WebSphere eXtreme Scale バージョン 7.0
管理ガイド

WebSphere eXtreme Scale 管理ガイド

WebSphere eXtreme Scale バージョン 7.0
管理ガイド

WebSphere eXtreme Scale 管理ガイド

本書は、WebSphere eXtreme Scale のバージョン 7、リリース 0、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 Version 7.0
Administration Guide
WebSphere eXtreme Scale Administration Guide

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 2009.11

目次

図

表

管理ガイド 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。

- *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。
- *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。

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 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。

本書の対象者

本書は、主にシステム管理者、セキュリティー管理者、およびシステム・オペレーターの方々を対象としています。

本書の構成

本書には、以下の主要なトピックに関する情報が入っています。

- **第 1 章**には、入門に関する情報が含まれています。
- **第 2 章**には、アプリケーション・デプロイメントの計画に関する情報が含まれています。
- **第 3 章**には、キャパシティー・プランニングに関する情報が含まれています。
- **第 4 章**には、製品のインストールに関する情報が含まれています。
- **第 5 章**には、z/OS プラットフォームのカスタマイズに関する情報が含まれています。
- **第 6 章**には、製品の構成に関する情報が含まれています。
- **第 7 章**には、環境の管理に関する情報が含まれています。
- **第 8 章**には、デプロイメント環境のモニターに関する情報が含まれています。
- **第 9 章**には、デプロイメント環境のセキュアに関する情報が含まれています。
- **第 10 章**には、環境のトラブルシューティングに関する情報が含まれています。

- **第 11 章**には、製品の用語集が含まれています。

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。

ご意見の送付方法

文書チームにご連絡ください。必要な情報が見つかりましたか? それは正確で完全な情報でしたか? 本書に関するご意見は、電子メールで wasdoc@us.ibm.com までお寄せください。

第 1 章 WebSphere eXtreme Scale 入門

WebSphere eXtreme Scale をスタンドアロン環境にインストールすると、以下の手順でメモリー内のデータ・グリッドとしてのその機能を簡単に導入できます。

スタンドアロンでインストールした WebSphere eXtreme Scale に組み込まれているサンプルを使用すると、インストールした製品を検証し、単純な eXtreme Scale グリッドおよびクライアントの使用方法を実際に確かめることができます。この入門サンプルは *installRoot*/ObjectGrid/gettingstarted ディレクトリーにあります。

この入門サンプルは、eXtreme Scale の機能および基本的な操作を紹介するものです。このサンプルは、シェル・スクリプトおよびバッチ・スクリプトからなります。これらは、ほんの少しカスタマイズするだけで単純なグリッドを開始できるよう設計されています。 さらに、この基本的なグリッドに対して単純な作成、読み取り、更新、および削除 (CRUD) 機能を実行するためのクライアント・プログラムが、ソースを含めて提供されています。

スクリプトとその機能

このサンプルには、以下の 4 つのスクリプトがあります。

env.sh|bat: このスクリプトは、他のスクリプトから呼び出されて、必要な環境変数を設定します。通常は、このスクリプトを変更する必要はありません。

- UNIX Linux `./env.sh`
- Windows `env.bat`

runcat.sh|bat: このスクリプトは、ローカル・システム上で eXtreme Scale カタログ・サービス・プロセスを開始します。

- UNIX Linux `./runcat.sh`
- Windows `runcat.bat`

runcontainer.sh|bat: このスクリプトは、コンテナー・サーバー・プロセスを開始します。指定した固有のサーバー名を使用してこのスクリプトを複数回実行すれば、いくつでもコンテナーを開始できます。それらのインスタンスが一緒に、グリッド内の区画化された冗長な情報をホストすることができます。

- UNIX Linux `./runcontainer.sh` *`unique_server_name`*
- Windows `runcontainer.bat` *`unique_server_name`*

runclient.sh|bat: このスクリプトは、単純な CRUD クライアントを実行し、指定された操作を開始します。

- UNIX Linux `./runclient.sh` *`command value1 value2`*
- Windows `runclient.sh` *`command value1 value2`*

command には、以下のいずれかのオプションを使用します。

- *value2* を、キー *value1* を持つグリッドに挿入するには、`i` と指定します。
- *value1* のキーのオブジェクトを *value2* に更新するには、`u` と指定します。
- *value1* のキーのオブジェクトを削除するには、`d` と指定します。
- *value1* のキーのオブジェクトを検索して表示するには、`g` を指定します。

注: *installRoot*`/ObjectGrid/ gettingstarted/src/Client.java` ファイルは、カタログ・サーバーへの接続、ObjectGrid インスタンスの取得、および ObjectMap API の使用をどのように行うのかを示すクライアント・プログラムです。

基本的な手順

次の手順で最初のグリッドを開始し、グリッドと対話するクライアントを実行します。

1. 端末セッションまたはコマンド行ウィンドウを開きます。
2. 次のコマンドを使用して、`gettingstarted` ディレクトリーに移動します。

   ```
   cd installRoot/ObjectGrid/gettingstarted
   ```

   installRoot の部分は、eXtreme Scale インストール・ルート・ディレクトリーへのパス、または、抽出した eXtreme Scale trial *installRoot* のルート・ファイル・パスに置き換えてください。
3. JAVA_HOME 環境変数を設定またはエクスポートして、有効な JDK または JRE バージョン 1.5 以降のインストール・ディレクトリーを指すようにします。

   UNIX Linux export JAVA_HOME=*Java_home_directory*

   Windows set JAVA_HOME=*Java_home_directory*
4. 次のスクリプトを実行して、ローカル・ホストでカタログ・サービス・プロセスを開始します。

   - UNIX Linux `./runcat.sh`
   - Windows `runcat.bat`

   カタログ・サービス・プロセスは、現行の端末ウィンドウで実行されます。
5. 別の端末セッションまたはコマンド行ウィンドウを開き、次のコマンドを実行して、コンテナー・サーバー・インスタンスを開始します。

   - UNIX Linux `./runcontainer.sh server0`
   - Windows `runcontainer.bat server0`

   コンテナー・サーバーは、現行の端末ウィンドウで実行されます。複製をサポートするためにさらに多くのコンテナー・サーバー・インスタンスを開始したい場合は、ステップ 5 と 6 を繰り返すことができます。
6. クライアント・コマンドを実行するため、別の端末セッションまたはコマンド行ウィンドウを開きます。

- グリッドへのデータの追加:
  - UNIX Linux `./runclient.sh i key1 helloWorld`
  - Windows `runclient.bat i key1 helloWorld`
- 値を検索して表示:
  - UNIX Linux `./runclient.sh g key1`
  - Windows `runclient.bat g key1`
- 値の更新:
  - UNIX Linux `./runclient.sh u key1 goodbyeWorld`
  - Windows `runclient.bat u key1 goodbyeWorld`
- 値の削除:
  - UNIX Linux `./runclient.sh d key1`
  - Windows `runclient.bat d key1`

7. <ctrl+c> を使用して、カタログ・サービス・プロセスおよびコンテナー・サーバーをそれぞれのウィンドウで停止します。

ObjectGrid の定義

サンプルでは、コンテナー・サーバーを開始するため、 *installRoot*/ObjectGrid/gettingstarted/xml ディレクトリーにある objectgrid.xml ファイルと deployment.xml ファイルが使用されます。objectgrid.xml ファイルは ObjectGrid 記述子 XML ファイルであり、deployment.xml ファイルは ObjectGrid デプロイメント・ポリシー記述子 XML ファイルです。両方のファイルが一緒になって、分散 ObjectGrid トポロジーが定義されます。

ObjectGrid 記述子 XML ファイル

ObjectGrid 記述子 XML ファイルは、アプリケーションによって使用される ObjectGrid の構造を定義するのに使用されます。このファイルには、BackingMap 構成のリストが含まれます。 これらの BackingMap はキャッシュ・データ用の実際のデータ・ストレージです。以下の例は、objectgrid.xml ファイルのサンプルです。ファイルの最初の数行には、各 ObjectGrid XML ファイルの必須ヘッダーが含まれています。このサンプル・ファイルは、Map1 と Map2 という BackingMap がある、Grid ObjectGrid を定義しています。

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

    <objectGrids>
        <objectGrid name="Grid">
            <backingMap name="Map1" />
            <backingMap name="Map2" />
        </objectGrid>
    </objectGrids>

</objectGridConfig>
```

デプロイメント・ポリシー記述子 XML ファイル

デプロイメント・ポリシー記述子 XML ファイルは、始動時に ObjectGrid コンテナー・サーバーに渡されます。デプロイメント・ポリシーは ObjectGrid XML ファイルと一緒に使用する必要があり、一緒に使用される ObjectGrid XML と互換でなければなりません。デプロイメント・ポリシー内の各 objectgridDeployment 要素ごとに、対応する 1 つの ObjectGrid 要素が ObjectGrid XML 内に必要です。objectgridDeployment 要素内に定義された backingMap 要素は、ObjectGrid XML 内にある backingMap と整合していなければなりません。すべての backingMap は、1 つの mapSet 内のみで参照する必要があります。

デプロイメント・ポリシー記述子 XML ファイルは、対応する ObjectGrid XML である `objectgrid.xml` ファイルと対で使用されることを想定しています。以下の例では、`deployment.xml` ファイルの最初の数行には、各デプロイメント・ポリシー XML ファイルの必須ヘッダーが含まれています。このファイルは、`objectgrid.xml` ファイル内に定義された Grid ObjectGrid の objectgridDeployment 要素を定義しています。 Grid ObjectGrid 内に定義された Map1 と Map2 の両 BackingMap は、mapSet mapSet に含まれていて、そこでは numberOfPartitions、 minSyncReplicas、および maxSyncReplicas 属性が構成されています。

```
<deploymentPolicy xmlns:xsi="http://www.w3.org/2001/XMLSchema-instance"
 xsi:schemaLocation="http://ibm.com/ws/objectgrid/deploymentPolicy ../deploymentPolicy.xsd"
 xmlns="http://ibm.com/ws/objectgrid/deploymentPolicy">

    <objectgridDeployment objectgridName="Grid">
        <mapSet name="mapSet" numberOfPartitions="13" minSyncReplicas="0" maxSyncReplicas="1" >
            <map ref="Map1"/>
            <map ref="Map2"/>
        </mapSet>
    </objectgridDeployment>

</deploymentPolicy>
```

mapSet エレメントの numberOfPartitions 属性は、mapSet の区画の数を指定します。これはオプションの属性であり、デフォルトは 1 です。この数は、グリッドに予想される容量に適した値である必要があります。

mapSet の minSyncReplicas 属性は、mapSet 内の各区画の同期複製の最小数を指定します。これはオプションの属性であり、デフォルトは 0 です。この同期複製の最小数をドメインがサポートできるまでは、プライマリーおよび複製は配置されません。minSyncReplicas 値をサポートするには、minSyncReplicas の値よりも 1 つだけ多いコンテナーが必要です。同期複製の数が minSyncReplicas の値よりも小さくなると、その区画に対しては書き込みトランザクションを行えなくなります。

mapSet の maxSyncReplicas 属性は、mapSet 内の各区画の同期複製の最大数を指定します。これはオプションの属性であり、デフォルトは 0 です。ある特定の区画でこの同期複製数にドメインが達すると、それ以降は、他の同期複製がその区画に対して配置されることはありません。まだ maxSyncReplicas 値を満たしていない場合には、この ObjectGrid をサポートできるコンテナーを追加すると、同期複製の数を増やすことができます。上のサンプルでは maxSyncReplicas は 1 に設定されていますが、これは、ドメインが最大 1 つの同期複製を置くことを意味しています。複数のコンテナー・サーバー・インスタンスを開始する場合、それらのコンテナー・サーバー・インスタンスの 1 つに、同期複製が 1 つだけ置かれます。

ObjectGrid の使用

installRoot/ObjectGrid/gettingstarted/src/ ディレクトリーにある Client.java ファイルは、カタログ・サーバーへの接続、ObjectGrid インスタンスの取得、および ObjectMap API の使用をどのように行うのかを示すクライアント・プログラムです。

クライアント・アプリケーションの観点から、WebSphere eXtreme Scale の使用は以下のステップに分解できます。

1. ClientClusterContext インスタンスを取得することによって、カタログ・サービスに接続する。
2. クライアント ObjectGrid インスタンスを取得する。
3. Session インスタンスを取得する。
4. ObjectMap インスタンスを取得する。
5. ObjectMap メソッドを使用する。

1. ClientClusterContext インスタンスを取得することによって、カタログ・サービスに接続する

カタログ・サーバーに接続するには、ObjectGridManager API の connect メソッドを使用します。このサンプルで使用されている connect メソッドが必要とするのは、hostname:port という形式のカタログ・サーバー・エンドポイントのみです (例えば localhost:2809)。カタログ・サーバーへの接続が成功すれば、connect メソッドは ClientClusterContext インスタンスを戻します。この ClientClusterContext インスタンスは、ObjectGridManager API から ObjectGrid を取得するのに必要です。以下のコード・スニペットは、カタログ・サーバーへの接続方法と ClientClusterContext インスタンスの取得方法を示します。

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

2. ObjectGrid インスタンスを取得する

ObjectGrid インスタンスを取得するには、ObjectGridManager API の getObjectGrid メソッドを使用します。 getObjectGrid メソッドは、ClientClusterContext インスタンスと、 ObjectGrid インスタンスの名前との両方を必要とします。
ClientClusterContext インスタンスは、カタログ・サーバーへの接続中に取得されます。ObjectGrid の名前は、objectgrid.xml ファイルに指定されている「Grid」です。以下のコード・スニペットは、ObjectGridManager API の getObjectGrid メソッドを呼び出すことによって ObjectGrid を取得する方法を示します。

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

3. Session インスタンスを取得する

取得した ObjectGrid インスタンスから、Session を取得することができます。
Session インスタンスは、ObjectMap の取得とトランザクション区分の実行のために必要です。以下のコード・スニペットは、ObjectGrid API の getSession メソッドを呼び出すことによって Session を取得する方法を示します。

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

4. ObjectMap インスタンスを取得する

Session を取得した後、Session API の getMap メソッドを呼び出すことによって、Session から ObjectMap を取得することができます。ObjectMap を取得するため、マップ名を getMap メソッドにパラメーターとして渡す必要があります。以下のコード・スニペットは、Session API の getMap メソッドを呼び出すことによって ObjectMap を取得する方法を示します。

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

5. ObjectMap メソッドを使用する

ObjectMap を取得した後、ObjectMap API を使用できます。ObjectMap はトランザクション・マップであり、Session API の begin メソッドおよび commit メソッドを使用することによるトランザクション区分を必要とすることに注意してください。明示的なトランザクション区分がない場合、ObjectMap 操作は自動コミット・トランザクションで実行します。

以下のコード・スニペットは、自動コミット・トランザクションでの ObjectMap API の使用方法を示しています。

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

以下のコード・スニペットは、明示的なトランザクション区分がある場合の ObjectMap API の使用方法を示しています。

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

追加情報

このサンプルは、カタログ・サーバーおよびコンテナー・サーバーを開始する方法と、スタンドアロン環境での ObjectMap API の使用を示しています。EntityManager API を使用することもできます。

WebSphere eXtreme Scale がインストールされているか使用可能にされている WebSphere Application Server 環境では、最も一般的なシナリオは、ネットワーク接続されたトポロジーです。 ネットワーク接続トポロジーでは、カタログ・サーバーは WebSphere Application Server デプロイメント・マネージャー・プロセス内でホストされ、各 WebSphere Application Server インスタンスが 1 つの eXtreme Scale サーバーを自動的にホストします。Java™ Platform, Enterprise Edition アプリケーションは、ObjectGrid 記述子 XML ファイルと ObjectGrid デプロイメント・ポリシー記述子 XML ファイルの両方を、任意のモジュールの `META-INF` ディレクトリーに含めることのみを必要とし、ObjectGrid は自動的に使用可能になります。 その後、アプリケーションはローカルで使用可能なカタログ・サーバーに接続し、使用する ObjectGrid インスタンスを取得できます。

第 2 章 アプリケーション・デプロイメントの計画

WebSphere eXtreme Scale にアプリケーションをデプロイする前に、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、デプロイメント構成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。

eXtreme Scale のデプロイメント構成

WebSphere eXtreme Scale は、ローカルまたは分散の 2 つのタイプの環境にデプロイできます。必要な構成は、デプロイメント環境のタイプによって異なります。

ローカル環境

ローカル環境にデプロイすると、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。ローカル環境では、ObjectGrid XML ファイルまたは ObjectGrid API が必要です。

分散環境

分散環境にデプロイすると、eXtreme Scale は Java 仮想マシンのセット内で稼働します。この構成では、データの複製および区画化の使用が可能です。分散環境は、必要に応じてサーバーを追加できる動的デプロイメント・トポロジーにさらに分類できます。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。さらに eXtreme Scale メモリー内のデータ・グリッドの構成詳細を持つデプロイメント・ポリシー XML ファイルを提供する必要があります。

ハードウェアおよびソフトウェアの要件

WebSphere eXtreme Scale を使用するための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は、特に制限はありません。 eXtreme Scale のサポートに関する公式文書は、「システム要件」ページのオンラインで提供されています。

WebSphere eXtreme Scale の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に関するオペレーティング・システム別の詳しいリストについては、 システム要件ページを参照してください。

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

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。

WebSphere Application Server 要件

分散環境で稼働する eXtreme Scale のクライアントとサーバー、およびローカルのメモリー内の ObjectGrid は、WebSphere Application Server バージョン 6.0.2 以降でサポートされます。

注: 動的キャッシュ・プロバイダーを使用する場合、システムは以下の最小要件のいずれかを満たしていなければなりません。

- WebSphere Application Server バージョン 6.1.0.25 以上 + 暫定修正 PK85622
- WebSphere Application Server バージョン 7.0.0.3 以上 + 暫定修正 PK85622

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 1.4.2 以降を使用する必要があります。

パフォーマンス調整

パフォーマンスを向上させるため、オペレーティング・システムとネットワークの調整、ネットワーク・ポートの計画、WebSphere eXtreme Scale 設定に対する Java 仮想マシン (JVM) 調整、フェイルオーバーの構成、その他のチューニングに関するトピックなどを考慮します。

ネットワーク・ポートの計画

WebSphere eXtreme Scale は、分散キャッシュであり、オブジェクト・リクエスト・ブローカー (ORB) および伝送制御プロトコル (TCP) スタックを使用して Java 仮想マシンと他のマシン間で通信するためにオープン・ポートを必要とします。特にファイアウォール環境では、ポートを計画し、制御することが必要です。例えば、いくつかのポートをオープンするためにカタログ・サービスとコンテナーを使用する場合などです。

カタログ・サービス・グリッド

カタログ・サービス・グリッドは、次のものを必要とします。

1. peerPort: 高可用性 (HA) マネージャーがピア・カタログ・サーバー間で TCP スタックを介して通信するために必要です
2. clientPort: カタログ・サーバーがカタログ・サービス・データにアクセスするために必要です
3. JMXServicePort: JMX サービスが使用することになるポートを指定するために必要です
4. ORB リスナー・ポート: コンテナーおよびクライアントが ORB を介してカタログ・サービスと通信するために必要です

上記のポートを指定するには、以下の例のように、スタンドアロンでオプションを指定した startOgServer コマンドを使用します。

```
–catalogServiceEndPoints cs1:host1:clientPort:peerPort, cs2:host2:clientPort:peerPort,
cs3:host3:clientPort:peerPort -listenerPort <orbPort> -JMXServicePort <jmxPort>
```

WebSphere Application Server 環境では、上記リスト中のポートは次のように、キー catalog.services.cluster の WebSphere セル・カスタム・プロパティーに指定されます。

- cell¥node1¥server1:host1:clientPort:peerPort:listenerPort
- cell¥node2¥server2:host2:clientPort:peerPort:listenerPort

WebSphere eXtreme Scale コンテナー・サーバーも、いくつかのポートが作動することを必要とします。デフォルトでは、eXtreme Scale コンテナー・サーバーは、HA マネージャー・ポートおよび ORB リスナー・ポートを、動的ポートと共に自動的に生成します。ファイアウォール環境では、ポートを計画し、制御することはユーザーにとって有益なので、eXtreme Scale コンテナー・サーバーを開始する際のオプションが用意されています。このコンテナー・サーバーは、以下の例に示すように、startOgServer コマンド中のオプションで HAManager ポートおよび ORB リスナー・ポートを指定して開始することができます。

```
-HaManagerPort <peerPort>
-listenerPort <orbPort>
```

ポート制御の適切な計画は有益ですが、数百の Java 仮想マシンを 1 台のマシンで開始する場合は、これらのポートの計画と管理には特有の難しさがあります。ポート競合があると、サーバーの始動が失敗します。

セキュリティーが有効になっている場合、上記のリストに示されたポートに加えて、Secure Socket Layer (SSL) ポートが必要です。
`Dcom.ibm.CSI.SSLPort=<sslPort>` を -jvmArgs 引数として使用すると、SSL ポートが `<sslPort>` に設定されます。

オペレーティング・システムおよびネットワークのチューニング

ネットワークのチューニングは、接続キープアライブ設定の変更によって伝送制御プロトコル (TCP) スタックの遅延を減らすことができ、TCP バッファーの変更によってスループットを改善することができます。

オペレーティング・システム

チューニングが最も少なくてすむのが Windows® システムで、最も必要なのが Solaris システムです。以下の説明は、指定された各システムに固有のものであり、WebSphere eXtreme Scale パフォーマンスを向上させる可能性があります。ご使用の

環境でのネットワークおよびアプリケーション負荷に応じて調整を行ってください。

Windows

```
HKEY_LOCAL_MACHINE¥SYSTEM¥CurrentControlSet¥Services¥
Tcpip¥Parameters
MaxFreeTcbs = dword:00011940
MaxHashTableSize = dword:00010000
MaxUserPort = dword:0000fffe
TcpTimedWaitDelay = dword:0000001e
```

Solaris

```
ndd -set /dev/tcp tcp_time_wait_interval 60000
fndd -set /dev/tcp tcp_keepalive_interval 15000
ndd -set /dev/tcp tcp_fin_wait_2_flush_interval 67500
ndd -set /dev/tcp tcp_conn_req_max_q 16384
ndd -set /dev/tcp tcp_conn_req_max_q0 16384
ndd -set /dev/tcp tcp_xmit_hiwat 400000
ndd -set /dev/tcp tcp_recv_hiwat 400000
ndd -set /dev/tcp tcp_cwnd_max 2097152
ndd -set /dev/tcp tcp_ip_abort_interval 20000
ndd -set /dev/tcp tcp_rexmit_interval_initial 4000
ndd -set /dev/tcp tcp_rexmit_interval_max 10000
ndd -set /dev/tcp tcp_rexmit_interval_min 3000
ndd -set /dev/tcp tcp_max_buf 4194304
```

AIX®

```
/usr/sbin/no -o tcp_sendspace=65536
/usr/sbin/no -o tcp_recvspace=65536
/usr/sbin/no -o udp_sendspace=65536
/usr/sbin/no -o udp_recvspace=65536
/usr/sbin/no -o somaxconn=10000
/usr/sbin/no -o tcp_nodelayack=1
/usr/sbin/no –o tcp_keepinit=40
/usr/sbin/no –o tcp_keepintvl=10
```

LINUX

```
sysctl -w net.ipv4.tcp_timestamps=0
sysctl -w net.ipv4.tcp_tw_reuse=1
sysctl -w net.ipv4.tcp_tw_recycle=1
sysctl -w net.ipv4.tcp_fin_timeout=30
sysctl -w net.ipv4.tcp_keepalive_time=1800
sysctl -w net.ipv4.tcp_rmem="4096 87380 8388608"
sysctl -w net.ipv4.tcp_wmem="4096 87380 8388608"
sysctl -w net.ipv4.tcp_max_syn_backlog=4096
```

HP-UX

```
ndd -set /dev/tcp tcp_ip_abort_cinterval 20000
```

ORB プロパティーおよびファイル記述子の設定

チューニング考慮事項には、Object Request Broker (ORB) プロパティーおよびファイル記述子の設定などが含まれます。

ORB プロパティー

ORB は WebSphere eXtreme Scale によって TCP スタックでの通信のために使用されます。必要な `orb.properties` ファイルは、`java/jre/lib` ディレクトリーにあります。大容量オブジェクトの重い負荷に備えるため、以下の設定値を指定して ORB フラグメント化を有効にしてください。

```
com.ibm.CORBA.FragmentSize=<right size>
```

スレッド・プールの増大を防止するため、次のように設定してください。

```
com.ibm.CORBA.ThreadPool.IsGrowable=false
```

異常なシチュエーションでの過剰スレッドを回避するため、以下を設定して適切なタイムアウトを設定してください。

- com.ibm.CORBA.RequestTimeout
- com.ibm.CORBA.ConnectTimeout
- com.ibm.CORBA.FragmentTimeout

ファイル記述子

UNIX® システムおよび Linux® システムでは、プロセス当たりに許容されるオープン・ファイルの数に制限があります。オペレーティング・システムが、許容されるオープン・ファイルの数を指定します。この値が小さすぎる場合、AIX ではメモリー割り振りエラーが発生し、多すぎるオープン・ファイルはログに記録されます。

UNIX システム端末ウィンドウで、この値をデフォルトのシステム値よりも大きく設定してください。クローンを持つ大容量 SMP マシンの場合、無限に設定してください。

AIX 構成では、コマンド `ulimit -n -1` を使用して、この値を `-1` (無限) に設定してください。

Solaris 構成の場合、コマンド `ulimit -n 16384` を使用して、この値を `16384` に設定してください。

コマンド `ulimit –a` を使用すれば、現行値を表示できます。

WebSphere eXtreme Scale 用の JVM の調整

このページでは、WebSphere eXtreme Scale 用の Java 仮想マシン (JVM) 調整に関するいくつかの具体的な問題を示します。パフォーマンス上の問題がある場合は、IBM® サポートに連絡してください。このページは、追加の調整情報が使用可能になり次第更新されます。

ほとんどの場合、WebSphere eXtreme Scale は、特殊な JVM 設定をほんとどまたはまったく必要としません。WebSphere eXtreme Scale に保管されるオブジェクトが多数ある場合は、ヒープ・サイズを適切なレベルに調整して、メモリー不足の状態で実行されるのを避けます。

プラットフォーム

パフォーマンス・テストは、まず AIX (32 way)、Linux (4 way)、Windows (8 way)、および AZUL (384 way) のコンピューターで実行されました。 AZUL システムおよびハイエンド AIX のコンピューターでは、マルチスレッドのシナリオを大量にプッシュして、競合ポイントを特定して修正することができます。AZUL システムにより、テスト中にスケーリング実行の品質を大幅に低下させる可能性があるガーベッジ・コレクションも除去できます。

オブジェクト・リクエスト・ブローカー (ORB) 要件

IBM SDK には、WebSphere Application Server および WebSphere eXtreme Scale を使用してテスト済みの IBM ORB 実装が組み込まれています。サポート・プロセスを簡単にするため、IBM 提供の JVM を使用してください。他の JVM 実装では、

異なる ORB が使用されます。IBM 提供の Java 仮想マシンと共にすぐに使用可能なのは、IBM ORB のみです。WebSphere eXtreme Scale には、操作する作業 ORB が必要です。WebSphere eXtreme Scale はサード・パーティーの ORB とともに使用できますが、ORB に関する問題が特定されている場合は、ORB ベンダーのサポートに連絡する必要があります。IBM ORB 実装は、サード・パーティーの Java 仮想マシンと互換性があり、必要な場合は置換できます。

ガーベッジ・コレクション

WebSphere eXtreme Scale は、要求や応答など各トランザクション、およびログ・シーケンスに関連した一時オブジェクトを作成します。これらのオブジェクトはガーベッジ・コレクションの効率に影響するため、ガーベッジ・コレクションのチューニングは重要です。

IBM の Java 仮想マシンの場合、更新率が高いシナリオ (トランザクションの 100% がエントリーを変更する) には optavgpause コレクターを使用してください。データがほとんど更新されない (10% 以下の頻度) ようなシナリオでは、optavgpause コレクターより gencon コレクターの方がより適切に機能します。両方のコレクターを使用して実験を行い、シナリオで最も適切に機能するコレクターを確認します。パフォーマンス上の問題を確認できる場合は、詳細ガーベッジ・コレクションをオンにして実行し、ガーベッジの収集に費やされている時間の割合を確認します。調整で問題が修正されるまでガーベッジ・コレクションで時間の 80% が費やされたシナリオが発生しました。

ガーベッジ・コレクションの構成について詳しくは、IBM の Java 仮想マシンの調整を参照してください。

JVM パフォーマンス

WebSphere eXtreme Scale は、異なるバージョンの Java 2 Platform, Standard Edition (J2SE) 上で実行できます。WebSphere eXtreme Scale バージョン 6.1 は、J2SE バージョン 1.4.2 以降をサポートしています。開発者の生産性およびパフォーマンスを向上させるためには、J2SE 5 またはそれ以降を使用して、アノテーションおよび改良されたガーベッジ・コレクションを活用してください。WebSphere eXtreme Scale は、32 ビットまたは 64 ビット版の Java 仮想マシンで動作します。

WebSphere eXtreme Scale がテストされたのは、使用可能な仮想マシンの一部ですが、サポートのリストは排他的なものではありません。バージョン 1.4.2 以上の任意のバージョンで WebSphere eXtreme Scale を実行できますが、JVM に関する問題が特定されている場合は、JVM ベンダーにサポートを依頼する必要があります。可能であれば、 WebSphere Application Server がサポートするどのプラットフォーム上でも、WebSphere ランタイムの JVM を使用してください。

WebSphere eXtreme Scale が使用されるほとんどのシナリオでは、JVM の Java Platform Standard Edition 6 のほうが、Edition 5 または 1.4 よりうまく機能します。Java 2 Platform, Standard Edition Version 1.4 は、gencon コレクターを使用するシナリオの場合は特に、パフォーマンスが悪くなります。Java Platform Standard Edition 5 は良好に動作しますが、Java Platform, Standard Edition 6 のほうが優れています。

大規模なヒープ

アプリケーションにより区画ごとに大量のデータを管理する必要がある場合は、ガーベッジ・コレクションが要因になっている可能性があります。ある世代のコレクターが使用されている場合は、非常に大きなヒープ (20 GB 以上) が存在しても読み取りシナリオの大部分は正常に機能します。ただし、保有ヒープがいっぱいになった後は、コンピューターで使用可能なヒープ・サイズおよびプロセッサー数に比例して一時停止が発生します。この一時停止は、大きいヒープを持つ小さいコンピューターで大きくなる可能性があります。

WebSphere eXtreme Scale は、WebSphere Real Time Java をサポートします。WebSphere Real Time Java を一緒に使用することによって、WebSphere eXtreme Scale のトランザクション処理応答はより一貫性のある、予測可能なものになり、ガーベッジ・コレクションおよびスレッド・スケジューリングの影響は大幅に小さくなります。応答時間の標準偏差が標準 Java の 10% よりも小さくなる程度まで、影響は少なくなります。

詳しくは、 230 ページの『WebSphere Real Time の使用』を参照してください。

スレッド数

スレッド数はいくつかの要因に依存します。単一の断片が管理できるスレッド数には制限があります。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とすることができます。JVM ごとの断片数が多いほど、それぞれ追加断片を持つスレッドが増えるので、データへの並行パスが多くなります。各断片は可能な限り並行ですが、それでも並行性について制限はあります。

フェイルオーバー検出の構成

障害の起きたサーバーがないかを調べるシステム・チェックの間の時間は構成できます。この値は、ハートビート間隔と呼ばれます。

このタスクについて

フェイルオーバーの構成は、使用している環境のタイプによって異なります。スタンドアロン環境を使用している場合は、コマンド行でフェイルオーバーを構成できます。 WebSphere Application Server Network Deployment 環境を使用している場合は、WebSphere Application Server Network Deployment 管理コンソールでフェイルオーバーを構成する必要があります。

- スタンドアロン環境のフェイルオーバーを構成します。

  ハートビート間隔は、startOgServer.bat | startOgServer.sh スクリプト・ファイルに **-heartbeat** パラメーターを使用してコマンド行で構成できます。このパラメーターは以下のいずれかの値に設定します。

表 1. ハートビート間隔

値	アクション	説明
0	標準 (デフォルト)	通常、30 秒以内にフェイルオーバーが検出されます。
-1	高速	通常、5 秒以内にフェイルオーバーが検出されます。
1	低速	通常、180 秒以内にフェイルオーバーが検出されます。

高速のハートビート間隔は、プロセスおよびネットワークが安定している場合に役立ちます。ネットワークまたはプロセスが最適に構成されていないと、ハートビートを見逃す可能性があり、そうなった場合は誤って障害検出が示されることがあります。

- WebSphere Application Server 環境のフェイルオーバーを構成します。

  WebSphere Application Server Network Deployment バージョン 6.0.2 以降は、WebSphere eXtreme Scale のフェイルオーバーを高速で行えるように構成できます。ハード障害の場合のデフォルトのフェイルオーバー時間は、約 200 秒です。ハード障害は、物理的なコンピューターまたはサーバーの破損、ネットワーク・ケーブルの切断、オペレーティング・システム・エラーのことです。プロセスの異常終了やソフト障害による障害は、一般的に 1 秒未満でフェイルオーバーされます。ソフト障害の障害検出は、デッド・プロセスのネットワーク・ソケットがそのプロセスをホスティングするサーバーのオペレーティング・システムにより自動的にクローズされるときに発生します。

  コア・グループのハートビート構成

  WebSphere Application Server プロセスで実行されているWebSphere eXtreme Scale は、アプリケーション・サーバーのコア・グループ設定のフェイルオーバー特性を継承します。以下のセクションでは、以下のようなさまざまなバージョンの WebSphere Application Server Network Deployment のコア・グループ・ハートビート設定を構成する方法について説明します。

  – **WebSphere Application Server Network Deployment バージョン 6.x または 7.x のコア・グループ設定を更新します。**

    ハートビート間隔は、WebSphere Application Server のバージョン 6.0 からバージョン 6.1.0.12 までは秒単位、バージョン 6.1.0.13 からはミリ秒単位で指定します。また、欠落ハートビートの数も指定する必要があります。この値は、ピア Java 仮想マシン (JVM) に障害が起きたと見なされるまでに、容認される欠落ハートビートの数を示します。ハード障害の検出時間は、ほぼハートビート間隔と欠落ハートビート数の積です。

    これらのプロパティーは、WebSphere 管理コンソールで、コア・グループに対してカスタム・プロパティーを使用して指定します。構成について詳しくは、コア・グループ・カスタム・プロパティーを参照してください。アプリケーションによって使用されるすべてのコア・グループに対して、以下のプロパティーを指定する必要があります。

    - ハートビート間隔は、IBM_CS_FD_PERIOD_SEC カスタム・プロパティー (秒単位) または IBM_CS_FD_PERIOD_MILLIS カスタム・プロパティー (ミリ秒単位、V6.1.0.13 以降が必要) を使用して指定します。
    - 欠落ハートビート数は、IBM_CS_FD_CONSECUTIVE_MISSED カスタム・プロパティーを使用して指定します。

    IBM_CS_FD_PERIOD_SEC プロパティーのデフォルト値は 20 で、IBM_CS_FD_CONSECUTIVE_MISSED プロパティーのデフォルト値は 10 です。IBM_CS_FD_PERIOD_MILLIS プロパティーを指定すると、設定されている IBM_CS_FD_PERIOD_SEC カスタム・プロパティーがオーバーライドされます。これらのプロパティーの値は、正の整数値です。

WebSphere Application Server Network Deployment 6.x サーバーで 1500 ミリ秒の障害検出時間を実現するには、以下の設定を使用します。

- `IBM_CS_FD_PERIOD_MILLIS = 750` を設定 (WebSphere Application Server Network Deployment バージョン 6.1.0.13 以降)
- `IBM_CS_FD_CONSECUTIVE_MISSED = 2` を設定

– **WebSphere Application Server Network Deployment バージョン 7.0 でのコア・グループ設定を更新します。**

バージョン 7.0 の WebSphere Application Server Network Deployment は、フェイルオーバー検出を増減するために調整できる以下の 2 つのコア・グループ設定を提供します。

- **ハートビート伝送期間。** デフォルト値は 30000 ミリ秒です。
- **ハートビート・タイムアウト期間。** デフォルト値は 180000 ミリ秒です。

これらの設定を変更する方法について詳しくは、WebSphere Application Server Network Deployment インフォメーション・センター: ディスカバリーおよび障害検出の設定を参照してください。

WebSphere Application Server Network Deployment バージョン 7 サーバーで 1500 ミリ秒の障害検出時間を実現するには、以下の設定を使用します。

- ハートビート伝送期間を 750 ミリ秒に設定します。
- ハートビート・タイムアウト期間を 1500 ミリ秒に設定します。

次のタスク

短いフェイルオーバー時間を指定するようにこれらの設定を変更すると、注意すべきシステム調整上の問題が生じます。まず Java はリアルタイム環境ではありません。JVM に長期のガーベッジ・コレクション時間が発生すると、スレッドが遅延する可能性があります。JVM をホスティングするマシンの負荷が大きくなった (JVM 自身またはマシンで実行中の他のプロセスが原因) 場合にも、スレッドが遅延する可能性があります。スレッドが遅延された場合、ハートビートが正確な時間で送信されない可能性があります。最悪の場合、必要なフェイルオーバー時間で遅延が生じる可能性があります。 スレッドが遅延すると、誤障害検出が発生します。実動環境で誤障害検出が発生しないように、システムを調整し、サイズ設定する必要があります。これを確実にするには、適切な負荷テストが最善の策です。

注: eXtreme Scale の現行バージョンは、WebSphere Real Time をサポートします。

高可用性カタログ・サービス

カタログ・サービスは、使用しているカタログ・サーバーのグリッドであり、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・サービスをグリッドにクラスター化する必要があります。

カタログ・サービスのコンポーネント

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・サーバー・グリッドのルーティング・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。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・サーバー・グリッドのルーティング・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・サーバー・グリッドとコンテナー・サーバー・グリッドは、まったく別のものです。カタログ・サーバー・グリッドは、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・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・サーバー・グリッドのルーティング・テーブルとサーバー・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。

カタログの責務は、一連のサービスに分割されます。コア・グループ・マネージャーは、ヘルスをモニターするためのピアのグループ化を実行し、配置サービスは、割り振りを行い、管理サービスは、管理のためのアクセスを提供し、ロケーション・サービスは局所性を管理します。

カタログ・グリッド・デプロイメント

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 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での断片の配置を管理します。物理リソース間でのリソース・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。それは、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、実行中のサービスのインスタンスは必ず 1 つとなります。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、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、すべてのグリッド・メンバーで実行されます。

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・サービスをグリッドに構成します。

計画

カタログ・グリッドが開始されると、グリッドのメンバーは相互にバインドされます。カタログ構成を実行時に変更することはできないので、カタログ・グリッド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、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・サーバー・グリッドの開始

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・グリッドへの接続

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・グリッドに接続できます。アプリケーション・サーバーをカタログ・グリッドに接続するのと同じプロパティーを使用します。しかし、プロパティーは、サーバーでカタログのライフサイクルを管理しません。その代わり、プロパティーは、コンテナーがリモート・カタログ・グリッドを見つけることができるようにします。プロパティーの設定については、 「*管理ガイド*」内の WebSphere Application Server 環境でのカタログ・サービス・プロセスの開始についての説明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、*製品概要*のカタログ・サーバー・クォーラムに関する説明を参照してください。

運用チェックリスト

この運用チェックリストを使用して、WebSphere eXtreme Scale のデプロイ用に環境を準備してください。

表 2. 運用チェックリスト

チェックリスト項目	詳細情報

AIX を使用している場合、以下のオペレーティング・システムの設定を調整してください。
 	
TCP_KEEPINTVL
TCP_KEEPINTVL 設定は、ネットワーク障害の検出を可能にする、ソケットのキープアライブ・プロトコルの一部です。このプロパティーは、接続を検証するために送信されるパケット間の間隔を指定します。 WebSphere eXtreme Scale を指定している場合、この値は 10 に設定します。現行設定を確認するには、次のコマンドを実行します。
`# no –o tcp_keepintvl`
現行設定を変更するには、次のコマンドを実行します。
`# no –o tcp_keepintvl=10`
TCP_KEEPINTVL 設定は、0.5 秒単位で設定します。
 	
TCP_KEEPINIT
TCP_KEEPINIT 設定は、ネットワーク障害の検出を可能にする、ソケットのキープアライブ・プロトコルの一部です。このプロパティーは、TCP 接続の初期タイムアウト値を指定します。 WebSphere eXtreme Scale を使用している場合、この値は 40 に設定します。現行設定を確認するには、次のコマンドを実行します。
`# no –o tcp_keepinit`
現行設定を変更するには、次のコマンドを実行します。
`# no –o tcp_keepinit=40`
TCP_KEEPINIT 設定は、0.5 秒単位で設定します。
	• AIX のシステム調整について詳しくは、 AIX システムの調整を参照してください。

表 2. 運用チェックリスト (続き)

チェックリスト項目	詳細情報
orb.properties ファイルを更新すると、グリッドのトランスポート動作を変更できます。 orb.properties ファイルは、java/jre/lib ディレクトリーにあります。	214 ページの『ORB プロパティー・ファイル』
startOgServer スクリプトのパラメーターを使用します。特に、以下のパラメーターを使用します。 • **-jvmArgs** パラメーターを使用してヒープ設定を設定します。 • **-jvmArgs** パラメーターを使用してアプリケーション・クラスパスとプロパティーを設定します。 • エージェント・モニターの構成用に **-jvmArgs** パラメーターを設定します。 **ポートの設定** WebSphere eXtreme Scale は、一部のトランスポートの通信用にポートを開く必要があります。これらのポートはすべて動的に定義されます。ただし、コンテナー間のファイアウォールが使用中の場合はポートを指定する必要があります。ポートに関する以下の情報を使用してください。 **リスナー・ポート** プロセス間の通信に使用するポートを指定する場合、**-listenerPort** 引数を使用できます。 **コア・グループ・ポート** 障害検出に使用するポートを指定する場合、**-haManagerPort** 引数を使用できます。コア・グループは、ゾーンをまたいで通信する必要がないことに注意してください。したがって、ファイアウォールが単一ゾーンのすべてのメンバーに対してオープンである場合は、このポートを設定する必要はありません。 **JMX サービス・ポート** JMX サービスが使用するポートを指定する場合、**-JMXServicePort** 引数を使用できます。 **SSL ポート** -Dcom.ibm.CSI.SSLPort=1234 を **-jvmArgs** 引数として引き渡すと、SSL ポートが 1234 に設定されます。この SSL ポートは、リスナー・ポートと対等のセキュア・ポートです。 **クライアント・ポート** カタログ・サービスのみで使用されます。この値は、**-catalogServiceEndPoints** 引数を使用して指定できます。このパラメーターの値は次の形式になります。 serverName:hostName:clientPort:peerPort	244 ページの『startOgServer スクリプト』
次のセキュリティー設定が正しく構成されていることを検証します。 • トランスポート (SSL) • アプリケーション (認証と許可) セキュリティー設定を検証するには、悪意のあるクライアントを使用してご使用の構成に接続を試みてください。例えば、SSL が必要な設定が構成されている場合に、TCP_IP 設定があるクライアント、あるいは、正しくないトラストストアのクライアントが、サーバーに接続できてはいけません。認証が必要な場合に、クレデンシャル (例えば、ユーザー ID とパスワードなど) を持たないクライアントは、サーバーに接続できてはいけません。許可が実行されている場合に、アクセス許可を持たないクライアントは、サーバー・リソースへのアクセスを認可されるべきではありません。	323 ページの『第 9 章 デプロイメント環境の保護』

表 2. 運用チェックリスト (続き)

チェックリスト項目	詳細情報
ご使用の環境をモニターする方法を選択します。 • xsAdmin – カタログ・サーバーの JMX ポートが、XSAdmin ツールから表示可能になっている必要があります。コンテナー・ポートも、コンテナーからの情報を収集する一部のコマンドにとってアクセス可能である必要があります。 • 以下のベンダー・モニター・ツールから選択できます。 – Tivoli® Enterprise Monitoring Agent – CA Wily Introscope – Hyperic HQ	• 300 ページの『xsAdmin サンプル・ユーティリティーの使用』 • 335 ページの『Java Management Extensions (JMX) セキュリティー』 • 304 ページの『IBM Tivoli Enterprise Monitoring Agent for WebSphere eXtreme Scale のモニター』 • 315 ページの『Hyperic HQ による eXtreme Scale のモニター』 • 311 ページの『CA Wily Introscope による eXtreme Scale アプリケーションのモニター』

第 3 章 キャパシティー・プランニング

初期データ・セット・サイズおよび予測されるデータ・セット・サイズがわかっている場合、WebSphere eXtreme Scale を実行するために必要なキャパシティーを計画できます。こうした計画は、将来の変更に向けて eXtreme Scale を効率よくデプロイするのに役立ちますが、eXtreme Scale の弾力性を最大限に生かすことができます。これにより、メモリー内のデータベースや他のタイプのデータベースなど、異なるシナリオを考える必要がなくなります。

グリッド、区画、および断片

eXtreme Scale 分散グリッドは、区画に分割されています。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複製断片) から構成されます。区画に必ず複製断片を置く必要はありませんが、複製断片は高可用性をもたらします。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、断片化の概念に大きく依存します。

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 つ以上の複製断片が含まれます。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、FIXED_PARTITIONS (デフォルト) と PER_CONTAINER の 2 つがあります。以下では、FIXED_PARTITIONS 戦略の使用に焦点を当てて説明します。

断片数

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 の複製断片が存在することになります。この場合もやはり各断片には、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) の複製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。少数の断片で構成すると、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。当初、グリッドをデプロイする場合、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

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。これら 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。

メモリー・サイズ設定および区画数の計算

構成に必要なメモリーの容量および区画数を計算できます。

WebSphere eXtreme Scale は、データを Java 仮想マシン (JVM) のアドレス・スペースに保管します。各 JVM は、JVM に保管されたデータの作成、取得、更新、および削除の呼び出しをサービスするためのプロセッサー・スペースを提供します。さらに、各 JVM は、データ・エントリーおよびレプリカ用のメモリー・スペースを提供します。Java オブジェクトのサイズはさまざまなので、必要なメモリー量を見積もるための測定が必要です。

必要なメモリーのサイズを設定するには、アプリケーション・データを 1 つの JVM にロードします。ヒープ使用量が 60% に達している場合、使用されるオブジェクトの数に注意してください。この数値は、Java 仮想マシン のそれぞれの推奨されるオブジェクトの最大数です。最も的確なサイズ設定を行うには、現実に近いデータを使用します。索引もまたメモリーを消費するので、定義されているすべての索引をサイズ設定に含めてください。メモリー使用量のサイズ見積もりを行う最良の方法は、ガーベッジ・コレクション verbosegc 出力を実行することです。この出力によって、ガーベッジ・コレクション後の数値が分かるからです。ヒープ使用量については MBean またはプログラムでいつでも照会できますが、この方法では消費メモリーは正確には示されません。そういった照会ではヒープの現行スナップショットしか取得できず、未収集のガーベッジが含まれている可能性があるからです。

構成の拡張

区画当たりの断片数 (numShardsPerPartition 値)

区画当たりの断片数である numShardsPerPartition 値を計算するには、プライマリー断片の 1 に、必要な複製断片の総数を加算します。

```
numShardsPerPartition = 1 + total_number_of_replicas
```

Java 仮想マシン 数 (minNumJVMs 値)

構成を拡張するには、まず保管する必要のある合計オブジェクトの最大数を決定します。必要な Java 仮想マシンの数を決めるには、次の式を使用します。

```
minNumJVMS=(numShardsPerPartition * numObjs) / numObjsPerJVM
```

この値を切り上げて、最も近い整数値にしてください。

断片数 (numShards 値)

最終的な増加サイズでは、それぞれの JVM に 10 個の断片が使用されます。前述したように、各 JVM には、1 つのプライマリー断片と (N-1) 個の複製断片があります。この場合、9 個のレプリカがあります。データ保管用に既に多数の Java 仮想マシンがあるため、Java 仮想マシンの数に 10 を掛けて、断片の数を決定することができます。

```
numShards = minNumJVMs * 10 shards/JVM
```

区画数

区画に 1 つのプライマリー断片と 1 つの複製断片がある場合、区画には 2 つの断片 (プライマリーとレプリカ) があります。区画数は、断片数を 2 で割って、一番近い素数に丸めたものです。区画に 1 つのプライマリーと 2 つのレプリカがある場合、区画数は、断片数を 3 で割って、一番近い素数に丸めたものになります。

```
numPartitions = numShards / numShardsPerPartition
```

拡張の例

この例では、エントリー数は当初 250,000,000 であるとします。毎年、エントリー数は 14% 増加します。7 年後には、エントリーの合計数が 500,000,000 となるため、それに応じた容量計画が必要になります。高可用性を実現するため、1 つのレ

プリカが必要になります。レプリカを使用すると、エントリー数は 2 倍、すなわち 1,000,000,000 となります。テストとして、2,000,000 のエントリーを各 JVM に保管できます。このシナリオの計算を使用すると、以下の構成が必要になります。

- 最終的な数のエントリーを保管するために 500 の Java 仮想マシン。
- 5000 の断片 (Java 仮想マシン数の 500 に 10 を掛けたもの)。
- 2500 の区画、すなわち次位の素数である 2503 (5000 の断片を 2 (プライマリー断片と複製断片) で割ったもの)。

構成の開始

前述の計算に基づいて、250 の Java 仮想マシンから始めて、5 年間で 500 の Java 仮想マシンに増やします。そうすると、最終的なエントリー数に達するまで段階的な増加を管理できます。

この構成では、区画ごとに約 200,000 (500,000,000 のエントリーを区画数の 2503 で割ったもの) のエントリーが保管されます。numberOfBuckets パラメーターを次位の素数 (この例では 70887) までのエントリーを収めるマップで設定します。これにより約 3 の比率が保たれます。

Java 仮想マシンの最大数に達した場合

500 という Java 仮想マシンの最大数に達しても、グリッドを拡張できます。Java 仮想マシン の数が 500 を超えるまでになると、各 JVM について断片数が 10 (推奨数) を下回り始めます。つまり断片が大きくなり始めます。これは問題となる場合があります。将来の成長を考慮しながら、サイズ設定処理を繰り返し、区画数を設定し直す必要があります。この作業では、グリッド全体の再始動が必要になります。さもないとグリッド不足となります。

サーバー数

重要: どのような場合にも、サーバーについてはページングは使用しないでください。

1 つの JVM は、ヒープ・サイズを超えるメモリーを使用します。例えば、JVM の 1 GB のヒープでは、実際には 1.4 GB の実メモリーが使用されます。サーバー上の使用可能な空き RAM を判別してください。RAM の容量を JVM あたりのメモリーで割って、サーバー上の Java 仮想マシンの最大数を計算してください。

トランザクションの区画ごとの CPU 見積もり

eXtreme Scale の主要な機能は、その弾力的なスケーリングですが、拡大のための CPU のサイズおよび理想的な数を調整することも重要です。

プロセッサーのコストには、以下が含まれます。

- クライアントからの作成、取得、更新、および削除操作にサービスを提供するコスト。
- 他の Java 仮想マシンの複製のコスト。
- 無効化のコスト。
- 除去ポリシーのコスト。

- ガーベッジ・コレクションのコスト。
- アプリケーション・ロジックのコスト。

サーバーごとの Java 仮想マシン

サーバーは 2 台使用し、サーバーごとに最大の JVM 数を開始します。前のセクションで計算した区画数を使用します。その後、それら 2 台のコンピューターに収まるだけの十分なデータと Java 仮想マシンをプリロードします。クライアントには別のサーバーを使用してください。この 2 台のサーバーからなるグリッドに対し、現実的なトランザクション・シミュレーションを実行します。

ベースラインを計算するには、プロセッサー使用が飽和状態になるようにしてください。プロセッサーを飽和状態にできない場合は、ネットワークが飽和している可能性があります。ネットワークが飽和している場合は、ネットワーク・カードをさらに追加し、複数のネットワーク・カードで Java 仮想マシンをラウンドロビンさせてください。

プロセッサー使用量 60% でコンピューターを実行し、作成、取得、更新、および削除トランザクションの速度を測定します。この測定により、2 台のサーバーでのスループットがわかります。この数値は、サーバー 4 台で倍になり、サーバー 8 台でさらにその倍になり、以降も同様の割合で大きくなります。この拡張の前提は、ネットワーク容量とクライアントのキャパシティーも同様に拡大可能であることです。

結果的に、eXtreme Scale 応答時間は、サーバーの数が増しても安定しています。トランザクション・スループットは、グリッドにコンピューターが追加されるにつれて直線的に増加します。

並列トランザクションの場合の CPU のサイズ設定

グリッドが拡張するにつれ、単一区画トランザクションのスループットが直線的に増加します。並列トランザクションは、サーバーのサブセット (すべてのサーバーである可能性があります) にタッチするので、単一区画トランザクションとは異なります。

トランザクションがすべてのサーバーにタッチする場合、スループットは、トランザクションを開始したクライアントまたはタッチされた最低速のサーバーのスループットに制限されます。グリッドが大きくなれば、データはより広く分散され、提供されるプロセッサー・スペース、メモリー、ネットワークなどが拡張されます。ただし、クライアントは最低速のサーバーの応答を待たなければならなくなり、クライアントはトランザクションの結果を消費しなければならなくなります。

トランザクションがサーバーのサブセットにタッチする場合、N 個のサーバーのうちの M 個が要求を受け取ります。この結果、スループットは、最低速のサーバーのスループットより、N/M 倍した分だけ速くなります。例えば、20 のサーバーがある場合に、トランザクションが 5 つのサーバーにタッチすると、スループットは、グリッド内の最低速のサーバーのスループットを 4 倍したものになります。

並列トランザクションが完了すると、結果がそのトランザクションを開始したクライアント・スレッドに送信されます。ここでこのクライアントは、単一スレッド化

された結果を集約する必要があります。トランザクションでタッチされるサーバーの数が増えると、この集約時間が増加します。ただし、グリッドが拡大すると、各サーバーが戻す結果が小さくなる可能性もあるので、この時間はアプリケーションに依存します。

一般的には、グリッド全体で区画が均等に配分されるので、並列トランザクションはグリッド内のすべてのサーバーにタッチします。この場合、スループットは、最初の事例に制限されます。

要約

このサイズ設定により、以下の 3 つのメトリックが得られます。

- 区画の数。
- 必須メモリーに必要なサーバーの数。
- 必須スループットに必要なサーバーの数。

メモリー所要量に対して 10 個のサーバーが必要であるが、プロセッサーが飽和状態であるため必要とするスループットの 50% しか得られない場合、2 倍の数のサーバーが必要になります。

最高の安定度を実現するために、60% のプロセッサー負荷でサーバー、さらに 60% のヒープ負荷で JVM ヒープを稼働してください。スパイクでプロセッサー使用量を 80% から 90% の間に引き上げることができますが、通常はこのレベルより高いレベルでサーバーを稼働しないようにしてください。

第 4 章 WebSphere eXtreme Scale のインストールとデプロイメント

WebSphere eXtreme Scale は、複数のサーバーにまたがるアプリケーション・データおよびビジネス・ロジックの区画化、複製、および管理を動的に行うために使用できるメモリー内のデータ・グリッドです。

始める前に

- WebSphere eXtreme Scale が現行トポロジーにどのように適合するかを設定します。詳しくは、*製品概要*の WebSphere eXtreme Scale アーキテクチャーとトポロジーの概要を参照してください。
- ご使用の環境が eXtreme Scale をインストールするための前提条件を満たしていることを確認してください。詳しくは、7 ページの『ハードウェアおよびソフトウェアの要件』を参照してください。

このタスクについて

サポートされる環境

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 および Java Platform, Enterprise Edition 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。 eXtreme Scale は、Java ランタイム環境 (JRE) バージョン 1.4.2 以降および WebSphere Application Server バージョン 6.0.2 以降をサポートします。

スタンドアロン eXtreme Scale のインストール

WebSphere Application Server または WebSphere Application Server Network Deployment を含まない環境に、スタンドアロンの eXtreme Scale をインストールすることができます。スタンドアロン・オプションで、eXtreme Scale サーバーをインストールする新規インストール・ロケーションを定義します。

WebSphere Application Server または WebSphere Application Server Network Deployment と製品の統合

eXtreme Scale をインストールして、既存の WebSphere Application Server または WebSphere Application Server Network Deployment のインストールと統合することができます。eXtreme Scale のクライアントとサーバーを両方インストールするか、あるいは、クライアントのみをインストールするかを選択できます。

プロファイルの作成および拡張

eXtreme Scale フィーチャーを使用するためのプロファイルを作成し、拡張します。WebSphere Application Server バージョン 6.1 またはバージョン 7.0 を実行している場合、プロファイル管理ツール・プラグインまたは manageprofiles コマンドが使用できます。WebSphere Application Server バージョン 6.0.2 を実行している場合、プロファイルの作成および拡張には、wasprofile コマンドを使用する必要があります。

保守の適用

ご使用の環境に保守を適用するには、IBM Update Installer バージョン 7.0.0.4 以降を使用してください。

WebSphere eXtreme Scale バージョン 7.0 へのマイグレーション

バージョン 6.1.0.x からバージョン 7.0 にアップグレードするには、変更済みの製品スクリプト・ファイルを新規の製品スクリプト・ファイルとマージして、変更の保守を行います。

始める前に

ご使用のシステムが、マイグレーションおよびインストールしようとしている製品バージョンの最小限の要件を満たしていることを確認してください。詳しくは、7 ページの『ハードウェアおよびソフトウェアの要件』を参照してください。

このタスクについて

変更済みの製品スクリプト・ファイルを /bin ディレクトリーにある新規の製品スクリプト・ファイルとマージして、変更の保守を行います。

ヒント: 製品にインストールされているスクリプト・ファイルを変更しない場合は、以下のマイグレーション・ステップを実行する必要はありません。代わりに、以前のバージョンをアンインストールしてから同じディレクトリーに新規バージョンをインストールすることで、バージョン 7.0 にアップグレードできます。

1. eXtreme Scale を使用しているすべてのプロセスを停止します。
   - スタンドアロン eXtreme Scale 環境で実行中のすべてのプロセスを停止するには、249 ページの『スタンドアロン eXtreme Scale サーバーの停止』を参照してください。
   - WebSphere Application Server または WebSphere Application Server Network Deployment 環境で実行中のすべてのプロセスを停止するには、 コマンド行ユーティリティーを参照してください。
2. 現行インストール・ディレクトリーから変更済みのすべてのスクリプトを一時ディレクトリーに保存します。
3. 製品をアンインストールします。
4. eXtreme Scale バージョン 7.0 をインストールします。 詳しくは、 27 ページの『第 4 章 WebSphere eXtreme Scale のインストールとデプロイメント』を参照してください。
5. 一時ディレクトリーにあるファイルから、/bin ディレクトリーにある新規の製品スクリプト・ファイルに必要な変更をマージします。

6. すべての eXtreme Scale プロセスを開始して、製品の使用を開始します。 詳しくは、 233 ページの『第 7 章 環境の管理』を参照してください。

スタンドアロン WebSphere eXtreme Scale のインストール

WebSphere Application Server または WebSphere Application Server Network Deployment を含まない環境に、スタンドアロンの WebSphere eXtreme Scale をインストールすることができます。

始める前に

- ターゲット・インストール・ディレクトリーが空であるか、存在していないことを確認します。

  注: バージョン 7.0 のインストール先に指定したディレクトリーに、以前のバージョンの eXtreme Scale または ObjectGrid コンポーネントが存在していると、この製品はインストールされません。他のインストール・ディレクトリーを選択するか、あるいは、インストールを取り消すことができます。次に、以前のインストールをアンインストールしてから、再度ウィザードを実行してください。
- パフォーマンスと保守容易性を良くするために、developerWorks® から IBM デベロッパー・キットをダウンロードして、インストールします。

  制約事項: Windows 独立系ベンダーのハードウェアを使用している場合、以下のいずれかのオプションを選択して、デベロッパー・キットをダウンロードしてインストールします。
  - Sun JDK をダウンロードします。
  - 別の独立系ソフトウェア・ベンダーから JDK または JRE をダウンロードします。

このタスクについて

この製品をスタンドアロンとしてインストールする場合、eXtreme Scale クライアントとサーバーを別々にインストールします。したがって、サーバーとクライアントのプロセスは、必要なすべてのリソースにローカル・アクセスします。また、スクリプトと Java アーカイブ (JAR) ファイルを使用して、eXtreme Scale を既存の Java Platform, Standard Edition アプリケーションに組み込むこともできます。

次の表に、このインストールに含まれる JAR ファイルをリストします。

表 3. /ObjectGrid/lib インストール・ディレクトリーのランタイム・ファイル

ファイル名	環境	説明
cglib.jar	ローカル、クライアント、およびサーバー	cglib.jar ファイルは、コピー・オン・ライト・コピー・モードを使用している場合、および EntityManager を使用してエンティティーの変更を追跡している場合に、cglib ユーティリティー関数によって読み取られます。提供されているスクリプトを使用すると、このファイルは自動的にサーバー・ランタイムに組み込まれます。このファイルをクライアントまたはローカル・ランタイムに追加してください。
objectgrid.jar	ローカル、クライアント、およびサーバー	objectgrid.jar ファイルは、Java Platform, Standard Edition バージョン 1.4.2 以降のサーバー・ランタイムによって使用されます。提供されているスクリプトを使用すると、このファイルは自動的にサーバー・ランタイムに組み込まれます。
ogagent.jar	ローカル、クライアント、およびサーバー	ogagent.jar ファイルには、EntityManager API と一緒に使用される Java インスツルメンテーション・エージェントの実行に必要なランタイム・クラスが含まれています。

表 3. /ObjectGrid/lib インストール・ディレクトリーのランタイム・ファイル (続き)

ファイル名	環境	説明
ogclient.jar	ローカルおよびクライアント	ogclient.jar ファイルには、ローカル・ランタイムとクライアント・ランタイムのみが含まれています。このファイルは、Java SE バージョン 1.4.2 以降で使用することができます。
wsogclient.jar	ローカルおよびクライアント	wsogclient.jar ファイルは、WebSphere Application Server バージョン 6.0.2 以降を含む環境に製品をインストールしたときに組み込まれます。このファイルには、ローカルおよびクライアント・ランタイムのみが含まれています。
wxsdynacache.jar	サーバーのみ	wxsdynacache.jar ファイルには、動的キャッシュ・プロバイダーと一緒に使用するために必要なクラスが含まれています。提供されているスクリプトを使用すると、このファイルは自動的にサーバー・ランタイムに組み込まれます。 **重要:** このファイルは、ObjectGrid/dynacache/lib ディレクトリーにあります。

1. ウィザードを使用して、インストールを完了します。 以下のスクリプトを実行して、ウィザードを開始します。
   - Linux UNIX *dvd_root*/install
   - Windows *dvd_root*¥install.bat
2. ウィザードのプロンプトに従って進み、「**終了**」をクリックしてインストールを完了します。

   注: オプション・フィーチャー・パネルに、インストールを選択できる機能がリストされます。ただし、製品のインストール後に、その製品環境にフィーチャーを順次追加することはできません。初期の製品インストール時にフィーチャーのインストールを選択しなかった場合、そのフィーチャーを追加するには、製品をアンインストールしてから再インストールする必要があります。

次のタスク

クライアント・アプリケーション・プロセスとサーバー・プロセスを構成します。詳しくは、65 ページの『第 6 章 WebSphere eXtreme Scale の構成』を参照してください。

関連資料

55 ページの『インストール・パラメーター』
製品のインストールをカスタマイズし、構成するためのパラメーターをコマンド行に指定します。

WebSphere eXtreme Scale プロセスでのオブジェクト・リクエスト・ブローカーの使用

WebSphere Application Server または WebSphere Application Server Network Deployment を含まない環境で、オブジェクト・リクエスト・ブローカー (ORB) を直接使用するアプリケーションを使用して WebSphere eXtreme Scale を使用することができます。

始める前に

eXtreme Scale に組み込まれていないアプリケーション、あるいは他のコンポーネントやフレームワークを実行中に、eXtreme Scale と同じプロセス内で ORB を使用する場合、追加のタスクを実行して、ご使用の環境で eXtreme Scale が正しく実行されていることを確認する必要がある場合があります。

このタスクについて

ご使用の環境で ORB の使用を初期化するには、orb.properties ファイルに ObjectGridInitializer プロパティーを追加します。 ORB を使用して、ご使用の環境にある eXtreme Scale プロセスと別のプロセスの間の通信を使用可能にします。orb.properties ファイルは、java/jre/lib ディレクトリーにあります。プロパティーと設定の説明は、214 ページの『ORB プロパティー・ファイル』を参照してください。

次の行を入力してから、変更を保存します。

```
org.omg.PortableInterceptor.ORBInitializerClass.com.ibm.ws.objectgrid.corba.ObjectGridInitializer
```

タスクの結果

eXtreme Scale は ORB を正しく初期化し、その ORB が使用可能に設定されている他のアプリケーションと共存できます。

eXtreme Scale でカスタム・バージョンの ORB を使用する場合は、58 ページの『カスタム・オブジェクト・リクエスト・ブローカーの構成』を参照してください。

関連資料

214 ページの『ORB プロパティー・ファイル』
Object Request Broker (ORB) がグリッドのトランスポート動作を変更するために使用するプロパティーが、orb.properties ファイルを使用して受け渡されます。

オブジェクト・リクエスト・ブローカーのカスタム・プロパティー

WebSphere eXtreme Scale の WebSphere Application Server との統合

WebSphere eXtreme Scale を WebSphere Application Server または WebSphere Application Server Network Deployment がインストールされている環境にインストールできます。 WebSphere Application Server または WebSphere Application Server Network Deployment の既存のフィーチャーを使用する場合に eXtreme Scale の機能を適用してご使用のアプリケーションを拡張することができます。

始める前に

- WebSphere Application ServerまたはWebSphere Application Server Network Deploymentをインストールします。詳しくは、アプリケーション・サービス提供環境のインストールを参照してください。
- インストールするバージョン (バージョン 6.0.x、バージョン 6.1、またはバージョン 7.0) に基づいて、WebSphere Application Server または WebSphere Application Server Network Deployment の最新のフィックスパックを適用して、製品レベルを更新してください。詳しくは、WebSphere Application Server の最新フィックスパックを参照してください。
- ターゲット・インストール・ディレクトリーに eXtreme Scale の既存インストールがないことを確認します。

- WebSphere Application Server または WebSphere Application Server Network Deployment で実行中のすべてのプロセスを停止します。詳しくは、コマンド行ユーティリティー (Command-line utilities) を参照してください。

このタスクについて

eXtreme Scale を WebSphere Application Server または WebSphere Application Server Network Deployment に統合して、eXtreme Scale の機能をご使用の Java Platform, Enterprise Edition アプリケーションに適用します。Java EE アプリケーションは eXtreme Scale グリッドをホストし、クライアント接続を使用してそのグリッドにアクセスします。

次の表に、このインストールに含まれる Java アーカイブ (JAR) ファイルをリストします。

表 4. /lib インストール・ディレクトリーのランタイム・ファイル

ファイル名	環境	説明
cglib.jar	ローカル、クライアント、およびサーバー	cglib.jar ファイルは、コピー・オン・ライト・コピー・モードを使用している場合、および EntityManager API を使用してエンティティーの変更を追跡している場合に、cglib ユーティリティー関数によって読み取られます。
ogagent.jar	ローカル、クライアント、およびサーバー	ogagent.jar ファイルには、EntityManager API と一緒に使用される Java インスツルメンテーション・エージェントの実行に必要なランタイム・クラスが含まれています。
wsobjectgrid.jar	ローカル、クライアント、およびサーバー	wsobjectgrid.jar ファイルには、 eXtreme Scale のローカル、クライアント、およびサーバー・ランタイムが含まれています。
wsogclient.jar	ローカルおよびクライアント	wsogclient.jar ファイルは、WebSphere Application Server バージョン 6.0.2 以降を含む環境に製品をインストールしたときに組み込まれます。このファイルには、ローカルおよびクライアント・ランタイムのみが含まれています。
wxsdynacache.jar	サーバーのみ	wxsdynacache.jar ファイルには、動的キャッシュ・プロバイダーと一緒に使用するために必要なクラスが含まれています。

1. ウィザードを使用して、インストールを完了します。 以下のスクリプトを実行して、ウィザードを開始します。
   - Linux UNIX *dvd_root*/install
   - Windows *dvd_root*¥install.bat
2. ウィザードのプロンプトに従います。

   オプション・フィーチャー・パネルに、インストールを選択できる機能がリストされます。ただし、製品のインストール後に、その製品環境にフィーチャーを順次追加することはできません。初期の製品インストール時にフィーチャーのインストールを選択しなかった場合、そのフィーチャーを追加するには、製品をアンインストールしてから再インストールする必要があります。

   プロファイル拡張パネルには、eXtreme Scale のフィーチャーで拡張するために選択できる既存プロファイルがリストされます。既に使用中の既存プロファイルを選択すると、警告パネルが表示されます。インストールを続行するには、そのプロファイルに構成されているサーバーを停止するか、「**戻る**」をクリックして選択からそのプロファイルを除去します。

次のタスク

WebSphere Application Server バージョン 6.1 またはバージョン 7.0 を実行している場合、プロファイル管理ツール・プラグインまたは manageprofiles コマンドが使用できます。WebSphere Application Server バージョン 6.0.2 を実行している場合、プロファイルの作成および拡張には、wasprofile コマンドを使用する必要があります。

ご使用の WebSphere Application Server 環境で、アプリケーションのデプロイ、カタログ・サービスの開始、およびコンテナーの開始を実行します。詳しくは、 252 ページの『WebSphere Application Server による WebSphere eXtreme Scale の管理』を参照してください。

関連資料

55 ページの『インストール・パラメーター』
製品のインストールをカスタマイズし、構成するためのパラメーターをコマンド行に指定します。

Installation Factory プラグインを使用したカスタマイズ・パッケージの作成およびインストール

カスタマイズ・インストール・パッケージ (CIP) または統合インストール・パッケージ (IIP) を作成するには、 IBM Installation Factory plug-in for WebSphere eXtreme Scale を使用します。 CIP には、単一の製品インストール・パッケージと各種のオプション資産が含まれます。 IIP は、1 つ以上のインストール・パッケージを組み合わせて、自分でデザインした 1 つのインストール・ワークフローにします。

始める前に

eXtreme Scale のカスタマイズ・パッケージを作成してインストールする前に、次の製品をまずダウンロードしてください。

- IBM Installation Factory for WebSphere Application Server
- IBM Installation Factory plug-in for WebSphere eXtreme Scale

このタスクについて

Installation Factory を使用して、単一の製品コンポーネントを保守パッケージ、カスタマイズ・スクリプト、その他のファイルと組み合わせることによって、CIP が作成できます。 IIP を作成した場合、個別のコンポーネントまたはインストール・パッケージを単一のインストール・パッケージに集約します。

ビルド定義ファイル

ビルド定義ファイルは、カスタマイズ・インストール・パッケージ (CIP) または統合インストール・パッケージ (IIP) をビルドしてインストールする方法を指定する XML 文書です。IBM Installation Factory for WebSphere eXtreme Scale は、ビルド定義ファイルのパッケージ詳細を読み取り、CIP または IIP を生成します。

CIP または IIP を作成する前に、各カスタマイズ・パッケージのビルド定義ファイルを作成する必要があります。ビルド定義ファイルは、インストールする製品コンポーネントまたはインストール・パッケージ、CIP または IIP のロケーション、組

み込む保守パッケージ、インストール・スクリプト、および組み込むように選択したその他のファイルを説明します。IIP のビルド定義ファイル内で Installation Factory が各インストール・パッケージをインストールする順序を指定することもできます。

ビルド定義ウィザードによって、ビルド定義ファイルの作成プロセスを実行することができます。また、ウィザードを使用して、既存のビルド定義ファイルを変更することもできます。ビルド定義ウィザードの各パネルには、パッケージ ID、ビルド定義のインストール・ロケーション、カスタマイズ・パッケージのインストール・ロケーションなど、カスタマイズ・パッケージに関する情報を求めるプロンプトが出ます。この情報はすべて新規のビルド定義ファイルに保存されるか、変更されて既存のビルド定義ファイルに保存されます。詳しくは、CIP ビルド定義ウィザード・パネルおよび IIP ビルド定義ウィザード・パネルを参照してください。

ビルド定義ファイルのみを作成するには、コマンド行インターフェース・ツールを使用して、GUI の外部でカスタマイズ・パッケージを生成することができます。詳しくは、41 ページの『CIP または IIP のサイレント・インストール』を参照してください。

ビルド定義ファイルの作成と CIP の生成

IBM Installation Factory plug-in for WebSphere eXtreme Scale は、ビルド定義ファイルに指定した詳細に従って、カスタマイズ・インストール・パッケージ (CIP) を生成します。ビルド定義では、インストールする製品パッケージ、CIP のロケーション、インストールに組み込む保守パッケージ、インストール・スクリプト・ファイル、および CIP に組み込むその他のファイルを指定します。

このタスクについて

ビルド定義ウィザードを使用して、ビルド定義ファイルを作成し、CIP を生成します。

1. *IF_HOME*/bin ディレクトリーから次のスクリプトを実行して、 Installation Factory を開始します。
   - UNIX Linux `ifgui.sh`
   - Windows `ifgui.bat`

   「**ビルド定義の新規作成**」アイコンをクリックします。
2. ビルド定義ファイルに組み込む製品を選択して「**終了**」をクリックし、ビルド定義ウィザードを開始します。
3. ウィザードのプロンプトに従います。

   「インストール・スクリプトとアンインストール・スクリプト」パネルで、「**スクリプトの追加...**」をクリックして、テーブルにカスタマイズ・インストール・スクリプトを取り込みます。スクリプト・ファイルのロケーションを入力し、エラー・メッセージが表示された場合に続行するチェック・ボックスをクリアします。デフォルトでは、操作は停止されます。「**OK**」をクリックしてパネルに戻ります。

タスクの結果

これで、ビルド定義ファイルが作成されてカスタマイズされ、接続モードでの作業を選択している場合には、CIP が生成されます。

ビルド定義ファイルから CIP を生成するオプションが、ビルド定義ウィザードにない場合は、 *IF_HOME*/bin ディレクトリーから ifcli.sh|bat スクリプトを実行して生成することができます。

次のタスク

CIP をインストールします。

CIP のインストール:

カスタマイズ・インストール・パッケージ (CIP) をインストールすることによって、製品のインストール処理が単純になります。 CIP とは、単一の製品インストール・イメージで、1 つ以上の保守パッケージ、構成スクリプト、その他のファイルを含みます。

始める前に

CIP をインストールする前に、ビルド定義ファイルを作成して、CIP に組み込むオプションを指定する必要があります。詳しくは、34 ページの『ビルド定義ファイルの作成と CIP の生成』を参照してください。

このタスクについて

CIP は、単一の製品コンポーネントを保守パッケージ、カスタマイズ・スクリプト、その他のファイルと組み合わせてインストールします。

1. インストールの準備を行うワークステーション上で実行されているすべてのプロセスを停止します。 デプロイメント・マネージャーを停止するには、次のスクリプトを実行します。
   - Linux UNIX *profile_root*/bin/stopManager.sh
   - Windows *profile_root*¥bin¥stopManager.bat

   ノードを停止するには、次のスクリプトを実行します。
   - Linux UNIX *profile_root*/bin/stopNode.sh
   - Windows *profile_root*¥bin¥stopNode.bat
2. 次のスクリプトを実行して、インストールを開始します。
   - Linux UNIX *CIP_home*/bin/install
   - Windows *CIP_home*¥bin¥install.bat
3. ウィザードのプロンプトに従って、インストールを完了します。

   オプション・フィーチャー・パネルに、インストールを選択できる機能がリストされます。ただし、製品のインストール後に、その製品環境にフィーチャーを順次追加することはできません。初期の製品インストール時にフィーチャーのイン

ストールを選択しなかった場合、そのフィーチャーを追加するには、製品をアンインストールしてから再インストールする必要があります。

プロファイル拡張パネルには、eXtreme Scale のフィーチャーで拡張するために選択できる既存プロファイルがリストされます。既に使用中の既存プロファイルを選択すると、警告パネルが表示されます。インストールを続行するには、そのプロファイルに構成されているサーバーを停止するか、「**戻る**」をクリックして選択からそのプロファイルを除去します。

タスクの結果

CIP が正常にインストールされました。

次のタスク

WebSphere Application Server バージョン 6.1 またはバージョン 7.0 を実行している場合、プロファイル管理ツール・プラグインまたは manageprofiles コマンドを使用して、プロファイルを作成および拡張することができます。 WebSphere Application Server バージョン 6.0.2 を実行している場合、プロファイルの作成および拡張には、wasprofile コマンドを使用する必要があります。詳しくは、43 ページの『WebSphere eXtreme Scale のプロファイルの作成および拡張』を参照してください。

インストール処理中に eXtreme Scale のプロファイルを拡張した場合には、ご使用の WebSphere Application Server 環境で、アプリケーションのデプロイ、カタログ・サービスの開始、およびコンテナーの開始を実行できます。詳しくは、252 ページの『WebSphere Application Server による WebSphere eXtreme Scale の管理』を参照してください。

既存の製品インストール済み環境に保守を適用するための CIP のインストール:

カスタマイズ・インストール・パッケージ (CIP) をインストールすることによって、既存の製品インストール済み環境に保守パッケージを適用することができます。 CIP で既存のインストール済み環境に保守を適用する処理を一般的にスリップ・インストール と呼びます。

始める前に

ビルド定義ファイルを作成して、CIP に組み込むオプションを指定します。詳しくは、34 ページの『ビルド定義ファイルの作成と CIP の生成』を参照してください。

このタスクについて

リフレッシュ・パックまたはフィックスパック、あるいはその両方を含む CIP で保守を適用する場合、以前にインストールされたすべてのプログラム診断依頼書 (APAR) はウィザードによってアンインストールされます。CIP が製品と同じレベルの場合、以前にインストールされた APAR は、CIP 内にパッケージ化されている場合に限り、その状態のままです。既存のインストール済み環境に保守を正常に適用するには、インストールされているフィーチャーを CIP に含める必要があります。

1. インストールの準備を行うワークステーション上で実行されているすべてのプロセスを停止します。 デプロイメント・マネージャーを停止するには、次のスクリプトを実行します。
   - Linux UNIX *profile_root*/bin/stopManager.sh
   - Windows *profile_root*¥bin¥stopManager.bat

   ノードを停止するには、次のスクリプトを実行します。
   - Linux UNIX *profile_root*¥bin¥stopNode.sh
   - Windows *profile_root*¥bin¥stopNode.bat
2. 次のスクリプトを実行して、インストールを開始します。
   - Linux UNIX *CIP_home*/bin/install
   - Windows *CIP_home*¥bin¥install.bat
3. ウィザードのプロンプトに従って、インストールを完了します。

   インストール・プレビュー要約には、結果として得られた製品バージョン、適用可能なフィーチャーおよび暫定修正がリストされます。次に、ウィザードが保守を正常に適用し、製品のフィーチャーを更新します。

タスクの結果

製品のバイナリー・ファイルが *was_home*/properties/version/nif/backup ディレクトリーにコピーされます。 IBM Update Installer を使用して更新をアンインストールし、ワークステーションを復元することができます。詳しくは、『既存の製品インストール済み環境からの CIP 更新のアンインストール』を参照してください。

既存の製品インストール済み環境からの CIP 更新のアンインストール:

製品全体を除去せずに、既存の製品インストール済み環境から CIP の更新を除去することができます。 CIP の更新をアンインストールするには、IBM Update Installer バージョン 7.0.0.4 を使用します。このタスクは、スリップ・アンインストール とも呼ばれます。

始める前に

製品の、少なくとも 1 つ以上の既存コピーがシステムにインストールされている必要があります。

1. Update Installer のバージョン 7.0.0.4 を次の FTP サイトからダウンロードします。

   ftp://ftp.software.ibm.com/software/websphere/cw/process_server/FEP/UPDI/7004
2. Update Installer をインストールします。 詳しくは、WebSphere Application Server インフォメーション・センターの Update Installer for WebSphere Software のインストールを参照してください。
3. CIP のインストール後に環境に追加したフィックスパック、リフレッシュ・パック、または暫定修正があれば、それをアンインストールします。

4. スリップ・インストールに組み込んだ暫定修正があれば、それをアンインストールします。 これは、単一のフィックスパックまたはリフレッシュ・パックをアンインストールするプロセスと同じです。ただし、CIP に組み込まれていた保守は現在、単一の操作で組み込まれるようになりました。
5. Update Installer を使用して CIP をアンインストールします。 保守レベルは更新前の状態に戻り、CIP のファイル名には前に CIP ID が付加されます。 次の例では、保守パッケージの選択パネルで、他の通常の保守パッケージとは異なって CIP がどのように表示されるかを示しています。

   CIP

   `com.ibm.ws.cip.7000.wxs.primary.ext.pak`

タスクの結果

既存の製品インストール済み環境から CIP 更新を正常に除去しました。

ビルド定義ファイルの作成と IIP の生成

IBM Installation Factory plug-in for WebSphere eXtreme Scale は、ビルド定義ファイルで指定されたプロパティー (IIP に組み込むインストール・パッケージ、Installation Factory で各パッケージをインストールする順序、IIP のロケーションなど) に基づいて IIP を生成します。

このタスクについて

ビルド定義ウィザードを使用して、ビルド定義ファイルを作成し、IIP を生成します。

1. *IF_HOME*/bin ディレクトリーから次のスクリプトを実行して、 Installation Factory を開始します。
   - UNIX Linux `ifgui.sh`
   - Windows `ifgui.bat`
2. 「**統合インストール・パッケージの新規作成**」アイコンをクリックして、ビルド定義ウィザードを開始します。
3. ウィザードのプロンプトに従います。
   a. 「IIP の構成 (Construct the IIP)」パネルで、サポートされるインストール・パッケージをリストから選択し、「**インストーラーの追加**」をクリックして、インストール・パッケージを IIP に追加します。 パッケージ名、パッケージ ID、パッケージ・プロパティーを示したパネルが表示されます。選択したパッケージに関する固有情報を表示するには、「**インストール・パッケージの情報を表示**」をクリックします。「**変更**」をクリックして、各オペレーティング・システムのインストール・パッケージのディレクトリー・パスを入力します。 WebSphere Extended Deployment のインストール・パッケージを現在追加している場合には、サポートされるすべてのオペレーティング・システムに同じパッケージを使用するオプションのチェック・ボックスを選択します。「**OK**」をクリックして、「IIP の構成 (Construct the IIP)」パネルに戻ります。デフォルトで、呼び出しが作成されます。

- インストール・パッケージのディレクトリー・パスを変更するには、 IIP リストに使用されているインストール・パッケージからパッケージを選択し、「**変更**」をクリックします。
- 呼び出しを変更するには、呼び出しを選択して、「**変更**」をクリックします。各オペレーティング・システムにおける呼び出しのデフォルトのインストール場所を指定します。デフォルトのインストール・モードにサイレント・インストールを選択する場合には、その場所を応答ファイルに指定します。
- 「**呼び出しの追加**」をクリックして、インストール・パッケージに呼び出しコントリビューションを追加します。呼び出しのプロパティーを指定できるパネルが表示されます。
- インストール・パッケージまたは呼び出しを除去するには、「**除去**」をクリックします。

4. 選択の要約を確認し、「**ビルド定義ファイルを保存し、統合インストール・パッケージを生成する**」オプションを選択してから、「**終了**」をクリックします。

   あるいは、IIP を生成せずにビルド定義ファイルを保存することも選択できます。このオプションを使用した場合には、*IF_home*/bin/ ディレクトリーから `ifcli.bat | ifcli.sh` スクリプトを実行することによって、ウィザードの外で IIP を実際に生成します。

タスクの結果

これで、IIP のビルド定義ファイルが作成され、カスタマイズされます。

次のタスク

IIP をインストールします。

IIP のインストール:

統合インストール・パッケージ (IIP) をインストールするには、IBM Installation Factory plug-in for WebSphere eXtreme Scale を使用します。 IIP は、1 つ以上のインストール・パッケージを組み合わせて、自分でデザインした 1 つのワークフローにします。

始める前に

CIP をインストールする前に、ビルド定義ファイルを作成して、CIP に組み込むオプションを指定する必要があります。詳しくは、38 ページの『ビルド定義ファイルの作成と IIP の生成』を参照してください。

このタスクについて

IIP には、1 つ以上の一般出荷可能インストール・パッケージ、 1 つ以上の CIP、その他のオプションのファイルおよびディレクトリーを含めることができます。 IIP をインストールすることによって、複数のインストール・パッケージ、つまりコントリビューション を 1 つのパッケージに集約し、コントリビューションを特定の順序でインストールしてすべてのインストールを完了します。

1. 以下のスクリプトを実行して、ウィザードを開始します。

- Linux UNIX *IIP_home*/bin/install
- Windows *IIP_home*¥bin¥install.bat

2. 「ようこそ」パネルの「**製品情報 (About)**」をクリックして、パッケージ ID、サポートされるオペレーティング・システム、組み込まれるインストール・パッケージなど、IIP の詳細を表示します。

   オプション: 各パッケージのインストール・オプションを変更するには、「**変更**」をクリックします。

   オプション: ウィザード・パネルには、「**ログの表示**」ボタンが 2 つ表示されます。各パッケージのログを表示する場合は、インストール・パッケージをリストした表の横に表示される「**ログの表示**」ボタンをクリックします。 IIP の全体のログ詳細を表示する場合は、状況情報の横に表示される「**ログの表示**」ボタンをクリックします。

3. 実行するインストール・パッケージを選択し、「**インストール**」をクリックします。 IIP に含まれるすべてのコントリビューションのリストが、呼び出し順に表示されます。インストール中に実行しないコントリビューション呼び出しを指定するには、「**インストール名**」フィールドの横にあるチェック・ボックスをクリアします。

タスクの結果

IIP が正常にインストールされました。

IIP の既存ビルド定義ファイルの変更:

IIP のプロパティーに編集や追加を行って、インストールをさらにカスタマイズすることができます。

このタスクについて

IIP のプロパティーを変更するには、既存のビルド定義ファイルを変更します。

1. *IF_HOME*/bin ディレクトリーから次のスクリプトを実行して、 Installation Factory を開始します。
   - UNIX Linux ifgui.sh
   - Windows ifgui.bat
2. 「**ビルド定義を開く**」アイコンをクリックし、変更するビルド定義ファイルを選択します。
3. 変更する IIP の具体的なプロパティーを選択します。 以下のリストには、可能な変更が含まれています。
   - 現行モード選択を変更します。接続モードでは、現行ワークステーションから、使用するビルド定義を作成し、また、オプションで IIP を生成します。切断モードでは、別のワークステーションで使用するビルド定義ファイルを作成します。
   - IIP がサポートする既存のオペレーティング・システムを追加または除去します。
   - IIP の既存の ID およびバージョンを編集します。

- ビルド定義ファイルのターゲット・ロケーションを編集します。
- IIP のターゲット・ロケーションを編集します。
- IIP のインストール・ウィザードを表示するかどうかを変更します。ウィザードでは、IIP に関する情報と、IIP 実行時のインストール・オプションが示されます。
- IIP に含まれるインストール・パッケージを追加、除去、および編集します。

  重要: サポートされるオペレーティング・システムを追加し、IIP 内のインストール・パッケージのプロパティーを更新していないと、選択したコントリビューションに、IIP がサポートするすべてのオペレーティング・システムに識別されているインストール・パッケージが含まれないことを示す警告メッセージを受け取ります。続行する場合には「**はい**」、インストール・パッケージを編集する場合には「**いいえ**」をクリックします。

4. 選択の要約を確認し、「**ビルド定義ファイルを保存し、統合インストール・パッケージを生成する**」を選択し、「**終了**」をクリックします。

CIP または IIP のサイレント・インストール

ニーズに具体的に対応して構成する完全修飾応答ファイル、またはコマンド行に受け渡すパラメーターのいずれかを使用して、製品のカスタマイズ・インストール・パッケージ (CIP) または統合インストール・パッケージ (IIP) をサイレントにインストールすることができます。

始める前に

CIP または IIP のビルド定義ファイルを作成します。詳しくは、34 ページの『ビルド定義ファイルの作成と CIP の生成』を参照してください。

このタスクについて

サイレント・インストールは、グラフィカル・ユーザー・インターフェース (GUI) バージョンが使用するのと同じインストール・プログラムを使用します。ただし、ウィザード・インターフェースを表示する代わりに、サイレント・インストールは、カスタマイズされたファイルから、あるいはコマンド行にパスされたパラメーターからすべての応答を読み取ります。 IIP をサイレントにインストールする場合、コントリビューションの呼び出しには、応答ファイルに指定したオプションのほかに、コマンド行に直接指定したオプションを組み合わせて使用できます。ただし、コマンド行にコントリビューション・オプションを渡すと、IIP インストーラーは、特定のコントリビューションの応答ファイルに指定されたオプションをすべて無視します。詳しくは、詳細な IIP インストール・オプションを参照してください。

注: 完全修飾応答ファイル名を指定してください。相対パスを指定すると、エラーが発生したことをまったく示さずにインストールが失敗します。

1. オプション: 応答ファイルを使用して CIP または IIP をインストールする場合は、まずファイルをカスタマイズします。
   a. 応答ファイル `wxssetup.response.txt` を製品 DVD からディスク・ドライブにコピーします。

b. 任意のテキスト・エディターで応答ファイルを開き、編集します。 このファイルには、構成プロセスを支援するコメントが含まれています。ファイルには、次のパラメーターを組み込む必要があります。
   - ご使用条件
   - 製品インストールのロケーション

   ヒント: インストーラーは、インストールに選択したロケーションを使用して WebSphere Application Server インスタンスのインストール場所を判別します。複数の WebSphere Application Server インスタンスが含まれるノードにインストールする場合、そのロケーションを明確に定義してください。

c. 次のスクリプトを実行して、カスタマイズ応答ファイルを開始します。
   - Linux UNIX `install -options /absolute_path/response_file.txt -silent`
   - Windows `install.bat -options C:¥drive_path¥response_file.txt -silent`

2. オプション: 特定のパラメーターをコマンド行に渡すことによって CIP または IIP をインストールする場合は、次のスクリプトを実行してインストールを開始します。
   - Linux UNIX `install -silent -OPT silentInstallLicenseAcceptance=true -OPT installLocation=install_location`
   - Windows `install.bat -silent -OPT silentInstallLicenseAcceptance=true -OPT installLocation=install_location`

   ここで、*install_location* は、既存の WebSphere Application Server インストールのロケーションです。
3. 結果ログを検討して、エラーやインストールの失敗を調べます。

タスクの結果

CIP または IIP がサイレントにインストールされました。

次のタスク

WebSphere Application Server バージョン 6.1 またはバージョン 7.0 を実行している場合、プロファイル管理ツール・プラグインまたは manageprofiles コマンドを使用して、プロファイルを作成および拡張することができます。 WebSphere Application Server バージョン 6.0.2 を実行している場合、プロファイルの作成および拡張には、wasprofile コマンドを使用する必要があります。

インストール処理中に eXtreme Scale のプロファイルを拡張した場合には、ご使用の WebSphere Application Server 環境で、アプリケーションのデプロイ、カタログ・サービスの開始、およびコンテナーの開始を実行できます。詳しくは、252 ページの『WebSphere Application Server による WebSphere eXtreme Scale の管理』を参照してください。

WebSphere eXtreme Scale のプロファイルの作成および拡張

製品のインストール後、WebSphere eXtreme Scale の固有のタイプのプロファイルを作成し、既存のプロファイルを拡張します。

始める前に

WebSphere eXtreme Scale をインストールします。詳しくは、31 ページの『WebSphere eXtreme Scale の WebSphere Application Server との統合』を参照してください。

このタスクについて

WebSphere Application Server バージョン 6.0.2 での実行

ご使用の環境に WebSphere Application Server バージョン 6.0.2 が含まれている場合、次の例に示すように、 wasprofile コマンドを使用して WebSphere eXtreme Scale のプロファイルを作成または拡張します。

```
install_root/bin/wasprofile.sh|bat -augment -profileName dmgr_01
-templatePath "C:/ProgramFiles/IBM/WebSphere/AppServer/profileTemplates/xs_augment/dmgr"
```

詳しくは、WebSphere Application Server インフォメーション・センターの wasprofile コマンドを参照してください。

WebSphere Application Server バージョン 6.1 またはバージョン 7.0 での実行

ご使用の環境に WebSphere Application Server バージョン 6.1 またはバージョン 7.0 が含まれている場合、プロファイル管理ツール・プラグインまたは manageprofiles コマンドを使用して、プロファイルを作成および拡張することができます。

次のタスク

実行するタスクに応じて、ファースト・ステップ・コンソールを起動して、製品環境の構成とテストを支援します。または、前のタスクを繰り返して追加プロファイルの作成または拡張を行います。

関連資料

55 ページの『インストール・パラメーター』
製品のインストールをカスタマイズし、構成するためのパラメーターをコマンド行に指定します。

46 ページの『manageprofiles コマンド』
manageprofiles ユーティリティーで、 WebSphere eXtreme Scale テンプレートを使用してプロファイルを作成したり、 eXtreme Scale 拡張テンプレートを使用して既存のアプリケーション・サーバー・プロファイルの拡張および拡張解除を行えます。製品のこの機能を使用するには、ご使用の環境に含まれる少なくとも 1 つのプロファイルが製品用に拡張されている必要があります。

プロファイルを作成するグラフィカル・ユーザー・インターフェースの使用

プロファイル管理ツール・プラグインで提供されているグラフィカル・ユーザー・インターフェース (GUI) を使用して WebSphere eXtreme Scale のプロファイルを作成します。プロファイルは、ランタイム環境を定義するファイル・セットです。

始める前に

注: WebSphere Application Server バージョン 6.0.2 または WebSphere Application Server Network Deployment バージョン 6.0.2 を実行している場合、次の例に示すように、wasprofile コマンドを使用して WebSphere eXtreme Scale のプロファイルを作成または拡張します。

```
install_root/bin/wasprofile.sh|bat -create -profileName dmgr_01
-templatePath "C:/ProgramFiles/IBM/WebSphere/AppServer/profileTemplates/xs_augment/dmgr"
```

詳しくは、WebSphere Application Server インフォメーション・センターの wasprofile コマンドを参照してください。

このタスクについて

製品の機能を使用するために、プロファイル管理ツール・プラグインでは、GUI を使用してプロファイル (WebSphere Application Server のプロファイル、デプロイメント・マネージャーのプロファイル、セルのプロファイル、およびカスタム・プロファイルなど) のセットアップを行うことができます。

プロファイル管理ツール GUI を使用してプロファイルを作成します。 ウィザードを開始するには、以下のオプションのいずれかを選択します。

- ファースト・ステップ・コンソールから「**プロファイル管理ツール**」を選択します。
- 「**スタート**」メニューからプロファイル管理ツールにアクセスします。
- *install_root*/bin/ProfileManagement ディレクトリーから./pmt.sh|bat スクリプトを実行します。

少なくとも 1 つのプロファイルおよび拡張テンプレートが存在する場合にのみ、「アクションの選択」ページが表示されます。

次のタスク

追加のプロファイルを作成したり、既存のプロファイルを拡張したりできます。プロファイル管理ツールを再始動するには、*install_root*/bin/ProfileManagement ディレクトリーから ./pmt.sh|bat コマンドを実行するか、ファースト・ステップ・コンソールで「**プロファイル管理ツール**」を選択します。

ご使用の WebSphere Application Server 環境で、カタログ・サービスの開始、コンテナーの開始、および TCP ポートの構成を実行します。詳しくは、252 ページの『WebSphere Application Server による WebSphere eXtreme Scale の管理』を参照してください。

関連資料

46 ページの『manageprofiles コマンド』

manageprofiles ユーティリティーで、 WebSphere eXtreme Scale テンプレートを使用してプロファイルを作成したり、 eXtreme Scale 拡張テンプレートを使用して既存のアプリケーション・サーバー・プロファイルの拡張および拡張解除を行えます。製品のこの機能を使用するには、ご使用の環境に含まれる少なくとも 1 つのプロファイルが製品用に拡張されている必要があります。

プロファイルを拡張するグラフィカル・ユーザー・インターフェースの使用

製品をインストールした後で、既存のプロファイルを拡張し、 WebSphere eXtreme Scale と互換性を持たせることができます。

始める前に

注: WebSphere Application Server バージョン 6.0.2 または WebSphere Application Server Network Deployment バージョン 6.0.2 を実行している場合、次の例に示すように、wasprofile コマンドを使用して WebSphere eXtreme Scale のプロファイルを作成または拡張します。

```
install_root/bin/wasprofile.sh|bat -augment -profileName dmgr_01
-templatePath "C:/ProgramFiles/IBM/WebSphere/AppServer/profileTemplates/xs_augment/dmgr"
```

詳しくは、WebSphere Application Server インフォメーション・センターの wasprofile コマンドを参照してください。

このタスクについて

既存のプロファイルを拡張する場合、製品固有の拡張テンプレートを適用してプロファイルの変更をします。例えば、WebSphere eXtreme Scale サーバーは、サーバー・プロファイルが xs_augment テンプレートで拡張されていない限り、自動始動されません。

- eXtreme Scale クライアントまたは、クライアントとサーバーをインストールしている場合、xs_augment テンプレートを使用してプロファイルを拡張します。
- 区画化機能のみをインストールしている場合は、pf_augment テンプレートを使用してプロファイルの拡張をします。
- ご使用の環境に eXtreme Scale クライアントと区画化機能が含まれている場合は、両方のテンプレートを適用します。

プロファイル管理ツール GUI を使用して、eXtreme Scale のプロファイルを拡張します。 ウィザードを開始するには、以下のオプションのいずれかを選択します。

- ファースト・ステップ・コンソールから「**プロファイル管理ツール**」を選択します。
- 「**スタート**」メニューからプロファイル管理ツールにアクセスします。
- *install_root*/bin/ProfileManagement ディレクトリーから./pmt.shlbat スクリプトを実行します。

次のタスク

追加のプロファイルを拡張することもできます。プロファイル管理ツールを再始動するには、*install_root*/bin/ProfileManagement ディレクトリーから ./pmt.shlbat コマンドを実行するか、ファースト・ステップ・コンソールで「**プロファイル管理ツール**」を選択します。

ご使用の WebSphere Application Server 環境で、カタログ・サービスの開始、コンテナーの開始、および TCP ポートの構成を実行します。詳しくは、 252 ページの『WebSphere Application Server による WebSphere eXtreme Scale の管理』を参照してください。

関連資料

『manageprofiles コマンド』
manageprofiles ユーティリティーで、 WebSphere eXtreme Scale テンプレートを使用してプロファイルを作成したり、 eXtreme Scale 拡張テンプレートを使用して既存のアプリケーション・サーバー・プロファイルの拡張および拡張解除を行えます。製品のこの機能を使用するには、ご使用の環境に含まれる少なくとも 1 つのプロファイルが製品用に拡張されている必要があります。

manageprofiles コマンド

manageprofiles ユーティリティーで、 WebSphere eXtreme Scale テンプレートを使用してプロファイルを作成したり、 eXtreme Scale 拡張テンプレートを使用して既存のアプリケーション・サーバー・プロファイルの拡張および拡張解除を行えます。製品のこの機能を使用するには、ご使用の環境に含まれる少なくとも 1 つのプロファイルが製品用に拡張されている必要があります。

- プロファイルを作成および拡張する前に、 eXtreme Scale をインストールする必要があります。詳しくは、 31 ページの『WebSphere eXtreme Scale の WebSphere Application Server との統合』を参照してください。
- ご使用の環境に WebSphere Application Server バージョン 6.0.2 が含まれている場合、次の例に示すように、 wasprofile コマンドを使用して eXtreme Scale のプロファイルを作成したり拡張する必要があります。

  ```
  install_root/bin/wasprofile.sh|bat -augment -profileName dmgr_01
  -templatePath "C:/ProgramFiles/IBM/WebSphere/AppServer/profileTemplates/xs_augment/dmgr"
  ```

  詳しくは、WebSphere Application Server インフォメーション・センターの wasprofile コマンドを参照してください。

目的

manageprofiles コマンドは、プロファイルと呼ばれる一連のファイルに、製品プロセスのランタイム環境を作成します。プロファイルは、ランタイム環境を定義します。manageprofiles コマンドを使用して、以下のアクションを行うことができます。

- デプロイメント・マネージャー・プロファイルの作成および拡張
- カスタム・プロファイルの作成および拡張
- スタンドアロン・アプリケーション・サーバー・プロファイルの作成および拡張
- セル・プロファイルの作成および拡張
- 任意のタイプのプロファイルの拡張解除

既存のプロファイルを拡張する場合、製品固有の拡張テンプレートを適用してプロファイルの変更をします。

- eXtreme Scale のクライアント、または、そのクライアントとサーバーの両方をインストールしている場合、xs_augment テンプレートを使用してプロファイルの拡張をします。
- 区画化機能のみをインストールしている場合は、pf_augment テンプレートを使用してプロファイルの拡張をします。
- ご使用の環境に eXtreme Scale クライアントと区画化機能が含まれている場合は、両方のテンプレートを適用します。

ロケーション

このコマンド・ファイルは、*install_root*/bin ディレクトリーにあります。

使用法

詳しいヘルプが必要な場合、以下のように **-help** パラメーターを使用してください。

```
./manageprofiles.sh|bat -create -templatePath install_root/profileTemplates/xs_augment/dmgr -help
```

以降のセクションで、 manageprofiles コマンドを使用して実行できる各タスクを、必須パラメーターのリストと共に説明します。各タスクに指定するオプション・パラメーターに関する詳細は、WebSphere Application Server インフォメーション・センターの manageprofiles コマンドを参照してください。

デプロイメント・マネージャー・プロファイルの作成

manageprofiles コマンドを使用して、 デプロイメント・マネージャー・プロファイルを作成できます。デプロイメント・マネージャーはセルに統合されているアプリケーション・サーバーを管理します。

パラメーター

-create
プロファイルを作成します。(必須)

-templatePath *template_path*
テンプレートへのファイル・パスを指定します。(必須)

以下のフォーマット設定を使用します。

```
-templatePath install_root/profileTemplates/template_type/dmgr
```

ここで、*template_type* は xs_augment または pf_augment です。

例

- xs_augment テンプレートを使用する場合:

```
./manageprofile.sh|bat -create -templatePath install_root/profileTemplates/xs_augment/dmgr
```

- pf_augment テンプレートを使用する場合:

```
./manageprofile.sh|bat -create -templatePath install_root/profileTemplates/pf_augment/dmgr
```

カスタム・プロファイルの作成

manageprofiles コマンドを使用して、 はカスタム・プロファイルを作成します。カスタム・プロファイルは、アプリケーション・サーバー、クラスター、またはその他の Java プロセスを組み込むようにデプロイメント・マネージャーを介してカスタマイズする空のノードです。

パラメーター

-create
プロファイルを作成します。(必須)

-templatePath *template_path*
テンプレートへのファイル・パスを指定します。(必須)

以下のフォーマット設定を使用します。

```
-templatePath install_root/profileTemplates/template_type/managed
```

ここで、*template_type* は xs_augment または pf_augment です。

例

- xs_augment テンプレートを使用する場合:

```
./manageprofile.sh|bat -create -templatePath install_root/profileTemplates/xs_augment/managed
```

- pf_augment テンプレートを使用する場合:

```
./manageprofile.sh|bat -create -templatePath install_root/profileTemplates/pf_augment/managed
```

スタンドアロン・アプリケーション・サーバー・プロファイルの作成

manageprofiles コマンドを使用して、 はスタンドアロン・アプリケーション・サーバー・プロファイルを作成します。

パラメーター

-create
プロファイルを作成します。(必須)

-templatePath *template_path*
テンプレートへのファイル・パスを指定します。(必須)

以下のフォーマット設定を使用します。

```
-templatePath install_root/profileTemplates/template_type/default
```

ここで、*template_type* は xs_augment または pf_augment です。

例

- xs_augment テンプレートを使用する場合:

```
./manageprofile.sh|bat -create -templatePath install_root/profileTemplates/xs_augment/default
```

- pf_augment テンプレートを使用する場合:

```
./manageprofile.sh|bat -create -templatePath install_root/profileTemplates/pf_augment/default
```

セル・プロファイルの作成

manageprofiles コマンドを使用して、デプロイメント・マネージャーおよびアプリケーション・サーバーからなるセル・プロファイルを作成します。

パラメーター

デプロイメント・マネージャー・テンプレートに以下のパラメーターを指定します。

-create
プロファイルを作成します。(必須)

-templatePath *template_path*
テンプレートへのファイル・パスを指定します。(必須)

以下のフォーマット設定を使用します。

```
-templatePath install_root/profileTemplates/template_type/cell/dmgr
```

ここで、*template_type* は `xs_augment` または `pf_augment` です。

アプリケーション・サーバー・テンプレートを使用して以下のパラメーターを指定します。

-create
プロファイルを作成します。(必須)

-templatePath *template_path*
テンプレートへのファイル・パスを指定します。(必須)

以下のフォーマット設定を使用します。

```
-templatePath install_root/profileTemplates/template_type/cell/default
```

ここで、*template_type* は `xs_augment` または `pf_augment` です。

例

- xs_augment テンプレートを使用する場合:

```
./manageprofile.sh|bat -create -templatePath install_root/profileTemplates/xs_augment/cell/dmgr
-nodeProfilePath install_root/profiles/AppSrv01 -cellName cell01dmgr -nodeName node01dmgr
-appServerNodeName node01

./manageprofile.sh|bat -create -templatePath install_root/profileTemplates/xs_augment/cell/default
-dmgrProfilePath install_root/profiles/Dmgr01 -portsFile
install_root/profiles/Dmgr01/properties/portdef.props -nodePortsFile
install_root/profiles/Dmgr01/properties/nodeportdef.props -cellName cell01dmgr
-nodeName node01dmgr -appServerNodeName node01
```

- pf_augment テンプレートを使用する場合:

```
./manageprofile.sh|bat -create -templatePath install_root/profileTemplates/pf_augment/cell/dmgr
-nodeProfilePath install_root/profiles/AppSrv01 -cellName cell01dmgr -nodeName node01dmgr
-appServerNodeName node01

./manageprofile.sh|bat -create -templatePath install_root/profileTemplates/pf_augment/cell/default
```

```
-dmgrProfilePath install_root/profiles/Dmgr01 -portsFile
install_root/profiles/Dmgr01/properties/portdef.props -nodePortsFile
install_root/profiles/Dmgr01/properties/nodeportdef.props -cellName cell01dmgr
-nodeName node01dmgr -appServerNodeName node01
```

デプロイメント・マネージャー・プロファイルの拡張

manageprofiles コマンドを使用して、デプロイメント・マネージャー・プロファイルを拡張します。

パラメーター

-augment
既存のプロファイルを拡張します。(必須)

-profileName
プロファイルの名前を指定します。(必須)

-templatePath *template_path*
インストール・ルート・ディレクトリーにあるテンプレート・ファイルへのパスを指定します。(必須)

以下のフォーマット設定を使用します。

```
-templatePath install_root/profileTemplates/template_type/dmgr
```

ここで、*template_type* は `xs_augment` または `pf_augment` です。

例

- xs_augment テンプレートを使用する場合:

```
./manageprofile.sh|bat -augment -profileName profile01
-templatePath install_root/profileTemplates/xs_augment/dmgr
```

- pf_augment テンプレートを使用する場合:

```
./manageprofile.sh|bat -augment -profileName profile01
-templatePath install_root/profileTemplates/pf_augment/dmgr
```

カスタム・プロファイルの拡張

manageprofiles コマンドを使用して、カスタム・プロファイルを拡張します。

パラメーター

-augment
既存のプロファイルを拡張します。(必須)

-profileName
プロファイルの名前を指定します。(必須)

-templatePath *template_path*
インストール・ルート・ディレクトリーにあるテンプレート・ファイルへのパスを指定します。(必須)

以下のフォーマット設定を使用します。

```
-templatePath install_root/profileTemplates/template_type/managed
```

ここで、*template_type* は `xs_augment` または `pf_augment` です。

例

- xs_augment テンプレートを使用する場合:

```
./manageprofile.sh|bat -augment -profileName profile01 -templatePath install_root/profileTemplates/xs_augment/managed
```

- pf_augment テンプレートを使用する場合:

```
./manageprofile.sh|bat -augment -profileName profile01 -templatePath install_root/profileTemplates/pf_augment/managed
```

スタンドアロン・アプリケーション・サーバー・プロファイルの拡張

manageprofiles コマンドを使用して、スタンドアロン・アプリケーション・サーバー・プロファイルを拡張します。

パラメーター

-augment
既存のプロファイルを拡張します。(必須)

-profileName
プロファイルの名前を指定します。(必須)

-templatePath *template_path*
インストール・ルート・ディレクトリーにあるテンプレート・ファイルへのパスを指定します。(必須)

以下のフォーマット設定を使用します。

```
-templatePath install_root/profileTemplates/template_type/default
```

ここで、*template_type* は `xs_augment` または `pf_augment` です。

例

- xs_augment テンプレートを使用する場合:

```
./manageprofile.sh|bat -augment -profileName profile01 -templatePath install_root/profileTemplates/xs_augment/default
```

- pf_augment テンプレートを使用する場合:

```
./manageprofile.sh|bat -augment -profileName profile01 -templatePath install_root/profileTemplates/pf_augment/default
```

セル・プロファイルの拡張

manageprofiles コマンドを使用して、セル・プロファイルを拡張します。

パラメーター

デプロイメント・マネージャー・プロファイルに以下のパラメーターを指定します。

-augment
既存のプロファイルを拡張します。(必須)

-profileName
プロファイルの名前を指定します。(必須)

-templatePath *template_path*
インストール・ルート・ディレクトリーにあるテンプレート・ファイルへのパスを指定します。(必須)

以下のフォーマット設定を使用します。

```
-templatePath install_root/profileTemplates/template_type/cell/dmgr
```

ここで、*template_type* は xs_augment または pf_augment です。

アプリケーション・サーバー・プロファイルに以下のパラメーターを指定します。

-augment
既存のプロファイルを拡張します。(必須)

-profileName
プロファイルの名前を指定します。(必須)

-templatePath *template_path*
インストール・ルート・ディレクトリーにあるテンプレート・ファイルへのパスを指定します。(必須)

以下のフォーマット設定を使用します。

```
-templatePath install_root/profileTemplates/template_type/cell/default
```

ここで、*template_type* は xs_augment または pf_augment です。

例

- xs_augment テンプレートを使用する場合:

```
./manageprofile.sh|bat -augment -profileName profile01 -templatePath install_root/profileTemplates/xs_augment/cell/dmgr
./manageprofile.sh|bat -augment -profileName profile01 -templatePath install_root/profileTemplates/xs_augment/cell/default
```

- pf_augment テンプレートを使用する場合:

```
./manageprofile.sh|bat -augment -profileName profile01 -templatePath install_root/profileTemplates/pf_augment/cell/dmgr
./manageprofile.sh|bat -augment -profileName profile01 -templatePath install_root/profileTemplates/pf_augment/cell/default
```

プロファイルの拡張解除

プロファイルを拡張解除する場合は、必須の **-unaugment** パラメーターと **-profileName** パラメーターを指定した上で、 **-ignoreStack** パラメーターを **-templatePath** パラメーターと共に指定します。

パラメーター

-unaugment
前に拡張されたプロファイルを拡張解除します。(必須)

-profileName
プロファイルの名前を指定します。このパラメーターは、値が指定されていない場合にデフォルトで発行されます。(必須)

-templatePath *template_path*
インストール・ルート・ディレクトリーにあるテンプレート・ファイルへのパスを指定します。(オプション)

以下のフォーマット設定を使用します。

```
-templatePath install_root/profileTemplates/template_type/profile_type
```

ここで、*template_type* は xs_augment または pf_augment で、*profile_type* は次の 4 つのプロファイル・タイプのいずれかです。

- dmgr: デプロイメント・マネージャー・プロファイル
- managed: カスタム・プロファイル
- default: スタンドアロン・アプリケーション・サーバー・プロファイル

- cell: セル・プロファイル

-ignoreStack
拡張されている特定のプロファイルを拡張解除するために、**-templatePath** パラメーターとともに使用されます。(オプション)

例

- xs_augment テンプレートを使用する場合:

```
./manageprofile.sh|bat -unaugment -profileName profile01 -ignoreStack
-templatePath install_root/profileTemplates/xs_augment/profile_type
```

- pf_augment テンプレートを使用する場合:

```
./manageprofile.sh|bat -unaugment -profileName profile01 -ignoreStack
-templatePath install_root/profileTemplates/pf_augment/profile_type
```

関連タスク

43 ページの『WebSphere eXtreme Scale のプロファイルの作成および拡張』
製品のインストール後、WebSphere eXtreme Scale の固有のタイプのプロファイルを作成し、既存のプロファイルを拡張します。

44 ページの『プロファイルを作成するグラフィカル・ユーザー・インターフェースの使用』
プロファイル管理ツール・プラグインで提供されているグラフィカル・ユーザー・インターフェース (GUI) を使用して WebSphere eXtreme Scale のプロファイルを作成します。プロファイルは、ランタイム環境を定義するファイル・セットです。

45 ページの『プロファイルを拡張するグラフィカル・ユーザー・インターフェースの使用』
製品をインストールした後で、既存のプロファイルを拡張し、 WebSphere eXtreme Scale と互換性を持たせることができます。

非 root プロファイル

非 root ユーザーが製品のプロファイルを作成できるように、非 root ユーザーにファイルおよびディレクトリーへのアクセス権を付与してください。非 root ユーザーは、root ユーザー、別の非 root ユーザー、または同じ非 root ユーザーが作成したプロファイルを拡張することもできます。

全般的に、非 root ユーザーではそれぞれの環境におけるプロファイルの作成、および使用に制限があります。プロファイル管理ツール・プラグインでは、非 root ユーザーに対して固有名とポート値は使用不可になっています。非 root ユーザーは、プロファイル名、ノード名、セル名、およびポートの割り当てについて、プロファイル管理ツールのデフォルト・フィールド値を変更する必要があります。各フィールドで、非 root ユーザーに一定範囲の値を割り当てることを検討してください。非 root ユーザーに対して、適切な値の範囲を守る責任と、独自の定義の整合性を維持する責任を割り当てることができます。

用語「インストーラー」は、root ユーザーまたは非 root ユーザーのいずれかを指します。インストーラーとして、非 root ユーザーにプロファイルを作成し、独自の製品環境を確立する許可を与えることができます。例えば、非 root ユーザーが所有するプロファイルを持ったアプリケーション・デプロイメントをテストする製品環境を作成する場合があります。非 root ユーザーにプロファイルの作成を許可するために完了する具体的なタスクには、次の項目があります。

- 非 root ユーザーが特定のプロファイルの場合に WebSphere Application Server を開始できるように、プロファイルを作成し、プロファイル・ディレクトリーの所有権を非 root ユーザーに割り当てます。
- 非 root ユーザーに適切なファイルおよびディレクトリーの書き込み許可を与えます。これにより、非 root ユーザーはプロファイルを作成できるようになります。このタスクで、プロファイルの作成を許可されたユーザーのグループを作成したり、個々のユーザーがプロファイルを作成できるようにすることができます。
- 製品の保守パッケージをインストールします。これには、非ユーザーにより所有されている既存のプロファイルに必要なサービスが含まれます。インストーラーであれば、保守パッケージが作成するすべての新規ファイルの所有者です。

詳しくは、WebSphere Application Server Network Deployment インフォメーション・センターで、上記タスクの実行手順の例を含む、非 root ユーザーの場合のプロファイル作成についての詳しい説明を参照してください。

インストーラーとして、非 root ユーザーがプロファイルを拡張する許可を与えることもできます。例えば、非 root ユーザーはインストーラーによって作成されたプロファイルを拡張したり、作成するプロファイルを拡張したりすることができます。これらのタスクを完了するには、 WebSphere Application Server Network Deployment 非 root ユーザー拡張プロセスに従ってください。

ただし、インストーラーによって作成されたプロファイルを非 root ユーザーが拡張するときには、非 root ユーザーは拡張前に以下のファイルを作成する必要はありません。これらのファイルはプロファイル作成プロセス中に作成済みであるためです。

- *app_server_root*/logs/manageprofiles.xml
- *app_server_root*/properties/fsdb.xml
- *app_server_root*/properties/profileRegistry.xml

root 以外のユーザーが作成したプロファイルを自ら拡張する場合は、 eXtreme Scale プロファイル・テンプレート内に位置する文書の権限を変更する必要があります。

WebSphere eXtreme Scale のサイレント・インストール

必要に応じて具体的に構成することができる完全修飾応答ファイルを使用するか、またはコマンド行にパラメーターを渡して WebSphere eXtreme Scale をサイレント・インストールします。

始める前に

WebSphere Application Server または WebSphere Application Server Network Deployment 環境で実行中のすべてのプロセスを停止します。stopManager コマンド、stopNode コマンド、および stopServer コマンドに関して詳しくは、コマンド行ユーティリティー (Command-line utilities) を参照してください。

このタスクについて

サイレント・インストールは、グラフィカル・ユーザー・インターフェース (GUI) バージョンが使用するのと同じインストール・プログラムを使用します。ただし、

ウィザード・インターフェースを表示する代わりに、サイレント・インストールは、カスタマイズされたファイルから、あるいはコマンド行にパスされたパラメーターからすべての応答を読み取ります。サンプルの応答ファイルと各オプションの説明については、wxssetup.response.txt を参照してください。

1. オプション: 応答ファイルを使用して eXtreme Scale をインストールする場合は、まずファイルをカスタマイズします。

   注: 完全修飾応答ファイル名を指定してください。相対パスを指定すると、エラーが発生したことをまったく示さずにインストールが失敗します。

   a. 応答ファイルを製品 DVD からディスク・ドライブにコピーします。
   b. 任意のテキスト・エディターで応答ファイルを開き、編集します。 以下のパラメーターを指定する必要があります。
      - ご使用条件
      - インストール・ディレクトリー

      ヒント: eXtreme Scale を WebSphere Application Server 環境にインストールする場合、インストーラーはインストール・ディレクトリーを使用して、既存の WebSphere Application Server インスタンスがインストールされている場所を判別します。複数の WebSphere Application Server インスタンスが含まれるノードにインストールする場合、そのロケーションを明確に定義してください。
   c. 次のスクリプトを実行して、インストールを開始します。

```
./install.sh|bat -options C:/drive_path/response_file.txt -silent
```

2. オプション: 特定のパラメーターをコマンド行に渡すことによって eXtreme Scale をインストールする場合は、次のスクリプトを実行してインストールを開始します。

```
./install.sh|bat -silent -OPT silentInstallLicenseAcceptance=true -OPT installLocation=install_location
```

インストール・パラメーター

製品のインストールをカスタマイズし、構成するためのパラメーターをコマンド行に指定します。

注: 完全修飾応答ファイル名を指定してください。相対パスを指定すると、エラーが発生したことをまったく示さずにインストールが失敗します。

パラメーター

コマンド行で、または製品のオプション・ファイルのインストール中に、次のパラメーターを渡すことができます。

-silent

グラフィカル・ユーザー・インターフェース (GUI) を抑止します。**-options** パラメーターを指定して、インストーラーに、カスタマイズしたオプション・ファイルに従ってインストールを実行するように指示します。 **-options** パラメーターを指定しないと、代わりにデフォルト値が使用されます。

使用例

```
./install.sh|bat -silent -options options_file.txt
```

-options *path_name/file_name*
サイレント・インストールを実行するためにインストーラーが使用するオプション・ファイルを指定します。コマンド行のプロパティーが優先されます。

使用例

```
./install.sh|bat -options c:/path_name/options_file.txt
```

-log #!file_name @*event_type*
次のイベント・タイプを記録するインストール・ログ・ファイルを生成します。

- err
- wrn
- msg1
- msg2
- dbg
- ALL

使用例

```
./install.sh|bat -log # !c:/temp/logfiles.txt @ALL
```

-is:log *path_name/file_name*
GUI の始動中に、インストーラーの Java 仮想マシン (JVM) 検索を含むログ・ファイルを作成します。ログ・ファイルは、指定しないと作成されません。

使用例

```
./install.sh|bat -is:log c:/logs/javalog.txt
```

-is:javaconsole
インストール・プロセス中にコンソール・ウィンドウを表示します。

使用例

```
./install.sh|bat -is:javaconsole
```

-is:silent
通常インストーラーの開始時に表示される Java 初期化ウィンドウを抑止します。

使用例

```
./install.sh|bat -is:silent
```

-is:tempdir *path_name*
インストーラーがインストール時に使用する一時ディレクトリーを指定します。

使用例

```
./install.sh|bat -is:tempdir c:/temp
```

関連タスク

29 ページの『スタンドアロン WebSphere eXtreme Scale のインストール』
WebSphere Application Server または WebSphere Application Server Network Deployment を含まない環境に、スタンドアロンの WebSphere eXtreme Scale をインストールすることができます。

60 ページの『WebSphere eXtreme Scale のアンインストール』
ご使用の環境から WebSphere eXtreme Scale を削除するには、ウィザードを使用するか、または、製品をサイレント・アンインストールすることができます。

31 ページの『WebSphere eXtreme Scale の WebSphere Application Server との統合』
WebSphere eXtreme Scale を WebSphere Application Server または WebSphere Application Server Network Deployment がインストールされている環境にインストールできます。 WebSphere Application Server または WebSphere Application Server Network Deployment の既存のフィーチャーを使用する場合に eXtreme Scale の機能を適用してご使用のアプリケーションを拡張することができます。

43 ページの『WebSphere eXtreme Scale のプロファイルの作成および拡張』
製品のインストール後、WebSphere eXtreme Scale の固有のタイプのプロファイルを作成し、既存のプロファイルを拡張します。

Update Installer を使用して保守パッケージをインストールする

IBM Update Installer を使用して、ご使用の WebSphere eXtreme Scale 環境をさまざまなタイプの保守 (暫定修正、フィックスパック、リフレッシュ・パックなど) で更新します。

このタスクについて

IBM Update Installer を使用して、WebSphere eXtreme Scale のさまざまなタイプの保守パッケージをインストールして、適用します。 Update Installer は定期的に保守されるため、そのツールの最新バージョンを使用する必要があります。

1. ご使用の環境で実行中のすべてのプロセスを停止します。
   - スタンドアロン eXtreme Scale 環境で実行中のすべてのプロセスを停止する場合の詳細は、 249 ページの『スタンドアロン eXtreme Scale サーバーの停止』を参照してください。
   - WebSphere Application Server 環境で実行中のすべてのプロセスを停止する場合は、コマンド行ユーティリティー (Command-line utilities) を参照してください。
2. Update Installer の最新バージョンをダウンロードします。 詳しくは、推奨フィックスを参照してください。
3. Update Installer をインストールします。 詳しくは、WebSphere Application Server インフォメーション・センターの Update Installer for WebSphere Software のインストールを参照してください。
4. インストールしようとする保守パッケージを *updi_root*/maintenance ディレクトリーにダウンロードします。 詳しくは、 サポート・サイトを参照してください。

5. Update Installer を使用して、暫定修正、修正パッケージ、またはリフレッシュ・パックをインストールします。 保守パッケージのインストールは、グラフィカル・ユーザー・インターフェース (GUI) を実行するか、Update Installer をサイレント・モードで実行することで行うことができます。

   updi_root ディレクトリーから次のコマンドを実行して、GUI を開始します。

   - Linux UNIX `update.sh`
   - Windows `update.bat`

   updi_root ディレクトリーから次のコマンドを実行して、Update Installer をサイレント・モードで実行します。

   - Linux UNIX `./update.sh -silent -options responsefile/file_name`
   - Windows `update.bat -silent -options responsefile¥file_name`

   インストール・プロセスが失敗した場合、 *updi_root*/logs/update/tmp ディレクトリーにある一時ログ・ファイルを参照してください。 Update Installer は、インストール・ログ・ファイルが入れられる *install_root*/logs/update/*maintenance_package*.install ディレクトリーを作成します。

カスタム・オブジェクト・リクエスト・ブローカーの構成

ご使用の環境でスタンドアロンの Java Platform, Standard Edition プロセスを実行している場合、WebSphere eXtreme Scale と一緒にオブジェクト・リクエスト・ブローカー (ORB) のカスタム・バージョンを使用できます。

始める前に

WebSphere eXtreme Scale と WebSphere Application Server は両方とも ORB を提供します。この ORB は eXtreme Scale と一緒に使用するように構成済みです。通常の環境では、その ORB を構成したり、あるいは、別の ORB を使用する必要はありません。

このタスクについて

WebSphere eXtreme Scale は、プロセス間の通信を使用可能にするために Object Request Broker (ORB) を使用します。 ORB は eXtreme Scale と WebSphere Application Server に組み込まれています。WebSphere Application Server で提供される IBM Developer Kit または SDK を使用している場合、その ORB は JRE に組み込まれてます。

eXtreme Scale で提供される ORB、IBM SDK で提供される ORB、もしくは WebSphere Application Server で提供される ORB を使用することができます。独立系ソフトウェア・ベンダーの ORB を使用して発生した問題は、IBM ORB で再現可能で、JRE との互換性がないとサポートに問い合わせることはできません。eXtreme Scale は、Sun Microsystems Java Development Kit (JDK) で提供される ORB はサポートしません。 eXtreme Scale は、ほとんどのベンダーのデベロッパー・キットをサポートしますが、eXtreme Scale で提供される ORB を使用することを推奨します。

- ご使用の環境に SDK のバージョン 5 以降が含まれている場合、代替ディレクトリーを指定することで、Java コマンドを開始するスクリプトを更新してください。
  1. カスタム ibmorb.jar ファイルと ibmorbapi.jar ファイルを空のディレクトリーにコピーします。
  - スタンドアロン eXtreme Scale 環境で製品スクリプトを使用する場合は、以下のステップを実行してください。
    1. setupCmdLine ファイルの OBJECTGRID_ENDORSED_DIRS 変数に対するパスを、カスタム ORB ディレクトリーを参照するように編集します。変更内容を保存します。

       objectgridRoot/bin/setupCmdLine.sh ファイルを編集します。

       objectgridRoot¥bin¥setupCmdLine.bat ファイルを編集します。
  - WebSphere Application Server 環境で製品スクリプトを使用する場合は、以下のステップを実行してください。
    1. startOgServer スクリプトに以下のシステム・プロパティーとパラメーターを追加します。

       -jvmArgs –Djava.endorsed.dirs=*custom_ORB_directory*
  - クライアント・アプリケーション・プロセスまたはサーバー・プロセスの開始にカスタム・スクリプトを使用する場合は、以下のステップを実行してください。
    1. カスタム・スクリプトに次のシステム・プロパティーを追加します。

       -Djava.endorsed.dirs=*custom_ORB_directory*
- ご使用の環境にバージョン 1.4.2 の SDK が含まれている場合は、IBM ORB を指定された SDK に組み込みます。
  1. IBM SDK から ORB をダウンロードし、解凍します。ご使用のプラットフォームで使用可能な IBM SDK がない場合は、 IBM Developer Kit for Linux、Java Technology Edition をダウンロードして解凍します。詳しくは、IBM developer kit を参照してください。
  2. java/jre/lib/ibmorb.jar ファイルと java/jre/lib/ibmorbapi.jar ファイルをターゲット SDK 上の java/jre/lib/ext ディレクトリーにコピーします。
  3. orb.properties ファイルを作成または編集します。このファイルは、SDK の java/jre/lib ディレクトリーに入ります。以下のプロパティーを追加するか、または以下のプロパティーがファイルに存在することを確認します。

     ```
     org.omg.CORBA.ORBClass=com.ibm.CORBA.iiop.ORB
     org.omg.CORBA.ORBSingletonClass=com.ibm.rmi.corba.ORBSingleton
     org.omg.PortableInterceptor.ORBInitializerClass.com.ibm.ws.objectgrid.corba.ObjectGridInitializer
     ```

     プロパティーと設定の説明は、 214 ページの『ORB プロパティー・ファイル』を参照してください。
  4. Xerces2 Java 2.9 をダウンロードします。詳しくは、The Apache Xerces Project - Downloads を参照してください。
  5. xercesImpl.jar ファイルと xml-apis.jar ファイルを lib/ext ディレクトリーにコピーします。

タスクの結果

スタンドアロン Java SE クライアントおよびサーバーのプロセスを実行すると、eXtreme Scale でカスタム ORB を使用できます。

関連資料

214 ページの『ORB プロパティー・ファイル』
Object Request Broker (ORB) がグリッドのトランスポート動作を変更するために使用するプロパティーが、orb.properties ファイルを使用して受け渡されます。

オブジェクト・リクエスト・ブローカーのカスタム・プロパティー

WebSphere eXtreme Scale のアンインストール

ご使用の環境から WebSphere eXtreme Scale を削除するには、ウィザードを使用するか、または、製品をサイレント・アンインストールすることができます。

始める前に

重要: アンインストーラーは、すべてのバイナリー・ファイルと、フィックスパックやインテリム・フィックスなどのすべての保守を同時に削除します。

1. eXtreme Scale を実行中のすべてのプロセスを停止します。 そうしないと、アンインストールは失敗します。
   - スタンドアロンの eXtreme Scale をインストールしている場合は、249 ページの『スタンドアロン eXtreme Scale サーバーの停止』を参照してください。
   - 既存の WebSphere Application Server インストール環境に eXtreme Scale をインストールしている場合、WebSphere Application Server プロセスの停止に関する詳細は、 コマンド行ユーティリティー (Command-line utilities) を参照してください。
2. 以下のスクリプトを実行します。
   - UNIX Linux `install_root/uninstall_wxs/uninstall`
   - Windows `install_root¥uninstall_wxs¥uninstall.exe`

タスクの結果

これで、ご使用の環境から eXtreme Scale の削除ができました。

関連資料

55 ページの『インストール・パラメーター』
製品のインストールをカスタマイズし、構成するためのパラメーターをコマンド行に指定します。

第 5 章 WebSphere eXtreme Scale for z/OS のカスタマイズ

WebSphere カスタマイズ・ツールを使用し、カスタマイズ・ジョブを生成して実行し、 WebSphere eXtreme Scale for z/OS® をカスタマイズできます。

始める前に

- システムに最新レベルの WebSphere Application Server Network Deployment が含まれていることを確認します。
  - バージョン 6.1 を実行している場合、最小でも Fix Pack 27 がシステムに必要です。詳しくは、 バージョン 6.1 のアプリケーション・サービス提供環境のインストール を参照してください。
  - バージョン 7.0 を実行している場合、最小でも Fix Pack 3 がシステムに必要です。詳しくは、 バージョン 7.0 のアプリケーション・サービス提供環境のインストール を参照してください。
- WebSphere eXtreme Scale for z/OS をインストールします。詳しくは、 Library ページの WebSphere eXtreme Scale Program Directory を参照してください。

このタスクについて

WebSphere カスタマイズ・ツールを使用して、カスタマイズ定義を生成し、カスタマイズ・ジョブをアップロードして実行し、 WebSphere eXtreme Scale for z/OS をカスタマイズします。詳しくは、次のトピックを参照してください。

- 『WebSphere カスタマイズ・ツールのインストール』
- 63 ページの『カスタマイズ定義の生成』
- 64 ページの『カスタマイズ・ジョブのアップロードおよび実行』

WebSphere カスタマイズ・ツールのインストール

WebSphere eXtreme Scale for z/OS 環境をカスタマイズするには、 WebSphere カスタマイズ・ツール バージョン 7.0.0.3 またはそれ以降をインストールします。

始める前に

WebSphere eXtreme Scale for z/OS をインストールします。詳しくは、 Library ページの WebSphere eXtreme Scale Program Directory を参照してください。

このタスクについて

WebSphere カスタマイズ・ツールはワークステーション・ベースのグラフィック・ツールで、 WebSphere eXtreme Scale for z/OS ランタイム環境を構築するカスタマイズ・ジョブの作成に使用します。

1. FTP を使用して、`xs.wct` と `xspf.wct` の拡張ファイルを z/OS システムから WebSphere カスタマイズ・ツールをインストールするワークステーションにコピーします。 拡張ファイルは、z/OS システム上の `/usr/lpp/zWebSphereXS/util/V7R0/WCT` ディレクトリーにあります。

2. WebSphere カスタマイズ・ツール バージョン 7.0.0.3 またはそれ以降を適切な Web サイトからダウンロードしてインストールします。
   - Windows WebSphere Customization Tools for Windows
   - Linux WebSphere Customization Tools for Linux
3. xs.wct ファイルを WebSphere カスタマイズ・ツール・アプリケーションにアップロードします。
   a. ワークステーションで WebSphere カスタマイズ・ツール・アプリケーションを開始します。
   b. 「**ヘルプ**」 → 「**ソフトウェアも更新**」 → 「**拡張のインストール**」をクリックします。
   c. 「WebSphere カスタマイズ・ツール拡張ロケーション」パネルから「**新規拡張ロケーションのインストール**」をクリックします。
   d. 「ソース・アーカイブ・ファイル」パネルから「**参照**」をクリックし、ステップ 1 で xs.wct ファイルをコピーしたディレクトリーにナビゲートし、「**開く**」をクリックします。
   e. 「要約」パネルで「**次へ**」をクリックします。

      注: 「インストールが正常に完了しました (Install Successful)」のパネルが表示されます。「**終了**」をクリックする前に、ロケーション・フィールドからデータをコピーして保存してください。

      ```
      C:¥Documents and Settings¥Administrator¥WCT¥workspace¥configuration¥
      com.ibm.ws.pmt.update¥com.ibm.ws390.pmt.xs_7.0.0.0¥eclipse
      ```

   f. 「製品構成 (Product Configuration)」パネルから「**拡張ロケーションの追加**」をクリックします。 前のステップでコピーしたデータを「ロケーション」フィールドに貼り付けて、「**OK**」をクリックします。
   g. 「**はい**」をクリックして WebSphere カスタマイズ・ツールを再始動します。
4. xspf.wct ファイルを WebSphere カスタマイズ・ツール・アプリケーションにアップロードします。
   a. 「**ヘルプ**」 → 「**ソフトウェアも更新**」 → 「**拡張のインストール**」をクリックします。
   b. 「WebSphere カスタマイズ・ツール拡張ロケーション」パネルから「**新規拡張ロケーションのインストール**」をクリックします。
   c. 「ソース・アーカイブ・ファイル」パネルから「**参照**」をクリックし、ステップ 1 で xspf.wct ファイルをコピーしたディレクトリーにナビゲートし、「**開く**」をクリックします。
   d. 「要約」パネルで「**次へ**」をクリックします。

      注: 「インストールが正常に完了しました (Install Successful)」のパネルが表示されます。「**終了**」をクリックする前に、ロケーション・フィールドからデータをコピーして保存してください。

      ```
      C:¥Documents and Settings¥Administrator¥WCT¥workspace¥configuration¥
      com.ibm.ws.pmt.update¥com.ibm.ws390.pmt.xs_7.0.0.0¥eclipse
      ```

   e. 「製品構成 (Product Configuration)」パネルから「**拡張ロケーションの追加**」をクリックします。 前のステップでコピーしたデータを「ロケーション」フィールドに貼り付けて、「**OK**」をクリックします。

f. 「**はい**」をクリックして WebSphere カスタマイズ・ツールを再始動します。

次のタスク

両方の拡張ファイルをアップロードして WebSphere カスタマイズ・ツールを再始動したら、プロファイル管理ツールを使用して、 eXtreme Scale for z/OS のカスタマイズ定義を生成することができます。詳しくは、『カスタマイズ定義の生成』を参照してください。

カスタマイズ定義の生成

WebSphere カスタマイズ・ツール内のプロファイル管理ツール機能を使用して、カスタマイズ定義を生成し、 WebSphere eXtreme Scale for z/OS のカスタマイズ・ジョブを作成します。

始める前に

WebSphere カスタマイズ・ツールをインストールし、 `xs.wct` および `xspf.wct` 拡張ファイルをアップロードします。詳しくは、61 ページの『WebSphere カスタマイズ・ツールのインストール』を参照してください。

このタスクについて

WebSphere カスタマイズ・ツールで提供されるプロファイル管理ツールを使用して、カスタマイズ定義を生成できます。*カスタマイズ定義* とは、 WebSphere eXtreme Scale for z/OS を構成する目的でカスタマイズ・ジョブを作成するために使用する一連のファイルです。

1. プロファイル管理ツールを開始します。
   - Windows 「**スタート**」 → 「**プログラム**」 → 「**IBM WebSphere**」 → 「**WebSphere カスタマイズ・ツール**」をクリックします。アプリケーションが開始したら、「**プロファイル管理ツール**」タブをクリックします。
   - Linux *operating_system_menus* → 「**IBM WebSphere**」 → 「**WebSphere カスタマイズ・ツール**」をクリックします。アプリケーションが開始したら、「**プロファイル管理ツール**」タブをクリックします。
2. 既存のロケーションを追加するか、作成するカスタマイズ定義の新しいロケーションを作成します。 「**カスタマイズのロケーション (Customization Locations)**」タブで「**追加**」をクリックします。新しいロケーションを作成した場合、「バージョン」ボックスは、ご使用の z/OS システムにインストールされている既存の WebSphere Application Server 製品のバージョンを指します。

   注: 他の eXtreme Scale カスタマイズ定義に使用しているものと同じロケーションを使用しないでください。
3. カスタマイズ定義を生成します。 「**カスタマイズの定義**」タブで「**拡張**」をクリックします。
4. 作成する定義環境のタイプを選択します。
   - スタンドアロン・アプリケーション・サーバー・ノード
   - デプロイメント・マネージャー

- アプリケーション・サーバー
- 管理対象 (カスタム) ノード

5. パネルのフィールドに入力します。 z/OS システムの作成に使用されるパラメーターの値を指定します。
6. 「**拡張**」をクリックして、カスタマイズ定義を生成します。

次のタスク

ターゲットの z/OS システムにカスタマイズ・ジョブをアップロードします。詳しくは、『カスタマイズ・ジョブのアップロードおよび実行』を参照してください。

カスタマイズ・ジョブのアップロードおよび実行

カスタマイズ定義を生成したら、定義に関連付けられたカスタマイズ・ジョブを WebSphere eXtreme Scale for z/OS システムにアップロードして実行できます。

始める前に

z/OS システムにアップロードするジョブのカスタマイズ定義を生成します。詳しくは、63 ページの『カスタマイズ定義の生成』を参照してください。

このタスクについて

WebSphere カスタマイズ・ツールを使用して作成したカスタマイズ・ジョブをアップロードして実行し、 WebSphere eXtreme Scale for z/OS 環境の管理およびモニターを行います。

1. カスタマイズ・ジョブをアップロードします。 「**カスタマイズの定義**」タブで、アップロードするジョブを選択し、「**プロセス**」をクリックします。
2. z/OS システム上の FTP サーバーにジョブをアップロードします。 「**カスタマイズ定義のアップロード**」パネルに必要な情報を指定します。
3. 「**終了**」をクリックします。
4. カスタマイズ・ジョブを実行します。 「**カスタマイズの指示**」タブをクリックし、各ジョブのカスタマイズの指示に従います。

第 6 章 WebSphere eXtreme Scale の構成

スタンドアロン環境で実行する場合は WebSphere eXtreme Scale を構成し、WebSphere Application Server または WebSphere Application Server Network Deployment を含む環境で実行する場合は、eXtreme Scale を構成します。 eXtreme Scale のデプロイメントでサーバー・グリッド・サイドの構成変更を反映させるには、動的な適用ではなく、プロセスを再始動して変更を有効にする必要があります。一方、クライアント・サイドでは、既存のクライアント・インスタンスの構成設定は変更できませんが、 XML ファイルを使用するか、またはプログラムによって、新しいクライアントを必要な設定で作成できます。クライアントの作成時には、現行のサーバー構成からのデフォルト設定をオーバーライド可能です。

ローカルのメモリー内 ObjectGrid の構成

ローカルのメモリー内 eXtreme Scale 構成は、ObjectGrid 記述子 XML ファイルまたは eXtreme Scale API を使用して作成できます。

このタスクについて

以下は、単純なサンプル XML ファイル、companyGrid.xml です。ファイルの最初の数行には、各 ObjectGrid XML ファイルの必須ヘッダーが含まれています。ファイルは、Customer、Item、OrderLine、および Order BackingMaps を使用して、CompanyGrid ObjectGrid を定義します。デプロイメント・ポリシー XML ファイルは、始動時に eXtreme Scale コンテナーに渡されます。

companyGrid.xml ファイル

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

 <objectGrids>
  <objectGrid name="CompanyGrid">
   <backingMap name="Customer" />
   <backingMap name="Item" />
   <backingMap name="OrderLine" />
   <backingMap name="Order" />
  </objectGrid>
 </objectGrids>

</objectGridConfig>
```

ObjectGridManager インターフェース内で createObjectGrid メソッドのうちの 1 つに XML ファイルを受け渡します。以下のコード・サンプルは、XML スキーマに対して companyGrid.xml ファイルの妥当性検査を行い、CompanyGrid ObjectGrid を作成します。新規に作成された ObjectGrid インスタンスはキャッシュされません。

```
ObjectGridManager objectGridManager = ObjectGridManagerFactory.getObjectGridManager();
ObjectGrid companyGrid = objectGridManager.createObjectGrid("CompanyGrid",
 new URL("file:etc/test/companyGrid.xml"), true, false);
```

XML を使用する代わりとして、ObjectGrid オブジェクトをプログラマチックに作成できます。以下のコード・サンプルは、前記の XML およびコードの代わりに使用できます。

```
ObjectGridManager objectGridManager = ObjectGridManagerFactory.getObjectGridManager();
ObjectGrid companyGrid = objectGridManager.createObjectGrid ("CompanyGrid", false);
BackingMap customerMap= companyGrid.defineMap("Customer");
BackingMap itemMap= companyGrid.defineMap("Item");
BackingMap orderLineMap= companyGrid.defineMap("OrderLine");
BackingMap orderMap = companyGrid.defineMap("Order");
```

eXtreme Scale には、多くのカスタマイズ可能なプラグインおよび属性があります。ObjectGrid XML ファイルについて詳しくは、eXtreme Scale 構成の解説書を参照してください。

ObjectGrid インターフェース

以下のメソッドを使用して ObjectGrid インスタンスと対話することができます。

概要

作成と初期化

ObjectGrid インスタンスの作成に必要なステップについては、ObjectGridManager インターフェースのトピックを参照してください。ObjectGrid インスタンスを作成する方法には、プログラマチックな方法と XML 構成ファイルを使用した 2 つの別の方法があります。詳しくは、API 資料を参照してください。

get メソッドまたは set メソッドとファクトリー・メソッド

どの set メソッドも、ObjectGrid インスタンスを初期化する前に呼び出す必要があります。初期設定メソッドを呼び出した後に set メソッドを呼び出すと、java.lang.IllegalStateException 例外が発生します。ObjectGrid インターフェースの各 getSession メソッドでも初期設定メソッドが暗黙的に呼び出されます。このため、いずれの getSession メソッドを呼び出す前にも、set メソッドを呼び出す必要があります。ただし、ObjectGridEventListener オブジェクトの設定、追加、および除去の場合のみは、このルールの例外となります。これらのオブジェクトは、初期化処理が完了した後に処理することができます。

ObjectGrid インターフェースには以下の主要メソッドが含まれています。

表 5. ObjectGrid インターフェースのメソッド

メソッド	説明
BackingMap defineMap(String name);	defineMap: 一意的に名付けた BackingMap を定義するファクトリー・メソッドです。バッキング・マップの詳細情報については、『BackingMap インターフェース』を参照してください。
BackingMap getMap(String name);	getMap: defineMap を呼び出して、以前に定義された BackingMap を戻します。このメソッドを使用すると、XML 構成でまだ構成されていない場合に BackingMap を構成することができます。
BackingMap createMap(String name);	createMap: BackingMap を作成しますが、この ObjectGrid での使用のために BackingMap をキャッシュすることはありません。このメソッドは、この ObjectGrid での使用のために BackingMap をキャッシュに入れる、 ObjectGrid インターフェースの setMaps(List) メソッドと共に使用してください。これらのメソッドは、Spring Framework を使用して ObjectGrid を構成する場合に使用します。
void setMaps(List mapList);	setMaps: 以前この ObjectGrid で定義されたすべての BackingMap をクリアし、提供されている BackingMap のリストに置き換えます。

表 5. *ObjectGrid* インターフェースのメソッド (続き)

メソッド	説明
public Session getSession() throws ObjectGridException, TransactionCallbackException;	getSession: セッションを戻します。このセッションは、作業単位の開始、コミット、ロールバックの機能を提供します。Session オブジェクトに関する詳細情報については、『Session インターフェース』を参照してください。
Session getSession(CredentialGenerator cg);	getSession(CredentialGenerator cg): CredentialGenerator オブジェクトを使用してセッションを取得します。このメソッドは、クライアント/サーバー環境の ObjectGrid クライアントによってのみ呼び出すことができます。
Session getSession(Subject subject);	getSession(Subject subject): Session を取得するために、ObjectGrid で構成されている Subject オブジェクトではなく、特定の Subject オブジェクトの使用を許可します。
void initialize() throws ObjectGridException;	initialize: ObjectGrid は初期化され、汎用可能です。このメソッドは、ObjectGrid が初期化済み状態にない場合、getSession メソッドが呼び出されると暗黙的に呼び出されます。
void destroy();	destroy: このメソッドが呼び出された後は、フレームワークは分解されて使用できません。
void setTxTimeout(int timeout);	setTxTimeout: この ObjectGrid インスタンスが作成した Session によって開始されたトランザクションに許可されている完了までの時間 (秒単位) を設定する場合にこのメソッドを使用します。指定された時間内にトランザクションが完了しなかった場合は、トランザクションを開始した Session が「タイムアウト」としてマーキングされます。 Session がタイムアウトとしてマークされると、タイムアウトになった Session によって起動される次の ObjectMap メソッドで、 例外が発生します。 Session が「ロールバックのみ」としてマークされると、TransactionTimeoutException 例外がアプリケーションによってキャッチされた後に、アプリケーションがロールバック・メソッドではなくコミット・メソッドを呼び出した場合でも、トランザクションはロールバックされます。タイムアウト値 0 は、トランザクションの完了までに許可される時間が無制限であることを示します。タイムアウト値 0 が使用されると、トランザクションはタイムアウトになりません。このメソッドが呼び出されない場合、このインターフェースの getSession メソッドによって返されるすべての Session のトランザクション・タイムアウト値は、デフォルトで 0 に設定されます。アプリケーションは、com.ibm.websphere.objectgrid.Session インターフェースの setTransactionTimeout メソッドを使用して、Session ごとにトランザクション・タイムアウト設定をオーバーライドできます。 分散の場合は、`objectGrid.xml` ファイルでトランザクション・タイムアウトを構成することもできます。
int getTxTimeout();	getTxTimeout: トランザクション・タイムアウト値 (秒単位) を返します。このメソッドは、setTxTimeout メソッドのタイムアウト・パラメーターとして渡されるものと同じ値を返します。 setTxTimeout メソッドが呼び出されなかった場合、このメソッドは 0 を返し、トランザクションの完了までに許可されている時間が無制限であることを示します。
public int getObjectGridType();	ObjectGrid のタイプを返します。戻り値は、このインターフェースで宣言された定数 LOCAL、SERVER、または CLIENT のうちの 1 つに相当します。
public void registerEntities(Class[] entities);	クラス・メタデータに基づいた 1 つ以上のエンティティーを登録します。エンティティー登録は、BackingMap および他の定義された索引とエンティティーを結合するため、ObjectGrid の初期化の前に必要になります。このメソッドは、複数回呼び出すことができます。
public void registerEntities(URL entityXML);	エンティティー XML ファイルで 1 つ以上のエンティティーを登録します。エンティティー登録は、BackingMap および他の定義された索引とエンティティーを結合するため、ObjectGrid の初期化の前に必要になります。このメソッドは、複数回呼び出すことができます。
void setQueryConfig(QueryConfig queryConfig);	この ObjectGrid の QueryConfig オブジェクトを設定します。QueryConfig オブジェクトにより、この ObjectGrid のマップ間でオブジェクト照会を実行している照会構成が提供されます。
//Keywords	
void associateKeyword(Serializable parent, Serializable child);	非推奨: 索引または照会関数を使用して、特定の属性を持つオブジェクトを取得してください。associateKeyword メソッドは、キーワードに基づいた柔軟な無効化の手段を提供します。このメソッドは、方向関係で 2 つのキーワードを結び付けます。親が無効な場合は、その子も無効になります。子を無効にしても親には影響しません。
//Security	
void setSecurityEnabled()	setSecurityEnabled: セキュリティーを使用可能にします。セキュリティーは、デフォルトでは使用不可です。

表 5. *ObjectGrid* インターフェースのメソッド (続き)

メソッド	説明
void setPermissionCheckPeriod(long period);	setPermissionCheckPeriod: このメソッドは、アクセス権を検査する頻度を示す単一パラメーターを持ちます。アクセス権は、クライアント・アクセスの許可に使用されます。パラメーターが 0 である場合、すべてのメソッドは、許可機構 (JAAS 許可、カスタム許可のどちらか) に問い合わせて、現行サブジェクトがアクセス権を持っているかどうかを確認します。この方法は、許可の実装に依存するので、パフォーマンスに問題を生じる可能性があります。しかし、このタイプの許可は、必要に応じて有効です。あるいは、パラメーターが 0 未満である場合は、許可機構に戻して更新を行うまでに、アクセス権のセットをキャッシュに入れておくミリ秒数を示しています。このパラメーターはパフォーマンスをかなり向上させますが、その間にバックエンド・アクセス権が変更されると、バックエンドのセキュリティー・プロバイダーが変更されていても、 ObjectGrid はアクセスを許可されたり、防止されたりします。
void setAuthorizationMechanism(int authMechanism);	setAuthorizationMechanism: 許可機構を設定します。デフォルトは SecurityConstants.JAAS_AUTHORIZATION です。
setMapAuthorization(MapAuthorization ma);	非推奨: カスタム許可をプラグインする代わりに setObjectGridAuthorization (ObjectGridAuthorization) メソッドを使用してください。 setMapAuthorization: この ObjectGrid インスタンスに対する MapAuthorization プラグインを設定します。このプラグインを使用すると、Subject オブジェクトに含まれているプリンシパルに対する ObjectMap または JavaMap アクセスを許可することができます。このプラグインの通常の実装は、Subject オブジェクトからプリンシパルを検索し、指定されたアクセス権がプリンシパルに付与されているかどうかを確認することです。
setSubjectSource(SubjectSource ss);	setSubjectSource: SubjectSource プラグインを設定します。このプラグインを使用すると、ObjectGrid クライアントを表す Subject オブジェクトを取得できます。このサブジェクトは、ObjectGrid 許可に使用されます。ObjectGrid.getSession メソッドを使用してセッションを取得するとき、セキュリティーが使用可能になっている場合は、ObjectGrid ランタイムによって SubjectSource.getSubject メソッドが呼び出されます。このプラグインは、既に認証済みのクライアントの場合に役立ちます。つまり、このプラグインは、認証済み Subject オブジェクトを検索して、ObjectGrid インスタンスに渡すことができます。他の認証は必要ありません。
setSubjectValidation(SubjectValidation sv);	setSubjectValidation: この ObjectGrid インスタンスに対する SubjectValidation プラグインを設定します。このプラグインを使用すると、ObjectGrid に渡される javax.security.auth.Subject サブジェクトが、改ざんされていない有効なサブジェクトであるかどうかを検証できます。 Subject オブジェクトが改ざんされたかどうかは作成者のみが知っているため、このプラグインの実装には、Subject オブジェクト作成者からのサポートが必要です。ただし、サブジェクト作成者が、Subject が改ざんされたかどうかを関知していない場合もあります。この場合、このプラグインは使用しないようにしてください。
void setObjectGridAuthorization(ObjectGridAuthorization ogAuthorization);	この ObjectGrid インスタンスに対する ObjectGridAuthorization を設定します。このメソッドに NULL を引き渡して、このメソッドの初期の呼び出しから以前の set ObjectGridAuthorization オブジェクトを除去し、この <code>ObjectGrid</code> が ObjectGridAuthorization オブジェクトに関連付けられていないことを示します。このメソッドは、ObjectGrid セキュリティーが有効な場合にのみ使用できます。ObjectGrid セキュリティーが無効な場合は、提供された ObjectGridAuthorization は使用できません。ObjectGridAuthorization プラグインは、ObjectGrid およびマップへのアクセスを許可するのに使用されます。XD 6.1 では、setMapAuthorization は非推奨で setObjectGridAuthorization の使用を推奨しています。ただし、MapAuthorization プラグインおよび ObjectGridAuthorization プラグインの両方が使用される場合は、非推奨であるとしても、ObjectGrid は提供された MapAuthorization を使用して、マップへのアクセスを許可します。

ObjectGrid インターフェース: プラグイン

ObjectGrid インターフェースには、対話をさらに拡張可能にするための、オプションのプラグイン・ポイントがいくつかあります。

```
void addEventListener(ObjectGridEventListener cb);
void setEventListeners(List cbList);
void removeEventListener(ObjectGridEventListener cb);
void setTransactionCallback(TransactionCallback callback);
int reserveSlot(String);
// Security related plug-ins
void setSubjectValidation(SubjectValidation subjectValidation);
void setSubjectSource(SubjectSource source);
void setMapAuthorization(MapAuthorization mapAuthorization);
```

- ObjectGridEventListener: ObjectGridEventListener インターフェースは、ObjectGrid で重大なイベントが発生したときに通知を受け取るために使用します。これらのイベントには、ObjectGrid の初期化、トランザクションの開始、トランザクションの終了、および ObjectGrid の破棄などがあります。これらのイベントを listen するには、 ObjectGridEventListener インターフェースを実装するクラスを作成して、ObjectGrid に追加します。これらのリスナーは、各 Session に関連付けられています。詳細情報については、『リスナーおよび Session インターフェース』を参照してください。
- TransactionCallback: TransactionCallback リスナー・インターフェースを使用すると、開始、コミット、およびロールバック・シグナルなどのトランザクション・イベントは、このインターフェースを送信することができます。一般に、TransactionCallback リスナー・インターフェースは、ローダーと併用されます。詳細情報については、『TransactionCallback プラグインおよびローダー』を参照してください。これらのイベントは、外部リソースと一緒に、または複数のローダー内で、トランザクションを調整する目的で使用できます。
- reserveSlot: この ObjectGrid のプラグインが、TxID のようなスロットを持つオブジェクト・インスタンスでの使用のためにスロットを予約できるようにします。
- SubjectValidation: セキュリティーが使用可能である場合、このプラグインは、ObjectGrid に渡される javax.security.auth.Subject クラスの妥当性検査に使用できます。
- MapAuthorization: セキュリティーが使用可能である場合、このプラグインは、サブジェクト・オブジェクトによって表されるプリンシパルへの ObjectMap アクセスを許可する目的で使用できます。
- SubjectSource: セキュリティーが使用可能である場合、このプラグインは、ObjectGrid クライアントを表すサブジェクト・オブジェクトを取得する目的で使用できます。そうすると、このサブジェクトは ObjectGrid 許可に使用されます。

プラグインについて詳しくは、「*プログラミング・ガイド*」内のプラグインの概要説明を参照してください。

BackingMap インターフェース

各 ObjectGrid インスタンスは、BackingMap オブジェクトのコレクションを含みます。各 BackingMap を指定したり ObjectGrid インスタンスに追加したりするには、ObjectGrid インターフェースの defineMap メソッドまたは createMap メソッドをそれぞれ使用します。これらのメソッドは BackingMap インスタンスを戻し、このインスタンスは個々の Map の振る舞いを定義するために使用されます。

Session インターフェース

Session インターフェースを使用してトランザクションを開始し、アプリケーションと BackingMap オブジェクト間のトランザクション対話を実行するのに必要な ObjectMap または JavaMap を取得します。ただし、トランザクションの変更は、トランザクションがコミットされるまで、BackingMap オブジェクトに適用されません。 BackingMap は、個々のマップのためにコミットされたデータのメモリー内キャッシュとみなすことができます。詳しくは、*プログラミング・ガイド* でセッションを使用したデータへのアクセスについての情報を参照してください。

BackingMap i インターフェースは、BackingMap 属性を設定するためのメソッドを提供します。 一部の set メソッドを使用して、いくつかのカスタム設計されたプラグインを介して BackingMap を拡張できます。以下は、属性を設定する set メソッド、およびカスタム設計されたプラグインをサポートする set メソッドのリストです。

```
// For setting BackingMap attributes.
public void setReadOnly(boolean readOnlyEnabled);
public void setNullValuesSupported(boolean nullValuesSupported);
public void setLockStrategy( LockStrategy lockStrategy );
public void setCopyMode(CopyMode mode, Class valueInterface);
public void setCopyKey(boolean b);
public void setNumberOfBuckets(int numBuckets);
public void setNumberOfLockBuckets(int numBuckets);
public void setLockTimeout(int seconds);
public void setTimeToLive(int seconds);
public void setTtlEvictorType(TTLType type);
public void setEvictionTriggers(String evictionTriggers);

// For setting an optional custom plug-in provided by application.
public abstract void setObjectTransformer(ObjectTransformer t);
public abstract void setOptimisticCallback(OptimisticCallback checker);
public abstract void setLoader(Loader loader);
public abstract void setPreloadMode(boolean async);
public abstract void setEvictor(Evictor e);
public void setMapEventListeners( List /*MapEventListener*/ eventListenerList );
public void addMapEventListener(MapEventListener eventListener );
public void removeMapEventListener(MapEventListener eventListener );
public void addMapIndexPlugin(MapIndexPlugin index);
public void setMapIndexPlugins(List /¥* MapIndexPlugin ¥*/ indexList );
public void createDynamicIndex(String name, boolean isRangeIndex,
String attributeName, DynamicIndexCallback cb);
public void createDynamicIndex(MapIndexPlugin index, DynamicIndexCallback cb);
public void removeDynamicIndex(String name);
```

リストされている各 set メソッドには、対応する get メソッドが存在します。

BackingMap 属性

各 BackingMap には、BackingMap の振る舞いを変更または制御するために設定可能な以下の属性があります。

- ReadOnly: この属性は、Map を読み取り専用か、読み取り/書き込み可能のいずれにするかを示します。この属性が Map に設定されていない場合は、Map は読み取り/書き込み Map にデフォルトで設定されています。 BackingMap が読み取り専用で設定されている場合、ObjectGrid は可能な場合のみ読み取りのパフォーマンスを最適化します。
- NullValuesSupported: この属性は、Map でヌル値を使用できるようにするかどうかを示します。この属性が設定されていない場合は、Map はヌル値をサポートしません。 Map がヌル値をサポートする場合は、ヌルを戻す get 操作は、値がヌルであること、またはマップが get 操作によって指定されたキーを含まないことを意味します。
- LockStrategy: この属性は、この BackingMap でロック・マネージャーを使用するかどうかを示します。ロック・マネージャーを使用している場合は、LockStrategy 属性を使用して、マップ・エントリーのロッキングにオプティミスティック・ロッキングまたはペシミスティック・ロッキングのいずれの方法が使用されているか示します。この属性が設定されていない場合は、オプティミスティック LockStrategy が使用されます。サポートされているロック・ストラテジーについて詳しくは、ロックのトピックを参照してください。
- CopyMode: この属性は、マップから値が読み取られた場合、またはトランザクションのコミット・サイクル中に BackingMap に値が入れられた場合に、値オブジェクトのコピーが BackingMap によって作成されるかどうかを決定します。さまざまなコピー・モードがサポートされて、アプリケーションがパフォーマンスと

データ保全性の間でトレードオフを行うことを可能にします。この属性が設定されていない場合は、COPY_ON_READ_AND_COMMIT コピー・モードが使用されます。このコピー・モードでは、最良のパフォーマンスが上げられませんが、データ保全性の問題に対しては、最大限の保護が得られます。 BackingMap が EntityManager API エンティティーに関連付けられている場合、CopyMode の設定は、値が COPY_TO_BYTES に設定されている場合を除いて、効力はありません。他の CopyMode が設定されている場合は、常に NO_COPY に設定されます。エンティティー・マップの CopyMode をオーバーライドするには、ObjectMap.setCopyMode メソッドを使用します。コピー・モードについて詳しくは、*プログラミング・ガイド* で、CopyMode メソッドのベスト・プラクティスについての情報を参照してください。

- CopyKey: この属性は、先にマップ内にエントリーが作成されたときに、BackingMap でキー・オブジェクトのコピーを作成するかどうかを決定します。キーは通常変更不可能なオブジェクトであるため、デフォルトのアクションでは、キー・オブジェクトのコピーは作成されません。
- NumberOfBuckets: この属性は、BackingMap で使用するハッシュ・バケット数を示します。BackingMap 実装は、その実装のためにハッシュ・マップを使用します。 BackingMap に多くのエントリーがある場合、バケットが多いほどパフォーマンスが良好であることを意味します。同じバケットを持つキーの数は、バケット数が増えるにつれて少なくなります。また、バケットを増やすことによって、並行性も増大します。この属性は、パフォーマンスを微調整する際に便利です。アプリケーションで NumberOfBuckets 属性が設定されない場合は、未定義のデフォルト値が使用されます。
- NumberOfLockBuckets: この属性は、この BackingMap のロック・マネージャーが使用するロック・バケット数を示します。LockStrategy が OPTIMISTIC または PESSIMISTIC に設定されると、BackingMap にロック・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・マップに多くのエントリーがある場合は、同じバケットについて衝突するキーの数がバケット数が増えるほど減少するため、ロック・バケットの数が多いほどパフォーマンスが良好になります。また、ロック・バケットが多いと並行性も高くなります。 LockStrategy 属性が NONE に設定されると、この BackingMap はロック・マネージャーを使用しません。この場合、numberOfLockBuckets を設定しても効果はありません。この属性を設定しない場合は、未定義の値が使用されます。
- LockTimeout: この属性は、BackingMap がロック・マネージャーを使用しているときに使用されます。LockStrategy 属性が OPTIMISTIC または PESSIMISTIC のいずれかに設定されている場合、BackingMap はロック・マネージャーを使用します。属性値は、ロック・マネージャーがロックが与えられるまで待機する時間を秒単位で決定します。この属性が設定されない場合は、LockTimeout 値として 15 秒が使用されます。発生するロック待機タイムアウト例外について詳しくは、『ペシミスティック・ロック』を参照してください。
- TtlEvictorType: 各 BackingMap には、時間ベースのアルゴリズムを使用して除去対象となるマップ・エントリーを判別する、組み込み型存続時間 Evictor があります。デフォルトでは、組み込み型存続時間 Evictor はアクティブではありません。 3 つの値 (CREATION_TIME、LAST_ACCESS_TIME、または NONE) のうちの 1 つを使用して、 setTtlEvictorType メソッドを呼び出すと、存続時間 Evictor をアクティブにできます。 CREATION_TIME の値は、Evictor が、

TimeToLive 属性を、マップ・エントリーが BackingMap で作成された時間に追加して、 Evictor が BackingMap からマップ・エントリーを除去する時点を判別することを意味します。 LAST_ACCESS_TIME の値は、アプリケーションが稼働している何らかのトランザクションによってマップ・エントリーが最後にアクセスを受けた時間に、Evictor が TimeToLive 属性を追加して、マップ・エントリーを除去する時点を判別することを示します。 TimeToLive 属性によって指定される期間に、マップ・エントリーがトランザクションのアクセスを受けない場合にのみ、マップ・エントリーは除去されます。値 NONE は、Evictor が非アクティブなままで、マップ・エントリーを除去しないことを示します。この属性が設定されないと、NONE はデフォルトとして使用され、存続時間 Evictor はアクティブになりません。組み込み型存続時間 Evictor について詳しくは、『Evictor』を参照してください。

- TimeToLive: TtlEvictorType 属性で説明したように、組み込み型存続時間 Evictor によって各エントリーの作成時または最後のアクセス時に追加する必要がある秒数をこの属性を使用して指定します。この属性が設定されていない場合は、特殊値ゼロを使用して、存続時間が無限大であることを示します。この属性が無限大に設定されると、マップ・エントリーは Evictor によって除去されません。

以下の例は、someGrid ObjectGrid インスタンスでの someMap BackingMap の定義方法および BackingMap インターフェースの set メソッドを使用した BackingMap のさまざまな属性の設定方法を示しています。

```
import com.ibm.websphere.objectgrid.BackingMap;
import com.ibm.websphere.objectgrid.LockStrategy;
import com.ibm.websphere.objectgrid.ObjectGrid;
import com.ibm.websphere.objectgrid.ObjectGridManagerFactory;

...

ObjectGrid og =
ObjectGridManagerFactory.getObjectGridManager().createObjectGrid("someGrid");
BackingMap bm = objectGrid.getMap("someMap");
bm.setReadOnly( true ); // override default of read/write
bm.setNullValuesSupported(false); // override default of allowing Null values
bm.setLockStrategy( LockStrategy.PESSIMISTIC ); // override default of OPTIMISTIC
bm.setLockTimeout( 60 ); // override default of 15 seconds.
bm.setNumberOfBuckets(251); // override default (prime numbers work best)
bm.setNumberOfLockBuckets(251); // override default (prime numbers work best)
```

BackingMap プラグイン

BackingMap インターフェースには、ObjectGrid とより拡張性のある対話を行うためのオプションのプラグ・ポイントが複数あります。

- **ObjectTransformer プラグイン**: 一部のマップ操作の場合、BackingMap は、BackingMap 内のエントリーのキーまたは値をシリアライズ、デシリアライズ、またはコピーする必要があります。 BackingMap は、ObjectTransformer インターフェースのデフォルトの実装を提供することによって、これらのアクションを実行することができます。アプリケーションは、BackingMap が、BackingMap 内のキーまたは値をシリアライズ、デシリアライズ、またはコピーするために使用する、カスタム設計された ObjectTransformer プラグインを提供することで、パフォーマンスを改善できます。詳しくは、「*プログラミング・ガイド*」で ObjectTransformer プラグインに関する説明を参照してください。
- **Evictor プラグイン** : 組み込み型存続時間 Evictor は、時間ベースのアルゴリズムを使用して、BackingMap 内のエントリーを除去する時点を判別します。一部のアプリケーションは、BackingMap 内のエントリーを除去する時点を判別するために、別のアルゴリズムを使用する場合があります。 Evictor プラグインは、カス

タム設計された Evictor を有効にして、BackingMap が使用できるようにします。Evictor プラグインは、組み込み型存続時間 Evictor に追加されます。このプラグインは、存続時間 Evictor を置き換えません。 ObjectGrid は、「Least Recently Used (LRU)」や「Least Frequently Used (LFU)」のような、既知のアルゴリズムを実装するカスタム Evictor プラグインを提供します。アプリケーションは、提供される Evictor プラグインのいずれか 1 つをプラグインすることも、独自の Evictor プラグインを提供することもできます。「*プログラミング・ガイド*」内の除去に関する説明を参照してください。

- **MapEventListener プラグイン**: アプリケーションは、マップ・エントリーの除去や BackingMap 完了のプリロードなどの BackingMap イベントを認識する必要がある場合があります。 BackingMap は、MapEventListener プラグインでメソッドを呼び出し、アプリケーションに BackingMap イベントを通知します。アプリケーションは、setMapEventListener メソッドを使用して、さまざまな BackingMap イベントの通知を受け取り、 BackingMap に 1 つ以上のカスタム設計された MapEventListener プラグインを提供することができます。アプリケーションは、addMapEventListener メソッドまたは removeMapEventListener メソッドを使用して、リストされた MapEventListener オブジェクトを変更することができます。詳しくは、「*プログラミング・ガイド*」で MapEventListener プラグインに関する説明を参照してください。
- **Loader プラグイン** : BackingMap は Map のメモリー内キャッシュです。ローダー・プラグインは、BackingMap がメモリー間でデータを移動させるために使用するオプションであり、 BackingMap の永続ストアに使用されます。例えば、Java Database Connectivity (JDBC) Loader を使用して、 BackingMap とリレーショナル・データベースの 1 つ以上のリレーショナル・テーブルとの間でデータを入れたり取り出したりすることができます。リレーショナル・データベースは、BackingMap の永続ストアとして使用する必要はありません。この Loader はまた、BackingMap と 1 つのファイル間、BackingMap と Hibernate マップ間、BackingMap と Java 2 Platform, Enterprise Edition (JEE) Entity Bean 間、および BackingMap と他のアプリケーション・サーバー間などで、データを移動させるために使用できます。アプリケーションは、使用されるすべての技術に関して、BackingMap と永続ストアの間でデータを移動させるために、カスタム設計されたローダー・プラグインを提供する必要があります。 Loader が提供されない場合は、BackingMap は単純なメモリー内キャッシュになります。詳しくは、「*プログラミング・ガイド*」でローダーの使用に関する説明を参照してください。
- **OptimisticCallback プラグイン**: BackingMap の LockStrategy 属性は、OPTIMISTIC に設定され、BackingMap または Loader プラグインはマップの値に関する比較操作を実行する必要があります。 BackingMap および Loader は OptimisticCallback プラグインを使用して、オプティミスティック・バージョン管理の比較演算を実行します。詳しくは、「*プログラミング・ガイド*」で OptimisticCallback プラグインに関する説明を参照してください。
- **MapIndexPlugin プラグイン**: MapIndexPlugin プラグイン (短縮名は Index) は、BackingMap が、格納されているオブジェクトの指定された属性に基づいて索引をビルドする場合に使用するオプションです。索引によって、アプリケーションは、特定の値または値の範囲を使用してオブジェクトを検索することができます。索引には、静的と動的の 2 つのタイプがあります。詳しくは、『索引付け』を参照してください。

プラグインについて詳しくは、「*プログラミング・ガイド*」でプラグインの概要を参照してください。

マップ・エントリー・ロック

ObjectGrid BackingMap は、複数のロック・ストラテジーをサポートし、キャッシュ・エントリーの整合性を維持します。

各 BackingMap は、以下のロックのストラテジーの 1 つを使用するよう構成できます。

1. オプティミスティック・ロック・モード
2. ペシミスティック・ロック・モード
3. なし

デフォルトのロック・ストラテジーは、OPTIMISTIC です。データの変更が頻繁でない場合は、このオプティミスティック・ロックを使用します。データがキャッシュから読み取られ、トランザクションにコピーされる間、ロックは短期間だけ保持されます。トランザクション・キャッシュがメイン・キャッシュと同期されると、更新されたあらゆるキャッシュ・オブジェクトが元のバージョンに対してチェックされます。チェックが失敗すると、トランザクションはロールバックされ、OptimisticCollisionException 例外となります。

ペシミスティック・ロック・ストラテジーは、キャッシュ・エントリーに対してロックを獲得するため、データが頻繁に変更される場合に使用するようにしてください。キャッシュ・エントリーが読み取られる場合は、必ずロックが獲得され、トランザクションが完了するまでロックが条件付きで保持されます。ロックによっては、セッションのトランザクション分離レベルを使用して、その期間を調整することができます。

データがまったく更新されないか、静止期間のみに更新されるため、ロックが必要ない場合は、NONE ロック・ストラテジーを使用すれば、ロックを使用不可にすることができます。このストラテジーは、ロック・マネージャーを必要としないため、非常に高速です。NONE ロック・ストラテジーは、ルックアップ表または読み取り専用のマップの場合に理想的です。

ロック・ストラテジーについて詳しくは、*製品概要*のロック・ストラテジーに関する説明を参照してください。

ロック・ストラテジーの指定

以下の例は、map1、map2、および map3 の BackingMap 上にロック・ストラテジーを設定する方法を示しており、各マップは異なるロック・ストラテジーを使用しています。最初のスニペットは、ロック・ストラテジー構成に XML を使用する方法を示し、2 番目のスニペットはプログラマチックな方法を示しています。

XML を用いた方法

BackingMap configuration - XML example

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

    <objectGrids>
        <objectGrid name="test">
            <backingMap name="map1"
```

```
            lockStrategy="PESSIMISTIC" numberOfLockBuckets="31"/>
        <backingMap name="map2"
            lockStrategy="OPTIMISTIC" numberOfLockBuckets="409"/>
        <backingMap name="map3"
            lockStrategy="NONE"/>
    </objectGrid>
  </objectGrids>
</objectGridConfig>
```

プログラマチックな方法

```
BackingMap configuration - programmatic example
import com.ibm.websphere.objectgrid.BackingMap;
import com.ibm.websphere.objectgrid.LockStrategy;
import com.ibm.websphere.objectgrid.ObjectGrid;
import com.ibm.websphere.objectgrid.ObjectGridManagerFactory;
...
ObjectGrid og =
 ObjectGridManagerFactory.getObjectGridManager().createObjectGrid("test");
BackingMap bm = og.defineMap("map1");
bm.setLockStrategy( LockStrategy.PESSIMISTIC );
bm.setNumberOfLockBuckets(31);
bm = og.defineMap("map2");
bm.setNumberOfLockBuckets(409);
bm.setLockStrategy( LockStrategy.OPTIMISTIC );
bm = og.defineMap("map3");
bm.setLockStrategy( LockStrategy.NONE );
```

java.lang.IllegalStateException 例外を避けるには、ローカル ObjectGrid インスタンスで initialize メソッドまたは getSession メソッドを使用する前に setLockStrategy メソッドを呼び出す必要があります。

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。上記の例は、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・ストラテジーの構成

WebSphere eXtreme Scale 構成の各 BackingMap に対するオプティミスティック、ペシミスティック、あるいはロックなしのストラテジーを定義できます。

このタスクについて

ロック・ストラテジーは、プログラムで指定するか、あるいは XML を使用して指定できます。ロックについて詳しくは、*製品概要* にあるロック・ストラテジーに関する情報を参照してください。

- **オプティミスティック・ロック・ストラテジーの構成**
  - プログラムによる構成

プログラムによるオプティミスティック・ストラテジーの指定
```
import com.ibm.websphere.objectgrid.BackingMap;
import com.ibm.websphere.objectgrid.LockStrategy;
import com.ibm.websphere.objectgrid.ObjectGrid;
import com.ibm.websphere.objectgrid.ObjectGridManagerFactory;
...
ObjectGrid og =
 ObjectGridManagerFactory.getObjectGridManager().createObjectGrid("test");
BackingMap bm = og.defineMap("optimisticMap");
bm.setLockStrategy( LockStrategy.OPTIMISTIC );
```

  - XML の使用

XML を使用したオプティミスティック・ストラテジーの指定
```
<?xml version="1.0" encoding="UTF-8"?>
<objectGridConfig xmlns:xsi="http://www.w3.org/2001/XMLSchema-instance"
    xsi:schemaLocation="http://ibm.com/ws/objectgrid/config ../objectGrid.xsd"
    xmlns="http://ibm.com/ws/objectgrid/config">
    <objectGrids>
        <objectGrid name="test">
            <backingMap name="optimisticMap"
                lockStrategy="OPTIMISTIC"/>
        </objectGrid>
    </objectGrids>
</objectGridConfig>
```

- **ペシミスティック・ロック・ストラテジーの構成**
  - プログラムによる構成

プログラムによるペシミスティック・ストラテジーの指定
```
import com.ibm.websphere.objectgrid.BackingMap;
import com.ibm.websphere.objectgrid.LockStrategy;
import com.ibm.websphere.objectgrid.ObjectGrid;
import com.ibm.websphere.objectgrid.ObjectGridManagerFactory;
...
ObjectGrid og =
 ObjectGridManagerFactory.getObjectGridManager().createObjectGrid("test");
BackingMap bm = og.defineMap("pessimisticMap");
bm.setLockStrategy( LockStrategy.PESSIMISTIC);
```

  - XML の使用

XML を使用したペシミスティック・ストラテジーの指定
```
<?xml version="1.0" encoding="UTF-8"?>
<objectGridConfig xmlns:xsi="http://www.w3.org/2001/XMLSchema-instance"
    xsi:schemaLocation="http://ibm.com/ws/objectgrid/config ../objectGrid.xsd"
    xmlns="http://ibm.com/ws/objectgrid/config">

    <objectGrids>
        <objectGrid name="test">
            <backingMap name="pessimisticMap"
                lockStrategy="PESSIMISTIC"/>
        </objectGrid>
    </objectGrids>
</objectGridConfig>
```

- **ロックなしストラテジーの構成**
  - プログラムによる構成

プログラムによるロックなしストラテジーの指定
```
import com.ibm.websphere.objectgrid.BackingMap;
import com.ibm.websphere.objectgrid.LockStrategy;
import com.ibm.websphere.objectgrid.ObjectGrid;
import com.ibm.websphere.objectgrid.ObjectGridManagerFactory;
...
ObjectGrid og =
 ObjectGridManagerFactory.getObjectGridManager().createObjectGrid("test");
BackingMap bm = og.defineMap("noLockingMap");
bm.setLockStrategy( LockStrategy.NONE);
```

  - XML の使用

XML を使用したロックなしストラテジーの指定
```
<?xml version="1.0" encoding="UTF-8"?>
<objectGridConfig xmlns:xsi="http://www.w3.org/2001/XMLSchema-instance"
    xsi:schemaLocation="http://ibm.com/ws/objectgrid/config ../objectGrid.xsd"
    xmlns="http://ibm.com/ws/objectgrid/config">
```

```
    <objectGrids>
        <objectGrid name="test">
            <backingMap name="noLockingMap"
                lockStrategy="NONE"/>
        </objectGrid>
    </objectGrids>
</objectGridConfig>
```

次のタスク

java.lang.IllegalStateException 例外を避けるために、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、setLockStrategy メソッドを呼び出す必要があります。

分散 eXtreme Scale グリッドの構成

デプロイメント・ポリシー記述子 XML ファイルを使用して、デプロイメント・トポロジーを管理します。

デプロイメント・ポリシーは、eXtreme Scale コンテナーに提供される XML ファイルとしてエンコードされます。XML ファイルでは、以下の情報が指定されます。

- 各マップ・セットに属するマップ
- 区画の数
- 同期複製および非同期複製の数

コンテナー・サーバーの始動については、 254 ページの『WebSphere Application Server 環境でのコンテナー・プロセスの開始』または 240 ページの『コンテナー・プロセスの開始』を参照してください。

デプロイメント・ポリシーでは、以下の配置の振る舞いも制御されます。

- 配置を実行する前のアクティブ・コンテナーの最小数
- 破損した断片の自動置き換え
- 単一区画の各断片の別のマシンへの配置

デプロイメント・ポリシー XML ファイルについて詳しくは、 156 ページの『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。

エンドポイント情報は、動的環境では事前構成されません。デプロイメント・ポリシーには、サーバー名も物理トポロジー情報もありません。グリッド内のすべての断片は、カタログ・サービスによって自動的にコンテナーに配置されます。カタログ・サービスは、デプロイメント・ポリシーで定義されている制約を使用して、断片配置を自動的に管理します。この自動断片配置により、大きなグリッドの構成が容易になります。また必要に応じて、使用している環境にサーバーを追加することもできます。

注: WebSphere Application Server 環境では、50 を超えるメンバーが入っているコア・グループ・サイズはサポートされません。

デプロイメント・ポリシー XML ファイルは、始動時に eXtreme Scale コンテナーに渡されます。デプロイメント・ポリシーは、ObjectGrid XML ファイルと一緒に使用する必要があります。デプロイメント・ポリシーはコンテナーの始動には不要ですが、使用されることをお勧めします。デプロイメント・ポリシーは、それと一緒に使用される ObjectGrid XML と互換性がある必要があります。デプロイメント・

ポリシー内の objectgridDeployment エレメントごとに、対応する objectGrid が ObjectGrid XML に必要になります。objectgridDeployment 内のマップは、ObjectGrid XML 内の backingMap と整合している必要があります。すべての backingMap は、1 つの mapSet 内のみで参照する必要があります。

以下の例では、companyGridDpReplication.xml ファイルは、対応する companyGrid.xml ファイルとペアになっていることが想定されています。

companyGridDpReplication.xml

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

 <objectgridDeployment objectgridName="CompanyGrid">
  <mapSet name="mapSet1" numberOfPartitions="11"
   minSyncReplicas="1" maxSyncReplicas="1"
   maxAsyncReplicas="0" numInitialContainers="4">
   <map ref="Customer" />
   <map ref="Item" />
   <map ref="OrderLine" />
   <map ref="Order" />
  </mapSet>
 </objectgridDeployment>

</deploymentPolicy>
```

companyGrid.xml

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

 <objectGrids>
  <objectGrid name="CompanyGrid">
   <backingMap name="Customer" />
   <backingMap name="Item" />
   <backingMap name="OrderLine" />
   <backingMap name="Order" />
  </objectGrid>
 </objectGrids>

</objectGridConfig>
```

companyGridDpReplication.xml ファイルには、11 個の区画に分割されている mapSet が 1 つあります。各区画に含めることができる同期複製は 1 つだけです。同期複製の数は、minSyncReplicas 属性および maxSyncReplicas 属性によって決まります。 minSyncReplicas 属性が 1 に設定されているため、書き込みトランザクションを処理するためには、mapSet 内の各区画では、少なくとも 1 つの同期複製が使用可能になっている必要があります。maxSyncReplicas が 1 に設定されているため、各区画の同期複製の数は 1 を超えることはできません。この mapSet 内の区画には、非同期レプリカは含まれていません。

カタログ・サービスは、この ObjectGrid をサポートするために、numInitialContainers 属性に従って、4 つのコンテナーが使用可能になるまで配置を据え置きます。numInitialContainers 属性は、コンテナーが指定数に到達した後は、無視されます。

companyGridDpReplication.xml ファイルはデプロイメント・ポリシーを構成するための一般的な方法を示していますが、デプロイメント・ポリシーによって ObjectGrid を使用環境に配置する方法とその時期をさらに細かく制御することができます。デプロイメント・ポリシー記述子ファイルについて詳しくは、156 ページの『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。

分散トポロジー

分散コヒーレント・キャッシュを使用すると、構成できるパフォーマンス、可用性、およびスケーラビリティーが改善されます。

WebSphere eXtreme Scale は自動的にサーバーのバランスを取ります。 WebSphere eXtreme Scale を再始動しなくても、追加サーバーを組み込むことができます。eXtreme Scale を再始動せずにサーバーを追加できることで、単純なデプロイメントだけでなく、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が可能になります。このデプロイメント・トポロジーは柔軟です。カタログ・サービスを使用すると、キャッシュ全体を除去することなく、サーバーを追加および除去して、リソースを効率的に使用できるようになります。サーバーを追加あるいは除去する場合、 startOgServer コマンドを使用してコンテナー・サーバーを始動し、このコンテナー・サーバーが **-catalogServiceEndPoints** オプションを使用するカタログ・サービスと通信します。分散トポロジーのすべてのクライアントは、Internet Interoperability Object Protocol (IIOP) を介してカタログ・サービスと通信します。すべてのクライアントは、ObjectGrid インターフェースを使用してサーバーと通信できます。

WebSphere eXtreme Scale の動的構成機能を使用すると、簡単にシステムにリソースを追加することができます。コンテナーは、データをホストし、カタログ・サービスは、グリッドのタッチ・ポイントとして機能します。コンテナーは、データの維持を担当します。カタログ・サービスは、最初の接触時に正しい場所への要求の転送、ホスト・コンテナー内でのスペースの割り振り、およびシステム全体のヘルスと可用性の管理を行います。クライアントは、カタログ・サービスに接続して、コンテナー/サーバー・トポロジーの説明を取得してから、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他のサーバーの障害によってサーバー・トポロジーが変更されると、クライアントは、データをホスティングする適切なサーバーに自動的にルーティングされます。

通常、カタログ・サービスは、自身の Java 仮想マシン 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、別のサーバーの他のコンテナーとともに任意の JVM にロードすることもできます。クライアントはどの JVM 内にも存在でき、1 つ以上のサーバーと通信できます。また、コンテナーと同じ JVM 内に存在することも可能です。

既存の Java プロセスまたはアプリケーションにコンテナーを組み込む際には、プログラムでデプロイメント・ポリシーを作成することもできます。詳しくは、eXtreme Scale DeploymentPolicy API を参照してください。

eXtreme Scale クライアントの構成

eXtreme Scale クライアントを要件に応じて構成でき、設定値をオーバーライドすることもできます。

次の方法で、eXtreme Scale クライアントを構成することができます。

- XML 構成
- プログラマチック構成

- Spring Framework 構成
- ニア・キャッシュの使用不可化

以下のプラグインをクライアントにオーバーライドすることができます。

- **ObjectGrid プラグイン**
  - TransactionCallback プラグイン
  - ObjectGridEventListener プラグイン
- **BackingMap プラグイン**
  - Evictor プラグイン
  - MapEventListener プラグイン
  - numberOfBuckets 属性
  - ttlEvictorType 属性
  - timeToLive 属性

XML を使用したクライアントの構成

ObjectGrid XML ファイルを使用して、クライアント・サイドで設定を変更できます。eXtreme Scale クライアントの設定を変更するには、eXtreme Scale サーバーに使用されたファイルに構造が類似している ObjectGrid XML ファイルを作成する必要があります。

以下の XML ファイルがデプロイメント・ポリシー XML ファイルと対になっており、これらのファイルを使用して eXtreme Scale サーバーが始動されたものと想定します。

companyGridServerSide.xml

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

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

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

eXtreme Scale サーバーでは、CompanyGrid は companyGridServerSide.xml ファイルによる定義に従って動作しますデフォルトでは CompanyGrid クライアントの設定

は、サーバーで実行している CompanyGrid の設定と同じです。ただし、設定のいくつかはクライアント・サイドでオーバーライドできます。

クライアント固有の ObjectGrid を作成して、これらの設定のいくつかをオーバーライドします。サーバーの開始に使用された ObjectGrid XML ファイルをコピーし、そのファイルを編集してクライアント・サイドでカスタマイズします。クライアントからプラグインを除去するには、className 属性の値として空ストリングを使用します。既存のプラグインを変更するには、className 属性に新しい値を指定します。プラグインがサポートされているオーバーライド・プラグインの 1 つである場合、そのプラグインをこのファイルに追加することもできます。

クライアントの属性の 1 つを設定するには、新しい値を指定します。

以下の ObjectGrid XML ファイルを使用すると、CompanyGrid クライアントの属性およびプラグインのいくつかを指定できます。

companyGridClientSide.xml

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

    <objectGrids>
        <objectGrid name="CompanyGrid">
            <bean id="TransactionCallback"
                className="com.company.MyClientTxCallback" />
            <bean id="ObjectGridEventListener" className="" />
            <backingMap name="Customer" numberOfBuckets="1429"
                pluginCollectionRef="customerPlugins" />
            <backingMap name="Item" />
            <backingMap name="OrderLine" numberOfBuckets="701"
                timeToLive="800" ttlEvictorType="LAST_ACCESS_TIME" />
            <backingMap name="Order" lockStrategy="PESSIMISTIC"
                pluginCollectionRef="orderPlugins" />
        </objectGrid>
    </objectGrids>

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

companyGridClientSide.xml ファイルは、CompanyGrid クライアントのいくつかの属性およびプラグインをオーバーライドします。クライアント側では、TransactionCallback は、サーバー上で実行している com.company.MyTxCallback ではなく、com.company.MyClientTxCallback になります。className 値が空ストリングであるため、クライアントの ObjectGridEventListener は除去されます。

Customer backingMap の numberOfBuckets は、クライアントでは 1429 に設定されます。Customer backingMap はその Evictor をサーバー構成のまま保持しますが、MapEventListener はクライアントから除去されます。

numberOfBuckets および timeToLive は、OrderLine backingMap のクライアント・サイドで調整されています。

このファイル内の lockStrategy 属性は、指定されている値に関係なく無視されます。この属性は、クライアント・サイドでのオーバーライド用としてはサポートされていません。

companyGridClientSide.xml ファイルを使用して CompanyGrid クライアントを作成するには、ObjectGrid XML ファイルを URL として、ObjectGridManager の接続メソッドの 1 つに渡します。

```
Creating the client for XML

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

クライアントのプログラマチック構成

クライアント・サイドの ObjectGrid 設定をプログラマチックにオーバーライドすることもできます。サーバー・サイド ObjectGrid と同様の構造を持つ ObjectGridConfiguration オブジェクトを作成します。以下のコードで、前にセクションで示された companyGridClientSide.xml を使用して作成されるものと機能的に同等なクライアント・サイド ObjectGrid が構成されます。

```
client-side override programmatically
ObjectGridConfiguration companyGridConfig = ObjectGridConfigFactory
    .createObjectGridConfiguration("CompanyGrid");
Plugin txCallbackPlugin = ObjectGridConfigFactory.createPlugin(
    PluginType.TRANSACTION_CALLBACK, "com.company.MyClientTxCallback");
companyGridConfig.addPlugin(txCallbackPlugin);

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

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

companyGridConfig.addBackingMapConfiguration(customerMapConfig);

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

companyGridConfig.addBackingMapConfiguration(orderLineMapConfig);

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

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

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

overrideMap に組み込まれている ObjectGridConfiguration オブジェクトおよび BackingMapConfiguration オブジェクトのみが、オーバーライドされたプラグインおよび属性をチェックされます。例えば、上記のコードは、OrderLine マップ上のバケットの数をオーバーライドします。ただし、クライアント・サイドの Order マップは、その構成が組み込まれていないので未変更のままです。

Spring Framework でのクライアントの構成

クライアント・サイドの ObjectGrid 設定は、Spring Framework を使用してオーバーライドすることもできます。以下の例の XML ファイルは、ObjectGridConfiguration をビルドし、それをクライアント・サイド設定をオーバーライドするために使用する方法を示しています。この例では、プログラマチック構成で示したのと同じ API が呼び出されます。またこの例は、ObjectGrid XML 構成の例と機能的に同等です。

client configuration with Spring

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

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

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

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

```
    <null />
      </constructor-arg>
    </bean>
</beans>
```

XML ファイルの作成後、そのファイルをロードし、以下のコード・スニペットで ObjectGrid をビルドします。

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

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

詳しくは、 217 ページの『Spring フレームワークとの統合』を参照してください。

ニア・キャッシュの使用不可化

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。

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 ObjectGrid 記述子ファイルで numberOfBuckets 属性を 0 に設定します。

詳しくは、「*管理ガイド*」でマップ・エントリーのロックに関する説明を参照してください。

クライアント無効化メカニズムの使用可能化

分散 WebSphere eXtreme Scale 環境では、optimistic ロック・ストラテジーが使用されているか、ロックが無効にされている場合、クライアント・サイドにはデフォルトでニア・キャッシュがあります。ニア・キャッシュにはそれ独自のローカル・キャッシュ・データがあります。eXtreme Scale クライアントが更新をコミットすると、この更新はクライアントのニア・キャッシュとサーバーに送られます。しかし、他の eXtreme Scale クライアントはこの更新情報を受け取らないので、それらのデータは古くなっていることがあります。

ニア・キャッシュ

アプリケーションは、eXtreme Scale クライアントのこの失効データの問題を認識しておく必要があります。Java Message Service (JMS) ベースの組み込み ObjectGridEventListener クラス com.ibm.websphere.objectgrid.plugins.builtins.JMSObjectGridEventListener を使用して、分散 eXtreme Scale 環境 (eXtreme Scale グリッドと呼ばれます) 内で、クライアント無効化メカニズムを使用可能にすることができます。

クライアント無効化メカニズムは、分散 eXtreme Scale 環境におけるクライアントのニア・キャッシュ内の失効データの問題に対する解決方法です。このメカニズムにより、クライアントのニア・キャッシュはサーバーまたは他のクライアントと同期します。ただし、この JMS ベースのクライアント無効化メカニズムを使用しても、クライアントのニア・キャッシュが即時に更新されるわけではありません。eXtreme Scale ランタイムが更新を公開するときに遅延が発生します。

分散 eXtreme Scale 環境におけるクライアント無効化メカニズムには、次の 2 つのモデルがあります。

- クライアント/サーバー・モデル: このモデルでは、すべてのサーバー・プロセスは、指定された JMS の宛先にすべてのトランザクション変更を公開する publisher ロールにあります。すべてのクライアント・プロセスは受信側ロールにあり、指定された JMS の宛先からすべてのトランザクション変更を受け取ります。
- 二重ロール・モデルとしてのクライアント: このモデルでは、すべてのサーバー・プロセスは JMS の宛先とは無関係です。すべてのクライアント・プロセスは JMS publisher ロールであり、かつ receiver ロールです。クライアントで発生するトランザクション変更は JMS の宛先に公開され、すべてのクライアントがこれらのトランザクション変更を受け取ります。

クライアント無効化メカニズムの使用可能化について詳しくは、150 ページの『JMS イベント・リスナー』を参照してください。

クライアント/サーバー・モデル

クライアント/サーバー・モデルでは、サーバーは JMS publisher ロールにあり、クライアントは JMS receiver ロールにあります。

client-server model XML example

```
<?xml version="1.0" encoding="UTF-8"?>
<?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="AgentObjectGrid">
      <bean id="ObjectGridEventListener"
        className="com.ibm.websphere.objectgrid.plugins.builtins.JMSObjectGridEventListener">
        <property name="invalidationModel" type="java.lang.String" value="CLIENT_SERVER_MODEL" description="" />
        <property name="invalidationStrategy" type="java.lang.String" value="PUSH" description="" />
        <property name="mapsToPublish" type="java.lang.String" value="agent;profile;pessimisticMap" description="" />
        <property name="jms_topicConnectionFactoryJndiName" type="java.lang.String" value="defaultTCF" description="" />
        <property name="jms_topicJndiName" type="java.lang.String" value="defaultTopic" description="" />
        <property name="jms_topicName" type="java.lang.String" value="defaultTopic" description="" />
        <property name="jms_userid" type="java.lang.String" value="" description="" />
        <property name="jms_password" type="java.lang.String" value="" description="" />
        <property name="jndi_properties" type="java.lang.String"
          value="java.naming.factory.initial=org.apache.activemq.jndi.ActiveMQInitialContextFactory;java.naming.provider.url=
    tcp://localhost:61616;connectionFactoryNames=defaultTCF;topic.defaultTopic=defaultTopic"
          description="jndi properties" />
      </bean>

      <backingMap name="agent" readOnly="false" pluginCollectionRef="agent" preloadMode="false"
        lockStrategy="OPTIMISTIC" copyMode="COPY_ON_READ_AND_COMMIT" ttlEvictorType="LAST_ACCESS_TIME"
        timeToLive="28800" />
      <backingMap name="profile" readOnly="false" pluginCollectionRef="profile" preloadMode="false"
        lockStrategy="OPTIMISTIC" copyMode="COPY_ON_READ_AND_COMMIT" ttlEvictorType="LAST_ACCESS_TIME"
        timeToLive="2700" />
      <backingMap name="pessimisticMap" readOnly="false" pluginCollectionRef="pessimisticMap" preloadMode="false"
        lockStrategy="PESSIMISTIC" copyMode="COPY_ON_READ_AND_COMMIT" ttlEvictorType="LAST_ACCESS_TIME"
        timeToLive="2700" />
      <backingMap name="excludedMap1" readOnly="false" pluginCollectionRef="excludedMap1" preloadMode="false"
        lockStrategy="OPTIMISTIC" copyMode="COPY_ON_READ_AND_COMMIT" ttlEvictorType="LAST_ACCESS_TIME"
        timeToLive="2700" />
      <backingMap name="excludedMap2" readOnly="false" pluginCollectionRef="excludedMap2" preloadMode="false"
        lockStrategy="OPTIMISTIC" copyMode="COPY_ON_READ_AND_COMMIT" ttlEvictorType="LAST_ACCESS_TIME"
        timeToLive="2700" />
    </objectGrid>
  </objectGrids>

  <backingMapPluginCollections>
    <backingMapPluginCollection id="agent">
      <bean id="ObjectTransformer" className="com.ibm.ws.objectgrid.test.scenario.AgentObjectTransformer" />
    </backingMapPluginCollection>
    <backingMapPluginCollection id="profile">
      <bean id="ObjectTransformer" className="com.ibm.ws.objectgrid.test.scenario.ProfileObjectTransformer" />
      <bean id="Evictor" className="com.ibm.websphere.objectgrid.plugins.builtins.LRUEvictor">
        <property name="maxSize" type="int" value="2000" description="set max size for LRU evictor" />
        <property name="sleepTime" type="int" value="15" description="evictor thread sleep time" />
        <property name="numberOfLRUQueues" type="int" value="50" description="set number of LRU queues" />
      </bean>
    </backingMapPluginCollection>

    <backingMapPluginCollection id="pessimisticMap" />
    <backingMapPluginCollection id="excludedMap1" />
```

```
    <backingMapPluginCollection id="excludedMap2" />
  </backingMapPluginCollections>

</objectGridConfig>
```

二重ロール・モデルとしてのクライアント

二重ロール・モデルとしてのクライアントでは、各クライアントは JMS publisher ロールと receiver ロールの両方を持っています。クライアントは、コミットされたすべてのトランザクション変更を、指定された JMS 宛先に公開し、コミットされたすべてのトランザクション変更を他のクライアントから受け取ります。このモデルでは、サーバーは JMS とは無関係です。

dual-roles model XML example

```
<?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="AgentObjectGrid">
      <bean id="ObjectGridEventListener"
        className="com.ibm.websphere.objectgrid.plugins.builtins.JMSObjectGridEventListener">
        <property name="invalidationModel" type="java.lang.String" value="CLIENT_AS_DUAL_ROLES_MODEL" description="" />
        <property name="invalidationStrategy" type="java.lang.String" value="PUSH" description="" />
        <property name="mapsToPublish" type="java.lang.String" value="agent;profile;pessimisticMap" description="" />
        <property name="jms_topicConnectionFactoryJndiName" type="java.lang.String" value="defaultTCF" description="" />
        <property name="jms_topicJndiName" type="java.lang.String" value="defaultTopic" description="" />
        <property name="jms_topicName" type="java.lang.String" value="defaultTopic" description="" />
        <property name="jms_userid" type="java.lang.String" value="" description="" />
        <property name="jms_password" type="java.lang.String" value="" description="" />
        <property name="jndi_properties" type="java.lang.String"
          value="java.naming.factory.initial=org.apache.activemq.jndi.ActiveMQInitialContextFactory;java.naming.provider.url=
     tcp://localhost:61616;connectionFactoryNames=defaultTCF;topic.defaultTopic=defaultTopic"
          description="jndi properties" />
      </bean>

      <backingMap name="agent" readOnly="false" pluginCollectionRef="agent" preloadMode="false"
        lockStrategy="OPTIMISTIC" copyMode="COPY_ON_READ_AND_COMMIT" ttlEvictorType="LAST_ACCESS_TIME"
        timeToLive="28800" />
      <backingMap name="profile" readOnly="false" pluginCollectionRef="profile" preloadMode="false"
        lockStrategy="OPTIMISTIC" copyMode="COPY_ON_READ_AND_COMMIT" ttlEvictorType="LAST_ACCESS_TIME"
        timeToLive="2700" />
      <backingMap name="pessimisticMap" readOnly="false" pluginCollectionRef="pessimisticMap" preloadMode="false"
        lockStrategy="PESSIMISTIC" copyMode="COPY_ON_READ_AND_COMMIT" ttlEvictorType="LAST_ACCESS_TIME"
        timeToLive="2700" />
      <backingMap name="excludedMap1" readOnly="false" pluginCollectionRef="excludedMap1" preloadMode="false"
        lockStrategy="OPTIMISTIC" copyMode="COPY_ON_READ_AND_COMMIT" ttlEvictorType="LAST_ACCESS_TIME"
        timeToLive="2700" />
      <backingMap name="excludedMap2" readOnly="false" pluginCollectionRef="excludedMap2" preloadMode="false"
        lockStrategy="OPTIMISTIC" copyMode="COPY_ON_READ_AND_COMMIT" ttlEvictorType="LAST_ACCESS_TIME"
        timeToLive="2700" />
    </objectGrid>
  </objectGrids>

  <backingMapPluginCollections>
    <backingMapPluginCollection id="agent">
      <bean id="ObjectTransformer" className="com.ibm.ws.objectgrid.test.scenario.AgentObjectTransformer" />
    </backingMapPluginCollection>
    <backingMapPluginCollection id="profile">
      <bean id="ObjectTransformer" className="com.ibm.ws.objectgrid.test.scenario.ProfileObjectTransformer" />
      <bean id="Evictor" className="com.ibm.websphere.objectgrid.plugins.builtins.LRUEvictor">
        <property name="maxSize" type="int" value="2000" description="set max size for LRU evictor" />
        <property name="sleepTime" type="int" value="15" description="evictor thread sleep time" />
        <property name="numberOfLRUQueues" type="int" value="50" description="set number of LRU queues" />
      </bean>
    </backingMapPluginCollection>

    <backingMapPluginCollection id="pessimisticMap" />
    <backingMapPluginCollection id="excludedMap1" />
    <backingMapPluginCollection id="excludedMap2" />
  </backingMapPluginCollections>

</objectGridConfig>
```

要求再試行タイムアウトの構成

信頼できるマップがあれば、WebSphere eXtreme Scale に再試行タイムアウトを指定できます。このタイムアウトは、トランザクション要求を再試行するために、トランザクション・タイムアウトと一緒に使用されます。

信頼できるマップを構成するには、2 つの方法があります。ゼロより大きい値に設定された場合、タイムアウト条件が満たされるか、永続的な障害 (DuplicateKeyException 例外など) が起こるまで、要求は再試行されます。値がゼロに設定された場合、フェイル・ファースト・モードが使用され、eXtreme Scale は再試行を行いません。

クライアント・プロパティー・ファイルまたはセッションで、タイムアウト値をミリ秒単位で指定します。セッションは常にクライアント・プロパティー設定に優先します。ランタイム中、トランザクション・タイムアウトが再試行タイムアウトと一緒に使用されます。これは、再試行タイムアウトがトランザクション・タイムアウトを超えないようにするためです。

自動コミット・トランザクション、および非自動コミット・トランザクション (明示的な begin および commit メソッドを使用するトランザクション) がどのように完了するのかには多様なバリエーションがあるため、再試行が有効な例外もさまざまです。

セッション内で呼び出されるトランザクションの場合、CORBA SystemException および eXtreme Scale TargetNotAvailable 例外に対して再試行は有効です。

自動コミット・トランザクションでは、再試行は CORBA SystemException、eXtreme Scale アベイラビリティー例外 (ReplicationVotedToRollbackTransactionException、TargetNotAvailable、AvailabilityException など) の場合に有効です。

詳しくは、「*プログラミング・ガイド*」に記載されているセッションを使用したグリッド内データへのアクセスに関するトピックを参照してください。

アプリケーション障害やその他の永続障害はすぐに再発するので、トランザクションは再試行されません。このような永続障害には DuplicateKeyException や KeyNotFoundException 例外があります。

フェイル・ファースト設定は、いかなる例外についても、再試行なしですべての例外を戻します。

以下のリストは例外を詳細に示しています。

クライアントが再試行を実施する例外

- ReplicationVotedToRollbackTransactionException (自動コミットの場合のみ)
- TargetNotAvailable
- org.omg.CORBA.SystemException
- AvailabilityException (自動コミットの場合のみ)
- LockTimeoutException (自動コミットの場合のみ)
- UnavailableServiceException (自動コミットの場合のみ)

他の例外

- DuplicateKeyException
- KeyNotFoundException
- LoaderException

- TransactionAffinityException
- LockDeadlockException
- OptimisticCollisionException

クライアント・プロパティー・ファイル内での requestRetryTimeout プロパティーの設定

クライアントの requestRetryTimeout 値を設定するには、 210 ページの『クライアント・プロパティー・ファイル』内の requestRetryTimeout プロパティーを追加または変更します。クライアント・プロパティーは、デフォルトでは objectGridClient.properties ファイルです。requestRetryTimeout プロパティーはミリ秒単位で指定されます。 ゼロより大きい値に設定すると、再試行可能な例外が起こったときに要求は再試行されます。値を 0 に設定すると、例外が起こったときに再試行なしで失敗します。デフォルトの動作を使用するには、このプロパティーを除去するか、値を -1 に設定します。

objectGridClient.properties

```
# eXtreme Scale client config
preferLocalProcess = false
preferLocalhost = false
requestRetryTimeout = 30000
```

requestRetryTimeout 値はミリ秒で指定されます。上の例で、設定した値が ObjectGrid インスタンスで使用されると、requestRetryTimeout 値は 30000 ミリ秒または 30 秒です。

ObjectGrid 接続を取得することによる、プログラムでのクライアント・プロパティーの設定

クライアント・プロパティーをプログラマチックに設定するには、まず最初にクライアント・プロパティー・ファイルを作成します。以下の例では、クライアント・プロパティー・ファイルは objectGridClient.properties ファイルです。ObjectGridManager との接続が確立された後、クライアント・プロパティーを設定します。次に、ObjectGrid インスタンスを取得します。クライアントのプロパティーはその ObjectGrid インスタンスで設定されます。クライアントのプロパティーに変更があったら、そのたびに新しい ObjectGrid インスタンスを取得してください。

```
ObjectGridManager manager = ObjectGridManager.instance();
String objectGridName = "testObjectGrid";
URL overRideObjectGridXml = null;
ClientClusterContext ccc = manager.connect("localhost:2809", null, overRideObjectGridXml);
File file= new File("test/objectGridClient.properties");
URL url=file.toURL();
ccc.setClientProperties(objectGridName, url);
ObjectGridManager ogManager = ObjectGridManagerFactory.getObjectGridManager();
ObjectGrid objectGrid = ogManager.getObjectGrid(ctx, objectGridName);
```

自動コミットでのセッションのオーバーライド例

要求再試行タイムアウトをセッションに設定するか、requestRetryTimeout クライアント・プロパティーをオーバーライドするには、Session インターフェースの setRequestRetryTimeout(long) メソッドを呼び出します。

```
Session sessionA = objectGrid.getSession();
sessionA.setRequestRetryTimeout(30000);
ObjectMap mapA = sessionA.getMap("payroll");
String key = "key:" + j;
mapA.insert(key, "valueA");
```

クライアント・プロパティー・ファイルに設定されている値に関係なく、このセッションでは現在 30000 ミリ秒または 30 秒という requestRetryTimeout 値が使用されています。セッション・インターフェースについて詳しくは、セッションを使用したグリッド内データへのアクセスを参照してください。

XML 構成のトラブルシューティング

eXtreme Scale の構成時に予想外の動作が発生する場合があります。

以下のセクションに、発生する可能性のあるさまざまな XML 構成上の問題を示します。

デプロイメント・ポリシーと ObjectGrid XML ファイルの不一致

デプロイメント・ポリシーと ObjectGrid XML ファイルは一致している必要があります。一致する ObjectGrid 名およびマップ名がない場合には、エラーが発生します。

正しくない BackingMap およびマップ参照

ObjectGrid XML ファイル内の backingMap リストがデプロイメント・ポリシー XML ファイル内のマップ参照リストと一致しない場合は、カタログ・サーバーでエラーが発生します。

例えば、以下の ObjectGrid XML ファイルおよびデプロイメント・ポリシー XML ファイルはコンテナー・プロセスを開始する場合に使用されます。デプロイメント・ポリシー・ファイルには、ObjectGrid XML ファイルでリストされているマップ参照よりも多くのマップ参照があります。

ObjectGrid.xml - incorrect example

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

deploymentPolicy.xml - incorrect example

```
<?xml version="1.0" encoding="UTF-8"?>
<deploymentPolicy xmlns:xsi="http://www.w3.org/2001/XMLSchema-instance"
 xsi:schemaLocation="http://ibm.com/ws/objectgrid/deploymentPolicy ../deploymentPolicy.xsd"
 xmlns="http://ibm.com/ws/objectgrid/deploymentPolicy">
    <objectgridDeployment objectgridName="accounting">
        <mapSet name="mapSet1" numberOfPartitions="4" minSyncReplicas="1"
     maxSyncReplicas="2" maxAsyncReplicas="1">
            <map ref="payroll"/>
            <map ref="ledger"/>
        </mapSet>
    </objectgridDeployment>
</deploymentPolicy>
```

メッセージ

IncompatibleDeploymentPolicyException の例外が原因となって、ObjectGridException が発生します。上記の例では、例外は以下のようなメッセージで表示されます。

com.ibm.websphere.objectgrid.IncompatibleDeploymentPolicyException: 会計 objectgridDeployment の mapSet1 mapSet 内にある元帳マップ参照が ObjectGrid XML の有効な backingMap を参照していません。

デプロイメント・ポリシーが backingMap (ObjectGrid XML ファイル内) の欠落したマップ参照である場合、IncompatibleDeploymentPolicyException の例外が原因となり、ObjectGridException も 発生します。以下に例を示します。

com.ibm.websphere.objectgrid.IncompatibleDeploymentPolicyException: 会計 objectGrid には ObjectGrid XML 内に従業員 backingMap が含まれています。このマップは、デプロイメント・ポリシー XML の会計 objectGridDeployment 内にある mapSet で参照されていません。

問題

ObjectGrid XML ファイル の backingMap リストとデプロイメント・ポリシーのマップ参照は一致している必要があります。

解決策

使用中の環境に対してどのリストが正しいかを決定し、正しくないリストが含まれている XML ファイルを訂正します。

正しくない ObjectGrid 名

ObjectGrid の名前は ObjectGrid XML ファイルおよびデプロイメント・ポリシー XML ファイルの両方で参照されます。

メッセージ

IncompatibleDeploymentPolicyException の例外が原因となって、ObjectGridException が発生します。以下に例を示します。

原因: com.ibm.websphere.objectgrid.IncompatibleDeploymentPolicyException: objectGridName が accountin の objectgridDeployment には、ObjectGrid XML 内に対応する objectGrid がありません。

問題

ObjectGrid XML ファイルは ObjectGrid 名のマスター・リストです。デプロイメント・ポリシーに ObjectGrid XML ファイルに含まれていない ObjectGrid 名がある場合、エラーが発生します。

解決策

ObjectGrid 名のスペルを検証します。ObjectGrid XML ファイルまたはデプロイメント・ポリシー XML ファイルに対し、余分な名前を除去するか、または欠落している ObjectGrid 名を追加します。このメッセージ例では、objectGridName が「accounting」のところが、ミススペルのため「accountin」になっています。

属性の XML 値が無効

XML ファイルの一部属性は一定の値にのみ割り当てできます。これらの属性には、スキーマによって列挙された許容値が含まれています。以下のリストには、属性の一部が挙げられています。

- objectGrid エレメントの authorizationMechanism 属性
- backingMap エレメントの copyMode 属性
- backingMap エレメントの lockStrategy 属性
- backingMap エレメントの ttlEvictorType 属性
- property エレメントの type 属性
- objectGrid エレメントの initialState
- backingMap エレメントの evictionTriggers

これら属性の 1 つに無効値が割り当てられていると、XML 妥当性検査は失敗します。以下の XML ファイルの例では、正しくない値 INVALID_COPY_MODE が使用されています。

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

次のメッセージがログに表示されます。

CWOBJ2403E: The XML file is invalid. A problem has been detected with < null > at line 5. The error message is cvc-enumeration-valid: Value 'INVALID_COPY_MODE' is not facet-valid with respect to enumeration '[COPY_ON_READ_AND_COMMIT, COPY_ON_READ, COPY_ON_WRITE, NO_COPY,COPY_TO_BYTES]'. It must be a value from the enumeration.

属性またはタグの欠落

XML ファイルで正しい属性またはタグが欠落している場合には、エラーが発生します。例えば、以下の ObjectGrid XML ファイルでは閉じる < /objectGrid > タグが欠落しています。

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

   </objectGrids>
</objectGridConfig>
```

次のメッセージがログに表示されます。

CWOBJ2403E: The XML file is invalid. A problem has been detected with < null > at line 7. The error message is The end-tag for element type ″objectGrid″ must end with a '>' delimiter.

無効な XML ファイルに関する ObjectGridException が、XML ファイルの名前で発生します。

構文エラー

XML ファイルが、正しくない構文または欠落している構文でフォーマット済みである場合、CWOBJ2403E がログに表示されます。例えば、XML 属性の 1 つで引用符が欠落している場合、以下のメッセージが表示されます。

CWOBJ2403E: The XML file is invalid. A problem has been detected with < null > at line 7. The error message is Open quote is expected for attribute ″maxSyncReplicas″ associated with an element type ″mapSet″.

また、無効な XML ファイルに関する ObjectGridException も発生します。

存在しないプラグイン・コレクションの参照

XML を使用して BackingMap プラグインを定義する場合、backingMap エレメントの pluginCollectionRef 属性は backingMapPluginCollection を参照する必要があります。pluginCollectionRef 属性は backingMapPluginCollection エレメントのいずれか 1 つの ID と一致している必要があります。

メッセージ

pluginCollectionRef 属性が backingMapPluginConfiguration エレメントのどの ID 属性とも一致しない場合は、以下のメッセージまたは同様のメッセージがログに表示されます。

```
[7/14/05 14:02:01:971 CDT] 686c060e XmlErrorHandl E CWOBJ9002E:
This is an English only Error message: Invalid XML file. Line: 14; URI:
null; Message: Key 'pluginCollectionRef' with
value 'bookPlugins' not found for identity constraint of
element 'objectGridConfig'.
```

問題

以下の XML ファイルを使用すると、エラーが生じます。BackingMap の名前 book の pluginCollectionRef 属性は bookPlugins に設定され、1 つの backingMapPluginCollection が collection1 の ID を持つことに注意してください。

```
referencing a non-existent attribute XML - example
<?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="bookstore">
         <backingMap name="book" pluginCollectionRef="bookPlugin" />
      </objectGrid>
   </objectGrids>
   <backingMapPluginCollections>
    <backingMapPluginCollection id="collection1">
     <bean id="Evictor"
       className="com.ibm.websphere.objectgrid.plugins.builtins.LRUEvictor" />
    </backingMapPluginCollection>
   </backingMapPluginCollections>
</objectGridConfig>
```

解決策

問題を修正するには、それぞれの pluginCollectionRef の値が backingMapPluginCollection エレメントのいずれか 1 つの ID に一致するようにしてください。このエラーが発生しないように、pluginCollectionRef の名前を collection1 に変更するだけです。あるいは、既存の backingMapPluginCollection の ID を pluginCollectionRef に一致するように変更するか、または pluginCollectionRef に一致する ID を持つ追加の backingMapPluginCollection を追加してエラーを訂正します。

実装のサポートなしの XML 妥当性検査

IBM Software Development Kit (SDK) バージョン 1.4.2 は、スキーマに対する XML 妥当性検査に使用できるいくつかの Java API for XML Processing (JAXP) 機能の実装を含みます。

この実装を含まない SDK を使用する場合、妥当性検査の試みが失敗する場合があります。この実装を含まない SDK を使用して XML の妥当性検査を行う場合は、Apache Xerces をダウンロードして、その Java アーカイブ (JAR) ファイルをクラスパスに組み込みます。

必要な実装を持たない SDK で XML の妥当性検査をしようとすると、ログに以下のエラーが表示されます。

```
XmlConfigBuil d XML validation is enabled
SystemErr R com.ibm.websphere.objectgrid
SystemErr R at com.ibm.ws.objectgrid.ObjectGridManagerImpl.getObjectGridConfigurations(ObjectGridManagerImpl.java:182)
SystemErr R at com.ibm.ws.objectgrid.ObjectGridManagerImpl.createObjectGrid(ObjectGridManagerImpl.java:309)
SystemErr R at com.ibm.ws.objectgrid.test.config.DocTest.main(DocTest.java:128)
SystemErr R Caused by: java.lang.IllegalArgumentException: No attributes are implemented
SystemErr R at org.apache.crimson.jaxp.DocumentBuilderFactoryImpl.setAttribute(DocumentBuilderFactoryImpl.java:93)
SystemErr R at com.ibm.ws.objectgrid.config.XmlConfigBuilder.<init>XmlConfigBuilder.java:133)
SystemErr R at com.ibm.websphere.objectgrid.ProcessConfigXML$2.runProcessConfigXML.java:99)...
```

使用した SDK は、スキーマに対して XML ファイルの妥当性検査をするのに必要な JAXP 機能の実装を含んでいません。

この問題を回避するために、Xerces をダウンロードして JAR ファイルをクラスパスに組み込むと、XML ファイルを正常に妥当性検査できます。

ローダー

eXtreme Scale ローダー・プラグインによって、eXtreme Scale マップは、通常は同じシステムまたは他のシステム上の永続ストアに保管されるデータのメモリー・キャッシュの働きをすることができます。通常、データベースまたはファイル・システムが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。eXtreme Scale がローダーとどのように対話できるかについての概要は、*製品概要*のキャッシング・シナリオに関する説明

を参照してください。

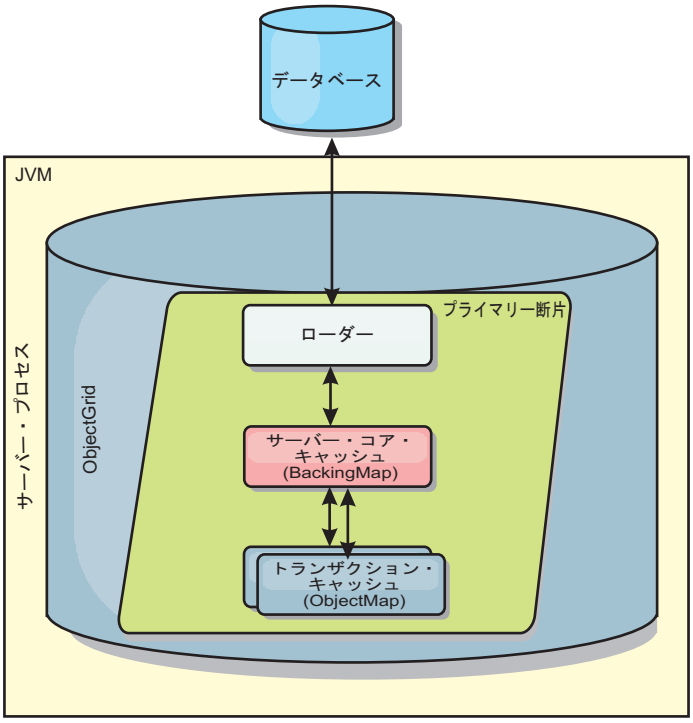


図 1. ローダー

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、「*製品概要*」で JPA ローダーに関する説明を参照してください。

ローダーの使用

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。

詳しくは、*製品概要*のローダーの作成に関するトピックを参照してください。

ローダーのプラグインの XML 構成アプローチ

アプリケーションが提供するローダーは、XML ファイルを使用して接続できます。以下の例は、「MyLoader」ローダーを「map1」バックアップ・マップに接続する方法を示しています。ローダーの className、データベース名と接続詳細、および分

離レベル・プロパティーを指定する必要があります。完全なローダー・クラス名ではなく、プリローダー・クラス名を指定して、プリローダーだけを使用している場合、同じ XML 構造を使用できます。

Loader configuration with XML
```
<?xml version="1.0" encoding="UTF-8" ?>
<objectGridConfig xmlns:xsi="http://www.w3.org/2001/XMLSchema-instance"
 xsi:schemaLocation="http://ibm.com/ws/objectgrid/config ../objectGrid.xsd"
 xmlns="http://ibm.com/ws/objectgrid/config">
<objectGrids>
    <objectGrid name="grid">
        <backingMap name="map1" pluginCollectionRef="map1" lockStrategy="OPTIMISTIC" />
    </objectGrid>
</objectGrids>
<backingMapPluginCollections>
    <backingMapPluginCollection id="map1">
        <bean id="Loader" className="com.myapplication.MyLoader">
            <property name="dataBaseName"
                      type="java.lang.String"
                      value="testdb"
                      description="database name" />
            <property name="isolationLevel"
                      type="java.lang.String"
                      value="read committed"
                      description="iso level" />
        </bean>
    </backingMapPluginCollection>
</backingMapPluginCollections>
</objectGridConfig>
```

ローダーのプログラマチックなプラグイン

プログラマチックな構成は、ローカルなメモリー内グリッドでのみ使用できます。以下のコード・スニペットは、ObjectGrid API を使用してアプリケーションが提供するローダーを map1 のバックアップ・マップに接続する方法を示しています。

Programmatic configuration of a Loader
```
import com.ibm.websphere.objectgrid.ObjectGridManagerFactory;
import com.ibm.websphere.objectgrid.ObjectGridManager;
import com.ibm.websphere.objectgrid.ObjectGrid;
import com.ibm.websphere.objectgrid.BackingMap;
ObjectGridManager ogManager = ObjectGridManagerFactory.getObjectGridManager();
ObjectGrid og = ogManager.createObjectGrid( "grid" );
BackingMap bm = og.defineMap( "map1" );
MyLoader loader = new MyLoader();
loader.setDataBaseName("testdb");
loader.setIsolationLevel("read committed");
bm.setLoader( loader );
```

このスニペットでは、 MyLoader クラスは、com.ibm.websphere.objectgrid.plugins.Loader インターフェースを実装するアプリケーション提供のクラスであることが前提になります。 ObjectGrid の初期化後は、ローダーとバックアップ・マップとの関連付けを変更できないので、呼び出されている ObjectGrid インターフェースの initialize メソッドを起動する前にコードを実行する必要があります。初期化が起こった後に setLoader メソッドが呼び出された場合、IllegalStateException 例外が発生します。

アプリケーションが提供する Loader には、set プロパティーがあります。例では、MyLoader ローダーを使用して、リレーショナル・データベースの表からデータを読み書きします。ローダーにより、データベースの名前と SQL 分離レベルが指定されることが必要です。 MyLoader ローダーには、setDataBaseName メソッドと setIsolationLevel メソッドがあり、アプリケーションはこれらのメソッドを使用してこれら 2 つの Loader プロパティーを設定できます。

ローダーの考慮事項

このトピックでは、ローダーを実装する際の考慮事項について説明します。

プリロードの考慮事項

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。eXtreme Scale がローダーとどのように対話するのかについての概要は、 *製品概要*のインライン・キャッシングのシナリオに関する説明を参照してください。

各バックアップ・マップには、マップのプリロードが非同期的に完了するかどうかを示すために設定できるブール値の preloadMode 属性があります。デフォルトでは、preloadMode 属性は false に設定されており、マップのプリロードが完了するまでバックアップ・マップの初期化が完了しないことを示します。例えば、preloadMap メソッドが戻されるまで、バックアップ・マップの初期化は完了しません。 preloadMap メソッドによりバックエンドから大量のデータが読み取られて、それがマップにロードされる場合は、完了するまでに比較的長い時間を要する場合があります。このような場合、preloadMode 属性を true に設定して、マップの非同期プリロードを使用するようにバックアップ・マップを構成できます。この設定により、バックアップ・マップ初期化コードが preloadMap メソッドを呼び出すスレッドを作成し、マップのプリロードの進行中に、バックアップ・マップの初期化を完了できるようになります。

以下のコード・スニペットは、非同期プリロードが有効になるよう preloadMode 属性を設定する方法を表しています。

```
BackingMap bm = og.defineMap( "map1" );
bm.setPreloadMode( true );
```

preloadMode 属性は、以下の例に示すように、 XML ファイルを使用して設定することもできます。

```
<backingMap name="map1" preloadMode="true" pluginCollectionRef="map1"
 lockStrategy="OPTIMISTIC" />
```

TxID と TransactionCallback インターフェースの使用

Loader インターフェースの get メソッドと batchUpdate メソッドの両方に、get 操作または batchUpdate 操作の実行を必要とするセッション・トランザクションを表す TxID オブジェクトが渡されます。 get メソッドおよび batchUpdate メソッドは、トランザクションごとに複数回呼び出すことが可能です。したがって、ローダーが必要とするトランザクション・スコープのオブジェクトは通常 TxID オブジェクトのスロットに保持されます。ローダーが TxID および TransactionCallback インターフェースをどのように使用するのかを示すため、Java Database Connectivity (JDBC) ローダーが使用されます。

いくつかの ObjectGrid マップを、同じデータベースに格納することも可能です。各マップは独自のローダーを持ち、各ローダーは同一のデータベースに接続する必要があります。同一のデータベースに接続するとき、各テーブルへの変更が同じデータベース・トランザクションのパーツとしてコミットされるようにするため、各ローダーは同じ JDBC 接続を使用しようとします。通常、ローダー実装を作成する同じ担当者が TransactionCallback 実装を作成します。最適な方法は、TransactionCallback インターフェースが拡張されて、ローダーにデータベースが接続され、ローダーが準備済みステートメントのキャッシングを必要とするメソッドを

追加する場合です。この方法論の理由は、ローダーが TransactionCallback インターフェースおよび TxID インターフェースを使用する方法を調査すると明らかになります。

例として、ローダーが、以下のように拡張される TransactionCallback インターフェースを必要とする場合を示します。

```
import java.sql.Connection;
import java.sql.PreparedStatement;
import java.sql.SQLException;
import com.ibm.websphere.objectgrid.TxID;
public interface MyTransactionCallback extends TransactionCallback
{
    Connection getAutoCommitConnection(TxID tx, String databaseName) throws SQLException;
    Connection getConnection(TxID tx, String databaseName, int isolationLevel ) throws SQLException;
    PreparedStatement getPreparedStatement(TxID tx, Connection conn, String tableName, String sql) throws SQLException;
    Collection getPreparedStatementCollection( TxID tx, Connection conn,  String tableName );
}
```

これらの新しいメソッドを使用すると、ローダーの get メソッドおよび batchUpdate メソッドにより、以下のようにして接続が取得されます。

```
import java.sql.Connection;
import java.sql.PreparedStatement;
import java.sql.SQLException;
import com.ibm.websphere.objectgrid.TxID;
private Connection getConnection(TxID tx, int isolationLevel)
{
    Connection conn = ivTcb.getConnection(tx, databaseName, isolationLevel );
    return conn;
}
```

前の例および以下の例において、ivTcb および ivOcb はプリロードの考慮事項のセクションで説明する方法で初期化されたローダーのインスタンス変数です。ivTcb 変数は MyTransactionCallback インスタンスへの参照であり、ivOcb は MyOptimisticCallback インスタンスへの参照です。 databaseName 変数は、ローダーのインスタンス変数であり、バックアップ・マップの初期化中に Loader プロパティーとして設定されています。 isolationLevel 引数は、JDBC がサポートするさまざまな分離レベルに対して定義されている JDBC 接続定数の 1 つです。ローダーがオプティミスティック実装を使用している場合は、get メソッドは通常 JDBC 自動コミット接続を使用してデータをデータベースからフェッチします。この場合、ローダーは以下のように実装される getAutoCommitConnection メソッドを備えている場合があります。

```
import java.sql.Connection;
import java.sql.PreparedStatement;
import java.sql.SQLException;
import com.ibm.websphere.objectgrid.TxID;
private Connection getAutoCommitConnection(TxID tx)
{
    Connection conn = ivTcb.getAutoCommitConnection(tx, databaseName);
    return conn;
}
```

batchUpdate メソッドに以下の switch ステートメントがあれば、再呼び出しします。

```
switch ( logElement.getType().getCode() )
{
    case LogElement.CODE_INSERT:
        buildBatchSQLInsert( tx, key, value, conn );
        break;
    case LogElement.CODE_UPDATE:
        buildBatchSQLUpdate( tx, key, value, conn );
        break;
    case LogElement.CODE_DELETE:
        buildBatchSQLDelete( tx, key, conn );
        break;
}
```

各 buildBatchSQL メソッドは、MyTransactionCallback インターフェースを使用して、準備済みステートメントを取得します。以下は、EmployeeRecord エントリーを更新する SQL UPDATE ステートメントをビルドして、それをバッチ更新用に追加する buildBatchSQLUpdate メソッドを示すコード・スニペットです。

```
private void buildBatchSQLUpdate( TxID tx, Object key, Object value, Connection conn )
throws SQLException, LoaderException
{
    String sql = "update EMPLOYEE set LASTNAME = ?, FIRSTNAME = ?, DEPTNO = ?,
    SEQNO = ?, MGRNO = ? where EMPNO = ?";
    PreparedStatement sqlUpdate = ivTcb.getPreparedStatement( tx, conn,
   "employee", sql );
    EmployeeRecord emp = (EmployeeRecord) value;
    sqlUpdate.setString(1, emp.getLastName());
    sqlUpdate.setString(2, emp.getFirstName());
    sqlUpdate.setString(3, emp.getDepartmentName());
    sqlUpdate.setLong(4, emp.getSequenceNumber());
    sqlUpdate.setInt(5, emp.getManagerNumber());
    sqlUpdate.setInt(6, key);
    sqlUpdate.addBatch();
}
```

batchUpdate ループは、準備済みステートメントをすべてビルドした後で、getPreparedStatementCollection メソッドを呼び出します。このメソッドは、以下のように実装されます。

```
private Collection getPreparedStatementCollection( TxID tx, Connection conn )
{
    return ( ivTcb.getPreparedStatementCollection( tx, conn, "employee" ) );
}
```

アプリケーションによりセッションの commit メソッドが呼び出されると、セッション・コードは、トランザクションによって変更された各マップのローダーに、トランザクションによって変更されたすべての変更がプッシュされた後で、TransactionCallback メソッドの commit メソッドを呼び出します。すべてのローダーにより、必要なすべての接続と準備済みステートメントを取得するために MyTransactionCallback メソッドが使用されたため、TransactionCallback メソッドは、バックエンドが変更をコミットすることを要求するために使用する接続を認識しています。したがって、各ローダーが必要とするメソッドを持つ TransactionCallback インターフェースを拡張することによって、以下の利点が得られます。

- TransactionCallback オブジェクトは、トランザクション・スコープ・データの TxID スロットの使用をカプセル化するので、ローダーは TxID スロットに関する情報を必要としません。ローダーは、ローダーが必要とする機能をサポートするための、 MyTransactionCallback インターフェースを使用する TransactionCallback に追加されるメソッドに関してのみ認識する必要があります。
- TransactionCallback オブジェクトは、2 フェーズ・コミット・プロトコルを回避できるようにするため、同じバックエンドに接続する各ローダー間で、接続の共用が確実に起こるようにすることができます。
- TransactionCallback オブジェクトは、バックエンドへの接続が適切な場合接続に呼び出されたコミットまたはロールバックを通して確実に完了できるようにします。
- TransactionCallback は、トランザクションの完了時にデータベース・リソースのクリーンアップが実行されることを保証します。

- TransactionCallback は、WebSphere Application Server、または他の Java 2 Platform, Enterprise Edition (J2EE) 準拠のアプリケーション・サーバーなどの管理された環境から、管理対象の接続を取得している場合は、隠蔽されます。この利点により、環境が管理されている、いないにかかわらず、同じローダーのコードを使用できます。 TransactionCallback プラグインのみを変更する必要があります。
- TransactionCallback の実装がトランザクション・スコープのデータの TxID スロットを使用する方法の詳細については、TransactionCallback プラグインを参照してください。

OptimisticCallback

これまでに述べたように、ローダーは、並行性制御にオプティミスティック・アプローチを使用する場合があります。その場合、オプティミスティック・アプローチを実装するために、buildBatchSQLUpdate メソッドの例に若干の変更を加える必要があります。オプティミスティック・アプローチを使用する方法は、いくつかあります。行の各更新をバージョン管理するために、タイム・スタンプの列かシーケンス番号のカウンターの列のいずれかを設ける方法が一般的です。従業員のテーブルには、行が更新されるたびに増分するシーケンス番号の列があります。次に、buildBatchSQLUpdate メソッドのシグニチャーを変更して、鍵と値のペアの代わりに LogElement オブジェクトが渡されるようにします。初期バージョンのオブジェクトを取得し、そのバージョンのオブジェクトを更新するには、バックアップ・マップにプラグインされた OptimisticCallback オブジェクトも使用する必要があります。以下は、preloadMap のセクションで説明されている、初期化された ivOcb インスタンス変数を使用する変更済み buildBatchSQLUpdate メソッドの例です。

```
modified batch-update method code exampleprivate void buildBatchSQLUpdate( TxID tx, LogElement le, Connection conn )
 throws SQLException, LoaderException
{
    // Get the initial version object when this map entry was last read
    // or updated in the database.
    Employee emp = (Employee) le.getCurrentValue();
    long initialVersion = ((Long) le.getVersionedValue()).longValue();
    // Get the version object from the updated Employee for the SQL update
    //operation.
    Long currentVersion = (Long)ivOcb.getVersionedObjectForValue( emp );
    long nextVersion = currentVersion.longValue();
    // Now build SQL update that includes the version object in where clause
    // for optimistic checking.
    String sql = "update EMPLOYEE set LASTNAME = ?, FIRSTNAME = ?,
    DEPTNO = ?,SEQNO = ?, MGRNO = ? where EMPNO = ? and SEQNO = ?";
    PreparedStatement sqlUpdate = ivTcb.getPreparedStatement( tx, conn,
   "employee", sql );
    sqlUpdate.setString(1, emp.getLastName());
    sqlUpdate.setString(2, emp.getFirstName());
    sqlUpdate.setString(3, emp.getDepartmentName());
    sqlUpdate.setLong(4, nextVersion );
    sqlUpdate.setInt(5, emp.getManagerNumber());
    sqlUpdate.setInt(6, key);
    sqlUpdate.setLong(7, initialVersion);
    sqlUpdate.addBatch();
}
```

この例は、初期バージョンの値を取得するために LogElement が使用されることを示しています。トランザクションがマップ・エントリーに最初にアクセスするとき、マップから取得した初期の従業員のオブジェクトに関して LogElement が作成されます。この初期 Employee オブジェクトは、OptimisticCallback インターフェースの getVersionedObjectForValue メソッドにも渡され、その結果は LogElement に保存されます。この処理が実行されるのは、初期 Employee オブジェクトへの参照がアプリケーションに与えられ、そのアプリケーションが初期 Employee オブジェクトの状態を変更する何らかのメソッドを呼び出す時の前です。

この例は、Loader が getVersiondObjectForValue メソッドを使用して、現行の更新済み Employee オブジェクトのバージョン・オブジェクトを取得しているところを示しています。Loader インターフェースの batchUpdate メソッドを呼び出す前に、eXtreme Scale は OptimisticCallback インターフェースの updateVersionedObjectForValue メソッドを呼び出して、更新された Employee オブジェクトに対する新しいバージョン・オブジェクトが生成されるようにします。batchUpdate メソッドが ObjectGrid に戻された後、LogElement は新規の初期バージョン・オブジェクトになるように、現行バージョン・オブジェクトで更新されます。アプリケーションは Session の commit メソッドの代わりに、マップ上の flush メソッドを呼び出す可能性があるので、このステップが必要になります。同一のキー用の単一のトランザクションによって、ローダーを複数回呼び出すことは可能です。その理由のため、eXtreme Scale は、従業員テーブル内の行が更新されるたびに LogElement が新しいバージョン・オブジェクトで更新されることを保証しています。

ローダーには、初期バージョンのオブジェクトと次期バージョンのオブジェクトが用意されているので、次期バージョンのオブジェクト値に SEQNO 列を設定し、where 文節で初期バージョンのオブジェクト値を使用する SQL UPDATE ステートメントを実行できます。この方法は、過剰 update ステートメントと呼ばれることがあります。この過剰 update ステートメントを使用することにより、リレーショナル・データベースは、このトランザクションがデータベースからデータを読み取り後データベースを更新するまでの間に、別のトランザクションにより行が変更されていないかどうかを検証できます。別のトランザクションが行を変更していた場合、バッチ更新によって戻されるカウント配列は、このキーに関してゼロ行が更新されたことを示します。ローダーは、SQL update 操作が実際に行を更新したことを検証します。更新されていない場合は、ローダーは com.ibm.websphere.objectgrid.plugins.OptimisticCollisionException 例外を表示して、複数の並行トランザクションがデータベース表の同一行に対して更新を試みたため、batchUpdate メソッドが失敗したことをセッションに通知します。この例外はセッションにロールバックを行わせるので、アプリケーションはトランザクション全体を再試行する必要があります。この方法は、再試行が成功することを予測して行われるため、オプティミスティックと呼ばれます。データがまれにしか変更されないか、または並行トランザクションによる同一行の更新がほとんど試行されない場合は、オプティミスティック・アプローチは実際に適切に機能します。

ローダーが OptimisticCollisionException コンストラクターのキー・パラメーターを使用して、オプティミスティック batchUpdate メソッドの失敗の原因になったキーまたはキーのセットを識別することが重要です。キー・パラメーターには、キー・オブジェクトそのものを使用することもできますし、複数のキーが原因でオプティミスティック更新が失敗した場合は、キー・オブジェクトの配列とすることもできます。 eXtreme Scale は、OptimisticCollisionException コンストラクターの getKey メソッドを使用して、どのマップ・エントリーに失効データが含まれていて、例外の発生原因となったのかを判別します。ロールバック処理の一環として、失効した各マップ・エントリーをマップから除去します。失効したエントリーを除去する必要があるのは、同じキーまたは複数のキーにアクセスする後続のいずれかのトランザクションで、Loader インターフェースの get メソッドが呼び出されて、データベースの現在のデータによってマップ・エントリーが更新されるようにするためです。

ローダーがオプティミスティック・アプローチを実施するそれ以外の方法として、以下のようなものがあります。

- タイム・スタンプまたはシーケンス番号の列を無くします。この場合、OptimisticCallback インターフェースの getVersionObjectForValue メソッドは、単に、値オブジェクト自身をバージョンとして戻します。この方法では、ローダーは初期バージョン・オブジェクトの各フィールドを組み込む where 文節をビルドする必要があります。この方法は効率的ではなく、列タイプのすべてが過剰 SQL UPDATE ステートメントの where 文節での使用に適しているわけではありません。この方法は通常使用しません。
- タイム・スタンプまたはシーケンス番号の列を無くします。ただし、前の方法とは異なり、where 文節にはトランザクションによって変更された値フィールドのみが含まれています。変更されたフィールドを検出する方法の 1 つに、バックアップ・マップのコピー・モードを CopyMode.COPY_ON_WRITE モードに設定することがあります。このコピー・モードは、BackingMap インターフェースの setCopyMode メソッドに渡される値インターフェースを必要とします。BackingMap は、提供される値インターフェースを実装する動的プロキシー・オブジェクトを作成します。このコピー・モードにより、ローダーは com.ibm.websphere.objectgrid.plugins.ValueProxyInfo オブジェクトに各値をキャストできます。 ValueProxyInfo インターフェースには、トランザクションによって変更された属性名のリストをローダーが取得できるメソッドがあります。このメソッドにより、ローダーは属性名の値インターフェースで get メソッドを呼び出して、変更されたデータを取得し、変更された属性のみを設定する SQL UPDATE ステートメントをビルドすることができます。 where 文節は、基本キーの列と変更された各属性の列を持つようにビルドされます。この方法は前の方法よりも効果的ですが、ローダーにさらに多くのコードを書き込む必要があり、さまざまな置換を処理するために、さらに多くの準備済みステートメントのキャッシュが必要になる可能性があります。ただし、トランザクションが、通常ごく一部の属性しか変更しない場合、この制限は問題になりません。
- 一部のリレーショナル・データベースには API があるため、オプティミスティックなバージョン管理に役立つ列データを自動的に保守します。ご使用のデータベースの資料を参照して、この可能性が該当するかどうかを判断してください。

JPA ローダーの構成

Java Persistence API (JPA) ローダーは、JPA を使用してデータベースと対話するローダー・プラグイン実装です。ローダーを構成するには、必要なパラメーターを構成し、XML 構成ファイルを更新する必要があります。

始める前に

- Hibernate または OpenJPA などの JPA 実装が必要です。
- データベースには、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。

このタスクについて

Java Persistence API (JPA) ローダーがどのように機能するかについて詳しくは、*製品概要*にある情報を参照してください。

1. データベースと対話するために JPA が必要とする必須パラメーターを構成します。

   次のパラメーターが必要です。これらのパラメーターは、JPALoader または JPAEntityLoader Bean、および JPATxCallback Bean 内で構成されます。

   - **persistenceUnitName**: パーシスタンス・ユニット名を指定します。このパラメーターは、JPA EntityManagerFactory の作成、および `persistence.xml` ファイル内の JPA エンティティー・メタデータの検索という 2 つの目的のために必要です。この属性は、JPATxCallback Bean に設定されます。
   - **JPAPropertyFactory**: パーシスタンス・プロパティー・マップを作成し、デフォルトのパーシスタンス・プロパティーをオーバーライドするためのファクトリーを指定します。この属性は、JPATxCallback Bean に設定されます。この属性を設定するために、 Spring スタイルの構成が必要です。
   - **entityClassName**: JPA メソッド (EntityManager.persist、EntityManager.find など) を使用する場合に必要なエンティティー・クラス名を指定します。JPALoader にはこのパラメーターが必要ですが、**JPAEntityLoader** ではこのパラメーターはオプションです。 JPAEntityLoader の場合、**entityClassName** パラメーターが構成されていないと、ObjectGrid エンティティー・マップに構成されているエンティティー・クラスが使用されます。 eXtreme Scale EntityManager と JPA プロバイダーに同じクラスを使用する必要があります。この属性は、JPALoader または JPAEntityLoader Bean に設定されます。
   - **preloadPartition**: マップ・プリロードが開始される区画を指定します。プリロード区画がゼロより小さいか、または「区画総数マイナス 1」よりも大きい場合、マップ・プリロードは開始されません。デフォルト値は -1 ですから、デフォルトではプリロードは開始されません。この属性は、JPALoader または JPAEntityLoader Bean に設定されます。

   この 4 つの JPA パラメーターが eXtreme Scale に構成されるほか、JPA エンティティーからキーを取得するため、JPA メタデータが使用されます。JPA メタデータは、アノテーションとして構成するか、 `persistence.xml` ファイル内に指定される `orm.xml` ファイルとして構成できます。これは、eXtreme Scale 構成には含まれません。

2. JPA 構成用に XML ファイルを構成します。

   JPALoader あるいは JPAEntityLoader を構成する場合は、*プログラミング・ガイド* にあるローダー・プラグインに関する情報を参照してください。

   ローダー構成と一緒に、JPATxCallback トランザクション・コールバックを構成します。以下に、JPAEntityLoader と JPATxCallback が構成されている ObjectGrid XML 記述子ファイル (`objectgrid.xml`) の例を示します。

   コールバックを含むローダーの構成 - XML の例

```
<?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="JPAEM" entityMetadataXMLFile="jpaEMD.xml">
        <bean id="TransactionCallback"
           className="com.ibm.websphere.objectgrid.jpa.JPATxCallback">
           <property
              name="persistenceUnitName"
              type="java.lang.String"
              value="employeeEMPU" />
        </bean>
        <backingMap name="Employee" pluginCollectionRef="Employee" />
    </objectGrid>
  </objectGrids>

  <backingMapPluginCollections>
    <backingMapPluginCollection id="Employee">
        <bean id="Loader"
          className="com.ibm.websphere.objectgrid.jpa.JPAEntityLoader">
       <property
               name="entityClassName"
               type="java.lang.String"
               value="com.ibm.ws.objectgrid.jpa.test.entity.Employee"/>
        </bean>
    </backingMapPluginCollection>
  </backingMapPluginCollections>
</objectGridConfig>
```

JPAPropertyFactory を構成する場合は、Spring スタイルの構成を使用する必要があります。以下に示すのは、Spring Bean が eXtreme Scale 構成に使用されるように構成している XML 構成ファイル・サンプル例 (JPAEM_spring.xml) です。

JPA プロパティー・ファクトリーを含むローダーの構成 - XML の例

```
<?xml version="1.0" encoding="UTF-8"?>
<beans xmlns="http://www.springframework.org/schema/beans"
       xmlns:xsi="http://www.w3.org/2001/XMLSchema-instance"
       xmlns:aop="http://www.springframework.org/schema/aop"
       xmlns:tx="http://www.springframework.org/schema/tx"
       xmlns:objectgrid="http://www.ibm.com/schema/objectgrid"
       xsi:schemaLocation="http://www.springframework.org/schema/beans
           http://www.springframework.org/schema/beans/spring-beans-2.0.xsd">

  <objectgrid:JPAEntityLoader id="jpaLoader"
  entityClassName="com.ibm.ws.objectgrid.jpa.test.entity.Employee"/>
  <objectgrid:JPATxCallback id="jpaTxCallback" persistenceUnitName="employeeEMPU" />
</beans>
```

次に Objectgrid.xml 構成 XML ファイルを示します。 ObjectGrid の名前は JPAEM であり、これは、JPAEM_spring.xml Spring 構成ファイルの ObjectGrid 名に一致しています。

JPAEM ローダー構成 - XML の例

```
<?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="JPAEM" entityMetadataXMLFile="jpaEMD.xml">
      <bean id="TransactionCallback"
           className="{spring}jpaTxCallback"/>
        <backingMap name="Employee" pluginCollectionRef="Employee"
                    writeBehind="T4"/>
      </objectGrid>
  </objectGrids>

  <backingMapPluginCollections>
    <backingMapPluginCollection id="Employee">
       <bean id="Loader" className="{spring}jpaLoader" />
    </backingMapPluginCollection>
  </backingMapPluginCollections>
</objectGridConfig>
```

エンティティーには、JPA アノテーションおよび eXtreme Scale エンティティー・マネージャー・アノテーションの両方でアノテーションを付けることができます。各アノテーションには、使用可能な等価 XML があります。そのため、eXtreme Scale は Spring 名前空間を追加しています。この Spring 名前空間サポートを使用してこれらを構成することもできます。Spring 名前空間サポートに関して詳しくは、217 ページの『Spring フレームワークとの統合』を参照してください。

JPA 時間ベース・データ・アップデーターの構成

時間ベース・データベース更新は、ローカルまたは分散 eXtreme Scale 構成の場合、XML を使用して構成することができます。ローカル構成はプログラムでも構成できます。

このタスクについて

Java Persistence API (JPA) 時間ベース・データ・アップデーターがどのように機能するかについて詳しくは、 「*プログラミング・ガイド*」の情報を参照してください。

timeBasedDBUpdate 構成を作成します。

- **XML ファイルを使用:**

  次に示すのは、timeBasedDBUpdate 構成を含む `objectgrid.xml` ファイルの例です。

  JPA 時間ベース・アップデーター - XML の例

```
<?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="changeOG"
            entityMetadataXMLFile="userEMD.xml">
            <backingMap name="user" >
                <timeBasedDBUpdate timestampField="rowChgTs"
                      persistenceUnitName="userderby"
                      entityClass="com.test.UserClass"
                      mode="INVALIDATE_ONLY"
                   />
            </backingMap>
        </objectGrid>
    </objectGrids>
    <backingMapPluginCollections>
</objectGridConfig>
```

  この例では、マップ ″user″ が、時間ベース・データベース更新で構成されています。データベースの更新モードは、INVALIDATE_ONLY、タイム・スタンプ・フィールドは、rowChgTs です。

  分散 ObjectGrid ″changeOG″ が、コンテナー・サーバーで開始されると、時間ベース・データベース更新スレッドが、区画 0 で自動的に始動されます。

- **プログラムで:**

  ローカル ObjectGrid を作成する場合、TimeBasedDBUpdateConfig オブジェクトを作成し、これを BackingMap インスタンスに設定することもできます。

```
public void setTimeBasedDBUpdateConfig(TimeBasedDBUpdateConfig dbUpdateConfig);
```

BackingMap インスタンスへのオブジェクトの設定に関して詳しくは、 API 資料にある BackingMap インターフェースに関する情報を参照してください。

また、com.ibm.websphere.objectgrid.jpa.dbupdate.annotation.Timestamp アノテーションを使用して、エンティティー・クラスのタイム・スタンプ・フィールドにアノテーションを付けることができます。このクラスの値を構成すれば、XML 構成の timestampField を構成する必要はありません。

次のタスク

JPA 時間ベース・データ・アップデーターを開始します。詳しくは、「*プログラミング・ガイド*」で JPA 時間ベース・データ・アップデーターの開始に関する情報を参照してください。

JPA 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。

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。

OpenJPA および Hibernate ObjectGrid キャッシュ・プラグインを使用すると、組み込み、組み込みの区画化、およびリモートの 3 つのトポロジー・タイプが作成できます。

組み込みトポロジー

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 eXtreme Scale 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。非同期複製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。

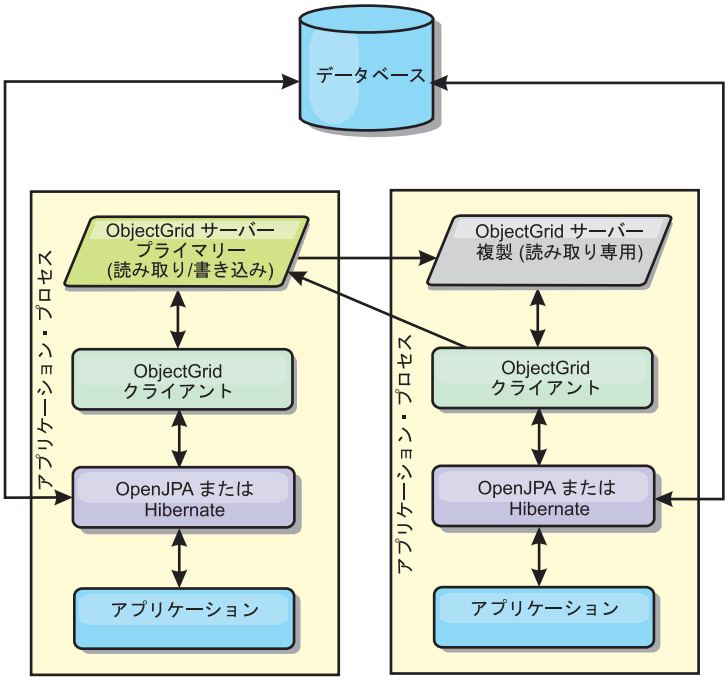


図 2. JPA 組み込みトポロジー

長所:

- すべてのキャッシュ読み取りが非常に速く、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が 1 つのプロセスに送られる。

組み込みの区画化トポロジー

キャッシュ・データの量が多く、1 つのプロセスに収まらない場合、組み込みの区画化トポロジーでは、ObjectGrid 区画を使用して、データを複数のプロセスに分割します。多くのキャッシュ読み取りがリモートになるため、パフォーマンスは組み込みトポロジーほど高くありません。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。

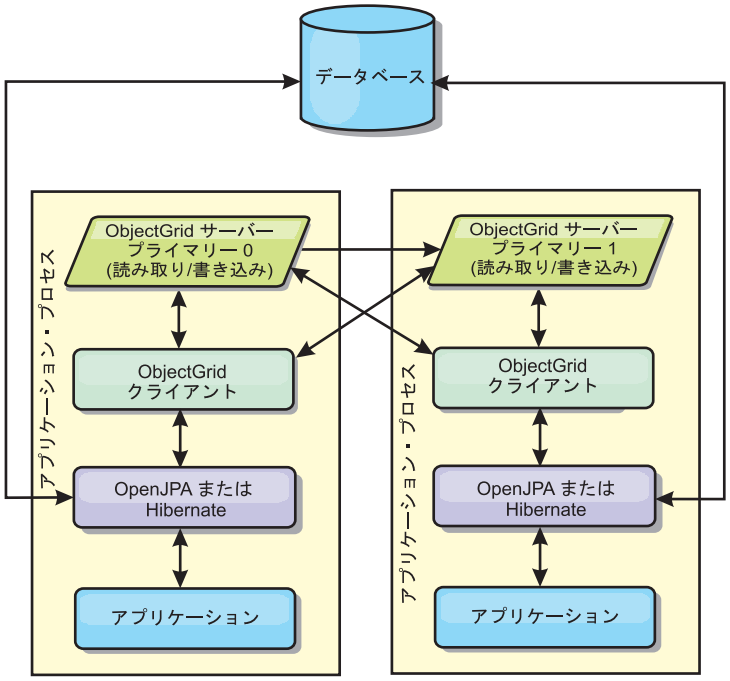


図 3. JPA 組み込みの区画化トポロジー

長所:

- 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、区画数は素数に設定され、各断片が適当な量のメモリーを保管することが望ましいと言えます。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。複製を使用可能にする場合、システム内の Java 仮想マシン 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。

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

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、複製されるように、eXtreme Scaleを構成できます。リモート構成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します。

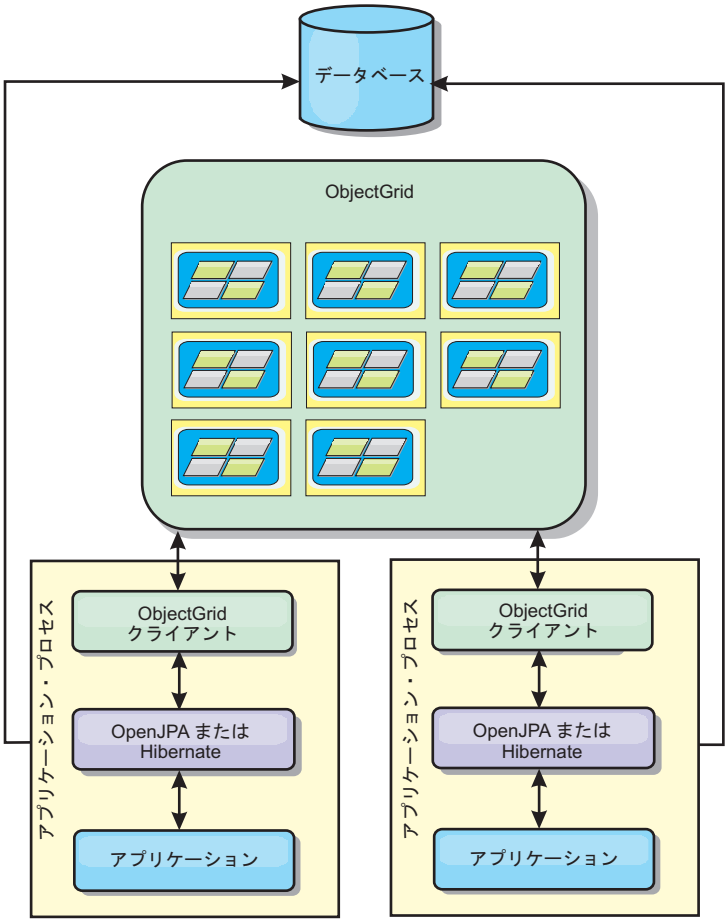


図 4. JPA リモート・トポロジー

長所:

- 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。
- 非常に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。

JPA キャッシュ・プラグイン構成

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 のキャッシュ・プラグインが組み込まれています。

ObjectGrid JPA キャッシュ構成プロパティー

JPA キャッシュ・プラグインは、以下のプロパティーで構成することができます。プロパティーはすべてオプションです。

ObjectGridName

固有の ObjectGrid 名を指定します。デフォルト値は、定義済みのパーシスタンス・ユニット名になります。パーシスタンス・ユニット名が JPA プロバイダーから使用できない場合、生成名が使用されます。

ObjectGridType

ObjectGrid のタイプを指定します。

有効な値:

- **EMBEDDED**: デフォルトおよび推奨構成タイプ。そのデフォルト設定には、NumberOfPartitions=1、ReplicaMode=SYNC、ReplicaReadEnabled=true および MaxNumberOfReplicas=47 が含まれます。 **ReplicaMode** パラメーターは複製モードの設定に、**MaxNumberOfReplicas** パラメーターはレプリカの最大数の設定に使用します。システムに 47 を超える Java 仮想マシンがある場合、**MaxNumberOfReplicas** 値を Java 仮想マシン の数に等しくなるように設定します。
- **EMBEDDED_PARTITION**: システムが、分散システムで大量のデータをキャッシュに入れる必要がある場合に使用するタイプ。デフォルトの区画数は 47 でレプリカ・モードは NONE です。少数の Java 仮想マシン のみを持つ小規模のシステムでは、**NumberOfPartitions** を Java 仮想マシン の数以下に設定します。システムを調整するために、**ReplicaMode**、**NumberOfPartitions**、および **ReplicaReadEnabled** 値を指定できます。
- **REMOTE**: キャッシュは、カタログ・サービスからリモートの分散 ObjectGrid に接続しようとします。

NumberOfPartitions

有効な値: 1 以上キャッシュに使用される区画の数を指定します。このプロパティーは、 ObjectGridType の値が EMBEDDED_PARTITION に設定されている場合に適用されます。デフォルト値は 47 です。EMBEDDED タイプの場合、 **NumberOfPartitions** 値は常に 1 です。

ReplicaMode

有効な値: SYNC/ASYNC/NONEキャッシュを複製にコピーする場合に使用するメソッドを指定します。このプロパティーは、ObjectGridType の値を EMBEDDED または EMBEDDED_PARTITION に設定している場合に適用されます。デフォルト値は、EMBEDDED_PARTITION タイプの場合が NONE で、EMBEDDED タイプの場合は SYNC です。**ReplicaMode** が、EMBEDDED ObjectGridType の場合に NONE に設定されても、EMBEDDED タイプではやはり SYNC の **ReplicaMode** が使用されます。

ReplicaReadEnabled

有効な値: TRUE または FALSE使用可能にする場合、クライアントはレプリカから読み込みます。このプロパティーは、EMBEDDED_PARTITION タイプに適用されます。デフォルト値は、EMBEDDED_PARTITION タイプの場合 FALSE です。 EMBEDDED タイプの場合は、常に **ReplicaReadEnabled** の値は TRUE に設定されます。

MaxUsedMemory

有効な値: TRUE または FALSEメモリーに余裕がなくなった場合にキャッシュ・エントリーの除去を使用可能にします。デフォルト値は TRUE で、JVM ヒープ使用率しきい値が 70 % を超えると、データの除去が開始されます。デフォルトの JVM ヒープ使用率しきい値の比率 (%) は、objectGridServer.properties ファイルの memoryThresholdPercentage プロパティーを設定し、このファイルをクラスパスに配置することによって変更することができます。 Evictor について詳しくは、「*製品概要*」で Evictor に関する情報を参照してください。サーバー・プロパティー・ファイルに関して詳しくは、「*管理ガイド*」を参照してください。

MaxNumberOfReplicas

有効な値: 1 以上キャッシュに使用されるレプリカの最大数を指定します。この値は、EMBEDDED タイプのみに適用されます。この数値は、システム内の Java 仮想マシン 数以上にする必要があります。デフォルト値は 47 です。

NumberOfPartitions、ReplicaMode、ReplicaReadEnabled、および MaxNumberOfReplicas プロパティーは、ObjectGrid デプロイメントの要素となります。NumberOfPartitions、ReplicaMode、および ReplicaReadEnabled は、EMBEDDED_PARTITION タイプに適用されます。
ReplicaMode、MaxNumberOfReplicas は、いずれも EMBEDDED タイプに適用されます。

EMBEDDED および EMBEDDED_PARTITION の考慮事項

組み込み ObjectGrid タイプでは、前述の構成プロパティーを使用して、必要に応じ、ObjectGrid コンテナー・サーバーのセットおよびカタログ・サービスを構成およびデプロイします。コンテナーのライフサイクルは、JPA アプリケーションにバインドされ、アプリケーションのクラスパス内に連結されます。アプリケーションが開始されると、プラグインでは、自動的にカタログ・サービスを検出または開始し、コンテナーを開始して、カタログ・サービスに接続します。それから、プラグインは、クライアント接続を使用して別のアプリケーション・サーバー・プロセスで実行されている ObjectGrid コンテナーおよびそのピアと通信します。

各 JPA エンティティーには、エンティティーのクラス名を使用して独立したバックアップ・マップが割り当てられます。各 BackingMap には、次の属性があります。

- readOnly="false"
- copyKey="false"
- lockStrategy="NONE"
- copyMode="NO_COPY"

注: EMBEDDED または EMBEDDED_PARTITION ObjectGridType を Java SEJava SE 環境で使用する場合、組み込み eXtreme Scale サーバーを停止するため、プログラムの末尾に System.exit(0) メソッドを使用してください。そうでないと、プログラムが反応しなくなったように見えます。

ObjectGridType のデフォルト値

ObjectGridType 値は、ObjectGrid キャッシュがデプロイされるトポロジーを指定します。デフォルトで、パフォーマンスが最善のタイプは EMBEDDED です。以下のセクションでは、ObjectGridType 値について、デフォルトのプロパティーを説明します。

EMBEDDED ObjectGrid JPA キャッシュ・トポロジーのデフォルト

EMBEDDED の ObjectGrid タイプを使用する場合、構成で何の値も指定されない場合、次のデフォルト・プロパティー値が使用されます。

- **ObjectGridName**: パーシスタンス・ユニット名
- **ObjectGridType**: EMBEDDED
- **NumberOfPartitions**: 1 (ObjectGrid タイプが EMBEDDED の場合、変更不可)
- **ReplicaMode**: SYNC
- **ReplicaReadEnabled**: TRUE (ObjectGrid タイプが EMBEDDED の場合、変更不可)
- **MaxUsedMemory**: TRUE
- **MaxNumberOfReplicas**: 47 (分散システムでは、Java 仮想マシン の数以下)

名前が競合しないように、固有の ObjectGridName 値を指定する必要があります。MaxNumberOfReplicas 値は、システム内の合計の Java 仮想マシン 数以上にする必要があります。

REMOTE ObjectGrid キャッシュ・トポロジー

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。

カタログ・サービスの考慮事項および構成

EMBEDDED または EMBEDDED_PARTITION トポロジーで実行する場合、JPA キャッシュ・プラグインでは、必要に応じてアプリケーション・プロセスの 1 つの内部で、単一のカタログ・サービスを自動的に開始します。実稼働環境では、カタログ・サーバー・グリッドを作成する必要があります。カタログ・サービスの定義について詳しくは、 *製品概要* にある高可用性カタログ・サービスに関する情報を参照してください。

WebSphere Application Server プロセス内で実行する場合、JPA キャッシュ・プラグインでは、WebSphere Application Server セルに定義されたカタログ・サービスまた

はカタログ・サービス・グリッドに自動的に接続します。カタログ・サービス・グリッドの定義について詳しくは、*管理ガイド*にあるカタログ・サービスの開始に関する情報を参照してください。

WebSphere Application Server プロセス内でサーバーを実行しない場合、カタログ・サービス・グリッドのホストおよびポートは、objectGridServer.properties というプロパティー・ファイルを使用して指定されます。このファイルは、アプリケーションのクラスパスに保管し、**catalogServiceEndPoints** プロパティーを定義しておく必要があります。カタログ・サービス・グリッドは、アプリケーション・プロセスとは別に開始され、アプリケーション・プロセスが開始される前に開始されなければなりません。

objectGridServer.properties ファイルのフォーマットは次のとおりです。

```
catalogServiceEndPoints=<hostname1>:<port1>,<hostname2>:<port2>
```

Hibernate キャッシュ・プラグイン構成

構成ファイルのプロパティーを設定することにより、Hibernate に対して eXtreme Scale キャッシュを使用可能にすることができます。

設定

persistence.xml ファイルのプロパティーを設定するための構文は、以下のとおりです。

persistence.xml

```
<property name="hibernate.cache.provider_class"
          value="com.ibm.websphere.objectgrid.hibernate.cache.ObjectGridHibernateCacheProvider" />
<property name="hibernate.cache.use_query_cache" value="true"/>
<property name="objectgrid.configuration" value="<property>=<value>,..." />
<property name="objectgrid.hibernate.regionNames" value="<regionName>,.." />
```

hibernate.cfg.xml ファイルのプロパティーを設定するための構文は、以下のとおりです。

hibernate.cfg.xml

```
<property name="cache.provider_class">com.ibm.websphere.objectgrid.
          hibernate.cache.ObjectGridHibernateCacheProvider</property>
<property name="cache.use_query_cache">true</property>
<property name="objectgrid.configuration"><property>=<value>,...</property>
<property name="objectgrid.hibernate.regionNames"><regionName>,...</property>
```

provider_class プロパティーは、com.ibm.websphere.objectgrid.hibernate.cache.ObjectGridHibernateCacheProvider プロパティーです。照会キャッシュを使用可能にするには、use_query_cache プロパティーの値を「true」に設定してください。objectgrid.configuration プロパティーを使用して、eXtreme Scale キャッシュ構成プロパティーを指定します。

名前が競合する可能性をなくすため、固有の ObjectGridName プロパティー値を指定する必要があります。他の eXtreme Scale キャッシュ構成プロパティーはオプションです。

objectgrid.hibernate.regionNames プロパティーはオプションですが、eXtreme Scale キャッシュの初期化後に regionNames 値が定義される場合は指定する必要があります。regionName にマップされるエンティティー・クラスで、そのエンティティー・クラスが persistence.xml ファイルに指定されていないか、または Hibernate マッピング・ファイルに含まれていない例を考えます。さらに、これにエンティティ

ー・アノテーションが設定されているとします。これにより、このエンティティー・クラスの regionName は、eXtreme Scale キャッシュが初期化される場合、クラス・ロード時に解決されます。別の例としては、eXtreme Scale キャッシュ初期化後に実行される Query.setCacheRegion(String regionName) メソッドがあります。こうした状態では、動的決定の可能性があるすべての regionNames を objectgrid.hibernate.regionNames プロパティーに組み入れ、eXtreme Scale キャッシュがすべての regionNames の BackingMaps を準備できるようにします。

以下に示すのは、persistence.xml および hibernate.cfg.xml ファイルの例です。

persistence.xml

```
<persistence-unit name="testPU2">
    <provider>org.hibernate.ejb.HibernatePersistence</provider>
    <class>com.ibm.websphere.objectgrid.jpa.test.Department</class>
    <properties>
      <property name="hibernate.show_sql" value="true" />
      <property name="hibernate.connection.url" value="jdbc:derby:DB_testPU2;create=true" />
      <property name="hibernate.connection.driver_class" value="org.apache.derby.jdbc.EmbeddedDriver" />

      <property name="hibernate.cache.provider_class" value="com.ibm.websphere.objectgrid.
               hibernate.cache.ObjectGridHibernateCacheProvider" />
      <property name="hibernate.cache.use_query_cache" value="true"/>
      <property name="objectgrid.configuration" value="ObjectGridName=myOGName,ObjectGridType=
               EMBEDDED,MaxNumberOfReplicas=4" />
      <property name="objectgrid.hibernate.regionNames" value="queryRegion1, queryRegion2" />
    </properties>
</persistence-unit>
```

hibernate.cfg.xml

```
<hibernate-configuration>
    <session-factory>
        <!-- Database connection settings -->
        <property name="connection.driver_class">org.apache.derby.jdbc.EmbeddedDriver</property>
        <property name="connection.url">jdbc:derby:DB_testPU2;create=true</property>
        <!-- ObjectGrid cache setting-->
        <property name="cache.provider_class">com.ibm.websphere.objectgrid.hibernate.cache.
                 ObjectGridHibernateCacheProvider</property>
        <property name="cache.use_query_cache">true</property>
        <property name="objectgrid.configuration">ObjectGridName=myOGName,
                 ObjectGridType=EMBEDDED,MaxNumberOfReplicas=4 </property>
        <property name="objectgrid.hibernate.regionNames">queryRegion1, queryRegion2</property>

        <mapping resource="com/ibm/websphere/objectgrid/jpa/test/Employee.hbm.xml"/>

    </session-factory>
</hibernate-configuration>
```

ObjectGrid キャッシュに対するデータのプリロード

ObjectGridHibernateCacheProvider クラスの preload メソッドを使用して、特定のエンティティー・クラスの ObjectGrid キャッシュにデータをプリロードできます。

例 1

EntityManagerFactory の使用

```
EntityManagerFactory emf = Persistence.createEntityManagerFactory("testPU");
ObjectGridHibernateCacheProvider.preload("objectGridName", emf, TargetEntity.class, 100, 100);
```

例 2

SessionFactory の使用

```
org.hibernate.cfg.Configuration cfg = new Configuration();
// use addResource, addClass, and setProperty method of Configuration to prepare
// configuration required to create SessionFactor
SessionFactory sessionFactory= cfg.buildSessionFactory();
ObjectGridHibernateCacheProvider.preload("objectGridName", sessionFactory,
TargetEntity.class, 100, 100);
```

注:

1. 分散システムでは、このプリロード・メカニズムは、1 つの Java 仮想マシンからのみ呼び出すことができます。プリロード・メカニズムは、複数の Java 仮想マシン から同時に実行することはできません。
2. プリロードを実行する前に、eXtreme Scale キャッシュを初期化する必要があります。この初期化は、対応するすべての BackingMap が作成されるようにするため、EntityManagerFactory を使用して EntityManager を作成することによって行います。そうしないでプリロードを実行すると、1 つのみのデフォルト BackingMap がすべてのエンティティーをサポートするようにキャッシュが初期化されてしまいます。これは、単一の BackingMap がすべてのエンティティーで共用されることを示しています。

XML による Hibernate キャッシュ構成のカスタマイズ

ほとんどのシナリオでは、キャッシュ・プロパティーを設定すれば十分です。キャッシュによって使用される ObjectGrid をさらにカスタマイズするには、Hibernate ObjectGrid 構成 XML ファイルを、 `persistence.xml` ファイルと同様に `META-INF` ディレクトリーに提供することができます。初期化時に、キャッシュはこれらの XML ファイルを検索し、検出されるとそれらのファイルを処理します。

Hibernate ObjectGrid 構成 XML ファイルには、`hibernate-objectGrid.xml` (ObjectGrid 構成)、`hibernate-objectGridDeployment.xml` (デプロイメント・ポリシー)、および `hibernate-objectGrid-client-override.xml` (クライアント ObjectGrid オーバーライド構成) の 3 つのタイプがあります。構成された eXtreme Scale トポロジーに応じて、これらの 3 つの XML ファイルのいずれか任意のファイルを提供してそのトポロジーをカスタマイズすることができます。

EMBEDDED と EMBEDDED_PARTITION の両方のタイプの場合、3 つの XML ファイルの任意のいずれかを提供し、ObjectGrid、デプロイメント・ポリシー、およびクライアント ObjectGrid オーバーライド構成をカスタマイズできます。

REMOTE ObjectGrid の場合、キャッシュは動的 ObjectGrid を作成しません。キャッシュは、カタログ・サービスからクライアント・サイドの ObjectGrid を取得するだけです。`hibernate-objectGrid-client-override.xml` ファイルを提供して、クライアント ObjectGrid オーバーライド構成をカスタマイズできるだけです。

1. **ObjectGrid 構成**: `META-INF/hibernate-objectGrid.xml` ファイルを使用します。このファイルは、EMBEDDED および EMBEDDED_PARTITION タイプの両方に対して ObjectGrid 構成をカスタマイズする場合に使用されます。REMOTE タイプの場合、このファイルは無視されます。デフォルトでは、各エンティティー・クラスには、regionName が関連付けられ (エンティティー・クラス名にデフォルト設定)、その regionName が、ObjectGrid 構成内の regionName として名付けられた BackingMap 構成にマップされます。例えば、com.mycompany.Employee エンティティー・クラスには、com.mycompany.Employee BackingMap とデフォルト設定される regionName が関連付けられます。デフォルト BackingMap 構成は、readOnly=″false″、copyKey=″false″、lockStrategy=″NONE″、および `copyMode="NO_COPY"` です。一部の BackingMap を選択された構成でカスタマイズできます。予約キーワード「ALL_ENTITY_MAPS」を使用すれば、`hibernate-objectGrid.xml` ファイルにリストされた他のカスタマイズ・マップ

を除くすべてのマップを示すことができます。この hibernate-objectGrid.xml ファイルにリストされていない BackingMap は、デフォルト構成を使用します。

2. **ObjectGridDeployment 構成**: META-INF/hibernate-objectGridDeployment.xml ファイルを使用します。このファイルは、デプロイメント・ポリシーのカスタマイズに使用されます。デプロイメント・ポリシーをカスタマイズする場合、hibernate-objectGridDeployment.xml が提供されている場合は、デフォルトのデプロイメント・ポリシーは廃棄されます。すべてのデプロイメント・ポリシー属性値は、この提供された hibernate-objectGridDeployment.xml ファイルから得られます。
3. **クライアント・オーバーライド ObjectGrid 構成**: META-INF/hibernate-objectGrid-client-override.xml ファイルを使用します。このファイルは、クライアント・サイド ObjectGrid のカスタマイズに使用されます。デフォルトでは、ObjectGrid キャッシュは、ニア・キャッシュを使用不可にするデフォルト・クライアント・オーバーライド構成を適用します。アプリケーションがニア・キャッシュを必要とする場合、アプリケーションはこのファイルを提供し、numberOfBuckets="xxx" を指定することができます。デフォルトのクライアント・オーバーライドでは、numberOfBuckets="0" を設定し、ニア・キャッシュを使用不可にします。ニア・キャッシュは、hibernate-objectGrid-client-override.xml ファイルによって numberOfBuckets 属性の値をゼロより大きい値に再設定すれば、アクティブにすることができます。 hibernate-objectGrid-client-override.xml ファイルの機能は、 hibernate-objectGrid.xml に似ています。このファイルは、デフォルトのクライアント ObjectGrid オーバーライド構成をオーバーライドまたは拡張します。

Hibernate ObjectGrid XML ファイルの例

Hibernate ObjectGrid XML ファイルは、パーシスタンス・ユニットの構成に基づいて作成する必要があります。

パーシスタンス・ユニットの構成を表す persistence.xml ファイルの例を以下に示します。

persistence.xml

```
<persistence xmlns="http://java.sun.com/xml/ns/persistence" xmlns:xsi="http://www.w3.org/2001/XMLSchema-instance"
  version="1.0">
  <persistence-unit name="AnnuityGrid">
    <provider>org.hibernate.ejb.HibernatePersistence</provider>

    <class>com.ibm.wssvt.acme.annuity.common.bean.jpa.AnnuityPersistebleObject</class>
    <class>com.ibm.wssvt.acme.annuity.common.bean.jpa.Annuity</class>
    <class>com.ibm.wssvt.acme.annuity.common.bean.jpa.FixedAnnuity</class>
    <class>com.ibm.wssvt.acme.annuity.common.bean.jpa.EquityAnnuity</class>
    <class>com.ibm.wssvt.acme.annuity.common.bean.jpa.Payout</class>
    <class>com.ibm.wssvt.acme.annuity.common.bean.jpa.Rider</class>
    <class>com.ibm.wssvt.acme.annuity.common.bean.jpa.Payor</class>
    <class>com.ibm.wssvt.acme.annuity.common.bean.jpa.Person</class>
    <class>com.ibm.wssvt.acme.annuity.common.bean.jpa.AnnuityHolder</class>
    <class>com.ibm.wssvt.acme.annuity.common.bean.jpa.Contact</class>
    <class>com.ibm.wssvt.acme.annuity.common.bean.jpa.Address</class>

    <exclude-unlisted-classes>true</exclude-unlisted-classes>

    <properties>
      <property name="hibernate.show_sql" value="false" />
      <property name="hibernate.connection.url" value="jdbc:db2:Annuity" />
      <property name="hibernate.connection.driver_class" value="com.ibm.db2.jcc.DB2Driver" />
      <property name="hibernate.default_schema" value="EJB30" />

      <!-- Cache -->
      <property name="hibernate.cache.provider_class"
        value="com.ibm.websphere.objectgrid.hibernate.cache.ObjectGridHibernateCacheProvider" />
      <property name="hibernate.cache.use_query_cache" value="true" />
      <property name="objectgrid.configuration" value="ObjectGridType=EMBEDDED,
               ObjectGridName=Annuity, MaxNumberOfReplicas=4" />
```

```
    </properties>
  </persistence-unit>

</persistence>
```

以下に示すのは、この persistence.xml ファイルに適合する hibernate-objectGrid.xml ファイルです。

hibernate-objectGrid.xml

```
<?xml version="1.0" encoding="UTF-8"?>
<objectGridConfig xmlns:xsi="http://www.w3.org/2001/XMLSchema-instance"
  xsi:schemaLocation="http://ibm.com/ws/objectgrid/config ../objectGrid.xsd"
  xmlns="http://ibm.com/ws/objectgrid/config">
  <objectGrids>
    <objectGrid name="Annuity">
      <backingMap name="com.ibm.wssvt.acme.annuity.common.bean.jpa.Annuity" readOnly="false" copyKey="false"
                  lockStrategy="NONE" copyMode="NO_COPY" evictionTriggers="MEMORY_USAGE_THRESHOLD"
                  pluginCollectionRef="com.ibm.wssvt.acme.annuity.common.bean.jpa.Annuity" />
      <backingMap name="defaultCacheMap" readOnly="false" copyKey="false"
                  lockStrategy="NONE" copyMode="NO_COPY" evictionTriggers="MEMORY_USAGE_THRESHOLD"
                  pluginCollectionRef="defaultCacheMap" />
      <backingMap name="com.ibm.wssvt.acme.annuity.common.bean.jpa.Payor" readOnly="false" copyKey="false"
                  lockStrategy="NONE" copyMode="NO_COPY" evictionTriggers="MEMORY_USAGE_THRESHOLD"
                  pluginCollectionRef="com.ibm.wssvt.acme.annuity.common.bean.jpa.Payor" />
      <backingMap name="com.ibm.wssvt.acme.annuity.common.bean.jpa.Contact" readOnly="false" copyKey="false"
                  lockStrategy="NONE" copyMode="NO_COPY" evictionTriggers="MEMORY_USAGE_THRESHOLD"
                  pluginCollectionRef="com.ibm.wssvt.acme.annuity.common.bean.jpa.Contact" />
      <backingMap name="com.ibm.wssvt.acme.annuity.common.bean.jpa.Person" readOnly="false" copyKey="false"
                  lockStrategy="NONE" copyMode="NO_COPY" evictionTriggers="MEMORY_USAGE_THRESHOLD"
                  pluginCollectionRef="com.ibm.wssvt.acme.annuity.common.bean.jpa.Person" />
      <backingMap name="com.ibm.wssvt.acme.annuity.common.bean.jpa.Rider" readOnly="false" copyKey="false"
                  lockStrategy="NONE" copyMode="NO_COPY" evictionTriggers="MEMORY_USAGE_THRESHOLD"
                  pluginCollectionRef="com.ibm.wssvt.acme.annuity.common.bean.jpa.Rider" />
      <backingMap name="com.ibm.wssvt.acme.annuity.common.bean.jpa.Payout" readOnly="false" copyKey="false"
                  lockStrategy="NONE" copyMode="NO_COPY" evictionTriggers="MEMORY_USAGE_THRESHOLD"
                  pluginCollectionRef="com.ibm.wssvt.acme.annuity.common.bean.jpa.Payout" />
      <backingMap name="org.hibernate.cache.UpdateTimestampsCache" readOnly="false" copyKey="false"
                  lockStrategy="NONE" copyMode="NO_COPY" evictionTriggers="MEMORY_USAGE_THRESHOLD"
                  pluginCollectionRef="org.hibernate.cache.UpdateTimestampsCache" />
      <backingMap name="org.hibernate.cache.StandardQueryCache" readOnly="false" copyKey="false"
                  lockStrategy="NONE" copyMode="NO_COPY" evictionTriggers="MEMORY_USAGE_THRESHOLD"
                  pluginCollectionRef="org.hibernate.cache.StandardQueryCache" />
    </objectGrid>
  </objectGrids>
  <backingMapPluginCollections>
    <backingMapPluginCollection id="com.ibm.wssvt.acme.annuity.common.bean.jpa.Annuity">
      <bean id="Evictor" className="com.ibm.websphere.objectgrid.plugins.builtins.LRUEvictor" >
      </bean>
    </backingMapPluginCollection>
    <backingMapPluginCollection id="defaultCacheMap">
      <bean id="Evictor" className="com.ibm.websphere.objectgrid.plugins.builtins.LRUEvictor" >
      </bean>
    </backingMapPluginCollection>
    <backingMapPluginCollection id="com.ibm.wssvt.acme.annuity.common.bean.jpa.Payor">
      <bean id="Evictor" className="com.ibm.websphere.objectgrid.plugins.builtins.LRUEvictor" >
      </bean>
    </backingMapPluginCollection>
    <backingMapPluginCollection id="com.ibm.wssvt.acme.annuity.common.bean.jpa.Contact">
      <bean id="Evictor" className="com.ibm.websphere.objectgrid.plugins.builtins.LRUEvictor" >
      </bean>
    </backingMapPluginCollection>
    <backingMapPluginCollection id="com.ibm.wssvt.acme.annuity.common.bean.jpa.Person">
      <bean id="Evictor" className="com.ibm.websphere.objectgrid.plugins.builtins.LRUEvictor" >
      </bean>
    </backingMapPluginCollection>
    <backingMapPluginCollection id="com.ibm.wssvt.acme.annuity.common.bean.jpa.Rider">
      <bean id="Evictor" className="com.ibm.websphere.objectgrid.plugins.builtins.LRUEvictor" >
      </bean>
    </backingMapPluginCollection>
    <backingMapPluginCollection id="com.ibm.wssvt.acme.annuity.common.bean.jpa.Payout">
      <bean id="Evictor" className="com.ibm.websphere.objectgrid.plugins.builtins.LRUEvictor" >
      </bean>
    </backingMapPluginCollection>
    <backingMapPluginCollection id="org.hibernate.cache.UpdateTimestampsCache">
      <bean id="Evictor" className="com.ibm.websphere.objectgrid.plugins.builtins.LRUEvictor" >
      </bean>


    </backingMapPluginCollection>
    <backingMapPluginCollection id="org.hibernate.cache.StandardQueryCache">
      <bean id="Evictor" className="com.ibm.websphere.objectgrid.plugins.builtins.LRUEvictor" >
      </bean>
    </backingMapPluginCollection>
  </backingMapPluginCollections>
</objectGridConfig>
```

注: org.hibernate.cache.UpdateTimestampsCache、
org.hibernate.cache.StandardQueryCache および defaultCacheMap マップが必要です。

以下に示すのは、この persistence.xml に適合する hibernate-objectGridDeployment.xml ファイルです。

hibernate-objectGridDeployment.xml

```
<?xml version="1.0" encoding="UTF-8" ?>
<deploymentPolicy xmlns:xsi="http://www.w3.org/2001/XMLSchema-instance"
  xsi:schemaLocation="http://ibm.com/ws/objectgrid/deploymentPolicy ../deploymentPolicy.xsd"
  xmlns="http://ibm.com/ws/objectgrid/deploymentPolicy">
  <objectgridDeployment objectgridName="Annuity">
    <mapSet name="MAPSET_Annuity" numberOfPartitions="1" numInitialContainers="1" minSyncReplicas="0"
          maxSyncReplicas="4" maxAsyncReplicas="0" replicaReadEnabled="true">
      <map ref="com.ibm.wssvt.acme.annuity.common.bean.jpa.Annuity" />
      <map ref="defaultCacheMap" />
      <map ref="com.ibm.wssvt.acme.annuity.common.bean.jpa.Payor" />
      <map ref="com.ibm.wssvt.acme.annuity.common.bean.jpa.Contact" />
      <map ref="com.ibm.wssvt.acme.annuity.common.bean.jpa.Person" />
      <map ref="com.ibm.wssvt.acme.annuity.common.bean.jpa.Rider" />
      <map ref="com.ibm.wssvt.acme.annuity.common.bean.jpa.Payout" />
      <map ref="org.hibernate.cache.UpdateTimestampsCache" />
      <map ref="org.hibernate.cache.StandardQueryCache" />
    </mapSet>
  </objectgridDeployment>
</deploymentPolicy>
```

注: org.hibernate.cache.UpdateTimestampsCache、
org.hibernate.cache.StandardQueryCache および defaultCacheMap マップが必要です。

REMOTE ObjectGrid タイプのキャッシュ用の外部システム

REMOTE ObjectGrid タイプでキャッシュを構成する場合、外部 eXtreme Scale システムをセットアップする必要があります。外部システムをセットアップするためには、persistence.xml ファイルに基づく ObjectGrid および ObjectGridDeployment 構成 XML ファイルの両方が必要になります。 Hibernate ObjectGrid XML ファイルの例のセクションに記載された Hibernate ObjectGrid および ObjectGridDeployment 構成 XML ファイルを使用しても外部 eXtreme Scale システムをセットアップできます。

外部 ObjectGrid システムには、カタログ・サービスおよび ObjectGrid サーバー・プロセスの両方があります。コンテナー・サーバーを始動する前に、カタログ・サービスを始動する必要があります。詳しくは、「*管理ガイド*」内のスタンドアロン eXtreme Scale サーバーの開始およびコンテナー・プロセスの開始に関する説明を参照してください。

トラブルシューティング

1. **CacheException: ObjectGrid サーバーを取得できません。**

   EMBEDDED または EMBEDDED_PARTITION ObjectGridType の場合、キャッシュは、eXtreme Scale ランタイムからサーバー・インスタンスを取得しようとします。Java Platform, Standard Edition 環境では、組み込みカタログ・サービスの eXtreme Scale サーバーが始動されます。組み込みカタログ・サービスでは、ポート 2809 を listen しようとします。このポートが別のプロセスによって使用中の場合、このエラーが発生します。外部カタログ・サービス・エンドポイントが、例えば objectGridServer.properties ファイルにより指定されている場合、ホスト名またはポートの指定に誤りがあると、このエラーが発生します。

2. **CacheException: 構成済みの REMOTE ObjectGrid に対して REMOTE ObjectGrid を取得できません。objectGridName = [ObjectGridName]、PU 名 = [persistenceUnitName]**

   このエラーは、キャッシュが、指定されたカタログ・サービス・エンドポイントから ObjectGrid を取得できない場合に発生します。一般的には、このエラーは、ホスト名またはポートに誤りがあることが原因です。

3. **CacheException: 2 つの PU [persistenceUnitName_1, persistenceUnitName_2] を EMBEDDED ObjectGridType の同一の ObjectGridName [ObjectGridName] では構成できません。**

   多数のパーシスタンス・ユニットを持つ構成があり、これらのユニットのキャッシュが同じ ObjectGrid 名および EMBEDDED ObjectGridType で構成されている場合、この例外が発生します。これらのパーシスタンス・ユニット構成は、同じか、または異なる persistence.xml ファイルに入れることができます。ObjectGridType が EMBEDDED の場合、各パーシスタンス・ユニットについて ObjectGrid 名が固有であることを確認する必要があります。

4. **CacheException: REMOTE ObjectGrid [ObjectGridName] に必要な BackingMaps [mapName_1, mapName_2,...] が含まれていません。**

   REMOTE ObjectGrid タイプの場合に、取得されたクライアント・サイド ObjectGrid にパーシスタンス・ユニットのキャッシュをサポートする完全なエンティティー BackingMap がないと、この例外が発生します。例えば、パーシスタンス・ユニットの構成に 5 つのエンティティー・クラスがリストされているが、取得された ObjectGrid には 2 つの BackingMap しかない場合などです。取得された ObjectGrid に 10 の BackingMap があったとしても、この 10 の BackingMap に必要な 5 つのエンティティー BackingMap のいずれかが検出されないと、やはりこの例外が発生します。

OpenJPA キャッシュ・プラグイン構成

WebSphere eXtreme Scale では、OpenJPA に対して DataCache 実装および QueryCache 実装の両方を提供しています。OpenJPA ObjectGrid キャッシュ、あるいは省略して ObjectGrid キャッシュは、DataCache 実装および QueryCache 実装の両方に共通の用語です。

設定

eXtreme Scale キャッシュは、persistence.xml ファイルに openjpa.DataCache および openjpa.QueryCache 構成プロパティーを設定することにより、OpenJPA に対して使用可能または使用不可にします。このプロパティー設定の構文は以下のとおりです。

```
<property name="openjpa.DataCache"
          value="<object_grid_datacache_class(<property>=<value>,...)"/>
<property name="openjpa.QueryCache"
          value="<object_grid_querycache_class(<property>=<value>,...)"/>
```

DataCache、QueryCache のいずれも eXtreme Scale キャッシュ・プロパティーを利用して、パーシスタンス・ユニットで使用されるキャッシュを構成することができます。

eXtreme Scale キャッシュ設定に加え、openjpa.RemoteCommitProvider プロパティーを sjvm に設定する必要があります。

```
<property name="openjpa.RemoteCommitProvider" value="sjvm"/>
```

各エンティティー・クラスの @DataCache アノテーションで指定されたタイムアウト値は、BackingMap に渡され、そこに各エンティティーがキャッシュされます。ただし、@DataCache アノテーションで指定された名前値は、eXtreme Scale キャッシュで無視されます。完全修飾エンティティー・クラス名が、キャッシュ・マップ名になります。

OpenJPA StoreCache および QueryCache の pin および unpin メソッドはサポートされておらず、何の機能も果たしません。

ObjectGrid キャッシュ・クラスのプロパティー・リストに ObjectGridName プロパティー、ObjectGridType プロパティー、その他の単純なデプロイメント・ポリシー関連のプロパティーを指定すれば、キャッシュ構成をカスタマイズすることができます。以下に例を示します。

```
<property name="openjpa.DataCache"
          value="com.ibm.websphere.objectgrid.openjpa.ObjectGridDataCache(
          ObjectGridName=BasicTestObjectGrid,ObjectGridType=EMBEDDED,
          maxNumberOfReplicas=4)"/>
<property name="openjpa.QueryCache"
          value="com.ibm.websphere.objectgrid.openjpa.ObjectGridQueryCache()"/>
<property name="openjpa.RemoteCommitProvider" value="sjvm"/>
```

DataCache および QueryCache 構成は、互いに独立しています。いずれかの構成を使用可能にすることができます。ただし、両方の構成を使用可能にした場合、QueryCache 構成では、DataCache 構成と同じ構成が使用され、その構成プロパティーは廃棄されます。

XML による OpenJPA キャッシュ構成のカスタマイズ

ほとんどのシナリオでは、eXtreme Scale キャッシュ・プロパティーを設定すれば十分です。キャッシュによって使用される ObjectGrid をさらにカスタマイズするには、OpenJPA ObjectGrid 構成 XML ファイルを persistence.xml ファイルと同様に META-INF ディレクトリーに提供することができます。キャッシュの初期化時に、ObjectGrid キャッシュはこれらの XML ファイルを検索し、検出されるとそれらのファイルを処理しようとします。

OpenJPA ObjectGrid 構成 XML ファイルには、openjpa-objectGrid.xml (ObjectGrid 構成)、 openjpa-objectGridDeployment.xml (デプロイメント・ポリシー)、および openjpa-objectGrid-client-override.xml (クライアント ObjectGrid オーバーライド構成) の 3 つのタイプがあります。構成された ObjectGrid タイプに応じて、これらの 3 つの XML ファイルのいずれか任意のファイルを提供して ObjectGrid をカスタマイズすることができます。

EMBEDDED と EMBEDDED_PARTITION のタイプの場合、3 つの XML ファイルの任意のいずれかを提供し、ObjectGrid、デプロイメント・ポリシー、およびクライアント ObjectGrid オーバーライド構成をカスタマイズできます。

REMOTE ObjectGrid の場合、ObjectGrid キャッシュは動的 ObjectGrid を作成しません。代わりにキャッシュは、カタログ・サービスからクライアント・サイドの ObjectGrid を取得するだけです。openjpa-objectGrid-client-override.xml ファイルを提供して、クライアント ObjectGrid オーバーライド構成をカスタマイズできるだけです。

1. **ObjectGrid 構成**: META-INF/openjpa-objectGrid.xml ファイルを使用します。このファイルは、EMBEDDED および EMBEDDED_PARTITION タイプの両方に対して ObjectGrid 構成をカスタマイズする場合に使用されます。REMOTE タイプの場合、このファイルは無視されます。デフォルトでは、各エンティティー・クラスは、ObjectGrid 構成内のエンティティー・クラス名として名付けられた独自の BackingMap 構成にマップされます。例えば、com.mycompany.Employee エンティティー・クラスは、com.mycompany.Employee BackingMap にマップされます。デフォルト BackingMap 構成は、readOnly="false"、copyKey="false"、lockStrategy="NONE"、および copyMode="NO_COPY" です。一部の BackingMap を選択された構成でカスタマイズできます。ALL_ENTITY_MAPS 予約キーワードを使用すれば、openjpa-objectGrid.xml ファイルにリストされた他のカスタマイズ・マップを除くすべてのマップを示すことができます。この openjpa-objectGrid.xml ファイルにリストされていない BackingMap は、デフォルト構成を使用します。カスタマイズされた BackingMaps が BackingMaps 属性またはプロパティーを指定しておらず、それらの属性がデフォルト構成で指定されている場合は、デフォルト構成の属性値が適用されます。例えば、エンティティー・クラスが timeToLive=30 でアノテーションを付けられている場合、そのエンティティーのデフォルトの BackingMap 構成には timeToLive=30 が設定されます。カスタム openjpa-objectGrid.xml ファイルにもその BackingMap が含まれているが、timeToLive 値を指定していない場合、カスタマイズされた BackingMap には、デフォルトで、timeToLive=30 値が設定されます。openjpa-objectGrid.xml ファイルは、デフォルト構成をオーバーライドまたは拡張しようとします。
2. **ObjectGridDeployment 構成**: META-INF/openjpa-objectGridDeployment.xml ファイルを使用します。このファイルは、デプロイメント・ポリシーのカスタマイズに使用されます。デプロイメント・ポリシーをカスタマイズする場合、openjpa-objectGridDeployment.xml ファイルが提供されている場合は、デフォルトのデプロイメント・ポリシーは廃棄されます。デプロイメント・ポリシー属性値は、すべて openjpa-objectGridDeployment.xml ファイルから提供されます。
3. **クライアント・オーバーライド ObjectGrid 構成**: META-INF/openjpa-objectGrid-client-override.xml ファイルを使用します。このファイルは、クライアント・サイド ObjectGrid のカスタマイズに使用されます。デフォルトでは、ObjectGrid キャッシュは、ニア・キャッシュを使用不可にするデフォルト・クライアント・オーバーライド ObjectGrid 構成を適用します。アプリケーションがニア・キャッシュを必要とする場合、アプリケーションはこのファイルを提供し、numberOfBuckets="xxx" を指定することができます。デフォルトのクライアント・オーバーライドでは、numberOfBuckets="0" を設定し、ニア・キャッシュを使用不可にします。ニア・キャッシュは、openjpa-objectGrid-client-override.xml ファイルによって numberOfBuckets の値をゼロより大きい値に再設定すれば、アクティブにすることができます。 openjpa-objectGrid-client-override.xml ファイルの機能は、 openjpa-objectGrid.xml ファイルに似てい

ます。このファイルは、デフォルトのクライアント ObjectGrid オーバーライド構成をオーバーライドまたは拡張します。

OpenJPA ObjectGrid XML ファイルの例

OpenJPA ObjectGrid XML ファイルは、パーシスタンス・ユニットの構成に基づいて作成する必要があります。

パーシスタンス・ユニットの構成を表す persistence.xml ファイルの例を以下に示します。

persistence.xml

```
<persistence xmlns="http://java.sun.com/xml/ns/persistence"
             xmlns:xsi="http://www.w3.org/2001/XMLSchema-instance"
  version="1.0">
  <persistence-unit name="AnnuityGrid">
    <provider>org.apache.openjpa.persistence.PersistenceProviderImpl</provider>
    <class>com.ibm.wssvt.acme.annuity.common.bean.jpa.AnnuityPersistebleObject</class>
    <class>com.ibm.wssvt.acme.annuity.common.bean.jpa.Annuity</class>
    <class>com.ibm.wssvt.acme.annuity.common.bean.jpa.FixedAnnuity</class>
    <class>com.ibm.wssvt.acme.annuity.common.bean.jpa.EquityAnnuity</class>
    <class>com.ibm.wssvt.acme.annuity.common.bean.jpa.Payout</class>
    <class>com.ibm.wssvt.acme.annuity.common.bean.jpa.Rider</class>
    <class>com.ibm.wssvt.acme.annuity.common.bean.jpa.Payor</class>
    <class>com.ibm.wssvt.acme.annuity.common.bean.jpa.Person</class>
    <class>com.ibm.wssvt.acme.annuity.common.bean.jpa.AnnuityHolder</class>
    <class>com.ibm.wssvt.acme.annuity.common.bean.jpa.Contact</class>
    <class>com.ibm.wssvt.acme.annuity.common.bean.jpa.Address</class>
    <exclude-unlisted-classes>true</exclude-unlisted-classes>

    <properties>
    <!-- Database setting -->


    <!--  enable cache -->
      <property name="openjpa.DataCache"
        value="com.ibm.websphere.objectgrid.openjpa.ObjectGridDataCache(objectGridName=Annuity,
               objectGridType=EMBEDDED, maxNumberOfReplicas=4)" />
      <property name="openjpa.RemoteCommitProvider" value="sjvm" />
      <property name="openjpa.QueryCache"
                value="com.ibm.websphere.objectgrid.openjpa.ObjectGridQueryCache()" />
    </properties>
  </persistence-unit>
</persistence>
```

以下に示すのは、この persistence.xml ファイルに適合する openjpa-objectGrid.xml ファイルです。

openjpa-objectGrid.xml

```
<?xml version="1.0" encoding="UTF-8"?>
<?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="Annuity">
      <backingMap name="com.ibm.wssvt.acme.annuity.common.bean.jpa.Annuity" readOnly="false" copyKey="false"
                  lockStrategy="NONE" copyMode="NO_COPY" evictionTriggers="MEMORY_USAGE_THRESHOLD"
                  pluginCollectionRef="com.ibm.wssvt.acme.annuity.common.bean.jpa.Annuity" />
      <backingMap name="com.ibm.wssvt.acme.annuity.common.bean.jpa.Address" readOnly="false" copyKey="false"
                  lockStrategy="NONE" copyMode="NO_COPY" evictionTriggers="MEMORY_USAGE_THRESHOLD"
                  pluginCollectionRef="com.ibm.wssvt.acme.annuity.common.bean.jpa.Address" />
      <backingMap name="com.ibm.wssvt.acme.annuity.common.bean.jpa.Payor" readOnly="false" copyKey="false"
                  lockStrategy="NONE" copyMode="NO_COPY" evictionTriggers="MEMORY_USAGE_THRESHOLD"
                  pluginCollectionRef="com.ibm.wssvt.acme.annuity.common.bean.jpa.Payor" />
      <backingMap name="com.ibm.wssvt.acme.annuity.common.bean.jpa.Person" readOnly="false" copyKey="false"
                  lockStrategy="NONE" copyMode="NO_COPY" evictionTriggers="MEMORY_USAGE_THRESHOLD"
                  pluginCollectionRef="com.ibm.wssvt.acme.annuity.common.bean.jpa.Person" />
      <backingMap name="com.ibm.wssvt.acme.annuity.common.bean.jpa.Contact" readOnly="false" copyKey="false"
                  lockStrategy="NONE" copyMode="NO_COPY" evictionTriggers="MEMORY_USAGE_THRESHOLD"
                  pluginCollectionRef="com.ibm.wssvt.acme.annuity.common.bean.jpa.Contact" />
      <backingMap name="com.ibm.wssvt.acme.annuity.common.bean.jpa.AnnuityPersistebleObject"
                  readOnly="false" copyKey="false"
                  lockStrategy="NONE" copyMode="NO_COPY" evictionTriggers="MEMORY_USAGE_THRESHOLD"
                  pluginCollectionRef="com.ibm.wssvt.acme.annuity.common.bean.jpa.AnnuityPersistebleObject" />
      <backingMap name="com.ibm.wssvt.acme.annuity.common.bean.jpa.Rider" readOnly="false" copyKey="false"
                  lockStrategy="NONE" copyMode="NO_COPY" evictionTriggers="MEMORY_USAGE_THRESHOLD"
                  pluginCollectionRef="com.ibm.wssvt.acme.annuity.common.bean.jpa.Rider" />
      <backingMap name="com.ibm.wssvt.acme.annuity.common.bean.jpa.Payout" readOnly="false" copyKey="false"
                  lockStrategy="NONE" copyMode="NO_COPY" evictionTriggers="MEMORY_USAGE_THRESHOLD"
```

```
                pluginCollectionRef="com.ibm.wssvt.acme.annuity.common.bean.jpa.Payout" />
      <backingMap name="ObjectGridQueryCache" readOnly="false" copyKey="false"
                lockStrategy="NONE" copyMode="NO_COPY" pluginCollectionRef="ObjectGridQueryCache"
                evictionTriggers="MEMORY_USAGE_THRESHOLD"  />
    </objectGrid>
  </objectGrids>
  <backingMapPluginCollections>
    <backingMapPluginCollection id="com.ibm.wssvt.acme.annuity.common.bean.jpa.Annuity">
      <bean id="ObjectTransformer"
            className="com.ibm.ws.objectgrid.openjpa.ObjectGridPCDataObjectTransformer" />
      <bean id="Evictor" className="com.ibm.websphere.objectgrid.plugins.builtins.LRUEvictor" >
      </bean>
    </backingMapPluginCollection>
    <backingMapPluginCollection id="com.ibm.wssvt.acme.annuity.common.bean.jpa.Address">
      <bean id="ObjectTransformer"
            className="com.ibm.ws.objectgrid.openjpa.ObjectGridPCDataObjectTransformer" />
      <bean id="Evictor" className="com.ibm.websphere.objectgrid.plugins.builtins.LRUEvictor" >
      </bean>
    </backingMapPluginCollection>
    <backingMapPluginCollection id="com.ibm.wssvt.acme.annuity.common.bean.jpa.Payor">
      <bean id="ObjectTransformer"
            className="com.ibm.ws.objectgrid.openjpa.ObjectGridPCDataObjectTransformer" />
      <bean id="Evictor" className="com.ibm.websphere.objectgrid.plugins.builtins.LRUEvictor" >
      </bean>
    </backingMapPluginCollection>
    <backingMapPluginCollection id="com.ibm.wssvt.acme.annuity.common.bean.jpa.Person">
      <bean id="ObjectTransformer"
            className="com.ibm.ws.objectgrid.openjpa.ObjectGridPCDataObjectTransformer" />
      <bean id="Evictor" className="com.ibm.websphere.objectgrid.plugins.builtins.LRUEvictor" >
      </bean>
    </backingMapPluginCollection>
    <backingMapPluginCollection id="com.ibm.wssvt.acme.annuity.common.bean.jpa.Contact">
      <bean id="ObjectTransformer"
            className="com.ibm.ws.objectgrid.openjpa.ObjectGridPCDataObjectTransformer" />
      <bean id="Evictor" className="com.ibm.websphere.objectgrid.plugins.builtins.LRUEvictor" >
      </bean>
    </backingMapPluginCollection>
    <backingMapPluginCollection
            id="com.ibm.wssvt.acme.annuity.common.bean.jpa.AnnuityPersistebleObject">
      <bean id="ObjectTransformer"
            className="com.ibm.ws.objectgrid.openjpa.ObjectGridPCDataObjectTransformer" />
      <bean id="Evictor" className="com.ibm.websphere.objectgrid.plugins.builtins.LRUEvictor" >
      </bean>
    </backingMapPluginCollection>
    <backingMapPluginCollection id="com.ibm.wssvt.acme.annuity.common.bean.jpa.Rider">
      <bean id="ObjectTransformer"
            className="com.ibm.ws.objectgrid.openjpa.ObjectGridPCDataObjectTransformer" />
      <bean id="Evictor" className="com.ibm.websphere.objectgrid.plugins.builtins.LRUEvictor" >
      </bean>
    </backingMapPluginCollection>
    <backingMapPluginCollection id="com.ibm.wssvt.acme.annuity.common.bean.jpa.Payout">
      <bean id="ObjectTransformer"
            className="com.ibm.ws.objectgrid.openjpa.ObjectGridPCDataObjectTransformer" />
      <bean id="Evictor" className="com.ibm.websphere.objectgrid.plugins.builtins.LRUEvictor" >
      </bean>
    </backingMapPluginCollection>
    <backingMapPluginCollection id="ObjectGridQueryCache">
      <bean id="MapIndexPlugin" className="com.ibm.websphere.objectgrid.plugins.index.HashIndex" >
          <property name="Name" type="java.lang.String"
                    value="QueryCacheKeyIndex" description="name of index"/>
          <property name="POJOKeyIndex" type="boolean" value="true" description="POJO Key Index "/>
      </bean>
      <bean id="Evictor" className="com.ibm.websphere.objectgrid.plugins.builtins.LRUEvictor" >
      </bean>
    </backingMapPluginCollection>
  </backingMapPluginCollections>
</objectGridConfig>
```

注:

1. 各エンティティーは、完全修飾エンティティー・クラス名として名付けられた BackingMap にマップされます。
2. エンティティー・クラスが、継承の階層にある場合、子クラスは親の BackingMap にマップされます。継承の階層は単一の BackingMap を共用します。
3. QueryCache のサポートには ObjectGridQueryCache マップが必要です。
4. 各エンティティー・マップの backingMapPluginCollection には、com.ibm.ws.objectgrid.openjpa.ObjectGridPCDataObjectTransformer クラスを使用する ObjectTransformer がなければなりません。

5. ObjectGridQueryCache マップの backingMapPluginCollection には、サンプルに示されているように QueryCacheKeyIndex と名付けられたキー索引がなければなりません。
6. 各マップで、Evictor はオプションです。

以下に示すのは、この persistence.xml ファイルに適合する openjpa-objectGridDeployment.xml ファイルです。

openjpa-objectGridDeployment.xml

```
<?xml version="1.0" encoding="UTF-8" ?>
<deploymentPolicy xmlns:xsi="http://www.w3.org/2001/XMLSchema-instance"
  xsi:schemaLocation="http://ibm.com/ws/objectgrid/deploymentPolicy ../deploymentPolicy.xsd"
  xmlns="http://ibm.com/ws/objectgrid/deploymentPolicy">
  <objectgridDeployment objectgridName="Annuity">
    <mapSet name="MAPSET_Annuity" numberOfPartitions="1" numInitialContainers="1"
            minSyncReplicas="0" maxSyncReplicas="4" maxAsyncReplicas="0"
            replicaReadEnabled="true">
      <map ref="com.ibm.wssvt.acme.annuity.common.bean.jpa.Annuity" />
      <map ref="com.ibm.wssvt.acme.annuity.common.bean.jpa.Address" />
      <map ref="com.ibm.wssvt.acme.annuity.common.bean.jpa.Payor" />
      <map ref="com.ibm.wssvt.acme.annuity.common.bean.jpa.Person" />
      <map ref="com.ibm.wssvt.acme.annuity.common.bean.jpa.Contact" />
      <map ref="com.ibm.wssvt.acme.annuity.common.bean.jpa.AnnuityPersistebleObject" />
      <map ref="com.ibm.wssvt.acme.annuity.common.bean.jpa.Rider" />
      <map ref="com.ibm.wssvt.acme.annuity.common.bean.jpa.Payout" />
      <map ref="ObjectGridQueryCache" />
    </mapSet>
  </objectgridDeployment>
</deploymentPolicy>
```

注: QueryCache のサポートには ObjectGridQueryCache マップが必要です。

REMOTE ObjectGrid タイプのキャッシュ用の外部システム

REMOTE ObjectGrid タイプでキャッシュを構成する場合、外部システムをセットアップする必要があります。外部システムをセットアップするためには、persistence.xml ファイルに基づく ObjectGrid および ObjectGridDeployment 構成 XML ファイルの両方が必要になります。OpenJPA ObjectGrid XML ファイルの例のセクションに記載された OpenJPA ObjectGrid および ObjectGridDeployment 構成 XML ファイルを使用しても外部 eXtreme Scale システムをセットアップできます。

外部 eXtreme Scale システムには、カタログ・サービスおよびコンテナー・サーバー・プロセスの両方があります。コンテナー・サーバーを始動する前に、カタログ・サーバーを始動する必要があります。

トラブルシューティング

1. **CacheException: ObjectGrid サーバーを取得できません。**

   EMBEDDED または EMBEDDED_PARTITION ObjectGridType の場合、eXtreme Scale キャッシュは、ランタイムからサーバー・インスタンスを取得しようとします。Java Platform, Standard Edition 環境では、組み込みカタログ・サービスを持つ eXtreme Scale サーバーが始動されます。組み込みカタログ・サービスでは、ポート 2809 を listen しようとします。このポートが別のプロセスによって使用中の場合、このエラーが発生します。外部カタログ・サービス・エンドポイントが、例えば objectGridServer.properties ファイルにより指定されている場合、ホスト名またはポートの指定に誤りがあると、このエラーが発生します。

2. **CacheException: 構成済みの REMOTE ObjectGrid に対して REMOTE ObjectGrid を取得できません。objectGridName = [ObjectGridName]、PU 名 = [persistenceUnitName]**

   このエラーは、キャッシュが、指定されたカタログ・サービス・エンドポイントから ObjectGrid を取得できない場合に発生します。一般的には、このエラーは、ホスト名またはポートに誤りがあることが原因です。

3. **CacheException: 2 つの PU [persistenceUnitName_1, persistenceUnitName_2] を EMBEDDED ObjectGridType の同一の ObjectGridName [ObjectGridName] では構成できません。**

   多数のパーシスタンス・ユニット構成があり、これらのユニットの eXtreme Scale キャッシュが同じ ObjectGrid 名および EMBEDDED ObjectGridType で構成されている場合、この例外が発生します。これらのパーシスタンス・ユニット構成は、同じまたは異なる `persistence.xml` ファイルに入れることができます。ObjectGridType が EMBEDDED の場合、各パーシスタンス・ユニットについて ObjectGrid 名が固有であることを確認する必要があります。

4. **CacheException: REMOTE ObjectGrid [ObjectGridName] に必要な BackingMaps [mapName_1, mapName_2,...] が含まれていません。**

   REMOTE ObjectGrid タイプの場合に、取得されたクライアント・サイド ObjectGrid に、パーシスタンス・ユニットのキャッシュをサポートする完全なエンティティー BackingMap がないと、この例外が発生します。例えば、パーシスタンス・ユニット構成に 5 つのエンティティー・クラスがリストされているが、取得された ObjectGrid には 2 つの BackingMap しかない場合などです。取得された ObjectGrid に 10 の BackingMap があったとしても、この 10 の BackingMap に必要な 5 つのエンティティー BackingMap のいずれかが検出されないと、やはりこの例外が発生します。

注: OpenJPA eXtreme Scale キャッシュは、パフォーマンス強化のためにデータ・フォーマットを変更しました。eXtreme Scale を L2 キャッシュとして使用するよう構成された OpenJPA アプリケーションをホスティングするシステムは、WebSphere eXtreme Scale バージョン 7.0 へのマイグレーションの前に停止される必要があります。

後書きキャッシング・サポート

後書きキャッシングを使用して、バックエンド・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は、ローダー・プラグインへの更新情報をキューに入れます。

概要

後書きキャッシングでは、ローダー・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。

BackingMap での後書き構成により、ローダーとマップ間のスレッドが作成されます。その後は、ローダーが、BackingMap.setWriteBehind() メソッドの構成の設定値に従い、スレッド経由でデータ要求を委任します。 eXtreme Scale トランザクションが eXtreme Scale マップでエントリーの挿入、更新、または除去を行うと、これらのレコードのそれぞれに 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、LogElement オブジェクトの挿入、更新、および削除タイプのみを実際のローダーに送信します。他のすべてのタイプの LogElement オブジェクト (例えば、EVICT タイプ) は無視されます。

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。

- バックエンドの障害の分離: 後書きキャッシングは、バックエンドの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、引き続きトランザクションを eXtreme Scale に送ることができます。バックエンドが回復すると、キュー・マップ内のデータがバックエンドにプッシュされます。
- バックエンド負荷の軽減: 後書きローダーは、更新をキー単位でマージするため、キュー・マップ内にはキーごとにマージされた更新が 1 つしか存在しません。このマージにより、バックエンドに対する更新の数が削減されます。
- 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。

ObjectGrid 記述子 XML

eXtreme Scale 記述子 XML ファイルを使用して eXtreme Scale を構成する場合、backingMap タグで writeBehind 属性を設定すると、後書きローダーが使用可能になります。以下に例を示します。

```
<objectGrid name="library" >
     <backingMap name="book" writeBehind="T300;C900" pluginCollectionRef="bookPlugins"/>
```

前記の例では、「book」バックアップ・マップの後書きサポートがパラメーター「T300;C900」で使用可能になります。

後書き属性は、最大更新時間または最大キー更新数、あるいはその両方を指定します。後書きパラメーターの形式は以下のとおりです。

```
write-behind attribute  ::= <defaults> | <update time> | <update key count> | <update time> ";" <update key count>
update time  ::= "T" <positive integer>
update key count  ::= "C" <positive integer>
defaults  ::= "" {table}
```

ローダーに対する更新は、以下のいずれかのイベントが発生すると実行されます。

1. 最終更新以降、最大更新時間 (秒数) を経過する
2. キュー・マップ内の更新キーの数が最大更新キー数に達する。

これらのパラメーターはヒントにすぎません。実際の更新数および更新時間は、これらのパラメーターの近似値になります。ただし、実際の更新数または更新時間がパラメーターで定義されたものと同じであることを保証するものではありません。また、更新時間の範囲内で、最大 2 回まで更新が発生した後、最初の後書き更新が発生することがあります。これは、すべての区画でデータベースに同時にヒットしないよう、eXtreme Scale が更新開始時間をランダム化するためです。

前記の例の T300;C900 では、最終更新以降 300 秒が経過するか、900 個のキーが更新保留状態になると、ローダーはデータをバックエンドに書き込みます。

デフォルトの更新時間は 300 秒で、デフォルトの更新キー数は 1000 です。

下の表に、後書き属性の例がいくつか示されています。

注: 後書きローダーを空ストリング `writeBehind=""` として構成すると、後書きローダーはデフォルト値を使用して使用可能になります。したがって、後書きサポートを使用可能にしたくない場合、writeBehind 属性を指定しないでください。

表 6. いくつかの後書きオプション

属性値	時間
T100	更新時間は 100 秒で、更新キー数は 1000 (デフォルト値) です。
C2000	更新時間は 300 秒 (デフォルト値) で、更新キー数は 2000 です。
T300;C900	更新時間は 300 秒で、更新キー数は 900 です。
""	更新時間は 300 秒 (デフォルト値) で、更新キー数は 1000 (デフォルト値) です。

後書きサポートをプログラマチックに使用可能化

ローカルのメモリー内の eXtreme Scale 用のバックアップ・マップをプログラムで作成する場合、以下のメソッドを BackingMap インターフェースで使用すると、後書きサポートを使用可能または使用不可にできます。

```
public void setWriteBehind(String writeBehindParam);
```

setWriteBehind メソッドの使用法について詳しくは、69 ページの『BackingMap インターフェース』を参照してください。

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と、eXtreme Scale トランザクションにバックエンド・トランザクションが組み込まれます。eXtreme Scale 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。

後書きサポートが使用可能になっていると、eXtreme Scale 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終了します。 eXtreme Scale トランザクションとバックエンド・トランザクションは、互いに切り離されます。

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 eXtreme Scale トランザクションで異なるマップに対して更新されたデータ

は、異なるバックエンド・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。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。

更新の失敗

eXtreme Scale トランザクションが、バックエンド・トランザクションの開始前に終了するため、トランザクションの実行が誤った形の正常実行になる可能性があります。例えば、バックアップ・マップには存在しないが、バックエンドに存在するエントリーを eXtreme Scale トランザクションに挿入すると、重複キーになることになりますが、eXtreme Scale トランザクションは成功します。ただし、後書きスレッドがそのオブジェクトをバックエンドに挿入するトランザクションは失敗し、重複キーの例外が発生します。

このような障害の処理方法については、 130 ページの『失敗した後書き更新の処理』を参照してください。

キュー・マップのロックの振る舞い

トランザクションでのもう 1 つの振る舞い上の大きな相違は、ロックの振る舞いです。eXtreme Scale は、PESSIMISTIC、OPTIMISITIC、および NONE という 3 つの異なるロック戦略をサポートします。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、PESSIMISTIC ロック・ストラテジーを使用します。キュー・マップでのロックを獲得するために以下の 2 つの異なるタイプの操作が存在します。

- eXtreme Scale トランザクションがコミットするか、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) が発生すると、トランザクションは、キュー・マップ内のキーを読み取り、キーに対して S ロックを設定します。
- eXtreme Scale トランザクションがコミットすると、トランザクションはキーに対する 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振る舞いのため、ロックの振る舞い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、ロックの振る舞い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 ロックが獲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 ロックが獲得されます。
- 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 ロックが獲得されます。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

WebSphere eXtreme Scale 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途中でサーバーに障害があると、一部のバックエンド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。詳細は、 API 資料を参照してください。

ローダー障害

ローダー・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、定期的にデータ変更をローダーにプッシュしようとします。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存在することを WebSphere eXtreme Scale 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 例外または OptimisticCollisionException 例外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 例外としてスローまたは再スローされる必要があります。 WebSphere eXtreme Scale 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。この誤りは、データベースと対話する汎用ローダーを作成した場合に起こる可能性があります。

eXtreme Scale 提供の JPALoader はその 1 例です。 JPALoader は JPA API を使用してデータベース・バックエンドと対話します。ネットワークで障害が発生すると、JPALoader は javax.persistence.PersitenceException を取得しますが、チェーニングされた SQLException の SQL 状態および SQL エラー・コードが確認されない限り、障害の本質を認識しません。 JPALoader があらゆるタイプのデータベースを処理するように設計されているという事実は、ネットワーク・ダウン問題の場合は SQL 状態およびエラー・コードが異なるため、問題をいっそう複雑にします。これを解決するため、WebSphere eXtreme Scale は ExceptionMapper API を提供して、ユーザーによる実装のプラグインで Exception をより使いやすい例外にマップできるようにします。例えば、ユーザーは、SQL 状態またはエラー・コードがネットワーク・ダウンを示している場合であれば、汎用 javax.persistence.PersitenceException を LoaderNotAvailableException にマップすることができます。

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ので、データベース・スループットが増加します。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。

- 読み取りおよび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
- データベース更新待ち時間

後書きサポートの構成

後書きサポートを使用可能にするには、ObjectGrid 記述子 XML ファイルを使用するか、BackingMap インターフェースでプログラマチックに行います。

後書きサポートを使用可能にするには、ObjectGrid 記述子 XML ファイルを使用するか、BackingMap インターフェースでプログラマチックに行います。

ObjectGrid 記述子 XML ファイル

ObjectGrid 記述子 XML ファイルを使用して ObjectGrid を構成する場合、backingMap タグで writeBehind 属性を設定すると、後書きローダーが使用可能になります。以下に例を示します。

```
<objectGrid name="library" >
     <backingMap name="book" writeBehind="T300;C900" pluginCollectionRef="bookPlugins"/>
```

この例では、book バックアップ・マップの後書きサポートがパラメーター T300;C900 で使用可能になります。後書き属性は、最大更新時間または最大キー更新数、あるいはその両方を指定します。後書きパラメーターの形式は以下のとおりです。

```
::= <defaults> | <update time> | <update key count> | <update time> ";"
<update key count>::= "T" <positive integer>::= "C" <positive integer>::= ""
```

- 後書き属性
- 更新時間
- 更新キー数
- デフォルト

ローダーに対する更新は、以下のいずれかのイベントが発生すると実行されます。

1. 最終更新以降、最大更新時間 (秒数) を経過する
2. キュー・マップ内の更新キーの数が最大更新キー数に達する。

これらのパラメーターは、ヒントにすぎません。実際の更新数および更新時間は、これらのパラメーターの近似値になります。ただし、実際の更新数または更新時間がパラメーターで定義されたものと同じであることを保証するものではありません。また、更新時間の範囲内で、最大 2 回まで更新が発生した後、最初の後書き更新が発生することがあります。これは、すべての区画で同時にデータベースにアクセスしないように ObjectGrid が更新開始時間をランダム化するためです。

前記の例の T300;C900 では、最終更新以降 300 秒が経過するか、900 個のキーが更新保留状態になると、ローダーはデータをバックエンドに書き込みます。デフォルトの更新時間は 300 秒で、デフォルトの更新キー数は 1000 です。

失敗した後書き更新の処理

WebSphere eXtreme Scale トランザクションが、バックエンド・トランザクションの開始前に終了するため、トランザクションの実行が誤った形の正常実行になる可能性があります。

例えば、バックアップ・マップには存在しないが、バックエンドに存在するエントリーを eXtreme Scale トランザクションに挿入すると、重複キーになることになりますが、eXtreme Scale トランザクションは成功します。しかしながら、後書きスレッドがバックエンドにそのオブジェクトを挿入するトランザクションは、重複キー例外で失敗します。

失敗した後書き更新の処理: クライアント・サイド

このような更新、あるいはその他失敗したバックエンド更新は、失敗した後書き更新となります。失敗した後書き更新は、失敗した後書き更新マップに保管されます。このマップは、失敗した更新のイベント・キューとして機能します。更新のキーは、固有の Integer オブジェクトで、値は、FailedUpdateElement のインスタンスになります。失敗した後書き更新マップは、エビクターによって構成されます。エビクターは、レコードを挿入後 1 時間経過すると排除します。このため、失敗した更新レコードは、1 時間以内に取得されないと失われます。

失敗した後書き更新マップのエントリーを取り出すには、ObjectMap API を使用できます。失敗した後書き更新マップの名前は IBM_WB_FAILED_UPDATES_<map name> です。 各後書きシステム・マップの接頭部名については、WriteBehindLoaderConstants API の資料を参照してください。以下に例を示します。

```
process failed - example code
ObjectMap failedMap = session.getMap(
    WriteBehindLoaderConstants.WRITE_BEHIND_FAILED_UPDATES_MAP_PREFIX + "Employee");
Object key = null;

session.begin();
while(key = failedMap.getNextKey(ObjectMap.QUEUE_TIMEOUT_NONE)) {
    FailedUpdateElement element = (FailedUpdateElement) failedMap.get(key);
    Throwable throwable = element.getThrowable();
    Object failedKey = element.getKey();
    Object failedValue = element.getAfterImage();
    failedMap.remove(key);
    // Do something interesting with the key, value, or exception.

}
session.commit();
```

getNextKey 呼び出しは、各 eXtreme Scale トランザクションごとに特定の 1 つの区画について作業します。分散環境では、すべての区画からキーを取得するため、以下の例に示すように複数のトランザクションを開始する必要があります。

```
getting keys from all partitions - example code
ObjectMap failedMap = session.getMap(
    WriteBehindLoaderConstants.WRITE_BEHIND_FAILED_UPDATES_MAP_PREFIX + "Employee");
while (true) {
    session.begin();
    Object key = null;
    while(( key = failedMap.getNextKey(5000) )!= null ) {
        FailedUpdateElement element = (FailedUpdateElement) failedMap.get(key);
    Throwable throwable = element.getThrowable();
        Object failedKey = element.getKey();
        Object failedValue = element.getAfterImage();
        failedMap.remove(key);
        // Do something interesting with the key, value, or exception.

    }
    Session.commit();
}
```

注: 失敗した更新マップは、アプリケーションのヘルスをモニターする 1 つの手段です。システムが失敗した更新マップに数多くのレコードを作成した場合、それは、後書きサポートを使用するように、アプリケーションまたはアーキテクチャーを再評価または修正する必要があるというサインです。6.1.0.5 以降、xsadmin スクリプトを使用して、失敗した更新マップのエントリー・サイズを確認することができます。

失敗した後書き更新の処理: 断片リスナー

後書きトランザクションが失敗した場合、それを検出し、ログに記録することが重要です。後書きを使用するアプリケーションはすべて、失敗した後書き更新を処理するウォッチャーを実装する必要があります。これによって、アプリケーションが正しくない更新マップ内のレコードを処理することが期待されるため、それらが除去されずに潜在的なメモリー不足になることを防ぐことができます。

以下のコードは、そのようなウォッチャー (ダンパー) の接続方法を示しています。これは、ObjectGrid 記述子 XML にスニペットとして追加する必要があります。

```
<objectGrid name="Grid">

        <bean id="ObjectGridEventListener" className="utils.WriteBehindDumper"/>
```

ObjectGridEventListener Bean が追加されていることがわかります。これは、上記で取り上げた後書きウォッチャーです。このウォッチャーは、JVM 内のすべてのプラ

イマリー断片のマップと対話し、後書きが使用可能になったものを検索します。後書きが使用可能になったものを検出すると、ウォッチャーは最大 100 の不適切な更新をログに記録しようとします。ウォッチャーは、プライマリー断片が別の JVM に移動されるまで、その断片を監視します。後書きを使用するすべてのアプリケーションは、これと似たウォッチャーを使用する必要があります。使用しないと、このエラー・マップが除去されないため、Java 仮想マシン がメモリー不足になります。

詳しくは、後書きダンパー・クラスのサンプル・コードを参照してください。

後書きダンパー・クラス・サンプル・コード

このサンプル・ソース・コードは、失敗した後書き更新を扱うウォッチャー (ダンパー) の作成方法を示しています。

```
//
//This sample program is provided AS IS and may be used, executed, copied and
//modified without royalty payment by customer (a) for its own instruction and
//study, (b) in order to develop applications designed to run with an IBM
//WebSphere product, either for customer's own internal use or for redistribution
//by customer, as part of such an application, in customer's own products. "
//
//5724-J34 (C) COPYRIGHT International Business Machines Corp. 2009
//All Rights Reserved * Licensed Materials - Property of IBM
//
package utils;

import java.util.Collection;
import java.util.Iterator;
import java.util.concurrent.Callable;
import java.util.concurrent.ScheduledExecutorService;
import java.util.concurrent.ScheduledFuture;
import java.util.concurrent.ScheduledThreadPoolExecutor;
import java.util.concurrent.TimeUnit;
import java.util.logging.Logger;

import com.ibm.websphere.objectgrid.BackingMap;
import com.ibm.websphere.objectgrid.ObjectGrid;
import com.ibm.websphere.objectgrid.ObjectGridException;
import com.ibm.websphere.objectgrid.ObjectGridRuntimeException;
import com.ibm.websphere.objectgrid.ObjectMap;
import com.ibm.websphere.objectgrid.Session;
import com.ibm.websphere.objectgrid.UndefinedMapException;
import com.ibm.websphere.objectgrid.plugins.ObjectGridEventGroup;
import com.ibm.websphere.objectgrid.plugins.ObjectGridEventListener;
import com.ibm.websphere.objectgrid.writebehind.FailedUpdateElement;
import com.ibm.websphere.objectgrid.writebehind.WriteBehindLoaderConstants;

/**
 * Write behind expects transactions to the Loader to succeed. If a transaction for a key fails then
 * it inserts an entry in a Map called PREFIX + mapName. The application should be checking this
 * map for entries to dump out write behind transaction failures. The application is responsible for
 * analyzing and then removing these entries. These entries can be large as they include the key, before
 * and after images of the value and the exception itself. Exceptions can easily be 20k on their own.
 *
 * The class is registered with the grid and an instance is created per primary shard in a JVM. It creates a single thread
 * and that thread then checks each write behind error map for the shard, prints out the problem and then removes the
 * entry.
 *
 * This means there will be one thread per shard. If the shard is moved to another JVM then the deactivate method stops the
 * thread.
 * @author bnewport
 *
 */
public class WriteBehindDumper implements ObjectGridEventListener, ObjectGridEventGroup.ShardEvents, Callable<Boolean>
{
 static Logger logger = Logger.getLogger(WriteBehindDumper.class.getName());

 ObjectGrid grid;

 /**
  * Thread pool to handle table checkers. If the application has it's own pool
```

```
 * then change this to reuse the existing pool
 */
static ScheduledExecutorService pool = new ScheduledThreadPoolExecutor(2); // two threads to dump records

// the future for this shard
ScheduledFuture<Boolean> future;

// true if this shard is active
volatile boolean isShardActive;

/**
 * Normal time between checking Maps for write behind errors
 */
final long BLOCKTIME_SECS = 20L;

/**
 * An allocated session for this shard. No point in allocating them again and again
 */
Session session;
/**
 * When a primary shard is activated then schedule the checks to periodically check
 * the write behind error maps and print out any problems
 */
public void shardActivated(ObjectGrid grid)
{
 try
 {
  this.grid = grid;
  session = grid.getSession();

  isShardActive = true;
  future = pool.schedule(this, BLOCKTIME_SECS, TimeUnit.SECONDS); // check every BLOCKTIME_SECS seconds initially
 }
 catch(ObjectGridException e)
 {
  throw new ObjectGridRuntimeException("Exception activating write dumper", e);
 }
}

/**
 * Mark shard as inactive and then cancel the checker
 */
public void shardDeactivate(ObjectGrid arg0)
{
 isShardActive = false;
 // if it's cancelled then cancel returns true
 if(future.cancel(false) == false)
 {
  // otherwise just block until the checker completes
  while(future.isDone() == false) // wait for the task to finish one way or the other
  {
   try
   {
    Thread.sleep(1000L); // check every second
   }
   catch(InterruptedException e)
   {
   }
  }
 }
}

/**
 * Simple test to see if the map has write behind enabled and if so then return
 * the name of the error map for it.
 * @param mapName The map to test
 * @return The name of the write behind error map if it exists otherwise null
 */
static public String getWriteBehindNameIfPossible(ObjectGrid grid, String mapName)
{
 BackingMap map = grid.getMap(mapName);
 if(map != null && map.getWriteBehind() != null)
 {
  return WriteBehindLoaderConstants.WRITE_BEHIND_FAILED_UPDATES_MAP_PREFIX + mapName;
 }
 else
  return null;
}
```

```
/**
 * This runs for each shard. It checks if each map has write behind enabled and if it does
  * then it prints out any write behind
 * transaction errors and then removes the record.
 */
public Boolean call()
{
 logger.fine("Called for " + grid.toString());
 try
 {
  // while the primary shard is present in this JVM
  // only user defined maps are returned here, no system maps like write behind maps are in
  // this list.
  Iterator<String> iter = grid.getListOfMapNames().iterator();
  boolean foundErrors = false;
  // iterate over all the current Maps
  while(iter.hasNext() && isShardActive)
  {
   String origName = iter.next();

   // if it's a write behind error map
   String name = getWriteBehindNameIfPossible(grid, origName);
   if(name != null)
   {
    // try to remove blocks of N errors at a time
    ObjectMap errorMap = null;
    try
    {
     errorMap = session.getMap(name);
    }
    catch(UndefinedMapException e)
    {
     // at startup, the error maps may not exist yet, patience...
     continue;
    }
    // try to dump out up to N records at once
    session.begin();
    for(int counter = 0; counter < 100; ++counter)
    {
     Integer seqKey = (Integer)errorMap.getNextKey(1L);
     if(seqKey != null)
     {
      foundErrors = true;
      FailedUpdateElement elem = (FailedUpdateElement)errorMap.get(seqKey);
      //
      // Your application should log the problem here
      logger.info("WriteBehindDumper ( " + origName + ") for key (" + elem.getKey() + ") Exception: " +
       elem.getThrowable().toString());
      //
      //
      errorMap.remove(seqKey);
     }
     else
      break;
    }
    session.commit();
   }
  } // do next map
  // loop faster if there are errors
  if(isShardActive)
  {
   // reschedule after one second if there were bad records
   // otherwise, wait 20 seconds.
   if(foundErrors)
    future = pool.schedule(this, 1L, TimeUnit.SECONDS);
   else
    future = pool.schedule(this, BLOCKTIME_SECS, TimeUnit.SECONDS);
  }
 }
 catch(ObjectGridException e)
 {
  logger.fine("Exception in WriteBehindDumper" + e.toString());
  e.printStackTrace();

  //don't leave a transaction on the session.
  if(session.isTransactionActive())
  {
   try { session.rollback(); } catch(Exception e2) {}
  }
```

```
  }
  return true;
 }

 public void destroy() {
  // TODO Auto-generated method stub

 }

 public void initialize(Session arg0) {
  // TODO Auto-generated method stub

 }

 public void transactionBegin(String arg0, boolean arg1) {
  // TODO Auto-generated method stub

 }

 public void transactionEnd(String arg0, boolean arg1, boolean arg2,
   Collection arg3) {
  // TODO Auto-generated method stub

 }
}
```

Evictor の構成

Evictor の構成は、ObjectGrid 記述子 XML ファイルを使用するか、プログラムで行います。

このタスクについて

XML を使用した構成については、161 ページの『ObjectGrid 記述子 XML ファイル』を参照してください。

TTL Evictor の使用可能化

WebSphere eXtreme Scale には、キャッシュ・エントリーを除去するためのデフォルトのメカニズムと、カスタム Evictor を作成するためのプラグインが用意されています。Evictor は、各 BackingMap インスタンスのエントリーのメンバーシップを制御します。デフォルトの Evictor では、各 BackingMap インスタンスに対して存続時間 (TTL) 除去ポリシーを使用します。プラグ可能 Evictor 機構を提供すると、この機構では通常、時間ではなく、エントリーの数に基づいた除去ポリシーが使用されます。

TTL Evictor のプログラマチックな使用可能化

TTL Evictor は、BackingMap インスタンスと関連しています。次のコード・スニペットは、各エントリーの作成時に、そのエントリーの有効期限の時間が、作成から 10 分後に設定されるようにするために、BackingMap インターフェースを使用して必要な属性を設定する方法を示しています。

enabling time-to-live evictor programmatically

```
import com.ibm.websphere.objectgrid.ObjectGridManagerFactory;
import com.ibm.websphere.objectgrid.ObjectGridManager;
import com.ibm.websphere.objectgrid.ObjectGrid;
import com.ibm.websphere.objectgrid.BackingMap;
import com.ibm.websphere.objectgrid.TTLType;
ObjectGridManager ogManager = ObjectGridManagerFactory.getObjectGridManager();
ObjectGrid og = ogManager.createObjectGrid( "grid" );
BackingMap bm = og.defineMap( "myMap" );
bm.setTtlEvictorType( TTLType.CREATION_TIME );
bm.setTimeToLive( 600 );
```

setTimeToLive メソッドの引数は、存続時間の値が秒単位であることを指示するため、600 になっています。前掲のコードは、ObjectGrid インスタンスで initialize メソッドを呼び出す前に実行する必要があります。これらの BackingMap 属性は、ObjectGrid の初期化後に変更することはできません。コードが実行された後、myMap BackingMap 内に挿入されたエントリーには有効期限の時間が設定されます。有効期限の時間に達すると、TTL Evictor はそのエントリーを除去します。

アプリケーションで、有効期限の時間を最終アクセス時刻プラス 10 分に設定する必要がある場合は、前のコードを 1 行変更する必要があります。 setTtlEvictorType メソッドに渡される引数は、TTLType.CREATION_TIME から TTLType.LAST_ACCESS_TIME に変更されます。この値を使用すると、有効期限の時間は最終アクセス時刻プラス 10 分として計算されます。最初にエントリーが作成されたときの最終アクセス時刻は、作成時刻です。

TTLType.LAST_ACCESS_TIME 引数を使用する場合、ObjectMap および JavaMap インターフェースを使用して、BackingMap の存続時間の値をオーバーライドすることができます。この機構により、アプリケーションが、作成される各エントリーに対して異なる存続時間の値を使用できるようになります。 ttlType 属性を LAST_ACCESS_TIME に設定するために、前述のコード・スニペットが使用され、BackingMap インスタンスに対して存続時間の値が 10 分に設定されたと想定します。アプリケーションは、エントリーを作成または変更する前に次のコードを実行して、各エントリーの存続時間をオーバーライドします。

```
import com.ibm.websphere.objectgrid.Session;
import com.ibm.websphere.objectgrid.ObjectMap;
Session session = og.getSession();
ObjectMap om = session.getMap( "myMap" );
int oldTimeToLive1 = om.setTimeToLive( 1800 );
om.insert("key1", "value1" );
int oldTimeToLive2 = om.setTimeToLive( 1200 );
om.insert("key2", "value2" );
```

前掲のコード・スニペットでは、key1 キーを持つエントリーの有効期限の時間は、ObjectMap の setTimeToLive(1800) メソッドを呼び出した結果、挿入時刻プラス 30 分になります。oldTimeToLive1 変数は 600 に設定されます。それは、以前に ObjectMap の setTimeToLive メソッドが呼び出されていなかった場合は、デフォルト値として BackingMap の存続時間の値が使用されるからです。

key2 キーを持つエントリーの有効期限の時間は、ObjectMap の setTimeToLive(1200) メソッドを呼び出した結果、挿入時刻プラス 20 分になります。oldTimeToLive2 変数は 1800 に設定されます。それは、前の ObjectMap.setTimeToLive メソッド呼び出しで存続時間の値が 1800 に設定されたからです。

前の例では、キー値が key1 と key2 の 2 つのマップ・エントリーが、myMap マップに挿入されています。新規スレッドのアプリケーションは、後で新規のマップ値を用いてこれらのマップ・エントリーを更新することができます。しかし、このアプリケーションでは、挿入時にマップ・エントリーごとに使用された存続時間の値を保持しておく必要があります。以下の例では、ObjectMap インターフェースで定義されている定数を使用して、存続時間値を保持する方法を示しています。

```
Session session = og.getSession();
ObjectMap om = session.getMap( "myMap" );
om.setTimeToLive( ObjectMap.USE_DEFAULT );
session.begin();
```

```
om.update("key1", "updated value1" );
om.update("key2", "updated value2" );
om.insert("key3", "value3" );
session.commit();
```

ObjectMap.USE_DEFAULT 特殊値は setTimeToLive メソッド呼び出しで使用されるので、key1 キーには 1800 秒、key2 キーには 1200 秒の存続時間値が保管されます。これらの値が、前のトランザクションでこれらのマップ・エントリーが挿入されたときに使用されたためです。

前の例では、key3 キーの新規マップ・エントリーの挿入も示されています。この場合、USE_DEFAULT 特殊値は、このマップの存続時間値にデフォルト設定を使用することを示しています。デフォルト値は、BackingMap の存続時間属性により定義されます。BackingMap インスタンスに存続時間属性を定義する方法について詳しくは、BackingMap インターフェース属性を参照してください。

ObjectMap インターフェースおよび JavaMap インターフェースの setTimeToLive メソッドについては、API の資料を参照してください。この資料では、BackingMap.getTtlEvictorType() メソッドから TTLType.LAST_ACCESS_TIME 以外の値が戻された場合は、IllegalStateException 例外が生じることを警告しています。ObjectMap および JavaMap は、LAST_ACCESS_TIME TTL Evictor タイプを使用しているときに、存続時間値をオーバーライドする場合にのみ使用できます。このメソッドは、CREATION_TIME TTL Evictor タイプまたは NONE TTL Evictor タイプを使用しているときに、存続時間の値をオーバーライドする場合には使用できません。

TTL Evictor の使用可能化用の XML 構成アプローチ

BackingMap インターフェースを使用して、TTL Evictor が使用する BackingMap 属性をプログラマチックに設定する代わりに、 XML ファイルを使用して各 BackingMap インスタンスを構成することができます。以下のコードは、3 つの異なる BackingMap マップに対してこれらの属性を設定する方法を示しています。

```
enabling time-to-live evictor using XML
<?xml version="1.0" encoding="UTF-8"?>
<objectGridConfig xmlns:xsi="http://www.w3.org/2001/XMLSchema-instance"
 xsi:schemaLocation="http://ibm.com/ws/objectgrid/config ../objectGrid.xsd"
 xmlns="http://ibm.com/ws/objectgrid/config">
<objectGrids>
    <objectGrid name="grid1">
        <backingMap name="map1" ttlEvictorType="NONE" />
        <backingMap name="map2" ttlEvictorType="LAST_ACCESS_TIME" timeToLive="1800" />
        <backingMap name="map3" ttlEvictorType="CREATION_TIME" timeToLive="1200" />
    </objectgrid>
</objectGrids>
```

前掲の例は、map1 BackingMap が NONE TTL Evictor タイプを使用することを示しています。map2 BackingMap では、LAST_ACCESS_TIME TTL Evictor タイプが使用され、存続時間の値は 1800 秒、すなわち 30 分になります。map3 BackingMap は、CREATION_TIME TTL Evictor タイプを使用するように定義されており、その存続時間の値は 1200 秒、すなわち 20 分です。

オプションのプラグ可能 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。

オプションのプラグ可能 Evictor の使用

オプションのプラグ可能 Evictor を BackingMap 構成に追加する場合、プログラマチック構成または XML 構成を使用できます。

プラグ可能 Evictor のプログラマチックなプラグイン

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 を指定します。次のコード・スニペットは、map1 BackingMap インスタンス用に LRUEvictor Evictor、および map2 BackingMap インスタンス用に LFUEvictor Evictor を指定する例です。

```
plugging in an evictor programmatically
import com.ibm.websphere.objectgrid.ObjectGridManagerFactory;
import com.ibm.websphere.objectgrid.ObjectGridManager;
import com.ibm.websphere.objectgrid.ObjectGrid;
import com.ibm.websphere.objectgrid.BackingMap;
import com.ibm.websphere.objectgrid.plugins.builtins.LRUEvictor;
import com.ibm.websphere.objectgrid.plugins.builtins.LFUEvictor;
ObjectGridManager ogManager = ObjectGridManagerFactory.getObjectGridManager();
ObjectGrid og = ogManager.createObjectGrid( "grid" );
BackingMap bm = og.defineMap( "map1" );
LRUEvictor evictor = new LRUEvictor();
evictor.setMaxSize(1000);
evictor.setSleepTime( 15 );
evictor.setNumberOfLRUQueues( 53 );
bm.setEvictor(evictor);
bm = og.defineMap( "map2" );
LFUEvictor evictor2 = new LFUEvictor();
evictor2.setMaxSize(2000);
evictor2.setSleepTime( 15 );
evictor2.setNumberOfHeaps( 211 );
bm.setEvictor(evictor2);
```

上記のスニペットは、概算最大エントリー数が 53,000 (53 x 1000) である map1 BackingMap に使用される LRUEvictor Evictor を示しています。 LFUEvictor Evictor は、概算最大エントリー数が 422,000 (211 x 2000) である map2 BackingMap に使用されます。LRU Evictor と LFU Evictor はいずれもスリープ時間プロパティーを持ちます。このプロパティーは、ウェイクアップしてエントリーを除去する必要があるかどうかを検査するまで Evictor がスリープする時間を示します。スリープ時間は、秒単位で指定されます。値 15 秒は、パフォーマンスの影響と BackingMap が大きくなりすぎないようにするための妥当な値です。目標は、BackingMap のサイズが超過することなく、できる限り長いスリープ時間を使用することです。

setNumberOfLRUQueues メソッドは、LRUEvictor プロパティーを設定します。このプロパティーは、LRU 情報を管理するために Evictor が使用する LRU キューの数を示します。各エントリーが同じキュー内の LRU 情報を保持しないように、キューのコレクションを使用します。この方法により、同じキュー・オブジェクトで同期化する必要があるマップ・エントリーの数が最小化され、パフォーマンスが向上します。キューの数を増やすのは、LRU Evictor がパフォーマンスに与えるインパクトを最小化するための良い方法です。開始点としては、キューの数として最大エントリー数の 10 % を使用することが適切です。一般に、基本数以外を使用するよりも、基本数を使用するほうが適しています。 setMaxSize メソッドは、各キューで許容されるエントリーの数を示します。キューがその最大エントリー数に達すると、キュー内の最も使用頻度の少ない 1 つまたは複数のエントリーが、次回に Evictor がエントリーを除去する必要があるかどうかをチェックした時点で除去されます。

setNumberOfHeaps メソッドは、LFUEvictor プロパティーを設定します。このプロパティーは、LFU 情報を管理するために LFUEvictor が使用するバイナリー・ヒープ・オブジェクトの数を設定します。この場合も、コレクションを使用するとパフォーマンスが向上します。開始点としては、最大エントリー数の 10% を使用することが適切であり、一般に、基本数以外を使用するよりも、基本を使用するほうが適しています。 setMaxSize メソッドは、各ヒープで許容されるエントリーの数を示します。ヒープがその最大エントリー数に達すると、ヒープ内の最も使用頻度の低い 1 つまたは複数のエントリーが、次回に Evictor がエントリーを除去する必要があるかどうかをチェックした時点で除去されます。

プラグ可能 Evictor のプラグインの XML 構成アプローチ

さまざまな API を使用して、Evictor をプログラマチックにプラグインし、そのプロパティーを設定する代わりに、次の例に示すように、XML ファイルを使用して各 BackingMap を構成することができます。

```
plugging in an evictor using XML<?xml version="1.0" encoding="UTF-8"?>
<objectGridConfig xmlns:xsi="http://www.w3.org/2001/XMLSchema-instance"
 xsi:schemaLocation="http://ibm.com/ws/objectgrid/config ../objectGrid.xsd"
 xmlns="http://ibm.com/ws/objectgrid/config">
<objectGrids>
    <objectGrid name="grid">
        <backingMap name="map1" ttlEvictorType="NONE" pluginCollectionRef="LRU" />
        <backingMap name="map2" ttlEvictorType="NONE" pluginCollectionRef="LFU" />
    </objectGrid>
</objectGrids>
<backingMapPluginCollections>
    <backingMapPlugincollection id="LRU">
        <bean id="Evictor" className="com.ibm.websphere.objectgrid.plugins.builtins.LRUEvictor">
            <property name="maxSize" type="int" value="1000" description="set max size for each LRU queue" />
            <property name="sleepTime" type="int" value="15" description="evictor thread sleep time" />
```

```
            <property name="numberOfLRUQueues" type="int" value="53" description="set number of LRU queues" />
        </bean>
    </backingMapPluginCollection>
    <backingMapPluginCollection id="LFU">
        <bean id="Evictor" className="com.ibm.websphere.objectgrid.plugins.builtins.LFUEvictor">
            <property name="maxSize" type="int" value="2000" description="set max size for each LFU heap" />
            <property name="sleepTime" type="int" value="15" description="evictor thread sleep time" />
            <property name="numberOfHeaps" type="int" value="211" description="set number of LFU heaps" />
        </bean>
    </backingMapPluginCollection>
</backingMapPluginCollections>
</objectGridConfig>
```

メモリー・ベースの除去

組み込み Evictor はすべて、BackingMap の evictionTriggers 属性を「MEMORY_USAGE_THRESHOLD」に設定して、BackingMap インターフェースで使用可能にできるメモリー・ベースの除去をサポートします。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 eXtreme Scale 構成の解説書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。

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。 eXtreme Scale 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。containerServer.props ファイルおよび eXtreme Scale サーバー・プロセスの開始も参照してください。

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、そのままシステム・ランタイムによるメモリー消費が速い状態が続くと、除去速度が十分であっても、メモリー不足エラーの可能性がなくなるという保証はありません。

プラグ可能エビクターのプラグイン

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 を指定します。

プラグ可能 Evictor のプログラマチックなプラグイン

次のコード・スニペットは、map1 BackingMap 用の LRUEvictor Evictor、または map2 BackingMap 用の LFUEvictor Evictor を指定する例です。

```
import com.ibm.websphere.objectgrid.ObjectGridManagerFactory;
import com.ibm.websphere.objectgrid.ObjectGridManager;
import com.ibm.websphere.objectgrid.ObjectGrid;
import com.ibm.websphere.objectgrid.BackingMap;
import com.ibm.websphere.objectgrid.plugins.builtins.LRUEvictor;
import com.ibm.websphere.objectgrid.plugins.builtins.LFUEvictor;
ObjectGridManager ogManager = ObjectGridManagerFactory.getObjectGridManager();
ObjectGrid og = ogManager.createObjectGrid( "grid" );
BackingMap bm = og.defineMap( "map1" );
LRUEvictor evictor = new LRUEvictor();
```

```
evictor.setMaxSize(1000);
evictor.setSleepTime( 15 );
evictor.setNumberOfLRUQueues( 53 );
bm.setEvictor(evictor);
bm = og.defineMap( "map2" );
LFUEvictor evictor2 = new LFUEvictor();
evictor2.setMaxSize(2000);
evictor2.setSleepTime( 15 );
evictor2.setNumberOfHeaps( 211 );
bm.setEvictor(evictor2);
```

上記のスニペットは、概算最大エントリー数が 53,000 (53 x 1000) である map1 BackingMap に使用される LRUEvictor Evictor を示しています。 LFUEvictor Evictor は、概算最大エントリー数が 422,000 (211 x 2000) である map2 BackingMap に使用されます。LRU Evictor と LFU Evictor はいずれもスリープ時間プロパティーを持ちます。このプロパティーは、ウェイクアップしてエントリーを除去する必要があるかどうかを検査するまで Evictor がスリープする時間を示します。スリープ時間は、秒単位で指定されます。値 15 秒は、パフォーマンスの影響と BackingMap が大きくなりすぎないようにするための妥当な値です。目標は、BackingMap のサイズが超過することなく、できる限り長いスリープ時間を使用することです。

setNumberOfLRUQueues メソッドは、LRUEvictor プロパティーを設定します。このプロパティーは、LRU 情報を管理するために Evictor が使用する LRU キューの数を示します。各エントリーが同じキュー内の LRU 情報を保持しないように、キューのコレクションを使用します。この方法により、同じキュー・オブジェクトで同期化する必要があるマップ・エントリーの数が最小化され、パフォーマンスが向上します。キューの数を増やすのは、LRU Evictor がパフォーマンスに与えるインパクトを最小化するための良い方法です。開始点としては、キューの数として最大エントリー数の 10 % を使用することが適切です。一般に、基本数以外を使用するよりも、基本数を使用するほうが適しています。 setMaxSize メソッドは、各キューに許可されるエントリー数を指定します。キューがその最大エントリー数に達すると、キュー内の最も使用頻度の少ない 1 つまたは複数のエントリーが、次回に Evictor がエントリーを除去する必要があるかどうかをチェックした時点で除去されます。

setNumberOfHeaps メソッドは、LFUEvictor プロパティーを設定します。このプロパティーは、LFU 情報を管理するために LFUEvictor が使用するバイナリー・ヒープ・オブジェクトの数を設定します。この場合も、コレクションを使用するとパフォーマンスが向上します。開始点としては、最大エントリー数の 10% を使用することが適切であり、一般に、基本数以外を使用するよりも、基本を使用するほうが適しています。 setMaxSize メソッドは、各ヒープに許可されるエントリー数を指定します。ヒープがその最大エントリー数に達すると、ヒープ内の最も使用頻度の低い 1 つまたは複数のエントリーが、次回に Evictor がエントリーを除去する必要があるかどうかをチェックした時点で除去されます。

プラグ可能 Evictor のプラグインの XML 構成アプローチ

さまざまな API を使用して、Evictor をプログラマチックにプラグインし、そのプロパティーを設定する代わりに、次の例に示すように、XML ファイルを使用して各 BackingMap を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="grid">
        <backingMap name="map1" ttlEvictorType="NONE" pluginCollectionRef="LRU" />
        <backingMap name="map2" ttlEvictorType="NONE" pluginCollectionRef="LFU" />
    </objectGrid>
</objectGrids>
<backingMapPluginCollections>
    <backingMapPlugincollection id="LRU">
        <bean id="Evictor" className="com.ibm.websphere.objectgrid.plugins.builtins.LRUEvictor">
            <property name="maxSize" type="int" value="1000" description="set max size for each LRU queue" />
            <property name="sleepTime" type="int" value="15" description="evictor thread sleep time" />
            <property name="numberOfLRUQueues" type="int" value="53" description="set number of LRU queues" />
        </bean>
    </backingMapPluginCollection>
    <backingMapPluginCollection id="LFU">
        <bean id="Evictor" className="com.ibm.websphere.objectgrid.plugins.builtins.LFUEvictor">
            <property name="maxSize" type="int" value="2000" description="set max size for each LFU heap" />
            <property name="sleepTime" type="int" value="15" description="evictor thread sleep time" />
            <property name="numberOfHeaps" type="int" value="211" description="set number of LFU heaps" />
        </bean>
    </backingMapPluginCollection>
</backingMapPluginCollections>
</objectGridConfig>
```

HashIndex の構成

HashIndex プラグインは、MapIndex インターフェースと MapRangeIndex インターフェースの両方をサポートします。索引を適切に定義および使用すると、照会のパフォーマンスをかなり改善できます。

ObjectGrid デプロイメント記述子 XML ファイルを使用して、またはプログラマチックに、以下の属性を HashIndex の構成に使用できます。

- Name: 索引の名前。名前は各マップで固有でなければなりません。この名前は、BackingMap の ObjectMap インスタンスから索引オブジェクトを取り出すのに使用されます。
- AttributeName: 索引に対する属性の名前をコンマで区切ったリスト。フィールド・アクセス索引の場合、属性名はフィールド名と同じです。プロパティー・アクセス索引の場合、属性名は JavaBean 互換のプロパティー名です。HashIndex は、属性名が 1 つだけであれば単一属性索引であり、その属性がリレーションシップの場合はリレーションシップ索引でもあります。複数の属性名が含まれている場合は、HashIndex は複合索引です。
- FieldAccessAttribute: 非エンティティー・マップに使用されます。true の場合、フィールドを使用してオブジェクトに直接アクセスします。指定されていないか、false の場合、データのアクセスには、属性の getter メソッドが使用されます。
- POJOKeyIndex: 非エンティティー・マップに使用されます。true の場合、索引はマップのキー部分でオブジェクトをイントロスペクトします。これは、キーが複合キーで、値にキーが組み込まれていない場合に役立ちます。指定されていないか、false の場合、索引はマップの値部分でオブジェクトをイントロスペクトします。
- RangeIndex: true の場合、範囲索引付けが使用可能にされ、アプリケーションは取り出された索引オブジェクトを MapRangeIndex インターフェースにキャストできます。 RangeIndex プロパティーが「false」と構成されている場合は、アプリケーションは取り出された索引オブジェクトを MapIndex インターフェースにしかキャストできません。

単一属性 HashIndex と複合 HashIndex

HashIndex の AttributeName プロパティーに複数の属性名が含まれている場合、HashIndex は複合索引です。そうではなく、含まれている属性名が 1 つのみの場合は、単一属性索引です。例えば、 AttributeName プロパティー値が「city,state,zipcode」であるような、複合 HashIndex が考えられます。 この例では、3 つの属性がコンマで区切られています。もし AttributeName プロパティー値が単に「zipcode」であれば、属性は 1 つだけなので、単一属性 HashIndex であるということになります。

複合 HashIndex は、検索条件に多くの属性が関係するような場合に、キャッシュに入れられたオブジェクトを検索する効果的な方法を提供します。ただし、範囲索引はサポートしないため、 RangeIndex プロパティーは「false」に設定されている必要があります。

管理ガイド の複合 HashIndex に関するトピックを参照してください。

リレーションシップ HashIndex

単一属性 HashIndex の索引属性が、単一値または複数値のリレーションシップの場合、その HashIndex はリレーションシップ HashIndex です。リレーションシップ HashIndex の場合、 HashIndex の RangeIndex プロパティーは「false」に設定されている必要があります。

リレーションシップ HashIndex は、循環参照を使用する照会や、IS NULL、IS EMPTY、SIZE、および MEMBER OF 照会フィルターを使用する照会を速くすることができます。「*プログラミング・ガイド*」で、索引を使用する照会最適化についての説明を参照してください。

範囲 HashIndex

HashIndex の RangeIndex プロパティーが「true」に設定されている場合、その HashIndex は範囲索引であり、 MapRangeIndex インターフェースをサポートできます。MapRangeIndex は、範囲関数 greater than や less than、あるいは両方を使用するデータ検出をサポートしますが、MapIndex は equals 関数のみをサポートします。単一属性索引の場合、RangeIndex プロパティーを「true」に設定できるのは、索引付けられる属性のタイプが Comparable の場合に限ります。単一属性索引が照会によって使用される場合、RangeIndex プロパティーは「true」に設定されていなければならず、索引付けられる属性のタイプは Comparable でなければなりません。リレーションシップ HashIndex および複合 HashIndex の場合、RangeIndex プロパティーは「false」に設定されていなければなりません。

次の表は、範囲索引の使用についての要約です。

表 7. 範囲索引のサポート

HashIndex タイプ	**範囲索引のサポート**
単一属性 HashIndex: 索引付けられるキーまたは属性のタイプは Comparable である	あり
単一属性 HashIndex: 索引付けられるキーまたは属性のタイプは Comparable でない	なし

表 7. 範囲索引のサポート (続き)

HashIndex タイプ	範囲索引のサポート
複合 HashIndex	なし
リレーションシップ HashIndex	なし

HashIndex を使用した照会の最適化

索引を適切に定義および使用すると、照会のパフォーマンスをかなり改善できます。WebSphere® eXtreme Scale 照会は、組み込み HashIndex プラグインを使用して、照会のパフォーマンスを向上させることができます。索引の使用は、照会パフォーマンスを大きく向上させることができますが、トランザクションのマップ操作のパフォーマンスに影響を与えることもあります。

JMS を使用したピアツーピア複製の構成

Java Message Service (JMS) を使用したピアツーピア複製メカニズムは、分散およびローカルの両方の WebSphere eXtreme Scale 環境で使用されます。 JMS は、コアツーコア複製プロセスであり、これにより、ローカル ObjectGrid と分散 ObjectGrid の間でデータ更新の流れが可能になります。 例えば、このメカニズムを使用すると分散 eXtreme Scale グリッドからローカル eXtreme Scale グリッドに、あるいは、あるグリッドから別のシステム・ドメインにある別のグリッドに、データ更新を移動させることができます。

始める前に

JMS ベースのピアツーピア複製メカニズムは、組み込まれた JMS ベースの ObjectGridEventListener (com.ibm.websphere.objectgrid.plugins.builtins.JMSObjectGridEventListener) に基づいています。ピアツーピア複製メカニズムの使用可能化に関する詳細については、 150 ページの『JMS イベント・リスナー』を参照してください。

詳しくは、 84 ページの『クライアント無効化メカニズムの使用可能化』を参照してください。

以下は、eXtreme Scale 構成上でピアツーピア複製メカニズムを使用可能にする XML 構成の例です。

ピアツーピア複製構成 - XML の例

```
<bean id="ObjectGridEventListener"
className="com.ibm.websphere.objectgrid.plugins.JMSObjectGridEventListener">
 <property name="replicationRole" type="java.lang.String" value="DUAL_ROLES" description="" />
  <property name="replicationStrategy" type="java.lang.String" value="PUSH" description="" />
  <property name="jms_topicConnectionFactoryJndiName" type="java.lang.String" value="defaultTCF" description="" />
  <property name="jms_topicJndiName" type="java.lang.String" value="defaultTopic" description="" />
  <property name="jms_topicName" type="java.lang.String" value="defaultTopic" description="" />
  <property name="jms_userid" type="java.lang.String" value="" description="" />
  <property name="jms_password" type="java.lang.String" value="" description="" />
  <property name="jndi_properties" type="java.lang.String"
 value="java.naming.factory.initial=org.apache.activemq.jndi.ActiveMQInitialContextFactory;
 java.naming.provider.url=tcp://localhost:61616;connectionFactoryNames=defaultTCF;
 topic.defaultTopic=defaultTopic"
  description="jndi properties" />
     </bean>
```

クライアント無効化メカニズムの使用可能化

分散 WebSphere eXtreme Scale 環境では、optimistic ロック・ストラテジーが使用されているか、ロックが無効にされている場合、クライアント・サイドにはデフォルトでニア・キャッシュがあります。ニア・キャッシュにはそれ独自のローカル・キャッシュ・データがあります。eXtreme Scale クライアントが更新をコミットすると、この更新はクライアントのニア・キャッシュとサーバーに送られます。しかし、他の eXtreme Scale クライアントはこの更新情報を受け取らないので、それらのデータは古くなっていることがあります。

ニア・キャッシュ

アプリケーションは、eXtreme Scale クライアントのこの失効データの問題を認識しておく必要があります。Java Message Service (JMS) ベースの組み込み ObjectGridEventListener クラス com.ibm.websphere.objectgrid.plugins.builtins.JMSObjectGridEventListener を使用して、分散 eXtreme Scale 環境 (eXtreme Scale グリッドと呼ばれます) 内で、クライアント無効化メカニズムを使用可能にすることができます。

クライアント無効化メカニズムは、分散 eXtreme Scale 環境におけるクライアントのニア・キャッシュ内の失効データの問題に対する解決方法です。このメカニズムにより、クライアントのニア・キャッシュはサーバーまたは他のクライアントと同期します。ただし、この JMS ベースのクライアント無効化メカニズムを使用しても、クライアントのニア・キャッシュが即時に更新されるわけではありません。eXtreme Scale ランタイムが更新を公開するときに遅延が発生します。

分散 eXtreme Scale 環境におけるクライアント無効化メカニズムには、次の 2 つのモデルがあります。

- クライアント/サーバー・モデル: このモデルでは、すべてのサーバー・プロセスは、指定された JMS の宛先にすべてのトランザクション変更を公開する publisher ロールにあります。すべてのクライアント・プロセスは受信側ロールにあり、指定された JMS の宛先からすべてのトランザクション変更を受け取ります。
- 二重ロール・モデルとしてのクライアント: このモデルでは、すべてのサーバー・プロセスは JMS の宛先とは無関係です。すべてのクライアント・プロセスは JMS publisher ロールであり、かつ receiver ロールです。クライアントで発生するトランザクション変更は JMS の宛先に公開され、すべてのクライアントがこれらのトランザクション変更を受け取ります。

クライアント無効化メカニズムの使用可能化について詳しくは、 150 ページの『JMS イベント・リスナー』を参照してください。

クライアント/サーバー・モデル

クライアント/サーバー・モデルでは、サーバーは JMS publisher ロールにあり、クライアントは JMS receiver ロールにあります。

client-server model XML example

```
<?xml version="1.0" encoding="UTF-8"?>
<?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="AgentObjectGrid">
      <bean id="ObjectGridEventListener"
        className="com.ibm.websphere.objectgrid.plugins.builtins.JMSObjectGridEventListener">
        <property name="invalidationModel" type="java.lang.String" value="CLIENT_SERVER_MODEL" description="" />
        <property name="invalidationStrategy" type="java.lang.String" value="PUSH" description="" />
        <property name="mapsToPublish" type="java.lang.String" value="agent;profile;pessimisticMap" description="" />
        <property name="jms_topicConnectionFactoryJndiName" type="java.lang.String" value="defaultTCF" description="" />
        <property name="jms_topicJndiName" type="java.lang.String" value="defaultTopic" description="" />
        <property name="jms_topicName" type="java.lang.String" value="defaultTopic" description="" />
        <property name="jms_userid" type="java.lang.String" value="" description="" />
        <property name="jms_password" type="java.lang.String" value="" description="" />
        <property name="jndi_properties" type="java.lang.String"
          value="java.naming.factory.initial=org.apache.activemq.jndi.ActiveMQInitialContextFactory;java.naming.provider.url=
    tcp://localhost:61616;connectionFactoryNames=defaultTCF;topic.defaultTopic=defaultTopic"
          description="jndi properties" />
      </bean>

      <backingMap name="agent" readOnly="false" pluginCollectionRef="agent" preloadMode="false"
        lockStrategy="OPTIMISTIC" copyMode="COPY_ON_READ_AND_COMMIT" ttlEvictorType="LAST_ACCESS_TIME"
        timeToLive="28800" />
      <backingMap name="profile" readOnly="false" pluginCollectionRef="profile" preloadMode="false"
        lockStrategy="OPTIMISTIC" copyMode="COPY_ON_READ_AND_COMMIT" ttlEvictorType="LAST_ACCESS_TIME"
        timeToLive="2700" />
      <backingMap name="pessimisticMap" readOnly="false" pluginCollectionRef="pessimisticMap" preloadMode="false"
        lockStrategy="PESSIMISTIC" copyMode="COPY_ON_READ_AND_COMMIT" ttlEvictorType="LAST_ACCESS_TIME"
        timeToLive="2700" />
      <backingMap name="excludedMap1" readOnly="false" pluginCollectionRef="excludedMap1" preloadMode="false"
        lockStrategy="OPTIMISTIC" copyMode="COPY_ON_READ_AND_COMMIT" ttlEvictorType="LAST_ACCESS_TIME"
        timeToLive="2700" />
      <backingMap name="excludedMap2" readOnly="false" pluginCollectionRef="excludedMap2" preloadMode="false"
        lockStrategy="OPTIMISTIC" copyMode="COPY_ON_READ_AND_COMMIT" ttlEvictorType="LAST_ACCESS_TIME"
        timeToLive="2700" />
    </objectGrid>
  </objectGrids>

  <backingMapPluginCollections>
    <backingMapPluginCollection id="agent">
      <bean id="ObjectTransformer" className="com.ibm.ws.objectgrid.test.scenario.AgentObjectTransformer" />
    </backingMapPluginCollection>
    <backingMapPluginCollection id="profile">
      <bean id="ObjectTransformer" className="com.ibm.ws.objectgrid.test.scenario.ProfileObjectTransformer" />
      <bean id="Evictor" className="com.ibm.websphere.objectgrid.plugins.builtins.LRUEvictor">
        <property name="maxSize" type="int" value="2000" description="set max size for LRU evictor" />
        <property name="sleepTime" type="int" value="15" description="evictor thread sleep time" />
        <property name="numberOfLRUQueues" type="int" value="50" description="set number of LRU queues" />
      </bean>
    </backingMapPluginCollection>

    <backingMapPluginCollection id="pessimisticMap" />
    <backingMapPluginCollection id="excludedMap1" />
    <backingMapPluginCollection id="excludedMap2" />
  </backingMapPluginCollections>

</objectGridConfig>
```

二重ロール・モデルとしてのクライアント

二重ロール・モデルとしてのクライアントでは、各クライアントは JMS publisher ロールと receiver ロールの両方を持っています。クライアントは、コミットされたすべてのトランザクション変更を、指定された JMS 宛先に公開し、コミットされたすべてのトランザクション変更を他のクライアントから受け取ります。このモデルでは、サーバーは JMS とは無関係です。

dual-roles model XML example

```
<?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="AgentObjectGrid">
      <bean id="ObjectGridEventListener"
        className="com.ibm.websphere.objectgrid.plugins.builtins.JMSObjectGridEventListener">
        <property name="invalidationModel" type="java.lang.String" value="CLIENT_AS_DUAL_ROLES_MODEL" description="" />
        <property name="invalidationStrategy" type="java.lang.String" value="PUSH" description="" />
        <property name="mapsToPublish" type="java.lang.String" value="agent;profile;pessimisticMap" description="" />
        <property name="jms_topicConnectionFactoryJndiName" type="java.lang.String" value="defaultTCF" description="" />
        <property name="jms_topicJndiName" type="java.lang.String" value="defaultTopic" description="" />
        <property name="jms_topicName" type="java.lang.String" value="defaultTopic" description="" />
        <property name="jms_userid" type="java.lang.String" value="" description="" />
        <property name="jms_password" type="java.lang.String" value="" description="" />
        <property name="jndi_properties" type="java.lang.String"
          value="java.naming.factory.initial=org.apache.activemq.jndi.ActiveMQInitialContextFactory;java.naming.provider.url=
     tcp://localhost:61616;connectionFactoryNames=defaultTCF;topic.defaultTopic=defaultTopic"
          description="jndi properties" />
```

```
      </bean>

      <backingMap name="agent" readOnly="false" pluginCollectionRef="agent" preloadMode="false"
        lockStrategy="OPTIMISTIC" copyMode="COPY_ON_READ_AND_COMMIT" ttlEvictorType="LAST_ACCESS_TIME"
        timeToLive="28800" />
      <backingMap name="profile" readOnly="false" pluginCollectionRef="profile" preloadMode="false"
        lockStrategy="OPTIMISTIC" copyMode="COPY_ON_READ_AND_COMMIT" ttlEvictorType="LAST_ACCESS_TIME"
        timeToLive="2700" />
      <backingMap name="pessimisticMap" readOnly="false" pluginCollectionRef="pessimisticMap" preloadMode="false"
        lockStrategy="PESSIMISTIC" copyMode="COPY_ON_READ_AND_COMMIT" ttlEvictorType="LAST_ACCESS_TIME"
        timeToLive="2700" />
      <backingMap name="excludedMap1" readOnly="false" pluginCollectionRef="excludedMap1" preloadMode="false"
        lockStrategy="OPTIMISTIC" copyMode="COPY_ON_READ_AND_COMMIT" ttlEvictorType="LAST_ACCESS_TIME"
        timeToLive="2700" />
      <backingMap name="excludedMap2" readOnly="false" pluginCollectionRef="excludedMap2" preloadMode="false"
        lockStrategy="OPTIMISTIC" copyMode="COPY_ON_READ_AND_COMMIT" ttlEvictorType="LAST_ACCESS_TIME"
        timeToLive="2700" />
    </objectGrid>
  </objectGrids>

  <backingMapPluginCollections>
    <backingMapPluginCollection id="agent">
      <bean id="ObjectTransformer" className="com.ibm.ws.objectgrid.test.scenario.AgentObjectTransformer" />
    </backingMapPluginCollection>
    <backingMapPluginCollection id="profile">
      <bean id="ObjectTransformer" className="com.ibm.ws.objectgrid.test.scenario.ProfileObjectTransformer" />
      <bean id="Evictor" className="com.ibm.websphere.objectgrid.plugins.builtins.LRUEvictor">
        <property name="maxSize" type="int" value="2000" description="set max size for LRU evictor" />
        <property name="sleepTime" type="int" value="15" description="evictor thread sleep time" />
        <property name="numberOfLRUQueues" type="int" value="50" description="set number of LRU queues" />
      </bean>
    </backingMapPluginCollection>

    <backingMapPluginCollection id="pessimisticMap" />
    <backingMapPluginCollection id="excludedMap1" />
    <backingMapPluginCollection id="excludedMap2" />
  </backingMapPluginCollections>

</objectGridConfig>
```

ピア Java 仮想マシン間の変更の配布

LogSequence オブジェクトおよび LogElement オブジェクトは、ObjectGridEventListener プラグインを使用して、eXtreme Scale トランザクションで発生した変更を通信します。

Java Message Service (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、「*製品概要*」に記載されているトランザクションの変更を配布する Java Message Service の使用に関する説明を参照してください。

ObjectGrid インスタンスが ObjectGridManager によりキャッシュされていることが前提条件です。詳しくは、createObjectGrid メソッドを参照してください。cacheInstance ブール値は、true に設定する必要があります。

このメカニズムを実装する必要はありません。この機能を使用するためのピアツーピア複製メカニズムが組み込まれています。「*管理ガイド*」で JMS でのピアツーピア複製の構成に関する説明を参照してください。

オブジェクトは、アプリケーションがメッセージ・トランスポートを使用して、ObjectGrid で発生した変更をリモート Java 仮想マシン 内のピア ObjectGrids に簡単にパブリッシュし、それらの変更をその JVM 上で適用するための手段を提供します。LogSequenceTransformer クラスは、このサポートを使用可能にするために重要です。ここでは、メッセージを伝搬するために Java Message Service (JMS) メッセージング・システムを使用するリスナーをどのように作成すればいいのかを詳しく見ていきます。eXtreme Scale トランザクションのコミットが WebSphere Application Server クラスターの複数メンバーにわたって実行された結果として生じる LogSequence の伝送を、eXtreme Scale は IBM 提供のプラグインによってサポートします。この機能はデフォルトでは使用不可ですが、作動可能に構成すること

ができます。ただし、コンシューマーまたはプロデューサーのいずれかが WebSphere Application Server ではない場合、外部 JMS メッセージング・システムが必要となる可能性があります。

メカニズムの実装

LogSequenceTransformer クラス、ObjectGridEventListener、 LogSequence および LogElement API により、信頼性のあるパブリッシュおよびサブスクライブを使用して、変更内容を配布し、配布するマップをフィルターに掛けることができます。このトピックのスニペットは、これらの API を JMS とともに使用して、ピアツーピア ObjectGrid をビルドする方法を示します。これは、共通メッセージ・トランスポートを共用するさまざまなプラットフォームのセットでホストされるアプリケーションによって共用されます。

プラグインの初期化

ObjectGrid が開始するときに、ObjectGrid は、 ObjectGridEventListener インターフェース契約の一部である、プラグインの initialize メソッドを呼び出します。initialize メソッドは、接続、セッション、およびパブリッシャーを含む JMS リソースを取得し、JMS リスナーであるスレッドを開始する必要があります。

以下の例は初期化メソッドを示しています。

```
initialize method examplepublic void initialize(Session session) {
        mySession = session;
        myGrid = session.getObjectGrid();
        try {
            if (mode == null) {
                throw new ObjectGridRuntimeException("No mode specified");
            }
            if (userid != null) {
                connection = topicConnectionFactory.createTopicConnection(userid, password);
            } else
                connection = topicConnectionFactory.createTopicConnection();

            // need to start the connection to receive messages.
            connection.start();

            // the jms session is not transactional (false).
            jmsSession = connection.createTopicSession(false, javax.jms.Session.AUTO_ACKNOWLEDGE);
            if (topic == null)
                if (topicName == null) {
                    throw new ObjectGridRuntimeException("Topic not specified");
                } else {
                    topic = jmsSession.createTopic(topicName);
                }
            publisher = jmsSession.createPublisher(topic);
            // start the listener thread.
            listenerRunning = true;
            listenerThread = new Thread(this);
            listenerThread.start();
        } catch (Throwable e) {
            throw new ObjectGridRuntimeException("Cannot initialize", e);
        }
    }
```

スレッドを開始するためのコードは、Java 2 Platform, Standard Edition (Java SE) スレッドを使用します。WebSphere Application Server バージョン 6.x または WebSphere Application Server バージョン 5.x エンタープライズ・サーバーを実行している場合、非同期 Bean アプリケーション・プログラミング・インターフェース (API) を使用して、このデーモン・スレッドを開始します。共通 API を使用することもできます。以下に、作業マネージャーを使用する同じアクションを示す置換の断片の例を示します。

```
// start the listener thread.
listenerRunning = true;
workManager.startWork(this, true);
```

また、プラグインは、実行可能なインターフェースの代わりに、作業インターフェースを実装する必要があります。また、リリース・メソッドを追加して、listenerRunning 変数を false に設定する必要があります。プラグインは、コンストラクター、または Inversion of Control (IoC) コンテナーを使用している場合は注入によって、 WorkManager インスタンスに提供されている必要があります。

変更の送信

以下は、ObjectGrid 上で行われるローカル変更をパブリッシュするためのサンプル transactionEnd メソッドです。このサンプルでは JMS を使用していますが、信頼できるパブリッシュおよびサブスクライブ・メッセージングの機能を持つ任意のメッセージ・トランスポートを使用することもできます。

```
transactionEnd method example
// This method is synchronized to make sure the
    // messages are published in the order the transaction
    // were committed. If we started publishing the messages
    // in parallel then the receivers could corrupt the Map
    // as deletes may arrive before inserts etc.
    public synchronized void transactionEnd(String txid, boolean isWriteThroughEnabled,
    boolean committed,
            Collection changes) {
        try {
            // must be write through and commited.
            if (isWriteThroughEnabled && committed) {
                // write the sequences to a byte []
                ByteArrayOutputStream bos = new ByteArrayOutputStream();
                ObjectOutputStream oos = new ObjectOutputStream(bos);
                if (publishMaps.isEmpty()) {
                    // serialize the whole collection
                    LogSequenceTransformer.serialize(changes, oos, this, mode);
                } else {
                    // filter LogSequences based on publishMaps contents
                    Collection publishChanges = new ArrayList();
                    Iterator iter = changes.iterator();
                    while (iter.hasNext()) {
                        LogSequence ls = (LogSequence) iter.next();
                        if (publishMaps.contains(ls.getMapName())) {
                            publishChanges.add(ls);
                        }
                    }
                    LogSequenceTransformer.serialize(publishChanges, oos, this, mode);
                }
                // make an object message for the changes
                oos.flush();
                ObjectMessage om = jmsSession.createObjectMessage(bos.toByteArray());
                // set properties
                om.setStringProperty(PROP_TX, txid);
                om.setStringProperty(PROP_GRIDNAME, myGrid.getName());
                // transmit it.
                publisher.publish(om);
            }
        } catch (Throwable e) {
            throw new ObjectGridRuntimeException("Cannot push changes", e);
        }
    }
```

このメソッドは、いくつかのインスタンス変数を使用します。

- jmsSession 変数: メッセージをパブリッシュするために使用する JMS セッションです。これは、プラグインが初期設定される際に作成されます。
- mode 変数: 配布モードです。
- publishMaps 変数: パブリッシュする変更内容を持つ各マップの名前が含まれている集合です。この変数が空の場合、すべてのマップがパブリッシュされます。
- publisher 変数: プラグインの initialize メソッド中に作成される TopicPublisher オブジェクトです。

更新メッセージの受信および適用

以下は実行メソッドです。このメソッドは、アプリケーションがループを停止するまで、ループ内で実行します。各ループの反復は、JMS メッセージの受信、およびメッセージの ObjectGrid への適用を試行します。

```
JMS message run method example
private synchronized boolean isListenerRunning() {
        return listenerRunning;
    }

    public void run() {
        try {
            System.out.println("Listener starting");
            // get a jms session for receiving the messages.
            // Non transactional.
            TopicSession myTopicSession;
            myTopicSession = connection.createTopicSession(false, javax.jms.
       Session.AUTO_ACKNOWLEDGE);

            // get a subscriber for the topic, true indicates don't receive
            // messages transmitted using publishers
            // on this connection. Otherwise, we'd receive our own updates.
            TopicSubscriber subscriber = myTopicSession.createSubscriber(topic,
       null, true);
            System.out.println("Listener started");
            while (isListenerRunning()) {
                ObjectMessage om = (ObjectMessage) subscriber.receive(2000);
                if (om != null) {
                    // Use Session that was passed in on the initialize...
                    // very important to use no write through here
                    mySession.beginNoWriteThrough();
                    byte[] raw = (byte[]) om.getObject();
                    ByteArrayInputStream bis = new ByteArrayInputStream(raw);
                    ObjectInputStream ois = new ObjectInputStream(bis);
                    // inflate the LogSequences
                    Collection collection = LogSequenceTransformer.inflate(ois,
           myGrid);
                    Iterator iter = collection.iterator();
                    while (iter.hasNext()) {
                        // process each Maps changes according to the mode when
                        // the LogSequence was serialized
                        LogSequence seq = (LogSequence) iter.next();
                        mySession.processLogSequence(seq);
                    }
                    mySession.commit();
                } // if there was a message
            } // while loop
            // stop the connection
            connection.close();
        } catch (IOException e) {
            System.out.println("IO Exception: " + e);
        } catch (JMSException e) {
            System.out.println("JMS Exception: " + e);
        } catch (ObjectGridException e) {
            System.out.println("ObjectGrid exception: " + e);
            System.out.println("Caused by: " + e.getCause());
        } catch (Throwable e) {
            System.out.println("Exception : " + e);
        }
        System.out.println("Listener stopped");
    }
```

JMS イベント・リスナー

JMSObjectGridEventListener は、クライアント・サイド・ニア・キャッシュの無効化およびピアツーピア複製メカニズムをサポートするように設計されています。これは、ObjectGridEventListener インターフェースの Java Message Service (JMS) 実装です。

クライアント無効化メカニズムは、クライアントのニア・キャッシュ・データがサーバーまたは他のクライアントと同期するように、分散 eXtreme Scale 環境で使用できます。この機能がないと、クライアントのニア・キャッシュに失効データが保持される可能性があります。ただし、この JMS ベースのクライアント無効化メカニズムを使用しても、クライアント・ニア・キャッシュを更新する場合の時間帯を考慮に入れる必要があります。実行時に更新の公開に遅延があるためです。

ピアツーピア複製メカニズムは、分散とローカルの両方の eXtreme Scale環境で使用できます。これは、ObjectGrid コアツーコア複製プロセスであり、これにより、ローカル ObjectGrid と分散 ObjectGrid の間で流れるデータ更新が可能になります。例えば、このメカニズムを使用すると、分散グリッドからローカル ObjectGrid に、あるいはあるグリッドから別のシステム・ドメインにある別のグリッドに、データ更新を移動させることができます。

JMSObjectGridEventListener の場合、ユーザーは、必要な JMS リソースを取得するために、JMS および Java Naming and Directory Interface (JNDI) 情報を構成する必要があります。さらに、複製関連のプロパティーを正しく設定する必要があります。JEE 環境では、Web と Enterprise JavaBean (EJB) の両方のコンテナーで JNDI が使用可能になっている必要があります。この場合、外部 JMS リソースを取得したい場合を除いて JNDI プロパティーはオプションです。

このイベント・リスナーには、XML を使用するか、プログラマチックな方法を使用して構成できるプロパティーがあります。これは、クライアント無効化のみ、ピアツーピア複製のみ、またはその両方に使用できます。必要な機能を実現するための振る舞いをカスタマイズする場合は、ほとんどのプロパティーはオプションです。

詳しくは、JMSObjectGridEventListener API を参照してください。

JMSObjectGridEventListener プラグインの拡張

JMSObjectGridEventListener プラグインを使用すると、グリッド内のデータが変更または排除されたときに、ピア ObjectGrid インスタンスが更新を受信できます。また、eXtreme Scale グリッドからエントリーが更新または排除されたときに、クライアントが通知を受信することも可能です。このトピックでは、JMS メッセージが受信されたときに、アプリケーションが通知を受け取れるように、JMSObjectGridEventListener プラグインを拡張する方法について説明します。これは、クライアント無効化に CLIENT_SERVER_MODEL 設定を使用する場合に最も役立ちます。

receiver ロールで実行中の場合、JMSObjectGridEventListener インスタンスがグリッドから JMS メッセージ更新を受信すると、オーバーライドされた JMSObjectGridEventListener.onMessage メソッドが eXtreme Scale ランタイムによって自動的に呼び出されます。これらのメッセージは、LogSequence オブジェクトの集まりを折り返します。LogSequence オブジェクトは onMessage メソッドに送られ、アプリケーションはこの LogSequence を使用して挿入、削除、更新、または無効化されたキャッシュ・エントリーを判別します。

onMessage 拡張ポイントを使用するために、アプリケーションは以下のステップを実行します。

1. 新規クラスを作成します。これにより、JMSObjectGridEventListener クラスが拡張され、onMessage メソッドがオーバーライドされます。
2. ObjectGrid の ObjectGridEventListener と同様に拡張 JMSObjectGridEventListener を構成します。

拡張 JMSObjectGridEventListener は、JMSObjectGridEventListener の子クラスであり、initialize (オプション) と onMessage という 2 つのメソッドのみをオーバーライドできます。JMSObjectGridEventListener の子クラスが onMessage メソッド内の

ObjectGrid や Session などの ObjectGrid 成果物を使用する必要がある場合、その子クラスは、initialize メソッドでその成果物を取得して、それをインスタンス変数としてキャッシュできます。また onMessage メソッドでは、キャッシュされた ObjectGrid 成果物は、渡された LogSequences の集まりを処理する場合に使用できます。

注: オーバーライドされた initialize メソッドは、親 JMSObjectGridEventListener を適切に初期化するために、super.initialize メソッドを呼び出す必要があります。

以下は、拡張 JMSObjectGridEventListener クラスの例です。

```
package com.ibm.websphere.samples.objectgrid.jms.price;

import java.util.*;
import com.ibm.websphere.objectgrid.*;
import com.ibm.websphere.objectgrid.plugins.LogElement;
import com.ibm.websphere.objectgrid.plugins.LogSequence;
import com.ibm.websphere.objectgrid.plugins.builtins.JMSObjectGridEventListener;

public class ExtendedJMSObjectGridEventListener extends JMSObjectGridEventListener{
 protected static boolean debug = true;

    /**
     * This is the grid associated with this listener.
     */
    ObjectGrid grid;

    /**
     * This is the session associated with this listener.
     */
    Session session;

    String objectGridType;

    public List receivedLogSequenceList = new ArrayList();


 /* (non-Javadoc)
  * @see com.ibm.websphere.objectgrid.plugins.builtins.JMSObjectGridEventListener
    #initialize(com.ibm.websphere.objectgrid.Session)
  */
 public void initialize(Session session) {
  // Note: if need to use any ObjectGrid artifact, this class need to get ObjectGrid
  //  from the passed Session instance and get ObjectMap from session instance
  // for any transactional ObjectGrid map operation.

  super.initialize(session);  // must invoke super's initialize method.
  this.session = session; // cache the session instance, in case need to
  //   use it to perform map operation.
  this.grid = session.getObjectGrid(); // get ObjectGrid, in case need
  //   to get ObjectGrid information.

  if (grid.getObjectGridType() == ObjectGrid.CLIENT)
   objectGridType = "CLIENT";
  else if (grid.getObjectGridType() == ObjectGrid.SERVER)
   objectGridType = "Server";

  if (debug)
   System.out.println("ExtendedJMSObjectGridEventListener[" +
     objectGridType + "].initialize() : grid = " + this.grid);
 }

 /* (non-Javadoc)
  * @see com.ibm.websphere.objectgrid.plugins.builtins.JMSObjectGridEventListener
    #onMessage(java.util.Collection)
  */
 protected void onMessage(Collection logSequences) {
  System.out.println("ExtendedJMSObjectGridEventListener[" +
    objectGridType + "].onMessage(): ");

  Iterator iter = logSequences.iterator();

  while (iter.hasNext()) {
            LogSequence seq = (LogSequence) iter.next();

      StringBuffer buffer = new StringBuffer();
      String mapName = seq.getMapName();
      int size = seq.size();
      buffer.append("\nLogSequence[mapName=" + mapName + ", size=" + size + ",
     objectGridType=" + objectGridType
        + "]: ");

      Iterator logElementIter = seq.getAllChanges();
```

```
      for (int i = seq.size() - 1; i >= 0; --i) {
       LogElement le = (LogElement) logElementIter.next();
       buffer.append(le.getType() + " -> key=" + le.getCacheEntry().getKey() + ", ");
      }
      buffer.append("\n");

      receivedLogSequenceList.add(buffer.toString());

      if (debug) {
       System.out.println("ExtendedJMSObjectGridEventListener["
      + objectGridType + "].onMessage(): " + buffer.toString());
      }
  }
 }

 public String dumpReceivedLogSequenceList() {
  String result = "";
  int size = receivedLogSequenceList.size();
  result = result + "\nExtendedJMSObjectGridEventListener[" + objectGridType
    + "]: receivedLogSequenceList size = " + size + "\n";
  for (int i = 0; i < size; i++) {
   result = result + receivedLogSequenceList.get(i) + "\n";
  }
  return result;
 }

 public String toString() {
  return "ExtendedJMSObjectGridEventListener["
    + objectGridType + " - " + this.grid + "]";
 }
}
```

構成

拡張 JMSObjectGridEventListener クラスは、クライアント無効化の場合にも、ピアツーピア複製メカニズムの場合にも同様に構成する必要があります。以下は XML 構成の例です。

```
<objectGrid name="PRICEGRID">
   <bean id="ObjectGridEventListener"
    className="com.ibm.websphere.samples.objectgrid.jms.
      price.ExtendedJMSObjectGridEventListener">
    <property name="invalidationModel" type="java.lang.String"
     value="CLIENT_SERVER_MODEL" description="" />
    <property name="invalidationStrategy" type="java.lang.String"
     value="INVALIDATE" description="" />
    <property name="jms_topicConnectionFactoryJndiName" type="java.lang.String"
     value="jms/TCF" description="" />
    <property name="jms_topicJndiName" type="java.lang.String"
     value="GRID.PRICEGRID" description="" />
    <property name="jms_topicName" type="java.lang.String"
     value="GRID.PRICEGRID" description="" />
    <property name="jms_userid" type="java.lang.String" value=""
     description="" />
    <property name="jms_password" type="java.lang.String" value=""
     description="" />
   </bean>
   <backingMap name="PRICE" pluginCollectionRef="PRICE"></backingMap>
      </objectGrid>
```

注: ObjectGridEventListener Bean の className は、一般 JMSObjectGridEventListener と同じプロパティーを持つ拡張 JMSObjectGridEventListener クラスによって構成されます。

XML ファイルを使用した構成

WebSphere eXtreme Scale は、XML ファイルのコレクションにより構成されます。それぞれの XML ファイルが、製品の構成に役立ちます。

デプロイメント・トポロジー構成

デプロイメント・ポリシー記述子 XML を使用して、デプロイメント・トポロジーを管理します。

デプロイメント・ポリシーは、eXtreme Scale コンテナーに提供される XML ファイルとしてエンコードされます。XML ファイルでは、以下の情報が指定されます。

- 各マップ・セットに属するマップ
- 区画の数
- 同期複製および非同期複製の数

デプロイメント・ポリシーでは、以下の配置の振る舞いも制御されます。

- 配置を実行する前のアクティブ・コンテナーの最小数
- 破損した断片の自動置き換え
- 単一区画の各断片の別のマシンへの配置

デプロイメント・ポリシー XML ファイルについて詳しくは、 156 ページの『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。

エンドポイント情報は、動的環境では事前構成されません。デプロイメント・ポリシーには、サーバー名も物理トポロジー情報もありません。動的グリッド内のすべての断片は、カタログ・サービスによって自動的にコンテナーに配置されます。カタログ・サービスは、デプロイメント・ポリシーで定義されている制約を使用して、断片配置を自動的に管理します。この自動断片配置により、大きなグリッドの構成が容易になります。また必要に応じて、使用している環境にサーバーを追加することもできます。

注: WebSphere Application Server 環境では、50 を超えるメンバーが入っているコア・グループ・サイズはサポートされません。

デプロイメント・ポリシー XML ファイルは、始動時に eXtreme Scale コンテナーに渡されます。デプロイメント・ポリシーは、ObjectGrid XML ファイルと一緒に使用する必要があります。デプロイメント・ポリシーはコンテナーの始動には不要ですが、使用されることをお勧めします。デプロイメント・ポリシーは、それと一緒に使用される ObjectGrid XML と互換性がある必要があります。デプロイメント・ポリシー内の objectgridDeployment エレメントごとに、対応する objectGrid が ObjectGrid XML に必要になります。objectgridDeployment 内のマップは、ObjectGrid XML 内の backingMap と整合している必要があります。すべての backingMap は、1 つの mapSet 内のみで参照する必要があります。

以下の例では、`companyGridDpReplication.xml` ファイルは、対応する `companyGrid.xml` ファイルとペアになっていることが想定されています。

companyGridDpReplication.xml

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

 <objectgridDeployment objectgridName="CompanyGrid">
  <mapSet name="mapSet1" numberOfPartitions="11"
   minSyncReplicas="1" maxSyncReplicas="1"
   maxAsyncReplicas="0" numInitialContainers="4">
   <map ref="Customer" />
   <map ref="Item" />
   <map ref="OrderLine" />
   <map ref="Order" />
  </mapSet>
 </objectgridDeployment>

</deploymentPolicy>
```

companyGrid.xml

```
<?xml version="1.0" encoding="UTF-8"?>
<objectGridConfig xmlns:xsi="http://www.w3.org/2001/XMLSchema-instance"
```

```
 xsi:schemaLocation="http://ibm.com/ws/objectgrid/config ../objectGrid.xsd"
 xmlns="http://ibm.com/ws/objectgrid/config">

 <objectGrids>
  <objectGrid name="CompanyGrid">
   <backingMap name="Customer" />
   <backingMap name="Item" />
   <backingMap name="OrderLine" />
   <backingMap name="Order" />
  </objectGrid>
 </objectGrids>

</objectGridConfig>
```

companyGridDpReplication.xml ファイルには、11 個の区画に分割されている mapSet が 1 つあります。各区画に含めることができる同期複製は 1 つだけです。同期複製の数は、minSyncReplicas 属性および maxSyncReplicas 属性によって決まります。 minSyncReplicas 属性が 1 に設定されているため、書き込みトランザクションを処理するためには、mapSet 内の各区画では、少なくとも 1 つの同期複製が使用可能になっている必要があります。maxSyncReplicas が 1 に設定されているため、各区画の同期複製の数は 1 を超えることはできません。この mapSet 内の区画には、非同期複製は含まれていません。

カタログ・サービスは、この ObjectGrid をサポートするために、numInitialContainers 属性に従って、4 つのコンテナーが使用可能になるまで配置を据え置きます。numInitialContainers 属性は、コンテナーが指定数に到達した後は、無視されます。

companyGridDpReplication.xml ファイルはデプロイメント・ポリシーを構成するための一般的な方法を示していますが、デプロイメント・ポリシーによって ObjectGrid を使用環境に配置する方法とその時期をさらに細かく制御することができます。デプロイメント・ポリシー記述子ファイルについて詳しくは、156 ページの『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。

分散トポロジー

分散コヒーレント・キャッシュを使用すると、構成できるパフォーマンス、可用性、およびスケーラビリティーが改善されます。

WebSphere eXtreme Scale は自動的にサーバーのバランスを取ります。 WebSphere eXtreme Scale を再始動しなくても、追加サーバーを組み込むことができます。eXtreme Scale を再始動せずにサーバーを追加できることで、単純なデプロイメントだけでなく、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が可能になります。このデプロイメント・トポロジーは柔軟です。カタログ・サービスを使用すると、キャッシュ全体を除去することなく、サーバーを追加および除去して、リソースを効率的に使用できるようになります。サーバーを追加あるいは除去する場合、 startOgServer コマンドを使用してコンテナー・サーバーを始動し、このコンテナー・サーバーが **-catalogServiceEndPoints** オプションを使用するカタログ・サービスと通信します。分散トポロジーのすべてのクライアントは、Internet Interoperability Object Protocol (IIOP) を介してカタログ・サービスと通信します。すべてのクライアントは、ObjectGrid インターフェースを使用してサーバーと通信できます。

WebSphere eXtreme Scale の動的構成機能を使用すると、簡単にシステムにリソースを追加することができます。コンテナーは、データをホストし、カタログ・サービスは、グリッドのタッチ・ポイントとして機能します。コンテナーは、データの維

持を担当します。カタログ・サービスは、最初の接触時に正しい場所への要求の転送、ホスト・コンテナー内でのスペースの割り振り、およびシステム全体のヘルスと可用性の管理を行います。クライアントは、カタログ・サービスに接続して、コンテナー/サーバー・トポロジーの説明を取得してから、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他のサーバーの障害によってサーバー・トポロジーが変更されると、クライアントは、データをホスティングする適切なサーバーに自動的にルーティングされます。

通常、カタログ・サービスは、自身の Java 仮想マシン 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、別のサーバーの他のコンテナーとともに任意の JVM にロードすることもできます。クライアントはどの JVM 内にも存在でき、1 つ以上のサーバーと通信できます。また、コンテナーと同じ JVM 内に存在することも可能です。

デプロイメント・ポリシー記述子 XML ファイル

デプロイメント・ポリシーを構成するには、デプロイメント・ポリシー記述子 XML ファイルを使用します。

以下のセクションでは、デプロイメント・ポリシー記述子 XML ファイルのエレメントおよび属性が定義されます。デプロイメント・ポリシー XML スキーマの例は、160 ページの『deploymentPolicy.xsd ファイル』を参照してください。

deploymentPolicy エレメント

deploymentPolicy エレメントは、デプロイメント・ポリシー XML ファイルの最上位エレメントです。このエレメントは、ファイルの名前空間とスキーマ・ロケーションをセットアップします。スキーマは deploymentPolicy.xsd ファイルで定義されます。

- 出現回数: 1 回
- 子エレメント: objectgridDeployment エレメント

objectgridDeployment エレメント

objectgridDeployment エレメントは、ObjectGrid XML ファイルの ObjectGrid インスタンスを参照する場合に使用します。objectgridDeployment エレメント内では、マップをマップ・セットに分割できます。

- 出現回数: 1 回以上
- 子エレメント: mapSet エレメント

属性

objectgridName
デプロイする ObjectGrid の名前を指定します。この属性は、ObjectGrid XML ファイルで定義されている objectGrid エレメントを参照します。(必須)

```
<objectgridDeployment objectgridName="objectgridName"/>
```

例えば、objectgridName 属性は、companyGridDpReplication.xml ファイル内で CompanyGrid として設定されています。objectgridName 属性は、companyGrid.xml ファイルに定義されている CompanyGrid を参照します。詳しくは、161 ページの

『ObjectGrid 記述子 XML ファイル』を参照してください。

mapSet エレメント

mapSet エレメントは複数のマップをまとめてグループ化するために使用されます。mapSet エレメント内のマップは同じように区画に分割され、複製されます。各マップは、単一の mapSet のみに属している必要があります。

- 出現回数: 1 回以上
- 子エレメント: map エレメント

属性

name
MapSet の名前を指定します。この属性は、objectgridDeployment エレメント内では固有である必要があります。(必須)

numberOfPartitions
MapSet 用の区画の数を指定します。デフォルト値は 1 です。この数は、区画をホストしているコンテナーの数に適した値である必要があります。(オプション)

minSyncReplicas
同期複製の最小数を MapSet 内の区画ごとに指定します。デフォルト値は 0 です。断片は、ドメインが最小数の同期複製をサポートできるようになるまで配置されません。 minSyncReplicas 値をサポートするには、minSyncReplicas の値よりも 1 つだけ多いコンテナーが必要です。同期複製の数が minSyncReplicas の値よりも小さくなると、その区画に対しては書き込みトランザクションを行えなくなります。(オプション)

maxSyncReplicas
同期複製の最大数を MapSet 内の区画ごとに指定します。デフォルト値は 0 です。ドメインがある特定の区画でこの同期複製数に達すると、その区画に対しては他の同期複製は配置されません。まだ maxSyncReplicas 値を満たしていない場合には、この ObjectGrid をサポートできるコンテナーを追加すると、同期複製の数を増やすことができます。(オプション)

maxAsyncReplicas
非同期複製の最大数を MapSet 内の区画ごとに指定します。デフォルト値は 0 です。ある区画に対してプライマリーおよびすべての同期複製が配置された後は、maxAsyncReplicas 値になるまで、非同期複製が配置されます。(オプション)

replicaReadEnabled
この属性が `true` に設定されている場合、プライマリー区画とそのレプリカに読み取り要求が配布されます。 replicaReadEnabled 属性が `false` の場合は、読み取り要求はプライマリーにのみ送付されます。デフォルトは false です。(オプション)

numInitialContainers
この mapSet 内の断片に対して初期配置が行われる前に必要となる eXtreme Scale コンテナーの数を指定します。デフォルト値は 1 です。この属性は、ObjectGrid がオンラインになったときに CPU とネットワーク帯域幅を節約するのに役立ちます。(オプション)

eXtreme Scale コンテナーを開始すると、カタログ・サービスにイベントが送信されます。アクティブ・コンテナーの数が初めて mapSet の numInitialContainers 値と等しくなり、さらに minSyncReplicas も満たすことができる場合には、カタログ・サービスは mapSet の断片を配置します。numInitialContainers 値を満たすと、コンテナーによって始動された各イベントは、未配置の断片と既に配置されている断片の再バランシングを引き起こす場合があります。この mapSet に対して開始する予定のコンテナーのおおよその数がわかっている場合は、その数に近い numInitialContainers 値を設定して、すべてのコンテナーが開始した後の再バランシングを回避することができます。配置は、mapSet の numInitialContainers 値に達するまでは行われません。

autoReplaceLostShards
失われた断片がその他のコンテナーに配置されるかどうかを指定します。デフォルトは true です。コンテナーが停止するか、障害を発生すると、コンテナーで実行中の断片が失われます。失われたプライマリーが持っているレプリカの 1 つが新規のプライマリーにプロモートします。このプロモーションのため、レプリカの 1 つが失われます。失われた断片が使用可能なコンテナー内に自動的に置き換えられるようにしたくない場合があります。失われた断片を未配置のままにしておく場合は、autoReplaceLostShards 属性を `false` に設定します。この設定はプロモーション・チェーンには影響せず、そのチェーン内の最後の断片の置き換えにのみ影響します。(オプション)

developmentMode
この属性を使用すると、ある断片をそのピア断片との関係でどこに配置するかを制御できます。デフォルトは true です。developmentMode 属性が `false` に設定されている場合は、同じ区画の 2 つの断片は同じマシンには配置されません。developmentMode 属性が `true` に設定されている場合は、同じ区画の断片を同じマシンに配置することができます。いずれの場合も、同一の区画の 2 つの断片は、同一のコンテナーには配置されません。(オプション)

placementStrategy
配置ストラテジーには 2 つあります。デフォルトのストラテジーは `FIXED_PARTITION` です。この場合、使用可能なコンテナー全体で配置されるプライマリー断片の数は、定義されている区画数にレプリカ数を加えたものになります。代替のストラテジーは `PER_CONTAINER` です。この場合、各コンテナーに配置されるプライマリー断片の数は、定義されている区画数になり、同じ数のレプリカが他のコンテナーに配置されます。 (オプション)

```
<mapSet
(1)   name="mapSetName"
(2)   numberOfPartitions="numberOfPartitions"
(3)   minSyncReplicas="minimumNumber"
(4)    maxSyncReplicas="maximumNumber"
(5)   maxAsyncReplicas="maximumNumber"
(6)   replicaReadEnable="true" | "false"
(7)   numInitialContainers="numberOfInitialContainersBeforePlacement"
(8)   autoReplaceLostShards="true" | "false"
(9)   developmentMode="true" | "false"
(10)   placementStrategy="FIXED_PARTITION"|"PER_CONTAINER"
/>
```

以下の例では、mapSet エレメントがデプロイメント・ポリシーを構成するために使用されています。値は、`mapSet1` に設定され、10 区画に分割されています。これらの分割のそれぞれにおいて、少なくとも 1 つの同期複製と、2 つ以下の同期複製が使用可能になっている必要があります。環境が非同期複製をサポートできる場合は、各区画には非同期複製も含まれています。すべての同期複製は、非同期複製が

配置される前に配置されます。さらに、ドメインが minSyncReplicas 値をサポートできるようになるまでは、カタログ・サービスは mapSet1 の断片を配置しません。minSyncReplicas 値をサポートするには、複数のコンテナー、すなわちプライマリー用に 1 つ、同期複製用に 2 つのコンテナーが必要です。

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

 <objectgridDeployment objectgridName="CompanyGrid">
  <mapSet name="mapSet1" numberOfPartitions="10"
   minSyncReplicas="1" maxSyncReplicas="2" maxAsyncReplicas="1"
    numInitialContainers="10" autoReplaceLostShards="true"
   developmentMode="false" replicaReadEnabled="true">
   <map ref="Customer"/>
   <map ref="Item"/>
   <map ref="OrderLine"/>
   <map ref="Order"/>
  </mapSet>
 </objectgridDeployment>

</deploymentPolicy>
```

複製の設定を満たすのにコンテナーが 2 つしか必要ない場合でも、numInitialContainers 属性は、カタログ・サービスがこの mapSet に断片を配置する前に、使用可能なコンテナーを 10 個要求します。ドメインが、CompanyGrid ObjectGrid をサポートできるコンテナーを 10 個持つと、mapSet1 内のすべての断片が配置されます。

autoReplaceLostShards 属性が true に設定されているため、この mapSet 内の、コンテナー障害のために失われた断片は、失われた断片をコンテナーがホストできる場合には、自動的に別のコンテナーに置き換えられます。同じ区画内の断片は、mapSet1 の同じマシンには配置できません。developmentMode 属性が `false` に設定されているためです。読み取り専用要求は、区画ごとにプライマリーとそのレプリカ全体に配布されます。これは、replicaReadEnabled 値が `true` だからです。

map エレメント

mapSet エレメント内の各 map エレメントは、ObjectGrid XML ファイルで定義されている backingMap エレメントの 1 つを参照します。分散 eXtreme Scale 内のすべてのマップは、単一の mapSet エレメントに属している必要があります。ObjectGrid XML ファイルに関して詳しくは、179 ページの『`objectGrid.xsd` ファイル』を参照してください。

- 出現回数: 1 回以上
- 子エレメント: なし

属性

ref ObjectGrid XML ファイル内の backingMap エレメントを参照できるようにします。mapSet 内の各マップは、ObjectGrid XML ファイルの backingMap を参照する必要があります。ref 属性に割り当てられている値は、ObjectGrid XML ファイル内の backingMap エレメントの 1 つの name 属性に一致している必要があります。(必須)

```
<map
(1)   ref="backingMapReference"
/>
```

`companyGridDpMapSetAttr.xml` ファイルは、この map エレメントの ref 属性を使用して、`companyGrid.xml` ファイルの各 backingMap を参照します。

XML 構成矛盾の回避については、 89 ページの『XML 構成のトラブルシューティング』を参照してください。

deploymentPolicy.xsd ファイル:

デプロイメント・ポリシー XML スキーマを使用すると、デプロイメント記述子 XML ファイルを作成できます。

deploymentPolicy.xsd ファイルに定義されるエレメントおよび属性の説明は、 156 ページの『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。

```
<?xml version="1.0" encoding="UTF-8" ?>
<xsd:schema xmlns:dp="http://www.ibm.com/ws/objectgrid/deploymentPolicy"
 xmlns:xsd="http://www.w3.org/2001/XMLSchema"
 targetNameSpace="http://ibm.com/ws/objectgrid/deploymentPolicy"
 elementFormDefault="qualified">

 <xsd:element name="deploymentPolicy">
  <xsd:complexType>
   <xsd:choice>
    <xsd:element name="objectgridDeployment"
     type="dp:objectgridDeployment" minOccurs="1"
     maxOccurs="unbounded">
     <xsd:unique name="mapSetNameUnique">
      <xsd:selector xpath="dp:mapset" />
      <xsd:field xpath="@name" />
     </xsd:unique>
    </xsd:element>
   </xsd:choice>
  </xsd:complexType>
 </xsd:element>

 <xsd:complexType name="objectgridDeployment">
  <xsd:sequence>
   <xsd:element name="mapSet" type="dp:mapSet"
    maxOccurs="unbounded" minOccurs="1">
    <xsd:unique name="mapNameUnique">
     <xsd:selector xpath="dp:map" />
     <xsd:field xpath="@ref" />
    </xsd:unique>
   </xsd:element>
  </xsd:sequence>
  <xsd:attribute name="objectgridName" type="xsd:string"
   use="required" />
 </xsd:complexType>

 <xsd:complexType name="mapSet">
  <xsd:sequence>
   <xsd:element name="map" type="dp:map" maxOccurs="unbounded"
    minOccurs="1" />
   <xsd:element name="zoneMetadata" type="dp:zoneMetadata"
    maxOccurs="1" minOccurs="0">

    <xsd:key name="zoneRuleName">
     <xsd:selector xpath="dp:zoneRule" />
     <xsd:field xpath="@name" />
    </xsd:key>

    <xsd:keyref name="zoneRuleRef"
     refer="dp:zoneRuleName">
     <xsd:selector xpath="dp:shardMapping" />
     <xsd:field xpath="@zoneRuleRef" />
    </xsd:keyref>

   </xsd:element>
  </xsd:sequence>
  <xsd:attribute name=="name" type="xsd:string" use="required" />
  <xsd:attribute name="numberOfPartitions" type="xsd:int"
   use="optional" />
  <xsd:attribute name="minSyncReplicas" type="xsd:int"
   use="optional" />
  <xsd:attribute name="maxSyncReplicas" type="xsd:int"
   use="optional" />
  <xsd:attribute name="maxAsyncReplicas" type="xsd:int"
   use="optional" />
  <xsd:attribute name="replicaReadEnabled" type="xsd:boolean"
   use="optional" />
  <xsd:attribute name="numInitialContainers" type="xsd:int"
   use="optional" />
  <xsd:attribute name="autoReplaceLostShards" type="xsd:boolean"
   use="optional" />
  <xsd:attribute name="developmentMode" type="xsd:boolean"
   use="optional" />
```

```
 <xsd:attribute name="placementStrategy"
  type="dp:placementStrategy" use="optional" />
 </xsd:complexType>

 <xsd:simpleType name="placementStrategy">
  <xsd:restriction base="xsd:string">
   <xsd:enumeration value="FIXED_PARTITIONS" />
   <xsd:enumeration value="PER_CONTAINER" />
  </xsd:restriction>
 </xsd:simpleType>


 <xsd:complexType name="map">
  <xsd:attribute name="ref" use="required" />
 </xsd:complexType>

 <xsd:complexType name="zoneMetadata">
  <xsd:sequence>
   <xsd:element name="shardMapping" type="dp:shardMapping"
    maxOccurs="unbounded" minOccurs="1" />
   <xsd:element name="zoneRule" type="dp:zoneRule"
    maxOccurs="unbounded" minOccurs="1">

   </xsd:element>

  </xsd:sequence>
 </xsd:complexType>

 <xsd:complexType name="shardMapping">
  <xsd:attribute name="shard" use="required">
   <xsd:simpleType>
    <xsd:restriction base="xsd:string">
     <xsd:enumeration value="P"></xsd:enumeration>
     <xsd:enumeration value="S"></xsd:enumeration>
     <xsd:enumeration value="A"></xsd:enumeration>
    </xsd:restriction>
   </xsd:simpleType>
  </xsd:attribute>
  <xsd:attribute name="zoneRuleRef" type="xsd:string"
   use="required" />
 </xsd:complexType>

 <xsd:complexType name="zoneRule">
  <xsd:sequence>
   <xsd:element name="zone" type="dp:zone"
    maxOccurs="unbounded" minOccurs="1" />
  </xsd:sequence>
  <xsd:attribute name="name" type="xsd:string" use="required" />
  <xsd:attribute name="exclusivePlacement" type="xsd:boolean" />
   use="optional" />
 </xsd:complexType>

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

</xsd:schema>
```

ObjectGrid 記述子 XML ファイル

WebSphere eXtreme Scale を構成するには、ObjectGrid ディスクリプター XML ファイルおよび ObjectGrid API を使用します。

以下のセクションでは、さまざまな構成を解説するサンプル XML ファイルが提供されています。XML ファイルの各エレメントおよび属性について定義されています。 ObjectGrid 記述子 XML スキーマを使用して、記述 XML ファイルを作成します。ObjectGrid 記述子 XML スキーマの例については、179 ページの『objectGrid.xsd ファイル』 を参照してください。

オリジナルの companyGrid.xml ファイルを変更したバージョンが使用されます。次の companyGridSingleMap.xml ファイルは、companyGrid.xml ファイルによく似ています。companyGridSingleMap.xml ファイルにはマップが 1 つあり、companyGrid.xml ファイルにはマップが 4 つあります。ファイルのエレメントと属性については、この例に続いて詳しく説明します。

```
<?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="CompanyGrid">
      <backingMap name="Customer"/>
    </objectGrid>
  </objectGrids>
</objectGridConfig>
```

objectGridConfig エレメント

objectGridConfig エレメントは、XML ファイルの最上位レベルのエレメントです。前の例で示されているように、このエレメントを eXtreme Scale XML 文書に記述します。このエレメントは、ファイルの名前空間とスキーマ・ロケーションをセットアップします。スキーマは objectGrid.xsd ファイルで定義されます。

- 出現回数: 1 回
- 子エレメント: objectGrids エレメントおよび backingMapPluginCollections エレメント

objectGrids エレメント

objectGrids エレメントは、すべての objectGrid エレメントのコンテナーです。companyGridSingleMap.xml ファイルでは、objectGrids エレメントに CompanyGrid という objectGrid が 1 つ含まれています。

- 出現回数: 1 回以上
- 子エレメント: objectGrid エレメント

objectGrid エレメント

objectGrid エレメントを使用して ObjectGrid を定義します。objectGrid エレメント上の各属性は、ObjectGrid インターフェース上のメソッドに対応します。

- 出現回数: 1 回から複数回
- 子エレメント: Bean エレメント、backingMap エレメント、querySchema エレメント、および streamQuerySet エレメント

属性

name

ObjectGrid に割り当てられている名前を指定します。この属性が欠落している場合、XML 妥当性検査は失敗します。(必須)

securityEnabled

属性を true に設定したとき、ObjectGrid レベルのセキュリティーを使用可能にします。これにより、マップ内のデータに対するアクセス許可が使用可能になります。デフォルトは true です。 (オプション)

authorizationMechanism

エレメントの許可メカニズムを設定します。この属性は 2 つの値 (AUTHORIZATION_MECHANISM_JAAS または AUTHORIZATION_MECHANISM_CUSTOM) のいずれかに設定できます。デフォルトは AUTHORIZATION_MECHANISM_JAAS です。カスタム MapAuthorization プラグインを使用する場合、AUTHORIZATION_MECHANISM_CUSTOM に設定します。authorizationMechanism 属性を有効にするためには、securityEnabled 属性を true に設定する必要があります。(オプション)

permissionCheckPeriod

クライアント・アクセスを許可するために使用されるアクセス権の検査頻度を、秒単位の整数値で指定します。デフォルトは 0 です。属性値 0 を設定すると、すべての get、put、update、remove または evict メソッド呼び出しが許可メカニズム (Java Authentication and Authorization Service (JAAS) 権限またはカスタム権限のいずれか) に問い合わせ、現行サブジェクトがアクセス権を持っているかどうかを検査します。0 より大きな値は、アクセス権のセットを、更新のために許可メカニズムへ戻す前に、キャッシュに入れる秒数を示します。

permissionCheckPeriod 属性を有効にするためには、securityEnabled 属性を true に設定する必要があります。(オプション)

txTimeout

トランザクションを完了するのに許されている時間を秒で指定します。トランザクションがこの時間内に完了しなかった場合、そのトランザクションはロールバック対象としてマークされ、TransactionTimeoutException 例外が発生します。値 0 を設定すると、トランザクションがタイムアウトになることはありません。(オプション)

entityMetadataXMLFile

エンティティー・ディスクリプター XML ファイルへの相対パスを指定します。このパスは、Objectgrid® ディスクリプター・ファイルのロケーションを相対的に示します。この属性を XML ファイルで使用してエンティティー・スキーマを定義します。eXtreme Scale は、エンティティーを定義してから開始する必要があります。こうすることで、各エンティティーを BackingMap にバインドできます。(オプション)

```
<objectGrid
(1) name="objectGridName"
(2) securityEnabled="true" | "false"
(3) authorizationMechanism="AUTHORIZATION_MECHANISM_JASS" | "AUTHORIZATION_MECHANISM_CUSTOM"
(4)  permissionCheckPeriod="permission_check_period"
(5) txTimeout="seconds"
(6) entityMetadataXMLFile="URL"
/>
```

以下の例 (companyGridObjectGridAttr.xml ファイル) は、objectGrid エレメントの属性を構成するための 1 つの方法を説明するものです。セキュリティーは使用可能にされ、許可メカニズムは JAAS に設定され、アクセス権検査期間は 45 秒に設定されています。また、このファイルでは entityMetadataXMLFile 属性を指定してエンティティーを登録しています。

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

 <objectGrids>
  <objectGrid name="CompanyGrid" securityEnabled="true"
   authorizationMechanism="AUTHORIZATION_MECHANISM_JASS"
   permissionCheckPeriod="45"
   entityMetadataXMLFile="companyGridEntities.xml">
   <backingMap name="Customer"/>
  </objectGrid>
 </objectGrids>
</objectGridConfig>
```

以下のコード・サンプルは、前の例の companyGridObjectGridAttr.xml ファイルと同じ構成を達成するプログラマチックな方法を示すものです。

```
ObjectGridManager objectGridManager = ObjectGridManagerFactory.getObjectGridManager();
ObjectGrid companyGrid = objectGridManager.createObjectGrid("CompanyGrid", false);
```

```
companyGrid.setSecurityEnabled();
companyGrid.setAuthorizationMechanism(SecurityConstants.AUTHORIZATION_MECHANISM_JAAS);
companyGrid.setPermissionCheckPeriod(45);
companyGrid.registerEntities(new URL("file:companyGridEntities.xml"));
```

backingMap エレメント

backingMap エレメントは、ObjectGrid で BackingMap インスタンスを定義するために使用します。backingMap エレメント上の各属性は、BackingMap インターフェース上のメソッドに対応します。詳しくは、69 ページの『BackingMap インターフェース』を参照してください。

- 出現回数: 0 回から複数回
- 子エレメント: timeBasedDBUpdate エレメント

属性

name
backingMap インスタンスに割り当てられている名前を指定します。この属性が欠落している場合、XML 妥当性検査は失敗します。 (必須)

readOnly
この属性を `false` として指定すると、BackingMap インスタンスを読み取り/書き込みとして設定します。この属性を `true` として設定すると、BackingMap インスタンスは読み取り専用となります。 Session.getMap(String) を呼び出すと、メソッドに渡された名前が、この backingMap の name 属性に指定された正規表現に一致する場合、動的マップ作成が行われます。デフォルト値は `false` です。(オプション)

template
動的マップが使用できるかどうかを指定します。BackingMap マップがテンプレート・マップである場合、この値を `true` に設定します。テンプレート・マップを使用して、ObjectGrid の開始後に動的にマップを作成できます。
Session.getMap(String) を呼び出すと、メソッドに渡された名前が、backingMap の name 属性に指定された正規表現に一致する場合、動的マップが作成されます。デフォルト値は `false` です。(オプション)

pluginCollectionRef
backingMapPluginCollection プラグインへの参照を指定します。この属性の値は、backingMapCollection プラグインの ID 属性と一致しなければなりません。一致する ID が存在しない場合、妥当性検査は失敗します。BackingMap プラグインを再利用するように、この属性を設定してください。(オプション)

numberOfBuckets
BackingMap インスタンスが使用するバケットの数を指定します。 BackingMap インスタンスは実装でハッシュ・マップを使用します。 BackingMap の中に複数のエントリーが存在する場合は、バケット数が増えると衝突のリスクが低減するため、バケットの数が多くなるほどパフォーマンスが向上します。また、バケットが多いと並行性も高くなります。値 `0` を設定すると、eXtreme Scale とのリモート通信中クライアントのニア・キャッシュが使用不可になります。クライアントでこの値を `0` に設定するときは、この値をクライアント・オーバーライド ObjectGrid XML ディスクリプター・ファイルでのみ設定してください。(オプション)

preloadMode
ローダー・プラグインがこの BackingMap インスタンスに設定されている場合は、プリロード・モードを設定します。デフォルト値は `false` です。属性が `true` に設定されている場合は、Loader.preloadMap(Session, BackingMap) メソッドが非同期に起動されます。それ以外の場合は、プリロードが完了するまでキャッシュが使用不可になるように、データのロード中、メソッドの実行がブロックされます。プリロードは初期化中に発生します。 (オプション)

lockStrategy
トランザクションがマップ・エントリーにアクセスするたびに内部ロック・マネージャーを使用するかどうかを指定します。この属性は、`OPTIMISTIC`、`PESSIMISTIC` または `NONE` の 3 つの値のいずれかに設定できます。デフォルト値は `OPTIMISTIC` です。 (オプション)

オプティミスティック・ロック・ストラテジーは、通常、ローダー・プラグインを持たないマップで使用します。このようなマップは大部分が既読で、書き込みや更新は頻繁には行われません。また、eXtreme Scale をサイド・キャッシュとして使用するパーシスタンス・マネージャーからも、アプリケーションからもロックは提供されません。コミット時に挿入、更新、または除去されるマップ・エントリー上では排他ロックが取得されます。コミット中のトランザクションがオプティミスティック・バージョン・チェックを実行している間、別のトランザクションによってバージョン情報が変更されないことが、ロックによって保証されます。

ペシミスティック・ロック・ストラテジーは、通常、ローダー・プラグインを持たないマップで使用します。また、eXtreme Scale をサイド・キャッシュとして使用するパーシスタンス・マネージャーからも、ローダー・プラグインからも、アプリケーションからもロックは提供されません。ペシミスティック・ロック・ストラテジーは、同じマップ・エントリー上で更新トランザクションが頻繁に衝突を起こすことが原因で、オプティミスティック・ロック・ストラテジーの失敗回数が多すぎる場合に使用されます。

ロックなしストラテジーは、内部ロック・マネージャーが不要であることを示します。 eXtreme Scale の外部では、eXtreme Scale をサイド・キャッシュとして使用するパーシスタンス・マネージャー、またはアプリケーション、またはデータベース・ロックを使用して並行性を制御するローダー・プラグインのいずれかによって並行性制御が提供されます。

詳しくは、 74 ページの『マップ・エントリー・ロック』を参照してください。

numberOfLockBuckets
ロック・マネージャーが BackingMap インスタンスのために使用するロック・バケットの数を設定します。 lockStrategy 属性を `OPTIMISTIC` または `PESSIMISTIC` に設定すると、BackingMap インスタンス用のロック・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。多くのエントリーが存在する場合は、バケット数が増えるにつれて衝突のリスクが低減するため、ロック・バケットが多い方がパフォーマンスが向上します。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。 lockStrategy 属性を `NONE` に設定すると、BackingMap インスタンスがロック・マネージャーを使用しないことが指定されます。(オプション)

lockTimeout

ロック・マネージャーが BackingMap インスタンスのために使用するロック・タイムアウト設定します。 lockStrategy 属性を OPTIMISTIC または PESSIMISTIC に設定すると、BackingMap インスタンス用のロック・マネージャーが作成されます。デッドロックの発生を回避するために、ロック・マネージャーにはデフォルトのタイムアウト値 (15 秒) があります。このタイムアウト制限を超過すると、LockTimeoutException 例外が発生します。デフォルト値の 15 秒は、ほとんどのアプリケーションに対して適切ですが、負荷の過剰なシステム上では、デッドロックが存在しないときにタイムアウトが発生する可能性があります。 lockTimeout 属性を使用してデフォルトより大きい値を設定すれば、誤ったタイムアウト例外が発生しないようにすることができます。 lockStrategy 属性を NONE に設定すると、BackingMap インスタンスがロック・マネージャーを使用しないことが指定されます。(オプション)

CopyMode

BackingMap インスタンス内のエントリーの get 操作が実際の値、その値のコピー、またはその値のプロキシーを戻すかどうかを指定します。 CopyMode 属性を次の 5 つの値のいずれかに設定します。

COPY_ON_READ_AND_COMMIT

デフォルト値は COPY_ON_READ_AND_COMMIT です。値を COPY_ON_READ_AND_COMMIT に設定すると、アプリケーションが BackingMap インスタンス内にある値オブジェクトへの参照を持たないようにすることができます。代わりに、アプリケーションは BackingMap インスタンス内にある値のコピーを常に使用します。(オプション)

COPY_ON_READ

値を COPY_ON_READ に設定すると、トランザクションがコミットされたときに発生するコピーを除去することによって、COPY_ON_READ_AND_COMMIT 値以上にパフォーマンスを向上させることができます。 BackingMap データの整合性を保持するため、アプリケーションは、トランザクションがコミットされた後、エントリーに対するすべての参照を削除します。この値を設定すると、ObjectMap.get メソッドは、値に対する参照の代わりにその値のコピーを戻し、トランザクションがコミットされるまで、アプリケーションによってその値に行われた変更が BackingMap エレメントに影響を与えないことを保証します。

COPY_ON_WRITE

値を COPY_ON_WRITE に設定すると、指定したキーのトランザクションによって ObjectMap.get メソッドが初めて呼び出されたときに発生するコピーを除去することにより、COPY_ON_READ_AND_COMMIT 値以上にパフォーマンスを向上させることができます。その代わりに、ObjectMap.get メソッドは、値オブジェクトを直接参照するのではなく、その値にプロキシーを戻します。プロキシーは、アプリケーションが値インターフェース上に set メソッドを呼び出さない限りは、その値のコピーを作成しないことを保証します。

NO_COPY

値を NO_COPY に設定すると、アプリケーションは、ObjectMap.get メソッドを使用して取得した値オブジェクトをパフォーマンス向上と交換に

変更しないようにすることができます。 EntityManager API エンティティーに関連付けられているマップの場合は、値を NO_COPY に設定してください。

COPY_TO_BYTES
値を COPY_TO_BYTES に設定すると、オブジェクトのコピー処理がコピー作成のシリアライゼーションに依存している場合に、複雑なオブジェクト・タイプのメモリー・フットプリントを改善し、パフォーマンスを改善します。オブジェクトが Cloneable でないか、あるいは、効率的な copyValue メソッドを使用するカスタム ObjectTransformer が指定されていない場合、デフォルトのコピー・メカニズムは、コピー作成のためにオブジェクトをシリアライズしてインフレーションします。
COPY_TO_BYTES 設定を使用すると、読み取り時にのみインフレーションが実行されて、コミット時にのみシリアライズが実行されます。

これらの設定について詳しくは、「*プログラミング・ガイド*」で CopyMode のベスト・プラクティスの情報を参照してください。

valueInterfaceClassName
CopyMode 属性を COPY_ON_WRITE に設定する際に必要なクラスを指定します。この属性は、それ以外のモードでは無視されます。ObjectMap.get メソッド呼び出しが行われると、COPY_ON_WRITE 値はプロキシーを使用します。プロキシーは、アプリケーションが valueInterfaceClassName 属性として指定されたクラス上に set メソッドを呼び出さない限りは、その値のコピーを作成しないことを保証します。 (オプション)

copyKey
マップ・エントリーの作成時にキーのコピーが必要であるかどうかを指定します。キーのコピーによって、アプリケーションがそれぞれの ObjectMap 操作に対して同じキー・オブジェクトを使用することが可能になります。値を true に設定すると、マップ・エントリーの作成時にキー・オブジェクトがコピーされます。デフォルト値は false です。(オプション)

nullValuesSupported
値を true に設定すると、ObjectMap でヌル値がサポートされます。ヌル値がサポートされていると、NULL を戻す get 操作は、値が NULL であること、またはメソッドに渡されるキーがマップに含まれていないことを意味する場合があります。デフォルト値は true です。(オプション)

ttlEvictorType
BackingMap エントリーの有効期限の時間を算出する方法を指定します。この属性は、CREATION_TIME、LAST_ACCESS_TIME、または NONE の 3 つの値のいずれかに設定できます。CREATION_TIME 値の場合、エントリーの有効期限の時間は、このエントリーの作成時間と timeToLive 属性値の合計になります。
LAST_ACCESS_TIME 値の場合、エントリーの有効期限の時間は、このエントリーの最終アクセス時間と timeToLive 属性値の合計になります。 NONE 値 (デフォルト) の場合、エントリーには有効期限がなく、アプリケーションによって明示的に除去または無効化されるまで、BackingMap 内に存続できることを示します。(オプション)

timeToLive
各マップ・エントリーの存続可能時間 (秒数) を指定します。デフォルト値の 0

は、このマップ・エントリーが永久に存続するか、アプリケーションが明示的にこのエントリーを除去または無効化するまで存続することを意味します。その他の場合は、TTL Evictor がこの値に基づいてマップ・エントリーを除去します。(オプション)

streamRef

backingMap がストリーム・ソース・マップであることを指定します。backingMap に対するすべての挿入または更新は、ストリーム照会エンジンに対するストリーミング・イベントに変換されます。この属性は、streamQuerySet 内の有効なストリーム名を参照する必要があります。(オプション)

viewRef

backingMap がビュー・マップであることを指定します。ストリーム照会エンジンからのビュー出力は、eXtreme Scale タプル・フォーマットに変換され、マップに追加されます。(オプション)

evictionTriggers

追加使用する除去トリガーのタイプを設定します。バックアップ・マップ用の Evictor は、すべてこの追加トリガーのリストを使用します。IllegalStateException を回避するためには、ObjectGrid.initialize() メソッドを呼び出す前にこの属性を呼び出す必要があります。また、ObjectGrid.initialize() メソッドがアプリケーションによってまだ呼び出されていない場合は、ObjectGrid.getSession() メソッドがこのメソッドを暗黙的に呼び出すので、注意してください。トリガー・リスト内のエントリーはセミコロンで区切られます。現在の除去トリガーには MEMORY_USAGE_THRESHOLD などがあります。(オプション)

```
<backingMap
(1)   name="objectGridName"
(2)   readOnly="true" | "false"
(3)   template="true" | "false"
(4)   pluginCollectionRef="reference to backingMapPluginCollection"
(5)   numberOfBuckets="number of buckets"
(6)   preloadMode="true" | "false"
(7)   lockStrategy="OPTIMISTIC" | "PESSIMISTIC" | "NONE"
(8)   numberOfLockBuckets="number of lock buckets"
(9)   lockTimeout="lock timeout"
(10)   copyMode="COPY_ON_READ_AND_COMMIT" | "COPY_ON_READ" | "COPY_ON_WRITE"
       | "NO_COPY" | "COPY_TO_BYTES"
(11)   valueInterfaceClassName="value interface class name"
(12)   copyKey="true" | "false"
(13)   nullValuesSupported="true" | "false"
(14)   ttlEvictorType="CREATION_TIME" | "LAST_ACCESS_TIME|NONE"
(15)   timeToLive="time to live"
(16)   streamRef="reference to a stream"
(17)   viewRef="reference to a view"
(18)   writeBehind="write-behind parameters"
(19)   evictionTriggers="MEMORY_USAGE_THRESHOLD"
/>
```

次の例では、サンプルの backingMap 構成を例示するために companyGridBackingMapAttr.xml ファイルが使用されています。

```
<?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="CompanyGrid">
    <backingMap name="Customer" readOnly="true"
     numberOfBuckets="641" preloadMode="false"
     lockStrategy="OPTIMISTIC" numberOfLockBuckets="409"
     lockTimeout="30" copyMode="COPY_ON_WRITE"
     valueInterfaceClassName="com.ibm.websphere.samples.objectgrid.CounterValueInterface"
```

```
    copyKey="true" nullValuesSupported="false"
    ttlEvictorType="LAST_ACCESS_TIME" timeToLive="3000"/>
  </objectGrid>
 </objectGrids>
</objectGridConfig>
```

以下のサンプル・コードは、前述の companyGridBackingMapAttr.xml ファイルの例と同じ構成を達成するプログラマチックな方法を示すものです。

```
ObjectGridManager objectGridManager = ObjectGridManagerFactory.getObjectGridManager();
ObjectGrid companyGrid = objectGridManager.createObjectGrid("CompanyGrid", false);

BackingMap customerMap = companyGrid.defineMap("Customer");
customerMap.setReadOnly(true);
customerMap.setNumberOfBuckets(641);
customerMap.setPreloadMode(false);
customerMap.setLockStrategy(LockStrategy.OPTIMISTIC);
customerMap.setNumberOfLockBuckets(409);
customerMap.setLockTimeout(30);

// when setting copy mode to COPY_ON_WRITE, a valueInterface class is required
customerMap.setCopyMode(CopyMode.COPY_ON_WRITE,
 com.ibm.websphere.samples.objectgrid.CounterValueInterface.class);
customerMap.setCopyKey(true);
customerMap.setNullValuesSupported(false);
customerMap.setTtlEvictorType(TTLType.LAST_ACCESS_TIME);
customerMap.setTimeToLive(3000); // set time to live to 50 minutes
```

Bean エレメント

Bean エレメントを使用して、プラグインを定義します。プラグインを objectGrid および BackingMap エレメント・インスタンスに接続することができます。

- objectGrid エレメント内の出現回数: 0 回から複数回
- backingMapPluginCollection エレメント内の出現回数: 0 回から複数回
- 子エレメント: property エレメント

属性

id 作成するプラグインのタイプを指定します。(必須)

objectGrid エレメントの子エレメントである bean に有効なプラグインは次のリストのとおりです。

- TransactionCallback プラグイン
- ObjectGridEventListener プラグイン
- SubjectSource プラグイン
- MapAuthorization プラグイン
- SubjectValidation プラグイン

backingMapPluginCollection エレメントの子エレメントである bean に有効なプラグインは次のリストのとおりです。

- ローダー・プラグイン
- ObjectTransformer プラグイン
- OptimisticCallback プラグイン
- Evictor プラグイン
- MapEventListener プラグイン
- MapIndex プラグイン

className

プラグインを作成するためにインスタンス化するクラスまたは Spring Bean の名前を指定します。クラスはプラグイン・タイプのインターフェースを実装しなければなりません。例えば、id 属性の値として ObjectGridEventListener を指定する場合は、className 属性値が ObjectGridEventListener インターフェースを実装するクラスを参照する必要があります。 (必須)

```
<bean
(1) id="TransactionCallback" | "ObjectGridEventListener" |"SubjectSource" |
    "MapAuthorization" | "SubjectValidation" | "Loader" | "ObjectTransformer" |
    "OptimisticCallback" | "Evictor" | "MapEventListener" | "MapIndexPlugin"
(2) className="class name" | "(spring)bean name"
/>
```

次の例では、Bean エレメントを使用するプラグインを構成する方法を示すために companyGridBean.xml ファイルが使用されています。 ObjectGridEventListener プラグインが CompanyGrid ObjectGrid に追加されます。この Bean の className 属性は com.ibm.websphere.objectgrid.plugins.builtins.TranPropListener クラスです。このクラスは、必要に応じて com.ibm.websphere.objectgrid.plugins.ObjectGridEventListener インターフェースを実装します。

companyGridBean.xml ファイルでは BackingMap プラグインも定義されています。evictor プラグインが Customer BackingMap インスタンスに追加されています。bean の ID が Evictor であるため、className 属性には com.ibm.websphere.objectgrid.plugins.Evictor インターフェースを実装するクラスを指定する必要があります。com.ibm.websphere.objectgrid.plugins.builtins.LRUEvictor クラスがこのインターフェースを実装します。 backingMap は pluginCollectionRef 属性を使用して、そのプラグインを参照します。

```
<?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="CompanyGrid">
   bean id="ObjectGridEventListener"
    className="com.ibm.websphere.objectgrid.plugins.builtins.TranPropListener"/>
   <backingMap name="Customer"
    pluginCollectionRef="customerPlugins"/>
  </objectGrid>
 </objectGrids>
 <backingMapPluginCollections>
  <backingMapPluginCollection id="customerPlugins">
   <bean id="Evictor"
     className="com.ibm.websphere.objectGrid.plugins.builtins.LRUEvictor/>
  </backingMapPluginCollection>
 </backingMapPluginCollections>
</objectGridConfig>
```

以下のコード・サンプルは、前の例の companyGridBean.xml ファイルと同じ構成を達成するプログラマチックな方法を示すものです。

```
ObjectGridManager objectGridManager = ObjectGridManagerFactory.getObjectGridManager();
ObjectGrid companyGrid = objectGridManager.createObjectGrid("CompanyGrid", false);
TranPropListener tranPropListener = new TranPropListener();
companyGrid.addEventListener(tranPropListener);

BackingMap customerMap = companyGrid.defineMap("Customer");
Evictor lruEvictor = new com.ibm.websphere.objectgrid.plugins.builtins.LRUEvictor();
customerMap.setEvictor(lruEvictor);
```

property エレメント

property エレメントは、プラグインにプロパティーを追加するために使用します。このプロパティーの名前は、このプロパティーを含む Bean が参照するクラスの set メソッドに対応している必要があります。

- 出現回数: 0 回から複数回
- 子エレメント: なし

属性

name
プロパティーの名前を指定します。この属性に割り当てられる値は、このプロパティーを含む Bean の className 属性で指定されたクラスの set メソッドと対応している必要があります。例えば、Bean の className 属性を `com.ibm.MyPlugin` に設定し、指定したプロパティーの名前が `size` である場合、`com.ibm.MyPlugin` クラスに setSize メソッドが必要です。(必須)

type
プロパティーのタイプを指定します。このタイプは、name 属性により識別される set メソッドに受け渡されます。有効な値は、Java プリミティブ、それに対応する java.lang プリミティブ、および java.lang.String です。 name 属性と type 属性は、Bean の className 属性のメソッド・シグニチャーに対応していなければなりません。例えば、名前を `size` と設定し、タイプを `int` と設定する場合は、Bean の className 属性として指定されたクラスに setSize(int) メソッドが存在している必要があります。 (必須)

value
プロパティーの値を指定します。この値は type 属性によって指定されたタイプに変換され、次に name 属性と type 属性で識別された set メソッドへの呼び出しでパラメーターとして使用されます。この属性の値は、どんな方法でも妥当性検査されません。 (必須)

description
プロパティーの説明です。(オプション)

```
<bean
(1) name="name"
(2) type="java.lang.String" | "boolean" | "java.lang.Boolean" | "int" |
    "java.lang.Integer" | "double" | "java.lang.Double" | "byte" |
    "java.lang.Byte" | "short" | "java.lang.Short" | "long" |
    "java.lang.Long" | "float" | "java.lang.Float" | "char" |
    "java.lang.Character"
(3) value="value"
(4) description="description"
/>
```

以下の例では、`companyGridProperty.xml` ファイルを使用して、Bean に property エレメントを追加する方法を示しています。この例では、maxSize という名前で int 型を持つプロパティーが Evictor に追加されます。com.ibm.websphere.objectgrid.plugins.builtins.LRUEvictor Evictor は、 setMaxSize(int) メソッドと一致するメソッド・シグニチャーを持っています。整数値 499 が com.ibm.websphere.objectgrid.plugins.builtins.LRUEvictor クラス上の setMaxSize(int) メソッドに渡されます。

```
<?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="CompanyGrid">
   <backingMap name="Customer"
    pluginCollectionRef="customerPlugins"/>
  </objectGrid>
 </objectGrids>
 <backingMapPluginCollections>
  <backingMapPluginCollection id="customerPlugins">
   <bean id="Evictor"
     className="com.ibm.websphere.objectGrid.plugins.builtins.LRUEvictor>
     <property name="maxSize" type="int" value="449"
       description="The maximum size of the LRU Evictor"/>
   </bean>
  </backingMapPluginCollection>
 </backingMapPluginCollections>
</objectGridConfig>
```

以下のコード・サンプルは、前の例の companyGridProperty.xml ファイルと同じ構成をプログラムで実現する方法を示しています。

```
ObjectGridManager objectGridManager = ObjectGridManagerFactory.getObjectGridManager();
ObjectGrid companyGrid = objectGridManager.createObjectGrid("CompanyGrid", false);

BackingMap customerMap = companyGrid.defineMap("Customer");

LRUEvictor lruEvictor = new com.ibm.websphere.objectgrid.plugins.builtins.LRUEvictor();
// if the XML file is used instead,
// the property that was added would cause the following call to occur
lruEvictor.setMaxSize(449);
customerMap.setEvictor(lruEvictor);
```

backingMapPluginsCollections エレメント

backingMapPluginsCollections エレメントは、すべての backingMapPluginCollection エレメントのコンテナーです。前のセクションにある companyGridProperty.xml ファイルでは、backingMapPluginCollections エレメントに customerPlugins という ID の backingMapPluginCollection エレメントが 1 つ含まれています。

- 出現回数: 0 回から 1 回
- 子エレメント: backingMapPluginCollection エレメント

backingMapPluginCollection エレメント

backingMapPluginCollection エレメントは、BackingMap プラグインを定義し、**id** 属性によって識別されます。 pluginCollectionRef 属性を指定して、このプラグインを参照します。いくつかの BackingMaps プラグインを同様の方法で構成すると、各 BackingMap は同じ backingMapPluginCollection エレメントを参照できます。

- 出現回数: 0 回から複数回
- 子エレメント: Bean エレメント

属性

id backingMapPluginCollection を識別し、backingMap エレメントの pluginCollectionRef 属性によって参照されます。各 ID は固有である必要があります。pluginCollectionRef 属性の値が 1 つの backingMapPluginCollection エレメントの ID と一致しない場合、XML 妥当性検査は失敗します。任意の数の backingMap エレメントが、単一の backingMapPluginCollection エレメントを参照できます。 (必須)

```
<backingMapPluginCollection
(1)   id="id"
/>
```

以下の例では、companyGridCollection.xml ファイルを使用して、backingMapPluginCollection エレメントの使用法を示しています。このファイルでは、Customer BackingMap が customerPlugins backingMapPluginCollection を使用して、LRUEvictor を使用する Customer BackingMap を構成します。 Item および OrderLine の BackingMap は、collection2 backingMapPluginCollection を参照します。これらの BackingMap には、それぞれ LFUEvictor のセットが含まれています。

```
<?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="CompanyGrid">
   <backingMap name="Customer"
    pluginCollectionRef="customerPlugins"/>
   <backingMap name="Item" pluginCollectionRef="collection2"/>
   <backingMap name="OrderLine"
    pluginCollectionRef="collection2"/>
   <backingMap name="Order"/>
  </objectGrid>
 </objectGrids>
 <backingMapPluginCollections>
  <backingMapPluginCollection id="customerPlugins">
   <bean id="Evictor"
     className="com.ibm.websphere.objectGrid.plugins.builtins.LRUEvictor/>
  </backingMapPluginCollection>
  <backingMapPluginCollection id="collection2">
   <bean id="Evictor"
     className="com.ibm.websphere.objectgrid.plugins.builtins.LFUEvictor/>
   <bean id="OptimisticCallback"
     className="com.ibm.websphere.samples.objectgrid.EmployeeOptimisticCallBackImpl"/>
  </backingMapPluginCollection>
  </backingMapPluginCollections>
</objectGridConfig>
```

以下のコード・サンプルは、前の例の companyGridCollection.xml ファイルと同じ構成をプログラムで実現する方法を示しています。

```
ObjectGridManager objectGridManager = ObjectGridManagerFactory.getObjectGridManager();

ObjectGrid companyGrid = objectGridManager.createObjectGrid("CompanyGrid", false);
BackingMap customerMap = companyGrid .defineMap("Customer");
LRUEvictor customerEvictor = new LRUEvictor();
customerMap.setEvictor(customerEvictor);

BackingMap itemMap = companyGrid.defineMap("Item");
LFUEvictor itemEvictor = new LFUEvictor();
itemMap.setEvictor(itemEvictor);

BackingMap orderLineMap = companyGrid.defineMap("OrderLine");
LFUEvictor orderLineEvictor = new LFUEvictor();
orderLineMap.setEvictor(orderLineEvictor);

BackingMap orderMap = companyGrid.defineMap("Order");
```

querySchema エレメント

querySchema エレメントは、BackingMap 間のリレーションシップを定義し、各マップ内にあるオブジェクトのタイプを識別します。この情報は、照会言語ストリングをマップ・アクセス呼び出しに変換するために ObjectQuery によって使用されます。

- 出現回数: 0 回から 1 回
- 子エレメント: mapSchemas エレメント、relationships エレメント

mapSchemas エレメント

各 querySchema エレメントは、1 つ以上の mapSchema エレメントを含む mapSchemas エレメントを 1 つ持ちます。

- 出現回数: 1 回
- 子エレメント: mapSchema エレメント

mapSchema エレメント

mapSchema エレメントは、BackingMap に保管されるオブジェクトのタイプ、およびそのデータにアクセスする方法を定義します。

- 出現回数: 1 回以上
- 子エレメント: なし

属性

mapName
スキーマに追加する BackingMap の名前を指定します。(必須)

valueClass
BackingMap の値部分に保管されるオブジェクトのタイプを指定します。(必須)

primaryKeyField
valueClass 属性内の 1 次キー属性の名前を指定します。1 次キーも BackingMap のキー部分に保管する必要があります。(オプション)

accessType
照会エンジンが valueClass オブジェクト・インスタンス内の永続データをイントロスペクトしてそのデータにアクセスする方法を示します。値を FIELD に設定すると、クラスのフィールドがイントロスペクトされ、スキーマに追加されます。値が PROPERTY の場合、get メソッドおよび is メソッドに関連付けられた属性が使用されます。デフォルト値は PROPERTY です。 (オプション)

```
<mapSchema
(1)   mapName="backingMapName"
(2)   valueClass="com.mycompany.OrderBean"
(3)   primaryKeyField="orderId"
(4)   accessType="PROPERTY" | "FIELD"
/>
```

以下の例では、companyGridQuerySchemaAttr.xml ファイルを使用して、サンプル mapSchema 構成を示しています。

```
<?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="CompanyGrid">
   <backingMap name="Order"/>
   <backingMap name="Customer"/>

   <querySchema>
    <mapSchemas>
     <mapSchema mapName="Order"
      valueClass="com.mycompany.OrderBean"
      primaryKeyField="orderNumber"
      accessType="FIELD"/>
     <mapSchema mapName="Customer"
      valueClass="com.mycompany.CustomerBean"
      primaryKeyField="id"
      accessType="FIELD"/>
```

```
    </mapSchemas>
   </querySchema>
  </objectGrid>
 </objectGrids>
</objectGridConfig>
```

以下のコード・サンプルは、前の例の companyGridQuerySchemaAttr.xml ファイルと同じ構成をプログラムで実現する方法を示しています。

```
ObjectGridManager objectGridManager = ObjectGridManagerFactory.getObjectGridManager();
ObjectGrid companyGrid = objectGridManager.createObjectGrid("CompanyGrid", false);
companyGrid.defineMap("Order");
companyGrid.defineMap("Customer");

// Define the schema
QueryConfig queryCfg = new QueryConfig();
queryCfg.addQueryMapping(new QueryMapping(
    "Order", OrderBean.class.getName(), "orderNumber", QueryMapping.FIELD_ACCESS));
queryCfg.addQueryMapping(new QueryMapping(
    "Customer", CustomerBean.class.getName(), "id", QueryMapping.FIELD_ACCESS));
companyGrid.setQueryConfig(queryCfg);
```

relationships エレメント

各 querySchema エレメントは、relationships エレメントをまったく持たないか (ゼロ)、または 1 つ以上の relationship エレメントを含む relationships エレメントを 1 つ持ちます。

- 出現回数: 0 回または 1 回
- 子エレメント: relationship エレメント

relationship エレメント

relationship エレメントは、2 つの BackingMap 間のリレーションシップ、およびそのリレーションシップをバインドする valueClass 属性の属性を定義します。

- 出現回数: 1 回以上
- 子エレメント: なし

属性

source
リレーションシップのソース側 valueClass の名前を指定します。 (必須)

target
リレーションシップのターゲット側 valueClass の名前を指定します。 (必須)

relationField
ソース valueClass 内でターゲットを参照している属性の名前を指定します。(必須)

invRelationField
ターゲット valueClass 内でソースを参照している属性の名前を指定します。この属性が指定されていないと、リレーションシップは単一方向となります。(オプション)

```
<mapSchema
(1)   source="com.mycompany.OrderBean"
(2)   target="com.mycompany.CustomerBean"
(3)   relationField="customer"
(4)   invRelationField="orders"
/>
```

以下の例では、companyGridQuerySchemaWithRelationshipAttr.xml ファイルを使用して、双方向リレーションシップを含むサンプル mapSchema 構成を示しています。

```
<?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="CompanyGrid">
   <backingMap name="Order"/>
   <backingMap name="Customer"/>

   <querySchema>
    <mapSchemas>
     <mapSchema mapName="Order"
      valueClass="com.mycompany.OrderBean"
      primaryKeyField="orderNumber"
      accessType="FIELD"/>
     <mapSchema mapName="Customer"
      valueClass="com.mycompany.CustomerBean"
      primaryKeyField="id"
      accessType="FIELD"/>
    </mapSchemas>
    <relationships>
     <relationship
      source="com.mycompany.OrderBean"
      target="com.mycompany.CustomerBean"
      relationField="customer"/>
      invRelationField="orders"/>
    </relationships>
   </querySchema>
  </objectGrid>
 </objectGrids>
</objectGridConfig>
```

以下のコード・サンプルは、前の例の companyGridQuerySchemaWithRelationshipAttr.xml ファイルと同じ構成をプログラムで実現する方法を示しています。

```
ObjectGridManager objectGridManager = ObjectGridManagerFactory.getObjectGridManager();
ObjectGrid companyGrid = objectGridManager.createObjectGrid("CompanyGrid", false);
companyGrid.defineMap("Order");
companyGrid.defineMap("Customer");

// Define the schema
QueryConfig queryCfg = new QueryConfig();
queryCfg.addQueryMapping(new QueryMapping(
    "Order", OrderBean.class.getName(), "orderNumber", QueryMapping.FIELD_ACCESS));
queryCfg.addQueryMapping(new QueryMapping(
    "Customer", CustomerBean.class.getName(), "id", QueryMapping.FIELD_ACCESS));
queryCfg.addQueryRelationship(new QueryRelationship(
     OrderBean.class.getName(), CustomerBean.class.getName(), "customer", "orders"));
companyGrid.setQueryConfig(queryCfg);
```

streamQuerySet エレメント

streamQuerySet エレメントは、ストリーム照会セットを定義するための最上位レベルのエレメントです。

- 出現回数: 0 回から複数回
- 子エレメント: stream エレメント、view エレメント

stream エレメント

stream エレメントは、ストリーム照会エンジンへのストリームを表します。stream エレメントの各属性は、StreamMetadata インターフェースのメソッドに対応します。

- 出現回数: 1 回から複数回
- 子エレメント: basic エレメント

属性

name
ストリームの名前を指定します。この属性が指定されていないと、妥当性検査は失敗します。(必須)

valueClass
このストリーム ObjectMap に保管される値のクラス・タイプを指定します。このクラス・タイプは、オブジェクトをストリーム・イベントに変換し、SQL ステートメントが指定されていない場合は SQL ステートメントを生成するために使用されます。 (必須)

sql ストリームの SQL ステートメントを指定します。このプロパティーを指定しなかった場合、valueClass 属性で属性または accessor メソッドのリフレクション、またはエンティティー・メタデータの tuple 属性を使用してストリーム SQL が生成されます。(オプション)

access
値クラスの属性にアクセスするためのタイプを指定します。この値を FIELD に設定すると、属性は Java のリフレクションを使用してフィールドから直接取り出されます。そうでない場合、属性は accessor メソッドを使用して読み取られます。デフォルト値は PROPERTY です。 (オプション)

```
<stream
(1) name="streamName"
(2) valueClass="streamMapClassType"
(3) sql="streamSQL create stream stockQuote
     keyed by t ( transactionvolume INTEGER, price DECIMAL (9,2), issue VARCHAR(100) );"
(4) access="PROPERTY" | "FIELD"
/>
```

view エレメント

view エレメントはストリーム照会ビューを表します。各 stream エレメントが ViewMetadata インターフェースのメソッドに対応します。

- 出現回数: 1 回から複数回
- 子エレメント: basic エレメント、id エレメント

属性

name
ビューの名前を指定します。この属性が指定されていないと、妥当性検査は失敗します。(必須)

sql ビューの変換を定義する、ストリームの SQL を指定します。この属性が指定されていないと、妥当性検査は失敗します。(必須)

valueClass
ObjectMap のこのビューに保管される値のクラス・タイプを指定します。このクラス・タイプは、ビュー・イベントをこのクラス・タイプと互換性のある適切なタプル・フォーマットに変換するのに使用されます。クラス・タイプを指定しなかった場合、Stream Processing Technology Structured Query Language (SPTSQL) の列定義に従ってデフォルトのフォーマットが使用されます。このビュー・マッ

プにエンティティー・メタデータが定義されていると、この属性は使用されません。代わりに、エンティティー・メタデータが使用されます。 (オプション)

access
値クラスの属性にアクセスするためのタイプを指定します。アクセス・タイプを FIELD に設定すると、列値は、Java のリフレクションを使用して直接にフィールドに設定されます。そうでない場合、accessor メソッドを使用して属性が設定されます。デフォルト値は PROPERTY です。 (オプション)

```
<view
(1)   name="viewName"
(2)   valueClass="viewMapValueClass"
(3)   sql="viewSQL CREATE VIEW last5MinuteAvgPrice AS
      SELECT issue, avg(price) as totalVolume
        FROM (SELECT * FROM stockQuote FETCH LATEST 5 MINUTES) group by issue;"/>
(4)   access="PROPERTY" | "FIELD"
/>
```

basic エレメント

basic エレメントは、値クラスまたはエンティティー・メタデータの属性名から SPTSQL で定義された列へのマッピングを定義するために使用します。

- 出現回数: 0 回から複数回
- 子エレメント: なし

```
<basic
(1)   name="attributeName"
(2)   column="columnName"
/>
```

id エレメント

id エレメントは、キー属性のマッピングに使用されます。

- 出現回数: 0 回から複数回
- 子エレメント: なし

```
<id
(1)   name="idName"
(2)   column="columnName"
/>
```

以下の例では、StreamQueryApp2.xml ファイルを使用して、streamQuerySet の属性を構成する方法を示しています。ストリーム照会セット _stockQuoteSQS_ にはストリームおよびビューが 1 つずつあります。ストリームおよびビューの両方で、名前、valueClasss、sql、およびアクセス・タイプがそれぞれ定義されています。ストリームは basic エレメントも定義します。これにより、StockQuote クラスの volume 属性が、SQL ステートメントで定義される SQL 列トランザクション・ボリュームにマップされることが指定されます。

```
<?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="og1">
   <backingMap name="stockQuote" readOnly="false" copyKey="true" streamRef="stockQuote"/>
   <backingMap name="last5MinuteAvgPrice" readOnly="false" copyKey="false" viewRef="last5MinuteAvgPrice"/>

   <streamQuerySet name="stockQuoteSQS">
    <stream
     name="stockQuote"
     valueClass="com.ibm.ws.objectgrid.streamquery.sample.guide.StockQuote"
```

```
     sql="create stream stockQuote
         keyed by t ( transactionvolume INTEGER, price DECIMAL (9,2), issue VARCHAR(100) );"
     access="FIELD">
     <basic name="volume" column="transactionvolume"/>
    </stream>

    <view
     name="last5MinuteAvgPrice"
     valueClass="com.ibm.ws.objectgrid.streamquery.sample.guide.AveragePrice"
     sql="CREATE VIEW last5MinuteAvgPrice AS SELECT issue, avg(price) as avgPrice
       FROM (SELECT * FROM stockQuote FETCH LATEST 5 MINUTES) group by issue;"
     access="FIELD"
    </view>
   </streamQuerySet>
  </objectGrid>
 </objectGrids>
</objectGridConfig>
```

objectGrid.xsd ファイル

ObjectGrid 記述子 XML スキーマを使用して、WebSphere eXtreme Scale を構成します。

objectGrid.xsd ファイルに定義されるエレメントおよび属性の説明は、161 ページの『ObjectGrid 記述子 XML ファイル』を参照してください。 Spring の objectgrid.xsd ファイルについては、223 ページの『Spring 記述子 XML ファイル』を参照してください。

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

 <xsd:element name="objectGridConfig">
  <xsd:complexType>
   <xsd:sequence>
    <xsd:element maxOccurs="1" minOccurs="1" name="objectGrids"
       type="dgc:objectGrids">
     <xsd:unique name="objectGridNameUnique">
      <xsd:selector xpath="dgc:objectGrid"/>
      <xsd:field xpath="@name"/>
     </xsd:unique>
    </xsd:element>
    <xsd:element maxOccurs="1" minOccurs="0" name="backingMapPluginCollections"
       type="dgc:backingMapPluginCollections"/>
   </xsd:sequence>
  </xsd:complexType>

  <xsd:key name="backingMapPluginCollectionId">
   <xsd:selector xpath="dgc:backingMapPluginCollections/dgc:
     backingMapPluginCollection"/>
   <xsd:field xpath="@id"/>
  </xsd:key>

  <xsd:keyref name="pluginCollectionRef" refer="dgc:backingMapPluginCollectionId">
   <xsd:selector xpath="dgc:objectGrids/dgc:objectGrid/dgc:backingMap"/>
   <xsd:field xpath="@pluginCollectionRef"/>
  </xsd:keyref>

  <xsd:key name="streamName">
   <xsd:selector xpath="dgc:objectGrids/dgc:objectGrid/dgc:
     streamQuerySet/dgc:stream"/>
   <xsd:field xpath="@name"/>
  </xsd:key>

  <xsd:keyref name="streamRef" refer="dgc:streamName">
   <xsd:selector xpath="dgc:objectGrids/dgc:objectGrid/dgc:backingMap"/>
   <xsd:field xpath="@streamRef"/>
  </xsd:keyref>

  <xsd:key name="viewName">
   <xsd:selector xpath="dgc:objectGrids/dgc:objectGrid/dgc:streamQuerySet/dgc:view"/>
   <xsd:field xpath="@name"/>
```

```
 </xsd:key>

 <xsd:keyref name="viewRef" refer="dgc:viewName">
  <xsd:selector xpath="dgc:objectGrids/dgc:objectGrid/dgc:backingMap"/>
  <xsd:field xpath="@viewRef"/>
 </xsd:keyref>
</xsd:element>

<xsd:complexType name="objectGrids">
 <xsd:sequence>
  <xsd:element maxOccurs="unbounded" minOccurs="1" name="objectGrid"
     type="dgc:objectGrid">
   <xsd:unique name="backingMapNameUnique">
    <xsd:selector xpath="dgc:backingMap"/>
    <xsd:field xpath="@name"/>
   </xsd:unique>
   <xsd:unique name="streamQuerySetNameUnique">
    <xsd:selector xpath="dgc:streamQuerySet"/>
    <xsd:field xpath="@name"/>
   </xsd:unique>
  </xsd:element>
 </xsd:sequence>
</xsd:complexType>

<xsd:complexType name="backingMapPluginCollections">
 <xsd:sequence>
  <xsd:element maxOccurs="unbounded" minOccurs="0" name="backingMapPluginCollection"
    type="dgc:backingMapPluginCollection"/>
 </xsd:sequence>
</xsd:complexType>

<xsd:complexType name="objectGrid">
 <xsd:sequence>
  <xsd:element maxOccurs="unbounded" minOccurs="0" name="bean" type="dgc:bean"/>
  <xsd:element maxOccurs="unbounded" minOccurs="0" name="backingMap"
     type="dgc:backingMap"/>
  <xsd:element maxOccurs="1" minOccurs="0" name="querySchema" type="dgc:querySchema"/>
  <xsd:element maxOccurs="unbounded" minOccurs="0" name="streamQuerySet"
     type="dgc:streamQuerySet">
   <xsd:unique name="stream">
    <xsd:selector xpath="dgc:stream"/>
    <xsd:field xpath="@name"/>
   </xsd:unique>
   <xsd:unique name="view">
    <xsd:selector xpath="dgc:view"/>
    <xsd:field xpath="@name"/>
   </xsd:unique>
  </xsd:element>
 </xsd:sequence>
 <xsd:attribute name="name" type="xsd:string" use="required"/>
 <xsd:attribute name="authorizationMechanism" type="dgc:authorizationMechanism"
   use="optional"/>
 <xsd:attribute name="accessByCreatorOnlyMode" type="dgc:accessByCreatorOnlyMode"
   use="optional"/>
 <xsd:attribute name="securityEnabled" type="xsd:boolean" use="optional"/>
 <xsd:attribute name="txTimeout" type="xsd:int" use="optional"/>
 <xsd:attribute name="permissionCheckPeriod" type="xsd:int" use="optional"/>
 <xsd:attribute name="entityMetadataXMLFile" type="xsd:string" use="optional"/>
 <xsd:attribute name="initialState" type="dgc:initialState" use="optional"/>
</xsd:complexType>


<xsd:complexType name="backingMap">
 <xsd:sequence>
  <xsd:element maxOccurs="1" minOccurs="0" name="timeBasedDBUpdate" type="dgc:
    timeBasedDBUpdate"/>
 </xsd:sequence>
 <xsd:attribute name="name" type="xsd:string" use="required"/>
 <xsd:attribute name="readOnly" type="xsd:boolean" use="optional"/>
 <xsd:attribute name="pluginCollectionRef" type="xsd:string" use="optional"/>
 <xsd:attribute name="preloadMode" type="xsd:boolean" use="optional"/>
 <xsd:attribute name="lockStrategy" type="dgc:lockStrategy" use="optional"/>
 <xsd:attribute name="copyMode" type="dgc:copyMode" use="optional"/>
 <xsd:attribute name="valueInterfaceClassName" type="xsd:string" use="optional"/>
 <xsd:attribute name="numberOfBuckets" type="xsd:int" use="optional"/>
 <xsd:attribute name="nullValuesSupported" type="xsd:boolean" use="optional"/>
 <xsd:attribute name="lockTimeout" type="xsd:int" use="optional"/>
 <xsd:attribute name="numberOfLockBuckets" type="xsd:int" use="optional"/>
 <xsd:attribute name="copyKey" type="xsd:boolean" use="optional"/>
```

```
 <xsd:attribute name="timeToLive" type="xsd:int" use="optional"/>
 <xsd:attribute name="ttlEvictoryType" type="dgc:ttlEvictorType" use="optional"/>
 <xsd:attribute name="streamRef" type="xsd:string" use="optional"/>
 <xsd:attribute name="viewRef" type="xsd:string" use="optional"/>
 <xsd:attribute name="writeBehind" type="xsd:string" use="optional"/>
 <xsd:attribute name="evictionTriggers" type="xsd:string" use="optional"/>
 <xsd:attribute name="template" type="xsd:boolean" use="optional"/>
</xsd:complexType>

<xsd:complexType name="bean">
 <xsd:sequence>
  <xsd:element maxOccurs="unbounded" minOccurs="0" name="property" type="dgc:property"/>
 </xsd:sequence>
 <xsd:attribute name="className" type="xsd:string" use="required"/>
 <xsd:attribute name="id" type="dgc:beanId" use="required"/>
</xsd:complexType>

<xsd:complexType name="backingMapPluginCollection">
 <xsd:sequence>
  <xsd:element maxOccurs="unbounded" minOccurs="0" name="bean" type="dgc:bean"/>
 </xsd:sequence>
 <xsd:attribute name="id" type="xsd:string" use="required"/>
</xsd:complexType>

<xsd:complexType name="property">
 <xsd:attribute name="name" type="xsd:string" use="required"/>
 <xsd:attribute name="value" type="xsd:string" use="required"/>
 <xsd:attribute name="type" type="dgc:propertyType" use="required"/>
 <xsd:attribute name="description" type="xsd:string" use="optional"/>
</xsd:complexType>

<xsd:simpleType name="propertyType">
 <xsd:restriction base="xsd:string">
  <xsd:enumeration value="java.lang.Boolean"/>
  <xsd:enumeration value="boolean"/>
  <xsd:enumeration value="java.lang.String"/>
  <xsd:enumeration value="java.lang.Integer"/>
  <xsd:enumeration value="int"/>
  <xsd:enumeration value="java.lang.Double"/>
  <xsd:enumeration value="double"/>
  <xsd:enumeration value="java.lang.Byte"/>
  <xsd:enumeration value="byte"/>
  <xsd:enumeration value="java.lang.Short"/>
  <xsd:enumeration value="short"/>
  <xsd:enumeration value="java.lang.Long"/>
  <xsd:enumeration value="long"/>
  <xsd:enumeration value="java.lang.Float"/>
  <xsd:enumeration value="float"/>
  <xsd:enumeration value="java.lang.Character"/>
  <xsd:enumeration value="char"/>
 </xsd:restriction>
</xsd:simpleType>

<xsd:simpleType name="beanId">
 <xsd:restriction base="xsd:string">
  <xsd:enumeration value="TransactionCallBack"/>
  <xsd:enumeration value="ObjectGridEventListener"/>
  <xsd:enumeration value="SubjectSource"/>
  <xsd:enumeration value="MapAuthorization"/>
  <xsd:enumeration value="SubjectValidation"/>
  <xsd:enumeration value="ObjectGridAuthorization"/>

  <xsd:enumeration value="Loader"/>
  <xsd:enumeration value="ObjectTransformer"/>
  <xsd:enumeration value="OptimisticCallback"/>
  <xsd:enumeration value="Evictor"/>
  <xsd:enumeration value="MapEventListener"/>
  <xsd:enumeration value="MapIndexPlugin"/>

 </xsd:restriction>
</xsd:simpleType>

<xsd:simpleType name="copyMode">
 <xsd:restriction base="xsd:string">
  <xsd:enumeration value="COPY_ON_READ_AND_COMMIT"/>
  <xsd:enumeration value="COPY_ON_READ"/>
  <xsd:enumeration value="COPY_ON_WRITE"/>
  <xsd:enumeration value="NO_COPY"/>
  <xsd:enumeration value="COPY_TO_BYTES"/>
```

```
 </xsd:restriction>
</xsd:simpleType>

<xsd:simpleType name="lockStrategy">
 <xsd:restriction base="xsd:string">
  <xsd:enumeration value="OPTIMISTIC"/>
  <xsd:enumeration value="PESSIMISTIC"/>
  <xsd:enumeration value="NONE"/>
 </xsd:restriction>
</xsd:simpleType>

<xsd:simpleType name="ttlEvictorType">
 <xsd:restriction base="xsd:string">
  <xsd:enumeration value="CREATION_TIME"/>
  <xsd:enumeration value="LAST_ACCESS_TIME"/>
  <xsd:enumeration value="NONE"/>
 </xsd:restriction>
</xsd:simpleType>

<xsd:simpleType name="authorizationMechanism">
 <xsd:restriction base="xsd:string">
  <xsd:enumeration value="AUTHORIZATION_MECHANISM_JAAS"/>
  <xsd:enumeration value="AUTHORIZATION_MECHANISM_CUSTOM"/>
 </xsd:restriction>
</xsd:simpleType>

<xsd:simpleType name="accessByCreatorOnlyMode">
 <xsd:restriction base="xsd:string">
  <xsd:enumeration value="disabled"/>
  <xsd:enumeration value="complement"/>
  <xsd:enumeration value="supersede"/>
 </xsd:restriction>
</xsd:simpleType>

<xsd:complexType name="streamQuerySet">
 <xsd:sequence>
  <xsd:element maxOccurs="unbounded" minOccurs="0" name="stream" type="dgc:stream">
   <xsd:unique name="streamBasicColumnUnique">
    <xsd:selector xpath="dgc:basic"/>
    <xsd:field xpath="@column"/>
   </xsd:unique>
  </xsd:element>
  <xsd:element maxOccurs="unbounded" minOccurs="0" name="view" type="dgc:view">
   <xsd:unique name="viewBasicColumnUnique">
    <xsd:selector xpath="dgc:basic"/>
    <xsd:field xpath="@column"/>
   </xsd:unique>
  </xsd:element>
 </xsd:sequence>
 <xsd:attribute name="name" type="xsd:string" use="required"/>
 <xsd:attribute name="viewResultsToListenersOnly" type="xsd:boolean"
    default="false" use="optional"/>
 <xsd:attribute name="deployInPrimaryOnly" type="xsd:boolean" default="true"
    use="optional"/>
</xsd:complexType>

<xsd:complexType name="stream">
 <xsd:sequence>
  <xsd:element maxOccurs="unbounded" minOccurs="0" name="basic" type="dgc:basic"/>
 </xsd:sequence>
 <xsd:attribute name="valueClass" type="xsd:string" use="required"/>
 <xsd:attribute name="name" type="xsd:string" use="required"/>
 <xsd:attribute name="sql" type="xsd:string" use="optional"/>
 <xsd:attribute name="access" type="cc:accessType" use="optional"/>
</xsd:complexType>

<xsd:complexType name="view">
 <xsd:sequence>
  <xsd:element maxOccurs="unbounded" minOccurs="0" name="id" type="dgc:basic"/>
  <xsd:element element maxOccurs="unbounded" minOccurs="0" name="basic"
    type="dgc:basic"/>
 </xsd:sequence>
 <xsd:attribute name="name" type="xsd:string" use="required"/>
 <xsd:attribute name="sql" type="xsd:string" use="optional"/>
 <xsd:attribute name="valueClass" type="xsd:string" use="optional"/>
 <xsd:attribute name="access" type="cc:accessType" use="optional"/>
</xsd:complexType>

<xsd:complexType name="basic">
```

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

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

<xsd:complexType name="timeBasedDBUpdate">
 <xsd:attribute name="persistenceUnitName" type="xsd:string" use="optional"/>
 <xsd:attribute name="mode" type="cc:dbUpdateMode" use="optional"/>
 <xsd:attribute name="timestampField" type="xsd:string" use="optional"/>
 <xsd:attribute name="entityClass" type="xsd:string" use="required"/>
 <xsd:attribute name="jpaPropertyFactory" type="xsd:string" use="optional"/>
</xsd:complexType>

<xsd:simpleType name="dbUpdateMode">
 <xsd:restriction base="xsd:string"/>
  <xsd:enumeration value="INVALIDATE_ONLY"/>
  <xsd:enumeration value="UPDATE_ONLY"/>
  <xsd:enumeration value="INSERT_UPDATE"/>
 </xsd:restriction>
</xsd:simpleType>

<xsd:complexType name="querySchema">
 <xsd:sequence>
  <xsd:element maxOccurs="1" minOccurs="1" name="mapSchemas" type="dgc:mapSchemas">
   <xsd:unique name="mapNameUnique">
    <xsd:selector xpath="dgc:mapSchema"/>
    <xsd:field xpath="@mapName"/>
   </xsd:unique>
  </xsd:element>
  <xsd:element maxOccurs="1" minOccurs="0" name="relationships"
    type="dgc:relationships"/>
 </xsd:sequence>
</xsd:complexType>

<xsd:complexType name="mapSchemas">
 <xsd:sequence>
  <xsd:element maxOccurs="unbounded" minOccurs="1" name="mapSchema"
    type="dgc:mapSchema"/>
 </xsd:sequence>
</xsd:complexType>

<xsd:complexType name="relationships">
 <xsd:sequence>
  <xsd:element maxOccurs="unbounded" minOccurs="1" name="relationship"
    type="dgc:relationship"/>
 </xsd:sequence>
</xsd:complexType>

<xsd:complexType name="mapSchema">
 <xsd:attribute name="mapName" type="xsd:string" use="required"/>
 <xsd:attribute name="valueClass" type="xsd:string" use="required"/>
 <xsd:attribute name="primaryKeyField" type="xsd:string" use="optional"/>
 <xsd:attribute name="accessType" type="cc:accessType" use="optional"/>
</xsd:complexType>

<xsd:complexType name="relationship">
 <xsd:attribute name="source" type="xsd:string" use="required"/>
 <xsd:attribute name="target" type="xsd:string" use="required"/>
 <xsd:attribute name="relationField" type="xsd:string" use="required"/>
 <xsd:attribute name="invRelationField" type="xsd:string" use="optional"/>
</xsd:complexType>

<xsd:simpleType name="accessType">
 <xsd:restriction base="xsd:string">
  <xsd:enumeration value="PROPERTY"/>
  <xsd:enumeration value="FIELD"/>
 </xsd:restriction>
</xsd:simpleType>

<xsd:simpleType name="initialState">
 <xsd:restriction base="xsd:string">
  <xsd:enumeration value="OFFLINE"/>
  <xsd:enumeration value="PRELOAD"/>
  <xsd:enumeration value="ONLINE"/>
```

```
  </xsd:restriction>
 </xsd:simpleType>

</xsd:schema>
```

エンティティー・メタデータ記述子 XML ファイル

エンティティー・メタデータ記述子ファイルは、WebSphere eXtreme Scale のエンティティー・スキーマを定義するために使用される XML ファイルです。すべてのエンティティー・メタデータを XML ファイルで定義するか、エンティティー・メタデータをエンティティー Java クラス・ファイルでアノテーションとして定義します。主に Java アノテーションを使用できないエンティティーに使用されます。

XML ファイルに基づくエンティティー・メタデータを作成するには、XML 構成を使用します。XML 構成で定義されている属性の一部をアノテーションとともに使用すると、対応するアノテーションがオーバーライドされます。あるエレメントをオーバーライドできる場合は、そのオーバーライドが明示的に以降のセクションに出現します。エンティティー・メタデータ記述子 XML ファイルの例については、195 ページの『emd.xsd ファイル』を参照してください。

id エレメント

id エレメントは、属性がキーであることを暗黙に示しています。最低限、少なくとも 1 つの id エレメントを指定する必要があります。複数の id キーを指定して、1 つの複合キーとして使用することができます。

属性

name
属性の名前を指定します。この属性は Java ファイルに存在している必要があります。

alias
エレメントの別名を指定します。alias 値は、アノテーション付きエンティティーとともに使用されるとオーバーライドされます。

basic エレメント

basic エレメントは、属性が以下のようなプリミティブ型またはプリミティブ型のラッパーであることを暗黙に示しています。

- java.lang.String
- java.math.BigInteger
- java.math.BigDecimal
- java.util.Date
- java.util.Calendar
- java.sql.Date
- java.sql.Time
- java.sql.Timestamp
- byte[]
- Byte[]
- char[]

- Character[]
- Java Platform, Standard Edition バージョン 5 の列挙型

属性を basic として指定する必要はありません。basic エレメント属性は、リフレクションを使用して自動的に構成されます。

id-class エレメント

id_class エレメントは、複合キーを使用してエンティティーを見つけるときに役立つ複合キー・クラスを指定します。

属性

class-name
id-class エレメントで使用するクラス名 (すなわち id-class) を指定します。

transient

transient エレメントは、このエレメントが無視され、処理されないことを暗黙に示しています。アノテーション付きエンティティーと一緒に使用された場合は、オーバーライドすることもできます。

属性

name
無視される属性の名前を指定します。

version

transient エレメントは、このエレメントが無視され、処理されないことを暗黙に示しています。アノテーション付きエンティティーと一緒に使用された場合は、オーバーライドすることもできます。

属性

name
無視される属性の名前を指定します。

property エレメント

property エレメントは、プラグインにプロパティーを追加するために使用します。このプロパティーの名前は、このプロパティーを含む Bean が参照するクラスの set メソッドに対応している必要があります。

- 出現回数: 0 回から複数回
- 子エレメント: なし

属性

name
プロパティーの名前を指定します。この属性に割り当てられる値は、このプロパティーを含む Bean の className 属性で指定されたクラスの set メソッドと対応している必要があります。例えば、Bean の className 属性を `com.ibm.MyPlugin` に設定し、指定したプロパティーの名前が `size` である場合、`com.ibm.MyPlugin` クラスに setSize メソッドが必要です。(必須)

type
プロパティーのタイプを指定します。このタイプは、name 属性により識別される set メソッドに受け渡されます。有効な値は、Java プリミティブ、それに対応する java.lang プリミティブ、および java.lang.String です。 name 属性と type 属性は、Bean の className 属性のメソッド・シグニチャーに対応していなければなりません。例えば、名前を `size` と設定し、タイプを `int` と設定する場合は、Bean の className 属性として指定されたクラスに setSize(int) メソッドが存在している必要があります。 (必須)

value
プロパティーの値を指定します。この値は type 属性によって指定されたタイプに変換され、次に name 属性と type 属性で識別された set メソッドへの呼び出しでパラメーターとして使用されます。この属性の値は、どんな方法でも妥当性検査されません。 (必須)

description
プロパティーの説明です。(オプション)

```
<bean
(1)   name="name"
(2)   type="java.lang.String" | "boolean" | "java.lang.Boolean" | "int" |
       "java.lang.Integer" | "double" | "java.lang.Double" | "byte" |
       "java.lang.Byte" | "short" | "java.lang.Short" | "long" |
       "java.lang.Long" | "float" | "java.lang.Float" | "char" |
       "java.lang.Character"
(3)   value="value"
(4)   description="description"
/>
```

以下の例では、`companyGridProperty.xml` ファイルを使用して、Bean に property エレメントを追加する方法を示しています。この例では、maxSize という名前で int 型を持つプロパティーが Evictor に追加されます。com.ibm.websphere.objectgrid.plugins.builtins.LRUEvictor Evictor は、 setMaxSize(int) メソッドと一致するメソッド・シグニチャーを持っています。整数値 499 が com.ibm.websphere.objectgrid.plugins.builtins.LRUEvictor クラス上の setMaxSize(int) メソッドに渡されます。

```
<?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="CompanyGrid">
   <backingMap name="Customer"
    pluginCollectionRef="customerPlugins"/>
  </objectGrid>
 </objectGrids>
 <backingMapPluginCollections>
  <backingMapPluginCollection id="customerPlugins">
   <bean id="Evictor"
     className="com.ibm.websphere.objectGrid.plugins.builtins.LRUEvictor>
     <property name="maxSize" type="int" value="449"
       description="The maximum size of the LRU Evictor"/>
   </bean>
  </backingMapPluginCollection>
 </backingMapPluginCollections>
</objectGridConfig>
```

以下のコード・サンプルは、前の例の `companyGridProperty.xml` ファイルと同じ構成をプログラムで実現する方法を示しています。

```
ObjectGridManager objectGridManager = ObjectGridManagerFactory.getObjectGridManager();
ObjectGrid companyGrid = objectGridManager.createObjectGrid("CompanyGrid", false);

BackingMap customerMap = companyGrid.defineMap("Customer");

LRUEvictor lruEvictor = new com.ibm.websphere.objectgrid.plugins.builtins.LRUEvictor();
// if the XML file is used instead,
// the property that was added would cause the following call to occur
lruEvictor.setMaxSize(449);
customerMap.setEvictor(lruEvictor);
```

backingMapPluginsCollections エレメント

backingMapPluginsCollections エレメントは、すべての backingMapPluginCollection エレメントのコンテナーです。前のセクションにある companyGridProperty.xml ファイルでは、backingMapPluginCollections エレメントに customerPlugins という ID の backingMapPluginCollection エレメントが 1 つ含まれています。

- 出現回数: 0 回から 1 回
- 子エレメント: backingMapPluginCollection エレメント

backingMapPluginCollection エレメント

backingMapPluginCollection エレメントは、BackingMap プラグインを定義し、**id** 属性によって識別されます。 pluginCollectionRef 属性を指定して、このプラグインを参照します。いくつかの BackingMaps プラグインを同様の方法で構成すると、各 BackingMap は同じ backingMapPluginCollection エレメントを参照できます。

- 出現回数: 0 回から複数回
- 子エレメント: Bean エレメント

属性

id backingMapPluginCollection を識別し、backingMap エレメントの pluginCollectionRef 属性によって参照されます。各 ID は固有である必要があります。pluginCollectionRef 属性の値が 1 つの backingMapPluginCollection エレメントの ID と一致しない場合、XML 妥当性検査は失敗します。任意の数の backingMap エレメントが、単一の backingMapPluginCollection エレメントを参照できます。 (必須)

```
<backingMapPluginCollection
(1)   id="id"
/>
```

以下の例では、companyGridCollection.xml ファイルを使用して、backingMapPluginCollection エレメントの使用法を示しています。このファイルでは、Customer BackingMap が customerPlugins backingMapPluginCollection を使用して、LRUEvictor を使用する Customer BackingMap を構成します。 Item および OrderLine の BackingMap は、collection2 backingMapPluginCollection を参照します。これらの BackingMap には、それぞれ LFUEvictor のセットが含まれています。

```
<?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="CompanyGrid">
   <backingMap name="Customer"
    pluginCollectionRef="customerPlugins"/>
```

```
   <backingMap name="Item" pluginCollectionRef="collection2"/>
   <backingMap name="OrderLine"
    pluginCollectionRef="collection2"/>
   <backingMap name="Order"/>
  </objectGrid>
 </objectGrids>
 <backingMapPluginCollections>
  <backingMapPluginCollection id="customerPlugins">
   <bean id="Evictor"
     className="com.ibm.websphere.objectGrid.plugins.builtins.LRUEvictor/>
  </backingMapPluginCollection>
  <backingMapPluginCollection id="collection2">
   <bean id="Evictor"
     className="com.ibm.websphere.objectgrid.plugins.builtins.LFUEvictor/>
   <bean id="OptimisticCallback"
     className="com.ibm.websphere.samples.objectgrid.EmployeeOptimisticCallBackImpl"/>
  </backingMapPluginCollection>
  </backingMapPluginCollections>
</objectGridConfig>
```

以下のコード・サンプルは、前の例の companyGridCollection.xml ファイルと同じ構成をプログラムで実現する方法を示しています。

```
ObjectGridManager objectGridManager = ObjectGridManagerFactory.getObjectGridManager();

ObjectGrid companyGrid = objectGridManager.createObjectGrid("CompanyGrid", false);
BackingMap customerMap = companyGrid .defineMap("Customer");
LRUEvictor customerEvictor = new LRUEvictor();
customerMap.setEvictor(customerEvictor);

BackingMap itemMap = companyGrid.defineMap("Item");
LFUEvictor itemEvictor = new LFUEvictor();
itemMap.setEvictor(itemEvictor);

BackingMap orderLineMap = companyGrid.defineMap("OrderLine");
LFUEvictor orderLineEvictor = new LFUEvictor();
orderLineMap.setEvictor(orderLineEvictor);

BackingMap orderMap = companyGrid.defineMap("Order");
```

querySchema エレメント

querySchema エレメントは、BackingMap 間のリレーションシップを定義し、各マップ内にあるオブジェクトのタイプを識別します。この情報は、照会言語ストリングをマップ・アクセス呼び出しに変換するために ObjectQuery によって使用されます。詳しくは、*プログラミング・ガイド* で ObjectQuery スキーマの定義の詳細を参照してください。

- 出現回数: 0 回から 1 回
- 子エレメント: mapSchemas エレメント、relationships エレメント

mapSchemas エレメント

各 querySchema エレメントは、1 つ以上の mapSchema エレメントを含む mapSchemas エレメントを 1 つ持ちます。

- 出現回数: 1 回
- 子エレメント: mapSchema エレメント

mapSchema エレメント

mapSchema エレメントは、BackingMap に保管されるオブジェクトのタイプ、およびそのデータにアクセスする方法を定義します。

- 出現回数: 1 回以上
- 子エレメント: なし

属性

mapName
スキーマに追加する BackingMap の名前を指定します。(必須)

valueClass
BackingMap の値部分に保管されるオブジェクトのタイプを指定します。(必須)

primaryKeyField
valueClass 属性内の 1 次キー属性の名前を指定します。1 次キーも BackingMap のキー部分に保管する必要があります。(オプション)

accessType
照会エンジンが valueClass オブジェクト・インスタンス内の永続データをイントロスペクトしてそのデータにアクセスする方法を示します。値を FIELD に設定すると、クラスのフィールドがイントロスペクトされ、スキーマに追加されます。値が PROPERTY の場合、get メソッドおよび is メソッドに関連付けられた属性が使用されます。デフォルト値は PROPERTY です。 (オプション)

```
<mapSchema
(1)   mapName="backingMapName"
(2)   valueClass="com.mycompany.OrderBean"
(3)   primaryKeyField="orderId"
(4)   accessType="PROPERTY" | "FIELD"
/>
```

以下の例では、companyGridQuerySchemaAttr.xml ファイルを使用して、サンプル mapSchema 構成を示しています。

```
<?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="CompanyGrid">
   <backingMap name="Order"/>
   <backingMap name="Customer"/>

   <querySchema>
    <mapSchemas>
     <mapSchema mapName="Order"
      valueClass="com.mycompany.OrderBean"
      primaryKeyField="orderNumber"
      accessType="FIELD"/>
     <mapSchema mapName="Customer"
      valueClass="com.mycompany.CustomerBean"
      primaryKeyField="id"
      accessType="FIELD"/>
    </mapSchemas>
   </querySchema>
  </objectGrid>
 </objectGrids>
</objectGridConfig>
```

以下のコード・サンプルは、前の例の companyGridQuerySchemaAttr.xml ファイルと同じ構成をプログラムで実現する方法を示しています。

```
ObjectGridManager objectGridManager = ObjectGridManagerFactory.getObjectGridManager();
ObjectGrid companyGrid = objectGridManager.createObjectGrid("CompanyGrid", false);
companyGrid.defineMap("Order");
companyGrid.defineMap("Customer");

// Define the schema
QueryConfig queryCfg = new QueryConfig();
queryCfg.addQueryMapping(new QueryMapping(
    "Order", OrderBean.class.getName(), "orderNumber", QueryMapping.FIELD_ACCESS));
queryCfg.addQueryMapping(new QueryMapping(
    "Customer", CustomerBean.class.getName(), "id", QueryMapping.FIELD_ACCESS));
companyGrid.setQueryConfig(queryCfg);
```

relationships エレメント

各 querySchema エレメントは、relationships エレメントをまったく持たないか (ゼロ)、または 1 つ以上の relationship エレメントを含む relationships エレメントを 1 つ持ちます。

- 出現回数: 0 回または 1 回
- 子エレメント: relationship エレメント

relationship エレメント

relationship エレメントは、2 つの BackingMap 間のリレーションシップ、およびそのリレーションシップをバインドする valueClass 属性の属性を定義します。

- 出現回数: 1 回以上
- 子エレメント: なし

属性

source
リレーションシップのソース側 valueClass の名前を指定します。 (必須)

target
リレーションシップのターゲット側 valueClass の名前を指定します。 (必須)

relationField
ソース valueClass 内でターゲットを参照している属性の名前を指定します。(必須)

invRelationField
ターゲット valueClass 内でソースを参照している属性の名前を指定します。この属性が指定されていないと、リレーションシップは単一方向となります。(オプション)

```
<mapSchema
(1)   source="com.mycompany.OrderBean"
(2)   target="com.mycompany.CustomerBean"
(3)   relationField="customer"
(4)   invRelationField="orders"
/>
```

以下の例では、companyGridQuerySchemaWithRelationshipAttr.xml ファイルを使用して、双方向リレーションシップを含むサンプル mapSchema 構成を示しています。

```
<?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="CompanyGrid">
   <backingMap name="Order"/>
   <backingMap name="Customer"/>

   <querySchema>
    <mapSchemas>
     <mapSchema mapName="Order"
      valueClass="com.mycompany.OrderBean"
      primaryKeyField="orderNumber"
      accessType="FIELD"/>
     <mapSchema mapName="Customer"
      valueClass="com.mycompany.CustomerBean"
      primaryKeyField="id"
      accessType="FIELD"/>
    </mapSchemas>
```

```
    <relationships>
     <relationship
      source="com.mycompany.OrderBean"
      target="com.mycompany.CustomerBean"
      relationField="customer"/>
      invRelationField="orders"/>
    </relationships>
   </querySchema>
  </objectGrid>
 </objectGrids>
</objectGridConfig>
```

以下のコード・サンプルは、前の例の companyGridQuerySchemaWithRelationshipAttr.xml ファイルと同じ構成をプログラムで実現する方法を示しています。

```
ObjectGridManager objectGridManager = ObjectGridManagerFactory.getObjectGridManager();
ObjectGrid companyGrid = objectGridManager.createObjectGrid("CompanyGrid", false);
companyGrid.defineMap("Order");
companyGrid.defineMap("Customer");

// Define the schema
QueryConfig queryCfg = new QueryConfig();
queryCfg.addQueryMapping(new QueryMapping(
    "Order", OrderBean.class.getName(), "orderNumber", QueryMapping.FIELD_ACCESS));
queryCfg.addQueryMapping(new QueryMapping(
    "Customer", CustomerBean.class.getName(), "id", QueryMapping.FIELD_ACCESS));
queryCfg.addQueryRelationship(new QueryRelationship(
     OrderBean.class.getName(), CustomerBean.class.getName(), "customer", "orders"));
companyGrid.setQueryConfig(queryCfg);
```

timeBasedDBUpdate エレメント

timeBasedDBUpdate エレメントは、時間ベースのデータベース・アップデーターのための構成を定義します。 timeBasedDBUpdate エレメントには、Java Persistence API (JPA) を使用して新しく挿入および更新されたレコードをデータベースから取り出す頻度、および対応する ObjectGrid マップ内のデータを更新する方法に関する情報が含まれています。

- 出現回数: 0 回または 1 回
- 子エレメント: なし

属性

entityClass

JPA プロバイダーと対話するために使用されるエンティティー・クラス名を指定します。このエンティティー・クラス名は、エンティティー照会を使用して JPA エンティティーを検索するために使用されます。(必須)

persistenceUnitName

JPA エンティティー・マネージャー・ファクトリーを作成するための JPA パーシスタンス・ユニット名を指定します。デフォルト値は、persistence.xml ファイル内で最初に定義されたパーシスタンス・ユニットの名前です。(オプション)

mode

時間ベースのデータベース更新モードを指定します。デフォルトでは、時間ベースのデータベース更新モードは、INVALIDATE_ONLY に設定されます。
INVALIDATE_ONLY タイプは、データベース内の対応するレコードが変更された場合に、ObjectGrid マップ内のエントリーを無効にすることを示します。
UPDATE_ONLY タイプは、ObjectGrid マップ内の既存のエントリーをデータベースの最新の値で更新することを示します。ただし、データベースに新たに挿入されたレコードは、すべて無視されます。INSERT_UPDATE タイプは、ObjectGrid マップ内の既存のエントリーをデータベースの最新の値で更新するこ

とを示します。またデータベースに新しく挿入されたレコードも、すべて ObjectGrid マップに挿入されます。(オプション)

timestampField
タイム・スタンプ・フィールドの名前を指定します。タイム・スタンプ・フィールドの値は、データベース・バックエンド・レコードが最後に更新された日時または順序を識別するために使用されます。 (オプション)

jpaPropertyFactory
JPAPropertyFactory 実装クラス名または spring Bean を示します。デフォルトの JPA プロパティーをオーバーライドするパーシスタンス・プロパティー・マップをプラグインするために、com.ibm.websphere.objectgrid.jpa.JPAPropertyFactory インターフェースが使用されます。JPAPropertyFactory インスタンス上で追加の属性を設定する必要がある場合は、Spring Framework Beans を使用してください。詳しくは、 217 ページの『Spring フレームワークとの統合』を参照してください。(オプション)

```
<timeBasedDBUpdate
(1)   persistenceUnitName="SamplePU"
(2)   mode="INVALIDATE_ONLY" | "UPDATE_ONLY" | "INSERT_UPDATE"
(3)   timestampField="TIMESTAMP"
(4)   entityClass="entity class"
(5)   jpaPropertyFactory="JPA property factory class" | "{spring}bean name"
/>
```

streamQuerySet エレメント

streamQuerySet エレメントは、ストリーム照会セットを定義するための最上位レベルのエレメントです。

- 出現回数: 0 回から複数回
- 子エレメント: stream エレメント、view エレメント

stream エレメント

stream エレメントは、ストリーム照会エンジンへのストリームを表します。stream エレメントの各属性は、StreamMetadata インターフェースのメソッドに対応します。

- 出現回数: 1 回から複数回
- 子エレメント: basic エレメント

属性

name
ストリームの名前を指定します。この属性が指定されていないと、妥当性検査は失敗します。(必須)

valueClass
このストリーム ObjectMap に保管される値のクラス・タイプを指定します。このクラス・タイプは、オブジェクトをストリーム・イベントに変換し、SQL ステートメントが指定されていない場合は SQL ステートメントを生成するために使用されます。 (必須)

sql ストリームの SQL ステートメントを指定します。このプロパティーを指定しな

かった場合、valueClass 属性で属性または accessor メソッドのリフレクション、またはエンティティー・メタデータの tuple 属性を使用してストリーム SQL が生成されます。(オプション)

access
値クラスの属性にアクセスするためのタイプを指定します。この値を FIELD に設定すると、属性は Java のリフレクションを使用してフィールドから直接取り出されます。そうでない場合、属性は accessor メソッドを使用して読み取られます。デフォルト値は PROPERTY です。 (オプション)

```
<stream
(1)   name="streamName"
(2)   valueClass="streamMapClassType"
(3)   sql="streamSQL create stream stockQuote
       keyed by t ( transactionvolume INTEGER, price DECIMAL (9,2),
        issue VARCHAR(100) );"
(4)   access="PROPERTY" | "FIELD"
/>
```

view エレメント

view エレメントはストリーム照会ビューを表します。各 stream エレメントが ViewMetadata インターフェースのメソッドに対応します。

- 出現回数: 1 回から複数回
- 子エレメント: basic エレメント、id エレメント

属性

name
ビューの名前を指定します。この属性が指定されていないと、妥当性検査は失敗します。(必須)

sql ビューの変換を定義する、ストリームの SQL を指定します。この属性が指定されていないと、妥当性検査は失敗します。(必須)

valueClass
ObjectMap のこのビューに保管される値のクラス・タイプを指定します。このクラス・タイプは、ビュー・イベントをこのクラス・タイプと互換性のある適切なタプル・フォーマットに変換するのに使用されます。クラス・タイプを指定しなかった場合、Stream Processing Technology Structured Query Language (SPTSQL) の列定義に従ってデフォルトのフォーマットが使用されます。このビュー・マップにエンティティー・メタデータが定義されていると、この属性は使用されません。代わりに、エンティティー・メタデータが使用されます。 (オプション)

access
値クラスの属性にアクセスするためのタイプを指定します。アクセス・タイプを FIELD に設定すると、列値は、Java のリフレクションを使用して直接にフィールドに設定されます。そうでない場合、accessor メソッドを使用して属性が設定されます。デフォルト値は PROPERTY です。 (オプション)

```
<view
(1)   name="viewName"
(2)   valueClass="viewMapValueClass"
(3)   sql="viewSQL CREATE VIEW last5MinuteAvgPrice AS
```

```
      SELECT issue, avg(price) as totalVolume
        FROM (SELECT * FROM stockQuote FETCH LATEST 5 MINUTES) group by issue;"/>
(4)   access="PROPERTY" | "FIELD"
/>
```

basic エレメント

basic エレメントは、値クラスまたはエンティティー・メタデータの属性名から SPTSQL で定義された列へのマッピングを定義するために使用します。

- 出現回数: 0 回から複数回
- 子エレメント: なし

属性

```
<basic
(1)   name="attributeName"
(2)   column="columnName"
/>
```

id エレメント

id エレメントは、キー属性のマッピングに使用されます。

- 出現回数: 0 回から複数回
- 子エレメント: なし

属性

```
<id
(1)   name="idName"
(2)   column="columnName"
/>
```

以下の例では、StreamQueryApp2.xml ファイルを使用して、streamQuerySet の属性を構成する方法を示しています。ストリーム照会セット _stockQuoteSQS_ にはストリームおよびビューが 1 つずつあります。ストリームおよびビューの両方で、名前、valueClass、sql、およびアクセス・タイプがそれぞれ定義されています。ストリームは basic エレメントも定義します。これにより、StockQuote クラスの volume 属性が、SQL ステートメントで定義される SQL 列 transactionvolume にマップされることが指定されます。

```
<?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="og1">
   <backingMap name="stockQuote" readOnly="false" copyKey="true"
    streamRef="stockQuote"/>
   <backingMap name="last5MinuteAvgPrice" readOnly="false" copyKey="false"
    viewRef="last5MinuteAvgPrice"/>

   <streamQuerySet name="stockQuoteSQS">
    <stream
     name="stockQuote"
     valueClass="com.ibm.ws.objectgrid.streamquery.sample.guide.StockQuote"
     sql="create stream stockQuote
         keyed by t ( transactionvolume INTEGER, price DECIMAL (9,2),
        issue VARCHAR(100) );"
     access="FIELD">
     <basic name="volume" column="transactionvolume"/>
    </stream>

    <view
```

```
      name="last5MinuteAvgPrice"
      valueClass="com.ibm.ws.objectgrid.streamquery.sample.guide.AveragePrice"
      sql="CREATE VIEW last5MinuteAvgPrice AS SELECT issue, avg(price) as avgPrice
        FROM (SELECT * FROM stockQuote FETCH LATEST 5 MINUTES) group by issue;"
      access="FIELD"
     </view>
    </streamQuerySet>
   </objectGrid>
  </objectGrids>
</objectGridConfig>
```

emd.xsd ファイル

エンティティー・メタデータ XML スキーマ定義を使用して記述子 XML ファイルを作成し、 WebSphere eXtreme Scale のエンティティー・スキーマを定義します。

emd.xsd ファイルの各エレメントおよび属性の説明は、184 ページの『エンティティー・メタデータ記述子 XML ファイル』を参照してください。

emd.xsd ファイル

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

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

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

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

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

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

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

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

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

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

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

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

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

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

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

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

<xsd:complexType name="emptyType/>
```

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

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

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

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

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

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

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

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

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

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

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

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

</xsd:schema>
```

セキュリティー記述子 XML ファイル

ObjectGrid セキュリティー記述子 XML ファイルを使用して、セキュリティーを使用可能にした eXtreme Scale デプロイメント・トポロジーを構成できます。このトピックでは、各種構成を説明するサンプル XML ファイルを提供します。

クラスター XML ファイルの各エレメントおよび属性の説明は、以下のリストに示されています。各サンプルでは、これらのエレメントおよび属性を使用して環境を構成する方法について説明しています。

securityConfig エレメント

securityConfig エレメントは、ObjectGrid セキュリティー XML ファイルの最上位エレメントです。このエレメントは、ファイルの名前空間とスキーマ・ロケーションをセットアップします。スキーマは objectGridSecurity.xsd ファイルで定義されます。

- 出現回数: 1 回
- 子エレメント: security

security エレメント

security エレメントは、ObjectGrid セキュリティーの定義に使用します。

- 出現回数: 1 回
- 子エレメント: authenticator、adminAuthorization、および systemCredentialGenerator

属性

securityEnabled
true に設定されているとき、グリッドのセキュリティーを使用可能にします。デフォルト値は false です。値を false に設定すると、グリッド全体のセキュリティーが使用不可になります。詳しくは、 324 ページの『グリッド・セキュリティー』を参照してください。 (オプション)

singleSignOnEnabled
値が true に設定されている場合は、クライアントがいずれか 1 つのサーバーに認識された後、任意のサーバーに接続できます。そうでない場合は、クライアントは接続のたびに各サーバーに対して認証を行う必要があります。デフォルト値は false です。(オプション)

loginSessionExpirationTime
ログイン・セッションが期限切れになるまでの時間を秒数で指定します。ログイン・セッションの有効期限が切れると、クライアントは再度認証する必要があります。(オプション)

adminAuthorizationEnabled
管理許可を使用可能にします。この値が true に設定されている場合は、すべての管理用タスクに許可が必要です。使用される許可メカニズムは、adminAuthorizationMechanism 属性の値により指定されます。デフォルト値は false です。(オプション)

adminAuthorizationMechanism
使用する許可メカニズムを示します。WebSphere eXtreme Scale は、2 種類の許可メカニズム、すなわち、 Java 認証・承認サービス (JAAS) とカスタム許可をサポートします。JAAS 許可メカニズムは、標準の JAAS ポリシー・ベースのアプローチを使用します。許可メカニズムとして JAAS を指定するには、値に AUTHORIZATION_MECHANISM_JAAS を設定します。カスタム許可メカニズムは、ユーザー・プラグイン AdminAuthorization 実装を使用します。カスタム許可メカニズムを指定するには、値に AUTHORIZATION_MECHANISM_CUSTOM を設定します。これら 2 つのメカニズムがどのように使用されるかについての詳細は、 328 ページの『アプリケーション・クライアントの許可』を参照してください。(オプション)

以下の `security.xml` ファイルは、 eXtreme Scale グリッド・セキュリティーを使用可能にするためのサンプル構成です。

security.xml

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

```
    <security securityEnabled="true" singleSignOnEnabled="true"
        loginSessionExpirationTime="20"
        adminAuthorizationEnabled="true"
        adminAuthorizationMechanism="AUTHORIZATION_MECHANISM_JAAS" >

        <authenticator className ="com.ibm.websphere.objectgrid.security.plugins.builtins.WSTokenAuthenticator">
        </authenticator>

        <systemCredentialGenerator className ="com.ibm.websphere.objectgrid.security.plugins.builtins.WSTokenCredentialGenerator">
            <property name="properties" type="java.lang.String" value="runAs" description="Using runAs subject" />
        </systemCredentialGenerator>

    </security>
</securityConfig>
```

authenticator エレメント

グリッド内の eXtreme Scale サーバーに対してクライアントを認証します。className 属性によって指定されるクラスは、com.ibm.websphere.objectgrid.security.plugins.Authenticator インターフェースを実装している必要があります。オーセンティケーターはプロパティーを使用し、className 属性によって指定されるクラスのメソッドを呼び出すことができます。プロパティーの使用について詳しくは、property エレメントを参照してください。

前記の `security.xml` ファイルの例では、com.ibm.websphere.objectgrid.security.plugins.builtins.WSTokenAuthenticator クラスがオーセンティケーターとして指定されています。このクラスは com.ibm.websphere.objectgrid.security.plugins.Authenticator インターフェースを実装します。

- 出現回数: 0 回または 1 回
- 子エレメント: property

属性

className

com.ibm.websphere.objectgrid.security.plugins.Authenticator インターフェースを実装するクラスを指定します。このクラスを使用して、eXtreme Scale グリッド内のサーバーに対してクライアントを認証します。(必須)

adminAuthorization エレメント

adminAuthorization エレメントは、グリッドへの管理アクセスをセットアップする場合に使用します。

- 出現回数: 0 回または 1 回
- 子エレメント: property

属性

className

com.ibm.websphere.objectgrid.security.plugins.AdminAuthorization インターフェースを実装するクラスを指定します。 (必須)

systemCredentialGenerator エレメント

systemCredentialGenerator エレメントを使用すると、システム・クレデンシャル生成プログラムがセットアップされます。このエレメントは、動的環境にのみ適用されます。動的構成モデルでは、動的コンテナー・サーバーは、 eXtreme Scale クライ

アントとしてカタログ・サーバーに接続し、カタログ・サーバーもクライアントとして eXtreme Scale コンテナー・サーバーに接続できます。このシステム・クレデンシャル生成プログラムは、システム・クレデンシャルのファクトリーを表すために使用します。

- 出現回数: 0 回または 1 回
- 子エレメント: property

属性

className
com.ibm.websphere.objectgrid.security.plugins.CredentialGenerator インターフェースを実装するクラスを指定します。 (必須)

systemCredentialGenerator の使用例については、前の `security.xml` をファイル参照してください。この例では、システム・クレデンシャル生成プログラムは com.ibm.websphere.objectgrid.security.plugins.builtins. WSTokenCredentialGenerator で、これはスレッドから RunAs Subject オブジェクトを取得します。

property エレメント

authenticator クラスおよび adminAuthorization クラスにおいて set メソッドを呼び出します。プロパティーの名前は、authenticator エレメントまたは adminAuthorization エレメントの className 属性の set メソッドに対応しています。

- 出現回数: 0 回以上
- 子エレメント: property

属性

name
プロパティーの名前を指定します。この属性に割り当てられる値は、このプロパティーを含む Bean の className 属性で指定されたクラスの set メソッドと対応している必要があります。例えば、Bean の className 属性が com.ibm.MyPlugin に設定され、指定されているプロパティーの名前が size である場合、com.ibm.MyPlugin クラスには setSize メソッドが必要です。(必須)

type
プロパティーのタイプを指定します。パラメーターのタイプは、name 属性により識別される set メソッドに渡されます。有効な値は、 Java プリミティブ、それに対応する java.lang プリミティブ、および java.lang.String です。name 属性と type 属性は、Bean の className 属性のメソッド・シグニチャーに対応していなければなりません。例えば、名前が size であり、タイプが int である場合は、Bean の className 属性で指定されたクラスに setSize(int) メソッドが存在している必要があります。(必須)

value
プロパティーの値を指定します。この値は type 属性によって指定されたタイプに変換され、次に name 属性と type 属性で識別された set メソッドへの呼び出しでパラメーターとして使用されます。この属性の値は、どんな方法でも妥当性検査されません。プラグイン・インプリメンターは、渡された値が有効であることを検証しなければなりません。 (必須)

description
プロパティーの説明を入力します。 (オプション)

詳しくは、『objectGridSecurity.xsd ファイル』を参照してください。

objectGridSecurity.xsd ファイル

次の ObjectGrid セキュリティー XML スキーマを使用して、eXtreme Scale デプロイメントに対するセキュリティーを使用可能にすることができます。

objectGridSecurity.xsd ファイルに定義されるエレメントおよび属性の説明は、197 ページの『セキュリティー記述子 XML ファイル』を参照してください。

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

 <xsd:element name="securityConfig">
  <xsd:complexType>
   <xsd:sequence>
    <xsd:element name="security" type="cc:security" />
   </xsd:sequence>
  </xsd:complexType>
 </xsd:element>

 <xsd:complexType name="security">
  <xsd:sequence>
   <xsd:element name="authenticator" type="cc:bean" minOccurs="0"
    maxOccurs="1" />
   <xsd:element name="adminAuthorization" type="cc:bean" minOccurs="0"
    maxOccurs="1" />
   <xsd:element name="systemCredentialGenerator" type="cc:bean" minOccurs="0"
    maxOccurs="1" />
  </xsd:sequence>
  <xsd:attribute name="securityEnabled" type="xsd:boolean" use="optional" />
  <xsd:attribute name="singleSignOnEnabled" type="xsd:boolean" use="optional"/>
  <xsd:attribute name="loginSessionExpirationTime" type="xsd:int" use="optional"/>
  <xsd:attribute name="adminAuthorizationMechanism" type="cc:adminAuthorizationMechanism"
   use="optional"/>
  <xsd:attribute name="adminAuthorizationEnabled" type="xsd:boolean" use="optional" />
 </xsd:complexType>

 <xsd:complexType name="bean">
  <xsd:sequence>
   <xsd:element name="property" type="cc:property" maxOccurs="unbounded" minOccurs="0" />
  </xsd:sequence>
  <xsd:attribute name="className" type="xsd:string" use="required" />
 </xsd:complexType>

 <xsd:complexType name="property">
  <xsd:attribute name="name" type="xsd:string" use="required" />
  <xsd:attribute name="value" type="xsd:string" use="required" />
  <xsd:attribute name="type" type="cc:propertyType" use="required" />
  <xsd:attribute name="description" type="xsd:string" use="optional" />
 </xsd:complexType>

 <xsd:simpleType name="propertyType">
  <xsd:restriction base="xsd:string">
   <xsd:enumeration value="java.lang.Boolean" />
   <xsd:enumeration value="boolean" />
   <xsd:enumeration value="java.lang.String" />
   <xsd:enumeration value="java.lang.Integer" />
   <xsd:enumeration value="int" />
   <xsd:enumeration value="java.lang.Double" />
   <xsd:enumeration value="double" />
   <xsd:enumeration value="java.lang.Byte" />
   <xsd:enumeration value="byte" />
   <xsd:enumeration value="java.lang.Short" />
   <xsd:enumeration value="short" />
   <xsd:enumeration value="java.lang.Long" />
   <xsd:enumeration value="long" />
   <xsd:enumeration value="java.lang.Float" />
```

```
    <xsd:enumeration value="float" />
    <xsd:enumeration value="java.lang.Character" />
    <xsd:enumeration value="char" />
   </xsd:restriction>
  </xsd:simpleType>

  <xsd:simpleType name="adminAuthorizationMechanism">
   <xsd:restriction base="xsd:string">
    <xsd:enumeration value="AUTHORIZATION_MECHANISM_JAAS" />
    <xsd:enumeration value="AUTHORIZATION_MECHANISM_CUSTOM" />
   </xsd:restriction>
  </xsd:simpleType>

</xsd:schema>
```

プロパティー・ファイルの解説

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。

サンプル・プロパティー・ファイル

extremescale_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・ファイルを作成することができます。

- sampleServer.properties
- sampleClient.properties

非推奨システム・プロパティー

-Dcom.ibm.websphere.objectgrid.CatalogServerProperties
このプロパティーは、WebSphere eXtreme Scale バージョン 7.0 で使用すべきではありません。**-Dobjectgrid.server.props** プロパティーを使用してください。

-Dcom.ibm.websphere.objectgrid.ClientProperties
このプロパティーは、WebSphere eXtreme Scale バージョン 7.0 で使用すべきではありません。**-Dobjectgrid.client.props** プロパティーを使用してください。

-Dobjectgrid.security.server.prop
このプロパティーは、WebSphere eXtreme Scaleバージョン 6.1.0.3 で使用すべきではありません。 **-Dobjectgrid.server.prop** プロパティーを使用してください。

-serverSecurityFile
この引数は、WebSphere eXtreme Scale バージョン 6.1.0.3 で使用すべきではありません。このオプションは、 startOgServer スクリプトに渡されます。 **-serverProps** 引数を使用してください。

関連概念

332 ページの『トランスポート層セキュリティーおよび Secure Sockets Layer』
WebSphere eXtreme Scale は、動的デプロイメント・モデルにおいてクライアントとサーバーとの間のセキュア通信に TCP/IP も、Transport Layer Security/Secure Sockets Layer (TLS/SSL) もサポートします。

関連タスク

236 ページの『スタンドアロン WebSphere eXtreme Scale サーバーの始動』
スタンドアロン WebSphere eXtreme Scale 構成を実行しているとき、環境はカタログ・サーバー、コンテナー・サーバー、および eXtreme Scale クライアント・プロセスで構成されています。これらのプロセスは手動で構成して開始する必要があります。

252 ページの『WebSphere Application Server による WebSphere eXtreme Scale の管理』
WebSphere Application Server でカタログ・サービスおよびコンテナー・サーバー・プロセスを実行できます。これらのサーバーを構成するプロセスは、スタンドアロン構成の場合とは異なります。カタログ・サービスは、WebSphere Application Server サーバーまたはデプロイメント・マネージャーで自動的に開始できます。eXtreme Scale アプリケーションが WebSphere Application Server 環境にデプロイされて、開始されるときに、コンテナー・プロセスは開始されます。

サーバー・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、サーバーのさまざまな設定 (例えば、トレース設定、ロギング、およびセキュリティー構成など) を定義する複数のプロパティーが含まれます。サーバー・プロパティー・ファイルは、カタログ・サービスとコンテナー・サービスによって使用されます。

サンプル・サーバー・プロパティー・ファイル

extremescale_root/properties ディレクトリーにある sampleServer.properties ファイルを使用して、プロパティー・ファイルを作成することができます。

サーバー・プロパティー・ファイルの指定

サーバー・プロパティー・ファイルは、以下のいずれかの方法で指定できます。このリスト内の項目のいずれかを使用して後で設定を指定すると、前の設定はオーバーライドされます。例えば、サーバー・プロパティー・ファイルのシステム・プロパティー値を指定すると、そのファイルのプロパティーにより、クラスパスにある objectGridServer.properties ファイルの値が指定変更されます。

1. クラスパス内のわかりやすい名前のファイルで指定。このわかりやすい名前のファイルを現行ディレクトリーに置いても、その現行ディレクトリーがクラスパスにない限りそのファイルは検出されません。使用される名前は次のようになります。

   ```
   objectGridServer.properties
   ```

2. システム現行ディレクトリー内のファイルを指定する、スタンドアロンまたは WebSphere Application Server 構成のいずれかのシステム・プロパティーとして指定。このファイルをクラスパスに入れることはできません。

   ```
   -Dobjectgrid.server.props=file_name
   ```

3. startOgServer コマンドを実行する際のパラメーターとして指定。次のように、これらのプロパティーを手動で指定変更して、システム現行ディレクトリーにファイルを指定することができます。

   `-serverProps` *`file_name`*

4. ServerFactory.getServerProperties メソッドおよび ServerFactory.getCatalogServerProperties メソッドを使用したプログラマチックな指定変更。オブジェクトのデータに、プロパティー・ファイルからのデータが取り込まれます。

サーバー・プロパティー

一般プロパティー

workingDirectory

コンテナー・サーバー出力が書き込まれるロケーションを指定します。この値が指定されない場合、出力は、現行ディレクトリー内の `log` ディレクトリーに書き込まれます。このプロパティーは、コンテナー・サーバーとカタログ・サービスの両方に適用されます。

デフォルト: 値なし

traceSpec

コンテナー・サーバーのトレースおよびトレース仕様ストリングを使用可能にします。トレースは、デフォルトで使用不可に設定されています。このプロパティーは、コンテナー・サーバーとカタログ・サービスの両方に適用されます。

デフォルト: `*=all=disabled`

traceFile

トレース情報を書き込むファイル名を指定します。このプロパティーは、コンテナー・サーバーとカタログ・サービスの両方に適用されます。

systemStreamToFileEnabled

コンテナーから、ファイルに SystemOut、SystemErr、およびトレース出力を書き込めるように設定します。このプロパティーが `false` に設定されていると、出力はファイルに書き込まれず、代わりにコンソールに書き込まれます。

デフォルト: `true`

enableMBeans

ObjectGrid コンテナーの Managed Beans (MBean) を使用可能にします。このプロパティーは、コンテナー・サーバーとカタログ・サービスの両方に適用されます。

デフォルト: `true`

serverName

サーバーを識別するために使用されるサーバーの名前を指定します。このプロパティーは、コンテナー・サーバーとカタログ・サービスの両方に適用されます。

zoneName

サーバーが含まれるゾーンの名前を設定します。このプロパティーは、コンテナー・サーバーとカタログ・サービスの両方に適用されます。

HAManagerPort

HA マネージャーが使用するポート番号を指定します。このプロパティーが設定されていない場合は、カタログ・サービスは使用可能なポートを自動的に生成します。このプロパティーは、コンテナー・サーバーとカタログ・サービスの両方に適用されます。

listenerHost

オブジェクト・リクエスト・ブローカー (ORB) のバインド先のホスト名を指定します。このプロパティーは、コンテナー・サーバーとカタログ・サービスの両方に適用されます。

listenerPort

オブジェクト・リクエスト・ブローカー (ORB) のバインド先のポート番号を指定します。このプロパティーは、コンテナー・サーバーとカタログ・サービスの両方に適用されます。

JMXServicePort

MBean サーバーが listen するポート番号を指定します。このプロパティーは、コンテナー・サーバーとカタログ・サービスの両方に適用されます。

コンテナー・サーバー・プロパティー

statsSpec

コンテナー・サーバーの統計仕様を指定します。

例:

```
all=disabled
```

memoryThresholdPercentage

メモリー・ベース除去のメモリーしきい値を設定します。このパーセンテージは、Java 仮想マシン (JVM) で使用される最大ヒープを指定し、これを超えると除去が行われます。デフォルト値は -1 で、メモリーしきい値が設定されていないことを示します。 memoryThresholdPercentage プロパティーが設定されていると、MemoryPoolMXBean 値は指定された値に設定されます。詳しくは、Java API 仕様の MemoryPoolMXBean インターフェースを参照してください。ただし、除去は Evictor で使用可能に設定されている場合にのみ行われます。メモリー・ベースの除去を使用可能に設定するには、*製品概要*を参照してください。このプロパティーは、コンテナー・サーバーにのみ適用されます。

catalogServiceEndPoints

カタログ・サービス・クラスターに接続するエンドポイントを指定します。この値は、形式 `host:port<,host:port>` で指定します。ここで、ホスト値は、 listenerHost 値で、ポート値は、カタログ・サーバーの listenerPort 値です。このプロパティーは、コンテナー・サーバーにのみ適用されます。

カタログ・サービス・プロパティー

domainName

複数ドメインに経路指定する際にクライアントに対してこのカタログ・サービス・グリッドを固有に識別するために使用するドメイン・ネームを指定します。このプロパティーは、カタログ・サービスにのみ適用されます。

enableQuorum

カタログ・サービスのクォーラムを使用可能にします。使用可能なコンテナ

ー・サーバー上の区画の配置を変更できるようにする前に、クォーラムを使用して、大半のカタログ・サービス・グリッドを確実に使用可能にします。クォーラムを使用可能にするには、この値を true または enabled に設定します。デフォルト値は disabled です。このプロパティーは、カタログ・サービスにのみ適用されます。

catalogClusterEndpoints
カタログ・サービスのカタログ・サービス・グリッド・エンドポイントを指定します。このプロパティーで、カタログ・サービス・エンドポイントを指定して、カタログ・サービス・グリッドを開始します。以下のフォーマット設定を使用します。

```
serverName:hostName:clientPort:peerPort<serverName:hostName:clientPort:peerPort>
```

このプロパティーは、カタログ・サービスにのみ適用されます。

heartBeatFrequencyLevel
ハートビートを発生させる頻度を指定します。 ハートビートの頻度レベルは、リソースの使用量と障害発見時間のトレードオフです。頻繁にハートビートが発生するほど、より多くのリソースが使用されますが、より迅速に障害は発見されます。このプロパティーは、カタログ・サービスにのみ適用されます。以下のいずれかの値を使用します。

- 0: 標準的な速度でのハートビート・レベルを指定します。この値を使用すると、リソースを過剰使用することなく適正な速度でフェイルオーバーの検出が行われます。 (デフォルト)
- -1: 積極的なハートビート・レベルを指定します。この値を指定すると、障害はより迅速に検出されますが、プロセッサーおよびネットワークのリソースもより多く使用します。このレベルは、サーバーが混んでいる場合に、欠落ハートビートの検出精度が高くなります。
- 1: 緩やかなハートビート・レベルを指定します。この値を使用すると、ハートビート頻度の減少により障害を検出するまでの時間は増加しますが、プロセッサーおよびネットワークの使用も減少します。

セキュリティー・サーバー・プロパティー

サーバー・プロパティー・ファイルは、 eXtreme Scale サーバー・セキュリティーの構成にも使用されます。単一サーバー・プロパティー・ファイルを使用して、基本的なプロパティーおよびセキュリティー・プロパティーの両方を指定できます。

一般セキュリティー・プロパティー

securityEnabled
true に設定すると、コンテナー・サーバー・セキュリティーが使用可能になります。デフォルト値は false です。このプロパティーは、カタログ・サーバーに提供される objectGridSecurity.xml ファイルに指定されている securityEnabled property と一致する必要があります。

credentialAuthentication
このサーバーが、クレデンシャル認証をサポートするかどうかを示します。次のいずれかの値を選択します。

- Never: サーバーは、クレデンシャル認証をサポートしません。

- Supported: クライアントがクレデンシャル認証をサポートする場合に、このサーバーもクレデンシャル認証をサポートします。
- Required: クライアントはクレデンシャル認証を必要とします。

クレデンシャル認証に関して詳しくは、326 ページの『アプリケーション・クライアントの認証』を参照してください。

Transport Layer Security の設定

transportType
サーバーのトランスポート・タイプを指定します。以下のいずれかの値を使用します。

- TCP/IP: サーバーが、TCP/IP 接続のみをサポートすることを示します。
- SSL-Supported: サーバーが TCP/IP 接続と Secure Sockets Layer (SSL) 接続の両方をサポートすることを示します。 (デフォルト)
- SSL-Required: サーバーが SSL 接続を必要とすることを示します。

SSL 構成プロパティー

別名 (alias)
鍵ストア内の別名を指定します。鍵ストアに複数の鍵ペア証明書があり、いずれか 1 つの証明書を選択したい場合は、このプロパティーを使用します。

デフォルト: 値なし

contextProvider
トラスト・サービスのコンテキスト・プロバイダーの名前を指定します。有効でない値を指定すると、コンテキスト・プロバイダー・タイプが正しくないことを示すセキュリティー例外が発生します。

有効値: IBMJSSE2、IBMJSSE、IBMJSSEFIPS など。

プロトコル
クライアントに使用するセキュリティー・プロトコルのタイプを指定します。このプロトコルの値は、使用する Java Secure Socket Extension (JSSE) プロバイダーに基づいて設定します。有効でない値を指定すると、プロトコル値が正しくないことを示すセキュリティー例外が発生します。

有効値: SSL、SSLv2、SSLv3、TLS、TLSv1 など。

keyStoreType
鍵ストアのタイプを示します。有効でない値を指定すると、ランタイム・セキュリティー例外が発生します。

有効値: JKS、JCEK、PKCS12 など。

trustStoreType
トラストストアのタイプを指定します。有効でない値を指定すると、ランタイム・セキュリティー例外が発生します。

有効値: JKS、JCEK、PKCS12 など。

keyStore
鍵ストア・ファイルへの完全修飾パスを指定します。

例:

etc/test/security/client.private

trustStore
トラストストア・ファイルへの完全修飾パスを指定します。

例:

etc/test/security/server.public

keyStorePassword
鍵ストアのストリング・パスワードを指定します。この値はエンコードすることも、実際の値を使用することもできます。

trustStorePassword
トラストストアのストリング・パスワードを指定します。この値はエンコードすることも、実際の値を使用することもできます。

clientAuthentication
このプロパティーを true に設定した場合は、SSL クライアントを認証する必要があります。SSL クライアントの認証は、クライアント証明書の認証とは異なります。クライアント証明書の認証とは、証明書チェーンに基づくユーザー・レジストリーと照合してクライアントを認証することです。このプロパティーは、サーバーが正しいクライアントに接続していることを保証します。

SecureTokenManager の設定

SecureTokenManager の設定は、サーバー相互認証の秘密ストリングの保護とシングル・サインオン・トークンの保護に使用されます。 324 ページの『グリッド・セキュリティー』を参照してください。

secureTokenManagerType
SecureTokenManager 設定のタイプを指定します。次の設定のいずれかを使用できます。

- none: セキュア・トークン・マネージャーが使用されないことを示します。
- default: WebSphere eXtreme Scale 製品で提供されるトークン・マネージャーが使用されることを示します。 SecureToken 鍵ストア構成を指定する必要があります。
- custom: SecureTokenManager 実装クラスを使用して指定した独自のトークン・マネージャーがあることを示しています。

customTokenManagerClass
SecureTokenManagerType プロパティー値を custom と指定した場合に、SecureTokenManager 実装クラスの名前を指定します。この実装クラスでは、デフォルトのコンストラクターがインスタンス化される必要があります。

customSecureTokenManagerProps
カスタム SecureTokenManager 実装クラス・プロパティーを指定します。このプロパティーは、secureTokenManagerType 値が custom の場合にのみ使用されます。この値は、setProperties(String) メソッドを使用して SecureTokenManager オブジェクトに設定されます。

セキュア・トークンの鍵ストア構成

secureTokenKeyStore
公開鍵と秘密鍵のペアおよび秘密鍵が保管されている鍵ストアのファイル・パス名を指定します。

secureTokenKeyStoreType
鍵ストアのタイプ (例、JCKES など) を指定します。この値は、使用している Java Secure Socket Extension (JSSE) プロバイダーに基づいて設定します。ただし、鍵ストアが秘密鍵をサポートしている必要があります。

secureTokenKeyPairAlias
署名および検証に使用される公開鍵と秘密鍵のペアの別名を指定します。

secureTokenKeyPairPassword
署名および検証に使用される鍵ペアの別名を保護するパスワードを指定します。

secureTokenSecretKeyAlias
暗号化に使用される秘密鍵の別名を指定します。

secureTokenSecretKeyPassword
秘密鍵を保護するパスワードを指定します。

secureTokenCipherAlgorithm
暗号の指定に使用されるアルゴリズムを指定します。この値は、使用している Java Secure Socket Extension (JSSE) プロバイダーに基づいて設定します。

secureTokenSignAlgorithm
オブジェクトの署名に使用されるアルゴリズムを指定します。この値は、使用される JSSE プロバイダーに基づいて設定します。

認証ストリング

authenticationSecret
サーバーのチャレンジに使用される秘密ストリングを指定します。サーバーは始動すると、このストリングをプレジデント・サーバーあるいはカタログ・サーバーに提示する必要があります。秘密ストリングが、プレジデント・サーバーにあるものと一致した場合に、このサーバーは参加できます。

関連概念

332 ページの『トランスポート層セキュリティーおよび Secure Sockets Layer』
WebSphere eXtreme Scale は、動的デプロイメント・モデルにおいてクライアントとサーバーとの間のセキュア通信に TCP/IP も、Transport Layer Security/Secure Sockets Layer (TLS/SSL) もサポートします。

関連タスク

236 ページの『スタンドアロン WebSphere eXtreme Scale サーバーの始動』
スタンドアロン WebSphere eXtreme Scale 構成を実行しているとき、環境はカタログ・サーバー、コンテナー・サーバー、および eXtreme Scale クライアント・プロセスで構成されています。これらのプロセスは手動で構成して開始する必要があります。

252 ページの『WebSphere Application Server による WebSphere eXtreme Scale の管理』
WebSphere Application Server でカタログ・サービスおよびコンテナー・サーバー・プロセスを実行できます。これらのサーバーを構成するプロセスは、スタンドアロン構成の場合とは異なります。カタログ・サービスは、WebSphere Application Server サーバーまたはデプロイメント・マネージャーで自動的に開始できます。eXtreme Scale アプリケーションが WebSphere Application Server 環境にデプロイされて、開始されるときに、コンテナー・プロセスは開始されます。

クライアント・プロパティー・ファイル

eXtreme Scale クライアント・プロセスの要件に基づいて、プロパティー・ファイルを作成できます。

サンプル・クライアント・プロパティー・ファイル

extremescale_root¥properties ディレクトリーにある sampleClient.properties ファイルを使用して、プロパティー・ファイルを作成することができます。

クライアント・プロパティー・ファイルの指定

クライアント・プロパティー・ファイルは、以下のいずれかの方法で指定できます。このリスト内の項目のいずれかを使用して後で設定を指定すると、前の設定はオーバーライドされます。例えば、クライアント・プロパティー・ファイルのシステム・プロパティー値を指定すると、そのファイルのプロパティーにより、クラスパスにある objectGridClient.properties ファイルの値が指定変更されます。

1. クラスパス内の任意の場所にあるわかりやすい名前のファイルで指定。このファイルをシステム現行ディレクトリーに置くことはサポートされていません。

   ```
   objectGridClient.properties
   ```

2. スタンドアロンまたは WebSphere Application Server 構成のいずれかでシステム・プロパティーとして指定。この値に、システム現行ディレクトリー内のファイルを指定することはできますが、クラスパス内のファイルは指定できません。

   ```
   -Dobjectgrid.client.props=file_name
   ```

3. ClientClusterContext.getClientProperties メソッドを使用したプログラマチックな指定変更。オブジェクトのデータに、プロパティー・ファイルからのデータが取り込まれます。セキュリティー・プロパティーは、このメソッドで構成できません。

クライアント・プロパティー

preferLocalProcess

ルーティングについてローカル・プロセスが優先されているかどうかを指定します。 true に設定すると、要求は、それが適切な場合は、クライアントと同じプロセスに配置されている断片に経路指定されます。

デフォルト: `true`

preferLocalHost

ルーティングについてローカル・ホストが優先されているかどうかを指定します。 true に設定すると、要求は、それが適切な場合はクライアントと同じホストに配置されている断片に経路指定されます。

デフォルト: `true`

preferZones

優先ルーティング・ゾーンのリストを指定します。各ゾーンは、`preferZones=ZoneA,ZoneB,ZoneC` の形式でコンマで区切って指定します。

デフォルト: 値なし

requestRetryTimeout

要求を再試行する時間をミリ秒で指定します。以下の有効値のいずれかを使用します。

- 値が `0` の場合、要求は、フェイル・ファーストになり、内部の再試行ロジックは飛ばされます。
- 値が `-1` の場合、要求再試行のタイムアウトは設定されません。すなわち、要求の所要時間は、トランザクション・タイムアウトによって制御されます。 (デフォルト)
- `0` より大きい値は、要求再試行のタイムアウト値をミリ秒で示しています。再試行しても正常に処理できない例外 (DuplicateException 例外など) は、即時に戻されます。トランザクション・タイムアウトは、最大待ち時間として引き続き使用されます。

セキュリティー・クライアント・プロパティー

一般セキュリティー・プロパティー

securityEnabled

WebSphere eXtreme Scale クライアント・セキュリティーを使用可能にします。このセキュリティー使用可能化設定は、WebSphere eXtreme Scale サーバー・プロパティー・ファイルの securityEnabled 設定と一致している必要があります。この設定が一致しない場合、例外が発生します。

デフォルト: `false`

クレデンシャル認証の構成プロパティー

credentialAuthentication

クライアントのクレデンシャル認証のサポートを指定します。以下の有効値のいずれかを使用します。

- `Never`: クライアントは、クレデンシャル認証をサポートしません。
- `Supported`: サーバーもクレデンシャル認証をサポートする場合に、クライアントはクレデンシャル認証をサポートします。 (デフォルト)

- Required: クライアントはクレデンシャル認証を必要とします。

authenticationRetryCount
クレデンシャルの有効期限が切れている場合に認証を再試行できる回数を指定します。この値が 0 に設定されていると、認証の再試行はできません。

デフォルト: 3

credentialGeneratorClass
com.ibm.websphere.objectgrid.security.plugins.CredentialGenerator インターフェースを実装するクラスの名前を指定します。このクラスを使用して、クライアントのクレデンシャルが取得されます。

デフォルト: 値なし

credentialGeneratorProps
CredentialGenerator 実装クラスのプロパティーを指定します。このプロパティーが、setProperties(String) メソッドを使用してオブジェクトに設定されます。 credentialGeneratorprops 値は、credentialGeneratorClass プロパティーの値が非ヌルの場合にのみ使用されます。

Transport Layer Security の構成プロパティー

transportType
クライアントのトランスポート・タイプを指定します。可能な値は以下のとおりです。

- TCP/IP: クライアントが、TCP/IP 接続のみをサポートすることを示します。
- SSL-Supported: クライアントが TCP/IP 接続と Secure Sockets Layer (SSL) 接続の両方をサポートすることを示します。(デフォルト)
- SSL-Required: クライアントが SSL 接続を必要とすることを示します。

SSL 構成プロパティー

別名 (alias)
鍵ストア内の別名を指定します。鍵ストアに複数の鍵ペア証明書があり、いずれか 1 つの証明書を選択したい場合は、このプロパティーを使用します。

デフォルト: 値なし

contextProvider
トラスト・サービスのコンテキスト・プロバイダーの名前を指定します。有効でない値を指定すると、コンテキスト・プロバイダー・タイプが正しくないことを示すセキュリティー例外が発生します。

有効値: IBMJSSE2、IBMJSSE、IBMJSSEFIPS など。

プロトコル
クライアントに使用するセキュリティー・プロトコルのタイプを指定します。このプロトコルの値は、使用する Java Secure Socket Extension (JSSE) プロバイダーに基づいて設定します。有効でない値を指定すると、プロトコル値が正しくないことを示すセキュリティー例外が発生します。

有効値: SSL、SSLv2、SSLv3、TLS、TLSv1 など。

keyStoreType
鍵ストアのタイプを示します。有効でない値を指定すると、ランタイム・セキュリティー例外が発生します。

有効値: JKS、JCEK、PKCS12 など。

trustStoreType
トラストストアのタイプを指定します。有効でない値を指定すると、ランタイム・セキュリティー例外が発生します。

有効値: JKS、JCEK、PKCS12 など。

keyStore
鍵ストア・ファイルへの完全修飾パスを指定します。

例:

```
etc/test/security/client.private
```

trustStore
トラストストア・ファイルへの完全修飾パスを指定します。

例:

```
etc/test/security/server.public
```

keyStorePassword
鍵ストアのストリング・パスワードを指定します。この値はエンコードすることも、実際の値を使用することもできます。

trustStorePassword
トラストストアのストリング・パスワードを指定します。この値はエンコードすることも、実際の値を使用することもできます。

関連概念

332 ページの『トランスポート層セキュリティーおよび Secure Sockets Layer』
WebSphere eXtreme Scale は、動的デプロイメント・モデルにおいてクライアントとサーバーとの間のセキュア通信に TCP/IP も、Transport Layer Security/Secure Sockets Layer (TLS/SSL) もサポートします。

関連タスク

236 ページの『スタンドアロン WebSphere eXtreme Scale サーバーの始動』
スタンドアロン WebSphere eXtreme Scale 構成を実行しているとき、環境はカタログ・サーバー、コンテナー・サーバー、および eXtreme Scale クライアント・プロセスで構成されています。これらのプロセスは手動で構成して開始する必要があります。

252 ページの『WebSphere Application Server による WebSphere eXtreme Scale の管理』
WebSphere Application Server でカタログ・サービスおよびコンテナー・サーバー・プロセスを実行できます。これらのサーバーを構成するプロセスは、スタンドアロン構成の場合とは異なります。カタログ・サービスは、WebSphere Application Server サーバーまたはデプロイメント・マネージャーで自動的に開始できます。eXtreme Scale アプリケーションが WebSphere Application Server 環境にデプロイされて、開始されるときに、コンテナー・プロセスは開始されます。

ORB プロパティー・ファイル

Object Request Broker (ORB) がグリッドのトランスポート動作を変更するために使用するプロパティーが、orb.properties ファイルを使用して受け渡されます。

ロケーション

orb.properties ファイルは、java/jre/lib ディレクトリーにあります。WebSphere Application Server java/jre/lib ディレクトリー内のファイルを変更すると、そのインストール済み環境に構成されているアプリケーション・サーバーも、そのファイルの設定を使用します。

ベースラインの設定

以下の設定は、適切なベースラインですが、必ずしもすべての環境に最適な設定とは限りません。ご使用の環境においてどの値が適切であるか、正しい決定をできるようにこれらの設定をよく理解してください。

```
com.ibm.CORBA.RequestTimeout=30
com.ibm.CORBA.ConnectTimeout=10
com.ibm.CORBA.FragmentTimeout=30
com.ibm.CORBA.ThreadPool.MinimumSize=256
com.ibm.CORBA.ThreadPool.MaximumSize=256
com.ibm.CORBA.ThreadPool.IsGrowable=false
com.ibm.CORBA.ConnectionMultiplicity=1
com.ibm.CORBA.MinOpenConnections=1024
com.ibm.CORBA.MaxOpenConnections=1024
com.ibm.CORBA.ServerSocketQueueDepth=1024
com.ibm.CORBA.FragmentSize=0
com.ibm.CORBA.iiop.NoLocalCopies=true
com.ibm.CORBA.NoLocalInterceptors=true
```

プロパティーの説明

タイムアウト設定

以下の設定は、ORB が要求の操作に見切りをつけるまで待機する時間に関係しています。

要求タイムアウト

プロパティー名: com.ibm.CORBA.RequestTimeout

値: 秒数を表す整数値。

説明: 要求 (任意) が応答を待機する秒数。その秒数を過ぎると待機を止めます。このプロパティーは、ネットワーク停止の障害が発生した場合にクライアントがフェイルオーバーするまでに要する時間に影響します。このプロパティーの値を極端に低く設定すると、要求が誤ってタイムアウトになる可能性があります。不用意なタイムアウトを回避するためにこのプロパティーの値は慎重に考慮してください。

接続タイムアウト

プロパティー名: com.ibm.CORBA.ConnectTimeout

値: 秒数を表す整数値。

説明: ソケット接続試行で待機する秒数。その秒数を過ぎると待機を止めます。このプロパティーは、要求タイムアウトと同様に、ネットワーク停止の障害が発生した場合にクライアントがフェイルオーバーするまでに要する時間に影響します。一般に、このプロパティーは要求タイムアウト値よりも小さい値に設定します。接続の確立に要する時間は比較的一定であるためです。

フラグメント・タイムアウト

プロパティー名: com.ibm.CORBA.FragmentTimeout

値: 秒数を表す整数値。

説明: フラグメント要求が待機する秒数。その秒数を過ぎると待機を止めます。このプロパティーは、要求タイムアウト・プロパティーと類似しています。

スレッド・プールの設定

このプロパティーは、スレッド・プール・サイズを特定のスレッド数に制約します。サーバー要求がソケットで受信されると、そのサーバー要求をスピンオフさせるために、ORB によってスレッドが使用されます。このプロパティー値を低い値に設定すると、ソケットのキュー項目数が増加して、タイムアウトになる可能性もあります。

接続多重度

プロパティー名: com.ibm.CORBA.ConnectionMultiplicity

値: クライアントとサーバーの間の接続数を表す整数値。デフォルト値は 1 です。これより大きい値に設定すると、複数接続にまたがる多重化の設定になります。**説明:** ORB が任意のサーバーとの複数の接続を使用できるように許可します。理論的に、この値の設定は、複数接続にまたがる並列性を促

進します。実際には、接続多重度の設定によるパフォーマンス上の利点はありません。このパラメーターは設定しないでください。

オープン接続

プロパティー名:
com.ibm.CORBA.MinOpenConnections、com.ibm.CORBA.MaxOpenConnections

値: 接続数を表す整数値。**説明:** オープン接続の最小数と最大数。 ORB は、クライアントとの間に確立された接続のキャッシュを保持します。 com.ibm.CORBA.MaxOpenConnections 値を超過すると、その接続は消去されます。接続の消去は、グリッド内の動作の低下の原因になる可能性があります。

成長可能

プロパティー名: com.ibm.CORBA.ThreadPool.IsGrowable

値: ブール値。 `true` または `false` に設定します。**説明:** これを使用可能に設定すると、 ORB が着信要求用に使用するスレッド・プールは、そのプールで現在サポートするものよりもさらに大きく成長します。プール・サイズを上回ると、要求の処理のために新規スレッドが作成されますが、そのスレッドはプールされません。

サーバー・ソケットのキュー項目数

プロパティー名: com.ibm.CORBA.ServerSocketQueueDepth

値: 接続数を表す整数値。**説明:** クライアントからの着呼接続のキューの長さを指定します。 ORB は、クライアントからの着呼接続をキューに入れます。キューがフルになると、接続は拒否されます。接続の拒否は、グリッド内の動作の低下の原因になる可能性があります。

フラグメント・サイズ

プロパティー名: com.ibm.CORBA.FragmentSize

値: バイト数を指定する整数。デフォルトは 1024 です。**説明:** ORB が要求の送信時に使用する最大パケット・サイズを指定します。要求がフラグメント・サイズ制限より大きい場合、その要求は要求フラグメントに分割されて、それぞれ別々に送信されて、サーバー上で再組み立てされます。要求のフラグメント化は、パケットの再送が必要になる可能性のある不安定なネットワークで有効です。ただし、ネットワークの信頼性が高い場合、要求をフラグメントに分割すると、オーバーヘッドの原因になる可能性があります。

ローカル・コピーなし

プロパティー名: com.ibm.CORBA.iiop.NoLocalCopies

値: ブール値。 `true` または `false` に設定します。 **説明:** ORB が参照による受け渡しをするかどうかを指定します。 ORB は、デフォルトで、値の呼び出しによる受け渡しを使用します。インターフェースがローカルで呼び出される際、値の呼び出しによる受け渡しは、パスに余分なガーベッジやシリアライゼーションのコストをもたらす原因になります。この値を true に設定すると、ORB は、値の呼び出しによる受け渡しよりも効率的な参照による受け渡し方式を使用します。

ローカル・インターセプターなし

プロパティー名: com.ibm.CORBA.NoLocalInterceptors

値: ブール値。 `true` または `false` に設定します。 **説明:** ローカル要求 (内部プロセス) の場合にも、ORB が要求インターセプターを呼び出すかどうかを指定します。 WebSphere eXtreme Scale が使用するインターセプターは、セキュリティーと経路処理を目的とし、要求が実行中のプロセス内で処理される場合には必須ではありません。プロセス間を仲介するインターセプターは、リモート・プロシージャー・コール (RPC) 操作の場合にのみ必要です。ローカル・インターセプターなしを設定すると、ローカル・インターセプターを使用することにより生じる余分なオーバーヘッドを回避できます。

ObjectGrid のクライアントとサーバー間でトランスポート・セキュリティーを実現するには、 `orb.properties` ファイルにさらにプロパティーを追加する必要があります。これらのプロパティーについて詳しくは、332 ページの『トランスポート層セキュリティーおよび Secure Sockets Layer』にあるトランスポート・セキュリティー・サポートの `orb.properties` ファイルに関するセクションを参照してください。

関連概念

332 ページの『トランスポート層セキュリティーおよび Secure Sockets Layer』
WebSphere eXtreme Scale は、動的デプロイメント・モデルにおいてクライアントとサーバーとの間のセキュア通信に TCP/IP も、Transport Layer Security/Secure Sockets Layer (TLS/SSL) もサポートします。

関連タスク

58 ページの『カスタム・オブジェクト・リクエスト・ブローカーの構成』
ご使用の環境でスタンドアロンの Java Platform, Standard Edition プロセスを実行している場合、WebSphere eXtreme Scale と一緒にオブジェクト・リクエスト・ブローカー (ORB) のカスタム・バージョンを使用できます。

30 ページの『WebSphere eXtreme Scale プロセスでのオブジェクト・リクエスト・ブローカーの使用』
WebSphere Application Server または WebSphere Application Server Network Deployment を含まない環境で、オブジェクト・リクエスト・ブローカー (ORB) を直接使用するアプリケーションを使用して WebSphere eXtreme Scale を使用することができます。

Spring フレームワークとの統合

Spring は、Java アプリケーションの開発によく使用されるフレームワークです。WebSphere eXtreme Scale では、 Spring を使用して eXtreme Scale トランザクションを管理し、デプロイされたメモリー内データ・グリッドに含まれるクライアントおよびサーバーの構成を行うことがサポートされています。

Spring 管理ネイティブ・トランザクション

Spring は、 Java Platform, Enterprise Edition アプリケーション・サーバーに似たコンテナー管理トランザクションを提供します。しかし、Spring メカニズムはさまざまな実装環境でプラグ可能です。WebSphere eXtreme Scale が提供するトランザクション・マネージャー統合は、Spring が ObjectGrid トランザクションのライフサイクルを管理することを可能にします。詳しくは、「*プログラミング・ガイド*」内のネ

イティブ・トランザクションに関する説明を参照してください。

Spring 管理拡張 Bean および名前空間のサポート

また、eXtreme Scale が Spring と統合されることによって、拡張ポイントまたはプラグイン用に Spring スタイルの Bean を定義することが可能になります。この機能によって、拡張ポイントの構成の柔軟性が高まり、洗練された構成ができるようになります。

Spring 管理の拡張 Bean に加えて、 eXtreme Scale は、「objectgrid」という名前の Spring 名前空間を提供します。Bean および組み込みの実装がこの名前空間に事前定義されていて、ユーザーが eXtreme Scale をより簡単に構成できるようになっています。これらのトピックに関する詳しい説明と、Spring 構成を使用して eXtreme Scale コンテナー・サーバーを開始する方法の例については、『Spring 拡張 Bean および名前空間のサポート』を参照してください。

断片有効範囲サポート

従来のスタイルの Spring 構成では、 ObjectGrid Bean は singleton タイプかプロトタイプ・タイプのどちらかです。ObjectGrid は、「断片」有効範囲と呼ばれる新しい有効範囲もサポートします。Bean が断片有効範囲と定義されている場合、断片当たり 1 つの Bean のみが作成されます。同じ断片内でその Bean 定義に一致する ID を持つ Bean に対する要求はすべて、その 1 つの特定の Bean インスタンスが Spring コンテナーによって戻される結果になります。

以下の例に示す com.ibm.ws.objectgrid.jpa.plugins.JPAPropFactoryImpl Bean の定義では、有効範囲が断片であると設定されています。したがって、断片当たり、JPAPropFactoryImpl クラスの 1 つのインスタンスのみが作成されます。

```
<bean id="jpaPropFactory" class="com.ibm.ws.objectgrid.jpa.plugins.JPAPropFactoryImpl" scope="shard" />
```

Spring Web Flow

Spring Web Flow は、デフォルトではセッション状態を HTTP セッションに保管します。Web アプリケーションがセッション管理に eXtreme Scale を使用するよう構成されている場合、この状態を保管するために Spring によってそれが自動的に使用され、セッションと同じ方法でフォールト・トレラントにされます。

パッケージ化

eXtreme Scale Spring 拡張は `ogspring.jar` ファイルに入っています。Spring サポートが正しく機能するためには、この Java アーカイブ (JAR) ファイルがクラスパスになければなりません。 WebSphere Extended Deployment で実行している JEE アプリケーションが WebSphere Application Server Network Deployment を拡張した場合、そのアプリケーションは `spring.jar` ファイルおよびその関連ファイルをエンタープライズ・アーカイブ (EAR) モジュールに入れる必要があります。同じ場所に `ogspring.jar` ファイルも入れる必要があります。

Spring 拡張 Bean および名前空間のサポート

WebSphere eXtreme Scale には、 `objectgrid.xml` ファイル内で拡張ポイントとして使用するために Plain Old Java Object (POJO) を宣言する機能があり、Bean を指定してからクラス名を指定する方法が提供されています。通常、指定されたクラスの

インスタンスが作成され、それらのオブジェクトはプラグインとして使用されます。eXtreme Scale は、これらのプラグイン・オブジェクトのインスタンスの取得を Spring に委任できます。アプリケーションが Spring を使用する場合は、通常、このような POJO をアプリケーションの残り部分に接続する必要があります。

場合によっては、特定のプラグイン・オブジェクトを構成するのに Spring を使用する必要があります。例として以下の構成を参考にしてください。

```
<objectGrid name="Grid">
    <bean id="TransactionCallback" className="com.ibm.websphere.objectgrid.jpa.JPATxCallback">
        <property name="persistenceUnitName" type="java.lang.String"  value="employeePU" />
    </bean>
    ...
</objectGrid>
```

組み込み TransactionCallback 実装である com.ibm.websphere.objectgrid.jpa.JPATxCallback クラスは、 TransactionCallback クラスとして構成されます。このクラスは上の例のように、1 つのプロパティー persistenceUnitName を使用して構成されます。 JPATxCallback クラスには JPAPropertyFactory 属性もあり、これのタイプは java.lang.Object です。ObjectGrid XML 構成は、このタイプの構成をサポートできません。

eXtreme Scale Spring 統合は Bean 作成を Spring フレームワークに委任することでこの問題を解決します。修正後の構成は、次のようになります。

```
<objectGrid name="Grid">
    <bean id="TransactionCallback" className="{spring}jpaTxCallback"/>
    ...
</objectGrid>
```

″Grid″ オブジェクト用の Spring ファイルには以下の情報が入っています。

```
<bean id="jpaTxCallback" class="com.ibm.websphere.objectgrid.jpa.JPATxCallback" scope="shard">
    <property name="persistenceUnitName" value="employeeEMPU"/>
    <property name="JPAPropertyFactory" ref ="jpaPropFactory"/>
</bean>

<bean id="jpaPropFactory" class="com.ibm.ws.objectgrid.jpa.plugins.
JPAPropFactoryImpl" scope="shard">
</bean>
```

ここでは、上の例に示されているように、`{spring}jpaTxCallback` として TransactionCallback が指定され、 Spring ファイル内に jpaTxCallback および jpaPropFactory Bean が構成されています。このような Spring 構成によって、JPAPropertyFactory Bean を JPATxCallback オブジェクトのパラメーターとして構成することが可能になります。

デフォルトの Spring Bean ファクトリー

eXtreme Scale が、接頭部 `{spring}` で始まる classname 値を持つプラグインまたは拡張 Bean (ObjectTransformer、Loader、TransactionCallback など) を検出した場合、eXtreme Scale は名前の残りの部分を Spring Bean 名として使用し、Spring Bean ファクトリーを使用して Bean インスタンスを取得します。

デフォルトでは、与えられた ObjectGrid 用に登録された Bean ファクトリーがない場合、`ObjectGridName_spring.xml` ファイルを見つけようとします。例えば、グリッドの名前が ″Grid″ の場合は、XML ファイルの名前は `/Grid_spring.xml` です。このファイルはクラスパスにあるか、クラスパス内の `META-INF` ディレクトリーに

あるはずです。このファイルが見つかったら、eXtreme Scale は、そのファイルを使用して ApplicationContext を作成し、その Bean ファクトリーから Bean を作成します。

カスタム Spring Bean ファクトリー

WebSphere eXtreme Scale には ObjectGridSpringFactory API もあり、これを使用して、特定の指定された ObjectGrid のために使用するよう Spring Bean ファクトリー・インスタンスを登録できます。この API は、以下の静的メソッドを使用して、BeanFactory のインスタンスを eXtreme Scale に登録します。

```
void registerSpringBeanFactoryAdapter(String objectGridName, Object
springBeanFactory)
```

名前空間サポート

バージョン 2.0 以降の Spring には、Bean の定義と構成のため、基本的な Spring XML フォーマットをスキーマ・ベースで拡張するメカニズムが備わっています。ObjectGrid はこの新しい機能を使用して、ObjectGrid Bean の定義と構成を行います。Spring XML スキーマ拡張では、eXtreme Scale プラグインのいくつかの組み込み実装、およびいくつかの ObjectGrid Bean が ″objectgrid″ 名前空間に事前定義されます。 Spring 構成ファイルを作成するとき、これらの組み込みの完全クラス名を指定する必要はありません。代わりに、事前定義された Bean を参照するようにできます。

また、XML スキーマ内に Bean の属性が定義されていることによって、間違った属性名を指定する可能性が減少します。XML スキーマに基づいた XML 妥当性検査は、この種のエラーを開発サイクルの初期にキャッチできます。

XML スキーマ拡張に定義されている Bean は、以下のとおりです。

- transactionManager
- register
- server
- カタログ (catalog)
- コンテナー
- JPALoader
- JPATxCallback
- JPAEntityLoader
- LRUEvictor
- LFUEvictor
- HashIndex

これらの Bean は objectgrid.xsd XML スキーマ内に定義されています。この XSD ファイルは、`ogspring.jar` ファイル中の `com/ibm/ws/objectgrid/spring/namespace/objectgrid.xsd` ファイルとして出荷されます。XSD ファイルおよび XSD ファイルで定義された Bean について詳しくは、「*管理ガイド*」に記載されている Spring 記述子ファイルに関する説明を参照してください。

前のセクションにある JPATxCallback 例をまた使用します。前のセクションでは、JPATxCallback Bean は次のように構成されていました。

```
<bean id="jpaTxCallback" class="com.ibm.websphere.objectgrid.jpa.JPATxCallback" scope="shard">
    <property name="persistenceUnitName" value="employeeEMPU"/>
    <property name="JPAPropertyFactory" ref ="jpaPropFactory"/>
</bean>

<bean id="jpaPropFactory" class="com.ibm.ws.objectgrid.jpa.plugins.JPAPropFactoryImpl" scope="shard">
</bean>
```

この名前空間フィーチャーを使用して、Spring XML 構成を次のようにコーディングできます。

```
<objectgrid:JPATxCallback id="jpaTxCallback" persistenceUnitName="employeeEMPU"
 jpaPropertyFactory="jpaPropFactory" />

<bean id="jpaPropFactory" class="com.ibm.ws.objectgrid.jpa.plugins.JPAPropFactoryImpl"
 scope="shard">
</bean>
```

ここでは、前の例でのように ″com.ibm.websphere.objectgrid.jpa.JPATxCallback″ クラスを指定する代わりに、事前定義された ″objectgrid:JPATxCallback″ Bean を直接使用することに注意してください。見て分かるように、この構成のほうが冗長でなく、誤りがないかチェックするのも簡単です。

Spring 拡張 Bean を使用したコンテナー・サーバーの開始

この例では、ObjectGrid Spring 管理拡張 Bean および名前空間のサポートを使用して、ObjectGrid サーバーを開始する方法を示します。

ObjectGrid XML ファイル

まず最初に、1 つの ObjectGrid ″Grid″ と 1 つのマップ ″Test″ が含まれているだけの、単純な ObjectGrid XML ファイルを定義します。この ObjectGrid には ″partitionListener″ という名前の ObjectGridEventListener プラグインがあり、マップ ″Test″ には ″testLRUEvictor″ という名前の Evictor プラグインがあります。ObjectGridEventListener プラグインと Evictor プラグインの両方とも、名前に ″{spring}″ が含まれるため、Spring を使用して構成されることに注意してください。

```
<?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="Grid">
     <bean id="ObjectGridEventListener" className="{spring}partitionListener" />
            <backingMap name="Test" pluginCollectionRef="test" />
        </objectGrid>
    </objectGrids>

    <backingMapPluginCollections>
        <backingMapPluginCollection id="test">
            <bean id="Evictor" className="{spring}testLRUEvictor"/>
        </backingMapPluginCollection>
    </backingMapPluginCollections>
</objectGridConfig>
```

ObjectGrid デプロイメント XML ファイル

次に、以下に示すように単純な ObjectGrid デプロイメント XML ファイルを作成します。これは ObjectGrid を 5 個の区画に分けます。複製は必要ありません。

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

```
        <mapSet name="mapSet" numInitialContainers="1" numberOfPartitions="5" minSyncReplicas="0"
            maxSyncReplicas="1" maxAsyncReplicas="0">
            <map ref="Test"/>
        </mapSet>
    </objectgridDeployment>
</deploymentPolicy>
```

ObjectGrid Spring XML ファイル

次に、 ObjectGrid Spring 管理拡張 Bean および名前空間のサポート機能を両方とも使用して、ObjectGrid Bean を構成します。spring xml ファイルの名前は ″Grid_spring.xml″ です。この XML ファイルには 2 つのスキーマが含まれていることに注意してください。spring-beans-2.0.xsd は Spring 管理 Bean を使用するためのもので、objectgrid.xsd は objectgrid 名前空間内に事前定義された Bean を使用するためのものです。

```
<beans xmlns="http://www.springframework.org/schema/beans"
       xmlns:xsi="http://www.w3.org/2001/XMLSchema-instance"
       xmlns:aop="http://www.springframework.org/schema/aop"
       xmlns:tx="http://www.springframework.org/schema/tx"
       xmlns:objectgrid="http://www.ibm.com/schema/objectgrid"
       xsi:schemaLocation="
       http://www.ibm.com/schema/objectgrid
   http://www.ibm.com/schema/objectgrid/objectgrid.xsd
       http://www.springframework.org/schema/beans
   http://www.springframework.org/schema/beans/spring-beans-2.0.xsd">

    <objectgrid:register id="ogregister" gridname="Grid"/>

    <objectgrid:server id="server" isCatalog="true" name="server">
        <objectgrid:catalog host="localhost" port="2809"/>
    </objectgrid:server>

    <objectgrid:container id="container"
   objectgridxml="com/ibm/ws/objectgrid/test/springshard/objectgrid.xml"
      deploymentxml="com/ibm/ws/objectgrid/test/springshard/deployment.xml"
   server="server"/>

    <objectgrid:LRUEvictor id="testLRUEvictor" numberOfLRUQueues="31"/>

    <bean id="partitionListener"
   class="com.ibm.websphere.objectgrid.springshard.ShardListener" scope="shard"/>
</beans>
```

この spring XML ファイルには、次の 6 個の Bean が定義されました。

1. *objectgrid:register*: これは、ObjectGrid ″Grid″ に対してデフォルトの Bean ファクトリーを登録します。
2. *objectgrid:server*: これは、″server″ という名前で ObjectGrid サーバーを定義します。objectgrid:catalog Bean がネストされているので、このサーバーはカタログ・サービスも提供します。
3. *objectgrid:catalog*: これは、″localhost:2809″ に設定された ObjectGrid カタログ・サービス・エンドポイントを定義します。
4. *objectgrid:container*: これは、前述したように、指定された objectgrid XML ファイルおよびデプロイメント XML ファイルと共に ObjectGrid コンテナーを定義します。server プロパティーは、このコンテナーがどのサーバーでホストされているのかを指定します。
5. *objectgrid:LRUEvictor*: これは、使用する LRU キューの数を 31 に設定して LRUEvictor を定義します。
6. *bean partitionListener*: これは ShardListener プラグインを定義します。このクラスは、ユーザーによってプラグインされるクラスであるため、事前定義された Bean を使用することはできません。また、この Bean の有効範囲は ″shard″ (断片) に設定されています。これは、この ShardListener のインスタンスが ObejctGrid 断片当たり 1 つのみであることを意味します。

サーバーの始動

以下のスニペットは、コンテナー・サービスとカタログ・サービスの両方をホストする ObjectGrid サーバーを開始します。これを見て分かるように、サーバーを開始するために呼び出す必要のあるメソッドは、Bean ファクトリーからの Bean ″container″ の get だけです。これは、ロジックの大部分を Spring 構成に移すことになり、プログラミングの複雑さが軽減されます。

```
public class ShardServer extends TestCase
{
    Container container;
    org.springframework.beans.factory.BeanFactory bf;

    public void startServer(String cep)
    {
        try
        {
            bf = new org.springframework.context.support.ClassPathXmlApplicationContext(
                "/com/ibm/ws/objectgrid/test/springshard/Grid_spring.xml", ShardServer.class);
            container = (Container)bf.getBean("container");
        }
        catch(Exception e)
        {
            throw new ObjectGridRuntimeException("Cannot start OG container", e);
        }
    }

    public void stopServer()
    {
        if(container != null)
            container.teardown();
    }
}
```

Spring 記述子 XML ファイル

Spring 記述子 XML ファイルを使用して、eXtreme Scale を構成して Spring と統合します。

以下のセクションにおいて、Spring objectgrid.xsd ファイルの各エレメントおよび属性が定義されています。 Spring objectgrid.xsd ファイルは、ogspring.jar ファイル、および objectgrid 名前空間 com/ibm/ws/objectgrid/spring/namespace にあります。記述子 XML スキーマの例については、 228 ページの『Spring objectgrid.xsd ファイル』を参照してください。

register エレメント

register エレメントを使用して、ObjectGrid のデフォルト Bean ファクトリーを登録します。

- 出現回数: 0 回から複数回
- 子エレメント: なし

属性

id 特定 ObjectGrid のデフォルト Bean ディレクトリーの名前を指定します。

gridname
ObjectGrid インスタンスの名前を指定します。この属性に割り当てられる値は、ObjectGrid 記述子ファイルに構成される有効な ObjectGrid に対応している必要があります。

```
<register
(1) id="register id"
(2) gridname="ObjectGrid name"
/>
```

server エレメント

server エレメントを使用して eXtreme Scale サーバーを定義します。eXtreme Scale サーバーは、コンテナー、カタログ・サービス、またはその両方をホストすることができます。

- 出現回数: 0 回から複数回
- 子エレメント: なし

属性

id eXtreme Scale サーバーの名前を指定します。

tracespec
トレースのタイプを示し、サーバーのトレースおよびトレース仕様を使用可能にします。

tracefile
作成および使用する traceFile のパスと名前を指定します。

statspec
サーバーの統計仕様を指定します。

jmxport
JMX/RMI 接続を使用可能にする未使用のポート番号を指定します。 JMX は、リモート・システムのモニターおよび管理を使用可能にします。

isCatalog
特定のサーバーがカタログ・サービスをホストするかどうかを指定します。デフォルト値は `false` です。

name
サーバーの名前を指定します。

```
<server
(1) id="server id"
(2) tracespec="the server trace specification"
(3) tracefile="the server trace file"
(4) statspec="the server statistic specification"
(5) jmxport="JMX port number"
(6) isCatalog="true"|"false"
(7) name="the server name"
/>
```

catalog エレメント

catalog エレメントを使用して、データ・グリッド内のコンテナー・サーバーにルーティングします。

- 出現回数: 0 回から複数回
- 子エレメント: なし

属性

ホスト (host)
カタログ・サービスが稼働しているワークステーションのホスト名を指定します。

ポート (port)
ホスト名と対になっているポート番号を指定します。これでクライアントが接続できるカタログ・サービス・ポートを決定します。

```
<catalog
(1) host="catalog service host name"
(2) port="catalog service port number"
/>
```

container エレメント

container エレメントを使用して、データそのものを保管します。

- 出現回数: 0 回から複数回
- 子エレメント: なし

属性

objectgridxml
使用する記述子 XML ファイルのパスと名前を指定します。このファイルは、マップ、ロック・ストラテジー、プラグインなどの ObjectGrid の特性を指定します。

deploymentxml
記述子 XML ファイルと一緒に使用して、区画化、複製、初期コンテナーの数、およびその他の設定を決定する XML ファイルのパスと名前を指定します。

server
コンテナーがホストされるサーバーを指定します。

```
<server
(1) objectgridxml="the objectgrid descriptor XML file"
(2) deploymentxml ="the objectgrid deployment descriptor XML file "
(3) server="the server reference "
/>
```

JPALoader エレメント

JPALoader エレメントを使用して、ObjectMap API の使用時に ObjectGrid キャッシュと既存のバックエンド・データ・ストアを同期します。

- 出現回数: 0 回から複数回
- 子エレメント: なし

属性

entityClassName
EntityManager.persist および EntityManager.find などの JPA の使用を可能にします。 **entityClassName** 属性は、JPALoader で必須です。

preloadPartition
マップ・プリロードが開始される区画番号を指定します。この値がゼロより小さいか、あるいは、(totalNumberOfPartition – 1) より大きい場合、マップ・プリロードは開始されません。

```
<JPALoader
(1) entityClassName="the entity class name"
(2) preloadPartition ="int"
/>
```

JPATxCallback エレメント

JPATxCallback エレメントを使用して、 JPA および ObjectGrid トランザクションを調整します。

- 出現回数: 0 回から複数回
- 子エレメント: なし

属性

persistenceUnitName
JPA EntityManagerFactory を作成し、persistence.xml ファイルで JPA エンティティー・メタデータを検索します。 **persistenceUnitName** 属性は必須です。

jpaPropertyFactory
パーシスタンス・プロパティー・マップを作成し、デフォルトのパーシスタンス・プロパティーをオーバーライドするためのファクトリーを指定します。この属性は、Bean を参照します。

exceptionMapper
JPA 固有あるいはデータベース固有の例外マッピング機能に使用できる ExceptionMapper プラグインを指定します。この属性は、Bean を参照します。

```
<JPATxCallback
(1) persistenceUnitName="the JPA persistence unit name"
(2) jpaPropertyFactory ="JPAPropertyFactory bean reference"
(3) exceptionMapper="ExceptionMapper bean reference"
/>
```

JPAEntityLoader エレメント

JPAEntityLoader エレメントを使用して、EntityManager API の使用時に ObjectGrid キャッシュと既存のバックエンド・データ・ストアを同期します。

- 出現回数: 0 回から複数回
- 子エレメント: なし

属性

entityClassName
EntityManager.persist および EntityManager.find などの JPA の使用を可能にします。 **entityClassName** 属性は、JPAEntityLoader エレメントでオプションです。このエレメントが構成されていないと、ObjectGrid エンティティー・マップに構成されているエンティティー・クラスが使用されます。 ObjectGrid EntityManager と JPA プロバイダーには、同じクラスが使用される必要があります。

preloadPartition
マップ・プリロードが開始される区画番号を指定します。この値がゼロより小さいか、あるいは、(totalNumberOfPartition – 1) より大きい場合、マップ・プリロードは起動されません。

```
<JPAEntityLoader
(1) entityClassName="the entity class name"
(2) preloadPartition ="int"
/>
```

LRUEvictor エレメント

LRUEvictor エレメントを使用して、マップがその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します。

- 出現回数: 0 回から複数回
- 子エレメント: なし

属性

maxSize
evictor の介入が必要になる、キュー内の合計エントリー数を指定します。

sleepTime
マップへの必要なアクションを決定するための、evictor のマップ・キューに対するスイープの間の時間を秒単位で設定します。

numberOfLRUQueues
マップ全体の大きさの単一キューにならないように、evictor がスキャンする必要があるキュー数の設定を指定します。

useMemoryUsageThresholdEviction
エントリーによって使用されるメモリー量に基づいて除去の設定を決めます。デフォルト値は `false` です。

```
<LRUEvictor
(1) maxSize="int"
(2) sleepTime ="seconds"
(3) numberOfLRUQueues ="int"
(4) useMemoryUsageThresholdEviction ="true"|"false"
/>
```

LFUEvictor エレメント

LFUEvictor エレメントを使用して、マップがその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します。

- 出現回数: 0 回から複数回
- 子エレメント: なし

属性

maxSize
evictor の対処が必要になる、各ヒープに許可される合計エントリー数を指定します。

sleepTime
マップへの必要なアクションを決定するための、evictor のマップ・ヒープに対するスイープの間の時間を秒単位で設定します。

numberOfHeaps
マップ全体の大きさの単一ヒープにならないように、evictor がスキャンする必要があるヒープ数の設定を指定します。

useMemoryUsageThresholdEviction
エントリーによって使用されるメモリー量に基づいて除去の設定を決めます。

```
<LFUEvictor
(1) maxSize="int"
(2) sleepTime ="seconds"
(3) numberOfHeaps ="int"
(4) useMemoryUsageThresholdEviction ="true"|"false"
/>
```

HashIndex エレメント

HashIndex エレメントを Java リフレクションで使用し、オブジェクトの更新時にマップに保管されるオブジェクトを動的にイントロスペクトします。

- 出現回数: 0 回から複数回
- 子エレメント: なし

属性

name
索引の名前を指定します。これは各マップで固有である必要があります。

attributeName
索引付けする属性の名前を指定します。フィールド・アクセス索引の場合、属性名は、フィールド名と同等です。プロパティー・アクセス索引の場合、属性名は、JavaBean と互換性のあるプロパティー名です。

rangeIndex
範囲の索引付けを使用可能にするかどうかを示します。デフォルト値は `false` です。

fieldAccessAttribute
非エンティティー・マップに使用されます。データのアクセスに getter メソッドが使用されます。デフォルト値は `false` です。値に `true` を指定した場合、フィールドを使用してオブジェクトに直接アクセスします。

POJOKeyIndex
非エンティティー・マップに使用されます。デフォルト値は `false` です。値に `true` を指定した場合、索引は、マップのキー部分でオブジェクトをイントロスペクトします。これは、キーが複合キーで、値にキーが組み込まれていない場合に有用です。値を設定しないか、または値に `false` を設定した場合、索引は、マップの値部分でオブジェクトをイントロスペクトします。

```
<HashIndex
(1) name="index name"
(2) attributeName="attribute name"
(3) rangeIndex ="true"|"false"
(4) fieldAccessAttribute ="true"|"false"

(5) POJOKeyIndex ="true"|"false"
/>
```

Spring objectgrid.xsd ファイル

Spring `objectgrid.xsd` ファイルを使用して、 eXtreme Scale を Spring と統合し、eXtreme Scale トランザクションの管理と、クライアントおよびサーバーの構成を行います。

Spring `objectgrid.xsd` ファイルに定義されるエレメントおよび属性の説明は、 223 ページの『Spring 記述子 XML ファイル』を参照してください。

Spring objectgrid.xsd ファイル

```
<xsd:schema xmlns="http://www.ibm.com/schema/objectgrid"
 xmlns:xsd="http://www.w3.org/2001/XMLSchema"
 xmlns:beans="http://www.springframework.org/schema/beans"
 targetNamespace="http://www.ibm.com/schema/objectgrid"
 elementFormDefault="qualified"
 attributeFormDefault="unqualified">

 <xsd:import namespace="http://www.springframework.org/schema/beans" />

  <xsd:element name="transactionManager">
  <xsd:complexType>
   <xsd:attribute name="id" type="xsd:ID" />
  </xsd:complexType>
 </xsd:element>

  <xsd:element name="register">
  <xsd:complexType>
   <xsd:attribute name="id" type="xsd:ID" />
   <xsd:attribute name="gridname" type="xsd:string" />
  </xsd:complexType>
```

```
</xsd:element>

 <xsd:element name="server">
 <xsd:complexType>
   <xsd:choice minOccurs="0" maxOccurs="unbounded">
   <xsd:element ref="catalog" />
  </xsd:choice>
  <xsd:attribute name="id" type="xsd:ID" />
  <xsd:attribute name="tracespec" type="xsd:string" />
  <xsd:attribute name="tracefile" type="xsd:string" />
  <xsd:attribute name="statspec" type="xsd:string" />
  <xsd:attribute name="jmxport" type="xsd:integer" />
  <xsd:attribute name="isCatalog" type="xsd:boolean" />
  <xsd:attribute name="name" type="xsd:string" />
 </xsd:complexType>
 </xsd:element>

 <xsd:element name="catalog">
  <xsd:complexType>
   <xsd:attribute name="host" type="xsd:string" />
   <xsd:attribute name="port" type="xsd:integer" />
  </xsd:complexType>
 </xsd:element>

 <xsd:element name="container">
 <xsd:complexType>
   <xsd:attribute name="id" type="xsd:ID" />
   <xsd:attribute name="objectgridxml" type="xsd:string" />
   <xsd:attribute name="deploymentxml" type="xsd:string" />
   <xsd:attribute name="server" type="xsd:string" />
  </xsd:complexType>
</xsd:element>

  <xsd:element name="JPALoader">
  <xsd:complexType>
   <xsd:attribute name="id" type="xsd:ID" />
   <xsd:attribute name="entityClassName" type="xsd:string" />
   <xsd:attribute name="preloadPartition" type="xsd:integer" />
  </xsd:complexType>
</xsd:element>

<xsd:element name="JPATxCallback">
 <xsd:complexType>
  <xsd:attribute name="id" type="xsd:ID" />
  <xsd:attribute name="persistenceUnitName" type="xsd:string" />
   <xsd:attribute name="jpaPropertyFactory" type="xsd:string" />
   <xsd:attribute name="exceptionMapper" type="xsd:string" />
  </xsd:complexType>
</xsd:element>

<xsd:element name="JPAEntityLoader">
 <xsd:complexType>
  <xsd:attribute name="id" type="xsd:ID />
  <xsd:attribute name="entityClassName" type="xsd:string" />
  <xsd:attribute name="preloadPartition" type="xsd:integer" />
 </xsd:complexType"
</xsd:element>

<xsd:element name="LRUEvictor">
 <xsd:complexType>
  <xsd:attribute name="id" type="xsd:ID" />
  <xsd:attribute name="maxSize" type="xsd:integer" />
  <xsd:attribute name="sleepTime" type="xsd:integer" />
  <xsd:attribute name="numberOfLRUQueues" type="xsd:integer" />
  <xsd:attribute name="useMemoryUsageThresholdEviction" type="xsd:boolean" />
 </xsd:complexType>
</xsd:element>

<xsd:element name="LFUEvictor">
 <xsd:complexType>
  <xsd:attribute name="id" type="xsd:ID" />
  <xsd:attribute name="maxSize" type="xsd:integer" />
  <xsd:attribute name="sleepTime" type="xsd:integer" />
  <xsd:attribute name="numberOfHeaps" type="xsd:integer" />
  <xsd:attribute name="useMemoryUsageThresholdEviction" type="xsd:boolean" />
 </xsd:complexType>
</xsd:element>

<xsd:element name="HashIndex">
 <xsd:complexType>
  <xsd:attribute name="id" type="xsd:ID" />
  <xsd:attribute name="name" type="xsd:string" />
  <xsd:attribute name="attributeName" type="xsd:string" />
  <xsd:attribute name="rangeIndex" type="xsd:boolean" />
  <xsd:attribute name="fieldAccessAttribute" type="xsd:boolean" />
  <xsd:attribute name="POJOKeyIndex" type="xsd:boolean" />
 </xsd:complexType"
</xsd:element>

</xsd:schema>
```

WebSphere Real Time の使用

WebSphere eXtreme Scale のもとで WebSphere Real Time を使用することができます。 WebSphere Real Time を使用可能にすることにより、ガーベッジ・コレクションはより予測可能になり、スタンドアロン eXtreme Scale 環境でのトランザクションの応答時間とスループットは安定した一貫性のあるものになります。

WebSphere Real Time を使用する理由

WebSphere eXtreme Scale は、各トランザクションに関連付けられる多数の一時オブジェクトを作成します。これらの一時オブジェクトは、要求、応答、ログ・シーケンス、およびセッションを処理します。 WebSphere Real Time がない場合は、トランザクションの応答時間が数百ミリ秒まで増大することがあります。しかし、WebSphere eXtreme Scale のもとで WebSphere Real Time を使用すると、ガーベッジ・コレクションの効率が上がり、応答時間がスタンドアロン構成応答時間の 10% に減少します。

WebSphere Real Time の使用可能化

Install eXtreme Scale を実行するコンピューターに、WebSphere Real Time とスタンドアロン WebSphere eXtreme Scale をインストールしてください。 JAVA_HOME 環境変数が標準 Java SE Runtime Environment (JRE) を指すように設定してください。

インストールされた WebSphere Real Time をポイントするよう、JAVA_HOME 環境変数を設定します。その後、次のようにして WebSphere Real Time を使用可能にします。

1. 次の行からコメントを除去することによって、スタンドアロン・インストール `objectgridRoot/bin/setupCmdLine.sh | .bat` ファイルを編集します。

   ```
   WXS_REAL_TIME_JAVA="-Xrealtime -Xgcpolicy:metronome
   -Xgc:targetUtilization=80"
   ```

2. ファイルを保存します。

これで、WebSphere Real Time が使用可能になります。WebSphere Real Time を使用不可にしたい場合は、同じ行にコメントを戻します。

ベスト・プラクティス

WebSphere WebSphere Real Time によって、eXtreme Scale トランザクションの応答時間はより予測可能なものになります。その結果、eXtreme Scale トランザクションの応答時間の偏差は、WebSphere Real Time を使用すると、デフォルトのガーベッジ・コレクターを使用する標準 Java と比較して、大幅に改善されます。 eXtreme Scale と共に WebSphere Real Time を使用可能にすることは、安定度および応答時間が重要なアプリケーションを使用している場合に最適です。

このセクションで説明するベスト・プラクティスは、WebSphere eXtreme Scale をより効率的にする調整方法と、予期される負荷に応じたコード例を示します。

- アプリケーションおよびガーベッジ・コレクター用のプロセッサー使用量を正しく設定する。

WebSphere Real Time にはプロセッサー使用量を制御する機能があり、ガーベッジ・コレクションがアプリケーションに与える影響を制御し、最小化することができます。 -Xgc:targetUtilization=NN パラメーターを使用して、20 秒ごとにアプリケーションが使用するプロセッサーの NN パーセントを指定します。WebSphere eXtreme Scale のデフォルトは 80% ですが、それとは異なる値 (例えば 70 など、ガーベッジ・コレクターにより多くのプロセッサー容量を提供する値) を設定するように objectgridRoot/bin/setupCmdLine.sh ファイル内のスクリプトを変更することができます。使用するアプリケーションのプロセッサー負荷を 80% 未満に保つことができる十分な数のサーバーをデプロイします。

- ヒープ・メモリーのサイズを大きく設定する。

  WebSphere Real Time は正規の Java より多くのメモリーを使用するため、大きいヒープ・メモリーを持つ WebSphere eXtreme Scale を計画して、カタログ・サーバーおよびコンテナーを開始する際、ogStartServer コマンドの –jvmArgs –XmxNNNM パラメーターでヒープ・サイズを設定してください。例えば、–jvmArgs –Xmx500M パラメーターを使用してカタログ・サーバーを開始し、適切なメモリー・サイズを使用してコンテナーを開始します。メモリー・サイズは、JVM ごとに予想データ・サイズの 60-70% に設定することができます。この値を設定しないと、OutOfMemoryError エラーが発生するおそれがあります。さらに、必要ならば、–jvmArgs –Xgc:noSynchronousGCOnOOM パラメーターを使用して、JVM がメモリー不足になったときの非決定的振る舞いを回避することができます。

- ガーベッジ・コレクションのスレッドを調整する。

  WebSphere eXtreme Scale は、各トランザクションおよびリモート・プロシージャー・コール (RPC) スレッドに関連付けられる多数の一時オブジェクトを作成します。ご使用のコンピューターに十分なプロセッサー・サイクルがある場合は、ガーベッジ・コレクションに対してパフォーマンスが有益に働きます。デフォルトのスレッド数は 1 です。スレッド数は –Xgcthreads n 引数によって変更できます。この引数の推奨値は、コンピューターごとの Java 仮想マシン数を考慮して使用可能となるコアの数です。

- WebSphere eXtreme Scale のもとで短時間実行されるアプリケーションのパフォーマンスを調整する。

  WebSphere Real Time は、長時間実行するアプリケーション向けに調整されています。通常、信頼できるパフォーマンス・データを取得するには、WebSphere eXtreme Scale のトランザクションを連続して 2 時間実行する必要があります。–Xquickstart パラメーターを使用して、短時間実行アプリケーションのパフォーマンスを最適にすることができます。このパラメーターは、JIT (Just-In-Time) コンパイラーに対して、低レベルの最適化を使用するように指示を出します。

- WebSphere eXtreme Scale クライアント・キューおよび WebSphere eXtreme Scale クライアント・リレーを最小化する。

  WebSphere eXtreme Scale を WebSphere Real Time と共に使用する主な利点は、信頼性の高いトランザクション応答時間を実現できることです。通常、トランザクション応答時間の偏差について、桁が数桁も違うような改善が見られます。キューに入れられたクライアント要求、および他のソフトウェアを経由するクライアント要求リレーは応答時間に影響しますが、それは WebSphere Real Time および WebSphere eXtreme Scale の制御範囲外です。スレッドおよびソケットのパラ

メーターを変更して、顕著な遅延のない安定的かつ平滑な負荷を維持し、キュー項目数を減らすようにする必要があります。

- WebSphere Real Time スレッド化を使用する WebSphere eXtreme Scale アプリケーションを開発する。

  アプリケーションを変更することなく、信頼性の高い WebSphere eXtreme Scale トランザクション応答時間を実現でき、応答時間の偏差に関して桁が数桁も違うほど改善されます。トランザクションを処理するユーザー・アプリケーションは、通常の Java スレッドではなく、スレッド優先順位およびスケジューリングの制御に優れた RealtimeThread を使用することで、スレッド使用の利点をさらに活用できます。

  アプリケーションが現在は以下のようなコードを含んでいるとします。

  ```
  public class WXSCacheAppImpl extends Thread implements WXSCacheAppIF
  ```

  必要ならば、このコードを次のもので置き換えることができます。

  ```
  public class WXSCacheAppImpl extends RealtimeThread implements
  WXSCacheAppIF
  ```

第 7 章 環境の管理

環境の管理には、スタンドアロン・モードおよび WebSphere Application Server 内の両方の場合のサーバーの始動および停止が含まれます。また、WebSphere eXtreme Scale を WebSphere Application Server 環境内のセッション・マネージャーとして使用することもできます。

このタスクについて

サーバー・タイプ

WebSphere eXtreme Scale には、*カタログ・サーバー* とコンテナー・サーバー の 2 タイプのサーバーがあります。カタログ・サーバーは、断片の配置を制御し、コンテナー・サーバーの検出とモニターをします。複数のカタログ・サーバーがまとまってカタログ・サービスを構成します。コンテナー・サーバーは、グリッドのアプリケーション・データを保管する Java 仮想マシンです。

スタンドアロン・モード

スタンドアロン・モードは、他のアプリケーション・サーバー製品を使用せずに単独で実行している WebSphere eXtreme Scale 構成のことを言います。

WebSphere Application Server 内での実行

WebSphere Application Server の上に WebSphere eXtreme Scale を実行している場合、カタログ・サーバーは WebSphere Application Server のサーバー上で自動的に始動します。コンテナー・サーバーを始動するには、ObjectGrid XML ファイルを使用してアプリケーションをパッケージ化し、デプロイする必要があります。

ObjectGrid の可用性の設定

ObjectGrid インスタンスの可用性状態は、特定の時点でどの要求を処理できるかを決定します。

以下の 4 つの可用性状態があります。

- ONLINE
- QUIESCE
- OFFLINE
- PRELOAD

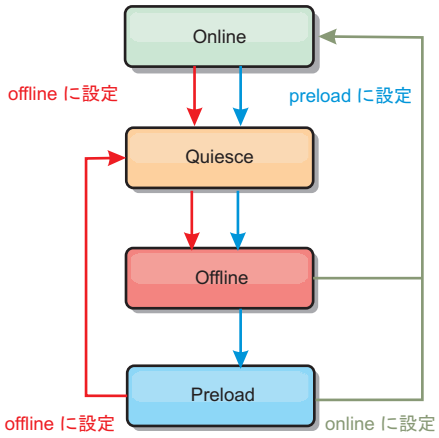


図 5. ObjectGrid の可用性状態

可用性状態の設定

ObjectGrid のデフォルトの可用性状態は ONLINE です。 ONLINE 状態の ObjectGrid は、標準 eXtreme Scale クライアントからのどの要求でも処理できます。ただし、プリロード・クライアントからの要求は、ObjectGrid が ONLINE である間は拒否されます。

QUIESCE 状態は、遷移的状態です。QUIESCE 状態にある ObjectGrid は、やがて OFFLINE 状態になります。QUIESCE 状態にある ObjectGrid は、未解決のトランザクションを処理できます。ただし新規トランザクションは拒否されます。ObjectGrid が QUIESCE 状態でいられるのは、30 秒までです。この時間を過ぎると、可用性状態は OFFLINE に移行します。

OFFLINE 状態にある ObjectGrid は、すべてのトランザクションを拒否します。

PRELOAD 状態は、プリロード・クライアントからデータを ObjectGrid にロードする場合に使用できます。ObjectGrid が PRELOAD 状態である間は、プリロード・クライアントしか、ObjectGrid に対してトランザクションをコミットできません。他のすべてのトランザクションは拒否されます。

ObjectGrid の可用性状態を設定するには、StateManager を使用します。クライアント ObjectGrid も、サーバー・サイド ObjectGrid も、StateManager インターフェースを使用してその可用性状態を変更できます。以下のコードは、クライアント ObjectGrid の可用性状態を変更する方法を示したものです。

```
ClientClusterContext client = ogManager.connect("localhost:2809", null, null);
ObjectGrid myObjectGrid = ogManager.getObjectGrid(client, "myObjectGrid");
StateManager stateManager = StateManagerFactory.getStateManager();
stateManager.setObjectGridState(AvailabilityState.OFFLINE, myObjectGrid);
```

StateManager で setObjectGridState メソッドが呼び出されると、ObjectGrid の各断片が要求状態に遷移します。メソッドが戻ると、ObjectGrid 内のすべての断片は、適切な状態になります。

サーバー・サイド ObjectGrid の可用性状態を変更するには、ObjectGridEventListener プラグインを使用します。サーバー・サイド ObjectGrid の可用性状態の変更は、その ObjectGrid の区画が 1 つだけの場合にのみ行ってください。ObjectGrid が複数

の区画を持っている場合、各プライマリーで shardActivated メソッドが呼び出され、結果として ObjectGrid の状態変更のために必要以上の呼び出しが行われることになります。

```
public class OGListener implements ObjectGridEventListener,
   ObjectGridEventGroup.ShardEvents {
    public void shardActivated(ObjectGrid grid) {
            StateManager stateManager = StateManagerFactory.getStateManager();
            stateManager.setObjectGridState(AvailabilityState.PRELOAD, grid);
    }
}
```

QUIESCE は遷移的状態であるため、ObjectGrid を QUIESCE 状態にするのに StateManager インターフェースを使用することはできません。ObjectGrid は、この状態を経てから OFFLINE 状態に移行します。

可用性状態の取得

特定の ObjectGrid の可用性状態を取得するには、StateManager の getObjectGridState メソッドを使用します。

```
StateManager stateManager = StateManagerFactory.getStateManager();
AvailabilityState state = stateManager.getObjectGridState(inventoryGrid);
```

getObjectGridState メソッドは、ObjectGrid 内のランダム・プライマリーを選択して、その AvailabilityState を返します。 ObjectGrid のすべての断片は同じ可用性状態になっているか、同じ可用性状態に遷移中である必要があるので、このメソッドにより、ObjectGrid の現在の可用性状態に関する妥当な結果がもたらされます。

さまざまな要求の適切な可用性状態

ObjectGrid が要求をサポートするのに適切な可用性状態にない場合、その要求は拒否されます。要求が拒否されると、必ず AvailabilityException 例外が発生します。

initialState 属性

initialState 属性は、ObjectGrid でその始動時の状態を示すのに使用できます。通常、初期化が完了した ObjectGrid は、ルーティングに使用可能になります。トラフィックが ObjectGrid にルーティングされないように、後で状態を変更できます。ObjectGrid を初期化する必要があるが、すぐには使用可能でない場合、initialState 属性を使用できます。

initialState 属性は、ObjectGrid の構成 XML ファイルで設定されます。デフォルト状態は ONLINE です。有効な値には、次のものが含まれます。

- ONLINE (デフォルト)
- PRELOAD
- OFFLINE

詳しくは、AvailabilityState API の資料を参照してください。

ObjectGrid で initialState が設定されている場合、その状態を明示的にオンラインに戻す必要があります。さもないと、ObjectGrid は使用不可のままになります。この場合 AvailabilityExceptions がスローされます。

プリロードのための initialState 属性の使用

ObjectGrid にデータがプリロードされる場合、ObjectGrid が使用可能になる時点と、クライアント・トラフィックをブロックするためにプリロード状態に切り替わる時点との間に、しばらく時間があくことがあります。この時間を回避するため、ObjectGrid の初期状態を PRELOAD に設定できます。この場合にも、ObjectGrid は必要な初期化をすべて完了しますが、状態が変更され、プリロードを実行できるようになるまで、トラフィックをブロックします。

PRELOAD 状態も OFFLINE 状態もトラフィックをブロックしますが、プリロードを開始する場合は PRELOAD 状態を使用する必要があります。

フェイルオーバーおよびバランシングの振る舞い

レプリカがプライマリーにプロモートされても、そのレプリカは initialState 設定を使用しません。プライマリーが再バランシングのために移動された場合、移動が完了する前に、データがプライマリーの新しいロケーションにコピーされるため、initialState 設定は使用されません。複製が構成されていない場合、フェイルオーバーが発生すると、プライマリーは initialState 値になるため、新規プライマリーを配置する必要があります。

スタンドアロン WebSphere eXtreme Scale サーバーの始動

スタンドアロン WebSphere eXtreme Scale 構成を実行しているとき、環境はカタログ・サーバー、コンテナー・サーバー、および eXtreme Scale クライアント・プロセスで構成されています。これらのプロセスは手動で構成して開始する必要があります。

始める前に

WebSphere Application Server がインストールされていない環境で WebSphere eXtreme Scale サーバーを始動できます。 WebSphere Application Server を使用している場合は、 252 ページの『WebSphere Application Server による WebSphere eXtreme Scale の管理』を参照してください。

次のタスク

eXtreme Scale プロセスを停止します。詳しくは、 249 ページの『スタンドアロン eXtreme Scale サーバーの停止』を参照してください。

関連概念

342 ページの『セキュア eXtreme Scale サーバーの開始と停止』
デプロイメント環境ではサーバーは保護される必要があることが多く、そのためには特別な構成が必要です。

関連資料

203 ページの『サーバー・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、サーバーのさまざまな設定 (例えば、トレース設定、ロギング、およびセキュリティー構成など) を定義する複数のプロパティーが含まれます。サーバー・プロパティー・ファイルは、カタログ・サービスとコンテナー・サービスによって使用されます。

210 ページの『クライアント・プロパティー・ファイル』
eXtreme Scale クライアント・プロセスの要件に基づいて、プロパティー・ファイルを作成できます。

202 ページの『プロパティー・ファイルの解説』
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。

244 ページの『startOgServer スクリプト』
startOgServer スクリプトはサーバーを始動します。サーバーの始動時に各種パラメーターを使用して、トレースを使用可能にしたり、ポート番号を指定するなど、さまざまな設定を行うことができます。

317 ページの『ログおよびトレース』
ログおよびトレースを使用して、環境のモニターおよびトラブルシューティングを実行できます。ログは、構成によってさまざまなロケーションにあります。IBM サポートに協力を依頼する場合、サーバーに関するトレースを提供する必要がある場合があります。

スタンドアロン環境でのカタログ・サービスの開始

WebSphere Application Server が実行されていない分散 WebSphere eXtreme Scale 環境を使用している場合は、カタログ・サービスを手動で開始する必要があります。

始める前に

WebSphere Application Server を使用している場合、カタログ・サービスは既存のいずれかのプロセス内で自動的に始動します。詳しくは、WebSphere Application Server でのカタログ・サービスの開始を参照してください。

このタスクについて

カタログ・サービスは単一のプロセスで実行することができます。また、複数のカタログ・サーバーを組み込んでカタログ・サーバー・グリッドを形成することもできます。実稼働環境では、高可用性を確保するためカタログ・サーバー・グリッドが必要となります。カタログ・サーバー・グリッドの構成について詳しくは、*製品概要* におけるカタログ・サービスのクラスター化に関する情報を参照してください。カタログ・サービスは、グリッドに配置されている場合も単一プロセス内にあ

る場合も、startOgServer スクリプトを使用して開始されます。また、このスクリプトに追加のパラメーターを指定することで、オブジェクト・リクエスト・ブローカー (ORB) を特定のホストおよびポートにバインドしたり、ドメインを指定したり、セキュリティーを使用可能にしたりできます。

開始コマンドを呼び出すには、UNIX プラットフォームでは startOgServer.sh スクリプトを、また、Windows では startOgServer.bat を使用します。

- **単一カタログ・サーバー・プロセスの開始**

  単一のカタログ・サーバーを始動するには、コマンド行から以下のコマンドを入力します。

  1. bin ディレクトリーに移動します。

     ```
     cd objectgridRoot/bin
     ```

  2. startOgServer コマンドを実行します。

     ```
     startOgServer.bat|sh catalogServer
     ```

  使用可能なすべてのコマンド行パラメーターのリストについては、244 ページの『startOgServer スクリプト』を参照してください。実稼働環境では、単一の Java 仮想マシン (JVM) を使用してカタログ・サービスを実行しないようにしてください。カタログ・サービスが失敗すると、新規クライアントをデプロイ済みの eXtreme Scale に経路指定することも、新規 ObjectGrid インスタンスをドメインに追加することもできません。これらの理由により、Java 仮想マシンのセットを始動してカタログ・サーバー・グリッドを実行するようにしてください。

- **マルチプロセス・カタログ・サーバー・グリッドの開始**

  サーバーのセットを開始してカタログ・サービスを実行するには、startOgServer スクリプトで **-catalogServiceEndPoints** オプションを使用する必要があります。この引数は、serverName:hostName:clientPort:peerPort の形式のカタログ・サービス・エンドポイントのリストを受け入れます。各属性の定義は次のとおりです。

  serverName
  起動しようとしているプロセスを識別する名前を指定します。

  hostName
  サーバーを起動するコンピューターのホスト名を指定します。

  clientPort
  ピア・カタログ・グリッド通信に使用されるポートを指定します。

  peerPort
  ピア・カタログ・グリッド通信に使用されるポートを指定します。

  以下の例は、カタログ・サービスをホストする 3 つの Java 仮想マシンのうち、最初のものを始動する方法を示しています。

  1. bin ディレクトリーに移動します。

     ```
     cd objectgridRoot/bin
     ```

  2. startOgServer コマンドを実行します。

```
startOgServer.bat|sh cs1 -catalogServiceEndpoints
cs1:MyServer1.company.com:6601:6602,
cs2:MyServer2.company.com:6601:6602,
cs3:MyServer3.company.com:6601:6602
```

この例では、MyServer1.company.com ホスト上の cs1 サーバーが始動されます。このサーバーの名前は、スクリプトに渡される最初の引数です。 cs1 サーバーの初期化時に、catalogServiceEndpoints パラメーターが検査されて、このプロセスに割り振られるポートが決定されます。このリストは、cs1 サーバーが他のサーバー (cs2 および cs3) からの接続を受け入れることができるようにするためにも使用されます。

3. リスト内の残りのカタログ・サーバーを開始するには、以下の引数を startOgServer スクリプトに渡します。MyServer2.company.com ホスト上の cs2 サーバーを始動します。

```
startOgServer.bat|sh cs2 -catalogServiceEndpoints
cs1:MyServer1.company.com:6601:6602,
cs2:MyServer2.company.com:6601:6602,
cs3:MyServer3.company.com:6601:6602
```

MyServer3.company.com 上の cs3 を始動します。

```
startOgServer.bat|sh cs3 -catalogServiceEndpoints
cs1:MyServer1.company.com:6601:6602,
cs2:MyServer2.company.com:6601:6602,
cs3:MyServer3.company.com:6601:6602
```

重要: すべてのカタログ・サーバーを同時に始動してください。

グリッドに入っているカタログ・サーバーは、それぞれのサーバーが、他のカタログ・サーバーがコア・グループに参加するのを休止して待つため、同時に始動する必要があります。グリッド用に構成されているカタログ・サーバーは、グループ内の他のメンバーを識別するまで始動しません。カタログ・サーバーは、他のサーバーがいずれも使用可能にならないと、最終的にタイムアウトになります。

- **特定のホストおよびポートへの ORB のバインド**

catalogServiceEndpoints 引数で定義されたポートを別にすれば、各カタログ・サービスも、オブジェクト・リクエスト・ブローカー (ORB) を使用して、クライアントおよびコンテナーからの接続を受け入れます。デフォルトでは、ORB はローカル・ホストのポート 2809 で listen します。カタログ・サービス JVM で特定のホストおよびポートに ORB をバインドする場合は、**-listenerHost** および **-listenerPort** 引数を使用します。次の例は、ORB が MyServer1.company.com のポート 7000 にバインドされた単一の JVM カタログ・サーバーを始動する方法を示しています。

```
startOgServer.sh catalogServer -listenerHost MyServer1.company.com
-listenerPort 7000
```

各 eXtreme Scale コンテナーおよびクライアントにカタログ・サービス ORB エンドポイント・データが提供されるようにする必要があります。クライアントにはこのデータのサブセットのみが必要ですが、高可用性のために少なくとも 2 つのエンドポイントを使用するようにしてください。

- **ドメインのネーミング**

カタログ・サービスを開始するとき、ドメイン・ネームは必要ではありません。デフォルトのドメイン・ネームは defaultDomain です。ドメインに名前を付ける

には、**-domain** オプションを使用します。次の例は、ドメイン・ネーム myDomain を持つ単一カタログ・サービス JVM の開始方法を示しています。

```
startOgServer.sh catalogServer -domain myDomain
```

- **セキュア・カタログ・サービスの開始**

  セキュア・カタログ・サービスを開始するには、以下の引数を指定します。

  -clusterSecurityFile および -clusterSecurityUrl
  これらの引数は objectGridSecurity.xml ファイルを指定します。このファイルは、すべてのサーバー (カタログ・サーバーおよびコンテナー・サーバーを含む) に共通するセキュア・プロパティーを記述するものです。プロパティー例の 1 つは、ユーザー・レジストリーおよび認証メカニズムを表すオーセンティケーター構成です。

  -serverProps
  サーバー固有のセキュリティー・プロパティーが含まれているサーバー・プロパティー・ファイルを指定します。このプロパティーに対して指定されるファイル名の形式は、プレーン・ファイル・パス形式です。例えば、c:/tmp/og/catalogserver.props などです。

  セキュア・カタログ・サービスを開始する方法の例は、*製品概要* にある Java SE セキュリティー・チュートリアルのステップ 2 を参照してください。
  objectGridSecurity.xml ファイルの例については、197 ページの『セキュリティー記述子 XML ファイル』を参照してください。

関連概念

342 ページの『セキュア eXtreme Scale サーバーの開始と停止』
デプロイメント環境ではサーバーは保護される必要があることが多く、そのためには特別な構成が必要です。

関連資料

244 ページの『startOgServer スクリプト』
startOgServer スクリプトはサーバーを始動します。サーバーの始動時に各種パラメーターを使用して、トレースを使用可能にしたり、ポート番号を指定するなど、さまざまな設定を行うことができます。

317 ページの『ログおよびトレース』
ログおよびトレースを使用して、環境のモニターおよびトラブルシューティングを実行できます。ログは、構成によってさまざまなロケーションにあります。IBM サポートに協力を依頼する場合、サーバーに関するトレースを提供する必要がある場合があります。

コンテナー・プロセスの開始

eXtreme Scale は、デプロイメント・トポロジー、または server.properties ファイルを使用して、コマンド行から開始できます。

このタスクについて

コンテナー・プロセスを開始するには、ObjectGrid XML ファイルが必要です。ObjectGrid XML ファイルは、コンテナーがホストする eXtreme Scale サーバーを指定します。コンテナーに渡される XML の各 ObjectGrid をホストするようにコンテナーが装備されていることを確認してください。これらの ObjectGrid が必要とす

るクラスは、すべてコンテナーのクラスパスになければなりません。 ObjectGrid XML ファイルに関して詳しくは、 179 ページの『objectGrid.xsd ファイル』を参照してください。

- **コマンド行からコンテナー・プロセスを開始します。**

  1. コマンド行から、bin ディレクトリーに移動します。

     ```
     cd objectgridRoot/bin
     ```

  2. 以下のコマンドを実行します。

     ```
     startOgServer.sh c0 -objectGridFile ../xml/companyGrid.xml
     -catalogServiceEndPoints MyServer1.company.com:2809
     ```

     重要: コンテナーでは、**-catalogServiceEndPoints** オプションを使用して、カタログ・サービス上のオブジェクト・リクエスト・ブローカー (ORB) のホストとポートを参照します。カタログ・サービスでは、**-listenerHost** および **-listenerPort** オプションを使用して ORB バインディング用のホストとポートを指定するか、またはデフォルトのバインディングを受け入れます。コンテナーを開始する場合は、**-catalogServiceEndPoints** オプションを使用して、カタログ・サービスで **-listenerHost** および **-listenerPort** オプションに渡される値を参照します。カタログ・サービスの開始時に **-listenerHost** および **-listenerPort** オプションが使用されなかった場合、ORB は、カタログ・サービスのローカル・ホストでポート 2809 にバインドします。カタログ・サービスで **-catalogServiceEndPoints** オプションに渡されたホストとポートを参照する場合に、**-catalogServiceEndPoints** オプションを使用しないでください。カタログ・サービスでは、**-catalogServiceEndPoints** オプションを使用して、静的サーバー構成に必要なポートを指定します。
     このプロセスは、スクリプトに渡される最初の引数 c0 によって識別されます。companyGrid.xml を使用してコンテナーを開始してください。カタログ・サーバー ORB が、コンテナーとは異なるホストで実行されているか、またはデフォルト以外のポートを使用している場合は、 **-catalogServiceEndPoints** 引数を使用してその ORB に接続する必要があります。この例では、単一のカタログ・サービスが MyServer1.company.com のポート 2809 で実行されているものと仮定します。

- **デプロイメント・ポリシーを使用してコンテナーを開始します。**

  必須ではありませんが、コンテナーの開始時には、デプロイメント・ポリシーが推奨されます。デプロイメント・ポリシーは、eXtreme Scale の区画化および複製をセットアップするために使用されます。デプロイメント・ポリシーは、配置方法に影響を与えるためにも使用されます。前述の例では、デプロイメント・ポリシー・ファイルが提供されなかったため、複製、区画化、および配置に関して、すべてのデフォルト値を受け取ります。したがって、CompanyGrid にあるマップは 1 つの mapSet 内に入ります。この mapSet は区画化も複製もされません。デプロイメント・ポリシー・ファイルについて詳しくは、 156 ページの『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。次の例では、companyGridDpReplication.xml ファイルを使用してコンテナー JVM (c0 JVM) を開始します。

  1. コマンド行から、bin ディレクトリーに移動します。

     ```
     cd objectgridRoot/bin
     ```

  2. 以下のコマンドを実行します。

```
startOgServer.sh c0 -objectGridFile ../xml/companyGrid.xml
-deploymentPolicyFile ../xml/companyGridDpReplication.xml
-catalogServiceEndPoints MyServer1.company.com:2809
```

注: Java クラスが特定のディレクトリーに保管されている場合には、StartOgServer スクリプトを変更する代わりに、`-jvmArgs -cp C:¥ . . . ¥DirectoryPOJOs¥POJOs.jar` というように引数を指定してサーバーを起動することができます。

. companyGridDpReplication.xml ファイルでは、単一のマップ・セットにすべてのマップが含まれています。この mapSet は 10 個の区画に分割されます。各区画には 1 つの同期複製が存在し、非同期複製は存在しません。また、companyGrid.xml ObjectGrid XML ファイルとペアになる companyGridDpReplication.xml デプロイメント・ポリシーを使用するコンテナーは、CompanyGrid 断片をホストできます。別のコンテナー JVM (c1 JVM) を開始します。

1. コマンド行から、bin ディレクトリーに移動します。

   ```
   cd objectgridRoot/bin
   ```

2. 以下のコマンドを実行します。

   ```
   startOgServer.sh c1 -objectGridFile ../xml/companyGrid.xml
   -deploymentPolicyFile ../xml/companyGridDpReplication.xml
   -catalogServiceEndPoints MyServer1.company.com:2809
   ```

各デプロイメント・ポリシーには、1 つ以上の objectgridDeployment エレメントが含まれています。コンテナーが開始されると、コンテナーは、デプロイメント・ポリシーをカタログ・サービスに公開します。カタログ・サービスは各 objectgridDeployment エレメントを検査します。 objectgridName 属性が、前に受信された objectgridDeployment エレメントの objectgridName 属性と一致する場合、最新の objectgridDeployment エレメントは無視されます。特定の objectgridName 属性用に受信された最初の objectgridDeployment エレメントがマスターとして使用されます。例えば、c2 JVM が、mapSet を異なる数の区画に分割するデプロイメント・ポリシーを使用するとします。

companyGridDpReplicationModified.xml

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

    <objectgridDeployment objectgridName="CompanyGrid">
        <mapSet name="mapSet1" numberOfPartitions="5"
            minSyncReplicas="1" maxSyncReplicas="1"
            maxAsyncReplicas="0">
            <map ref="Customer" />
            <map ref="Item" />
            <map ref="OrderLine" />
            <map ref="Order" />
        </mapSet>
    </objectgridDeployment>

</deploymentPolicy>
```

これで、3 番目の JVM である c2 JVM を開始できます。

1. コマンド行から、bin ディレクトリーに移動します。

   ```
   cd objectgridRoot/bin
   ```

2. 以下のコマンドを実行します。

```
startOgServer.sh c2 -objectGridFile ../xml/companyGrid.xml
-deploymentPolicyFile ../xml/companyGridDpReplicationModified.xml
-catalogServiceEndPoints MyServer1.company.com:2809
```

c2 JVM 上のコンテナーが、mapSet1 に 5 つの区画を指定するデプロイメント・ポリシーで開始されます。しかし、カタログ・サービスは、CompanyGrid の objectgridDeployment のマスター・コピーを既に保持しています。c0 JVM は開始されたときに、この mapSet に 10 個の区画を指定しました。 c0 が、デプロイメント・ポリシーを開始および公開する最初のコンテナーであったため、c0 のデプロイメント・ポリシーがマスターになりました。したがって、後続のデプロイメント・ポリシー内の CompanyGrid に等しい objectgridDeployment 属性値はすべて無視されます。

- **サーバー・プロパティー・ファイルを使用してコンテナーを開始します。**

  サーバー・プロパティー・ファイルを使用して、コンテナーでのトレースをセットアップし、セキュリティーを構成することができます。次のコマンドを実行し、サーバー・プロパティー・ファイルを使用してコンテナー c3 を開始します。

  1. コマンド行から、bin ディレクトリーに移動します。

     ```
     cd objectgridRoot/bin
     ```

  2. 以下のコマンドを実行します。

```
startOgServer.sh c3 -objectGridFile ../xml/companyGrid.xml
-deploymentPolicyFile ../xml/companyGridDpReplicationModified.xml
-catalogServiceEndPoints MyServer1.company.com:2809
-serverProps ../serverProps/server.properties
```

server.properties ファイルの例を次に示します。

```
server.properties
workingDirectory=
traceSpec=*=all=disabled
systemStreamToFileEnabled=true
enableMBeans=true
memoryThresholdPercentage=50
```

これは、セキュリティーを有効にしていない基本的なサーバー・プロパティー・ファイルです。 server.properties ファイルに関して詳しくは、 203 ページの『サーバー・プロパティー・ファイル』を参照してください。

関連概念

342 ページの『セキュア eXtreme Scale サーバーの開始と停止』
デプロイメント環境ではサーバーは保護される必要があることが多く、そのためには特別な構成が必要です。

関連資料

『startOgServer スクリプト』
startOgServer スクリプトはサーバーを始動します。サーバーの始動時に各種パラメーターを使用して、トレースを使用可能にしたり、ポート番号を指定するなど、さまざまな設定を行うことができます。

317 ページの『ログおよびトレース』
ログおよびトレースを使用して、環境のモニターおよびトラブルシューティングを実行できます。ログは、構成によってさまざまなロケーションにあります。IBM サポートに協力を依頼する場合、サーバーに関するトレースを提供する必要がある場合があります。

startOgServer スクリプト

startOgServer スクリプトはサーバーを始動します。サーバーの始動時に各種パラメーターを使用して、トレースを使用可能にしたり、ポート番号を指定するなど、さまざまな設定を行うことができます。

目的

startOgServer スクリプトを使用してサーバーを始動することができます。

ロケーション

startOgServer スクリプトは、ルート・ディレクトリーの bin ディレクトリーにあります。例えば、次のように入力します。

```
cd objectgridRoot/bin
```

注: Java クラスが特定のディレクトリーに保管されている場合には、 StartOgServer スクリプトを変更する代わりに、`-jvmArgs -cp C:¥ . . . ¥DirectoryPOJOs¥POJOs.jar` というように引数を指定してサーバーを起動することができます。

カタログ・サーバーの場合の使用法

カタログ・サーバーを始動する場合:

Windows

```
startOgServer.bat <server> [options]
```

UNIX

```
startOgServer.sh <server>[options]
```

デフォルトの構成済みカタログ・サーバーを始動するには、以下のコマンドを使用します。

Windows

```
startOgServer.bat catalogServer
```

UNIX

```
startOgServer.sh catalogServer
```

カタログ・サーバーの始動のオプション

カタログ・サーバーの始動のためのパラメーター:

-catalogServiceEndPoints
<server:serverHost:clientPort:peerPort,server:serverHost:clientPort:peerPort>
コンテナーでは、**-catalogServiceEndpoints** オプションを使用して、カタログ・サービスでオブジェクト・リクエスト・ブローカー (ORB) のホストとポートが参照されます。

-clusterSecurityFile <cluster security xml file>
セキュリティー記述子 XML ファイルが置かれているロケーションを指定します。

-clusterSecurityUrl <cluster security xml URL>

-domain <domain name>

-listenerHost <host name>
デフォルト: `localhost`

Internet Inter-ORB Protocol (IIOP) との通信に使用するリスナー・ホストを指定します。

-listenerPort <port>
デフォルト: `2809`

IIOP との通信に使用するリスナー・ポートを指定します。

-serverProps <server properties file>
サーバー固有のセキュリティー・プロパティーが含まれているサーバー・プロパティー・ファイルを指定します。このプロパティーに対して指定されるファイル名の形式は、単なるプレーン・ファイル・パス形式です。例えば、`c:/tmp/og/catalogserver.props` などです。

-JMXServicePort <port>
デフォルト: `1099`

Java Management Extensions (JMX) との通信に使用するポート番号を指定します。

-traceSpec <trace specification>
サーバーが始動したとき使用可能になるトレースの有効範囲を指定するストリングを指定します。

例:

- `ObjectGrid=all=enabled`
- `ObjectGrid*=all=enabled`

-traceFile <trace file>
トレース情報を保存するファイルのパスを指定します。

例: `../logs/c4Trace.log`

-timeout <seconds>
サーバーの始動がタイムアウトになる秒数を指定します。

-script <script file>

-jvmArgs <JVM arguments>
JVM 引数 (複数可) を指定します。 **-jvmArgs** パラメーターより後にあるパラメーターは、すべてサーバー Java 仮想マシン (JVM) を始動するために使用されるものです。 **-jvmArgs** パラメーターを使用するときは、最後に指定されるスクリプト引数 (オプション) になるようにしてください。

例: `-jvmArgs -Xms256M -Xmx1G`

コンテナー・サーバーの場合の使用法

Windows

```
startOgServer.bat <server> -objectgridFile <xml file>
-deploymentPolicyFile <xml file> [options]
```

Windows

```
startOgServer.bat <server> -objectgridUrl <xml URL>
-deploymentPolicyUrl <xml URL> [options]
```

UNIX

```
startOgServer.sh <server> -objectgridFile <xml file>
-deploymentPolicyFile <xml file> [options]
```

UNIX

```
 startOgServer.sh <server> -objectgridUrl <xml URL>
-deploymentPolicyUrl <xml URL> [options]
```

コンテナー・サーバーのオプション

-catalogServiceEndPoints<hostName:port,hostName:port>
デフォルト: localhost:2809

-deploymentPolicyFile <deployment policy xml file>
デプロイメント・ポリシー・ファイルへのパスを指定します。

例: `../xml/SimpleDP.xml`

-deploymentPolicyUrl <deployment policy xml url>

-listenerHost <host name>
デフォルト: `localhost`

Internet Inter-ORB Protocol (IIOP) との通信に使用するリスナー・ホストを指定します。

-listenerPort <port>
デフォルト: `2809`

IIOP との通信に使用するリスナー・ポートを指定します。

-serverProps <server properties file>
サーバー・プロパティー・ファイルへのパスを指定します。

例:`../security/server.props`

-zone <zone name>
サーバー内のすべてのコンテナーのために使用するゾーンを指定します。

-traceSpec <trace specification>
サーバーが始動したとき使用可能になるトレースの有効範囲を指定するストリングを指定します。

例:

- `ObjectGrid=all=enabled`
- `ObjectGrid*=all=enabled`

-traceFile <trace file>
トレース情報を保存するファイルのパスを指定します。

例: `../logs/c4Trace.log`

-timeout <seconds>
サーバーの始動がタイムアウトになる秒数を指定します。

-script <script file>

-jvmArgs <JVM arguments>
JVM 引数 (複数可) を指定します。 **-jvmArgs** パラメーターより後にあるパラメーターは、すべてサーバー Java 仮想マシン (JVM) を始動するために使用されるものです。 **-jvmArgs** パラメーターを使用するときは、最後に指定されるスクリプト引数 (オプション) になるようにしてください。

例:`-jvmArgs -Xms256M -Xmx1G`

関連概念

342 ページの『セキュア eXtreme Scale サーバーの開始と停止』
デプロイメント環境ではサーバーは保護される必要があることが多く、そのためには特別な構成が必要です。

関連タスク

236 ページの『スタンドアロン WebSphere eXtreme Scale サーバーの始動』
スタンドアロン WebSphere eXtreme Scale 構成を実行しているとき、環境はカタログ・サーバー、コンテナー・サーバー、および eXtreme Scale クライアント・プロセスで構成されています。これらのプロセスは手動で構成して開始する必要があります。

237 ページの『スタンドアロン環境でのカタログ・サービスの開始』
WebSphere Application Server が実行されていない分散 WebSphere eXtreme Scale 環境を使用している場合は、カタログ・サービスを手動で開始する必要があります。

240 ページの『コンテナー・プロセスの開始』
eXtreme Scale は、デプロイメント・トポロジー、または `server.properties` ファイルを使用して、コマンド行から開始できます。

スタンドアロン・モードでの TCP ポートの構成

カタログ・サービス・インスタンスをホストする Java 仮想マシンには 4 つのポートが必要です。内部使用のために 2 つのポートが使用され、クライアントおよびコンテナー断片が Internet Inter-ORB Protocol (IIOP) を使用して通信するために 3 番目のポートが使用され、 Java Management Extensions (JMX) の通信のために 4 番目のポートが使用されます。

このタスクについて

カタログ・サービス Java 仮想マシン (JVM) エンドポイント

eXtreme Scale は、主に Java 仮想マシン 間の通信のために IIOP を使用します。カタログ・サービス Java 仮想マシン は、IIOP サービスおよびグループ・サービス・ポートのためにポートの明示的構成を必要とする唯一の Java 仮想マシン です。内部ポートは **-catalogServiceEndPoints** コマンド行オプションを使用して指定されます。

```
-catalogServiceEndPoints <server:host:port:port,server:host:port:port>
```

-catalogServiceEndPoints コマンド行オプションで、サーバーあたり 2 つのポートを構成できます。 IIOP ポートは次のコマンド行オプションを使用して構成されます。

```
-listenerHost <host_name>
-listenerPort <port>
```

各カタログ・サービス JVM の始動時に、その JVM 用の単一リスナー・ポートとともに、すべて揃った カタログ・サービス・エンドポイントのセットを指定します。

コンテナー JVM エンドポイント

コンテナー Java 仮想マシン は 2 つのポートを使用します。 1 つのポートは内部使用で、もう 1 つのポートは IIOP 通信用です。コンテナー Java 仮想マシン は通

常、未使用ポートを自動的に検出し、次いで 2 つの動的に作成されたポートを使用するように自己構成します。この自動処理により構成は最小化されます。ただし、ファイアウォールが使用されていたり、明示的にポートを構成する場合は、コマンド行オプションにより、使用するオブジェクト・リクエスト・ブローカー (ORB) ポートを指定することができます。

```
-listenerHost <host_name>
-listenerPort <port>
```

これらのコマンド行オプションを使用して、ホスト名 (正しいネットワーク・カードにバインドするために重要) とその JVM 用に指定されるポートを指定できます。 **-haManagerPort** コマンド行引数を使用して内部ポートを指定することができます。ただし、最も単純な構成では、実行時にポートを選択させることができます。

1. hostA で最初のカタログ・サーバーを始動します。 コマンドの例は以下のとおりです。

   ```
   ./startOgServer.sh cs1 -listenerHost hostA -listenerPort 2809
   -catalogServiceEndPoints cs1:hostA:6601:6611,cs2:hostB:6601:6611
   ```

2. hostB で 2 番目のカタログ・サーバーを始動します。 コマンドの例は以下のとおりです。

   ```
   ./startOgServer.sh cs2 -listenerHost hostB -listenerPort 2809
   -catalogServiceEndPoints cs1:hostA:6601:6611,cs2:hostB:6601:6611
   ```

   クライアントは、カタログ・サービスのリスナー・エンドポイントを知ってさえいれば十分です。クライアントは、カタログ・サービスから自動的にコンテナー Java 仮想マシン のエンドポイント (すなわち、データを保管する Java 仮想マシン) を取得します。前述の例でカタログ・サービスに接続するには、クライアントは、以下の `host:port` のペアのリストを接続 API に渡す必要があります。

   ```
   hostA:2809,hostB:2809
   ```

   コンテナー JVM を始動して、カタログ・サービス例を使用するには、次のコマンドを使用します。

   ```
   ./startOgServer.sh c0 -catalogServiceEndPoints hostA:2809,hostB:2809
   ```

スタンドアロン eXtreme Scale サーバーの停止

`stopOgServer.sh` スクリプト (Unix の場合) または `stopOgServer.bat` スクリプト (Windows の場合) を使用してサーバー・プロセスを停止することができます。

- eXtreme Scale コンテナー・プロセスを停止します。
  1. コマンド行から、bin ディレクトリーに移動します。

     ```
     cd objectGridRoot/bin
     ```

  2. stopOgServer スクリプトを実行してサーバーを停止します。 次の例では、c0 サーバーを停止します。

     ```
     stopOgServer.sh c0 -catalogServiceEndPoints MyServer1.company.com:2809
     ```

重要: コンテナーでは、**-catalogServiceEndPoints** オプションを使用して、カタログ・サービス上のオブジェクト・リクエスト・ブローカー (ORB) のホストとポートを参照します。カタログ・サービスでは、**-listenerHost** および **-listenerPort** オプションを使用して ORB バインディング用のホストとポートを指定するか、またはデフォルトのバインディングを受け入れます。コンテナーを停止する場合は、**-catalogServiceEndPoints** オプションを使用して、カタログ・サービスで **-listenerHost** および **-listenerPort** に渡された値を参照します。カタログ・サービスの開始時に **-listenerHost** および **-listenerPort** オプションが使用されなかった場合、ORB はカタログ・サービスのために localhost のポート 2809 にバインドします。

カタログ・サービスで **-catalogServiceEndPoints** オプションに渡されたホストとポートを参照する場合に、**-catalogServiceEndPoints** オプションを使用しないでください。カタログ・サービスでは、**-catalogServiceEndPoints** オプションを使用して、静的サーバー構成に必要なポートが指定されます。

- eXtreme Scale カタログ・サービス・プロセスを停止します。
  1. コマンド行から、bin ディレクトリーに移動します。

     ```
     cd objectGridRoot/bin
     ```

  2. stopOgServer スクリプトを実行してサーバーを停止します。

     ```
     stopOgServer.sh catalogServer -bootstrap MyServer1.company.com:6601
     ```

  スクリプトに最初に渡される引数である catalogServer 引数を使用して、停止するプロセスを識別します。例えば、clientAccessPort 値が 6601 である MyServer1.company.com domain でカタログ・サービスが開始されたとします。
- サーバー停止プロセスのトレースを使用可能にします。 コンテナーを停止できない場合は、トレースを使用可能にして問題のデバッグに役立てることができます。サーバーの停止時にトレースを使用可能にするには、**-traceSpec** および **-traceFile** パラメーターを停止コマンドに追加します。 **-traceSpec** パラメーターは使用可能にするトレースのタイプを指定し、**-traceFile** パラメーターはそのトレース・データのために作成して使用するファイルのパスと名前を指定します。
  1. コマンド行から、bin ディレクトリーに移動します。

     ```
     cd objectGridRoot/bin
     ```

  2. トレースを使用可能にして stopOgServer スクリプトを実行します。

     ```
     stopOgServer.sh c4 -catalogServiceEndPoints MyServer1.company.com:2809
      -traceFile ../logs/c4Trace.log -traceSpec ObjectGrid=all=enabled
     ```

     トレースが取得されたら、ポート競合、欠落クラス、欠落または正しくない XML ファイルに関連したエラーがないか、またはスタック・トレースないか調べます。推奨される開始トレース仕様は、以下のとおりです。

     – ObjectGrid=all=enabled

     – ObjectGrid*=all=enabled

     すべてのトレース仕様オプションについては、 320 ページの『トレース・オプション』を参照してください。

関連資料

『stopOgServer スクリプト』
stopOgServer スクリプトはサーバーを停止します。

stopOgServer スクリプト

stopOgServer スクリプトはサーバーを停止します。

目的

stopOgServer スクリプトを使用してサーバーを停止することができます。

ロケーション

stopOgServer スクリプトは、ルート・ディレクトリーの bin ディレクトリーにあります。例えば、次のように入力します。

```
cd objectgridRoot/bin
```

使用法

カタログ・サーバーを停止する場合:

Windows

```
stopOgServer.bat <server> -catalogServiceEndPoints
<csHost:csListenerPort,csHost:csListenerPort> [options]
```

UNIX

```
stopOgServer.sh <server> -catalogServiceEndPoints
<csHost:csListenerPort,csHost:csListenerPort> [options]
```

ObjectGrid コンテナー・サーバーを停止する場合:

Windows

```
stopOgServer.bat <server> -catalogServiceEndPoints
<csHost:csListenerPort,csHost:csListenerPort> [options]
```

UNIX

```
startOgServer.sh -catalogServiceEndPoints
<csHost:csListenerPort,csHost:csListenerPort> [options]
```

オプション

-clientSecurityFile <security properties file>

-traceSpec <trace specification>
サーバーが始動したとき使用可能になるトレースの有効範囲を指定するストリングを指定します。

例:

- `ObjectGrid=all=enabled`
- `ObjectGrid*=all=enabled`

-traceFile <trace file>
トレース情報を保存するファイルのパスを指定します。

例: `../logs/c4Trace.log`

-jvmArgs <JVM arguments>

-jvmArgs オプションより後にあるパラメーターは、すべてサーバー Java 仮想マシン (JVM) を始動するために使用されるものです。 -jvmArgs オプションが使用される場合には、そのオプションが、指定された最後のオプション・スクリプト引数であることを確認してください。

関連タスク

249 ページの『スタンドアロン eXtreme Scale サーバーの停止』
`stopOgServer.sh` スクリプト (Unix の場合) または `stopOgServer.bat` スクリプト (Windows の場合) を使用してサーバー・プロセスを停止することができます。

WebSphere Application Server による WebSphere eXtreme Scale の管理

WebSphere Application Server でカタログ・サービスおよびコンテナー・サーバー・プロセスを実行できます。これらのサーバーを構成するプロセスは、スタンドアロン構成の場合とは異なります。カタログ・サービスは、WebSphere Application Server サーバーまたはデプロイメント・マネージャーで自動的に開始できます。eXtreme Scale アプリケーションが WebSphere Application Server 環境にデプロイされて、開始されるときに、コンテナー・プロセスは開始されます。

関連資料

203 ページの『サーバー・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、サーバーのさまざまな設定 (例えば、トレース設定、ロギング、およびセキュリティー構成など) を定義する複数のプロパティーが含まれます。サーバー・プロパティー・ファイルは、カタログ・サービスとコンテナー・サービスによって使用されます。

210 ページの『クライアント・プロパティー・ファイル』
eXtreme Scale クライアント・プロセスの要件に基づいて、プロパティー・ファイルを作成できます。

202 ページの『プロパティー・ファイルの解説』
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。

WebSphere Application Server 環境でのカタログ・サービス・プロセスの開始

WebSphere Application Server または WebSphere Application Server Network Deployment 環境では、eXtreme Scale カタログ・サービスを 1 つだけ自動的に開始することができます。カタログ・サービスを WebSphere セル内のどのプロセスでも実行するように構成することができます。開発環境には単一のクラスター化されていないカタログ・サービスが適しています。実稼働環境の場合は、カタログ・サービス・グリッドを使用する必要があります。

始める前に

WebSphere Application Server および WebSphere eXtreme Scale をインストールする必要があります。詳しくは、27 ページの『第 4 章 WebSphere eXtreme Scale のインストールとデプロイメント』を参照してください。

このタスクについて

カタログ・サービス・プロセスは WebSphere Application Server プロセス内で実行されます。カタログ・サービス・プロセスを含むプロセスが実行されると、カタログ・サービス・プロセスも開始されます。基本 WebSphere Application Server の場合、カタログ・サービスはサーバー・プロセスで実行されます。 WebSphere Application Server Network Deployment の場合は、カタログ・サービス・プロセスはデプロイメント・マネージャー内で自動的に実行されますが、ノード・エージェント・プロセスやアプリケーション・サーバー・プロセス内で実行されるように構成することもできます。

- **基本 WebSphere Application Server での単一カタログ・サーバーの始動**

  WebSphere eXtreme Scale が WebSphere Application Server にインストールされている場合は、カタログ・サービスはサーバー・プロセス内で自動的に開始されます。この機能により、カタログ・サービスを明示的に開始する必要がなくなるため、Rational® Application Developer などの開発環境での単体テストが簡略化されます。

  WebSphere eXtreme Scale が WebSphere Application Server Network Deployment にインストールされている場合は、カタログ・サービスはデプロイメント・マネージャー・プロセス内で自動的に開始されます。

- **WebSphere Application Server Network Deployment でのカタログ・サービス・グリッドの開始**

  eXtreme Scale カタログ・サーバー・グリッドは、WebSphere Application Server セル上で設定される catalog.services.cluster カスタム・プロパティーを使用して、デプロイメント・マネージャー、ノード・エージェント・プロセス、またはアプリケーション・サーバー・プロセスで明示的に定義することができます。カスタム・プロパティーの値の形式は *serverName*:*hostName*:*clientPort*:peerPort:*listenerPort* です。複数のサーバーを指定する場合は、値をコンマで区切ります。

  実稼働環境では、カタログ・サービスを eXtreme Scale コンテナーと連結しないようにしてください。カタログ・サービスを、複数のノード・エージェント・プロセスまたは eXtreme Scale アプリケーションをホスティングしていないアプリケーション・サーバー・グリッドに組み込んでください。いくつかの例外はありますが、catalog.services.cluster カスタム・プロパティーは、**-listenerPort** パラメーターを付けてスタンドアロン・カタログ・サーバーを始動する際に使用される **-catalogServiceEndPoints** パラメーターと類似しています。カスタム・プロパティー値の各属性の定義は次のとおりです。

  serverName
  : カタログ・サービスをホストするサーバーの WebSphere プロセス (cellName、nodeName、serverName など) の完全修飾名を指定します。

例: `cellA¥node1¥nodeagent`

hostName
ホスティング・サーバーの名前を指定します。

clientPort
ピア・カタログ・グリッド通信に使用されるポートを指定します。

peerPort
ピア・カタログ・グリッド通信に使用されるポートを指定します。

listenerPort
listenerPort は、WebSphere サーバー構成で定義される BOOTSTRAP_ADDRESS 値と一致しなければなりません。

管理コンソールでカスタム・プロパティーを作成するには、「**システム管理**」 → 「**セル**」 → 「**カスタム・プロパティー**」 → 「**新規**」をクリックします。定義されている属性を使用して、名前を `catalog.services.cluster` および適切な形式の値として指定します。有効な値の例を次に示します。

```
cellA¥node1¥nodeagent:host.local.domain:6600:6601:2809,cellA¥node2¥
nodeagent:host.foreign.domain:6600:6601:2809
```

カスタム・プロパティーを設定したならば、識別された各サーバーを再始動する必要があります。これらのサーバーを再始動すると、カタログ・サービス・グリッドが自動的に始動します。

制約事項: カタログ・グリッドは WebSphere Application Server のコア・グループ・メカニズムを使用します。その結果として、カタログ・グリッドのメンバーがコア・グループの境界にまたがらないため、カタログ・グリッドはセルにまたがらなくなります。ただし、eXtreme Scale は、セル境界を越えてカタログ・サーバー (スタンドアロン・カタログ・グリッドや別のセルに組み込まれたカタログ・グリッドなど) に接続することで、セルにまたがることがあります。

WebSphere Application Server 環境でのコンテナー・プロセスの開始

WebSphere Application Server 環境内のコンテナー・プロセスは、eXtreme Scale XML ファイルを組み込んだモジュールが開始されると自動的に開始されます。

始める前に

WebSphere Application Server および WebSphere eXtreme Scale をインストールする必要があります。さらに、WebSphere Application Server 管理コンソールにアクセスできなければなりません。

このタスクについて

Java Platform, Enterprise Edition アプリケーションのクラス・ローダー規則は複雑であるため、Java EE サーバー内で共用 WebSphere eXtreme Scale を使用しているときは、ロード元クラスが非常に複雑になります。 Java EE アプリケーションは通常、1 つ以上の Enterprise JavaBeans™ (EJB) または Web アーカイブ (WAR) モジュールがデプロイされる単一のエンタープライズ・アーカイブ (EAR) ファイルです。

WebSphere eXtreme Scale は各モジュールの開始を監視し、eXtreme Scale XML ファイルを検査します。 WebSphere eXtreme Scale は、XML ファイルによるモジュールの開始を検出すると、アプリケーション・サーバーを eXtreme Scale コンテナー Java 仮想マシン (JVM) としてカタログ・サービスに登録します。コンテナーをカタログ・サービスに登録することにより、さまざまなグリッドに同じアプリケーションをデプロイし、カタログ・サービスで引き続き単一グリッドとして処理することができます。カタログ・サービスは、セル、グリッド、または動的グリッドとの関連はありません。すべてが eXtreme Scale コンテナー JVM であるか、またはそうでないかのいずれかです。必要に応じて、単一の論理グリッドを複数の WebSphere Extended Deployment セルに分散させることができます。

1. META-INF フォルダーに eXtreme Scale XML ファイルが含まれるモジュールを持つように EAR ファイルをパッケージ化します。 WebSphere eXtreme Scale は、EJB および WEB モジュールが開始されると、これらのモジュールの META-INF フォルダーで `objectGrid.xml` および `objectGridDeployment.xml` ファイルの存在を検出します。 `objectGrid.xml` ファイルのみが検出されると、JVM はクライアントと見なされ、その他の場合は、この JVM が `objectGridDeployment.xml` ファイルで定義されているグリッドのコンテナーであると見なされます。

   これらの XML ファイルの正しい名前を使用しなければなりません。ファイル名には大/小文字の区別があります。これらのファイルが存在しないと、コンテナーは開始されません。断片が配置されることを示すメッセージは `systemout.log` ファイルで確認することができます。 eXtreme Scale を使用する EJB モジュールまたは WAR モジュールの `META-INF` ディレクトリーに eXtreme Scale XML ファイルがなければなりません。

   eXtreme Scale XML ファイルには以下のものがあります。

   - `objectGrid.xml` ファイル。通常は eXtreme Scale 記述子 XML ファイルと呼ばれているものです。
   - `objectGridDeployment.xml` ファイル。通常はデプロイメント記述子 XML ファイルと呼ばれているものです。
   - `entity.xml` ファイル。エンティティーが使用された場合 (オプション)。`entity.xml` ファイル名は、`objectGrid.xml` ファイルに指定されている名前と一致しなければなりません。

   eXtreme Scale ランタイムはこれらのファイルを検出し、カタログ・サービスに連絡して、別のコンテナーを使用してこの eXtreme Scale の断片をホストできることを通知します。

   ヒント: 複数のエンティティーが 1 つの eXtreme Scale サーバーを使用するアプリケーションを試みる場合は、デプロイメント記述子 XML ファイル内の minSyncReplicas 値を `0` に設定することで、「`CWPRJ1005E: 使用不可の EntityMetadata リポジトリーによって引き起こされたエンティティー関連例外を解決中にエラーが発生しました。タイムアウトのしきい値に達しました。`」メッセージを回避してください。
2. アプリケーションをデプロイして開始します。

   モジュールが開始されると、コンテナーが自動的に開始されます。カタログ・サービスが開始され、できるだけ速やかに区画のプライマリーおよびレプリカ (断

片) が配置されます。 objectGridDeployment.xml ファイルで numInitialContainers 属性を定義していない限り、この配置は直ちに行われます。numInitialContainers 属性を定義した場合は、指定した数のコンテナーが開始された時点で配置が開始されます。

次のタスク

コンテナーと同じセル内にあるアプリケーションは、ObjectGridManager.connect(null, null) メソッドを使用してこれらのグリッドに接続した後、getObjectGrid(ccc, ″object grid name″) メソッドを呼び出すことができます。 connect または getObjectGrid メソッドはコンテナーが断片の配置を完了するまでブロックされることがありますが、このブロックはグリッドが開始されるときだけ問題となります。

ClassLoader

eXtreme Scale に格納されたプラグインまたはオブジェクトは、特定のクラス・ローダーにロードされます。ロードされたオブジェクトは、同じ EAR 内の 2 つの EJB モジュールに含めることができます。これらのオブジェクトは同じですが、別の ClassLoader を使用してロードされています。アプリケーション A がサーバーに対してローカルな eXtreme Scale マップに Person オブジェクトを格納した場合、アプリケーション B がこのオブジェクトを読み取ろうとすると、ClassCastException を受け取ります。この例外は、アプリケーション B が Person オブジェクトを別のクラス・ローダーにロードしたために発生します。

この問題を解決する方法の 1 つは、eXtreme Scale に格納された必要なプラグインおよびオブジェクトをルート・モジュールが含むようにすることです。 eXtreme Scale を使用する各モジュールは、ルート・モジュール内でクラスを参照する必要があります。もう 1 つの解決方法は、これらの共用オブジェクトを、モジュールとアプリケーションの両方が共用する共通クラス・ローダー上のユーティリティー jar 内に配置することです。オブジェクトは、WebSphere クラスまたは lib/ext ディレクトリーにも配置することができますが、デプロイメントが複雑になるため、推奨しません。

EAR ファイル内の EJB モジュールは通常、同じ ClassLoader を共用するため、この問題の影響を受けません。各 WAR モジュールには独自の ClassLoader があり、この問題の影響を受けます。

クライアントとしてのみグリッドに接続

catalog.services.cluster プロパティーがセル、ノード、またはサーバー・カスタム・プロパティーで定義されている場合は、EAR ファイル内のすべてのモジュールが ObjectGridManager.connect (ServerFactory.getServerProperties() .getCatalogServiceBootstrap(), null, null) メソッドを呼び出して ClientClusterContext を取得し、さらに ObjectGridManager.getObjectGrid(ccc, ″grid name″) メソッドを呼び出して eXtreme Scale の参照を取得することができます。アプリケーション・オブジェクトがマップに格納されている場合は、それらのオブジェクトが共通の ClassLoader に存在することを確認してください。

Java Platform, Standard Edition クライアントまたはセル外のクライアントは、カタログ・サービス (WebSphere でのデフォルトはデプロイメント・マネージャー) の

ブートストラップ IIOP ポートを使用して接続して ClientClusterContext を取得し、次に通常の方法で、指定された eXtreme Scale を取得することができます。

エンティティー・マネージャー

エンティティー・マネージャーは役に立ちます。これは、タプルがアプリケーション・オブジェクトではなくマップに格納されていて、クラス・ローダー問題にあまり関係しないためです。ただし、プラグインが問題になることがあります。また、エンティティー (ObjectGridManager.connect("host:port[,host:port], null, objectGridOverride) または ObjectGridManager.connect(null, objectGridOverride)) が定義されている eXtreme Scale に接続する際はクライアント・オーバーライド eXtreme Scale 記述子 XML ファイルが常に必要となるので注意してください。

WebSphere Application Server 環境での TCP (Transmission Control Protocol) ポートの構成

カタログ・サービスは、デフォルトではデプロイメント・マネージャー内で単一インスタンスを実行し、デプロイメント・マネージャー Java 仮想マシンの Internet Inter-ORB Protocol (IIOP) ブートストラップ・ポートを使用します。

このタスクについて

セル内の Web アプリケーションまたは Enterprise JavaBeans (EJB) アプリケーションは、カタログ・サービス・ブートストラップ・ポートを指定するのではなく、`null,null` 接続呼び出しを使用して、同じセル内のグリッドに接続できます。

WebSphere Application Server のカタログ・サービス・グリッドがデプロイメント・マネージャーでホストされている場合、セル外部のクライアント (Java Platform, Standard Edition クライアントを含む) は、デプロイメント・マネージャー・ホスト名および IIOP ブートストラップ・ポートを使用してカタログ・サービスに接続する必要があります。

カタログ・サービスが WebSphere Application Server セルで実行され、クライアントがそのセル外で実行されているとき、`catalog.services.cluster` 名を使用してセルのカスタム・プロパティーを検索し、クライアント接続ポートを見つける必要があります。このエントリーがあった場合は、それを使用してクライアントをカタログ・サービスに接続しますが、ない場合は、デプロイメント・マネージャーの IIOP ポートをクライアント接続に使用します。クライアントが WebSphere Application Server セル内にある場合は、 CatalogServerProperties インターフェースから直接ポートを取得できます。

eXtreme Scale は、グループ・メンバーシップ用の HA マネージャーの分散および整合性サービス (DCS) ポートを再利用します。 Java Management Extensions (JMX) ポートも再利用されます。

HTTP セッション管理

WebSphere Application Server バージョン 5 以降の環境では、アプリケーション・サーバーにあるデフォルトのセッション・マネージャーを、WebSphere eXtreme Scale に同梱されている HTTP セッション・マネージャーでオーバーライドすることができます。

WebSphere eXtreme Scale HTTP セッション・マネージャーも WebSphere Application Server バージョン 6.0.2 以降で、または、WebSphere Application Server Community Edition や Apache Tomcat など、WebSphere Application Server を実行しない環境でも実行できます。

フィーチャー

セッション・マネージャーは、いずれの Java Platform, Enterprise Edition バージョン 1.4 コンテナーでも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 をサポートすることができます。

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管理機能を提供します。セッション・マネージャーは、そのアプリケーションに関連付けられた HTTP セッションを作成し、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 ObjectGrid インスタンス内にパーシストします。このインスタンスは、ローカルのメモリー内インスタンスか、あるいは完全に複製されたインスタンス、クラスター化されたインスタンス、および区画に分割されたインスタンスです。後者のトポロジーを使用すると、セッション・マネージャーが、アプリケーション・サーバーがシャットダウンまたは予期せずに終了した場合の HTTP セッション・フェイルオーバー・サポートを提供できます。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。

- 典型的マイグレーション・シナリオなど、さまざまなバージョンの WebSphere Application Server をアプリケーション・サーバーとして使用する環境。
- 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。

- サーバーのフェイルオーバー中、WebSphere Application Server のデフォルトのサービスの品質 (QoS) レベルよりも高い QoS レベルで、かつセッションの可用性もより強く保証された状態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。
- セッションの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ため、アフィニティー・メカニズムをそのロード・バランサー用にカスタマイズする必要がある環境。
- セッション管理のオーバーヘッド、および外部 Java プロセスに対するストレージの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。

セッション・マネージャーの動作

セッション・マネージャーは、それ自体をサーブレット・フィルターの形で要求パスに導入します。WebSphere eXtreme Scale に付属しているツールを使用して、このサーブレット・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。このフィルターは、ターゲット・アプリケーション内のサーブレットまたは JSP ファイルの前に要求を受け取ります。このときフィルターは、そのフィルターの実装環境で、HTTPRequest オブジェクトおよび HTTPResponse オブジェクトを代行受信し、ラッパー・オブジェクトを作成します。

このフィルターの実装環境は、HTTPRequest オブジェクトおよび HTTPResponse オブジェクトで作成された、HTTP セッションに関連するすべての呼び出しを代行受信します。これらの呼び出しは、セッション・マネージャーによって処理され、基礎となる Java Platform, Enterprise Edition サーバー実装環境内の基本セッション・マネージャーには渡されません。セッション・マネージャーは、セッションおよびそのセッションのライフサイクルを作成および管理します。これには、タイムアウト・ベースの無効化、セッション無効化のリスナーの始動、およびその他のライフサイクル・イベントが含まれます。

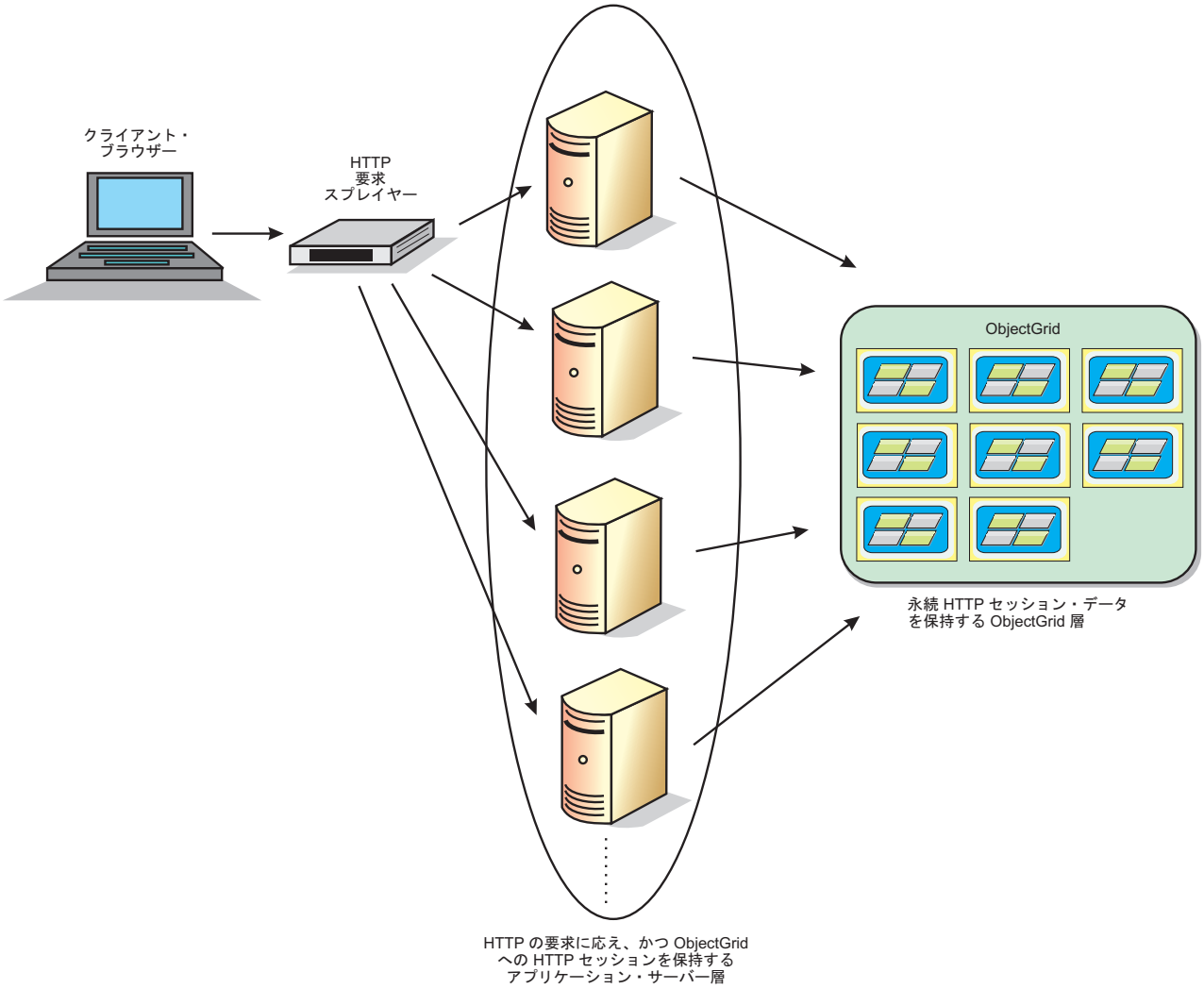


図 6. リモート・コンテナー構成を含む HTTP セッション管理トポロジー

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**

  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。

- **リモートの、ネットワーク接続 eXtreme Scale コンテナー**

  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 eXtreme Scale と通信します。

セッション・マネージャー・アプリケーションにおける eXtreme Scale セッションの取得

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

        HttpSession session = req.getSession(true);

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

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

getAttribute メソッド呼び出しから返されるセッションでマップが変更された場合、その変更は、基礎となるセッションがコミットされた時点でコミットされます。

関連タスク

『WebSphere Application Server と連動する WebSphere eXtreme Scale セッション・マネージャーの構成』
WebSphere Application Server はセッション管理機能を備えていますが、このサポートは極度の要求ロード下では拡張しません。 WebSphere eXtreme Scale は、Web コンテナー用のデフォルト・セッション・マネージャーをオーバーライドするセッション管理実装とバンドルされており、より良いスケーラビリティーとより強固な構成オプションを備えています。

267 ページの『WebSphere Application Server Community Edition と連動する HTTP セッション・マネージャーの構成』
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 Application Server Community Edition モジュラー・インフラストラクチャー (GBean) を使用すれば、WebSphere eXtreme Scale をセッション状態パーシスタンス・メカニズムとして組み込むことができます。

関連資料

264 ページの『サーブレット・コンテキスト初期化パラメーター』
以下に示すサーブレット・コンテキスト初期化パラメーターのリストは、スクリプトまたは ANT ベースの接続メソッドで必要なプロパティー・ファイルに指定できるものです。

WebSphere Application Server と連動する WebSphere eXtreme Scale セッション・マネージャーの構成

WebSphere Application Server はセッション管理機能を備えていますが、このサポートは極度の要求ロード下では拡張しません。 WebSphere eXtreme Scale は、Web コンテナー用のデフォルト・セッション・マネージャーをオーバーライドするセッション管理実装とバンドルされており、より良いスケーラビリティーとより強固な構成オプションを備えています。

このタスクについて

WebSphere eXtreme Scale HTTP セッション・マネージャーは、キャッシング用に、組み込みサーバーとリモート・サーバーの両方をサポートします。

組み込みシナリオ

組み込みシナリオでは、サーブレットが実行される同じプロセス内で WebSphere eXtreme Scale サーバーが相互に連結されています。セッション・マネージャーはローカルの ObjectGrid インスタンスと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。

WebSphere Application Server を使用している場合は、提供された objectgridRoot/session/samples/objectGrid.xml および objectgridRoot/session/samples/objectGridDeployment.xml ファイルを、ご使用の Web アーカイブ (WAR) ファイルの META-INF ディレクトリーに配置してください。アプリケーションが始動して、セッション・マネージャーと同じプロセス内の eXtreme Scale コンテナーを自動的に始動すると、eXtreme Scale がこれらのファイルを自動的に検出します。

使用する複製が同期複製か非同期複製かということ、および構成するレプリカの数に応じて、objectGridDeployment.xml ファイルを変更できます。

リモート・サーバー・シナリオ

リモート・サーバー・シナリオでは、サーブレットとは異なるプロセスで eXtreme Scale サーバーが実行されます。セッション・マネージャーはリモートの eXtreme Scale サーバーと通信します。リモートのネットワーク接続 eXtreme Scale サーバーを使用するためには、eXtreme Scale カタログ・サービス・クラスターのホスト名およびポート番号によってセッション・マネージャーを構成する必要があります。そうすると、セッション・マネージャーは、eXtreme Scale クライアント接続を使用して、カタログ・サーバーおよび eXtreme Scale サーバーと通信します。

eXtreme Scale サーバーを独立したスタンドアロン・プロセス内で始動する場合は、セッション・マネージャー・サンプル・ディレクトリーで提供される objectGridStandAlone.xml および objectGridDeploymentStandAlone.xml ファイルを使用して、eXtreme Scale コンテナーを始動してください。

1. **WebSphere eXtreme Scale のインストールなしで WebSphere Application Server または WebSphere Application Server Network Deployment を実行している場合:** eXtreme Scale ライブラリーをインストールします。

   objectgridRoot/session/lib/sessionobjectgrid.jar および objectgridRoot/lib/wsobjectgrid.jar ファイルを、新しい HTTP セッション管理対応アプリケーションをホスティングしているアプリケーション・サーバーのクラスパスにコピーします。これらの Java アーカイブ (JAR) ファイルを <WAS_HOME>/lib フォルダーに入れて、これらのクラスが可視になるようにサーバーを再始動します。
2. アプリケーションを接合することで、アプリケーションがセッション・マネージャーを使用できるようにします。 セッション・マネージャーを使用するためには、適切なフィルター宣言をアプリケーションの Web デプロイメント記述子に

追加する必要があります。さらに、セッション・マネージャー構成パラメーターが、デプロイメント記述子内のサーブレット・コンテキスト初期化パラメーターという形式でセッション・マネージャーに渡されます。この情報は、以下に示す 3 とおりの方法でアプリケーションに導入することができます。

- **addObjectGridFilter スクリプト**

  eXtreme Scale とともに提供されるコマンド行スクリプトを使用して、フィルター宣言と構成によってアプリケーションをサーブレット・コンテキスト初期化パラメーターの形式で接合します。このスクリプト (objectgridRoot/bin/addObjectGridFilter.sh または objectgridRoot/bin/addObjectGridFilter.bat) は、接合する必要があるアプリケーション (エンタープライズ・アーカイブ・ファイルへの絶対パス) および各種構成プロパティーを含むプロパティー・ファイルへの絶対パスという 2 つのパラメーターを取ります。このスクリプトの使用形式は以下の通りです。 Windows

  ```
  addObjectGridFilter.bat [EAR ファイルのロケーション]
  [プロパティー・ファイルのロケーション]
  ```

  UNIX

  ```
  addObjectGridFilter.sh [EAR ファイルのロケーション]
  [プロパティー・ファイルのロケーション]
  ```

  UNIX **Unix 上の WebSphere Application Server にインストールされている eXtreme Scale の使用例:**

  a. `cd objectgridRoot/optionalLibraries/ObjectGrid/session/bin`

  b. `addObjectGridFilter.sh /tmp/mySessionTest.ear wasRoot/optionalLibraries/ObjectGrid/session/samples/splicer.properties`

  UNIX **Unix 上のスタンドアロン・ディレクトリーにインストールされている eXtreme Scale の使用例:**

  a. `cd objectgridRoot/session/bin`

  b. `addObjectGridFilter.sh /tmp/mySessionTest.ear objectgridRoot/session/samples/splicer.properties`

  接合されるサーブレット・フィルターは構成値のデフォルトを保持します。これらのデフォルト値は、2 番目の引数にあるプロパティー・ファイルで指定する構成オプションでオーバーライドできます。使用できるパラメーターのリストについては、 264 ページの『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。

  eXtreme Scale インストールとともに提供されるサンプルの splicer.properties ファイルを変更して使用することができます。また、各サーブレットを拡張することによってセッション・マネージャーを挿入する、addObjectGridServlets スクリプトも使用できます。ただし、推奨スクリプトは addObjectGridFilter スクリプトです。

- **Ant ビルド・スクリプト**

  WebSphere eXtreme Scale には Apache Ant で使用できる build.xml ファイルが同梱されています。このファイルは WebSphere Application Server インス

トールの wasRoot/bin フォルダーに含まれています。 build.xml を修正してセッション・マネージャー構成プロパティーを変更することができます。構成プロパティーは splicer.properties ファイル内のプロパティー名と同一です。 build.xml が変更されたならば、ant.sh, ws_ant.sh (UNIX) または ant.bat, ws_ant.bat (Windows) を実行することによって Ant プロセスが起動されます。

- **手動による Web 記述子の更新**

  Web アプリケーションに同梱されている web.xml ファイルを編集して、フィルター宣言、そのサーブレット・マッピング、およびサーブレット・コンテキスト初期化パラメーターが組み込まれるようにします。この方法はエラーを起こしやすいため、使用しないようにしてください。

使用できるパラメーターのリストについては、『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。

3. アプリケーションをデプロイします。 サーバーやクラスターに対して通常使用する手順に従ってアプリケーションをデプロイしてください。アプリケーションをデプロイした後、アプリケーションを始動することができます。
4. アプリケーションにアクセスします。 これで、セッション・マネージャーおよび WebSphere eXtreme Scale と対話するアプリケーションにアクセスすることができます。

次のタスク

アプリケーションの装備時にセッション・マネージャーの構成属性の大多数を変更して、セッション・マネージャーを使用するようにすることができます。これらの属性には、複製タイプ (同期または非同期) のバリエーション、セッション ID の長さ、メモリー内セッション・テーブルのサイズなどがあります。アプリケーションの装備時に変更できる属性を別にすれば、アプリケーションのデプロイメント後に変更できるその他の構成属性は、WebSphere eXtreme Scale サーバー・クラスター・トポロジーと、それらのクラスターのクライアント (セッション・マネージャー) がそれらのクラスターに接続する方法に関係する属性のみです。

関連概念

258 ページの『HTTP セッション管理』
WebSphere Application Server バージョン 5 以降の環境では、アプリケーション・サーバーにあるデフォルトのセッション・マネージャーを、WebSphere eXtreme Scale に同梱されている HTTP セッション・マネージャーでオーバーライドすることができます。

関連資料

『サーブレット・コンテキスト初期化パラメーター』
以下に示すサーブレット・コンテキスト初期化パラメーターのリストは、スクリプトまたは ANT ベースの接続メソッドで必要なプロパティー・ファイルに指定できるものです。

サーブレット・コンテキスト初期化パラメーター

以下に示すサーブレット・コンテキスト初期化パラメーターのリストは、スクリプトまたは ANT ベースの接続メソッドで必要なプロパティー・ファイルに指定できるものです。

パラメーター

affinityManager

HTTP セッション・ルーティング・アフィニティーを容易にするための Servlet フィルター・プラグインの完全修飾パッケージおよび Java クラス名を指定します。 WebSphere Application Server の場合は、アフィニティー・サポートが組み込まれているため、この値は無視されます。次のいずれかの値を指定してください。

- **アフィニティーなし (デフォルト):** com.ibm.ws.httpsession.NoAffinityManager
- **ベンダー・アフィニティー・メカニズム:** com.ibm.ws.httpsession.AssumeAffinityManager
- **新規アフィニティー・マネージャーの構成:** com.ibm.wsspi.session.ISessionAffinityManager インターフェースを使用してください。

persistenceMechanism

セッションを eXtreme Scale 内に保管する方法を定義するストリング値を指定します。次のいずれかの値を指定してください。

- **ObjectGridStore**: 各セッション属性は異なるエントリーとして eXtreme Scale テーブルに保管されます。
- **ObjectGridAtomicSessionStore**: セッション全体が単一のエントリーとして eXtreme Scale テーブルに保管されます。

objectGridName

特定の Web アプリケーションに使用されるテーブルの名前を定義するストリング値を指定します。デフォルト名は、ServletContext 値から派生した Web アプリケーション名から取られます。このパラメーターを設定すると、その値がオーバーライドされます。

catalogHostPort

カタログ・サーバー接続情報を指定します。値の形式は `host:port<,host:port>` でなければなりません。このリストは任意の長さにすることができ、最初の実行可能なアドレスが使用されます。このプロパティーは、リモートのネットワーク接続 eXtreme Scale シナリオの場合にのみ必要です。

replicationType

セッションに対する更新を eXtreme Scale に書き込む方法を定義するストリング値。有効な値は、`asynchronous` と `synchronous` です。デフォルトは `asynchronous` で、アフィニティーが使用される場合にのみ適用可能です。その他の場合は、synchronous のみがサポートされます。

replicationInterval

replicationType パラメーター値が `asynchronous` であるとき、更新されたセッションを eXtreme Scale に書き込む間隔を秒数で定義する整数値。デフォルト値は 10 秒です。

sessionTableSize

eXtreme Scale に保管するほか、Servlet フィルターとともにメモリーに保存するセッションの数を定義する整数値。デフォルトは 1000 です。

defaultSessionTimeout

セッションが無効化されてシステムから除去される前に、セッションを非アクテ

ィブ (例えば、サーブレットからアクセスされないように) にできる時間を分数で定義する整数値。デフォルトは 30 分です。

sessionIDLength
HTTP セッション用に作成されるストリング ID の長さを定義する整数値。デフォルトは 23 です。

shareSessionsAcrossWebApps
true または false のいずれかのストリング値を指定します。デフォルトは false です。サーブレット仕様では、HTTP セッションを Web アプリケーション間で共用することはできません。この共用を可能にするため、サーブレット仕様の拡張が提供されます。

cookieName
この Web アプリケーションの Cookie の名前を定義するストリング値を指定します。デフォルトは JSESSIONID です。固有の Cookie 名を使用する場合は、このプロパティーを `splicer.properties` ファイルに追加してください。

関連概念

258 ページの『HTTP セッション管理』
WebSphere Application Server バージョン 5 以降の環境では、アプリケーション・サーバーにあるデフォルトのセッション・マネージャーを、WebSphere eXtreme Scale に同梱されている HTTP セッション・マネージャーでオーバーライドすることができます。

関連タスク

261 ページの『WebSphere Application Server と連動する WebSphere eXtreme Scale セッション・マネージャーの構成』
WebSphere Application Server はセッション管理機能を備えていますが、このサポートは極度の要求ロード下では拡張しません。 WebSphere eXtreme Scale は、Web コンテナー用のデフォルト・セッション・マネージャーをオーバーライドするセッション管理実装とバンドルされており、より良いスケーラビリティーとより強固な構成オプションを備えています。

267 ページの『WebSphere Application Server Community Edition と連動する HTTP セッション・マネージャーの構成』
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 Application Server Community Edition モジュラー・インフラストラクチャー (GBean) を使用すれば、WebSphere eXtreme Scale をセッション状態パーシスタンス・メカニズムとして組み込むことができます。

WebSphere eXtreme Scale を使用した SIP セッション管理

Session Initiation Protocol (SIP) セッション複製用のデータ複製サービス (DRS) の代わりに、WebSphere eXtreme Scale を、信頼できる SIP 複製メカニズムとして使用できます。

SIP セッション管理の構成

WebSphere eXtreme Scale を SIP 複製メカニズムとして使用するには、com.ibm.sip.ha.replicator.type カスタム・プロパティーを設定します。このカスタム・プロパティーを追加する各サーバーごとに、管理コンソールで、「**アプリケーション・サーバー**」 → ***my_application_server*** → 「**SIP コンテナー**」 → 「**カスタム・プロパティー**」を選択します。「名前」には `com.ibm.sip.ha.replicator.type` と入力し、「値」には `OBJECTGRID` と入力します。

以下のプロパティーを使用して、SIP セッションの保管に使用する ObjectGrid の振る舞いをカスタマイズします。このカスタム・プロパティーを追加する各サーバーごとに、管理コンソールで、「**アプリケーション・サーバー**」 → ***my_application_server*** → 「**SIP コンテナー**」 → 「**カスタム・プロパティー**」をクリックします。「**名前**」および「**値**」を入力します。各サーバーは、機能のプロパティーに設定されているものと同じプロパティーを所有する必要があります。

表 8. ObjectGrid を使用した SIP セッション管理のためのカスタム・プロパティー

property	値	デフォルト
com.ibm.sip.ha.replicator.type	`OBJECTGRID`: SIP セッション・ストアとして ObjectGrid を使用	
min.synchronous.replicas	同期複製の最小数	0
max.synchronous.replicas	同期複製の最大数	0
max.asynchronous.replicas	非同期複製の最大数	1
auto.replace.lost.shards	詳しくは、 153 ページの『デプロイメント・トポロジー構成』を参照してください。	true
development.mode	• `true` - プライマリーと同じノード上で複製をアクティブにできる • `false` - 複製はプライマリーと異なるノード上になければならない	false

WebSphere Application Server Community Edition と連動する HTTP セッション・マネージャーの構成

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 Application Server Community Edition モジュラー・インフラストラクチャー (GBean) を使用すれば、WebSphere eXtreme Scale をセッション状態パーシスタンス・メカニズムとして組み込むことができます。

始める前に

Geronimo または WebSphere Application Server Community Edition および WebSphere eXtreme Scale を、ご使用のシステムに unzip またはインストールする必要があります。

このタスクについて

WebSphere Application Server Community Edition のバックボーン (カーネル) は、GBean という外部モジュールのもとで機能を提供します。

1. GBean を配布します。
   a. *extremeScale_root*/wasce/bin ディレクトリーにナビゲートします。
   b. 稼働中のシステムに対して addToCeRepository スクリプトを実行します。
      - Linux UNIX addToRepository.sh ceRoot
      - Windows addToRepository.bat ceRoot
2. 構成ファイルを編集します。 GBean が配布されたので、WebSphere Application Server Community Edition を、サーバー始動時に起動されることを認識するように構成します。 GBean を登録して、実行時に config.xml ファイルと一緒に使用できる構成可能引数を GBean に指定することができます。
   a. WasCeRoot/var/config ディレクトリーに移動します。
   b. プレーン・テキスト・エディターで config.xml ファイルを開きます。
   c. config.xml ファイルで次の行を検索します。

      ```
      <module name="org.apache.geronimo.plugins/mconsole-ds/2.1.1/car"/>
       </attributes>
      ```

   d. 次の抜粋を </attributes> タグより前に付加します。

      ```
      <module name="com.ibm.websphere.objectgrid/gbean/1.0/car">
         <gbean name="objectgrid/BringupPlugin">
         <attribute name="objectgridHome">c:¥objectgrid</attribute>
         <attribute name="geronimoHome">C:¥websphereCE</attribute>
         <attribute name="serverName">server1</attribute>
         <attribute name="catalogServiceEndPoints">host:port</attribute>
         <attribute name="replicationDisabled">false</attribute>
         <attribute name="traceSpecification">*=all=disabled</attribute>
         </gbean>
      </module>
      ```

      上記の抜粋中にある GBean パラメーターの値は例です。これらの値を構成できます。

      objectgridHome
      objectgridRoot のインストール・ディレクトリーを指定します。

      geronimoHome -
      WebSphere Application Server Community Edition または Apache Geronimo のインストール・ディレクトリーを指定します。

      serverName
      サーバー名を指定します。この名前は、すべての WebSphere Application Server Community Edition および eXtreme Scale サーバー・インスタンスに固有な名前でなければなりません。

      catalogServiceEndPoints -
      最後のカタログ・サーバーのホストおよびポートを指定します。

      replicationDisabled -
      セッション状態の複製が使用可能になっているかどうかを指定します。

traceSpecification -

Java 仮想マシン (JVM) 内の eXtreme Scale コンテナーのトレース用のストリングを指定します。

3. カタログ・サーバーを始動します。 WebSphere Application Server Community Edition イメージでは、その JVM 内の eXtreme Scale サーバーをサーバー始動時に正常に起動できます。ただし、eXtreme Scale をブートストラップするには、まずカタログ・サーバーを始動する必要があります。カタログ・サーバーは、WebSphere Application Server トポロジー内のデプロイメント・マネージャーに類似しており、WebSphere Application Server Community Edition が起動する前に始動する必要があります。カタログ・サーバーが始動されたならば、GBean エントリー内の `config.xml` ファイルにあるカタログ・サーバーのホスト名とポートを入力します。詳しくは、237 ページの『スタンドアロン環境でのカタログ・サービスの開始』を参照してください。特定のホスト名およびポートへの ORB ポートのバインドに関するセクションを必ずお読みください。 GBean 構成内のエンドポイントを指定する必要があります。サーバーの始動時にポートの競合があった場合は、`ceRoot/var/config/config-substitutions.properties` ファイルを開いて、NamingPort キーを `1099` 以外の値に変更する必要があります。
4. WebSphere Application Server Community Edition を始動してブートストラップをテストします。
   a. `WasCeRoot/bin` ディレクトリーに移動します。
   b. JAVA_HOME 環境変数を設定します。
      - UNIX Linux `export JAVA_HOME=javaHome`
      - Windows `set JAVA_HOME=javaHome`
   c. 始動コマンドを実行します。
      - UNIX Linux `geronimo.sh run`
      - Windows `geronimo.bat run`

   そうすると、サーバーが始動し、サーバーのコンポーネントまたは GBean の初期化を開始するはずです。 eXtreme Scale プラグインは、最後にロードするプラグインであり、必要な開始トレースおよび情報ステートメントを出力します。

関連概念

258 ページの『HTTP セッション管理』
WebSphere Application Server バージョン 5 以降の環境では、アプリケーション・サーバーにあるデフォルトのセッション・マネージャーを、WebSphere eXtreme Scale に同梱されている HTTP セッション・マネージャーでオーバーライドすることができます。

関連資料

264 ページの『サーブレット・コンテキスト初期化パラメーター』
以下に示すサーブレット・コンテキスト初期化パラメーターのリストは、スクリプトまたは ANT ベースの接続メソッドで必要なプロパティー・ファイルに指定できるものです。

WebSphere Application Server Community Edition におけるセッション状態の複製のための WebSphere eXtreme Scale の使用

WebSphere Application Server Community Edition にデプロイされるすべての Web アプリケーションは、Java Platform, Enterprise Edition 仕様および `geronimo-web.xml` 記述子ファイルに準拠していることが必要です。このファイルには、WebSphere Application Server Community Edition のランタイム環境に固有のアクセス情報と従属情報が含まれています。 Java EE Web アプリケーションで WebSphere eXtreme Scale のセッション複製フィーチャーを使用できるようにするには、 Web アプリケーションは、セッション固有属性として eXtreme Scale プラグインに関するデータを提供する必要があります。

始める前に

このプロセスは、IDE またはユーティリティーを使用せずにどのオペレーティング・システムでも作成できるほどシンプルです。

このタスクについて

後で eXtreme Scale 内でセッション・パーシスタンスを使用する基本 Web アプリケーションをセットアップできます。これは、ユーザーが特定サーブレットにアクセスした回数をカウントして保管します。

1. サンプル Web アプリケーションの作成
   a. ファイル・システム内の任意のロケーションにディレクトリーを 1 つ作成します。例えば、*app_home* という名前のディレクトリーを作成するとします。
   b. *app_home* ディレクトリーにナビゲートして、以下のディレクトリー構造を作成します。

      ```
      > app_home
      --> META-INF
      --> WEB-INF
      -----> lib
      -----> classes
      ```

   c. *app_home*/*WEB-INF* ディレクトリーで、 `web.xml` ファイルを作成します。以下のコードをコピーして `web.xml` ファイルに貼り付けます。 すべてのコン

テキスト・パラメーターが web.xml ファイルに追加されていることを確認してください。そうでないとフィルターが正しく機能しない可能性があります。

```
<?xml version="1.0" encoding="UTF-8"?>
<web-app id="sample" version="2.4" xmlns="http://java.sun.com/xml/ns/j2ee"
  xmlns:xsi="http://www.w3.org/2001/XMLSchema-instance"
  xsi:schemaLocation="http://java.sun.com/xml/ns/j2ee
  http://java.sun.com/xml/ns/j2ee/web-app_2_4.xsd">
 <display-name>
 sample</display-name>
 <context-param>
    <param-name>shareSessionsAcrossWebApps</param-name>
    <param-value>false</param-value>
  </context-param>
  <context-param>
      <param-name>sessionIDLength</param-name>
      <param-value>23</param-value>
  </context-param>
  <context-param>
      <param-name>sessionTableSize</param-name>
      <param-value>1000</param-value>
  </context-param>
  <context-param>
      <param-name>replicationInterval</param-name>
      <param-value>10</param-value>
  </context-param>
  <context-param>
      <param-name>replicationType</param-name>
      <param-value>synchronous</param-value>
  </context-param>
  <context-param>
      <param-name>defaultSessionTimeout</param-name>
      <param-value>30</param-value>
  </context-param>
  <context-param>
      <param-name>affinityManager</param-name>
      <param-value>com.ibm.ws.httpsession.NoAffinityManager</param-value>
  </context-param>
  <context-param>
      <param-name>useURLEncoding</param-name>
      <param-value>true</param-value>
  </context-param>
  <context-param>
      <param-name>objectGridName</param-name>
      <param-value>session</param-value>
  </context-param>
  <context-param>
      <param-name>persistenceMechanism</param-name>
      <param-value>ObjectGridStore</param-value>
  </context-param>
  <filter>
      <filter-name>HttpSessionFilter</filter-name>
      <filter-class>com.ibm.ws.httpsession.HttpSessionFilter</filter-class>
  </filter>
  <filter-mapping>
      <filter-name>HttpSessionFilter</filter-name>
      <url-pattern>/*</url-pattern>
  </filter-mapping>
 <servlet>
  <description>
  </description>
  <display-name>
  SampleServlet</display-name>
  <servlet-name>SampleServlet</servlet-name>
  <servlet-class>
  SampleServlet</servlet-class>
 </servlet>
 <servlet-mapping>
  <servlet-name>SampleServlet</servlet-name>
  <url-pattern>/SampleServlet</url-pattern>
 </servlet-mapping>
 <welcome-file-list>
  <welcome-file>index.html</welcome-file>
  <welcome-file>index.htm</welcome-file>
  <welcome-file>index.jsp</welcome-file>
  <welcome-file>default.html</welcome-file>
  <welcome-file>default.htm</welcome-file>
  <welcome-file>default.jsp</welcome-file>
 </welcome-file-list>
</web-app>
```

d. *app_home*/*WEB-INF* ディレクトリーで、 geronimo-web.xml ファイルを作成します。 以下のコードをコピーして、geronimo-web.xml ファイルに貼り付けます。

```
<?xml version="1.0" encoding="UTF-8"?>
<web-app xmlns="http://geronimo.apache.org/xml/ns/j2ee/web-1.1"
 xmlns:nam="http://geronimo.apache.org/xml/ns/naming-1.1"
 xmlns:sec="http://geronimo.apache.org/xml/ns/security-1.1"
 xmlns:sys="http://geronimo.apache.org/xml/ns/deployment-1.1">
 <sys:environment>
  <sys:moduleId>
   <sys:groupId>com.ibm.websphere.objectgrid</sys:groupId>
   <sys:artifactId>sample</sys:artifactId>
   <sys:version>1.0</sys:version>
   <sys:type>war</sys:type>
  </sys:moduleId>
  <sys:dependencies>
   <sys:dependency>
    <sys:groupId>com.ibm.websphere.objectgrid
    </sys:groupId>
    <sys:artifactId>session</sys:artifactId>
    <sys:version>1.0</sys:version>
    <sys:type>jar</sys:type>
   </sys:dependency>
  </sys:dependencies>
 </sys:environment>
 <context-root>/sample</context-root>
</web-app>
```

e. 付属の SampleServlet.class を *WEB-INF*/*classes* ディレクトリーの下に配置します。

f. *app_home* ディレクトリーにナビゲートします。

g. 次のコマンドを実行して、Web アーカイブ (WAR) ファイルをパッケージします。

```
jar -cf sample.war
```

h. これで Web アプリケーションをデプロイする準備ができました。

2. Web アプリケーションをデプロイします。

a. 指定された hostname:port で WebSphere Application Server Community Edition ホーム・ページにログインします。 デフォルトで、このホーム・ページは http://localhost:8080/ です。

b. 「**管理コンソール**」をクリックします。

c. ユーザー名とパスワードを入力して、「**OK**」をクリックします。 デフォルトで、クレデンシャルは次のようになります。

- ユーザー名: System
- パスワード: Manager

d. 左側のメニューから「**アプリケーション**」 → 「**新規デプロイ**」を選択します。

e. アーカイブの右にある「**参照**」をクリックして、前のセクションで作成した sample.war ファイルを選択します。

f. この計画は Web アプリケーションにバンドルしているため、「**計画**」フィールドは空白にしたままにします。

g. 「**インストール後にアプリケーションを開始**」を選択してから、「**インストール**」をクリックします。

アプリケーションがインストールされると、インストールが正常終了したことを伝えるメッセージが表示されます。

3. アプリケーションをテストします。

a. /sample/SampleServlet コンテキスト・ルートにナビゲートして、アプリケーションをテストします。デフォルトで、コンテキスト・ルートは、http://localhost:8080/sample/SampleServlet です。
b. HttpSession 実装の前にある、このサーブレットにアクセスした回数を確認します。 eXtreme Scale パーシスタンス層を使用している場合は、HttpSessionImpl を読み取り、Web コンテナーのデフォルトのセッション・ストレージ・メカニズムを使用している場合は、 StandardSessionFacade を読み取ります。

第 8 章 デプロイメント環境のモニター

API、MBean、ログ、およびユーティリティーを使用して、アプリケーション環境のパフォーマンスをモニターできます。

関連概念

『統計の概説』
WebSphere eXtreme Scale での統計は、内部統計ツリーから作成されます。内部ツリーからは、StatsAccessor API、Performance Monitoring Infrastructure (PMI) モジュール、および MBean API が作成されます。

303 ページの『ベンダー・ツール』
一般的によく使われるいくつかのエンタープライズ・モニタリング・ソリューションを使用して、WebSphere eXtreme Scale をモニターすることができます。 パブリックにアクセス可能な管理 Bean を使用して WebSphere eXtreme Scale をモニターする IBM Tivoli Monitoring および Hyperic HQ 用に、プラグイン・エージェントが組み込まれています。CA Wily Introscope は Java メソッドのインスツルメンテーションを使用して、統計情報を収集します。

関連資料

298 ページの『Managed Bean (MBean) を使用した環境の管理』
デプロイメントを管理およびモニターするには、さまざまなタイプの Java Management Extensions (JMX) MBeans を使用できます。各 MBean は、マップ、eXtreme Scale、サーバー、複製グループ、または複製グループ・メンバーなどの特定のエンティティーを参照します。

統計の概説

WebSphere eXtreme Scale での統計は、内部統計ツリーから作成されます。内部ツリーからは、StatsAccessor API、Performance Monitoring Infrastructure (PMI) モジュール、および MBean API が作成されます。

次の図は、eXtreme Scale の統計の一般的なセットアップを示しています。

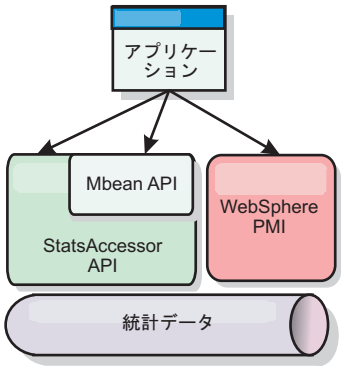


図 7. 統計の概説

これらの API のそれぞれは統計ツリーに対するビューを提供しますが、使用される理由は異なります。

- **Statistics API**: Statistics API は、内部統計ツリー内のデータの汎用ルックアップ・メカニズムとして使用します。クライアントまたは組み込みアプリケーションに Statistics API を使用することができます。分散 eXtreme Scale の場合には、Statistics API を使用することはできません。
- **MBean API**: MBean API は、モニター用の仕様ベースのメカニズムです。MBean API は、Statistics API を使用し、サーバー Java 仮想マシン (JVM) に対してローカルに実行します。API および MBean の構造は、他のベンダーのユーティリティーと容易に統合できるように設計されています。分散 eXtreme Scale を実行中の場合は、MBean API を使用します。
- **WebSphere Application Server Performance Monitoring Infrastructure (PMI) モジュール** : PMI は、WebSphere Application Server 内で WebSphere eXtreme Scale を実行中の場合に使用します。これらのモジュールは、内部統計ツリーのビューを提供します。

Statistics API

ツリー・マップに非常によく似ており、特定のモジュールを取得するための対応するパスおよびキー、すなわちこの場合は細分度または集約レベルがあります。例えば、ツリー内に常に任意のルート・ノードがあって、「accounting」という名前の ObjectGrid に属している「payroll」という名前のマップに関して統計が収集されると想定します。例えば、マップの集約レベルまたは細分度についてモジュールにアクセスするには、パスの String[] を渡すことができます。この場合、各 String がノードのパスを表わすので、これは String[] {root, ″accounting″, ″payroll″} と同等です。この構造の利点は、ユーザーがパス内のどのノードにも配列を指定でき、そのノードの集約レベルを取得できるという点です。このため、String[] {root, ″accounting″} を渡すと、マップ統計が取得されますが、これは「accounting」というグリッド全体に対するものです。これにより、ユーザーは、モニターする統計のタイプを、アプリケーションに必要などのような集約レベルでも指定できます。

WebSphere Application Server PMI モジュール

WebSphere eXtreme Scale には、WebSphere Application Server PMI で使用するための統計モジュールが組み込まれています。WebSphere Application Server プロファイルが WebSphere eXtreme Scale で拡張されると、拡張スクリプトにより WebSphere eXtreme Scale モジュールが自動的に WebSphere Application Server 構成ファイルに統合されます。 PMI を使用すると、統計モジュールを使用可能および使用不可にしたり、さまざまな細分度で統計を自動的に集約したり、また組み込み Tivoli Performance Viewer を使用してデータをグラフ化することさえできます。詳しくは、283 ページの『WebSphere Application Server PMI によるパフォーマンスのモニター』を参照してください。

ベンダー製品と Managed Bean (MBean) との統合

eXtreme Scale API および Managed Bean は、サード・パーティーのモニタリング・アプリケーションと簡単に統合できるように設計されています。eXtreme Scale トポロジーに関する情報を分析するために使用できる単純な Java Management Extensions (JMX) コンソールの例のいくつかとして、JConsole や MC4J がありま

す。またプログラマチック API を使用して、eXtreme Scale パフォーマンスのスナップショットを作成するか、そのパフォーマンスを追跡するアダプター実装を作成することもできます。WebSphere eXtreme Scale には、すぐに使用可能なモニター機能を持つサンプルのモニター・アプリケーションが含まれており、拡張したカスタム・モニター・ユーティリティーを作成するためのテンプレートとしてこれを使用できます。

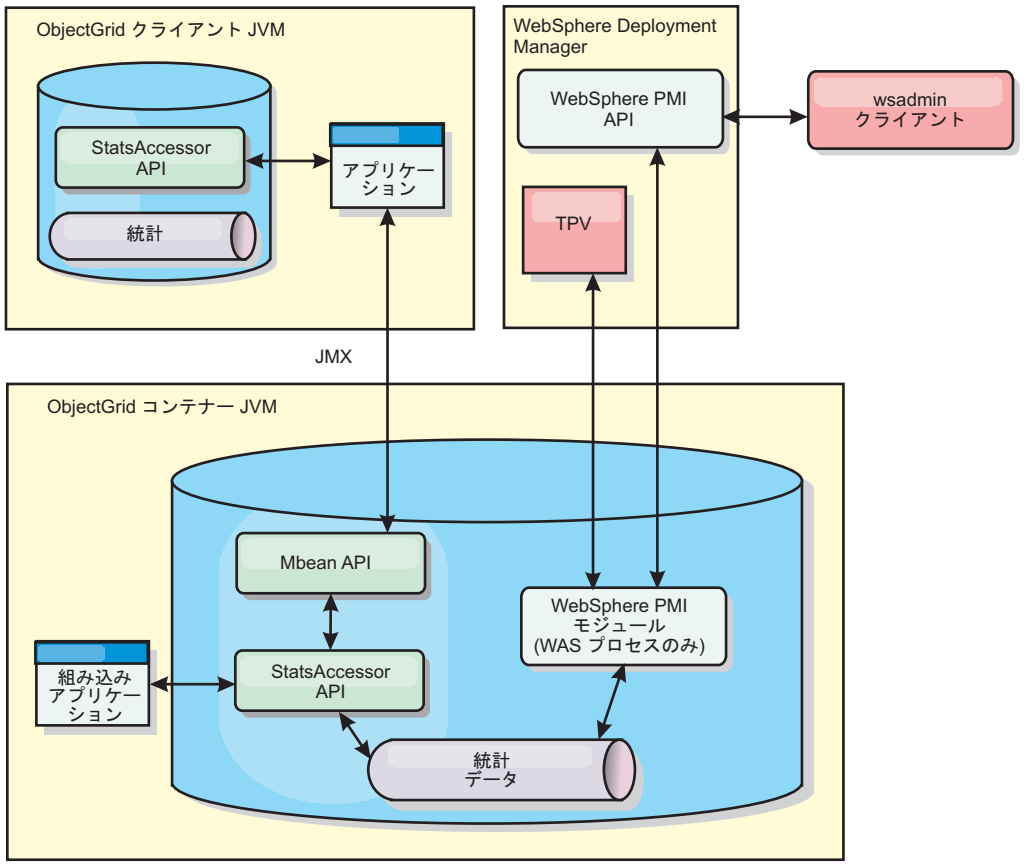


図 8. MBean の概説

詳しくは、 300 ページの『xsAdmin サンプル・ユーティリティーの使用』を参照してください。特定のベンダー・アプリケーションとの統合について詳しくは、以下のトピックを参照してください。

- IBM Tivoli モニター・エージェントでの eXtreme Scale のモニター
- 315 ページの『Hyperic HQ による eXtreme Scale のモニター』
- 311 ページの『CA Wily Introscope による eXtreme Scale アプリケーションのモニター』

関連タスク

275 ページの『第 8 章 デプロイメント環境のモニター』
API、MBean、ログ、およびユーティリティーを使用して、アプリケーション環境のパフォーマンスをモニターできます。

279 ページの『統計 API によるモニター』
統計 API は、内部統計ツリーに直接接続するインターフェースです。統計はデフォルトでは使用不可になっていますが、StatsSpec インターフェースを設定することで使用可能にすることができます。 StatsSpec インターフェースは、WebSphere eXtreme Scale がどのように統計をモニターするかを定義します。

283 ページの『WebSphere Application Server PMI によるパフォーマンスのモニター』
WebSphere eXtreme Scale は、WebSphere Application Server または WebSphere Extended Deployment アプリケーション・サーバーで実行されているとき、Performance Monitoring Infrastructure (PMI) をサポートします。 PMI は、ランタイム・アプリケーションでパフォーマンス・データを収集し、パフォーマンス・データをモニターするための外部アプリケーションをサポートするインターフェースを提供します。管理コンソールまたは wsadmin ツールを使用して、モニター・データにアクセスすることができます。

285 ページの『PMI の使用可能化』
WebSphere Application Server Performance Monitoring Infrastructure (PMI) を使用して、任意のレベルで統計を使用可能または使用不可にすることができます。例えば、特定のマップのマップ・ヒット率統計を使用可能にするが、エントリー数統計またはローダー・バッチ更新時間統計は使用可能にしないことを選択できます。管理コンソール内またはスクリプトを使用して PMI を使用可能にすることができます。

288 ページの『PMI 統計の取得』
PMI 統計を取得することによって、eXtreme Scale アプリケーションのパフォーマンスを確認できます。

300 ページの『xsAdmin サンプル・ユーティリティーの使用』
xsAdmin サンプル・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。

315 ページの『Hyperic HQ による eXtreme Scale のモニター』
Hyperic HQ は、サード・パーティーのモニター・ソリューションで、オープン・ソース・ソリューションあるいはエンタープライズ製品として自由に使用可能です。WebSphere eXtreme Scale には、あるプラグインが含まれていて、このプラグインにより Hyperic HQ エージェントは eXtreme Scale コンテナー・サーバーを検出し、eXtreme Scale 管理 Bean を使用して統計のレポートおよび集約を行うことができます。 Hyperic HQ を使用すると、スタンドアロン eXtreme Scale デプロイメントをモニターできます。

304 ページの『IBM Tivoli Enterprise Monitoring Agent for WebSphere eXtreme Scale のモニター』
IBM Tivoli Enterprise Monitoring Agent は、分散環境およびホスト環境のデータベース、オペレーティング・システム、およびサーバーをモニターするために使用で

きる機能の豊富なモニタリング・ソリューションです。 WebSphere eXtreme Scale には、eXtreme Scale 管理 Bean をイントロスペクトする場合に使用できるカスタマイズ・エージェントが含まれています。このソリューションは、スタンドアロン eXtreme Scale および WebSphere Application Server デプロイメントの両方で効果的に機能します。

関連資料

298 ページの『Managed Bean (MBean) を使用した環境の管理』
デプロイメントを管理およびモニターするには、さまざまなタイプの Java Management Extensions (JMX) MBeans を使用できます。各 MBean は、マップ、eXtreme Scale、サーバー、複製グループ、または複製グループ・メンバーなどの特定のエンティティーを参照します。

289 ページの『PMI モジュール』
Performance Monitoring Infrastructure (PMI) モジュールを使用してアプリケーションのパフォーマンスをモニターすることができます。

289 ページの『PMI モジュール』
Performance Monitoring Infrastructure (PMI) モジュールを使用してアプリケーションのパフォーマンスをモニターすることができます。

統計 API によるモニター

統計 API は、内部統計ツリーに直接接続するインターフェースです。統計はデフォルトでは使用不可になっていますが、StatsSpec インターフェースを設定することで使用可能にすることができます。 StatsSpec インターフェースは、WebSphere eXtreme Scale がどのように統計をモニターするかを定義します。

このタスクについて

ローカルの StatsAccessor API を使用して、実行中のコードと同じ Java 仮想マシン (JVM) にある ObjectGrid インスタンス上のデータおよびアクセス統計を照会することができます。個々のインターフェースについて詳しくは、API 資料を参照してください。次の手順で、内部統計ツリーのモニターを使用可能にします。

1. StatsAccessor オブジェクトを検索します。 StatsAccessor インターフェースは singleton パターンに従います。したがって、クラス・ローダーに関連する問題を別にすれば、JVM ごとに 1 つの StatsAccessor インスタンスが存在するはずです。このクラスはすべてのローカル統計操作のメイン・インターフェースとして機能します。以下のコードは、accessor クラスの検索方法の例です。この操作は、他のすべての ObjectGrid 呼び出しより前に呼び出します。

```
public class LocalClient {

   public static void main(String[] args) {

      // retrieve a handle to the StatsAccessor
      StatsAccessor accessor = StatsAccessorFactory.getStatsAccessor();

   }

}
```

2. グリッド StatsSpec インターフェースを設定します。 すべての統計を ObjectGrid レベルでのみ収集するように、この JVM を設定します。トランザクションを開始する前に、必要と思われるすべての統計をアプリケーションが使用可能にするようにする必要があります。次の例は、static 定数フィールドと spec ストリングの両方を使用して StatsSpec インターフェースを設定するものです。

static 定数フィールドは既に仕様が定義されているため、このフィールドを使用する方が簡単です。ただし、spec ストリングを使用すれば、必要な統計のどんな組み合わせでも使用可能にすることができます。

```
public static void main(String[] args) {

      // retrieve a handle to the StatsAccessor
      StatsAccessor accessor = StatsAccessorFactory.getStatsAccessor();

      // Set the spec via the static field
      StatsSpec spec = new StatsSpec(StatsSpec.OG_ALL);
      accessor.setStatsSpec(spec);

      // Set the spec via the spec String
      StatsSpec spec = new StatsSpec("og.all=enabled");
      accessor.setStatsSpec(spec);

}
```

3. トランザクションをグリッドに送信して、モニター用のデータが収集されるようにします。 統計用に有効なデータを収集するには、トランザクションをグリッドに送る必要があります。次のコード抜粋は、ObjectGridA 内の MapA にレコードを挿入するものです。統計は、ObjectGrid レベルであるため、ObjectGrid 内のマップはいずれも同じ結果を示します。

   ```
   public static void main(String[] args) {

         // retrieve a handle to the StatsAccessor
         StatsAccessor accessor = StatsAccessorFactory.getStatsAccessor();

         // Set the spec via the static field
         StatsSpec spec = new StatsSpec(StatsSpec.OG_ALL);
         accessor.setStatsSpec(spec);

         ObjectGridManager manager =
       ObjectGridmanagerFactory.getObjectGridManager();
         ObjectGrid grid = manager.getObjectGrid("ObjectGridA");
         Session session = grid.getSession();
         Map map = session.getMap("MapA");

         // Drive insert
         session.begin();
         map.insert("SomeKey", "SomeValue");
         session.commit();
   }
   ```

4. StatsAccessor API を使用して StatsFact を照会します。 すべての統計パスは StatsFact インターフェースに関連付けられます。 StatsFact インターフェースは、StatsModule オブジェクトを編成して組み込むために使用される汎用プレースホルダーです。実際の統計モジュールにアクセスするためには、前もって StatsFact オブジェクトを検索する必要があります。

```
public static void main(String[] args) {

      // retrieve a handle to the StatsAccessor
      StatsAccessor accessor = StatsAccessorFactory.getStatsAccessor();

      // Set the spec via the static field
      StatsSpec spec = new StatsSpec(StatsSpec.OG_ALL);
      accessor.setStatsSpec(spec);

      ObjectGridManager manager =
    ObjectGridManagerFactory.getObjectGridManager();
      ObjectGrid grid = manager.getObjectGrid("ObjectGridA");
      Session session = grid.getSession();
      Map map = session.getMap("MapA");

      // Drive insert
      session.begin();
      map.insert("SomeKey", "SomeValue");
      session.commit();

      // Retrieve StatsFact
```

```
      StatsFact fact = accessor.getStatsFact(new String[] {"EmployeeGrid"},
    StatsModule.MODULE_TYPE_OBJECT_GRID);

}
```

5. StatsModule オブジェクトと対話します。 StatsModule オブジェクトは StatsFact インターフェース内に含まれています。 StatsFact インターフェースを使用してモジュールへの参照を取得できます。 StatsFact インターフェースは汎用インターフェースであるため、戻されたモジュールを予期された StatsModule タイプにキャストする必要があります。このタスクは eXtreme Scale の統計を収集するため、戻された StatsModule オブジェクトは OGStatsModule タイプにキャストされます。モジュールがキャストされたならば、使用可能なすべての統計にアクセスすることができます。

```
public static void main(String[] args) {

      // retrieve a handle to the StatsAccessor
      StatsAccessor accessor = StatsAccessorFactory.getStatsAccessor();

      // Set the spec via the static field
      StatsSpec spec = new StatsSpec(StatsSpec.OG_ALL);
      accessor.setStatsSpec(spec);

      ObjectGridManager manager =
    ObjectGridmanagerFactory.getObjectGridManager();
      ObjectGrid grid = manager.getObjectGrid("ObjectGridA");
      Session session = grid.getSession();
      Map map = session.getMap("MapA");

      // Drive insert
      session.begin();
      map.insert("SomeKey", "SomeValue");
      session.commit();

      // Retrieve StatsFact
      StatsFact fact = accessor.getStatsFact(new String[] {"EmployeeGrid"},
    StatsModule.MODULE_TYPE_OBJECT_GRID);

      // Retrieve module and time
      OGStatsModule module = (OGStatsModule)fact.getStatsModule();
      ActiveTimeStatistic timeStat =
    module.getTransactionTime("Default", true);
      double time = timeStat.getMeanTime();

}
```

関連概念

275 ページの『統計の概説』
WebSphere eXtreme Scale での統計は、内部統計ツリーから作成されます。内部ツリーからは、StatsAccessor API、Performance Monitoring Infrastructure (PMI) モジュール、および MBean API が作成されます。

303 ページの『ベンダー・ツール』
一般的によく使われるいくつかのエンタープライズ・モニタリング・ソリューションを使用して、WebSphere eXtreme Scale をモニターすることができます。 パブリックにアクセス可能な管理 Bean を使用して WebSphere eXtreme Scale をモニターする IBM Tivoli Monitoring および Hyperic HQ 用に、プラグイン・エージェントが組み込まれています。CA Wily Introscope は Java メソッドのインスツルメンテーションを使用して、統計情報を収集します。

『統計モジュール』
WebSphere eXtreme Scale は、内部統計モデルを使用して、データの追跡およびフィルター処理を行います。このモデルは、すべてのデータ・ビューで統計のスナップショットを収集するために使用される基礎となる構造です。統計モジュールから情報を取得するには、いくつかの方法を使用できます。

関連資料

298 ページの『Managed Bean (MBean) を使用した環境の管理』
デプロイメントを管理およびモニターするには、さまざまなタイプの Java Management Extensions (JMX) MBeans を使用できます。各 MBean は、マップ、eXtreme Scale、サーバー、複製グループ、または複製グループ・メンバーなどの特定のエンティティーを参照します。

統計モジュール

WebSphere eXtreme Scale は、内部統計モデルを使用して、データの追跡およびフィルター処理を行います。このモデルは、すべてのデータ・ビューで統計のスナップショットを収集するために使用される基礎となる構造です。統計モジュールから情報を取得するには、いくつかの方法を使用できます。

概説

WebSphere eXtreme Scale での統計は、StatsModules コンポーネント内で追跡され、収容されます。統計モデルには、以下のいくつかのタイプの統計モジュールが存在します。

OGStatsModule
トランザクション応答時間など、ObjectGrid インスタンスの統計を提供します。

MapStatsModule
エントリー数やヒット率など、単一マップの統計を提供します。

QueryStatsModule
計画作成や実行時間など、照会の統計を提供します。

AgentStatsModule
シリアライズ時間や実行時間など、DataGrid API エージェントの統計を提供します。

HashIndexStatsModule
HashIndex 照会および保守の実行時間の統計を提供します。

SessionStatsModule
HTTP セッション・マネージャー・プラグインの統計を提供します。

統計モジュールについて詳しくは、API 資料中の com.ibm.websphere.objectgrid.stats パッケージを参照してください。

ローカル環境での統計

モデルは、前のリストで説明したすべての StatsModule タイプから構成される n 進ツリー (すべてのノードについて同じ次数を持つツリー構造) に似た編成になっています。この編成構造のため、ツリー内のすべてのノードは、StatsFact インターフェースで表現されます。 StatsFact インターフェースは、集約の目的で個別のモジュールまたはモジュールのグループを表わすことができます。例えば、ツリー内のいくつかのリーフ・ノードが特定の MapStatsModule オブジェクトを表わす場合、これらのノードの親 StatsFact ノードには、すべての子モジュールついて集約された統計が含まれます。StatsFact オブジェクトのフェッチ後には、インターフェースを使用して対応する StatsModule を取得することができます。

ツリー・マップに非常によく似ており、対応するパスまたはキーを使用して特定の StatsFact を取得することができます。パスは、要求されたファクトへのパスに沿ったすべてのノードから構成される String[] 値です。例えば、MapA と MapB という 2 つのマップを含む ObjectGridA という名前の ObjectGrid を作成したとします。MapA の StatsModule へのパスは、`[ObjectGridA, MapA]` のようになります。両方のマップの集約統計へのパスは、`[ObjectGridA]` となります。

分散環境での統計

分散環境では、統計モジュールは異なるパスを使用して取得されます。サーバーには複数の区画を入れることができるため、統計ツリーは、各モジュールが属する区画を追跡する必要があります。結果として、特定の StatsFact オブジェクトをルックアップするためのパスは異なります。前の例を使用しますが、マップが区画 1 に存在するという点を付け加えると、MapA の StatsFact オブジェクトを取得するためのパスは `[1,ObjectGridA, MapA]` となります。

関連タスク

279 ページの『統計 API によるモニター』
統計 API は、内部統計ツリーに直接接続するインターフェースです。統計はデフォルトでは使用不可になっていますが、StatsSpec インターフェースを設定することで使用可能にすることができます。 StatsSpec インターフェースは、WebSphere eXtreme Scale がどのように統計をモニターするかを定義します。

WebSphere Application Server PMI によるパフォーマンスのモニター

WebSphere eXtreme Scale は、WebSphere Application Server または WebSphere Extended Deployment アプリケーション・サーバーで実行されているとき、Performance Monitoring Infrastructure (PMI) をサポートします。 PMI は、ランタイム・アプリケーションでパフォーマンス・データを収集し、パフォーマンス・デー

タをモニターするための外部アプリケーションをサポートするインターフェースを提供します。管理コンソールまたは wsadmin ツールを使用して、モニター・データにアクセスすることができます。

始める前に

- WebSphere eXtreme Scale を WebSphere Application Server と組み合わせて使用しているとき、PMI を使用してご使用の環境をモニターすることができます。

このタスクについて

WebSphere eXtreme Scale は、WebSphere Application Server のカスタム PMI 機能を使用して、独自の PMI 装備を追加します。この方法で、管理コンソールまたは wsadmin ツールの Java Management Extensions (JMX) インターフェースを使用して、WebSphere eXtreme Scale PMI を使用可能および使用不可にすることができます。さらに、標準 PMI、および Tivoli Performance Viewer を含むモニター・ツールによって使用される JMX インターフェースを使用して WebSphere eXtreme Scale 統計にアクセスすることができます。

1. eXtreme Scale PMI を使用可能にします。 PMI 統計を表示するには、PMI を使用可能にする必要があります。詳しくは、285 ページの『PMI の使用可能化』を参照してください。
2. eXtreme Scale PMI 統計を取得します。 Tivoli Performance Viewer を使用して、eXtreme Scale アプリケーションのパフォーマンスを表示します。詳しくは、288 ページの『PMI 統計の取得』を参照してください。

次のタスク

wsadmin ツールについて詳しくは、297 ページの『wsadmin ユーティリティーの使用』を参照してください。

関連概念

275 ページの『統計の概説』
WebSphere eXtreme Scale での統計は、内部統計ツリーから作成されます。内部ツリーからは、StatsAccessor API、Performance Monitoring Infrastructure (PMI) モジュール、および MBean API が作成されます。

303 ページの『ベンダー・ツール』
一般的によく使われるいくつかのエンタープライズ・モニタリング・ソリューションを使用して、WebSphere eXtreme Scale をモニターすることができます。 パブリックにアクセス可能な管理 Bean を使用して WebSphere eXtreme Scale をモニターする IBM Tivoli Monitoring および Hyperic HQ 用に、プラグイン・エージェントが組み込まれています。CA Wily Introscope は Java メソッドのインスツルメンテーションを使用して、統計情報を収集します。

関連資料

298 ページの『Managed Bean (MBean) を使用した環境の管理』
デプロイメントを管理およびモニターするには、さまざまなタイプの Java Management Extensions (JMX) MBeans を使用できます。各 MBean は、マップ、eXtreme Scale、サーバー、複製グループ、または複製グループ・メンバーなどの特定のエンティティーを参照します。

289 ページの『PMI モジュール』
Performance Monitoring Infrastructure (PMI) モジュールを使用してアプリケーションのパフォーマンスをモニターすることができます。

PMI の使用可能化

WebSphere Application Server Performance Monitoring Infrastructure (PMI) を使用して、任意のレベルで統計を使用可能または使用不可にすることができます。例えば、特定のマップのマップ・ヒット率統計を使用可能にするが、エントリー数統計またはローダー・バッチ更新時間統計は使用可能にしないことを選択できます。管理コンソール内またはスクリプトを使用して PMI を使用可能にすることができます。

始める前に

アプリケーション・サーバーを始動し、eXtreme Scale 対応アプリケーションがインストールされている必要があります。また、スクリプトを使用して PMI を使用可能にするには、wsadmin ツールにログインして使用できなければなりません。
wsadmin ツールについて詳しくは、WebSphere Application Server インフォメーション・センターの wsadmin ツールのトピックを参照してください。

このタスクについて

WebSphere Application Server PMI を使用して、任意のレベルで統計を使用可能または使用不可にできる細かいメカニズムを提供します。例えば、特定のマップのマップ・ヒット率統計を使用可能にするが、エントリー数統計またはローダー・バッチ更新時間統計は使用可能にしないことを選択できます。このセクションでは、管理コンソールおよび wsadmin スクリプトを使用して ObjectGrid PMI を使用可能にする方法を示します。

- **管理コンソールで PMI を使用可能にします。**

1. 管理コンソールで、「**モニターおよびチューニング**」 → 「**Performance Monitoring Infrastructure**」 → 「***server_name***」をクリックします。
2. 「Performance Monitoring Infrastructure (PMI) を使用可能にする」が選択されていることを確認します。この設定は、デフォルトで使用可能になっています。この設定が使用可能になっていない場合は、チェック・ボックスを選択して、サーバーを再始動します。
3. 「**カスタム**」をクリックします。構成ツリーで、ObjectGrid および ObjectGrid マップ・モジュールを選択します。各モジュールの統計を使用可能にします。

ObjectGrid 統計のトランザクション・タイプ・カテゴリーが実行時に作成されます。「**Runtime**」タブには、ObjectGrid と Map 統計のサブカテゴリーのみを表示できます。

- **スクリプトを使用して PMI を使用可能にします。**
  1. コマンド行プロンプトを開きます。install_root/bin ディレクトリーへナビゲートします。wsadmin と入力して、wsadmin コマンド行ツールを開始します。
  2. eXtreme Scale PMI ランタイム構成を変更します。 以下のコマンドを使用して、サーバーに対して PMI が使用可能になっていることを確認します。

```
wsadmin>set s1 [$AdminConfig getid /Cell:CELL_NAME/Node:NODE_NAME/Server:APPLICATION_SERVER_NAME/]
wsadmin>set pmi [$AdminConfig list PMIService $s1]
wsadmin>$AdminConfig show $pmi.
```

PMI が使用可能になっていない場合は、以下のコマンドを実行して、PMI を使用可能にします。

```
wsadmin>$AdminConfig modify $pmi {{enable true}}
wsadmin>$AdminConfig save
```

PMI を使用可能にする必要がある場合は、サーバーを再始動します。

3. 以下のコマンドを使用して、統計セットをカスタム・セットに変更するための変数を設定します。

```
wsadmin>set perfName [$AdminControl completeObjectName type=Perf,
process=APPLICATION_SERVER_NAME,*]
wsadmin>set perfOName [$AdminControl makeObjectName $perfName]
wsadmin>set params [java::new {java.lang.Object[]} 1]
wsadmin>$params set 0 [java::new java.lang.String custom]
wsadmin>set sigs [java::new {java.lang.String[]} 1]
wsadmin>$sigs set 0 java.lang.String
```

4. 以下のコマンドを使用して、統計セットをカスタムに設定します。

```
wsadmin>$AdminControl invoke_jmx $perfOName setStatisticSet $params $sigs
```

5. 以下のコマンドを使用して、objectGridModule PMI 統計を使用可能にするための変数を設定します。

```
wsadmin>set params [java::new {java.lang.Object[]} 2]
wsadmin>$params set 0 [java::new java.lang.String objectGridModule=1]
wsadmin>$params set 1 [java::new java.lang.Boolean false]
wsadmin>set sigs [java::new {java.lang.String[]} 2]
wsadmin>$sigs set 0 java.lang.String
wsadmin>$sigs set 1 java.lang.Boolean
```

6. 以下のコマンドを使用して、統計ストリングを設定します。

```
wsadmin>set params2 [java::new {java.lang.Object[]} 2]
wsadmin>$params2 set 0 [java::new java.lang.String mapModule=*]
wsadmin>$params2 set 1 [java::new java.lang.Boolean false]
wsadmin>set sigs2 [java::new {java.lang.String[]} 2]
wsadmin>$sigs2 set 0 java.lang.String
wsadmin>$sigs2 set 1 java.lang.Boolean
```

7. 以下のコマンドを使用して、統計ストリングを設定します。

```
wsadmin>$AdminControl invoke_jmx $perfOName setCustomSetString $params2 $sigs2
```

これらのステップにより、eXtreme Scale ランタイム PMI は使用可能になりますが、PMI 構成は変更されません。アプリケーション・サーバーを再始動すると、メイン PMI の使用可能化を除いて、PMI 設定は失われます。

例

以下のステップを実行して、サンプル・アプリケーションの PMI 統計を使用可能にすることができます。

1. `http://host:port/ObjectGridSample` Web アドレスを使用してアプリケーションを立ち上げます。ここで、host および port は、サンプルをインストールするサーバーのホスト名および HTTP ポート番号です。
2. サンプル・アプリケーションで ObjectGridCreationServlet をクリックし、次にアクション・ボタン 1、2、3、4、および 5 をクリックして、ObjectGrid およびマップに対するアクションを生成します。この時点では、このサーブレット・ページを閉じないでください。
3. 管理コンソールで、「**モニターおよびチューニング**」 → 「**Performance Monitoring Infrastructure**」 → ***server_name*** をクリックします。「**ランタイム**」タブをクリックします。
4. 「**カスタム**」ラジオ・ボタンをクリックします。
5. ランタイム・ツリーで「ObjectGrid Maps」モジュールを展開し、「clusterObjectGrid」リンクをクリックします。「ObjectGrid マップ」グループの下に、clusterObjectGrid という名前の 1 つの ObjectGrid インスタンスが存在し、この clusterObjectGrid グループの下に、counters、employees、offices、および sites という 4 つのマップが存在します。ObjectGrids インスタンスには clusterObjectGrid インスタンスが存在し、そのインスタンスの下には、DEFAULT という名前のトランザクション・タイプがあります。
6. 興味のある統計を使用可能にすることができます。例えば、従業員マップのマップ・エントリー数、および DEFAULT トランザクション・タイプのトランザクション応答時間を使用可能にできます。

次のタスク

PMI が使用可能になった後、管理コンソールまたはスクリプトを介して PMI 統計を表示することができます。

関連概念

275 ページの『統計の概説』
WebSphere eXtreme Scale での統計は、内部統計ツリーから作成されます。内部ツリーからは、StatsAccessor API、Performance Monitoring Infrastructure (PMI) モジュール、および MBean API が作成されます。

関連資料

289 ページの『PMI モジュール』
Performance Monitoring Infrastructure (PMI) モジュールを使用してアプリケーションのパフォーマンスをモニターすることができます。

PMI 統計の取得

PMI 統計を取得することによって、eXtreme Scale アプリケーションのパフォーマンスを確認できます。

始める前に

- ご使用の環境の PMI 統計追跡を使用可能にします。詳しくは、285 ページの『PMI の使用可能化』を参照してください。
- このタスクにあるパスはサンプル・アプリケーションの統計を取得することを前提としたものですが、これらの統計は類似のステップを含む他のアプリケーションに対しても使用できます。
- 管理コンソールを使用して統計を取得する場合は、管理コンソールにログインできなければなりません。スクリプトを使用する場合は、wsadmin にログインできなければなりません。

このタスクについて

管理コンソールまたはスクリプトでステップを完了すれば、PMI 統計を取得して Tivoli Performance Viewer で表示することができます。

- 管理コンソールのステップ
- スクリプトのステップ

取得できる統計についての詳細は、289 ページの『PMI モジュール』を参照してください。

- 管理コンソールで PMI 統計を取得します。
  1. 管理コンソールで、「**モニターおよびチューニング**」 → 「**パフォーマンス・ビューアー**」 → 「**現行アクティビティー**」をクリックします。
  2. Tivoli Performance Viewer を使用してモニターするサーバーを選択してから、モニターを使用可能にします。
  3. サーバーをクリックして、「Performance viewer」ページを表示します。
  4. 構成ツリーを展開します。「**ObjectGrid マップ**」 → 「**clusterObjectGrid**」をクリックし、「**従業員**」を選択します。「**ObjectGrids**」 → 「**clusterObjectGrid**」を展開し、「**DEFAULT**」を選択します。
  5. ObjectGrid サンプル・アプリケーションで、ObjectGridCreationServlet サーブレットに移動し、ボタン 1 をクリックしてから、マップを取り込みます。ビューアーに統計が表示されます。
- スクリプトを使用して PMI 統計を取得します。

1. コマンド行プロンプトで、install_root/bin ディレクトリーに移動します。wsadmin と入力して wsadmin ツールを開始します。
2. 以下のコマンドを使用して、環境の変数を設定します。

```
wsadmin>set perfName [$AdminControl completeObjectName type=Perf,*]
wsadmin>set perfOName [$AdminControl makeObjectName $perfName]
wsadmin>set mySrvName [$AdminControl completeObjectName type=Server,
 name=APPLICATION_SERVER_NAME,*]
```

3. 以下のコマンドを使用して、mapModule 統計を取得するための変数を設定します。

```
wsadmin>set params [java::new {java.lang.Object[]} 3]
wsadmin>$params set 0 [$AdminControl makeObjectName $mySrvName]
wsadmin>$params set 1 [java::new java.lang.String mapModule]
wsadmin>$params set 2 [java::new java.lang.Boolean true]
wsadmin>set sigs [java::new {java.lang.String[]} 3]
wsadmin>$sigs set 0 javax.management.ObjectName
wsadmin>$sigs set 1 java.lang.String
wsadmin>$sigs set 2 java.lang.Boolean
```

4. 以下のコマンドを使用して、mapModule 統計を取得します。

```
wsadmin>$AdminControl invoke_jmx $perfOName getStatsString $params $sigs
```

5. 以下のコマンドを使用して、objectGridModule 統計を取得するための変数を設定します。

```
wsadmin>set params2 [java::new {java.lang.Object[]} 3]
wsadmin>$params2 set 0 [$AdminControl makeObjectName $mySrvName]
wsadmin>$params2 set 1 [java::new java.lang.String objectGridModule]
wsadmin>$params2 set 2 [java::new java.lang.Boolean true]
wsadmin>set sigs2 [java::new {java.lang.String[]} 3]
wsadmin>$sigs2 set 0 javax.management.ObjectName
wsadmin>$sigs2 set 1 java.lang.String
wsadmin>$sigs2 set 2 java.lang.Boolean
```

6. 以下のコマンドを使用して、objectGridModule 統計を取得します。

```
wsadmin>$AdminControl invoke_jmx $perfOName getStatsString $params2 $sigs2
```

タスクの結果

Tivoli Performance Viewer で統計を表示することができます。

関連概念

275 ページの『統計の概説』

WebSphere eXtreme Scale での統計は、内部統計ツリーから作成されます。内部ツリーからは、StatsAccessor API、Performance Monitoring Infrastructure (PMI) モジュール、および MBean API が作成されます。

関連資料

『PMI モジュール』

Performance Monitoring Infrastructure (PMI) モジュールを使用してアプリケーションのパフォーマンスをモニターすることができます。

PMI モジュール

Performance Monitoring Infrastructure (PMI) モジュールを使用してアプリケーションのパフォーマンスをモニターすることができます。

objectGridModule

objectGridModule は時間統計 (トランザクション応答時間) を含みます。トランザクションは、Session.begin メソッド呼び出しと Session.commit メソッド呼び出しの間の所要時間として定義されます。この所要時間は、トランザクション応答時間として追跡されます。objectGridModule のルート・エレメント (root) は、WebSphere eXtreme Scale 統計への入り口点として機能します。このルート・エレメントは ObjectGrid を子エレメントとして持ち、さらにこの子エレメントはトランザクション・タイプを子エレメントとして持ちます。応答時間統計はそれぞれのトランザクション・タイプと関連しています。

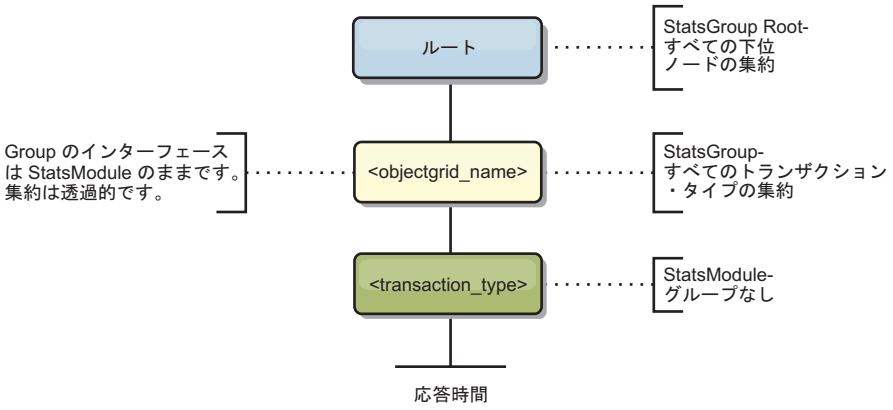


図 9. ObjectGridModule モジュールの構造

次の図は ObjectGridModule 構造の例です。この例では、2 つの ObjectGrid インスタンス (ObjectGrid A と ObjectGrid B) がシステムに存在します。ObjectGrid A インスタンスには 2 つのトランザクション・タイプ (A とデフォルト) があります。ObjectGrid B インスタンスにはデフォルトのトランザクション・タイプのみがあります。

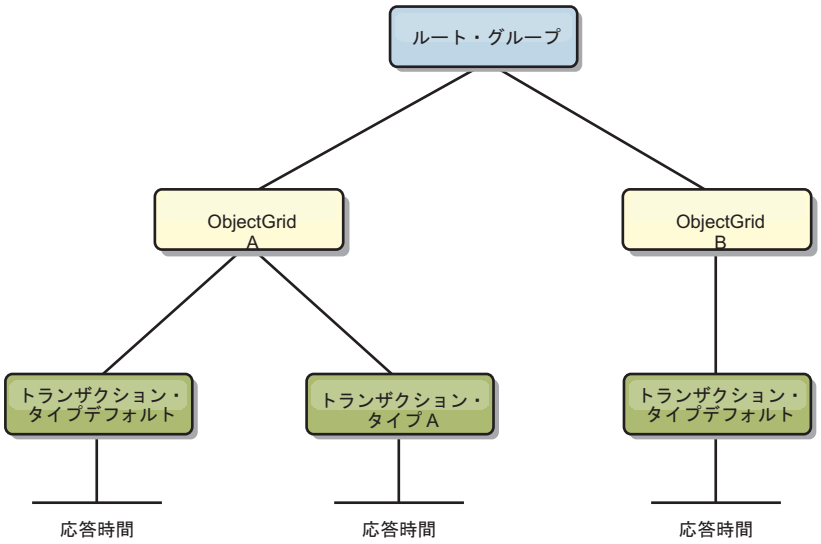


図 10. ObjectGridModule モジュール構造の例

アプリケーション開発者は、アプリケーションがどのタイプのトランザクションを使用するのかを知っているため、トランザクション・タイプはアプリケーション開発者によって定義されます。トランザクション・タイプの設定は、次の Session.setTransactionType(String) メソッドを使用して行われます。

```
/**
* Sets the transaction type for future transactions.
*
* After this method is called, all of the future transactions have the
* same type until another transaction type is set. If  no transaction
* type is set, the default  TRANSACTION_TYPE_DEFAULT transaction type
* is used.
*
* Transaction types are used mainly for statistical data tracking purpose.
* Users can predefine types of transactions that run in an
* application. The idea is to categorize transactions with the same characteristics
* to one category (type), so one transaction response time statistic can be
* used to track each transaction type.
*
* This tracking is useful when your application has different types of
* transactions.
* Among them, some types of transactions, such as update transactions, process
* longer than other transactions, such as read-only transactions. By using the
* transaction type, different transactions are tracked by different statistics,
* so the statistics can be more useful.
*
* @param tranType the transaction type for future transactions.
*/
void setTransactionType(String tranType);
```

次の例は、 updatePrice へのトランザクション・タイプを設定します。

```
// Set the transaction type to updatePrice
// The time between session.begin() and session.commit() will be
// tracked in the time statistic for "updatePrice".
session.setTransactionType("updatePrice");
session.begin();
map.update(stockId, new Integer(100));
session.commit();
```

最初の行は、後続のトランザクション・タイプが updatePrice であることを示します。updatePrice 統計は、例にあるセッションに対応する ObjectGrid インスタンスに置かれています。Java Management Extensions (JMX) インターフェースを使用して、updatePrice トランザクション用のトランザクション応答時間を取得できます。指定した ObjectGrid インスタンスで、トランザクションのすべてのタイプの集約統計も取得できます。

mapModule

mapModule は、eXtreme Scale マップに関連した 3 つの統計を含んでいます。

- **マップ・ヒット率** - *BoundedRangeStatistic*: マップのヒット率を追跡します。ヒット率は 0 以上 100 以下の浮動値で、マップ取得操作に関係するマップ・ヒットの比率です。
- **エントリー数** -*CountStatistic*: マップのエントリー数を追跡します。
- **ローダー・バッチ更新応答時間** -*TimeStatistic*: ローダー・バッチ更新操作に使用される応答時間を追跡します。

mapModule のルート・エレメント (root) は、ObjectGrid マップ統計への入り口点として機能します。このルート・エレメントは ObjectGrid を子エレメントとして持ち、さらにこの子エレメントはマップを子エレメントとして持ちます。すべてのマップ・インスタンスは、3 つのリスト統計を持っています。次の図は mapModule 構造です。

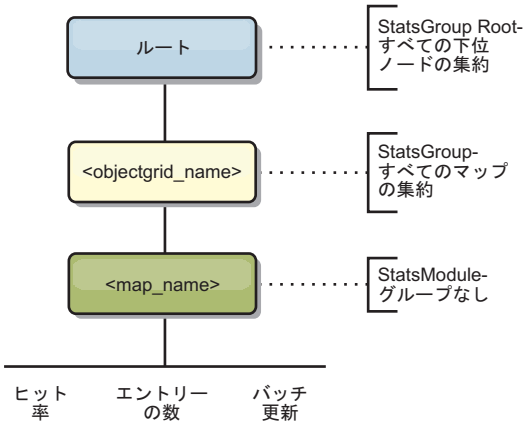
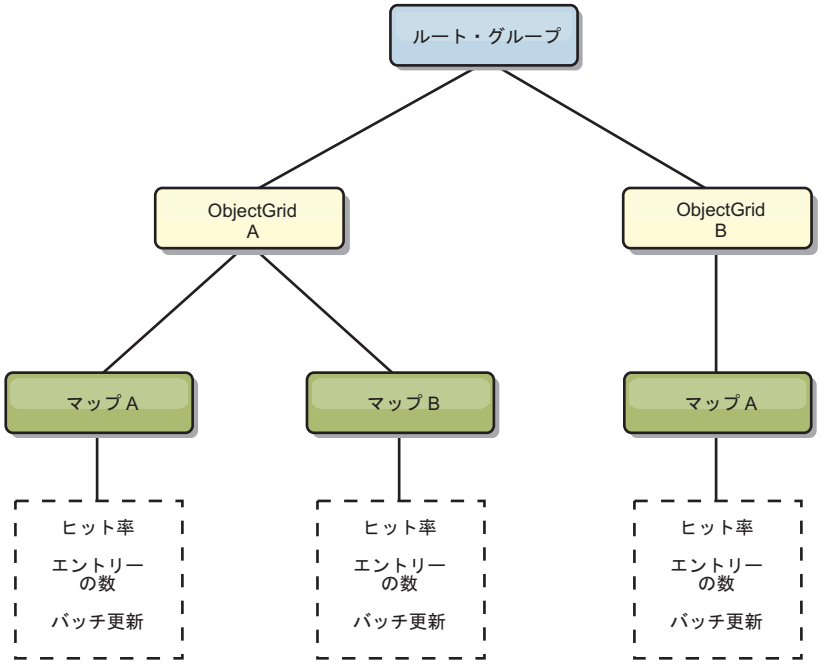


図 11. mapModule 構造

次の図は mapModule 構造の例です。

図 12. mapModule モジュール構造の例



hashIndexModule

hashIndexModule は、マップ・レベルの索引に関連する次の統計を含みます。

- **検索カウント** *-CountStatistic*: 索引検索操作の呼び出し回数。
- **衝突カウント** *-CountStatistic*: 検索操作の衝突回数。
- **障害カウント** *-CountStatistic*: 検索操作の障害件数。
- **結果カウント** *-CountStatistic*: 検索操作から戻されたキーの数。

- **バッチ更新カウント** *-CountStatistic*: この索引に対するバッチ更新の回数。対応するマップが何らかの方法で変更されると、索引で、その doBatchUpdate() メソッドが呼び出されます。この統計からは、索引の変更または更新頻度が分かります。
- **検索操作所要時間** *-TimeStatistic*: 検索操作が完了するまでに要する時間。

hashIndexModule のルート・エレメント (root) は、HashIndex 統計への入り口点として機能します。このルート・エレメントは ObjectGrid を子エレメントとして持ちます。ObjectGrid はマップを子エレメントとして持ち、最終的にこの子エレメントは HashIndexe を子エレメントおよびツリーのリーフ・ノードとして持ちます。すべての HashIndex インスタンスは 3 つのリスト統計を持っています。次の図は hashIndexModule の構造です。

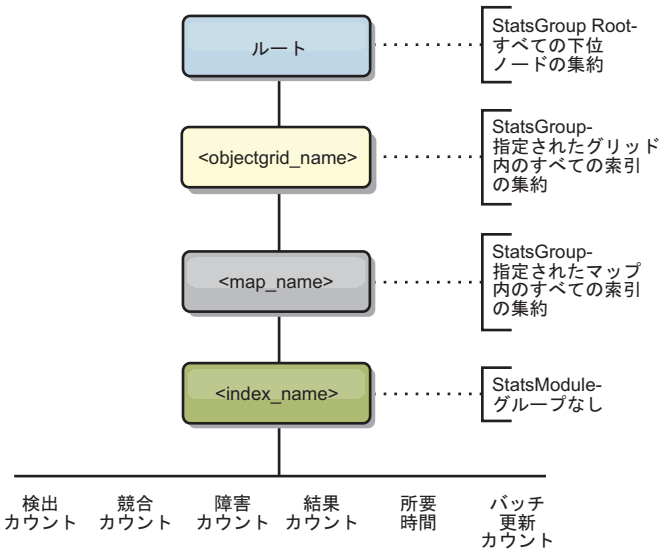


図 13. hashIndexModule モジュール構造

次の図は hashIndexModule 構造の例です。

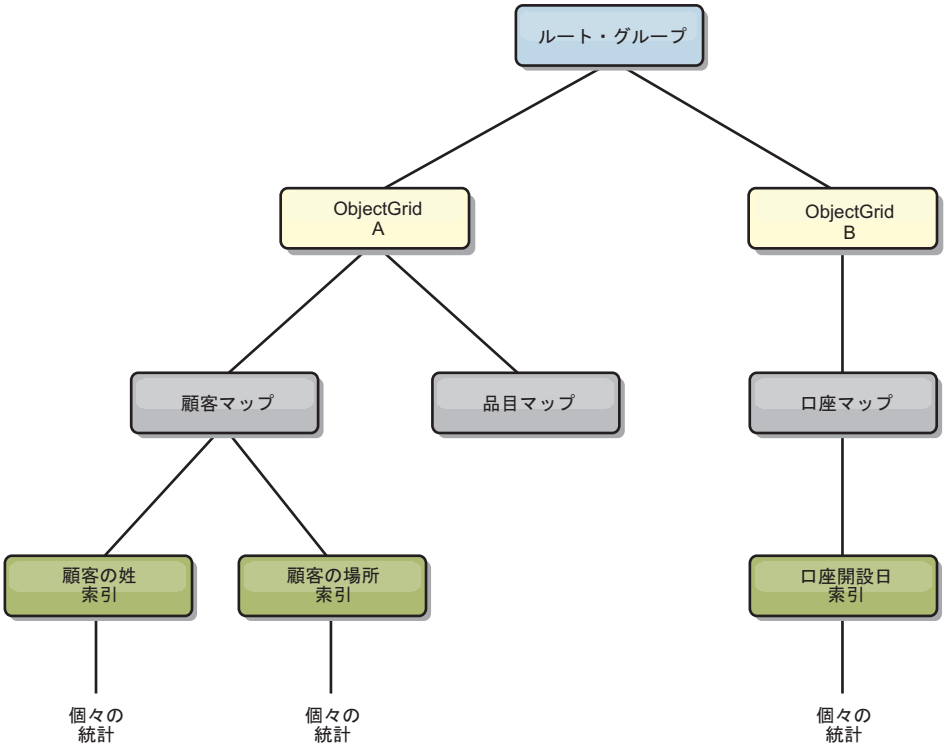


図 14. hashIndexModule モジュール構造の例

agentManagerModule

agentManagerModule は、マップ・レベルのエージェントに関連した統計を含みます。

- **削減時間**: *TimeStatistic* - エージェントが削減操作を完了するまでの時間。
- **合計所要時間**: *TimeStatistic* - エージェントがすべての操作を完了するまでの合計時間。
- **エージェント・シリアライゼーション時間**: *TimeStatistic* - エージェントをシリアライズするための時間。
- **エージェント・インフレーション時間**: *TimeStatistic* - サーバー上でエージェントをインフレーションするのに要する時間。
- **結果シリアライゼーション時間**: *TimeStatistic* - エージェントからの結果をシリアライズするための時間。
- **結果インフレーション時間**: *TimeStatistic* - エージェントからの結果をインフレーションするための時間。
- **障害カウント**: *CountStatistic* - エージェントが障害を起こした回数。
- **呼び出しカウント**: *CountStatistic* - AgentManager が呼び出された回数。
- **区画カウント**: *CountStatistic* - エージェントが送られる相手区画の数。

agentManagerModule のルート・エレメント (root) は、AgentManager 統計への入り口点として機能します。このルート・エレメントは ObjectGrid を子エレメントとして持ちます。ObjectGrid はマップを子エレメントとして持ち、最終的にこの子エレメントは AgentManager インスタンスを子エレメントおよびツリーのリーフ・ノードとして持ちます。すべての AgentManager インスタンスは 3 つのリスト統計を持

っています。

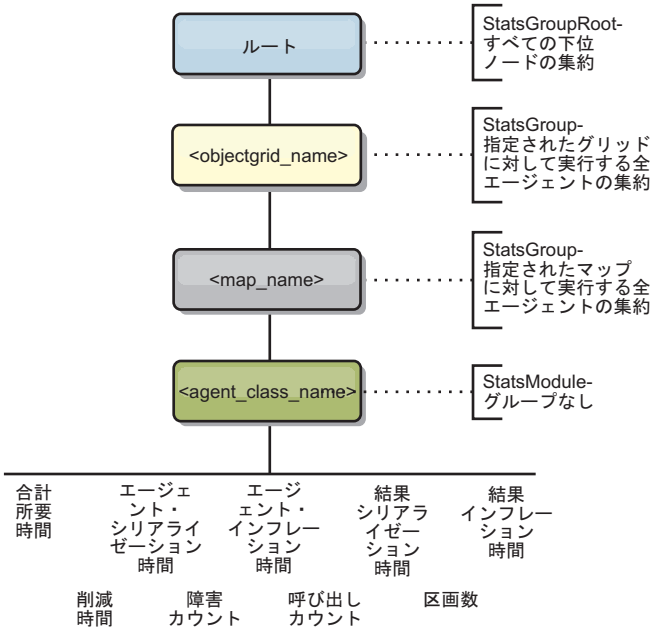


図 15. agentManagerModule 構造

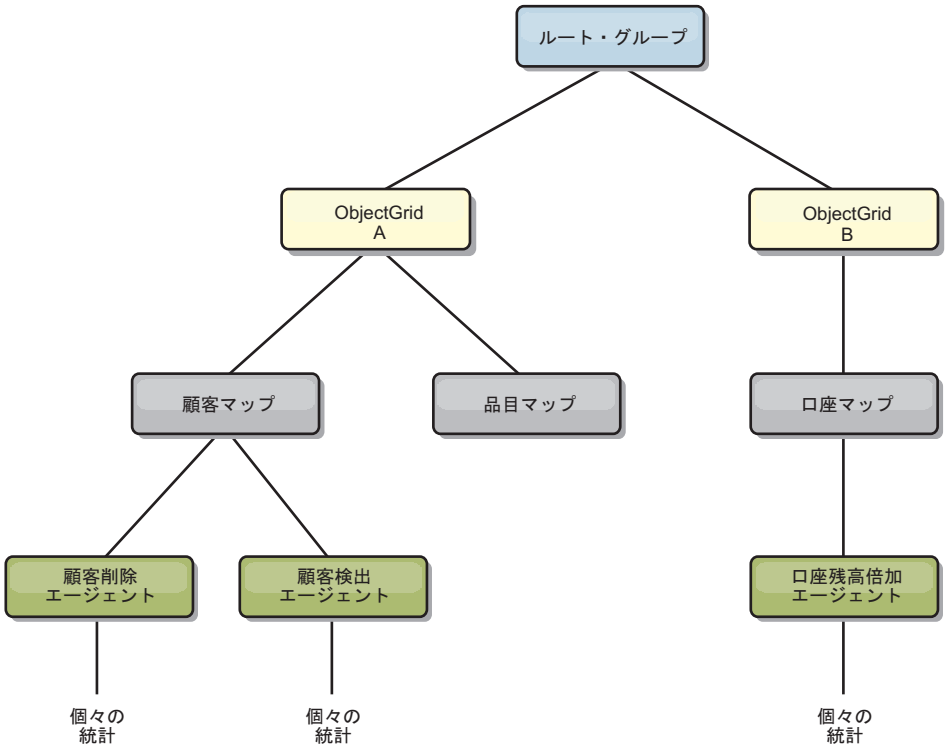


図 16. agentManagerModule 構造の例

queryModule

queryModule は、eXtreme Scale 照会に関連した統計を含みます。

- **計画作成時間**: *TimeStatistic* - 照会計画を作成するための時間。
- **実行時間**: *TimeStatistic* - 照会を実行するための時間。
- **実行カウント**: *CountStatistic* - 照会が実行された回数。
- **結果カウント**: *CountStatistic* - 実行された各照会の結果セットごとのカウント。
- **障害カウント**: *CountStatistic* - 照会が失敗した回数。

queryModule のルート・エレメント (root) は、Query 統計への入り口点として機能します。このルート・エレメントは ObjectGrid を子エレメントとして持ち、さらにこの子エレメントは照会オブジェクトを子エレメントおよびツリーのリーフ・ノードとして持ちます。すべての Query インスタンスは 3 つのリスト統計を持っています。

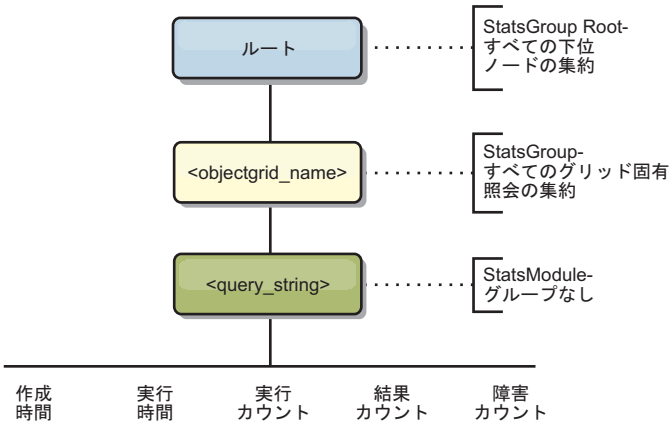


図 17. queryModule の構造

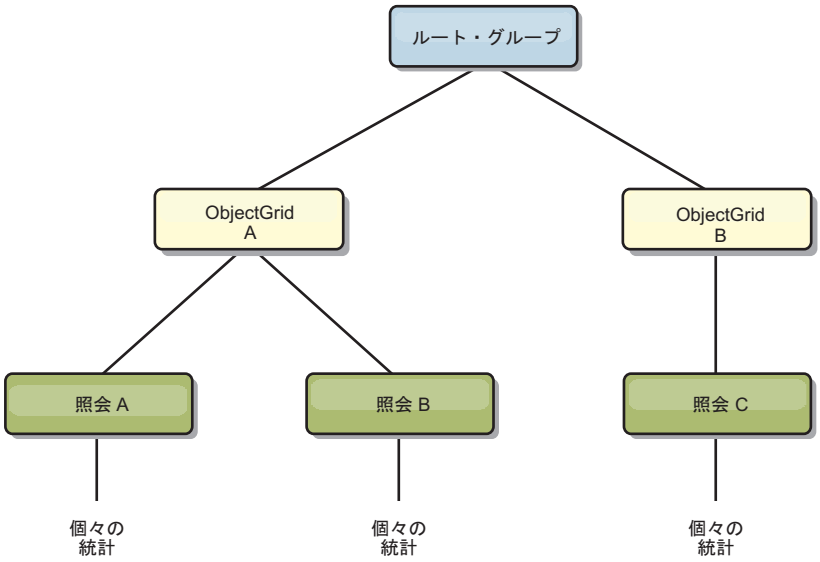


図 18. QueryStats.jpg queryModule 構造の例

関連概念

275 ページの『統計の概説』
WebSphere eXtreme Scale での統計は、内部統計ツリーから作成されます。内部ツリーからは、StatsAccessor API、Performance Monitoring Infrastructure (PMI) モジュール、および MBean API が作成されます。

関連タスク

283 ページの『WebSphere Application Server PMI によるパフォーマンスのモニター』
WebSphere eXtreme Scale は、WebSphere Application Server または WebSphere Extended Deployment アプリケーション・サーバーで実行されているとき、Performance Monitoring Infrastructure (PMI) をサポートします。 PMI は、ランタイム・アプリケーションでパフォーマンス・データを収集し、パフォーマンス・データをモニターするための外部アプリケーションをサポートするインターフェースを提供します。管理コンソールまたは wsadmin ツールを使用して、モニター・データにアクセスすることができます。

285 ページの『PMI の使用可能化』
WebSphere Application Server Performance Monitoring Infrastructure (PMI) を使用して、任意のレベルで統計を使用可能または使用不可にすることができます。例えば、特定のマップのマップ・ヒット率統計を使用可能にするが、エントリー数統計またはローダー・バッチ更新時間統計は使用可能にしないことを選択できます。管理コンソール内またはスクリプトを使用して PMI を使用可能にすることができます。

288 ページの『PMI 統計の取得』
PMI 統計を取得することによって、eXtreme Scale アプリケーションのパフォーマンスを確認できます。

300 ページの『xsAdmin サンプル・ユーティリティーの使用』
xsAdmin サンプル・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。

wsadmin ユーティリティーの使用

WebSphere Application Server で提供される wsadmin ユーティリティーを使用して、MBean 情報にアクセスすることができます。

wsadmin ツールを使用した MBean へのアクセス

WebSphere Application Server インストール内の bin ディレクトリーから wsadmin ツールを実行します。次の例は、動的 eXtreme Scale における現在の断片配置のビューを取得するものです。 wsadmin は、eXtreme Scale が稼働している任意のインストール済み環境から実行できます。 wsadmin をカタログ・サービスで実行する必要はありません。

```
$ wsadmin.sh -lang jython
wsadmin>placementService = AdminControl.queryNames
 ("com.ibm.websphere.objectgrid:*,type=PlacementService")
wsadmin>print AdminControl.invoke(placementService,
 "listObjectGridPlacement","library ms1")

<objectGrid name="library" mapSetName="ms1">
```

```
  <container name="container-0" zoneName="DefaultDomain"
  hostName="host1.company.org" serverName="server1">
     <shard type="Primary" partitionName="0"/>
     <shard type="SynchronousReplica" partitionName="1"/>
  </container>
  <container name="container-1" zoneName="DefaultDomain"
  hostName="host2.company.org" serverName="server2">
     <shard type="SynchronousReplica" partitionName="0"/>
     <shard type="Primary" partitionName="1"/>
  </container>
  <container name="UNASSIGNED" zoneName="_ibm_SYSTEM"
  hostName="UNASSIGNED" serverName="UNNAMED">
    <shard type="SynchronousReplica" partitionName="0"/>
    <shard type="AsynchronousReplica" partitionName="0"/>
  </container>
</objectGrid>
```

Managed Bean (MBean) を使用した環境の管理

デプロイメントを管理およびモニターするには、さまざまなタイプの Java Management Extensions (JMX) MBeans を使用できます。各 MBean は、マップ、eXtreme Scale、サーバー、複製グループ、または複製グループ・メンバーなどの特定のエンティティーを参照します。

JMX MBean インターフェースおよび WebSphere eXtreme Scale

各 MBean には、属性値を表す get メソッドがあります。この get メソッドは、プログラムから直接呼び出すことはできません。JMX 仕様では、属性の扱い方が操作のときと異なります。ベンダー JMX コンソールを使用して属性を表示し、プログラムまたはベンダー JMX コンソールで操作を実行することができます。

関連概念

275 ページの『統計の概説』
WebSphere eXtreme Scale での統計は、内部統計ツリーから作成されます。内部ツリーからは、StatsAccessor API、Performance Monitoring Infrastructure (PMI) モジュール、および MBean API が作成されます。

303 ページの『ベンダー・ツール』
一般的によく使われるいくつかのエンタープライズ・モニタリング・ソリューションを使用して、WebSphere eXtreme Scale をモニターすることができます。 パブリックにアクセス可能な管理 Bean を使用して WebSphere eXtreme Scale をモニターする IBM Tivoli Monitoring および Hyperic HQ 用に、プラグイン・エージェントが組み込まれています。CA Wily Introscope は Java メソッドのインスツルメンテーションを使用して、統計情報を収集します。

関連タスク

275 ページの『第 8 章 デプロイメント環境のモニター』
API、MBean、ログ、およびユーティリティーを使用して、アプリケーション環境のパフォーマンスをモニターできます。

279 ページの『統計 API によるモニター』
統計 API は、内部統計ツリーに直接接続するインターフェースです。統計はデフォルトでは使用不可になっていますが、StatsSpec インターフェースを設定することで使用可能にすることができます。 StatsSpec インターフェースは、WebSphere eXtreme Scale がどのように統計をモニターするかを定義します。

283 ページの『WebSphere Application Server PMI によるパフォーマンスのモニター』
WebSphere eXtreme Scale は、WebSphere Application Server または WebSphere Extended Deployment アプリケーション・サーバーで実行されているとき、Performance Monitoring Infrastructure (PMI) をサポートします。 PMI は、ランタイム・アプリケーションでパフォーマンス・データを収集し、パフォーマンス・データをモニターするための外部アプリケーションをサポートするインターフェースを提供します。管理コンソールまたは wsadmin ツールを使用して、モニター・データにアクセスすることができます。

Managed Bean 使用の概要

ご使用の環境の統計をトラッキングするために Managed Bean (MBean) を使用できます。

始める前に

属性が記録されるようにするには、統計を使用可能に設定する必要があります。統計は、以下のいずれかの方法で使用可能にできます。

- **サーバー・プロパティー・ファイルを使用:**

  サーバー・プロパティー・ファイルの `statsSpec=<StatsSpec>` というキー値エントリーを設定して統計を使用可能にすることができます。次に、設定可能な例をいくつか示します。

  – すべての統計を使用可能にする場合、`statsSpec=all=enabled` または `statisticsSpec=all=enabled` を使用します。

– ObjectGrid 統計のみを使用可能にする場合は、`statsSpec=og.all=enabled` を使用します。使用可能なすべての統計の仕様の説明を確認するには、API 資料の StatsSpec API を参照してください。

サーバー・プロパティー・ファイルに関して詳しくは、203 ページの『サーバー・プロパティー・ファイル』を参照してください。

- **プログラムで:**

  StatsAccessor インターフェースを使用して、統計をプログラムで使用可能にすることもできます。このインターフェースは、 StatsAccessorFactory クラスを使用して取得されます。このインターフェースは、クライアント環境で使用するか、現行プロセスで実行中の eXtreme Scale をモニターする必要がある場合に使用します。

例

Managed Bean の使用例は、『xsAdmin サンプル・ユーティリティーの使用』を参照してください。

xsAdmin サンプル・ユーティリティーの使用

xsAdmin サンプル・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

WebSphere eXtreme Scale がインストールされていなければなりません。

このタスクについて

xsAdmin サンプル・ユーティリティーを使用することで、グリッドの現在のレイアウトおよび特定の状態に関するフィードバック (マップの内容など) を提供することができます。この例では、このタスクにおけるグリッドのレイアウトは *ObjectGridA* という単一のグリッドで構成されています。このグリッドは、*MapA* という定義済みのマップを 1 つ持ち、*MapSetA* というマップ・セットに属しています。この例は、グリッド内のすべてのアクティブ・コンテナーを表示し、*MapA* のマップ・サイズに関するフィルタリング済みメトリックを印刷する方法を示しています。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。

1. コマンド行で、JAVA_HOME 環境変数を設定します。
   - UNIX `export JAVA_HOME=javaHome`
   - Windows `set JAVA_HOME=javaHome`
2. bin ディレクトリーに移動します。

   ```
   cd objectGridRoot/bin
   ```
3. xsAdmin ユーティリティーを起動します。

- **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
  ```

  ヘルプ・メッセージの必須引数セクションに注意してください。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 1 つだけを渡すようにする必要があるからです。 **-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。
- **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。

  ```
   This administrative utility is provided as a sample only and is not to be
  considered a fully supported component of the WebSphere eXtreme Scale product
  Connecting to Catalog service at localhost:1099
  *** Show all online containers for grid - ObjectGridA & mapset - MapSetA
  Host: 192.168.0.186
    Container: server1_C-0, Server:server1, Zone:DefaultZone
      P:0 Primary
    Num containers matching = 1
    Total known containers = 1
    Total known hosts = 1
  ```
- **ある特定のグリッドについてすべてのマップのエントリー数を表示するには、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。

  ```
  This administrative utility is provided as a sample only and is not to be
  considered a fully supported component of the WebSphere eXtreme Scale product

  Connecting to Catalog service at localhost:1099

  ****Displaying Results for Grid - ObjectGridA, MapSet - MapSetA*****
  ```

```
*** Listing Maps for server1 ***
 Map Name: MapA  Partition #: 0  Map Size: 0  Shard Type: Primary
Server Total: 0
```

- **カタログ・サービスの JMX ポートを指定するには、次のコマンドを実行します。** xsAdmin サンプル・ユーティリティーは、カタログ・サーバーで実行されている 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 プロセスで実行できますが、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。

  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
  ```

  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。

  ```
  This administrative utility is provided as a sample only and is not to be
  considered a fully supported component of the WebSphere eXtreme Scale product

  Connecting to Catalog service at 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 wasRoot/bin
     ```

b. 2. 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
```

c. 指定したマップのサイズが表示されます。

```
This administrative utility is provided as a sample only and is not to be
considered a fully supported component of the WebSphere eXtreme Scale product

Connecting to Catalog service at localhost: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
```

関連概念

275 ページの『統計の概説』

WebSphere eXtreme Scale での統計は、内部統計ツリーから作成されます。内部ツリーからは、StatsAccessor API、Performance Monitoring Infrastructure (PMI) モジュール、および MBean API が作成されます。

関連資料

289 ページの『PMI モジュール』

Performance Monitoring Infrastructure (PMI) モジュールを使用してアプリケーションのパフォーマンスをモニターすることができます。

ベンダー・ツール

一般的によく使われるいくつかのエンタープライズ・モニタリング・ソリューションを使用して、WebSphere eXtreme Scale をモニターすることができます。 パブリックにアクセス可能な管理 Bean を使用して WebSphere eXtreme Scale をモニターする IBM Tivoli Monitoring および Hyperic HQ 用に、プラグイン・エージェントが組み込まれています。CA Wily Introscope は Java メソッドのインスツルメンテーションを使用して、統計情報を収集します。

関連タスク

275 ページの『第 8 章 デプロイメント環境のモニター』
API、MBean、ログ、およびユーティリティーを使用して、アプリケーション環境のパフォーマンスをモニターできます。

279 ページの『統計 API によるモニター』
統計 API は、内部統計ツリーに直接接続するインターフェースです。統計はデフォルトでは使用不可になっていますが、StatsSpec インターフェースを設定することで使用可能にすることができます。 StatsSpec インターフェースは、WebSphere eXtreme Scale がどのように統計をモニターするかを定義します。

283 ページの『WebSphere Application Server PMI によるパフォーマンスのモニター』
WebSphere eXtreme Scale は、WebSphere Application Server または WebSphere Extended Deployment アプリケーション・サーバーで実行されているとき、Performance Monitoring Infrastructure (PMI) をサポートします。 PMI は、ランタイム・アプリケーションでパフォーマンス・データを収集し、パフォーマンス・データをモニターするための外部アプリケーションをサポートするインターフェースを提供します。管理コンソールまたは wsadmin ツールを使用して、モニター・データにアクセスすることができます。

315 ページの『Hyperic HQ による eXtreme Scale のモニター』
Hyperic HQ は、サード・パーティーのモニター・ソリューションで、オープン・ソース・ソリューションあるいはエンタープライズ製品として自由に使用可能です。WebSphere eXtreme Scale には、あるプラグインが含まれていて、このプラグインにより Hyperic HQ エージェントは eXtreme Scale コンテナー・サーバーを検出し、eXtreme Scale 管理 Bean を使用して統計のレポートおよび集約を行うことができます。 Hyperic HQ を使用すると、スタンドアロン eXtreme Scale デプロイメントをモニターできます。

『IBM Tivoli Enterprise Monitoring Agent for WebSphere eXtreme Scale のモニター』
IBM Tivoli Enterprise Monitoring Agent は、分散環境およびホスト環境のデータベース、オペレーティング・システム、およびサーバーをモニターするために使用できる機能の豊富なモニタリング・ソリューションです。 WebSphere eXtreme Scale には、eXtreme Scale 管理 Bean をイントロスペクトする場合に使用できるカスタマイズ・エージェントが含まれています。このソリューションは、スタンドアロン eXtreme Scale および WebSphere Application Server デプロイメントの両方で効果的に機能します。

関連資料

298 ページの『Managed Bean (MBean) を使用した環境の管理』
デプロイメントを管理およびモニターするには、さまざまなタイプの Java Management Extensions (JMX) MBeans を使用できます。各 MBean は、マップ、eXtreme Scale、サーバー、複製グループ、または複製グループ・メンバーなどの特定のエンティティーを参照します。

IBM Tivoli Enterprise Monitoring Agent for WebSphere eXtreme Scale のモニター

IBM Tivoli Enterprise Monitoring Agent は、分散環境およびホスト環境のデータベース、オペレーティング・システム、およびサーバーをモニターするために使用で

きる機能の豊富なモニタリング・ソリューションです。 WebSphere eXtreme Scale には、eXtreme Scale 管理 Bean をイントロスペクトする場合に使用できるカスタマイズ・エージェントが含まれています。このソリューションは、スタンドアロン eXtreme Scale および WebSphere Application Server デプロイメントの両方で効果的に機能します。

始める前に

- WebSphere eXtreme Scale バージョン 7.0.0 以降をインストールします。
- IBM Tivoli Monitoring バージョン 6.2.1 (フィックスパック 2 以降の適用済み) をインストールします。
- eXtreme Scale サーバーが実行される各サーバーまたはホストに Tivoli OS エージェントをインストールします。
- WebSphere eXtreme Scale エージェントをインストールします。 (IBM Open Process Automation Library (OPAL) サイトから無料でダウンロードできます。)

以下の手順を実行して、Tivoli Monitoring Agent をインストールおよび構成します。

1. WebSphere eXtreme Scale 用に Tivoli Monitoring Agent をインストールします。

   Tivoli インストール・イメージをダウンロードし、そのファイルを一時ディレクトリーに解凍します。
2. eXtreme Scale アプリケーション・サポート・ファイルをインストールします。

   以下の各デプロイメントに eXtreme Scale アプリケーション・サポートをインストールします。
   - Tivoli Enterprise Portal Server (TEPS)
   - Enterprise Desktop クライアント (TEPD)
   - Tivoli Enterprise Monitoring Server (TEMS)

   a. 作成した一時ディレクトリーから、新しいコマンド・ウィンドウを開始し、ご使用のプラットフォームに合った実行可能ファイルを実行します。 インストール・スクリプトが、ご使用の Tivoli デプロイメント・タイプ (TEMS、TEPD、または TEPS) を自動的に検出します。タイプによらず単一のホストでも複数のホストでもインストールできますが、デプロイメントの場合は 3 つのタイプのすべてについて eXtreme Scale エージェントのアプリケーション・サポート・ファイルをインストールする必要があります。

   b. 「**インストーラー**」ウィンドウで、デプロイされた Tivoli コンポーネントの選択が正しいことを確認します。「**次へ**」をクリックします。

   c. プロンプトが出されたら、ホスト名および管理クレデンシャルをサブミットします。「**次へ**」をクリックします。

   d. 「**Monitoring Agent for WebSphere eXtreme Scale**」を選択します。「**次へ**」をクリックします。

   e. 実行されるインストール・アクションについて通知があります。「**次へ**」をクリックすると、インストールの進行状況を完了まで確認することができます。

   手順を完了すると、WebSphere eXtreme Scale エージェントが必要とするすべてのアプリケーション・サポート・ファイルがインストールされます。

3. 各 eXtreme Scale ノードにエージェントをインストールします。

   各コンピューターに Tivoli OS エージェントをインストールします。このエージェントを構成または開始する必要はありません。上記のステップと同じインストール・イメージを使用して、プラットフォーム固有の実行可能ファイルを実行します。

   指針としては、ホストごとにエージェントを 1 つだけインストールするようにする必要があります。 1 つのエージェントで eXtreme Scale サーバーの多数のインスタンスをサポートすることができます。 1 つのエージェント・インスタンスで約 50 の eXtreme Scale サーバーをモニターすると、最適のパフォーマンスが得られます。

   a. 「インストール・ウィザード」初期画面で「**次へ**」をクリックすると、インストール・パス情報を指定する画面が開きます。
   b. 「**Tivoli Monitoring インストール・ディレクトリー**」フィールドに、`C:¥IBM¥ITM` (または `/opt/IBM/ITM`) と入力するか、またはこのパスを参照します。 次に「**インストール可能なメディアのロケーション**」フィールドで、表示されている値が正しいことを確認してから「**次へ**」をクリックします。
   c. 追加するコンポーネント (「**ソリューションのローカル・インストールの実行**」など) を選択して、「**次へ**」をクリックします。
   d. サポートを追加する対象のアプリケーション (「**Monitoring Agent for WebSphere eXtreme Scale**」など) を選択して、「**次へ**」をクリックします。
   e. アプリケーション・サポートが正常に追加されるまで進行状況を確認することができます。

   注: それぞれの eXtreme Scale ノードに これまでのステップを繰り返します。サイレント・インストールを使用することもできます。サイレント・インストールに関して詳しくは、IBM Tivoli Monitoring インフォメーション・センターを参照してください。

4. WebSphere eXtreme Scale エージェントを構成します。

   インストールした各エージェントを、カタログ・サーバー、eXtreme Scale サーバー、またはその両方をモニターするように構成する必要があります。

   Windows プラットフォームと UNIX プラットフォームとでは構成手順が異なります。 Windows プラットフォームの場合の構成は、**Tivoli Monitoring サービスの管理**ユーザー・インターフェースを使用して実行します。 UNIX プラットフォームの場合の構成はコマンド行に基づいて行います。

   Windows 以下のステップを使用して、まず Windows 上のエージェントを構成します Windows

   a. 「**Tivoli Enterprise Monitoring サービスの管理**」ウィンドウで、「**スタート**」 → 「**すべてのプログラム**」 → 「**IBM Tivoli Monitoring**」 → 「**Tivoli Monitoring サービスの管理**」の順にクリックします。
   b. 「**Monitoring Agent for WebSphere eXtreme Scale**」を右クリックし、「**デフォルトを使用して構成**」を選択します。そうすると、エージェントに固有のインスタンスを作成するウィンドウが開きます。

c. 固有の名前 (例えば「instance1」など) を入力し、「**次へ**」をクリックします。

- Windows スタンドアロンの eXtreme Scale サーバーをモニターする場合は、次のステップを実行します。
  a. Java パラメーターを更新し、「**Java Home**」の値が正しいことを確認します。 JVM 引数は空のままでもかまいません。「**次へ**」をクリックします。
  b. 「**MBean サーバー接続タイプ**」のタイプを選択し、スタンドアロン eXtreme Scale サーバーの場合は「JSR-160 準拠サーバー」を使用します。「**次へ**」をクリックします。
  c. セキュリティーが有効な場合、「**ユーザー ID**」および「**パスワード**」値を更新します。「**JMX サービス URL**」の値は、そのままにしておきます。この値は、後でオーバーライドします。「**JMX クラスパス情報**」フィールドはそのままにしておきます。「**次へ**」をクリックします。

  Windows でエージェント用にサーバーを構成するには、以下のステップを実行します。
  a. 「**WebSphere eXtreme Scale グリッド・サーバー**」ペインで eXtreme Scale サーバーのサブノード・インスタンスをセットアップします。ご使用のコンピューター上にコンテナー・サーバーがない場合、「**次へ**」をクリックして、カタログ・サービス・ペインに進みます。
  b. ご使用のコンピューターに複数の eXtreme Scale コンテナー・サーバーがある場合、エージェントで各サーバーをモニターするように構成します。
  c. それぞれの名前とポートが固有な場合は、「**新規**」をクリックし、eXtreme Scale サーバーを必要なだけいくつでも追加することができます。(eXtreme Scale サーバーが始動されるときは、JMXPort 値が指定されていなければなりません。)
  d. コンテナー・サーバーの構成を完了したならば、「**次へ**」をクリックして「**WebSphere eXtreme Scale カタログ・サーバー**」ペインに進みます。
  e. カタログ・サーバーがない場合は、「**OK**」をクリックします。カタログ・サーバーがある場合は、コンテナー・サーバーについて実行したのと同様に各サーバーに新しい構成を追加します。ここでもまた、固有の名前 (できればカタログ・サービスを開始するときに使用するものと同じ名前) を選択します。「**OK**」をクリックして終了します。

- Windows WebSphere Application Server プロセス内に組み込まれている eXtreme Scale サーバー上でエージェントのサーバーをモニターする場合、以下のステップを実行します。
  a. Java パラメーターを更新し、「**Java Home**」の値が正しいことを確認します。 JVM 引数は空のままでもかまいません。「**次へ**」をクリックします。
  b. 「**MBean サーバー接続タイプ**」を選択します。ご使用の環境に適した WebSphere Application Server バージョンを選択します。「**次へ**」をクリックします。
  c. パネルの WebSphere Application Server 情報が正しいことを確認します。「**次へ**」をクリックします。

d. サブノード定義を 1 つのみ追加します。サブノード定義に名前を指定し、ポート定義は更新しないでください。 WebSphere Application Server 環境内で、そのコンピューター上で実行中のノード・エージェントによって管理されるすべてのアプリケーション・サーバーからデータが収集されます。「**次へ**」をクリックします。

e. この環境にカタログ・サーバーが存在しない場合は、「**OK**」をクリックします。カタログ・サーバーがある場合は、コンテナー・サーバーと同様、それぞれのカタログ・サーバーについて新しい構成を追加します。カタログ・サービスに固有の名前、できればカタログ・サービスを開始するときに使用するものと同じ名前を選択します。「**OK**」をクリックして終了します。

注: コンテナー・サーバーは、カタログ・サービスと連結する必要はありません。

これでエージェントとサーバーが構成されて作動可能になるので、次のウィンドウで「instance1」を右クリックしてエージェントを開始します。

UNIX UNIX プラットフォーム上のエージェントをコマンド行で構成する場合は、以下のステップを実行します。

次に JSR160 準拠の接続タイプを使用するスタンドアロン・サーバーの例を示します。この例では、単一ホスト (rhea00b02) 上に 3 つの eXtreme Scale コンテナーがあり、JMX リスナーのアドレスは、それぞれ 15000、15001 および 15002 です。カタログ・サーバーはありません。

構成ユーティリティーからの出力は*モノスペースのイタリック体* で、一方ユーザー応答は**モノスペースの太字体**で示されています。 (ユーザー応答が不要であった場合は、Enter キーを押してデフォルトが選択されました。) UNIX

```
rhea00b02 # ./itmcmd config -A xt
Agent configuration started...
Enter instance name (default is: ): inst1
Edit "Monitoring Agent for WebSphere eXtreme Scale" settings? [ 1=Yes, 2=No ] (default is: 1):
Edit 'Java' settings? [ 1=Yes, 2=No ] (default is: 1):
Java home (default is: C:¥Program Files¥IBM¥Java50): /opt/OG61/java
Java trace level [ 1=Error, 2=Warning, 3=Information, 4=Minimum Debug, 5=Medium Debug, 6=Maximum Debug, 7=All ] (default is: 1):
JVM arguments (default is: ):
Edit 'Connection' settings? [ 1=Yes, 2=No ] (default is: 1):
MBean server connection type [ 1=JSR-160-Compliant Server, 2=WebSphere Application Server version 6.0,
3=WebSphere Application Server version 6.1, 4=WebSphere Application Server version 7.0 ] (default is: 1): 1
Edit 'JSR-160-Compliant Server' settings? [ 1=Yes, 2=No ] (default is: 1):
JMX user ID (default is: ):
Enter JMX password (default is: ):
Re-type : JMX password (default is: ):
JMX service URL (default is: service:jmx:rmi:///jndi/rmi://localhost:port/objectgrid/MBeanServer):
----------------------------------------
JMX Class Path Information
JMX base paths (default is: ):
JMX class path (default is: ):
JMX JAR directories (default is: ):
Edit 'WebSphere eXtreme Scale Catalog Service' settings? [ 1=Yes, 2=No ] (default is: 1): 2
Edit 'WebSphere eXtreme Scale Grid Servers' settings? [ 1=Yes, 2=No ] (default is: 1): 1
No 'WebSphere eXtreme Scale Grid Servers' settings available?
Edit 'WebSphere eXtreme Scale Grid Servers' settings, [1=Add, 2=Edit, 3=Del, 4=Next, 5=Exit] (default is: 4): 1
WebSphere eXtreme Scale Grid Servers (default is: ): rhea00b02_c0
JMX service URL (default is: service:jmx:rmi:///jndi/rmi://localhost:<port>/objectgrid/MBeanServer):
service:jmx:rmi:///jndi/rmi://localhost:15000/objectgrid/MBeanServer

'WebSphere eXtreme Scale Grid Servers' settings:  WebSphere eXtreme Scale Grid Servers=ogx
Edit 'WebSphere eXtreme Scale Grid Servers' settings, [1=Add, 2=Edit, 3=Del, 4=Next, 5=Exit] (default is: 4): 1
WebSphere eXtreme Scale Grid Servers (default is: ): rhea00b02_c1
JMX service URL (default is: service:jmx:rmi:///jndi/rmi://localhost:<port>/objectgrid/MBeanServer):
service:jmx:rmi:///jndi/rmi://localhost:15001/objectgrid/MBeanServer

'WebSphere eXtreme Scale Grid Servers' settings:  WebSphere eXtreme Scale Grid Servers= rhea00b02_c1
```

```
Edit 'WebSphere eXtreme Scale Grid Servers' settings, [1=Add, 2=Edit, 3=Del, 4=Next, 5=Exit] (default is: 4): 1
WebSphere eXtreme Scale Grid Servers (default is: ): rhea00b02_c2
JMX service URL (default is: service:jmx:rmi:///jndi/rmi://localhost:<port>/objectgrid/MBeanServer):
service:jmx:rmi:///jndi/rmi://localhost:15002/objectgrid/MBeanServer

'WebSphere eXtreme Scale Grid Servers' settings:  WebSphere eXtreme Scale Grid Servers= rhea00b02_c2
Edit 'WebSphere eXtreme Scale Grid Servers' settings, [1=Add, 2=Edit, 3=Del, 4=Next, 5=Exit] (default is: 4): 5

Will this agent connect to a TEMS? [1=YES, 2=NO] (Default is: 1):
TEMS Host Name (Default is: rhea00b00):

Network Protocol [ip, sna, ip.pipe or ip.spipe] (Default is: ip.pipe):

     Now choose the next protocol number from one of these:
     - ip
     - sna
     - ip.spipe
     - 0 for none
Network Protocol 2 (Default is: 0):
IP.PIPE Port Number (Default is: 1918):
Enter name of KDC_PARTITION (Default is: null):

Configure connection for a secondary TEMS? [1=YES, 2=NO] (Default is: 2):
Enter Optional Primary Network Name or 0 for "none" (Default is: 0):
Agent configuration completed...
```

上記の例では、「inst1」というエージェント・インスタンスが作成され、Java ホーム設定が更新されます。 eXtreme Scale コンテナー・サーバーは構成されますが、カタログ・サービスは構成されません。

注: 上記の手順では、次の形式のテキスト・ファイルがディレクトリー <ITM_install>/config/<host>_xt_<instance name>.cfg に作成されます。

例: rhea00b02_xt_inst1.cfg

自分で選んだプレーン・テキスト・エディターを使用してこのファイルを編集することをお勧めします。そうしたファイルの内容の例を以下に示します。

```
INSTANCE=inst2 [        SECTION=KQZ_JAVA [ { JAVA_HOME=/opt/OG61/java }  { JAVA_TRACE_LEVEL=ERROR }  ]
SECTION=KQZ_JMX_CONNECTION_SECTION [ { KQZ_JMX_CONNECTION_PROPERTY=KQZ_JMX_JSR160_JSR160 }  ]
SECTION=KQZ_JMX_JSR160_JSR160 [ { KQZ_JMX_JSR160_JSR160_CLASS_PATH_TITLE= }
{ KQZ_JMX_JSR160_JSR160_SERVICE_URL=service:jmx:rmi:///jndi/rmi://localho
st:port/objectgrid/MBeanServer }  { KQZ_JMX_JSR160_JSR160_CLASS_PATH_SEPARATOR= }  ]
SECTION=OGS:rhea00b02_c1 [ { KQZ_JMX_JSR160_JSR160_SERVICE_URL=service:jmx:
rmi:///jndi/rmi://localhost:15001/objectgrid/MBeanServer }  ]
SECTION=OGS:rhea00b02_c0 [ { KQZ_JMX_JSR160_JSR160_SERVICE_URL=service:jmx:
rmi:///jndi/rmi://localhost:15002/objectgrid/MBeanServer }  ]
SECTION=OGS:rhea00b02_c2 [ { KQZ_JMX_JSR160_JSR160_SERVICE_URL=service:jmx:
rmi:///jndi/rmi://localhost:15002/objectgrid/MBeanServer }  ]]
```

次に、WebSphere Application Server デプロイメント上の構成の例を示します。

```
rhea00b02 # ./itmcmd config -A xt
Agent configuration started...
Enter instance name (default is: ): inst1
Edit "Monitoring Agent for WebSphere eXtreme Scale" settings? [ 1=Yes, 2=No ] (default is: 1): 1
Edit 'Java' settings? [ 1=Yes, 2=No ] (default is: 1): 1
Java home (default is: C:¥Program Files¥IBM¥Java50): /opt/WAS61/java
Java trace level [ 1=Error, 2=Warning, 3=Information, 4=Minimum Debug, 5=Medium Debug, 6=Maximum Debug, 7=All ] (default is: 1):
JVM arguments (default is: ):
Edit 'Connection' settings? [ 1=Yes, 2=No ] (default is: 1):
MBean server connection type [ 1=JSR-160-Compliant Server, 2=WebSphere Application Server version 6.0,
3=WebSphere Application Server version 6.1, 4=WebSphere Application Server version 7.0 ] (default is: 1): 4
Edit 'WebSphere Application Server version 7.0' settings? [ 1=Yes, 2=No ] (default is: 1):WAS user ID (default is: ):
Enter WAS password (default is: ):
Re-type : WAS password (default is: ):
WAS host name (default is: localhost): rhea00b02
WAS port (default is: 2809):
WAS connector protocol [ 1=rmi, 2=soap ] (default is: 1):
WAS profile name (default is: ): default
----------------------------------------
WAS Class Path Information
WAS base paths (default is: C:¥Program Files¥IBM¥WebSphere¥AppServer;/opt/IBM/WebSphere/AppServer): /opt/WAS61
WAS class path (default is: runtimes/com.ibm.ws.admin.client_6.1.0.jar;runtimes/com.ibm.ws.ejb.thinclient_7.0.0.jar):
WAS JAR directories (default is: lib;plugins):
Edit 'WebSphere eXtreme Scale Grid Servers' settings? [ 1=Yes, 2=No ] (default is: 1):
No 'WebSphere eXtreme Scale Grid Servers' settings available?
Edit 'WebSphere eXtreme Scale Grid Servers' settings, [1=Add, 2=Edit, 3=Del, 4=Next, 5=Exit] (default is: 4): 1
WebSphere eXtreme Scale Grid Servers (default is: ): rhea00b02
JMX service URL (default is: service:jmx:rmi:///jndi/rmi://localhost:<port>/objectgrid/MBeanServer):
```

```
'WebSphere eXtreme Scale Grid Servers' settings:  WebSphere eXtreme Scale Grid Servers=rhea00b02
Edit 'WebSphere eXtreme Scale Grid Servers' settings, [1=Add, 2=Edit, 3=Del, 4=Next, 5=Exit] (default is: 4): 5
Edit 'WebSphere eXtreme Scale Catalog Service' settings? [ 1=Yes, 2=No ] (default is: 1): 2
Will this agent connect to a TEMS? [1=YES, 2=NO] (Default is: 1):
TEMS Host Name (Default is: rhea00b02):

Network Protocol [ip, sna, ip.pipe or ip.spipe] (Default is: ip.pipe):

     Now choose the next protocol number from one of these:
     - ip
     - sna
     - ip.spipe
     - 0 for none
Network Protocol 2 (Default is: 0):
IP.PIPE Port Number (Default is: 1918):
Enter name of KDC_PARTITION (Default is: null):

Configure connection for a secondary TEMS? [1=YES, 2=NO] (Default is: 2):
Enter Optional Primary Network Name or 0 for "none" (Default is: 0):
Agent configuration completed...
rhea00b02 #
```

WebSphere Application Server デプロイメントの場合、複数のサブノードを作成する必要はありません。eXtreme Scale エージェントは、ノード・エージェントに接続して、管理下のアプリケーション・サーバーからすべての情報を収集します。

SECTION=CAT はカタログ・サービス行を意味し、一方 SECTION=OGS は eXtreme Scale サーバー構成行を意味します。

5. eXtreme Scale サーバーを構成します。

-JMXServicePort 引数が指定されないまま eXtreme Scale コンテナー・サーバーが始動されるときは、MBean サーバーに動的ポートが割り当てられます。エージェントは通信相手の JMX ポートをあらかじめ知っておく必要があります。エージェントは動的ポートとは連動しません。

サーバーの始動時、startOgServer.sh | .bat コマンドを使用して eXtreme Scale サーバーを開始する場合は -JMXServicePort <port_number> 引数を指定します。このコマンドの実行により、プロセス内の JMX サーバーが静的事前定義ポートを listen するようになります。

UNIX インストールの上記の例の場合は、2 つの eXtreme Scale サーバーを次のように設定されたポートで始動する必要があります。

a. ″-JMXServicePort″ ″15000″ (rhea00b02_c0 の場合)
b. ″-JMXServicePort″ ″15001″ (rhea00b02_c1 の場合)

a. WebSphere eXtreme Scale エージェントを開始します。

   inst1 インスタンスが作成されたとすると、上記の例のように、以下のコマンドを発行します。

   1) cd <ITM_install>/bin
   2) itmcmd agent –o inst1 start xt

b. WebSphere eXtreme Scale エージェントを停止します。

   「inst1」が上記の例のようにして作成されたインスタンスであったとして、以下のコマンドを発行します。

   1) cd <ITM_install>/bin
   2) itmcmd agent –o inst1 stop xt

タスクの結果

すべてのサーバーが構成されて始動されたならば、IBM Tivoli Portal コンソールに MBean データが表示されます。事前定義ワークスペースには、グラフとデータ・メトリックがノード・レベルごとに表示されます。

モニターされるすべてのノードの「**WebSphere eXtreme Scale グリッド・サーバー**」ノードについて、以下のワークスペースが定義されています。

- eXtreme Scale トランザクション・ビュー
- eXtreme Scale プライマリー断片ビュー
- eXtreme Scale メモリー・ビュー
- eXtreme Scale ObjectMap ビュー

独自のワークスペースも構成することができます。詳しくは、 IBM Tivoli Monitoring インフォメーション・センターのワークスペースのカスタマイズに関する情報を参照してください。

関連概念

303 ページの『ベンダー・ツール』
一般的によく使われるいくつかのエンタープライズ・モニタリング・ソリューションを使用して、WebSphere eXtreme Scale をモニターすることができます。 パブリックにアクセス可能な管理 Bean を使用して WebSphere eXtreme Scale をモニターする IBM Tivoli Monitoring および Hyperic HQ 用に、プラグイン・エージェントが組み込まれています。CA Wily Introscope は Java メソッドのインスツルメンテーションを使用して、統計情報を収集します。

275 ページの『統計の概説』
WebSphere eXtreme Scale での統計は、内部統計ツリーから作成されます。内部ツリーからは、StatsAccessor API、Performance Monitoring Infrastructure (PMI) モジュール、および MBean API が作成されます。

CA Wily Introscope による eXtreme Scale アプリケーションのモニター

CA Wily Introscope は、エンタープライズ・アプリケーション環境のパフォーマンス上の問題を検出して診断する場合に使用できるサード・パーティーの管理製品です。 eXtreme Scale には、CA Wily Introscope を構成して eXtreme Scale ランタイムの選択部分をイントロスペクトし、eXtreme Scale アプリケーションを迅速に表示、検証する場合の詳細が含まれています。 CA Wily Introscope は、スタンドアロン環境と WebSphere Application Server デプロイメント環境の両方で効果的に機能します。

概説

CA Wily Introscope を使用して eXtreme Scale アプリケーションをモニターするには、eXtreme Scale のモニター情報へのアクセスを可能にする設定を ProbeBuilderDirective (PBD) ファイルに作成する必要があります。

重要: Introscope のインスツルメンテーション・ポイントは、各フィックスパックまたはリリースで変わる可能性があります。新しいフィックスパックまたはリリースをインストールしたら、インスツルメンテーション・ポイントに変更がないか、資料を確認してください。

eXtreme Scale アプリケーションをモニターするように、CA Wily Introscope の ProbeBuilderDirective (PBD) ファイルを構成できます。CA Wily Introscope は、アプリケーション管理製品で、これを使用すると複雑な複合 Web アプリケーション環境におけるパフォーマンス上の問題を積極的に検出、選別、および診断することができます。

カタログ・サービスのモニター用の PBD ファイル設定

カタログ・サービスをモニターするには、PBD ファイルの以下の設定を 1 つ以上を使用できます。

```
TraceOneMethodOfClass: com.ibm.ws.objectgrid.hamanager.HAControllerImpl
  changeDefinedCompleted
BlamePointTracerDifferentMethods "OGcatalog|{classname}|{method}"
TraceOneMethodOfClass: com.ibm.ws.objectgrid.hamanager.HAControllerImpl
  viewChangeCompleted
BlamePointTracerDifferentMethods "OGcatalog|{classname}|{method}"
TraceOneMethodOfClass: com.ibm.ws.objectgrid.hamanager.HAControllerImpl
  viewAboutToChange
BlamePointTracerDifferentMethods "OGcatalog|{classname}|{method}"
TraceOneMethodOfClass: com.ibm.ws.objectgrid.container.ServerAgent
  heartbeat
BlamePointTracerDifferentMethods "OGcatalog|{classname}|{method}"
TraceOneMethodOfClass: com.ibm.ws.objectgrid.container.ServerAgent
  heartbeatCluster
BlamePointTracerDifferentMethods "OGcatalog|{classname}|{method}"
TraceOneMethodOfClass: com.ibm.ws.objectgrid.container.ServerAgent
  heartbeatCurrentLeader
BlamePointTracerDifferentMethods "OGcatalog|{classname}|{method}"
TraceOneMethodOfClass: com.ibm.ws.objectgrid.container.ServerAgent
  heartbeatDeadServer
BlamePointTracerDifferentMethods "OGcatalog|{classname}|{method}"
TraceOneMethodOfClass: com.ibm.ws.objectgrid.container.ServerAgent
  heartbeatNewLeader
BlamePointTracerDifferentMethods "OGcatalog|{classname}|{method}"
TraceOneMethodOfClass: com.ibm.ws.objectgrid.container.ServerAgent
  heartbeatNewServer
BlamePointTracerDifferentMethods "OGcatalog|{classname}|{method}"
TraceOneMethodOfClass: com.ibm.ws.objectgrid.catalog.placement.
  PlacementServiceImpl
importRouteInfo BlamePointTracerDifferentMethods
  "OGcatalog|{classname}|{method}"
TraceOneMethodOfClass: com.ibm.ws.objectgrid.catalog.placement.
  PlacementServiceImpl heartbeat
BlamePointTracerDifferentMethods "OGcatalog|{classname}|{method}"
TraceOneMethodOfClass: com.ibm.ws.objectgrid.catalog.placement.
  PlacementServiceImpl joinPlacementGroup
BlamePointTracerDifferentMethods "OGcatalog|{classname}|{method}"
  TraceOneMethodOfClass:
com.ibm.ws.objectgrid.catalog.placement.PlacementServiceImpl
  classifyServer
BlamePointTracerDifferentMethods "OGcatalog|{classname}|{method}"
TraceOneMethodOfClass: com.ibm.ws.objectgrid.catalog.placement.
  BalanceGridEventListener shardActivated
BlamePointTracerDifferentMethods "OGcatalog|{classname}|{method}"
TraceOneMethodOfClass: com.ibm.ws.objectgrid.catalog.placement.
  BalanceGridEventListener shardDeactivate
BlamePointTracerDifferentMethods "OGcatalog|{classname}|{method}"
```

カタログ・サービスのモニター用のクラス

HAControllerImpl

HAControllerImpl クラスは、コア・グループのライフサイクル・イベントおよびフィードバック・イベントを処理します。このクラスをモニターすると、コア・グループの構造および変更を確認できます。

ServerAgent

ServerAgent クラスは、コア・グループ・イベントとカタログ・サービスの

通信を担当します。さまざまなハートビート呼び出しをモニターして、主要なイベントを見極めることができます。

PlacementServiceImpl

PlacementServiceImpl クラスは、コンテナーを調整します。このクラスのメソッドは、サーバーの結合イベントおよび配置イベントをモニターするために使用できます。

BalanceGridEventListener

BalanceGridEventListener クラスは、カタログのリーダーシップを制御します。このクラスをモニターすると、現在リーダーとして実行中のカタログ・サービスを確認できます。

コンテナー・モニター用の PBD ファイル設定

コンテナーをモニターするには、PBD ファイルの以下の設定を 1 つ以上使用できます。

```
TraceOneMethodOfClass: com.ibm.ws.objectgrid.ShardImpl processMessage
BlamePointTracerDifferentMethods "OGcontainer|{classname}|{method}"
TraceOneMethodOfClass: com.ibm.ws.objectgrid.plugins.
  CommittedLogSequenceListenerProxy applyCommitted
BlamePointTracerDifferentMethods "OGcontainer|{classname}|{method}"
TraceOneMethodOfClass: com.ibm.ws.objectgrid.plugins.
  CommittedLogSequenceListenerProxy sendApplyCommitted
BlamePointTracerDifferentMethods "OGcontainer|{classname}|{method}"
TraceOneMethodOfClass: com.ibm.ws.objectgrid.map.BaseMap evictMapEntries
BlamePointTracerDifferentMethods "OGcontainer|{classname}|{method}"
TraceOneMethodOfClass: com.ibm.ws.objectgrid.checkpoint.
  CheckpointMapImpl$CheckpointIterator activateListener
BlamePointTracerDifferentMethods "OGcontainer|{classname}|{method}"
TraceOneMethodOfClass: com.ibm.ws.objectgrid.hamanager.HAControllerImpl
  changeDefinedCompleted
BlamePointTracerDifferentMethods "OGcontainer|{classname}|{method}"
TraceOneMethodOfClass: com.ibm.ws.objectgrid.hamanager.HAControllerImpl
  viewChangeCompleted
BlamePointTracerDifferentMethods "OGcontainer|{classname}|{method}"
TraceOneMethodOfClass: com.ibm.ws.objectgrid.hamanager.HAControllerImpl
  viewAboutToChange
BlamePointTracerDifferentMethods "OGcontainer|{classname}|{method}"
TraceOneMethodOfClass: com.ibm.ws.objectgrid.container.ServerAgent
  batchProcess
BlamePointTracerDifferentMethods "OGcontainer|{classname}|{method}"
TraceOneMethodOfClass: com.ibm.ws.objectgrid.container.ServerAgent
  heartbeat
BlamePointTracerDifferentMethods "OGcontainer|{classname}|{method}"
TraceOneMethodOfClass: com.ibm.ws.objectgrid.container.ServerAgent
  heartbeatCluster
BlamePointTracerDifferentMethods "OGcontainer|{classname}|{method}"
TraceOneMethodOfClass: com.ibm.ws.objectgrid.container.ServerAgent
  heartbeatCurrentLeader
BlamePointTracerDifferentMethods "OGcontainer|{classname}|{method}"
TraceOneMethodOfClass: com.ibm.ws.objectgrid.container.ServerAgent
  heartbeatDeadServer
BlamePointTracerDifferentMethods "OGcontainer|{classname}|{method}"
TraceOneMethodOfClass: com.ibm.ws.objectgrid.container.ServerAgent
  heartbeatNewLeader
BlamePointTracerDifferentMethods "OGcontainer|{classname}|{method}"
TraceOneMethodOfClass: com.ibm.ws.objectgrid.container.ServerAgent
  heartbeatNewServer
BlamePointTracerDifferentMethods "OGcontainer|{classname}|{method}"
```

コンテナー・モニター用のクラス

ShardImpl

ShardImpl クラスには、processMessage メソッドがあります。
processMessage メソッドは、クライアント要求のためのメソッドです。このメソッドを使用すると、サーバー・サイドの応答時間および要求数を確認できます。すべてのサーバー全体でカウントを監視し、ヒープ使用状況をモニターすることにより、グリッドのバランスが取れているかどうかを判別できます。

CheckpointIterator

CheckpointIterator クラスには、プライマリーをピア・モードにする activateListener メソッド呼び出しがあります。プライマリーがピア・モードになると、メソッド完了後に、レプリカがプライマリーにより更新されます。レプリカがプライマリー全体から再生成される場合、この操作に時間がかかることがあります。この操作が完了するまでは、システムの回復状態は不十分であるため、このクラスを使用してこの操作の進行状況をモニターできます。

CommittedLogSequenceListenerProxy

CommittedLogSequenceListenerProxy クラスには、2 つの興味深いメソッドがあります。applyCommitted メソッドは、すべてのトランザクションで実行され、sendApplyCommitted メソッドは、レプリカが情報をプルしているときに実行されます。これら 2 つのメソッドの実行頻度により、レプリカがプライマリーにどの程度後れを取らずに対応できているかがある程度分かります。

クライアント・モニター用の PBD ファイル設定

クライアントをモニターするには、PBD ファイルの以下の設定を 1 つ以上使用できます。

```
TraceOneMethodOfClass: com.ibm.ws.objectgrid.client.ORBClientCoreMessageHandler
 sendMessage
BlamePointTracerDifferentMethods "OGclient|{classname}|{method}"
TraceOneMethodOfClass: com.ibm.ws.objectgrid.corba.cluster.ClusterStore
  bootstrap
BlamePointTracerDifferentMethods "OGclient|{classname}|{method}"
TraceOneMethodOfClass: com.ibm.ws.objectgrid.corba.cluster.ClusterStore
  epochChangeBootstrap
BlamePointTracerDifferentMethods "OGclient|{classname}|{method}"
TraceOneMethodOfClass: com.ibm.ws.objectgrid.map.BaseMap evictMapEntries
BlamePointTracerDifferentMethods "OGclient|{classname}|{method}"
TraceOneMethodOfClass: com.ibm.ws.objectgrid.cluster.orb.routing.
  SelectionServiceImpl routeFailed
BlamePointTracerDifferentMethods "OGclient|{classname}|{method}"
TraceOneMethodOfClass: com.ibm.ws.objectgrid.cluster.orb.routing.
  SelectionServiceImpl routeFailed
BlamePointTracerDifferentMethods "OGclient|{classname}|{method}"
TraceOneMethodOfClass: com.ibm.ws.objectgrid.SessionImpl getMap
BlamePointTracerDifferentMethods "OGclient|{classname}|{method}"
TraceOneMethodOfClass: com.ibm.ws.objectgrid.ObjectGridImpl getSession
BlamePointTracerDifferentMethods "OGclient|{classname}|{method}"
TurnOn: ObjectMap
SetFlag: ObjectMap
IdentifyClassAs: com.ibm.ws.objectgrid.ObjectMapImpl ObjectMap
TraceComplexMethodsifFlagged: ObjectMap BlamePointTracerDifferentMethods
"OGclient|{classname}|{method}"
```

クライアント・モニター用のクラス

ORBClientCoreMessageHandler

ORBClientCoreMessageHandler クラスは、コンテナーへのアプリケーション要求の送信を担当します。sendMessage メソッドでクライアントの応答時間および要求数をモニターできます。

ClusterStore

ClusterStore クラスには、クライアント・サイドでのルーティング情報が保持されます。

BaseMap

BaseMap クラスには、Evictor がマップからエントリーを除去するときに呼び出される evictMapEntries メソッドがあります。

SelectionServiceImpl

SelectionServiceImpl クラスは、ルーティング上の決定を行います。クライアントによりフェイルオーバーに関する決定が下される場合、このクラスを使用すると、その決定から実行されるアクションを判別できます。

ObjectGridImpl

ObjectGridImpl クラスには、このメソッドに対する要求数を判別するためにモニターできる getSession メソッドがあります。

Hyperic HQ による eXtreme Scale のモニター

Hyperic HQ は、サード・パーティーのモニター・ソリューションで、オープン・ソース・ソリューションあるいはエンタープライズ製品として自由に使用可能です。WebSphere eXtreme Scale には、あるプラグインが含まれていて、このプラグインにより Hyperic HQ エージェントは eXtreme Scale コンテナー・サーバーを検出し、eXtreme Scale 管理 Bean を使用して統計のレポートおよび集約を行うことができます。 Hyperic HQ を使用すると、スタンドアロン eXtreme Scale デプロイメントをモニターできます。

始める前に

- この一連の説明は、Hyperic バージョン 4.0 を対象としています。これよりも新しいバージョンの Hyperic の場合、パス名やエージェントとサーバーの始動方法などの情報について Hyperic 資料を参照してください。
- Hyperic サーバーとエージェントのインストールをダウンロードします。サーバーのインストール済み環境が 1 つ実行中である必要があります。 eXtreme Scale サーバーのすべてを検出するには、eXtreme Scale サーバーが稼働している各マシン上で Hyperic エージェントが実行中である必要があります。ダウンロード情報および資料のサポートは、Hyperic Web サイトを参照してください。
- `objectgrid-plugin.xml` および `hqplugin.jar` ファイルにアクセスする必要があります。これらのファイルは、`objectgridRoot/hyperic/etc` ディレクトリーにあります。

このタスクについて

eXtreme Scale を Hyperic HQ モニター・ソフトウェアと統合することで、ご使用の環境のパフォーマンスに関するメトリックをグラフィカルにモニターおよび表示することができます。この統合をセットアップするには、各エージェントでプラグインの実装を使用します。

1. eXtreme Scale サーバーを始動します。 Hyperic プラグインは、eXtreme Scale を実行している Java 仮想マシンに接続するローカル・プロセスを調べます。Java 仮想マシンに適切に接続する場合、各サーバーは **-jmxServicePort** オプションを指定して開始する必要があります。また、サーバー・プロパティー・ファイルの統計を使用可能に設定する必要もあります。このフィルターではカタログ・サーバーは検出されません。
   - **-jmxServicePort** オプションを指定したサーバーの始動に関して詳しくは、244 ページの『startOgServer スクリプト』を参照してください。
   - サーバー・プロパティー・ファイルの統計の使用可能化に関して詳しくは、203 ページの『サーバー・プロパティー・ファイル』の **statsSpec** プロパティーに関する情報を参照してください。

2. ご使用の Hyperic の構成で、extremescale-plugin.xml ファイルと wxshyperic.jar ファイルを適切なサーバーとエージェントのプラグイン・ディレクトリーに置きます。 Hyperic と統合するためには、エージェント・インストールとサーバー・インストールの両方でプラグインおよび Java アーカイブ (JAR) ファイルにアクセスする必要があります。サーバーは構成を動的にスワップできますが、いずれのエージェントを開始する場合もその前に統合を完了しておく必要があります。
   a. extremescale-plugin.xml ファイルを、次のロケーションにあるサーバーの plugin ディレクトリーに置きます。

      hyperic_home/server_home/hq-engine/server/default/deploy/hq.ear/hq-plugins

   b. extremescale-plugin.xml ファイルを、次のロケーションにあるエージェントの plugin ディレクトリーに置きます。

      agent_home/bundles/gent-4.0.2-939/pdk/plugins

   c. wshyperic.jar ファイルを次のロケーションにあるエージェントの lib ディレクトリーに置きます。

      agent_home/bundles/gent-4.0.2-939/pdk/lib

3. エージェントを構成します。 agent.properties ファイルは、エージェント・ランタイムの構成ポイントとして機能します。このプロパティーは、*agent_home*/conf ディレクトリーにあります。以下のキーはオプションですが、eXtreme Scale プラグインに対しては重要です。
   - autoinventory.defaultScan.interval.millis=<*time_in_milliseconds*>

     エージェント・ディスカバリーの間隔をミリ秒単位に設定します。
   - log4j.logger.org.hyperic.hq.plugin.extremescale.XSServerDetector=DEBUG

     : eXtreme Scale プラグインからの詳細のデバッグ・ステートメントを有効にします。
   - username=<*username*>: セキュリティーが有効に設定されている場合に Java Management Extensions (JMX) ユーザー名を設定します。
   - password=<*password*>: セキュリティーが有効に設定されている場合に、JMX パスワードを設定します。
   - sslEnabled=<*true*|*false*>: プラグインに対して、Secure Sockets Layer (SSL) を使用するかどうかを指示します。デフォルトではこの値は false です。
   - trustPath=<*path*>: SSL 接続のトラスト・パスを設定します。
   - trustType=<*type*>: SSL 接続のトラスト・タイプを設定します。
   - trustPass=<*password*>: SSL 接続のトラスト・パスワードを設定します。
4. エージェント・ディスカバリーを開始します。 Hyperic エージェントはディスカバリーおよびメトリック情報をサーバーに送ります。サーバーを使用してデータ・ビューをカスタマイズし、論理インベントリー・オブジェクトをグループ化して有用な情報を生成します。サーバーが使用可能になったならば、エージェントの起動スクリプトを実行するか、または Windows サービスを開始する必要があります。
   - Linux agent_home/bin/hq-agent.sh start

- Windows Windows サービスを使用してエージェントを開始します。

エージェントの始動後に、サーバーが検出されて、グループが構成されます。サーバー・コンソールにログインし、サーバーのインベントリー・データベースに追加するリソースを選択することができます。サーバー・コンソールは、デフォルトで URL: `http://<server_host_name>:7080/` にあります。

5. Hyperic コンソールでサーバーをモニターします。 サーバーがインベントリー・モデルに追加されると、そのサービスはもはや必要なくなります。
   - **ダッシュボード・ビュー**: リソース検出イベントを調べたとき、メイン・ダッシュボード・ビューにログインしました。ダッシュボード・ビューは、カスタマイズ可能なメッセージ・センターとなる汎用ビューです。このメイン・ダッシュボードにグラフやインベントリー・オブジェクトをエクスポートすることができます。
   - **リソース・ビュー**: このページからインベントリー・モデル全体を照会して調べることができます。サービスが追加されたならば、サーバー・セクションの下で正しくラベル付けされて一緒にリストされたすべての eXtreme Scale サーバーを確認することができます。個々のサーバーをクリックすると、基本メトリックを参照できます。
6. 「リソース・ビュー」のページでサーバー全体のインベントリーを表示できます。 このページで、複数の ObjectGrid サーバーを選択し、それらをグループにまとめることができます。リソースのセットをグループ化すると、共通のメトリックがグラフ化されて、グループ・メンバー間のオーバーレイや相違点が表示されます。オーバーレイを表示するには、ご使用のサーバー・グループの画面でメトリックを選択します。そうすると、メトリックが図表域に表示されます。すべてのグループ・メンバーのオーバーレイを表示するには、下線の付いたメトリック名をクリックします。「**ツール**」メニューを使用して、必要なグラフ、ノード・ビュー、および比較オーバーレイをメイン・ダッシュボードにエクスポートすることができます。

関連概念

303 ページの『ベンダー・ツール』
一般的によく使われるいくつかのエンタープライズ・モニタリング・ソリューションを使用して、WebSphere eXtreme Scale をモニターすることができます。 パブリックにアクセス可能な管理 Bean を使用して WebSphere eXtreme Scale をモニターする IBM Tivoli Monitoring および Hyperic HQ 用に、プラグイン・エージェントが組み込まれています。CA Wily Introscope は Java メソッドのインスツルメンテーションを使用して、統計情報を収集します。

275 ページの『統計の概説』
WebSphere eXtreme Scale での統計は、内部統計ツリーから作成されます。内部ツリーからは、StatsAccessor API、Performance Monitoring Infrastructure (PMI) モジュール、および MBean API が作成されます。

ログおよびトレース

ログおよびトレースを使用して、環境のモニターおよびトラブルシューティングを実行できます。ログは、構成によってさまざまなロケーションにあります。IBM サポートに協力を依頼する場合、サーバーに関するトレースを提供する必要がある場合があります。

WebSphere Application Server によるロギング

詳しくは、WebSphere Application Server インフォメーション・センターを参照してください。

スタンドアロン環境での WebSphere eXtreme Scale によるロギング

スタンドアロン・カタログおよびコンテナー・サービスを使用して、ログのロケーションおよびトレース仕様を設定します。カタログ・サーバー・ログは、サーバー始動コマンドを実行したロケーションにあります。

コンテナー・サーバーのログ・ロケーションの設定

デフォルトでは、コンテナーのログは、サーバー・コマンドが実行されたディレクトリーにあります。*<eXtremeScale_home>*/bin ディレクトリーでサーバーを始動する場合、ログおよびトレース・ファイルは bin ディレクトリーの logs/*<server_name>* ディレクトリー内にあります。コンテナー・サーバー・ログの代替ロケーションを指定するには、以下のコンテンツを使用して server.properties ファイルなどのプロパティー・ファイルを作成します。

```
workingDirectory=<directory>
traceSpec=
systemStreamToFileEnabled=true
```

workingDirectory プロパティーは、ログおよびオプションのトレース・ファイルのルート・ディレクトリーです。WebSphere eXtreme Scale は、traceSpec オプションでトレースが使用可能になっていると、SystemOut.log ファイル、SystemErr.log ファイル、およびトレース・ファイルを使用して、コンテナー・サーバーの名前を持つディレクトリーを作成します。コンテナー開始中にプロパティー・ファイルを使用するには、**-serverProps** オプションを使用して、サーバー・プロパティー・ファイルのロケーションを指定します。

詳しくは、 236 ページの『スタンドアロン WebSphere eXtreme Scale サーバーの始動』および 244 ページの『startOgServer スクリプト』を参照してください。

SystemOut.log ファイル内で参照する共通の情報メッセージは、開始確認メッセージです。特定のメッセージについて詳しくは、 352 ページの『メッセージ』を参照してください。

WebSphere Application Server によるトレース

詳しくは、WebSphere Application Server インフォメーション・センターを参照してください。

カタログ・サービスでのトレース

カタログ・サービスの始動中に、 **-traceSpec** および **-traceFile** パラメーターを使用して、カタログ・サービスでトレースを設定できます。以下に例を示します。

```
startOgServer.sh catalogServer -traceSpec
ObjectGridPlacement=all=enabled -traceFile
/home/user1/logs/trace.log
```

<eXtremeScale_home>/bin ディレクトリーでカタログ・サービスを始動する場合、ログおよびトレース・ファイルは bin ディレクトリーの logs/

<catalog_service_name> ディレクトリー内にあります。 カタログ・サービスの開始に関して詳しくは、237 ページの『スタンドアロン環境でのカタログ・サービスの開始』を参照してください。

スタンドアロン・コンテナー・サーバーでのトレース

コンテナー・サーバーでトレースを使用可能にするには 2 つの方法があります。ログ・セクションでの説明どおりに、サーバー・プロパティー・ファイルを作成するか、始動時にコマンド行を使用してトレースを使用可能にすることができます。サーバー・プロパティー・ファイルを使用してコンテナー・トレースを使用可能にするには、必要なトレース仕様で **traceSpec** プロパティーを更新します。始動パラメーターを使用して、コンテナー・トレースを使用可能にするには、**-traceSpec** および **-traceFile** パラメーターを使用します。以下に例を示します。

```
startOgServer.sh c0 -objectGridFile ../xml/myObjectGrid.xml
-deploymentPolicyFile ../xml/myDepPolicy.xml -catalogServiceEndpoints
server1.rchland.ibm.com:2809 -traceSpec
ObjectGridPlacement=all=enabled -traceFile /home/user1/logs/trace.log
```

<eXtremeScale_home>/bin ディレクトリーでサーバーを始動する場合、ログおよびトレース・ファイルは bin ディレクトリーの logs/*<server_name>* ディレクトリー内にあります。 コンテナー・プロセスの開始について詳しくは、240 ページの『コンテナー・プロセスの開始』を参照してください。

ObjectGridManager インターフェースによるトレース

別の方法として、ObjectGridManager インターフェースで実行時にトレースを設定する方法があります。 ObjectGridManager インターフェースでのトレース設定を使用すると、eXtreme Scale に接続してトランザクションをコミットしている間に eXtreme Scale クライアント上でトレースを取得することができます。ObjectGridManager インターフェースでトレースを設定するには、トレース仕様およびトレース・ログを指定します。

```
ObjectGridManager manager = ObjectGridManagerFactory.getObjectGridManager();
...
manager.setTraceEnabled(true);
manager.setTraceFileName("logs/myClient.log");
manager.setTraceSpecification("ObjectGridReplication=all=enabled");
```

xsadmin ユーティリティーでトレースを使用可能にする

xsadmin ユーティリティーを使用してトレースを使用可能にする場合、**setTraceSpec** オプションを使用します。 xsadmin ユーティリティーを使用して、開始時ではなく実行時にスタンドアロン環境でトレースを使用可能にすることができます。すべてのサーバーおよびカタログ・サービスに対してトレースを使用可能にすることができます。あるいは、ObjectGrid 名などでサーバーをフィルタリングすることもできます。例えば、カタログ・サービス・サーバーへのアクセスを使用して ObjectGridReplication トレースを使用可能にするには、以下を実行します。

```
<eXtremeScale_home>/bin>xsadmin.bat -setTraceSpec "ObjectGridReplication=all=enabled"
```

トレース仕様を *=all=disabled に設定することでトレースを使用不可にすることもできます。

詳しくは、300 ページの『xsAdmin サンプル・ユーティリティーの使用』を参照してください。

ffdc ディレクトリーおよびファイル

FFDC ファイルは、IBM サポートがデバッグの補助とするファイルです。これらのファイルは、問題が生じた場合に IBM サポートによって要求される場合があります。

これらのファイルは、ffdc という名前のディレクトリーに存在し、以下のファイルに類似したファイルが含まれています。

```
server2_exception.log
server2_20802080_07.03.05_10.52.18_0.txt
```

関連概念

342 ページの『セキュア eXtreme Scale サーバーの開始と停止』
デプロイメント環境ではサーバーは保護される必要があることが多く、そのためには特別な構成が必要です。

関連タスク

236 ページの『スタンドアロン WebSphere eXtreme Scale サーバーの始動』
スタンドアロン WebSphere eXtreme Scale 構成を実行しているとき、環境はカタログ・サーバー、コンテナー・サーバー、および eXtreme Scale クライアント・プロセスで構成されています。これらのプロセスは手動で構成して開始する必要があります。

237 ページの『スタンドアロン環境でのカタログ・サービスの開始』
WebSphere Application Server が実行されていない分散 WebSphere eXtreme Scale 環境を使用している場合は、カタログ・サービスを手動で開始する必要があります。

240 ページの『コンテナー・プロセスの開始』
eXtreme Scale は、デプロイメント・トポロジー、または server.properties ファイルを使用して、コマンド行から開始できます。

トレース・オプション

トレースを使用可能にすることで、ご使用の環境に関する情報を IBM サポートに提供することができます。

トレースについて

WebSphere eXtreme Scale のトレースは、いくつかの異なるコンポーネントに分けられます。 WebSphere Application Server のトレースと同様、使用するトレース・レベルを指定することができます。一般的なトレースのレベルには、all、debug、entryExit、および event があります。

トレース・ストリングの例は、以下のとおりです。

```
ObjectGridComponent=level=enabled
```

トレース・ストリングは連結することができます。* (アスタリスク) 記号を使用してワイルドカード値を指定します (例: ObjectGrid*=all=enabled)。トレースを IBM サポートに提供する必要がある場合は、特定のトレース・ストリングが要求されます。 例えば、複製に関する問題が発生した場合には、ObjectGridReplication=debug=enabled トレース・ストリングが要求される可能性があります。

トレース仕様

ObjectGrid
汎用・コア・キャッシュ・エンジン。

ObjectGridCatalogServer
汎用カタログ・サービス。

ObjectGridChannel
静的デプロイメント・トポロジー通信。

ObjectgridCORBA
動的デプロイメント・トポロジー通信。

ObjectGridDataGrid
AgentManager API。

ObjectGridDynaCache
WebSphere eXtreme Scale 動的キャッシュ・プロバイダー

ObjectGridEntityManager
EntityManager API。Projector オプションとともに使用。

ObjectGridEvictors
ObjectGrid 組み込み Evictor。

ObjectGridJPA
Java Persistence API (JPA) ローダー

ObjectGridJPACache
JPA キャッシュ・プラグイン

ObjectGridLocking
ObjectGrid キャッシュ・エントリー・ロック・マネージャー。

ObjectGridMBean
管理 Bean

ObjectGridPlacement
カタログ・サーバー断片配置サービス。

ObjectGridQuery
ObjectGrid 照会。

ObjectGridReplication
レプリケーション・サービス。

ObjectGridRouting
クライアント/サーバー・ルーティングの詳細。

ObjectGridSecurity
セキュリティー・トレース。

ObjectGridStats
ObjectGrid 統計。

ObjectGridStreamQuery
ストリーム照会 API。

ObjectGridWriteBehind
ObjectGrid 後書き。

Projector
EntityManager API 内のエンジン。

QueryEngine
オブジェクト照会 API および EntityManager 照会 API のための照会エンジン。

QueryEnginePlan
照会計画診断。

第 9 章 デプロイメント環境の保護

WebSphere eXtreme Scale データを保護するために、eXtreme Scale は、外部のセキュリティー・プロバイダーと統合することができます。

WebSphere eXtreme Scale は、外部のセキュリティー実装と統合できます。この外部実装は、eXtreme Scale に認証サービスおよび許可サービスを提供する必要があります。eXtreme Scale には、セキュリティー実装と統合するためのプラグイン・ポイントがあります。eXtreme Scale は、以下のコンポーネントと正常に統合されています。

- Lightweight Directory Access Protocol (LDAP)
- Kerberos
- ObjectGrid セキュリティー
- Tivoli Access Manager
- Java 認証・承認サービス (JAAS)

eXtreme Scale は、以下のタスクにセキュリティー・プロバイダーを使用します。

- クライアントをサーバーに認証する。
- クライアントに対して、特定の eXtreme Scale 成果物へアクセスする権限、または eXtreme Scale 成果物に対して行うことができる操作を指定する権限を与える。

eXtreme Scale には、以下のタイプの許可があります。

マップ許可
: クライアントまたはグループに、マップに対する挿入、読み取り、更新、排除、および削除の操作を許可することができます。

ObjectGrid 許可
: クライアントまたはグループに、オブジェクト・グリッドでオブジェクト照会またはエンティティー照会を実行する許可を与えることができます。

DataGrid エージェント許可
: クライアントまたはグループに、DataGrid エージェントの ObjectGrid へのデプロイを許可することができます。

サーバー・サイド・マップ許可
: クライアントまたはグループに、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できます。

管理許可
: クライアントまたはグループに、管理タスク実行の許可を与えることができます。

注: バックエンドに対して既にセキュリティーを有効にしている場合、こうしたセキュリティー設定ではもはや十分にデータを保護できないことに注意してください。データベースまたは他のデータ・ストアのセキュリティー設定は、決してキャ

ッシュに転送されません。認証、許可、トランスポートのレベルのセキュリティーなど、eXtreme Scale のセキュリティー・メカニズムを使用して、現在キャッシュされているデータを個別に保護する必要があります。

グリッド・セキュリティー

WebSphere eXtreme Scale グリッド・セキュリティーによって、結合サーバーのクレデンシャルが適切であることが保証されるため、悪意のあるサーバーはグリッドを結合することができません。この目的のために、共有秘密ストリング・メカニズムが使用されています。

カタログ・サーバーを含むすべての WebSphere eXtreme Scale サーバーが、共有秘密ストリングと一致しています。サーバーがグリッドに結合する場合、秘密ストリングの表示を求められます。結合サーバーの秘密ストリングがプレジデント・サーバーまたはカタログ・サーバーのいずれかの秘密ストリングと一致する場合は、結合サーバーは受け入れられます。ストリングが一致しない場合、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。WebSphere eXtreme Scale セキュリティー・インフラストラクチャーには、セキュア・トークン・マネージャー・プラグインが用意されており、サーバーはこの機密事項を送信する前にセキュアにできます。セキュア操作の実装方法を決定する必要があります。WebSphere eXtreme Scale は、すぐに使用可能な実装を提供し、これによりセキュア操作が実装され、機密事項が暗号化されて署名が行われます。

秘密ストリングは `server.properties` ファイルに設定されます。authenticationSecret プロパティーについての詳細は、203 ページの『サーバー・プロパティー・ファイル』 を参照してください。

SecureTokenManager プラグイン

セキュア・トークン・マネージャー・プラグインは、com.ibm.websphere.objectgrid.security.plugins.SecureTokenManager インターフェースによって表されます。

SecureTokenManager プラグインについて詳しくは、SecureTokenManager API 資料を参照してください。

generateToken(Object) メソッドはオブジェクトを取得し、外部に識別されないトークンを生成します。verifyTokens(byte[]) メソッドは逆に、トークンを元のオブジェクトに変換して戻します。

単純な SecureTokenManager 実装は、排他 OR (XOR) アルゴリズムなどの単純なエンコード・アルゴリズムを使用して、シリアライズ形式でオブジェクトをエンコードし、対応するデコード・アルゴリズムを使用してトークンをデコードします。この実装は保護されません。

WebSphere eXtreme Scale には、このインターフェースに対してすぐに使用可能な実装が用意されています。

デフォルトの実装では鍵ペアを使用して、署名し、シグニチャーを検査します。また、秘密鍵を使用して、コンテンツを暗号化します。すべてのサーバーには JCKES タイプの鍵ストアが備えられており、鍵ペア、秘密鍵と公開鍵、および秘密鍵が保管されています。鍵ストアは、秘密鍵を保管する JCKES タイプである必要があります。

これらの鍵は、送信側で秘密ストリングを暗号化し、署名または検証する場合に使用されます。また、トークンは有効期限の時間に関連付けられています。受信側で、データの検証、暗号化解除、および受信側の秘密ストリングとの比較が行われます。サーバーのペアの間での認証には、Secure Sockets Layer (SSL) 通信プロトコルは必要ありません。これは、秘密鍵と公開鍵の目的が同じであるためです。ただし、サーバー通信が暗号化されていない場合は、通信時に侵入者にデータを盗まれる可能性があります。トークンの有効期限が近いため、リプレイ・アタックの危険性は少なくなっています。この可能性は、すべてのサーバーをファイアウォールの後ろにデプロイすると、非常に小さくなります。

この方法の欠点は、WebSphere eXtreme Scale 管理者が鍵を生成し、生成した鍵をすべてのサーバーに移送する必要があるため、移送中にセキュリティー・ブリーチが発生する可能性があることです。

構成

セキュア・トークン・マネージャーの構成に使用するプロパティーについて詳しくは、サーバー・プロパティーを参照してください。

ローカル・セキュリティーの使用可能化

WebSphere eXtreme Scale によりいくつかのセキュリティー・エンドポイントが提供され、カスタム・メカニズムを統合できるようになります。ローカル・プログラミング・モデルにおける主なセキュリティー機能は許可で、認証サポートはありません。既存の WebSphere Application Server 認証とは別個に認証を行う必要があります。ただし、Subject オブジェクトを取得および検証するプラグインは備えられています。

以下のセクションでは、ローカル・セキュリティーを使用可能にする 2 とおりの方法について説明します。

ObjectGrid XML ファイルを使用して ObjectGrid を定義し、その ObjectGrid に対するセキュリティーを使用可能にできます。ObjectGridSample エンタープライズ・アプリケーションの例で使用される `secure-objectgrid-definition.xml` ファイルを以下の例に示します。セキュリティーを使用可能にするには、securityEnabled attribute 属性を `true` に設定します。

```
<objectGrids>
    <objectGrid name="secureClusterObjectGrid" securityEnabled="true"
        authorizationMechanism="AUTHORIZATION_MECHANISM_JAAS">
    ...
</objectGrids>
```

セキュリティーをプログラマチックに使用可能にすることもできます。ObjectGrid.setSecurityEnabled メソッドを使用して ObjectGrid を作成するには、ObjectGrid インターフェース上で以下のメソッドを呼び出します。

```
/**
 * Enable the ObjectGrid security
 */
void setSecurityEnabled();
```

詳しくは、API 資料を参照してください。

アプリケーション・クライアントの認証

アプリケーション・クライアントの認証は、クライアント/サーバー・セキュリティーおよびクレデンシャル認証の使用可能化と、オーセンティケーターおよびシステム・クレデンシャル生成プログラムの構成からなります。

クライアント/サーバー・セキュリティーの使用可能化

ObjectGrid による認証を正常に行うには、クライアントとサーバーの両方でセキュリティーを使用可能にする必要があります。

クライアント・セキュリティーの使用可能化

WebSphere eXtreme Scale は、クライアント・プロパティー・サンプル・ファイル (sampleClient.properties ファイル) を WebSphere Extended Deployment インストール済み環境では *WAS_HOME*/optionalLibraries/ObjectGrid/properties ディレクトリー内、混合サーバー・インストール済み環境では /ObjectGrid/properties ディレクトリー内に提供しています。このテンプレート・ファイルを、適切な値で変更することができます。objectgridClient.properties ファイル内の securityEnabled プロパティーを true に設定してください。securityEnabled プロパティーは、セキュリティーが使用可能かどうかを示します。クライアントがサーバーに接続されている場合、クライアント・サイドとサーバー・サイドのこの値は、両方とも true か、両方とも false である必要があります。例えば、接続されているサーバーのセキュリティーが使用可能な場合、クライアントがサーバーに接続するには、このプロパティー値をクライアント・サイドで true に設定する必要があります。

com.ibm.websphere.objectgrid.security.config.ClientSecurityConfiguration インターフェースは、security.ogclient.props ファイルを表しています。com.ibm.websphere.objectgrid.security.config. ClientSecurityConfigurationFactory public API を使用して、このインターフェースのインスタンスをデフォルト値で作成することができます。または ObjectGrid クライアント・セキュリティー・プロパティー・ファイルを渡して、インスタンスを作成することもできます。security.ogclient.props ファイルには、その他のプロパティーが含まれています。詳しくは、ClientSecurityConfiguration API 資料および ClientSecurityConfigurationFactory API 資料を参照してください。

サーバー・セキュリティーの使用可能化

サーバー・サイドでセキュリティーを使用可能にするには、security.xml ファイル内の securityEnabled プロパティーを true に設定します。セキュリティー記述子 XML ファイルを使用してグリッド・セキュリティー構成を指定し、グリッド全体のセキュリティー構成を非セキュリティー構成から分離します。

クレデンシャル認証の使用可能化

eXtreme Scale クライアントが CredentialGenerator オブジェクトを使用して Credential オブジェクトを取得すると、この Credential オブジェクトがクライアント要求とともに eXtreme Scale サーバーに送信されます。サーバーは、要求の処理前に Credential オブジェクトの認証を行います。 Credential オブジェクトが正常に認証されると、この Credential オブジェクトを表す Subject オブジェクトが戻されます。その後、この Subject オブジェクトは要求の認証に使用されます。

クライアントおよびサーバーのプロパティー・ファイルで credentialAuthentication を設定して、クレデンシャル認証を使用可能にします。詳しくは、210 ページの『クライアント・プロパティー・ファイル』および 203 ページの『サーバー・プロパティー・ファイル』を参照してください。

以下の 2 つの表に、さまざまな設定で、いずれの認証メカニズムが使用されるかを示します。

表 9. クライアントおよびサーバーの設定におけるクレデンシャル認証

クライアント・クレデンシャル認証	サーバー・クレデンシャル認証	結果
なし	常になし	使用不可
なし	サポートされる	使用不可
なし	必須	Error case
サポートされる	常になし	使用不可
サポートされる	サポートされる	使用可能
サポートされる	必須	使用可能
必須	常になし	Error case
必須	サポートされる	使用可能
必須	必須	使用可能

オーセンティケーターの構成

eXtreme Scale サーバーは、Authenticator プラグインを使用して、Credential オブジェクトの認証を行います。 Authenticator インターフェースの実装では、Credential オブジェクトを取得し、Lightweight Directory Access Protocol (LDAP) サーバーなどのユーザー・レジストリーに対してこのオブジェクトを認証します。eXtreme Scale は、レジストリー構成を提供しません。ユーザー・レジストリーへの接続およびユーザー・レジストリーに対する認証は、このプラグインで実装する必要があります。

例えば、1 つのオーセンティケーター実装では、クレデンシャルからユーザー ID とパスワードが抽出され、このユーザー ID とパスワードを使用して、LDAP サーバーに対する接続と検証が行われます。認証の結果として、Subject オブジェクトが作成されます。この実装では、Java 認証・承認サービス (JAAS) ログイン・モジュールを使用できます。認証の結果として、Subject オブジェクトが戻されます。

以下の例のように、セキュリティー記述子 XML ファイルでオーセンティケーターを構成できます。

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

 <security securityEnabled="true"
        loginSessionExpirationTime="300">

  <authenticator className ="com.ibm.websphere.objectgrid.security.plugins.builtins.KeyStoreLoginAuthenticator">
        </authenticator>

 </security>

</securityConfig>
```

セキュア・サーバーを開始して、セキュリティー XML ファイルを設定する場合は、–clusterSecurityFile オプションを使用します。詳しくは、*製品概要*の Java SE セキュリティーのチュートリアルを参照してください。

システム・クレデンシャル生成プログラムの構成

このシステム・クレデンシャル生成プログラムは、システム・クレデンシャルのファクトリーを表すために使用されます。システム・クレデンシャルは、管理者クレデンシャルに似ています。以下の例のように、カタログ・セキュリティー XML 内で SystemCredentialGenerator 要素を構成できます。

```
<systemCredentialGenerator className ="com.ibm.websphere.objectgrid.security.plugins.builtins.UserPasswordCredentialGenerator">
          <property name="properties" type="java.lang.String" value="manager manager1" description="username password" />
    </systemCredentialGenerator>
```

デモンストレーション用のため、ユーザー名およびパスワードは平文で保管されます。実稼働環境では、ユーザー名およびパスワードは平文で保管しないでください。

WebSphere eXtreme Scale が提供するデフォルトのシステム・クレデンシャル生成プログラムは、サーバー・クレデンシャルを使用します。システム・クレデンシャル生成プログラムを明示的に指定しないと、このデフォルトのシステム・クレデンシャル生成プログラムが使用されます。

アプリケーション・クライアントの許可

アプリケーション・クライアントの許可は、ObjectGrid 許可クラス、許可メカニズム、許可検査期間、および作成者限定アクセス許可から構成されます。

eXtreme Scale の許可は Subject オブジェクトおよびアクセス権に基づいています。本製品は、2 種類の許可メカニズム、つまり、Java 認証・承認サービス (JAAS) とカスタム許可をサポートしています。

ObjectGrid 許可クラス

許可はアクセス権に基づいています。許可クラスには次の 4 つの異なるタイプがあります。

- MapPermission: このクラスは、ObjectGrid マップ内のデータへのアクセスの許可を表します。
- ObjectGridPermission: このクラスは、ObjectGrid へのアクセスの許可を表します。
- ServerMapPermission: このクラスは、クライアントからサーバー・サイドの ObjectGrid マップへのアクセスの許可を表します。

- AgentPermission: このクラスは、サーバー・サイドのエージェントを開始する許可を表します。

API および関連許可について詳しくは、*プログラミング・ガイド*のクライアント許可プログラミングに関するトピックを参照してください。

許可検査期間

eXtreme Scale は、パフォーマンス上の理由で、マップ許可検査結果のキャッシングをサポートしています。このメカニズムがないと、メソッドおよびそれらに必要な許可のリストにあるメソッドが呼び出されたときに、ランタイムは、構成された許可メカニズムを呼び出してアクセスを許可します。この許可検査期間が設定されていると、許可メカニズムは、許可検査期間に基づいて定期的に呼び出されます。

アクセス権の許可情報は Subject オブジェクトに基づいています。クライアントがメソッドにアクセスしようとすると、eXtreme Scale ランタイムは、Subject オブジェクトに基づいてキャッシュ内を検索します。キャッシュ内でオブジェクトが見つからない場合、ランタイムは、この Subject オブジェクトに付与されている許可を確認し、この許可をキャッシュに格納します。

許可検査期間は、ObjectGrid が初期化される前に定義しておく必要があります。許可検査期間は、以下の 2 とおりの方法で構成できます。

ObjectGrid XML ファイルを使用して ObjectGrid を定義し、許可検査期間を設定できます。以下の例では、許可検査期間が 45 秒に設定されています。

```
<objectGrids>
 <objectGrid name="secureClusterObjectGrid" securityEnabled="true"
 authorizationMechanism="AUTHORIZATION_MECHANISM_JAAS"
 permissionCheckPeriod="45">
  <bean id="bean id="TransactionCallback"
className="com.ibm.websphere.samples.objectgrid.HeapTransactionCallback" />
...
</objectGrids>
```

API を使用して ObjectGrid を作成する場合、以下のメソッドを呼び出して、許可検査期間を設定します。このメソッドは、ObjectGrid インスタンスを初期化する前にのみ呼び出すことができます。このメソッドは、ObjectGrid を直接インスタンス化する場合のローカル eXtreme Scale プログラミング・モデルにのみ適用されます。

```
/**
 * This method takes a single parameter indicating how often you
 * want to check the permission used to allow a client access. If the
 * parameter is 0 then every single get/put/update/remove/evict call
 * asks the authorization mechanism, either JAAS authorization or custom
 * authorization, to check if the current subject has permission. This might be
 * prohibitively expensive from a performance point of view depending on
 * the authorization implementation, but if you need to have ever call check the
 * authorization mechanism, then set the parameter to 0.
 * Alternatively, if the parameter is > 0 then it indicates the number
 * of seconds to cache a set of permissions before returning to
 * the authorization mechanism to refresh them. This value provides much
 * better performance, but if the back-end
 * permissions are changed during this time then the ObjectGrid can
 * allow or prevent access even though the back-end security
 * provider was modified.
 *
 * @param period the permission check period in seconds.
 */
void setPermissionCheckPeriod(int period);
```

作成者限定アクセス許可

作成者限定アクセス許可を使用すると、エントリーを ObjectGrid マップに挿入したユーザーのみ (関連付けられた Principal オブジェクトによって表される) が、そのエントリーにアクセス (read、update、invalidate、および remove) できます。

既存の ObjectGrid マップの許可モデルは、アクセス・タイプに基づいていて、データ・エントリーには基づいていません。すなわち、ユーザーは、
read、write、insert、delete、または invalidate などの特定のアクセス・タイプをマップ内のすべてのデータに対して保持しているか、またはどのデータに対しても保持していないかのいずれかです。しかし、eXtreme Scale は、個別のデータ・エントリーに対するユーザーの許可は行いません。この機能は、データ・エントリーに対してユーザーを許可するための新しい方法を提供します。

さまざまなユーザーが異なるデータのセットにアクセスするようなシナリオでは、このモデルが便利です。ユーザーがデータを永続ストアから ObjectGrid マップにロードするときに、永続ストアによってアクセスを許可できます。このケースでは、ObjectGrid マップ層で別の許可を実行する必要がありません。必要な処理は、作成者限定アクセスの機能を使用可能にして、データをマップにロードするユーザーが、確実にそのデータにアクセスできるようにするのみです。

作成者限定アクセスには、次の 3 つのモードがあります。

disabled
: 作成者限定アクセス機能は使用不可になっています。

complement
: 作成者限定アクセス機能が使用可能になり、マップ許可を補完します。すなわち、マップ許可と作成者限定アクセスの機能の両方が有効になります。結果、データに対する操作をさらに制限することができます。例えば、作成者はデータを無効化できないようにすることができます。

supersede
: 作成者限定アクセス機能が使用可能になり、マップ許可を置き換えます。すなわち、作成者限定アクセス機能がマップ許可に取って代わり、マップ許可は実行されなくなります。

作成者限定アクセス・モードは、次の 2 とおりの方法で構成できます。

以下の例のように、ObjectGrid XML ファイルを使用して ObjectGrid を定義し、作成者限定アクセス・モードを `disabled`、`complement`、または `supersede` のいずれかに設定できます。

```
<objectGrids>
    <objectGrid name="secureClusterObjectGrid" securityEnabled="true"
         accessByCreatorOnlyMode="supersede"
        <bean id="TransactionCallback"
              classname="com.ibm.websphere.samples.objectgrid.HeapTransactionCallback" />
    ...
</objectGrids>
```

ObjectGrid をプログラマチックに作成する場合、以下のメソッドを呼び出して作成者限定アクセス・モードを設定できます。このメソッドの呼び出しは、直接 ObjectGrid インスタンスを生成する場合のローカル eXtreme Scale プログラミング・モデルにのみ適用されます。

```
/**
 * Set the "access by creator only" mode.
 * Enabling "access by creator only" mode ensures that only the user (represented
 * by the Principals associated with it), who inserts the record into the map,
 * can access (read, update, invalidate, and remove) the record.
 * The "access by creator only" mode can be disabled, or can complement the
 * ObjectGrid authorization model, or it can supersede the ObjectGrid
 * authorization model. The default value is disabled:
 * {@link SecurityConstants#ACCESS_BY_CREATOR_ONLY_DISABLED}.
 * @see SecurityConstants#ACCESS_BY_CREATOR_ONLY_DISABLED
 * @see SecurityConstants#ACCESS_BY_CREATOR_ONLY_COMPLEMENT
 * @see SecurityConstants#ACCESS_BY_CREATOR_ONLY_SUPERSEDE
 *
 * @param accessByCreatorOnlyMode the access by creator mode.
 *
 * @since WAS XD 6.1 FIX3
*/
void setAccessByCreatorOnlyMode(int accessByCreatorOnlyMode);
```

詳細を示すために、ObjectGrid マップ・アカウントがバンキング・グリッドにあり、Manager1 と Employee1 が 2 人のユーザーであるようなシナリオを考えてみます。この場合、eXtreme Scale 許可ポリシーは、「Manager1」に対してはすべてのアクセス権を付与しますが、「Employee1」に対しては読み取りアクセス権しか付与しません。以下の例に示すのは、ObjectGrid マップ許可の JAAS ポリシーです。

```
grant codebase "http://www.ibm.com/com/ibm/ws/objectgrid/security/PrivilegedAction"
    Principal com.acme.PrincipalImpl "Manager1" {
    permission com.ibm.websphere.objectgrid.security.MapPermission
        "banking.account", "all"
};
grant codebase "http://www.ibm.com/com/ibm/ws/objectgrid/security/PrivilegedAction"
    Principal com.acme.PrincipalImpl "Employee1" {
    permission com.ibm.websphere.objectgrid.security.MapPermission
        "banking.account", "read"
};
```

作成者限定アクセスの機能が、どのように許可に影響するか検討してください。

- **disabled**: 作成者限定アクセスの機能が使用不可に設定された場合、マップ許可への影響はありません。ユーザー「Manager1」は、「account」マップ内のすべてのデータにアクセスできます。ユーザー「Employee1」は、マップ内のすべてのデータの read は許可されますが、マップ内のデータに対する update、invalidate、remove はできません。
- **complement**: 「complement」オプションを使用して作成者限定アクセスの機能を使用可能にした場合、マップ許可と作成者限定アクセスの機能の両方が有効になります。ユーザー「Manager1」は、自身が単独でデータをマップにロードした場合のみ、「account」マップ内のデータにアクセスできます。ユーザー「Employee1」は、自身が単独でデータをマップにロードした場合のみ、「account」マップ内のデータを読み取ることができます。(しかし、このユーザーは、マップ内のデータに対する update、invalidate、または remove は許可されません。)
- **supersede**: 「supersede」オプションを使用して作成者限定アクセスの機能を使用可能にした場合、マップ許可は実施されません。許可ポリシーは、作成者限定アクセスの許可のみになります。ユーザー「Manager1」には、「complement」モードの場合と同じ特権が与えられます。このユーザーは、自身がデータをマップにロードした場合のみ、「account」マップ内のデータにアクセスできます。しかし、今回はユーザー「Employee1」も、自身がデータをマップにロードすれば、「account」マップ内のデータに対する全アクセス権限が与えられます。つまり、Java 認証・承認サービス (JAAS) ポリシーに定義されている許可ポリシーは実施されません。

トランスポート層セキュリティーおよび Secure Sockets Layer

WebSphere eXtreme Scale は、動的デプロイメント・モデルにおいてクライアントとサーバーとの間のセキュア通信に TCP/IP も、Transport Layer Security/Secure Sockets Layer (TLS/SSL) もサポートします。

TLS/SSL は、クライアントとサーバーとの間のセキュア通信を可能にします。使用される通信メカニズムは、クライアントおよびサーバーの構成ファイル内に指定された transportType パラメーターの値によって決まります。

以下のクライアントおよびサーバー構成ファイル内に transportType プロパティーを設定できます。

- クライアント・セキュリティー構成内でこのプロパティーを設定するには、 210 ページの『クライアント・プロパティー・ファイル』を参照してください。
- コンテナー・サーバー・セキュリティー構成内でこのプロパティーを設定するには、 203 ページの『サーバー・プロパティー・ファイル』を参照してください。
- カタログ・サーバー・セキュリティー構成内でこのプロパティーを設定するには、 203 ページの『サーバー・プロパティー・ファイル』を参照してください。

表 10. クライアント・トランスポートおよびサーバー・トランスポートの設定で使用されるトランスポート・プロトコル

クライアントの transportType プロパティー	サーバーの transportType プロパティー	結果のプロトコル
TCP/IP	TCP/IP	TCP/IP
TCP/IP	SSL サポート	TCP/IP
TCP/IP	SSL 必須	エラー
SSL サポート	TCP/IP	TCP/IP
SSL サポート	SSL サポート	SSL (SSL が失敗した場合は TCP/IP)
SSL サポート	SSL 必須	SSL
SSL 必須	TCP/IP	エラー
SSL 必須	SSL サポート	SSL
SSL 必須	SSL 必須	SSL

SSL が使用される場合、クライアント・サイドとサーバー・サイドの両方で SSL 構成パラメーターが指定されている必要があります。Java SE 環境では、SSL 構成はクライアントまたはサーバーのプロパティー・ファイル内で構成されます。クライアントまたはサーバーが WebSphere Application Server 内にある場合、WebSphere Application Server の トランスポート・セキュリティー・サポートを使用して SSL パラメーターを構成できます。

トランスポート・セキュリティー・サポート用の orb.properties ファイルの構成

transportType プロパティーの値が SSL-Supported の場合は、TLS/SSL を使用することができます。

Java Platform, Standard Edition 環境でセキュア・トランスポートをサポートするには、 214 ページの『ORB プロパティー・ファイル』 ファイルを変更して、以下のプロパティーが含まれるようにする必要があります。

```
# IBM JDK properties
org.omg.CORBA.ORBClass=com.ibm.CORBA.iiop.ORB
org.omg.CORBA.ORBSingletonClass=com.ibm.rmi.corba.ORBSingleton
javax.rmi.CORBA.StubClass=com.ibm.rmi.javax.rmi.CORBA.StubDelegateImpl
```

```
javax.rmi.CORBA.PortableRemoteObjectClass=com.ibm.rmi.javax.rmi.PortableRemoteObject
javax.rmi.CORBA.UtilClass=com.ibm.ws.orb.WSUtilDelegateImpl

# WS Plugins
com.ibm.CORBA.ORBPluginClass.com.ibm.ws.orbimpl.transport.WSTransport
com.ibm.CORBA.ORBPluginClass.com.ibm.ws.orbimpl.WSORBPropertyManager
com.ibm.CORBA.ORBPluginClass.com.ibm.ISecurityUtilityImpl.SecurityPropertyManager

# WS Interceptors
org.omg.PortableInterceptor.ORBInitializerClass.com.ibm.ws.objectgrid.corba.ObjectGridInitializer
org.omg.PortableInterceptor.ORBInitializerClass.com.ibm.ISecurityLocalObjectBaseL13Impl.SecurityComponentFactory

# WS ORB & Plugins properties
com.ibm.ws.orb.transport.ConnectionInterceptorName=com.ibm.ISecurityLocalObjectBaseL13Impl.SecurityConnectionInterceptor
com.ibm.ws.orb.transport.WSSSLClientSocketFactoryName=com.ibm.ws.security.orbssl.WSSSLClientSocketFactoryImpl

com.ibm.CORBA.TransportMode=Pluggable
com.ibm.CORBA.ServerName=ogserver
```

eXtreme Scale クライアント用の SSL パラメーターの構成

次の方法でクライアントの SSL パラメーターを構成することができます。

1. com.ibm.websphere.objectgrid.security.config. ClientSecurityConfigurationFactory ファクトリー・クラスを使用して、com.ibm.websphere.objectgrid.security.config.SSLConfiguration オブジェクトを作成します。詳しくは、ClientSecurityConfigurationFactory API 資料を参照してください。
2. client.propeties ファイル内でパラメーターを構成し、次に、ClientSecurityConfigurationFactory.getClientSecurityConfiguration(String) メソッドを使用してオブジェクト・インスタンスを取り込みます。

クライアントに設定できるプロパティーの例については、210 ページの『クライアント・プロパティー・ファイル』でセキュリティー・クライアント・プロパティーに関するセクションを参照してください。

eXtreme Scale サーバー用の SSL パラメーターの構成

サーバーの SSL パラメーターの構成は、上記の server.properties ファイルの例のようなサーバー・プロパティー・ファイルを使用して行います。このプロパティー・ファイルは、eXtreme Scale サーバーの始動時にパラメーターとして渡すことができます。eXtreme Scale サーバーに対して設定できる SSL パラメーターについて詳しくは、203 ページの『サーバー・プロパティー・ファイル』を参照してください。

WebSphere Application Server でのトランスポート・セキュリティー・サポート

eXtreme Scale クライアント、コンテナー・サーバー、またはカタログ・サーバーが WebSphere Application Server プロセスで実行している場合、eXtreme Scale トランスポート・セキュリティーは Application Server CSIV2 トランスポート設定によって管理されます。 SSL 設定を構成するために eXtreme Scale クライアントまたはサーバーのプロパティーを使用するべきではありません。すべての SSL 設定は、WebSphere Application Server 構成で指定するようにしてください。

注:

プロパティー・ファイルを使用して SSL 設定を構成した場合は、既存の設定をオーバーライドすることになります。

関連資料

202 ページの『プロパティー・ファイルの解説』
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。

203 ページの『サーバー・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、サーバーのさまざまな設定 (例えば、トレース設定、ロギング、およびセキュリティー構成など) を定義する複数のプロパティーが含まれます。サーバー・プロパティー・ファイルは、カタログ・サービスとコンテナー・サービスによって使用されます。

210 ページの『クライアント・プロパティー・ファイル』
eXtreme Scale クライアント・プロセスの要件に基づいて、プロパティー・ファイルを作成できます。

214 ページの『ORB プロパティー・ファイル』
Object Request Broker (ORB) がグリッドのトランスポート動作を変更するために使用するプロパティーが、`orb.properties` ファイルを使用して受け渡されます。

グリッド認証

セキュア・トークン・マネージャー・プラグインは、サーバー間認証に使用される新たなプラグインです。セキュア・トークン・マネージャー・プラグインは、com.ibm.websphere.objectgrid.security.plugins.SecureTokenManager インターフェースによって表されます。

generateToken(Object) メソッドは保護されるオブジェクトを取得し、外部に識別されないトークンを生成します。verifyTokens(byte[]) メソッドは逆に、トークンを元のオブジェクトに変換して戻します。

単純な SecureTokenManager 実装は XOR アルゴリズムなど単純なエンコード・アルゴリズムを使用して、オブジェクトをシリアライズ済みフォームでエンコードし、対応するデコード・アルゴリズムを使用してトークンをデコードします。この実装は保護されていないため、簡単に中断されます。

WebSphere eXtreme Scale デフォルト実装

WebSphere eXtreme Scale には、このインターフェース用のすぐに使用可能な実装が用意されています。このデフォルト実装は、鍵ペアを使用して署名し、署名を検査します。また、秘密鍵を使用してコンテンツを暗号化します。すべてのサーバーには JCKES タイプの鍵ストアが備えられており、鍵ペア、秘密鍵と公開鍵、および秘密鍵が保管されています。鍵ストアは、秘密鍵を保管する JCKES タイプである必要があります。これらの鍵は、送信側で秘密ストリングを暗号化し、署名または検証する場合に使用されます。また、トークンは有効期限の時間に関連付けられています。受信側で、データの検証、暗号化解除、および受信側の秘密ストリングとの比較が行われます。サーバーのペアの間での認証には、Secure Sockets Layer (SSL) 通信プロトコルは必要ありません。これは、秘密鍵と公開鍵の目的が同じであるためです。ただし、サーバー通信が暗号化されていない場合は、通信時に侵入

者にデータを盗まれる可能性があります。トークンの有効期限が近いため、リプレイ・アタックの危険性は少なくなっています。この可能性は、すべてのサーバーをファイアウォールの後ろにデプロイすると、非常に小さくなります。

この方法の欠点は、WebSphere eXtreme Scale 管理者が鍵を生成し、生成した鍵をすべてのサーバーにトランスポートする必要があるため、トランスポート中にセキュリティー・ブリーチ (抜け穴) が発生する可能性があることです。

Java Management Extensions (JMX) セキュリティー

動的デプロイメント環境での Managed Bean (MBean) 呼び出しを保護することができます。

使用可能な MBean に関する詳細は、298 ページの『Managed Bean (MBean) を使用した環境の管理』を参照してください。

動的デプロイメント・トポロジーでは、MBean は、カタログ・サーバーおよびコンテナー・サーバーで直接ホストされます。一般に、動的デプロイメント・トポロジーの JMX セキュリティーは、JavaTM Management Extensions (JMX) 仕様に指定された JMX セキュリティー仕様に従います。これは、以下の 3 つのパートで構成されます。

1. 認証 - リモート・クライアントは、コネクター・サーバー内で認証される必要があります。
2. アクセス制御 - MBean アクセス制御は、MBean 情報にアクセスできるユーザー、および MBean 操作を実行できるユーザーを制限します。
3. セキュア・トランスポート - JMX クライアントとサーバー間のトランスポートは、TLS/SSL を使用して保護できます。

認証

JMX では、コネクター・サーバーがリモート・クライアントを認証するメソッドを提供しています。RMI コネクターの場合、認証は、コネクター・サーバーが作成される場合に JMXAuthenticator インターフェースを実装するオブジェクトを提供することにより実行されます。eXtreme Scale は、この JMXAuthenticator インターフェースを実装し、 ObjectGrid Authenticator プラグインを使用してリモート・クライアントを認証します。 eXtreme Scale がクライアントをどのように認証するのかについて詳しくは、「*製品概要*」でセキュリティーに関するチュートリアルを参照してください。

JMX クライアントは、JMX API に従って、コネクター・サーバーに接続するためのクレデンシャルを提供します。JMX フレームワークは、クレデンシャルをコネクター・サーバーに渡し、認証のため、JMXAuthenticator 実装を呼び出します。前述のように、JMXAuthenticator 実装は、ObjectGrid Authenticator 実装に認証を委任します。

以下の例は、クレデンシャルを使用してコネクター・サーバーに接続する方法を説明していますので、参考にしてください。

```
javax.management.remote.JMXServiceURL jmxUrl = new JMXServiceURL(
        "service:jmx:rmi:///jndi/rmi://localhost:1099/objectgrid/MBeanServer");

    environment.put(JMXConnector.CREDENTIALS, new UserPasswordCredential("admin", "xxxxxx"));
```

```
// Create the JMXCconnectorServer
JMXConnector cntor = JMXConnectorFactory.newJMXConnector(jmxUrl, null);

// Connect and invoke an operation on the remote MBeanServer
cntor.connect(environment);
```

上記の例では、UserPasswordCredential が、ユーザー ID が admin に、パスワードが xxxxx に設定されて提供されます。この UserPasswordCredential オブジェクトは、環境マップに設定され、これは、JMXConnector.connect(Map) メソッドで使用されます。次に、この UserPasswordCredential オブジェクトは、JMX フレームワークによってサーバーに渡され、最終的に認証のために ObjectGrid 認証フレームワークに渡されます。

クライアント・プログラミング・モデルは、厳格に JMX 仕様に従います。

アクセス制御

JMX MBean サーバーは、機密情報に対するアクセス権を持つことがあり、機密操作を実行することができる場合があります。JMX では、どのクライアントがその情報にアクセスでき、どのユーザーがそうした操作を実行できるかを識別する、必要なアクセス制御を提供しています。アクセス制御は、標準 Java セキュリティー・モデルに基づいて、MBean サーバーおよびその操作へのアクセスを制御する許可を定義することによって構築されます。

JMX 操作のアクセス制御または許可に関して、eXtreme Scale は、JMX 実装で提供される JAAS サポートに依存しています。プログラム実行の任意の時点で、実行のスレッドが保持する現行の許可セットがあります。そのようなスレッドが、JMX 仕様操作を呼び出す場合、これらは、保持許可と呼ばれています。 JMX 操作が実行されると、セキュリティー・チェックが行われ、必要な許可が保持許可によって暗黙的に示されているかどうかがチェックされます。

MBean ポリシー定義は、Java ポリシー形式に従います。例えば、以下のポリシーでは、すべての署名者およびすべてのコード・ベースに PlacementServiceMBean のサーバー JMX アドレスを取得する権限を付与していますが、com.ibm.websphere.objectgrid ドメインに対しては制限しています。

```
grant {
    permission javax.management.MBeanPermission
        "com.ibm.websphere.objectgrid.management.PlacementServiceMBean#retrieveServerJMXAddress
     [com.ibm.websphere.objectgrid:*,type=PlacementService]",
        "invoke";
}
```

以下のポリシー・サンプルを使用すれば、リモート・クライアント ID に基づく許可を完了することができます。このポリシーでは、前の例に示されたものと同じ MBean 許可を付与していますが、X500Principal 名が CN=Administrator、OU=software、O=IBM、L=Rochester、ST=MN、C=US のユーザーだけは除きます。

```
grant principal javax.security.auth.x500.X500Principal "CN=Administrator,OU=software,O=IBM,L=Rochester,ST=MN,C=US" {
    permission javax.management.MBeanPermission
        "com.ibm.websphere.objectgrid.management.PlacementServiceMBean#retrieveServerJMXAddress
    [com.ibm.websphere.objectgrid:*,type=PlacementService]",
        "invoke";
}
```

Java ポリシーは、セキュリティー・マネージャーがオンになっている場合に限ってチェックされます。-Djava.security.manager JVM 引数を設定してカタログ・サーバーおよびコンテナー・サーバーを始動し、MBean 操作のアクセス制御を強制するようにしてください。

セキュア・トランスポート

JMX クライアントとサーバー間のトランスポートは、TLS/SSL を使用して保護することができます。カタログ・サーバーまたはコンテナー・サーバーの transportType が SSL_Required または SSL_Supported に設定されている場合、SSL を使用して JMX サーバーに接続する必要があります。

SSL を使用するには、-D システム・プロパティーを使用して、トラストストア、トラストストア・タイプ、およびトラストストア・パスワードを MBean クライアントに構成する必要のある必要があります。

1. -Djavax.net.ssl.trustStore=TRUST_STORE_LOCATION
2. -Djavax.net.ssl.trustStorePassword=TRUST_STORE_PASSWORD
3. -Djavax.net.ssl.trustStoreType=TRUST_STORE_TYPE

JAVA_HOME/jre/lib/security/java.security ファイルで SSL ソケット・ファクトリーとして com.ibm.websphere.ssl.protocol.SSLSocketFactory を使用する場合は、次のプロパティーを使用します。

1. -Dcom.ibm.ssl.trustStore=TRUST_STORE_LOCATION
2. -Dcom.ibm.ssl.trustStorePassword=TRUST_STORE_PASSWORD
3. -Dcom.ibm.ssl.trustStoreType=TRUST_STORE_TYPE

セキュリティー記述子 XML ファイル

ObjectGrid セキュリティー記述子 XML ファイルを使用して、セキュリティーを使用可能にした eXtreme Scale デプロイメント・トポロジーを構成できます。このトピックでは、各種構成を説明するサンプル XML ファイルを提供します。

クラスター XML ファイルの各エレメントおよび属性の説明は、以下のリストに示されています。各サンプルでは、これらのエレメントおよび属性を使用して環境を構成する方法について説明しています。

securityConfig エレメント

securityConfig エレメントは、ObjectGrid セキュリティー XML ファイルの最上位エレメントです。このエレメントは、ファイルの名前空間とスキーマ・ロケーションをセットアップします。スキーマは `objectGridSecurity.xsd` ファイルで定義されます。

- 出現回数: 1 回
- 子エレメント: security

security エレメント

security エレメントは、ObjectGrid セキュリティーの定義に使用します。

- 出現回数: 1 回
- 子エレメント: authenticator、adminAuthorization、および systemCredentialGenerator

属性

securityEnabled
true に設定されているとき、グリッドのセキュリティーを使用可能にします。デフォルト値は false です。値を false に設定すると、グリッド全体のセキュリティーが使用不可になります。詳しくは、 324 ページの『グリッド・セキュリティー』を参照してください。 (オプション)

singleSignOnEnabled
値が true に設定されている場合は、クライアントがいずれか 1 つのサーバーに認識された後、任意のサーバーに接続できます。そうでない場合は、クライアントは接続のたびに各サーバーに対して認証を行う必要があります。デフォルト値は false です。(オプション)

loginSessionExpirationTime
ログイン・セッションが期限切れになるまでの時間を秒数で指定します。ログイン・セッションの有効期限が切れると、クライアントは再度認証する必要があります。(オプション)

adminAuthorizationEnabled
管理許可を使用可能にします。この値が true に設定されている場合は、すべての管理用タスクに許可が必要です。使用される許可メカニズムは、adminAuthorizationMechanism 属性の値により指定されます。デフォルト値は false です。(オプション)

adminAuthorizationMechanism
使用する許可メカニズムを示します。WebSphere eXtreme Scale は、2 種類の許可メカニズム、すなわち、 Java 認証・承認サービス (JAAS) とカスタム許可をサポートします。JAAS 許可メカニズムは、標準の JAAS ポリシー・ベースのアプローチを使用します。許可メカニズムとして JAAS を指定するには、値に AUTHORIZATION_MECHANISM_JAAS を設定します。カスタム許可メカニズムは、ユーザー・プラグイン AdminAuthorization 実装を使用します。カスタム許可メカニズムを指定するには、値に AUTHORIZATION_MECHANISM_CUSTOM を設定します。これら 2 つのメカニズムがどのように使用されるかについての詳細は、 328 ページの『アプリケーション・クライアントの許可』を参照してください。(オプション)

以下の `security.xml` ファイルは、 eXtreme Scale グリッド・セキュリティーを使用可能にするためのサンプル構成です。

security.xml

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

    <security securityEnabled="true" singleSignOnEnabled="true"
        loginSessionExpirationTime="20"
        adminAuthorizationEnabled="true"
        adminAuthorizationMechanism="AUTHORIZATION_MECHANISM_JAAS" >

        <authenticator className ="com.ibm.websphere.objectgrid.security.plugins.builtins.WSTokenAuthenticator">
        </authenticator>

        <systemCredentialGenerator className ="com.ibm.websphere.objectgrid.security.plugins.builtins.WSTokenCredentialGenerator">
            <property name="properties" type="java.lang.String" value="runAs" description="Using runAs subject" />
        </systemCredentialGenerator>

    </security>
</securityConfig>
```

authenticator エレメント

グリッド内の eXtreme Scale サーバーに対してクライアントを認証します。className 属性によって指定されるクラスは、com.ibm.websphere.objectgrid.security.plugins.Authenticator インターフェースを実装している必要があります。オーセンティケーターはプロパティーを使用し、className 属性によって指定されるクラスのメソッドを呼び出すことができます。プロパティーの使用について詳しくは、property エレメントを参照してください。

前記の `security.xml` ファイルの例では、com.ibm.websphere.objectgrid.security.plugins.builtins.WSTokenAuthenticator クラスがオーセンティケーターとして指定されています。このクラスは com.ibm.websphere.objectgrid.security.plugins.Authenticator インターフェースを実装します。

- 出現回数: 0 回または 1 回
- 子エレメント: property

属性

className

com.ibm.websphere.objectgrid.security.plugins.Authenticator インターフェースを実装するクラスを指定します。このクラスを使用して、eXtreme Scale グリッド内のサーバーに対してクライアントを認証します。(必須)

adminAuthorization エレメント

adminAuthorization エレメントは、グリッドへの管理アクセスをセットアップする場合に使用します。

- 出現回数: 0 回または 1 回
- 子エレメント: property

属性

className

com.ibm.websphere.objectgrid.security.plugins.AdminAuthorization インターフェースを実装するクラスを指定します。 (必須)

systemCredentialGenerator エレメント

systemCredentialGenerator エレメントを使用すると、システム・クレデンシャル生成プログラムがセットアップされます。このエレメントは、動的環境にのみ適用されます。動的構成モデルでは、動的コンテナー・サーバーは、 eXtreme Scale クライアントとしてカタログ・サーバーに接続し、カタログ・サーバーもクライアントとして eXtreme Scale コンテナー・サーバーに接続できます。このシステム・クレデンシャル生成プログラムは、システム・クレデンシャルのファクトリーを表すために使用します。

- 出現回数: 0 回または 1 回
- 子エレメント: property

属性

className

com.ibm.websphere.objectgrid.security.plugins.CredentialGenerator インターフェースを実装するクラスを指定します。 (必須)

systemCredentialGenerator の使用例については、前の `security.xml` をファイル参照してください。この例では、システム・クレデンシャル生成プログラムは com.ibm.websphere.objectgrid.security.plugins.builtins. WSTokenCredentialGenerator で、これはスレッドから RunAs Subject オブジェクトを取得します。

property エレメント

authenticator クラスおよび adminAuthorization クラスにおいて set メソッドを呼び出します。プロパティーの名前は、authenticator エレメントまたは adminAuthorization エレメントの className 属性の set メソッドに対応しています。

- 出現回数: 0 回以上
- 子エレメント: property

属性

name

プロパティーの名前を指定します。この属性に割り当てられる値は、このプロパティーを含む Bean の className 属性で指定されたクラスの set メソッドと対応している必要があります。例えば、Bean の className 属性が com.ibm.MyPlugin に設定され、指定されているプロパティーの名前が size である場合、com.ibm.MyPlugin クラスには setSize メソッドが必要です。(必須)

type

プロパティーのタイプを指定します。パラメーターのタイプは、name 属性により識別される set メソッドに渡されます。有効な値は、 Java プリミティブ、それに対応する java.lang プリミティブ、および java.lang.String です。name 属性と type 属性は、Bean の className 属性のメソッド・シグニチャーに対応していなければなりません。例えば、名前が size であり、タイプが int である場合は、Bean の className 属性で指定されたクラスに setSize(int) メソッドが存在している必要があります。(必須)

value

プロパティーの値を指定します。この値は type 属性によって指定されたタイプに変換され、次に name 属性と type 属性で識別された set メソッドへの呼び出しでパラメーターとして使用されます。この属性の値は、どんな方法でも妥当性検査されません。プラグイン・インプリメンターは、渡された値が有効であることを検証しなければなりません。 (必須)

description

プロパティーの説明を入力します。 (オプション)

詳しくは、 201 ページの『`objectGridSecurity.xsd` ファイル』を参照してください。

WebSphere Application Server とのセキュリティー統合

WebSphere eXtreme Scale には、 WebSphere Application Server セキュリティー・インフラストラクチャーと統合するためのいくつかのセキュリティー・フィーチャーがあります。

認証統合

eXtreme Scale クライアントおよびサーバーが WebSphere Application Server および同じセキュリティー・ドメインで稼働中の場合、 WebSphere Application Server セキュリティー・インフラストラクチャーを使用して、クライアント認証クレデンシャルを eXtreme Scale サーバーに伝搬することができます。例えば、サーブレットが eXtreme Scale クライアントとして動作して、同じセキュリティー・ドメインの eXtreme Scale サーバーに接続し、そのサーブレットが既に認証されている場合、認証トークンをクライアント (サーブレット) からサーバーに伝搬し、その後、WebSphere Application Server セキュリティー・インフラストラクチャーを使用して、認証トークンを元のクライアント・クレデンシャルに変換することができます。

WebSphere Application Server との分散セキュリティー統合

分散 ObjectGrid モデルの場合、セキュリティー統合は、以下のクラスを使用して完了できます。

```
com.ibm.websphere.objectgrid.security.plugins.builtins.WSTokenCredentialGenerator

com.ibm.websphere.objectgrid.security.plugins.builtins.WSTokenAuthenticator

com.ibm.websphere.objectgrid.security.plugins.builtins.WSTokenCredential
```

詳しくは、 326 ページの『アプリケーション・クライアントの認証』を参照してください。以下に、WSTokenCredentialGenerator クラスの使用方法の例を示します。

```
/**
 * connect to the ObjectGrid Server.
 */
protected ClientClusterContext connect() throws ConnectException {
  ClientSecurityConfiguration csConfig = ClientSecurityConfigurationFactory
  .getClientSecurityConfiguration(proFile);

  CredentialGenerator gen = getWSCredGen();

  csConfig.setCredentialGenerator(gen);

  return objectGridManager.connect(csConfig, null);
}

/**
 * Get a WSTokenCredentialGenerator
 *
private CredentialGenerator getWSCredGen() {
    WSTokenCredentialGenerator gen = new WSTokenCredentialGenerator(
        WSTokenCredentialGenerator.RUN_AS_SUBJECT);
    return gen;
}
```

サーバー・サイドで、WSTokenAuthentication オーセンティケーターを使用して、WSTokenCredential オブジェクトを認証します。

WebSphere Application Server とのローカル・セキュリティー統合

ローカル ObjectGrid モデルの場合、セキュリティー統合は、以下の 2 つのクラスを使用して完了できます。

- com.ibm.websphere.objectgrid.security.plugins.builtins.WSSubjectSourceImpl

- com.ibm.websphere.objectgrid.security.plugins.builtins.WSSubjectValidationImpl

これらのクラスについての詳細は、*プログラミング・ガイド* のローカル・セキュリティーに関する情報を参照してください。 WSSubjectSourceImpl クラスを SubjectSource プラグインとして構成し、WSSubjectValidationImpl クラスを SubjectValidation プラグインとして構成することができます。

セキュア eXtreme Scale サーバーの開始と停止

デプロイメント環境ではサーバーは保護される必要があることが多く、そのためには特別な構成が必要です。

Java SE 環境でのセキュア・サーバーの開始

カタログ・サービスまたはコンテナー・サーバーは次のように開始できます。

セキュア eXtreme Scale カタログ・サービスの開始

セキュア eXtreme Scale カタログ・サービス・プロセスには、追加で 2 つのセキュリティー構成ファイルが必要です。

セキュリティー記述子 XML ファイル: セキュリティー記述子 XML ファイルは、すべてのサーバー (カタログ・サーバーおよびコンテナー・サーバーを含む) に共通するセキュリティー・プロパティーを記述します。プロパティーの例の 1 つは、ユーザー・レジストリーおよび認証メカニズムを表すオーセンティケーター構成です。

サーバー・プロパティー・ファイル。サーバー・プロパティー・ファイルは、サーバーに固有のセキュリティー・プロパティーを構成します。

startOgServer.sh または startOgServer.cat コマンドを使用してセキュア eXtreme Scale カタログ・サービス・プロセスを開始するときに、-clusterSecurityFile または -clusterSecurityUrl を使用して、ファイル・タイプまたは URL タイプとしてセキュリティー記述子 XML ファイルを設定することができ、–serverProps を使用してサーバー・プロパティー・ファイルを設定することができます。

セキュア eXtreme Scale コンテナー・サーバーの開始

セキュア eXtreme Scale コンテナー・サーバーの開始には、1 つのセキュリティー構成ファイルが必要です。

- **サーバー・プロパティー・ファイル:** サーバー・プロパティー・ファイルは、サーバーに固有のセキュリティー・プロパティーを構成します。詳しくは、 203 ページの『サーバー・プロパティー・ファイル』を参照してください。

startOgServer.sh または startOgServer.cat コマンドを使用してセキュア eXtreme Scale コンテナー・サーバーを開始するときに、 –serverProps を使用してサーバー・プロパティー・ファイルを設定できます。サーバー・プロパティー・ファイルを設定する方法は他にもあります。詳しくは、サーバー・プロパティー・ファイルを参照してください。

startOgServer.sh または startOgServer.bat コマンドとそのオプションの使用方法について詳しくは、 244 ページの『startOgServer スクリプト』を参照してください。

セキュア eXtreme Scale サーバーの停止

セキュア eXtreme Scale カタログ・サービス・プロセスまたはコンテナー・サーバーの停止には、1 つのセキュリティー構成ファイルが必要です。

- **クライアント・プロパティー・ファイル:** クライアント・プロパティー・ファイルを使用して、クライアント・セキュリティー・プロパティーを構成できます。クライアント・セキュリティー・プロパティーは、クライアントがセキュア・サーバーに接続するために必要です。詳しくは、 210 ページの『クライアント・プロパティー・ファイル』を参照してください。

stopOgServer.sh または stopOgServer.cat コマンドを使用してセキュア eXtreme Scale カタログ・サービス・プロセスまたはコンテナー・サーバーを停止するときに、-clientSecurityFile を使用してクライアント・セキュリティー・プロパティーを設定できます。

stopOgServer.sh または stopOgServer.cat コマンドとそのオプションの使用方法について詳しくは、 251 ページの『stopOgServer スクリプト』を参照してください。

WebSphere Application Server でのセキュア・サーバーの始動

WebSphere Application Server でのセキュア ObjectGrid サーバーの開始は、セキュリティー構成ファイルを渡す必要がある点を除いて、非セキュア ObjectGrid サーバーの開始と似ています。 Java SE 環境でコマンド中に -[PROPERTY_FILE] (例えば -serverProps) を使用する代わりに、汎用 Java 仮想マシン (JVM) 引数で -D[PROPERTY_FILE] を使用します。

Websphere Application Server でのセキュア・カタログ・サービスの開始

カタログ・サーバーには、以下の 2 つの異なるレベルのセキュリティー情報が含まれます。

- `-Dobjectgrid.cluster.security.xml.url`: これは、すべてのサーバー (カタログ・サーバーおよびコンテナー・サーバーを含む) に共通するセキュリティー・プロパティーを記述する objectGridSecurity.xml ファイルを指定します。ユーザー・レジストリーおよび認証メカニズムを表わすオーセンティケーター構成はその一例です。このプロパティーに指定するファイル名は、URL 形式 (例えば「file:///tmp/og/objectGridSecurity.xml」) でなければなりません。
- `-Dobjectgrid.server.props`: これは、サーバー固有のセキュリティー・プロパティーが入っているサーバー・プロパティー・ファイルを指定します。このプロパティーに指定するファイル名は、普通のファイル・パス形式 (例えば「c:/tmp/og/catalogserver.props」) です。 -Dobjectgrid.security.server.props の使用は推奨されませんが、後方互換性のために引き続き使用できることに注意してください。

WebSphere Application Server 内でセキュア・カタログ・サービスを開始するには、324 ページの『グリッド・セキュリティー』の『WebSphere Application Server への組み込み』の説明に従ってください。

その後、プロセスの汎用 JVM 引数でセキュリティー・プロパティーを設定します。

```
-Dobjectgrid.cluster.security.xml.url=file:///tmp/og/
objectGridSecurity.xml-Dobjectgrid.server.props=/tmp/og/
catalog.server.props
```

汎用 JVM 引数を追加する手順は以下のとおりです。

- 左側のタスク・ビューで「システム管理」を展開します。
- カタログ・サービスがデプロイされる WebSphere Application Server プロセス (例えば、「デプロイメント・マネージャー」) をクリックします。
- 右側のページで、「サーバー・インフラストラクチャー」の下の「Java およびプロセス管理」を展開します。
- 「プロセス定義」をクリックします。
- 「追加プロパティー」の下の「Java 仮想マシン」をクリックします。
- 「汎用 JVM 引数」テキスト・ボックスにプロパティーを入力します。

WebSphere Application Server でのセキュア・コンテナー・サーバーの開始

カタログ・サーバーに接続されたコンテナー・サーバーは、オーセンティケーター構成やログイン・セッション・タイムアウト設定など、objectGridSecurity.xml に構成されたすべてのセキュリティー構成を取得します。またコンテナー・サーバーは、-Dobjectgrid.server.props プロパティーにそのサーバー固有のセキュリティー・プロパティーを構成する必要があります。

このプロパティー・ファイルには、セキュリティーに関係しない他のプロパティーも入れるため、-Dobjectgrid.security.server.props プロパティーの代わりに -Dobjectgrid.server.props プロパティーを使用する必要があります。このプロパティーに対して指定されるファイル名の形式は、単なるプレーン・ファイル・パス形式です。例えば、`c:/tmp/og/server.props` などです。

上記と同じ手順に従って、セキュリティー・プロパティーを汎用 JVM 引数に追加してください。

関連タスク

236 ページの『スタンドアロン WebSphere eXtreme Scale サーバーの始動』
スタンドアロン WebSphere eXtreme Scale 構成を実行しているとき、環境はカタログ・サーバー、コンテナー・サーバー、および eXtreme Scale クライアント・プロセスで構成されています。これらのプロセスは手動で構成して開始する必要があります。

237 ページの『スタンドアロン環境でのカタログ・サービスの開始』
WebSphere Application Server が実行されていない分散 WebSphere eXtreme Scale 環境を使用している場合は、カタログ・サービスを手動で開始する必要があります。

240 ページの『コンテナー・プロセスの開始』
eXtreme Scale は、デプロイメント・トポロジー、または `server.properties` ファイルを使用して、コマンド行から開始できます。

関連資料

244 ページの『startOgServer スクリプト』
startOgServer スクリプトはサーバーを始動します。サーバーの始動時に各種パラメーターを使用して、トレースを使用可能にしたり、ポート番号を指定するなど、さまざまな設定を行うことができます。

317 ページの『ログおよびトレース』
ログおよびトレースを使用して、環境のモニターおよびトラブルシューティングを実行できます。ログは、構成によってさまざまなロケーションにあります。IBM サポートに協力を依頼する場合、サーバーに関するトレースを提供する必要がある場合があります。

第 10 章 トラブルシューティング

ログとトレース、メッセージ、およびリリース情報の他に、モニター・ツールを使用して構成のトラブルシューティングを行えます。

トラブルシューティングのためのモニター・ツールの使用

ログとトレース、メッセージ、およびリリース情報の他に、モニター・ツールを使用して環境内のデータのロケーション、グリッド内のサーバーのアベイラビリティーなどの問題を把握することができます。 WebSphere Application Server 環境で実行している場合、 Performance Monitoring Infrastructure (PMI) を使用できます。スタンドアロン環境で実行している場合は、ベンダーのモニター・ツール (CA Wily Introscope あるいは Hyperic HQ など) を使用できます。また、xsAdmin サンプル・ユーティリティーを使用し、これをカスタマイズすれば、ご使用の環境に関するテキスト情報を表示させることができます。

モニター・ツールに関して詳しくは、 275 ページの『第 8 章 デプロイメント環境のモニター』を参照してください。

ログおよびトレース

ログおよびトレースを使用して、環境のモニターおよびトラブルシューティングを実行できます。ログは、構成によってさまざまなロケーションにあります。IBM サポートに協力を依頼する場合、サーバーに関するトレースを提供する必要がある場合があります。

WebSphere Application Server によるロギング

詳しくは、WebSphere Application Server インフォメーション・センターを参照してください。

スタンドアロン環境での WebSphere eXtreme Scale によるロギング

スタンドアロン・カタログおよびコンテナー・サービスを使用して、ログのロケーションおよびトレース仕様を設定します。カタログ・サーバー・ログは、サーバー始動コマンドを実行したロケーションにあります。

コンテナー・サーバーのログ・ロケーションの設定

デフォルトでは、コンテナーのログは、サーバー・コマンドが実行されたディレクトリーにあります。*<eXtremeScale_home>*/bin ディレクトリーでサーバーを始動する場合、ログおよびトレース・ファイルは bin ディレクトリーの logs/*<server_name>* ディレクトリー内にあります。コンテナー・サーバー・ログの代替ロケーションを指定するには、以下のコンテンツを使用して server.properties ファイルなどのプロパティー・ファイルを作成します。

```
workingDirectory=<directory>
traceSpec=
systemStreamToFileEnabled=true
```

workingDirectory プロパティーは、ログおよびオプションのトレース・ファイルのルート・ディレクトリーです。WebSphere eXtreme Scale は、traceSpec オプションでトレースが使用可能になっていると、SystemOut.log ファイル、SystemErr.log ファイル、およびトレース・ファイルを使用して、コンテナー・サーバーの名前を持つディレクトリーを作成します。コンテナー開始中にプロパティー・ファイルを使用するには、**-serverProps** オプションを使用して、サーバー・プロパティー・ファイルのロケーションを指定します。

詳しくは、 236 ページの『スタンドアロン WebSphere eXtreme Scale サーバーの始動』および 244 ページの『startOgServer スクリプト』を参照してください。

SystemOut.log ファイル内で参照する共通の情報メッセージは、開始確認メッセージです。特定のメッセージについて詳しくは、 352 ページの『メッセージ』を参照してください。

WebSphere Application Server によるトレース

詳しくは、WebSphere Application Server インフォメーション・センターを参照してください。

カタログ・サービスでのトレース

カタログ・サービスの始動中に、 **-traceSpec** および **-traceFile** パラメーターを使用して、カタログ・サービスでトレースを設定できます。以下に例を示します。

```
startOgServer.sh catalogServer -traceSpec
ObjectGridPlacement=all=enabled -traceFile
/home/user1/logs/trace.log
```

<eXtremeScale_home>/bin ディレクトリーでカタログ・サービスを始動する場合、ログおよびトレース・ファイルは bin ディレクトリーの logs/*<catalog_service_name>* ディレクトリー内にあります。 カタログ・サービスの開始に関して詳しくは、 237 ページの『スタンドアロン環境でのカタログ・サービスの開始』を参照してください。

スタンドアロン・コンテナー・サーバーでのトレース

コンテナー・サーバーでトレースを使用可能にするには 2 つの方法があります。ログ・セクションでの説明どおりに、サーバー・プロパティー・ファイルを作成するか、始動時にコマンド行を使用してトレースを使用可能にすることができます。サーバー・プロパティー・ファイルを使用してコンテナー・トレースを使用可能にするには、必要なトレース仕様で **traceSpec** プロパティーを更新します。始動パラメーターを使用して、コンテナー・トレースを使用可能にするには、**-traceSpec** および **-traceFile** パラメーターを使用します。以下に例を示します。

```
startOgServer.sh c0 -objectGridFile ../xml/myObjectGrid.xml
-deploymentPolicyFile ../xml/myDepPolicy.xml -catalogServiceEndpoints
server1.rchland.ibm.com:2809 -traceSpec
ObjectGridPlacement=all=enabled -traceFile /home/user1/logs/trace.log
```

<eXtremeScale_home>/bin ディレクトリーでサーバーを始動する場合、ログおよびトレース・ファイルは bin ディレクトリーの logs/*<server_name>* ディレクトリー内にあります。 コンテナー・プロセスの開始について詳しくは、 240 ページの『コンテナー・プロセスの開始』を参照してください。

ObjectGridManager インターフェースによるトレース

別の方法として、ObjectGridManager インターフェースで実行時にトレースを設定する方法があります。 ObjectGridManager インターフェースでのトレース設定を使用すると、eXtreme Scale に接続してトランザクションをコミットしている間に eXtreme Scale クライアント上でトレースを取得することができます。ObjectGridManager インターフェースでトレースを設定するには、トレース仕様およびトレース・ログを指定します。

```
ObjectGridManager manager = ObjectGridManagerFactory.getObjectGridManager();
...
manager.setTraceEnabled(true);
manager.setTraceFileName("logs/myClient.log");
manager.setTraceSpecification("ObjectGridReplication=all=enabled");
```

xsadmin ユーティリティーでトレースを使用可能にする

xsadmin ユーティリティーを使用してトレースを使用可能にする場合、**setTraceSpec** オプションを使用します。 xsadmin ユーティリティーを使用して、開始時ではなく実行時にスタンドアロン環境でトレースを使用可能にすることができます。すべてのサーバーおよびカタログ・サービスに対してトレースを使用可能にすることができます。あるいは、ObjectGrid 名などでサーバーをフィルタリングすることもできます。例えば、カタログ・サービス・サーバーへのアクセスを使用して ObjectGridReplication トレースを使用可能にするには、以下を実行します。

```
<eXtremeScale_home>/bin>xsadmin.bat -setTraceSpec "ObjectGridReplication=all=enabled"
```

トレース仕様を `*=all=disabled` に設定することでトレースを使用不可にすることもできます。

詳しくは、 300 ページの『xsAdmin サンプル・ユーティリティーの使用』を参照してください。

ffdc ディレクトリーおよびファイル

FFDC ファイルは、IBM サポートがデバッグの補助とするファイルです。これらのファイルは、問題が生じた場合に IBM サポートによって要求される場合があります。

これらのファイルは、`ffdc` という名前のディレクトリーに存在し、以下のファイルに類似したファイルが含まれています。

```
server2_exception.log
server2_20802080_07.03.05_10.52.18_0.txt
```

関連概念

342 ページの『セキュア eXtreme Scale サーバーの開始と停止』
デプロイメント環境ではサーバーは保護される必要があることが多く、そのためには特別な構成が必要です。

関連タスク

236 ページの『スタンドアロン WebSphere eXtreme Scale サーバーの始動』
スタンドアロン WebSphere eXtreme Scale 構成を実行しているとき、環境はカタログ・サーバー、コンテナー・サーバー、および eXtreme Scale クライアント・プロセスで構成されています。これらのプロセスは手動で構成して開始する必要があります。

237 ページの『スタンドアロン環境でのカタログ・サービスの開始』
WebSphere Application Server が実行されていない分散 WebSphere eXtreme Scale 環境を使用している場合は、カタログ・サービスを手動で開始する必要があります。

240 ページの『コンテナー・プロセスの開始』
eXtreme Scale は、デプロイメント・トポロジー、または `server.properties` ファイルを使用して、コマンド行から開始できます。

トレース・オプション

トレースを使用可能にすることで、ご使用の環境に関する情報を IBM サポートに提供することができます。

トレースについて

WebSphere eXtreme Scale のトレースは、いくつかの異なるコンポーネントに分けられます。 WebSphere Application Server のトレースと同様、使用するトレース・レベルを指定することができます。一般的なトレースのレベルには、
all、debug、entryExit、および event があります。

トレース・ストリングの例は、以下のとおりです。

```
ObjectGridComponent=level=enabled
```

トレース・ストリングは連結することができます。* (アスタリスク) 記号を使用してワイルドカード値を指定します (例: `ObjectGrid*=all=enabled`)。トレースを IBM サポートに提供する必要がある場合は、特定のトレース・ストリングが要求されます。 例えば、複製に関する問題が発生した場合には、`ObjectGridReplication=debug=enabled` トレース・ストリングが要求される可能性があります。

トレース仕様

ObjectGrid
汎用・コア・キャッシュ・エンジン。

ObjectGridCatalogServer
汎用カタログ・サービス。

ObjectGridChannel
静的デプロイメント・トポロジー通信。

ObjectgridCORBA
動的デプロイメント・トポロジー通信。

ObjectGridDataGrid
AgentManager API。

ObjectGridDynaCache
WebSphere eXtreme Scale 動的キャッシュ・プロバイダー

ObjectGridEntityManager
EntityManager API。Projector オプションとともに使用。

ObjectGridEvictors
ObjectGrid 組み込み Evictor。

ObjectGridJPA
Java Persistence API (JPA) ローダー

ObjectGridJPACache
JPA キャッシュ・プラグイン

ObjectGridLocking
ObjectGrid キャッシュ・エントリー・ロック・マネージャー。

ObjectGridMBean
管理 Bean

ObjectGridPlacement
カタログ・サーバー断片配置サービス。

ObjectGridQuery
ObjectGrid 照会。

ObjectGridReplication
レプリケーション・サービス。

ObjectGridRouting
クライアント/サーバー・ルーティングの詳細。

ObjectGridSecurity
セキュリティー・トレース。

ObjectGridStats
ObjectGrid 統計。

ObjectGridStreamQuery
ストリーム照会 API。

ObjectGridWriteBehind
ObjectGrid 後書き。

Projector
EntityManager API 内のエンジン。

QueryEngine
オブジェクト照会 API および EntityManager 照会 API のための照会エンジン。

QueryEnginePlan
照会計画診断。

メッセージ

製品インターフェースのログまたはその他の部分にメッセージが表示された場合は、そのコンポーネントの接頭部でメッセージを検索して、詳細情報を確認してください。

メッセージの検索

ログにメッセージが表示された場合、そのメッセージ番号を (接頭部の文字と番号と共に) コピーして、インフォメーション・センターで検索します (例えば、CWOBJ1526I)。メッセージを検索すると、そのメッセージの詳細説明や、問題解決のために実行できる可能性のあるアクションを確認できます。

製品メッセージの索引については、インフォメーション・センターを参照してください。

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』
- 『製品資料へのアクセス』
- 『製品のサポート Web サイトへのアクセス』
- 353 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。

- 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・ページ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。

パフォーマンスおよびベスト・プラクティス

WebSphere eXtreme Scale のパフォーマンスを向上させることができます。

WebSphere eXtreme Scale は、アプリケーションのパフォーマンスを向上させることを目的として設計されています。eXtreme Scale を使用する場合に、アプリケーションのパフォーマンスに影響を与える要因が多く存在します。例えば、トランザクション・サイズ、コピー・モード、シリアライゼーション技法、および区画アーキテクチャーなどです。最高のパフォーマンスを実現するには、アーキテクチャーを定義し、アプリケーションを注意深く実装する必要があります。

パフォーマンスの向上を図る前に、 「*製品概要*」に記載されているトランザクションに関する情報を参照して、ご使用の環境に合ったロック・ストラテジーを選択できるようにしてください。

eXtreme Scale 成果物のパフォーマンス

パフォーマンスについては、eXtreme Scale のいくつかの異なる側面に関する作業を行うことができます。

- **シリアライゼーションのパフォーマンス:** 詳しくは、「*製品概要*」でシリアライゼーションのパフォーマンスに関する説明を参照してください。
- **トランザクション・サイズ:** 詳しくは、「*製品概要*」でトランザクション・サイズに関する説明を参照してください。
- **複製のパフォーマンス:** 詳しくは、「*製品概要*」で複製のパフォーマンスに関する説明を参照してください。
- **エンティティー・マネージャーのパフォーマンスの影響:** 詳しくは、「プログラミング・ガイド」に記載されているエンティティー・マネージャーのパフォーマンスに関する説明を参照してください。
- **照会の調整:** 詳しくは、「プログラミング・ガイド」でパフォーマンスのための照会の調整に関する説明を参照してください。

ベスト・プラクティス

マップのパフォーマンスは、いくつかのベスト・プラクティスによって改善することができます。各アプリケーションと環境で、パフォーマンスに対して別々のソリューションを使用し、eXtreme Scaleでは、パフォーマンスを向上するための組み込みのカスタマイズを提供しています。アプリケーション・アーキテクチャー内でも

さらにパフォーマンスの向上が可能です。向上が導入されるのは、アプリケーションとそのアーキテクチャー関連のみです。

- **ロックのパフォーマンスに関するベスト・プラクティス:** アプリケーションのパフォーマンスに影響を与えるさまざまなロック・ストラテジーの中から選択します。詳しくは、「*プログラミング・ガイド*」に記載されているロック・パフォーマンスのベスト・プラクティスに関する説明を参照してください。
- **CopyMode メソッドに関するベスト・プラクティス:** eXtreme Scale がエントリーの保守とコピーを行う方法を変更するために使用される、さまざまなコピー・モードの中から選択します。詳しくは、「*プログラミング・ガイド*」に記載されている CopyMode メソッドのベスト・プラクティスに関する説明を参照してください。
- **ObjectTransformer インターフェースに関するベスト・プラクティス:** ObjectTransformer インターフェースを使用すると、アプリケーションへのコールバックが可能になり、一般的でコストのかかる操作 (オブジェクト・シリアライゼーションやオブジェクトのディープ・コピーなど) のカスタム実装が提供されます。詳しくは、「*プログラミング・ガイド*」に記載されている ObjectTransformer インターフェースのベスト・プラクティスに関する説明を参照してください。
- **プラグイン・エビクターのパフォーマンスに関するベスト・プラクティス:** 最少使用頻度 (LFU) 除去ストラテジーと最長未使用時間 (LRU) 除去ストラテジーの中から選択します。詳しくは、「*プログラミング・ガイド*」に記載されているプラグイン・エビクターのパフォーマンスに関するベスト・プラクティスの説明を参照してください。
- **デフォルト・エビクターに関するベスト・プラクティス:** デフォルト存続時間 (TTL) エビクター (これは各 backingMap と共に作成されるデフォルトのエビクターです) のプロパティー。詳しくは、「*プログラミング・ガイド*」に記載されているデフォルト・エビクターに関するベスト・プラクティスの説明を参照してください。
- JVM の調整
- 230 ページの『WebSphere Real Time の使用』: このフィーチャーによって、ガーベッジ・コレクションはより予測可能になり、eXtreme Scale を使用するトランザクションの応答時間とスループットは安定した一貫性のあるものになります。

パフォーマンスのモニター

eXtreme Scale は、パフォーマンス・モニター機構も提供します。パフォーマンス・モニター統計およびツールに関しては、以下のセクションを参照してください。

- 289 ページの『PMI モジュール』
- 283 ページの『WebSphere Application Server PMI によるパフォーマンスのモニター』

第 11 章 用語集

この用語集には、WebSphere eXtreme Scale の用語と定義が含まれています。

この用語集では以下の相互参照が使用されています。

1. 「からを参照」は、ある用語から使用が推奨される同義語への参照、または頭字語あるいは省略語から完全な表現形式の定義への参照を読者に指示します。
2. 「からも参照」は、関連用語または対比用語を読者に示すものです。

他の IBM 製品の用語集を表示するには、www.ibm.com/software/globalization/terminology を参照してください。

アップグレード可能ロック. ペシミスティック・ロックを使用する場合に、キャッシュ・エントリーの更新意図を識別するロック。

アップストリーム (upstream). プロセスの開始 (アップストリーム) からプロセスの終了 (ダウンストリーム) へと流れる、フローの方向に関する用語。

宛先 (destination). バックエンド・システムまたは取引先に文書を配信するのに使用する出口点。

後書きキャッシュ (write-behind cache). ローダーを使用して、データベースに対する各書き込み操作が非同期に行われるキャッシュ。

アプリケーション (application). 特定のビジネス・プロセス (複数可) を直接サポートする機能を実現する 1 つ以上のコンピューター・プログラムまたはソフトウェア・コンポーネント。

アプリケーション・サーバー (application server). 分散ネットワーク内のサーバー・プログラムであり、アプリケーション・プログラムのための実行環境を提供する。

アプリケーション・プログラミング・インターフェース (API) (application programming interface (API)). 高水準言語で記述されたアプリケーション・プログラムがオペレーティング・システムまたは別のプログラムの特定のデータまたは機能を使用できるようにするインターフェース。

イテレーター (iterator). オブジェクトの集合を一度にステップスルーするために使用するクラスまたは構造。

イベント (event).

1. 操作、ビジネス・プロセス、またはヒューマン・タスクの完了または失敗などの状態の変更で、イベント・データのデータ・リポジトリーへの保管や、別のビジネス・プロセスを呼び出すなどの後続アクションをトリガーすることができる。

2. エンタープライズ情報システム (EIS) のデータに対する変更の 1 つ。アダプターによって処理され、EIS からのビジネス・オブジェクトを、変更を通知する必要があるエンドポイント (アプリケーション) に送信するために使用される。

インスタンス (instance). あるクラスに属するオブジェクトの特定のオカレンス。

インスタンス化 (instantiate). 抽象概念を具象化されたインスタンスで表現すること。

インストール・ターゲット (installation target). 選択されたインストール・パッケージがインストールされるシステム。

インストール・パッケージ (installation package). ソフトウェア製品のインストール可能単位。ソフトウェア製品パッケージは、そのソフトウェア製品の他のパッケージとは無関係に作動できる別々にインストール可能な単位である。

インターネット・プロトコル (Internet Protocol (IP)). 1 つのネットワークまたは相互接続ネットワークを介してデータを送付するプロトコル。このプロトコルは、高位プロトコル層と物理ネットワークの間の仲介として機能する。

インターフェース (interface). クラスのサービスまたはコンポーネントを指定するために使われる一連のオペレーションのこと。

インテリム・フィックス (interim fix). 正規にスケジュールされたフィックスパック、リフレッシュ・パック、またはリリースまでの間に、すべてのお客様で一般出荷可能になる認定された修正のこと。「フィックスパック (fix pack)」も参照。

インフォメーション・センター (information center). 製品に関する情報がまとめられており、複数製品へのサポートをユーザーに提供する。それぞれの製品から起動でき、ナビゲーション、検索エンジン、およびトピックのリストを表示するパネルが提供される。

インポート (import).

1. モジュール外部にあるサービスを取り込む(インポートする)ためのもの。

2. SCA モジュールが外部サービス (SCA モジュールにない外部サービス) に対して、ローカルであるかのようにしてアクセスするためのポイントとして定義される。インポートは、SCA モジュールとサービス・プロバイダー間のインターフェースを定義する。インポートには 1 つのバインディングと 1 つ以上のインターフェースが定義できる。

ウェイター (waiter). 接続を待機しているスレッド。

エージェント. ユーザーによる介入や定期スケジュールなしにユーザーまたは他のプログラムのためにアクションを実行し、結果をユーザーまたはプログラムに報告するプログラム。

永続データ・ストア (persistent data store). セッションの境界を越えて維持され、作成元のプログラムまたはプロセスの実行後にも継続して存在するイベント・データ用の不揮発性ストレージ (データベース・システムなど)。

エクスポート (export). Service Component Architecture (SCA) モジュールからの公開インターフェースで、モジュール外部にビジネス・サービスを提供する。エクスポートは、サービス・リクエスターにサービスへアクセスさせる方法 (例えば Web サービスとして) を定義するバインディングを持つ。

エクスポート・ファイル (export file).

1. インバウンド操作の開発過程で作成された、インバウンド処理の構成設定を含むファイル。

2. エクスポートしたデータを含むファイル。

エディション. 成果物セットの、特定バージョンにおける連続したデプロイメントの世代。

エディター領域 (editor area). Eclipse および Eclipse ベースの製品では、編集作業のためにファイルが開かれる、ワークベンチ・ウィンドウ内のエリア。

エラー (error). 値や状態が、計算したものと実際のものとの間で、監視したものと指定したものとの間で、あるいは測定したものと理論的に正しいものとの間で、一致しない状態。

エラー・ログ・ストリーム (error log stream). 事前定義フォーマットを使用して伝送されるエラー情報の連続フロー。

エンタープライズ Bean (enterprise bean). ビジネス・タスクまたはビジネス・エンティティーを実装し、EJB コンテナー内に常駐するコンポーネント。エンティティー Bean、セッション Bean、およびメッセージ駆動型 Bean はすべてエンタープライズ Bean である。Bean も参照。

エンタープライズ・アーカイブ (enterprise archive (EAR)). Java EE 標準で定義され、Java EE アプリケーションを Java EE アプリケーション・サーバーにデプロイするために使用される、特殊なタイプの JAR ファイル。EAR ファイルには、EJB コンポーネント、デプロイメント記述子、および個々の Web アプリケーション用の Web アーカイブ (WAR) ファイルが含まれる。「Web アーカイブ (Web archive)」も参照。

エンタープライズ・アプリケーション・プロジェクト (enterprise application project (EAR project)). デプロイメント記述子および IBM 拡張文書、デプロイメント記述子に定義されているすべての Java EE モジュールに共通のファイルを含むフォルダーやファイルの構造および階層。

エンタープライズ・サービス・バス (enterprise service bus (ESB)). アプリケーションとサービスを統合するための高い柔軟性を持つ接続インフラストラクチャー。柔軟で扱いやすいサービス指向アーキテクチャーの実装への手引きとなる。

エンティティー.

1. データベース表の行またはマップ内のエントリーを表わす単一の Java クラス。

2. XML などのマークアップ言語において、例えば文書内に頻繁に繰り返されるテキストや特殊文字を組み込むために、1 単位として参照できる文字の集合。

エンティティー Bean (entity bean). EJB プログラミングにおいて、データベース内で保守される永続的データを表すエンタープライズ Bean。各エンティティー Bean は独自の ID を持つ。

エンドポイント (endpoint).

1. JCA アプリケーションまたはエンタープライズ情報システムからのイベントを利用するその他のクライアント利用者。

2. セッションの発信元または宛先であるシステム。

エンドポイント・リスナー (endpoint listener). Web サービスの着信メッセージが、サービス統合バスで受信されるポイントまたはアドレス。

エントリー・ブレークポイント (entry breakpoint). コンポーネント・エレメントに設定されるブレークポイント。コンポーネント・エレメントが呼び出される前にヒットする。

オートディスカバリー (autodiscovery). ファイル・システム、外部レジストリー、またはその他のソース内のサービス成果物を発見すること。

オートノミック・マネージャー (autonomic manager). 他のソフトウェアまたはハードウェア・コンポーネントの動作を、人間が管理するように管理するポリシーによって構成された、一連のソフトウェアまたはハードウェア・コンポーネント。オートノミック・マネージャーには、モニター、分析、計画、実行の各コンポーネントからなる制御ループが組み込まれている。

オープン・ソース (open source). ソース・コードを公に使用または修正することができるソフトウェアに関する用語。通常、オープン・ソース・ソフトウェアは、公開のコラボレーションとして開発され、無償で使用可能にされる。ただし、その使用と再配布はライセンスの制約を受ける。 Linux は、オープン・ソース・ソフトウェアとしてよく知られている。

オブジェクト (object). オブジェクト指向設計およびプログラミングにおいて、データとそのデータに関連付けられた操作で構成される、1 つのクラスを具体的に実現したもの (インスタンス)。オブジェクトには、クラスで定義されたインスタンス・データが含まれているが、クラスはデータに関連付けられた操作を持っている。

オブジェクト・リクエスト・ブローカー (Object Request Broker (ORB)). オブジェクト指向プログラミングでは、オブジェクトが要求や応答を交換することを透過的に可能にすることによって、仲介としてサービスを提供するソフトウェアのこと。

オブジェクト指向プログラミング (object-oriented programming). データの抽象化と継承の概念に基づいたプログラミング・アプローチ。プロシージャー型プログラミング技法と異なり、オブジェクト指向プログラミングは、何らかのものを達成する方法に専念するのではなく、代わりに、問題がどのデータ・オブジェクトから構成されているか、およびそれらを操作する方法に専念する。

ガーベッジ・コレクション (garbage collection). プログラム・セグメントまたは非アクティブ・データのスペースを再利用するため、メモリーを検索するルーチン。

下位ノード (child node). 別のノードの有効範囲内にあるノード。

カスタマイズ・インストール・パッケージ (CIP). カスタマイズされたインストール・イメージ。1 つ以上の保守パッケージ、スタンドアロン・サーバー・プロファイルからの構成アーカイブ・ファイル、1 つ以上のエンタープライズ・アーカイブ・ファイル、スクリプト、および結果としてのインストールのカスタマイズに役立つその他のファイルを組み込むことができる。

仮想化 (virtualization). 他のシステムがリソースと対話する手段からリソースの特性をカプセル化する技法。

仮想ホスト (virtual host). 単一のホスト・マシンを複数のホスト・マシンのように機能させることが可能な構成。ある仮想ホストに関連付けられたリソースは、別の仮想ホストに関連付けられたリソースとデータを共有することはできない。このことは、これらの仮想ホストが同じ物理マシンを共有している場合であっても該当する。

仮想マシン (virtual machine). コンピューティング・デバイスの抽象的な仕様。さまざまな方法でソフトウェアおよびハードウェアに実装できる。

カタログ (catalog). コンテナーのタイプに基づいて、プロセス、データ、リソース、組織、またはレポートをプロジェクト・ツリーで保持するコンテナー。

カタログ・サービス. 断片の配置を制御し、コンテナーのヘルスをディスカバーおよびモニターするサービス。

カテゴリー (category). 共用の属性または品質に基づいてエレメントをグループ化する、構造ダイアグラムで使用されるコンテナー。

可用性.

1. ユーザーにアプリケーションとデータのアクセスおよび使用を許可する条件。

2. リソースがアクセス可能となる時間。例えばある請負業者のアベイラビリティーは、平日は毎日午前 9 時から午後 5 時、土曜日は午前 9 時から午後 3 時までとなる。

環境 (environment). 機能のパフォーマンスをサポートするのに使用される論理リソースおよび物理リソースの名前付きコレクション。

環境変数 (environment variable). オペレーティング・システムまたは他のプログラムの動作方法を指定する変数、あるいはオペレーティング・システムが認識するデバイスを指定する変数。

管理 Bean (MBean) (Managed Bean (MBean)). Java Management Extensions (JMX) 仕様において、リソースとそのインスツルメンテーションを実装する Java オブジェクト。

管理者 (administrator). アクセス許可およびコンテンツ・マネージメントなどの管理タスクの担当者。管理者は権限レベルをユーザーに付与することもできる。

キー.

1. 暗号の数値であり、メッセージをデジタル署名、検証、暗号化、または暗号化解除するために使用される。

2. モニター・コンテキストによってトラッキングされる実際のエンティティーを特徴づけ、一意的に識別するための情報。

キーワード (keyword). プログラミング言語、人工言語、アプリケーション、またはコマンドの事前定義語。

キャッシュ・インスタンス・リソース (cache instance resource). Java Platform, Enterprise Edition (Java EE) アプリケーションがデータを保管、配布、および共有できる場所。

キャッシュ複製 (cache replication). 同じ複製ドメイン内の別のサーバーとの、キャッシュ ID、キャッシュ・エントリー、およびキャッシュ無効化の共用。

共用ロック. 同時に実行するアプリケーション・プロセスを、データベース・データの読み取り専用操作に制限するロック。

許可 (authorization). ユーザー、システム、またはプロセスに、オブジェクト、リソース、または機能への完全なアクセス権限または制限付きのアクセス権限を付与するプロセス。

許可テーブル (authorization table). 特定のリソースに対してクライアントに許可されたアクセス権限を識別する、ユーザーのロールまたはグループ・マッピング情報を含むテーブル。

許可ポリシー (authorization policy). ビジネス・サービスをポリシー・ターゲットとするポリシーで、その契約には、チャネル・アクションを実行するための許可を付与する 1 つ以上のアサーションが含まれる。

区画化機能. エンタープライズ Bean、HTTP トラフィック、およびデータベース・アクセスの区画化の概念をサポートする、プログラミング・フレームワークおよびシステム管理インフラストラクチャー。

組み込みサーバー (embedded server). 既存のプロセス内に存在し、そのプロセス内で始動および停止されるカタログ・サービスまたはコンテナー・サーバー。

クライアント (client). サーバーに対してサービスを要求するソフトウェア・プログラムまたはコンピューター。「ホスト (host)」も参照。

クライアント/サーバー (client/server). 分散データ処理において、あるサイトのプログラムが他のサイトのプログラムへの要求を送信して応答を待つ形の対話モデル。要求を出すプログラムをクライアントと言い、応答するプログラムをサーバーと言う。

クライアント・アプリケーション (client application). ワークステーション上で実行されるクライアントにリンクするアプリケーションで、サーバーのキューイング・サービスを利用して、アプリケーションからのアクセスを提供する。

クラス. オブジェクト指向の設計またはプログラミングにおいて、オブジェクトを生成するために用いる、オブジェクトに共通の定義および共通のプロパティー、操作、振る舞いを持つモデルまたはテンプレート。オブジェクトは、クラスのインスタンスである。

クラス・ファイル (class file). Java ソース・ファイルをコンパイルして生成される中間コード・ファイル。

クラス・ローダー (class loader). クラス・ファイルの検索およびロードを行う Java 仮想マシン (JVM) のパーツ。クラス・ローダーは、アプリケーションのパッケージ化、およびアプリケーション・サーバーにデプロイされたパッケージ済みアプリケーションの実行時動作に影響を与える。

クラスター (cluster). ワークロード・バランシングおよびフェイルオーバーの目的で協調して動作する複数のアプリケーション・サーバーから構成されるグループ。

クラスパス (class path). プログラムが実行時に動的にロードできるリソース・ファイルまたは Java クラスを含むディレクトリーおよび JAR ファイルのリスト。

クラス階層 (class hierarchy). 単一継承を共有するクラス間の関係。

グループ.

1. 保護リソースに対するアクセス権限を共用できるユーザーの集合。

2. 交換内の関連する文書セット。交換には、多数のグループを含めることができる (グループを含めないことも可能)。

3. プレースでは、1 つのプレースでメンバーシップのためにグループになっている複数の人物。

クレデンシャル. Java 認証・承認サービス (JAAS) フレームワークにおいて、セキュリティー関連属性を所有するサブジェクト・クラス。これらの属性には、新規サービスに対してサブジェクトを認証するのに使用される情報を含めることができる。

グローバル (global).

1. ワークスペースの任意のプロセスが使用可能なエレメントに関する用語。グローバル・エレメントは、プロジェクト・ツリーに表示され、複数のプロセスで使用できる。タスク、プロセス、リポジトリー、およびサービスは、グローバル (プロジェクトの任意のプロセスによって参照される) かローカル (単一のプロセスに固有) のいずれかとなる。

2. 複数のプログラムまたはサブルーチンに有効な情報に関すること。

グローバル・インスタンス ID (global instance identifier). アプリケーションまたはエミッターによって生成され、イベント識別の 1 次キーとして使用されるグローバルな固有 ID。

グローバル・エレメント (global element). XML において、複合型定義の一部としてではなく、スキーマ・エレメントの子として宣言されるエレメント。グローバル・エレメントは、ref 属性を使用する 1 つ以上のコンテンツ・モデル内で参照できる。

グローバル・セキュリティー (global security). 環境内で実行されているすべてのアプリケーションに適用され、セキュリティーが使用されるかどうか、認証の際に使用されるレジストリーのタイプ、およびその他の値 (多くはデフォルトで動作する) を決定する。

グローバル・トランザクション (global transaction). 分散トランザクション環境で 1 つ以上のリソース・マネージャーによって実行され、外部トランザクション・マネージャーによって調整されるリカバリー可能な作業単位。

グローバル属性 (global attribute). XML において、複合型定義の一部としてではなく、スキーマ・エレメントの子として宣言される属性。グローバル属性は、ref 属性を使用する 1 つ以上のコンテンツ・モデル内で参照できる。

グローバル変数 (global variable). 変換中に割り当てられた値を保持および操作するために使用する変数。マップ間および文書変換の間で共用される。Data Interchange Services のマッピング・コマンド言語でサポートされる 3 つの変数タイプの一つである。

継承 (inheritance). 既存のクラスを他のクラスを作成するための基礎として使用するオブジェクト指向プログラミング技法。継承によって、より一般化されたエレメントの構造および動作が、より特殊化されたエレメントに組み込まれる。

結合 (join).

1. 決定または fork の後に並列処理のパスを再結合および同期化するプロセス要素。結合は、プロセスの続行を許可する前に、各着信ブランチで入力の到着を待機する。

2. 2 つの表から (通常、結合列を指定する結合条件に基づいて) データを取得できる SQL 関係演算。

3. リンクの動作を決定する着信リンク上の構成。

公開 (public).

1. オブジェクト指向プログラミングにおいて、すべてのクラスにアクセス可能なクラス・メンバーのこと。

2. Java プログラミング言語において、他のクラス内に存在するエレメントがアクセスできるメソッドまたは変数を指す。

高可用性 (high availability (HA)). ノードまたはデーモンに障害が発生した場合に、ワークロードをクラスター内に残っているノードに再配布できるように再構成されるクラスター化されたシステムを指す。

構造化照会言語 (SQL) (Structured Query Language (SQL)). リレーショナル・データベース内のデータを定義および操作するための標準化言語。

構文 (syntax). コマンドまたはステートメントを構成する際の規則。

コヒーレント・キャッシュ. すべてのクライアントが同一のデータを見れるよう、整合性を維持するキャッシュ。

コマンド Bean (command bean). execute() メソッドを使用して、単一操作を呼び出すことができるプロキシー。

コマンド行. コマンド、オプション番号、または選択を入力できるモニター上のブランク行。

コレクション証明書ストア (collection certificate store). 中間証明書または証明書取り消しリスト (CRL) のコレクション。検証のための証明書チェーンを構築するために証明書パスで使用される。

コンテナー・サーバー. 複数の断片をホスティングできるサーバー・インスタンス。1 台の Java 仮想マシン (JVM) は、複数のコンテナー・サーバーをホスティングできる。

コンバーター (converter). Enterprise JavaBeans (EJB) プログラミングでは、データベース表記をオブジェクト・タイプに (またはその逆に) 変換するクラス。

コンパイル時間 (compile time). コンピューター・プログラムを実行可能プログラムにコンパイルするための時間。

コンパイル単位 (compilation unit). プログラムのソース・コード群を分割してコンパイルする際に、分割された個々の部分。

コンポーネント (component).

1. 特定の機能を実行し、他のコンポーネントやアプリケーションと共に動作する、再使用可能なオブジェクトまたはプログラム。

2. Eclipse では、別個の機能セットを供給するために、一緒に動作する 1 つ以上のプラグイン。

コンポーネント・インスタンス (component instance). 同じコンポーネントの他のインスタンスと並列に実行できる実行コンポーネント。

コンポーネント・エレメント (component element). ビジネス・プロセスのアクティビティーまたは Java Snippet、あるいはメディエーション・フローのメディエーション・プリミティブまたはノードのような、ブレークポイントを設定できるコンポーネント内のエンティティー。

コンポーネント・テスト (component test). エンタープライズ・アプリケーションの 1 つ以上のコンポーネント (Java クラス、EJB Bean、または Web サービスが含まれる場合がある) の自動化されたテスト。

コンマ区切りファイル (comma delimited file). レコード内のフィールドがコンマで区切られているファイル。

サーバー (server). 別のソフトウェア・プログラムまたは別のコンピューターにサービスを提供するソフトウェア・プログラムまたはコンピューター。「ホスト (host)」も参照。

サーバー・クラスター (server cluster). 通常は別々の物理マシン上に配置され、内部に同じアプリケーションが構成されているが、単一の論理サーバーとして機能するサーバーのグループ。

サーバント領域 (servant region). 負荷が増大すると動的に開始し、負荷が軽減されると自動的に停止する仮想ストレージの連続区域。

サービス・レベル・アグリーメント (service level agreement (SLA)). 可用性やパフォーマンスなどの測定可能な目標に関して、期待されるサービス・レベルを明記した顧客とサービス・プロバイダー間の契約。

サービスの品質 (QoS) (quality of service (QoS)). アプリケーションが要求する一連の通信特性。サービスの品質 (QoS) は、特定の伝送優先順位、経路信頼性のレベル、およびセキュリティー・レベルを定義する。

サーブレット (servlet). Web サーバー上で稼働し、Web クライアントの要求に応答して動的コンテンツを生成することにより、サーバー機能を拡張する Java プログラム。一般に、サーブレットは、データベースを Web に接続するために使用される。

再帰 (recursion). プログラムまたはルーチンが自分自身を呼び出して、ある操作中で一連のステップを実行するプログラミング手法。この手法では、各ステップが前のステップからの出力内容を使用する。

サイレント・インストール. コンソールに対してメッセージは送信されず、メッセージおよびエラーがログ・ファイルに格納されるインストール。サイレント・インストールでは、データ入力に応答ファイルを使用できる。

サイレント・モード (silent mode). GUI 表示なしでコマンド行から製品コンポーネントをインストールまたはアンインストールする方法。サイレント・モードを使用する場合、インストールまたはアンインストール・プログラムで必要なデータは、コマンド行で直接指定するか、(オプション・ファイルまたは応答ファイルと呼ばれる) ファイル内に指定する。

索引. キーの値によって論理的に順序付けられているポインターのセット。索引を利用すると、データに迅速にアクセスでき、また表にある行のキー値の固有性を高めることができる。

サブクラス (subclass). Java において、特定のクラスから継承を通じて派生したクラス。

シェル・スクリプト (shell script). オペレーティング・システムのシェルで解釈されるプログラムまたはスクリプト。

しきい値 (threshold). シミュレーションの中断に適用される設定。あるイベントが指定の比率で発生した場合、既存の条件に基づいて、いつプロセス・シミュレーションを停止すべきかを定義する。

システム分析者 (systems analyst). ビジネス要件からシステム定義およびソリューションを作成する責任を持つ専門家。

実行トレース (execution trace). 統合テスト・クライアントの「イベント」ページで、階層形式で記録および表示される一連のイベント。

シャーシ (chassis). 各種電子部品が取り付けられる金属製のフレーム。

修飾子 (qualifier). 別の一般的な複合エレメントまたは単一エレメントに固有の意味を提供する単一エレメント。修飾子は、単一または複数のオカレンスをマッピングする際に使用される。修飾子を使用すると、名前の 2 番目の部分 (通常は ID と呼ばれる) の解釈で使用する名前空間を指定することもできる。

種別 (classifier). プロセス要素をグループ化およびカラー・コーディングするのに使用される特殊属性。

照会 (query).

1. 特定の条件に基づいてデータベースからの情報を求める要求。例えば、顧客テーブル内で ¥10,000 を上回る残高のすべてのお客様のリストを求める要求。

2. 1 つ以上のモデル・エレメントについての情報を求める再使用可能な要求。

署名者証明書 (signer certificate). 通常はトラストストア・ファイルにあるトラステッド証明書エントリー。

シリアライザー (serializer). オブジェクト・データを別のフォーム (例えば、バイナリー、または XML) に変換するためのメソッド。

シリアライゼーション (serialization). オブジェクト指向プログラミングにおいて、プログラム・メモリーから通信メディアに順次データを書き込むこと。

シン・アプリケーション・クライアント (thin application client). エンタープライズ Bean との対話が可能な、軽量でダウンロード可能な Java アプリケーション・ランタイム。

シン・クライアント (thin client). ソフトウェアがほとんどまたはまったくインストールされていないが、接続先のネットワーク・サーバーで管理および配信されるソフトウェアへのアクセス権限を持つクライアント。シン・クライアントは、ワークステーションなどの全機能を搭載したクライアントの代替である。

スクリプティング (scripting). アプリケーション構築の基礎として既存のコンポーネントを再利用するプログラミング・スタイル。

スクリプト (script). 一連のコマンドをファイルにまとめたもの。ファイルの実行時に特定の機能を実行する。スクリプトは、その実行時に解釈される。

スケーラビリティー. プロセッサー、メモリー、ストレージなどのリソースを追加する際のシステムの拡張能力。

スケルトン (skeleton). 実装クラスのスケルトン。

スコープ (scope).

1. システム・リソースをその範囲内で使用できる境界の指定。

2. Web サービスにおいて、呼び出し要求のサービスを行うオブジェクトの存続期間を識別するプロパティー。

スタック (stack). 一般に一時的なレジスター情報、パラメーター値、サブルーチンの戻りアドレスなどの情報を保管するメモリー内の領域。後入れ先出し (LIFO) の原則に基づいている。

スタンドアロン (stand-alone). ほかのどのデバイス、プログラム、システムからも独立していること。ネットワーク環境において、スタンドアロン・マシンは、必要なすべてのリソースにローカルにアクセスする。

スタンドアロン・サーバー (stand-alone server). サーバー・プロセスの開始および停止を行う、オペレーティング・システムから管理されるカタログ・サービスまたはコンテナー・サーバー

ストリング (string). プログラム言語における、テキストを保管および操作するために使用するデータの形式。

スループット (throughput). 一定期間に渡ってコンピューターやプリンターなどのデバイスで実行される作業量の指標 (1 日当たりのジョブ数など)。

スレッド (thread). プロセスの制御下にあるコンピューター命令のストリーム。 オペレーティング・システムによっては、スレッドとはプロセスでの最小単位の演算命令のこと。複数のスレッドを並行して実行し、それぞれのスレッドで異なるジョブを実行することができる。

スレッド競合 (thread contention). あるスレッドが、別のスレッドが保持しているロックまたはオブジェクトを待機している状態。

静的 (static). Java プログラミング言語のキーワードの一つであり、変数をクラス変数として定義するために使用される。

セキュリティー・トークン (security token). クライアントによって生成された資格証明のセットを表し、名前、パスワード、ID、キー、証明書、グループ、特権などを含めることができる。

セキュリティー管理者 (security administrator). ビジネス・データおよびプログラム機能へのアクセスを制御する担当者。

セッション.

1. ネットワーク上の 2 つのステーション、ソフトウェア・プログラム、またはデバイス間の論理接続または仮想接続。これにより、2 つのエレメントがデータ通信およびデータ交換を行うことができる。

2. 同じブラウザーで同じユーザーから発信される、サーブレットへの一連の要求。

3. Java EE において、複数の HTTP 要求にわたる Web アプリケーションとユーザーとの対話を追跡するためにサーブレットが使用するオブジェクト。

セッション・アフィニティー (session affinity). クライアントが常に同じサーバーに接続するようなアプリケーションの構成方法。この構成では、最初に接続した後、クライアント要求が常に同じサーバーに送られるので、ワークロード管理を行うことはできない。

セル (cell).

1. 同じデプロイメント・マネージャーにフェデレートされて、高可用性を持つコア・グループを含めることができる、管理対象プロセスのグループ。

2. ランタイム・コンポーネントをホストする 1 つ以上のプロセス。それぞれが名前付きのコア・グループを 1 つ以上持つ。

セル・スコープ・バインディング (cell-scoped binding). バインディングがノードまたはサーバーに固有でなく、関連がない場合のバインディング・スコープ。このタイプの名前バインディングは、セルの永続的なルート・コンテキストに従って作成される。

ゾーン・ベース・サポート (zone-based support). ルール・ベースの断片配置を有効にして、階、建物、地域などが異なるさまざまなデータ・センターにまたがって断片を配置することで、グリッドの可用性を高める機能。

操作 (operation). あるオブジェクトが呼び出されて実行する、機能や照会の実装。

粗視化 (coarse-grained). オブジェクト群を論理的にハイレベル、要約レベルから観察する手法。

組織 (organization). 規定の目標を達成するために人々が協力し合うエンティティーのこと。例えば、企業、会社、工場など。

存続時間 (time to live). キャッシュに存在する項目が破棄されるまでの時間を秒単位で表したもの。

ダーティー読み取り (dirty read). いかなるロック・メカニズムも伴わない読み取り要求。つまり、データを読み取ることはできるが、その後ロールバックされた結果として、読み取られたものとデータベースに入っているものとが一致しなくなることがある。

タイマー (timer). 特定の時点で出力を生成するタスク。

タイミング制約 (timing constraint). 1 つのメソッド呼び出しまたは一連のメソッド呼び出しの期間を測定するために使用される特殊な検証アクション。

ダウンストリーム (downstream). フローの方向に関して、プロセスの最初のノード (アップストリーム) からプロセスの最後のノード (ダウンストリーム) に向かう方向のこと。

ダッシュボード (dashboard). ビジネス・データをグラフィカルに示す 1 つ以上のビューアーを含むことが可能な Web ページ。

断片. 区画のインスタンス。断片は基本またはレプリカとすることができる。

データ・グリッド (data grid). テラバイトまたはペタバイトのデータにアクセスするためのシステム。

デーモン (daemon). ネットワーク制御など、連続的または周期的な機能をバックグラウンドで実行するプログラム。

デジタル証明書 (digital certificate). 個人、システム、サーバー、会社、またはその他のエンティティーを識別するために使用され、公開鍵をそのエンティティーに関連付けるために使用される電子文書。デジタル証明書は、認証局によって発行され、その認証局によってデジタル署名される。

デシリアライゼーション (deserialization). シリアライズされた変数をオブジェクト・データに変換するメソッド。

デッドロック (deadlock). 2 つの独立した制御スレッドがブロックされ、一方が何らかのアクションを実行するため他方を待っている状態。競合状態を避けるため、同期メカニズムの追加からデッドロックが生じることがよくある。

デプロイ. 作動環境にファイルを置いたりソフトウェアをインストールしたりすること。Java Platform, Enterprise Edition (Java EE) では、デプロイされるアプリケーションのタイプに適したデプロイメント記述子の作成を伴う。

デプロイ・フェーズ (deploy phase). 「デプロイメント・フェーズ (deployment phase)」も参照。

デプロイメント・コード (deployment code). アプリケーション開発者によって記述された Bean 実装コードが特定の EJB ランタイム環境で動作できるようにする追加コード。デプロイメント・コードは、アプリケーション・サーバー・ベンダーが提供するツールで生成できる。

デプロイメント・ディレクトリー (deployment directory). アプリケーション・サーバーがインストールされたマシン上で公開サーバー構成と Web アプリケーションが配置されるディレクトリー。

デプロイメント・トポロジー (deployment topology). デプロイメント環境でのサーバーおよびクラスターの構成と、それらの間の物理関係および論理関係。

デプロイメント・フェーズ (deployment phase). アプリケーションのホスティング環境の作成とそれらのアプリケーションのデプロイメントの組み合わせを含むフェーズ。これにはアプリケーションのリソース依存、操作条件、キャパシティー要件、および保全性とアクセス権限の制約の解決を含む。

デプロイメント・ポリシー (deployment policy). システム数、サーバー数、区画数、レプリカ数 (レプリカ・タイプを含む)、各サーバーのヒープ・サイズなど、さまざまな項目に基づいて eXtreme Scale 環境を構成するためのオプションの手段。

デプロイメント・マネージャー. 論理グループまたは他のサーバーのセルの操作を管理するサーバー。

デプロイメント環境 (deployment environment). 構成済みのクラスター、サーバー、およびミドルウェアの組み合わせによって、ソフトウェア・モジュールをホストするための環境を提供する。例えば、デプロイメント環境はメッセージの宛先のホスト、ビジネス・イベントのプロセッサーまたはソーター、および管理プログラムを含むことがある。

デプロイメント記述子 (deployment descriptor). 構成オプションおよびコンテナー・オプションを指定することにより、モジュールまたはアプリケーションをデプロイする方法を記述している XML ファイル。例えば、 EJB デプロイメント記述子は、エンタープライズ Bean を管理、制御する方法に関する情報を EJB コンテナーに渡す。

伝送制御プロトコル/インターネット・プロトコル (Transmission Control Protocol/Internet Protocol (TCP/IP)). 業界標準の独占されていない通信プロトコルのセットのことで、異なる種類の相互接続ネットワークにおいて、アプリケーション間の信頼性のあるエンドツーエンド接続を提供する。

トークン (token).

1. シミュレーションの実行中にプロセス・インスタンスの現在の状態を追跡するために使用するマーカー。
2. ネットワーク上で転送を行うときの許可または一時的な制御を示す特定のメッセージまたはビット・パターン。

同期化 (synchronize). ある機能または成果物を別のものと一致するように加算、減算、または変更すること。

同期複製 (synchronous replica). データの整合性を保証するため、プライマリー断片においてトランザクションの一部として更新を受信する断片。この場合、非同期複製に比べて応答時間が増す可能性がある。

同期プロセス (synchronous process). 要求/応答オペレーションを起動することによって開始されるプロセス。プロセスの結果は、同じオペレーションによって戻される。

統合開発環境 (IDE) (integrated development environment (IDE)). ソース・エディター、コンパイラー、デバッガーなど、一連のソフトウェア開発ツールのこと。単一ユーザー・インターフェースからアクセス可能。

動的キャッシュ (dynamic cache). あるサービスの中のサーブレット、Web サービス、WebSphere コマンドを含むいくつかのキャッシング・アクティビティーの集まりで、構成情報を共有しパフォーマンスが向上するように機能する。

動的クラスター. クラスター・メンバーから収集されたパフォーマンス情報に基づき、重みを使用して、クラスター・メンバーのワークロードを動的にバランスさせるサーバー・クラスター。

トポロジー (topology). ネットワーク内のネットワーキング・コンポーネントまたはノードの場所に関する物理的または論理的なマッピング。一般的なネットワーク・トポロジーとしては、バス、リング、スター、ツリーなどがある。

ドメイン (domain). あるドメイン内のリソースを表現する別のオブジェクトが入っているオブジェクト、アイコン、およびコンテナー。ドメイン・オブジェクトを使用すると、これらのリソースを管理できる。

ドメイン・ネーム・システム (DNS). ドメイン・ネームを IP アドレスにマップする分散データベース・システム。

トラストストア・ファイル (truststore file). トラステッド・エンティティーの公開鍵が入っている鍵データベース・ファイル。

ドロップダウン (drop-down). 「プルダウン (pull-down)」を参照。

名前空間 (namespace). すべての名前が固有である論理コンテナー。成果物の固有 ID は、名前空間と、成果物のローカル名で構成される。

認証 (authentication). コンピューター・システムのユーザーが本人であることを証明するセキュリティー・サービス。このサービスを実装する一般的な手段として、パスワードやデジタル署名などがある。認証は許可とは異なり、システム・リソースへのアクセスの許可または拒否とは関係がない。

認証済みユーザー (authenticated user). 有効なアカウント (ユーザー ID およびパスワード) でポータルにログインしたポータル・ユーザー。認証済みユーザーはすべてのパブリック・プレースへのアクセス権限を持つ。

認証別名 (authentication alias). リソース・アダプターおよびデータ・ソースへのアクセスを許可する別名。認証別名にはユーザー ID およびパスワードなどの認証データが含まれる。

ノード・エージェント (node agent). ノード上のすべてのアプリケーション・サーバーを管理し、管理セル内のノードを表す管理エージェント。

パーシスタンス (persistence).

1. セッション境界を超えて保持されるデータ、または作成元のプログラムまたはプロセスの実行後も引き続き存在するオブジェクトの特性。通常は、データベース・システムなどの不揮発性ストレージに存在する。

2. Java EE において、エンティティー Bean の状態をそのインスタンス変数と基本データベース間で転送するためのプロトコル。

パーシスト (persist). 通常、データベース・システムやディレクトリーなどの不揮発性ストレージ内で、セッション境界を越えて保持されること。

ハートビート (heartbeat). エンティティーがまだアクティブであることを通知するために別のエンティティーに送信するシグナル。

パーミッション (permission). ローカル・ファイルの読み取りと書き込み、ネットワーク接続の作成、ネイティブ・コードのロードなどのアクティビティーを実行する権限。

排他ロック. 同時に実行するアプリケーション・プロセスがデータベースのデータにアクセスできないようにするロック。「共用ロック (shared lock)」も参照。

バイトコード (bytecode). Java コンパイラーによって生成され、Java インタープリターによって実行される、マシンから独立したコード。

バイナリー形式 (binary format). 各フィールドの長さが 2 バイトまたは 4 バイトであるような 10 進値表現。フィールドの左端のビットは符号 (+ または -) であり、フィールドの残りのビットは数値である。正数の符号ビットは 0 である。正数は true 形式で表現される。負数の符号ビットは 1 である。負数は 2 の補数形式で表現される。

派生 (derivation). オブジェクト指向プログラミングで、1 つのクラスから別のクラスへの改良または拡張。

パッケージ (package).

1. Java プログラミングにおけるタイプのグループ。パッケージは、パッケージ・キーワードによって宣言される。

2. 文書の内容を囲むラッパーで、インターネット経由で文書を送信するのに使用するフォーマットを定義する。RNIF、AS1、および AS2 など。

3. コンポーネントを組み立ててモジュールにし、モジュールを組み立ててエンタープライズ・アプリケーションにすること。

発生 (fire). オブジェクト指向プログラミングにおいて、状態遷移を起こすこと。

反復 (iteration). 「ループ (loop)」を参照。

汎用オブジェクト (generic object). 概念、カスタム・エンティティー、またはコレクションを参照するために API 呼び出しおよび XPath 式で使用するオブジェクト。例えば、XPath 式 /WSRR/GenericObject は、WebSphere Service Registry and Repository からすべての概念を取得する。

微細化 (fine-grained). オブジェクトを個別に詳細に見ていくこと。

非推奨 (deprecated). サポートされているが推奨されなくなり、廃止される可能性のあるエンティティー (プログラミング・エレメントまたはフィーチャーなど) について使用される言葉。

非同期 (asynchronous). 時間内に同期しないイベント、あるいは定期的または予測可能な時間間隔で発生しないイベントに関する用語。

非同期複製 (asynchronous replica). トランザクションのコミット後に更新を受信する断片。この方式は、同期複製に比べて高速であるが、プライマリー断片の背後のいくつかのトランザクションが非同期複製となる場合があるため、データ損失が発生する可能性がある。

非同期メッセージング (asynchronous messaging). プログラムがメッセージ・キューにメッセージを入れたら、メッセージへの応答を待たずに次の処理に進める、プログラム間での通信方式。

非武装地帯 (demilitarized zone (DMZ)). インターネットに見られるような、企業のイントラネットと公衆ネットワークとの間に保護層として追加された、複数のファイアウォールを含む構成。

ビルド・パス (build path). Java ソース・コードのコンパイル中に、別のプロジェクトにある参照クラスを検出するために使用されるパス。

ビルド計画 (build plan). 成果物をビルドするために必要な処理を定義し、処理が行われるマシンを指定するための XML ファイル。

ビルド時のデータ (build time data). EDI 標準、レコード指向データ文書タイプ、およびマップなど、変換プログラムで使用されないオブジェクト。

ビルド定義ファイル (build definition file). カスタマイズ・インストール・パッケージ (CIP) のコンポーネントと特性を特定する XML ファイル。

ブートストラッピング (bootstrapping). ネーミング・サービスの初期参照を取得するプロセス。ブートストラップ設定およびホスト名が、Java Naming and Directory Interface (JNDI) 参照の初期コンテキストを形成する。

ブートストラップ (bootstrap). システムを初期化する一連の処理。

ファイアウォール (firewall). セキュア・ネットワークに入ったり出たりする承認されないトラフィックを阻止するために使われるネットワーク構成のこと。通常はハードウェアおよびソフトウェアの両方が使われる。

ファクトリー (factory). オブジェクト指向プログラミングにおいて、別のクラスのインスタンスを作成するために使用するクラスのこと。ファクトリーを使用すると、新たな機能追加をしたい特定クラスのオブジェクト作成をそこだけで行うことができ、あちこちコード変更をしないで済む。

フィックスパック (fix pack). 出荷スケジュールが決められたリフレッシュ・パック、製造リフレッシュ、リリースの間に提供される、累積フィックスがまとめられたもの。お客様が特定の保守レベルに移行できることを意図したもの。「インテリム・フィックス (interim fix)」も参照。

フェイルオーバー (failover). ソフトウェア、ハードウェア、またはネットワークの障害が発生した場合に、冗長システムまたは待機システムに自動的に切り替わること。

フォーク (fork). 同時に並列処理される処理パスに対して、入力のコピーをそれらに渡すためのプロセス要素のこと。

フォルダー (folder). オブジェクトをまとめるために使用するコンテナー。

副照会 (subquery). SQL の述部で使用される副選択。他の SQL ステートメントの WHERE 節または HAVING 節内の select ステートメントなど。

複製 (replication). 複数のロケーションで定義済みのデータ・セットを保守するプロセス。複製には、あるロケーション (ソース) の指定された変更を、別のロケーション (ターゲット) にコピーして、両方のロケーションのデータを同期化することが含まれる。

プラグイン. 既存のプログラム、アプリケーション、またはインターフェースに機能を追加する、個別にインストール可能なソフトウェア・モジュール。

プリミティブ型 (primitive type). Java におけるデータ型のカテゴリー。その型に対する適切なサイズおよび形式 (数値、文字、またはブール値) の単一の値を含む変数を記述する。プリミティブ型の種類の例としては、byte、short、int、long、float、double、char、boolean がある。

ブレークポイント (breakpoint). プロセスまたはプログラマチック・フローでのマークを付けられたポイントで、ポイントに到達するとフローが一時停止し、通常はデバッグまたはモニターが可能になる。

プロキシー (proxy). 特定のネットワーク・アプリケーション用 (Telnet や FTP など) に、あるネットワークから別のネットワークへ転送するアプリケーション・ゲートウェイ。例えば、ファイアウォール・プロキシー Telnet サーバーがユーザーの認証を実行すると、トラフィックは、プロキシーが存在しないかのようにそのプロキシーを流れる。機能はクライアント・ワークステーションではなくファイアウォールで実行されるため、ファイアウォールの負荷が増す。

プロキシー・クラスター (proxy cluster). HTTP 要求をクラスター全体にわたって配布するプロキシー・サーバーのグループ。

プロキシー・サーバー (proxy server).

1. アプリケーションまたは Web サーバーによってホストされる、HTTP Web 要求の仲介として動作するサーバー。プロキシー・サーバーは、エンタープライズ内のコンテンツ・サーバーの代理の役割を果たす。

2. 別のサーバーを対象とした要求を受信し、要求されたサービスを獲得するために、クライアントに代わって (クライアントのプロキシーとして) 働くサーバー。プロキシー・サーバーは、クライアントとサーバーが、直接接続するには非互換であるという場合によく使用される。例えば、クライアントはサーバーのセキュリティー認証要件に合わせることができないが、一部のサービスの許可が必要な場合がこれに該当する。

プロキシー・ピア・アクセス・ポイント (proxy peer access point). 直接にはアクセスできないピア・アクセス・ポイントの通信設定を識別する手段。

プログラム一時修正 (program temporary fix (PTF)). System i、System p、および System z 製品の場合、すべてのお客様が入手できるようにしてある IBM テスト済みの修正。「フィックスパック (fix pack)」も参照。

プログラム診断依頼書 (authorized program analysis report (APAR)). サポート対象リリースの IBM 提供プログラムにおける問題点に対する修正要求。

プロセス (process).

1. 特定の結果または結末に体系的に導かれる一連の制御されたアクティビティーで構成された、連続的に続く手順。

2. ビジネス・トランザクションを実行するためにコミュニティー・マネージャーと参加プログラムの間で交換される文書またはメッセージのシーケンス。

プロトコル・バインディング (protocol binding). エンタープライズ・サービス・バスが通信プロトコルとは無関係にメッセージを処理できるようにするバインディング。

プロパティー (property). オブジェクトを記述するオブジェクトの特性。プロパティーは変更できる。プロパティーは、オブジェクト名前、タイプ、値、振る舞いなどの事項を記述できる。

プロファイル (profile). ユーザー、グループ、リソース、プログラム、デバイス、またはリモート・ロケーションの特性を記述しているデータ。

プロンプト (prompt). フィールドにユーザー入力し、出力画面へ遷移することを確認できるコンポーネント。

分散 eXtreme Scale (distributed eXtreme Scale). サーバーおよびクライアントが複数のプロセスに存在する場合に、eXtreme Scale と対話するための使用パターン。

文書タイプ定義 (DTD) (document type definition (DTD)). SGML または XML 文書の個々のクラスの構造を指定する規則。DTD は、エレメント、属性、および表記法を使用して構造を定義する。また、各エレメント、属性、および表記法を、文書の個々のクラス内で使用する方法に関する制約も規定する。

ペシミスティック・ロック (pessimistic locking). 行が選択されてから、その行に対して検索更新操作または検索削除操作が試みられるまでの間、ロックが保たれるようなロック戦略。

ヘルス (health). データベース環境の全般的条件または状態。

変数 (variable). 可変値を表す。

ポート (port). Web サービス記述言語 (WSDL) の資料に定義されているように、バインディングとネットワーク・アドレスの組み合わせとして定義される単一エンドポイント。

ポート番号 (port number). インターネット通信において、アプリケーション・エンティティーとトランスポート・サービスの間の論理結合子の ID。

保守モード (maintenance mode). 管理者が実稼働環境の着信トラフィックを中断することなく、ノードまたはサーバーの診断、保守、またはチューニングに使用できる、ノードまたはサーバーの状態。

ホスト (host).

1. ネットワークに接続され、そのネットワークへのアクセス・ポイントを提供するコンピューター。ホストはクライアント、サーバー、または同時にその両方である場合がある。

2. パフォーマンス・プロファイル作成では、プロファイル作成されているプロセスを所有しているマシン。サーバー (server) も参照。

ホスト・システム (host system). 3270 アプリケーションをホストするエンタープライズ・メインフレーム・コンピューター・システム。3270 端末サービス開発ツールでは、開発者は 3270 端末サービス・レコーダーを使用してホスト・システムに接続する。

ホスト名 (host name).

1. インターネット通信では、コンピューターに付けられた名前。ホスト名は、完全修飾ドメイン名 (例: mycomputer.city.company.com) の場合も、あるいは、固有のサブネーム (例: mycomputer) の場合もあります。

2. ノードがインストールされている物理マシン上のネットワーク・アダプターのネットワーク名。

ボトムアップ開発 (bottom-up development). Web サービスにおいて、Web サービス記述言語 (WSDL) ファイルからではなく、Java Bean またはエンタープライズ Bean などの既存の成果物からサービスを開発するプロセス。

ボトルネック. リソースの競合がパフォーマンスに影響を与えるシステムの場所。

ポリシー. 管理対象リソースまたはユーザーの振る舞いに影響を与える一連の考慮事項。

マップ (map).

1. キーを値にマップするデータ構造。

2. ソースとターゲットの間の変換を定義するファイル。

3. EJB 開発環境で、エンタープライズ Bean のコンテナー管理の永続フィールドが、リレーショナル・データベースの表または他の永続ストレージにある列に対応する方法の指定。

メソッド (method). オブジェクト指向プログラミングにおいて、オブジェクトが実行できるオペレーション。オブジェクトには多数のメソッドがある。

メトリック (metric). モニター関連の用語で、情報 (通常は業績測定) のホルダー。

メモリー・リーク (memory leak). 既に不要なために新たに再生すべきオブジェクト参照をプログラムが保持し続けることによる悪影響のある現象。

呼び出し (invocation). プログラムまたはプロシージャーを活動化すること。

ライトスルー・キャッシュ (write-through cache). ローダーを使用して、データベースに対する各書き込み操作が同期的に行われるキャッシュ。

ライフサイクル. ソフトウェア開発における方向付け、推敲、作成、および移行の 4 つのフェーズを一巡すること。

ライブラリー (library).

1. ビジネス・アイテム、プロセス、タスク、リソース、組織などのモデル・エレメントの集合。

2. 開発、バージョン管理、および共用リソースの編成のために使用されるプロジェクト。ビジネス・オブジェクトやインターフェースなど、成果物タイプのサブセットのみをライブラリーに作成および保管することができる。

ランタイム (run time). コンピューター・プログラムが実行している間の時間枠。

ランタイム・トポロジー (runtime topology). 環境の現行の状態を表したもの。

リードスルー・キャッシュ (read-through cache). 要求されたデータ・エントリーをキーによってロードするスパース・キャッシュ。データがキャッシュに見つからない場合、その欠落データがローダーによって検索され、このローダーがそのデータをバックエンド・データ・リポジトリーからロードしてキャッシュに挿入する。

リスナー (listener). 接続要求を受け付け、関連チャネルを開始するプログラム。

リスナー・ポート (listener port). 接続ファクトリー、宛先、およびデプロイされたメッセージ駆動型 Bean 間の関連を定義するオブジェクト。リスナー・ポートは、これらのリソース間の関連の管理を単純化する。

リソース (resource).

1. 離散的アセット。例えば、アプリケーション・スイート、アプリケーション、ビジネス・サービス、インターフェース、エンドポイント、ビジネス・イベントなど。

2. ジョブ、タスク、または実行中のプログラムが必要とするコンピューター・システムまたはオペレーティング・システムの機能。リソースには、メイン・ストレージ、入出力装置、処理装置、データ・セット、ファイル、ライブラリー、フォルダー、アプリケーション・サーバー、制御プログラム、処理プログラムなどがある。

3. タスクまたはプロジェクトを実行するための個人、装置、または資料。各リソースは、リソース定義の特定の存在または例である。

リフレッシュ・パック (refresh pack). 修正の累積コレクションで、新規機能を含む。 フィックスパック (fix pack)、暫定修正 (interim fix) も参照。

領域 (region). 共通特性を備え、プロセス間で共用可能な仮想ストレージの連続区域。

ループ (loop). 反復して実行される命令のシーケンス。

例外 (exception). 通常の処理では扱うことのできない条件またはイベント。

例外ハンドラー (exception handler). 異常条件に対応するルーチンのセット。例外ハンドラーは割り込みを行い、通常の処理の実行を再開できる。

レプリカ. ディレクトリーのコピー、または別のサーバーのディレクトリーのコピーが含まれるサーバー。レプリカにより、パフォーマンスや応答時間の改善のため、またはデータ保全性の維持のために、サーバーがバックアップされます。

ローカル.

1. ユーザー・システムから、通信回線を使用せずに直接アクセスする装置、ファイル、またはシステムを示す用語。

2. 特定のプロセス内でのみ使用可能なエレメントに関する用語。

ローカル・データベース (local database). 使用しているワークステーションに配置されているデータベース。

ローダー. 永続ストアでデータの読み書きを行うコンポーネント。

ロード・バランシング (load balancing). アプリケーション・サーバーの監視と、サーバー上のワークロード管理。あるサーバーのワークロードが超過すると、要求は、容量のより大きい別のサーバーへ転送される。

ロール (role).

1. 個人またはバルク・リソースによって実行される機能の記述、および機能を遂行するために必要な資格。シミュレーションおよび分析において、ロールという用語は、資格のあるリソースを指すためにも使用される。

2. ユーザーが実行できるタスク、およびユーザーがアクセスできるリソースを識別するジョブの機能。1 人のユーザーに 1 つ以上のロールを割り当てることができる。

3. 一連の許可を提供するプリンシパルの論理グループ。オペレーションへのアクセスは、ロールへのアクセスを認可することにより制御される。

4. リレーションにおいて、ロールは、エンティティーの機能および関与を決定する。ロールは、関与するエンティティーと関与の方法に対する構造および制約の要件を取り込む。例えば、雇用関係におけるロールは、雇用者と被雇用者である。

ロギング (logging). エラーなど、システム上の特定のイベントについてのデータを記録すること。

ロック (lock). 1 つのアプリケーション・プロセスによって行われたコミットされていない変更が別のアプリケーション・プロセスに感知されないようにし、1 つのアプリケーション・プロセスが別のアプリケーション・プロセスでアクセス中のデータを更新できないようにする手段。ロックにより、各ユーザーが同時に不整合データにアクセスできなくなるので、データの保全性が保たれる。

ロング・ネーム (long name). z/OS プラットフォーム上のサーバーに論理名を指定するプロパティー。

ワークスペース (workspace).

1. すべてのプロジェクト・ファイルとプリファレンスなどの情報を含むディスク上のディレクトリー。

2. 管理クライアントが使用する構成情報の一時的なリポジトリー。

3. Eclipse では、現在ユーザーがワークベンチで開発を行っているプロジェクトおよびその他のリソースの集合。これらのリソースに関するメタデータは、ファイル・システム上のディレクトリーにある。リソースが同じディレクトリーにある場合もある。

ワークロード・マネージャー (WLM) (Workload Manager (WLM)). z/OS のコンポーネントの 1 つ。単一の z/OS イメージまたは複数のイメージで同時に複数のワークロードを実行するための機能を提供する。

ワークロード管理 (workload management). アプリケーション・サーバーやエンタープライズ Bean、サーブレットなど、要求を効率的に処理できるオブジェクトに対して、着信した作業要求を最適な方法で分配すること。

1 次キー (primary key).

1. 特定のタイプのエンティティー Bean を一意的に識別するオブジェクト。

2. リレーショナル・データベースで、データベース・テーブルのある 1 つの行を一意的に識別するキー。

APAR. 「プログラム診断依頼書 (authorized program analysis report)」を参照。

API. 「アプリケーション・プログラミング・インターフェース (application programming interface)」を参照。

Bean. JavaBeans コンポーネントの定義およびインスタンス。「JavaBeans」、「エンタープライズ Bean (Enterprise Bean)」も参照。

Bean Scripting Framework. スクリプト言語機能を Java アプリケーションに取り込むアーキテクチャー。

Bean 管理トランザクション (BMT) (bean-managed transaction (BMT)). トランザクションをコンテナー経由ではなく直接管理するための、セッション Bean、サーブレット、またはアプリケーション・クライアント・コンポーネントの機能。

Bean 管理パーシスタンス (BMP) (bean-managed persistence (BMP)). エンティティー Bean の変数とリソース・マネージャーの間で行われるデータ転送がエンティティー Bean によって管理されるときに使用されるメカニズム。

Bean 管理メッセージング (bean-managed messaging). メッセージング・インフラストラクチャー全体の完全な制御をエンタープライズ Bean に与える非同期メッセージングの機能。

Bean クラス (bean class). Enterprise JavaBeans (EJB) プログラミングにおいて、javax.ejb.EntityBean クラスまたは javax.ejb.SessionBean クラスを実装する Java クラス。

BMP. 「Bean 管理パーシスタンス (bean-managed persistence)」を参照。

BMT. 「Bean 管理トランザクション (bean-managed transaction)」を参照。

CIP. 「カスタマイズ・インストール・パッケージ (customized installation package)」を参照。

cloudscape. 組み込み可能で、すべてが Java で書かれたオブジェクト・リレーショナル・データベース管理システム (ORDBMS)。

create メソッド (create method). エンタープライズ Bean では、エンタープライズ Bean を作成するために、ホーム・インターフェース内に定義され、クライアントによって呼び出されるメソッド。

DB2. リレーショナル・データベース管理用 IBM ライセンス・プログラム・ファミリー。

DNS. 「ドメイン・ネーム・システム (Domain Name System)」を参照。

do while ループ (do-while loop). ある条件が満足される限りアクティビティーの同じシーケンスを反復するループ。do while ループは while ループと異なり、ループの終わりで条件をテストする。つまり、アクティビティーのシーケンスは最低 1 回は必ず実行されることを意味する。

DTD. 「文書タイプ定義 (document type definition)」を参照。

DTD 文書定義 (DTD document definition). XML DTD に基づく XML 文書の記述またはレイアウト。

EAR. 「エンタープライズ・アーカイブ (enterprise archive)」を参照。

EAR プロジェクト (EAR project). 「エンタープライズ・アプリケーション・プロジェクト (enterprise application project)」を参照。

Eclipse. ISV やその他のツール・デベロッパーに対して、互換性のある開発ツールを作成するための標準プラットフォームを提供する、オープン・ソース・イニシアチブ。

EJB. 「Enterprise JavaBeans」を参照。

EJB JAR ファイル (EJB JAR file). EJB モジュールを含む Java アーカイブ。

EJB オブジェクト (EJB object). エンタープライズ Bean において、エンタープライズ Bean リモート・インターフェースを実装するクラスを持つオブジェクト。

EJB 継承 (EJB inheritance). エンタープライズ Bean が、同じグループ内の他のエンタープライズ Bean からプロパティー、メソッド、およびメソッド・レベルの制御記述子属性を継承する際の形式。

EJB コンテキスト (EJB context). エンタープライズ Bean において、コンテナーによって提供されるサービスを呼び出すこと、およびクライアントに呼び出されたメソッドの呼び出し側についての情報を取得することをエンタープライズ Bean に許可するオブジェクト。

EJB コンテナー (EJB container). Java EE アーキテクチャーの EJB コンポーネント規約を実装するコンテナー。この規約は、エンタープライズ Bean に対してランタイム環境を規定する。これには、セキュリティー、並行性、ライフサイクル管理、トランザクション、デプロイメント、およびその他のサービスが含まれる。

EJB サーバー (EJB server). EJB コンテナーにサービスを提供するソフトウェア。 EJB サーバーは、1 つ以上の EJB コンテナーをホスティングできる。

EJB 参照 (EJB reference). ターゲットの動作環境で、エンタープライズ Bean のホーム・インターフェースの位置を指定するためにアプリケーションが使用する論理名。

EJB 照会 (EJB query). EJB 照会言語において、戻される EJB オブジェクトを特定するオプションの SELECT 節、Bean コレクションを指定する FROM 節、コレクションでの検索述部を含むオプションの WHERE 節、結果のコレクションの順序を指定するオプションの ORDER BY 節、および finder メソッドの引数に対応する入力パラメーターを含むストリング。

EJB ファクトリー (EJB factory). エンタープライズ Bean インスタンスの作成と検索を単純化するアクセス Bean。

EJB プロジェクト (EJB project). エンタープライズ Bean、ホーム・インターフェース、ローカル・インターフェース、リモート・インターフェース、JSP ファイル、サーブレット、およびデプロイメント記述子など、EJB アプリケーションに必要なリソースを含むプロジェクト。

EJB ホーム・オブジェクト (EJB home object). Enterprise JavaBeans (EJB) プログラミングにおいて、エンタープライズ Bean に対してライフサイクル操作 (create、remove、find) を実行するオブジェクトのこと。

EJB モジュール. 1 つ以上のエンタープライズ Bean および EJB デプロイメント記述子からなるソフトウェア単位。

Enterprise JavaBeans (EJB). オブジェクト指向の分散型エンタープライズ・レベル・アプリケーション (Java EE) の開発とデプロイメントのため、Sun Microsystems によって定義されたコンポーネント・アーキテクチャー。

ESB. 「エンタープライズ・サービス・バス (enterprise service bus)」を参照。

Evictor. 各 BackingMap インスタンス内にあるエントリーのメンバーシップを制御するコンポーネント。スパース・キャッシュでは、エビクターを使用して、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る。

expression. 1 つの SQL または XQuery オペランド、あるいは SQL または XQuery 演算子とオペランドからなる 1 つのコレクション。単一の値を生成する。

Extensible Markup Language (XML). Standard Generalized Markup Language (SGML) に基づくマークアップ言語を定義する標準メタ言語。

eXtreme Scale グリッド (eXtreme Scale grid). データおよびクライアントがすべて 1 つのプロセスにある場合に、eXtreme Scale と対話するために使用されるパターン。

for ループ (for loop). 同一の複数アクティビティーの順序処理を、指定した回数だけ繰り返すループのこと。

General Inter-ORB Protocol (GIOP). Common Object Request Broker Architecture (CORBA) がメッセージのフォーマットを定義するために使用するプロトコル。

getter メソッド (getter method). インスタンスの値、またはクラス変数を取得することを目的としたメソッド。これにより、別のオブジェクトがその変数のうちの 1 つの値を取得できる。

GIOP. 「General Inter-ORB Protocol」を参照。

HA. 「高可用性 (high availability)」を参照。

HA グループ. プロセスの高可用性を実現するために使用されるメンバーの集まり。

HA ポリシー. HA グループのために定義するルールのセットで、0 またはそれ以上のメンバーをアクティブにするかどうかを決定する。このポリシーは、ポリシー一致基準をグループ名と突き合わせることによって、特定の HA グループと関連付けられる。

HA マネージャー (high availability manager). コア・グループ・メンバーシップが判別され、状況がコア・グループ・メンバー間で伝達されるフレームワーク。

HTTP over SSL (HTTPS). トランザクションを保護するための Web プロトコル。ユーザー・ページ要求および Web サーバーで戻されるページを暗号化、および復号化を行う。

HTTPS.

1. 「SSL を使用する HTTP (HTTP over SSL)」を参照。
2. 「Hypertext Transfer Protocol Secure」を参照。

Hypertext Transfer Protocol Secure (HTTPS). ハイパーメディア文書をインターネット経由で安全に転送および表示するために、Web サーバーと Web ブラウザーで使用されるインターネット・プロトコル。

IDE. 「統合開発環境 (integrated development environment)」を参照。

if-then ルール (if-then rule). 条件 (if 部分) が真のときにのみ、アクション (then 部分) が実行されるルール。

IIOP. 「Internet Inter-ORB Protocol」を参照。

Internet Inter-ORB Protocol (IIOP). Common Object Request Broker Architecture (CORBA) オブジェクト・リクエスト・ブローカー間の通信に使用されるプロトコル。

IP. 「インターネット・プロトコル (Internet Protocol)」を参照。

IP スプレイヤー (IP sprayer). 複数ユーザーからのインバウンド要求と複数アプリケーション・サーバー・ノードの間に位置し、リクエストを複数ノードに転送する装置。

JAAS . 「Java 認証・承認サービス (Java Authentication and Authorization Service)」を参照。

JAF. 「JavaBeans Activation Framework」を参照。

JAR ファイル (JAR file). Java アーカイブ・ファイル。「Web アーカイブ (Web archive)」、「エンタープライズ・アーカイブ (enterprise archive)」も参照。

Java. リモート・オブジェクト内の相互作用をサポートする、移植可能な解釈コード用のオブジェクト指向プログラム言語。Java は、Sun Microsystems によって開発および指定されたものである。

Java API for XML (JAX). Extensible Markup Language (XML) を介して定義されたデータに関連するさまざまな操作を処理するための Java ベースの API のセット。

Java Authentication and Authorization Service (JAAS). Java EE テクノロジーにおいて、セキュリティー・ベースのオペレーションを実行するための標準 API。サービスは、JAAS を介して、ユーザーを認証および承認し、基礎となるテクノロジーからアプリケーションを独立させておくことを可能にする。

Java Command Language. Java 環境用のスクリプト言語で、Web コンテンツの作成および Java アプリケーションの制御に使用される。

Java Database Connectivity (JDBC). Java プラットフォームと広範なデータベースとの間のデータベース独立の接続用の業界標準。JDBC インターフェースは、SQL ベースおよび XQuery ベースのデータベース・アクセス用にコール・レベル・インターフェースを提供する。

Java EE. 「Java Platform, Enterprise Edition」を参照。

Java EE アプリケーション (Java EE application). Java EE 機能のデプロイ可能な任意の単位。この単位には、Java EE アプリケーションのデプロイメント記述子と一緒にエンタープライズ・アーカイブ (EAR) ファイルにパッケージされた、単一モジュールまたはモジュール・グループがある。

Java EE コネクター・アーキテクチャー (JCA). Java EE プラットフォームを異機種混合のエンタープライズ情報システム (EIS) に接続するための標準アーキテクチャー。

Java EE サーバー (Java EE server). EJB コンテナーまたは Web コンテナーを提供するランタイム環境。

Java Management Extensions (JMX). Java テクノロジーを介して Java テクノロジーの管理を行う手段のこと。JMX は、管理用の Java プログラミング言語のユニバーサルかつオープンな拡張機能であり、管理が必要とされるすべての業界でデプロイできる。

Java Message Service (JMS). メッセージ処理用の Java 言語機能を提供する、アプリケーション・プログラミング・インターフェース。

Java Naming and Directory Interface (JNDI). Java プラットフォームの拡張により、異機種の命名サービスとディレクトリー・サービス用の標準インターフェースが提供される。

Java Platform, Enterprise Edition (Java EE). エンタープライズ・アプリケーションを開発およびデプロイするための環境であり、Sun Microsystems によって定義されている。Java EE プラットフォームは、多層化された Web ベース・アプリケーションを開発するための機能を提供する、一連のサービス、アプリケーション・プログラミング・インターフェース (API)、およびプロトコルで構成される。

Java Platform, Standard Edition (Java SE). Java テクノロジー・プラットフォームの中核。

Java SE. 「Java Platform, Standard Edition」を参照。

Java SE Development Kit (JDK). Sun Microsystems が提供する Java プラットフォーム用のソフトウェア開発キットの名前。

Java Secure Socket Extension (JSSE). セキュア・インターネット通信を可能にする Java パッケージ。この Java パッケージは、Java バージョンの Secure Sockets Layer (SSL) および Transport Layer Security (TLS) プロトコルを実装して、データ暗号化、サーバー認証、メッセージ保全性、およびオプションで、クライアント認証をサポートする。

Java Specification Request (JSR). Java プラットフォームに対して、公式に提案された仕様。

Java virtual machine Profiler Interface (JVMPI). ガーベッジ・コレクションに関するデータなどの情報の収集や、アプリケーション・サーバーを実行する Java 仮想マシン (JVM) API をサポートするプロファイル作成ツール。

Java アーカイブ (Java archive). Java プログラムをインストールおよび実行するために必要なすべてのリソースを単一ファイルで保管するための、圧縮されたファイル・フォーマット。「Web アーカイブ (Web archive)」、「エンタープライズ・アーカイブ (enterprise archive)」も参照。

Java 仮想マシン (Java virtual machine (JVM)). コンパイルされた Java コード (アプレットおよびアプリケーション) を実行するプロセッサーのソフトウェア実装。

Java クラス (Java class). Java 言語で記述されるクラス。

Java コネクター・セキュリティー (Java Connector security). Java EE ベースのアプリケーションのエンドツーエンド・セキュリティー・モデルを拡張して、エンタープライズ情報システム (EIS) を組み込むよう設計されたアーキテクチャー。

Java ファイル (Java file). 編集可能なソース・ファイル (拡張子 .java)。バイトコード (.class ファイル) にコンパイルが可能。

Java プラットフォーム (Java platform). プログラム作成のための Java 言語、また、プログラムの開発、コンパイルおよびエラー・チェックに使用する API のセット、クラス・ライブラリー、およびその他プログラム、そしてクラス・ファイルをロードし実行する Java 仮想マシンに対する総称。

Java プロジェクト (Java project). Eclipse では、コンパイル可能な Java ソース・コードを含み、ソース・フォルダーまたはパッケージのコンテナーとなるプロジェクト。

Java ランタイム環境 (Java runtime environment). 標準的 Java プラットフォームを構成する中核の実行可能プログラムおよびファイルを含む Java Developer Kit のサブセット。JRE には Java 仮想マシン (JVM)、コア・クラス、およびそれらをサポートするファイルが含まれる。

JavaBeans. Sun Microsystems によって Java 用に定義された、ポータブルでプラットフォーム非依存の、再使用可能なコンポーネント・モデル。「Bean」も参照。

JavaBeans Activation Framework (JAF). 任意のデータ・タイプおよび使用可能な操作を判別し、関連するサービスを実行するように Bean をインスタンス化できる、Java プラットフォームに対する標準拡張。

Javadoc.

1. 1 組のソース・ファイルの中の宣言およびドキュメンテーション・コメントを解析して、クラス、内部クラス、インターフェース、コンストラクター、メソッド、およびフィールドを記述する 1 組の HTML ページを作成するツール。

2. 1 組のソース・ファイルの中の宣言およびドキュメンテーション・コメントを解析して、クラス、内部クラス、インターフェース、コンストラクター、メソッド、およびフィールドを記述する 1 組の HTML ページを作成するツールに関する用語。

JavaMail API. Java ベースのメール・クライアント・アプリケーションを構築するための、プラットフォームとプロトコルに依存しないフレームワーク。

JavaScript. ブラウザーと Web サーバーの両方で使用される、Web スクリプト言語の 1 つ。

JavaScript Object Notation. JavaScript のオブジェクト・リテラル記法に基づく単純なデータ交換形式。JSON はプログラミング言語に対して中立的だが、C、C++、C#、Java、JavaScript、Perl、Python などの言語の規則を使用する。

JavaServer Pages (JSP). サーバー・サイド・スクリプト・テクノロジーの 1 つで、これにより、Java コードが Web ページ (HTML ファイル) 内に動的に組み込まれ、そのページにサービスが提供されると、実行されて動的コンテンツをクライアントに戻すことが可能になる。

JAX. 「Java API for XML」を参照。

JCA. 「Java EE コネクター・アーキテクチャー (Java EE Connector Architecture)」を参照。

JDBC. 「Java Database Connectivity」を参照。

JDK. 「Java SE Development Kit」を参照。

JMS. 「Java Message Service」を参照。

JMS データ・バインディング (JMS data binding). 外部 JMS メッセージによって使用されるフォーマットと、サービス・コンポーネント・アーキテクチャー (SCA) モジュールによって使用されるサービス・データ・オブジェクト (SDO) の間のマッピングを提供するデータ・バインディング。

JMX. 「Java Management Extensions」を参照。

JNDI. 「Java Naming and Directory Interface」を参照。

JSP. 「JavaServer Pages」を参照。

JSP ファイル (JSP file). .jsp 拡張子を持ち、Web ページへの動的コンテンツの組み込みを可能にする、スクリプト化された HTML ファイル。JSP ファイルは、URL として直接要求するか、サーブレットで呼び出すか、HTML ページ内から呼び出すことができる。

JSP ページ (JSP page). 応答を作成する要求の処理方法を説明する、固定テンプレート・データおよび JSP エレメントを使用したテキスト・ベースの文書。

JSR. 「Java Specification Request」を参照。

JSSE. 「Java Secure Socket Extension」を参照。

JVM. 「Java 仮想マシン (Java virtual machine)」を参照。

JVMPI. 「Java Virtual Machine Profiler Interface」を参照。

Jython. Python プログラミング言語を Java プラットフォームに組み込んで実装したもの。

LDAP. 「Lightweight Directory Access Protocol」を参照。

LDAP ディレクトリー (LDAP directory). 人、組織、およびその他のリソースに関する情報を保管するリポジトリーの一種。LDAP プロトコルを使用してアクセスされる。リポジトリー内の項目は、階層構造に編成される。場合によっては、階層構造は組織の構造または組織の地理的分布を表す。

Lightweight Directory Access Protocol (LDAP). TCP/IP を使用して、X.500 モデルをサポートするディレクトリーを知る方法を提供し、より複雑な X.500 Directory Access Protocol (DAP) のリソース要件を発生させない公開プロトコル。例えば、LDAP を使用して、インターネットまたはイントラネット・ディレクトリー内の個人、組織、およびその他のリソースを見つけることができる。

Lightweight Third Party Authentication (LTPA). 分散環境において、暗号方式を使用してセキュリティーをサポートするプロトコル。

LTPA. 「Lightweight Third Party Authentication」を参照。

MBean. 「Managed Bean」を参照。

MBean プロバイダー (MBean provider). Java Management Extensions (JMX) MBean の実装および MBean の Extensible Markup Language (XML) 記述子ファイルを含むライブラリー。

node.

1. 複数の管理対象サーバーから成る論理グループ。

2. ツリー制御の項目。単一エレメント、複合エレメント、マッピング・コマンド、コメント、またはグループ・ノードが含まれる。

3. XML では、文書における有効で完全な構造の最小単位。

4. 図を構成する基本の形状。

ObjectGrid. Java で書かれたアプリケーション用のグリッドに対応したメモリー・データベース。ObjectGrid は、メモリー内のデータベースとして使用することも、ネットワーク全体にデータを分散するために使用することもできる。

ODBC. 「Open Database Connectivity」を参照。

Open Database Connectivity (ODBC). リレーショナルおよび非リレーショナルの両方のデータベース管理システムのデータにアクセスするための、標準的なアプリケーション・プログラミング・インターフェース (API)。各データベース管理システムが異なるデータ・ストレージ形式およびプログラミング・インターフェースを採用している場合でも、データベース・アプリケーションは、この API を使用することにより、さまざまなコンピューター上のデータベース管理システムに保管されているデータにアクセスできます。

ORB. 「オブジェクト・リクエスト・ブローカー (Object Request Broker)」を参照。

Performance Monitoring Infrastructure (PMI). パフォーマンス・データを収集、配送、処理、および表示するために割り当てられたパッケージおよびライブラリーの集合。

PMI. 「Performance Monitoring Infrastructure」を参照。

point-to-point. メッセージの宛先が送信側のアプリケーションによって認識されている、メッセージング・アプリケーションのスタイル。

PTF. 「プログラム一時修正 (program temporary fix)」を参照。

QoS. 「サービスの品質 (quality of service)」を参照。

root. 最大の権限を持つシステム・ユーザーのユーザー名。

SDK. 「Software Development Kit」を参照。

Secure Sockets Layer (SSL). 通信のプライバシーを提供するセキュリティー・プロトコルの 1 つ。 SSL を使用すれば、盗聴、改ざん、およびメッセージ偽造を防止するよう設計された方法で、クライアント/サーバー・アプリケーションは通信することができる。

setter メソッド (setter method). インスタンスの値、またはクラス変数を設定することを目的としたメソッド。この能力により、他のオブジェクトがその変数の 1 つの値を設定できるようになる。

SLA. 「サービス・レベル・アグリーメント (service level agreement)」を参照。

Software Development Kit (SDK). 特定のコンピューター言語または特定の稼働環境用のソフトウェア開発を支援するツールのセット、API、およびドキュメンテーションのこと。

SQL. 「構造化照会言語 (Structured Query Language)」を参照。

SQL 照会 (SQL query). 結果テーブルを指定する特定の SQL ステートメントのコンポーネント。

SSL. 「Secure Sockets Layer」を参照。

SSL チャネル (SSL channel). トランスポート・チェーン内のチャネルの一種。Secure Sockets Layer (SSL) 構成レパートリーをトランスポート・チェーンに関連付ける。

TCP. 「伝送制御プロトコル (Transmission Control Protocol)」を参照。

TCP チャネル (TCP channel). トランスポート・チェーン内のチャネルの一種。これにより、クライアント・アプリケーションは、ローカル・エリア・ネットワーク (LAN) 内で永続的な接続を行うことができる。

TCP/IP. 「伝送制御プロトコル/インターネット・プロトコル (Transmission Control Protocol/Internet Protocol)」を参照。

TCP/IP モニター・サーバー (TCP/IP monitoring server). TCP/IP アクティビティーだけでなく、Web ブラウザーとアプリケーション・サーバー間のすべての要求と応答をモニターするランタイム環境。

timeout. あるイベントが発生または完了するのを待機する時間間隔。これを過ぎると操作が中断される。

Tivoli Performance Viewer. アプリケーション・サーバーから Performance Monitoring Infrastructure (PMI) データを取得して、それを各種の形式で表示する Java クライアント。WAS Version 6.0 以降は Web アプリケーションとして管理コンソールに統合。

transaction. トランザクション中に行われたデータ変更がすべて一緒に 1 単位としてコミットまたは 1 単位としてロールバックされるプロセス。

Transmission Control Protocol (TCP). インターネット、および Internet Engineering Task Force (IETF) のインターネットワーク・プロトコル標準に準拠するネットワークで使用される通信プロトコル。TCP は、パケット交換通信ネットワークとそのようなネットワークで相互接続されたシステムで、信頼できるホスト間プロトコルを提供する。

type.

1. Java プログラミングにおけるクラス、またはインターフェース。

2. WSDL 文書では、何らかの型システム (XSD など) を使用するデータ型定義を含むエレメントのこと。

UDDI. 「Universal Description, Discovery, and Integration」を参照。

Uniform Resource Identifier (URI).

1. 抽象的または物理的なリソースを識別するための簡潔な文字ストリング。

2. テキストのページ、ビデオ・クリップやサウンド・クリップ、静止画や動画、またはプログラムなどの Web 上のコンテンツを識別するのに使用する固有のアドレス。URI の最も一般的な形式は Web アドレスである。Web アドレスは URI の特別な形式またはサブセットで URL (Uniform Resource Locator) と呼ばれる。通常 URI が表すのは、リソースへのアクセス方法、リソースを含むコンピューター、およびコンピューター上のリソース名 (ファイル名) を表す。

Uniform Resource Locator (URL). インターネットなどのネットワークでアクセス可能な情報リソースの固有アドレス。URL には、情報リソースへのアクセスに使用されるプロトコルの省略名と情報リソースを位置指定するためにプロトコルが使用する情報が含まれている。

Uniform Resource Name (URN). クライアントに対し Web サービスを一意的に識別する名前。

Universal Description, Discovery, and Integration (UDDI). 会社およびアプリケーションがインターネット上で迅速かつ容易に Web サービスを検索および利用できるようにする、標準ベースの仕様のセット。

Universally Unique Identifier (UUID). 2 つのコンポーネントが同じ ID を持たないようにするために使用される 128 ビットの数値 ID。

UNIX システム・サービス (UNIX System Services). XPG4 UNIX 1995 仕様に準拠した UNIX 環境を構築する z/OS のエレメント。 z/OS オペレーティング・システム上で、アプリケーション・プログラミング・インターフェース (API) および対話式シェル・インターフェースという 2 つのオープン・システム・インターフェースを提供する。

URI. 「Uniform Resource Identifier」を参照。

URL. 「Uniform Resource Locator」を参照。

URL スキーム (URL scheme). 別のオブジェクト参照を含む形式。

URN. 「Uniform Resource Name」を参照。

UUID. 「汎用固有 ID (Universally Unique Identifier)」を参照。

version. 通常では重要な新規コードまたは新規機能を備えている、別個にライセンスされるプログラム。

WAR. 「Web アーカイブ (Web archive)」を参照。

WCCM. 「WebSphere Common Configuration Model」を参照。

Web アーカイブ (Web archive (WAR)). 単一ファイルで Web アプリケーションをインストールおよび実行するために必要なすべてのリソースを保管するための、Java EE 標準で定義された圧縮ファイル・フォーマット。「エンタープライズ・アーカイブ (enterprise archive)」も参照。

Web クローラー (Web crawler). Web 文書を検索して、その文書内のリンクをたどることにより Web を探索するタイプのクローラー。

Web コンテナー (Web container). Java EE アーキテクチャーの Web コンポーネント規約を実装するコンテナー。

Web コンテナー・チャネル (Web container channel). トランスポート・チェーン内のチャネルの一種。HTTP インバウンド・チャネルとサーブレットまたは JavaServer Pages (JSP) エンジン間のトランスポート・チェーン内にブリッジを作成する。

Web コンポーネント (Web component). サーブレット、JavaServer Pages (JSP) ファイル、またはハイパーテキスト・マークアップ言語 (HTML) ファイル。 Web モジュールは、1 つ以上の Web コンポーネントによって構成される。

Web サーバー (Web server). Hypertext Transfer Protocol (HTTP) 要求のサービスを提供できるソフトウェア・プログラム。

Web サーバー・プラグイン (Web server plug-in). サーブレットのような動的コンテンツの要求においてアプリケーション・サーバーと通信するために、Web サーバーをサポートするソフトウェア・モジュール。

Web サーバーの分離 (Web server separation). Web サーバーがアプリケーション・サーバーから物理的に分離しているトポロジー。

Web サイト (Web site). 単一エンティティー (1 組織または 1 個人) によって管理され、そのユーザーのためにハイパーテキストの形での情報が含まれている、Web 上で使用可能な関連するファイルの集合。Web サイトには、他の Web サイトへのハイパーテキスト・リンクが含まれていることがある。

Web ブラウザー (Web browser). Web サーバーへの要求を開始し、サーバーが戻す情報を表示するクライアント・プログラム。

WebSphere. e-business アプリケーションおよび Web アプリケーションを実行するミドルウェアの開発用のツールを包含する IBM 製品ブランド名。

WebSphere Common Configuration Model (WCCM). 構成データにプログラムによってアクセスするためのモデル。

what you see is what you get (WYSIWYG). 印刷、またはレンダリング時とまったく同様のページを表示するエディターの機能。

while ループ (while loop). ある条件が満足される限りアクティビティーの同じシーケンスを反復するループ。while ループでは、ループを開始するたびにその条件がテストされる。開始から条件が偽である場合、そのループに含まれる一連のアクティビティーは実行されない。

WLM. 「ワークロード・マネージャー (Workload Manager)」を参照。

WYSIWYG. 「what you see is what you get」を参照。

X/Open XA. X/Open 分散トランザクション処理 XA インターフェース。分散トランザクション通信用に提案された標準である。この標準では、トランザクション内の共用リソースを知る方法を提供するリソース・マネージャー間の双方向インターフェース、およびトランザクションをモニターして解決するトランザクション・サービス間の双方向インターフェースが規定されている。

XA. 共用リソースへのアクセスを可能にする 1 つ以上のリソース・マネージャーとトランザクションをモニターして解決するトランザクション・マネージャーとの間にある双方向インターフェース。

XML. 「Extensible Markup Language」を参照。

z/OS. 64 ビットの実ストレージを使用する IBM のメインフレーム用オペレーティング・システム。

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。

Java およびすべての Java 関連の商標およびロゴは、Sun Microsystems, Inc. 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]

[ナ行]

[ハ行]

[マ行]

[ラ行]

C

E

H

I

J

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