g ObjectGridA -m MapSetA -mapsizes -fm MapA
  ```

  指定したマップのサイズが表示されます。出力の例を次に示します。

  ```
  Connecting to Catalog service at localhost:1099

  ****Displaying Results for Grid - ObjectGridA, MapSet - MapSetA*****


  *** Listing Maps for server1 ***
   Map Name Partition Map Size Used Bytes (B) Shard Type
   MapA     0         0        0              Primary
  ```

- 特定の JMX ポートを使用してカタログ・サービスに接続して MapA マップに関する情報を表示するには、以下のコマンドを実行します。 UNIX

  ```
  xsadmin.sh -g ObjectGridA -m MapSetA -mapsizes -fm MapA
    -ch CatalogMachine -p 6645
  ```

  Windows

  ```
  xsadmin.bat -g ObjectGridA -m MapSetA -mapsizes -fm MapA
    -ch CatalogMachine -p 6645
  ```

  xsadmin サンプル・ユーティリティーは、カタログ・サーバーを実行している MBean サーバーに接続します。カタログ・サーバーは、スタンドアロン・プロセスまたは WebSphere Application Server プロセスとして実行できます。あるいは、カスタム・アプリケーション・プロセス内に組み込むこともできます。カタログ・サービス・ホスト名を指定するには **-ch** オプションを、カタログ・サービス・ネーミング・ポートを指定するには **-p** オプションを、それぞれ使用します。

  指定したマップのサイズが表示されます。出力の例を次に示します。

  ```
  Connecting to Catalog service at CatalogMachine:6645

  *****Displaying Results for Grid - ObjectGridA, MapSet - MapSetA*****


  *** Listing Maps for server1 ***
  Map Name: MapA  Partition #: 0  Map Size: 0  Shard Type: Primary
  Server Total: 0
  ```

- WebSphere Application Server プロセスでホストされるカタログ・サービスに接続するには、以下のステップを実行します。

  WebSphere Application Server プロセスまたはプロセスのクラスターがホストするカタログ・サービスに接続する際には、**-dmgr** オプションが必要です。localhost ではない場合ホスト名を指定するには **-ch** オプションを、カタログ・サービス・ブートストラップ・ポートをオーバーライドするには **-p** オプションを、それぞれ使用します。後者の場合は、BOOTSTRAP_ADDRESS プロセスが使用されます。 **-p** オプションは、BOOTSTRAP_ADDRESS がデフォルトの 9809 に設定されていない場合にのみ必要となります。

  注: WebSphere Application Server プロセスがホストするカタログ・サービスに接続する場合は、WebSphere eXtreme Scale のスタンドアロン・バージョンは使用できません。 *was_root*/bin ディレクトリーに含まれている **xsadmin** スクリプトを使用してください。このスクリプトは、WebSphere eXtreme Scale を WebSphere Application Server または WebSphere Application Server Network Deployment にインストールすると使用可能になります。

  a. WebSphere Application Server bin ディレクトリーに移動します。

     ```
     cd was_root/bin
     ```

  b. 次のコマンドを使用して、**xsadmin** ユーティリティーを起動します。

     UNIX

     ```
     xsadmin.sh -g ObjectGridA -m MapSetA -mapsizes -fm MapA -dmgr
     ```

     Windows

     ```
     xsadmin.bat -g ObjectGridA -m MapSetA -mapsizes -fm MapA -dmgr
     ```

  指定したマップのサイズが表示されます。

  ```
  Connecting to Catalog service at localhost:9809

  ****Displaying Results for Grid - ObjectGridA, MapSet - MapSetA*****


  *** Listing Maps for server1 ***
  Map Name: MapA  Partition #: 0  Map Size: 0  Shard Type: Primary
  Server Total: 0
  ```

- 構成内の構成されている配置とランタイムの配置を表示するには、次のいずれかのコマンドを実行します。

  ```
  xsadmin -placementStatus
  xsadmin -placementStatus -g myOG -m myMapSet
  xsadmin -placementStatus -m myMapSet
  xsadmin -placementStatus -g myOG
  ```

  配置情報については、全体の構成、1 つのデータ・グリッド、1 つのマップ・セット、またはデータ・グリッドとマップ・セットの組み合わせを表示するそれぞれのコマンドを使用できます。出力の例を次に示します。

  ```
  **********Printing Placement Status for Grid - Grid, MapSet - mapSet**************

  <objectGrid name="Grid" mapSetName="mapSet">
    <configuration>
      <attribute name="placementStrategy" value="FIXED_PARTITIONS"/>
      <attribute name="numInitialContainers" value="3"/>
      <attribute name="minSyncReplicas" value="0"/>
      <attribute name="developmentMode" value="true"/>
    </configuration>
  ```

```
  <runtime>
    <attribute name="numContainers" value="3"/>
    <attribute name="numMachines" value="1"/>
    <attribute name="numOutstandingWorkItems" value="0"/>
  </runtime>
</objectGrid>
```

xsadmin ユーティリティーの構成プロファイルの作成

xsadmin ユーティリティーでよく指定するパラメーターをプロパティー・ファイルに保存できます。その結果として、**xsadmin** ユーティリティーの呼び出しを短くできます。

始める前に

カタログ・サーバーとコンテナー・サーバーを少なくとも 1 つずつ含む WebSphere eXtreme Scale の基本デプロイメントを作成します。詳しくは、**startOgServer** スクリプトを参照してください。

このタスクについて

xsadmin ユーティリティーの構成プロファイルに設定できるプロパティーのリストについては、『**xsadmin** ユーティリティー・リファレンス』を参照してください。プロパティー・ファイルと (コマンド行引数として) 対応するパラメーターの両方を指定した場合、コマンド行引数がプロパティー・ファイルの値をオーバーライドします。

手順

1. 構成プロファイルのプロパティー・ファイルを作成します。 このプロパティー・ファイルには、すべての **xsadmin** コマンド呼び出しで使用するグローバル・プロパティーを含めてください。

   プロパティー・ファイルを任意の名前で保存します。例えば、ファイルは次のパスに配置できます: `/opt/ibm/WebSphere/wxs71/ObjectGrid/security/<my.properties>`。

   `<my.properties>` を実際のファイルの名前に置き換えてください。 例えば、次のプロパティーをファイルに設定できます。

   - `XSADMIN_TRUST_TYPE=jks`
   - `XSADMIN_TRUST_PATH=/opt/ibm/WebSphere/wxs71/ObjectGrid/bin/security/key.jks`
   - `XSADMIN_USERNAME=ogadmin`
2. 作成したプロパティー・ファイルを指定して xsadmin ユーティリティーを実行します。 **-profile** パラメーターを使用して、プロパティー・ファイルの場所を指示します。**-v** パラメーターを使用して、詳細な出力を表示することもできます。

   ```
   ./xsadmin.sh -l -v -password xsadmin -ssl -trustPass ogpass -profile
   /opt/ibm/WebSphere/wxs71/ObjectGrid/security/<my.properties>
   ```

xsadmin ユーティリティー・リファレンス

xsadmin ユーティリティーには、コマンド行引数を使用する方式、またはプロパティー・ファイルを使用する方式で、引数を渡すことができます。

xsadmin 引数

xsadmin ユーティリティー用のプロパティー・ファイルは、、バージョン 7.1 フィックス 1 以上で定義できます。プロパティー・ファイルを作成すると、ユーザー名などの使用頻度の高い引数の一部を保存できます。プロパティー・ファイルに追加できるプロパティーは、次の表のとおりです。プロパティー・ファイル内のプロパティーとそれに相当するコマンド行引数を両方指定した場合、コマンド行引数値がプロパティー・ファイルの値に優先します。

xsadmin ユーティリティー用のプロパティー・ファイルの定義方法について詳しくは、225 ページの『**xsadmin** ユーティリティーの構成プロファイルの作成』を参照してください。

表 10. **xsadmin** *ユーティリティーの引数*

コマンド行引数	プロパティー・ファイル内で相当するプロパティー名	説明と有効な値
-bp	n/a	リスナー・ポートを指示します。 **デフォルト:**2809
-ch	n/a	カタログ・サーバーの JMX ホスト名を指示します。 **デフォルト:**localhost
-clear	n/a	指定されたマップをクリアします。 **使用できるフィルター:** -fm
-containers	n/a	データ・グリッドおよびマップ・セットのそれぞれについて、コンテナー・サーバーのリストを表示します。 **使用できるフィルター:** -fnp
-continuous	n/a	データ・グリッドをモニターするためにマップ・サイズの結果を継続的に表示させたい場合は、このフラグを指定してください。 **-mapsizes** 引数を指定してこのコマンドを実行すると、20 秒ごとにマップ・サイズが表示されます。
-coregroups	n/a	カタログ・サーバーのすべてのコア・グループを表示します。この引数は、拡張診断に使用されます。
-dismissLink <*catalog_service_domain*>	n/a	2 つのカタログ・サービス・ドメイン間のリンクを削除します。前に **-establishLink** 引数を使用して接続した外部カタログ・サービス・ドメインの名前を指定します。
-dmgr	n/a	WebSphere Application Server がホストするカタログ・サービスへの接続かどうかを指示します。 **デフォルト:**false
-empties	n/a	出力に空のコンテナーを表示する場合、このフラグを指定します。
-establishLink <*foreign_domain_name*> <*host1:port1,host2:port2...*>	n/a	カタログ・サービス・ドメインを外部カタログ・サービス・ドメインに接続します。次の形式を使用してください: -establishLink <*foreign_domain_name*> <*host1:port1,host2:port2...*>。ここで、*foreign_domain_name* は外部カタログ・サービス・ドメインの名前で、*host1:port1,host2:port2...* は、このカタログ・サービス・ドメイン内で実行中のカタログ・サーバー・ホスト名とオブジェクト・リクエスト・ブローカー (ORB) ポートのコンマ区切りリストです。
-fc	n/a	このコンテナーのみにフィルタリングします。 WebSphere Application Server Network Deployment 環境でコンテナー・サーバーをフィルタリングする場合は、次の形式を使用します。 <*cell_name*>/<*node_name*>/<*serverName_containerSuffix*> **この形式を指定できる引数: -mapsizes**、**-teardown**、**-revisions**、**-getTraceSpec**、**-setTraceSpec**、**-getStatsSpec**、**-setStatsSpec**

表 10. **xsadmin** *ユーティリティーの引数 (続き)*

コマンド行引数	プロパティー・ファイル内で相当するプロパティー名	説明と有効な値
-fh	n/a	このホストのみにフィルタリングします。 **この形式を指定できる引数: -mapsizes、-teardown、-revisions、-getTraceSpec、-setTraceSpec、-getStatsSpec、-setStatsSpec、-routetable**
-fm	n/a	このマップのみにフィルタリングします。 **この形式を指定できる引数: -clear、-mapsizes**
-fnp	n/a	プライマリー断片を持たないサーバーをフィルタリングします。 **この形式を指定できる引数: -containers**
-fp	n/a	この区画のみにフィルタリングします。 **この形式を指定できる引数: -mapsizes、-teardown、-revisions、-getTraceSpec、-setTraceSpec、-getStatsSpec、-setStatsSpec、-routetable**
-fs	n/a	このサーバーのみにフィルタリングします。 WebSphere Application Server Network Deployment 環境でアプリケーション・サーバーをフィルタリングする場合は、次の形式を使用します。 *<cell_name>/<node_name>/<server_name>* **この形式を指定できる引数: -mapsizes、-teardown、-revisions、-getTraceSpec、-setTraceSpec、-getStatsSpec、-setStatsSpec**
-fst	n/a	この断片タイプのみにフィルタリングします。プライマリー断片のみの場合は P を、非同期レプリカ断片のみの場合は A を、同期レプリカ断片のみの場合は S を指定します。 **この形式を指定できる引数: -mapsizes、-teardown、-revisions、-getTraceSpec、-setTraceSpec、-getStatsSpec、-setStatsSpec**
-fz	n/a	このゾーンのみにフィルタリングします。 **この形式を指定できる引数: -mapsizes、-teardown、-revisions、-getTraceSpec、-setTraceSpec、-getStatsSpec、-setStatsSpec、-routetable**
-force	n/a	いずれのプリエンプティブなプロンプトも無効にして、コマンド内のアクションを強制します。この引数はバッチ・コマンドを実行するときに便利です。
-g	n/a	ObjectGrid 名を指定します。
-getstatsspec	n/a	現在の統計仕様を表示します。統計仕様は、**-setstatsspec** 引数で設定できます。 **使用できるフィルター:** `-fst -fc -fz -fs -fh -fp`
-getTraceSpec	n/a	現在のトレース仕様を表示します。トレース仕様は、**-settracespec** 引数で設定できます。 **使用できるフィルター:** `-fst -fc -fz -fs -fh -fp`
-h	n/a	**xsadmin** ユーティリティーのヘルプを表示します。ヘルプには引数のリストも含まれます。
-hosts	n/a	構成内のすべてのホストを表示します。
-jmxUrl	XSADMIN_JMX_URL	JMX API コネクター・サーバーのアドレスを次の形式で指定します: `service:jmx:`*`protocol`*`:`*`sap`*。 *protocol* および *sap* 変数の定義は、次のとおりです。 *protocol* コネクター・サーバーに接続するために使用するトランスポート・プロトコルを指定します。 *sap* コネクター・サーバーが検出されるアドレスを指定します。 JMX サービス URL の形式の詳細については、Class JMXServiceURL (Java 2 Platform SE 5.0) を参照してください。
-l	n/a	すべての既知のデータ・グリッドとマップ・セットを表示します。
-m	n/a	マップ・セットの名前を指定します。
-mapsizes	n/a	断片間でキーの分布が一様か確認するため、カタログ・サーバー上の各マップのサイズを表示します。 **使用できるフィルター:** `-fm -fst -fc -fz -fs -fh -fp`

表 10. **xsadmin** ユーティリティーの引数 (続き)

コマンド行引数	プロパティー・ファイル内で相当するプロパティー名	説明と有効な値
-mbeanservers	n/a	すべての MBean サーバー・エンドポイントのリストを表示します。
-overridequorum	n/a	データ・センターの障害発生時のシナリオでコンテナー・サーバー・イベントが無視されないよう、クォーラム設定を無効にします。
-password	XSADMIN_PASSWORD	**xsadmin** ユーティリティーにログインするためのパスワードを指定します。パスワードを保護する必要がある場合、プロパティー・ファイル内にパスワードは指定しないでください。
-p	n/a	カタログ・サーバー・ホストの JMX ポートを示します。 **デフォルト:** WebSphere Application Server ホストの場合は 1099 または 9809、スタンドアロン構成の場合は 1099。
-placementStatus	n/a	構成内の構成されている配置とランタイムの配置を表示します。出力の有効範囲はデータ・グリッドとマップ・セットの組み合わせにしたり、構成全体にしたりできます。 • 構成全体: `-placementStatus` • 特定のデータ・グリッドの場合: `-placementStatus -g` *`my_grid`* • 特定のマップ・セットの場合: `-placementStatus -m` *`my_mapset`* • 特定のデータ・グリッドとマップ・セットの場合: `-placementStatus -g` *`my_grid`* `-m` *`my_mapset`*
-primaries	n/a	プライマリー断片のリストを表示します。
-profile	n/a	**xsadmin** ユーティリティー用のプロパティー・ファイルへの完全修飾パスを指定します。
-quorumstatus	n/a	カタログ・サービスのクォーラムの状況を表示します。
-releaseShard <*container_server_name*> <*objectgrid_name*> <*map_set_name*> <*partition_name*>	n/a	**-reserveShard** 引数と一緒に使用されます。**-releaseShard** 引数は、断片が予約されて配置された後に呼び出されます。**-releaseShard** 引数は、ContainerMBean.release() メソッドを呼び出します。
-reserved	n/a	**-reserveShard** 引数によって予約された断片のみを表示するために **-containers** 引数と一緒に使用されます。
-reserveShard <*container_server_name*> <*objectgrid_name*> <*map_set_name*> <*partition_name*>	n/a	プライマリー断片を指定されたコンテナー・サーバーに移動します。この引数によって、ContainerMBean.reserve() メソッドが呼び出されます。
-resumeBalancing <*objectgrid_name*> <*map_set_name*>	n/a	要求のバランスを取ること、および指定された ObjectGrid とマップ・セットに対する将来の再バランシング試行を許可することを試みます。
-revisions	n/a	カタログ・サービス・ドメインの改訂 ID を表示します。これには、特定の断片ごとの各データ・グリッド、区画番号、区画タイプ (プライマリーまたはレプリカ)、カタログ・サービス・ドメイン、存続期間 ID、およびデータ改訂の回数が含まれます。この引数を使用して、非同期レプリカまたはリンクされたドメインがキャッチアップされたかどうかを判別できます。この引数は、ObjectGridMBean.getKnownRevisions() メソッドを呼び出します。 **使用できるフィルター:** `-fst -fc -fz -fs -fh -fp`

表 10. **xsadmin** *ユーティリティーの引数 (続き)*

コマンド行引数	プロパティー・ファイル内で相当するプロパティー名	説明と有効な値
-routetable	n/a	クライアント・サーバーの観点から、データ・グリッドの現在の状態を表示します。経路テーブルは、ObjectGrid クライアント・サーバーがデータ・グリッドとの通信に使用する情報です。接続問題または TargetNotAvailable 例外を特定する場合に、診断援助として経路テーブルを使用します。 **必須引数:** スタンドアロン環境では、カタログ・サーバー・ホストのブートストラップ・リスナー・ポートおよび JMX ポートにデフォルト値を使用していない場合、この引数と一緒に **-bp** パラメーターおよび **-p** パラメーターを指定しなければなりません。 **使用できるフィルター:** -fz -fh -fp
-settracespec *<trace_string>*	n/a	実行時にサーバーのトレースを有効にします。次の例を参照してください。 -setTraceSpec "ObjectGridReplication=all=enabled" 指定できるトレース・ストリングについて詳しくは、トレースの収集とトレース・オプションを参照してください。 **使用できるフィルター:** -fst -fc -fz -fs -fh -fp
-swapShardWithPrimary *<container_server_name>* *<objectgrid_name>* *<map_set_name>* *<partition_name>*	n/a	指定されたコンテナー・サーバーからの指定されたレプリカ断片を、プライマリー断片と交換します。このコマンドを実行することで、必要なときに手動でプライマリー断片をバランシングできます。
-setstatsspec *<stats_spec>*	n/a	統計の収集を有効にします。この引数は、DynamicServerMBean.setStatsSpec メソッドと DynamicServerMBean.getStatsSpec メソッドを呼び出します。モニターできる統計モジュールについて詳しくは、クラス StatsSpec を参照してください。 **使用できるフィルター:** -fm -fst -fc -fz -fs -fh -fp
-suspendBalancing *<objectgrid_name>* *<map_set_name>*	n/a	指定された ObjectGrid およびマップ・セットの将来のバランシング試行をさせません。
-ssl	n/a	Secure Sockets Layer (SSL) が有効なことを示します。
-teardown	n/a	カタログおよびコンテナー・サーバーのリストまたはカタログおよびコンテナー・サーバーのグループを停止します。 **使用できるフィルター:** -fst -fc -fz -fs -fh -fp **サーバーのリストを指定する場合の形式:** *server_name_1*,*server_name_2 ...* **特定ゾーン内のすべてのサーバーを停止する場合、-fz 引数を組み込みます。** –fz *<zone_name>* **特定ホスト上のすべてのサーバーを停止する場合、-fh 引数を組み込みます。** –fh *<host_name>*
-triggerPlacement	n/a	デプロイメント XML ファイルに構成されている **numInitialContainers** の値を無視して、断片配置の実行を強制します。サーバーのメンテナンスを実行中、この引数を使用すると、**numInitialContainers** の値が構成値より小さい場合でも断片配置の実行を続行できるようになります。
-trustPass	XSADMIN_TRUST_PASS	指定したトラストストアのパスワードを指定します。
-trustPath	XSADMIN_TRUST_PATH	トラストストア・ファイルへのパスを指定します。 例: etc/test/security/server.public
-trustType	XSADMIN_TRUST_TYPE	トラストストアのタイプを指定します。 有効値: JKS、JCEK、PKCS12 など。
-unassigned	n/a	データ・グリッド上に配置できない断片のリストを表示します。配置サービスに配置を妨げる制約があると、断片は配置できません。

表 10. **xsadmin** *ユーティリティーの引数 (続き)*

コマンド行引数	プロパティー・ファイル内で相当するプロパティー名	説明と有効な値
-username	XSADMIN_USERNAME	**xsadmin** ユーティリティーにログインするためのユーザー名を指定します。
-v	n/a	詳細コマンド行アクションを使用可能にします。環境変数、プロパティー・ファイル、またはその両方を使用して特定のコマンド行引数を指定し、それらの値を表示する必要がある場合、このフラグを使用します。 詳しくは、『**xsadmin** ユーティリティーの詳細オプション』を参照してください。
-xml	n/a	PlacementServiceMBean.listObjectGridPlacement() メソッドからのフィルタリングされていない出力を表示します。他の **xsadmin** 引数は、このメソッドの出力をフィルタリングし、データをさらに消費可能な形式に編成します。

xsadmin ユーティリティーの詳細オプション

問題のトラブルシューティングに、**xsadmin** 詳細オプションを使用できます。xsadmin -v コマンドを実行すると、構成済みのすべてのパラメーターがリストされます。詳細オプションは、コマンド行引数、プロパティー・ファイル引数、および環境指定の引数を含めて、すべてのスコープ内のすべての値を表示します。複数のスコープを使用して同じプロパティーを指定した場合、Effective arguments セクションには、環境で使用される設定が含まれます。

詳細オプションの例

xsadmin コマンド引数:

次のテキストは、プロパティー値を指定して次のコマンドを実行した後、コマンド行から詳細オプションを指定したときの出力例です。

```
./xsadmin -l -v -username xsadmin -password xsadmin -ssl -trustPass ogpass
-profile /opt/ibm/WebSphere/wxs71/ObjectGrid/security/my.properties
```

プロパティー・ファイル引数:

/opt/ibm/WebSphere/wxs71/ObjectGrid/security/my.properties プロパティー・ファイルの内容は次のとおりです。

```
XSADMIN_TRUST_PASS=ogpass
XSADMIN_TRUST_TYPE=jks
XSADMIN_TRUST_PATH=/opt/ibm/WebSphere/wxs71/ObjectGrid/bin/security/key.jks
XSADMIN_USERNAME=ogadmin
XSADMIN_PASSWORD=ogpass
```

コマンド結果:

上記の **xsadmin** コマンドからの次の出力の中で、***太字イタリック体*** のテキストは、コマンド行とプロパティー・ファイルの両方で指定されたプロパティーと値を示します。Effective command line arguments セクションで、コマンド行で指定された引数が、プロパティー・ファイル内の値をオーバーライドすることを確認できます。

```
Command line specified arguments
*********************************
XSADMIN_USERNAME=xsadmin
XSADMIN_PASSWORD=xsadmin
XSADMIN_TRUST_PATH=<unspecified>
XSADMIN_TRUST_TYPE=<unspecified>
XSADMIN_TRUST_PASS=ogpass
XSADMIN_PROFILE=/opt/ibm/WebSphere/wxs71/ObjectGrid/security/my.properties
```

```
XSADMIN_JMX_URL=<unspecified>
*********************************
Properties file specified arguments
*********************************
XSADMIN_USERNAME=ogadmin
XSADMIN_PASSWORD=ogpass
XSADMIN_TRUST_PATH=/opt/ibm/WebSphere/wxs71/ObjectGrid/bin/security/key.jks
XSADMIN_TRUST_TYPE=jks
XSADMIN_TRUST_PASS=ogproppass
XSADMIN_JMX_URL=<unspecified>
*********************************
Environment-specified arguments
*********************************
XSADMIN_USERNAME=<unspecified>
XSADMIN_PASSWORD=<unspecified>
XSADMIN_TRUST_PATH=<unspecified>
XSADMIN_TRUST_TYPE=<unspecified>
XSADMIN_TRUST_PASS=<unspecified>
XSADMIN_JMX_URL=<unspecified>
*********************************
Effective arguments
*********************************
XSADMIN_USERNAME=xsadmin
XSADMIN_PASSWORD=xsadmin
XSADMIN_TRUST_PATH=/opt/ibm/WebSphere/wxs71/ObjectGrid/bin/security/key.jks
XSADMIN_TRUST_TYPE=jks
XSADMIN_TRUST_PASS=ogpass
XSADMIN_PROFILE=/opt/ibm/WebSphere/wxs71/ObjectGrid/security/my.properties
XSADMIN_JMX_URL=<unspecified>
SSL authentication enabled: true
*********************************
Connecting to Catalog service at localhost:1099
*** Show all 'objectGrid:mapset' names
Grid Name  MapSet Name
accounting defaultMapSet
```

重要: XSADMIN_PROFILE プロパティーは、詳細出力で表示されますが、プロパティー・ファイルに指定できる有効なキーではありません。詳細出力内でのこのプロパティーの値は、**-profile** コマンド行引数に示されるように、使用されているプロパティー値を示します。

詳細オプションを指定しない場合の出力

同じコマンドで、詳細オプションを有効にしていない場合の出力例を次に示します。

```
./xsadmin -l -username xsadmin -password xsadmin -ssl -trustPass ogpass
-profile /opt/ibm/WebSphere/wxs71/ObjectGrid/security/my.properties

Connecting to Catalog service at localhost:1099
*** Show all 'objectGrid:mapset' names
Grid Name  MapSet Name
accounting defaultMapSet
```

特記事項

本書に記載の製品、プログラム、またはサービスが日本においては提供されていない場合があります。日本で利用可能な製品、プログラム、またはサービスについては、日本 IBM の営業担当員にお尋ねください。本書で IBM 製品、プログラム、またはサービスに言及していても、その IBM 製品、プログラム、またはサービスのみが使用可能であることを意味するものではありません。IBM 製品、プログラムまたはサービスに代えて、IBM の知的所有権を侵害することのない機能的に同等のプログラムまたは製品を使用することができます。ただし、IBM によって明示的に指定されたものを除き、他社の製品と組み合わせた場合の動作の評価と検証はお客様の責任で行っていただきます。

IBM は、本書に記載されている内容に関して特許権 (特許出願中のものを含む) を保有している場合があります。本書の提供は、お客様にこれらの特許権について実施権を許諾することを意味するものではありません。実施権についてのお問い合わせは、書面にて下記宛先にお送りください。

〒242-8502
神奈川県大和市下鶴間1623番14号
日本アイ・ビー・エム株式会社
法務・知的財産
知的財産権ライセンス渉外

本プログラムのライセンス保持者で、(i) 独自に作成したプログラムとその他のプログラム (本プログラムを含む) との間での情報交換、および (ii) 交換された情報の相互利用を可能にすることを目的として、本プログラムに関する情報を必要とする方は、下記に連絡してください。

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

本プログラムに関する上記の情報は、適切な使用条件の下で使用することができますが、有償の場合もあります。

商標

IBM、IBM ロゴおよび ibm.com は、世界の多くの国で登録された International Business Machines Corp. の商標です。他の製品名およびサービス名等は、それぞれ IBM または各社の商標である場合があります。現時点での IBM の商標リストについては、http://www.ibm.com/legal/copytrade.shtml をご覧ください。

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

Java およびすべての Java 関連の商標およびロゴは、Oracle やその関連会社の米国およびその他の国における商標または登録商標です。

LINUX は、Linus Torvalds の米国およびその他の国における商標です。

Microsoft、Windows、Windows NT および Windows ロゴは、Microsoft Corporation の米国およびその他の国における商標です。

UNIX は、The Open Group の米国およびその他の国における登録商標です。

索引

日本語, 数字, 英字, 特殊文字の順に配列されています。なお, 濁音と半濁音は清音と同等に扱われています。

[ア行]

[カ行]

[サ行]

[タ行]

[ハ行]

[マ行]

[ヤ行]

[ラ行]

A

E

H

J

O

R

X

Printed in Japan

日本アイ・ビー・エム株式会社
〒103-8510 東京都中央区日本橋箱崎町19-21