Red Hat JBoss Data Grid 7.0

スタートガイド

Red Hat JBoss Data Grid 7.0 向け

Last Updated: 2017-11-21

Red Hat JBoss Data Grid 7.0 スタートガイド

Red Hat JBoss Data Grid 7.0 向け

Misha Husnain Ali
Red Hat Engineering Content Services
mhusnain@redhat.com

Gemma Sheldon
Red Hat Engineering Content Services
gsheldon@redhat.com

Rakesh Ghatvisave
Red Hat Engineering Content Services
rghatvis@redhat.com

Christian Huffman
Red Hat Engineering Content Services
chuffman@redhat.com

法律上の通知

概要

本ガイドでは Red Hat JBoss Data Grid 7.0 における初歩的な概念や操作について説明します。

目次

パート I. JBOSS DATA GRID の紹介

第1章 RED HAT JBOSS DATA GRID

Red Hat JBoss Data Grid は分散型インメモリーデータグリッドで、以下の機能を提供します。

- スキーマレスなキーバリューストア – JBoss Data Grid は、固定のデータモデルを用いずに異なるオブジェクトを格納する柔軟な NoSQL データベースです。
- グリッドベースのデータストレージ – JBoss Data Grid は、複数のノードにまたがったデータのレプリケートが簡単に行えるよう設計されています。
- エラスティックスケーリング – 処理を中断せずに簡単にノードを追加および削除できます。
- 複数のアクセスプロトコル – REST、Memcached、Hot Rod、またはシンプルなマップのような API を使用して簡単にデータグリッドへアクセスできます。

バグを報告する

1.1. サポートされる構成

Red Hat JBoss Data Grid (現行および過去のバージョン) のサポートされる機能、設定、および統合については、 https://access.redhat.com/articles/115883 のサポートされる構成を参照してください。

バグを報告する

1.2. コンポーネントおよびバージョン

Red Hat JBoss Data Grid には、ライブラリーおよびリモートクライアントサーバーモード用の多くのコンポーネントが含まれています。各使用モードの各バージョンに含まれるコンポーネントの包括的な最新リストは、https://access.redhat.com/articles/488833 の 『Red Hat JBoss Data Grid Component の詳細』ページを参照してください。

バグを報告する

1.3. RED HAT JBOSS DATA GRID の使用モード

Red Hat JBoss Data Grid には 2 つの使用モードがあります。

- リモートクライアントサーバーモード
- ライブラリーモード

バグを報告する

1.3.1. リモートクライアントサーバーモード

リモートクライアントサーバーモードは、管理対象で分散型のクラスター化可能なデータグリッドサーバーを提供します。クライアントサーバーモードでは、サーバーは、JBoss EAP にをベースとするコンテナーを使用し、自己完結型のプロセスとして実行されます。またクライアントアプリケーションは **Hot Rod**、**Memcached**、または **REST** クライアント API を使用して、データグリッドサーバーにリモートアクセスできます。

リモートクライアントサーバーモードでの Red Hat JBoss Data Grid 操作はすべて非トランザクションです。そのため、JBoss Data Grid をリモートクライアントサーバーモードで実行すると、数多くの機能は実行できません。

ライブラリーモードの機能が必要でない場合に JBoss Data Grid をリモートクライアントサーバーモードで実行する利点は数多くあります。リモートクライアントサーバーモードは、選択したプロトコルのクライアントライブラリーがあればクライアント言語を指定しません。そのため、リモートクライアントサーバーモードは、以下を実現します。

- データグリッドのスケーリングがより簡単です。
- クライアントアプリケーションの影響を与えずにデータグリッドをより簡単にアップグレードできます。

以下のコマンドを実行して、スタンドアロンの JBoss Data Grid インスタンスをリモートクライアントサーバーモードで起動します。

Linux の場合:

```
$JBOSS_HOME/bin/standalone.sh
```

Windows の場合:

```
$JBOSS_HOME\bin\standalone.bat
```

以下のコマンドを実行して、管理対象のドメイン JBoss Data Grid インスタンスをリモートクライアントサーバーモードで起動します。

Linux の場合:

```
$JBOSS_HOME/bin/domain.sh
```

Windows の場合:

```
$JBOSS_HOME\bin\domain.bat
```

バグを報告する

1.3.2. ライブラリーモード

ライブラリーモードでは、カスタムのランタイム環境を構築およびデプロイできます。ライブラリーモードはアプリケーションプロセスで単一のデータグリッドノードをホストし、他の JVM でホストされるノードへリモートアクセスできます。Red Hat JBoss Data Grid 6 のライブラリーモード用にテストされたコンテナーには、Red Hat JBoss Enterprise Web Server 2.x と JBoss Enterprise Application Platform 6.x が含まれます。

JBoss Data Grid のライブラリーモードで使用できる機能の中には、リモートクライアントサーバーモードで使用できないものが数多くあります。

ライブラリーモードが必要になる機能は次のとおりです。

- トランザクション
- リスナーおよび通知

JBoss Data Grid は、Java SE のスタンドアロンアプリケーションとして実行することもできます。スタンドアロンモードは、JBoss Data Grid をコンテナーで実行する代替としてサポートされます。

バグを報告する

1.4. RED HAT JBOSS DATA GRID の利点

Red Hat JBoss Data Grid には以下の利点があります。

JBoss Data Grid の利点

パフォーマンス

リモートデータストア (データベースなど) からオブジェクトにアクセスするよりも、ローカルメモリーからオブジェクトにアクセスした方が高速になります。JBoss Data Grid は速度が遅いデータソースから送信されるインメモリーオブジェクトを効率的に保存する方法を提供するため、リモートデータストアよりもパフォーマンスが高速になります。また、JBoss Data Grid はクラスター化されたキャッシュとクラスター化されていないキャッシュの両方に対して最適化を実現するため、パフォーマンスをさらに向上します。

一貫性

キャッシュにデータを格納すると、固有のリスクをともないます。アクセス時にデータが古くなっている (陳腐) 可能性があります。このリスクに対応するため、JBoss Data Grid はキャッシュインバリデーションおよびエクスパレーションを使用して、キャッシュから陳腐データエントリーを削除します。さらに、JBoss Data Grid はトラザクションリカバリーやバージョン API とともに、JTA、分散 (XA)、および 2 フェーズコミットトランザクションをサポートし、保存されたバージョンを基にデータを削除または置換します。

巨大なヒープと高可用性

JBoss Data Grid では、パフォーマンス向上のためにアプリケーションがデータのルックアップ処理のほとんどを大型のサーバーデータベースへ委譲する必要がありません。JBoss Data Grid は、レプリケーションや分散などの技術を導入し、現在のエンタープライズアプリケーションのほとんどに存在するボトルネックを完全に取り除きます。

> **例1.1 巨大なヒープと高可用性の例**
>
> 16 個のブレードサーバーを持つグリッドの例として、各ノードがレプリケートされたキャッシュ専用の 2 GB のストレージ領域を持っているとします。この場合、グリッドの全データが 2 GB のデータのコピーとなります。逆に、分散グリッド (データ項目ごとに 1 つのコピーが必要であることを仮定し、ヒープ全体の容量を 2 で割る) では、メモリーがサポートする仮想ヒープに 16 GB のデータが含まれます。グリッドのどこからでも効果的にこのデータへアクセスできるようになります。サーバーに障害が発生した場合、グリッドによって損失データの新しいコピーが即座に作成され、グリッドの運用サーバーに置かれます。

スケーラビリティー

分散データグリッドがレプリケートされたクラスター化キャッシュよりも優れているのは、容量とパフォーマンスの両方でデータグリッドがスケーラブルである点です。グリッド全体のスループットと容量を増やすには、ノードを JBoss Data Grid に追加します。JBoss Data Grid は、ノードの追加または削除による影響を、グリッドのすべてのノードではなくノードのサブセットに限定する、一貫したハッシュ化アルゴリズムを使用します。

JBoss Data Grid ではデータが均一に分散されるため、グリッドサイズの唯一の上限はネットワーク上でのグループ通信になります。ネットワークのグループ通信は最小限で、新規ノードの発見のみに制限されています。すべてのデータアクセスパターンは、ノードによるピアツーピア接続を介した直接通信を許可するため、スケーラビリティーのさらなる向上を容易にします。
JBoss Data Grid のクラスターは、リアルタイムでスケールアップまたはスケールダウンすることができ、インフラストラクチャーを再起動する必要はありません。スケーリングポリシーへの変更をリアルタイムで適用できるため、非常にフレキシブルな環境を実現することができます。

データ分散

JBoss Data Grid は一貫したハッシュアルゴリズムを使用して、クラスターでのキーの場所を判断します。一貫したハッシュ化に関する利点には次のようなものがあります。

- コスト効果
- 速度
- 追加のメタデータやネットワークトラフィックの必要がない、決定論的なキーの場所

データ分散は、永続性とフォールトトラレンスを提供するため、余分のない十分なコピーがクラスター内に確実に存在するようにします。余分なコピーは環境のスケーラビリティーを低下させます。

永続性

JBoss Data Grid は、**CacheStore** インターフェースと複数の高パフォーマンス実装 (JDBC キャッシュストアやファイルシステムベースのキャッシュストアなど) を公開します。キャッシュストアを使用すると、起動時にキャッシュを事前設定し、関連データが破損しないように保持できます。また、キャッシュストアは必要時にデータをディスクへオーバーフローし、メモリー不足にならないようにします。

言語のバインディング

JBoss Data Grid は、多くの一般的なプログラミング言語向けの既存クライアントを持つ Memcached プロトコルと、Hot Rod と呼ばれる最適化された JBoss Data Grid 固有のプロトコルの両方をサポートします。そのため、JBoss Data Grid は Java に限定されず、すべての主要な Web サイトやアプリケーションに使用できます。さらに、RESTful API を介して HTTP プロトコルを使用すると、リモートキャッシュにアクセスできます。

管理

数百個またはそれ以上のサーバーが存在するグリッド環境では、管理の実行は重要な機能になります。エンタープライズネットワーク管理ソフトウェアである JBoss Operations Network は、複数の JBoss Data Grid インスタンスを管理するのに最適なツールです。JBoss Operations Network の機能を使用すると、キャッシュマネージャーやキャッシュインスタンスを簡単かつ効率的に監視できます。

リモートデータグリッド

JBoss Data Grid は、アプリケーションサーバーアーキテクチャー全体をスケールアップしてデータグリッドをスケールアップする代わりに、アプリケーションサーバーアーキテクチャーに依存せずにデータグリッドインフラストラクチャーを独自にアップグレードできるリモートクライアントサーバーモードを提供します。さらに、データグリッドサーバーをアプリケーションサーバー以外のリソースへ割り当てることができます。また、独自にデータグリッドをアップグレードでき、データグリッド内でアプリケーションを再デプロイできます。

バグを報告する

1.5. RED HAT JBOSS DATA GRID のバージョン情報

Red Hat JBoss Data Grid は、データグリッドソフトウェアのオープンソースコミュニティー版である Infinispan が基盤となっています。Infinispan は、高ストレス環境で試行やテストを行って証明された JBoss Cache のコード、設計、およびアイデアを使用します。このようなデプロイメントの前歴があるため、JBoss Data Grid の初版リリースはバージョン 6.0 になっています。

以下の表は、JBoss Data Grid と Infinispan のバージョンを比較したものです。

表1.1 JBoss Data Grid および Infinispan の関係

JBoss Data Grid 製品	Infinispan バージョン
JBoss Data Grid 6.0.0	Infinispan 5.1.5
JBoss Data Grid 6.0.1	Infinispan 5.1.7
JBoss Data Grid 6.1.0	Infinispan 5.2.4
JBoss Data Grid 6.2.0	Infinispan 6.0.1
JBoss Data Grid 6.3.0	Infinispan 6.1.0
JBoss Data Grid 6.3.1	Infinispan 6.1.1
JBoss Data Grid 6.4.0	Infinispan 6.2.0
JBoss Data Grid 6.4.1	Infinispan 6.2.1
JBoss Data Grid 6.5.0	Infinispan 6.3.0
JBoss Data Grid 6.5.1	Infinispan 6.3.1
JBoss Data Grid 6.6.0	Infinispan 6.4.0
JBoss Data Grid 7.0.0	Infinispan 8.3.0

注記

Red Hat JBoss Data Grid 6.2.0 の後にリリースされた JBoss Data Grid に直接関係するバージョンの Infinispan はありません。JBoss Data Grid 6.3.0 以降の Infinispan バージョンは、Infinispan の内部バージョン番号 (未リリースのバージョンも含む) が基になっています。

バグを報告する

1.6. RED HAT JBOSS DATA GRID のキャッシュアーキテクチャー

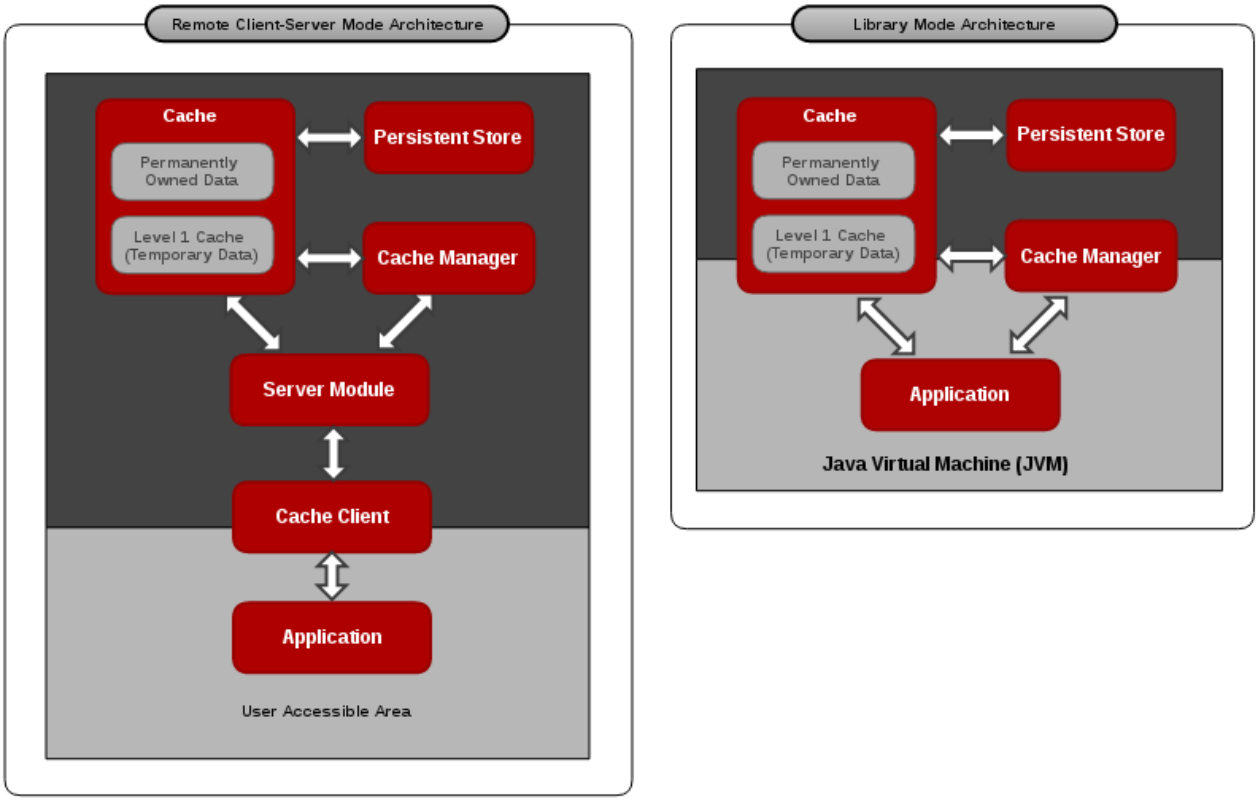


図1.1 Red Hat JBoss Data Grid のキャッシュアーキテクチャー

Red Hat JBoss Data Grid のキャッシュインフラストラクチャーは、JBoss Data Grid の各使用モード (ライブラリーおよびリモートクライアントサーバー) での個々の要素と要素同士の対話を表しています。明確にするため、キャッシュアーキテクチャーの図は以下の 2 つの部分に分割されています。

- 図の濃い灰色部分にあるのは、ユーザーが直接対話できない要素です。リモートクライアントサーバーモードでは、永続ストア、キャッシュ、キャッシュマネージャー、1 次キャッシュ、およびサーバーモジュールが含まれます。ライブラリーモードでは、ユーザーは永続ストアおよび 1 次キャッシュと直接対話できません。

- 図の薄い灰色部分にあるのは、ユーザーが直接対話できる要素です。リモートクライアントサーバーモードでは、アプリケーションとキャッシュクライアントが含まれます。ライブラリーモードでは、ユーザーはキャッシュ、キャッシュマネージャー、およびアプリケーションと対話できます。

キャッシュアーキテクチャーの要素

JBoss Data Grid のキャッシュアーキテクチャーには以下の要素が含まれています。

1. 永続ストアはオプションのコンポーネントです。データグリッドのシャットダウン後に、修復用にキャッシュされたエントリーを永久に格納します。

2. 1 次キャッシュ (L1 キャッシュ) は、リモートキャッシュエントリーが最初にアクセスされた後にそれらのエントリーを格納し、同じエントリーがその後使用されるたびに不必要なリモートフェッチ操作が行われないようにします。

3. キャッシュマネージャーはキャッシュインスタンスのライフサイクルを制御し、必要時にキャッシュインスタンスを格納および読み出しできます。

4. キャッシュは、キーと値のエントリーを格納および読み出しするための主なコンポーネントです。

ライブラリーおよびリモートクライアントサーバーモードのアーキテクチャー

ライブラリーモードでは、アプリケーション (ユーザーコード) はキャッシュおよびキャッシュマネージャーコンポーネントと直接対話できます。この場合、アプリケーションは同じ Java 仮想マシン (JVM) にあり、Cache および Cache Manager Java API メソッドを直接呼び出せます。

リモートクライアントサーバーモードでは、アプリケーションはキャッシュと直接対話しません。また、通常アプリケーションは異なる物理ホストの別の JVM にあるか、Java アプリケーションである必要がありません。この場合、アプリケーションは Memcached、Hot Rod、REST などのサポートされるプロトコルの 1 つを使用してネットワーク上でリモートの JBoss Data Grid サーバーと通信するキャッシュクライアントを使用します。適切なサーバーモジュールがサーバー側の通信を処理します。リクエストがリモートでサーバーに送信されると、プロトコルを変換してキャッシュコンポーネントで実行される操作に戻し、データを格納および読み出しします。

バグを報告する

1.7. RED HAT JBOSS DATA GRID の API

Red Hat JBoss Data Grid は以下のプログラム可能な API を提供します。

- キャッシュ
- バッチ化
- グループ化
- 永続性 (旧名 CacheStore)
- ConfigurationBuilder
- 外部化
- 通知 (通知とリスナーを処理するため、リスナー API とも呼ばれます)

JBoss Data Grid は、リモートクライアントサーバーモードでデータグリッドと対話するために以下の API を提供します。

- 非同期 API (リモートクライアントサーバーモードで Hot Rod クライアントを併用する場合のみ使用可能)
- REST インターフェース
- Memcached インターフェース
- Hot Rod インターフェース
    - RemoteCache API

バグを報告する

パート II. RED HAT JBOSS DATA GRID のダウンロードおよびインストール

第2章 RED HAT JBOSS DATA GRID のダウンロード

2.1. RED HAT JBOSS DATA GRID インストールの前提条件

Java 仮想マシン (Java 6.0 以降と互換性がある) がインストールされていることが Red Hat JBoss Data Grid をセットアップするための前提条件になります。

バグを報告する

2.2. JAVA 仮想マシン

Java 仮想マシン (JVM) は、ホストオペレーティングシステムで Java プログラムを実行する仮想環境です。JVM は抽象コンピューターとして動作し、Java プログラミングコードをマシン言語に変換する、プラットフォームに依存しない実行環境です。Java 仮想マシン (JVM) は、コンパイルされた Java プログラムと基礎のハードウェアプラットフォームおよびオペレーティングシステムの間に抽象層を提供し、Java をポータブルにします。

Red Hat は、Red Hat Enterprise Linux システムで快適に動作し、オープンソースのサポートされる Java 仮想マシンである OpenJDK Java プラットフォームの使用を推奨します。Windows では、Oracle JDK 1.6 インストールが推奨されます。

バグを報告する

2.3. RED HAT ENTERPRISE LINUX に OPENJDK をインストール

手順2.1 Red Hat Enterprise Linux に OpenJDK をインストール

1. **ベースチャンネルのサブスクライブ**
   RHN ベースチャンネルより OpenJDK を取得します。Red Hat Enterprise Linux のインストールはデフォルトでこのチャンネルにサブスクライブされています。

2. **パッケージのインストール**
   yum ユーティリティを使用して OpenJDK をインストールします。

   ```
   $ sudo yum install java-1.6.0-openjdk-devel
   ```

3. **OpenJDK がシステムデフォルトであることを確認**
   次のように、正しい JDK がシステムデフォルトとして設定されていることを確認します。

   a. root 特権を持つユーザーとしてログインし、alternatives コマンドを実行します。

   ```
   $ /usr/sbin/alternatives --config java
   ```

   b. OpenJDK のバージョンに応じて **/usr/lib/jvm/jre-1.6.0-openjdk.x86_64/bin/java** または **/usr/lib/jvm/jre-1.7.0-openjdk.x86_64/bin/java** を選択します。

   c. 次のコマンドを使用して **javac** を設定します。

   ```
   $ /usr/sbin/alternatives --config javac
   ```

d. 使用した OpenJDK のバージョンに応じて、**/usr/lib/jvm/java-1.6.0-openjdk/bin/java** または **/usr/lib/jvm/java-1.7.0-openjdk/bin/java** を選択します。

バグを報告する

2.4. JBOSS DATA GRID のダウンロードおよびインストール

次の手順を使用して Red Hat JBoss Data Grid をダウンロードおよびインストールします。

1. Red Hat カスタマーポータルから JBoss Data Grid をダウンロードします。
2. ダウンロードしたファイルを検証します。
3. JBoss Data Grid をインストールします。

バグを報告する

2.4.1. Red Hat JBoss Data Grid のダウンロード

以下の手順にしたがって、カスタマーポータルから Red Hat JBoss Data Grid をダウンロードします。

手順2.2 JBoss Data Grid のダウンロード

1. カスタマーポータル (https://access.redhat.com) にログインします。
2. ページ最上部にある **ダウンロード** ボタンをクリックします。
3. **製品のダウンロード** ページで**Red Hat JBoss Data Grid** をクリックします。
4. **Version:** ドロップダウンメニューから適切な JBoss Data Grid バージョンを選択します。
5. 表示されるリストから適切なファイルをダウンロードします。

バグを報告する

2.4.2. Red Hat カスタマーポータルについて

*Red Hat カスタマーポータル*は、Red Hat のナレッジリソースやサブスクリプションリソースを管理する集中プラットフォームです。*Red Hat カスタマーポータル*では、以下のことを行えます。

- Red Hat エンタイトルメントやサポート契約の管理および維持。
- 正式サポートされたソフトウェアのダウンロード。
- 製品ドキュメントや Red Hat ナレッジベースの利用。
- グローバルサポートサービスへの連絡。
- Red Hat 製品のバグの登録。

カスタマーポータルは https://access.redhat.com からアクセスしてください。

バグを報告する

2.4.3. チェックサムの検証

チェックサムの検証は、ダウンロードされたファイルが破損していないことを確認するために使用されます。チェックサムの検証には、デジタルデータの任意のブロックから固定サイズのデータ (またはチェックサム) を算出するアルゴリズムが使用されます。異なるユーザーが同じアルゴリズムを使用して特定ファイルのチェックサムを算出しても、結果は同じになります。そのため、サプライヤーと同じアルゴリズムを使用してダウンロードされたファイルのチェックサムを算出する場合、チェックサムが一致すればファイルの整合性が確認されます。チェックサムが一致しない場合は、ダウンロードの処理中にファイルが破損したことになります。

バグを報告する

2.4.4. ダウンロードされたファイルの検証

手順2.3 ダウンロードされたファイルの検証

1. Red Hat カスタマーポータルからダウンロードしたファイルにエラーがないことをポータルサイト上で検証するには、パッケージの **Software Details** ページに移動します。このページには、**MD5** および **SHA256** のチェックサム値が記載されています。チェックサム値を使用してファイルの整合性をチェックします。

2. ターミナルウインドウを開き、ダウンロードしたファイルを引数として指定して **md5sum** または **sha256sum** コマンドを実行します。コマンドの出力として、ファイルのチェックサム値が表示されます。

3. コマンドによって返されたチェックサム値と、そのファイルの **Software Details** ページに記載されている値を比較します。

注記

Microsoft Windows にはチェックサムツールが同梱されていません。Windows オペレーティングシステムのユーザーはサードパーティーの製品をダウンロードする必要があります。

結果

2 つのチェックサム値が同じである場合、ファイルに変更や破損がないため、安心して使用することができます。

2 つのチェックサム値が同じでない場合、再度ファイルをダウンロードします。チェックサム値が異なる場合、ダウンロード中にファイルが破損したか、サーバーへアップロードされた後に変更が加えられたことを意味します。数回ダウンロードしてもチェックサムの検証に失敗する場合は、Red Hat サポートまでご連絡ください。

バグを報告する

2.4.5. Red Hat JBoss Data Grid のインストール

要件

適切なバージョン、プラットフォームおよびファイルタイプを見つけ、カスタマーポータルから Red Hat JBoss Data Grid をダウンロードする必要があります。

手順2.4 JBoss Data Grid のインストール

1. ダウンロードした JBoss Data Grid パッケージをマシン上の希望の場所にコピーします。
2. 次のコマンドを実行して、ダウンロードした JBoss Data Grid パッケージを展開します。

```
$ unzip JDG_PACKAGE
```

   JDG_PACKAGE は、Red Hat カスタマーポータルからダウンロードした JBoss Data Grid 使用モードパッケージの名前に置き換えます。

3. 展開したディレクトリーは *$JDG_HOME* とします。

バグを報告する

2.4.6. Red Hat のドキュメントサイト

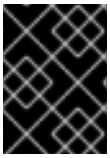
Red Hat の公式ドキュメントサイトは https://access.redhat.com/site/documentation/ になります。本書を含む最新バージョンのドキュメントをご覧になれます。

バグを報告する

第3章 MAVEN リポジトリーのインストールおよび使用

3.1. MAVEN

Apache Maven は、ソフトウェアプロジェクトの作成、管理、構築を行う Java アプリケーションの開発で使用される分散型ビルド自動化ツールです。Maven は Project Object Model (POM) と呼ばれる標準の設定ファイルを利用して、プロジェクトの定義や構築プロセスの管理を行います。POM ファイルはモジュールやコンポーネントの依存関係、ビルドの順番、結果となるプロジェクトパッケージのターゲットを記述し、XML ファイルを使用して出力します。こうすることで、プロジェクトが正しく統一された状態で構築されるようにします。



重要

Red Hat JBoss Data Grid のすべてのクイックスタートおよび一般的な使用には、Maven 3 以上が必要になります。

Maven のダウンロードおよびインストール手順については、Maven の Download ページ (http://maven.apache.org/download.html) を参照してください。

バグを報告する

3.2. 必要な MAVEN リポジトリー

Red Hat JBoss Data Grid のクイックスタートを使用するには、以下の Maven リポジトリーが設定されている必要があります。

- JBoss Data Grid Maven リポジトリー
- **techpreview-all-repository** (https://maven.repository.redhat.com/techpreview/all/)

両方の Maven リポジトリーは同じ方法でインストールされます。よって、次の手順は両方のリポジトリーに適用されます。

バグを報告する

3.3. MAVEN リポジトリーのインストール

必要なリポジトリーをインストールする方法は 3 つあります。

1. ローカルファイルシステム上 (「ローカルファイルシステムリポジトリーのインストール」)。
2. Apache Web Server 上 (「Apache httpd リポジトリーのインストール」).
3. Maven リポジトリーマネージャーを使用 (「Maven リポジトリーマネージャーのインストール」)。

ご使用の環境に最も適した方法を使用してください。

バグを報告する

3.3.1. ローカルファイルシステムリポジトリーのインストール

これは小チームでの初期テストに最も適した方法です。 手順にしたがって、Red Hat JBoss Data Grid と JBoss Enterprise Application Platform の Maven リポジトリーをローカルファイルシステムのディレクトリーに展開します。

手順3.1 ローカルファイルシステムリポジトリーのインストール (JBoss Data Grid)

1. **カスタマーポータルへのログイン**
   ブラウザーでカスタマーポータルページ (https://access.redhat.com/home) に移動し、ログインします。

2. **JBoss Data Grid リポジトリーファイルのダウンロード**
   Red Hat カスタマーポータルから **jboss-datagrid-*{VERSION}*-maven-repository.zip** をダウンロードします。

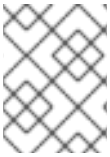
3. ローカルファイルシステムのディレクトリー (例: **$JDG_HOME/projects/maven-repositories/**) にファイルを展開します。

上記の手順により、**maven-repository/** という名前のサブディレクトリーに Maven レポジトリーが含まれる新しい **jboss-datagrid-${jdg-version}-maven-repository** ディレクトリーが作成されます。

バグを報告する

3.3.2. Apache httpd リポジトリーのインストール

この例では、Apache httpd で使用する JBoss Data Grid Maven リポジトリーをダウンロードする手順を示します。Web サーバーにアクセスできる開発者は Maven リポジトリーにもアクセスできるため、このオプションはマルチユーザーの開発環境や複数のチームにまたがる開発環境に適しています。



注記

以下の手順を機能させるには Apache httpd を設定する必要があります。Apache の設定手順については、Apache HTTP Server Project を参照してください。

1. Web ブラウザーを開き、https://access.redhat.com/jbossnetwork/restricted/listSoftware.html?product=data.grid にアクセスします。

2. リストから **Red Hat JBoss Data Grid 7.0.0 Maven Repository** を見つけます。

3. **ダウンロード** ボタンをクリックし、リポジトリーが含まれる .zip ファイルをダウンロードします。

4. Apache サーバー上で Web にアクセス可能なディレクトリーにファイルを展開します。

5. Apache を設定し、作成されたディレクトリーの読み取りアクセスとディレクトリの閲覧を許可します。

バグを報告する

3.3.3. Maven リポジトリーマネージャーのインストール

これは、リポジトリーマネージャーをすでに使用している場合に最適な方法です。

Red Hat JBoss Data Grid および JBoss Enterprise Application Server リポジトリーは、Maven リポジトリーマネージャーとそのドキュメントを使用してインストールします。このようなリポジトリーマネージャーの例を以下に示します。

- Apache Archiva: http://archiva.apache.org/
- JFrog Artifactory: http://www.jfrog.com/products.php
- Sonatype Nexus: http://nexus.sonatype.org/ (詳細は「Nuxus を用いた JBoss Enterprise Application Platform リポジトリーのインストール」 を参照してください)

バグを報告する

3.4. MAVEN レポジトリーの設定

インストールされた Red Hat JBoss Data Grid の Maven リポジトリーを設定するには、**settings.xml** ファイルを編集します。このファイルは以下の 2 つの場所のいずれかで設定できます。

1. ユーザーレベル: Maven ユーザーの設定は **${user.home}/.m2/** ディレクトリーにあります。
   - Linux または Mac 環境の場合、通常これは **~/.m2/** になります。
   - Windows 環境の場合、通常これは **\Documents and Settings\.m2** または **\Users\.m2** になります。
2. グローバルレベル: マシン上のすべてのユーザーの設定です。これらのユーザーがすべて同じ Maven インストールを使用していることが想定されます。この設定は通常 **${maven.home}/conf/settings.xml** で行われます。

サンプルの Maven 設定を表示するには、「Maven リポジトリーの設定例」を参照してください。また、Maven の設定方法についての詳細は、Maven ドキュメントを参照してください。

バグを報告する

3.4.1. JBoss Data Grid Maven リポジトリーのオフライン環境での設定

一部の環境では Maven リポジトリーをオフラインで利用可能にすることが推奨されます。この設定を行うには、以下の手順を実行します。

前提条件:

- JBoss Data Grid Maven リポジトリーが、これが参照される内部ネットワークにダウンロードされている。
- Sonatype Nexus または Apache Archiva などの Maven の依存関係を含む内部リポジトリーがネットワークで利用できる。

手順3.2 オフラインでの使用に向けた JBoss Data Grid Maven リポジトリーの設定

1. 「ローカルファイルシステムリポジトリーのインストール」の手順に従って、ダウンロードされた JBoss Data Grid Maven リポジトリーをローカルにインストールします。
2. 「Maven レポジトリーの設定」にあるように **settings.xml** がローカルに抽出されたリポジトリーを指すように更新します。サンプル設定は以下のようになります。

```
<?xml version="1.0" encoding="UTF-8"?>

<settings xmlns="http://maven.apache.org/SETTINGS/1.0.0"
xmlns:xsi="http://www.w3.org/2001/XMLSchema-instance"
xsi:schemaLocation="http://maven.apache.org/SETTINGS/1.0.0
http://maven.apache.org/xsd/settings-1.0.0.xsd">

   <profiles>
      <profile>
         <id>jboss-datagrid-repository</id>
         <repositories>
            <repository>
               <id>jboss-datagrid-repository</id>
               <name>JBoss Data Grid Maven Repository</name>
               <url>file:///path/to/jboss-datagrid-7.0.0.GA-maven-
repository/maven-repository</url>
               <layout>default</layout>
               <releases>
                  <enabled>true</enabled>
                  <updatePolicy>never</updatePolicy>
               </releases>
               <snapshots>
                  <enabled>false</enabled>
                  <updatePolicy>never</updatePolicy>
               </snapshots>
            </repository>
         <pluginRepositories>
            <pluginRepository>
               <id>jboss-datagrid-repository</id>
               <name>JBoss Data Grid Maven Repository</name>
               <url>file:///path/to/jboss-datagrid-7.0.0.GA-maven-
repository/maven-repository</url>
               <layout>default</layout>
               <releases>
                  <enabled>true</enabled>
                  <updatePolicy>never</updatePolicy>
               </releases>
               <snapshots>
                  <enabled>false</enabled>
                  <updatePolicy>never</updatePolicy>
               </snapshots>
            </pluginRepository>
         </pluginRepositories>
      </profile>
   </profiles>
   <activeProfiles>
      <!-- make the profile active by default -->
      <activeProfile>jboss-datagrid-repository</activeProfile>
   </activeProfiles>

</settings>
```

3. プロジェクトがローカルにビルドできることを確認します。

バグを報告する

3.4.2. 次のステップ

最新の入手可能なバージョンの Red Hat JBoss Data Grid をインストールし、Maven を設定した後、「新しい Red Hat JBoss Data Grid プロジェクトの作成」を参照して JBoss Data Grid を初めて使用する方法を確認してください。

バグを報告する

パート III. JBOSS DATA GRID でサポートされるコンテナー

第4章 JBOSS DATA GRID をサポートされるコンテナーと使用

Red Hat JBoss Data Grid は以下のランタイムで使用できます。

- アプリケーションによって起動された Java SE。
- スタンドアロン JBoss Data Grid サーバーとして。
- アプリケーションのライブラリーとしてバンドルされ、アプリケーションサーバーへデプロイされ、アプリケーションによって起動 (たとえば、JBoss Data Grid は Tomcat または Weblogic と使用できます)。
- OSGi ランタイム環境内 (この場合、Apache Karaf)。

Red Hat JBoss Data Grid でサポートされるコンテナーの一覧は、『リリースノート』または https://access.redhat.com/knowledge/articles/115883 のサポート情報を参照してください。

バグを報告する

4.1. JBOSS EAP での JBOSS DATA GRID のデプロイ (ライブラリーモード)

Red Hat JBoss Data Grid は Red Hat JBoss Enterprise Application Platform 6.x 用のモジュールを提供します。これらのモジュールを使用する場合は、JBoss Data Grid ライブラリーをユーザーのデプロイメントに含める必要がありません。JBoss EAP にすでに含まれている Infinispan モジュールとの競合を防ぐため、JBoss Data Grid モジュールは別のスロット内に置かれ、JBoss Data Grid のバージョン (*major.minor*) によって識別されます。

注記

JBoss EAP モジュールは JBoss EAP には含まれていません。http://access.redhat.com のカスタマーポータルに移動し、これらのモジュールを Red Hat JBoss Data Grid のダウンロードページからダウンロードします。

JBoss EAP で JBoss Data Grid をデプロイするには、以下の方法の 1 つを用いて JBoss Data Grid モジュールの依存関係をアプリケーションのクラスパス (JBoss EAP デプロイヤー) に追加します。

- 依存関係を **jboss-deployment-structure.xml** ファイルに追加します。
- 依存関係を **MANIFEST.MF** ファイルに追加します。
- Maven より **MANIFEST.MF** ファイルを生成します。

jboss-deployment-structure.xml ファイルへ依存関係を追加

以下の設定を **jboss-deployment-structure.xml** ファイルに追加します。

```
<jboss-deployment-structure xmlns="urn:jboss:deployment-structure:1.2">
    <deployment>
        <dependencies>
            <module name="org.infinispan" slot="jdg-7.0"
services="export"/>
        </dependencies>
    </deployment>
</jboss-deployment-structure>
```

注記

jboss-deployment-structure.xml ファイルの詳細は、Red Hat JBoss Enterprise Application Platform のドキュメントを参照してください。

MANIFEST.MF ファイルへ依存関係を追加

次のように依存関係を **MANIFEST.MF** ファイルに追加します。

例4.1 MANIFEST.MF ファイルの例

```
Manifest-Version: 1.0
Dependencies: org.infinispan:jdg-7.0 services
```

最初の行はこの例と同じになります。必要な依存関係に応じて、以下のいずれかをファイルの 2 行目に追加します。

- JBoss Data Grid コア

```
Dependencies: org.infinispan:jdg-7.0 services
```

- 埋め込みクエリー

```
Dependencies: org.infinispan:jdg-7.0 services,
org.infinispan.query:jdg-7.0 services, org.infinispan.query.dsl:jdg-
7.0 services
```

- JDBC キャッシュストア

```
Dependencies: org.infinispan:jdg-7.0 services,
org.infinispan.persistence.jdbc:jdg-7.0 services
```

- JPA キャッシュストア

```
Dependencies: org.infinispan:jdg-7.0 services,
org.infinispan.persistence.jpa:jdg-7.0 services
```

- LevelDB キャッシュストア

```
Dependencies: org.infinispan:jdg-7.0 services,
org.infinispan.persistence.leveldb:jdg-7.0 services
```

- CDI

```
Dependencies: org.infinispan:jdg-7.0 services,
org.infinispan.cdi:jdg-7.0 meta-inf
```

Maven より MANIFEST.MF ファイルを生成

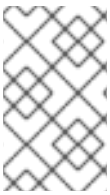
MANIFEST.MF ファイルはビルド中 (JAR または WAR プロセス中) に生成されます。**MANIFEST.MF** ファイルに依存関係を追加する代わりに、以下を **pom.xml** ファイルに追加し、Maven で直接依存関係を設定します。

```
<plugin>
 <artifactId>maven-war-plugin</artifactId>
 <version>2.4</version>
 <configuration>
  <failOnMissingWebXml>false</failOnMissingWebXml>
  <archive>
   <manifestEntries>
    <Dependencies>org.infinispan:jdg-7.0 services</Dependencies>
   </manifestEntries>
  </archive>
 </configuration>
</plugin>
```

バグを報告する

4.2. JBOSS EAP での JBOSS DATA GRID のデプロイ (リモートクライアントサーバーモード)

Red Hat JBoss Data Grid は Red Hat JBoss Enterprise Application Platform 6.x 用のモジュールを提供します。これらのモジュールを使用する場合は、JBoss Data Grid ライブラリーをユーザーのデプロイメントに含める必要がありません。JBoss EAP にすでに含まれている Infinispan モジュールとの競合を防ぐため、JBoss Data Grid モジュールは別のスロット内に置かれ、JBoss Data Grid のバージョン (*major.minor*) によって識別されます。



注記

JBoss EAP モジュールは JBoss EAP には含まれていません。http://access.redhat.com のカスタマーポータルに移動し、これらのモジュールを Red Hat JBoss Data Grid のダウンロードページからダウンロードします。

JBoss EAP で JBoss Data Grid をデプロイするには、以下の方法の 1 つを用いて JBoss Data Grid モジュールの依存関係をアプリケーションのクラスパス (JBoss EAP デプロイヤー) に追加します。

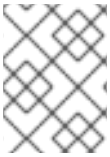
- 依存関係を **jboss-deployment-structure.xml** ファイルに追加します。
- 依存関係を **MANIFEST.MF** ファイルに追加します。

jboss-deployment-structure.xml ファイルへ依存関係を追加

以下の設定を **jboss-deployment-structure.xml** ファイルに追加します。

```
<jboss-deployment-structure xmlns="urn:jboss:deployment-structure:1.2">
    <deployment>
        <dependencies>
            <module name="org.infinispan.commons" slot="jdg-7.0"
services="export"/>
            <module name="org.infinispan.client.hotrod" slot="jdg-7.0"
services="export"/>
```

```
        </dependencies>
    </deployment>
</jboss-deployment-structure>
```



注記

jboss-deployment-structure.xml ファイルの詳細は、Red Hat JBoss Enterprise Application Platform のドキュメントを参照してください。

MANIFEST.MF ファイルへ依存関係を追加

次のように依存関係を **MANIFEST.MF** ファイルに追加します。

例4.2 MANIFEST.MF ファイルの例

```
Manifest-Version: 1.0
Dependencies: org.infinispan.commons:jdg-7.0 services,
org.infinispan.client.hotrod:jdg-7.0 services
```

最初の行はこの例と同じになります。必要な依存関係に応じて、以下のいずれかをファイルの 2 行目に追加します。

- 基本の Hot Rod クライアント

```
org.infinispan.commons:jdg-7.0 services,
org.infinispan.client.hotrod:jdg-7.0 services
```

- リモートクエリー機能を持つ Hot Rod クライアント

```
org.infinispan.commons:jdg-7.0 services,
org.infinispan.client.hotrod:jdg-7.0 services,
org.infinispan.query.dsl:jdg-7.0 services, org.jboss.remoting3
```

バグを報告する

4.2.1. Hot Rod クライアントでのカスタムクラスの使用

以下の 2 つの方法のいずれかを使用して Hot Rod クライアントでカスタムクラスを使用できます。

- **オプション 1:** 以下の例のように、Hot Rod クライアントの設定ビルダーでデプロイメントのクラスローダーを参照します。

例4.3 ConfigurationBuilder インスタンスでのカスタムクラスローダーの参照

```
import
org.infinispan.client.hotrod.configuration.ConfigurationBuilder;
[...]
ConfigurationBuilder config = new ConfigurationBuilder();
config.marshaller(new
GenericJBossMarshaller(Thread.currentThread().getContextClassLoade
r()));
```

- **オプション 2:** カスタムクラスを JBoss EAP 内の独自のモジュールとしてインストールし、新規に作成されたモジュールの依存関係は JBoss Data Grid モジュール (**${EAP_HOME}/modules/system/layers/base/org/infinispan/commons/jdg-6.x/module.xml**) に追加する必要があります。

バグを報告する

4.3. JBOSS ENTERPRISE WEB SERVER での JBOSS DATA GRID のデプロイ

Red Hat JBoss Data Grid は JBoss Enterprise Web Server を ライブラリーおよびリモートクライアントサーバーモードでサポートします。JBoss Data Grid を JBoss Enterprise Web Server と使用するには、Web アプリケーションで JDG ライブラリーをバンドルし、サーバーでアプリケーションをデプロイします。

JBoss Data Grid を JBoss Enterprise Web Server にデプロイする方法の詳細は「JBoss Enterprise Web Server を用いた (非トランザクション) CarMart クイックスタート」および「リモートクライアントサーバーモードでの (非トランザクション) CarMart クイックスタート (JBoss Enterprise Web Server)」を参照してください。

バグを報告する

4.4. WEBLOGIC サーバーでの WEB アプリケーションのデプロイ (ライブラリーモード)

Red Hat JBoss Data Grid は WebLogic 12c アプリケーションサーバーをライブラリーモードでサポートします。 以下の手順は、WebLogic サーバーで Web アプリケーションをデプロイする方法になります。

前提条件

Web アプリケーションをデプロイするための前提条件は次のとおりです。

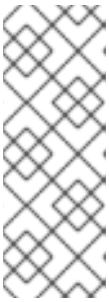
1. WebLogic Server 12c
2. JBoss Data Grid のライブラリー (埋め込み) モード

手順4.1 WebLogic サーバーでの Web アプリケーションのデプロイ

1. **Web アプリケーションの作成**
   Web アプリケーションを作成し、ライブラリーを **WEB-INF** フォルダーに追加します。
2. **weblogic.xml デプロイメント記述子の作成**
   以下の要素が含まれる **weblogic.xml** デプロイメント記述子を作成します。

```
<?xml version="1.0" encoding="UTF-8"?>
<weblogic-web-app
        xmlns="http://www.bea.com/ns/weblogic/90"
        xmlns:xsi="http://www.w3.org/2001/XMLSchema-instance"
        xsi:schemaLocation="http://www.bea.com/ns/weblogic/90
http://www.bea.com/ns/weblogic/90/weblogic-web-app.xsd">
    <container-descriptor>
        <prefer-web-inf-classes>true</prefer-web-inf-classes>
    </container-descriptor>
</weblogic-web-app>
```

-



注記

prefer-web-inf-classes クラスは、**WEB-INF/lib** フォルダーにあるライブラリーおよびクラスが WebLogic サーバーにバンドルされたデフォルトのライブラリーよりも優先されることを示します。たとえば、WebLogic サーバーの **commons-pool.jar** ファイルのバージョンは 1.4 で、クラスローダーによって自動的にロードされますが、Hot Rod クライアントはこのライブラリーの新しいバージョンを使用します。

3. **Web アプリケーションを Web アーカイブファイルへパック**
   Web アプリケーションの Web アプリケーションアーカイブ (WAR) ファイルを作成し、JBoss Data Grid ライブラリーと WebLogic デプロイメント記述子ファイルが **WEB-INF** フォルダーにあることを確認します。

4. **WebLogic サーバーでのアプリケーションのデプロイ**
   Infinispan CDI モジュールを使用して Web アプリケーションをデプロイするには、WebLogic サーバーが稼働している場合は停止し、パッチ (パッチファイル **p17424706_121200_Generic.zip**) を適用した後、WebLogic サーバーを再起動します。Infinispan CDI モジュールが使用されない場合は、通常どおり Web アプリケーションをデプロイします。

   WebLogic サーバーへのパッチ適用に関する詳細は、Oracle の Web サイトにある『Oracle patch database』を参照してください。

バグを報告する

4.5. WEBLOGIC サーバーでの WEB アプリケーションのデプロイ (リモートクライアントサーバーモード)

Red Hat JBoss Data Grid は WebLogic 12c アプリケーションサーバーをリモートクライアントサーバーモードでサポートします。 以下の手順は、WebLogic サーバーで Web アプリケーションをデプロイする方法になります。

手順4.2 WebLogic サーバーでの Web アプリケーションのデプロイ

1. WebLogic サーバーをインストールするには http://docs.oracle.com/cd/E24329_01/doc.1211/e24492/toc.htm を参照してください。

2. リモートクライアントサーバーモードの JBoss Data Grid を設定し、キャッシュ、キャッシュコンテナー、およびエンドポイント設定を定義します。設定後、JBoss Data Grid を起動し、Hot Rod エンドポイントが設定されたポートでリッスンしていることを確認します。リモートクライアントサーバーモードの JBoss Data Grid の設定に関する詳細は、*7章Red Hat JBoss Data Grid のリモートクライアントサーバーモードでの実行*を参照してください。

3. Web アプリケーションを作成し、Maven が使用される場合は **infinispan-remote** ライブラリーを依存関係として追加します。

4. 以下の要素が含まれる **weblogic.xml** デプロイメント記述子を作成します。

```
<?xml version="1.0" encoding="UTF-8"?>
<weblogic-web-app
        xmlns="http://www.bea.com/ns/weblogic/90"
        xmlns:xsi="http://www.w3.org/2001/XMLSchema-instance"
```

```
        xsi:schemaLocation="http://www.bea.com/ns/weblogic/90
http://www.bea.com/ns/weblogic/90/weblogic-web-app.xsd">
    <container-descriptor>
        <prefer-web-inf-classes>true</prefer-web-inf-classes>
    </container-descriptor>
</weblogic-web-app>
```

注記

prefer-web-inf-classes クラスは、**WEB-INF/lib** フォルダーにあるライブラリーおよびクラスが WebLogic サーバーにバンドルされたデフォルトのライブラリーよりも優先されることを示します。たとえば、WebLogic サーバーの **commons-pool.jar** ファイルのバージョンは 1.4 で、クラスローダーによって自動的にロードされますが、Hot Rod クライアントはこのライブラリーの新しいバージョンを使用します。

5. デプロイメント記述子ファイルを **WEB-INF** フォルダーに追加します。
6. **infinispan-remote** 依存関係が **pom.xml** ファイルに追加されたことを確認した後、Maven プラグインを使用して Web アーカイブを作成します。

   この代わりに、手動で Web アーカイブを作成し、手動でライブラリーに追加することもできます。
7. WebLogic サーバーでアプリケーションをデプロイし、Web アプリケーション内に埋め込まれた Hot Rod クライアントがリモート JBoss Data Grid サーバーに接続することを確認します。

バグを報告する

4.6. KARAF (OSGI) での RED HAT JBOSS DATA GRID の実行

Apache Karaf は強力で軽量な OSGi ベースのランタイムコンテナーで、コンポーネントとアプリケーションがデプロイされます。OSGi は、スタンドアロン JVM 環境に存在しない動的コンポーネントモデルを実装します。Karaf などの OSGi コンテナーには、アプリケーションのライフサイクルを管理するためのツールが含まれています。

バージョン番号を含む、各モジュール間の依存関係はすべて明示的に指定する必要があります。同じ名前のクラスが複数存在する場合は、OSGi の厳格なルールにしたがって、バンドルによって使用されるクラスが指定されます。

バグを報告する

4.6.1. Karaf で Red Hat JBoss Data Grid のデプロイメントを実行 (リモートクライアントサーバー)

Red Hat JBoss Data Grid の Hot Rod クライアントは、Karaf などの OSGi ベースのコンテナーで実行できます。 Karaf にデプロイされたクライアントアプリケーションは既存の JBoss Data Grid サーバーに接続できます。

JBoss Data Grid の Maven リポジトリーを使用して Karaf を設定します。また、JBoss Data Grid は **org/infinispan/infinispan-remote/*${VERSION}*** にある **features** ファイルを必要とします。このファイルは OSGi の Hot Rod クライアントの依存関係をすべてリストし、さらに Karaf (バージョン 2.3.3 または 3.0) への機能のインストールを単純化します。

バグを報告する

4.6.2. Karaf への Hot Rod クライアント機能のインストール

Red Hat JBoss Data Grid の Hot Rod 機能は、次のように Karaf にインストールされます。

前提条件

Red Hat JBoss Data Grid の Maven リポジトリーを設定する必要があります。

手順4.3 Karaf への Hot Rod クライアント機能のインストール

1. **Karaf 2.3.3**
   Karaf 2.3.3 では次のコマンドを使用します。

   a.
   ```
   karaf@root> features:addUrl mvn:org.infinispan/infinispan-remote/${VERSION}/xml/features
   ```

   b.
   ```
   karaf@root> features:install infinispan-remote
   ```

   c. 機能が正しくインストールされたことを確認します。

   ```
   karaf@root> features:list
    //output
   ```

2. **Karaf 3.0.0**
   次のコマンドを使用します。

   a.
   ```
   karaf@root> feature:repo-add mvn:org.infinispan/infinispan-remote/${VERSION}/xml/features
   ```

   b.
   ```
   karaf@root> feature:install infinispan-remote
   ```

   c. 機能が正しくインストールされたことを確認します。

   ```
   karaf@root> feature:list
   ```

   この代わりに、**-i** コマンドパラメーターを以下のように使用して、Hot Rod クライアント機能をインストールすることもできます。

   ```
   karaf@root()> feature:repo-add -i mvn:org.infinispan/infinispan-remote/${VERSION}/xml/features
   ```

バグを報告する

4.6.3. Karaf での Red Hat JBoss Data Grid のインストール (ライブラリーモード)

Red Hat JBoss Data Grid の JAR ファイルには必要な OSGi マニフェストヘッダーが含まれ、これらのヘッダーは OSGi ランタイム環境内で OSGi バンドルとして使用されます。また、必要なサードパーティーの依存関係をインストールする必要があります。これらの依存関係は、個別にインストールできますが、必要な依存関係をすべて定義する **features** ファイルを用いて同時にインストールする

こともできます。

features ファイルを使用してバンドルをインストールするには、以下を実行します。

- Karaf 内の feature リポジトリーを登録します。
- リポジトリーに含まれる機能をインストールします。

手順4.4 features ファイルを使用したバンドルのインストール

1. **Karaf コンソールの起動**
   以下のコマンドを使用して Karaf コンソールを起動します。

```
$ cd $APACHE_KARAF_HOME/bin
$ ./karaf
```

2. **feature リポジトリーの登録**
   次のように feature リポジトリーを登録します。
   - Karaf 2.3.3 の場合

```
karaf@root()> features:addUrl mvn:org.infinispan/infinispan-embedded/${VERSION}/xml/features
```

```
karaf@root> features:install infinispan-embedded
```

   - Karaf 3.0.0 の場合

```
karaf@root()> feature:repo-add mvn:org.infinispan/infinispan-embedded/${VERSION}/xml/features
```

```
karaf@root> feature:install infinispan-embedded
```

結果

JBoss Data Grid が Karaf を使用してライブラリーモードで実行されます。

feature リポジトリーの URL は、以下の形式を使用して Maven アーティファクト座標から構成されます。

```
mvn:<groupId>/<artifactId>/<version>/xml/features
```

重要

JPA キャッシュストアは、JBoss Data Grid 7.0 の Apache Karaf でサポートされません。

重要

ライブラリーモードでのクエリー (JBoss Data Grid 『Developer Guide』 で取り上げています) は JBoss Data Grid 7.0 の Apache Karaf ではサポートされません。

バグを報告する

第5章 APACHE CAMEL を用いた RED HAT JBOSS DATA GRID の実行

Apache Camel はオープンソースの統合およびルーティングシステムです。さまざまなソースから異なる宛先へメッセージを送信できるようにし、プロトコルやデータタイプに関わらず、同じ API を使用してさまざまなシステムと対話できる統合フレームワークを提供します。Red Hat JBoss Data Grid および Red Hat JBoss Fuse に Camel を使用すると、接続性を追加するさまざまなトランスポートや API が提供され、大型のエンタープライズアプリケーションの統合が簡素化されます。

JBoss Data Grid は、特に Ehcache の置き換えで JBoss Fuse の Camel ルートにおけるキャッシングをサポートします。JBoss Data Grid は、埋め込みキャッシュ (ローカルまたはクラスター化) または Camel ルートのリモートキャッシュとしてサポートされます。

バグを報告する

5.1. CAMEL-JBOSSDATAGRID コンポーネント

Red Hat JBoss Data Grid の **camel-jbossdatagrid** コンポーネントには、以下の機能が含まれています。

- **ローカル Camel コンシューマー**

  キャッシュ変更通知を受け取り、処理のために送信します。同期または非同期で実行でき、レプリケートされたキャッシュまたは分散キャッシュでもサポートされます。

- **ローカル Camel プロデューサー**

  プロデューサーはメッセージを作成し、エンドポイントへ送信します。**camel-jbossdatagrid** プロデューサーは **GET**、**PUT**、**REMOVE**、および **CLEAR** 操作を使用します。ローカルプロデューサーはレプリケートされたキャッシュまたは分散キャッシュでもサポートされます。

- **リモート Camel プロデューサー**

  リモートクライアントサーバーモードでは、Camel プロデューサーは Hot Rod を使用してメッセージを送信できます。

- **リモート Camel コンシューマー**

  クライアントサーバーモードでは、キャッシュ変更通知を受け取り、それを処理のために送信します。イベントは非同期で処理されます。

Camel を用いて JBoss Data Grid を実行するには、以下の **camel-jbossdatagrid** 依存関係を **pom.xml** ファイルに追加する必要があります。

```
<dependency>
    <groupId>org.apache.camel</groupId>
    <artifactId>camel-jbossdatagrid</artifactId>
    <version>7.0.0.ER4-redhat-1</version>
    <!-- use the same version as your JBoss Data Grid version -->
</dependency>
```

注記

camel-jbossdatagrid コンポーネントは、JBoss Data Grid に同梱されていますが、JBoss Fuse 6.1 や JBoss Fuse Service Works 6.0 ディストリビューションには含まれていません。

Camel コンポーネントは Camel の主な拡張ポイントです。URI で使用される名前に関連付けられ、エンドポイントのファクトリーとして動作します。たとえば、**FileComponent** は URI では**file** と呼ばれ、**FileEndpoints** を作成します。

URI の形式

camel-jbossdatagrid には以下の URI 形式が使用されます。

```
infinispan://hostname?[options]
```

URI オプション

プロデューサーはメッセージを作成し、レジストリーで設定されたローカルまたはリモートの JBoss Data Grid キャッシュへメッセージを送信できます。**cacheContainer** が存在する場合、**cacheContainer** インスタンスが**DefaultCacheManager** または**RemoteCacheManager** であるかによってキャッシュがローカルまたはリモートになります。存在しない場合は、指定のホスト名やポートを使用してリモートキャッシュへの接続を試みます。

コンシューマーは、レジストリーからアクセスできるローカルの JBoss Data Grid キャッシュからイベントをリッスンします。

表5.1 URI オプション

Name	デフォルト値	タイプ	コンテキスト	説明
cacheContainer	null	CacheContainer	共有	レジストリーの **org.infinispan.manager.CacheContainer** への参照。
cacheName	null	文字列	共有	使用するキャッシュ名。指定されていないと、デフォルトのキャッシュが使用されます。
command	PUT	文字列	プロデューサー	実行する操作。現在 PUT、GET、REMOVE、および CLEAR のみが値としてサポートされています。

Name	デフォルト値	タイプ	コンテキスト	説明
eventTypes	null	Set<String>	コンシューマー	登録するイベントタイプのコンマ区切りのリスト。デフォルトでは、すべてのイベントタイプをリッスンします。可能な値は **org.infinispan.notifications.cachelistener.event.Event.Type** で定義されます。 例: `...?eventTypes=CACHE_ENTRY_EXPIRED,CACHE_ENTRY_EVICTED,...`
sync	true	ブール値	コンシューマー	デフォルトでは、コンシューマーはキャッシュ操作を処理する同じスレッドによって同期的に通知を受け取ります。リモート HotRod リスナーは非同期の通知のみをサポートします。

Name	デフォルト値	タイプ	コンテキスト	説明
clustered	false	ブール値	コンシューマー	デフォルトでは、コンシューマーはローカルイベントのみを受信します。このオプションを使用すると、コンシューマーはクラスターの他のノードからのイベントもリッスンします。クラスター化されたリスナーに利用できるイベントは **CACHE_ENTRY_CREATED**、**CACHE_ENTRY_REMOVED**、および **CACHE_ENTRY_MODIFIED** のみです。

Camel の操作

使用できる操作とヘッダー情報のリストは次のとおりです。

表5.2 Put 操作

操作名	コンテキスト	説明	必要なヘッダー	任意のヘッダー	結果のヘッダー
CamelInfinispanOperationPut	埋め込み/リモート	キーと値のペアをキャッシュに追加し、オプションでエクスパレーションを指定します。	**CamelInfinispanKey**、**CamelInfinispanValue**	**CamelInfinispanLifespanTime**、**CamelInfinispanLifespanTimeUnit**、**CamelInfinispanMaxIdleTime**、**CamelInfinispanMaxIdleTimeUnit**、**CamelInfinispanIgnoreReturnValues**	**CamelInfinispanOperationResult**
CamelInfinispanOperationPutAsync		非同期的にキーと値のペアをキャッシュに追加し、オプションでエクスパレーションを指定します。			

操作名	コンテキスト	説明	必要なヘッダー	任意のヘッダー	結果のヘッダー
CamelInfinispanOperationPutIfAbsent		存在しない場合にキーと値のペアをキャッシュに追加し、オプションでエクスパレーションを指定します。			
CamelInfinispanOperationPutIfAbsentAsync		存在しない場合にキーと値のペアを非同期的にキャッシュへ追加し、オプションでエクスパレーションを指定します。			

表5.3 Put All 操作

操作名	コンテキスト	説明	必要なヘッダー	任意のヘッダー	結果のヘッダー
CamelInfinispanOperationPutAll	埋め込み/リモート	複数のエントリーをキャッシュに追加します (オプションでエクスパレーションを指定します)。	**CamelInfinispanMap**	**CamelInfinispanLifespanTime**、**CamelInfinispanLifespanTimeUnit**、**CamelInfinispanMaxIdleTime**、**CamelInfinispanMaxIdleTimeUnit**	
CamelInfinispanOperationPutAllAsync		非同期的に複数のエントリーをキャッシュに追加します (オプションでエクスパレーションを指定します)。			

表5.4 Get 操作

操作名	コンテキスト	説明	必要なヘッダー	任意のヘッダー	結果のヘッダー
CamelInfinispanOperationGet	埋め込み/リモート	特定のキーに関連する値をキャッシュから取得します。	**CamelInfinispanKey**		

表5.5 Contains Key 操作

操作名	コンテキスト	説明	必要なヘッダー	任意のヘッダー	結果のヘッダー
CamelInfinispanOperationContainsKey	埋め込み/リモート	キャッシュに特定のキーが含まれるかどうかを判断します。	**CamelInfinispanKey**		**CamelInfinispanOperationResult**

表5.6 Contains Value 操作

操作名	コンテキスト	説明	必要なヘッダー	任意のヘッダー	結果のヘッダー
CamelInfinispanOperationContainsValue	埋め込み/リモート	キャッシュに特定の値が含まれるかどうかを判断します。	**CamelInfinispanKey**		

表5.7 Remove 操作

操作名	コンテキスト	説明	必要なヘッダー	任意のヘッダー	結果のヘッダー
CamelInfinispanOperationRemove	埋め込み/リモート	キャッシュからエントリーを削除します (オプションで、指定の値と一致した場合のみ削除します)。	**CamelInfinispanKey**	**CamelInfinispanValue**	**CamelInfinispanOperationResult**

操作名	コンテキスト	説明	必要なヘッダー	任意のヘッダー	結果のヘッダー
CamelInfinispanOperationRemoveAsync		非同期的にキャッシュからエントリーを削除します (オプションで、指定の値と一致した場合のみ削除します)。			

表5.8 Replace 操作

操作名	コンテキスト	説明	必要なヘッダー	任意のヘッダー	結果のヘッダー
CamelInfinispanOperationReplace	埋め込み/リモート	キャッシュのエントリーを条件的に置き換えます (オプションでエクスパレーションを指定します)。	**CamelInfinispanKey**、**CamelInfinispanValue**、**CamelInfinispanOldValue**	**CamelInfinispanLifespanTime**、**CamelInfinispanLifespanTimeUnit**、**CamelInfinispanMaxIdleTime**、**CamelInfinispanMaxIdleTimeUnit**、**CamelInfinispanIgnoreReturnValues**	**CamelInfinispanOperationResult**
CamelInfinispanOperationReplaceAsync		キャッシュのエントリーを非同期的かつ条件的に置き換えます (オプションでエクスパレーションを指定します)。			

表5.9 Clear 操作

操作名	コンテキスト	説明	必要なヘッダー	任意のヘッダー	結果のヘッダー
CamelInfinispanOperationClear	埋め込み/リモート	キャッシュを消去します。			

表5.10 Size 操作

操作名	コンテキスト	説明	必要なヘッダー	任意のヘッダー	結果のヘッダー
CamelInfinispanOperationSize	埋め込み/リモート	キャッシュにあるエントリーの数を返します。			**CamelInfinispanOperationResult**

表5.11 Query 操作

操作名	コンテキスト	説明	必要なヘッダー	任意のヘッダー	結果のヘッダー
CamelInfinispanOperationQuery	リモート	キャッシュでクエリーを実行します。	**CamelInfinispanQueryBuilder**		**CamelInfinispanOperationResult**

注記

CamelInfinispanIgnoreReturnValues を取る操作はすべて null を結果として受け取ります。

表5.12 メッセージヘッダー

Name	デフォルト値	タイプ	コンテキスト	説明
CamelInfinispanCacheName	null	文字列	共有	操作またはイベントに参加しているキャッシュ。
CamelInfinispanMap	null	マップ	プロデューサー	CamelInfinispanOperationPutAll 操作の場合に使用されるマップ。
CamelInfinispanKey	null	オブジェクト	共有	操作を実行するキーまたはイベントを生成するキー。
CamelInfinispanValue	null	オブジェクト	プロデューサー	操作に使用する値。
CamelInfinispanOperationResult	null	オブジェクト	プロデューサー	操作の結果。

Name	デフォルト値	タイプ	コンテキスト	説明
CamelInfinispanEventType	null	文字列	コンシューマー	ローカルキャッシュリスナー (非クラスター) の場合は次の値の 1 つ: **CACHE_ENTRY_ACTIVATED**、**CACHE_ENTRY_PASSIVATED**、**CACHE_ENTRY_VISITED**、**CACHE_ENTRY_LOADED**、**CACHE_ENTRY_EVICTED**、**CACHE_ENTRY_CREATED**、**CACHE_ENTRY_REMOVED**、**CACHE_ENTRY_MODIFIED** リモート HotRod リスナーの場合は次の値の 1 つ: **CLIENT_CACHE_ENTRY_CREATED**、**CLIENT_CACHE_ENTRY_MODIFIED**、**CLIENT_CACHE_ENTRY_REMOVED**、**CLIENT_CACHE_FAILOVER**
CamelInfinispanIsPre	null	ブール値	コンシューマー	ローカルの非クラスター化リスナーが使用されると、Infinispan は各操作に対して操作の前と後に 1 つずつ 2 つのイベントを発火します。クラスター化リスナーおよびリモート HotRod リスナーの場合は、Infinispan は操作の後に 1 つのイベントのみを発火します。

Name	デフォルト値	タイプ	コンテキスト	説明
CamelInfinispanQueryBuilder	null	InfinispanQueryBuilder	プロデューサー	**InfinispanQueryBuilder** のインスタンスで、キャッシュで実行されるクエリーを **build()** 内で定義します。
CamelInfinispanLifespanTime	null	long	プロデューサー	キャッシュ内の値のライフスパン時間。負の値は無限として解釈されます。
CamelInfinispanTimeUnit	null	文字列	プロデューサー	エントリーのライフスパン時間の時間単位。
CamelInfinispanMaxIdleTime	null	long	プロデューサー	エントリーがアイドル状態でいられる最大期間。この期間以降は期限切れと見なされます。
CamelInfinispanMaxIdleTimeUnit	null	文字列	プロデューサー	エントリーの最大アイドル時間の時間単位。

バグを報告する

5.2. JBOSS DATA GRID での CAMEL を用いたルーティング

Camel のルーティングは、バックグラウンドでメッセージを移動するプロセスチェーンです。以下は、特定のキーのキャッシュから値を読み出すルートの例になります。

```
from("direct:start")
    .setHeader(InfinispanConstants.OPERATION,
constant(InfinispanConstants.GET))
    .setHeader(InfinispanConstants.KEY, constant("123"))
    .to("infinispan://localhost?cacheContainer=#cacheContainer");
```

XML 設定を使用してルーティングを実行することもできます。以下の例は、 camel-jbossdatagrid の **local-camel-producer** を示しています。これは、**camel-jbossdatagrid** コンポーネントを使用して、**local-cache** モジュールによって作成された埋め込みキャッシュにデータを送信する Camel ルートです。

```
<camelContext id="local-producer"
xmlns="http://camel.apache.org/schema/blueprint">
    <route>
```

```
        <from uri="timer://local?fixedRate=true&amp;period=5000"/>
        <setHeader headerName="CamelInfinispanKey">
            <constant>CamelTimerCounter</constant>
        </setHeader>
        <setHeader headerName="CamelInfinispanValue">
            <constant>CamelTimerCounter</constant>
        </setHeader>
        <to uri="infinispan://foo?cacheContainer=#cacheManager"/>
        <to uri="log:local-put?showAll=true"/>
    </route>
</camelContext>
```

この例では、**cacheManager** がインスタンス化される必要があります。

Spring XML の **cacheManager** Bean は以下のようにインスタンス化できます。

```
<bean id="cacheManager" class="org.infinispan.manager.DefaultCacheManager"
init-method="start" destroy-method="stop">
    <constructor-arg type="java.lang.String" value="infinispan.xml"/>
</bean>
```

以下は、Blueprint XML を使用して **cacheManager** Bean をインスタンス化する方法を示しています。

```
<bean id="cacheManager" class="org.infinispan.manager.DefaultCacheManager"
init-method="start" destroy-method="stop">
    <argument value="infinispan.xml" />
</bean>
```

注記

Spring XML および Blueprint XML の例は、キャッシュの設定に設定ファイル **infinispan.xml** を使用します。このファイルはクラスパスに存在する必要があります。

バグを報告する

5.3. リモートクエリー

リモートクエリーを実行するとき、cacheManager は **RemoteCacheManager** のインスタンスである必要があります。Java および blueprint.xml で **RemoteCacheManager** を使用する設定例を以下に示します。

Java のみを使用

```
from("direct:start")
    .setHeader(InfinispanConstants.OPERATION, InfinispanConstants.QUERY)
    .setHeader(InfinispanConstants.QUERY_BUILDER,
      new InfinispanQueryBuilder() {
        public Query build(QueryFactory<Query> queryFactory) {
          return
queryFactory.from(User.class).having("name").like("%abc%")
                      .toBuilder().build();
```

```
        }
      })
    .to("infinispan://localhost?
cacheContainer=#cacheManager&cacheName=remote_query_cache") ;
```

Blueprint および Java を使用

Java **RemoteCacheManagerFactory** クラス:

```
public class RemoteCacheManagerFactory {
    ConfigurationBuilder clientBuilder;
    public RemoteCacheManagerFactory(String hostname, int port) {
        clientBuilder = new ConfigurationBuilder();
        clientBuilder.addServer()
            .host(hostname).port(port);
    }
    public RemoteCacheManager newRemoteCacheManager() {
        return new RemoteCacheManager(clientBuilder.build());
    }
}
```

Java **InfinispanQueryExample** クラス:

```
public class InfinispanQueryExample {
    public InfinispanQueryBuilder getBuilder() {
        return new InfinispanQueryBuilder() {
            public Query build(QueryFactory<Query> queryFactory) {
                return queryFactory.from(User.class)
                        .having("name")
                        .like("%abc%")
                        .toBuilder().build();
            }
        }
    }
}
```

blueprint.xml:

```
<bean id=”remoteCacheManagerFactory”
class=“com.jboss.datagrid.RemoteCacheManagerFactory”>
    <argument value=”localhost”/>
    <argument value="11222”/>
</bean>

<bean id=”cacheManager”
    factory-ref=”remoteCacheManagerFactory”
    factory-method=“newRemoteCacheManager”>
</bean>

<bean id="queryBuilder" class="org.example.com.InfinispanQueryExample"/>

<camelContext id="route" xmlns="http://camel.apache.org/schema/blueprint">
    <route>
        <from uri="direct:start"/>
            <setHeader headerName="CamelInfinispanOperation">
                <constant>CamelInfinispanOperationQuery</constant>
```

```
            </setHeader>
            <setHeader headerName="CamelInfinispanQueryBuilder">
                <method ref="queryBuilder" method="getBuilder"/>
            </setHeader>
        <to uri="infinispan://localhost?
cacheContainer=#cacheManager&cacheName=remote_query_cache"/>
    </route>
</camelContext>
```

remote_query_cache は、データを保持するキャッシュの任意名で、クエリーの結果は **CamelInfinispanOperationResult** ヘッダーとして保存されるドメインオブジェクトのリストになります。

さらに、要件を以下に示します。

- **RemoteCacheManager** は **ProtoStreamMarshaller** を使用するよう設定される必要があります。
- **ProtoStreamMarshaller** は **RemoteCacheManager** のシリアライズコンテキストで登録される必要があります。
- ドメインオブジェクトの .proto 記述子はリモート JBoss Data Grid サーバーで登録される必要があります。

RemoteCacheManager の設定方法の詳細は、『Red Hat JBoss Data Grid Infinispan Query Guide』 の **Remote Querying** の項を参照してください。

バグを報告する

5.4. 埋め込みキャッシュのカスタムリスナー

埋め込みキャッシュのカスタムリスナーは、以下のように ***customListener*** パラメーターを使用して登録できます。

Java を使用

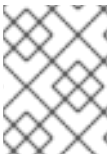
```
from("infinispan://?
cacheContainer=#myCustomContainer&cacheName=customCacheName&customListener
=#myCustomListener")
  .to("mock:result");
```

Blueprint を使用

```
<bean id="myCustomContainer" org.infinispan.manager.DefaultCacheManager"
      init-method="start" destroy-method="stop">
      <argument value="infinispan.xml" />
</bean>

<bean id="myCustomListener" class="org.example.com.CustomListener"/>

<camelContext id="route" xmlns="http://camel.apache.org/schema/blueprint">
    <route>
        <from uri="infinispan://?
cacheContainer=#myCustomContainer&cacheName=customCacheName&customListener
=#myCustomListener"/>
```

```
        <to uri="mock:result"/>
    </route>
</camelContext>
```

myCustomListener のインスタンスが存在する必要があります。ユーザーは **org.apache.camel.component.infinispan.embedded.InfinispanEmbeddedCustomListener** を拡張し、結果となるクラスにorg.infinispan.notifications にある **@Listener** アノテーションを付けることが推奨されます。



注記

埋め込みキャッシュのカスタムフィルターおよびコンバーターは現在サポートされていません。

バグを報告する

5.5. リモートキャッシュのカスタムリスナー

リモートキャッシュのカスタムリスナーは、埋め込みキャッシュと同様に登録できますが、***sync=false*** が存在する必要があります。例を以下に示します。

Java のみを使用

```
from(infinispan://?
cacheContainer=#cacheManager&sync=false&customListener=#myCustomListener")
  .to(mock:result);
```

Blueprint および Java を使用

Java クラス:

```
public class RemoteCacheManagerFactory {
    ConfigurationBuilder clientBuilder;
    public RemoteCacheManagerFactory(String hostname, int port) {
        clientBuilder = new ConfigurationBuilder();
        clientBuilder.addServer()
            .host(hostname).port(port);
    }
    public RemoteCacheManager newRemoteCacheManager() {
        return new RemoteCacheManager(clientBuilder.build());
    }
}
```

blueprint.xml:

```
<bean id=”remoteCacheManagerFactory”
class=“com.jboss.datagrid.RemoteCacheManagerFactory”>
    <argument value=”localhost”/>
    <argument value="11222”/>
</bean>

<bean id=”cacheManager”
    factory-ref=”remoteCacheManagerFactory”
    factory-method=“newRemoteCacheManager”>
```

```
</bean>

<bean id="myCustomListener" class="org.example.com.CustomListener"/>

<camelContext id="route" xmlns="http://camel.apache.org/schema/blueprint">
    <route>
        <from uri="infinispan://?
cacheContainer=#cacheManager&sync=false&customListener=#myCustomListener"/
>
        <to uri="mock:result"/>
    </route>
</camelContext>
```

myCustomListener のインスタンスが存在する必要があります。ユーザーは **org.apache.camel.component.infinispan.remote.InfinispanRemoteCustomListener** クラスを拡張し、結果となるクラスに **@ClientListener** アノテーションを付けることが推奨されます。このアノテーションは org.infinispan.client.hotrod.annotation にあります。

リモートリスナーも以下のようにカスタムフィルターおよびコンバーターと関連付けられることがあります。

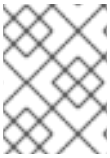
```
@ClientListener(includeCurrentState=true, filterFactoryName = "static-
filter-factory", converterFactoryName = "static-converter-factory")
  private static class MyCustomListener extends
InfinispanRemoteCustomListener {
}
```

カスタムフィルターまたはコンバーターを使用するには、**@NamedFactory** アノテーションが付けられたクラスを実装する必要があります。必要なメソッドを実装するスケルトンは以下のとおりです。

```
import org.infinispan.notifications.cachelistener.filter;

@NamedFactory(name = "static-converter-factory")
public static class StaticConverterFactory implements
CacheEventConverterFactory {
  @Override
  public CacheEventConverter<Integer, String, CustomEvent>
getConverter(Object[] params) {
    ...
  }

  static class StaticConverter implements CacheEventConverter<Integer,
String, CustomEvent>, Serializable {
    @Override
    public CustomEvent convert(Integer key, String previousValue, Metadata
previousMetadata,
                               String value, Metadata metadata, EventType
eventType) {
      ...
    }
  }
}

@NamedFactory(name = "static-filter-factory")
public static class StaticCacheEventFilterFactory implements
CacheEventFilterFactory {
```

```
  @Override
  public CacheEventFilter<Integer, String> getFilter(final Object[]
params) {
    ...
  }

  static class StaticCacheEventFilter implements CacheEventFilter<Integer,
String>, Serializable {
    @Override
    public boolean accept(Integer key, String previousValue, Metadata
previousMetadata,
                          String value, Metadata metadata, EventType
eventType) {
      ...
    }
  }
}
```

カスタムフィルターおよびコンバーターはサーバーに登録する必要があります。これらのクラスの登録については、『Red Hat JBoss Data Grid Developer Guide』 の **Remote Event Listeners** の項を参照してください。



注記

リモート HotRod イベントをリッスンするには、cacheManager のタイプが **RemoteCacheManager** で、インスタンス化される必要があります。

バグを報告する

5.6. RED HAT JBOSS DATA GRID および RED HAT JBOSS FUSE

5.6.1. Red Hat JBoss Fuse 向けの camel-jbossdatagrid のインストール

Red Hat JBoss Fuse は OSGi コンテナーベースの Karaf コンテナーです。 **camel-jbossdatagrid** を使用して Red Hat JBoss Data Grid と JBoss Fuse を実行するには、JBoss Data Grid 7.0 と JBoss Fuse 6.1 (フルインストール) の両方がインストールされている必要があります。

手順5.1 JBoss Data Grid のインストール

- JBoss Data Grid のインストールに関する情報は、パートII「Red Hat JBoss Data Grid のダウンロードおよびインストール」を参照してください。JBoss Fuse で Camel コンポーネントを実行するには、以下の JBoss Data Grid コンポーネントのみが必要になります。
  - JBoss Data Grid Maven リポジトリー。
  - The JBoss Data Grid Server パッケージ (Hot Rod クライアントを使用するため)。

camel-jbossdatagrid ライブラリーは、**jboss-datagrid-7.0.0-camel-library** という個別のディストリビューションから入手することもできます。

手順5.2 JBoss Fuse のインストール

前提条件

Red Hat JBoss Fuse をインストールする前に、ご使用のシステムが最低の条件を満たしていることを確認してください。サポートされるプラットフォームと推奨される Java ランタイムプラットフォームについては『Red Hat JBoss Fuse Installation Guide』を参照してください。

JBoss Fuse 6.1 フルインストールには以下のハードウェアが必要になります。

- 700 MB の空きディスク容量
- 2 GB の RAM

ベースインストールに必要なディスク容量以外に、システムの実行にはキャッシング、永続メッセージストア、およびその他の機能の容量が必要になります。

1. **JBoss Fuse フルインストールのダウンロード**
   Red Hat JBoss Fuse アーカイブは、Red Hat カスタマーポータルでアカウントを登録してログインした後に、**ダウンロード>Red Hat JBoss Fuse** と選択すると表示される Software Downloads のページからダウンロードできます。

   ログインしている状態で以下を行います。

   a. **Software Downloads** ページの **Product** ドロップダウンメニューで **Fuse** を選択します。

   b. **Software Downloads** ページの Version ドロップダウンリストで **6.1.0** を選択します。

   c. Red Hat JBoss Fuse 6.1.0 ディストリビューションファイルの横にある **Download** ボタンをクリックし、ダウンロードします。

   JBoss Fuse では、異なる機能セットが含まれるインストールを選択できます。JBoss Data Grid を JBoss Fuse と実行するには、フルインストールが必要になります。フルインストールには以下が含まれます。

   - Apache Karaf
   - Apache Camel
   - Apache ActiveAMQ
   - Apache CXF
   - Fuse 管理
   - コンソール (hawtio)
   - JBI コンポーネント

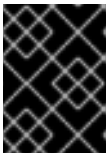
2. **アーカイブの展開**
   Red Hat JBoss Fuse は、アーカイブをシステムで展開するとインストールされます。JBoss Fuse は zip ファイルとしてパッケージ化されています。適切なアーカイブツールを使用して、Red Hat JBoss Fuse を完全なアクセス権限を持つディレクトリーで展開します。

警告

パス名に空白文字が含まれるフォルダーにアーカイブファイルを展開しないでください。たとえば、**C:\Documents and Settings\Greco Roman\Desktop\fusesrc.** で展開しないでください。

さらに、パス名に特殊文字 #、%、^、または " が含まれるフォルダーにアーカイブファイルを展開しないでください。

3. **リモートコンソールユーザーの追加**
サーバーのリモートコマンドコンソールはデフォルトユーザーには設定されていません。サーバーのコンソールへリモート接続する前に、ユーザーを設定に追加します。



重要

このファイルの情報は暗号化されていないため、厳重なセキュリティーを必要とする環境には適していません。

ユーザーを追加するには、以下を実行します。

a. テキストエディターで **InstallDir/etc/users.properties** を開きます。

b. 行 **#admin=admin,admin** を見つけます。この行は、パスワードが**admin** のユーザー **admin** と、ロール **admin** を指定します。

c. 先頭の # を削除し、行をアンコメントします。

d. 最初の **admin** をユーザーの名前に置き換えます。

e. 2 つ目の **admin** をユーザーのパスワードに置き換えます。

f. 最後の **admin** はそのままにし、変更を保存します。

注記

Fuse 管理コンソールにアクセスし、 Camel ルート、ActiveMQ ブローカー、Web アプリケーションなどを監視および管理するには、Red Hat JBoss Fuse の起動後にブラウザーで
http://localhost:8181/hawtio
にアクセスします。

4. **Red Hat JBoss Fuse Maven リポジトリー**
Maven を使用してプロジェクトをビルドするには、Maven **settings.xml** ファイルでアーティファクトの場所を指定します。

次の JBoss Fuse Maven リポジトリーには Camel に必要な依存関係が含まれ、**settings.xml** ファイルに追加する必要があります。

```
https://repo.fusesource.com/nexus/content/groups/public/
```

JBoss Fuse リポジトリーは JBoss Data Grid リポジトリーと平行して実行されます。

JBoss Data Grid には、**camel-jbossdatagrid** コンポーネントに必要なすべてのアーティファクトをデプロイする Karaf 用の **features.xml** ファイルが含まれています。このファイルは、JBoss Fuse コンテナーディストリビューションには含まれていません。**features.xml** ファイルは **jboss-datagrid-7.0.0-maven-repository/org/apache/camel/camel-jbossdatagrid/${version}/** にあります。JBoss Data Grid リポジトリーに必要な他の設定はありません。

JBoss Fuse のインストールや初めて使用する場合の詳細は、Red Hat カスタマーポータルの Red Hat JBoss Fuse ドキュメントを参照してください。

バグを報告する

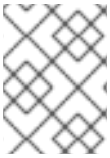
5.7. RED HAT JBOSS DATA GRID および RED HAT JBOSS EAP

5.7.1. Red Hat JBoss Enterprise Application Platform への camel-jbossdatagrid のインストール

Red Hat JBoss Enterprise Application Platform 6 (JBoss EAP 6) は、オープンな標準に基づいて構築され、Java Enterprise Edition 6 の仕様に準拠するミドルウェアプラットフォームです。

Camel は Red Hat JBoss Fuse を介してのみサポートされます。以下の製品すべての有効なエンタイトルメントが必要になります。

- Red Hat JBoss EAP
- Red Hat JBoss Fuse
- Red Hat JBoss Data Grid



注記

Red Hat JBoss Fuse Service Works のエンタイトルメントには、Red Hat JBoss EAP および Red Hat JBoss Fuse のエンタイトルメントが含まれます。

インストール手順では、特定のバージョン番号の代わりに以下の変数が使用されます。

- **jdg.version** - Red Hat JBoss Data Grid の最新バージョン

  fuse.version - Red Hat JBoss Fuse の最新バージョン

- **infinispan.version** - 最新バージョンの Red Hat JBoss Data Grid に含まれる Infinispan のバージョン。
- **camel.version** - 最新バージョンの Red Hat JBoss Fuse に含まれる Apache Camel のバージョン。

camel-jbossdatagrid のテスト済みの構成に関する詳細は https://access.redhat.com/articles/115883 を参照してください。

手順5.3 JBoss Data Grid のインストール

- JBoss Data Grid のインストールに関する情報は、パートII「Red Hat JBoss Data Grid のダウンロードおよびインストール」を参照してください。JBoss EAP で Camel コンポーネントを実行するには、以下の JBoss Data Grid コンポーネントのみが必要になります。
  - JBoss Data Grid Maven リポジトリー。
  - The JBoss Data Grid Server パッケージ (Hot Rod を使用するため)。

camel-jbossdatagrid ライブラリーは、**jboss-datagrid-${jdg.version}-camel-library** という個別のディストリビューションから入手することもできます。

手順5.4 JBoss EAP のインストール

1. Red Hat JBoss EAP をインストールして使用する前に、ご使用のシステムが 『Red Hat JBoss EAP インストールガイド』 に記載されている最低限の要件を満たしていることを確認してください。
2. **アーカイブの展開**
   Red Hat JBoss EAP は zip ファイルとしてパッケージ化されているため、アーカイブをシステムで展開するとインストールされます。適切なアーカイブツールを使用して、Red Hat JBoss EAP を完全なアクセス権限を持つディレクトリーで展開します。

警告

パス名に空白文字が含まれるフォルダーにアーカイブファイルを展開しないでください。たとえば、**C:\Documents and Settings\Greco Roman\Desktop\JBoss** では展開しないでください。

さらに、パス名に特殊文字 #、%、^、または " が含まれるフォルダーにアーカイブファイルを展開しないでください。

3. アーカイブが展開されると、JBoss EAP が正常にインストールされます。インストールオプションの詳細は 『Red Hat JBoss EAP インストールガイド』を参照してください。
4. JBoss Data Grid がライブラリーモードで使用されている場合は 「JBoss EAP での JBoss Data Grid のデプロイ (ライブラリーモード)」 を参照し、必要な依存関係がインストールされていることを確認してください。
5. JBoss Data Grid がリモートクライアントサーバーモードで使用されている場合は 「JBoss EAP での JBoss Data Grid のデプロイ (リモートクライアントサーバーモード)」 を参照し、必要な依存関係がインストールされていることを確認してください。

バグを報告する

5.7.2. EAP を用いた Camel のデプロイ

5.7.2.1. 開発およびランタイム依存関係の追加

アプリケーションをコンパイルするには、Camel および JBoss Data Grid の依存ライブラリーを **pom.xml** に追加する必要があります (Maven を使用する場合)。

手順5.5 Fuse からの Camel の追加

1. Fuse リポジトリーが **pom.xml** に追加されていることを確認します。

```
<repository>
  <id>fusesource</id>
  <name>FuseSource Release Repository</name>

<url>https://repo.fusesource.com/nexus/content/groups/public/</url>
  <snapshots>
    <enabled>false</enabled>
  </snapshots>
  <releases>
    <enabled>true</enabled>
  </releases>
</repository>
```

2. **pom.xml** で Camel コンポーネントを依存関係として追加します。

```
<dependency>
  <groupId>org.apache.camel</groupId>
  <artifactId>camel-core</artifactId>
  <version>${camel.version}</version>
</dependency>
```

手順5.6 camel-jbossdatagrid のデプロイメントへの追加

1. 3章*Maven リポジトリーのインストールおよび使用*の手順に従って、maven リポジトリーを追加します。

2. **pom.xml** で camel-jbossdatagrid を依存関係として追加します。

```
<dependency>
  <groupId>org.apache.camel</groupId>
  <artifactId>camel-jbossdatagrid</artifactId>
  <version>${jdg.version}</version>
</dependency>
```

3. 使用している機能に応じて、残りの JBoss Data Grid の依存関係を追加します。

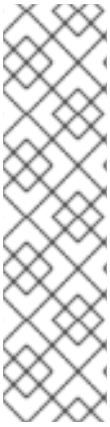
```
<!-- If Remote Camel Producer is used add the following dependency -
->
<dependency>
 <groupId>org.infinispan</groupId>
 <artifactId>infinispan-remote</artifactId>
 <version>${infinispan.version}</version>
</dependency>

<!-- If the Local Camel Producer or Local Camel Consumer are in use
add -->
<!-- the following dependency -->
<dependency>
 <groupId>org.infinispan</groupId>
```

```
 <artifactId>infinispan-embedded</artifactId>
 <version>${infinispan.version}</version>
</dependency>
```

バグを報告する

5.7.2.2. 任意設定: ランタイム依存関係を JBoss EAP モジュールとして追加

場合によっては JBoss EAP の他の製品ライブラリーをモジュールとして維持したいことがあります。これらのモジュールの作成には追加の手順が必要になります。



注記

依存関係のモジュールを使用する場合、**pom.xml** にてモジュールとして提供される依存関係の **scope** を provided に設定する必要があります。例を以下に示します。

```
<dependency>
 <groupId>org.apache.camel</groupId>
 <artifactId>camel-jbossdatagrid</artifactId>
 <scope>provided</scope>
 <version>${jdg.version}</version>
</dependency>
```

jboss-datagrid-${jdg.version}-camel-library.zip をカスタマーポータルからダウンロードします。

Red Hat JBoss Fuse の **extras/** ディレクトリーにある **apache-camel-${camel.version}.zip** を展開します。

```
user@example modules] unzip /path/to/jboss-fuse-
${fuse.version}/extras/apache-camel-${camel.version}
```

手順5.7 JBoss Fuse からの Camel コンポーネントの追加

1. $EAP_HOME/modules 以下にディレクトリーを作成します。

   ```
   user@example jboss-eap-6.4] cd modules
   user@example modules] mkdir -p org/apache/camel/core
   ```

2. jar を格納する **main** サブディレクトリーを作成します。

   ```
   user@example modules] mkdir org/apache/camel/core/main
   ```

3. camel-core jar を **apache-camel-${camel.version}.zip** から新たに作成された **main** ディレクトリーへコピーします。

   ```
   user@example modules] cp /path/to/jboss/fuse/extras/apache-camel-
   ${camel.version}/lib/camel-core-${camel.version}.jar
   ./org/apache/camel/core/main/
   ```

4. 以下のテキストを **org/apache/camel/core/main/module.xml** に追加し、**module.xml** 記述子を作成します。

```
<?xml version="1.0" encoding="UTF-8"?>
<module xmlns="urn:jboss:module:1.1" name="org.apache.camel.core">
  <resources>
    <resource-root path="camel-core-${camel.version}.jar"/>
  </resources>
</module>
```

5. 上記の手順を繰り返し、使用中の各依存関係に対してモジュールを作成します。『Red Hat JBoss 管理および設定ガイド』に記載されているとおり、モジュールによっては他のモジュールとの依存関係がある場合があります。

手順5.8 JBoss Data Grid からの Camel コンポーネントの追加

1. JDG Camel コンポーネントの **main** サブディレクトリーを作成します。

```
user@example jboss-eap-6.4] mkdir -p modules/org/apache/camel/main
```

2. **jboss-datagrid-${jdg.version}-camel-library.zip** を展開します。
3. **camel-jbossdatagrid-${jdg.version}.jar** を新たに作成されたディレクトリーにコピーします。

```
user@example jboss-eap-6.4] cp jboss-datagrid-${jdg.version}-camel-
library/camel-jbossdatagrid-${jdg.version}.jar
modules/org/apache/camel/main/
```

4. 以下のテキストを **org/apache/camel/main/module.xml** に追加し、**module.xml** 記述子を作成します。

```
<?xml version="1.0" encoding="UTF-8"?>
<module xmlns="urn:jboss:module:1.1" name="org.apache.camel">
  <resources>
    <resource-root path="camel-jbossdatagrid-${jdg.version}.jar"/>
  </resources>
  <dependencies>
    <module name="org.apache.camel.core" />
  </dependencies>
</module>
```

war の **WEB-INF** に **jboss-deployment-structure.xml** を作成し、新たに作成されたモジュールに依存関係を追加します。

```
<?xml version="1.0" encoding="UTF-8"?>
<jboss-deployment-structure>
    <deployment>
        <dependencies>
            <module name="org.apache.camel" meta-inf="import"/>

            <!-- Add the following lines if Library mode is used -->
            <module name="org.infinispan" slot="jdg-7.0" />
            <module name="org.jgroups" slot="jdg-7.0" />

            <!-- Add the following lines if Remote Client-Server mode is
used -->
```

```
            <module name="org.infinispan.client.hotrod" slot="jdg-7.0" />
        </dependencies>
    </deployment>
</jboss-deployment-structure>
```

バグを報告する

パート IV. RED HAT JBOSS DATA GRID の実行

第6章 MAVEN を用いた RED HAT JBOSS DATA GRID JAR ファイルの実行

6.1. JBOSS DATA GRID の実行 (リモートクライアントサーバーモード)

以下の手順に従って、Maven を用いて Red Hat JBoss Data Grid JAR ファイルリモートクライアントサーバーモードで実行します。

クエリーを用いる Hot Rod クライアント

以下の依存関係を **pom.xml** ファイルに追加します。

1. **infinispan-remote** 依存関係を追加します。

```
<dependency>
   <groupId>org.infinispan</groupId>
   <artifactId>infinispan-remote</artifactId>
   <version>${infinispan.version}</version>
</dependency>
```

2. **Remote Cache Store** が使用されているインスタンスの場合も以下のように**infinispan-embedded** 依存関係を追加します。

```
<dependency>
   <groupId>org.infinispan</groupId>
   <artifactId>infinispan-embedded</artifactId>
   <version>${infinispan.version}</version>
</dependency>
```

3. **JSR-107** が使用されているインスタンスの場合、必ず起動時に**cache-api** パッケージが使用できるようにしてください。以下のいずれかの方法で、これらのパッケージを利用可能にできます。

    1. **オプション 1:** Boss EAP が使用されている場合、「JBoss EAP での JBoss Data Grid のデプロイ (リモートクライアントサーバーモード)」の説明どおりに JBoss Data Grid モジュールをこのインスタンスに追加します。

       javax.cache.api モジュールをアプリケーションの**jboss-deployment-structure.xml** に追加します。例を以下に示します。

```
<jboss-deployment-structure xmlns="urn:jboss:deployment-
structure:1.2">
    <deployment>
        <dependencies>
            <module name="javax.cache.api" slot="jdg-7.0"
services="export"/>
        </dependencies>
    </deployment>
</jboss-deployment-structure>
```

    2. **オプション 2:** カスタマーポータルから**jboss-datagrid-${jdg.version}-library** ファイルをダウンロードします。

ダウンロードしたアーカイブを展開します。

jboss-datagrid-${jdg.version}-library/lib/cache-api-${jcache.version}.jar ファイルを希望のアプリケーションに埋め込みます。

3. **オプション 3:** JBoss Data Grid の Maven リポジトリーを使用できる場合は、以下のようにプロジェクトの **pom.xml** に明示的な依存関係を追加します。

```
<dependency>
    <groupId>javax.cache</groupId>
    <artifactId>cache-api</artifactId>
    <version>${jcache.version}</version>
</dependency>
```

警告

Infinispan クエリー API は、直接 Hibernate Search と Lucene API を公開し、**infinispan-embedded-query.jar** ファイル内に埋め込みできません。他のバージョンの Hibernate Search と Lucene が **infinispan-embedded-query** として同じデプロイメントに含まれないようにしてください。含まれると、クラスパスの競合が発生する原因となり、予期せぬ動作が実行されます。

バグを報告する

6.2. JBOSS DATA GRID の実行 (ライブラリーモード)

以下の手順に従って、Maven を用いて Red Hat JBoss Data Grid をライブラリーモードで実行します。

注記

Red Hat JBoss Data Grid を直接アプリケーションに埋め込むことを簡素化するため、JBoss Data Grid には少数の統合された jar が含まれています。サポートされる jar ファイルの一覧は、『Release Notes』の Packaging Revisions を参照してください。

クエリーを用いずに埋め込まれた Infinispan

以下の依存関係を **pom.xml** ファイルに追加します。

1. **infinispan-embedded** 依存関係を追加します。

```
<dependency>
   <groupId>org.infinispan</groupId>
   <artifactId>infinispan-embedded</artifactId>
   <version>${infinispan.version}</version>
</dependency>
```

クエリーを用いて埋め込まれた Infinispan

以下の依存関係を **pom.xml** ファイルに追加します。

1. **infinispan-embedded-query** 依存関係を追加します。

```
<dependency>
   <groupId>org.infinispan</groupId>
   <artifactId>infinispan-embedded-query</artifactId>
   <version>${infinispan.version}</version>
</dependency>
```

警告

Infinispan クエリー API は、直接 Hibernate Search と Lucene API を公開し、**infinispan-embedded-query.jar** ファイル内に埋め込みできません。他のバージョンの Hibernate Search と Lucene が **infinispan-embedded-query** として同じデプロイメントに含まれないようにしてください。含まれると、クラスパスの競合が発生する原因となり、予期せぬ動作が実行されます。

バグを報告する

第7章 RED HAT JBOSS DATA GRID のリモートクライアントサーバーモードでの実行

7.1. 前提条件

Red Hat JBoss Data Grid を初めてリモートクライアントサーバーモードで実行するための前提条件は次のとおりです。

- 適切なバージョンの OpenJDK がインストールされている必要があります。詳細は「Red Hat Enterprise Linux に OpenJDK をインストール」を参照してください。
- 最新バージョンの JBoss Data Grid をダウンロードし、インストールする必要があります。「Red Hat JBoss Data Grid のダウンロード」を参照してください。

バグを報告する

7.2. RED HAT JBOSS DATA GRID のスタンドアロンモードでの実行

スタンドアロンモードは、ローカルモードで動作する Red Hat JBoss Data Grid の単一のインスタンスです。ローカルモードの JBoss Data Grid は、簡単な単一ノードのインメモリーデータキャッシュとして動作します。

次のスクリプトを実行し、JBoss Data Grid をスタンドアロンモードで起動します。

```
$JDG_HOME/bin/standalone.sh
```

このコマンドは **$JDG_HOME/standalone/configuration/standalone.xml** ファイルのデフォルト設定情報を使用して JBoss Data Grid を起動します。

バグを報告する

7.3. RED HAT JBOSS DATA GRID のクラスターモードでの実行

クラスターモードは、複数の Red Hat JBoss Data Grid 標準インスタンスで構成されるクラスターを参照します。

次のスクリプトを実行し、JBoss Data Grid をクラスターモードで起動します。

```
$JDG_HOME/bin/standalone.sh -c clustered.xml
```

このコマンドは **$JDG_HOME/standalone/configuration/clustered.xml** ファイルのデフォルト設定情報を使用して JBoss Data Grid を起動します。

バグを報告する

7.4. 管理対象ドメインでの RED HAT JBOSS DATA GRID の実行

管理対象ドメインにより、複数のサーバーインスタンスとグループをドメインコントローラーの管理コンソールを使って一元管理することができます。

次のスクリプトを実行し、JBoss Data Grid サーバーを管理対象ドメインで起動します。

```
$JDG_HOME/bin/domain.sh
```

このコマンドは **$JDG_HOME/standalone/configuration/domain.xml** ファイルと **$JDG_HOME/domain/configuration/host.xml** ファイルのデフォルト設定情報を使用して JBoss Data Grid を起動します。

バグを報告する

7.5. カスタム設定を用いた RED HAT JBOSS DATA GRID の実行

カスタム設定を使用して Red Hat JBoss Data Grid を実行するには、設定ファイルを **$JDG_HOME/standalone/configuration** ディレクトリーに追加します。

以下のコマンドを使用して、スタンドアロンモード用に作成されたカスタム設定ファイルを指定します。

```
$JDG_HOME/bin/standalone.sh -c ${FILENAME}
```

このスクリプトに使用される **-c** は絶対パスを許可しないため、指定したファイルは **$JDG_HOME/standalone/configuration** ディレクトリーにある必要があります。

-c パラメーターを使用せずにコマンドを実行すると、JBoss Data Grid はデフォルト設定を使用します。

管理対象ドメインは、デフォルトで **domain.xml** および **host.xml** の 2 つの個別ファイルで設定されるため、カスタム設定ファイルを指定するための 2 つの個別フラグがあります。

サーバーグループプロファイルのカスタム設定ファイルを定義するには、上記のと以下のコマンドで示されるように **-c** パラメーターを使用します。

```
$JDG_HOME/bin/domain.sh -c ${FILENAME}
```

サーバーのカスタム設定ファイルを定義するには、以下のコマンドに示されるように **--host-config** パラメーターを使用します。

```
$JDG_HOME/bin/domain.sh --host-config=${FILENAME}
```

バグを報告する

7.6. IP アドレスを設定して RED HAT JBOSS DATA GRID を実行

本番で使用する場合、Red Hat JBoss Data Grid を **127.0.0.1/localhost** にバインドせずに、指定の IP アドレスへバインドする必要があります。スクリプトで **-b** パラメーターを使用して IP アドレスを指定します。

スタンドアロンモードでは IP アドレスを次のように設定します。

```
$JDG_HOME/bin/standalone.sh -b ${IP_ADDRESS}
```

ドメインモードでは、ホストコントローラーとサーバーの IP アドレスを次のように設定します。

```
$JDG_HOME/bin/domain.sh -b ${IP_ADDRESS}
```

バグを報告する

7.7. RED HAT JBOSS DATA GRID の実行

JBoss Data Grid は、以下の 3 つの方法の 1 つを使用して実行できます。

- (**$JDG_HOME/standalone/configuration**) にある **standalone.xml** ファイルを使用して JBoss Data Grid を実行するには、以下のコマンドを実行します。

```
$JDG_HOME/bin/standalone.sh
```

- デフォルト以外の設定ファイルを使用して JBoss Data Grid を実行するには、**-c** の後に設定ファイル名を指定して以下のコマンドを実行します。

```
$JDG_HOME/bin/standalone.sh -c clustered.xml
```

- (**$JDG_HOME/domain/configuration/** にある) **domain.xml** および **host.xml** ファイルに定義された設定を使用して JBoss Data Grid を実行するには、以下のコマンドを実行します。

```
$JDG_HOME/bin/domain.sh
```

バグを報告する

第8章 エンドポイントのないノードとして RED HAT JBOSS DATA GRID を実行

サービスはチャネルを使用してメッセージを送信し、他のメッセージと通信します。エンドポイントはこれらのサービスの通信ポイントで、チャネルで送信されたメッセージを送受信するために使用されます。そのため、エンドポイントのないノードは同じクラスターで他のノードと通信できますが、クライアントとは通信できません。

バグを報告する

8.1. エンドポイントのないノードの利点

Red Hat JBoss Data Grid でエンドポイントのないノードを作成する主な利点には、データレプリケーションが関係します。

クライアントはエンドポイントのないノードに直接アクセスできません。そのため、主にクライアントと通信できる他のノードからデータをレプリケートするために使用されます。その結果、ノードにクライアントがアクセスできないデータのバックアップコピーが作成され、クライアントが送信したエラーによって発生した障害から保護することができます。

バグを報告する

8.2. エンドポイントのないノードの設定例

Red Hat JBoss Data Grid は、エンドポイントのないノードを設定するための設定例を提供します。

手順8.1 エンドポイントのないノードの JBoss Data Grid 設定例を検索

1. **JBoss Data Grid ZIP ファイルの展開**
   JBoss Data Grid リモートクライアントサーバーモードの ZIP ファイルを展開します。このファイルの名前は **jboss-datagrid-server-*${version}*** です。ファイル名に適切なバージョンを追加します。

2. **適切なフォルダーへ移動**
   展開したフォルダー内で、**$JDG_HOME/docs/examples/config** フォルダーに移動します。

3. **設定例ファイルの検索**
   エンドポイントのないノードの設定が含まれる **clustered-storage-only.xml** を見つけます。

バグを報告する

8.3. エンドポイントのないノードの設定

スタンドアロン高可用性設定などの標準的な設定は、エンドポイントのないノード向けに変更できます。以下の手順に従います。

1. ***datagrid*** サブシステムを削除します。
2. ***modcluster*** を削除します。
3. ***datasource*** 定義を削除します。
4. ***mod_cluster***、**Hot Rod**、および **memcached** の ***socket-bindings*** を削除します。

上記の項目を削除し、すべてのエンドポイントが確実に設定から削除されるようにし、クラスター化を不可能にします。結果、エンドポイントのないノードが設定されます。

バグを報告する

第9章 RED HAT JBOSS DATA GRID のライブラリーモードでの実行

ここでは、Red Hat JBoss Data Grid をライブラリーモードで使用する方法について取り上げます。

- 以降の章での前提条件として、「新しい Red Hat JBoss Data Grid プロジェクトの作成」の手順に従って新しいプロジェクトを設定します。
- 次に、埋め込みキャッシュ (詳細は *10章Red Hat JBoss Data Grid のライブラリーモードでの実行 (単一ノードの設定)* を参照) またはクラスター化キャッシュ (*11章Red Hat JBoss Data Grid のライブラリーモードでの実行 (マルチノードの設定)* を参照) のいずれかとして JBoss Data Grid を使用します。各チュートリアルは Infinispan クイックスタートを基にしています。
- 最後に、*12章Red Hat JBoss EAP での Red Hat JBoss Data Grid アプリケーションの監視* の手順に従って、JBoss Data Grid を使用して Red Hat JBoss EAP アプリケーションを監視します。

バグを報告する

9.1. 新しい RED HAT JBOSS DATA GRID プロジェクトの作成

本章は、新しい Red Hat JBoss Data Grid プロジェクトを作成するためのガイドになります。ここで説明するタスクは *10章Red Hat JBoss Data Grid のライブラリーモードでの実行 (単一ノードの設定)* および *11章Red Hat JBoss Data Grid のライブラリーモードでの実行 (マルチノードの設定)* のクイックスタートタスクの前提条件になります。

バグを報告する

9.2. プロジェクトへの依存関係の追加

プロジェクトに依存関係を追加し、Red Hat JBoss Data Grid を設定します。Maven や Maven の依存関係をサポートするその他のビルドシステムを使用している場合、Maven リポジトリーフォルダーにある **pom.xml** ファイルに以下を追加します。

```
<dependency>
          <groupId>org.infinispan</groupId>
          <artifactId>infinispan-embedded</artifactId>
          <version>$VERSION</version>
</dependency>
```

注記

version の値を、JBoss Data Grid に含まれるライブラリーの適切なバージョンに置き換えます。

バグを報告する

9.3. プロジェクトへのプロファイルの追加

プロジェクトに対して JBoss Maven リポジトリーを有効にするには、次のようにプロファイルを **$HOME/.m2/settings.xml** の **settings.xml** ファイルに追加します。

例9.1 プロファイルの追加

```
<profiles>

    <!-- Configure the JBoss GA Maven repository -->
    <profile>
      <id>jboss-ga-repository</id>
      <repositories>
        <repository>
          <id>jboss-ga-repository</id>
          <url>http://maven.repository.redhat.com/techpreview/all</url>
          <releases>
            <enabled>true</enabled>
          </releases>
          <snapshots>
            <enabled>false</enabled>
          </snapshots>
        </repository>
      </repositories>
      <pluginRepositories>
        <pluginRepository>
          <id>jboss-ga-plugin-repository</id>
          <url>http://maven.repository.redhat.com/techpreview/all</url>
          <releases>
            <enabled>true</enabled>
          </releases>
          <snapshots>
            <enabled>false</enabled>
          </snapshots>
        </pluginRepository>
      </pluginRepositories>
    </profile>
    <!-- Configure the JBoss Early Access Maven repository -->
    <profile>
      <id>jboss-earlyaccess-repository</id>
      <repositories>
        <repository>
          <id>jboss-earlyaccess-repository</id>

<url>http://maven.repository.redhat.com/earlyaccess/all/</url>
          <releases>
            <enabled>true</enabled>
          </releases>
          <snapshots>
            <enabled>false</enabled>
          </snapshots>
        </repository>
      </repositories>
      <pluginRepositories>
        <pluginRepository>
          <id>jboss-earlyaccess-plugin-repository</id>

<url>http://maven.repository.redhat.com/earlyaccess/all/</url>
          <releases>
            <enabled>true</enabled>
          </releases>
```

```
            <snapshots>
              <enabled>false</enabled>
            </snapshots>
          </pluginRepository>
        </pluginRepositories>
      </profile>

    </profiles>

  <!-- Add active profiles information here -->
```

以下が **settings.xml** ファイルに含まれていることを確認し、プロファイルを有効にします。

例9.2 プロファイルの有効化

```
    <activeProfiles>
      <!-- Optionally, make the repositories active by default -->
      <activeProfile>jboss-ga-repository</activeProfile>
      <activeProfile>jboss-earlyaccess-repository</activeProfile>
    </activeProfiles>
```

宣言型の依存関係管理をサポートしないビルドシステムを使用している場合は、Red Hat JBoss Data Grid パッケージに含まれる **client/java/** ディレクトリーの内容を、ビルドクラスパスへ追加します。

バグを報告する

第10章 RED HAT JBOSS DATA GRID のライブラリーモードでの実行 (単一ノードの設定)

10.1. クイックスタートクラスの実行

前提条件

これらのクイックスタートは https://github.com/infinispan/infinispan-quickstart にある Infinispan クイックスタートを使用します。以下の手順は **infinispan-quickstart/embedded-cache** クイックスタートを使用します。

手順10.1 クイックスタートクラスの実行

1. **Quickstart.java ファイルを開きます。**
   infinispan-quickstart/embedded-cache にある **Quickstart.java** というファイルを開きます。

2. **クイックスタートクラスの追加**
   Quickstart.java ファイルを調べると、このクラスが**DefaultCacheManager** を作成する後に、キャッシュ名が指定されていないので **デフォルト**キャッシュへのローカル参照を取得することを確認できます。

```
package org.infinispan.quickstart.embeddedcache;

import org.infinispan.Cache;
import org.infinispan.manager.DefaultCacheManager;

public class Quickstart {

   public static void main(String args[]) throws Exception {
      Cache<Object, Object> c = new
DefaultCacheManager().getCache();
   }

}
```

3. **依存関係のコピーおよび Java クラスのコンパイル**
   次のコマンドを使用して、すべてのプロジェクト依存関係をディレクトリーにコピーし、プロジェクトから Java クラスをコンパイルします。

```
$ mvn clean compile dependency:copy-dependencies -DstripVersion
```

4. **main メソッドの実行**
   次のコマンドを用いて main メソッドを実行します。

```
$ java -cp target/classes/:target/dependency/* package
org.infinispan.quickstart.embeddedcache.Quickstart
```

バグを報告する

10.2. デフォルトキャッシュの使用

10.2.1. キャッシュのデータ追加および削除

Red Hat JBoss Data Grid は、キャッシュに格納されたデータにアクセスして変更を行うため、提案されている JSR-107 API と似たインターフェースを提供します。

次の手順は、DefaultCacheQuickstart.java ファイルに入力された各行が何を実行するかを定義する例になります。

手順10.2 キャッシュのデータ追加および削除

1. エントリーを追加し、*key* と *value* を希望のキーと値に置き換えます。

```
cache.put("key", "value");
```

2. キャッシュにエントリーが存在することを確認します。

```
assertEquals(1, cache.size());
assertTrue(cache.containsKey("key"));
```

3. キャッシュからエントリーを削除します。

```
Object v = cache.remove("key");
```

4. エントリーがキャッシュに存在しないことを確認します。

```
assertEquals("value", v);
assertTrue(cache.isEmpty());
```

バグを報告する

10.2.2. キー値の追加と置換

Red Hat JBoss Data Grid はスレッドセーフなデータ構造を提供します。

次の手順は、**DefaultCacheQuickstart.java** ファイルに入力された各行が何を実行するかを定義する例になります。

手順10.3 キー値の追加と置換

- **value** をキーの値とするエントリー**key** を追加します。

```
cache.put("key", "value");
```

手順10.4 キー値の置換

1. 次のコードは、**key** および **key2** という名前のキーを検索します。検索した 2 つのキーが見つからなかった場合、JBoss Data Grid は指定のキー名と値を持つ 2 つの新しいキーを作成します。

```
cache.putIfAbsent("key", "newValue");
cache.putIfAbsent("key2", "value2");
```

2. 以下のコードは、格納されたキーの値が格納したい値と同じであることを確認します。

```
assertEquals(cache.get("key"), "value");
assertEquals(cache.get("key2"), "value2");
```

関連トピック:

- 「キーバリューペアについて」

バグを報告する

10.2.3. エントリーの削除

JBoss Data Grid の使用方法に応じて、エントリーを削除する別々のメソッドが使用されます。

ライブラリーモード

以下のメソッドはすべて **org.infinispan.Cache** とそのサブクラスにあります。

- **remove(key)**: 単一キーをキャッシュから削除します。
- **removeAsync(key)**: 単一キーをキャッシュから非同期的に削除します。
- **clear()**: マッピングのすべてをキャッシュから削除し、呼び出しが完了するとキャッシュを空にします。
- **clearAsync()**: マッピングのすべてをキャッシュから非同期的に削除し、呼び出しが完了するとキャッシュを空にします。
- **cache.evict(key)**: エントリーをキャッシュから削除し、キャッシュストアが定義されている場合はエントリーをキャッシュストアに移動します。キャッシュストアが定義されていない場合、エントリーはキャッシュから削除され、失われます。

リモートクライアントサーバーモード

以下のメソッドはすべて **org.infinispan.client.hotrod.RemoteCache** とそのサブクラスにあります。

- **remove(key)**: 単一キーをキャッシュから削除します。
- **removeAsync(key)**: 単一キーをキャッシュから非同期的に削除します。
- **clear()**: マッピングのすべてをキャッシュから削除し、呼び出しが完了するとキャッシュを空にします。
- **clearAsync()**: マッピングのすべてをキャッシュから非同期的に削除し、呼び出しが完了するとキャッシュを空にします。
- **removeWithVersion(key, version)**: 単一キーの現行バージョンが指定バージョンと一致する場合にのみ、単一キーをキャッシュから削除します。
- **removeWithVersionAsync(key, value)**: 単一キーの現行バージョンが指定バージョンと一致する場合にのみ、単一キーをキャッシュから非同期的に削除します。

上記メソッドのいずれかについての詳細は、『API ドキュメント』を参照してください。

バグを報告する

10.2.4. データセットの配置および取得

AdvancedCache および **RemoteCache** インターフェースには、既存データの**Map** の Put または Get のいずれかを一括で実行するためのメソッドが含まれます。通常これらの操作は、複数のトランザクションではなく単一ネットワーク操作が実行されるため、クライアントサーバーモードの場合はとくに個別の操作の同等のシーケンスよりも大幅に効率がよくなります。

一括処理を実行すると、メモリーオーバーヘッドが処理中に高くなります。**get** または **put** 処理が単一実行における完全な **Map** に対応する必要があるためです。

各クラスのメソッドは以下のようになります。

- **AdvancedCache**:
  - **Map<K,V> getAll(Set<?> keys)**: 要求されたキーセットに関連する値を含む**Map** を返します。
  - void **putAll(Map<? extends K, ? extends V> map, Metadata metadata)**: 指定されたマップからこのキャッシュにすべてのマッピングをコピーします。ここでは **Metadata** のインスタンスが取られ、保存されるエントリーについてのライフスパン、バージョンなどのメタ情報が提供されます。
- **RemoteCache**:
  - **Map<K,V> getAll(Set<? extends K> keys)**: 要求されたキーセットに関連する値を含む **Map** を返します。
  - void **putAll(Map<? extends K, ? extends V> map)**: 指定されたマップからこのキャッシュにすべてのマッピングをコピーします。
  - void **putAll(Map<? extends K, ? extends V> map, long lifespan, TimeUnit unit)**: エントリーの期限が切れる前のライフスパンと共に、指定されたマップからこのキャッシュにすべてのマッピングをコピーします。
  - void **putAll(Map<? extends K, ? extends V> map, long lifespan, TimeUnit lifespanUnit, long maxIdleTime, TimeUnit maxIdleTimeUnit)**: エントリーの期限が切れる前のタイムスパンと、エントリーの期限が切れる前のエントリーのアイドル状態が許容される最大時間と共に、指定されたマップからこのキャッシュにすべてのマッピングをコピーします。

バグを報告する

10.2.5. データライフの調整

デフォルトでは、Red Hat JBoss Data Grid のエントリーは期限なし (immortal) ですが、この設定は変更できます。

次の手順は、**DefaultCacheQuickstart.java** ファイルに入力された各行が何を実行するかを定義する例になります。

手順10.5 データライフの調整

1. キーの ***lifespan*** 値を変更します。

```
cache.put("key", "value", 5, TimeUnit.SECONDS);
```

2. キャッシュにキーが含まれていることを確認します。

```
assertTrue(cache.containsKey("key"));
```

3. 割り当てられた ***lifespan*** 時間が期限切れになると、キーはキャッシュから削除されます。

```
Thread.sleep(10000);
assertFalse(cache.containsKey("key"));
```

バグを報告する

10.2.6. デフォルトのデータ期限

デフォルトでは、新規に作成されたエントリーにはライフスパンや最大アイドル時間値セットがありません。これらの 2 つの値がない場合、データエントリーは永久に期限切れにならないため、期限なし (immortal) データと呼ばれます。

バグを報告する

10.2.7. XML を用いた名前付きキャッシュの登録

プログラムを使用せずに、名前付きキャッシュを宣言的に (XML を使用) 設定するには、**infinispan.xml** ファイルを設定します。

サンプルの **infinispan.xml** ファイルは、**secure-embedded-cache/src/main/resources/** フォルダー内の https://github.com/jboss-developer/jboss-jdg-quickstarts/ にあり、完全スキーマは 『Red Hat JBoss Data Grid Library』 ディストリビューションの **docs/schema/** ディレクトリーで利用できます。

バグを報告する

第11章 RED HAT JBOSS DATA GRID のライブラリーモードでの実行 (マルチノードの設定)

11.1. JGROUPS チャネルの共有

Red Hat JBoss Data Grid は、ネットワークトランスポートとして JGroups を使用し、簡単に使用できるクラスタリングを提供します。そのため、JBoss Data Grid でクラスターを形成するために必要な初期操作を JGroups が管理します。

単一の CacheManager から作成されたすべてのキャッシュは、デフォルトでは同じ JGroups チャネルを共有します。JGroups チャネルは、レプリケーションメッセージや分散メッセージを多重化するために使用されます。

次の例では、3 つのキャッシュがすべて同じ JGroups チャネルを使用します。

例11.1 共有 JGroups チャネル

```
EmbeddedCacheManager cm = $LOCATION
Cache<Object, Object> cache1 = cm.getCache("replSyncCache");
Cache<Object, Object> cache2 = cm.getCache("replAsyncCache");
Cache<Object, Object> cache3 = cm.getCache("invalidationSyncCache");
```

$LOCATION を CacheManager の場所に置き換えてください。

バグを報告する

11.2. クラスターでの RED HAT JBOSS DATA GRID の実行

Red Hat JBoss Data Grid のクラスター化クイックスタートは、https://github.com/infinispan/infinispan-quickstart/tree/master/clustered-cache のクイックスタートを基にしています。

バグを報告する

11.2.1. プロジェクトのコンパイル

Maven を使用し、次のコマンドを用いてプロジェクトをコンパイルします。

```
$ mvn clean compile dependency:copy-dependencies -DstripVersion
```

バグを報告する

11.2.2. レプリケーションモードでクラスター化されたキャッシュを実行

クラスター化されたキャッシュに関する Red Hat JBoss Data Grid のレプリケーションモードの例を実行するには、異なるコンソールで 2 つのノードを開始します。

手順11.1 レプリケーションモードでクラスター化されたキャッシュを実行

1. 次のコマンドを用いて最初のノードを立ち上げます。

```
$ java -cp target/classes/:target/dependency/*
org.infinispan.quickstart.clusteredcache.replication.Node0
```

2. 次のコマンドを用いて、2 つ目のノードを立ち上げます。

```
$ java -cp target/classes/:target/dependency/*
org.infinispan.quickstart.clusteredcache.replication.Node1
```

結果

両方のノードで JGroups と JBoss Data Grid が初期化されます。約 15 秒後に、キャッシュエントリーのログメッセージが最初のノードのコンソール上に表示されます。

バグを報告する

11.2.3. ディストリビューションモードでクラスター化されたキャッシュを実行

クラスター化されたキャッシュに関する Red Hat JBoss Data Grid のディストリビューションモードの例を実行するには、異なるコンソールで 2 つのノードを開始します。

手順11.2 ディストリビューションモードでクラスター化されたキャッシュを実行

1. 次のコマンドを用いて最初のノードを立ち上げます。

```
$ java -cp target/classes/:target/dependency/*
org.infinispan.quickstart.clusteredcache.distribution.Node0
```

2. 次のコマンドを用いて、2 つ目のノードを立ち上げます。

```
$ java -cp target/classes/:target/dependency/*
org.infinispan.quickstart.clusteredcache.distribution.Node1
```

3. 次のコマンドを用いて、3 つ目のノードを立ち上げます。

```
$ java -cp target/classes/:target/dependency/*
org.infinispan.quickstart.clusteredcache.distribution.Node2
```

結果

3 つのノードで JGroups と JBoss Data Grid が初期化されます。約 15 秒後に、3 番目のノードによって追加された 10 個のエントリーが、最初のノードと 2 番目のノードに分散されたように表示されます。

バグを報告する

11.2.4. クラスターの設定

次の手順に従って、クラスターを追加および設定します。

手順11.3 クラスターの設定

1. 新しいクラスターのデフォルト設定を追加します。

2. ネットワークの要件に従って、デフォルトのクラスター設定をカスタマイズします。カスタマイズは宣言的 (XML を使用) またはプログラム的に行います。

3. レプリケートされたデータグリッドまたは分散されたデータグリッドを設定します。

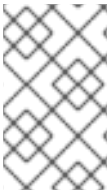
バグを報告する

11.2.4.1. デフォルトクラスター設定の追加

クラスター設定を追加して、クラスターが存在し、定義されていることを Red Hat JBoss Data Grid が確実に認識するようにします。この目的を達成するデフォルト設定は次の通りです。

例11.2 デフォルト設定

```
new ConfigurationBuilder()
   .clustering().cacheMode(CacheMode.REPL_SYNC)
   .build()
```



注記

新しい **GlobalConfigurationBuilder().clusteredDefault()** を使用して、事前設定されたクラスター対応の **GlobalConfiguration** を迅速に作成します。この設定はカスタマイズ可能です。

バグを報告する

11.2.4.2. デフォルトクラスター設定のカスタマイズ

ネットワークの要件によっては、JGroups 設定のカスタマイズが必要なことがあります。

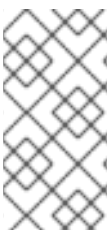
プログラミングによる設定:

以下の GlobalConfiguration コードを使用して、JGroups の設定に使用するファイルの名前を指定します。

```
new
GlobalConfigurationBuilder().transport().addProperty("configurationFile",
"jgroups.xml")
   .build()
```

jgroups.xml を希望のファイル名に置き換えます。

jgroups.xml ファイルは **Infinispan-Quickstart/clustered-cache/src/main/resources/** にあります。



注記

JGroups を ループバックインターフェースのみにバインドする場合は (設定されたファイアウォールを避けるため)、システムプロパティー **-Djgroups.bind_addr="127.0.0.1"** を使用します。これは、すべてのノードが 1 つのマシン上にある状態でクラスターをテストする場合に特に便利です。

宣言的な設定:

infinispan.xml ファイルにある以下の XML スニペットを使用して、 Red Hat JBoss Data Grid の XML 設定を使用するよう JGroups プロパティーを設定します。

```
<global>
   <transport>
      <properties>
         <property name="configurationFile" value="jgroups.xml"/>
      </properties>
   </transport>
</global>
```

バグを報告する

11.2.4.3. レプリケートされたデータグリッドの設定

Red Hat JBoss Data Grid のレプリケートモードは、データグリッドのすべてのノードで各エントリーが確実にレプリケートされるようにします。

このモードは、ノード障害によるデータの損失に対応するセキュリティーを提供し、優れたデータの可用性を提供します。ストレージ容量を、最小メモリーのノードで使用できるストレージの量に制限して、これらの利点を実現します。

プログラミングによる設定:

以下のコードスニペットを使用して、レプリケーションモードのキャッシュをプログラムを用いて設定します (同期または非同期)。

```
private static EmbeddedCacheManager createCacheManagerProgramatically() {
   return new DefaultCacheManager(
      new GlobalConfigurationBuilder()
         .transport().addProperty("configurationFile", "jgroups.xml")
         .build(),
      new ConfigurationBuilder()
         .clustering().cacheMode(CacheMode.REPL_SYNC)
         .build()
   );
}
```

宣言的な設定:

infinispan.xml ファイルを編集して次の XML コードが含まれるように、レプリケーションモードのキャッシュを宣言的に設定します (同期または非同期)。

```
<infinispan xsi:schemaLocation="urn:infinispan:config:8.3
http://www.infinispan.org/schemas/infinispan-config-8.3.xsd"
xmlns:xsi="http://www.w3.org/2001/XMLSchema-instance"
xmlns="urn:infinispan:config:8.3">
   <global>
      <transport>
         <properties>
            <property name="configurationFile" value="jgroups.xml"/>
         </properties>
      </transport>
   </global>
```

```
   <default>
      <clustering mode="replication">
         <sync/>
      </clustering>
   </default>
</infinispan>
```

次のコードを使用して、XML 設定ファイルを持つ DefaultCacheManager を初期化し、返します。

```
private static EmbeddedCacheManager createCacheManagerFromXml() throws
IOException {
   return new DefaultCacheManager("infinispan.xml");}
```

注記

JBoss EAP には独自の基礎となる JMX が含まれています。そのため、JBoss EAP でサンプルコードを使用するときに競合が発生し、**org.infinispan.jmx.JmxDomainConflictException: Domain already registered org.infinispan** などのエラーが表示されることがあります。

この問題を回避するには、以下のようにグローバル設定を指定します。

```
GlobalConfiguration glob = new GlobalConfigurationBuilder()
 .clusteredDefault()
        .globalJmxStatistics()
          .allowDuplicateDomains(true)
          .enable()
        .build();
```

バグを報告する

11.2.4.4. 分散されたデータグリッドの設定

Red Hat JBoss Data Grid のディストリビューションモードは、データグリッドの全ノードのサブセット上に各エントリーが確実に格納されるようにします。サブセットのノード数は ***numOwners*** パラメーターによって制御され、各エントリーの「所有者」の数が示されます。

ディストリビューションモードではストレージ容量が増えますが、アクセス時間も長くなり、永続性 (ノード障害に対する保護) も低下します。***numOwners*** の値を調整し、容量、永続性、および可用性のバランスが取れるように設定してください。JBoss Data Grid のトポロジーに対応する一貫性のあるハッシュによって、永続性がさらに向上されます。このようなハッシュは、さまざまなデータセンター、ラック、およびノードの全体でエントリーの「所有者」を検索します。

プログラミングによる設定:

次のように、ディストリビューションモードのキャッシュをプログラムを用いて設定します (同期または非同期)。

```
new ConfigurationBuilder()
   .clustering()
      .cacheMode(CacheMode.DIST_SYNC)
      .hash().numOwners(2)
   .build()
```

宣言的な設定:

宣言的な設定:

infinispan.xml ファイルに次の XML コードが含まれるようにし、ディストリビューションモードのキャッシュを宣言的に設定します (同期または非同期)。

```
<default>
   <clustering mode="distribution">
      <sync/>
      <hash numOwners="2"/>
   </clustering>
</default>
```

バグを報告する

第12章 RED HAT JBOSS EAP での RED HAT JBOSS DATA GRID アプリケーションの監視

Red Hat JBoss Data Grid ライブラリーアプリケーション (WAR または EAR ファイルの形式) を JBoss Enterprise Application Server 6 (またはこれ以降) 内にデプロイし、JBoss Operations Network を使用して監視することができます。

バグを報告する

12.1. 前提条件

JBoss Enterprise Application Platform で Red Hat JBoss Data Grid ライブラリーアプリケーションを監視するための前提条件は次のとおりです。

- JBoss Enterprise Application Platform 6 (またはそれ以降) をインストールし、設定する必要があります。
- JBoss Operations Network 3.2.2 (またはそれ以降) をインストールし、設定する必要があります。
- JBoss Data Grid (6.3 以降) ライブラリーモードプラグインをインストールし、設定する必要があります。

バグを報告する

12.2. RED HAT JBOSS EAP での RED HAT JBOSS DATA GRID アプリケーションの監視

前提条件をすべて満たすようにしてください。手順に従って、JBoss Operations Network または RHQ を使用して JBoss Enterprise Application Platform で Red Hat JBoss Data Grid アプリケーションを監視します。

手順12.1 JBoss Enterprise Application Platform での JBoss Data Grid アプリケーションの監視

1. **RHQ/JBoss Operations Network の設定**
   以下のように、RHQ/JBoss Operations Network 固有のプロパティー (***org.rhq.resourceKey***) を **/bin/standalone.conf** ファイルに追加します。

   ```
   JAVA_OPTS="$JAVA_OPTS -Dorg.rhq.resourceKey=MyEAP"
   ```

   このコマンドは、プロパティーを間接的に JBoss Enterprise Application Platform のコマンドラインに追加します。

2. **フル JDK を使用して RHQ/JBoss Operations Network が実行されていることを確認**
   RHQ/JBoss Operations Network エージェントが JRE ではなくフル JDK を使用して起動したことを確認してください。これは、エージェントは JDK の **tools.jar** ファイルにアクセスする必要があるためです。

   RHQ/JBoss Operations Network エージェントが JDK を使用するよう設定するため、ご使用のオペレーティングシステムに対応する以下の手順に従います。

   a. Linux の場合は、***RHQ_AGENT_JAVA_HOME*** 環境変数をエージェントの **rhq-agent-env.sh** ファイルの JDK ホームディレクトリーに設定します。

b. Windows の場合は、***RHQ_AGENT_JAVA_HOME*** 環境変数をエージェントの **rhq-agent-env.bat** ファイルの JDK ホームディレクトリーに設定します。

3. **エージェントが JBoss Enterprise Application Platform インスタンスのローカルであることを確認**
   JBoss Application Platform インスタンスと同じユーザーが、JBoss Application Platform インスタンスのローカルになるよう RHQ/JBoss Operations Network エージェントを実行するようにしてください。これは、Java Attach API がプロセスに接続するために必要です。

4. **リソースをエージェントインベントリーにインポート**
   RHQ/JBoss Operations Network がリソースを検索できるようになりました。これらのリソースはエージェントインベントリーへインポートできます。

   JBoss Data Grid のユーザーデプロイメントが JMX 統計による JBoss Data Grid キャッシュマネージャーまたはキャッシュの公開を有効すると、リソースは JBoss Enterprise Application Platform インスタンスの子リソースとして表示されます。

バグを報告する

パート V. キャッシュマネージャーのセットアップ

第13章 キャッシュマネージャー

キャッシュマネージャーは、Red Hat JBoss Data Grid においてキャッシュインスタンスを取得するための主なメカニズムであり、キャッシュを使用する際のスタートポイントになります。

JBoss Data Grid では、キャッシュマネージャーは以下の理由により役に立ちます。

- 複数のインスタンスをオンデマンドで作成します。
- 既存のキャッシュインスタンスを読み出します (すでに作成されたキャッシュ)。

バグを報告する

13.1. キャッシュマネージャーの種類

Red Hat JBoss Data Grid は、次のキャッシュマネージャーを提供します。

- **EmbeddedCacheManager** は、クライアントが使用する Java 仮想マシン (JVM) 内で実行されるキャッシュマネージャーです。現在 JBoss Data Grid は、**EmbeddedCacheManager** インターフェースの **DefaultCacheManager** 実装のみを提供しています。
- **RemoteCacheManager** は、リモートキャッシュにアクセスするために使用されます。**RemoteCacheManager** は、起動時に Hot Rod サーバー (または複数の Hot Rod サーバー) への接続をインスタンス化します。次に **RemoteCacheManager** は、それが実行されている間に永続的な **TCP** 接続を管理します。結果的に、**RemoteCacheManager** はリソースを集中的に使用します。そのため、それぞれの Java 仮想マシン (JVM) に対して単一の **RemoteCacheManager** インスタンスを設定する方法が推奨されます。

バグを報告する

13.2. CACHEMANAGERS の作成

13.2.1. 新しい RemoteCacheManager の作成

手順13.1 新しい RemoteCacheManager の設定

```
import org.infinispan.client.hotrod.RemoteCache;
import org.infinispan.client.hotrod.RemoteCacheManager;
import org.infinispan.client.hotrod.configuration.Configuration;
import org.infinispan.client.hotrod.configuration.ConfigurationBuilder;

Configuration conf = new
ConfigurationBuilder().addServer().host("localhost").port(11222).build();
RemoteCacheManager manager = new RemoteCacheManager(conf);
RemoteCache defaultCache = manager.getCache();
```

1. **ConfigurationBuilder()** コンストラクターを使用して新しい設定ビルダーを作成します。**.addServer()** メソッドは、**.host(<hostname|ip>)** プロパティーと **.port(<port>)** プロパティーで設定されたリモートサーバーを追加します。
2. 指定された設定を使用して新しい **RemoteCacheManager** を作成します。
3. リモートサーバーからデフォルトキャッシュを取得します。

バグを報告する

13.2.2. 新しい組み込みキャッシュマネージャーの作成

CDI を使用せずに新規の EmbeddedCacheManager を作成するには、以下の手順を実行します。

手順13.2 新しい組み込みキャッシュマネージャーの作成

1. 設定 XML ファイルを作成します。たとえば、クラスパス上 (**resources/** フォルダー内) に **my-config-file.xml** ファイルを作成し、このファイルに設定情報を追加します。
2. 設定ファイルを使用してキャッシュマネージャーを作成するには、以下のプログラムを使用した設定を使用します。

```
EmbeddedCacheManager manager = new DefaultCacheManager("my-config-
file.xml");
Cache defaultCache = manager.getCache();
```

上記の手順を完了すると、**my-config-file.xml** で指定された設定を使用して新規の EmbeddedCacheManager が作成されます。

バグを報告する

13.2.3. CDI の使用による新しい組み込みキャッシュマネージャーの作成

CDI を使用して新規の EmbeddedCacheManager インスタンスを作成するには、以下の手順を実行します。

手順13.3 CDI を使用した新規 EmbeddedCacheManager の作成

1. 次のようにデフォルト設定を指定します。

```
public class Config
   @Produces
   public EmbeddedCacheManager defaultCacheManager() {
      ConfigurationBuilder builder = new ConfigurationBuilder();
      Configuration configuration =
builder.eviction().strategy(EvictionStrategy.LRU).maxEntries(100).bu
ild();
      return new DefaultCacheManager(configuration);
   }
}
```

2. デフォルトのキャッシュマネージャーを挿入します。

```
<!-- Additional configuration information here -->
@Inject
EmbeddedCacheManager cacheManager;
<!-- Additional configuration information here -->
```

バグを報告する

13.3. 複数のキャッシュマネージャー

キャッシュマネージャーは、キャッシュを使用する時の開始点であり、Red Hat JBoss Data Grid では、ユーザーが複数のキャッシュマネージャーを作成することができます。それぞれのキャッシュマネージャーは、JMX、エグゼキューターおよびクラスタリングなどの設定を含む、異なるグローバル設定を使用して設定されます。

バグを報告する

13.3.1. 単一のキャッシュマネージャーを用いた複数キャッシュの作成

Red Hat JBoss Data Grid では、同じキャッシュマネージャーを使用し、異なるキャッシュモード (同期または非同期キャッシュモード) を持つ複数のキャッシュを作成することが可能です。

バグを報告する

13.3.2. 複数のキャッシュマネージャーの使用

Red Hat JBoss Data Grid では、複数のキャッシュマネージャーを使用することが可能です。レプリケーションやネットワーキングコンポーネントなどほとんどの場合で、キャッシュインスタンスは内部コンポーネントを共有するため単一のキャッシュマネージャーで十分です。

ただし、1 つのキャッシュが **TCP** プロトコルを使用し、他のキャッシュが**UDP** プロトコルを使用する場合など、複数のキャッシュに異なるネットワーク特性が必要な場合は、複数のキャッシュマネージャーを使用する必要があります。

バグを報告する

13.3.3. 複数のキャッシュマネージャーの作成

Red Hat JBoss Data Grid では、ユーザーは最初のキャッシュマネージャーを作成するために使用された手順を繰り返す (さらに、必要な場合は設定ファイルの内容を調整する) ことにより、さまざまな種類の複数のキャッシュマネージャーを作成することができます。

複数の新規キャッシュマネージャーを作成するために宣言的 API を使用するには、**infinispan.xml** ファイルの内容を新規の設定ファイルにコピーします。新規ファイルで必要な設定についての編集を行ってから、新しいキャッシュマネージャー用にこの新規ファイルを使用します。

バグを報告する

パート VI. RED HAT JBOSS DATA GRID のクイックスタート

以下の表は本ガイドに含まれるクイックスタートの一覧で、使用されるコンテナーおよびモードが記載されています。

表14 クイックスタートの情報

クイックスタート名	コンテナー	JBoss Data Grid モード	詳細のリンク
Hello World	JBoss EAP	ライブラリーモード	14章*Hello World クイックスタート*
Carmart Non-Transactional	JBoss EAP および JBoss Enterprise Web Server	ライブラリーモード	「JBoss EAP を用いた (非トランザクション) CarMart クイックスタート」
Carmart Non-Transactional	JBoss EAP および JBoss Enterprise Web Server	リモートクライアントサーバーモード	「リモートクライアントサーバーモードでの (非トランザクション) CarMart クイックスタート (JBoss EAP)」および「リモートクライアントサーバーモードでの (非トランザクション) CarMart クイックスタート (JBoss Enterprise Web Server)」
Carmart Transactional	JBoss EAP および JBoss Enterprise Web Server	ライブラリーモード	「JBoss EAP を用いた (トランザクション) CarMart クイックスタート」および「JBoss Enterprise Web Server を用いた (トランザクション) CarMart クイックスタート」
Football Application	コンテナーなし	リモートクライアントサーバーモード	16章*Football クイックスタートエンドポイントの例*
Rapid Stock Market	コンテナーなし	リモートクライアントサーバーモード	17章*Rapid Stock Market クイックスタート*
Cluster App	JBoss EAP	ライブラリーモード	18章*Cluster App クイックスタート*

クイックスタート名	コンテナー	JBoss Data Grid モード	詳細のリンク
camel-jbossdatagrid-fuse	JBoss Fuse	ライブラリーモード	19章*camel-jbossdatagrid-fuse クイックスタート*

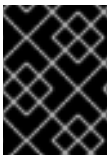
バグを報告する

第14章 HELLO WORLD クイックスタート

Hello World は、Red Hat JBoss Data Grid を使用してキャッシュにデータを保存する方法やキャッシュからデータを読み出す方法を実演する簡単なクイックスタートです。このクイックスタートでは、ユーザーは以下の 2 つの方法でキャッシュにアクセスできます。

- サーブレットからアクセスする方法。
- リクエストスコープ Bean を使用して JSF からアクセスする方法。

アプリケーションにバンドルされるすべてのライブラリー (JAR ファイル) は、JBoss Enterprise Application Platform 6.x へデプロイされます。JBoss Data Grid のライブラリーモードは、分散されたクラスターの単一ノードへのローカルアクセスのみを許可します。また、このモードでは、アプリケーションはターゲットコンテナーの仮想マシン内でデータグリッド機能にアクセスすることもできます。



重要

Hello World クイックスタートは JBoss Data Grid のライブラリーモードでのみ動作します。

場所

JBoss Data Grid の Hello World クイックスタートは、**jboss-datagrid-{VERSION}-quickstarts/** にあります。

バグを報告する

14.1. クイックスタートの前提条件

このクイックスタートの前提条件は次のとおりです。

- Java 6.0 (Java SDK 1.6) 以上
- JBoss Enterprise Application Platform 6.x または JBoss Enterprise Web Server 2.x
- Maven 3.0 以上
- Maven リポジトリーを設定します。詳細は3章*Maven リポジトリーのインストールおよび使用*を参照してください。

バグを報告する

14.2. 2 つのアプリケーションサーバーインスタンスの起動

Hello World クイックスタートをデプロイする前に、アプリケーションサーバー (JBoss Enterprise Application Platform 6.x) の 2 つのインスタンスを起動します。

手順14.1 最初のアプリケーションサーバーインスタンスの起動

1. **ルートディレクトリーへ移動**
   コマンドラインターミナルで、JBoss サーバーディレクトリーのルートへ移動します。
2. **最初のアプリケーションサーバーを起動**

ご使用のオペレーティングシステムに適した以下のコマンドを使用して、選択したアプリケーションサーバーの最初のインスタンスを起動します。

a. Linux の場合:

```
$JBOSS_HOME/bin/standalone.sh
```

b. Windows の場合:

```
$JBOSS_HOME\bin\standalone.bat
```

手順14.2 2 番目のアプリケーションサーバーインスタンスの起動

1. **アプリケーションサーバーのクローン**
   選択した JBoss Server のコピーを作成し、2 番目のインスタンスを作成します。
2. **ルートディレクトリーへ移動**
   コマンドラインターミナルで、JBoss サーバーディレクトリーのルートへ移動します。
3. **2 番目のアプリケーションサーバーの起動**
   ご使用のオペレーティングシステムに適した以下のコマンドを使用します。このコマンドは提供されるポートオフセットでサーバーを起動し、両方のサーバーインスタンスが同じホストで実行されるようにします。

   a. Linux の場合:

   ```
   $JBOSS_HOME2/bin/standalone.sh -Djboss.socket.binding.port-offset=100
   ```

   b. Windows の場合:

   ```
   $JBOSS_HOME2\bin\standalone.bat -Djboss.socket.binding.port-offset=100
   ```

バグを報告する

14.3. HELLO WORLD クイックスタートのビルドおよびデプロイ

クイックスタートをビルドおよびデプロイする前に、前提条件をすべて満たし、2 つのアプリケーションサーバーインスタンスが実行されているようにしてください (詳細は「2 つのアプリケーションサーバーインスタンスの起動」を参照してください)。

手順14.3 Hello World クイックスタートのビルドおよびデプロイ

1. **必要なディレクトリーへの移動**
   コマンドラインターミナルで、クイックスタートのルートディレクトリーへ移動します。
2. **最初のアプリケーションサーバーインスタンスのビルドおよびデプロイ**
   以下のコマンドを使用して、クイックスタートを最初のアプリケーションサーバーインスタンスにビルドおよびデプロイします。

   ```
   # mvn clean package jboss-as:deploy
   ```

このコマンドは、**target/jboss-helloworld-jdg.war** を稼働している最初のサーバーインスタンスにデプロイします。

3. **2 番目のアプリケーションサーバーインスタンスのビルドおよびデプロイ**
以下のコマンドを使用して、クイックスタートを指定のポートを持つ 2 番目のアプリケーションサーバーインスタンスにビルドおよびデプロイします。

```
# mvn clean package jboss-as:deploy -Djboss-as.port=10099
```

このコマンドは、**target/jboss-helloworld-jdg.war** を稼働している 2 番目のサーバーインスタンスにデプロイします。

バグを報告する

14.4. 実行中のアプリケーションへのアクセス

Hello World クリックスタートアプリケーションは以下の URL で実行されます。

- 最初のサーバーインスタンス: http://localhost:8080/jboss-helloworld-jdg
- 2 番目のサーバーインスタンス: http://localhost:8180/jboss-helloworld-jdg

バグを報告する

14.5. アプリケーションでのレプリケーションのテスト

以下の手順に従って、キャッシュエントリーが指定どおりに最初のサーバーインスタンスから 2 番目のサーバーインスタンスへレプリケートすることをテストします。

手順14.4 アプリケーションでのレプリケーションのテスト

1. **最初のサーバーへのアクセス**
最初のアプリケーションサーバーへアクセスし、キーと値を入力します。
   a. ブラウザーのウインドウで以下の URL を指定し、最初のアプリケーションサーバーへアクセスします。

```
http://localhost:8080/jboss-helloworld-jdg
```

   b. キー **foo** を挿入します。
   c. 値 **bar** を挿入します。
2. **2 番目のサーバーへのアクセス**
2 番目のアプリケーションサーバーへアクセスし、キーと値を入力します。
   a. ブラウザーのウインドウで以下の URL を指定し、2 番目のアプリケーションサーバーへアクセスします。

```
http://localhost:8180/jboss-helloworld-jdg
```

   b. **Get Some** をクリックします。
   c. キー **foo** を取得します。

d. **Put Some More** をクリックします。

e. キー **mykey** を挿入します。

f. 値 **myvalue** を挿入します。

3. **すべてのキーおよび値の取得**
最初のサーバーへアクセスし、すべてのキーを要求します。

a. ブラウザーのウインドウで以下の URL を指定し、最初のアプリケーションサーバーへアクセスします。

```
http://localhost:8080/jboss-helloworld-jdg
```

b. **Get Some** をクリックします。

c. **Get All** をクリックし、すべてのキーおよび値を要求します。

結果

最後の手順の実行後、各サーバーに追加されたすべてのデータが別のサーバーにレプリケートされます。

注記

エントリーは最新の更新を行った後、 60 秒後に期限切れになります。

キャッシュのキーへの直接アクセス

事前定義されたサーブレットと対話する場合や、キーを直接キャッシュに保存したり読み出したりする場合は、以下の URL を使用します。

```
http://localhost:8080/jboss-helloworld-jdg/TestServletPut
```

```
http://localhost:8180/jboss-helloworld-jdg/TestServletGet
```

バグを報告する

14.6. アプリケーションの削除

以下の手順に従って、Hello World アプリケーションを削除します。

手順14.5 アプリケーションの削除

1. **アプリケーションサーバーの起動**
両方のサーバーインスタンスが実行されているようにしてください。

2. **ルートディレクトリーへ移動**
コマンドラインターミナルで、クイックスタートのルートディレクトリーへ移動します。

3. **アーカイブの削除**
以下のコマンドを使用して両方のサーバーインスタンスからアーカイブを削除します。

a. 以下のとおり、最初のサーバーインスタンスからアーカイブを削除します。

```
mvn jboss-as:undeploy
```

b. 以下のとおり、2 番目のサーバーインスタンスからアーカイブを削除します。

```
mvn jboss-as:undeploy -Djboss-as.port=10099
```

バグを報告する

第15章 CARMART クイックスタート

Red Hat JBoss Data Grid にはトランザクションおよび非トランザクションの CartMart クイックスタートが含まれています。CartMart クイックスタートは、リレーショナルデータベースの代わりに Red Hat JBoss Data Grid を使用する簡単な Web アプリケーションです。各車 (car) に関する情報はキャッシュに格納されます。キャッシュは、使用モードに応じて宣言的またはプログラムを用いて設定されます。

機能

CarMart クイックスタートは次の機能を提供します。

- 全車の一覧
- 新しい車の追加
- 車の削除
- ヒット数、保存、読み出しなど、キャッシュの統計を表示

使用モード

CarMart クイックスタートは、次の JBoss Data Grid の使用モードで使用可能です。

- リモート JBoss Data Grid サーバーと通信するため Hot Rod クライアントが含まれる、リモートクライアントサーバーモード。
- すべてのライブラリーが **jar** ファイル形式でアプリケーションとバンドルされる、ライブラリーモード。

場所

JBoss Data Grid の CarMart クイックスタートは、**jboss-datagrid-{VERSION}-quickstarts/** にあります。

バグを報告する

15.1. CARMART トランザクションクイックスタート

トランザクションバージョンの CartMart クイックスタートは、リレーショナルデータベースの代わりに Red Hat JBoss Data Grid を使用する簡単な Web アプリケーションです。各車 (car) に関する情報はキャッシュに格納されます。キャッシュは、使用モードに応じて宣言的またはプログラムを用いて設定され、Web アプリケーションと同じ Java 仮想マシン (JVM) で実行されます。

機能

トランザクション CarMart クイックスタートは次の機能を提供します。

- 全車の一覧
- 新しい車の追加
- ロールバックで新しい車を追加
- 車の削除
- ヒット数、保存、読み出しなど、キャッシュの統計を表示

使用モード

トランザクション CarMart クイックスタートは JBoss Data Grid のライブラリモードでのみ使用できます。トランザクション CarMart クイックスタートが Red Hat JBoss Enterprise Web Server 2.x で実行されると、JBoss Transactions からタンドアロントランザクションマネージャーが使用されます。

場所

JBoss Data Grid のトランザクション CarMart クイックスタートは、**jboss-datagrid-{VERSION}-quickstarts/carmart-tx** にあります。

バグを報告する

15.2. CARMART とトランザクションクイックスタートの違い

トランザクションと非トランザクションの CarMart クイックスタートでは、ビルド、デプロイ、および削除の手順は似ていますが、違いもあります。相違点は次のとおりです。

- CarMart はリモートクライアントサーバーモードとライブラリーモードの両方で使用可能です。リモートクライアントサーバーモードではトランザクションが使用できないため、トランザクション CarMart はライブラリーモードでのみ使用可能です。
- また、トランザクションクイックスタートは、トランザクションロールバックがどのように発生するかを表示します。**Add car with rollback** ボタンを使用してロールバックを表示します。CarMart の例には、簡単な **Add car** ボタンがあります。

バグを報告する

15.3. JBOSS EAP を用いた (非トランザクション) CARMART クイックスタート

Carmart (非トランザクション) クイックスタートは、JBoss EAP コンテナーを用いる JBoss Data Grid のライブラリーモードでサポートされます。

バグを報告する

15.3.1. クイックスタートの前提条件

このクイックスタートの前提条件は次のとおりです。

- Java 6.0 (Java SDK 1.6) 以上
- JBoss Enterprise Application Platform 6.x または JBoss Enterprise Web Server 2.x
- Maven 3.0 以上
- Maven リポジトリーを設定します。詳細は3章*Maven リポジトリーのインストールおよび使用*を参照してください。

バグを報告する

15.3.2. CarMart クイックスタートの JBoss EAP へのビルドおよびデプロイ

次の手順では、CarMart アプリケーションを JBoss EAP にビルドおよびデプロイする方法を説明します。

前提条件

この手順の前提条件は次のとおりです。

1. サポートされる JBoss Data Grid ライブラリーモードの分散ファイルを取得します。
2. JBoss Data Grid および JBoss Enterprise Application Platform Maven リポジトリーがインストールされ、設定されているようにしてください。詳細は *3章Maven リポジトリーのインストールおよび使用* を参照してください。
3. JBoss Enterprise Application Platform 6 (またはそれ以上) または JBoss EAP 6 (またはそれ以上) を使用する JBoss サーバーを選択します。

手順15.1 CarMart の JBoss EAP へのビルドおよびデプロイ

1. **JBoss EAP の起動**
   ご使用のオペレーティングシステムに適した以下のコマンドを使用して、選択したアプリケーションサーバーの最初のインスタンスを起動します。

   Linux の場合

   ```
   $JBOSS_HOME/bin/standalone.sh
   ```

   Windows の場合

   ```
   $JBOSS_HOME\bin\standalone.bat
   ```

2. **ルートディレクトリーへ移動**
   コマンドラインターミナルを開き、このクイックスタートのルートディレクトリーへ移動します。

3. **アプリケーションのビルドおよびデプロイ**
   以下のコマンドを実行し、Maven を使用してアプリケーションをビルドおよびデプロイします。

   ```
   $ mvn clean package jboss-as:deploy
   ```

結果

ターゲット war ファイル(**target/jboss-carmart.war**) が実行中の JBoss EAP インスタンスにデプロイされます。

バグを報告する

15.3.3. JBoss EAP での CarMart クイックスタートの表示

次の手順では、JBoss EAP で CarMart クイックスタートを表示する方法を説明します。

前提条件

表示する CarMart クリックスタートがビルドおよびデプロイされている必要があります。

手順15.2 JBoss EAP での CarMart クイックスタートの表示

- アプリケーションを表示するには、ブラウザーを使用して次のリンクに移動します。
  -

```
http://localhost:8080/jboss-carmart
```

バグを報告する

15.3.4. JBoss EAP での CarMart クイックスタートの削除

次の手順では、デプロイされたアプリケーションを JBoss EAP から削除する方法を説明します。

手順15.3 JBoss EAP からアプリケーションを削除

- アプリケーションを削除するには、このクイックスタートのルートディレクトリーから次のコマンドを使用します。

```
$ mvn jboss-as:undeploy
```

バグを報告する

15.4. JBOSS ENTERPRISE WEB SERVER を用いた (非トランザクション) CARMART クイックスタート

Carmart (非トランザクション) クイックスタートは、JBoss Enterprise Web Server コンテナーを用いる JBoss Data Grid のライブラリーモードでサポートされます。

バグを報告する

15.4.1. CarMart クイックスタートの JBoss Enterprise Web Server へのビルドおよびデプロイ

次の手順では、CarMart アプリケーションを JBoss Enterprise Web Server にビルドおよびデプロイする方法を説明します。

前提条件

この手順の前提条件は次のとおりです。

1. JBoss Data Grid および JBoss Enterprise Application Platform Maven リポジトリーがインストールされ、設定されているようにしてください。詳細は *3章Maven リポジトリーのインストールおよび使用* を参照してください。
2. ご使用のアプリケーションに JBoss Enterprise Web Server 2 (またはそれ以上) を選択し、インストールしてください。

手順15.4 CarMart クイックスタートのサーバーへのビルド (ライブラリーモード)

1. **サーバーの起動**
   ご使用のオペレーティングシステムに適した以下のコマンドを使用して、選択したアプリケーションサーバーの最初のインスタンスを起動します。

   Linux の場合

   ```
   $JBOSS_EWS_HOME/tomcat7/bin/catalina.sh run
   ```

   Windows の場合

   -

```
$JBOSS_EWS_HOME\tomcat7\bin\catalina.bat run
```

2. **ルートディレクトリーへ移動**
   コマンドラインターミナルを開き、このクイックスタートのルートディレクトリーへ移動します。

3. **アプリケーションのビルドおよびデプロイ**
   以下のコマンドを実行し、Maven を使用してアプリケーションをビルドおよびデプロイします。

```
$ mvn -Plibrary-tomcat clean package tomcat:deploy
```

結果

ターゲット war ファイル(**target/jboss-carmart.war**) が実行中の選択したサーバーのインスタンスにデプロイされます。

バグを報告する

15.4.2. JBoss Enterprise Web Server を用いた CarMart クイックスタートの表示

次の手順では、JBoss Enterprise Web Server で CarMart クイックスタートを表示する方法を説明します。

前提条件

表示する CarMart クリックスタートがビルドおよびデプロイされている必要があります。

手順15.5 CarMart クイックスタートの表示

- アプリケーションを表示するには、ブラウザーを使用して次のリンクに移動します。

```
http://localhost:8080/jboss-carmart
```

バグを報告する

15.4.3. JBoss Enterprise Web Server での CarMart クイックスタートの削除

次の手順では、すでにデプロイされているアプリケーションを JBoss Enterprise Web Server から削除する方法を説明します。

手順15.6 JBoss Enterprise Web Server からアプリケーションを削除

- アプリケーションを削除するには、このクイックスタートのルートディレクトリーから次のコマンドを使用します。

```
$ mvn tomcat:undeploy -Plibrary-tomcat
```

バグを報告する

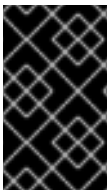
15.5. リモートクライアントサーバーモードでの (非トランザクション) CARMART クイックスタート (JBOSS EAP)

Carmart (非トランザクション) クイックスタートは、JBoss Enterprise Application Platform コンテナーを用いる JBoss Data Grid のリモートクライアントサーバーモードでサポートされます。

バグを報告する

15.5.1. リモートクライアントサーバーモードでの CarMart クイックスタートのビルドおよびデプロイ

このクイックスタートは Hot Rod 経由で Red Hat JBoss Data Grid にアクセスします。この機能は、トランザクション CarMart クイックスタートでは使用できません。



重要

このクイックスタートは JBoss Enterprise Application Platform へデプロイされます。アプリケーションのデプロイメントがサポートされないため、JBoss Data Grid へはデプロイできません。

前提条件

この手順の前提条件は次のとおりです。

1. サポートされる最新の JBoss Data Grid リモートクライアントサーバーモードディストリビューションファイルを Red Hat から取得します。
2. JBoss Data Grid および JBoss Enterprise Application Platform Maven リポジトリーがインストールされ、設定されているようにしてください。詳細は *3章Maven リポジトリーのインストールおよび使用* を参照してください。
3. 使用する JBoss サーバー (JBoss Enterprise Application Platform 6 以降) を選択します。ターミナルウインドウで、JBoss サーバーディレクトリーのルートに移動し、以下のコマンドを入力します。

   Linux の場合

   ```
   $JBOSS_HOME/bin/standalone.sh
   ```

   Windows の場合

   ```
   $JBOSS_HOME\bin\standalone.bat
   ```

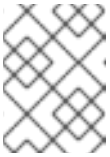
手順15.7 リモートクライアントサーバーモードでの CarMart クイックスタートのビルドおよびデプロイ

1. **スタンドアロンファイルの設定**
   $JDG_HOME/standalone/configuration/ ディレクトリーにある **standalone.xml** ファイルに、次の設定を追加します。

   a. infinispan サブシステムのタグ内に次の設定を追加します。

   ```
   <local-cache name="carcache"
         start="EAGER"
   ```

```
      batching="false"
      statistics="true">
 <eviction strategy="LIRS"
    max-entries="4"/>
</local-cache>
```



注記

設定に ***carcache*** 要素がすでに存在する場合は、提供される設定に置き換えてください。

2. **JBoss Data Grid サーバーの起動**
   以下のスクリプトを実行して、JBoss Data Grid サーバーを起動します。

```
$JDG_HOME/bin/standalone.sh -Djboss.socket.binding.port-offset=100
```

3. **JBoss サーバーの起動**
   次のスクリプトを実行し、アプリケーションがデプロイされる場所で JBoss サーバーインスタンスを起動します。

```
$JBOSS_HOME/bin/standalone.sh
```

4. **任意の設定: ホストおよびポートアドレスの指定**
   アプリケーションは **jboss-datagrid-{VERSION}-quickstarts/carmart/src/main/resources/META-INF/datagrid.properties** ファイルの値を使用して JBoss Data Grid サーバーを見つけます。JBoss Data Grid サーバーがデフォルトのホストおよびポートの値を使用して実行されていない場合、次のようにファイルを編集し、正しいホストおよびポートの値を挿入します。

```
datagrid.host=localhost
datagrid.hotrod.port=11322
```

5. **ルートディレクトリーへ移動**
   コマンドラインターミナルを開き、このクイックスタートのルートディレクトリーへ移動します。

6. **アプリケーションのビルドおよびデプロイ**
   以下のコマンドを実行し、Maven を使用してアプリケーションをビルドおよびデプロイします。

```
$ mvn clean package jboss-as:deploy -Premote-jbossas
```

バグを報告する

15.5.2. リモートクライアントサーバーモードでの CarMart クイックスタートの表示

次の手順では、Red Hat JBoss Data Grid のリモートクライアントサーバーモードで CarMart クイックスタートを表示する方を説明します。

前提条件

表示する CarMart クリックスタートがビルドおよびデプロイされている必要があります。

手順15.8 リモートクライアントサーバーモードでの CarMart クイックスタートの表示

- ブラウザーウインドウで次のリンクにアクセスし、アプリケーションを表示します。

```
http://localhost:8080/jboss-carmart
```

バグを報告する

15.5.3. リモートクライアントサーバーモードでの CarMart クイックスタートの削除

次の手順では、Red Hat JBoss Data Grid のリモートクライアントサーバーモードにすでにデプロイされているアプリケーションを削除する方法を説明します。

手順15.9 リモートクライアントサーバーモードでのアプリケーションの削除

- アプリケーションを削除するには、このクイックスタートのルートディレクトリーから次のコマンドを使用します。

```
$ mvn jboss-as:undeploy -Premote-jbossas
```

バグを報告する

15.6. リモートクライアントサーバーモードでの (非トランザクション) CARMART クイックスタート (JBOSS ENTERPRISE WEB SERVER)

Carmart (非トランザクション) クイックスタートは、JBoss Enterprise Web Server コンテナーを用いる JBoss Data Grid のリモートクライアントサーバーモードでサポートされます。

バグを報告する

15.6.1. リモートクライアントサーバーモードでの CarMart クイックスタートのビルドおよびデプロイ

このクイックスタートは Hot Rod 経由で Red Hat JBoss Data Grid にアクセスします。この機能は、トランザクション CarMart クイックスタートでは使用できません。

重要

このクイックスタートは JBoss Enterprise Web Server または Tomcat へデプロイされます。アプリケーションのデプロイメントがサポートされないため、JBoss Data Grid へはデプロイできません。

前提条件

この手順の前提条件は次のとおりです。

1. サポートされる最新の JBoss Data Grid リモートクライアントサーバーモードディストリビューションファイルを Red Hat から取得します。
2. JBoss Data Grid および JBoss Enterprise Application Platform Maven リポジトリーがインストールされ、設定されているようにしてください。詳細は *3章Maven リポジトリーのインストールおよび使用* を参照してください。

3. **server** 要素を Maven の**settings.xml** ファイルに追加します。適切な Tomcat のクレデンシャルを **server** 内の ***id*** 要素に追加します。

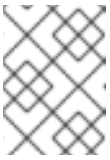
```
<server>
    <id>tomcat</id>
    <username>admin</username>
    <password>admin</password>
</server>
```

手順15.10 リモートクライアントサーバーモードでの CarMart クイックスタートのビルドおよびデプロイ

1. **スタンドアロンファイルの設定**
   $JDG_HOME/standalone/configuration/ ディレクトリーにある**standalone.xml** ファイルに、次の設定を追加します。

   a. infinispan サブシステムのタグ内に次の設定を追加します。

```
<local-cache name="carcache"
      start="EAGER"
      batching="false"
      statistics="true">
 <eviction strategy="LIRS"
    max-entries="4"/>
</local-cache>
```



注記

設定に ***carcache*** 要素がすでに存在する場合は、提供される設定に置き換えてください。

2. **コンテナーの起動**
   アプリケーションがデプロイされる場所で JBoss サーバーインスタンスを起動します。

   Linux の場合:

```
$JBOSS_EWS_HOME/tomcat7/bin/catalina.sh run
```

   Windows の場合:

```
$JBOSS_EWS_HOME\tomcat7\bin\catalina.bat run
```

3. **アプリケーションのビルド**
   次のコマンドを実行し、関連するディレクトリーでアプリケーションをビルドします。

```
$ mvn clean package -Premote-tomcat
```

4. **アプリケーションのデプロイ**
   次のコマンドを実行し、関連するディレクトリーでアプリケーションをデプロイします。

```
mvn tomcat:deploy -Premote-tomcat
```

バグを報告する

15.6.2. リモートクライアントサーバーモードでの CarMart クイックスタートの表示

次の手順では、Red Hat JBoss Data Grid のリモートクライアントサーバーモードで CarMart クイックスタートを表示する方を説明します。

前提条件

表示する CarMart クリックスタートがビルドおよびデプロイされている必要があります。

手順15.11 リモートクライアントサーバーモードでの CarMart クイックスタートの表示

- ブラウザーウインドウで次のリンクにアクセスし、アプリケーションを表示します。

```
http://localhost:8080/jboss-carmart
```

バグを報告する

15.6.3. リモートクライアントサーバーモードでの CarMart クイックスタートの削除

次の手順では、Red Hat JBoss Data Grid のリモートクライアントサーバーモードにすでにデプロイされているアプリケーションを削除する方法を説明します。

手順15.12 リモートクライアントサーバーモードでのアプリケーションの削除

- アプリケーションを削除するには、このクイックスタートのルートディレクトリーから次のコマンドを使用します。

```
$ mvn tomcat:undeploy -Premote-tomcat
```

バグを報告する

15.7. JBOSS EAP を用いた (トランザクション) CARMART クイックスタート

この CarMart Transactional クイックスタートには、JBoss Enterprise Application Platform コンテナーを用いる JBoss Data Grid のライブラリーモードが必要になります。

必要なライブラリー (jar ファイル) はすべてアプリケーションにバンドルされ、サーバーにデプロイされます。キャッシュはプログラムを用いて設定され、Web アプリケーションと同じ JVM で実行されます。

すべての操作はトランザクションで、**CacheContainerProvider** インターフェースの **JBossASCacheContainerProvider**/**TomcatCacheContainerProvider** 実装クラスで設定されます。

バグを報告する

15.7.1. クイックスタートの前提条件

このクイックスタートの前提条件は次のとおりです。

- Java 6.0 (Java SDK 1.6) 以上
- JBoss Enterprise Application Platform 6.x または JBoss Enterprise Web Server 2.x
- Maven 3.0 以上
- Maven リポジトリーを設定します。詳細は3章*Maven リポジトリーのインストールおよび使用*を参照してください。

バグを報告する

15.7.2. トランザクション CarMart クイックスタートのビルドおよびデプロイ

前提条件

CarMart クイックスタートをビルドおよびデプロイする前に以下の前提条件を満たしていることを確認してください。

1. Maven の設定 (「クイックスタートの前提条件」を参照してください)
2. JBoss Enterprise Application Platform の起動
    1. コマンドラインターミナルで、JBoss EAP サーバーディレクトリーのルートへ移動します。
    2. 以下のコマンドの 1 つを使用し、Web プロファイルでサーバーを起動します。

        Linux の場合:

        ```
        $JBOSS_HOME/bin/standalone.sh
        ```

        Windows の場合:

        ```
        %JBOSS_HOME%\bin\standalone.bat
        ```

手順15.13 トランザクションクイックスタートのビルドおよびデプロイ

1. コマンドラインターミナルを開き、このクイックスタートのルートディレクトリーへ移動します。
2. 以下のコマンドを入力し、アーカイブをビルドおよびデプロイします。

    ```
    mvn clean package jboss-as:deploy
    ```

3. **target/jboss-carmart-tx.war** ファイルが稼働中のサーバーインスタンスへデプロイされます。

バグを報告する

15.7.3. トランザクション CarMart クイックスタートの表示

次の手順では、 CarMart クイックスタートを表示する方法を説明します。

前提条件

表示する CarMart クリックスタートがビルドおよびデプロイされている必要があります。

手順15.14 CarMart クイックスタートの表示

- アプリケーションを表示するには、ブラウザーを使用して次のリンクに移動します。

```
http://localhost:8080/jboss-carmart-tx
```

バグを報告する

15.7.4. トランザクション CarMart クイックスタートのアンデプロイ

次のように、トランザクション CarMart クイックスタートをアンデプロイします。

1. コマンドラインターミナルで、クイックスタートのルートディレクトリーへ移動します。
2. 次のようにアーカイブをアンデプロイします。

```
mvn jboss-as:undeploy
```

バグを報告する

15.7.5. トランザクション CarMart クイックスタートのテスト

JBoss Data Grid クイックスタートには、Arquillian Selenium テストが含まれています。これらのテストを実行するには、以下を行います。

1. JBoss EAP が実行されている場合は停止します。
2. コマンドラインターミナルで、クイックスタートのルートディレクトリーへ移動します。
3. 次のように、クイックスタートをビルドします。

```
mvn clean package
```

4. 以下のとおり、テストを実行します。

```
mvn test -Puitests-jbossas -Das7home=/path/to/server
```

バグを報告する

15.8. JBOSS ENTERPRISE WEB SERVER を用いた (トランザクション) CARMART クイックスタート

この CarMart Transactional クイックスタートには、JBoss Enterprise Web Server コンテナーを用いる JBoss Data Grid のライブラリーモードが必要になります。

必要なライブラリー (jar ファイル) はすべてアプリケーションにバンドルされ、サーバーにデプロイされます。キャッシュはプログラムを用いて設定され、このクイックスタートの Web アプリケーションと同じ JVM で実行されます。

さらに、すべての操作はトランザクションです。JBoss Enterprise Web Server でアプリケーションを実行するため、JBoss Transactions のスタンドアロントランザクションマネージャーが使用されます。

JBoss Enterprise Web Server を用いてこのクイックスタートを実行する場合、ライブラリーモードを有効にする **library-tomcat** プロファイルのみを使用できます。

バグを報告する

15.8.1. クイックスタートの前提条件

このクイックスタートの前提条件は次のとおりです。

- Java 6.0 (Java SDK 1.6) 以上
- JBoss Enterprise Application Platform 6.x または JBoss Enterprise Web Server 2.x
- Maven 3.0 以上
- Maven リポジトリーを設定します。詳細は3章*Maven リポジトリーのインストールおよび使用*を参照してください。

バグを報告する

15.8.2. トランザクション CarMart クイックスタートのビルドおよびデプロイ

前提条件

CarMart クイックスタートをビルドおよびデプロイする前に以下の前提条件を満たしていることを確認してください。

1. Maven の設定 (「クイックスタートの前提条件」を参照してください)
2. JBoss Enterprise Web Server を設定するには、以下の行を **conf/tomcat-users.xml** ファイルに追加します。

```
<role rolename="manager-script"/>
    <user username="admin" password="admin" roles="manager-script"/>
```

3. 適切なクレデンシャルを使用して以下の設定情報を Maven の **settings.xml** ファイルに追加し、Maven を設定します。

```
<server>
 <id>tomcat</id>
 <username>admin</username>
 <password>admin</password>
</server>
```

4. JBoss Enterprise Web Server を起動します。
    1. コマンドラインターミナルで、JBoss Enterprise Web Server ディレクトリーのルートへ移動します。
    2. 以下のコマンドの 1 つを使用し、Web プロファイルでサーバーを起動します。

        Linux の場合:

```
$TOMCAT_HOME/bin/catalina.sh run
```

Windows の場合:

```
%TOMCAT_HOME%\bin\catalina.bat run
```

手順15.15 トランザクション CarMart クイックスタートのビルドおよびデプロイ

1. コマンドラインターミナルで、クイックスタートのルートディレクトリーへ移動します。
2. 以下のコマンドを入力し、アーカイブをビルドおよびデプロイします。

```
mvn -Plibrary-tomcat clean package tomcat:deploy
```

3. **target/jboss-carmart-tx.war** ファイルが稼働中の JBoss Enterprise Web Server インスタンスへデプロイされます。

バグを報告する

15.8.3. トランザクション CarMart クイックスタートの表示

次の手順では、 CarMart クイックスタートを表示する方法を説明します。

前提条件

表示する CarMart クリックスタートがビルドおよびデプロイされている必要があります。

手順15.16 CarMart クイックスタートの表示

- アプリケーションを表示するには、ブラウザーを使用して次のリンクに移動します。

```
http://localhost:8080/jboss-carmart-tx
```

バグを報告する

15.8.4. トランザクション CarMart クイックスタートのアンデプロイ

次のように、トランザクション CarMart クイックスタートをアンデプロイします。

1. コマンドラインターミナルで、クイックスタートのルートディレクトリーへ移動します。
2. 次のようにアーカイブをアンデプロイします。

```
mvn -Plibrary-tomcat tomcat:undeploy
```

バグを報告する

15.8.5. トランザクション CarMart クイックスタートのテスト

JBoss Data Grid クイックスタートには、Arquillian Selenium テストが含まれています。これらのテストを実行するには、以下を行います。

1. アーカイブをアンデプロイします (「トランザクション CarMart クイックスタートのアンデプロイ」を参照してください)
2. JBoss Enterprise Web Server のサーバーが稼動している場合は停止します。
3. コマンドラインターミナルで、クイックスタートのルートディレクトリーへ移動します。
4. 次のように、クイックスタートをビルドします。

```
mvn clean package
```

5. 以下のように、テストを実行します。

```
mvn test -Puitests-jbossas -Das7home=/path/to/server
```

バグを報告する

第16章 FOOTBALL クイックスタートエンドポイントの例

Football アプリケーションは、Red Hat JBoss Data Grid のエンドポイント (Hot Rod、REST、および Memcached) の使用方法を実例を用いて説明する簡単な例になります。各例は、これらプロトコルの 1 つを使用して JBoss Data Grid に接続し、キャッシュからデータをリモートで保存、読み出し、および削除する方法を示しています。

各アプリケーションは、コンソールアプリケーションとする簡単なフットボールチームマネージャーユーティリティの 1 つです。

機能

Football Manager アプリケーションの例で使用できる機能を以下に示します。

- チームの追加
- 選手の追加
- すべてのエンティティーの削除 (チームと選手)
- すべてのチームおよび選手のリスト

場所

JBoss Data Grid の Football クイックスタートは以下の場所にあります。

- **`jboss-datagrid-{VERSION}-quickstarts/rest-endpoint`**
- **`jboss-datagrid-{VERSION}-quickstarts/hotrod-endpoint`**
- **`jboss-datagrid-{VERSION}-quickstarts/memcached-endpoint`**

バグを報告する

16.1. クイックスタートの前提条件

このクイックスタートの前提条件は次のとおりです。

- Java 6.0 (Java SDK 1.6) 以上
- JBoss Enterprise Application Platform 6.x または JBoss Enterprise Web Server 2.x
- Maven 3.0 以上
- Maven リポジトリーを設定します。詳細は3章*Maven リポジトリーのインストールおよび使用*を参照してください。

バグを報告する

16.2. FOOTBALL アプリケーションのビルド

次の手順では、Red Hat JBoss Data Grid の REST、Hot Rod、および memcached エンドポイントの例としてフットボールマネージャーアプリケーションをビルドする方法を説明します。

重要

JBoss Data Grid はデプロイするアプリケーションをサポートしないため、このクイックスタートをデプロイメントとしてインストールすることはできません。

前提条件

この手順の前提条件は次のとおりです。

1. サポートされる最新の JBoss Data Grid リモートクライアントサーバーモードディストリビューションファイルを Red Hat から取得します。
2. JBoss Data Grid および JBoss Enterprise Application Platform Maven リポジトリーがインストールされ、設定されていることを確認します。詳細は *3章Maven リポジトリーのインストールおよび使用* を参照してください。

手順16.1 Football アプリケーションのビルド

1. **設定の追加**
   $JDG_HOME/standalone/configuration/ にある **standalone.xml** ファイルを編集し、データソースおよび infinispan サブシステムの定義を追加します。

   a. データソースに対する次のサブシステム定義を追加します。

```
<subsystem xmlns="urn:jboss:domain:datasources:4.0">

<!-- Define this Datasource with jndi
name java:jboss/datasources/ExampleDS -->

    <datasources>
        <datasource jndi-name="java:jboss/datasources/ExampleDS"
      pool-name="ExampleDS"
      enabled="true"
      use-java-context="true">

            <!-- The connection URL uses H2 Database
  Engine with in-memory database called test -->

            <connection-url>jdbc:h2:mem:test;DB_CLOSE_DELAY=-
1</connection-url>

            <!-- JDBC driver name -->
            <driver>h2</driver>

            <!-- Credentials -->
            <security>
                <user-name>sa</user-name>
                <password>sa</password>
            </security>
        </datasource>

        <!-- Define the JDBC driver called 'h2' -->
        <drivers>
            <driver name="h2" module="com.h2database.h2">
                <xa-datasource-
class>org.h2.jdbcx.JdbcDataSource</xa-datasource-class>
```

```
            </driver>
        </drivers>

    </datasources>
</subsystem>
```

b. infinispan に対する次のサブシステム定義を追加します。

```
<subsystem xmlns="urn:infinispan:server:core:8.0"
      default-cache-container="local">

 <cache-container name="local"
    default-cache="default"
    statistics="true">

         <local-cache name="default"
        start="EAGER"
        statistics="true">

   <locking isolation="NONE"
     acquire-timeout="30000"
     concurrency-level="1000"
     striping="false"/>

   <transaction mode="NONE"/>

  </local-cache>

  <local-cache name="memcachedCache"
        start="EAGER"
        statistics="true">

   <locking isolation="NONE"
     acquire-timeout="30000"
     concurrency-level="1000"
     striping="false"/>
   <transaction mode="NONE"/>

  </local-cache>

         <local-cache name="namedCache"
        start="EAGER"
        statistics="true"/>

         <!-- ADD a local cache called 'teams' -->

  <local-cache name="teams"
               start="EAGER"
                batching="false"
                statistics="true">

            <!-- Disable transactions for this cache -->
   <transaction mode="NONE" />

            <!-- Define the JdbcBinaryStores
   to point to the ExampleDS previously
```

```
    defined -->

    <string-keyed-jdbc-store
     datasource="java:jboss/datasources/ExampleDS"
      passivation="false"
     preload="false"
     purge="false">

                   <!-- Define the database dialect -->
                   <property name="databaseType">H2</property>

                  <!-- specifies information about
         database table/column names
         and data types -->

     <string-keyed-table prefix="JDG">
                       <id-column name="id"
                    type="VARCHAR"/>
        <data-column name="datum"
                     type="BINARY"/>
        <timestamp-column name="version"
                          type="BIGINT"/>
     </string-keyed-table>

    </string-keyed-jdbc-store>

          </local-cache>

          <!-- End of local cache called 'teams' definition -->

  </cache-container>
</subsystem>
```

注記

Hot Rod および REST エンドポイントは **teams** という名前のキャッシュを使用し、memcached エンドポイントはデフォルトで **memcachedCache** を使用します。

2. **REST セキュリティーの無効化**
   デフォルトでは、**standalone.xml** 設定ファイルは **BASIC** 認証で REST エンドポイントを保護します。このクイックスタートは認証を実行できないため、***security-domain*** および ***auth-method*** パラメーターを削除して REST コネクターで REST 認証を無効にする必要があります。REST 認証が無効化された設定は次のようになります。

   ```
   <rest-connector virtual-server="default-host"
                   cache-container="local" />
   ```

   セキュリティーに関する詳細は、JBoss Data Grid 『Developer Guide』 の REST 認証の章を参照してください。

3. **サブモジュール設定ファイルの編集**
   各サブモジュール (**hotrod-endpoint**、 **rest-endpoint**、および **memcached-endpoint**) には設定ファイルが含まれています

(**$JDG_QUICKSTART/src/main/resources/jdg.properties** にあります)。設定ファイルのデフォルト値を編集し、指定の JBoss Data Grid インストールに必要な値を設定します。

4. **ルートディレクトリーへ移動**
   コマンドラインターミナルを開き、このクイックスタートのルートディレクトリーへ移動します。

5. **アプリケーションのビルド**
   次のコマンドを使用して、サンプルアプリケーションのディレクトリーでサンプルアプリケーションをビルドします。

   ```
   mvn clean package
   ```

   これにより、すべての依存関係を 1 つの jar ファイルにバンドルして使いやすくする、Maven のシェードプラグインが使用されます。このファイルには **{PROTOCOL}-endpoint-quickstart.jar** という名前が付けられます。たとえば、REST の場合は**rest-endpoint-quickstart.jar** という名前になります。

6. **JBoss Data Grid の起動**
   次のスクリプトを実行し、JBoss Data Grid を起動します。

   ```
   $JDG_HOME/bin/standalone.sh
   ```

7. **アプリケーションの実行**
   以下のコマンドを実行し、サンプルアプリケーションのディレクトリーでサンプルアプリケーションを実行します。

   ```
   mvn exec:java
   ```

バグを報告する

第17章 RAPID STOCK MARKET クイックスタート

Rapid Stock Market クイックスタートは、JBoss Data Grid の互換性モードが Hot Rod クライアント (データの格納) および REST を使用する HTTP クライアント (データの呼び出し) とどのように動作するかを実演します。このクイックスタートは、JBoss Data Grid のリモートクライアントサーバーモードでのみ使用でき、コンテナーは使用しません。

Rapid Stock Market クイックスタートには、サーバー側とクライアント側のアプリケーションが含まれています。

バグを報告する

17.1. RAPID STOCK MARKET クイックスタートのビルドおよび実行

Rapid Stock Market クイックスタートは、アプリケーションのサーバー側およびクライアント側に以下の設定が必要になります。

手順17.1 Rapid Stock Market クイックスタートのサーバー側の設定

1. **ルートディレクトリーへ移動**
   コマンドラインターミナルを開き、このクイックスタートのルートディレクトリーへ移動します。

2. クライアントとサーバーに共通するクラスを jar ファイルにパッケージ化して、JBoss Data Grid のサーバーモジュールをビルドします。

   ```
   $ mvn clean package -Pprepare-server-module
   ```

   新しい jar ファイルを、サーバーモジュールに似たディレクトリー構造に置きます。

3. サーバーモジュールをサーバーにインストールします。

   a. 準備したモジュールをサーバーへコピーします。

      ```
      $ cp -r target/modules ${JDG_SERVER_HOME}/
      ```

   b. 以下を **modules/system/layers/base/org/infinispan/commons/main/module.xml** ファイルへ追加し、新しいモジュールを **org.infinispan.commons** モジュールの依存関係として追加します。

      ```
      <module name="org.infinispan.quickstart.compatibility.common"/>
      ```

4. アプリケーションをビルドします。

   ```
   $ mvn clean package
   ```

5. 適切な設定ファイルを使用するように JBoss Data Grid を設定します。

   a. 互換性モードのサンプル設定ファイルを、JBoss Data Grid サーバーが見つけて使用できる場所にコピーします。

```
$ cp ${JDG_SERVER_HOME}/docs/examples/configs/standalone-
compatibility-mode.xml
${JDG_SERVER_HOME}/standalone/configuration
```

b. **rest-connector** 要素から **security-domain** および **auth-method** 属性を削除し、REST セキュリティーを無効にします。

6. JBoss Data Grid サーバーを互換性モードで起動します。

```
$ ${JDG_SERVER_HOME}/bin/standalone.sh -c standalone-compatibility-
mode.xml
```

手順17.2 Rapid Stock Market クイックスタートのクライアント側の設定

1. 別のコマンドラインターミナルウインドウで、サーバー側アプリケーションを起動します。

```
$ mvn exec:java -Pclient
```

2. クライアントアプリケーションのヘルプメニューにある手順を使用します。

バグを報告する

第18章 CLUSTER APP クイックスタート

EAP Cluster App クイックスタートは、JBoss Data Grid キャッシュをライブラリーモードで使用する方法を実証します。

以下を実証する 3 つのアプリケーションがあります。

- JBoss EAP クラスターがない状態で JBoss Data Grid のクラスター化キャッシュを作成および使用する方法。
- JBoss EAP クラスターが Infinispan クラスターから独立している設定。
- 1 つの JBoss EAP インスタンスは、異なる JBoss Data Grid クラスターのメンバーである JBoss Data Grid キャッシュを使用できる。
- JBoss Data Grid API を使用したキャッシュ設定のプログラミング。
- 管理アプリケーション App1Cache で行われるファイルベースの設定。
- CDI を使用したキャッシュマネージャーのインジェクト。

各アプリケーションには、同じアプリケーションのステートレス EJB によってアクセスされる埋め込み JBoss Data Grid キャッシュが含まれています。非クラスター化アプリケーションが JBoss Data Grid キャッシュを共有できることを実証するため、AppTwo 以外の JBoss EAP サーバーはクラスター化されていません。また、2 つのキャッシュマネージャーがありますが、どちらも同じ JBoss Data Grid クラスターのメンバーではありません。AdminApp は、これら両方のキャッシュにアクセスし、変更します。AppOne は App1Cache のみを読み取りますが、AppTwo へのクラスター化 EJB 呼び出しを使用して App2Cache から読み取ります。AppTwo はクラスター化 EJB アプリケーションとしてデプロイされ、App2Cache のみを読み出します。

バグを報告する

18.1. EAP CLUSTER APP の前提条件

EAP Cluster App の前提条件は次のとおりです。

- Java 6.0 (Java SDK 1.6) 以上
- JBoss Enterprise Application Platform 6.1 以降
- Maven 3.0 以上
- Maven リポジトリーの設定。詳細は *3章Maven リポジトリーのインストールおよび使用*を参照してください。
- JBoss Data Grid モジュールが JBoss EAP サーバーにインストールされている必要があります。

バグを報告する

18.2. アプリケーションサーバーインスタンスの起動

手順18.1 スタンドアロンモードでのサーバーの設定および起動

1. コピーする JBoss EAP のインスタンスを準備します。

1. **jboss-datagrid-${version}-eap-modules-library.zip** を展開します。
2. モジュールをサーバーモジュールディレクトリーにコピーします。
   - Linux の場合:

     ```
     cp -a jboss-datagrid-${version}-eap-modules-library/modules
     EAP_HOME
     ```

     Windows の場合:

     ```
      xcopy /e/i/f jboss-datagrid-${version}-eap-modules-
     library/modules EAP_HOME\modules
     ```
3. EJB にアクセスするため、ユーザーを各サーバーに追加します。
   - Linux の場合: EAP_HOME/bin/add-user.sh -a -u quickuser -p quick-123
   - Windows の場合: EAP_HOME\bin\add-user.bat -a -u quickuser -p quick-123

2. 準備した JBoss EAP サーバーを EAP_HOME[1-4] というラベルの 4 つディレクトリーにそれぞれコピーします。
3. ノードごとに 1 つのコマンドラインターミナルを開き、JBoss EAP サーバーディレクトリーのルートに移動します。
4. 以下のコマンドを実行して、4 つの JBoss EAP サーバーすべてを起動します。
   - Linux の場合:

     ```
     EAP_HOME1/bin/standalone.sh -Djboss.node.name=node1
     EAP_HOME2/bin/standalone.sh -Djboss.node.name=node2 -
     Djboss.socket.binding.port-offset=100
     EAP_HOME3/bin/standalone.sh -Djboss.node.name=node3 -
     Djboss.socket.binding.port-offset=200 -c standalone-ha.xml
     EAP_HOME4/bin/standalone.sh -Djboss.node.name=node4 -
     Djboss.socket.binding.port-offset=300 -c standalone-ha.xml
     ```
   - Windows の場合:

     ```
     EAP_HOME1\bin\standalone.bat -Djboss.node.name=node1
     EAP_HOME2\bin\standalone.bat -Djboss.node.name=node2 -
     Djboss.socket.binding.port-offset=100
     EAP_HOME3\bin\standalone.bat -Djboss.node.name=node3 -
     Djboss.socket.binding.port-offset=200 -c standalone-ha.xml
     EAP_HOME4\bin\standalone.bat -Djboss.node.name=node4 -
     Djboss.socket.binding.port-offset=300 -c standalone-ha.xml
     ```
5. EJB のサーバー対サーバーの呼び出しを使用するよう、node2 (AppOne) の設定を追加します。
   - Linux の場合:

     ```
     EAP_HOME2/bin/jboss-cli.sh -c --controller=localhost:10099 --
     file=QUICKSTART_HOME/install-appOne-standalone.cli
     ```

Windows の場合:

```
EAP_HOME2\bin\jboss-cli.bat -c --controller=localhost:10099 --
file=QUICKSTART_HOME/install-appOne-standalone.cli
```

手順18.2 ドメインモードでのサーバーの設定および起動

1. JBoss EAP のフレッシュインストールを EAP_HOME にコピーします。
    1. **jboss-datagrid-${version}-eap-modules-library.zip** を展開します。
    2. モジュールをサーバーモジュールディレクトリーにコピーします。
        - Linux の場合:

        ```
        cp -a jboss-datagrid-${version}-eap-modules-library/modules
        EAP_HOME
        ```

        - Windows の場合:

        ```
        xcopy /e/i/f jboss-datagrid-${version}-eap-modules-
        library/modules EAP_HOME\modules
        ```

2. コマンドラインターミナルを開き、JBoss EAP のルートへ移動します。
3. ユーザーを追加します。
    - Linux の場合:

    ```
    EAP_HOME/bin/add-user.sh -a -u quickuser -p quick-123
    ```

    - Windows の場合:

    ```
    EAP_HOME\bin\add-user.bat -a -u quickuser -p quick-123
    ```

4. 以下はドメインを起動するコマンドになります。
    - Linux の場合:

    ```
    EAP_HOME/bin/domain.sh
    ```

    - Windows の場合:

    ```
    EAP_HOME\bin\domain.bat
    ```

5. クイックスタートの設定を適用します。ドメインには 4 つのノードが含まれるようになります。
    - Linux の場合:

    ```
    EAP_HOME/bin/jboss-cli.sh -c --file=QUICKSTART_HOME/install-
    domain.cli
    ```

    - Windows の場合:

```
EAP_HOME\bin\jboss-cli.bat -c --file=QUICKSTART_HOME/install-
domain.cli
```

バグを報告する

18.3. アプリケーションのビルド

手順18.3 アプリケーションのビルド

1. コマンドラインターミナルを開き、このクイックスタートのルートディレクトリーへ移動します。
2. 以下のコマンドを実行し、アーカイブをビルドします。

```
mvn clean install
```

3. アプリケーションを適切なサーバーへコピーします。
   - Linux の場合:

```
cp adminApp/ear/target/jboss-eap-application-adminApp.ear
EAP_HOME1/standalone/deployments
cp appOne/ear/target/jboss-eap-application-AppOne.ear
EAP_HOME2/standalone/deployments
cp appTwo/ear/target/jboss-eap-application-AppTwo.ear
EAP_HOME3/standalone/deployments
cp appTwo/ear/target/jboss-eap-application-AppTwo.ear
EAP_HOME4/standalone/deployments
```

   - Windows の場合:

```
copy adminApp\ear\target\jboss-eap-application-adminApp.ear
EAP_HOME1\standalone\deployments
copy appOne\ear\target\jboss-eap-application-AppOne.ear
EAP_HOME2\standalone\deployments
copy appTwo\ear\target\jboss-eap-application-AppTwo.ear
EAP_HOME3\standalone\deployments
copy appTwo\ear\target\jboss-eap-application-AppTwo.ear
EAP_HOME4\standalone\deployments
```

4. ドメインモードが使用されている場合は、以下の方法でアプリケーションをデプロイします。
   - Linux の場合:

```
EAP_HOME/bin/jboss-cli.sh -c --file=QUICKSTART_HOME/deploy-
domain.cli
```

   - Windows の場合:

```
EAP_HOME\bin\jboss-cli.bat -c --file=QUICKSTART_HOME/deploy-
domain.cli
```

バグを報告する

18.4. アプリケーションの実行

ビルド後、含まれるクラスを使用してデプロイされたアプリケーションにアクセスできます。

AdminClient の使用

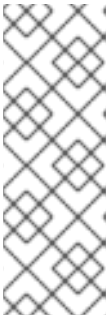
この例では、**AdminServer** および **AppOneServer** のいずれも JBoss EAP レベルではクラスター化されません。JBoss Data Grid インスタンスのみがアプリケーションによる設定どおりにクラスター化されます。このアプリケーションによって以下が実行されます。

- **AdminApp** を使用して **App1** キャッシュに値を追加し、**AppOne** のサーバーインスタンスへレプリケートされることを検証します。
- **App2** キャッシュへ値を追加した後、トランザクションをロールバックし、値がロールバック後にキャッシュへ追加されないことを確認します。

手順18.4 AdminClient アプリケーションの実行

1. コマンドラインターミナルを開き、**$QUICKSTART_HOME/client/** ディレクトリーへ移動します。
2. 以下のコマンドを実行します。

```
mvn -
Dexec.mainClass=org.jboss.as.quickstarts.datagrid.eap.app.AdminClien
t exec:java
```



注記

デフォルトでは、アプリケーションは **localhost:4447** および **localhost:4547** (**AdminHost** および **AppOneHost** 用) へアクセスしようとします。この動作を変更するには、以下のパラメーターを **mvn** コマンドに追加します。

```
-Dexec.args="AdminHost AdminPort AppOneHost AppOnePort"
```

3. 以下のテキストがコンソールに表示されることを確認します。予期せぬ結果の場合は例外が発生します。

```
Add a value to App1Cache with the AdminApp and check on the same
instance that the value is correct added
    success
Check the previous added value of App1Cache by accessing the AppOne
Server
    success
Add a value to App2Cache and check on the same instance that the
value is correct added
    success
Check whether changes to a cache are rollbacked if the transaction
fail
    The cache App2 work as expected on rollback
```

AppOneClient の使用

この例では、JBoss EAP および JBoss Data Grid キャッシュの両方がクラスター化されますが、この例の目的はクラスターはそれぞれ独立していることを実証することです。JBoss Data Grid クラスターは JBossEAP サーバーとは異なる JGroups 実装を使用できます。この例は以下を行います。

- **AdminApp** を使用して **App2** キャッシュへ値を追加します。**AppOne** にアクセスして、EJB 呼び出しがクラスター化され、両方の **AppTwo** インスタンスが使用されることを実証します。

手順18.5 AppOneClient の実行

1. コマンドラインターミナルを開き、**$QUICKSTART_HOME/client/** ディレクトリーへ移動します。
2. 以下のコマンドを実行します。

```
mvn -
Dexec.mainClass=org.jboss.as.quickstarts.datagrid.eap.app.AppOneClie
nt exec:java
```

3. 以下のテキストがコンソールに表示されることを確認します。予期せぬ結果の場合は例外が発生します。

```
Add a value to App2Cache with the AdminApp
Access the App2Cache from the AppOneServer by using the clustered
EJB@AppTwoServer
    success : received the following node names for EJB invocation :
[node3, node4]
```

バグを報告する

18.5. アプリケーションのデバッグ

クイックスタートまたはその関連するライブラリーのソースコードをデバッグまたは確認するには、以下のコマンドのどちらかを実行し、ローカルリポジトリーへプルします。

```
mvn dependency:sources
mvn dependency:resolve -Dclassifier=javadoc
```

バグを報告する

第19章 CAMEL-JBOSSDATAGRID-FUSE クイックスタート

このクイックスタートでは、JBoss Fuse 上で「camel-jbossdatagrid コンポーネント」に記載されているコンポーネントを使用し、JBoss Data Grid と対話する方法を実証します。

このクイックスタートでは、**child1** コンテナーにある**local_cache_producer** バンドルおよび**child2** コンテナーにある**local_cache_consumer** バンドルの 2 つのバンドルを Fuse にデプロイします。各バンドルの詳細は次のとおりです。

- **local_cache_producer**:「id, firstName, lastName, age」の形式を取る受信 CSV ファイルのフォルダー (/tmp/incoming) をスキャンします。ファイルが指定形式のエントリーとドロップされた場合、各エントリーが読み取られ、Person POJO に変換された後、データグリッドに保存されます。
- **local_cache_consumer**:RESTful インターフェースを使用して POJO をクエリーできるようにし、指定のキーのデータグリッドに保存された Person POJO の JSON 表現を受け取れるようにします。

バンドルは 2 つの異なるコンテナーにあります。**infinispan.xml** および**jgroups.xml** ファイルで同じ設定が使用されるため、コンシューマーはプロデューサーによって追加されたものを抽出できます。**infinispan.xml** ファイルは **camel-cache** という名前の**REPL** (レプリケートされた) キャッシュを定義し、コンシューマーとプロデューサーの両方がこのキャッシュと対話します。

バグを報告する

19.1. クイックスタートの前提条件

このクイックスタートの前提条件は次のとおりです。

- Java 7.0 (Java SDK 1.7) 以上
- Maven 3.0 以上
- JBoss Fuse 6.2.0 以上
- Maven リポジトリーを設定します。詳細は3章*Maven リポジトリーのインストールおよび使用*を参照してください。

バグを報告する

19.2. 設定

1. Red Hat カスタマーポータルより、Fuse フルインストールバイナリーをダウンロードします。
2. 以下のコマンドを実行し、CSV ファイルが置かれるフォルダーへのパスをエクスポートします。

```
export incomingFolderPath=[Full path to the CSV folder]
```

3. クイックスタートのルートディレクトリーで以下のコマンドを実行します。

```
mvn clean install -DincomingFolderPath=$incomingFolderPath
```

4. 同じシェルで **FUSE_INSTALL_PATH** および **FUSE_BINARY_PATH** 変数を設定します。

```
export FUSE_INSTALL_PATH = [Full path to the folder where Fuse will
be installed]
export FUSE_BINARY_PATH = [Full path to the Fuse zip file downloaded
in step 1]
```

5. JBoss Fuse 6.2.1 で camel-jbossdatagrid-fuse クイックスタートを実行する場合、**setupEverythingOnFuse.sh** スクリプトに以下の変更を実行する必要があります。変更しない場合は、次の手順に進みます。

   1. 6.2.1 コンポーネントを参照するようにエクスポートされる Fuse のバージョンを変更します。

   2. コンテナー名を **child1** に更新します。

```
# Original line with old version
# export FUSE_VERSION=jboss-fuse-6.2.0.redhat-133
# Updated line for Fuse 6.2.1:
export FUSE_VERSION=jboss-fuse-6.2.1.redhat-084

[...]

# Original line exporting the profile
#sh client -r 2 -d 10 "fabric:container-add-profile child demo-
local_producer" > /dev/null 2>&1
# Updated line for Fuse 6.2.1:
sh client -r 2 -d 10 "fabric:container-add-profile child1 demo-
local_producer" > /dev/null 2>&1
```

6. 環境変数の設定後、クイックスタートのルートディレクトリーから以下のコマンドを実行します。

```
./setupEverythingOnFuse.sh
```

7. スクリプトの完了後、エラーなしで Fuse Hawtio Console へアクセスできることを確認します。このコンソールは、デフォルトでは **http://127.0.0.1:8181/hawtio/index.html#/login** で実行され、ユーザー名とパスワードは共に **admin** になります。

8. **http://127.0.0.1:8181/hawtio/index.html#/fabric/containers** の Fuse Fabric にアクセスし、 **child1** および **child2** コンテナーの両方が作成されたことを確認します。両方のコンテナーは、準備ができたことを示す緑色で強調表示される必要があります。

バグを報告する

19.3. CAMEL-JBOSSDATAGRID-FUSE クイックスタートのテスト

local_cache_producer をテストするには、前に指定した**incomingFolderPath** で CSV ファイルを作成します。以下のコマンドを実行すると、単一のエントリーを持つファイルが作成されます。

```
echo "1,Bill,Gates,59" > $incomingFolderPath/sample.csv
```

プロデューサーによって正常にファイルが解析されると、ディレクトリーからファイルが削除されます。コンシューマーのテストに進みます。

local_cache_consumer をテストするには、Web ブラウザーで **http://127.0.0.1:8282/cache/get/1** に移動します。これにより、上記で指定した**1** を ID として持つエントリーのキャッシュがクエリーされます。作成された POJO の以下の JSON が返されるはずです。

```
{"id":1,"firstName":"Bill","lastName":"Gates","age":59}
```

バグを報告する

パート VII. RED HAT JBOSS DATA GRID のアンインストール

第20章 RED HAT JBOSS DATA GRID の削除

20.1. LINUX システムから RED HAT JBOSS DATA GRID を削除

以下の手順には、Linux システムから Red Hat JBoss Data Grid を削除する手順が含まれます。

警告

JBoss Data Grid を削除すると、すべての設定が永久に失われます。

手順20.1 Linux システムから JBoss Data Grid を削除

1. **サーバーのシャットダウン**
   JBoss Data Grid サーバーがシャットダウンしたことを確認します。
2. **JBoss Data Grid のホームディレクトリーへの移動**
   コマンドラインを使用して、**$JDG_HOME** フォルダーより 1 つ上の階層へ移動します。
3. **JBoss Data Grid のホームディレクトリーの削除**
   ターミナルで以下のコマンドを入力し、JBoss Data Grid を削除します。**$JDG_HOME** は JBoss Data Grid ホームディレクトリーの名前に置き換えます。

   ```
   $ rm -Rf $JDG_HOME
   ```

バグを報告する

20.2. WINDOWS システムから RED HAT JBOSS DATA GRID を削除

以下の手順には、Microsoft Windows システムから Red Hat JBoss Data Grid を削除する手順が含まれます。

警告

JBoss Data Grid を削除すると、すべての設定が永久に失われます。

手順20.2 Windows システムから JBoss Data Grid を削除

1. **サーバーのシャットダウン**
   JBoss Data Grid サーバーがシャットダウンしたことを確認します。
2. **JBoss Data Grid のホームディレクトリーへの移動**

Windows Explorer を使用して、**$JDG_HOME** フォルダーが存在するディレクトリーへ移動します。

3. **JBoss Data Grid のホームディレクトリーの削除**
$JDG_HOME フォルダーを選択し、削除します。

バグを報告する

付録A 参考資料

A.1. キーバリューペアについて

キーバリューペア (KVP) とは、キーと値で構成されるデータセットのことです。

- キーは特定のデータエントリーに一意です。関連するエントリーのエントリーデータ属性から構成されます。
- バリュー (値) は、キーによって割り当てられ、キーによって識別されるデータです。

バグを報告する

付録B MAVEN の設定情報

B.1. NUXUS を用いた JBOSS ENTERPRISE APPLICATION PLATFORM リポジトリーのインストール

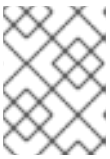
この例では、Sonatype Nexus Maven Repository Manager を使用して JBoss Enterprise Application Platform 6 Maven リポジトリーをインストールする手順を説明します。手順の詳細は http://www.sonatype.org/nexus/ を参照してください。

手順B.1 JBoss Enterprise Application Platform 6 の Maven リポジトリー ZIP アーカイブのダウンロード

1. Web ブラウザーを開き、URL https://access.redhat.com/jbossnetwork/restricted/listSoftware.html?product=appplatform にアクセスします。
2. リストに **Application Platform 6 Maven Repository** があることを確認します。
3. **Download** をクリックし、リポジトリーが含まれる ZIP ファイルをダウンロードします。
4. 希望のディレクトリーにファイルを展開します。

手順B.2 Nexus Maven リポジトリーマネージャーを使用した JBoss Enterprise Application Platform 6 Maven リポジトリーの追加

1. 管理者として Nexus にログインします。
2. リポジトリーマネージャーの左側にある **Views → Repositories** メニューより **Repositories** セクションを選択します。
3. **Add...** ドロップダウンメニューをクリックし、**Hosted Repository** を選択します。
4. 新しいリポジトリーの名前と ID を指定します。
5. **Override Local Storage Location** フィールドに展開されていないリポジトリーのパスを入力します。
6. リポジトリーグループでアーティファクトを使用できる必要がある場合は設定を続行します。必要がない場合はこの手順を続行しないでください。
7. リポジトリーグループを選択します。
8. **Configure** タブをクリックします。
9. **Available Repositories** リストにある新しい JBoss Maven リポジトリーを左側の **Ordered Group Repositories** へドラッグします。



注記

このリストの順番により Maven アーティファクトの検索優先度が決定されます。

結果:

Nexus Maven リポジトリーマネージャーを使用してリポジトリーが設定されます。

バグを報告する

B.2. MAVEN リポジトリーの設定例

展開後、Maven リポジトリーフォルダーのルートディレクトリーにある **example-settings.xml** という名前のサンプル Maven リポジトリーファイルを利用できます。以下は、**example-settings.xml** ファイルに関連する箇所が含まれる抜粋になります。

例B.1 Maven リポジトリーの設定例

```
<?xml version="1.0" encoding="UTF-8"?>

<settings xmlns="http://maven.apache.org/SETTINGS/1.0.0"
xmlns:xsi="http://www.w3.org/2001/XMLSchema-instance"
xsi:schemaLocation="http://maven.apache.org/SETTINGS/1.0.0
http://maven.apache.org/xsd/settings-1.0.0.xsd">

   <profiles>
      <profile>
         <id>jboss-datagrid-repository</id>
         <repositories>
            <repository>
               <id>jboss-datagrid-repository</id>
               <name>JBoss Data Grid Maven Repository</name>
               <url>JDG_REPOSITORY_URL</url>
               <layout>default</layout>
               <releases>
                  <enabled>true</enabled>
                  <updatePolicy>never</updatePolicy>
               </releases>
               <snapshots>
                  <enabled>false</enabled>
                  <updatePolicy>never</updatePolicy>
               </snapshots>
            </repository>
         <pluginRepositories>
            <pluginRepository>
               <id>jboss-datagrid-repository</id>
               <name>JBoss Data Grid Maven Repository</name>
               <url>JDG_REPOSITORY_URL</url>
               <layout>default</layout>
               <releases>
                  <enabled>true</enabled>
                  <updatePolicy>never</updatePolicy>
               </releases>
               <snapshots>
                  <enabled>false</enabled>
                  <updatePolicy>never</updatePolicy>
               </snapshots>
            </pluginRepository>
         </pluginRepositories>
      </profile>
   </profiles>
   <activeProfiles>
      <!-- make the profile active by default -->
```

```
        <activeProfile>jboss-datagrid-repository</activeProfile>
    </activeProfiles>

</settings>
```

「JBoss Data Grid リポジトリーの URL の判別」の説明に従って **JDG_REPOSITORY_URL** を確認できます。

バグを報告する

B.3. JBOSS DATA GRID リポジトリーの URL の判別

リポジトリーの URL は、リポジトリーのある場所によって異なります。以下のいずれかのリポジトリーの場所を使用するよう Maven を設定できます。

- オンラインの JBoss Data Grid Maven リポジトリーを使用するには、URL (https://maven.repository.redhat.com/ga/) を指定します。
- ローカルファイルシステムにインストールされた JBoss Data Grid Maven リポジトリーを使用するには、リポジトリーをダウンロードし、URL のローカルファイルパスを使用する必要があります (例: **file:///path/to/repo/jboss-datagrid-7.0.0-maven-repository/maven-repository/**)。
- Nexus リポジトリーマネージャーを使用して JBoss Data Grid Maven リポジトリーをインストールする場合、URL は https://intranet.acme.com/nexus/content/repositories/jboss-datagrid-7.0.0-maven-repository/maven-repository/ のようになります。

バグを報告する

付録C 改訂履歴

改訂 7.0.0-2 **Mon 18 Jul 2016** **Christian Huffman**

ドメインモードの説明を組み込む。

GA リリース向けのバージョンを更新。

改訂 7.0.0-0 **Tue 19 Apr 2016** **Christian Huffman**

7.0.0 の初回ドラフト。