Red Hat JBoss Data Grid 7.0

管理および設定ガイド

Red Hat JBoss Data Grid 7.0 向け

Last Updated: 2017-11-21

Red Hat JBoss Data Grid 7.0 管理および設定ガイド

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 の管理および設定について説明します。

目次

第1章 RED HAT JBOSS DATA GRID のセットアップ

1.1. 前提条件

Red Hat JBoss Data Grid のセットアップには、Java 仮想マシンのみが必要となり、この製品のサポートされている最新バージョンがシステム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

1.2. RED HAT JBOSS DATA GRID のセットアップ手順

以下に、Red Hat JBoss Data Grid の初回に行う基本設定に必要な手順 (および指定がある場合はオプションの手順) を示します。オプションの手順であると指定されていない限り、これらの手順を指定される順番で実行することをお勧めします。

手順1.1 JBoss Data Grid のセットアップ

1. **キャッシュマネージャーのセットアップ**
   JBoss Data Grid の設定手順は、キャッシュマネージャーから始まります。キャッシュマネージャーにより、事前に設定された設定テンプレートを使ってキャッシュインスタンスをすばやくかつ簡単に取得し、作成することができます。キャッシュマネージャーのセットアップについてさらに詳しくは、JBoss Data Grid の『スタートガイド』の**キャッシュマネージャー**のセクションを参照してください。

2. **JVM メモリー管理のセットアップ**
   JBoss Data Grid の設定における重要な手順は、お使いの Java 仮想マシン (JVM) のメモリー管理のセットアップです。JBoss Data Grid は、JVM メモリーの管理に役立つ、エビクションおよびエクスパレーションなどの各種機能を提供します。

   a. **エビクションのセットアップ**
      エビクションを使用し、エントリーが使用される頻度に基づいてインメモリーキャッシュの実装からエントリーを削除するために使用するロジックを指定します。JBoss Data Grid は、データグリッド内のエントリーのエビクションに対する詳細な制御を実行するための複数の異なるエビクションストラテジーを提供します。エビクションのストラテジーおよびエビクションを設定する手順については、*2章エビクションのセットアップ*を参照してください。

   b. **エクスパレーションのセットアップ**
      キャッシュにおけるエントリーの時間の上限を設定するために、エクスパレーション情報を各エントリーに添付します。エクスパレーションを使用して、エントリーがキャッシュ内に留まる最長期間や、取得されたエントリーがキャッシュから削除される前にアイドル状態として有効となる期間をセットアップします。さらに詳しくは、*3章エクスパレーションのセットアップ*を参照してください。

3. **キャッシュのモニタリング**
   JBoss Data Grid では、JBoss ロギングによるロギング機能を使用し、ユーザーのキャッシュをモニタリングする支援を行います。

   a. **ロギングのセットアップ**
      JBoss Data Grid にロギングをセットアップするのは必須ではありませんが、このセットアップを強く推奨します。JBoss Data Grid は JBoss ロギングを使用します。これにより、ユーザーはデータグリッド内の各種操作に対する自動化されたロギングを簡単にセット

アップできます。ログは、その後エラーのトラブルシューティングや予想外の失敗の原因を特定するために使用することができます。さらに詳しくは、*4章ロギングのセットアップ*を参照してください。

4. **キャッシュモードのセットアップ**
   キャッシュモードは、キャッシュがローカル (単純なインメモリーキャッシュ) か、またはクラスター化されたキャッシュ (ノードの小さなサブセット上で状態変更をレプリケートする) のいずれかを指定するために使用されます。さらに、キャッシュがクラスター化されている場合、変更をノードのサブセットにどのように伝搬させるかを定めるために、レプリケーション、ディストリビューションまたはインバリデーションモードのいずれかを適用する必要があります。さらに詳しくは、パートIII「キャッシュモードのセットアップ」を参照してください。

5. **キャッシュのロックのセットアップ**
   レプリケーションまたはディストリビューションが有効な場合、エントリーのコピーは複数のノードでアクセスできます。結果として、データのコピーは、異なる複数のスレッドで同時にアクセスしたり、変更したりすることができます。複数のノード間ですべてのコピーの一貫性を維持するには、ロックを設定します。さらに詳しくは、パートVI「キャッシュのロックのセットアップ」および*17章分離レベルのセットアップ*を参照してください。

6. **キャッシュストアのセットアップと設定**
   JBoss Data Grid は、メモリーから削除されたエントリーを永続外部キャッシュストアに一時的に保存するために、パッシベーション機能 (またはパッシベーションがオフになっている場合はキャッシュ書き込みストラテジー) を提供します。パッシベーションまたはキャッシュ書き込みストラテジーをセットアップするには、まずキャッシュストアをセットアップする必要があります。

   a. **キャッシュストアのセットアップ**
      キャッシュストアは永続ストアへの接続として機能します。キャッシュストアは、エントリーを永続ストアから取得し、変更を永続ストアに戻すために主に使用されます。さらに詳しくは、パートVII「キャッシュストアのセットアップと設定」を参照してください。

   b. **パッシベーションのセットアップ**
      パッシベーションは、メモリーからエビクトされたエントリーをキャッシュストアに保存します。この機能により、エントリーがメモリー内に存在しない場合でも使用可能な状態となり、永続キャッシュへの高いコストが発生する可能性のある操作を回避できます。さらに詳しくは、パートVIII「パッシベーションのセットアップ」を参照してください。

   c. **キャッシュ書き込みストラテジーのセットアップ**
      パッシベーションが無効な場合、キャッシュへの書き込みが試行されるたびに、キャッシュストアへの書き込みが行なわれます。これは、デフォルトのライトスルーキャッシュ書き込みストラテジーです。これらのキャッシュの書き込みが同期的または非同期的に実行されるかを定めるためにキャッシュライティングストラテジーを設定します。さらに詳しくは、パートIX「キャッシュ書き込みのセットアップ」を参照してください。

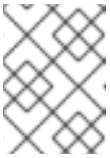
7. **キャッシュとキャッシュマネージャーのモニタリング**
   JBoss Data Grid には、データグリッドが実行中の場合にキャッシュとキャッシュマネージャーをモニタリングするための 3 つの主なツールが含まれます。

   a. **JMX のセットアップ**
      JMX は、JBoss Data Grid に使用される標準的な統計および管理ツールです。ユースケースに応じて、JMX はキャッシュまたはキャッシュマネージャー、またはそれら両方のレベルで設定することができます。さらに詳しくは、*22章 Java Management Extensions (JMX) のセットアップ*を参照してください。

   b. **管理コンソールへのアクセス**
      Red Hat JBoss Data Grid 7.0.0 では、キャッシュおよびキャッシュマネージャーの Web

ベースのモニタリングおよび管理を可能にする管理コンソールを導入しています。使用方法については、「Red Hat JBoss Data Grid 管理コンソールの使用開始」 を参照してください。

c. **Red Hat JBoss Operations Network (JON) のセットアップ**
Red Hat JBoss Operations Network (JON) は、JBoss Data Grid で利用できる 2 番目のモニタリングソリューションです。JBoss Operations Network (JON) は、キャッシュおよびキャッシュマネージャーのランタイムパラメーターおよび統計をモニタリングするためのグラフィカルインターフェースを提供します。さらに詳しくは、*23章JBoss Operations Network (JON) のセットアップ*を参照してください。



注記

JON プラグインは JBoss Data Grid 7.0 で非推奨となり、以降のバージョンでは除去される予定です。

8. **トポロジー情報の導入**
オプションとして、データグリッド内の特定タイプの情報またはオブジェクトが置かれる場所を指定するために、データグリッドにトポロジー情報を提供します。サーバーヒンティングは、トポロジー情報を JBoss Data Grid に導入する方法の内の 1 つです。

a. **サーバーヒンティングのセットアップ**
サーバーヒンティングがセットアップされると、データのオリジナルおよびバックアップコピーが同じ物理サーバー、ラックまたはデータセンターに保存されていないことを確認でき、高可用性が提供されます。これは、すべてのデータがすべてのサーバー、ラックおよびデータセンターでバックアップされるレプリケートキャッシュなどの場合のオプション設定になります。さらに詳しくは、*34章 サーバーヒンティングを用いた高可用性*を参照してください。

後続の章では、JBoss Data Grid の標準設定のセットアップに関する各手順を詳細に説明します。

バグを報告する

パート I. JVM メモリー管理のセットアップ

第2章 エビクションのセットアップ

2.1. エビクションについて

エビクションとは、メモリー不足にならないようにするためにメモリーからエントリーを削除するプロセスのことです。永久的なデータの損失を防ぐために、メモリーからエビクトされたエントリーは設定されたキャッシュストアと残りのクラスターに留まります。キャッシュストアが設定されておらず、エビクションが有効にされている場合、データの損失が発生する可能性があります。

Red Hat JBoss Data Grid は、すでにデータコンテナーと対話しているユーザースレッドを利用してエビクションのタスクを実行します。JBoss Data Grid は別のスレッドを使用して、期限切れのキャッシュエントリーをキャッシュから排除します。

エビクションはクラスター全体の操作として発生せず、ノードごとに個別に発生します。各ノードはエビクションスレッドを使用してインメモリーコンテナーの内容を分析し、エビクションが必要なエントリーを判別します。Java 仮想マシン (JVM) の空きメモリーはエビクションの分析時には考慮されず、エントリーのエビクションを初期化するしきい値としても考慮されません。

JBoss Data Grid では、エビクションは、キャッシュのインメモリー表現からエントリーを効率的に削除し、削除されたエントリーを (設定されている場合) キャッシュストアにプッシュするメカニズムを提供します。これにより、メモリーは新規のエントリーがフェッチされると常にこれらを格納でき、かつエビクトされたエントリーは、失われるのではなくクラスターに保存されます。

さらに、エビクションストラテジーは、エビクトされるエントリーや、エビクションが発生するタイミングをセットアップするための設定で必要になる場合に使用することができます。

関連トピック:

- 「エビクションとエクスパレーションの比較」

バグを報告する

2.2. エビクションストラテジー

各エビクションストラテジーには、以下に記載されているような特定の利点およびユースケースがあります。

表2.1 エビクションストラテジー

ストラテジー名	操作	説明
`EvictionStrategy.NONE`	エビクションは一切発生しません。	これは、Red Hat JBoss Data Grid でのデフォルトのエビクションストラテジーです。
`EvictionStrategy.LRU`	LRU (Least Recently Used) 方式のエビクションストラテジーです。このストラテジーは、最も長い期間にわたって使用されてこなかったエントリーをエビクトします。これにより、定期的に再利用されるエントリーが確実にメモリーに残ります。	

ストラテジー名	操作	説明
`EvictionStrategy.UNORDERED`	順序付けのないエビクションストラテジーです。このストラテジーは、順序付けのあるアルゴリズムを使用せずにエントリーをエビクトするため、後で必要になるエントリーをエビクトする可能性があります。しかし、このストラテジーでは、エビクションの前にアルゴリズムに関連する計算が不要であるため、リソースを節約します。	テストを目的とする場合はこのストラテジーが推奨されますが、実際の作業の実装にあたっては推奨されません。
`EvictionStrategy.LIRS`	LIRS (Low Inter-Reference Recency Set) 方式のエビクションストラテジーです。	LIRS は、さまざまな本番稼働ユースケースに適しているエビクションアルゴリズムです。

バグを報告する

2.2.1. LRU エビクションアルゴリズムの制限

LRU (Least Recently Used) エビクションアルゴリズムでは、最も長い間使用されていないエントリーが最初にエビクトされます。キャッシュから最初にエビクトされるのは、最も長い間アクセスされていないエントリーです。ただし、LRU エビクションアルゴリズムは、アクセスローカリティーが弱い場合には最適に実行されないことがあります。この弱いアクセスローカリティーとは、キャッシュに入れられ、長い間アクセスされていないエントリーについて使用される技術用語であり、エントリーの置き換えは最も早くアクセスされる順番で行われます。このケースでは、次のような問題が発生する可能性があります。

- 1 回限りの使用のためにアクセスされるエントリーが時間内に置き換えられない。
- 最初にアクセスされるエントリーが不必要に置き換えられる。

バグを報告する

2.3. エビクションの使用

Red Hat JBoss Data Grid では、エビクションはデフォルトでは無効にされています。空の ***<eviction />*** 要素を使用して、ストラテジーや最大エントリー数の設定なしにエビクションを有効にすると、次のデフォルト値が使用されます。

- ストラテジー: 指定されたエビクションストラテジーがない場合、***`EvictionStrategy.NONE`*** がデフォルトとみなされます。
- サイズ: 指定された値がない場合、***`size`*** の値は無制限のエントリーを許可する **`-1`** に設定されます。

バグを報告する

2.3.1. エビクションの初期化

エビクションを初期化するには、エビクション要素の ***size*** 属性の値をゼロよりも大きい数に設定します。***size*** に設定された値を調整して、使用する設定に最適な値を探します。***size*** に設定する値が大きすぎると、Red Hat JBoss Data Grid のメモリーが不足することに注意してください。

以下の手順は、JBoss Data Grid でエビクションを初期化する手順を簡単に説明しています。

手順2.1 エビクションの初期化

1. **エビクションタグの追加**
   <eviction> タグを次のようにプロジェクトの **<cache>** タグに追加します。

   ```
   <eviction />
   ```

2. **エビクションストラテジーの設定**
   使用するエビクションストラテジーを設定するために ***strategy*** の値を設定します。使用可能な値は、**LRU**、**UNORDERED** および **LIRS** (またはエビクションが不要な場合は **NONE**) です。以下は、この手順の例になります。

   ```
   <eviction strategy="LRU" />
   ```

3. **エビクションに使用する最大サイズの設定**
   size 要素を定義してメモリー内で許可されるエントリーの最大数を設定します。無制限のエントリーを許可するためのデフォルト値は **-1** です。以下はこの手順について説明しています。

   ```
   <eviction strategy="LRU" size="200" />
   ```

結果

エビクションがターゲットキャッシュ用に設定されます。

バグを報告する

2.3.2. エビクションの設定例

エビクションは Red Hat JBoss Data Grid でプログラムを使用するか、または XML ファイルを使用して設定できます。エビクションの設定はキャッシュごとに実行されます。

XML 設定例は以下のようになります。

```
<eviction strategy="LRU" size="2000"/>
```

バグを報告する

2.3.3. メモリーベースのエビクションの利用

Red Hat JBoss Data Grid 7 では、メモリーベースのエビクションを導入し、エントリー数ではなくエントリーのメモリー使用に基づくエントリーのエビクションが可能になりました。これはエントリーのサイズが異なる場合にとくに役立ちます。

キー/値の制限

メモリーベースのエビクションでは、プリミティブ、プリミティブラッパー (**java.lang.Integer** など)、**java.lang.String** インスタンス、またはこれらの値の**アレイ**として保存されるキーおよび値のみを使用できます。

この制限により、カスタムクラスが使用される場合 ***store-as-binary*** をキャッシュで有効にする必要があります。またはカスタムクラスのデータをシリアライズし、これをバイトアレイに格納することができます。

ただし互換性モードはバイトアレイへのシリアライズを禁止するため、両者は相互に排他的です。

エビクションストラテジーの制限

メモリーベースのエビクションは、**LRU** エビクションストラテジーでのみサポートされます。

メモリーベースのエビクションの有効化

このエビクションメソッドは、以下の例にあるように **MEMORY** をエビクションタイプとして定義することによって使用できます。

```
<local-cache name="local">
    <eviction size="10000000000" strategy="LRU" type="MEMORY"/>
</local-cache>
```

バグを報告する

2.3.4. エビクションとパッシベーション

エントリーの1つのコピーがメモリーまたはキャッシュストアに維持されるようにするため、パッシベーションはエビクションと共に使用してください。

通常のキャッシュストアの代わりにパッシベーションを使用する主な理由は、パッシベーションを使用するとエントリーの更新に必要なリソースが少なくなることにあります。パッシベーションではキャッシュストアへの更新が不要になるためです。

バグを報告する

第3章 エクスパレーションのセットアップ

3.1. エクスパレーションについて

Red Hat JBoss Data Grid は、以下の値のいずれかまたは両方をエントリーに追加するためにエクスパレーションを使用します。

- ライフスパンの値。
- 最大アイドル時間の値。

エクスパレーションはエントリーまたはキャッシュごとに指定でき、エントリーごとの設定は、キャッシュごとの設定よりも優先されます。エクスパレーションがキャッシュレベルで設定される場合、エクスパレーションのデフォルトが ***lifespan*** または ***max-idle*** 値を明示的に指定しないすべてのエントリーに適用されます。

エクスパレーションがキャッシュレベルで設定されない場合、キャッシュエントリーは、デフォルトで期限なし (immortal) (つまり期限切れにならないもの) として作成されます。***lifespan*** または ***max-idle*** が定義されたすべてのエントリーについては、いずれかの条件を満たす場合にキャッシュから最終的に削除されるため、これらには期限が設定されます。

エビクトされたエントリーとは異なり、期限切れのエントリーはグローバルに削除されます。そのため、期限切れのエントリーはメモリー、キャッシュストア、およびクラスターから削除されます。

エクスパレーションは、指定した期間中に使用されなかったエントリーのメモリーからの削除を自動化します。エクスパレーションとエビクションは、以下の点で異なっています。

- エクスパレーションは、エントリーがメモリーに存在していた期間に基づいてエントリーを削除します。エクスパレーションは、ライフスパンの期間が終了するか、またはエントリーが指定したアイドル時間よりも長くアイドル状態になっていた場合のみ、エントリーを削除します。
- エビクションは、エントリーがどの程度最近 (および頻繁) に使用されるかに基づいてエントリーを削除します。エビクションは、メモリーに存在するエントリーが多すぎる場合にエントリーを削除します。キャッシュストアが設定されている場合、エビクトされたエントリーがキャッシュストアで永続化します。

バグを報告する

3.2. エクスパレーションの操作

Red Hat JBoss Data Grid のエクスパレーションによって、キャッシュに格納されたキーバリューペアに対してライフスパンと最大アイドル時間の値を設定することができます。

ライフスパンまたは最大アイドル時間は、キャッシュ API を使用してキャッシュ全体に適用するために設定したり、キーバリューペアごとに定義することができます。キーバリューペアごとに定義されたライフスパン (***lifespan***) または最大アイドル時間 (***max-idle***) は、エントリーのキャッシュ全体のデフォルトよりも優先されます。

バグを報告する

3.3. エビクションとエクスパレーションの比較

エクスパレーションは Red Hat JBoss Data Grid のトップレベルのコンストラクトで、グローバル設定およびキャッシュ API で表されます。

エビクションは使用されるキャッシュインスタンスに限定されますが、エクスパレーションはクラスター全体で有効です。エクスパレーションのライフスパン (***lifespan***) とアイドル時間 (***max-idle***) の値は、各キャッシュエントリーと共にレプリケートされます。

バグを報告する

3.4. キャッシュエントリーのエクスパレーション動作

Red Hat JBoss Data Grid は、タイムアウト直後のエントリーの削除を保証しません。その代わりに、複数のメカニズムを連携して使用し、効率的な削除が確実に行われるようにします。以下のいずれかに該当する場合、期限切れのエントリーがキャッシュから削除されます。

- エントリーがディスクへパッシベートまたはオーバフローされ、期限切れであることが判明した場合。
- エクスパレーションメンテナンススレッドが見つけたエントリーが期限切れであることが判明した場合。

期限が切れているが削除されていないエントリーをユーザーが要求した場合は、null 値がユーザーに送信されます。このメカニズムにより、ユーザーは期限切れのエントリーを受け取らなくなります。このエントリーは結果的にエクスパレーションスレッドにより削除されます。

バグを報告する

3.5. エクスパレーションの設定

Red Hat JBoss Data Grid では、エクスパレーションはエビクションと同様の方法で設定されます。

手順3.1 エクスパレーションの設定

1. **エクスパレーションタグの追加**
   <expiration> タグを次のようにプロジェクトの **<cache>** タグに追加します。

   ```
   <expiration />
   ```

2. **エクスパレーションのライフスパンの設定**
   エントリーがメモリーに留まる時間 (ミリ秒単位) を設定するために ***lifespan*** の値を設定します。以下はこの手順の例になります。

   ```
   <expiration lifespan="1000" />
   ```

3. **最大アイドル時間の設定**
   エントリーが削除された後にアイドル (未使用) の状態のままにすることのできる時間 (ミリ秒単位) を設定します。無制限にするためのデフォルト値は **-1** です。

   ```
   <expiration lifespan="1000" max-idle="1000" />
   ```

バグを報告する

3.6. エクスパレーションのトラブルシューティング

エクスパレーションが機能していないように見える場合、エントリーがエクスパレーション用にマークされていても削除されていないためである可能性があります。

put() のような複数キャッシュの操作では、ライフスパン値がパラメーターとして渡されます。この値は間隔を定義し、この間隔の後にエントリーが期限切れになります。エビクションが設定されていない状態でライフスパンが期限切れになると、Red Hat JBoss Data Grid がエントリーを削除しなかったように表示されます。たとえば、**number of entries** などの JMX の統計が表示される場合、無効の数字が表示されたり、JBoss Data Grid に関連する永続ストアにこのエントリーが依然として含まれていることがあります。この場合、JBoss Data Grid は背後でこのエントリーを期限切れエントリーとしてマーク付けしても、削除していません。このようなエントリーの削除は、以下の場合に行われます。

- エントリーがディスクへパッシベートまたはオーバフローされ、期限切れであることが判明した場合。
- エクスパレーションメンテナンススレッドにより検出されたエントリーが期限切れであることが判明した場合。

期限切れエントリーに対して **get()** または **containsKey()** の使用を試みると、JBoss Data Grid が null 値を返します。期限切れのエントリーはエクスパレーションスレッドによって後に削除されます。

バグを報告する

パート II. キャッシュのモニタリング

第4章 ロギングのセットアップ

4.1. ロギング

Red Hat JBoss Data Grid は、独自の内部使用とデプロイされたアプリケーションによる使用のために設定可能な高度なロギング機能を提供します。ロギングサブシステムは JBoss LogManager を基盤とし、JBoss Logging 以外にも複数のサードパーティーアプリケーションのロギングフレームワークをサポートします。

ロギングサブシステムは、ログカテゴリーとログハンドラーのシステムを使用して設定されます。ログカテゴリーはキャプチャーするメッセージを定義し、ログハンドラーはこれらのメッセージの処理方法を定義します (ディスクへの書き込みやコンソールへの送信など)。

JBoss Data Grid キャッシュは、エビクションおよびエクスパレーションなどの操作と共に設定されると、ロギングは関連アクティビティー (エラーまたは障害を含む) を追跡します。

正しくセットアップされると、ロギングは所定の環境でいつ、何が発生したかについての詳細情報を提供します。さらに、ロギングは環境内でクラッシュまたは問題が起こる直前に生じたアクティビティーの追跡を行うのに役立ちます。この情報は、トラブルシューティングやクラッシュまたはエラーの原因の特定を試行する際に役立ちます。

バグを報告する

4.2. サポート対象のアプリケーションロギングフレームワーク

Red Hat JBoss LogManager は以下のロギングフレームワークに対応しています。

- Red Hat JBoss Data Grid 7 に含まれる JBoss ロギング。
- Apache Commons Logging
- Simple Logging Facade for Java (SLF4J)
- Apache log4j
- Java SE Logging (java.util.logging)

バグを報告する

4.2.1. JBoss ロギングについて

Red Hat JBoss Logging は JBoss Enterprise Application Platform 7 に含まれているアプリケーションロギングフレームワークです。そのため、Red Hat JBoss Data Grid 7 でも JBoss ロギングを使用します。

JBoss ロギングはアプリケーションにロギングを追加する簡単な方法を提供します。フレームワークを使用するアプリケーションにコードを追加し、定義された形式でログメッセージを送信します。アプリケーションサーバーにアプリケーションがデプロイされると、これらのメッセージはサーバーによってキャプチャーされ、サーバーの設定に応じて表示されたり、ファイルに書き込まれたりします。

バグを報告する

4.2.2. JBoss ロギングの機能

JBoss のロギングには次の機能が含まれます。

- 革新的で使いやすい**型指定された**ロガーを提供します。
- 国際化およびローカリゼーションを完全にサポートします。翻訳者は properties ファイルのメッセージバンドルを、開発者はインターフェースやアノテーションを使って作業を行います。
- 実稼働用の型指定されたロガーを生成し、開発用の型指定されたロガーをランタイムに生成する build-time ツールを提供します。

バグを報告する

4.3. ブートロギング

ブートログは、サーバーの起動中 (またはブート中) に発生したイベントの記録です。Red Hat JBoss Data Grid には、サーバーがブート処理を完了した後に生成されるログエントリーが含まれるサーバーログも組み込まれます。

バグを報告する

4.3.1. ブートロギングの設定

ブートログを設定するには、**`logging.properties`** ファイルを編集します。このファイルは標準的な Java プロパティーファイルであるため、テキストエディターで編集することができます。このファイルの各行の形式は **`property=value`** になります。

Red Hat JBoss Data Grid では、**`logging.properties`** ファイルは **`$JDG_HOME/standalone/configuration`** フォルダーにあります。

バグを報告する

4.3.2. デフォルトのログファイルの場所

以下の表は、Red Hat JBoss Data Grid のログファイルとそれらの場所のリストです。

表4.1 デフォルトのログファイルの場所

ログファイル	場所	説明
`boot.log`	**`$JDG_HOME/standalone/log/`**	サーバーブートログ。サーバーの起動に関連するログメッセージが含まれます。 デフォルトでは、このファイルは **`server.log`** の前に追加されます。このファイルは、**`logging.properties`** に **`org.jboss.boot.log`** プロパティーを定義することで、**`server.log`** とは切り離して作成できます。
`server.log`	**`$JDG_HOME/standalone/log/`**	サーバーログ。サーバー起動後のすべてのログメッセージが含まれます。

バグを報告する

4.4. ロギング属性

4.4.1. ログレベルについて

ログレベルとは、ログメッセージの性質と重大度を示す列挙値の順序付けされたセットです。特定ログメッセージのレベルは、そのメッセージを送信するために選択したロギングフレームワークの適切なメソッドを使用して開発者が指定します。

Red Hat JBoss Data Grid は、サポート対象のアプリケーションロギングフレームワークによって使用されるすべてのログレベルをサポートします。最も一般的に使用される 6 つのログレベルは次のとおりです (重大度の低いものから順に記載）。

1. **TRACE**
2. **DEBUG**
3. **INFO**
4. **WARN**
5. **ERROR**
6. **FATAL**

ログレベルはログカテゴリーとログハンドラーによって使用され、それらが対象とするメッセージを限定します。各ログレベルには、他のログレベルとの相対的な順序を示す数値が割り当てられます。ログカテゴリーとハンドラーにはログレベルが割り当てられ、その数値以上のログメッセージのみを処理します。たとえば、**WARN** レベルのログハンドラーは、**WARN**、**ERROR**、および **FATAL** レベルのメッセージのみを記録します。

バグを報告する

4.4.2. サポート対象のログレベル

以下の表は Red Hat JBoss Data Grid でサポートされるログレベルの一覧になります。ログレベル、ログレベルの値、および説明が記載されています。ログレベルの値は、他のログレベルに対する相対的な値になります。フレームワークが異なるとログレベルの名前が異なることがありますが、ログの値はこの一覧と一致します。

表4.2 サポート対象のログレベル

ログレベル	値	説明
FINEST	300	-
FINER	400	-

ログレベル	値	説明
TRACE	400	アプリケーションの実行状態について詳細な情報を提供するメッセージに使用されます。**TRACE** レベルが有効な状態でサーバーが実行されている時に **TRACE** レベルのログメッセージがキャプチャーされます。
DEBUG	500	個々の要求の進捗やアプリケーションのアクティビティーを示すメッセージに使用されます。**DEBUG** レベルが有効な状態でサーバーが実行されている時に **DEBUG** レベルのログメッセージがキャプチャーされます。
FINE	500	-
CONFIG	700	-
INFO	800	アプリケーションの全体的な進捗を示すメッセージに使用されます。アプリケーションの起動やシャットダウン、その他の主なライフサイクルイベントに対して使用されます。
WARN	900	エラーではないが、理想的とは見なされない状況を示すために使用されます。将来的にエラーをもたらす可能性のある状況を示します。
WARNING	900	-
ERROR	1000	発生したエラーの中で、現在のアクティビティーや要求の完了を妨げる可能性があるが、アプリケーション実行の妨げにはならないエラーを示すために使用されます。
SEVERE	1000	-
FATAL	1100	重大なサービス障害を引き起こしたり、アプリケーションをシャットダウンしたり、場合によっては JBoss Data Grid をシャットダウンしたりする可能性があるイベントを示すために使用されます。

バグを報告する

4.4.3. ログカテゴリーについて

ログカテゴリーは、キャプチャーするログメッセージのセットと、メッセージを処理する 1 つまたは複数のログハンドラーを定義します。

キャプチャーするログメッセージは、元の Java パッケージとログレベルによって定義されます。そのパッケージ内のクラスからのメッセージおよびそのログレベル以上の (数値がその値以上の) メッセージがログカテゴリーによってキャプチャーされ、指定のログハンドラーに送信されます。たとえば、**WARNING** ログレベルでは、**900**、**1000**、および **1100** のログの値がキャプチャーされます。

ログカテゴリーは、独自のハンドラーの代わりにルートロガーのログハンドラーを任意で使用することができます。

バグを報告する

4.4.4. ルートロガーについて

ルートロガーは、サーバーに送信された (指定レベルの) ログメッセージの中でログカテゴリーによってキャプチャーされないすべてのログメッセージをキャプチャーします。これらのメッセージは単一または複数のハンドラーに送信されます。

デフォルトでは、ルートロガーはコンソールおよび定期ログハンドラーを使用するように設定されています。定期ログハンドラーは、**server.log** ファイルに書き込むように設定されています。このファイルはサーバーログと呼ばれる場合もあります。

バグを報告する

4.4.5. ログハンドラーについて

ログハンドラーは、キャプチャーされたログメッセージがどのように Red Hat JBoss Data Grid によって記録されるかを定義します。JBoss Data Grid で設定可能な 6 種類のログハンドラーは次のとおりです。

- **Console**
- **File**
- **Periodic**
- **Size**
- **Async**
- **Custom**

ログハンドラーは、指定されたログオブジェクトをさまざまな出力 (コンソールまたは指定されたログファイルを含む) に配信します。JBoss Data Grid で使用される一部のログハンドラーは、他のログハンドラーの動作を送信するために使用されるラッパーログハンドラーです。

ログハンドラーは、ソートを容易にするためにログ出力を特定のファイルに送信したり、特定の間隔でログを書き込むために使用されます。ログハンドラーは主として、必要なログの種類や、それらが保存または表示される場所、または JBoss Data Grid におけるロギング動作を指定するのに役立ちます。

バグを報告する

4.4.6. ログハンドラーのタイプ

以下の表は、Red Hat JBoss Data Grid で利用可能なログハンドラーの各種タイプのリストです。

表4.3 ログハンドラーのタイプ

ログハンドラーのタイプ	説明	ユースケース
コンソール (Console)	コンソールログハンドラーは、ログメッセージをホストオペレーティングシステムの標準出力 (`stdout`) または標準エラー (`stderr`) ストリームに書き込みます。これらのメッセージは、JBoss Data Grid がコマンドラインプロンプトから実行される場合に表示されます。	コンソールログハンドラーは、JBoss Data Grid がコマンドラインを使って管理されている場合に推奨されます。この場合、コンソールログハンドラーからのメッセージは、オペレーティングシステムが標準出力や標準エラーストリームをキャプチャーするように設定されていない限り、保存されません。
ファイル (File)	ファイルログハンドラーは最も単純なログハンドラーです。主に、ログメッセージを指定のファイルへ書き込むために使用されます。	ファイルログハンドラーは、時間に基づいてすべてのログエントリーを 1 つの場所に保存することが要件である場合に最も役に立ちます。
定期 (Periodic)	定期ファイルハンドラーは、指定した時間が経過するまで、ログメッセージを指定ファイルに書き込みます。その時間が経過した後は、指定のタイムスタンプがファイル名に追加されます。その後、ハンドラーは元の名前で新たに作成されたログファイルへの書き込みを継続します。	定期ファイルハンドラーは、環境の要件に応じて、週ごと、日ごと、時間ごと、またはその他の単位ごとにログメッセージを蓄積するために使用することができます。
サイズ (Size)	サイズログハンドラーは、指定のファイルが指定サイズに到達するまで、そのファイルにログメッセージを書き込みます。ファイルが指定のサイズに到達すると、名前に数値のプレフィックスを追加して名前が変更され、ハンドラーは元の名前で新規に作成されたログファイルへの書き込みを継続します。各サイズログハンドラーは、このような方式で保管されるファイルの最大数を指定する必要があります。	サイズハンドラーは、ログファイルのサイズが一致している必要のある環境に最も適しています。

ログハンドラーのタイプ	説明	ユースケース
非同期 (Async)	非同期ログハンドラーは、単一または複数の他のログハンドラーを対象とする非同期動作を提供するラッパーログハンドラーです。非同期ログハンドラーは、待ち時間が長かったり、ネットワークファイルシステムへのログファイルの書き込みなどの他のパフォーマンス上の問題があるログハンドラーに対して有用です。	非同期ログハンドラーは、待ち時間が長いことが問題になる環境や、ネットワークファイルシステムへログファイルを書き込む際に最も適しています。
カスタム (Custom)	カスタムログハンドラーにより、実装されている新たなタイプのログハンドラーを設定することが可能になります。カスタムハンドラーは、**`java.util.logging.Handler`** を拡張する Java クラスとして実装し、モジュール内に格納する必要があります。	カスタムログハンドラーは、ログハンドラーのカスタマイズしたタイプを作成するもので、高度なユーザー用として推奨されます。

バグを報告する

4.4.7. ログハンドラーの選択

以下は、Red Hat JBoss Data Grid で利用可能なログハンドラーのそれぞれのタイプについての最も一般的な使用例です。

- **コンソール (Console)** ログハンドラーは、JBoss Data Grid がコマンドラインを使って管理される場合に推奨されます。このような場合、エラーやログメッセージはコンソールウィンドウに表示され、保存されるように別途設定されない限り保存されません。
- **ファイル (File)** ログハンドラーは、ログエントリーを指定のファイルに送信するために使用されます。この単純なログハンドラーは、時間に基づいてすべてのログエントリーを 1 つの場所に保存することが要件である場合に役に立ちます。
- **定期 (Periodic)** ログハンドラーは、**ファイル (File)** ハンドラーと似ていますが、指定された期間に応じてファイルを作成します。例として、このハンドラーは環境の要件に応じて、週ごと、日ごと、時間ごと、またはその他の単位ごとにログメッセージを蓄積するために使用することができます。
- **サイズ (Size)** ログハンドラーも、指定されたファイルにログメッセージを書き込みますが、ログファイルのサイズが指定の制限内にある場合にのみ、これが実行されます。ファイルサイズが指定したサイズまで達すると、ログファイルは新規のログファイルに書き込まれます。このハンドラーは、ログファイルのサイズに一貫性が必要な環境に最も適しています。
- **非同期 (Async)** ログハンドラーは、他のログハンドラーが非同期に動作するように強制するラッパーです。このログハンドラーは、待ち時間が長いことが問題となる環境や、ネットワークファイルシステムへの書き込み時に最も適しています。
- **カスタム (Custom)** ログハンドラーは、新規のカスタマイズされたタイプのログハンドラーを作成します。これは高度なログハンドラーです。

バグを報告する

4.4.8. ログフォーマッターについて

ログフォーマッターは、ログハンドラーの設定プロパティーで、関連するログハンドラーから発信されるログメッセージの外観を定義します。ログフォーマッターは **java.util.Formatter** クラスと同じ構文を使用する文字列です。

さらに詳しくは、http://docs.oracle.com/javase/6/docs/api/java/util/Formatter.html を参照してください。

バグを報告する

4.5. ロギングの設定例

4.5.1. ロギングの設定例の場所

このセクションで示されるすべての設定例は、通常スタンドアロンインスタンスの場合は **standalone.xml** または **clustered.xml** のいずれか、または管理ドメインインスタンスの場合は **domain.xml** であるサーバーの設定ファイル内にあります。

バグを報告する

4.5.2. ルートロガーの XML 設定例

以下の手順は、ルートロガーの設定例を示しています。

手順4.1 ルートロガーの設定

1. ***level* プロパティーを設定します。**
   level プロパティーは、ルートロガーが記録するログメッセージの最大レベルを設定します。

```
<subsystem xmlns="urn:jboss:domain:logging:3.0">
   <root-logger>
      <level name="INFO"/>
```

2. ***handlers* を一覧表示します。**
   handlers は、ルートロガーによって使用されるログハンドラーの一覧です。

```
<subsystem xmlns="urn:jboss:domain:logging:3.0">
     <root-logger>
        <level name="INFO"/>
        <handlers>
           <handler name="CONSOLE"/>
           <handler name="FILE"/>
        </handlers>
     </root-logger>
  </subsystem>
```

バグを報告する

4.5.3. ログカテゴリーの XML 設定例

以下の手順は、ログカテゴリーの設定例を示しています。

手順4.2 ログカテゴリーの設定

```
<subsystem xmlns="urn:jboss:domain:logging:3.0">
   <logger category="com.company.accounts.rec" use-parent-handlers="true">
      <level name="WARN"/>
      <handlers>
         <handler name="accounts-rec"/>
      </handlers>
   </logger>
</subsystem>
```

1. ログメッセージがキャプチャーされるログカテゴリーを指定するために、***category*** プロパティーを使用します。

   use-parent-handlers はデフォルトで **"true"** に設定されています。**"true"** に設定した場合、このカテゴリーは、割り当てられた他のハンドラーだけでなく、ルートロガーのログハンドラーを使用します。

2. ログカテゴリーが記録するログメッセージの最大レベルを設定するために、***level*** プロパティーを使用します。

3. ***handlers*** 要素には、ログハンドラーのリストが含まれます。

バグを報告する

4.5.4. コンソールログハンドラーの XML 設定例

以下の手順は、コンソールログハンドラーの設定例を示しています。

手順4.3 コンソールログハンドラーの設定

```
<subsystem xmlns="urn:jboss:domain:logging:3.0">
   <console-handler name="CONSOLE" autoflush="true">
      <level name="INFO"/>
      <encoding value="UTF-8"/>
      <target value="System.out"/>
      <filter-spec value="not(match(&quot;JBAS.*&quot;))"/>
      <formatter>
         <pattern-formatter pattern="%K{level}%d{HH:mm:ss,SSS} %-5p [%c]
(%t) %s%E%n"/>
      </formatter>
   </console-handler>
</subsystem>
```

1. **ログハンドラーの ID 情報を追加します。**
   name プロパティーは、このログハンドラーの一意の ID を設定します。

   autoflush を **"true"** に設定すると、ログメッセージは要求直後にハンドラーのターゲットに送信されます。

2. ***level*** **プロパティーを設定します。**
   level プロパティーは、記録されるログメッセージの最大レベルを設定します。

3. ***encoding*** **出力を設定します**
   出力に使用する文字エンコーディングスキームを設定するには、***encoding*** を使用します。

4. ***target*** **値を定義します。**
   target プロパティーは、ログハンドラーの出力先となるシステム出力ストリームを定義します。これはシステムエラーストリームの場合は **System.err**、標準出力ストリームの場合は **System.out** とすることができます。

5. ***filter-spec*** **プロパティーを定義します。**
   filter-spec プロパティーはフィルターを定義する式の値です。以下の例では、**not(match("JBAS.*"))** はパターンに一致しないフィルターを定義します。

6. ***formatter*** **を指定します。**
   このログハンドラーで使用するログフォーマッターの一覧を表示するには、***formatter*** を使用します。

バグを報告する

4.5.5. ファイルログハンドラーの XML 設定例

以下の手順は、ファイルログハンドラーの設定例を示しています。

手順4.4 ファイルログハンドラーの設定

```
<file-handler name="accounts-rec-trail" autoflush="true">
    <level name="INFO"/>
    <encoding value="UTF-8"/>
    <file relative-to="jboss.server.log.dir" path="accounts-rec-
trail.log"/>
    <formatter>
        <pattern-formatter pattern="%d{HH:mm:ss,SSS} %-5p [%c] (%t)
%s%E%n"/>
    </formatter>
    <append value="true"/>
</file-handler>
```

1. **ファイルログハンドラーの ID 情報を追加します。**
   name プロパティーは、このログハンドラーの一意の ID を設定します。

   autoflush を **"true"** に設定すると、ログメッセージは要求直後にハンドラーのターゲットに送信されます。

2. ***level*** **プロパティーを設定します。**
   level プロパティーは、ルートロガーが記録するログメッセージの最大レベルを設定します。

3. ***encoding*** **出力を設定します。**
   出力に使用する文字エンコーディングスキームを設定するには、***encoding*** を使用します。

4. ***file*** **オブジェクトを設定します。**
   file オブジェクトは、このログハンドラーの出力が書き込まれるファイルを表します。***relative-to*** と ***path*** の 2 つの設定プロパティーが含まれます。

relative-to プロパティーは、ログファイルが書き込まれるディレクトリーです。JBoss Enterprise Application Platform 6 のファイルパス変数をここで指定できます。**jboss.server.log.dir** 変数はサーバーの **log/** ディレクトリーを指します。

path プロパティーは、ログメッセージが書き込まれるファイルの名前です。これは、完全パスを決定するために ***relative-to*** プロパティーの値に追加される相対パス名です。

5. ***formatter*** を指定します。
   このログハンドラーで使用するログフォーマッターの一覧を表示するには、***formatter*** を使用します。

6. ***append*** プロパティーを設定します。
   append プロパティーを **"true"** に設定した場合、このハンドラーが追加したすべてのメッセージが既存のファイルに追加されます。**"false"** に設定した場合、アプリケーションサーバーが起動するたびに新規ファイルが作成されます。***append*** への変更を反映させるには、サーバーの再起動が必要です。

バグを報告する

4.5.6. 定期ログハンドラーの XML 設定例

以下の手順は、定期ログハンドラーの設定例を示しています。

手順4.5 定期ログハンドラーの設定

```
<periodic-rotating-file-handler name="FILE" autoflush="true">
   <level name="INFO"/>
   <encoding value="UTF-8"/>
   <formatter>
      <pattern-formatter pattern="%d{HH:mm:ss,SSS} %-5p [%c] (%t)
%s%E%n"/>
   </formatter>
   <file relative-to="jboss.server.log.dir" path="server.log"/>
   <suffix value=".yyyy-MM-dd"/>
   <append value="true"/>
</periodic-rotating-file-handler>
```

1. 定期ログハンドラーの ID 情報を追加します。
   name プロパティーは、このログハンドラーの一意の ID を設定します。

   autoflush を **"true"** に設定すると、ログメッセージは要求直後にハンドラーのターゲットに送信されます。

2. ***level*** プロパティーを設定します。
   level プロパティーは、ルートロガーが記録するログメッセージの最大レベルを設定します。

3. ***encoding*** 出力を設定します。
   出力に使用する文字エンコーディングスキームを設定するには、***encoding*** を使用します。

4. ***formatter*** を指定します。
   このログハンドラーで使用するログフォーマッターの一覧を表示するには、***formatter*** を使用します。

5. ***file*** オブジェクトを設定します。

file オブジェクトは、このログハンドラーの出力が書き込まれるファイルを表します。***relative-to*** と ***path*** の 2 つの設定プロパティーが含まれます。

relative-to プロパティーは、ログファイルが書き込まれるディレクトリーです。JBoss Enterprise Application Platform 6 のファイルパス変数をここで指定できます。**jboss.server.log.dir** 変数はサーバーの **log/** ディレクトリーを指します。

path プロパティーは、ログメッセージが書き込まれるファイルの名前です。これは、完全パスを決定するために ***relative-to*** プロパティーの値に追加される相対パス名です。

6. ***suffix*** **値を設定します。**
   suffix は、ローテーションされたログのファイル名に追加され、ローテーションの周期を決定するために使用されます。***suffix*** の形式では、ドット **(.)** の後に **java.text.SimpleDateFormat** クラスで解析できる日付文字列が指定されます。ログは ***suffix*** で定義された最小時間単位に基づいてローテーションされます。たとえば、**yyyy-MM-dd** の場合は、ログが毎日ローテーションされます。http://docs.oracle.com/javase/6/docs/api/index.html?java/text/SimpleDateFormat.html を参照してください。

7. ***append*** **プロパティーを設定します。**
   append プロパティーを **"true"** に設定した場合、このハンドラーが追加したすべてのメッセージが既存のファイルに追加されます。**"false"** に設定した場合、アプリケーションサーバーが起動するたびに新規ファイルが作成されます。***append*** への変更を反映させるには、サーバーの再起動が必要です。

バグを報告する

4.5.7. サイズログハンドラーの XML 設定例

以下の手順は、サイズログハンドラーの設定例を示しています。

手順4.6 サイズログハンドラーの設定

```
<size-rotating-file-handler name="accounts_debug" autoflush="false">
   <level name="DEBUG"/>
   <encoding value="UTF-8"/>
   <file relative-to="jboss.server.log.dir" path="accounts-debug.log"/>
   <rotate-size value="500k"/>
   <max-backup-index value="5"/>
   <formatter>
        <pattern-formatter pattern="%d{HH:mm:ss,SSS} %-5p [%c] (%t)
%s%E%n"/>
    </formatter>
   <append value="true"/>
</size-rotating-file-handler>
```

1. **サイズログハンドラーの ID 情報を追加します。**
   name プロパティーは、このログハンドラーの一意の ID を設定します。

   autoflush を **"true"** に設定すると、ログメッセージは要求直後にハンドラーのターゲットに送信されます。

2. ***level*** **プロパティーを設定します。**
   level プロパティーは、ルートロガーが記録するログメッセージの最大レベルを設定します。

3. ***encoding*** **出力を設定します。**
   出力に使用する文字エンコーディングスキームを設定するには、***encoding*** を使用します。

4. ***file*** **オブジェクトを設定します。**
   file オブジェクトは、このログハンドラーの出力が書き込まれるファイルを表します。***relative-to*** と ***path*** の 2 つの設定プロパティーが含まれます。

   relative-to プロパティーは、ログファイルが書き込まれるディレクトリーです。JBoss Enterprise Application Platform 6 のファイルパス変数をここで指定できます。**jboss.server.log.dir** 変数はサーバーの **log/** ディレクトリーを指します。

   path プロパティーは、ログメッセージが書き込まれるファイルの名前です。これは、完全パスを決定するために ***relative-to*** プロパティーの値に追加される相対パス名です。

5. ***rotate-size*** **値を指定します。**
   ログファイルがローテーションされる前に到達できる最大サイズです。数字に追加された単一の文字はサイズ単位を示します。バイトの場合は **b**、キロバイトの場合は **k**、メガバイトの場合は **m**、ギガバイトの場合は **g** になります。たとえば、50 メガバイトの場合は、**50m** になります。

6. ***max-backup-index*** **数を設定します。**
   保持されるローテーションログの最大数です。この数字に達すると、最も古いログが再利用されます。

7. ***formatter*** **を指定します。**
   このログハンドラーで使用するログフォーマッターの一覧を表示するには、***formatter*** を使用します。

8. ***append*** **プロパティーを設定します。**
   append プロパティーを **"true"** に設定した場合、このハンドラーが追加したすべてのメッセージが既存のファイルに追加されます。**"false"** に設定した場合、アプリケーションサーバーが起動するたびに新規ファイルが作成されます。***append*** への変更を反映させるには、サーバーの再起動が必要です。

バグを報告する

4.5.8. 非同期ログハンドラーの XML 設定例

以下の手順は、非同期ログハンドラーの設定例を示しています。

手順4.7 非同期ログハンドラーの設定

```
<async-handler name="Async_NFS_handlers">
   <level name="INFO"/>
   <queue-length value="512"/>
   <overflow-action value="block"/>
   <subhandlers>
      <handler name="FILE"/>
      <handler name="accounts-record"/>
   </subhandlers>
</async-handler>
```

1. ***name*** プロパティーは、このログハンドラーの一意の ID を設定します。

2. ***level*** プロパティーは、ルートロガーが記録するログメッセージの最大レベルを設定します。

3. ***queue-length*** は、サブハンドラーの応答を待機する間に、このハンドラーが保持するログメッセージの最大数を定義します。

4. ***overflow-action*** は、キューの長さを超えたときにこのハンドラーがどのように応答するかを定義します。これは **BLOCK** または **DISCARD** に設定できます。**BLOCK** の場合、キューでスペースが利用可能になるまでロギングアプリケーションが待機します。これは、非同期ではないログハンドラーと同じ動作です。**DISCARD** の場合、ロギングアプリケーションは動作を続けますが、ログメッセージは削除されます。

5. ***subhandlers*** リストは、この非同期ハンドラーがログメッセージを渡すログハンドラーの一覧です。

バグを報告する

パート III. キャッシュモードのセットアップ

第5章 キャッシュモード

Red Hat JBoss Data Grid は次の 2 つのモードを提供します。

- ローカルモードは、JBoss Data Grid で提供される唯一のクラスターキャッシュモードではないモードです。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。ローカルモードは、スケーラビリティーおよびフェイルオーバーが不要な場合に最も効果的であり、クラスターモードに比べてパフォーマンスが高くなります。

- クラスターモードは、状態の変更をノードのサブセットにレプリケートするクラスターモードを提供します。サブセットのサイズは、フォールトトラレンスを実現するには十分なサイズですが、スケーラビリティーを妨げるほど大きくはありません。クラスターモードを使用する前に、クラスター化された設定に対して JGroups を設定することが重要です。JGroups の設定方法についてさらに詳しくは、「JGroups の設定 (ライブラリーモード)」 を参照してください。

バグを報告する

5.1. キャッシュコンテナーについて

キャッシュコンテナーは、キャッシュを使用する際の開始点として Red Hat JBoss Data Grid のリモートクライアントサーバーモードで使用されます。**cache-container** 要素は 1 つ以上の (ローカルまたはクラスター) キャッシュの親として動作します。クラスターキャッシュをコンテナーに追加するには、トランスポートを定義する必要があります。

次の手順は、キャッシュコンテナーの設定例を示しています。

手順5.1 キャッシュコンテナーの設定方法

```
<subsystem xmlns="urn:infinispan:server:core:8.3"
    default-cache-container="local">
 <cache-container name="local"
    default-cache="default"
    statistics="true"
    start="EAGER">
  <local-cache name="default"
        start="EAGER"
        statistics="false">
        <!-- Additional configuration information here -->
  </local-cache>
 </cache-container>
</subsystem>
```

1. **キャッシュコンテナーの設定**
   cache-container 要素は、次のパラメーターを使用してキャッシュコンテナーに関する情報を指定します。

   a. ***name*** パラメーターはキャッシュコンテナーの名前を定義します。

   b. ***default-cache*** パラメーターは、キャッシュコンテナーと共に使用されるデフォルトキャッシュの名前を定義します。

   c. ***statistics*** 属性は任意であり、デフォルトは **true** です。統計は、JMX または JBoss Operations Network 経由で JBoss Data Grid を監視する際に役立ちますが、パフォーマンスにはマイナスの影響を与えます。統計が不要な場合は、これを **false** に設定

してこの属性を無効にします。

d. ***start*** パラメーターはいつキャッシュコンテナーが起動するかを示します (要求時にレイジーに起動するか、またはサーバー起動時に「すぐに (eargerly)」起動するかなど)。このパラメーターの有効な値は **EAGER** と **LAZY** です。

2. **キャッシュごとの統計の設定**
statistics がコンテナーレベルで有効にされている場合、***statistics*** 属性を **false** に設定することにより、キャッシュごとの統計は、監視を必要としないキャッシュについては選択的に無効にすることができます。

バグを報告する

5.2. ローカルモード

マップの代わりに Red Hat JBoss Data Grid のローカルモードを使用すると、多くの利点があります。

簡単なマップでは提供されない次のような機能をキャッシュが提供します。

- データを永続化するライトスルーおよびライトビハインドキャッシュ。
- Java 仮想マシン (JVM) がメモリー不足にならないようにするためのエントリーエビクション。
- 定義された期間後に期限切れになるエントリーのサポート。

JBoss Data Grid は、楽観的および非観的ロックなどの技術を使用してロックの取得を管理する、高パフォーマンスで読み取りをベースとするデータコンテナーを中心に構築されます。

また、JBoss Data Grid は CAS (Compare and Swap) アルゴリズムやその他のロックフリーアルゴリズムも使用するため、スループットの高いマルチ CPU 環境やマルチコア環境を実現します。さらに、JBoss Data Grid のキャッシュ API は JDK の **ConcurrentMap** を拡張するため、マップから JBoss Data Grid への移行プロセスが簡単になります。

バグを報告する

5.2.1. ローカルモードの設定

ローカルキャッシュは、ライブラリーモードとリモートクライアントサーバーの両方で、すべてのキャッシュコンテナーに追加することができます。以下の例は、**local-cache** 要素を追加する方法について説明しています。

手順5.2 local-cache 要素

```
<cache-container name="local"
                 default-cache="default"
                 statistics="true">
    <local-cache name="default"
        start="EAGER"
        batching="false"
        statistics="true">
        <!-- Additional configuration information here -->
    </local-cache>
```

local-cache 要素は次のパラメーターを使用して、キャッシュコンテナーと共に使用されるローカルキャッシュに関する情報を指定します。

1. ***name*** パラメーターは使用するローカルキャッシュの名前を指定します。
2. ***start*** パラメーターはいつキャッシュコンテナーが起動するかを示します (要求時にレイジーに起動するか、またはサーバー起動時に「すぐに (eargerly)」起動するかなど)。このパラメーターの有効な値は **EAGER** と **LAZY** です。
3. ***batching*** パラメーターは、ローカルキャッシュに対してバッチ処理が有効であるかを指定します。
4. ***statistics*** がコンテナーレベルで有効にされている場合、***statistics*** 属性を **false** に設定することにより、キャッシュごとの統計は、監視を必要としないキャッシュについては選択的に無効にすることができます。

この代わりに、引数のないコンストラクターで **DefaultCacheManager** を作成することもできます。どちらの方法でも、ローカルのデフォルトキャッシュが作成されます。

ローカルキャッシュとクラスター化されたキャッシュは同じキャッシュコンテナーで共存できますが、コンテナーに **<transport/>** がない場合はローカルキャッシュのみ格納できます。例で使用されたコンテナーには **<transport/>** がないため、ローカルキャッシュのみを格納できます。

キャッシュインターフェースは **ConcurrentMap** を拡張し、複数のキャッシュシステムと互換性があります。

バグを報告する

5.3. クラスターモード

Red Hat JBoss Data Grid は、次のクラスターモードを提供します。

- レプリケーションモードは、クラスター内のすべてのキャッシュインスタンス間で追加されたエントリーをレプリケートします。
- インバリデーションモードはデータを共有しませんが、無効なエントリーの削除を開始するようリモートキャッシュに伝えます。
- ディストリビューションモードは、クラスターの全ノード上ではなく、ノードのサブセット上の各エントリーを保管します。

ネットワーク通信に同期または非同期トランスポートを使用するよう、クラスターモードを追加で設定することが可能です。

バグを報告する

5.3.1. 非同期および同期の操作

クラスターモード (インバリデーション、レプリケーション、ディストリビューションなど) が使用されると、データが同期的または非同期的に他のノードへ伝搬されます。

同期モードが使用されると、送信側はスレッドの継続を許可する前に受信側からの応答を待ちます。非同期モードでは、データを送信しても、クラスターの他のノードからの応答を待たずに操作を継続します。

非同期モードは一貫性よりも速度を優先するため、スティッキーセッションが有効な HTTP セッションレプリケーションなどのユースケースに適しています。このようなセッション (他のユースケースではデータ) は、ノードに障害が発生しない限り常に同じクラスターノード上でアクセスされます。

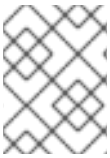
バグを報告する

5.3.2. 非同期通信について

Red Hat JBoss Data Grid では、ローカルモードは **local-cache** によって表され、ディストリビューションモードは **distributed-cache**、レプリケーションモードは **replicated-cache** によって表されます。これらの各要素には、***mode*** プロパティーが含まれ、同期通信の場合は **SYNC**、非同期通信の場合は **SYNC** に値を設定することができます。

例5.1 非同期通信の設定例

```
<replicated-cache name="default"
                  start="EAGER"
                  mode="ASYNC"
                  batching="false"
                  statistics="true">
                 <!-- Additional configuration information here -->
</replicated-cache>
```



注記

この設定は、JBoss Data Grid のどちらの使用モード (ライブラリーモードとリモートクライアントサーバーモード) でも有効です。

バグを報告する

5.3.3. キャッシュモードのトラブルシューティング

5.3.3.1. ReadExternal の無効なデータ

Cache.putAsync() を使用する場合、シリアライズを開始するとオブジェクトが変更される可能性があります。それによってデータストリームが破損されると、無効なデータが **readExternal** に渡されます。このような場合、オブジェクトへのアクセスを同期化すると、この問題を解決することができます。

バグを報告する

5.3.3.2. クラスター物理アドレスの読み出し

クラスターの物理アドレスの読み出し方法

インスタンスメソッド呼び出しを使用して物理アドレスを読み出すことができます。たとえば、**AdvancedCache.getRpcManager().getTransport().getPhysicalAddresses()** のように読み出します。

バグを報告する

第6章 ディストリビューションモードのセットアップ

6.1. ディストリビューションモードについて

Red Hat JBoss Data Grid のディストリビューションモードが有効になっている場合、全ノードの各エントリーをレプリケートせずに、グリッド内のノードのサブセット上に各エントリーが格納されます。冗長性とフォールトトラレンスを実現するため、通常各エントリーは複数のノード上に格納されます。

ディストリビューションモードの場合、クラスター全体の選択されたノード上にエントリーを格納するため、他のクラスターモードと比べスケーラビリティーが向上します。

ディストリビューションモードを使用するキャッシュは、一貫したハッシュアルゴリズムを使用し、クラスター全体で透過的にキーを検索することが可能です。

バグを報告する

6.2. ディストリビューションモードの一貫したハッシュアルゴリズム

Red Hat JBoss Data Grid のハッシュアルゴリズムは、一貫性のあるハッシュに基づきます。一貫性のあるハッシュという用語は、従来の一貫性のあるハッシュとは少し意味合いの異なるものが派生しているものの、この実装で依然として使われています。

分散モードは一貫したハッシュアルゴリズムを使用して、エントリーを格納するクラスターからノードを選択します。一貫したハッシュアルゴリズムは、クラスター内で維持される各キャッシュエントリーのコピー数で設定されます。汎用的な一貫性のあるハッシュとは異なり、JBoss Data Grid で使用される実装ではキー領域が固定セグメントに分割されます。セグメントの数は、***numSegments*** を使用して設定され、クラスターを再起動しても変更できません。キーとセグメントのマッピングも固定されます。クラスターのトポロジーがどのように変わるかに関係なく、キーは同じセグメントに対してマップされます。

パフォーマンスとフォールトトラレンスのバランスを考慮して、各データ項目のコピー数を設定する必要があります。エントリーのコピーが多すぎるとパフォーマンスが低下し、コピーが少なすぎるとノードの障害時にデータを損失する可能性があります。

各ハッシュセグメントはオーナーと呼ばれるノードのリストにマップされます。最初のオーナー (プライマリオーナーとも呼ばれます) は多くのキャッシュ操作 (ロックなど) で特殊な役割を担うため、この順序は重要です。他のオーナーはバックアップオーナーと呼ばれます。セグメントとオーナーのマッピングにはルールがありませんが、ハッシュアルゴリズムにより各ノードに割り当てられたセグメントの数が同時に分散され、ノードがクラスターに参加した後やクラスターから脱退した後に移動する必要があるセグメントの数が最小化されます。

バグを報告する

6.3. ディストリビューションモードにおけるエントリーの検索

Red Hat JBoss Data Grid のディストリビューションモードで使用される一貫したハッシュアルゴリズムは、要求をマルチキャストしたりコストのかかるメタデータを維持しなくてもエントリーを特定できるようにします。

PUT 操作が実行されると、リモート呼び出しが***owners*** に指定された回数実行されます。クラスターのいずれかのノードで **GET** 操作が実行されると、リモート呼び出しが 1 回実行されます。バックグラウンドでは、**GET** 操作が実行されると **PUT** 操作と同じ回数 (***owners*** パラメーターの値) のリモート呼び出しが行われますが、これらの呼び出しは同時に実行され、返されたエントリーは即座に呼び出し側に渡されます。

バグを報告する

6.4. ディストリビューションモードの戻り値

Red Hat JBoss Data Grid のディストリビューションモードでは、以前の戻り値がローカルで見つからない場合に同期要求を使用して以前の戻り値を読み出します。ディストリビューションモードが使用する処理が非同期か同期であるかに関係なく、この作業では同期要求が使用されます。

バグを報告する

6.5. ディストリビューションモードの設定

ディストリビューションモードは Red Hat JBoss Data Grid のクラスターモードです。以下の手順を使用して、ディストリビューションモードを、ライブラリーモードとリモートクライアントサーバーモードの両方でキャッシュコンテナーに追加することができます。

手順6.1 `distributed-cache` 要素

```
<cache-container name="clustered"
    default-cache="default"
    statistics="true">
  <!-- Additional configuration information here -->
  <distributed-cache name="default"
      mode="SYNC"
      segments="20"
      start="EAGER"
      owners="2"
      statistics="true">
    <!-- Additional configuration information here -->
  </distributed-cache>
</cache-container>
```

`distributed-cache` 要素は、以下のパラメーターを使用して分散キャッシュの設定を行います。

1. **`name`** パラメーターは、キャッシュの一意の ID を提供します。
2. **`mode`** パラメーターは、クラスター化されたキャッシュモードを設定します。有効な値は**`SYNC`** (同期) と **`ASYNC`** (非同期) です。
3. (オプションの) **`segments`** パラメーターは、クラスターごとのハッシュ領域セグメントの数を指定します。このパラメーターの推奨される値は、クラスターサイズに 10 を乗算した値であり、デフォルト値は **`20`** です。
4. **`start`** パラメーターは、サーバーの起動時か、サーバーが要求またはデプロイされるときにキャッシュを起動させるかどうかを指定します。
5. **`owners`** パラメーターは、ハッシュセグメントを含むノード数を示します。
6. ***`statistics`*** がコンテナーレベルで有効にされている場合、***`statistics`*** 属性を **`false`** に設定することにより、キャッシュごとの統計は、監視を必要としないキャッシュについては選択的に無効にすることができます。

重要

この設定をロードする前に、JGroups をクラスターモードに対して適切に設定する必要があります。

バグを報告する

6.6. 同期および非同期の分散

特定のパブリック API メソッドから意味のある戻り値を取得するには、ディストリビューションモードを使用するときに同期通信を使用する必要があります。

例6.1 通信モードの例

たとえば、**A**、**B**、**C** という 3 つのノードがクラスターにあり、ノード**A** と **B** をマップする **K** というキーがあるとします。また、戻り値の必要なクラスター **C** 上で、**`Cache.remove(K)`** のような操作を実行するとします。この場合、正常に実行するには、操作が最初にノード **A** と **B** の両方に呼び出しを同期転送し、ノード **A** または **B** より返される結果を待つ必要があります。非同期通信が使用された場合、操作が予期される通りに動作しても戻り値が有効であるかどうかは保証されません。

バグを報告する

第7章 レプリケーションモードのセットアップ

7.1. レプリケーションモードについて

Red Hat JBoss Data Grid のレプリケーションモードは、簡単なクラスターモードです。キャッシュインスタンスは、同じネットワーク上の異なる Java 仮想マシン (JVM) 上にある隣接したインスタンスを自動的に見つけ、見つけたインスタンスを用いてクラスターを形成します。キャッシュインスタンスへ追加されたエントリーは、クラスターのすべてのキャッシュインスタンス全体でレプリケートされ、すべてのクラスターキャッシュインスタンスよりローカルで読み出すことが可能です。

JBoss Data Grid のレプリケーションモードでは、レプリケーションが発生する前にローカルで戻り値を使用できます。

バグを報告する

7.2. 最適化されたレプリケーションモードの使用

レプリケーションモードは、クラスター間での状態の共有に使用されます。ただし、レプリケートされたキャッシュがあり、大量のノードが使用されている場合は、すべてのノードを同期するためにレプリケート済みキャッシュに多くの書き込みが行われます。実行される作業量は多くの要因と特定のユースケースに基づきます。このため、計画されたノードの数でレプリケートモードが適切であるかどうかを確認するために各ワークロードを完全にテストすることが推奨されます。多くの状況で、レプリケーションモードは推奨されません (10 台のサーバーがある場合)。ただし、ワークロードによっては (ロードの読み取りが重要な場合など)、このモードが適切であることがあります。

大型のクラスターのパフォーマンスをある程度向上させる UDP マルチキャストを使用するよう、Red Hat JBoss Data Grid を設定できます。

バグを報告する

7.3. レプリケーションモードの設定

レプリケーションモードは Red Hat JBoss Data Grid のクラスターモードです。以下の手順を使用して、レプリケーションモードを、ライブラリーモードとリモートクライアントサーバーモードの両方でキャッシュコンテナーに追加することができます。

手順7.1 *replicated-cache* 要素

```
<cache-container name="clustered"
   default-cache="default"
   statistics="true">
  <!-- Additional configuration information here -->
  <replicated-cache name="default"
      mode="SYNC"
      start="EAGER"
      statistics="true">
    <!-- Additional configuration information here -->
  </replicated-cache>
</cache-container>
```

重要

この設定をロードする前に、JGroups をクラスターモードに対して適切に設定する必要があります。

replicated-cache 要素は、以下のパラメーターを使用して分散キャッシュの設定を行います。

1. **name** パラメーターは、キャッシュの一意の ID を提供します。
2. **mode** パラメーターは、クラスター化されたキャッシュモードを設定します。有効な値は**SYNC** (同期) と **ASYNC** (非同期) です。
3. **start** パラメーターは、サーバーの起動時か、サーバーが要求またはデプロイされるときにキャッシュを起動させるかどうかを指定します。
4. ***statistics*** がコンテナーレベルで有効にされている場合、***statistics*** 属性を **false** に設定することにより、キャッシュごとの統計は、監視を必要としないキャッシュについては選択的に無効にすることができます。

cache-container および **locking** の詳細については、該当する章を参照してください。

バグを報告する

7.4. 同期および非同期のレプリケーション

対処する問題に応じて、レプリケーションモードは同期または非同期のいずれかになります。

- 同期レプリケーションは、クラスターの全ノードで変更がレプリケートされるまでスレッドや呼び出し側 (**put()** 操作の場合など) をブロックします。確認応答を待つために、同期レプリケーションでは操作が終了する前にすべてのレプリケーションが正常に適用されます。
- 非同期レプリケーションはノードからの応答を待つ必要がないため、同期レプリケーションよりもかなり高速になります。非同期レプリケーションはバックグラウンドでレプリケーションを実行し、呼び出しは即座に返されます。非同期レプリケーション中に発生したエラーはログに書き込まれます。そのため、クラスターのすべてのキャッシュインスタンスでトランザクションが正常にレプリケートされなくても、トランザクションは正常に終了することが可能です。

バグを報告する

7.4.1. 非同期レプリケーションの動作に対するトラブルシューティング

インスタンスによっては、非同期のレプリケーションや分散に対して設定されたキャッシュが、同期の場合と同様に応答を待ってしまうことがあります。これは、状態転送と非同期モードの両方が設定されていると、キャッシュは同期的に動作するためです。状態転送を予想通りに機能させるには、同期的な動作が必要となります。

この問題に対処するには、以下の方法の 1 つに従います。

- 状態転送を無効にし、 **ClusteredCacheLoader** を使用して必要な時にリモート状態をレイジーにルックアップします。
- 状態転送と ***REPL_SYNC*** を有効にします。非同期 API (**cache.putAsync(k, v)** など) を使用して「fire-and-forget」機能をアクティベートします。

- 状態転送と ***REPL_ASYNC*** を有効にします。PRC はすべて同期的になりますが、レプリケーションキューを有効にすると (非同期モードで推奨) クライアントスレッドは中断されません。

バグを報告する

7.5. レプリケーションキュー

レプリケーションモードでは、Red Hat JBoss Data Grid が以下に基づいてレプリケーションキューを使用し、ノード全体で変更をレプリケートします。

- 以前に設定された間隔。
- 要素数を超えるキューサイズ。
- 以前に設定された間隔と要素数を超えるキューサイズの組み合わせ。

レプリケーションキューは、レプリケート中にキャッシュ操作が個別に送信されるのではなく、一括送信されるようにします。そのため、送信されるレプリケーションメッセージの数が減り、使用されるエンベロープも少なくなるため、JBoss Data Grid のパフォーマンスが向上します。

レプリケーションキューを使用する場合の難点は、時間やキューサイズを基にキューが周期的にフラッシュされることです。このようなフラッシュ操作は、クラスターノード全体のレプリケーション、分散、または無効化の操作を遅延させます。レプリケーションキューを無効にすると、データは直接送信されるため、クラスターノードへ達する時間が短くなります。

レプリケーションキューは非同期モードと共に使用されます。

バグを報告する

7.5.1. レプリケーションキューの使用

レプリケーションキューを使用する場合、以下の 1 つを実行します。

- 非同期マーシャリングを無効にします。
- ***max-threads*** 数の値を、***transport*** 要素の ***executor*** 属性に対して **1** に設定します。**executor** はライブラリーモードでのみ使用できるため、以下のように設定ファイルに定義されます。

```
<transport executor="infinispan-transport"/>
```

これらの方法の 1 つを実装するには、レプリケーションキューを非同期モードで使用する必要があります。非同期モードは、キュータイムアウト (***queue-flush-interval***、値はミリ秒単位) やキューサイズ (***queue-size***) と共に次のように設定することができます。

例7.1 非同期モードのレプリケーションキュー

```
<replicated-cache name="asyncCache"
                  start="EAGER"
                  mode="ASYNC"
                  batching="false"
                  indexing="NONE"
                  statistics="true"
                  queue-size="1000"
```

```
                    queue-flush-interval="500">
                <!-- Additional configuration information here -->
</replicated-cache>
```

レプリケーションキューにより、要求がクライアントへ返されるまでの時間が短縮されるため、レプリケーションキューを非同期マーシャリングと共に使用しても大きな利点はありません。

バグを報告する

7.6. レプリケーション保証について

クラスター化されたキャッシュでは、ユーザーは同期レプリケーション保証と非同期レプリケーションに関連する並列性を得ることができます。Red Hat JBoss Data Grid はこの目的で非同期 API を提供します。

API で使用される非同期メソッドは、クエリー可能な Future を返します。クエリーは、使用されるネットワーク呼び出しが正常に行われたことの確認を受信するまでスレッドをブロックします。

バグを報告する

7.7. 内部ネットワークのレプリケーショントラフィック

クラウドプロバイダーによっては、内部 **IP** アドレスを介したトラフィックにパブリック **IP** アドレスを介したトラフィックよりも低い課金を行ったり、内部ネットワークトラフィックにまったく課金しないことがあります (GoGrid など)。低料金で利用できるよう、内部ネットワークを使用してレプリケーションのトラフィックを転送するよう Red Hat JBoss Data Grid を設定することが可能です。このような設定では、割り当てられた内部 **IP** アドレスを調べるのは簡単ではありませんが、JBoss Data Grid は JGroups インターフェースを使用してこの問題を解決します。

バグを報告する

第8章 インバリデーションモードのセットアップ

8.1. インバリデーションモードについて

無効化はクラスターモードで、データを共有しませんが、リモートキャッシュの潜在的に古いデータを削除します。このキャッシュモードを使用するには、データベースなどのさらに永久的なデータストアが別に必要になります。

このような状況で、Red Hat JBoss Data Grid は、数多くの読み取り操作を実行するシステムを最適化するために使用され、状態が必要となるたびにデータベースが使用されないようにします。

インバリデーションモードが使用されている場合、キャッシュのデータが変更になると、クラスターの他のキャッシュが古いデータをメモリーからエビクトします。

バグを報告する

8.2. インバリデーションモードの設定

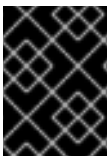
インバリデーションモードは Red Hat JBoss Data Grid のクラスターモードです。以下の手順を使用して、インバリデーションモードを、ライブラリーモードとリモートクライアントサーバーモードの両方でキャッシュコンテナーに追加することができます。

手順8.1 `invalidation-cache` 要素

```
<cache-container name="local"
        default-cache="default"
        statistics="true">
 <invalidation-cache name="default"
       mode="ASYNC"
       start="EAGER"
       statistics="true">
   <!-- Additional configuration information here -->
 </invalidation-cache>
</cache-container>
```

`invalidation-cache` 要素は、以下のパラメーターを使用して分散キャッシュの設定を行います。

1. **`name`** パラメーターは、キャッシュの一意の ID を提供します。
2. **`mode`** パラメーターは、クラスター化されたキャッシュモードを設定します。有効な値は**SYNC** (同期) と **ASYNC** (非同期) です。
3. **`start`** パラメーターは、サーバーの起動時か、サーバーが要求またはデプロイされるときにキャッシュを起動させるかどうかを指定します。
4. ***`statistics`*** がコンテナーレベルで有効にされている場合、***`statistics`*** 属性を **`false`** に設定することにより、キャッシュごとの統計は、監視を必要としないキャッシュについては選択的に無効にすることができます。



重要

この設定をロードする前に、JGroups をクラスターモードに対して適切に設定する必要があります。

cache-container、**locking**、および **transaction** 要素について詳しくは、該当する章を参照してください。

バグを報告する

8.3. 同期的/非同期の無効化

Red Hat JBoss Data Grid のライブラリーモードでは、無効化は同期的または非同期的に機能します。

- 同期的な無効化は、クラスターのすべてのキャッシュが無効化メッセージを受信し、古いデータをエビクトするまでスレッドをブロックします。
- 非同期的な無効化は、応答待ちのスレッドをブロックせずに無効化メッセージがブロードキャストされる fire-and-forget モードで機能します。

バグを報告する

8.4.1 次キャッシュと無効化

無効化メッセージはキーが更新されるたびに生成されます。このメッセージは、現在の 1 次キャッシュエントリーに対応するデータが含まれる各ノードへマルチキャストされます。無効化メッセージにより、これらのノードは関連エントリーを無効としてマークするようになります。

バグを報告する

第9章 状態転送

状態転送は、基本的なデータグリッドまたはクラスター化キャッシュ機能です。状態転送を使用しないと、ノードをクラスターに追加する場合やノードをクラスターから削除する場合にデータが失われます。

状態転送により、キャッシュメンバーシップでの変更に応じてキャッシュの内部状態が調整されます。変更は、ノードが参加または脱退したとき、2 つ以上のクラスターパーティションがマージしたとき、参加、脱退、およびマージが実行されたときに行われます。状態転送は、ノードがクラスターに参加したときやクラスターから脱退したときに必ず Red Hat JBoss Data Grid で自動的に実行されます。

Red Hat JBoss Data Grid のレプリケーションモードでは、キャッシュに参加する新しいノードは既存のノードから全体のキャッシュ状態を受け取ります。分散モードでは、新しいノードは既存のノードから一部の状態のみを受け取り、既存のノードは各キーの **owners** コピーをキャッシュに保持するために一部の状態を削除します (整合性のあるハッシュによって決定されます)。インバリデーションモードでは、最初の状態転送はレプリケーションモードと似ており、唯一の違いはノードの状態が同じであることが保証されないことです。ノードが脱退すると、レプリケーションモードまたはインバリデーションモードでキャッシュが状態転送を実行しません。分散キャッシュは、各キーの **owners** コピーを保持するために、脱退するノードに格納されたキーの追加コピーを作成する必要があります。

状態転送では、デフォルトでインメモリー状態と永続状態の両方が転送されますが、それらはどちらも設定で無効にできます。状態転送が無効にされる場合は **ClusterLoader** を設定する必要があります。設定しない場合、ノードは、データがそのキャッシュにロードされることなく、キーの所有者またはバックアップ所有者になります。さらに、状態転送が分散モードで無効にされると、キーの所有者は ***owners*** より少なくなることがあります。

バグを報告する

9.1. 非ブロッキング状態転送

Red Hat JBoss Data Grid における非ブロック状態転送は、状態転送が実行中であるためクラスターまたはノードが応答できない時間を最小化します。非ブロック状態転送はアーキテクチャー上の大きな機能向上であり、その目的は以下のとおりです。

- 状態転送が実行中であるためクラスター全体が要求に応答できない時間を最小化します。
- 状態転送が実行中であるため既存のメンバーが要求への応答を中止する時間を最小化します。
- 状態転送を実行することを可能にします (クラスターのパフォーマンスが低下します)。ただし、状態転送時にパフォーマンスが低下すると、例外がスローされず、プロセスを続行できます。
- 状態転送の実行中に null 値を返さずに **GET** 操作が別のノードからキーを正常に取得することを許可します。

簡素化のために、総オーダーベースコミットプロトコルは、現在実装されている状態転送メカニズムのブロックバージョンを使用します。通常の状態転送と総オーダー状態転送の主な違いは、以下のとおりです。

- ブロックプロトコルは、状態転送中にトランザクション配信をキューに格納します。
- 状態転送制御メッセージ (CacheTopologyControlCommand など) は、総オーダー情報に基づいて送信されます。

総オーダーベースコミットプロトコルは、すべてのトランザクションが同じオーダーで配信され、同じデータセットを認識することを前提として動作します。したがって、トランザクションは、すべてのノードでメモリーに最新のキーまたは値が含まれる必要があるため、状態転送中に検証されません。

状態転送とブロックプロトコルをこのように使用すると、すべてのノードでの状態転送とトランザクション配信を同期できます。ただし、状態転送にすでに関係があるトランザクション (状態転送が始まる前に送信され、状態転送が終了した後に配信されたトランザクション) は再び送信する必要があります。再送信された場合、これらのトランザクションは新しい参加者として扱われ、新しい総オーダー値が割り当てられます。

バグを報告する

9.2. JMX による状態転送の抑制

保守を行うためにクラスターの停止および再起動を行うにあたり、JMX を使用して状態転送を抑制することができます。この操作は、より効率的なクラスターのシャットダウンと起動を許可し、グリッドを停止する際のメモリー不足のエラーの発生リスクを取り除きます。

新規ノードがクラスターに参加し、再調整が抑制される際に、**`getCache()`** 呼び出しは、再調整が再度有効にされないか、または **`stateTransfer.awaitInitialTransfer`** が **`false`** に設定されない限り、**`stateTransfer.timeout`** が期限切れになった後にタイムアウトになります。

状態転送および再調整を無効にすることは、部分的なクラスターのシャットダウンや再起動の場合に有効ですが、状態転送が無効にされているために部分的なクラスターのシャットダウンでデータが失われる可能性があります。

バグを報告する

9.3. REBALANCINGENABLED 属性

再調整の抑制は、**`rebalancingEnabled`** JMX 属性によってのみトリガーでき、これには特定の設定は不要です。

`rebalancingEnabled` 属性は、いずれのノードでも **`LocalTopologyManager`** JMX Mbean からクラスター全体に対して変更することができます。この属性はデフォルトでは**`true`** であり、プログラムを使って設定することができます。

Hot Rod などのサーバーは、起動時に設定で宣言されるすべてのキャッシュを起動するよう試みます。再調整が無効にされる場合、キャッシュは起動に失敗します。そのため、サーバー環境で以下の設定を使用することが必須になります。

```
<await-initial-transfer="false"/>
```

バグを報告する

パート IV. API の有効化

第10章 API の宣言的な有効化

JBoss Data Grid が提供する各種の API については、JBoss Data Grid の 『Developer Guide』 で詳細に記述されていますが、管理者は要素を設定ファイルに追加することでこれらを宣言的に有効にできます。以下のセクションでは、各種 API を実装する方法について説明します。

バグを報告する

10.1. バッチ化 API

バッチ化により、トランザクションの原子性 (atomicity) と一部の特性が有効になりますが、本格的な JTA または XA 機能は許可されません。通常、バッチ化は本格的なトランザクションよりも軽量で、コスト安になりますが、トランザクションに参加するのが JBoss Data Grid クラスターのみである場合は常に使用する必要があります。トランザクションに複数のシステムが関係する場合は、JTA トランザクションが使用される必要があります。たとえば、ある銀行口座から別の銀行口座に送金するトランザクションについて考えてみましょう。どちらの口座も JBoss Data Grid クラスター内に保存される場合は、バッチを使用できますが、1 つの口座のみがクラスター内にあり、別の口座が外部データベースにあると、分散トランザクションが必要になります。

バッチ化 API の有効化

バッチ化は、**BATCH** のトランザクションモードを定義してキャッシュごとに有効にできます。以下の例はこれを説明しています。

```
<local-cache>
   <transaction mode="BATCH"/>
</local-cache>
```

デフォルトでは、呼び出しのバッチ化は無効にされています。さらにトランザクションマネージャーはバッチ化の使用に必要ではありません。

バグを報告する

10.2. グループ化 API

グループ化 API により、各エントリーがエントリーの計算されたハッシュコードに対応するノードに保存されるデフォルト動作とは異なり、エントリーのグループを同じノード上に共存させることができます。デフォルトで、JBoss Data Grid はキーの保存時に各キーのハッシュコードを取り、そのキーをハッシュセグメントにマップします。これにより、キーを含むノードを判別するためにアルゴリズムが使用され、クラスター内の各ノードが、所有権情報を配信しなくてもキーを含むノードを識別できます。この動作により、ノードの失敗時に所有権情報のレプリケートが不要になるため、オーバーヘッドや重複が削減されます。

グループ化 API を有効にすることで、エントリーを保存するノードを決定する際にキーのハッシュが無視されます。代わりに、グループのハッシュが取得および使用されます。その際、キーのハッシュはパフォーマンスの低下を防ぐために内部で使用されます。グループ API が使用される場合、すべてのノードはキーの所有者を依然として判別できます。そのため、グループは手動で指定できません。グループは、キークラスで生成されるエントリー固有のものか、または外部関数で生成される、エントリーの外部の値によって決められる可能性があります。

グループ化 API の有効化

グループ化 API は、以下の例にあるように **groups** 要素を追加することにより、キャッシュごとに有効にすることができます。

```
<distributed-cache>
    <groups enabled="true"/>
</distributed-cache>
```

外部グループの定義

カスタム **Grouper** が存在することを仮定した場合、以下のようにクラス名を渡すことによって定義できます。

```
<distributed-cache>
    <groups enabled="true">
        <grouper class="com.acme.KXGrouper" />
    </groups>
</distributed-cache>
```

バグを報告する

10.3. EXTERNALIZABLE API

Externalizer クラスは、以下を実行するクラスです。

- 該当するオブジェクトタイプをバイトアレイにマーシャリングします。
- バイトアレイの内容のオブジェクトタイプのインスタンスに対するマーシャリングを解除します。

エクスターナライザーは Red Hat JBoss Data Grid により使用され、ユーザーはオブジェクトタイプをどのようにシリアライズするかを指定できます。JBoss Data Grid で使用されるマーシャリングインフラストラクチャーは、JBoss Marshalling に基づいて構築され、効率的なペイロード配信を提供し、ストリームをキャッシュすることを可能にします。ストリームキャッシングを使用すると、データに複数回アクセスできますが、通常はストリームは 1 度だけ読み取ることができます。

Externalizable インターフェースはシリアライゼーションを使用し、拡張します。このインターフェースは、JBoss Data Grid でシリアライゼーションとシリアライゼーション解除を制御するために使用されます。

バグを報告する

10.3.1. 高度なエクスターナライザーの登録 (宣言的)

高度なエクスターナライザーがセットアップされた後に、Red Hat JBoss Data Grid で使用するためにこれを登録します。登録は以下のように宣言的に (XML で) 実行されます。

手順10.1 高度なエクスターナライザーの登録

```
<infinispan>
    <cache-container>
        <serialization>
            <advanced-externalizer class="Book$BookExternalizer" />
        </serialization>
    </cache-container>
</infinispan>
```

1. **serialization** 要素を **cache-container** 要素に追加します。

2. **advanced-externalizer** 要素を追加して、カスタムエクスターナライザーを**class** 属性で定義します。必要に応じて *Book$BookExternalizer* 値を置き換えます。

バグを報告する

10.3.2. カスタムエクスターナライザー ID 値

高度なエクスターナライザーには必要な場合にカスタム ID を割り当てることができます。一部の ID 範囲は他のモジュールやフレームワーク用に予約されるため、その使用を避ける必要があります。

表10.1 エクスターナライザーの予約される ID 範囲

ID 範囲	対象
1000-1099	Infinispan Tree モジュール
1100-1199	Red Hat JBoss Data Grid Server モジュール
1200-1299	Hibernate Infinispan 2 次キャッシュ
1300-1399	JBoss Data Grid Lucene Directory
1400-1499	Hibernate OGM
1500-1599	Hibernate Search
1600-1699	Infinispan Query モジュール
1700-1799	Infinispan Remote Query モジュール

バグを報告する

10.3.2.1. エクスターナライザー ID のカスタマイズ (宣言的)

高度なエクスターナライザー ID を以下のように宣言的に (XML で) カスタマイズします。

手順10.2 エクスターナライザー ID のカスタマイズ (宣言的)

```
<infinispan>
    <cache-container>
        <serialization>
            <advanced-externalizer id="123"
                                   class="Book$BookExternalizer"/>
        </serialization>
    </global>
</infinispan>
```

1. **serialization** 要素を **cache-container** 要素に追加します。

2. **advanced-externalizer** 要素を追加して、新規の高度なエクスターナライザーについての情報を追加します。

3. ***id*** 属性を使用してエクスターナライザー ID を定義します。選択した ID が他のモジュール用に予約された ID の範囲にないことを確認します。

4. ***class*** 属性を使用してエクスターナライザークラスを定義します。必要に応じて *Book$BookExternalizer* 値を置き換えます。

バグを報告する

第11章 INFINISPAN QUERY API のセットアップおよび設定

11.1. INFINISPAN QUERY のセットアップ

11.1.1. Infinispan Query の依存関係 (ライブラリーモード)

Maven で JBoss Data Grid Infinispan Query を使用するには、以下の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>
```

Maven 以外のユーザーは、JBoss Data Grid ディストリビューションからすべての **infinispan-embedded-query.jar** および **infinispan-embedded.jar** ファイルをインストールする必要があります。

警告

Infinispan Query API は Hibernate Search と Lucene API を直接公開し、**infinispan-embedded-query.jar** ファイル内に埋め込むことはできません。他のバージョンの Hibernate Search と Lucene を **infinispan-embedded-query** として同じデプロイメントに組み込まないようにしてください。これらが含まれると、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

11.2. インデックス化モード

11.2.1. インデックスの管理

Red Hat JBoss Data Grid の Query モジュールでは、インデックスを保管するための 2 つのオプションがあります。

1. 各ノードがグローバルインデックスの個別コピーを維持できる。
2. インデックスがすべてのノード間で共有される。

indexLocalOnly を **true** に設定してインデックスがローカルに保存されると、それらのインデックスを更新できるようにそれぞれのキャッシュへの書き込みは他のすべてのノードに転送される必要があります。***indexLocalOnly*** を **false** に設定してインデックスが共有される場合、書き込み元のノードのみが共有インデックスを更新する必要があります。

Lucene は、インデックスを格納するために使用される **directory provider** というディレクトリー構造の抽象化を行います。インデックスは、インメモリーとして、ファイルシステム上、または分散されたキャッシュ内に格納できます。

バグを報告する

11.2.2. インデックスの管理 (ローカルモード)

ローカルモードでは、Lucene Directory 実装を使用できます。***indexLocalOnly*** オプションはローカルモードでは意味がありません。

バグを報告する

11.2.3. インデックスの管理 (レプリケートモード)

レプリケーションモードでは、各ノードがインデックスのローカルコピーを独自に保管できます。各ノードでインデックスをローカルに保管するには、**indexLocalOnly** を **false** に設定して、各ノードがローカルで開始された更新と共に、他のノードから受信する必要な更新を適用できるようにします。

任意の Directory 実装を使用できます。新規ノードの起動時に、インデックスの最新コピーが受信される必要があります。通常、これは再同期で実行できますが、外部操作の場合でとくに更新が頻繁に実行される場合には、インデックスの同期が若干取れなくなる可能性があります。

または、インデックスの共有ストレージが使用される場合 (「Infinispan Directory Provider」 を参照してください)、**indexLocalOnly** を **true** に設定し、各ノードがローカルに発生した変更のみを適用できるようにします。これにより、インデックスが同期しないというリスクはなくても、インデックスの更新に使用されるノードには競合が生じます。

以下の図は、各ノードがローカルインデックスを持つレプリケートされたデプロイメントを示しています。

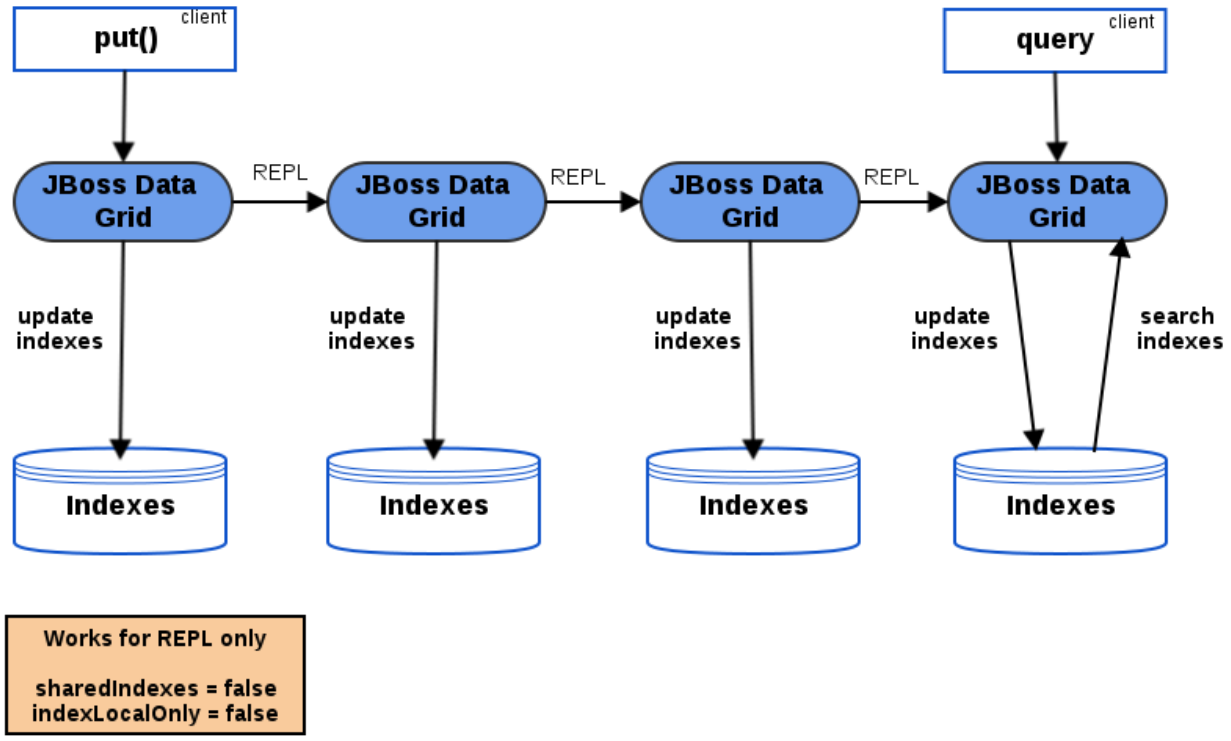


図11.1 レプリケートされたキャッシュクエリー

バグを報告する

11.2.4. インデックスの管理 (ディストリビューションモード)

いずれのディストリビューションモードでも、`indexLocalOnly` を `true` に設定して共有インデックスを使用する必要があります。

以下の図は、共有インデックスのあるデプロイメントを示しています。

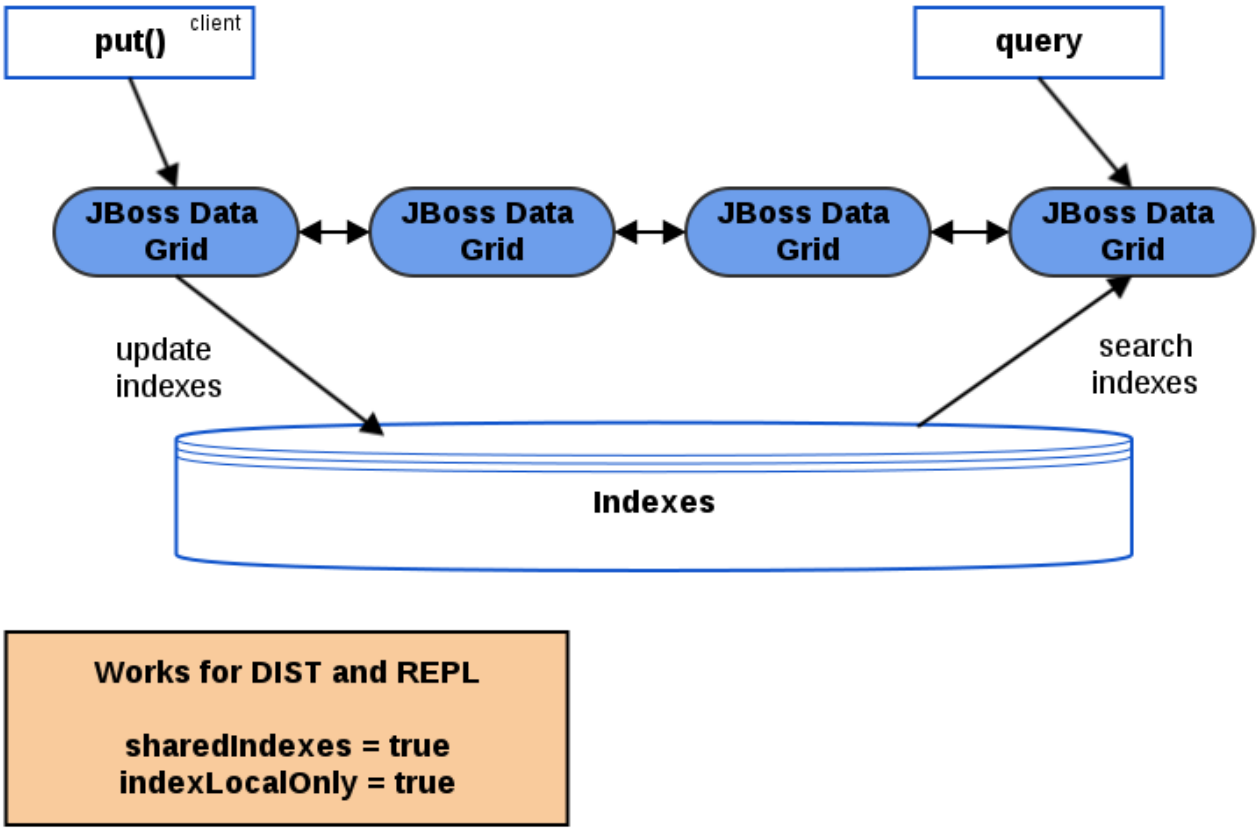


図11.2 共有インデックスによるクエリー

バグを報告する

11.2.5. インデックスの管理 (インバリデーションモード)

インバリデーションモードでの要素のインデックス化と検索はサポートされていません。

バグを報告する

11.3. DIRECTORY PROVIDER

以下のディレクトリープロバイダーは Infinispan Query でサポートされます。

- RAM Directory Provider
- Filesystem Directory Provider
- Infinispan Directory Provider

バグを報告する

11.3.1. RAM Directory Provider

グローバルインデックスを Red Hat JBoss Data Grid の Query モジュールにローカルに保存することにより、各ノードでは以下が可能になります。

- 独自のインデックスの保持。
- **Lucene** のインメモリーまたはファイルシステムベースのインデックスディレクトリーの使用。

以下の例は、インメモリーの RAM ベースのインデックスストアを示しています。

```
<local-cache name="indexesInMemory">
    <indexing index="LOCAL">
        <property name="default.directory_provider">ram</property>
    </indexing>
</local-cache>
```

バグを報告する

11.3.2. Filesystem Directory Provider

インデックスのストレージを設定するには、JBoss Data Grid 設定でインデックス化を有効にする場合に適切なプロパティーを設定します。

以下の例では、ディスクベースのインデックスストアを示しています。

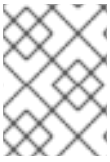
例11.1 ディスクベースのインデックスストア

```
<local-cache name="indexesInInfinispan">
    <indexing index="ALL">
        <property
name="default.directory_provider">filesystem</property>
        <property name="default.indexBase">/tmp/ispn_index</property>
    </indexing>
</local-cache>
```

バグを報告する

11.3.3. Infinispan Directory Provider

Lucene ディレクトリーの実装のほかにも、Red Hat JBoss Data Grid には **`infinispan-directory`** モジュールが同梱されます。

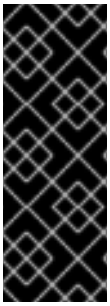


注記

Red Hat JBoss Data Grid は、**`infinispan-directory`** をスタンドアロンの機能としてではなく、クエリー機能との関連でのみサポートしています。

`infinispan-directory` により、**Lucene** は分散されたデータグリッド内にインデックスを保存できます。これにより、インデックスを分散し、メモリーに保存し、またオプションではキャッシュストアを使用してディスクに書き込むことで永続化することもできます。

Infinispan **Directory Provider** を使用して同じインデックスのインスタンスを共有することで、単一インスタンスで同じインデックスに同時に書き込まれる可能性があるため、書き込みの競合点が生じます。



重要

デフォルトで、**exclusive_index_use** は **`true`** に設定されます。これにより、パフォーマンスが大幅に向上します。ただし、外部アプリケーションが **infinispan** で使用されているものと同じインデックスにアクセスする場合、このプロパティーは **`false`** に設定される必要があります。ほとんどアプリケーションとユースケースにおいては、パフォーマンスが向上するためにデフォルト値の使用が推奨されます。そのため、この値は確実に変更する必要がある場合にのみ変更してください。

`InfinispanIndexManager` は、マスターにすべての更新を送信するデフォルトのバックエンドを提供します。マスターは後に更新をインデックスに適用します。マスターノードに障害が発生すると更新が失われる可能性があるため、キャッシュとインデックスは同期させません。デフォルト以外のバックエンドはサポートされません。

例11.2 共有インデックスの有効化

```
<local-cache name="indexesInInfinispan">
    <indexing index="ALL">
        <property
name="default.directory_provider">infinispan</property>
        <property
name="default.indexmanager">org.infinispan.query.indexmanager.Infinispan
IndexManager</property>
    </indexing>
</local-cache>
```

インデックス化され、クラスター化されたキャッシュを使用する場合、「Infinispan ディレクトリーのチューニング」にあるようにインデックスデータを含むキャッシュもクラスター化されていることを確認してください。

バグを報告する

11.4. インデックスの設定

11.4.1. リモートクライアントサーバーモードでのインデックスの設定

リモートクライアントサーバーモードでは、インデックスの設定はプロバイダーとその設定によって異なります。インデックスモードはプロバイダーや、それがローカルまたは分散されているものかどうかによって異なります。以下のインデックスモードがサポートされています。

- NONE
- LOCAL = indexLocalOnly="true"
- ALL = indexLocalOnly="false"

リモートクライアントサーバーモードでのインデックス設定は以下の通りです。

例11.3 リモートクライアントサーバーモードでの設定

```
<indexing index="LOCAL">
    <property name="default.directory_provider">ram</property>
    <!-- Additional configuration information here -->
</indexing>
```

Lucene キャッシュの設定

デフォルトで、Lucene キャッシュはローカルキャッシュとして作成されます。ただし、この設定では Lucene の検索結果がクラスターのノード間で共有されません。これを避けるには、以下の設定スニペットにあるようにクラスターモードで Lucene で必要とされるキャッシュを定義します。

例11.4 リモートクライアントサーバーモードでの Lucene キャッシュの設定

```
<cache-container name="clustered" default-cache="repltestcache">
    [...]
    <replicated-cache name="LuceneIndexesMetadata" mode="SYNC">
        <transaction mode="NONE"/>
        <indexing index="NONE"/>
    </replicated-cache>
    <distributed-cache name="LuceneIndexesData" mode="SYNC">
        <transaction mode="NONE"/>
        <indexing index="NONE"/>
    </distributed-cache>
    <replicated-cache name="LuceneIndexesLocking" mode="SYNC">
        <transaction mode="NONE"/>
        <indexing index="NONE"/>
    </replicated-cache>
    [...]
</cache-container>
```

これらのキャッシュについては、Red Hat JBoss Data Grid の 『Developer Guide』 で詳細に説明されています。

バグを報告する

11.4.2. インデックスの再構築

Lucene インデックスは、キャッシュ内のデータストアから再構築することによって、必要な場合に再構築できます。

インデックスは以下の場合に再構築する必要があります。

- タイプでインデックス化されている内容の定義が変更されている。
- ***Analyser*** などのインデックスの定義方法に影響を与えるパラメーターが変更されている。
- インデックスがシステム管理者のエラーにより、破壊または破損している。

インデックスを再構築するには、以下のように **MassIndexer** への参照を取得し、以下のようにこれを開始します。

```
SearchManager searchManager = Search.getSearchManager(cache);
searchManager.getMassIndexer().start();
```

この操作はグリッド内のすべてのデータを再処理するため、時間がかかる場合があります。

インデックスの再構築は JMX 操作としても有効です。

バグを報告する

11.5. インデックスのチューニング

11.5.1. Near-Realtime Index Manager

デフォルトでは、それぞれの更新がインデックスに即時にフラッシュされます。スループットを改善するには、更新をバッチ化することができます。ただし、これによって、更新とクエリー間にずれが発生する可能性があります。クエリーで古いデータが表示される可能性があります。これを許容できる場合、以下を設定して Near-Realtime Index Manager を使用できます。

```
<property name="default.indexmanager">near-real-time</property>
```

バグを報告する

11.5.2. Infinispan ディレクトリーのチューニング

Lucene ディレクトリーは 3 つのキャッシュを使用してインデックスを保存します。

- データキャッシュ
- メタデータキャッシュ
- ロッキングキャッシュ

これらのキャッシュは明示的に設定でき、以下の例にあるようにキャッシュ名を指定し、それらのキャッシュを通常どおりに設定できます。これらのキャッシュすべては、Infinispan ディレクトリーがローカルモードで使用されていない場合にクラスター化される必要があります。

例11.5 Infinispan ディレクトリーのチューニング

```
<distributed-cache name="indexedCache" mode="SYNC" owners="2">
    <indexing index="LOCAL">
        <property
name="default.indexmanager">org.infinispan.query.indexmanager.Infinispan
IndexManager</property>
        <property
name="default.metadata_cachename>lucene_metadata_repl</property>
        <property
name="default.data_cachename">lucene_data_dist</property>
        <property
name="default.locking_cachename">lucene_locking_repl</property>
    </indexing>
</distributed-cache>

<replicated-cache name="lucene_metadata_repl" mode="SYNC" />
```

```
<distributed-cache name="lucene_data_dist" mode="SYNC" owners="2" />

<replicated-cache name="lucene_locking_repl" mode="SYNC" />
```

バグを報告する

11.5.3. インデックス前の設定

上記の例のインデックスプロパティーはすべてのインデックスに適用されます。それは、各プロパティーに **`default.`** プレフィックスが使用されるためです。各インデックスに異なる設定を指定するには、**`default`** をインデックス名に置き換えます。デフォルトでは、これはインデックス化されたオブジェクトの完全クラス名ですが、**`@Indexed`** アノテーションのインデックス名は上書きできます。

バグを報告する

パート V. リモートクライアントサーバーモードインターフェース

Red Hat 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

第12章 REST インターフェース

Red Hat JBoss Data Grid は、REST インターフェースを提供します。REST API の主な利点は、クライアントとサーバー間で疎結合が可能になることです。クライアントライブラリーおよびバインディングの特定のバージョンに対する依存性がなくなります。REST API によりオーバーヘッドが発生し、REST クライアントまたはカスタムコードが REST 呼び出しを認識し、作成する必要があります。

JBoss Data Grid の REST API と対話するには、HTTP クライアントライブラリーのみが必要です。Java の場合は Apache HTTP Commons Client が推奨されます。または、java.net API を使用できます。

重要

以下の例では、REST セキュリティーが REST コネクターで無効にされていることを前提とします。REST セキュリティーを無効にするには、コネクターから ***security-domain*** および ***auth-method*** パラメーターを削除します。

バグを報告する

12.1. REST インターフェースコネクター

REST コネクターは Web サブシステムが必要である点で、Hot Rod および Memcached コネクターとは異なります。したがって、ソケットバインディング、ワーカースレッド、タイムアウトなどの設定は、Web サブシステムで実行する必要があります。

REST インターフェースがサーバーで有効にされると、通常これをデータの追加、削除、および取得のために使用できます。これらのプロセスについての詳細は、JBoss Data Grid の 『Developer Guide』を参照してください。

バグを報告する

12.1.1. REST コネクターの設定

次の手順を使用して、Red Hat JBoss Data Grid のリモートクライアントサーバーモードで **rest-connector** 要素を設定します。

手順12.1 リモートクライアントサーバーモード用 REST コネクターの設定

```
<subsystem xmlns="urn:infinispan:server:endpoint:8.0">
   <rest-connector cache-container="local"
                   context-path="${CONTEXT_PATH}"
                   security-domain="${SECURITY_DOMAIN}"
                   auth-method="${METHOD}"
                   security-mode="${MODE}" />
</subsystem>
```

rest-connector 要素は、REST コネクターの設定情報を指定します。

1. ***cache-container*** パラメーターは、REST コネクターで使用されるキャッシュコンテナーを指定します。これは必須パラメーターです。
2. ***context-path*** パラメーターは、REST コネクターのコンテキストパスを指定します。このパラメーターのデフォルト値は空の文字列 (**""**) です。これはオプションパラメーターです。

3. ***security-domain*** パラメーターは、REST エンドポイントへのアクセスを認証するためにセキュリティーサブシステムで宣言された指定済みドメインを使用することを指定します。これはオプションパラメーターです。このパラメーターが省略されると、認証は実行されません。

4. ***auth-method*** パラメーターは、エンドポイントのクレデンシャルを取得するために使用するメソッドを指定します。このパラメーターのデフォルト値は **BASIC** です。サポートされる別の値には **BASIC**、**DIGEST**、および **CLIENT-CERT** があります。これはオプションパラメーターです。

5. ***security-mode*** パラメーターは、書き込み操作 (PUT、POST、DELETE など) または読み取り操作 (GET や HEAD など) に対してのみ認証が必要かどうかを指定します。このパラメーターの有効な値は **WRITE** (書き込み操作のみを認証する場合) または **READ_WRITE** (読み書き操作を認証する場合) です。このパラメーターのデフォルト値は **READ_WRITE** です。

バグを報告する

第13章 MEMCACHED インターフェース

Memcached は、データベース駆動 Web サイトの応答時間と操作時間を改善するために使用されるインメモリーキャッシングシステムです。Memcached キャッシングシステムは、Memached プロトコルと呼ばれるテキストベースのクライアントサーバーキャッシングプロトコルを定義します。Memcached プロトコルはインメモリーオブジェクトを使用するか、(最後の手段として) 特殊な memcached データベースなどの永続ストアに渡します。

Red Hat JBoss Data Grid は、Memcached プロトコルを使用するサーバーを提供し、JBoss Data Grid と別に Memcached を使用する必要はありません。また、JBoss Data Grid のクラスタリング機能により、データフェールオーバー機能は Memcached で提供されるものよりも優れています。

バグを報告する

13.1. MEMCACHED サーバーについて

Red Hat JBoss Data Grid には、memcached プロトコルを実装するサーバーモジュールが含まれます。これにより、memcached クライアントは1 つまたは複数の JBoss Data Grid ベース memcached サーバーと対話できるようになります。

サーバーは以下のいずれかになります。

- スタンドアロン。各サーバーは、他の memcached サーバーと通信せずに独立して動作します。
- クラスター。サーバーはデータを他の memcached サーバーにレプリケートおよび分散します。

バグを報告する

13.2. MEMCACHED 統計

以下の表には、Red Hat JBoss Data Grid で memcached プロトコルを使用して利用できる有効な統計のリストが含まれます。

表13.1 memcached 統計

統計	データタイプ	説明
uptime	32 ビット符号なし整数。	memcached インスタンスが利用可能であり、実行されている時間 (秒数単位) を含みます。
time	32 ビット符号なし整数。	現在の時間を含みます。
version	文字列	現在のバージョンを含みます。
curr_items	32 ビット符号なし整数。	インスタンスが現在格納しているアイテムの数を含みます。
total_items	32 ビット符号なし整数。	存続期間中にインスタンスにより格納されたアイテムの合計数を含みます。

統計	データタイプ	説明
cmd_get	64 ビット符号なし整数	get 操作要求 (データ取得要求) の合計数を含みます。
cmd_set	64 ビット符号なし整数	設定された操作要求 (データ格納要求) の合計数を含みます。
get_hits	64 ビット符号なし整数	要求されたキーにあるキーの数を含みます。
get_misses	64 ビット符号なし整数	要求されたキーにないキーの数を含みます。
delete_hits	64 ビット符号なし整数	削除するキー (特定され正常に削除されたキー) の数を含みます。
delete_misses	64 ビット符号なし整数	削除するキー (特定されず、削除できなかったキー) の数を含みます。
incr_hits	64 ビット符号なし整数	増分するキー (特定され正常に増分されたキー) の数を含みます。
incr_misses	64 ビット符号なし整数	増分するキー (特定されず、増分できなかったキー) の数を含みます。
decr_hits	64 ビット符号なし整数	減分するキー (特定され正常に減分されたキー) の数を含みます。
decr_misses	64 ビット符号なし整数	減分するキー (特定されず、減分できなかったキー) の数を含みます。
cas_hits	64 ビット符号なし整数	比較し、スワップするキー (特定され正常に比較およびスワップされたキー) の数を含みます。
cas_misses	64 ビット符号なし整数	比較し、スワップするキー (特定されず、比較およびスワップされなかったキー) の数を含みます。
cas_badval	64 ビット符号なし整数	比較およびスワップが行われたが、元の値が提供された値に一致しなかったキーの数を含みます。
evictions	64 ビット符号なし整数	実行されたエビクションコールの数を含みます。

統計	データタイプ	説明
bytes_read	64 ビット符号なし整数	ネットワークからサーバーが読み取ったバイトの合計数を含みます。
bytes_written	64 ビット符号なし整数	ネットワークからサーバーが書き込んだバイトの合計数を含みます。

バグを報告する

13.3. MEMCACHED インターフェースコネクター

以下により、**memcached** ソケットバインディングを使用して Memcached サーバーが有効になり、**local** コンテナーで宣言された **memcachedCache** キャッシュが公開され、他のすべての設定にデフォルト値が使用されます。

```
<memcached-connector socket-binding="memcached"
       cache-container="local"/>
```

Memcached プロトコルの制限のため、1 つのコネクターで公開できるキャッシュは 1 つだけです。複数のキャッシュを公開するには、異なるソケットバインディングで追加の memcached コネクターを宣言します。「Memcached コネクターの設定」を参照してください。

バグを報告する

13.3.1. Memcached コネクターの設定

以下の手順は、Red Hat JBoss Data Grid のリモートクライアントサーバーモードの **connectors** 要素内にある memcached コネクターを設定するために使用する属性を示しています。

手順13.1 リモートクライアントサーバーモードでの Memcached コネクターの設定

memcached-connector 要素は、memcached で使用する設定要素を定義します。

```
<subsystem xmlns="urn:infinispan:server:endpoint:8.0">
<memcached-connector socket-binding="memcached"
                     cache-container="local"
                     worker-threads="${VALUE}"
                     idle-timeout="{VALUE}"
                     tcp-nodelay="{TRUE/FALSE}"
                     send-buffer-size="{VALUE}"
                     receive-buffer-size="${VALUE}" />
</subsystem>
```

1. ***socket-binding*** パラメーターは、memcached コネクターで使用されるソケットバインディングポートを指定します。これは必須パラメーターです。
2. ***cache-container*** パラメーターは、memcached コネクターで使用されるキャッシュコンテナーを指定します。これは必須パラメーターです。

3. ***worker-threads*** パラメーターは、memcached コネクターで利用可能なワーカースレッドの数を指定します。このパラメーターのデフォルト値は、160 です。これはオプションパラメーターです。

4. ***idle-timeout*** パラメーターは、接続がタイムアウトするまでコネクターがアイドル状態のままになる時間 (ミリ秒単位) を指定します。このパラメーターのデフォルト値は **-1** です (タイムアウト期間が設定されません)。これは、オプションパラメーターです。

5. ***tcp-no-delay*** パラメーターは、TCP パケットが遅延され一括して送信されるかを指定します。このパラメーターの有効な値は **true** と **false** になります。このパラメーターのデフォルト値は、**true** です。これはオプションパラメーターです。

6. ***send-buffer-size*** パラメーターは、memcached コネクターの送信バッファーのサイズを指定します。このパラメーターのデフォルト値は TCP スタックバッファーのサイズです。これはオプションパラメーターです。

7. ***receive-buffer-size*** パラメーターは、memcached コネクターの受信バッファーのサイズを指定します。このパラメーターのデフォルト値は TCP スタックバッファーのサイズです。これはオプションパラメーターです。

バグを報告する

第14章 HOT ROD インターフェース

14.1. HOT ROD について

Hot Rod は、Red Hat JBoss Data Grid で使用されるバイナリー TCP クライアントサーバープロトコルであり、Memcached などの他のクライアントサーバープロトコルの欠点を解消するために作成されました。

Hot Rod はサーバークラスターでフェールオーバーを行い、トポロジーが変更されます。Hot Rod は、クラスタートポロジーに関する更新をクライアントに定期的に提供することによりこれを行います。

Hot Rod では、クライアントはパーティション化された、または分散された JBoss Data Grid サーバークラスターで要求をスマートにルーティングできます。この実行において、Hot Rod ではクライアントはキーを格納するパーティションを決定し、キーがあるサーバーと直接通信できます。この機能は、クライアントでクラスタートポロジーを更新する Hot Rod に依存し、クライアントはサーバーと同じ一貫性のあるハッシュアルゴリズムを使用します。

JBoss Data Grid には、Hot Rod プロトコルを実装するサーバーモジュールが含まれます。Hot Rod プロトコルを使用すると、他のテキストベースのプロトコルに比べて、クライアントとサーバーの対話が促進され、クライアントが負荷分散、フェールオーバー、およびデータ場所運用に関する決定を行えるようになります。

バグを報告する

14.2. MEMCACHED ではなく HOT ROD を使用する利点

Red Hat JBoss Data Grid は、クライアントがリモートクライアントサーバー環境のサーバーと対話することを可能にするプロコトルを提供します。memcached または Hot Rod のいずれを使用するか決定する場合は、以下のことを考慮する必要があります。

Memcached

memcached プロコトルでは、サーバーエンドポイントが **memcached text wire protocol** を使用します。**memcached wire protocol** の利点は、一般的に使用されていることであり、これはほとんどのプラットフォームで利用できます。memcached を使用する場合は、クラスタリング、スケーラビリティーの状態共有、および高可用性を含む JBoss Data Grid のすべての機能を利用できます。

ただし、memcached プロトコルには dynamicity がなく、クラスターのいずれかのノードで障害が発生したときにクライアント上のサーバーノードのリストを手動で更新する必要があります。また、memcached クライアントはクラスターのデータの場所を認識しません。つまり、クライアントは非所有者のノードからデータを要求し、データをクライアントに返す前に、そのノードから実際の所有者への追加の要求のペナルティーが発生します。この結果、Hot Rod プロトコルは memcached よりも優れたパフォーマンスを提供できます。

Hot Rod

JBoss Data Grid の Hot Rod プロトコルは、memcached のすべての機能を提供するバイナリーワイヤープロトコルであり、優れたスケーリング、持続性、および弾力性を提供します。

Hot Rod プロトコルは、リモートキャッシュで各ノードのホスト名とポートを必要としませんが、memcached ではこれらのパラメーターを指定する必要があります。Hot Rod クライアントはクラスター化された Hot Rod サーバーのトポロジーの変更を自動的に検出します。新しいノードがクラスターに参加したり、クラスターから脱退したりすると、クライアントは Hot Rod サーバートポロジービューを更新します。この結果、Hot Rod では、設定と保守が容易になり、動的なロードバランシングとフェイルオーバーの利点が提供されます。

また、Hot Rod ワイヤープロトコルは分散キャッシュに接続するときにスマートルーティングを使用します。この場合に、サーバーノードとクライアント間で一貫したハッシュアルゴリズムが共有され、memcached よりも高速な読み取りおよび書き込み機能が提供されます。

警告

Hot Rod を介して JCache を使用する場合、リモートクラスター化キャッシュは作成できません (操作がクラスター全体ではなく単一のノードで実行されるため)。ただし、キャッシュがクラスターで作成された場合は、`cacheManager.getCache` メソッドを使用してキャッシュを取得できます。

設定ファイルまたは CLI のいずれかを使用してキャッシュを作成することが推奨されます。

バグを報告する

14.3. HOT ROD ハッシュ機能

Hot Rod は、サーバーと同じアルゴリズムを使用します。Hot Rod クライアントは常に、所有者のリストの最初のノードであるキーのプライマリー所有者に接続します。Red Hat JBoss Data Grid での整合性のあるハッシュの詳細については、「ディストリビューションモードの一貫したハッシュアルゴリズム」を参照してください。

バグを報告する

14.4. HOT ROD インターフェースコネクター

以下により、**hotrod** ソケットバインディングを使用して Hot Rod サーバーが有効になります。

```
<hotrod-connector socket-binding="hotrod"
    cache-container="local" />
```

コネクターは、サポートするトポロジーキャッシュをデフォルト設定で作成します。これらの設定は、以下のように ***<topology-state-transfer />*** 子要素をコネクターに追加することにより、調整できます。

```
<hotrod-connector socket-binding="hotrod"
    cache-container="local">
   <topology-state-transfer lazy-retrieval="false"
          lock-timeout="1000"
          replication-timeout="5000" />
</hotrod-connector>
```

Hot Rod コネクターは追加の設定で調整できます。Hot Rod コネクターの設定方法の詳細については、「Hot Rod コネクターの設定」を参照してください。

注記

Hot Rod コネクターは SSL を使用してセキュアにできます。詳細については、『Developer Guide』の「Hot Rod Authentication Using SASL」を参照してください。

バグを報告する

14.4.1. Hot Rod コネクターの設定

次の手順は、Red Hat JBoss Data Grid のリモートクライアントサーバーモードで Hot Rod コネクターを設定するために使用する属性を示しています。**hotrod-connector** 要素と **topology-state-transfer** 要素は、次の手順に基づいて設定する必要があります。

手順14.1 リモートクライアントサーバーモード用 Hot Rod コネクターの設定

```
<subsystem xmlns="urn:infinispan:server:endpoint:8.0">
 <hotrod-connector socket-binding="hotrod"
     cache-container="local"
     worker-threads="${VALUE}"
     idle-timeout="${VALUE}"
     tcp-nodelay="${TRUE/FALSE}"
     send-buffer-size="${VALUE}"
     receive-buffer-size="${VALUE}" >
  <topology-state-transfer lock-timeout"="${MILLISECONDS}"
     replication-timeout="${MILLISECONDS}"
     external-host="${HOSTNAME}"
     external-port="${PORT}"
     lazy-retrieval="${TRUE/FALSE}"  />
 </hotrod-connector>
</subsystem>
```

1. **hotrod-connector** 要素は、Hot Rod で使用する設定要素を定義します。

    a. ***socket-binding*** パラメーターは、Hot Rod コネクターで使用されるソケットバインディングポートを指定します。これは必須パラメーターです。

    b. ***cache-container*** パラメーターは、Hot Rod コネクターで使用されるキャッシュコンテナーを指定します。これは必須パラメーターです。

    c. ***worker-threads*** パラメーターは、Hot Rod コネクターで利用可能なワーカースレッドの数を指定します。このパラメーターのデフォルト値は、**160** です。これはオプションパラメーターです。

    d. ***idle-timeout*** パラメーターは、接続がタイムアウトするまでコネクターがアイドル状態のままになる時間 (ミリ秒単位) を指定します。このパラメーターのデフォルト値は **-1** です (タイムアウト期間が設定されません)。これは、オプションパラメーターです。

    e. ***tcp-no-delay*** パラメーターは、TCP パケットが遅延され一括して送信されるかを指定します。このパラメーターの有効な値は **true** と **false** になります。このパラメーターのデフォルト値は、**true** です。これはオプションパラメーターです。

    f. ***send-buffer-size*** パラメーターは、Hot Rod コネクターの送信バッファーのサイズを指定します。このパラメーターのデフォルト値は TCP スタックバッファーのサイズです。これはオプションパラメーターです。

g. ***receive-buffer-size*** パラメーターは、Hot Rod コネクターの受信バッファーのサイズを指定します。このパラメーターのデフォルト値は TCP スタックバッファーのサイズです。これはオプションパラメーターです。

2. **topology-state-transfer** 要素は、Hot Rod コネクターのトポロジー状態転送設定を指定します。この要素は **hotrod-connector** 要素内でのみ使用できます。

   a. ***lock-timeout*** パラメーターは、ロックを取得しようとする操作がタイムアウトする時間を指定します。このパラメーターのデフォルト値は **10** 秒です。これはオプションパラメーターです。

   b. ***replication-timeout*** パラメーターは、レプリケーション操作がタイムアウトする時間 (ミリ秒単位) を指定します。このパラメーターのデフォルト値は **10** 秒です。これはオプションパラメーターです。

   c. ***external-host*** パラメーターは、トポロジー情報にリストされたクライアントに Hot Rod サーバーが送信するホスト名を指定します。このパラメーターのデフォルト値は、ホストアドレスです。これはオプションパラメーターです。

   d. ***external-port*** パラメーターは、トポロジー情報にリストされたクライアントに Hot Rod サーバーが送信するポートを指定します。このパラメーターのデフォルト値は、設定されたポートです。これはオプションパラメーターです。

   e. ***lazy-retrieval*** パラメーターは、Hot Rod コネクターが取得操作をレイジーに実行するかどうかを指定します。このパラメーターのデフォルト値は **true** です。これはオプションパラメーターです。

バグを報告する

パート VI. キャッシュのロックのセットアップ

第15章 ロック

Red Hat JBoss Data Grid は、ダーティー読み出し (トランザクションが古くなった値に変更を適用する前にその古くなった値を読み出す) と反復不可能読み出しを防ぐためのロックメカニズムを提供します。

バグを報告する

15.1. ロックの設定 (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は、ロックは、キャッシュタグ (たとえば、**invalidation-cache**、**distributed-cache**、 **replicated-cache** または **local-cache**) 内で **locking** 要素を使用して設定されます。

注記

リモートクライアントサーバーモードのデフォルトの分離モードは **READ_COMMITTED** です。分離モードを明示的に指定するために ***isolation*** 属性が含まれる場合、この属性は無視され、警告がスローされて、デフォルト値が代わりに使用されます。

以下は、Red Hat JBoss Data Grid のリモートクライアントサーバーモードにおけるデフォルトキャッシュについての基本的なロック設定の手順例です。

手順15.1 ロックの設定 (リモートクライアントサーバーモード)

```
<distributed-cache>
 <locking acquire-timeout="30000"
          concurrency-level="1000"
          striping="false" />
          <!-- Additional configuration here -->
</distributed-cache>
```

1. ***acquire-timeout*** パラメーターは、ロックの取得がタイムアウトになった後のミリ秒数を指定します。
2. ***concurrency-level*** パラメーターは、LockManager によって使用されるロックストライプの数を定義します。
3. ***striping*** パラメーターは、ロックストライピングがローカルキャッシュに使用されるかどうかを指定します。

バグを報告する

15.2. ロックの設定 (ライブラリーモード)

ライブラリーモードの場合、**locking** 要素とそのパラメーターは、**default** 要素内で設定され、それぞれの名前付きキャッシュでは、**local-cache** 要素内で設定されます。この設定の例は、以下の通りです。

手順15.2 ロックの設定 (ライブラリーモード)

```
<local-cache name="default">
```

```
  <locking concurrency-level="${VALUE}"
   isolation="${LEVEL}"
   acquire-timeout="${TIME}"
   striping="${TRUE/FALSE}"
   write-skew="${TRUE/FALSE}" />
</local-cache>
```

1. ***concurrencyLevel*** パラメーターは、ロックコンテナーの平行性レベルを指定します。データグリッドと通信する同時スレッドの数に従ってこの値を設定します。

2. ***isolation*** パラメーターはキャッシュの分離レベルを指定します。有効な分離レベルは、**READ_COMMITTED** および **REPEATABLE_READ** です。分離レベルについてさらに詳しくは、「分離レベルについて」を参照してください。

3. ***acquire-timeout*** パラメーターは、ロック取得の試行がタイムアウトになった後の時間 (ミリ秒単位) を指定します。

4. ***striping*** パラメーターは、ロックを必要とするすべてのエントリーに対して、共有ロックのプールを維持するかどうかを指定します。**FALSE** に設定されると、ロックがキャッシュ内のそれぞれのエントリーに対して作成されます。さらに詳しくは、「ロックストライピングについて」を参照してください。

5. ***write-skew*** パラメーターは、***isolation*** が **REPEATABLE_READ** に設定された場合にのみ有効です。このパラメーターが **FALSE** に設定された場合、書き込み時に使用中のエントリーと基礎となるエントリー間の相違があると、使用中のエントリーが基礎となるエントリーを上書きします。このパラメーターが **TRUE** に設定された場合は、このような競合 (書き込みの競合など) により例外がスローされます。***write-skew*** パラメーターは、**OPTIMISTIC** トランザクションでのみ使用でき、**SIMPLE** バージョン管理スキームを使用してエントリーバージョン管理を有効にする必要があります。

バグを報告する

15.3. ロックのタイプ

15.3.1. 楽観的ロックについて

楽観的ロックは、ロックの取得をトランザクションの準備時間まで延期することで複数のトランザクションが同時に終了するようにします。

楽観的モードは、複数のトランザクションが競合せずに終了するようにします。トランザクションは、他のトランザクションロックがクリアされるまで待機しなくてもコミットできるため、同時に実行されている複数のトランザクション間でほとんど競合が発生しない場合に適しています。***write-skew*** が有効になっている場合、トランザクションが終了する前に、競合する変更が 1 つ以上データに加えられると、楽観的ロックモードのトランザクションはロールバックします。

バグを報告する

15.3.2. 悲観的ロックについて

悲観的ロック (Pessimistic locking) は一括ロック (Eager locking) とも呼ばれます。

悲観的ロックは、クラスター全体のロックを各書き込み操作に適用することにより、複数のトランザクションでキーの値が変更されないようにします。ロックは、コミットまたはロールバックによってトランザクションが完了したときにのみ開放されます。

悲観的モードはキーで競合が発生し、効率が悪くなったり、予期されないロールバック操作が発生する場合に使用されます。

バグを報告する

15.3.3. 悲観的ロックのタイプ

Red Hat JBoss Data Grid には、明示的な悲観的ロックと暗黙的な悲観的ロックが含まれています。

- 明示的な楽観的ロックは、JBoss Data Grid Lock API を使用してトランザクションの期間にキャッシュユーザーがキャッシュキーを明示的にロックできるようにします。ロック呼び出しは、クラスターの全ノードにおいて、指定されたキャッシュキー上でロックの取得を試みます。ロックはすべてコミットまたはロールバックフェーズ中に開放されます。

- 暗黙的な悲観的ロックは、キャッシュキーが変更操作のためアクセスされる時にキャッシュキーがバックグラウンドでロックされるようにします。暗黙的な悲観的ロックを使用すると、各変更操作に対してキャッシュキーが確実にローカルでロックされるよう JBoss Data Grid がチェックします。ロックされていないキャッシュキーが見つかると、JBoss Data Grid はロックされていないキャッシュキーのロックを取得するため、クラスターワイドのロックを要求します。

バグを報告する

15.3.4. 明示的な悲観的ロックの例

以下は、キャッシュノードの 1 つで実行されるトランザクションの明示的な悲観的ロックの例になります。

手順15.3 明示的な悲観的ロックによるトランザクション

```
tx.begin()
cache.lock(K)
cache.put(K,V5)
tx.commit()
```

1. 行 **`cache.lock(K)`** が実行されると、**K** でクラスター全体のロックが取得されます。
2. 行 **`cache.put(K,V5)`** が実行されると、取得の成功が保証されます。
3. 行 **`tx.commit()`** が実行されると、この処理のために保持されたロックが開放されます。

バグを報告する

15.3.5. 暗黙的な悲観的ロックの例

以下は、キャッシュノードの 1 つで実行されるトランザクションを使用する暗黙的な悲観的ロックの例になります。

手順15.4 暗黙的な悲観的ロックによるトランザクション

```
tx.begin()
cache.put(K,V)
cache.put(K2,V2)
```

```
cache.put(K,V5)
tx.commit()
```

1. 行 **cache.put(K,V)** が実行されると、**K** でクラスター全体のロックが取得されます。
2. 行 **cache.put(K2,V2)** が実行されると、**K2** でクラスター全体のロックが取得されます。
3. 行 **cache.put(K,V5)** が実行されると、**K** のクラスター全体のロックは以前取得されたため、ロックの取得は実行できませんが、**put** 操作は引き続き実行されます。
4. 行 **tx.commit()** が実行されると、このトランザクションのために保持されたすべてのロックが開放されます。

バグを報告する

15.3.6. ロックモードの設定 (リモートクライアントサーバーモード)

Red Hat JBoss Data Grid のリモートクライアントサーバーモードでロックモードを設定するには、以下のように **transaction** 要素を使用します。

```
<transaction locking="{OPTIMISTIC/PESSIMISTIC}" />
```

バグを報告する

15.3.7. ロックモードの設定 (ライブラリーモード)

Red Hat JBoss Data Grid のライブラリーモードでは、ロックモードが以下のように **transaction** 要素内で設定されます。

```
<transaction transaction-manager-lookup="{TransactionManagerLookupClass}"
      mode="{NONE, BATCH, NON_XA, NON_DURABLE_XA, FULL_XA}"
      locking="{OPTIMISTIC,PESSIMISTIC}">
</transaction>
```

トランザクションキャッシュに使用されるロックモードを設定するには、 ***locking*** 値を **OPTIMISTIC** または **PESSIMISTIC** に設定します。

バグを報告する

15.4. ロック操作

15.4.1. LockManager について

LockManager コンポーネントは、書き込み処理が始まる前にエントリーをロックします。**LockManager** は **LockContainer** を使用してロックを見つけたり、ロックを保持および作成します。JBoss Data Grid が内部的に使用する **LockContainers** には 2 つのタイプがあり、その決定は ***useLockStriping*** 設定に依存します。最初のタイプはロックストライピングのサポートを提供し、2 つ目のタイプはエントリーごとに 1 つのロックをサポートします。

関連トピック:

- *16章 ロックストライピングのセットアップ*

バグを報告する

15.4.2. ロックの取得について

Red Hat JBoss Data Grid はデフォルトでリモートロックをレイジーに取得します。ローカルでトランザクションを実行しているノードはロックを取得し、他のクラスターノードは 2 相準備/コミットフェーズに関与するキャッシュキーをロックしようとします。JBoss Data Grid では、明示的または暗示的な悲観的ロックにてキャッシュキーをロックすることが可能です。

バグを報告する

15.4.3. 平行性レベルについて

平行性は、データグリッドと同時に対話するスレッド数に言及する概念です。Red Hat JBoss Data Grid では、平行性レベルは、ロックコンテナー内で使用される同時スレッドの数を指します。

JBoss Data Data Grid では、平行性レベルがストライピングされたロックコンテナーのサイズを決定します。さらに、平行性レベルは **`DataContainers`** 内部のコレクションなど、関連するすべての JDK **`ConcurrentHashMap`** ベースのコレクションを調整します。

バグを報告する

第16章 ロックストライピングのセットアップ

16.1. ロックストライピングについて

ロックストライピングは、キャッシュにあるロック (固定サイズ) の共有コレクションからロックを割り当てます。ロック割り当ては各エントリーのキーに対するハッシュコードに基づきます。ロックストライピングは、オーバーヘッドが固定された非常にスケーラブルなロックメカニズムを提供しますが、同じロックによって関係ないエントリーがブロックされることがあります。

ロックストライピングは、Red Hat JBoss Data Grid ではデフォルトで無効になります。ロックストライピングが無効であると、各エントリーに対して新しいロックが作成されます。この方法では、より大きな同時スループットが提供されますが、メモリー使用量の増加やガベージコレクションチャーンなどのデメリットも発生します。

バグを報告する

16.2. ロックストライピングの設定 (リモートクライアントサーバーモード)

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ロックストライピングは、***striping*** 要素を **true** に設定して有効になります。

例16.1 ロックストライピング (リモートクライアントサーバーモード)

```
<locking acquire-timeout="20000"
  concurrency-level="500"
  striping="true" />
```

注記

リモートクライアントサーバーモードのデフォルトの分離モードは **READ_COMMITTED** です。分離モードを明示的に指定するために ***isolation*** 属性が含まれる場合、この属性は無視され、警告がスローされて、デフォルト値が代わりに使用されます。

locking 要素は以下の属性を使用します。

- ***acquire-timeout*** 属性は、ロックの取得を試行する最大時間を指定します。この属性のデフォルト値は **10000** ミリ秒です。
- ***concurrency-level*** 属性は、ロックコンテナーの同時実行レベルを指定します。JBoss Data Grid と通信する同時スレッドの数に従ってこの値を調整します。この属性のデフォルト値は **32** です。
- ***striping*** 属性は、ロックを必要とするすべてのエントリーに対してロックの共有プールを維持するかどうかを指定します (**true**)。**false** に設定された場合は、各エントリーに対してロックが作成されます。ロックストライピングによりメモリーフットプリントが制御され、システムでの同時実行性を削減できます。この属性のデフォルト値は **false** です。

バグを報告する

16.3. ロックストライピングの設定 (ライブラリーモード)

ロックストライピングは、Red Hat JBoss Data Grid ではデフォルトで無効にされています。JBoss Data Grid のライブラリーモードでのロックストライピングの設定は、以下の手順で示されたように ***striping*** パラメーターを使用して行います。

手順16.1 ロックストライピングの設定 (ライブラリーモード)

```
<local-cache>
  <locking concurrency-level="${VALUE}"
    isolation="${LEVEL}"
    acquire-timeout="${TIME}"
    striping="${TRUE/FALSE}"
    write-skew="${TRUE/FALSE}" />
</local-cache>
```

1. ***concurrency-level*** は、ロックストライピングが有効な場合に使用される共有ロックコレクションのサイズを指定するために使用されます。
2. ***isolation*** パラメーターは、キャッシュの分離レベルを指定します。有効な分離レベルは **READ_COMMITTED** と **REPEATABLE_READ** です。
3. ***acquire-timeout*** パラメーターは、ロック取得の試行がタイムアウトになった後の時間 (ミリ秒単位) を指定します。
4. ***striping*** パラメーターは、ロックを必要とするすべてのエントリーに対して、共有ロックのプールを維持するかどうかを指定します。**FALSE** に設定されると、ロックがキャッシュ内のそれぞれのエントリーに対して作成されます。**TRUE** に設定されると、ロックストライピングは有効にされ、共有ロックは必要に応じてプールから使用されます。
5. ***write-skew*** チェックは、異なるトランザクションのエントリーへの変更によってトランザクションのロールバックを実行するかどうかを決定します。書き込みスキューを true に設定するには、***isolation_level*** を **REPEATABLE_READ** に設定する必要があります。***write-skew*** および ***isolation_level*** のデフォルト値はそれぞれ **FALSE** と **READ_COMMITTED** です。***write-skew*** パラメーターは、**OPTIMISTIC** トランザクションでのみ使用でき、**SIMPLE** バージョン管理スキームを使用してエントリーバージョン管理を有効にする必要があります。

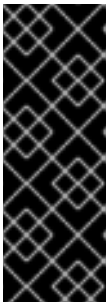
バグを報告する

第17章 分離レベルのセットアップ

17.1. 分離レベルについて

分離レベルは、リーダーが同時書き込みの読み取りを実行できるタイミングを決定します。Red Hat JBoss Data Grid で提供される分離モードは ***READ_COMMITTED*** と ***REPEATABLE_READ*** の 2 つです。

- **READ_COMMITTED**。この分離レベルはさまざまな要件に適用されます。これは、リモートクライアントサーバーおよびライブラリーモードでのデフォルト値です。
- **REPEATABLE_READ**。



重要

リモートクライアントサーバーモードのロックに有効な唯一の値はデフォルトの **READ_COMMITTED** 値です。***isolation*** 値で明示的に指定された値は無視されます。

locking 要素が設定に存在しない場合、デフォルトの分離値は **READ_COMMITTED** です。

JBoss Data Grid における分離モードの設定例については、ロックストライピングの設定例を参照してください。

- リモートクライアントサーバーモードの設定例については、「ロックストライピングの設定 (リモートクライアントサーバーモード)」を参照してください。
- ライブラリーモードの設定例については、「ロックストライピングの設定 (ライブラリーモード)」を参照してください。

バグを報告する

17.2. READ_COMMITTED について

READ_COMMITTED は Red Hat JBoss Data Grid で使用できる 2 つの分離モードの 1 つです。

JBoss Data Grid の ***READ_COMMITTED*** モードでは、書き込み操作はデータ自体ではなくデータのコピーとして作成されます。書き込み操作は他のデータの書き込みをブロックしますが、書き込みは読み出し操作をブロックしません。そのため、***READ_COMMITTED*** と ***REPEATABLE_READ*** の両モードは、書き込み操作がいつ発生するかに関係なく、いつでも読み取り操作を許可します。

READ_COMMITTED モードでは、読み取りの合間にデータを変更する別のトランザクションでの書き込み操作のため、トランザクション内の複数の読み取りで異なる結果が返されることがあります。 これは、反復不可能読み取りと呼ばれ、***REPEATABLE_READ*** モードでは回避されます。

バグを報告する

17.3. REPEATABLE_READ について

REPEATABLE_READ は Red Hat JBoss Data Grid で使用できる 2 つの分離モードの 1 つです。

従来、***REPEATABLE_READ*** は読み取り操作中の書き込み操作や、書き込み操作時の読み取り操作を許可しません。これにより、単一のトランザクションの同じ行に 2 つの読み取り操作があるのに取得した値

が異なる場合に発生する「反復不可能読み取り」が起こらないようにします (この原因は 2 つの読み取り操作の間に値を変更する書き込み操作にあると考えられます) 。

JBoss Data Grid の ***REPEATABLE_READ*** 分離モードは、変更が発生する前にエントリーの値を保存します。これにより、同じエントリーの 2 番目の読み取り操作は変更された新しい値ではなく、保存された値を読み取るため、「反復不可能読み取り」の発生を防ぐことができます。そのため、読み取りの間に別のトランザクションで書き込み操作が発生しても、1 つのトランザクションで 2 つの読み取り操作によって取得される 2 つの値は常に同じになります。

バグを報告する

パート VII. キャッシュストアのセットアップと設定

第18章 キャッシュストア

キャッシュストアは、Red Hat JBoss Data Grid を永続データストアに接続します。キャッシュストアは個別のキャッシュに関連付けられます。同じキャッシュマネージャーに接続された個々のキャッシュには、異なるキャッシュストア設定を指定できます。

注記

クラスター化キャッシュが共有されないキャッシュストアで設定された場合 (***shared*** が **false** に設定された場合) は、ノードの参加時に、クラスターから削除された可能性がある古いエントリーがストアにまだ存在し、再び現れることがあります。

バグを報告する

18.1. キャッシュローダーとキャッシュライター

永続ストアとの統合は、**org.infinispan.persistence.spi** にある以下の SPI を介して行われます。

- **CacheLoader**
- **CacheWriter**
- **AdvancedCacheLoader**
- **AdvancedCacheWriter**

CacheLoader と **CacheWriter** は、ストアに対して読み書きを行う基本的なメソッドを提供します。**CacheLoader** は、必要なデータがキャッシュにない場合にデータストアからデータを取得します。**CacheWriter** は、キャッシュのエビクション時にエントリーのパッシベーションとアクティベーションを強制実行するために使用されます。

AdvancedCacheLoader と **AdvancedCacheWriter** は、基礎となるストレージを一括で処理する並列反復、失効したエントリーの削除、クリアおよびサイズ指定などの操作を提供します。

org.infinispan.persistence.file.SingleFileStore を使用すると、独自のストア実装を簡単に作成できます。

注記

JBoss Data Grid では古い API (**CacheLoader**、**CacheStore** により拡張) が使用されていました (これは現時点でも利用可能です)。

バグを報告する

18.2. キャッシュストアの設定

18.2.1. キャッシュストアの設定

キャッシュストアはチェーンで設定されます。キャッシュの読み取り操作は、データの有効な null 以外の要素が見つかるまで、設定される順番でそれぞれのキャッシュストアをチェックします。書き込み操作は、***ignoreModifications*** 要素が特定のキャッシュストアに対して **"true"** にされない限り、

すべてのキャッシュストアに影響を与えます。

バグを報告する

18.2.2. XML を使用したキャッシュストアの設定 (ライブラリーモード)

次の例は、JBoss Data Grid のライブラリーモードで XML を使用したキャッシュストアの設定を示しています。

```
<persistence passivation="false">
   <file-store shared="false"
               preload="true"
               fetch-state="true"
               purge-startup="false"
               singleton="true"
               location="${java.io.tmpdir}" >
      <write-behind enabled="true"
             flush-lock-timeout="15000"
             thread-pool-size="5" />
   </singleFile>
</persistence>
```

この設定例で使用された要素とパラメーターの詳細については、「キャッシュストア設定の詳細 (ライブラリモード)」を参照してください。

バグを報告する

18.2.3. SKIP_CACHE_LOAD フラグについて

Red Hat JBoss Data Grid のリモートクライアントモードでは、キャッシュがキャッシュストアからプリロードされ、エビクションが無効な場合に、読み取り要求がメモリーに移動します。読み取り要求時にエントリーがメモリー内にないと、キャッシュストアがアクセスされ、読み取りパフォーマンスが影響を受けることがあります。

キーがメモリーにない場合にキャッシュストアの参照を回避するには、**SKIP_CACHE_LOAD** フラグを使用します。

バグを報告する

18.2.4. SKIP_CACHE_STORE フラグについて

SKIP_CACHE_STORE フラグが使用されると、キャッシュストアは指定されたキャッシュ操作に対して考慮されません。このフラグは、関連付けられたキャッシュストアからキャッシュを取得せずにエントリーがキャッシュ内にあるかどうかを判別できると同時に、設定済みのキャッシュストアにキャッシュを組み込まずにエントリーをキャッシュに配置する際に役立ちます。

バグを報告する

18.2.5. SKIP_SHARED_CACHE_STORE フラグについて

SKIP_SHARED_CACHE_STORE フラグが有効にされると、共有キャッシュストアは指定されたキャッシュ操作に対して考慮されません。このフラグは、共有キャッシュストアからキャッシュを取得せずにエントリーがキャッシュ内にあるかどうかを判別できると同時に、共有キャッシュストアにキャッシュを組み込まずにエントリーをキャッシュに配置する際に役立ちます。

バグを報告する

18.3. 要求キャッシュストア

共有キャッシュストアとは、複数のキャッシュインスタンスによって共有されるキャッシュストアのことです。

クラスターのすべてのインスタンスが、同じ JDBC 設定を使用して同じリモート共有データベースと通信する場合は、共有キャッシュストアが便利です。このようなインスタンスにおいて、共有キャッシュストアを設定すると、さまざまなキャッシュインスタンスが同じデータをキャッシュストアに書き込もうとした場合に不必要な書き込み操作が繰り返されません。

バグを報告する

18.3.1. インバリデーションモードと共有キャッシュストア

Red Hat JBoss Data Grid のインバリデーションモードを共有キャッシュストアと共に使用すると、リモートキャッシュが共有キャッシュストアを参照して、変更されたデータを読み出すようになります。

インバリデーションモードを共有キャッシュストアと共に使用する利点には次のようなものがあります。

- 更新されたデータが含まれるレプリケーションメッセージよりも無効化メッセージはかなり小型であるため、ネットワークトラフィックが軽減されます。
- 残りのクラスターキャッシュは、必要な場合のみ変更されたデータを共有キャッシュストアよりレイジーにルックアップするため、ネットワークトラフィックがさらに軽減されます。

バグを報告する

18.3.2. キャッシュストアとキャッシュパッシベーション

Red Hat JBoss Data Grid では、エントリーのパッシベーションを強制したり、キャッシュでエビクションをアクティベートしたりするためにキャッシュストアを使用することができます。パッシベーションモードまたはアクティベーションモードが使用されると、設定されたキャッシュストアがデータストアに対して読み書きを実行します。

JBoss Data Grid でパッシベーションが無効になっている場合、要素の変更、追加、または削除が実行された後にキャッシュストアがストアの変更を永続化します。

バグを報告する

18.3.3. アプリケーションキャッシュストア登録

分離されたデプロイメントに対してアプリケーションキャッシュストアを登録する必要はありません。レイジーデシリアライゼーションを使用するとこの問題を回避できるため、Red Hat JBoss Data Grid では必須ではありません。

バグを報告する

18.4. 接続ファクトリー

Red Hat JBoss Data Grid では、すべての JDBC キャッシュストアが **`ConnectionFactory`** 実装に依存してデータベースへの接続を取得します。このプロセスは接続管理またはプーリングとも呼ばれます。

接続ファクトリーは ***ConnectionFactoryClass*** 設定属性を使用して指定することができます。
JBoss Data Grid には次の **ConnectionFactory** 実装が含まれています。

- ManagedConnectionFactory。
- SimpleConnectionFactory。
- PooledConnectionFactory。

バグを報告する

18.4.1. ManagedConnectionFactory について

ManagedConnectionFactory は、アプリケーションサーバーなどの管理された環境内での使用に適した接続ファクトリーです。この接続ファクトリーは JNDI ツリー内の設定された場所を調査でき、接続管理を **DataSource** へ委譲できます。

バグを報告する

18.4.2. SimpleConnectionFactory について

SimpleConnectionFactory は呼び出しごとにデータベース接続を作成する接続ファクトリーです。この接続ファクトリーは実稼働環境での使用向けには設計されていません。

バグを報告する

18.4.3. PooledConnectionFactory について

PooledConnectionFactory は C3P0 に基づく接続ファクトリーであり、通常は JBoss EAP などのサーブレットコンテナーを施与うしたデプロイメントではなくスタンドアロンのデプロイメント用に推奨されます。この接続ファクトリーは、ユーザーがファクトリーで生成されるすべての **DataSource** インスタンスで使用できる一連のパラメーターを定義できるようにすることにより機能できます。

バグを報告する

第19章 キャッシュストアの実装

キャッシュストアは、Red Hat JBoss Data Grid を永続データストアに接続します。キャッシュストアは個別のキャッシュに関連付けられます。同じキャッシュマネージャーに接続された個々のキャッシュには、異なるキャッシュストア設定を指定できます。

注記

クラスター化キャッシュが共有されないキャッシュストアで設定された場合 (***shared*** が **`false`** に設定された場合) は、ノードの参加時に、クラスターから削除された可能性がある古いエントリーがストアにまだ存在し、再び現れることがあります。

バグを報告する

19.1. キャッシュストアの比較

要件に 基づいてキャッシュストアを選択します。以下に、Red Hat JBoss Data Grid で利用可能なキャッシュストア間の主な違いの概要を示します。

- 単一ファイルキャッシュストアはローカルのファイルキャッシュストアです。これにより、クラスター化されたキャッシュの各ノードに対してデータがローカルで永続化されます。単一ファイルキャッシュストアは、優れた読み書きパフォーマンスを提供しますが、キーがメモリーに保持され、各ノードで大きいデータセットを永続化するときに使用が制限されます。詳細については、「単一ファイルキャッシュストア」を参照してください。

- LevelDB ファイルキャッシュストアは、高い読み書きパフォーマンスを提供するローカルのファイルキャッシュストアです。キーがメモリに保持される単一ファイルキャッシュストアの制限はありません。詳細については、「LevelDB キャッシュストア」を参照してください。

- JDBC キャッシュストアは、必要に応じて共有できるキャッシュストアです。使用時に、クラスター化されたキャッシュのすべてのノードは、クラスターの各ノードに対して単一のデータベースまたはローカルの JDBC データベースに永続化されます。共有キャッシュストアには、LevelDB キャッシュストアなどのローカルキャッシュストアのスケーラビリティーとパフォーマンスがありませんが、永続化データに対して単一の場所が提供されます。JDBC キャッシュストアでは、エントリーがバイナリー blob として永続化され、JBoss Data Grid 外部で読み取ることができません。詳細については、「JDBC ベースのキャッシュストア」を参照してください。

- JPA キャッシュストア (ライブラリーモードでのみサポート) は JDBC キャッシュストアのような要求キャッシュストアですが、データベースに永続化するときにスキーマ情報が保持されます。したがって、永続化されたエントリーは、JBoss Data Grid 外部で読み取ることができます。詳細については、「JPA キャッシュストア」を参照してください。

バグを報告する

19.2. キャッシュストア設定の詳細 (ライブラリモード)

以下の一覧には、JBoss Data Grid のライブラリーモードでのキャッシュストア要素の設定要素とパラメーターに関する詳細が含まれます。以下の一覧は各要素の特定のパラメーターを強調表示するためのもので、詳細の一覧についてはスキーマを参照していただくことができます。

persistence 要素

- ***`passivation`*** パラメーターは、Red Hat JBoss Data Grid がストアと対話する方法に影響を与

えます。オブジェクトがインメモリーキャッシュから削除されると、パッシベーションによりオブジェクトがシステムやデータベースなどの 2 次データストアに書き込まれます。このパラメーターの有効な値は、**true** と **false** ですが、***passivation*** はデフォルトで **false** に設定されます。

file-store 要素

- ***shared*** パラメーターは、異なるキャッシュインスタンスによってキャッシュストアが共有されていることを示します。たとえば、クラスター内のすべてのインスタンスが、同じリモートの共有データベースと通信するために同じ JDBC 設定を使用する場合があります。***shared*** は、デフォルトで **false** になります。**true** に設定すると、異なるキャッシュインスタンスによって重複データがキャッシュストアに書き込まれることが避けられます。LevelDB キャッシュストアの場合は、このパラメーターを設定から除外するか、**false** に設定する必要があります (このキャッシュストアの共有はサポートされていません)。

- ***preload*** パラメーターはデフォルトで **false** に設定されます。**true** に設定されると、キャッシュストアに保存されたデータは、キャッシュの起動時にメモリーにプリロードされます。これにより、キャッシュストアのデータが起動後すぐに利用できるようになり、データのレイジーなロードの結果としてキャッシュ操作の遅れを防ぐことができます。プリロードされたデータは、ノード上でローカルにのみに保存され、プリロードされたデータのレプリケーションや分散は行われません。Red Hat JBoss Data Grid は、エビクションのエントリーの最大設定数までの数のエントリーをプリロードします。

- ***fetch-state*** パラメーターは、キャッシュの永続ステートを取り込むかどうかを決定し、クラスターに参加する際にこれをローカルキャッシュストアに適用します。キャッシュストアが共有される場合、キャッシュが同じキャッシュストアにアクセスする際に、fetch persistent 状態は無視されます。複数のキャッシュストアでこのプロパティーが **true** に設定されている場合にキャッシュサービスを起動すると、設定の例外がスローされます。***fetch-state*** プロパティーはデフォルトでは **false** です。

- ルックアップを迅速化するために、単一ファイルキャッシュストアはキーのインデックスとファイル内のそれらの場所を保持します。このインデックスがメモリー消費の問題の原因になるのを避けるために、このキャッシュストアは、***max-entries*** パラメーターで定義される、保存されるエントリーの最大数でバインドすることができます。この制限の上限を上回る場合は、LRU アルゴリズムを使用してエントリーをインメモリーインデックスと基礎となるファイルベースのキャッシュストアの両方から永久的に削除できます。デフォルト値は **-1** で、無制限のエントリーを許可します。

- ***singleton*** パラメーターは、シングルトンストアのキャッシュストアを有効にします。SingletonStore は、クラスター内の唯一のインスタンスが基礎となるストアと通信する場合にのみ使用される委譲するキャッシュストアです。ただし、***singleton*** パラメーターは **file-store** には推奨されません。デフォルト値は **false** です。

- ***purge*** パラメーターは、起動時にキャッシュストアがパージされるかどうかを制御します。

- ***location*** 設定要素は、ストアが書き込みできるディスクの場所を設定します。

ライトビハインド要素

write-behind 要素には、キャッシュストアのさまざまな側面を設定するパラメーターが含まれます。

- ***thread-pool-size*** パラメーターは、変更をストアに同時に適用するスレッドの数を指定します。このパラメーターのデフォルト値は **1** です。

- ***flush-lock-timeout*** パラメーターは、キャッシュストアに定期的にフラッシュされる状態を保護するロックを取得するための時間を指定します。このパラメーターのデフォルト値は **1** です。

- ***modification-queue-size*** パラメーターは、非同期ストアの変更キューのサイズを指定します。基礎となるキャッシュストアがこのキューを処理するよりも速く更新される場合に、その期間において非同期ストアは同期ストアのように動作し、キューがさらに要素を許可できるようになるまでブロックします。このパラメーターのデフォルト値は **1024** 要素です。

- ***shutdown-timeout*** パラメーターは、キャッシュストアを停止するのにかかる最大時間を指定します。このパラメーターのデフォルト値は **25000** ミリ秒です。

remote-store 要素

- ***cache*** 属性は、リモート Infinispan クラスターで接続するリモートキャッシュの名前を指定します。リモートキャッシュの名前が指定されないと、デフォルトのキャッシュが使用されます。

- ***fetch-state*** 属性が **true** に設定されると、リモートキャッシュがクラスターに参加したときに永続状態が取り込まれます。複数のキャッシュストアがチェーン化されている場合、1 つのキャッシュストアだけでこのプロパティーを **true** に設定できます。この値のデフォルトは **false** です。

- ***shared*** 属性は、複数のキャッシュインスタンスがキャッシュストアを共有する場合に **true** に設定されます。これにより、複数のキャッシュインスタンスが同じ変更内容を個別に書き込むことを防ぐことができます。この属性のデフォルト値は **false** です。

- ***preload*** 属性を使用すると、キャッシュストアデータがメモリーにプリロードされ、起動後すぐにアクセス可能になります。これを **true** に設定すると、起動時間が増加するという不利な点もあります。この属性のデフォルト値は **false** です。

- ***singleton*** パラメーターは、SingletonStore の委譲するキャッシュストアを有効にします。これは、クラスター内の唯一のインスタンスが基礎となるストアと通信する場合にのみ使用されます。デフォルト値は **false** です。

- ***purge*** 属性を使用すると、キャッシュストアが起動プロセスでパージされます。この属性のデフォルト値は **false** です。

- ***tcp-no-delay*** 属性は、**TCP *NODELAY*** スタックをトリガーします。この属性のデフォルト値は **true** です。

- ***ping-on-start*** 属性は、クラスタートポロジーを取り込むために ping 要求をバックエンドサーバーに送信します。この属性のデフォルト値は **true** です。

- ***key-size-estimate*** 属性は、キーサイズの推定値を提供します。この属性のデフォルト値は **64** です。

- ***value-size-estimate*** 属性は、値をシリアライズおよびデシリアライズする時のバイトバッファーのサイズを指定します。この属性のデフォルト値は **512** です。

- ***force-return-values*** 属性は、***FORCE_RETURN_VALUE*** をすべての呼び出しに対して有効にするかどうかを設定します。この属性のデフォルト値は **false** です。

remote-server 要素

remote-store 要素内の **remote-server** 要素を作成し、サーバー情報を定義します。

- ***host*** 属性はホストアドレスを設定します。
- ***port*** 属性は、リモートキャッシュストアで使用されるポートを設定します。これはデフォルトで **11222** になります。

connection-pool 要素 (リモートストア)

- ***max-active*** パラメーターは、一度に各サーバーに設定できるアクティブな接続の最大数を示します。この属性のデフォルト値は **-1** であり、これはアクティブな接続の無限な数を示します。
- ***max-idle*** パラメーターは、一度に各サーバーに設定できるアイドル状態の接続の最大数を示します。この属性のデフォルト値は **-1** であり、これはアイドル状態の接続の無限な数を示します。
- ***max-total*** パラメーターは、組み合わされたサーバーセット内の永続的な接続の最大数を示します。この属性のデフォルト設定は **-1** であり、これは接続の無限な数を示します。
- ***min-idle-time*** パラメーターは、常に利用可能にする必要のある (各サーバーの) アイドル状態の接続の最小数のターゲット値を設定します。パラメーターが正の数値と **timeBetweenEvictionRunsMillis** > 0 に設定される場合、アイドル状態の接続のエビクションスレッドが実行されるたびに、各サーバーで ***minIdle*** アイドルインスタンスを利用できるように十分な数のアイドルインスタンスを作成しようとします。このパラメーターのデフォルト設定は **1** です。
- ***eviction-interval*** パラメーターは、アイドル接続の検査の「実行」前にエビクションスレッドをスリープ状態にする必要のある時間を示します。正の数値でない場合、エビクションスレッドは起動しません。このパラメーターのデフォルト設定は **120000** ミリ秒、つまり 2 分です。
- ***min-evictable-idle-time*** パラメーターは、アイドル時間により接続がエビクションの候補となる前にプール内でアイドル状態になる最小時間を指定します。正の数値でない場合、アイドル時間によりプールからドロップされる接続はありません。この設定は **timeBetweenEvictionRunsMillis** > 0 でない限り影響を与えません。このパラメーターのデフォルト設定は **1800000** つまり (30 分) です。
- ***test-idle*** パラメーターは、アイドル接続のエビクションの実行時に TCP パケットをサーバーに送信してアイドル状態の接続を検証するかどうかを示します。検証に失敗する接続はプールからドロップします。この設定は **timeBetweenEvictionRunsMillis** > 0 でない限り影響を与えません。このパラメーターのデフォルト設定は **true** です。

leveldb-store 要素

- ***relative-to*** パラメーターは、キャッシュ状態を格納するためのベースディレクトリーを指定します。
- ***path*** パラメーターはキャッシュ状態を格納するための ***relative-to*** パラメーター内の位置を指定します。
- ***shared*** パラメーターは、キャッシュストアを共有するかどうかを指定します。LevelDB キャッシュストアのこのパラメーターについてサポートされる唯一の値は **false** です。
- ***preload*** パラメーターは、キャッシュストアをプリロードするかどうかを指定します。有効な値は **true** と **false** です。

- ***block-size*** パラメーターはキャッシュストアのブロックサイズを定義します。
- ***singleton*** パラメーターは、SingletonStore の委譲するキャッシュストアを有効にします。これは、クラスター内の唯一のインスタンスが基礎となるストアと通信する場合にのみ使用されます。デフォルト値は **false** です。
- ***cache-size*** パラメーターはキャッシュストアのキャッシュサイズを定義します。
- ***clear-threshold*** パラメーターはキャッシュストアのキャッシュクリアのしきい値を定義します。

jpa-store 要素

- ***persistence-unit*** 属性は、JPA キャッシュストアの名前を指定します。
- ***entity-class*** 属性は、キャッシュエントリー値を格納するために使用する JPA エントリーの完全修飾クラス名を指定します。
- ***batch-size*** (オプション) 属性は、キャッシュストアストリーミングのバッチサイズを指定します。この属性のデフォルト値は **100** です。
- ***store-metadata*** (オプション) 属性は、キャッシュストアがメタデータ (失効やバージョンに関する情報など) を保持するかどうかを指定します。この属性のデフォルト値は **true** です。
- ***singleton*** パラメーターは、SingletonStore の委譲するキャッシュストアを有効にします。これは、クラスター内の唯一のインスタンスが基礎となるストアと通信する場合にのみ使用されます。デフォルト値は **false** です。

binary-keyed-jdbc-store 要素、string-keyed-jdbc-store、および mixed-keyed-jdbc-store 要素

- ***fetch-state*** パラメーターは、クラスターへ参加する時に永続状態が取り込まれるかを決定します。クラスター環境でレプリケーションとインバリデーションを使用している場合は、これを **true** に設定します。さらに、複数のキャッシュストアがチェーン化されている場合、1 つのキャッシュストアのみがこのプロパティーを有効に設定できます。共有キャッシュストアが使用されている場合、キャッシュは、このプロパティーが **true** に設定されているにも関わらず、永続状態の転送を許可しません。***fetch-state*** パラメーターはデフォルトでは **false** に設定されます。
- ***singleton*** パラメーターは、SingletonStore の委譲するキャッシュストアを有効にします。これは、クラスター内の唯一のインスタンスが基礎となるストアと通信する場合にのみ使用されます。デフォルト値は **false** です。
- **purge** パラメーターは、初回の起動時にキャッシュストアをパージするかどうかを指定します。
- ***key-to-string-mapper*** パラメーターは、キーをデータベーステーブルの文字列にマップするために使用されるクラス名を指定します。

connection-pool 要素 (JDBC ストア)

- ***connection-url*** パラメーターは、JDBC ドライバー固有の接続 URL を指定します。
- ***username*** パラメーターには、***connection-url*** で接続するために使用されるユーザー名が含まれます。
- ***password*** には、***connection-url*** で接続する際に使用するパスワードが含まれます。

- ***driver*** パラメーターは、データベースに接続するために使用されるドライバーのクラス名を指定します。

binary-keyed-table 要素および string-keyed-table 要素

- ***prefix*** 属性は、キャッシュバケットテーブルの名前を作成する際に、ターゲットキャッシュの名前に付与される文字列を定義します。
- ***drop-on-exit*** パラメーターは、シャットダウン時にデータベーステーブルがドロップされるかどうかを指定します。
- ***create-on-start*** パラメーターは、スタートアップ時にストアによってデータベーステーブルが作成されるかどうかを指定します。
- ***fetch-size*** パラメーターは、このテーブルからクエリーを実行する際に使用するサイズを指定します。このパラメーターを使用し、クエリーのサイズが大きくなった場合のヒープメモリーの消費を防ぎます。
- ***batch-size*** パラメーターは、このテーブルの変更時に使用されるバッチサイズを指定します。

id-column 要素、data-column 要素、および timestamp-column 要素

- ***name*** パラメーターは、使用される列の名前を指定します。
- ***type*** パラメーターは、使用される列のタイプを指定します。

custom-store 要素

- ***class*** パラメーターは、キャッシュストア実装のクラス名を指定します。
- ***preload*** パラメーターは、起動中にエントリーをキャッシュにロードするかどうかを指定します。このパラメーターの有効な値は **true** と **false** です。
- ***shared*** パラメーターは、キャッシュストアを共有するかどうかを指定します。これは、複数のキャッシュインスタンスがキャッシュストアを共有する場合に使用されます。このパラメーターに有効な値は **true** と **false** です。

property 要素

プロパティーは、プロパティータグ間のエントリーを保存された値にしてキャッシュストアの内部に定義できます。たとえば、以下の例では **1** の値が **minOccurs** に定義されます。

```
<property name="minOccurs">1</property>
```

- ***name*** 属性は、プロパティーの名前を指定します。

バグを報告する

19.3. キャッシュストア設定の詳細 (リモートクライアントサーバーモード)

以下の表には、JBoss Data Grid のリモートクライアントサーバーモードでのキャッシュストア要素の設定要素とパラメーターに関する詳細が含まれます。以下の一覧は各要素の特定のパラメーターを強調表示するためのもので、詳細の一覧についてはスキーマを参照していただくことができます。

local-cache 要素

- キャッシュの名前を指定するため、**local-cache** 属性の ***name*** パラメーターが使用されます。
- ***statistics*** パラメーターは、コンテナーレベルで統計を有効にするかどうかを指定します。***statistics*** 属性を **false** に設定することにより、キャッシュごとに統計を有効または無効にします。

file-store 要素

- **file-store** 要素の ***name*** パラメーターが、ファイルストアの名前を指定するために使用されます。
- ***passivation*** パラメーターは、キャッシュのエントリーがパッシベートされるか **(true)** またはキャッシュストアが内容のコピーをメモリーに保持するか **(false)** を決定します。
- ***purge*** パラメーターは、起動時にキャッシュストアをパージするかどうかを指定します。 このパラメーターの有効な値は **true** と **false** です。
- ***shared*** パラメーターは、複数のキャッシュインスタンスがキャッシュストアを共有する場合に使用されます。このパラメーターは、複数のキャッシュインスタンスが同じ変更内容を複数回書き込まないようにするために設定することができます。このパラメーターに有効な値は **true** と **false** です。ただし、***shared*** パラメーターは LevelDB キャッシュストアには推奨されません (このキャッシュストアを共有できないため)。
- ***relative-to*** プロパティーは、**file-store** がデータを保存するディレクトリーです。これは名前付きのパスを定義するために使用されます。
- ***path*** プロパティーは、データが保存されるファイルの名前です。これは、完全パスを決定するために ***relative-to*** プロパティーの値に追加される相対パス名です。
- ***max-entries*** パラメーターは、許可されるエントリーの最大数を指定します。無制限のエントリーの場合のデフォルト値は -1 です。
- ***fetch-state*** パラメーターは、true に設定されている場合に、クラスターへ参加する際に永続状態を取り込みます。複数のキャッシュストアがチェーン化されている場合、その内の 1 つのみでこのプロパティーを有効にできます。共有キャッシュストアが使用されている場合、永続状態の転送は、データを提供する同じ永続ストアが永続状態を受信するだけなので意味をなしません。そのため、共有キャッシュストアが使用されている場合、キャッシュストアでこのプロパティーが **true** に設定されている場合であっても、永続状態の転送は許可されません。クラスター化環境でのみこのプロパティーを true に設定することが推奨されます。このパラメーターのデフォルト値は false です。
- ***preload*** パラメーターは、true に設定されている場合に、キャッシュの起動時にキャッシュストアに保存されたデータをメモリーにロードします。ただし、このパラメーターを true に設定することにより、起動時間が増加するためパフォーマンスへの影響があります。このパラメーターのデフォルト値は false です。
- ***singleton*** パラメーターは、シングルトンストアのキャッシュストアを有効にします。SingletonStore は、クラスター内の唯一のインスタンスが基礎となるストアと通信する場合にのみ使用される委譲するキャッシュストアです。ただし、***singleton*** パラメーターは **file-store** には推奨されません。デフォルト値は **false** です。

store 要素

- ***class*** パラメーターは、キャッシュストア実装のクラス名を指定します。

property 要素

- ***name*** パラメーターは、プロパティーの名前を指定します。
- ***value*** パラメーターは、プロパティーに割り当てられた値を指定します。

remote-store 要素

- ***cache*** パラメーターは、リモートキャッシュの名前を定義します。定義されない状態のままの場合、デフォルトのキャッシュが代わりに使用されます。
- ***socket-timeout*** パラメーターは、***SO_TIMEOUT*** で定義される値 (ミリ秒単位) が指定されるタイムアウトでリモート Hot Rod サーバーに適用されるかどうかを設定します。タイムアウト値が **0** の場合は、無限のタイムアウトを示します。デフォルト値は 60,000 ms (ミリ秒) または 1 分です。
- ***tcp-no-delay*** は、**TCP_NODELAY** がソケット接続でリモートの Hot Rod サーバーに適用されるかどうかを設定します。
- ***hotrod-wrapping*** は、リモートストア上でラッパーが Hot Rod に必要となるかどうかを設定します。
- ***singleton*** パラメーターは、SingletonStore の委譲するキャッシュストアを有効にします。これは、クラスター内の唯一のインスタンスが基礎となるストアと通信する場合にのみ使用されます。デフォルト値は **false** です。

remote-server 要素

- ***outbound-socket-binding*** パラメーターは、リモートサーバーのアウトバウンドソケットバインディングを設定します。

binary-keyed-jdbc-store 要素、string-keyed-jdbc-store、および mixed-keyed-jdbc-store 要素

- ***datasource*** パラメーターは、データソースの JNDI 名を定義します。
- ***passivation*** パラメーターは、キャッシュのエントリーがパッシベートされるか (**true**) またはキャッシュストアが内容のコピーをメモリーに保持するか (**false**) を決定します。
- ***preload*** パラメーターは、起動中にエントリーをキャッシュにロードするかどうかを指定します。このパラメーターの有効な値は **true** と **false** です。
- ***purge*** パラメーターは、起動時にキャッシュストアをパージするかどうかを指定します。 このパラメーターの有効な値は **true** と **false** です。
- ***shared*** パラメーターは、複数のキャッシュストアインスタンスがキャッシュストアを共有する場合に使用されます。複数のキャッシュインスタンスが同じ変更内容を複数回書き込まないようにするため、このパラメーターを設定することができます。このパラメーターに有効な値は **true** と **false** です。
- ***singleton*** パラメーターは、シングルトンストアのキャッシュストアを有効にします。SingletonStore は、クラスター内の唯一のインスタンスが基礎となるストアと通信する場合にのみ使用される委譲キャッシュストアです。

binary-keyed-table 要素および string-keyed-table 要素

- ***prefix*** パラメーターはデータベーステーブル名のプレフィックスの文字列を指定します。

id-column 要素、data-column 要素、および timestamp-column 要素

- ***name*** パラメーターはデータベース列の名前を指定します。
- ***type*** パラメーターはデータベース列のタイプを指定します。

leveldb-store 要素

- ***relative-to*** パラメーターは、キャッシュ状態を格納するためのベースディレクトリーを指定します。この値はデフォルトで **jboss.server.data.dir** になります。
- ***path*** パラメーターは、キャッシュ状態が格納される ***relative-to*** パラメーターで指定されたディレクトリー内の場所を指定します。未定義の場合、パスのデフォルト値はキャッシュコンテナーの名前になります。
- ***passivation*** パラメーターは、LevelDB キャッシュストアに対してパッシベーションを有効にするかどうかを指定します。有効な値は **true** と **false** です。
- ***singleton*** パラメーターは、SingletonStore の委譲するキャッシュストアを有効にします。これは、クラスター内の唯一のインスタンスが基礎となるストアと通信する場合にのみ使用されます。デフォルト値は **false** です。
- ***purge*** パラメーターは、起動時にキャッシュストアをパージするかどうかを指定します。有効な値は **true** と **false** です。

バグを報告する

19.4. 単一ファイルキャッシュストア

Red Hat JBoss Data Grid には、ファイルシステムベースのキャッシュストアの 1 つである **SingleFileCacheStore** が含まれています。

SingleFileCacheStore は、単純なファイルシステムベースの実装であり、古くなったファイルシステムベースのキャッシュストアである **FileCacheStore** の代わりになるものです。

SingleFileCacheStore は、単一ファイルに、すべてのキーバリューペアと、それらの対応するメタデータ情報を保存します。データ検索のスピードを速めるために、すべてのキーとそれらの値およびメタデータの位置をメモリーに保存します。そのため、単一ファイルキャッシュストアを使用すると、キーのサイズと保存されるキーの数量に応じて、必要なメモリーが若干増加します。そのため、**SingleFileCacheStore** は、キーが大きすぎる場合のユースケースでは推奨されません。

メモリー消費量を減らすために、キャッシュストアのサイズを、ファイルに保存するエントリーの固定数に設定することができます。ただし、これは JBoss Data Grid がキャッシュとして使用される場合にのみ機能します。JBoss Data Grid がこのように使用されると、キャッシュにないデータは、正式なデータストアから再計算されるか、または再度取り込まれ、JBoss Data Grid キャッシュに保存される可能性があります。この制限がある理由は、エントリーの最大数にいったん達すると、キャッシュストアの古いデータが削除されるため、このシナリオで JBoss Data Grid が正式なデータストアとして使用される場合、データ損失が発生する可能性があります。

SingleFileCacheStore には制限があるため、実稼働環境では制限内での使用が可能です。適切なファイルロックがなく、データが破損する原因となるため、共有ファイルシステム (NFS や Windows の共有など) 上では使用しないでください。また、ファイルシステムは本質的にトランザクションでは

ないため、トランザクションコンテキストでキャッシュが使用されると、コミットフェーズ中にファイルが障害を書き込む原因となります。

バグを報告する

19.4.1. 単一ファイルストアの設定 (リモートクライアントサーバーモード)

以下は、Red Hat JBoss Data Grid のリモートクライアントサーバーモードにおける単一ファイルストアの設定の例です。

```
<local-cache name="default" statistics="true">
    <file-store name="myFileStore"
                passivation="true"
                purge="true"
                relative-to="{PATH}"
                path="{DIRECTORY}"
                max-entries="10000"
                fetch-state="true"
                preload="false" />
</local-cache>
```

この設定例で使用された要素とパラメーターの詳細については、「キャッシュストア設定の詳細 (リモートクライアントサーバーモード)」を参照してください。

バグを報告する

19.4.2. 単一ファイルストアの設定 (ライブラリーモード)

Red Hat JBoss Grid のライブラリーモードで、単一ファイルキャッシュストアを以下のように設定します。

```
<local-cache name="writeThroughToFile">
      <persistence passivation="false">
         <file-store fetch-state="true"
                     purge="false"
                     shared="false"
                     preload="false"
                     location="/tmp/Another-FileCacheStore-Location"
                     max-entries="100">
            <write-behind enabled="true"
                threadPoolSize="500"
                flush-lock-timeout="1"
            modification-queue-size="1024"
            shutdown-timeout="25000"/>
        </singleFile>
      </persistence>
 </local-cache>
```

この設定例で使用された要素とパラメーターの詳細については、「キャッシュストア設定の詳細 (ライブラリモード)」を参照してください。

バグを報告する

19.4.3. JBoss Data Grid キャッシュストアのアップグレード

Red Hat JBoss Data Grid 7 は、以前のバージョンの JBoss Data Grid とは異なる形式でデータを保存します。そのため、新しいバージョンの JBoss Data Grid は、古いバージョンで保存されたデータを読み込むことはできません。ローリングアップグレードを使用すると、旧バージョンの JBoss Data Grid で使用された形式で永続化されたデータを新しい形式にアップグレードすることができます。さらに、新たなバージョンの JBoss Data Grid は、永続性設定情報を別の場所に保存します。

ローリングアップグレードは、JBoss Data Grid インストールをサービスをシャットダウンせずにアップグレードするプロセスです。JBoss Data Grid サーバーの場合は、サーバー側のコンポーネントを指します。このアップグレードは、ハードウェア変更または JBoss Data Grid のアップグレードなどのソフトウェアの変更のいずれかによって発生する場合があります。

ローリングアップグレードは、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ます。

バグを報告する

19.5. LEVELDB キャッシュストア

LevelDB は、文字列キーから文字列値への順序付けられたマッピングを提供するキーバリューのストレージエンジンです。

LevelDB キャッシュストアは 2 つのファイルシステムディレクトリーを使用します。それぞれのディレクトリーは、LevelDB データベースについて設定されます。1 つのディレクトリーは、期限が切れていないデータを保存し、2 つ目のディレクトリーは、永続的にパージされるよう保留状態にされているキーを保存します。

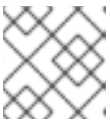
バグを報告する

19.5.1. LevelDB キャッシュストアの設定 （リモートクライアントサーバーモード)

手順19.1 LevelDB キャッシュストアの設定

- データベースを設定するには、**standalone.xml** のキャッシュ定義に以下の要素を追加します。

```
<leveldb-store path="/path/to/leveldb/data"
               passivation="false"
               purge="false" >
    <leveldb-expiration path="/path/to/leveldb/expires/data" />
    <implementation type="JNI" />
</leveldb-store>
```



注記

ディレクトリーがない場合は自動的に作成されます。

この設定例で使用された要素とパラメーターの詳細については、「キャッシュストア設定の詳細 （リモートクライアントサーバーモード)」を参照してください。

バグを報告する

19.5.2. LevelDB キャッシュストアの XML 設定例 (ライブラリーモード)

以下は、LevelDB キャッシュストアの XML 設定例です。

```
<local-cache name="vehicleCache">
      <persistence passivation="false">
          <leveldb-store xmlns="urn:infinispan:config:store:leveldb:8.0
                         relative-to="/path/to/leveldb/data"
                         shared="false"
                         preload="true"/>
      </persistence>
   </local-cache>
```

この設定例で使用された要素とパラメーターの詳細については、「キャッシュストア設定の詳細 （ライブラリモード)」を参照してください。

バグを報告する

19.5.3. JBoss Operations Network を使用した LevelDB キャッシュストアの設定

以下の手順に従い、JBoss Operations Network を使用して新しい LevelDB キャッシュストアをセットアップします。

手順19.2

1. Red Hat JBoss Operations Network 3.2 以上がインストールされ、起動されていることを確認します。
2. JBoss Operations Network 3.2.0 用 Red Hat JBoss Data Grid プラグインパックをインストールします。
3. JBoss Data Grid がインストールされ、起動されていることを確認します。
4. JBoss Data Grid サーバーをインベントリーにインポートします。
5. JBoss Data Grid 接続設定を実行します。
6. 以下のように新しい LevelDB キャッシュストアを作成します。

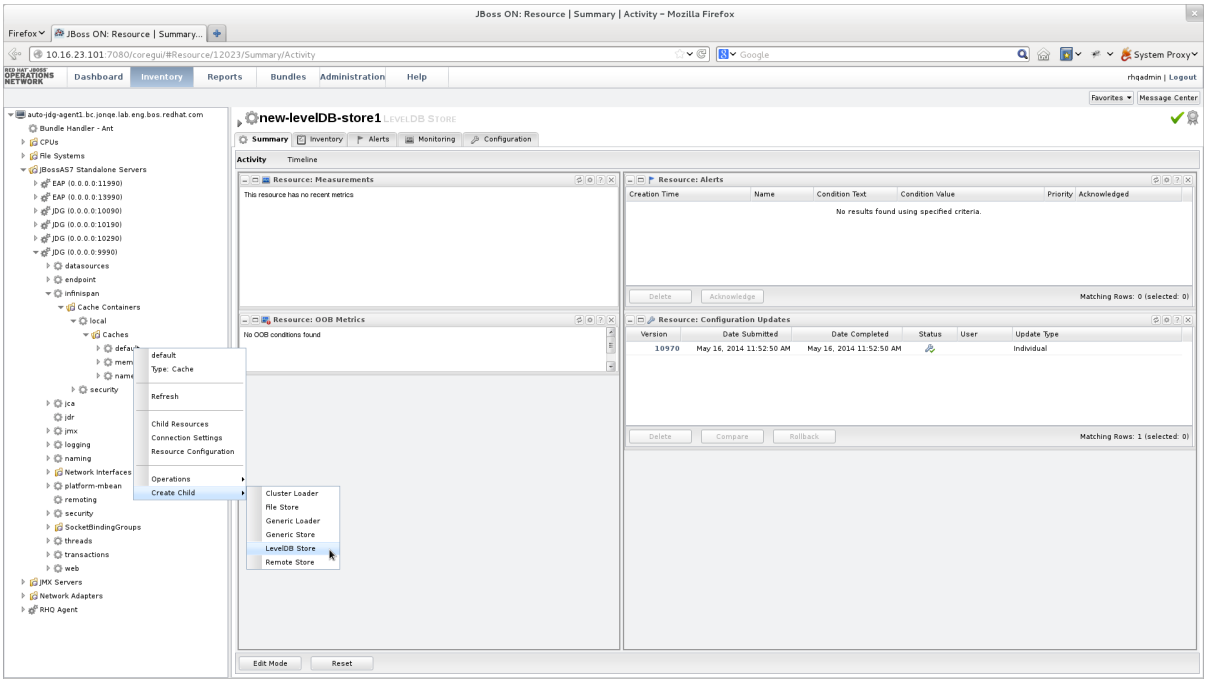


図19.1 新しい LevelDB キャッシュストアの作成

a. **default** キャッシュを右クリックします。

b. メニューで、**Create Child** オプションにカーソルを置きます。

c. サブメニューで、**LevelDB Store** をクリックします。

7. 以下のように新しい LevelDB キャッシュストアの名前を指定します。

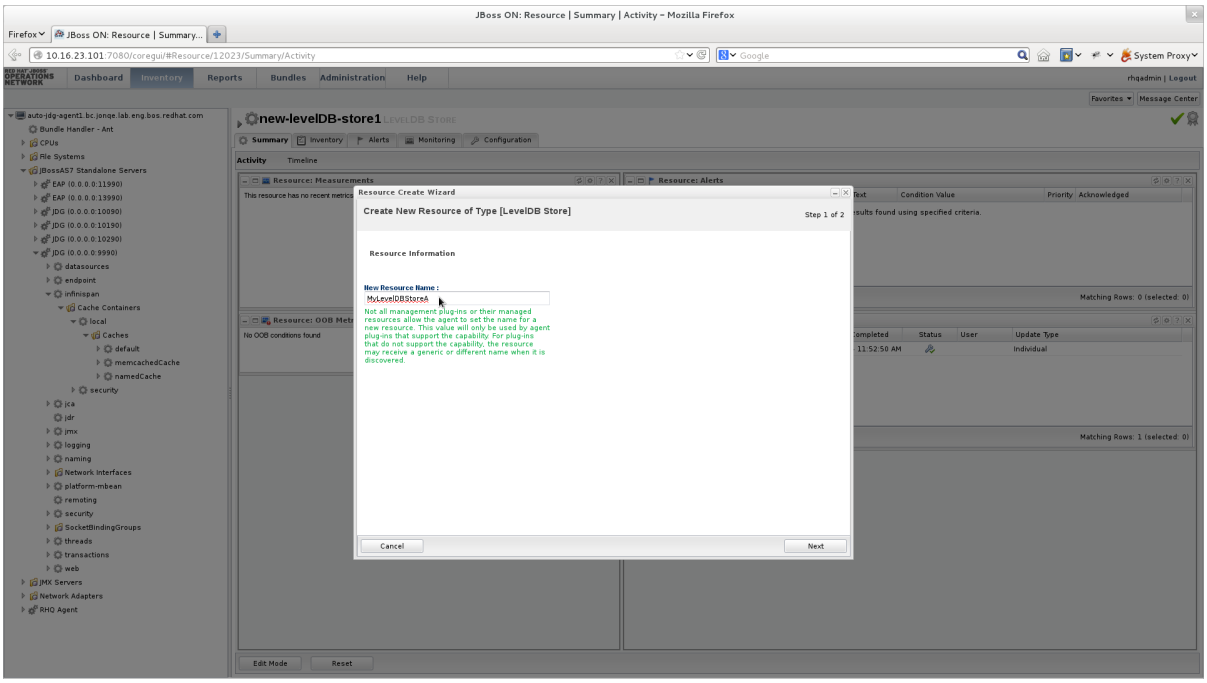


図19.2 新しい LevelDB キャッシュストアの名前指定

a. 表示された **Resource Create Wizard** で、新しい LevelDB キャッシュストアの名前を追加します。

b. **Next** をクリックして作業を継続します。

8. 以下のように LevelDB キャッシュストアを設定します。

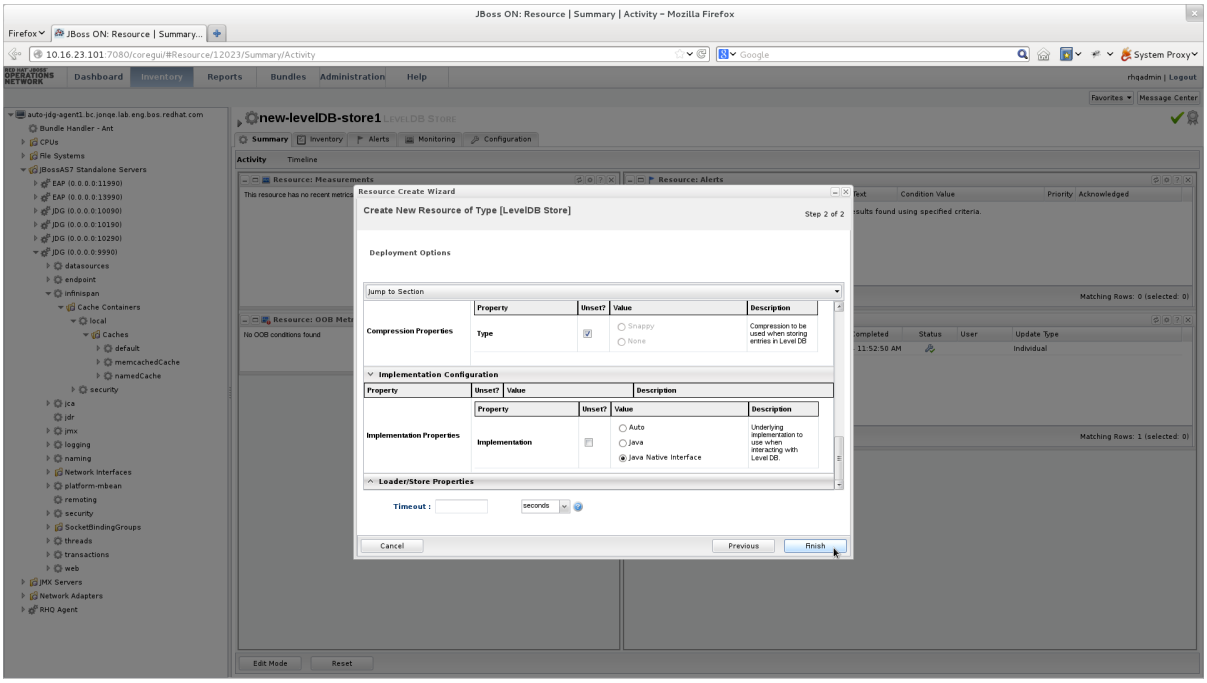


図19.3 LevelDB キャッシュストアの設定

a. 設定ウィンドウのオプションを使用して新しい LevelDB キャッシュストアを設定します。

b. **Finish** をクリックして設定を完了します。

9. 以下のように再起動操作をスケジュールします。

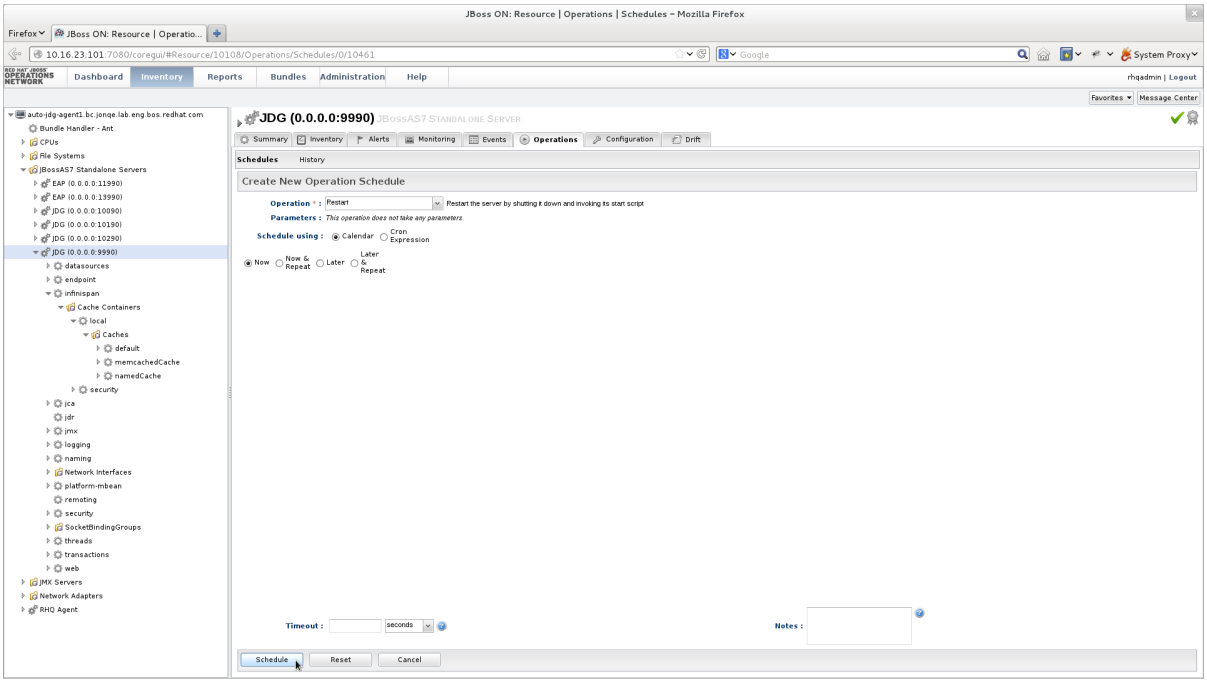


図19.4 再起動操作のスケジュール

a. 画面の左パネルで、**JBossAS7 Standalone Servers** エントリーを展開します (まだ展開されていない場合)。

b. 展開されたメニュー項目から **JDG (0.0.0.0:9990)** をクリックします。

c. 画面の右パネルに、選択されたサーバーの詳細が表示されます。**Operations** タブをクリックします。

d. **Operation** ドロップダウンボックスで、**Restart** 操作を選択します。

e. **Now** エントリーのラジオボタンを選択します。

f. **Schedule** をクリックしてサーバーをすぐに再起動します。

10. 以下のように新しい LevelDB キャッシュストアを検出します。

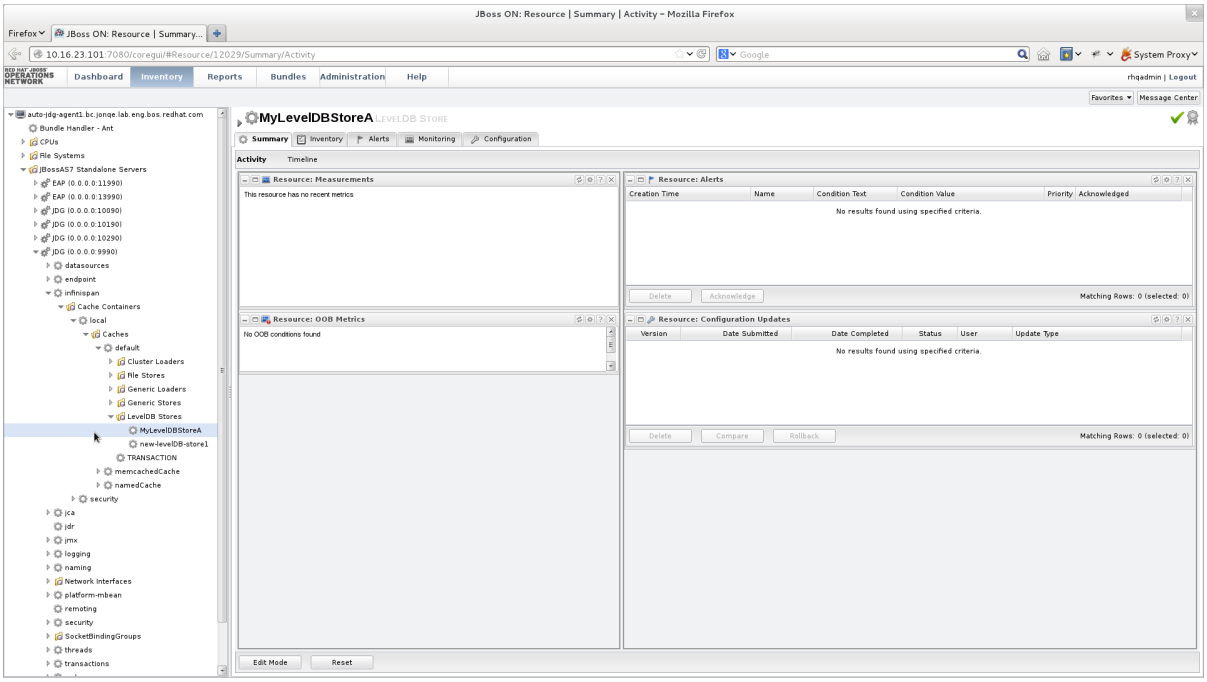


図19.5 新しい LevelDB キャッシュストアの検出

a. 画面の左パネルで、指定された順序で次の項目を選択して展開します: **JBossAS7 Standalong Servers → JDG (0.0.0.0:9990) → infinispan → Cache Containers → local → Caches → default → LevelDB Stores**

b. 新しい LevelDB キャッシュストアの名前をクリックして右パネルの設定情報を表示します。

バグを報告する

19.6. JDBC ベースのキャッシュストア

Red Hat JBoss Data Grid は、一般的なデータストレージ形式と共に使用される複数のキャッシュストアを提供します。JDBC ベースのキャッシュストアは、JDBC ドライバーを公開するキャッシュストアと共に使用されます。JBoss Data Grid は、永続化されるキーに応じて以下の JDBC ベースのキャッシュストアを提供します。

- **`JdbcBinaryStore`**。
- **`JdbcStringBasedStore`**。
- **`JdbcMixedStore`**。

バグを報告する

19.6.1. JdbcBinaryStores

JdbcBinaryStore はすべてのキータイプをサポートします。同じテーブル行/Blob の同じハッシュ値 (キー上の **hashCode** メソッド) を持つすべてのキーを格納します。組み込まれるキーに共通するハッシュ値が、テーブルの行/Blob の主キーとして設定されます。このハッシュ値により、**JdbcBinaryStore** は大変優れた柔軟性を提供しますが、これにより平行性とスループットのレベルが下がります。

たとえば、3 つのキー (**k1**、**k2**、**k3**) のハッシュコードが同じである場合、同じテーブル行に格納されます。3 つの異なるスレッドが **k1**、**k2**、**k3** を同時に更新しようとすると、すべてのキーが同じ行を共有するため同時には更新できないことから、順次更新する必要があります。

バグを報告する

19.6.1.1. JdbcBinaryStore の設定 (リモートクライアントサーバーモード)

以下は、Red Hat JBoss Data Grid のリモートクライアントサーバーモードを使用し、パッシベーションを有効にした **JdbcBinaryStore** の設定です。

```
<local-cache name="customCache">

 <!-- Additional configuration elements here -->
 <binary-keyed-jdbc-store datasource="java:jboss/datasources/JdbcDS"

     passivation="${true/false}"
     preload="${true/false}"
     purge="${true/false}">
                <binary-keyed-table prefix="JDG">
                 <id-column name="id"
       type="${id.column.type}"/>
                  <data-column name="datum"
         type="${data.column.type}"/>
                <timestamp-column name="version"
       type="${timestamp.column.type}"/>
               </binary-keyed-table>
        </binary-keyed-jdbc-store>
</local-cache>
```

この設定例で使用された要素とパラメーターの詳細については、「キャッシュストア設定の詳細 (リモートクライアントサーバーモード)」を参照してください。

バグを報告する

19.6.1.2. JdbcBinaryStore の設定 (ライブラリーモード)

以下は、**JdbcBinaryStore** の設定例です。

```
<infinispan
        xmlns:xsi="http://www.w3.org/2001/XMLSchema-instance"
        xsi:schemaLocation="urn:infinispan:config:8.3
http://www.infinispan.org/schemas/infinispan-config-8.3.xsd
            urn:infinispan:config:store:jdbc:8.0
http://www.infinispan.org/schemas/infinispan-cachestore-jdbc-config-
8.0.xsd"
        xmlns="urn:infinispan:config:8.3">
```

```
        <!-- Additional configuration elements here -->
 <persistence>
 <binary-keyed-jdbc-store xmlns="urn:infinispan:config:store:jdbc:8.0
                              fetch-state="false"
         purge="false">
  <connection-pool connection-
url="jdbc:h2:mem:infinispan_binary_based;DB_CLOSE_DELAY=-1"
    username="sa"
    driver="org.h2.Driver"/>
  <binary-keyed-table dropOnExit="true"
      createOnStart="true"
      prefix="ISPN_BUCKET_TABLE">
   <id-column name="ID_COLUMN"
      type="VARCHAR(255)" />
   <data-column name="DATA_COLUMN"
        type="BINARY" />
   <timestamp-column name="TIMESTAMP_COLUMN"
      type="BIGINT" />
  </binary-keyed-table>
 </binary-keyed-jdbc-store>
</persistence>
```

この設定例で使用された要素とパラメーターの詳細については、「キャッシュストア設定の詳細 （ライブラリモード)」を参照してください。

バグを報告する

19.6.2. JdbcStringBasedStores

JdbcStringBasedStore は複数のエントリーを各行にグループ化せずに、各エントリーをテーブルの独自の行に格納するため、同時負荷の下でスループットが増加します。また、各キーを **String** オブジェクトへマッピングする (プラグ可能な) バイジェクション (bijection) も使用します。**Key2StringMapper** インターフェースはバイジェクションを定義します。

Red Hat JBoss Data Grid には、プリミティブタイプを処理する **DefaultTwoWayKey2StringMapper** と呼ばれるデフォルトの実装が含まれています。

バグを報告する

19.6.2.1. JdbcStringBasedStore の設定 (リモートクライアントサーバーモード)

以下は、Red Hat JBoss Data Grid のリモートクライアントサーバーモードの **JdbcStringBasedStore** の設定例です。

```
<local-cache name="customCache">
 <!-- Additional configuration elements here -->
 <string-keyed-jdbc-store datasource="java:jboss/datasources/JdbcDS"
     passivation="true"
     preload="false"
     purge="false"
     shared="false"
     singleton="true">
               <string-keyed-table prefix="JDG">
                <id-column name="id"
       type="${id.column.type}"/>
```

```
    <data-column name="datum"
          type="${data.column.type}"/>
    <timestamp-column name="version"
        type="${timestamp.column.type}"/>
          </string-keyed-table>
 </string-keyed-jdbc-store>
</local-cache>
```

この設定例で使用された要素とパラメーターの詳細については、「キャッシュストア設定の詳細 （リモートクライアントサーバーモード)」を参照してください。

バグを報告する

19.6.2.2. JdbcStringBasedStore 設定 (ライブラリーモード)

JdbcStringBasedStore の設定例は次のとおりです。

```
<infinispan
        xmlns:xsi="http://www.w3.org/2001/XMLSchema-instance"
        xsi:schemaLocation="urn:infinispan:config:8.3
http://www.infinispan.org/schemas/infinispan-config-8.3.xsd
            urn:infinispan:config:store:jdbc:8.0
http://www.infinispan.org/schemas/infinispan-cachestore-jdbc-config-
8.0.xsd"
        xmlns="urn:infinispan:config:8.3">
        <!-- Additional configuration elements here -->
 <persistence>
 <string-keyed-jdbc-store xmlns="urn:infinispan:config:store:jdbc:8.0"
                      fetch-state="false"
         purge="false"

key2StringMapper="org.infinispan.loaders.keymappers.DefaultTwoWayKey2Strin
gMapper">
  <dataSource jndiUrl="java:jboss/datasources/JdbcDS"/>
  <string-keyed-table dropOnExit="true"
      createOnStart="true"
      prefix="ISPN_STRING_TABLE">
   <id-column name="ID_COLUMN"
      type="VARCHAR(255)" />
   <data-column name="DATA_COLUMN"
        type="BINARY" />
   <timestamp-column name="TIMESTAMP_COLUMN"
      type="BIGINT" />
  </string-keyed-table>
 </string-keyed-jdbc-store>
</persistence>
```

この設定例で使用された要素とパラメーターの詳細については、「キャッシュストア設定の詳細 （ライブラリモード)」を参照してください。

バグを報告する

19.6.2.3. JdbcStringBasedStore の複数ノード設定 (リモートクライアントサーバーモード)

以下は、Red Hat JBoss Data Grid のリモートクライアントサーバーモードにおける **JdbcStringBasedStore** の設定になります。この設定は、複数のノードを使用しなければならない場合に使用されます。

```
<subsystem xmlns="urn:infinispan:server:core:8.3" default-cache-
container="default">
 <cache-container <!-- Additional configuration information here --> >
  <!-- Additional configuration elements here -->
      <replicated-cache>
   <!-- Additional configuration elements here -->
       <string-keyed-jdbc-store datasource="java:jboss/datasources/JdbcDS"
                      fetch-state="true"
                      passivation="false"
                      preload="false"
                      purge="false"
                      shared="false"
                      singleton="true">
          <string-keyed-table prefix="JDG">
              <id-column name="id"
                         type="${id.column.type}"/>
              <data-column name="datum"
                           type="${data.column.type}"/>
              <timestamp-column name="version"
                                type="${timestamp.column.type}"/>
    </string-keyed-table>
   </string-keyed-jdbc-store>
  </replicated-cache>
 </cache-container>
</subsystem>
```

この設定例で使用された要素とパラメーターの詳細については、「キャッシュストア設定の詳細 （リモートクライアントサーバーモード)」を参照してください。

バグを報告する

19.6.3. JdbcMixedStores

JdbcMixedStore は、キーのタイプを基にキーを **JdbcBinaryStore** または **JdbcStringBasedStore** に委譲するハイブリッド実装です。

バグを報告する

19.6.3.1. JdbcMixedStore の設定 (リモートクライアントサーバーモード)

以下は、Red Hat JBoss Data Grid のリモートクライアントサーバーモードの **JdbcMixedStore** の設定です。

```
<local-cache name="customCache">
 <mixed-keyed-jdbc-store datasource="java:jboss/datasources/JdbcDS"
    passivation="true"
    preload="false"
    purge="false">
  <binary-keyed-table prefix="MIX_BKT2">
   <id-column name="id"
       type="${id.column.type}"/>
```

```
   <data-column name="datum"
         type="${data.column.type}"/>
   <timestamp-column name="version"
          type="${timestamp.column.type}"/>
  </binary-keyed-table>
  <string-keyed-table prefix="MIX_STR2">
   <id-column name="id"
       type="${id.column.type}"/>
   <data-column name="datum"
         type="${data.column.type}"/>
   <timestamp-column name="version"
          type="${timestamp.column.type}"/>
  </string-keyed-table>
 </mixed-keyed-jdbc-store>
</local-cache>
```

この設定例で使用された要素とパラメーターの詳細については、「キャッシュストア設定の詳細 （リモートクライアントサーバーモード)」を参照してください。

バグを報告する

19.6.3.2. JdbcMixedStore の設定 (ライブラリーモード)

以下は、**JdbcMixedStore** の設定例です。

```
<infinispan
        xmlns:xsi="http://www.w3.org/2001/XMLSchema-instance"
        xsi:schemaLocation="urn:infinispan:config:8.3
http://www.infinispan.org/schemas/infinispan-config-8.3.xsd
            urn:infinispan:config:store:jdbc:8.0
http://www.infinispan.org/schemas/infinispan-cachestore-jdbc-config-
8.0.xsd"
        xmlns="urn:infinispan:config:8.3">
        <!-- Additional configuration elements here -->
 <persistence>
 <mixed-keyed-jdbc-store xmlns="urn:infinispan:config:store:jdbc:8.0"
                      fetch-state="false"
         purge="false"
         key-to-string-
mapper="org.infinispan.persistence.keymappers.DefaultTwoWayKey2StringMappe
r">
  <connection-pool connection-
url="jdbc:h2:mem:infinispan_binary_based;DB_CLOSE_DELAY=-1"
    username="sa"
    driver="org.h2.Driver"/>
  <binary-keyed-table dropOnExit="true"
      createOnStart="true"
      prefix="ISPN_BUCKET_TABLE_BINARY">
   <id-column name="ID_COLUMN"
      type="VARCHAR(255)" />
   <data-column name="DATA_COLUMN"
        type="BINARY" />
   <timestamp-column name="TIMESTAMP_COLUMN"
      type="BIGINT" />
  </binary-keyed-table>
  <string-keyed-table dropOnExit="true"
```

```
      createOnStart="true"
      prefix="ISPN_BUCKET_TABLE_STRING">
    <id-column name="ID_COLUMN"
      type="VARCHAR(255)" />
    <data-column name="DATA_COLUMN"
        type="BINARY" />
    <timestamp-column name="TIMESTAMP_COLUMN"
      type="BIGINT" />
   </string-keyed-table>
  </mixed-keyed-jdbc-store>
</persistence>
```

この設定例で使用された要素とパラメーターの詳細については、「キャッシュストア設定の詳細 （ライブラリモード)」を参照してください。

バグを報告する

19.6.4. キャッシュストアのトラブルシューティング

19.6.4.1. JdbcStringBasedStore の IOExceptions

JdbcStringBasedStore を使用している時に IOException **Unsupported protocol version 48** エラーが発生した場合は、データ列タイプが正しいタイプである **BLOB** や **VARBINARY** ではなく、**VARCHAR** や **CLOB** などに設定されていることを示しています。**JdbcStringBasedStore** は文字列であるキーのみを必要とし、値はバイナリー列に保存されるため、すべてのデータタイプを値に使用できます。

バグを報告する

19.7. リモートキャッシュストア

RemoteCacheStore は、リモート Red Hat JBoss Data Grid クラスターにデータを保存するキャッシュローダーの実装です。**RemoteCacheStore** は Hot Rod クライアントサーバーアーキテクチャーを使用してリモートクラスターと通信します。

Hot Rod はリモートキャッシュストアに対して ロードバランシングやフォールトトラレンスを提供します。また、 **RemoteCacheStore** とクラスター間の接続を細かく調整する機能も提供します。

バグを報告する

19.7.1. リモートキャッシュストアの設定 （リモートクライアントサーバーモード)

以下は、Red Hat JBoss Data Grid のリモートクライアントサーバーモードにおけるリモートキャッシュストアの設定例を示しています。

```
<remote-store cache="default"
              socket-timeout="60000"
              tcp-no-delay="true"
              hotrod-wrapping="true">
 <remote-server outbound-socket-binding="remote-store-hotrod-server" />
</remote-store>
```

この設定例で使用された要素とパラメーターの詳細については、「キャッシュストア設定の詳細 （リモートクライアントサーバーモード)」を参照してください。

バグを報告する

19.7.2. リモートキャッシュストアの設定 （ライブラリーモード)

以下は、Red Hat JBoss Data Grid のライブラリーモードにおけるリモートキャッシュストアの設定例を示しています。

```
<persistence passivation="false">
 <remote-store xmlns="urn:infinispan:config:remote:8.3"
              cache="default"
       fetch-state="false"
       shared="true"
       preload="false"
       purge="false"
       tcp-no-delay="true"
       ping-on-start="true"
       key-size-estimate="62"
       value-size-estimate="512"
       force-return-values="false">
  <remote-server host="127.0.0.1"
   port="1971" />
  <connectionPool max-active="99"
    max-idle="97"
    max-total="98" />
 </remote-store>
</persistence>
```

この設定例で使用された要素とパラメーターの詳細については、「キャッシュストア設定の詳細 （ライブラリモード)」を参照してください。

バグを報告する

19.7.3. リモートキャッシュストアのアウトバウンドソケットの定義

リモートキャッシュストアによって使用される Hot Rod サーバーは、**standalone.xml** ファイルの **outbound-socket-binding** 要素を使用して定義されます。

standalone.xml ファイルにおけるこの設定の例は次のとおりです。

例19.1 アウトバウンドソケットの定義

```
<server>
    <!-- Additional configuration elements here -->
    <socket-binding-group name="standard-sockets"
         default-interface="public"
         port-offset="${jboss.socket.binding.port-offset:0}">
        <!-- Additional configuration elements here -->
        <outbound-socket-binding name="remote-store-hotrod-server">
            <remote-destination host="remote-host"
                                port="11222"/>
        </outbound-socket-binding>
    </socket-binding-group>
</server>
```

バグを報告する

19.8. JPA キャッシュストア

JPA (Java Persistence API) キャッシュストアは、正式なスキーマを使用してデータベースにキャッシュエントリーを格納します。これにより、他のアプリケーションは永続化データを読み取り、他のアプリケーションにより提供されたデータを Red Hat JBoss Data Grid にロードできます。他のアプリケーションが JBoss Data Grid でこのデータベースを同時に使用しないようにする必要があります。

重要

Red Hat JBoss Data Grid では、JPA キャッシュストアはライブラリーモードでのみサポートされます。

バグを報告する

19.8.1. JPA キャッシュストアの XML 設定例 (ライブラリーモード)

Red Hat JBoss Data Grid で XML を使用して JPA キャッシュストアを設定するには、以下の設定を **infinispan.xml** ファイルに追加します。

```
<local-cache name="users">
  <!-- Insert additional configuration elements here -->
 <persistence passivation="false">
           <jpa-store xmlns="urn:infinispan:config:store:jpa:8.0"
                      shared="true"
                      preload="true"
                      persistence-unit="MyPersistenceUnit"
                      entity-
class="org.infinispan.loaders.jpa.entity.User" />
 </persistence>
</local-cache>
```

この設定例で使用された要素とパラメーターの詳細については、「キャッシュストア設定の詳細 (ライブラリモード)」を参照してください。

バグを報告する

19.8.2. データベースへのメタデータの格納

storeMetadata が **true** (デフォルト値) に設定された場合、有効期限、作成および変更タイムスタンプ、バージョンなどのエントリーに関するメタ情報はデータベースに格納されます。エンティティーテーブルのレイアウトは固定され、メタデータを収めることができないため、JBoss Data Grid はメタデータを **__ispn_metadata__** という名前の追加テーブルにメタデータを格納します。

このテーブルの構造は、使用中のデータベースに依存します。テスト環境と同じデータベースを使用して、このテーブルの自動作成を有効にし、構造を本稼働データベースに転送します。

手順19.3 persistence.xml でのメタデータエンティティーの設定

1. Hibernate を JPA 実装として使用すると、以下のように **persistence.xml** のプロパティー ***hibernate.hbm2ddl.auto*** を使用してこれらのテーブルの自動作成を許可できます。

```
<property name="hibernate.hbm2ddl.auto" value="update"/>
```

2. 以下の内容を **persistence.xml** に追加して、JPA プロバイダーに対してメタデータエンティティークラスを宣言します。

```
<class>org.infinispan.persistence.jpa.impl.MetadataEntity</class>
```

前述のように、メタデータは常に新しいテーブルに格納されます。メタデータ情報の収集と格納が必要ない場合は、JPA ストア設定で ***storeMetadata*** 属性を **false** に設定します。

バグを報告する

19.8.3. 各種のコンテナーでの JPA キャッシュストアのデプロイ

Red Hat JBoss Data Grid の JPA キャッシュストア実装は、Red Hat JBoss Enterprise Application Platform を除くすべてのサポート対象コンテナーに対して正常にデプロイされます。JBoss Data Grid の JBoss EAP モジュールには、JPA キャッシュストアと関連ライブラリー (Hibernate など) が含まれます。結果として、関連ライブラリーはアプリケーション内部にパッケージ化されず、アプリケーションはインストールされた JBoss EAP モジュールのライブラリーを参照します。

これらのモジュールは、JBoss EAP 以外のコンテナーには必要ありません。結果として、すべての関連ライブラリーは、以下の Maven 依存関係の場合のようにアプリケーションの WAR/EAR ファイル内でパッケージ化されます。

```
<dependency>
    <groupId>org.infinispan</groupId>
    <artifactId>infinispan-cachestore-jpa</artifactId>
    <version>8.3.0.Final-redhat-1</version>
</dependency>
```

手順19.4 JBoss EAP 6.3.x およびそれ以前のバージョンでの JPA キャッシュストアのデプロイ

- 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に追加するには、以下のいずれかの方法で JBoss EAP デプロイヤーに依存関係のリストを提供します。

  a. 依存関係設定を **MANIFEST.MF** ファイルに追加します。

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

  b. 依存関係設定を **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.persistence.jpa"
slot="jdg-7.0" services="export"/>
            <module name="org.infinispan" slot="jdg-7.0"
services="export"/>
```

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

手順19.5 JBoss EAP 6.4 以降での JPA キャッシュストアのデプロイ

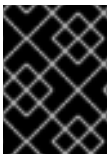
1. **persistence.xml** で以下のプロパティーを追加します。

```
<persistence-unit>
  [...]
  <properties>
    <property name="jboss.as.jpa.providerModule" value="application"
/>
  </properties>
</persistence-unit>
```

2. **jboss-deployment-structure.xml** で以下の依存関係を追加します。

```
<jboss-deployment-structure>
    <deployment>
        <dependencies>
            <module name="org.infinispan" slot="jdg-7.0"/>
            <module name="org.jgroups" slot="jdg-7.0"/>
            <module name="org.infinispan.persistence.jpa"
slot="jdg-7.0" services="export"/>
            <module name="org.hibernate"/>
        </dependencies>
    </deployment>
</jboss-deployment-structure>
```

3. 追加の JDG モジュールなどの依存関係を **jboss-deployment-structure.xml** の **dependencies** セクションに追加します。



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。

バグを報告する

19.9. CASSANDRA キャッシュストア

Red Hat JBoss Data Grid は、Apache Cassandra がキャッシュストアとして機能することを可能にし、分散データベースアーキテクチャーから実質的に無制限の、水平方向に拡張可能な、キャッシュエントリーの永続ストアを提供できるようにします。

Cassandra キャッシュストアを使用するには、適切なキースペースを Cassandra データベースに作成する必要があります。これは自動的に実行するか、またはキャッシュストア設定で **auto-create-keyspace** パラメーターを有効にして実行できます。キースペースの作成例は以下のとおりです。

```
CREATE KEYSPACE IF NOT EXISTS Infinispan WITH replication =
{'class':'SimpleStrategy', 'replication_factor':1};
CREATE TABLE Infinispan.InfinispanEntries (key blob PRIMARY KEY, value
```

```
blob, metadata blob);
```

バグを報告する

19.9.1. Cassandra キャッシュストアの有効化

Cassandra キャッシュストアは、ダウンロードされたディストリビューションをベースに組み込まれます。以下は、これが置かれる場所、および必要な場合にこれを有効にする手順を示しています。

- **ライブラリーモード:** **`infinispan-cachestore-cassandra-8.3.0.final-redhat-1-deployable.jar`** は **`jboss-datagrid-${jdg-version}-library/`** ディレクトリーに組み込まれており、Cassandra キャッシュストアを使用しているすべてのプロジェクトに追加できます。
- **リモートクライアントサーバーモード:** Cassandra キャッシュストアはサーバーの **`modules/`** ディレクトリーに事前にパッケージ化されており、追加の設定なしにデフォルトで使用できます。
- **JBoss EAP の JBoss Data Grid モジュール:** Cassandra キャッシュストアは配信されるモジュールに組み込まれており、モジュール名として **`org.infinispan.persistence.cassandra`** を使用して追加できます。

バグを報告する

19.9.2. Cassandra キャッシュストアの XML 設定例 (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は、Cassandra キャッシュストアは、クラス **`org.infinispan.persistence.cassandra.CassandraStore`** を使用し、プロパティーをストア内に個別に定義することによって定義されます。

以下の設定スニペットは、xml ファイル内に Cassandra キャッシュストアを定義する方法についての例を示しています。

```
<local-cache name="cassandracache" start="EAGER">
    <locking acquire-timeout="30000" concurrency-level="1000"
striping="false"/>
    <transaction mode="NONE"/>
    <store name="cassstore1"
           class="org.infinispan.persistence.cassandra.CassandraStore"
           shared="true"
           passivation="false">
        <property name="autoCreateKeyspace">true</property>
        <property name="keyspace">store1</property>
        <property name="entryTable">entries1</property>
        <property name="consistencyLevel">LOCAL_ONE</property>
        <property name="serialConsistencyLevel">SERIAL</property>
        <property name="servers">127.0.0.1[9042],127.0.0.1[9041]
</property>
        <property
name="connectionPool.heartbeatIntervalSeconds">30</property>
        <property name="connectionPool.idleTimeoutSeconds">120</property>
        <property name="connectionPool.poolTimeoutMillis">5</property>
    </store>
</local-cache>
```

バグを報告する

19.9.3. Cassandra キャッシュストアの XML 設定例 (ライブラリーモード)

ライブラリーモードでは、Cassandra キャッシュストアは 2 つの異なるメソッドを使って設定することができます。

- **オプション 1:** 「Cassandra キャッシュストアの XML 設定例 (リモートクライアントサーバーモード)」で説明されている リモートクライアントサーバーモードで使用されるのと同じメソッドの使用。
- **オプション 2: `cassandra-store`** スキーマの使用。以下のスニペットは、Cassandra キャッシュストアを定義する設定例を示しています。

```
<cache-container default-cache="cassandracache">
  <local-cache name="cassandracache">
    <persistence passivation="false">
      <cassandra-store
xmlns="urn:infinispan:config:store:cassandra:8.2"
            auto-create-keyspace="true"
            keyspace="Infinispan"
            entry-table="InfinispanEntries" shared="true">
        <cassandra-server host="127.0.0.1" port="9042" />
        <connection-pool heartbeat-interval-seconds="30"
            idle-timeout-seconds="120"
            pool-timeout-millis="5" />
      </cassandra-store>
    </persistence>
  </local-cache>
</cache-container>
```

バグを報告する

19.9.4. Cassandra 設定パラメーター

ライブラリーモードでバッキング Cassandra インスタンスを定義する場合に、1 つ以上の **`cassandra-server`** 要素を設定に指定できます。要素のそれぞれには以下のプロパティーがあります。

表19.1 Cassandra サーバー設定パラメーター

パラメーター名	説明	デフォルト値
`host`	Cassandra サーバーのホスト名または IP アドレス。	127.0.0.1
`port`	サーバーがリッスンしているポート。	9042

以下のプロパティーを Cassandra キャッシュストアに設定できます。

表19.2 Cassandra 設定パラメーター

パラメーター名	説明	デフォルト値
auto-create-keyspace	キースペースおよびエントリーテーブルを起動時に自動作成する必要があるかどうかを決定します。	true
keyspace	使用するキースペースの名前。	Infinispan
entry-table	エントリーを格納するテーブルの名前。	InfinispanEntries
consistency-level	クエリーに使用する整合性レベル	LOCAL_ONE
serial-consistency-level	クエリーに使用するシリアル整合性レベル	SERIAL

connection-pool も以下の要素で定義できます。

表19.3 接続プール設定パラメーター

パラメーター名	説明	デフォルト値
pool-timeout-millis	ホストプールからの接続が利用できない場合にドライバーがブロックする時間。このタイムアウトの後にドライバーは次のホストを試行します。	5
heartbeat-interval-seconds	アクティビティーが実行されていない場合に接続がドロップされるのを防ぐためのアプリケーション側のハートビート。無効にするには 0 に設定します。	30
idle-timeout-seconds	アイドル接続が削除される前のタイムアウト。	120

バグを報告する

19.10. カスタムキャッシュストア

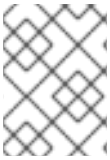
カスタムキャッシュストアは Red Hat JBoss Data Grid キャッシュストアのカスタマイズされた実装です。

カスタムキャッシュストア (またはローダー) を作成するには、必要に応じて以下のインターフェースのすべてまたはサブセットを実装します。

- **CacheLoader**

- **CacheWriter**
- **AdvancedCacheLoader**
- **AdvancedCacheWriter**
- **ExternalStore**
- **AdvancedLoadWriteStore**

インターフェースの個々の機能については、「キャッシュローダーとキャッシュライター」を参照してください。



注記

AdvancedCacheWriter が実装されない場合は、該当するライターを使用して、失効したエントリーをパージまたはクリアできません。

注記

AdvancedCacheLoader が実装されない場合、該当するローダーに格納されたエントリーはプリロードに使用されません。

既存のキャッシュストアを新しい API に移行するか、または新しいストア実装を作成するには、**SingleFileStore** などを使用します。**SingleFileStore** サンプルコードを参照するには、JBoss Data Grid ソースコードをダウンロードします。

以下の手順に従って、カスタマーポータルから **SingleFileStore** サンプルコードをダウンロードします。

手順19.6 JBoss Data Grid ソースコードのダウンロード

1. Red Hat カスタマーポータルにアクセスするには、ブラウザーで https://access.redhat.com/home に移動します。
2. **Downloads** をクリックします。
3. **JBoss 開発および管理** というラベルの付いたセクションで、**Red Hat JBoss Data Grid** をクリックします。
4. **Red Hat login** フィールドと **Password** フィールドに該当する認証情報を入力し、**Log In** をクリックします。
5. ダウンロード可能なファイルのリストから、**Red Hat JBoss Data Grid 7 ソースコード** を見つけ、**Download** をクリックします。ファイルを必要な場所に保存し、展開します。
6. **jboss-datagrid-7.0.0-sources/infinispan-8.3.0.Final-redhat-1-src/core/src/main/java/org/infinispan/persistence/file/SingleFileStore.java** で **SingleFileStore** ソースコードを見つけます。

バグを報告する

19.10.1. カスタムキャッシュストアの Maven アーキタイプ

カスタムキャッシュストアの開発は、Maven アーキタイプを使用して簡単に始めることができます。アーキタイプを作成すると、正しいディレクトリーレイアウトとサンプルコードと共に新しい Maven プロジェクトが生成されます。

手順19.7 Maven アーキタイプの生成

1. JBoss Data Grid Maven リポジトリーが Red Hat JBoss Data Grid の 『Getting Started Guide』に記載された手順に従ってインストールされていることを確認します。
2. コマンドプロンプトを開き、以下のコマンドを実行して現在のディレクトリーでアーキタイプを生成します。

```
mvn -Dmaven.repo.local="path/to/unzipped/jboss-datagrid-7.0.0-maven-
repository/"
  archetype:generate
  -DarchetypeGroupId=org.infinispan
  -DarchetypeArtifactId=custom-cache-store-archetype
  -DarchetypeVersion=8.3.0.Final-redhat-1
```

注記

読みやすさを目的として上記のコマンドは複数の行に分割されていますが、実行時にはコマンドとすべての引数を 1 つの行で指定する必要があります。

バグを報告する

19.10.2. カスタムキャッシュストアの設定 (リモートクライアントサーバーモード)

以下は、Red Hat JBoss Data Grid のリモートクライアントサーバーモードにおけるカスタムキャッシュストアの設定例になります。

例19.2 カスタムキャッシュストアの設定

```
<distributed-cache name="cacheStore" mode="SYNC" segments="20"
owners="2" remote-timeout="30000">
    <store class="my.package.CustomCacheStore">
        <property name="customStoreProperty">10</property>
    </store>
</distributed-cache>
```

この設定例で使用された要素とパラメーターの詳細については、「キャッシュストア設定の詳細 （リモートクライアントサーバーモード)」を参照してください。

バグを報告する

19.10.2.1. オプション 1: デプロイメントを使用してカスタムキャッシュストアを追加 (リモートクライアントサーバーモード)

手順19.8 デプロイメントを使用してカスタムキャッシュストア .jar ファイルを JDG サーバーにデプロイ

1. 以下の Java サービスローダーファイル **META-**

INF/services/org.infinispan.persistence.spi.AdvancedLoadWriteStore をモジュールに追加し、以下のように参照をカスタムキャッシュストアクラスに追加します。

```
my.package.CustomCacheStore
```

2. jar を **$JDG_HOME/standalone/deployments/** ディレクトリーにコピーします。
3. .jar ファイルがサーバーで利用可能な場合は、以下のメッセージがログに表示されます。

```
JBAS010287: Registering Deployed Cache Store service for store
'my.package.CustomCacheStore'
```

4. 「カスタムキャッシュストア」で示されたように、**infinispan-core** サブシステムで、インターフェースをオーバーライドするクラスを指定して **cache-container** 内部にキャッシュのエントリーを追加します。

```
<subsystem xmlns="urn:infinispan:server:core:8.3">
  [...]
  <distributed-cache name="cacheStore" mode="SYNC" segments="20"
owners="2" remote-timeout="30000"">
    <store class="my.package.CustomCacheStore">
      <!-- If custom properties are included these may be specified
as below -->
      <property name="customStoreProperty">10</property>
    </store>
  </distributed-cache>
  [...]
</subsystem>
```

バグを報告する

19.10.2.2. オプション 2: CLI を使用してカスタムキャッシュストアを追加 (リモートクライアントサーバーモード)

手順19.9 CLI を使用してカスタムキャッシュストア .jar ファイルを JDG サーバーにデプロイ

1. 以下のコマンドを実行して JDG サーバーに接続します。

```
[$JDG_HOME] $ bin/cli.sh --connect --controller=$IP:$PORT
```

2. 以下のコマンドを実行して、.jar ファイルをデプロイします。

```
deploy /path/to/artifact.jar
```

バグを報告する

19.10.2.3. オプション 3: JON を使用してカスタムキャッシュストアを追加 (リモートクライアントサーバーモード)

手順19.10 JBoss Operation Network を使用してカスタムキャッシュストア .jar ファイルを JDG サーバーにデプロイ

1. JON ログインします。
2. 上部のバーの **Bundles** に移動します。
3. **New** ボタンをクリックし、**Recipe** ラジオボタンを選択します。
4. 以下の例のように、ストアを参照するデプロイメントバンドルファイルの内容を挿入します。

```
<?xml version="1.0"?>
<project name="cc-bundle" default="main"
xmlns:rhq="antlib:org.rhq.bundle">

  <rhq:bundle name="Mongo DB Custom Cache Store" version="1.0"
description="Custom Cache Store">
   <rhq:deployment-unit name="JDG" compliance="full">
     <rhq:file name="custom-store.jar"/>
   </rhq:deployment-unit>
  </rhq:bundle>

  <target name="main" />

</project>
```

5. **Next** ボタンを押し **Bundle Groups** 設定ウィザードページに進み、もう一度 **Next** ボタンを押します。
6. ファイルアップローダーでカスタムキャッシュストア **.jar** ファイルを指定し、**Upload** を押してファイルをアップロードします。
7. **Next** ボタンを押し、**Summary** 設定ウィザードページに進みます。バンドル設定を終了するために **Finish** ボタンを押します。
8. 上部のバーの **Bundles** タブに戻ります。
9. 新しく作成されたバンドルを選択し、**Deploy** ボタンをクリックします。
10. **Destination Name** を入力し、適切なリソースグループを選択します。このグループは JDG サーバーでのみ構成される必要があります。
11. **Base Location** のラジオボックスグループから **Install Directory** を選択します。
12. 下の **Deployment Directory** テキストフィールドに **/standalone/deployments** と入力します。
13. デフォルトのオプションを使用してウィザードを続行します。
14. サーバーのホストで以下のコマンドを使用してデプロイメントを検証します。

```
find $JDG_HOME -name "custom-store.jar"
```

15. バンドルが **$JDG_HOME/standalone/deployments** にインストールされていることを確認します。

上記の手順が完了したら、.jar ファイルが正常にアップロードされ、JDG サーバーによって登録されます。

注記

JON プラグインは JBoss Data Grid 7.0 で非推奨となり、以降のバージョンでは除去される予定です。

バグを報告する

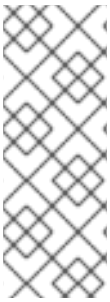
19.10.3. カスタムキャッシュストアの設定 （ライブラリーモード)

以下は、Red Hat JBoss Data Grid のライブラリーモードにおけるカスタムキャッシュストアの設定例になります。

例19.3 カスタムキャッシュストアの設定

```
<persistence>
 <store class="org.infinispan.custom.CustomCacheStore"
        preload="true"
        shared="true">
  <properties>
   <property name="customStoreProperty"
      value="10" />
  </properties>
 </store>
</persistence>
```

この設定例で使用された要素とパラメーターの詳細については、「キャッシュストア設定の詳細 （ライブラリモード)」を参照してください。



注記

カスタムキャッシュストアクラスは、Red Hat JBoss Data Grid が使用されるクラスパスに指定する必要があります。ほとんどの場合、これは、カスタムキャッシュストアをアプリケーションと共にパッケージ化して実行されます。ただし、これは Red Hat JBoss Enterprise Application Platform の 『管理および設定ガイド』で説明されているように、EAP に対してカスタムキャッシュストアをモジュールとして定義し、依存関係としてリストすることによっても実行できます。

バグを報告する

パート VIII. パッシベーションのセットアップ

第20章 アクティベーションモードとパッシベーションモード

アクティベーションは、エントリーをメモリーへロードし、キャッシュストアから削除する処理のことです。ストアに存在し、メモリーには存在しないエントリー (パッシベートされたエントリー) へスレッドがアクセスしようとした時にアクティベーションが発生します。

パッシベーションモードでは、エントリーがメモリーからエビクトされた後にエントリーをキャッシュストアに格納することができます。パッシベーションは、不必要で潜在的にコストのかかる書き込みが発生しないようにします。パッシベーションは頻繁に使用または参照されるため、メモリーからエビクトされないエントリーに使用されます。

パッシベーションが有効である場合、キャッシュストアはオーバーフロータンクとして使用されます。オーバーフロータンクは、メモリーページをディスクへスワップするオペレーティングシステムの仮想メモリー実装と似ています。

パッシベーションフラグは、パッシベーションモードの切り替えに使用されます。パッシベーションモードは、エントリーがメモリーからエビクトされた後でのみキャッシュストアにエントリーを格納するモードです。

バグを報告する

20.1. パッシベーションモードの利点

パッシベーションモードの主な利点は、不必要で潜在的に費用のかかる書き込みの発生を防ぐことです。これは、エントリーが頻繁に使用されたり参照されたりする場合に便利で、エントリーはメモリーからエビクトされません。

バグを報告する

20.2. パッシベーションの設定

Red Hat JBoss Data Grid のリモートクライアントサーバーモードでは、パッシベーションの切り替えを実行するために ***passivation*** パラメーターをキャッシュストア要素に追加します。

例20.1 リモートクライアントサーバーモードでのパッシベーションの切り替え

```
<local-cache name="customCache"/>
 <!-- Additional configuration elements for local-cache here -->
 <file-store passivation="true"
  <!-- Additional configuration elements for file-store here -->
</local-cache>
```

ライブラリーモードでは、パッシベーションを切り替えるために ***passivation*** パラメーターを **persistence** 要素に追加します。

例20.2 ライブラリーモードでのパッシベーションの切り替え

```
<persistence passivation="true">
   <!-- Additional configuration elements here -->
</persistence>
```

バグを報告する

20.3. エビクションとパッシベーション

エントリーの1つのコピーがメモリーまたはキャッシュストアに維持されるようにするため、パッシベーションはエビクションと共に使用してください。

通常のキャッシュストアの代わりにパッシベーションを使用する主な理由は、パッシベーションを使用するとエントリーの更新に必要なリソースが少なくなることにあります。パッシベーションではキャッシュストアへの更新が不要になるためです。

バグを報告する

20.3.1. エビクションとパッシベーションの用途

エビクションポリシーが原因で、パッシベーションが有効な間にキャッシュからエントリーがエビクトされた場合、結果として以下の処理が行われます。

- パッシベートされたエントリーに関する通知がキャッシュリスナーへ送信されます。
- エビクトされたエントリーが保存されます。

エビクトされたエントリーの読み出しを試みると、エントリーがキャッシュローダーよりメモリーへレイジーにロードされます。エントリーとその子エントリーは、ロードされた後にキャッシュローダーより削除され、エントリーのアクティベーションに関する通知がキャッシュリスナーへ送信されます。

バグを報告する

20.3.2. パッシベーションが無効な場合のエビクションの例

以下の例は、パッシベーションが無効な状態でエビクションを操作する間のメモリーおよび永続ストアの状態を示しています。

表20.1 パッシベーションが無効な場合のエビクション

手順	メモリー内のキー	ディスク上のキー
keyOne の挿入	メモリー: **keyOne**	ディスク: **keyOne**
keyTwo の挿入	メモリー: **keyOne**、 **keyTwo**	ディスク: **keyOne**、 **keyTwo**
エビクションスレッドが実行され、**keyOne** をエビクトする	メモリー: **keyTwo**	ディスク: **keyOne**、 **keyTwo**
keyOne の読み取り	メモリー: **keyOne**、 **keyTwo**	ディスク: **keyOne**、 **keyTwo**
エビクションスレッドが実行され、**keyTwo** をエビクトする	メモリー: **keyOne**	ディスク: **keyOne**、 **keyTwo**
keyTwo の削除	メモリー: **keyOne**	ディスク: **keyOne**

バグを報告する

20.3.3. パッシベーションが有効な場合のエビクションの例

以下の例は、パッシベーションが有効な状態でエビクションを操作する間のメモリーおよび永続ストアの状態を示しています。

表20.2 パッシベーションが有効な場合のエビクション

手順	メモリー内のキー	ディスク上のキー
keyOne の挿入	メモリー: **keyOne**	ディスク:
keyTwo の挿入	メモリー: **keyOne**、 **keyTwo**	ディスク:
エビクションスレッドが実行され、**keyOne** をエビクトする	メモリー: **keyTwo**	ディスク: **keyOne**
keyOne の読み取り	メモリー: **keyOne**、 **keyTwo**	ディスク:
エビクションスレッドが実行され、**keyTwo** をエビクトする	メモリー: **keyOne**	ディスク: **keyTwo**
keyTwo の削除	メモリー: **keyOne**	ディスク:

バグを報告する

パート IX. キャッシュ書き込みのセットアップ

第21章 キャッシュ書き込みモード

Red Hat JBoss Data Grid では、単一または複数のキャッシュストアを用いた設定オプションが使用できます。このようなオプションにより、共有の JDBC データベースやローカルファイルシステムなど、永続的な場所にデータを格納することができます。JBoss Data Grid は、以下の 2 つのキャッシングモードをサポートします。

- ライトスルー (同期)
- ライトビハインド (非同期)

バグを報告する

21.1. ライトスルーキャッシング

Red Hat JBoss Data Grid のライトスルー (または同期) モードは、クライアントがキャッシュエントリーを更新する時に (通常は **Cache.put()** 呼び出し経由)、JBoss Data Grid が基盤のキャッシュストアを見つけ、更新するまで呼び出しが返されないようにします。この機能により、キャッシュストアへの更新をクライアントスレッド境界内で終了させることができます。

バグを報告する

21.1.1. ライトスルーキャッシングのメリットとデメリット

ライトスルーキャッシングのメリット

ライトスルーモードの主な利点は、キャッシュとキャッシュストアが同時に更新されるため、キャッシュストアとキャッシュの内容の整合性を保つことができることです。

ライトスルーキャッシングのデメリット

キャッシュストアはキャッシュエントリーと同時に更新されるため、キャッシュストアへのアクセスや更新と同時に発生するキャッシュ操作のパフォーマンスが低下します。

バグを報告する

21.1.2. ライトスルーキャッシングの設定 (ライブラリーモード)

ライトスルーまたは同期キャッシュストアの設定には、特別な設定操作は必要ありません。すべてのキャッシュストアは、明示的にライトビハインドまたは非同期とマーク付けされていない限り、ライトスルーまたは同期になります。以下の手順は、ライトスルーの共有されないローカルファイルキャッシュストアの設定ファイルの例について説明しています。

手順21.1 ライトスルーのローカルファイルキャッシュストアの設定

```
<local-cache name="persistentCache">
  <persistence>
      <file-store fetch-state="true"
       purge="false"
       shared="false"
       location="${java.io.tmpdir}"/>
  </persistence>
</local-cache>
```

1. ***name*** パラメーターは、使用する **local-cache** の名前を指定します。

2. ***fetch-state*** パラメーターは、クラスターへ参加する時に永続状態が取り込まれるかを決定します。クラスター環境でレプリケーションとインバリデーションを使用している場合は、これを **true** に設定します。さらに、複数のキャッシュストアがチェーンされている場合、1 つのキャッシュストアのみがこのプロパティーを有効に設定できます。共有キャッシュストアが使用されている場合、キャッシュは、このプロパティーが **true** に設定されているにも関わらず、永続状態の転送を許可しません。***fetch-state*** パラメーターはデフォルトでは **false** に設定されます。

3. ***purge*** パラメーターは、キャッシュの初回起動時にキャッシュがパージされるかどうかを制御します。

4. ***shared*** パラメーターは、複数のキャッシュストアインスタンスがキャッシュストアを共有する場合に使用され、キャッシュストアレベルで定義されるようになりました。複数のキャッシュインスタンスが同じ変更内容を複数回書き込まないようにするため、このパラメーターを設定することができます。このパラメーターに有効な値は ***true*** と ***false*** です。

バグを報告する

21.2. ライトビハインドキャッシング

Red Hat JBoss Data Grid のライトビハインド (非同期) モードでは、キャッシュの更新が非同期的にキャッシュストアへ書き込みされます。非同期的な更新は、キャッシュと対話するクライアントスレッド以外のスレッドによってキャッシュストアの更新が実行されるようにします。

ライトビハインドモードの主な利点は、キャッシュ操作のパフォーマンスが基礎のストア更新によって影響されないことです。ただし、非同期の更新であるため、キャッシュと比較した場合にキャッシュストアに古いデータが短期間存在することになります。

バグを報告する

21.2.1. スケジュール外のライトビハインドストラテジーについて

スケジュール外のライトビハインドストラテジーモードでは、Red Hat JBoss Enterprise Data Grid は保留の変更を平行して適用し、変更をできるだけ早く保管しようとします。そのため、複数のスレッドが変更の完了を待つことになります。これらの変更が完了すると、スレッドが使用可能になり、基礎のキャッシュストアに適用されます。

このストラテジーは、待ち時間が短く、運用コストが低いキャッシュストアに適しています。例としては、キャッシュストアがキャッシュ自体に対してローカルとなる、共有されていないローカルのファイルベースキャッシュストアなどが挙げられます。このストラテジーを使用すると、キャッシュの内容とキャッシュストアの内容に不整合が生じる時間が、可能な限り最短の間隔まで削減されます。

バグを報告する

21.2.2. スケジュール外のライトビハインドストラテジーの設定 (リモートクライアントサーバーモード)

Red Hat JBoss Data Grid のリモートクライアントサーバーモードでライトビハインドストラテジーを設定するには、次のように **write-behind** 要素をターゲットキャッシュストアの設定に追加します。

手順21.2 write-behind 要素

```
<file-store passivation="false"
            path="${PATH}"
```

```
            purge="true"
            shared="false">
    <write-behind modification-queue-size="1024"
                  shutdown-timeout="25000"
                  flush-lock-timeout="15000"
                  thread-pool-size="5" />
</file-store>
```

write-behind 要素は次の設定パラメーターを使用します。

1. ***modification-queue-size*** パラメーターは、非同期ストアの変更キューサイズを設定します。更新がキャッシュストアがキューを処理するよりも速く行なわれる場合に、非同期ストアは同期ストアのように動作します。ストアの動作は、キューが要素を許可できるようになるまで同期された状態になり、要素をブロックします。その後ストアの動作は再び非同期になります。

2. ***shutdown-timeout*** パラメーターは、キャッシュストアがシャットダウンするまでのミリ秒単位の時間を指定します。ストアが停止している場合でも、いくらかの変更が依然として適用される必要がある場合があります。タイムアウトの値として大きな値を設定すると、データを損失する可能性が低くなります。このパラメーターのデフォルト値は **25000** です。

3. ***flush-lock-timeout*** パラメーターは、定期的にフラッシュされる状態を保護するロックを取得するための時間 (ミリ秒単位) を指定します。このパラメーターのデフォルト値は **15000** です。

4. ***thread-pool-size*** パラメーターはスレッドプールのサイズを指定します。このスレッドプール内のスレッドによって、変更がキャッシュストアに適用されます。このパラメーターのデフォルト値は **5** です。

バグを報告する

21.2.3. スケジュール外のライトビハインドストラテジーの設定 (ライブラリーモード)

キャッシュのストアへのエントリーのライトビハインドストラテジーを有効にするには、以下のように **async** 要素をストア設定に追加します。

手順21.3 async 要素

```
<persistence>
    <singleFile location="${LOCATION}">
        <async enabled="true"
                    modificationQueueSize="1024"
                    shutdownTimeout="25000"
                    flushLockTimeout="15000"
                    threadPoolSize="5"/>
    </singleFile>
</persistence>
```

async 要素は次の設定パラメーターを使用します。

1. ***modificationQueueSize*** パラメーターは、非同期ストアの変更キューサイズを設定します。更新がキャッシュストアがキューを処理するよりも速く行なわれる場合に、非同期ストアは同期ストアのように動作します。ストアの動作は、キューが要素を受け入れるまで同期が取られた状態のままになり、要素をブロックします。その後ストアの動作は再び非同期になります。

2. ***shutdownTimeout*** パラメーターは、キャッシュストアがシャットダウンされた後の時間 (ミリ秒単位) を指定します。これにより、キャッシュがシャットダウンされた際に非同期ライターがデータをストアへフラッシュする時間が提供されます。このパラメーターのデフォルト値は **25000** です。

3. ***flushLockTimeout*** パラメーターは、定期的にフラッシュする状態を保護するロックを取得するための時間 (ミリ秒単位) を指定します。このパラメーターのデフォルト値は **15000** です。

4. ***threadPoolSize*** パラメーターは、変更をストアに同時に適用するスレッドの数を指定します。このパラメーターのデフォルト値は **5** です。

バグを報告する

パート X. キャッシュとキャッシュマネージャーのモニタリング

第22章 JAVA MANAGEMENT EXTENSIONS (JMX) のセットアップ

22.1. JAVA MANAGEMENT EXTENSIONS (JMX) について

Java Management Extensions (JMX) は、アプリケーション、デバイス、システムオブジェクト、およびサービス指向ネットワークを管理および監視するツールを提供する Java ベースの技術です。これらの各オブジェクトは **MBeans** によって管理および監視されます。

JMX はミドルウェア管理の事実上の業界標準です。そのため、Red Hat JBoss Data Grid では管理および統計情報を公開するために JMX が使用されます。

バグを報告する

22.2. RED HAT JBOSS DATA GRID における JMX の使用

Red Hat JBoss Data Grid インスタンスの管理は、関連する統計情報をできるだけ多く公開することを目的としています。この統計情報により、管理者は各インスタンスの状態を確認することができます。1 つのインストールは数十または数百のインスタンスを構成することがあるため、各インスタンスの統計情報を明確で簡潔に公開し、表示する必要があります。

このような統計情報を公開し、管理者に対して適切な情報を見やすく表示するため、JBoss Data Grid では、JMX は JBoss Operations Network (JON) と共に使用されます。

バグを報告する

22.3. JMX 統計レベル

JMX 統計は、次の 2 つのレベルで有効にすることが可能です。

- 個別のキャッシュインスタンスによって管理情報が生成されるキャッシュレベル。
- **CacheManager** レベル。このレベルでは、**CacheManager** エンティティーが **CacheManager** より作成されたすべてのキャッシュインスタンスを管理します。そのため、管理情報は個別のキャッシュではなく、これらすべてのキャッシュインスタンスに対して生成されます。

重要

Red Hat JBoss Data Grid では、統計はリモートクライアントサーバーモードでデフォルトにより有効になり、ライブラリーモードでデフォルトにより無効になります。統計は JBoss Data Grid の状態を評価する際に役に立ちますが、パフォーマンスにはマイナスの影響を与えるため、統計が不要な場合には無効にする必要があります。

バグを報告する

22.4. キャッシュインスタンスに対して JMX を有効にする

キャッシュレベルでは、宣言的に、またはプログラムを用いて、次のように JMX 統計を有効にすることができます。

キャッシュレベルで JMX を宣言的に有効にする

デフォルトキャッシュインスタンスに対する **<default>** 要素内または特定のキャッシュに対するターゲット **<local-cache>** 要素下に、次のスニペットを追加します。

```
<jmxStatistics enabled="true"/>
```

バグを報告する

22.5. CACHEMANAGER に対して JMX を有効にする

CacheManager レベルでは、宣言的に、またはプログラムを用いて次のように JMX 統計を有効にすることができます。

CacheManager レベルで JMX を宣言的に有効にする

次の **<global>** 要素を追加して、**CacheManager** レベルで JMX を宣言的に有効にします。

```
<globalJmxStatistics enabled="true"/>
```

バグを報告する

22.6. ローリングアップグレードの使用時に JMX で CACHESTORE を無効にする

Red Hat JBoss Data Grid は、**RollingUpgradeManager** MBean で ***disconnectSource*** 操作を呼び出すことで、JMX を使用して CacheStore を無効にすることができます。

関連トピック:

- 「RollingUpgradeManager」

バグを報告する

22.7. 複数の JMX ドメイン

複数の **CacheManager** インスタンスが 1 つの仮想マシンに存在したり、キャッシュインスタンスの名前が **CacheManager** と異なる場合に、複数の JMX ドメインが使用されます。

この問題を解決するには、JMX や JBoss Operations Network などの監視ツールが簡単に識別および使用できるような名前を各 **CacheManager** に付けるようにします。

CacheManager の名前を宣言的に設定する

次のスニペットを関係する **CacheManager** 設定に追加します。

```
<globalJmxStatistics enabled="true" cacheManagerName="Hibernate2LC"/>
```

バグを報告する

22.8. MBEANS

MBean は、サービス、コンポーネント、デバイス、またはアプリケーションなどの管理可能なリソースを表します。

Red Hat JBoss Data Grid は複数の側面を監視および管理する **MBeans** を提供します。たとえば、トランスポート層で統計を提供する **MBeans** などが提供されます。JBoss Data Grid サーバーは、JMX 統計で設定されます。JMX 統計はホスト名、ポート、読み取りバイト、書き込みバイト、およびワーカースレッドの数を提供する **MBean** で、次の場所に存在します。

```
jboss.infinispan:type=Server,name=<Memcached|Hotrod>,component=Transport
```

MBeans は、2 つの JMX ドメイン下で利用可能です。

- jboss.as - これらの **MBeans** はサーバーサブシステムにより作成されます。
- jboss.infinispan - これらの **MBeans** は、内蔵モードで作成されたものと対称的になります。

Red Hat JBoss Data Grid では、jboss.infinispan 下の **MBeans** のみを使用する必要があります (jboss.as 下にあるものは Red Hat JBoss Enterprise Application Platform 用に使用されます)。

注記

利用可能な MBeans の一覧、それらのサポートされている操作と属性については、付録で参照することができます。

バグを報告する

22.8.1. MBean について

キャッシュマネージャーまたはキャッシュレベルのいずれかで JMX レポートが有効になっている場合、JConsole や VisualVM などの標準的な JMX GUI を使用して Red Hat JBoss Data Grid を実行する Java 仮想マシンに接続します。接続した場合、次の **MBean** を使用できます。

- キャッシュマネージャーレベルの JMX 統計が有効になっている場合、***jboss.infinispan:type=CacheManager,name="DefaultCacheManager"*** という名前の **MBean** が存在し、キャッシュマネージャー **MBean** によってプロパティーが指定されます。
- キャッシュレベルの JMX 統計が有効になっている場合、使用される設定に応じて複数の **MBean** が表示されます。たとえば、ライトビハインドキャッシュストアが設定されている場合、キャッシュストアコンポーネントに属するプロパティーを公開する **MBean** が表示されます。すべてのキャッシュレベルの **MBeans** は同じ形式を使用します。

  ```
  jboss.infinispan:type=Cache,name="<name-of-cache>(<cache-
  mode>)",manager="<name-of-cache-manager>",component=<component-name>
  ```

  この形式の詳細は次のとおりです。

  - **cache-container** 要素の ***default-cache*** 属性を使用してキャッシュのデフォルト名を指定します。
  - ***cache-mode*** はキャッシュのキャッシュモードに置き換えられます。可能な列挙値を小文字にしたものがキャッシュモードを表します。
  - ***component-name*** は、 JMX 参考ドキュメントにある JMX コンポーネント名の 1 つに置き換えられます。

たとえば、同期分散に対して設定されたデフォルトキャッシュのキャッシュストア JMX コンポーネント **MBean** の名前は次のようになります。

```
jboss.infinispan:type=Cache,name="default(dist_sync)",
manager="default",component=CacheStore
```

ユーザー定義の名前でサポートされていない文字が使用されないようにするため、キャッシュおよびキャッシュマネージャーの名前は引用符で囲まれています。

バグを報告する

22.8.2. デフォルトでない MBean サーバーでの MBean の登録

使用されるすべての MBean がデフォルトで登録される場所は、標準の JVM MBeanServer プラットフォームです。ユーザーは代替の MBeanServer インスタンスを設定することもできます。MBeanServerLookup インターフェースを実装して、確実に **getMBeanServer()** メソッドが必要な (デフォルト以外の) MBeanServer を返すようにします。

デフォルト以外の場所を設定して MBeans を登録するには、実装を作成してからクラスの完全修飾名を用いて Red Hat JBoss Data Grid を設定します。例は次のとおりです。

完全修飾ドメイン名を宣言的に追加する

次のスニペットを追加します。

```
<globalJmxStatistics enabled="true"
mBeanServerLookup="com.acme.MyMBeanServerLookup"/>
```

バグを報告する

第23章 JBOSS OPERATIONS NETWORK (JON) のセットアップ

23.1. JBOSS OPERATIONS NETWORK (JON) について

JBoss Operations Network (JON) は、アプリケーションのライフサイクルを開発、テスト、デプロイおよび監視するために使用される JBoss の管理プラットフォームです。JBoss Operations Network は JBoss のエンタープライズ管理ソリューションで、サーバーにまたがる複数の Red Hat JBoss Data Grid インスタンスを管理するための使用が推奨されます。JBoss Operations Network のエージェントおよび自動ディスカバリー機能は、JBoss Data Grid のキャッシュマネージャーおよびキャッシュインスタンスの監視を円滑にします。JBoss Operations Network は主なランタイムパラメーターと統計をグラフィカルに表示し、管理者がしきい値を設定したり、使用率が設定したしきい値を上回ったり、下回ったりした時に通知を受けるようにすることが可能です。

重要

Red Hat JBoss Data Grid のリモートクライアントサーバーモードでは、統計がデフォルトで有効になります。統計は JBoss Data Grid の状態を評価する際に役に立ちますが、パフォーマンスにはマイナスの影響を与えるため、統計が不要な場合には無効にする必要があります。JBoss Data Grid のライブラリーモードでは、統計はデフォルトで無効になり、必要な場合は明示的に有効にする必要があります。

重要

JBoss Data Grid のライブラリーモード用 JBoss Operations Network ライブラリープラグインの完全な機能を実現するには、**`Update 04`** 以上のパッチバージョンで JBoss Operations Network 3.3.0 にアップグレードします。JBoss Operations Network のアップグレードについては、JBoss Operations Network 『Installation Guide』 の「Upgrading JBoss ON」のセクションを参照してください。

注記

JON プラグインは JBoss Data Grid 7.0 で非推奨となり、以降のバージョンでは除去される予定です。

バグを報告する

23.2. JBOSS OPERATIONS NETWORK (JON) のダウンロード

23.2.1. JBoss Operations Network (JON) インストールの前提条件

JBoss Operations Network を Red Hat JBoss Data Grid にインストールするには、以下が必要です。

- Linux、Windows、または Mac OSX オペレーティングシステム、および x86_64、i686、または ia64 プロセッサー。
- JBoss Operations Network サーバーおよび JBoss Operations Network エージェントの両方を実行するには、Java 6 以上が必要です。
- JBoss Operations Network サーバーとエージェントで同期されたクロック。
- 外部データベースをインストールする必要があります。

バグを報告する

23.2.2. JBoss Operations Network のダウンロード

以下の手順を使用して、カスタマーポータルから Red Hat JBoss Operations Network (JON) をダウンロードします。

手順23.1 JBoss Operations Network のダウンロード

1. Red Hat カスタマーポータルにアクセスするには、ブラウザーで https://access.redhat.com/home に移動します。
2. **Downloads** をクリックします。
3. **JBoss 開発および管理** というラベルの付いたセクションで、**Red Hat JBoss Data Grid** をクリックします。
4. **Red Hat login** フィールドと **Password** フィールドに該当する認証情報を入力し、**Log In** をクリックします。
5. **Version** ドロップダウンメニューリストから適切なバージョンを選択します。
6. 必要なダウンロードファイルの横にある **Download** ボタンをクリックします。

バグを報告する

23.2.3. リモート JMX ポートの値

Red Hat JBoss Data Grid インスタンスを見つけられるようにするには、ポートの値を提供する必要があります。使用できるポートの値を使用します。

一意 (使用可能な) のリモート JMX ポートを提供し、単一のマシン上で複数の JBoss Data Grid インスタンスを実行します。ローカルで実行されている JBoss Operations Network エージェントは、リモートポートの値を使用して各インスタンスを見つけることができます。

バグを報告する

23.2.4. JBoss Operations Network (JON) プラグインのダウンロード

Red Hat カスタマーポータルから Red Hat JBoss Data Grid の JBoss Operations Network (JON) プラグインをダウンロードするにはこのタスクを実行します。

手順23.2 インストールファイルのダウンロード

1. Web ブラウザーで http://access.redhat.com を開きます。
2. ページ上部にあるメニュー内の **ダウンロード**をクリックします。
3. **JBoss Development and Management** の下にあるリストで**Red Hat JBoss Operations Network** をクリックします。
4. ログイン情報を入力します。

   「Software Downloads」ページに移動します。

5. **JBoss Operations Network プラグインをダウンロードします。**
   JBoss Data Grid の JBoss Operations Network プラグインを使用する予定であれば、**Product** ドロップダウンボックスか、または左側のメニューのいずれかから **JBoss ON for Data Grid** を選択します。

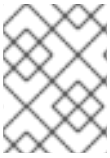
   a. **Red Hat JBoss Operations Network *VERSION* Base Distribution Download** ボタンをクリックします。

   b. **Data Grid Management Plugin Pack for JBoss ON *VERSION*** をダウンロードする手順を繰り返します。

バグを報告する

23.3. JBOSS OPERATIONS NETWORK サーバーのインストール

JBoss Operations Network のコアとなるのはサーバーです。このサーバーは、エージェントと通信し、インベントリーを維持し、リソース設定を管理し、コンテンツプロバイダーと対話し、中央管理 UI を提供します。



注記

JBoss Operations Network の設定方法についてさらに詳しくは、『JBoss Operations Network インストールガイド』を参照してください。

バグを報告する

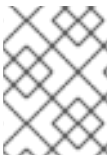
23.4. JBOSS OPERATIONS NETWORK エージェント

JBoss Operations Network エージェントはスタンドアロンの Java アプリケーションです。エージェントが管理を要求されるリソース数にかかわらず、1 台のマシンにつき 1 つのエージェントのみが必要です。

JBoss Operations Network エージェントは、完全に設定された状態では出荷されません。エージェントのインストールおよび設定が完了すると、エージェントをコンソールから Windows サービスとして実行したり、UNIX 環境でデーモンまたは **init.d** スクリプトとして実行したりできます。

JBoss Operations Network エージェントは、データを収集するために監視されるそれぞれのマシンにインストールする必要があります。

JBoss Operations Network Agent は、通常 Red Hat JBoss Data Grid が実行されているのと同じマシンにインストールされますが、複数のマシンがある場合は、エージェントはそれぞれのマシンにインストールされる必要があります。



注記

JBoss Operations Network エージェントの設定方法についてさらに詳しくは、JBoss Operations Network の『インストールガイド』を参照してください。

バグを報告する

23.5. リモートクライアントサーバーモードの JBOSS OPERATIONS NETWORK

Red Hat JBoss Data Grid のリモートクライアントサーバーモードでは、JBoss Operations Network プラグインを使って以下が実行されます。

- インストールおよび設定操作の開始および実行。
- リソースおよびメトリックスの監視。

リモートクライアントサーバーモードでは、JBoss Operations Network プラグインは、メトリックスを取得し、JBoss Data Grid サーバー上で各種操作を実行するために
JBoss Enterprise Application Platform の管理プロトコルを使用します。

バグを報告する

23.5.1. JBoss Operations Network プラグインのインストール (リモートクライアントサーバーモード)

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用に
JBoss Operations Network プラグインをインストールする方法について詳細に説明します。

1. **プラグインのインストール**
   - JBoss Data Grid サーバーの RHQ プラグインを ***$JON_SERVER_HOME*/plugins** にコピーします。
   - JBoss Enterprise Application Platform プラグインを ***$JON_SERVER_HOME*/plugins** にコピーします。

   サーバーはここでプラグインを自動的に検出し、これらをデプロイします。プラグインはデプロイメントが成功すると、プラグインディレクトリーから削除されます。
2. **プラグインの取得**
   JBoss Operations Network サーバーから利用可能なすべてのプラグインを取得します。これを実行するには、以下をエージェントのコンソールに入力します。

   ```
   plugins update
   ```
3. **インストール済みプラグインのリスト**
   以下のコマンドを使用して、JBoss Enterprise Application Platform プラグインと
   JBoss Data Grid サーバー rhq プラグインが正しくインストールされていることを確認します。

   ```
   plugins info
   ```

JBoss Operation Network が実行中の JBoss Data Grid サーバーを検出できるようになります。

バグを報告する

23.6. JBOSS OPERATIONS NETWORK リモートクライアントサーバーのプラグイン

23.6.1. JBoss Operations Network プラグインのメトリックス

表23.1 キャッシュコンテナーの JBoss Operations Network 特性 (キャッシュマネージャー)

特性名	表示名	説明
cache-manager-status	Cache Container Status	キャッシュコンテナーの現在のランタイム状態。
cluster-name	Cluster Name	クラスターの名前。
members	Cluster Members	クラスターのメンバーの名前。
coordinator-address	Coordinator Address	コーディネーターノードのアドレス。
local-address	Local Address	ローカルノードのアドレス。
version	Version	キャッシュマネージャーバージョン。
defined-cache-names	Defined Cache Names	このマネージャーに定義されたキャッシュ。

表23.2 キャッシュコンテナーの JBoss Operations Network メトリックス (キャッシュマネージャー)

メトリック名	表示名	説明
cluster-size	Cluster Size	クラスター内のメンバーの数。
defined-cache-count	Defined Cache Count	このマネージャーに定義されたキャッシュの数。
running-cache-count	Running Cache Count	このマネージャーで実行中のキャッシュの数。
created-cache-count	Created Cache Count	このマネージャーで実際に作成されたキャッシュの数。

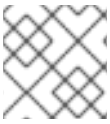
表23.3 キャッシュの JBoss Operations Network 特性

特性名	表示名	説明
cache-status	Cache Status	キャッシュの現在のランタイム状態。
cache-name	Cache Name	キャッシュの現在の名前。
version	Version	キャッシュバージョン。

表23.4 キャッシュについての JBoss Operations Network メトリックス

メトリック名	表示名	説明
cache-status	Cache Status	キャッシュの現在のランタイム状態。
number-of-locks-available	[LockManager] Number of locks available	現在利用可能な排他ロックの数。
concurrency-level	[LockManager] Concurrency level	LockManager の設定済みの平行性レベル。
average-read-time	[Statistics] Average read time	キャッシュでの読み取り操作が完了するまでに必要な平均のミリ秒数です。
hit-ratio	[Statistics] Hit ratio	ヒット数 (試行が成功した回数) を試行の合計数で割った結果 (パーセント単位) です。
elapsed-time	[Statistics] Seconds since cache started	キャッシュ起動後の秒数。
read-write-ratio	[Statistics] Read/write ratio	キャッシュの読み取り/書き込み比率 (パーセント単位)。
average-write-time	[Statistics] Average write time	キャッシュでの書き込み操作の完了に必要な平均のミリ秒数。
hits	[Statistics] Number of cache hits	キャッシュヒット数。
evictions	[Statistics] Number of cache evictions	キャッシュエビクション操作の数。
remove-misses	[Statistics] Number of cache removal misses	キーが見つからなかった場合のキャッシュ除去の回数。
time-since-reset	[Statistics] Seconds since cache statistics were reset	最後にキャッシュ統計がリセットされてからの秒数。
number-of-entries	[Statistics] Number of current cache entries	キャッシュ内の現在のエントリーの数。
stores	[Statistics] Number of cache puts	キャッシュの put 操作の回数。
remove-hits	[Statistics] Number of cache removal hits	キャッシュの remove 操作のヒット数。
misses	[Statistics] Number of cache misses	キャッシュミスの数。

メトリック名	表示名	説明
success-ratio	[RpcManager] Successful replication ratio	数値 (double) 形式でのレプリケーションの合計数に対する正常なレプリケーションの比率です。
replication-count	[RpcManager] Number of successful replications	成功したレプリケーションの数。
replication-failures	[RpcManager] Number of failed replications	失敗したレプリケーションの数。
average-replication-time	[RpcManager] Average time spent in the transport layer	トランスポート層で費やされた平均時間 (ミリ秒単位)。
commits	[Transactions] Commits	最終リセット時から実行されるトランザクションのコミット数。
prepares	[Transactions] Prepares	最終リセット時から実行されるトランザクションの準備回数。
rollbacks	[Transactions] Rollbacks	最終リセット時から実行されるトランザクションのロールバック回数。
invalidations	[Invalidation] Number of invalidations	インバリデーションの数。
passivations	[Passivation] Number of cache passivations	パッシベーションイベントの数。
activations	[Activations] Number of cache entries activated	アクティベーションイベントの数。
cache-loader-loads	[Activation] Number of cache store loads	キャッシュストアからロードされるエントリーの数。
cache-loader-misses	[Activation] Number of cache store misses	キャッシュストアに存在しなかったエントリーの数。
cache-loader-stores	[CacheStore] Number of cache store stores	キャッシュストアに保存されるエントリーの数。



注記

これらの統計の一部の収集はデフォルトで無効にされます。

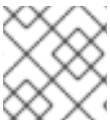
コネクターついての JBoss Operations Network メトリックス

Red Hat JBoss Data Grid の JBoss Operations Network (JON) プラグインによって提供されるメトリッ

クスは、REST と Hot Rod エンドポイント用のみです。REST プロトコルの場合、データは Web サブシステムのメトリックスから取得する必要があります。これらのエンドポイントのそれぞれについてさらに詳しくは、『スタートガイド』を参照してください。

表23.5 コネクターついての JBoss Operations Network メトリックス

メトリック名	表示名	説明
bytesRead	Bytes Read	読み込まれるバイト数。
bytesWritten	Bytes Written	書き込まれるバイト数。



注記

これらの統計の収集はデフォルトで無効にされます。

バグを報告する

23.6.2. JBoss Operations Network プラグイン操作

表23.6 キャッシュについての JBoss ON プラグイン操作

操作名	説明
Start Cache	キャッシュを起動します。
Stop Cache	キャッシュを停止します。
Clear Cache	キャッシュ内容をクリアします。
Reset Statistics	キャッシュによって収集される統計をリセットします。
Reset Activation Statistics	キャッシュによって収集されるアクティベーション統計をリセットします。
Reset Invalidation Statistics	キャッシュによって収集されるインバリデーション統計をリセットします。
Reset Passivation Statistics	キャッシュによって収集されるパッシベーション統計をリセットします。
Reset Rpc Statistics	キャッシュによって収集されるレプリケーション統計をリセットします。
Remove Cache	キャッシュコンテナーから所定のキャッシュを削除します。

操作名	説明
Record Known Global Keyset	アップグレードプロセスで取得するために、グローバルな既知のキーセットを既知のキーに記録します。
Synchronize Data	指定された移行プログラムを使用して、古いクラスターのデータをこれに同期します。
Disconnect Source	指定される移行プログラムに従って、ターゲットクラスターをソースクラスターから切り離します。

キャッシュバックアップについての JBoss Operations Network プラグイン操作

これらの操作に使用されるキャッシュバックアップは、データセンター間レプリケーションを使用して設定されます。JBoss Operations Network (JON) ユーザーインターフェースでは、それぞれのキャッシュバックアップはキャッシュの子になります。データセンター間のレプリケーションについてさらに詳しくは、35章*データセンター間のレプリケーションのセットアップ*を参照してください。

表23.7 キャッシュバックアップについての JBoss Operations Network プラグイン操作

操作名	説明
status	サイトの状態を表示します。
bring-site-online	サイトをオンラインにします。
take-site-offline	サイトをオフラインにします。

キャッシュ (トランザクション)

Red Hat JBoss Data Grid は、リモートクライアントサーバーモードでのトランザクションの使用をサポートしません。結果として、いずれのエンドポイントでもトランザクションを使用することができません。

バグを報告する

23.6.3. JBoss Operations Network プラグイン属性

表23.8 キャッシュの JBoss ON プラグイン属性 (トランスポート)

属性名	タイプ	説明
cluster	string	グループ通信クラスターの名前。
executor	string	トランスポートに使用されるエグゼキューター。

属性名	タイプ	説明
lock-timeout	long	トランスポートにおけるロックのタイムアウト期間。デフォルト値は **240000** です。
machine	string	トランスポートのマシン ID。
rack	string	トランスポートのラック ID。
site	string	トランスポートのサイト ID。
stack	string	トランスポートに使用される JGroups スタック。

バグを報告する

23.6.4. JBoss Operations Network (JON) を使用した新しいキャッシュの作成

以下の手順に従い、リモートクライアントサーバーモード向けの JBoss Operations Network (JON) を使用して新しいキャッシュを作成します。

手順23.3 リモートクライアントサーバーモードでの新しいキャッシュの作成

1. JBoss Operations Network コンソールにログインします。
    a. JBoss Operations Network コンソールに で **Inventory** をクリックします。
    b. コンソールの左側にある **Resources** リストから **Servers** を選択します。
2. サーバーリストから特定の Red Hat JBoss Data Grid サーバーを選択します。
    a. サーバー名の下にある **infinispan** をクリックし、次に **Cache Containers** をクリックします。
3. 新しく作成されたキャッシュの親になる任意のキャッシュコンテナーを選択します。
    a. 選択されたキャッシュコンテナーを右クリックします (例: **clustered**)。
    b. コンテキストメニューで、**Create Child** に移動し、**Cache** を選択します。
4. リソース作成ウィザードで新しいキャッシュを作成します。
    a. 新しいキャッシュ名を入力し、**Next** をクリックします。
    b. デプロイメントオプションでキャッシュ属性を設定し、**Finish** をクリックします。

注記

新しく追加されたリソースを表示するためにキャッシュのビューを更新します。リソースがインベントリーに表示されるまで数分かかることがあります。

バグを報告する

23.7. ライブラリーモードの JBOSS OPERATIONS NETWORK

Red Hat JBoss Data Grid のライブラリーモードでは、JBoss Operations Network プラグインを使って以下が実行されます。

- インストールおよび設定操作の開始および実行。
- リソースおよびメトリックスの監視。

ライブラリーモードでは、JBoss Operations Network プラグインは JBOSS Data Grid ライブラリーを使ってメトリックスを取得し、アプリケーション上で各種操作を実行するために JMX を使用します。

バグを報告する

23.7.1. JBoss Operations Network プラグインのインストール　(ライブラリーモード)

次の手順を使用して、Red Hat JBoss Data Grid のライブラリーモード向け JBoss Operations Network プラグインをインストールします。

手順23.4 JBoss Operations Network ライブラリーモードプラグインのインストール

1. **JBoss Operations Network コンソールを開きます。**
   a. JBoss Operations Network コンソールから、**Administration** を選択します。
   b. コンソールの左側にある **Configuration** オプションから **Agent Plugins** を選択します。

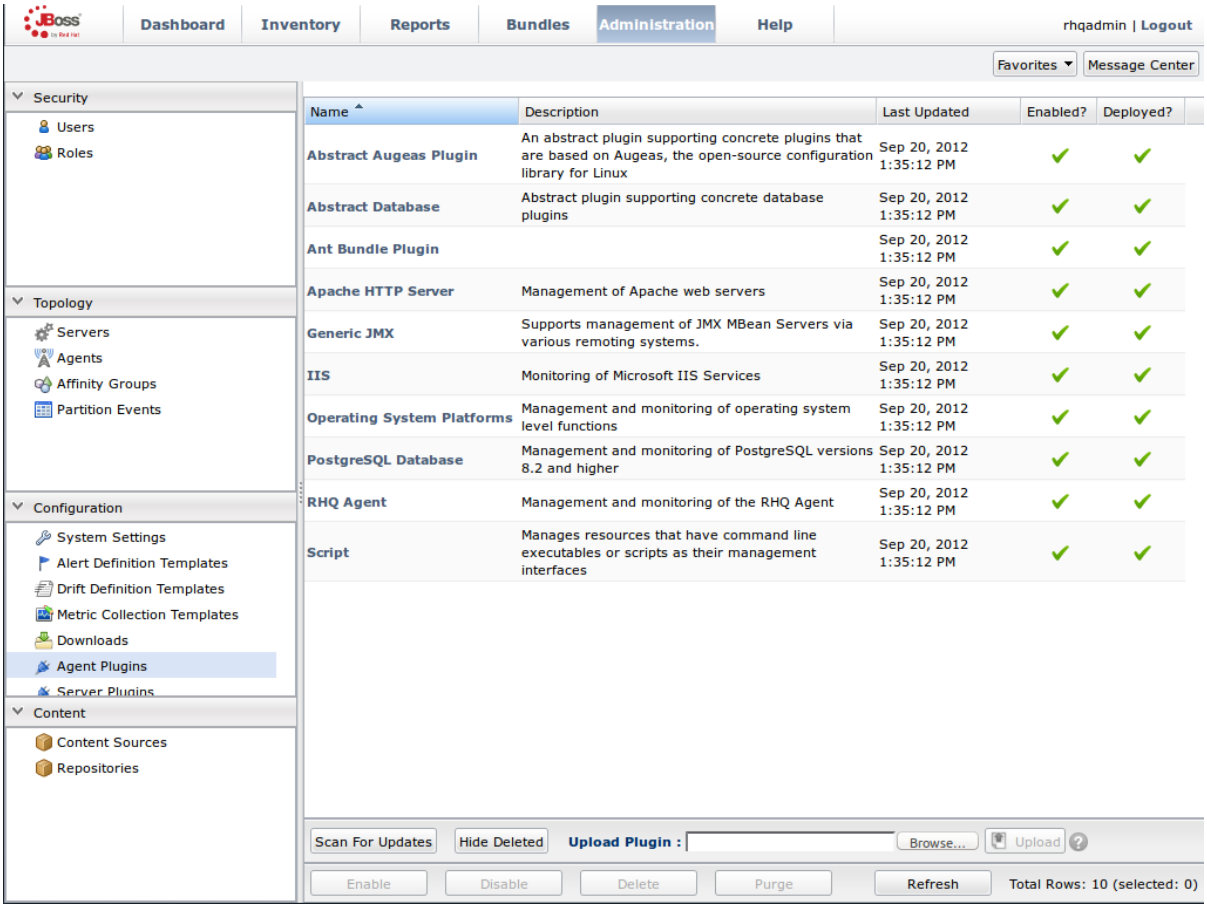


図23.1 JBoss Data Grid 用の JBoss Operations Network コンソール

2. ライブラリーモードプラグインをアップロードします。

   a. **Browse** をクリックし、ローカルファイルシステムで**InfinispanPlugin** を見つけます。

   b. **Upload** をクリックして、プラグインを JBoss Operations Network サーバーに追加します。

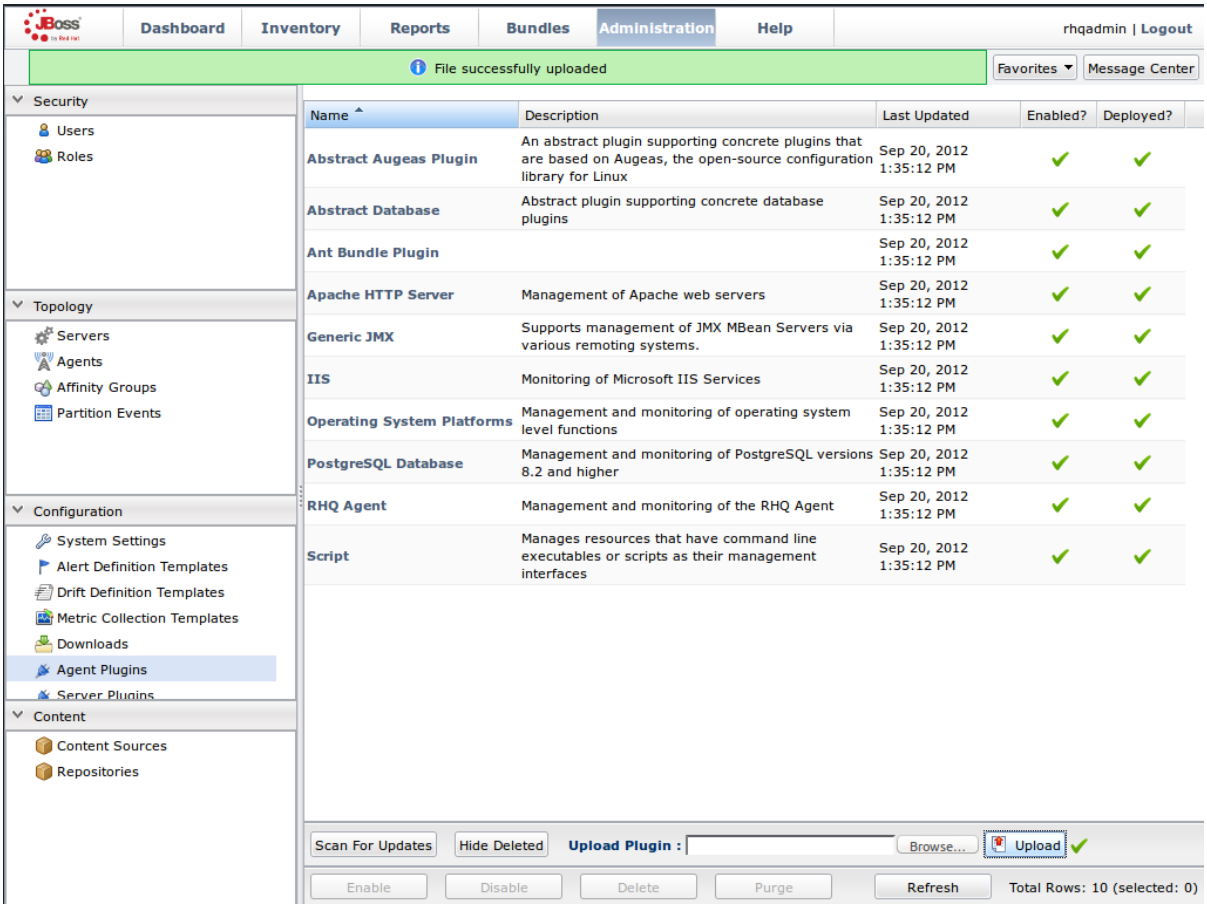


図23.2 `InfinispanPlugin` のアップロード。

3. **更新のためのスキャン**

   a. ファイルが正常にアップロードされたら、画面の下部にある **`Scan For Updates`** をクリックします。

   b. **`InfinispanPlugin`** がインストール済みプラグインのリストに表示されます。

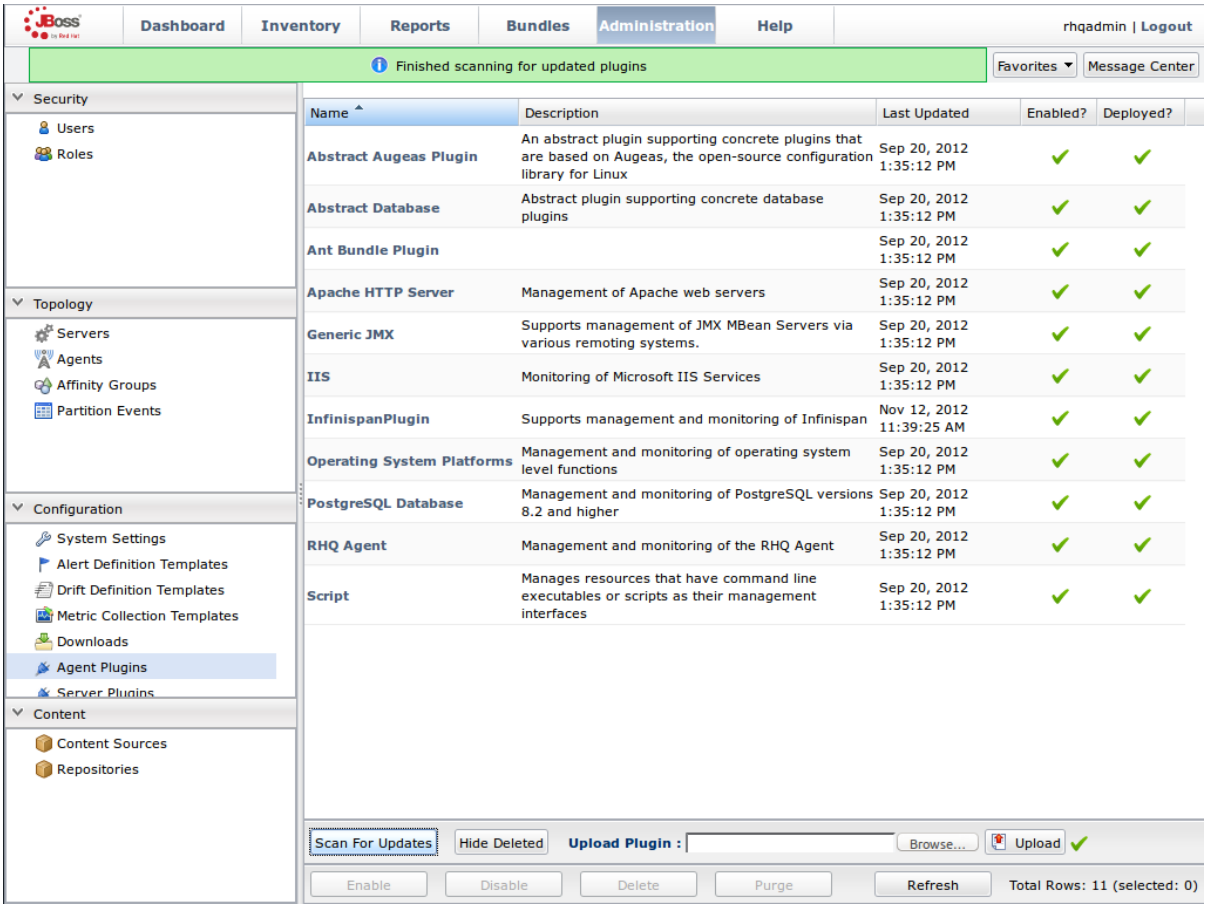


図23.3 更新済みプラグインのためのスキャン

バグを報告する

23.7.2. ライブラリーモードでの JBoss Data Grid インスタンスの追加

23.7.2.1. 前提条件

以下は、「スタンドアロンモードでデプロイされたアプリケーションの監視」、「ドメインモードでデプロイされたアプリケーションの監視」、および「ライブラリーモードでの JBoss Data Grid インスタンスの手動による追加」において共通する前提条件の一覧です。

- パッチが **Update 02** 以上の JBoss Operations Network (JON) 3.2.0 の正しく設定されたインスタンス。
- アプリケーションが実行されるサーバー上の JON Agent の実行中インスタンス。詳細については、「JBoss Operations Network エージェント」を参照してください。
- 完全な JDK を含む RHQ エージェントの操作インスタンス。エージェントに JDK の **tools.jar** ファイルへのアクセス権があることを確認します。JON エージェントの環境ファイル (**bin/rhq-env.sh**) で、完全な JDK ホームを参照するよう **RHQ_AGENT_JAVA_HOME** プロパティーの値を設定します。
- RHQ エージェントは、JBoss Enterprise Application Platform インスタンスと同じユーザーを使用して起動している必要があります。たとえば、JON エージェントを root 権限を持つユーザーとして実行し、JBoss Enterprise Application Platform プロセスを異なるユーザーとして実行しても予想どおりには機能しないため、この実行は避けるようにしてください。

- JBoss Data Grid Library Mode 用のインストール済み JON プラグイン。詳細については、「JBoss Operations Network プラグインのインストール （ライブラリーモード） 」を参照してください。
- パッチが **Update 02** 以上の JBoss Operation Networks 3.2.0 の **Generic JMX plugin**。
- 統計機能と監視機能を動作させるために、ライブラリーモードキャッシュの JMX 統計が有効な Red Hat JBoss Data Grid のライブラリーモードを使用したカスタムアプリケーション。キャッシュインスタンスの JMX 統計を有効にする方法については「キャッシュインスタンスに対して JMX を有効にする」 を参照し、キャッシュマネージャー用に JMX を有効にする方法については「 CacheManager に対して JMX を有効にする」を参照してください。
- Java 仮想マシン (JVM) は、JMX MBean Server を公開するために設定する必要があります。Oracle/Sun JDK については、http://docs.oracle.com/javase/1.5.0/docs/guide/management/agent.html を参照してください。
- JBoss Enterprise Application Platform 用に適切に追加され、設定された管理ユーザー。

バグを報告する

23.7.2.2. ライブラリーモードでの JBoss Data Grid インスタンスの手動による追加

Red Hat JBoss Data Grid インスタンスを JBoss Operations Network に手動で追加するには、JBoss Operations Network インターフェースで次の手順を使用します。

手順23.5 ライブラリーモードでの JBoss Data Grid インスタンスの追加

1. **プラットフォームのインポート**
   a. **Inventory** にナビゲートし、コンソールの左側の**Resources** リストから **Discovery Queue** を選択します。
   b. アプリケーションが実行されているプラットフォームを選択し、画面の下部で **Import** をクリックします。

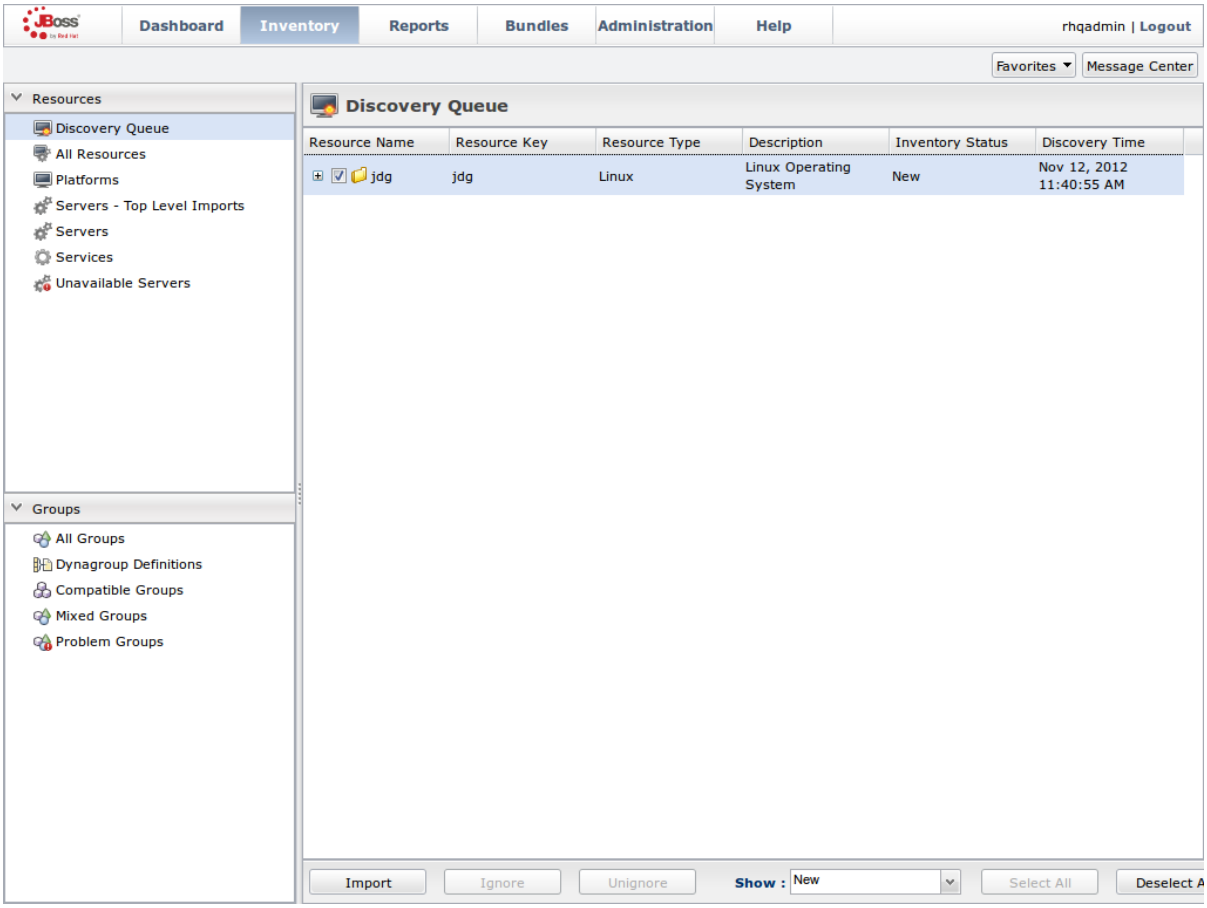


図23.4 Discovery Queue からのプラットフォームのインポート。

2. プラットフォーム上のサーバーへのアクセス

   a. **`jdg`** プラットフォームが **`Platforms`** リストに表示されます。

   b. 実行中のサーバーにアクセスするためにプラットフォームをクリックします。

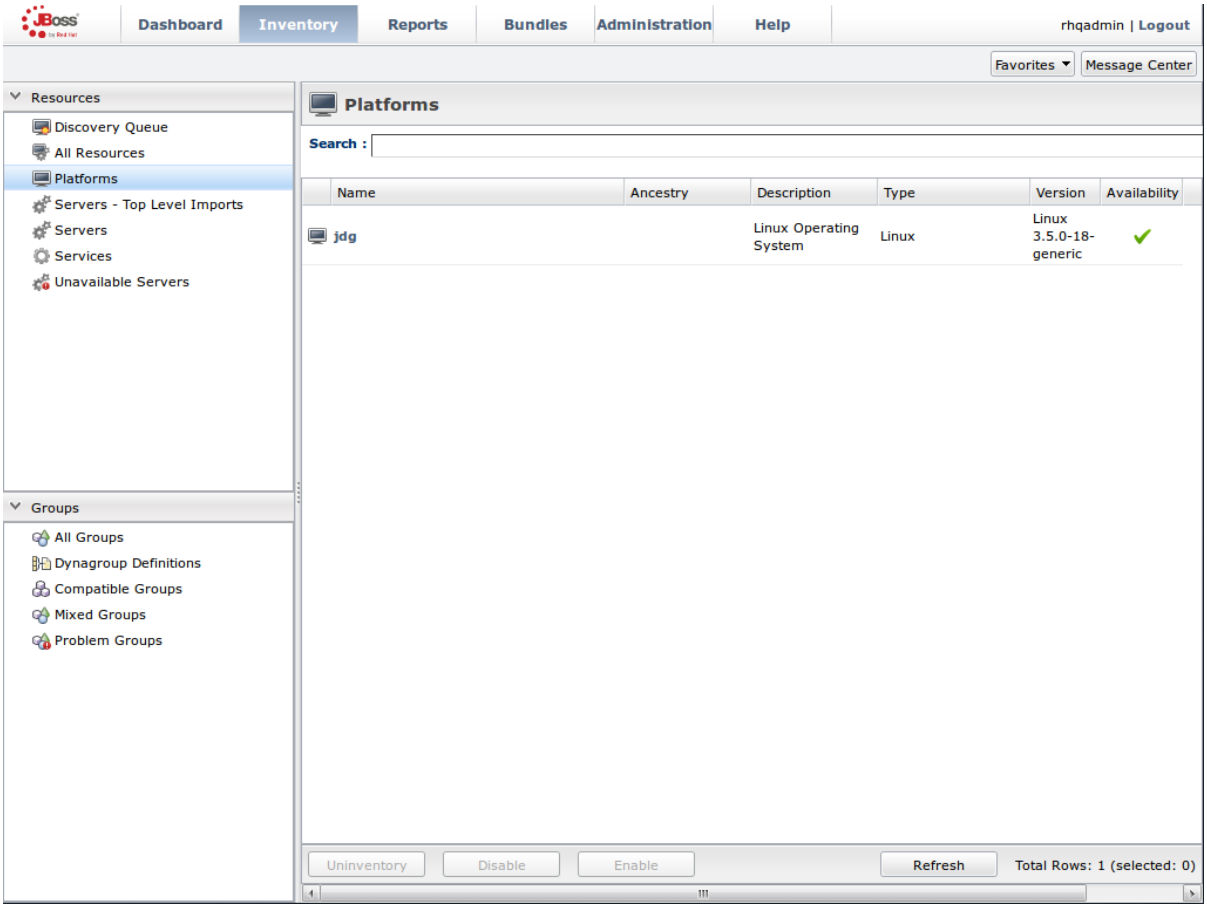


図23.5 jdg プラットフォームを開いてサーバーのリストを表示。

3. **JMX サーバーのインポート**

   a. **Inventory** タブから、**Child Resources** を選択します。

   b. 画面の下部で **Import** ボタンをクリックし、リストから **JMX Server** オプションを選択します。

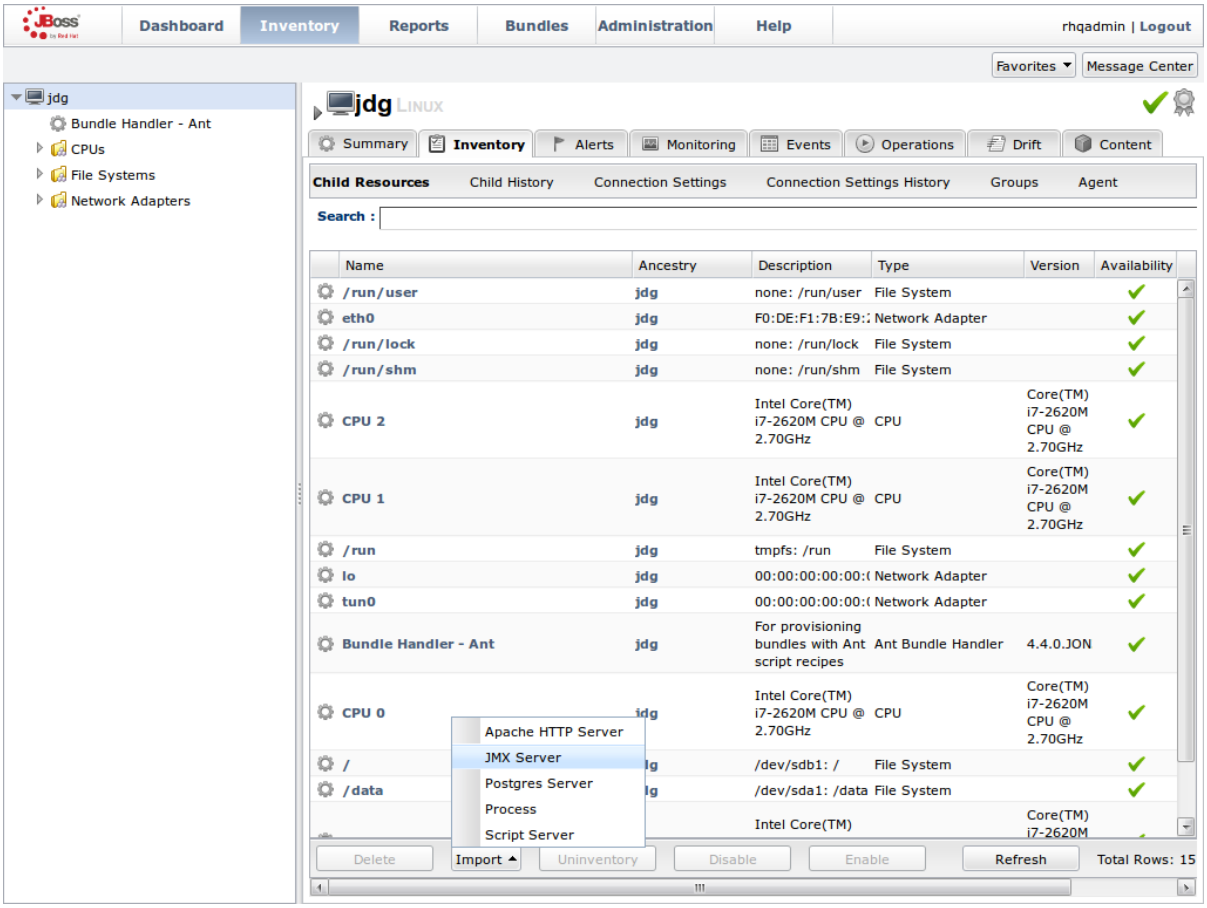


図23.6 JMX サーバーのインポート

4. **JDK 接続設定を有効にします。**

    a. **Resource Import Wizard** ウィンドウで、**Connection Settings Template** オプションのリストから **JDK 5** を指定します。

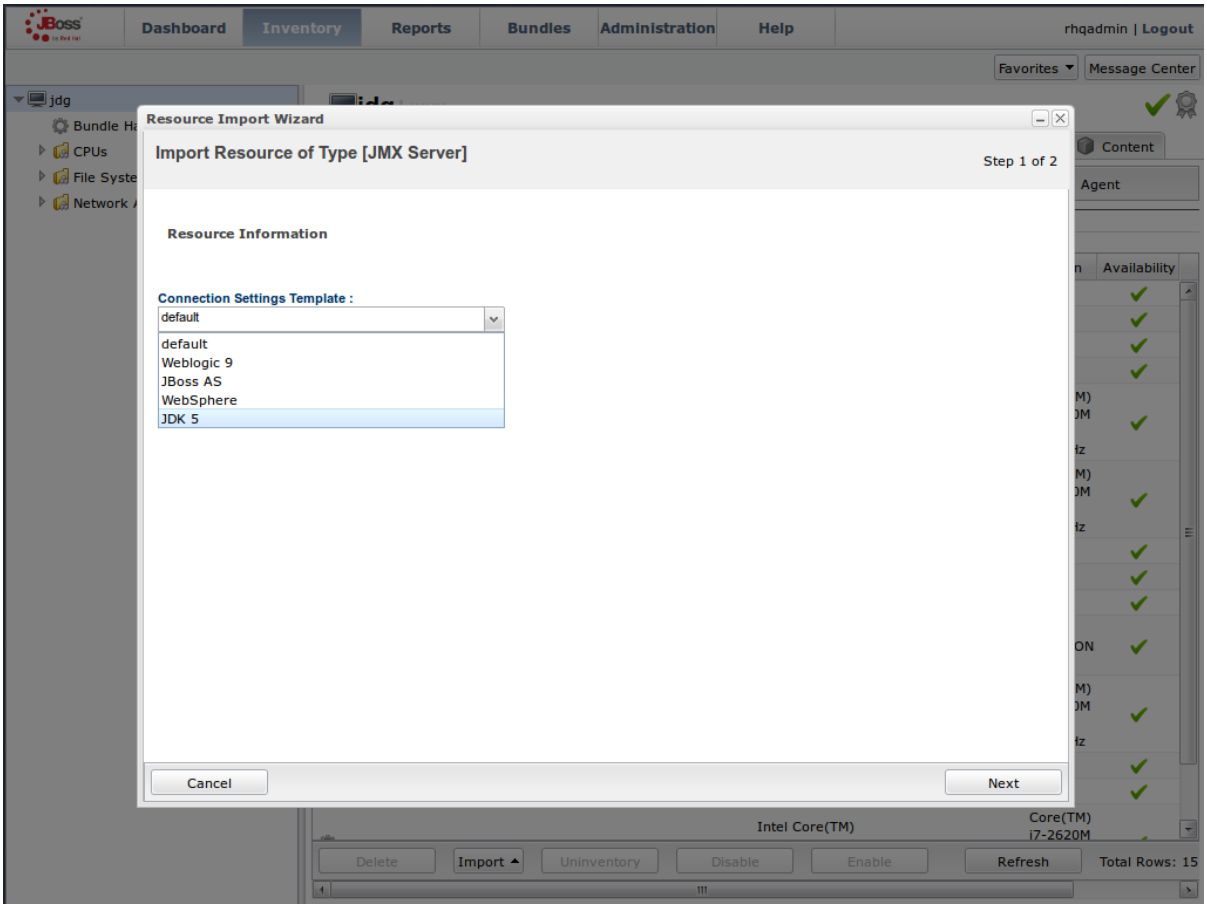


図23.7 JDK 5 テンプレートの選択

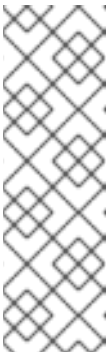
5. **コネクターアドレスを変更します。**

   a. **Deployment Options** メニューで、指定された **Connector Address** の修正を Infinispan ライブラリーを含むプロセスのホスト名と JMX ポートを使って行います。

   b. 監視する新規 JBoss Data Grid インスタンスの JMX コネクターのアドレスを入力します。以下に例を示します。

      コネクターアドレス:

      ```
      service:jmx:rmi://127.0.0.1/jndi/rmi://127.0.0.1:7997/jmxrmi
      ```

      注記

      コネクターアドレスは、新規インスタンスに割り当てられたホストと JMX ポートによって異なります。この場合、インスタンスには起動時に以下のシステムプロパティーが必要です。

      ```
      -Dcom.sun.management.jmxremote.port=7997 -
      Dcom.sun.management.jmxremote.ssl=false -
      Dcom.sun.management.jmxremote.authenticate=false
      ```

   c. 必要な場合は、**Principal** および **Credentials** 情報を指定します。

   d. **Finish** をクリックします。

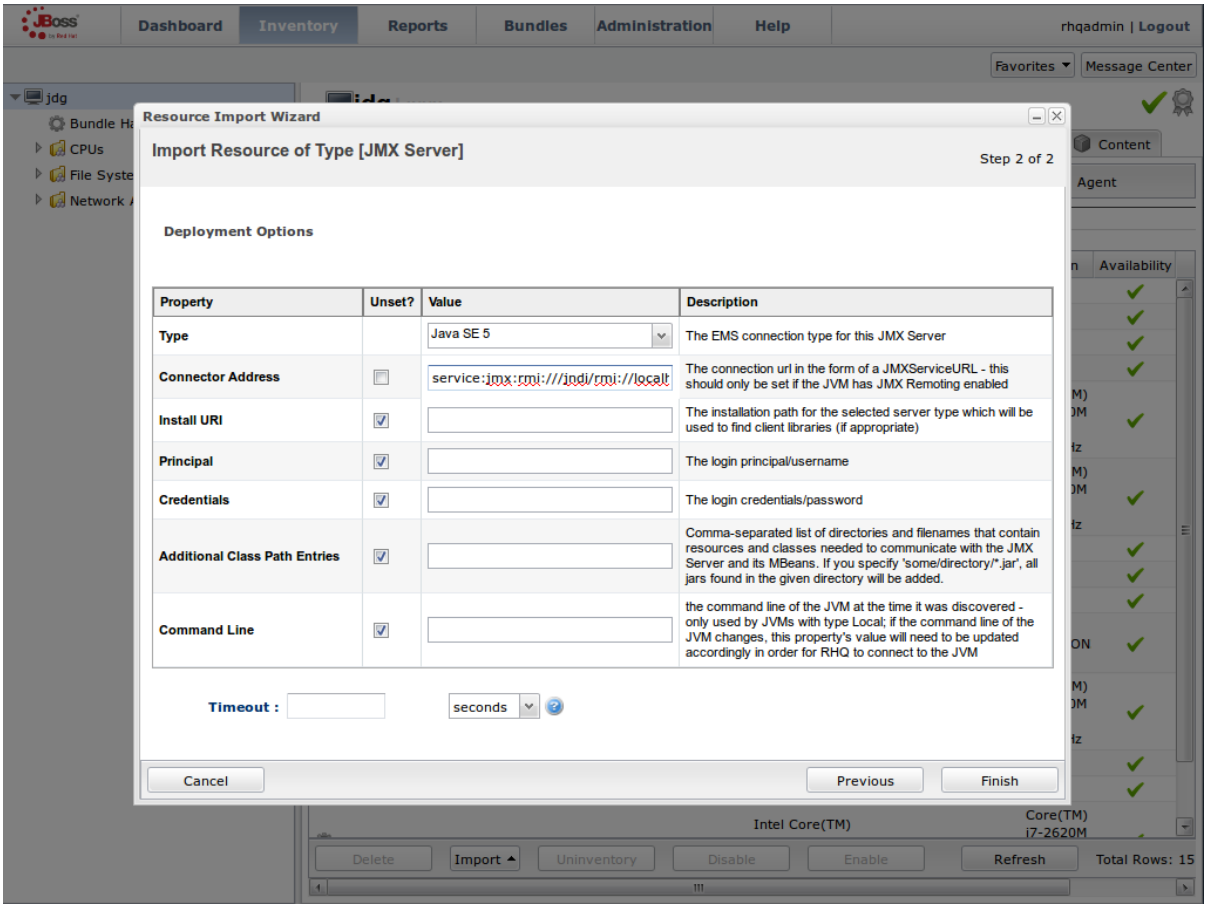


図23.8 Deployment Options 画面での値の修正

6. **キャッシュ統計および操作を表示します。**

   a. **Refresh** をクリックして、サーバーの一覧を更新します。

   b. 画面の左側のパネルにある **JMX Servers** ツリーには、**Infinispan Cache Managers** ノードが含まれ、これには利用可能なキャッシュマネージャーが含まれます。このキャッシュマネージャーには利用可能なキャッシュが含まれます。

   c. メトリックスを表示するために利用可能なキャッシュからキャッシュを選択します。

   d. **Monitoring** タブを選択します。

   e. **Tables** ビューは、統計およびメトリックスを表示します。

   f. **Operations** タブは、サービスで実行できるさまざまな操作へのアクセスを提供します。

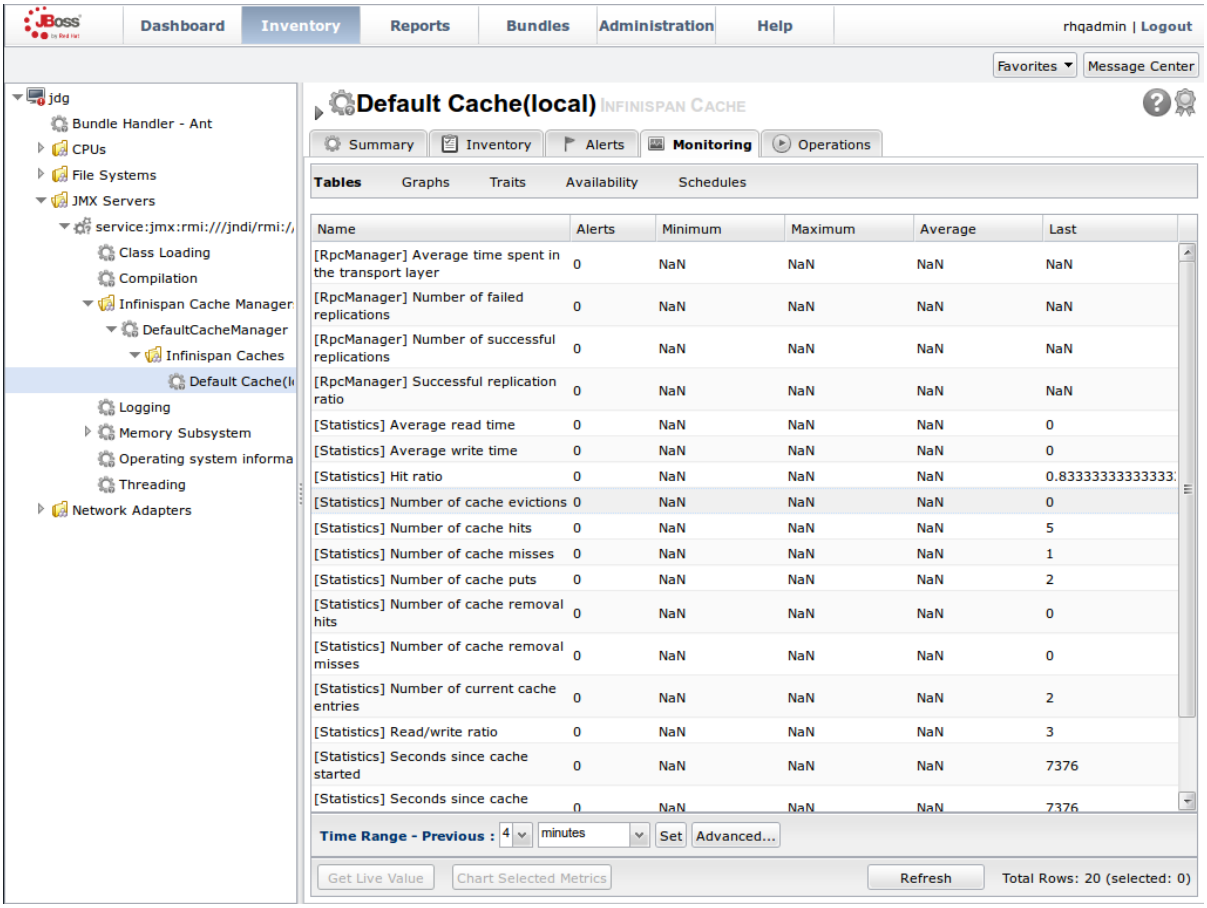


図23.9 JBoss Operations Network コンソールで利用可能な JMX 経由でリレーされるメトリックスと操作データ。

バグを報告する

23.7.2.3. JBoss Enterprise Application Platform にデプロイされたアプリケーションのライブラリーモードでの監視

23.7.2.3.1. スタンドアロンモードでデプロイされたアプリケーションの監視

スタンドアロンモードを使って JBoss Enterprise Application Platform でデプロイされたアプリケーションを監視するには以下の手順を使用します。

手順23.6 スタンドアロンモードでデプロイされたアプリケーションの監視

1. **JBoss Enterprise Application Platform インスタンスを起動します。**
   JBoss Enterprise Application Platform インスタンスを以下のように起動します。

   a. 以下のコマンドをコマンドラインに入力するか、スタンドアロン設定ファイル (`/bin/standalone.conf`) を個別に変更します。

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

   b. JBoss Enterprise Application Platform インスタンスをスタンドアロンモードで以下のように起動します。

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

2. **Red Hat JBoss Data Grid アプリケーションをデプロイします。**
   globalJmxStatistics および **jmxStatistics** を有効にした JJBoss Data Grid ライブラリーモードアプリケーションが含まれる WAR ファイルをデプロイします。

3. **JBoss Operations Network (JON) 検出を実行します。**
   JBoss Operations Network (JON) エージェントで **discovery --full** コマンドを実行します。

4. **アプリケーションサーバープロセスを見つけます。**
   JBoss Operations Network (JON) web インターフェースに、
   JBoss Enterprise Application Platform プロセスが JMX サーバーとしてリストされます。

5. **プロセスをインベントリーにインポートします。**
   プロセスを JBoss Operations Network (JON) インベントリーにインポートします。

6. **オプション: 検出を再度実行します。**
   必要な場合は、**discovery --full** コマンドを再び実行し、新規リソースを検出します。

結果:

JBoss Data Grid ライブラリーモードアプリケーションが JBoss Enterprise Application Platform のスタンドアロンモードでデプロイされ、JBoss Operations Network (JON) を使用して監視することができるようになります。

バグを報告する

23.7.2.3.2. ドメインモードでデプロイされたアプリケーションの監視

ドメインモードを使って JBoss Enterprise Application Platform 6 でデプロイされたアプリケーションを監視するには以下の手順を使用します。

手順23.7 ドメインモードでデプロイされたアプリケーションの監視

1. **ホスト設定の編集**
   domain/configuration/host.xml ファイルを編集し、**server** 要素を以下の設定に置き換えます。

```
<servers>
 <server name="server-one" group="main-server-group">
  <jvm name="default">
   <jvm-options>
    <option value="-Dorg.rhq.resourceKey=EAP1"/>
   </jvm-options>
  </jvm>
 </server>
 <server name="server-two" group="main-server-group" auto-
start="true">
  <socket-bindings port-offset="150"/>
  <jvm name="default">
   <jvm-options>
    <option value="-Dorg.rhq.resourceKey=EAP2"/>
   </jvm-options>
  </jvm>
 </server>
</servers>
```

2. **JBoss Enterprise Application Platform の起動**
   ドメインモードによる JBoss Enterprise Application Platform 6 の起動:

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

3. **Red Hat JBoss Data Grid アプリケーションのデプロイ**
   `globalJmxStatistics` および **`jmxStatistics`** を有効にした JJBoss Data Grid ライブラリーモードアプリケーションが含まれる WAR ファイルをデプロイします。

4. **JBoss Operations Network (JON) での検出の実行**
   必要な場合は、新規リソースを検出するために JBoss Operations Network (JON) エージェントについて **`discovery --full`** コマンドを実行します。

結果

JBoss Data Grid ライブラリーモードアプリケーションが、JBoss Enterprise Application Platform のドメインモードでデプロイされ、JBoss Operations Network (JON) を使用して監視することができるようになります。

バグを報告する

23.8. JBOSS OPERATIONS NETWORK のプラグインクイックスタート

単一の JBoss Operation Network エージェントを使ったテストおよびデモの目的で、プラグインをサーバーにアップロードし、エージェントのコマンドラインに「**plugins update**」と入力して、サーバーから最新のプラグインを強制的に読み出します。

バグを報告する

23.9. 他の管理ツールと操作

Red Hat JBoss Data Grid インスタンスの管理には、関連する統計情報を大量に公開する必要があります。管理者は統計情報より、 JBoss Data Grid の各ノードの状態を明確に把握することができます。1 つのインストールが、何十または何百もの JBoss Data Grid ノードによって構成されることもあるため、明確で簡潔に情報を提供することが重要になります。JBoss Operations Network はランタイムを可視化するツールの 1 つです。JMX が有効である場合、**JConsole** などの他のツールも使用できます。

バグを報告する

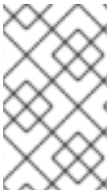
23.9.1. URL 経由のデータアクセス

REST インターフェースで設定されたキャッシュは、RESTful HTTP アクセスを使用して Red Hat JBoss Data Grid へアクセスできます。

RESTful サービスは HTTP クライアントライブラリーのみを必要とするため、密結合されたクライアントライブラリーやバインディングは必要ありません。REST インターフェースを使用したデータの取得方法の詳細については、JBoss Data Grid の 『Developer Guide』 を参照してください。

HTTP **`put()`** および **`post()`** メソッドは、キャッシュにデータを格納します。使用される URL より使用されるキャッシュ名とキーを判断することができます。データはキャッシュに格納される値で、要求の本文に置かれます。

これらのメソッドに対して Content-Type ヘッダーを設定する必要があります。データの読み出しには **GET** および **HEAD** メソッドが使用され、他のヘッダーはキャッシュの設定と挙動を制御します。



注記

競合するサーバーモジュールがデータグリッドとやりとりすることはできません。JBoss Data Grid にアクセスするには、互換性のあるインターフェースでキャッシュを設定する必要があります。

バグを報告する

23.9.2. Map メソッドの制限

size()、 **values()**、 **keySet()**、 **entrySet()** などの特定の Map メソッドは不安定であるため、Red Hat JBoss Data Grid で一定の制限付きで使用することが可能です。これらのメソッドはロック (グローバルまたはローカル) を取得せず、同時編集、追加、および削除はこれらの呼び出しでは考慮されません。

一覧表示されるメソッドはパフォーマンスに大きく影響します。そのため、情報収集やデバッグの目的でのみこれらのメソッドを使用することが推奨されます。

パフォーマンスの問題

JBoss Data Grid 7.0 では、マップメソッド **size()**、**values()**、**keySet()**、および **entrySet()** には、デフォルトでキャッシュローダーのエントリーが含まれます。使用されるキャッシュローダーはこれらのコマンドのパフォーマンスに直接影響を与えます。たとえば、データベースを使用している場合、これらのメソッドはデータが格納されるテーブルの完全なスキャンを実行し、処理が遅くなることがあります。キャッシュローダーからエントリーをロードしないようにし、パフォーマンスの低下を避けるには、必要なメソッドを実行する前に ***Cache.getAdvancedCache().withFlags(Flag.SKIP_CACHE_LOAD)*** を使用します。

size() サイズの概要 (組み込みキャッシュ)

JBoss Data Grid 7.0 では、**Cache.size()** メソッドは、クラスター全体で、このキャッシュとキャッシュローダーの両方にあるすべての要素の数を提供します。ローダーまたはリモートエントリーを使用している場合、メモリー関連の問題の発生を防げるようにエントリーのサブセットのみが指定時にメモリーに保持されます。すべてのエントリーのロードする場合、その速度が遅くなる場合があります。

この操作モードでは、**size()** メソッドで返される結果は、ローカルノードにあるエントリー数を返すよう強制実行する ***org.infinispan.context.Flag#CACHE_MODE_LOCAL*** フラグと、パッシベートされたエントリーを無視する ***org.infinispan.context.Flag#SKIP_CACHE_LOAD*** フラグによって影響を受けます。これらのフラグのいずれかを使用すると、クラスター全体ですべての要素の数を返さない代わりにこのメソッドのパフォーマンスを上げることができます。

size() メソッドの概要 (リモートキャッシュ)

JBoss Data Grid 7.0 では、Hot Rod プロトコルには専用の **SIZE** 操作が含まれ、クライアントはこの操作を使用してすべてのエントリーのサイズを計算します。

バグを報告する

パート XI. RED HAT JBOSS DATA GRID の WEB 管理

第24章 RED HAT JBOSS DATA GRID 管理コンソール

24.1. JBOSS DATA GRID 管理コンソールについて

Red Hat JBoss Data Grid 管理コンソールを使用する管理者は、キャッシュおよび JBoss Data Grid クラスターを監視できます。

バグを報告する

24.2. RED HAT JBOSS DATA GRID 管理コンソールの前提条件

Red Hat JBoss Data Grid 管理コンソールを実行するには、以下が必要です。

- Java 8
- JBoss Data Grid サーバーがインストール済みで、ドメインモードで実行できる。

バグを報告する

24.3. RED HAT JBOSS DATA GRID 管理コンソースの使用開始

24.3.1. Red Hat JBoss Data Grid 管理コンソールの使用開始

JBoss Data Grid 管理コンソールの使用を開始するには、JBoss Data Grid サーバーのバージョンをダウンロードし、これをインストールし、管理ユーザーを追加してから Web インターフェースにログインします。

バグを報告する

24.3.2. JBoss Data Grid Server のダウンロードおよびインストール

- Red Hat カスタマーポータルから Red Hat JBoss Data Grid サーバーのバージョンをダウンロードします。
- ダウンロードしたパッケージをシステムの優先ディレクトリーに解凍して JBoss Data Grid をインストールします。

注記

ダウンロードおよびインストールの詳細は、『Red Hat JBoss Data Grid Getting Started Guide』 の 『Download and Install JBoss Data Grid』 セクションを参照してください。

バグを報告する

24.3.3. 管理ユーザーの追加

JBoss Data Grid 管理コンソールを使用するには、新規管理ユーザーを作成する必要があります。新規ユーザーを追加するには、JBoss Data Grid Server インストールの bin フォルダー内で add-user.sh ユーティリティースクリプトを実行し、必要な情報を入力します。

以下の手順は、新規管理ユーザーを追加する方法を説明しています。

手順24.1 管理ユーザーの追加

1. 以下のように bin フォルダー内で add-user スクリプトを実行します。

```
./add-user.sh
```

2. 追加するユーザーのタイプのオプションを選択します。管理ユーザーの場合は、オプション ***a*** を選択します。
3. 一覧表示される推奨に基づいてユーザー名とパスワードを設定します。
4. ユーザーを追加する必要のあるグループの名前を入力します。グループがない場合は空白にします。

注記

ダウンロードおよびインストールの詳細は、『Red Hat JBoss Data Grid Getting Started Guide』 の 『Download and Install JBoss Data Grid』 セクションを参照してください。

5. Apache Spark プロセス接続にユーザーを使用する必要があるかどうかを確認します。

注記

次に進む前に、$JBOSS_HOME が別のインストールに設定されていないことを確認します。この確認をしないと、予期しない結果が出される可能性があります。

結果

管理ユーザーが正常に追加されます。

バグを報告する

24.3.4. JBoss Data Grid Server の起動

JBoss Data Grid Server をドメインモードで起動するには、以下を実行します。

```
./domain.sh
```

JBoss Data Grid Server はドメインモードで起動すると、JBoss Data Grid 管理コンソールにアクセスできます。

バグを報告する

24.3.5. JBoss Data Grid 管理コンソールへのログイン

以下のリンクを web ブラウザーに入力し、JBoss Data Grid 管理コンソールのログインページにアクセスします。

```
http://localhost:9990/console/index.html
```

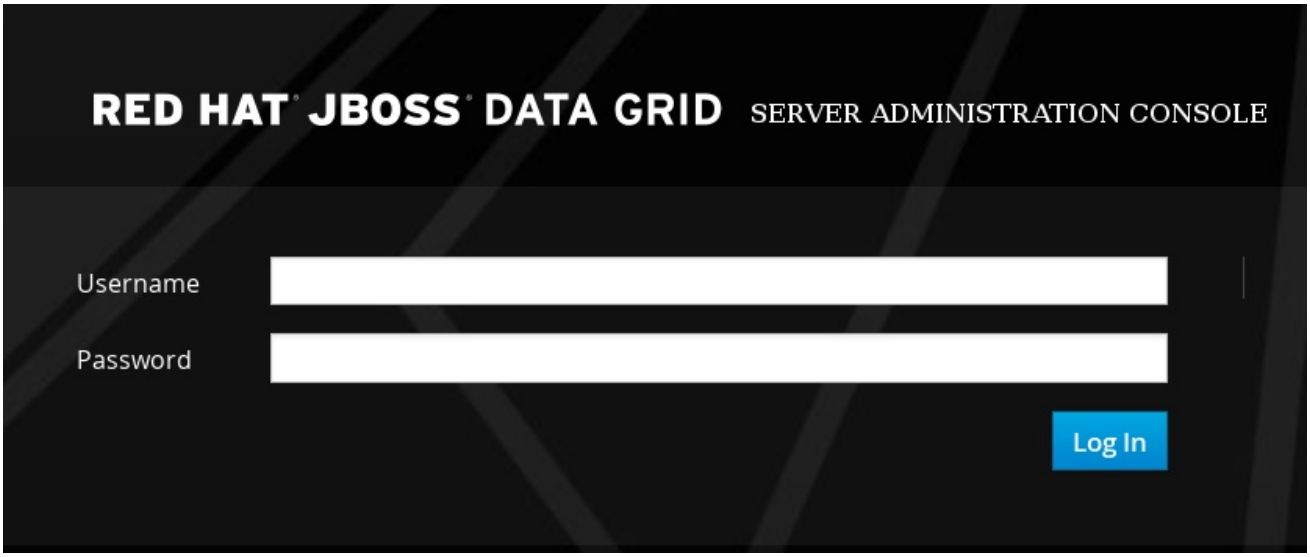


図24.1 JBoss Data Grid 管理コンソールログイン画面

クレデンシャルを入力してログインします。ログイン後にキャッシュコンテナービューが表示されます。

バグを報告する

24.4. ダッシュボードビュー

ダッシュボードビューは 3 つのタブに分割されます。

- キャッシュ
- クラスター
- ステータスイベント

バグを報告する

24.4.1. キャッシュコンテナービュー

キャッシュコンテナーの一覧はログイン後の最初のデフォルトビューです。キャッシュコンテナーは、キャッシュインスタンスを処理するための主なメカニズムであり、キャッシュ自体を使用する際の開始地点として使用されます。

キャッシュセントリックビューには、設定済みキャッシュの一覧が表示されます。これは、キャッシュの表示およびクラスターへの追加、新規のキャッシュ設定の追加および調整、エンドポイントおよびその他の関連する管理タスクの追加および設定を行うために使用されます。

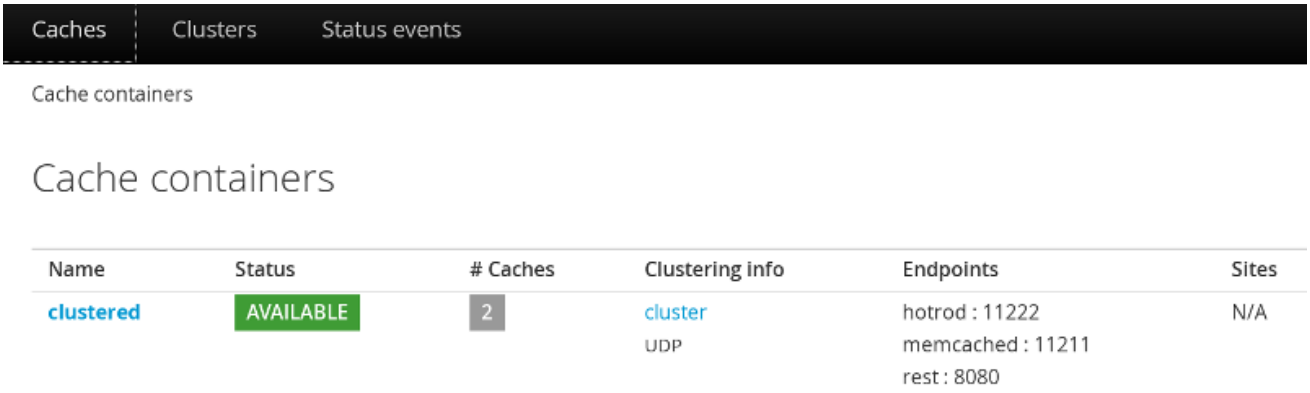


図24.2 キャッシュコンテナービュー

この例では、UDP トランスポートとエンドポイントが接続されたクラスターグループに 2 つのキャッシュがデプロイされた **clustered** という名前のキャッシュコンテナーが 1 つあります。このキャッシュコンテナーにはリモートサイトが設定されていません。

バグを報告する

24.4.2. クラスタービュー

「Cluster」タブには、クラスターの概要が、クラスターのステータス、ホスト数およびノード数と共に表示されます。

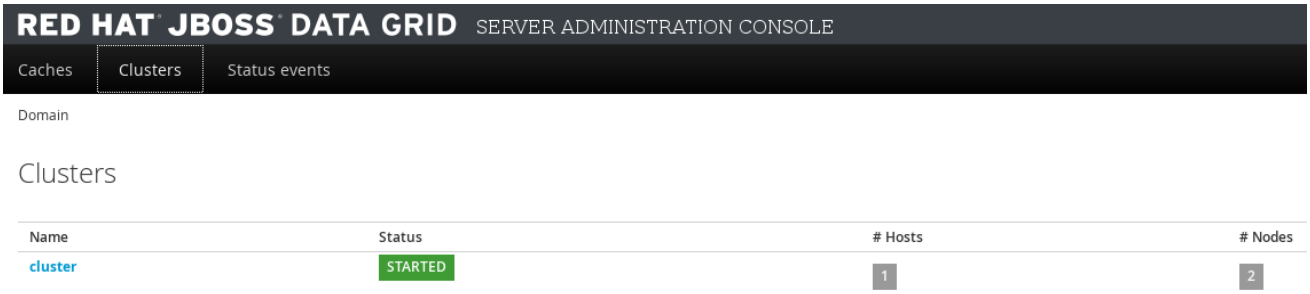


図24.3 クラスタービュー

バグを報告する

24.4.3. ステータスイベントビュー

JBoss Data Grid 管理コンソールは、統合セクションにローカルの再調整、クラスターの開始および停止、クラスター分割、クラスターマージイベントなどのクラスター全体のイベントを表示します。詳細のステータスイベントを表示するには、ダッシュボードから「Status Events」タブに移動します。

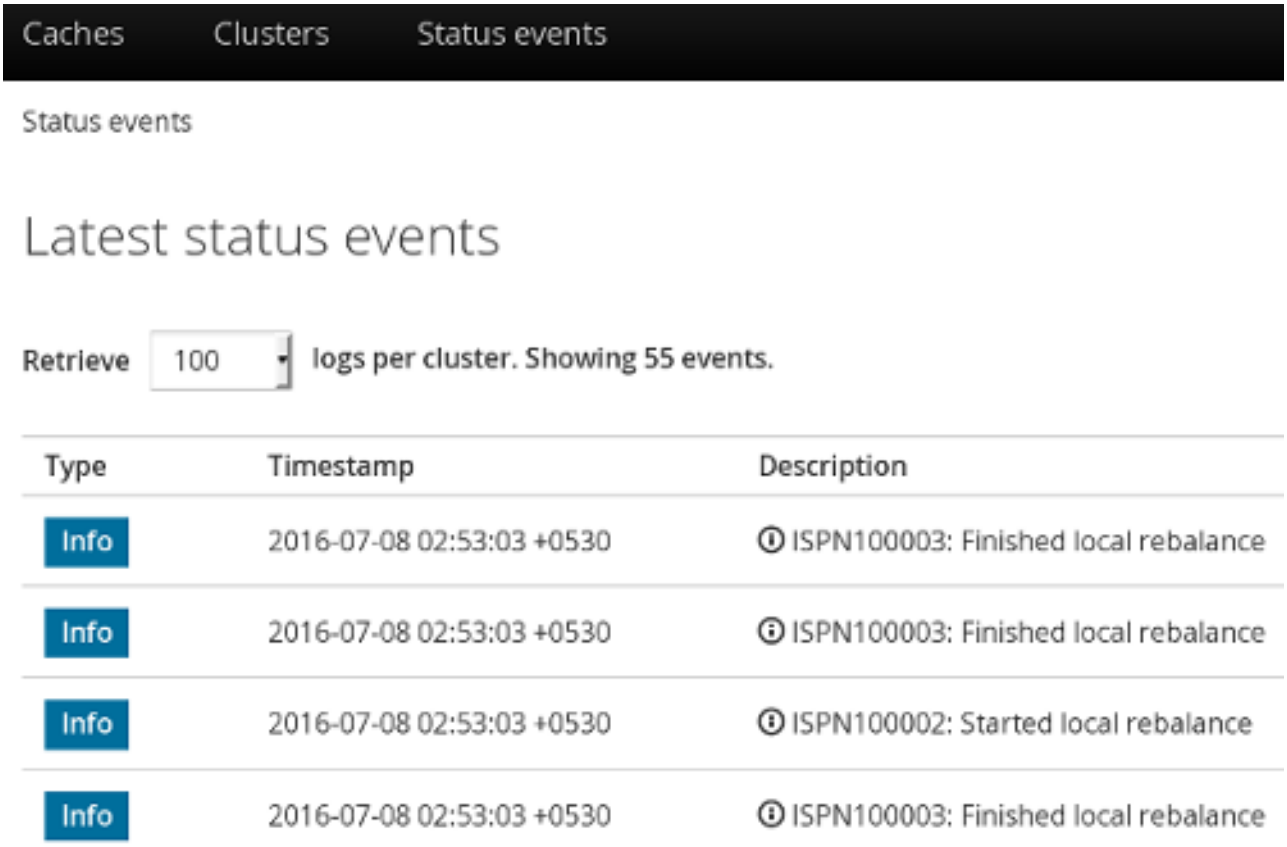


図24.4 ステータスイベントビュー

ステータスイベントは、関連付けられたタイムスタンプとイベントの説明と共に表示されます。

バグを報告する

24.5. キャッシュ管理

24.5.1. 新規キャッシュの追加

新規キャッシュを追加するには、以下の手順を実行します。

手順24.2 新規キャッシュの追加

1. 「**Cache Containers**」ビューで、キャッシュコンテナーの名前をクリックします。

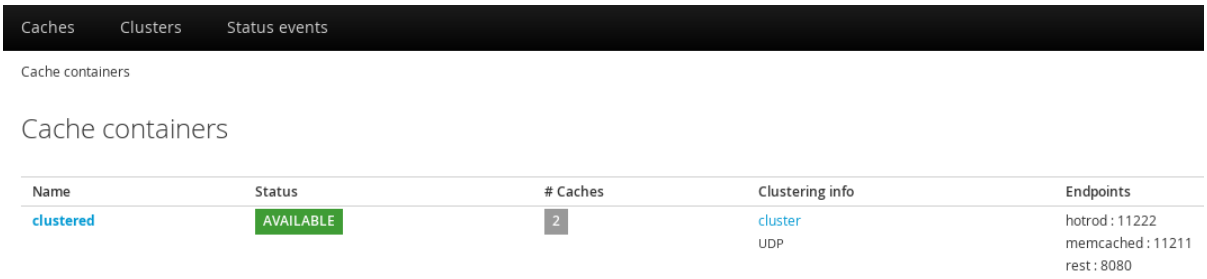


図24.5 「Cache Containers」ビュー

2. すべての設定済みのキャッシュを一覧表示するキャッシュビューが表示されます。「**Add Cache**」をクリックして新規キャッシュを追加し、設定します。新規キャッシュの作成ウィンドウが開かれます。

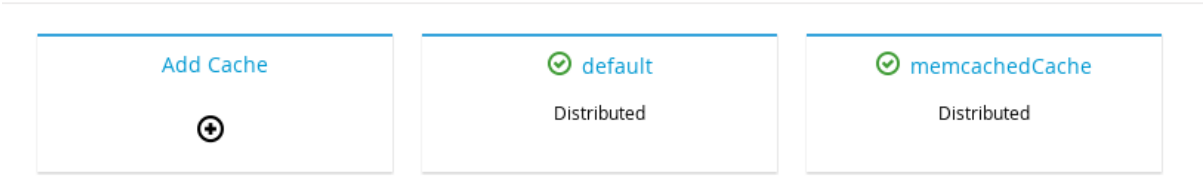


図24.6 キャッシュの追加

3. 新規キャッシュ名を入力し、ドロップダウンメニューからベース設定テンプレートを選択し、「**Next**」をクリックします。

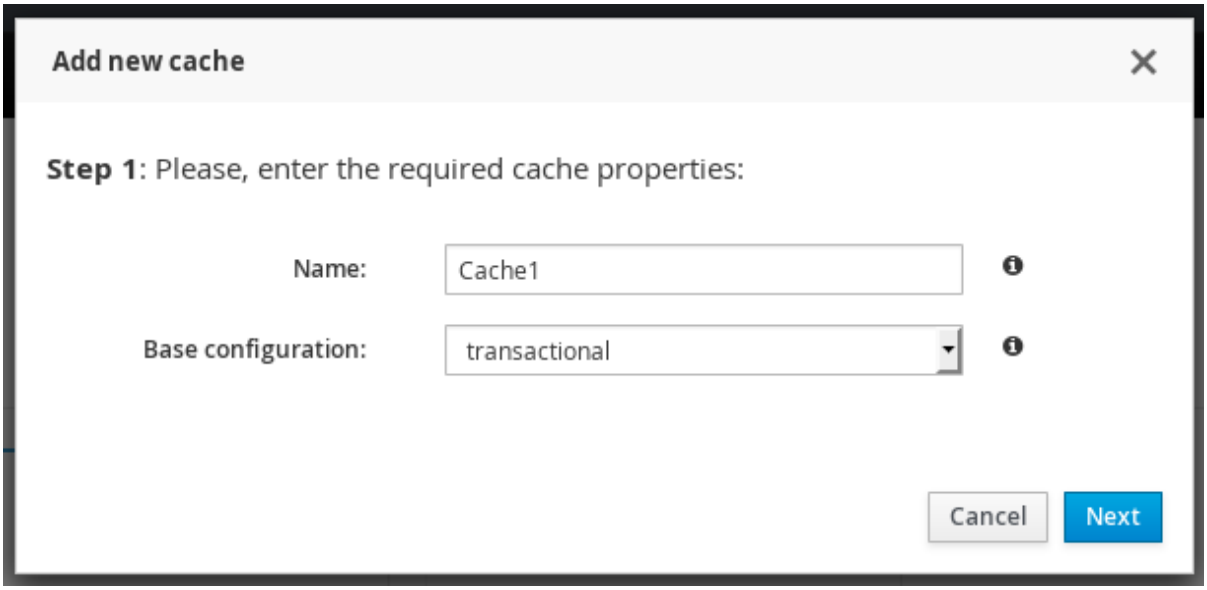


図24.7 キャッシュプロパティー

4. キャッシュ設定画面が表示されます。キャッシュパラメーターを入力し、「**Create**」をクリックします。

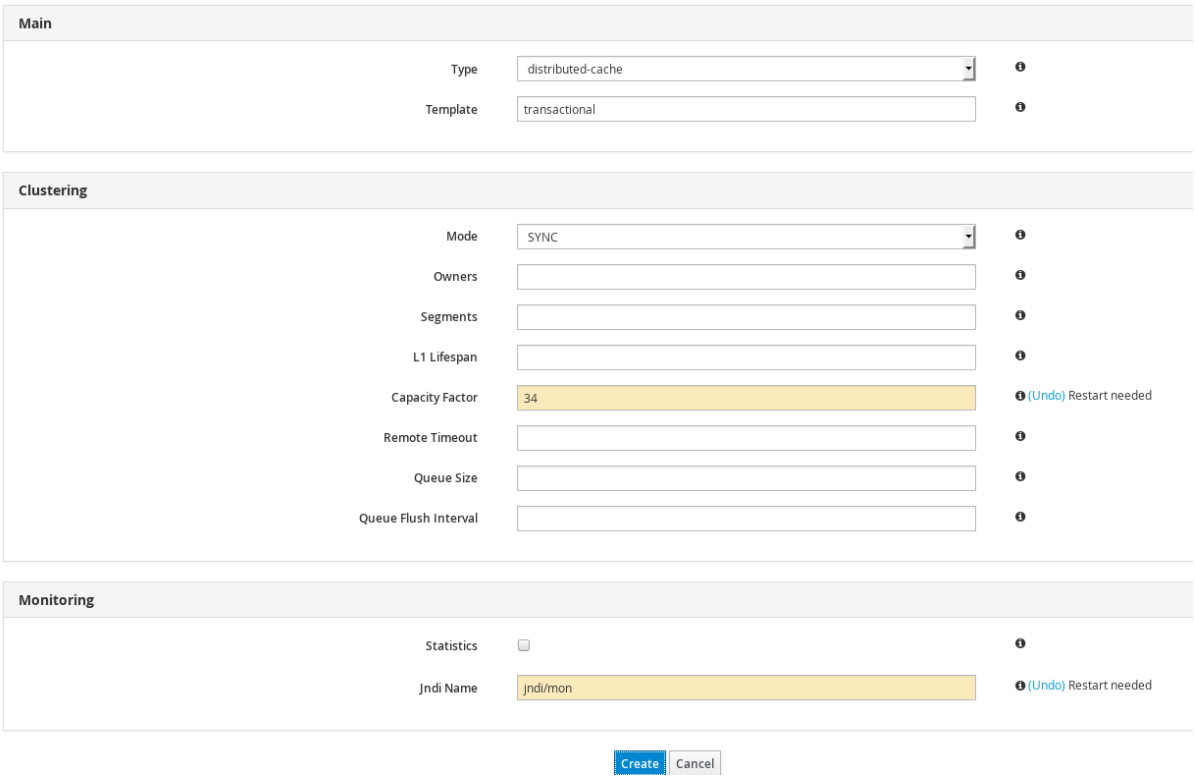


図24.8 キャッシュの設定

5. 確認画面が表示されます。「**Create**」をクリックしてキャッシュを作成します。

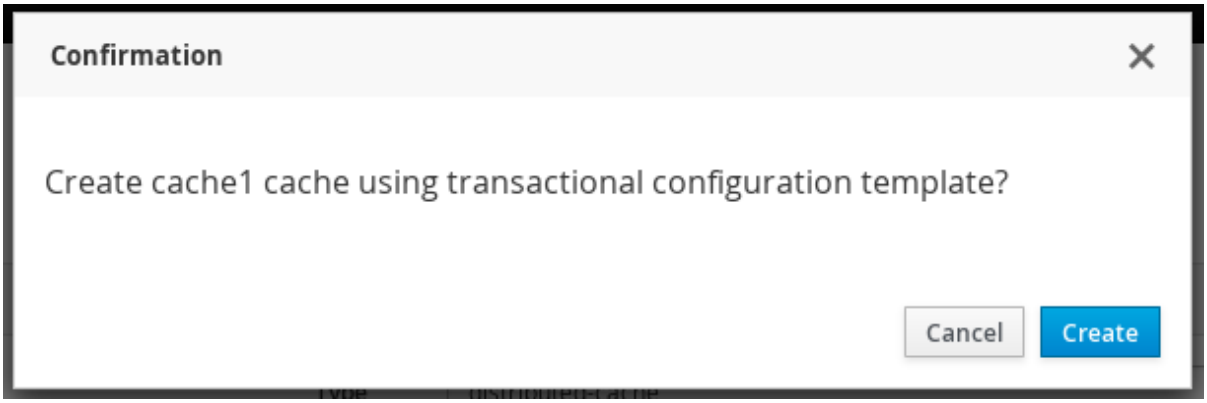


図24.9 キャッシュの確認

結果

新規キャッシュが正常に追加されます。

バグを報告する

24.5.2. キャッシュ設定の編集

JBoss Data Grid 管理コンソールを使用すると、管理者は既存キャッシュの設定を編集することができます。

以下の手順は、キャッシュ設定を編集する手順を簡単に説明しています。

手順24.3 キャッシュ設定の編集

1. JBoss Data Grid 管理コンソールにログインし、キャッシュコンテナーの名前をクリックします。

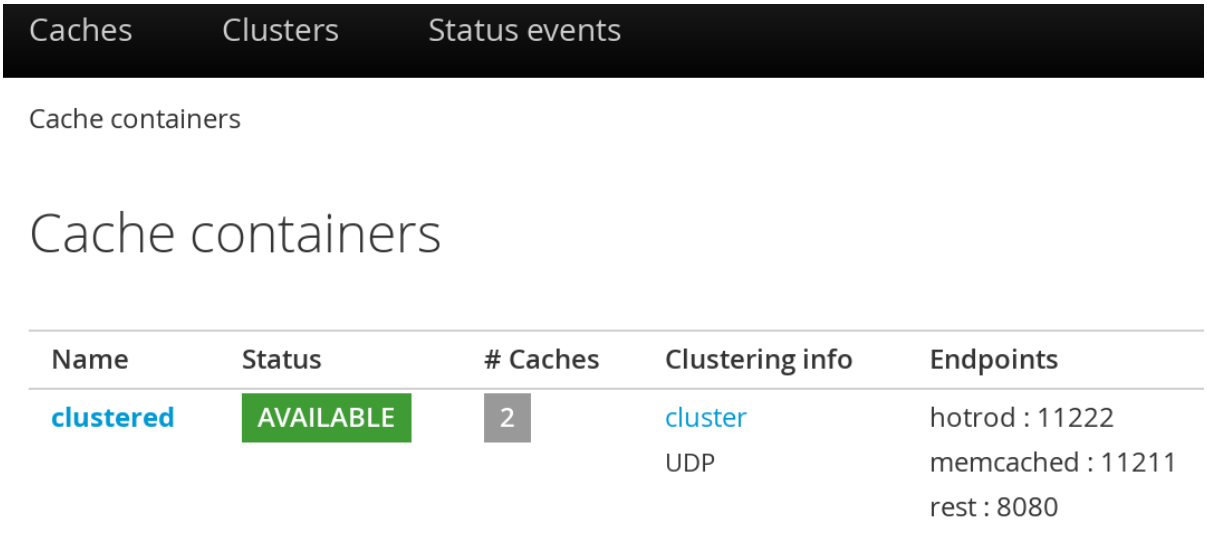


図24.10 キャッシュコンテナー

2. キャッシュビューで、キャッシュ名をクリックします。

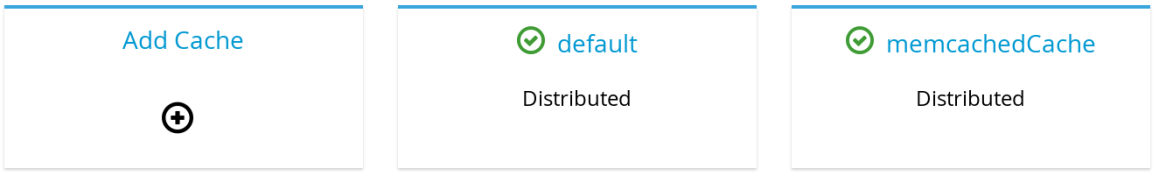


図24.11 キャッシュビュー

3. キャッシュの統計およびプロパティーのページが表示されます。右側にある「**Configuration**」タブをクリックします。

図24.12 キャッシュ設定ボタン

4. キャッシュの設定を編集するためのインターフェースが開かれます。編集可能なキャッシュプロパティーは、右側のキャッシュプロパティーのメニューで確認できます。

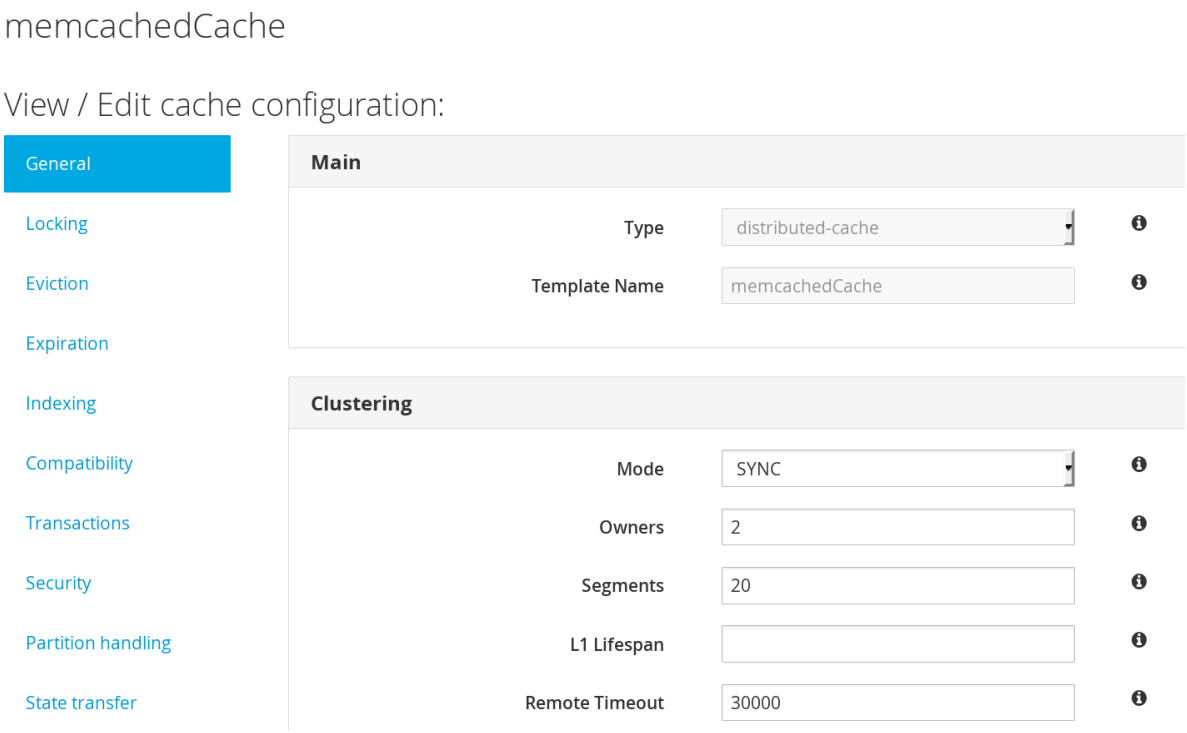


図24.13 キャッシュ設定の編集インターフェース

5. キャッシュプロパティーメニューから編集する設定プロパティーを選択します。キャッシュ設定パラメーターの説明を確認するには、情報アイコン上にカーソルを移動します。パラメーターの説明はツールチップの形式で表示されます。

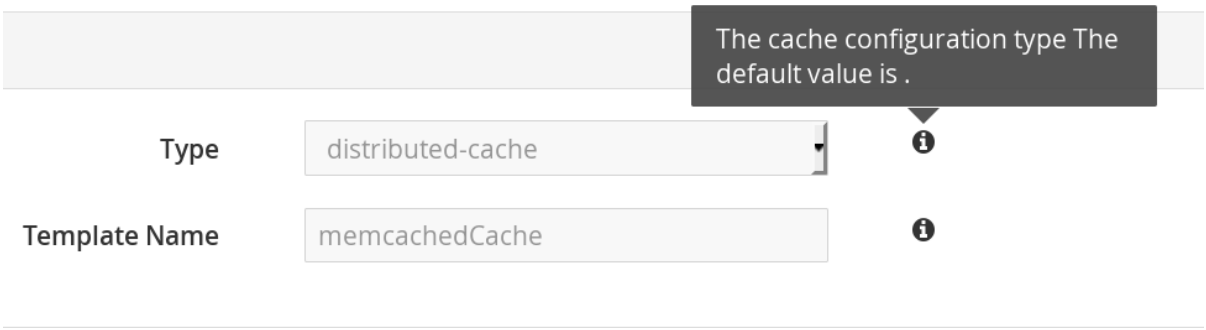


図24.14 キャッシュ設定パラメーター

6. 「**General**」プロパティーがデフォルトで選択されているとします。指定されたパラメーターの入力フィールドに必要な値を入力します。スクロールダウンして「**Apply changes**」をクリックします。

7. 確認のためのダイアログボックスが表示されます。「**Update**」をクリックします。

図24.15

8. 再起動するためのダイアログボックスが表示されます。「**`Restart Now`**」をクリックして変更を適用します。

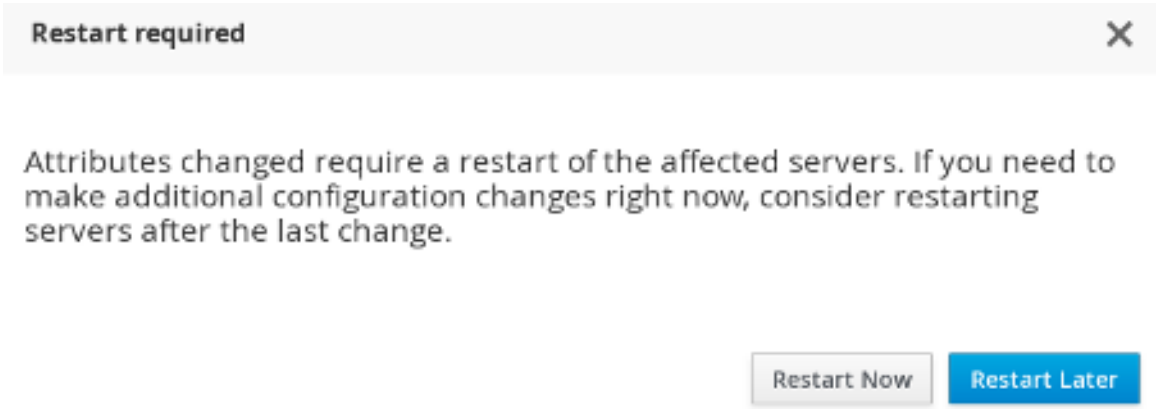
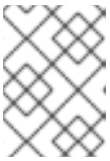


図24.16 再起動ダイアログボックス



注記

キャッシュプロパティーの編集を継続するには、「**`Restart Later`**」をクリックします。

バグを報告する

24.5.3. キャッシュ統計およびプロパティービュー

JBoss Data Grid 管理コンソールにより、管理者は読み取りの平均時間、書き込みの平均時間、エントリーの合計数、読み取りの合計数、失敗した読み取りの合計数、書き込みの合計数を含む、すべてのキャッシュ統計を表示できます

キャッシュ統計を表示するには、以下の手順を実行します。

手順24.4 キャッシュ統計の表示

1. 「Cache Container」ビューでキャッシュコンテナーの名前をクリックし、キャッシュの一覧に移動します。
2. キャッシュの一覧からキャッシュの名前をクリックします。オプションで、キャッシュをフィルターするために左側のキャッシュフィルターを使用できます。キャッシュは、キーワード、サブ文字列で、またはタイプおよび特性を選択してフィルターできます。

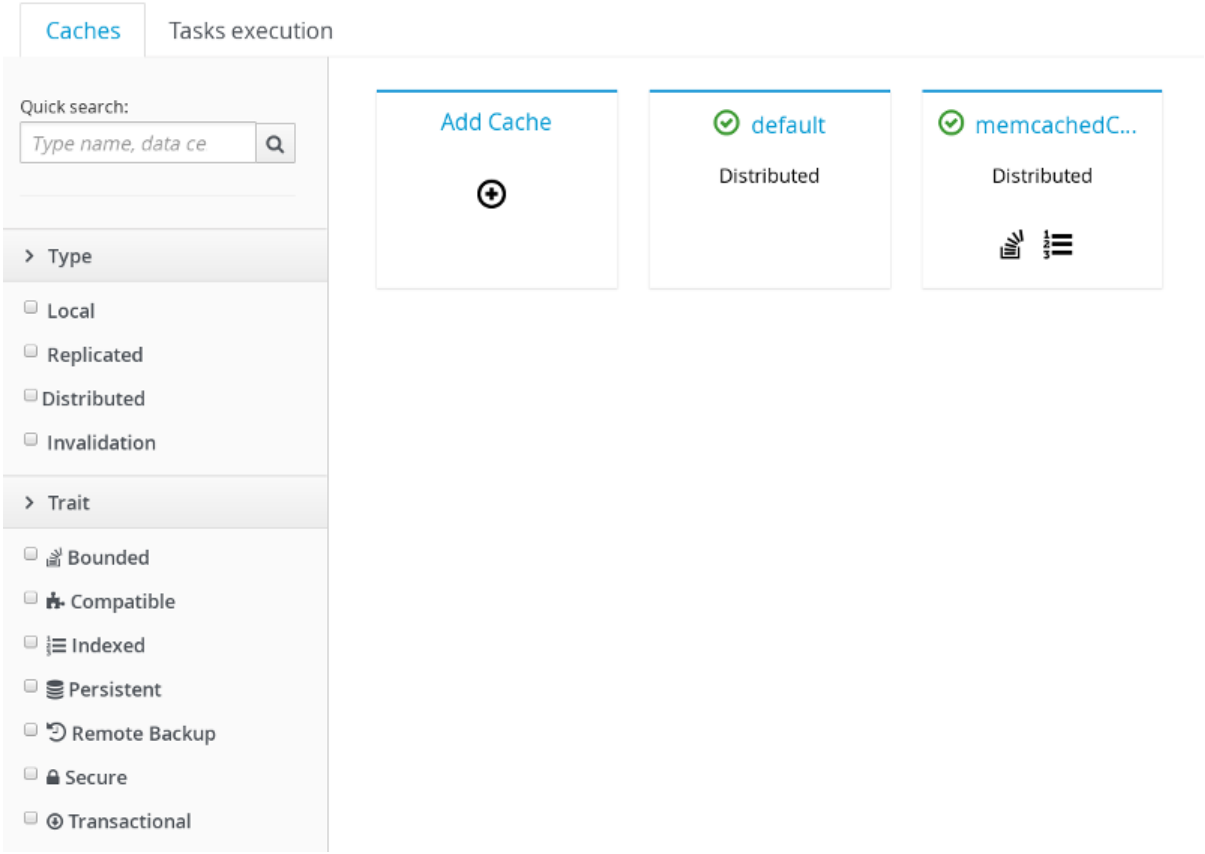


図24.17 キャッシュビュー

3. 以下のページでは、「***Cache content***」、「***Operations performance***」および「***Caching Activity***」の見出しの下に総合的なキャッシュ統計が表示されます。

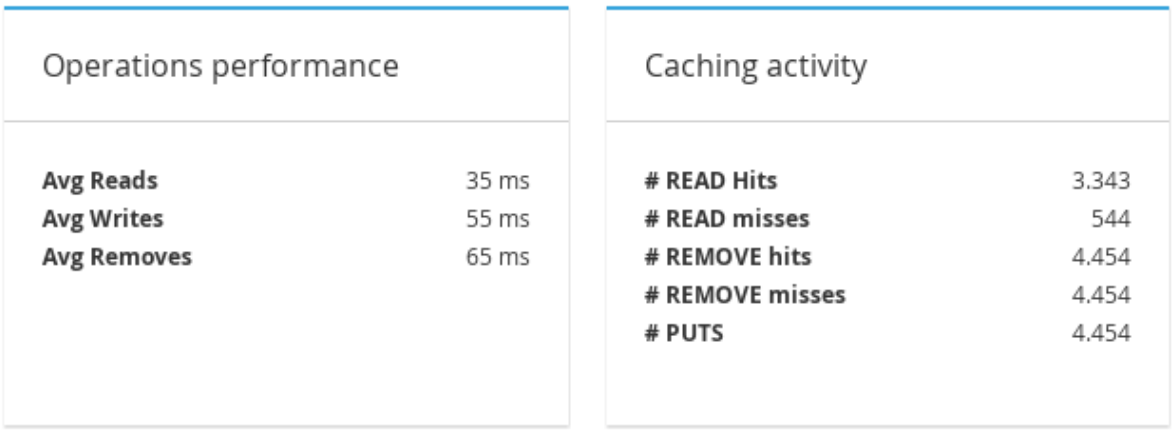


図24.18 キャッシュ統計

4. 追加のキャッシュ統計は、「***Entries Lifecycle***」、「***Cache Loader***」および「***Locking***」の見出しの下に表示されます。

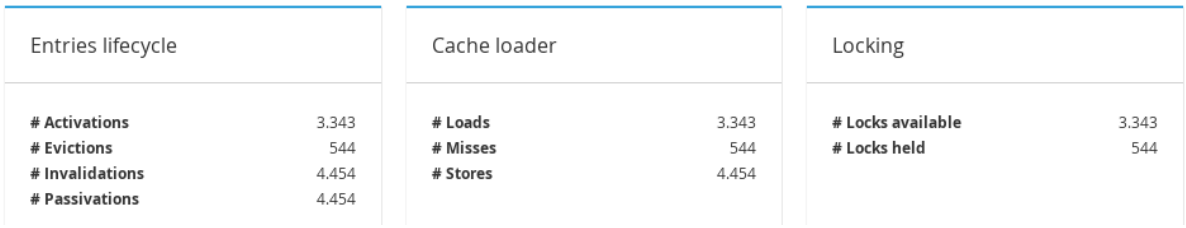


図24.19 キャッシュ統計

5. キャッシュプロパティーを表示するには、右側にある「**Configuration**」をクリックします。

図24.20 設定ボタン

6. キャッシュプロパティーのメニューが右側に表示されます。

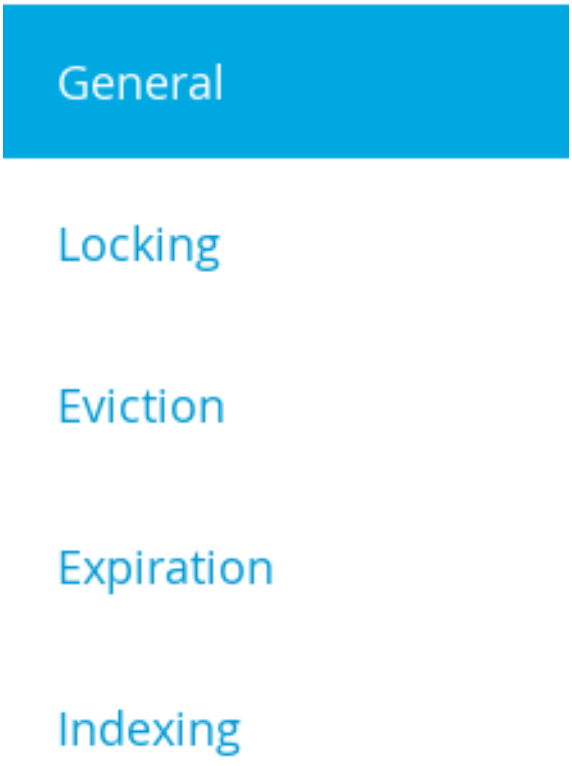


図24.21 キャッシュプロパティーメニュー

キャッシュのあるノードを表示するには、キャッシュ統計ページの「**General Status**」タブの横にある「**Nodes**」タブをクリックします。

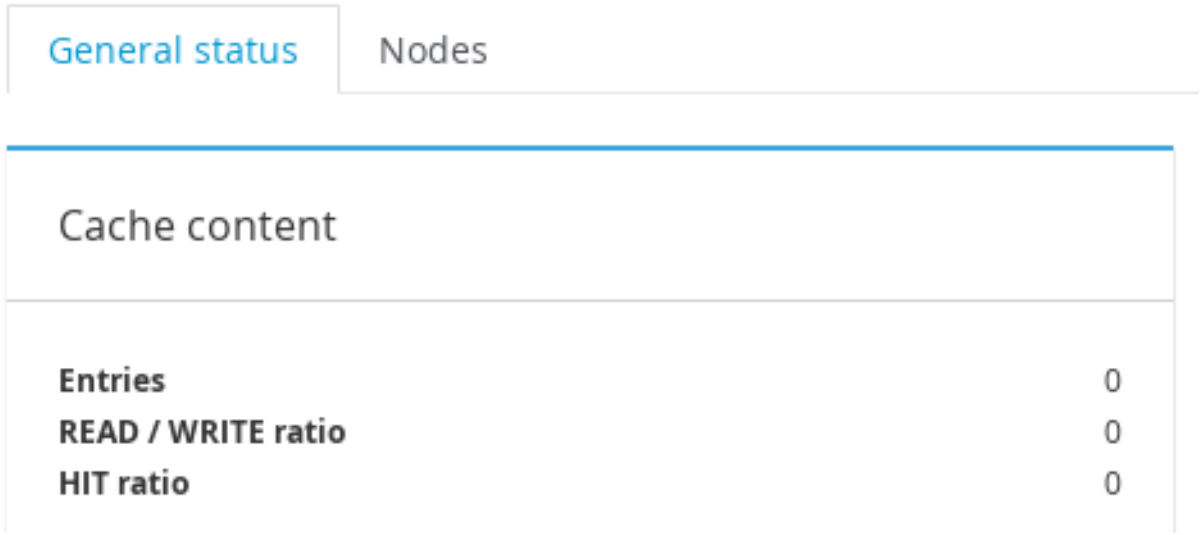


図24.22 「General Status」タブ

ノードの名前は、読み取り/書き込み統計と共に表示されます。

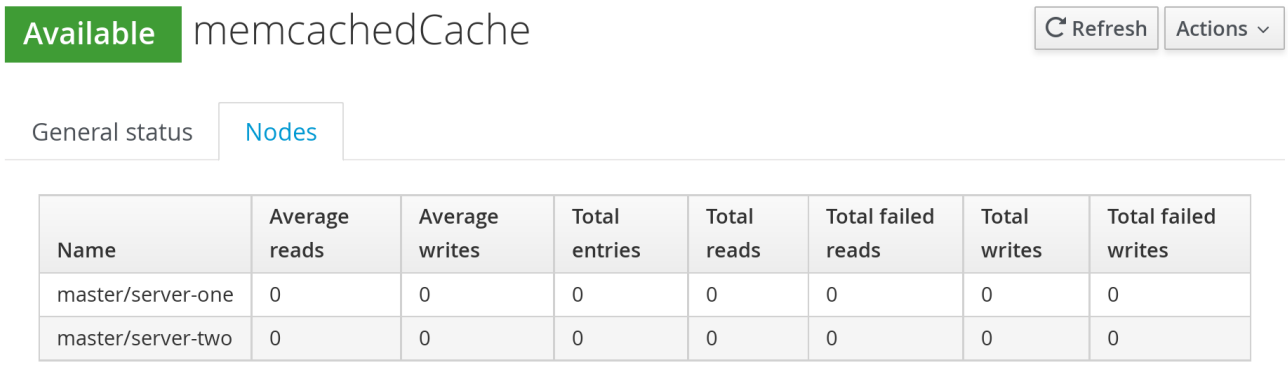


図24.23 キャッシュノードラベル

バグを報告する

24.5.4. キャッシュの有効化および無効化

以下の手順は、キャッシュを無効にする手順を簡単に説明しています。

手順24.5 キャッシュの無効化

1. 「Cache Container 」ビューでキャッシュコンテナーの名前をクリックし、キャッシュビューに移動します。無効にするキャッシュの名前をクリックします。

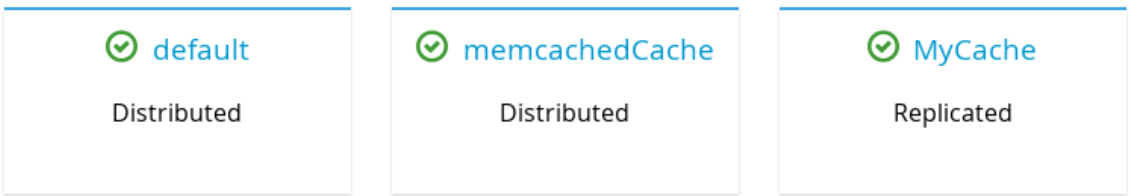


図24.24 キャッシュビュー

2. キャッシュ統計が表示されます。インターフェースの右側にある「 **Actions**」タグをクリックしてから「**`Disable`**」をクリックします。

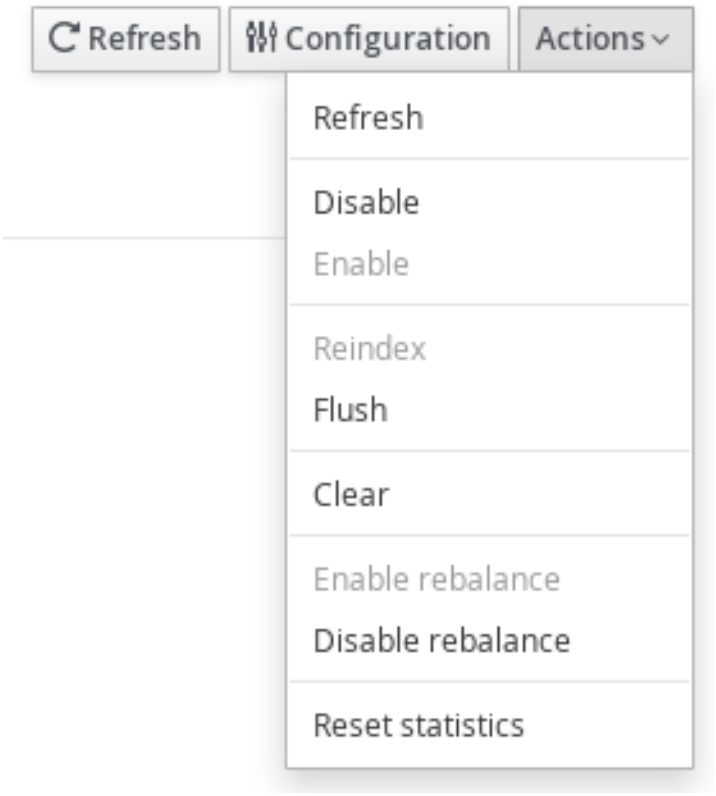


図24.25 キャッシュビューの無効化

3. 確認ダイアログボックスが表示されます。「**`Disable`**」をクリックしてキャッシュを無効にします。

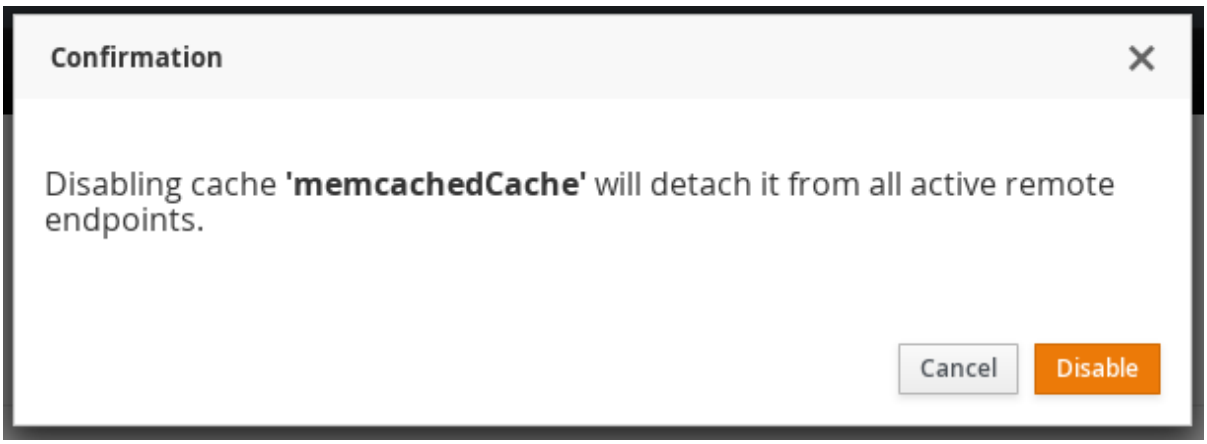


図24.26 キャッシュの無効化を確認

4. 後続のダイアログボックスが表示されます。「**`Ok`**」をクリックします。

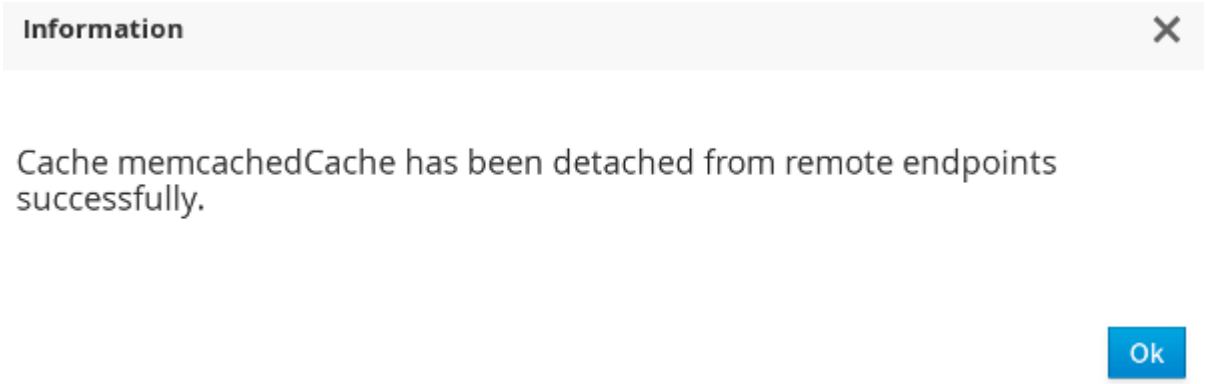


図24.27 確認ボックス

5. 選択されたキャッシュは正常に無効にされ、キャッシュ名ラベルの横にあるインジケーター「**Disabled**」と共に表示されます。

図24.28 無効にされたキャッシュ

結果

キャッシュが正常に無効にされます。

以下の手順では、キャッシュを有効にする手順を説明しています。

手順24.6 キャッシュの有効化

1. キャッシュを有効にするには、キャッシュビューから特定の無効にされたキャッシュをクリックします。

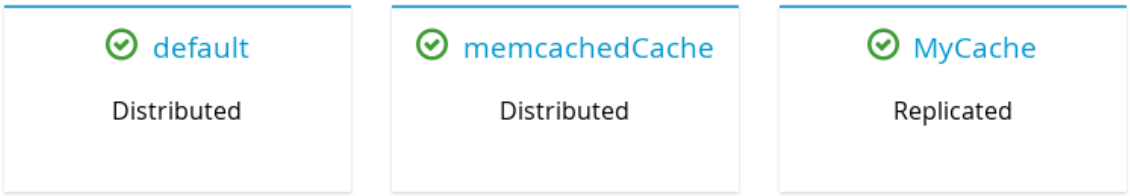


図24.29 キャッシュビュー

2. インターフェースの右側にある「**Actions**」タブをクリックします。

図24.30

3. 「Actions」タブから「**Enable**」をクリックします。

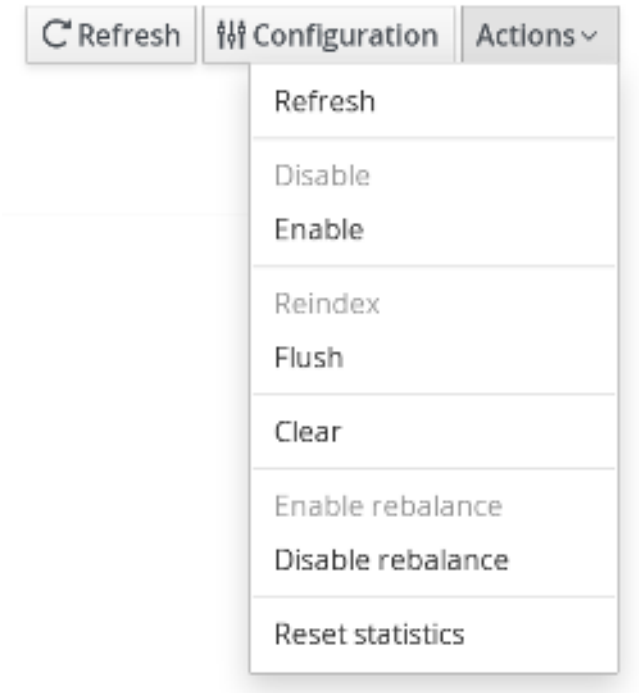


図24.31 アクションメニュー

4. 確認ダイアログボックスが表示されます。「**Enable**」をクリックします。

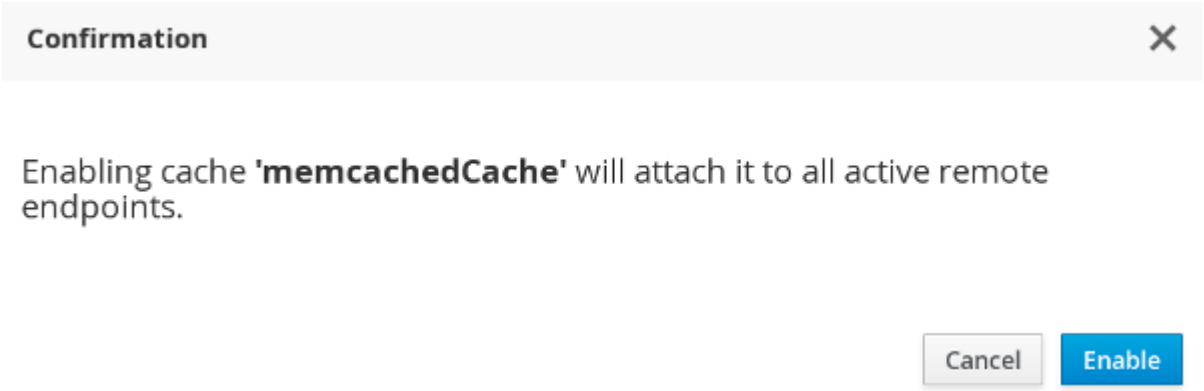


図24.32 確認ボックス

5. 後続のダイアログボックスが表示されます。「**Ok**」をクリックします。

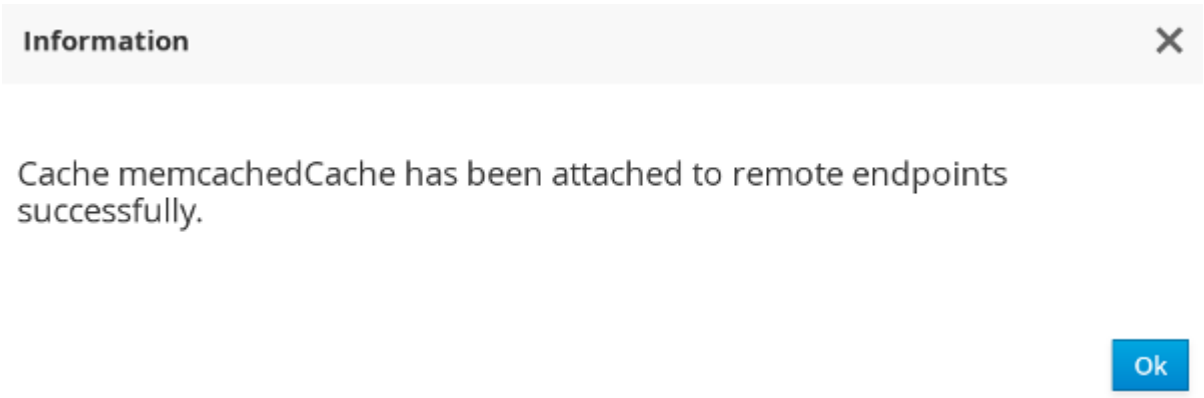


図24.33 情報ボックス

6. 選択されたキャッシュは正常に無効にされ、キャッシュ名ラベルの横にあるインジケーター「**Enabled**」と共に表示されます。

図24.34 有効にされたキャッシュ

バグを報告する

24.5.5. キャッシュのフラッシュとクリア

JBoss Data Grid 管理コンソールにより、管理者はキャッシュクリア操作でキャッシュおよびキャッシュストアからすべてのエントリーを削除できます。さらに、コンソールからのフラッシュ操作により、キャッシュメモリーからキャッシュストアにエントリーを保存することができます。

キャッシュのフラッシュ

キャッシュをフラッシュするには、以下の手順を実行します。

手順24.7 キャッシュのフラッシュ

1. 「**Cache Containers**」ビューで、キャッシュコンテナーの名前をクリックします。
2. キャッシュビューが表示されます。クリアするキャッシュをクリックします。

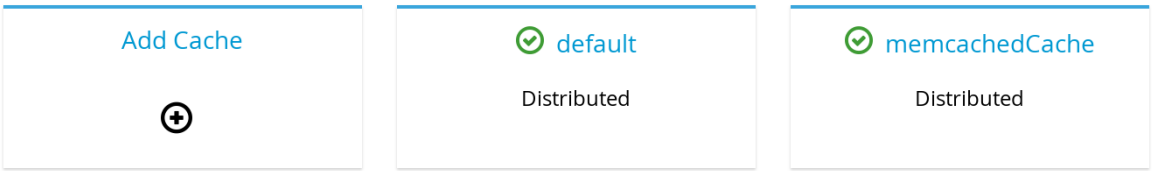


図24.35 キャッシュビュー

3. キャッシュの統計ページが表示されます。右側にある「**Actions**」をクリックします。

図24.36 アクションボタン

4. 「**Actions**」メニューから「**Flush**」をクリックします。

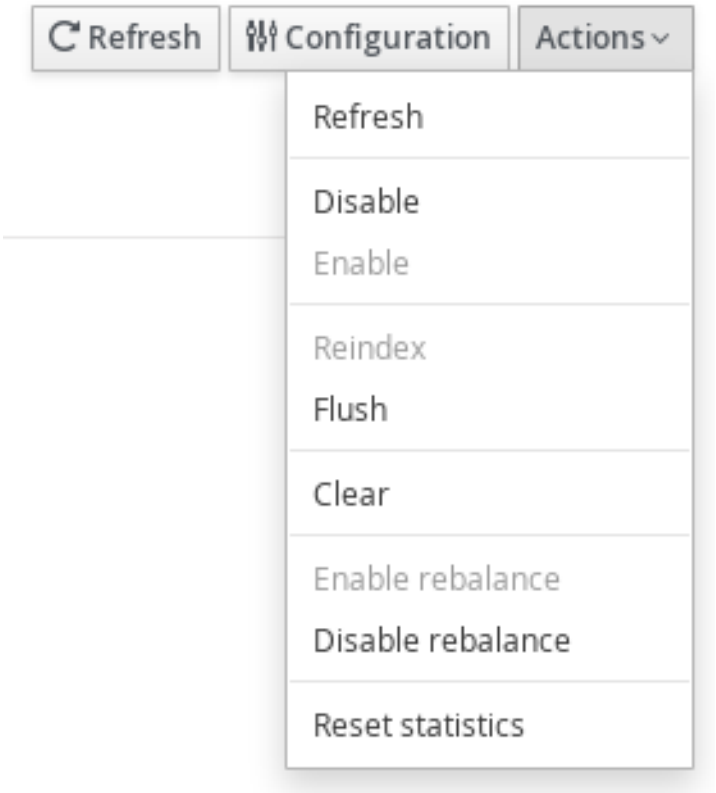


図24.37 アクションメニュー

5. 確認ダイアログボックスが表示されます。「**Flush**」をクリックします。

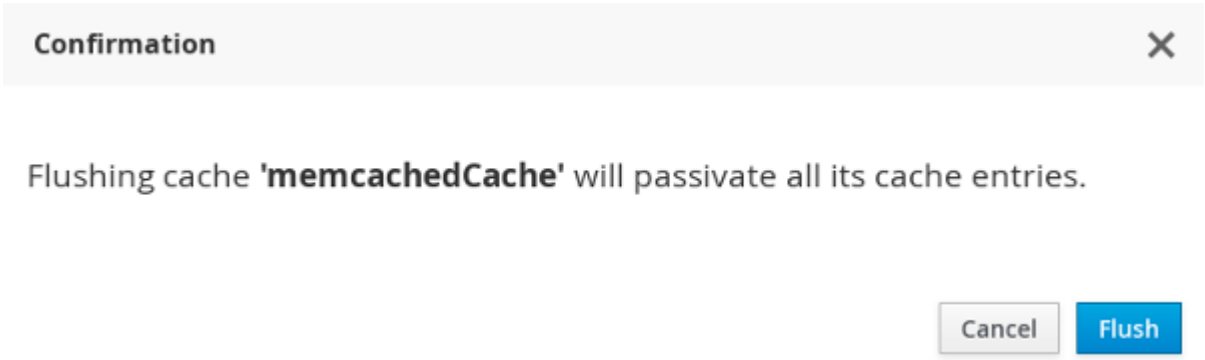


図24.38 キャッシュのフラッシュ確認ボックス

6. キャッシュが正常にフラッシュされます。「**Ok**」をクリックします。

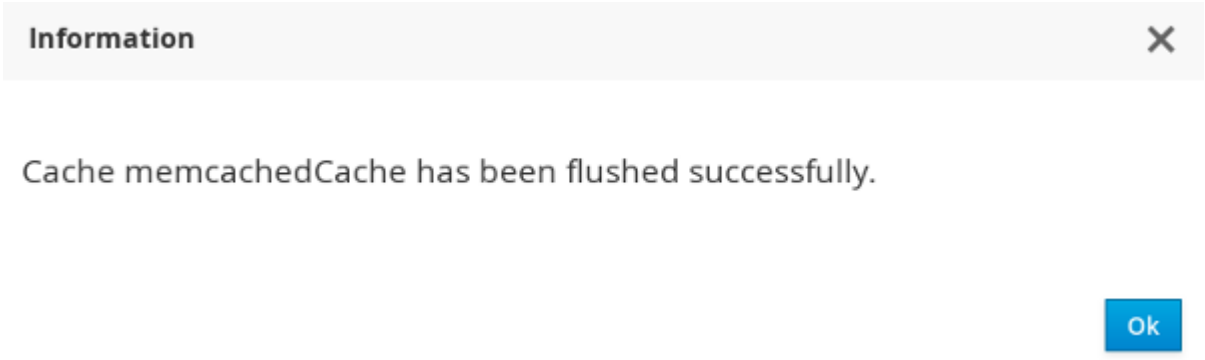


図24.39 キャッシュのフラッシュの情報ボックス

キャッシュのクリア

キャッシュをクリアするには、以下の手順を実行します。

手順24.8 キャッシュのクリア

1. 「Cache Containers」ビューで、キャッシュコンテナーの名前をクリックします。
2. キャッシュビューが表示されます。クリアするキャッシュをクリックします。

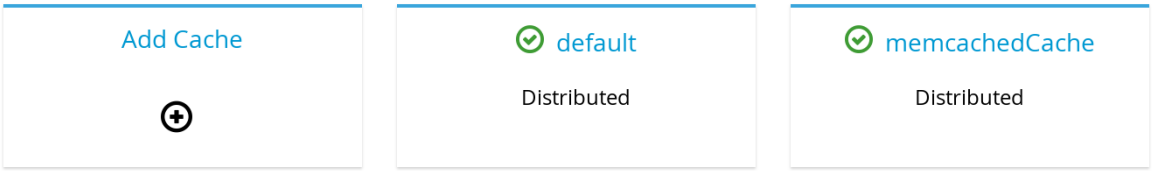


図24.40 キャッシュビュー

3. キャッシュの統計ページで、右側にある「**Actions**」をクリックします。

図24.41

4. 「**Actions**」メニューから、「**Clear**」をクリックします。

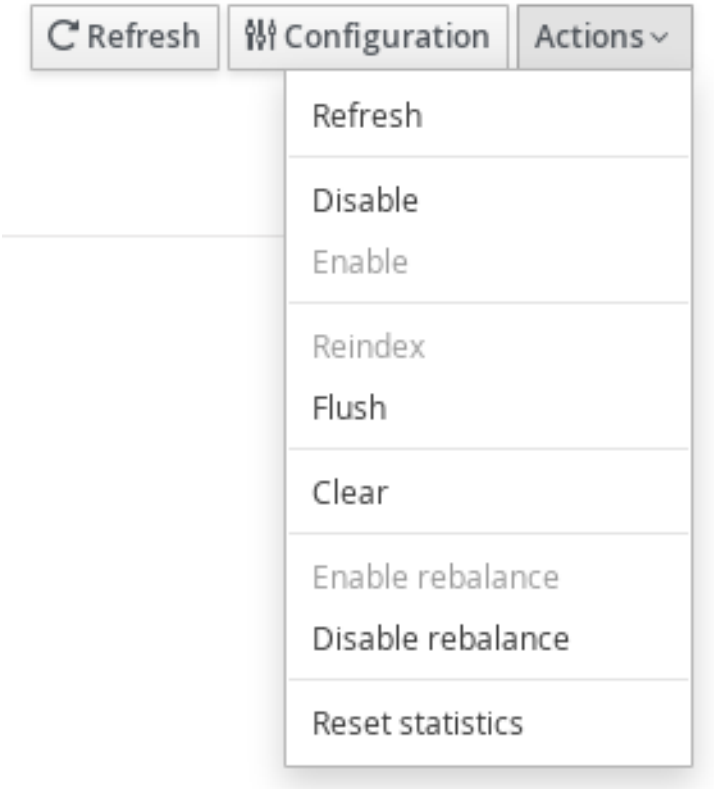


図24.42 クリアボタン

5. 確認ダイアログボックスが表示されます。「**Clear**」をクリックします。

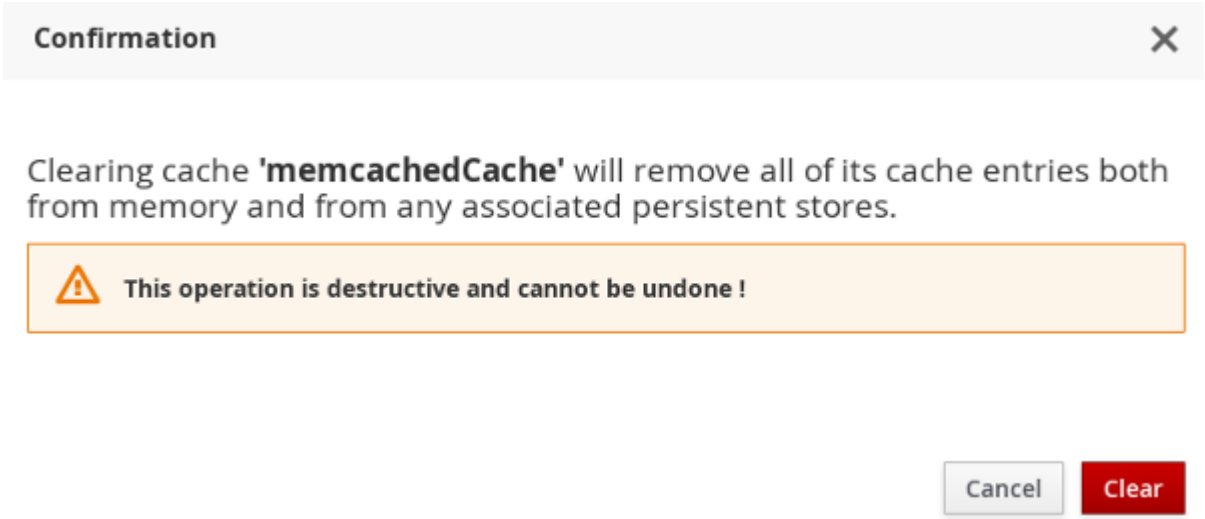


図24.43 確認ボックス

6. キャッシュが正常にフラッシュされます。「**Ok**」をクリックします。

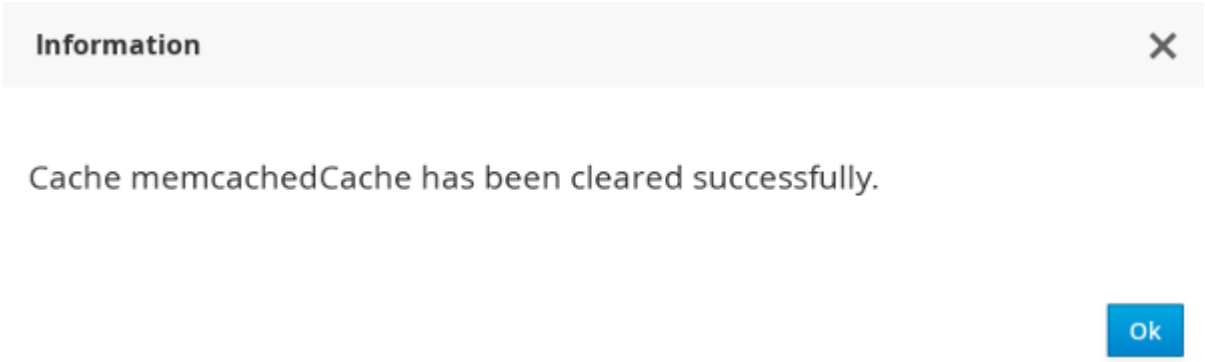


図24.44 情報ボックス

バグを報告する

24.5.6. サーバータスクの実行

JBoss Data Grid 管理コンソールを使用する管理者は、JBoss Data Grid クラスターでサーバーのスクリプトジョブを開始できます。

バグを報告する

24.5.7. サーバータスク

24.5.7.1. 新規サーバータスク

以下の手順では、新規サーバータスクを起動する方法を説明しています。

手順24.9 新規サーバータスクの起動

1. JBoss Data Grid 管理コンソールの「Cache Containers」ビューで、キャッシュコンテナーの名前をクリックします。

2. キャッシュビューページで、「**Task Execution**」タブをクリックします。

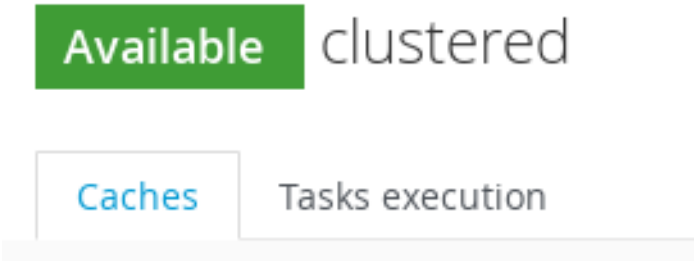


図24.45 タスクの実行

3. 「Tasks execution」タブで、「**Launch new task**」をクリックします。

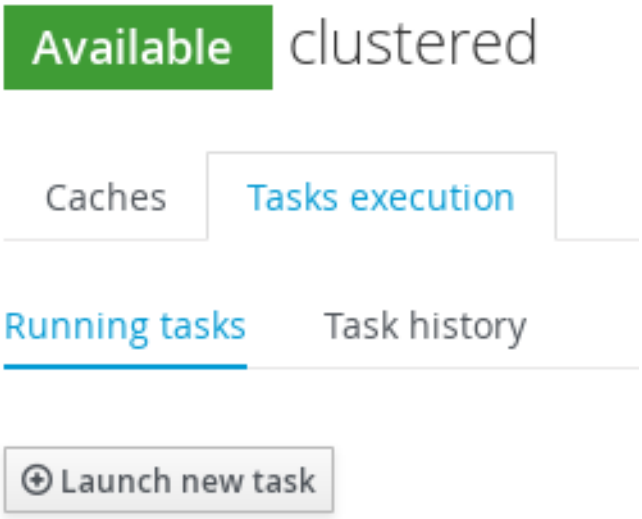


図24.46 新規タスクの起動

4. 新規タスクプロパティーを入力し、「**Launch task**」をクリックします。

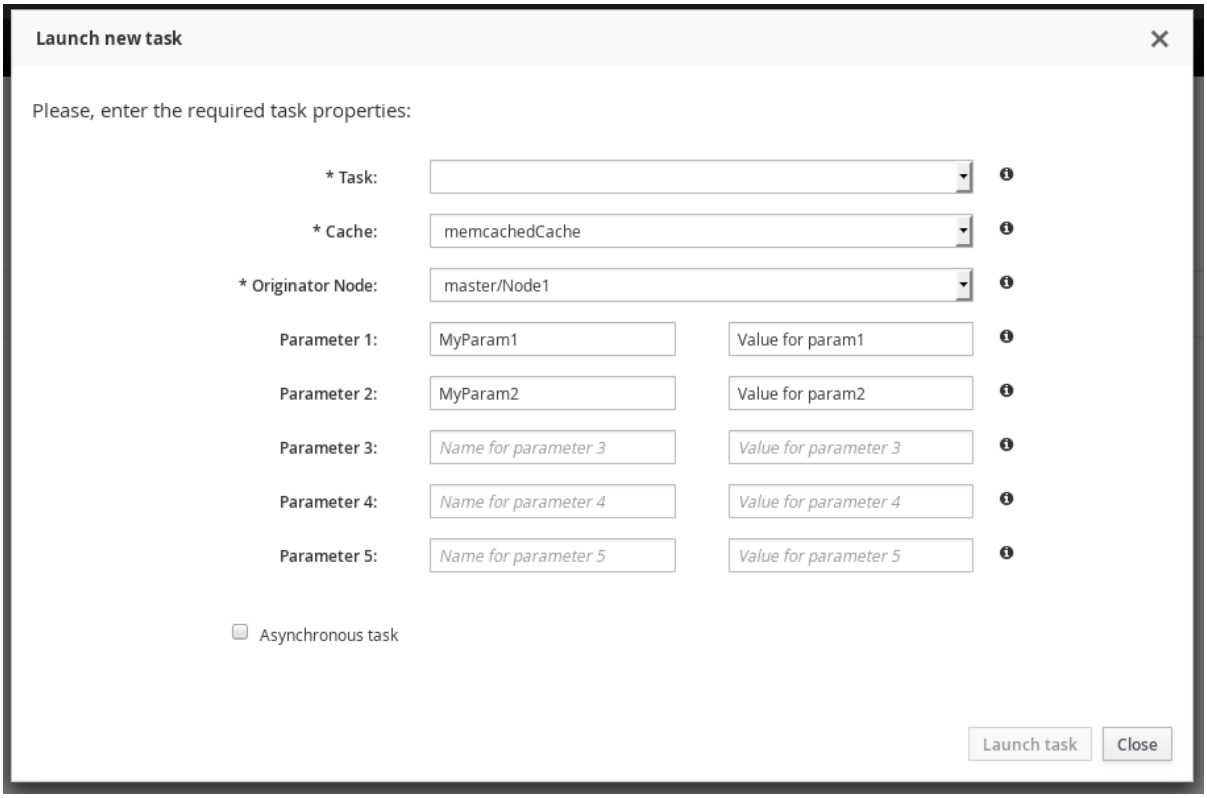


図24.47 タスクプロパティー

結果

新規サーバータスクが正常に作成されます。

バグを報告する

24.5.7.2. サーバータスクビュー

サーバータスクの起動後は、サーバータスクは他の実行中のタスクと共に「**Tasks execution**」タブで表示できます。完了したサーバースクリプトジョブのセットを開始時間と終了時間と共に表示できます。さらに、成功した実行回数と失敗回数も表示できます。

Status	Cache name	Task name	Parameters
Error	My cache #1	Do some calculations	MyParam1 = 'Value for param1' AnotherParam2 = 'Another value' TheParam3 = 'Final value'
Success	My cache #2	Calculate aggregate stuff	-
Error	My cache #1	Do some calculations	MyParam1 = 'Value for param1' AnotherParam2 = 'Another value' TheParam3 = 'Final value'

図24.48 サーバータスクビュー

Start time	End time
Thu Sep 3 17:10:02 CEST 2015	Thu Sep 3 17:14:02 CEST 2015
Thu Sep 3 17:10:02 CEST 2015	Thu Sep 3 17:14:02 CEST 2015
Thu Sep 3 17:10:02 CEST 2015	Thu Sep 3 17:14:02 CEST 2015

図24.49 タスクの開始/終了時間

バグを報告する

24.6. キャッシュコンテナーの設定

JBoss Data Grid 管理コンソールにより、ユーザーは、トランスポートやスレッドプール、セキュリティー、キャッシュテンプレート、リモートの実行可能ファイル/スクリプトのデプロイメントなどの、キャッシュコンテナーのレベル設定を表示し、設定することができます。それぞれのキャッシュコンテナーはクラスターに関連付けられます。

以下の手順では、キャッシュコンテナー設定にアクセスする手順を簡単に説明しています。

手順24.10 キャッシュコンテナー設定へのアクセス

1. 「Cache Containers」ビューで、キャッシュコンテナーの名前をクリックします。

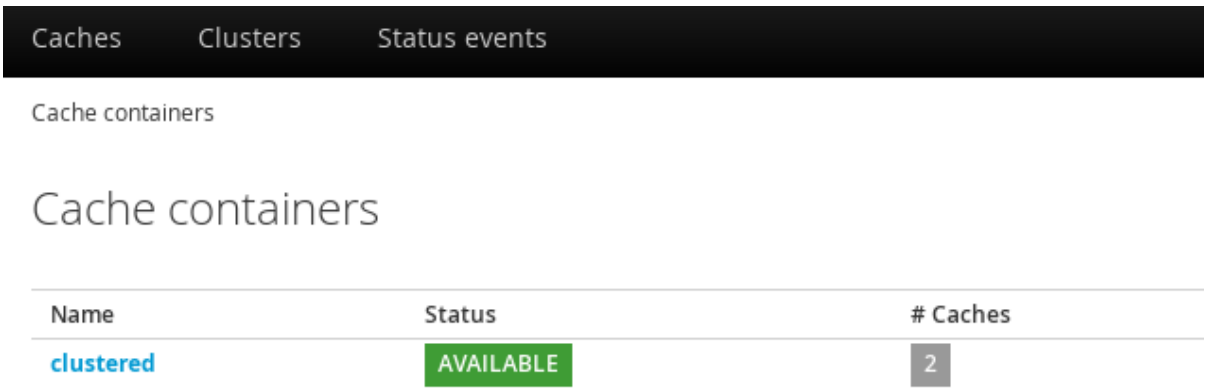


図24.50 キャッシュコンテナービュー

2. インターフェースの右側にある「Configuration」設定ボタンをクリックします。

図24.51 設定

キャッシュコンテナー設定インターフェースが表示されます。

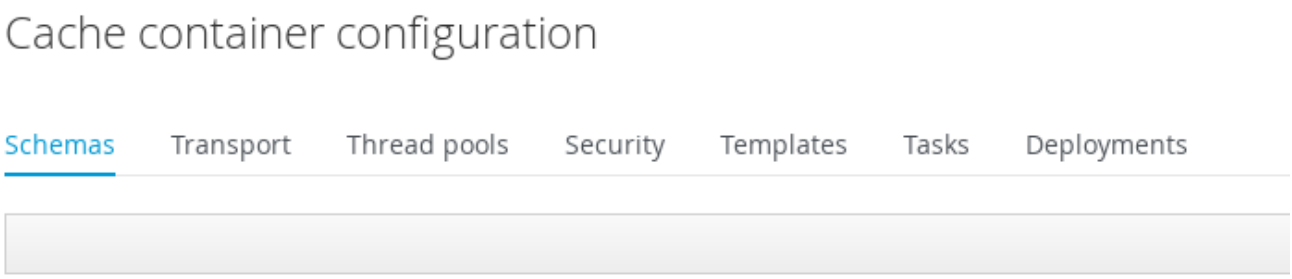


図24.52 キャッシュコンテナーの設定

バグを報告する

24.6.1. プロトコルバッファースキーマの定義

プロトコルバッファースキーマは、キャッシュコンテナー設定インターフェースで定義されます。

以下の手順は、プロトコルスキーマを定義する手順を簡単に説明しています。

手順24.11 プロトコルスキーマの定義

1. 「Schema」タブの右上にある「**Add**」をクリックしてスキーマ作成ウィンドウを起動します。
2. それぞれのフィールドにスキーマ名とスキーマを入力し、「**Create Schema**」をクリックします。

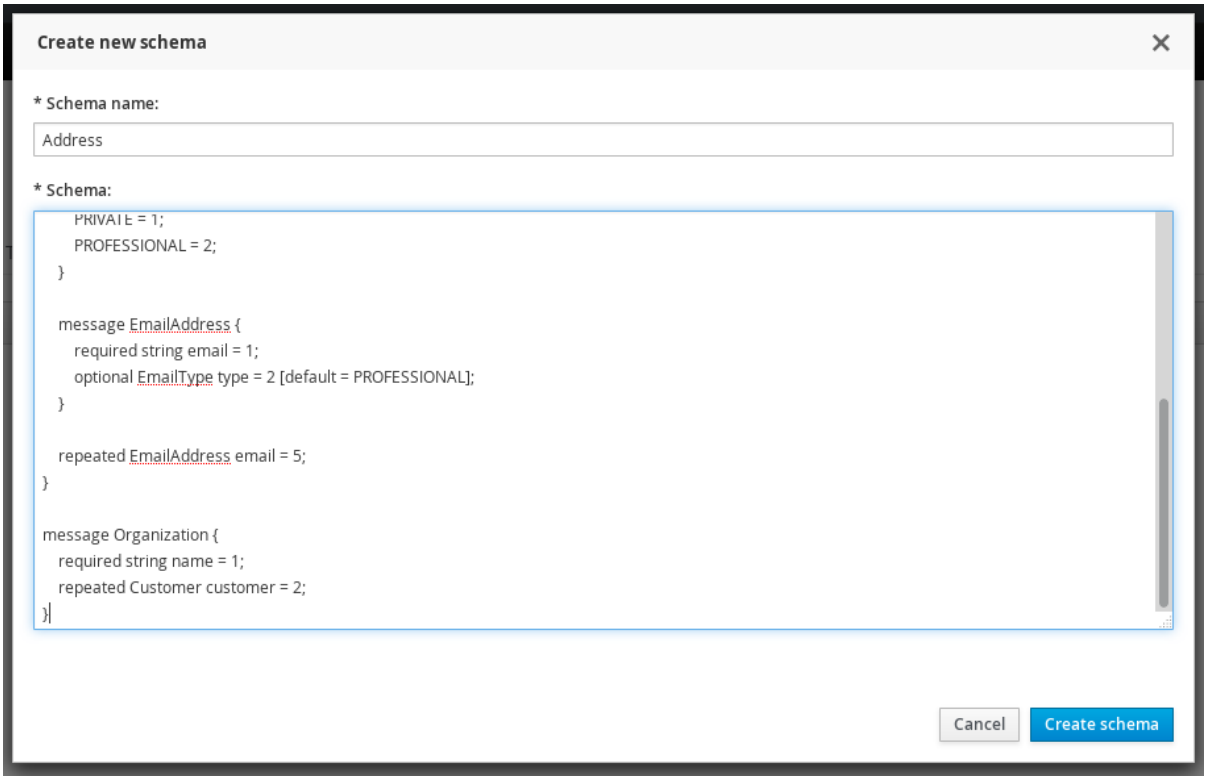


図24.53 新規スキーマ

3. プロトコルバッファースキーマが追加されます。

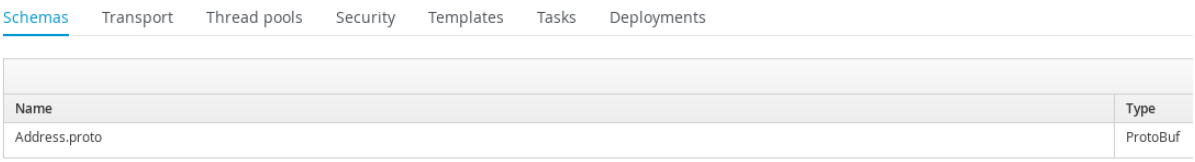


図24.54 プロトコルバッファー

バグを報告する

24.6.2. トランスポート設定

トランスポート設定にアクセスするには、キャッシュコンテナー設定インターフェースの「**Transport**」タブをクリックします。トランスポート設定を入力し、「**Save**」をクリックします。

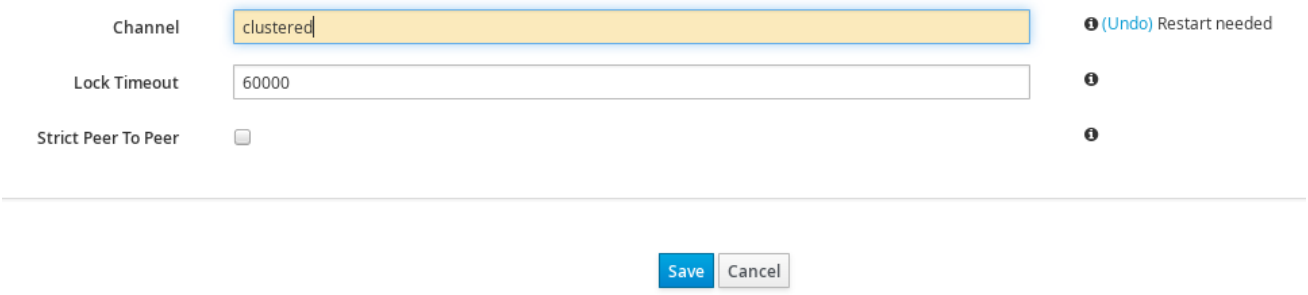


図24.55 トランスポート設定

設定変更により、サーバーの再起動を求めるダイアログがプロンプトされます。再起動して変更を適用します。

図24.56 再起動の確認

バグを報告する

24.6.3. スレッドプールの定義

異なるキャッシュ関連操作のスレッドプールを定義するには、キャッシュコンテナー設定インターフェースで「Thread Pools」タブをクリックします。

JBoss Data Grid 管理コンソールを使用すると、管理者は以下のキャッシュレベルの操作のスレッドプール値を設定できます。

`Async Operations (非同期操作)`

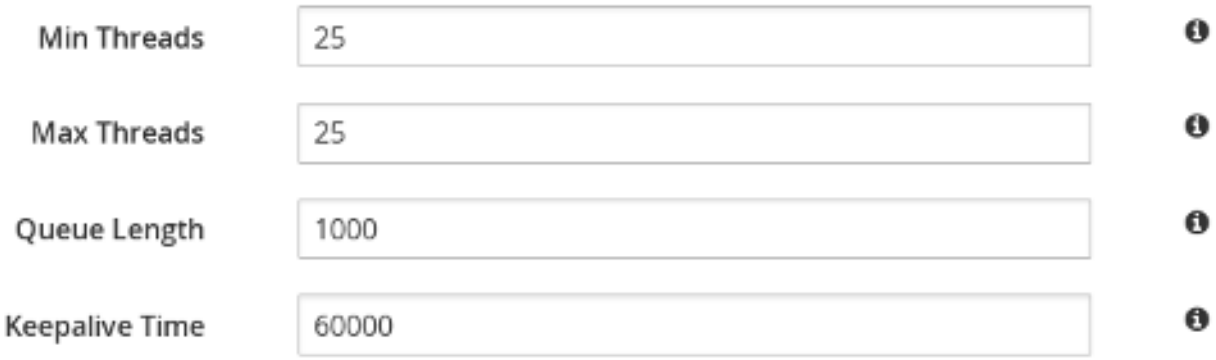


図24.57 非同期操作

各パラメーターのデフォルト値はコンソールで設定されます。ツールチップ形式でパラメーターの説明を表示するには、情報アイコン上にカーソルを移動します。スレッドプール値を変更するには、パラメーターフィールドに新規の値を入力し、「**Save**」をクリックします。サーバーの再起動が値を変更するたびに必要になります。

`Expiration (エクスパレーション)`

エクスパレーションの設定については、ユーザーは以下のパラメーターの値を設定できます。

図24.58 エクスパレーションの値

`Listener` (リスナー)

リスナーの設定については、ユーザーは以下のパラメーターの値を設定できます。

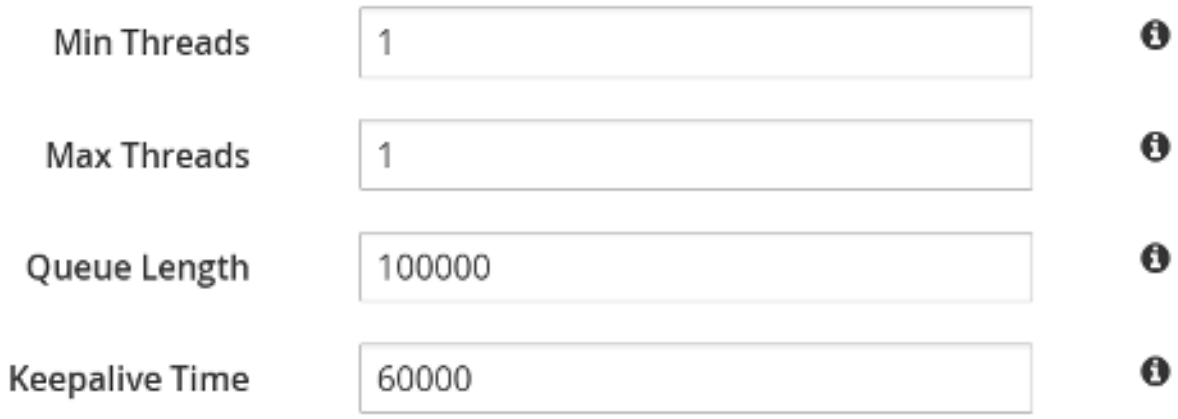


図24.59 リスナーの値

`Persistence` (永続性)

永続性の設定については、ユーザーは以下のパラメーターの値を設定できます。

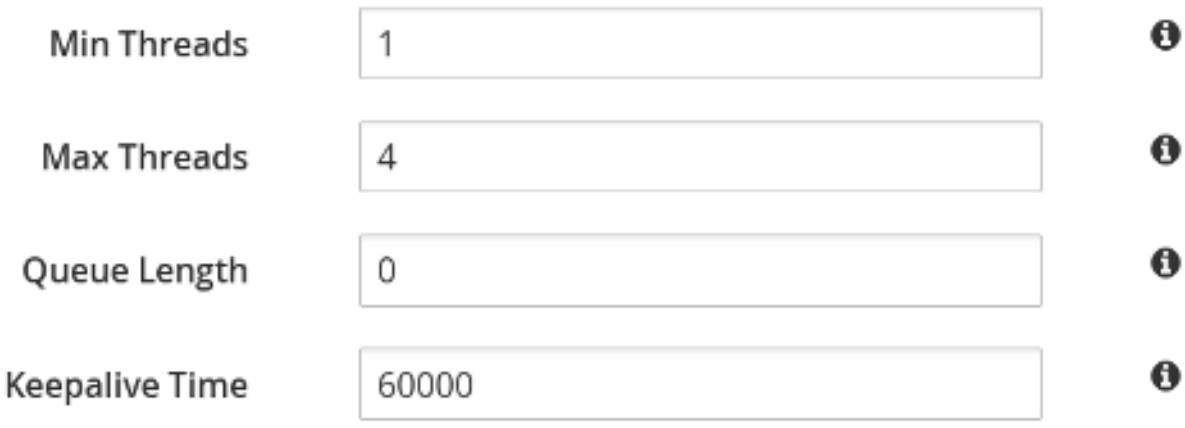


図24.60 永続性の値

`Remote Commands` (リモートコマンド)

リモートコマンドの設定については、ユーザーは以下のパラメーターの値を設定できます。

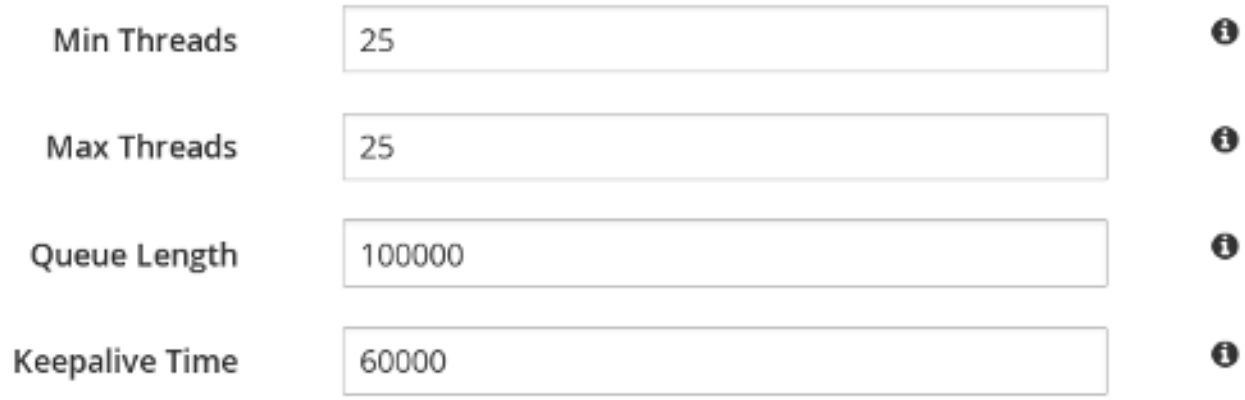


図24.61 リモートコマンド

`Replication Queue (レプリケーションキュー)`

レプリケーションキューの設定については、ユーザーは以下のパラメーターの値を設定できます。

図24.62 レプリケーションキューの値

`State Transfer (状態転送)`

リスナーの設定については、ユーザーは以下のパラメーターの値を設定できます。

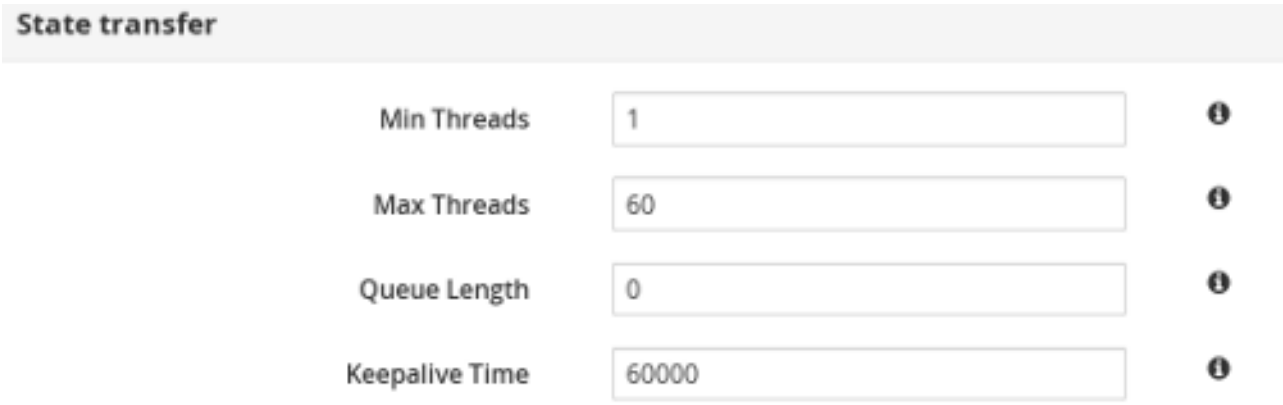


図24.63 状態転送の値

`Transport (トランスポート)`

トランスポートの設定については、ユーザーは以下のパラメーターの値を設定できます。

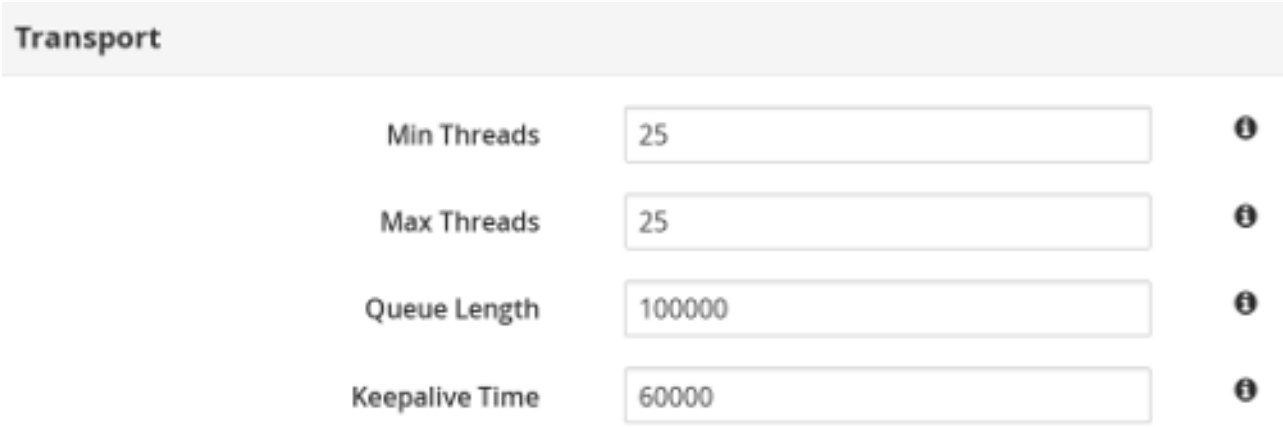


図24.64 トランスポートの値

バグを報告する

24.6.4. 新規セキュリティーロールの追加

以下の手順は、新規セキュリティーロールを追加する方法を簡単に説明しています。

手順24.12 セキュリティーロールの追加

1. 「**Security**」タブをクリックします。キャッシュコンテナーに権限が定義されていない場合は、**Yes** をクリックして定義します。

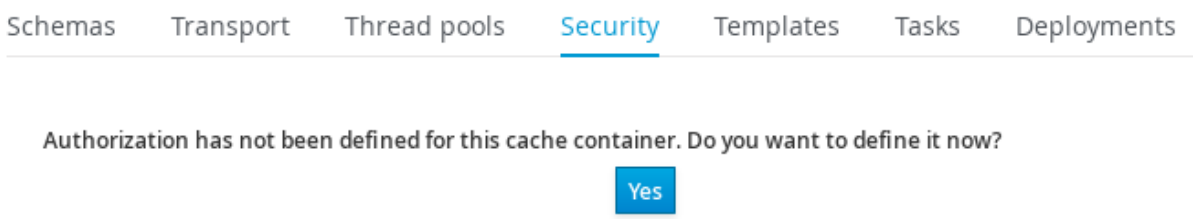


図24.65 権限の定義

2. ドロップダウンメニューからロールマッパーを選択します。「**Add**」をクリックしてパーミッションウィンドウを起動します。

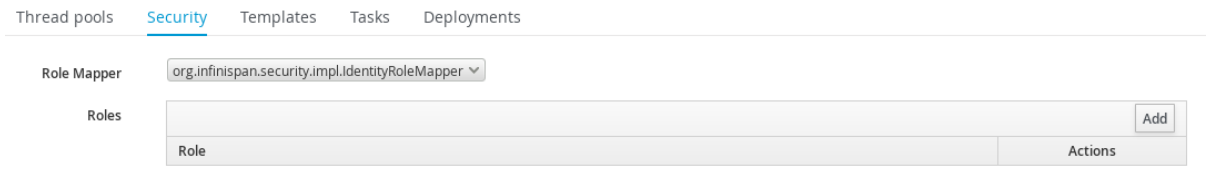


図24.66 ロールマッパーの選択

3. 「**Permissions**」ウィンドウで、新規ロールの名前を入力し、チェックボックスにチェックを付けてパーミッションを割り当てます。「**Save changes**」をクリックしてロールを保存します。

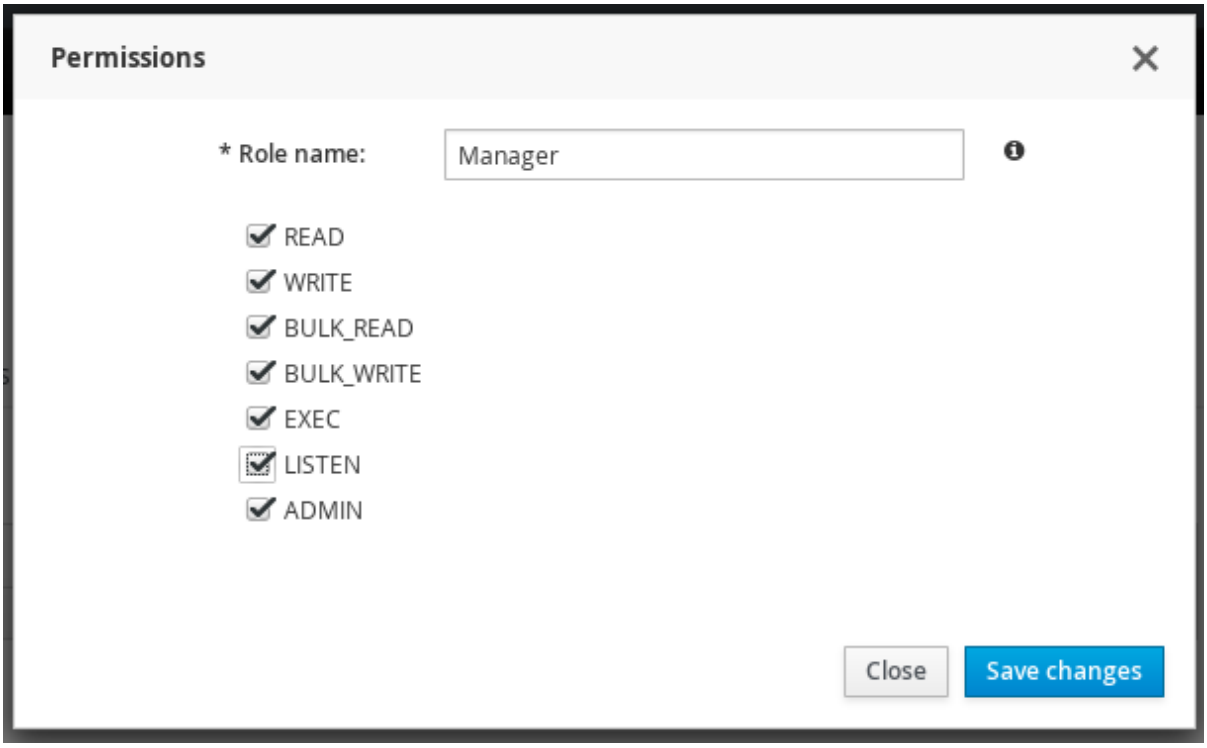


図24.67 ロールパーミッション

4. 新規のセキュリティーロールが追加されます。

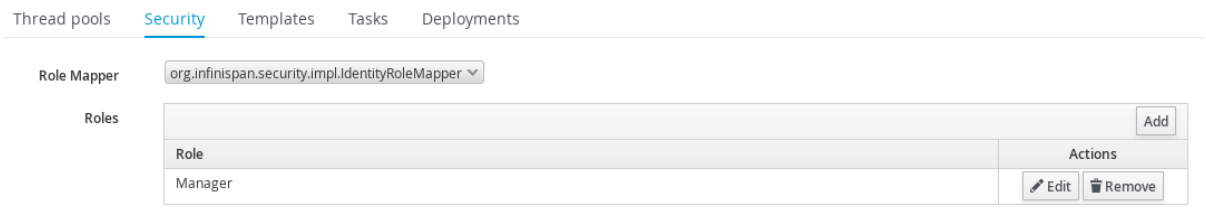


図24.68 新規セキュリティーロール

バグを報告する

24.6.5. キャッシュ設定テンプレートの作成

キャッシュコンテナー設定インターフェースの「**Templates**」タグは、設定済みの利用可能なキャッシュテンプレートすべてを一覧表示します。

Schemas Transport Thread pools Security Templates Tasks Deployments

Create new template

Name	Type	Mode	Traits	Actions
transactional	distributed-cache	SYNC	Transactional	Edit Remove
persistent-file-store	distributed-cache	SYNC		Edit Remove
indexed	distributed-cache	SYNC	Indexed	Edit Remove
memory-bounded	distributed-cache	SYNC	Bounded	Edit Remove
persistent-file-store-passivation	distributed-cache	SYNC	Bounded	Edit Remove
persistent-file-store-write-behind	distributed-cache	SYNC		Edit Remove
persistent-leveldb-store	distributed-cache	SYNC		Edit Remove
persistent-jdbc-string-keyed	distributed-cache	SYNC		Edit Remove

図24.69 キャッシュテンプレートビュー

以下の手順は、新規キャッシュ設定テンプレートを作成する方法を簡単に説明しています。

手順24.13 新規キャッシュ設定テンプレートの作成

1. テンプレート一覧の右側にある「**`Create new Template`**」をクリックします。
2. キャッシュ設定テンプレート名を入力し、ドロップダウンからベース設定を選択して「**`Next`**」をクリックします。

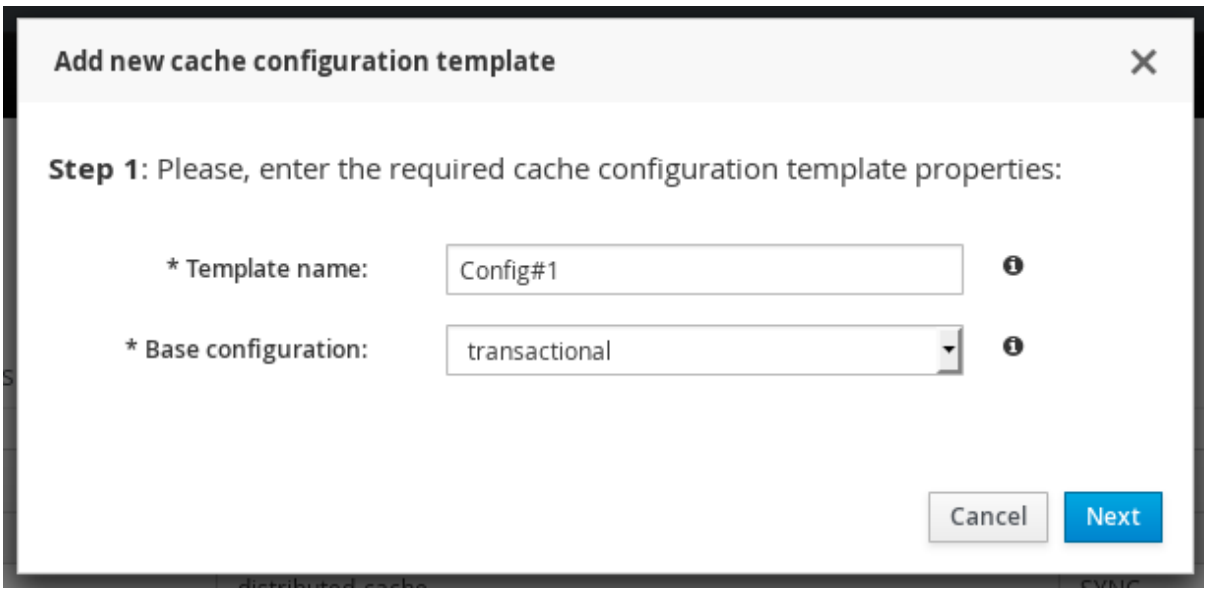


図24.70 キャッシュ設定テンプレート

3. ロックやエクスパレーション、インデックスなどの各種のキャッシュ操作のキャッシュテンプレート属性を設定します。

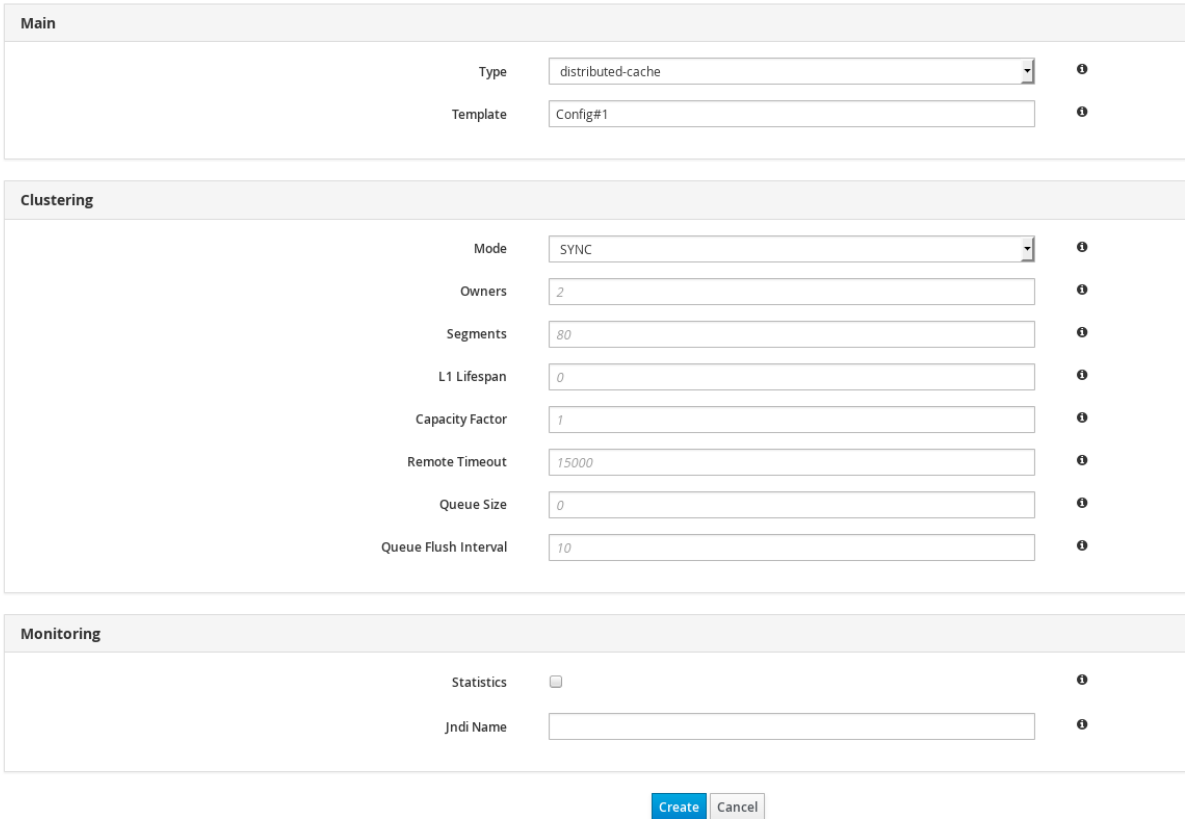


図24.71 キャッシュ設定テンプレート

4. 値を入力した後に「**Create**」をクリックしてキャッシュテンプレートを作成します。

バグを報告する

24.7. クラスターの管理

24.7.1. クラスターノードビュー

クラスターセントリックビューにより、各サーバーグループについて作成されるノードを表示し、デプロイされたサーバーの一覧を表示できます。クラスタービューでは、新規ノードをクラスターグループに追加でき、特定ノードのパフォーマンスメトリックスを表示できます。

クラスタービューにアクセスするには、ダッシュボードから「**Clusters**」タブに移動し、クラスターの名前をクリックします。

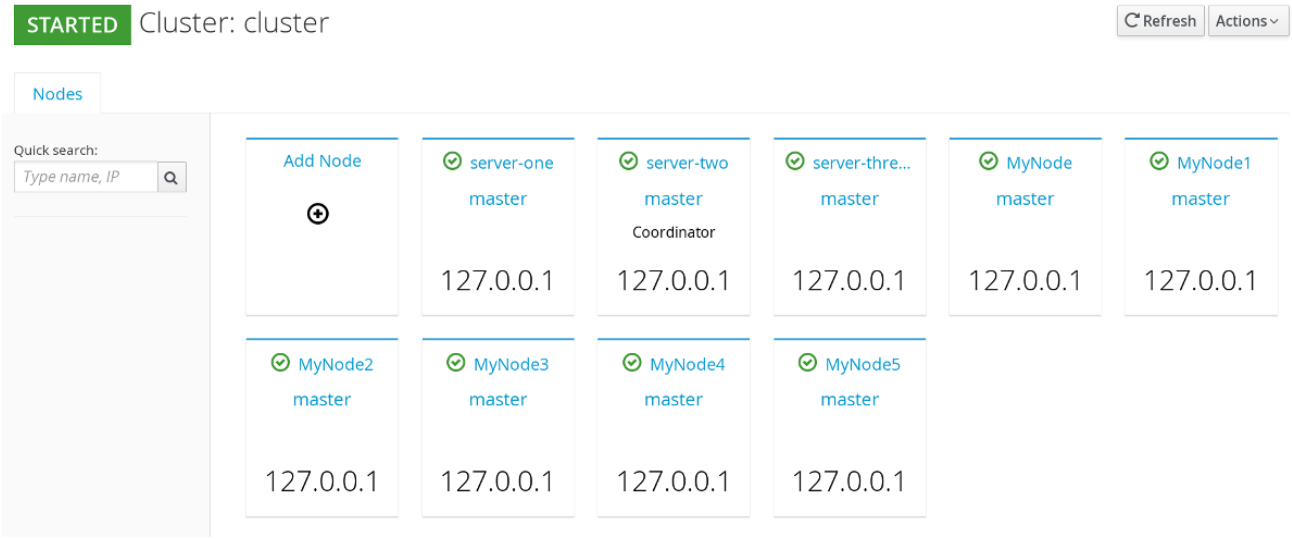


図24.72 ノードビュー

バグを報告する

24.7.2. クラスターノードの不一致

JBoss Data Grid クラスターのサーバーノードの合計数は JBoss Data Grid 管理コンソールに表示されるノード数に一致することが望ましいと言えます。万一何かの理由によりコンソールで予想されるノードが JBoss Data Grid の物理クラスターのノードの合計数に一致しない場合、コンソールは検出されたノード数と予想されるノード数を表示して不一致についての警告を発行します。サーバーノードの予想される数を把握しておくことはネットワークパーティションの処理に役立ちます。

ノードの不一致が生じる場合は、クラスタービューのノードの一覧の上に表示される警告を確認できます。クラスタービューにアクセスするには、ダッシュボードから「**Clusters**」タブに移動してクラスターの名前をクリックします。

以下の画面で、コンソールは警告の形式でユーザーにアラートを出します。予想されるサーバーノードの数は 5 つですが、3 つのノードのみがコンソールで検出されています。

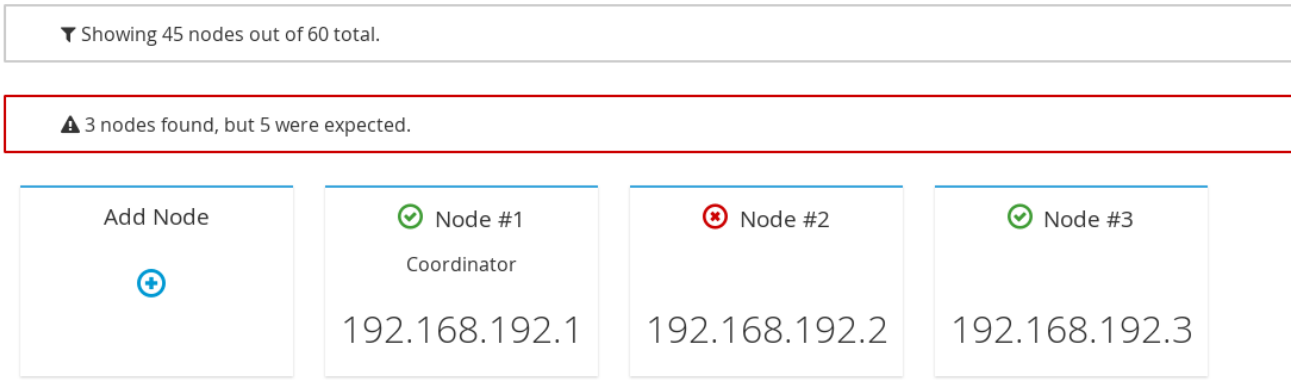
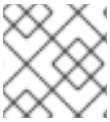


図24.73 クラスターノードの不一致

バグを報告する

24.7.3. クラスターの再調整

Red Hat JBoss Data Grid 管理コンソールを使うと、ユーザーはキャッシュコンテナーおよびキャッシュレベルでクラスターの再調整を有効および無効にできます。



注記

クラスターの再調整がデフォルトで有効にされます。

以下の手順は、クラスターの再調整をキャッシュコンテナーレベルで有効および無効にする手順を簡単に説明しています。

手順24.14 再調整の有効化および無効化

1. キャッシュコンテナービューで、キャッシュコンテナーの名前をクリックします。
2. キャッシュビューで、右側にある「**Actions**」をクリックします。

図24.74

3. コールアウトメニューが開かれます。「**Disable Rebalancing**」をクリックします。

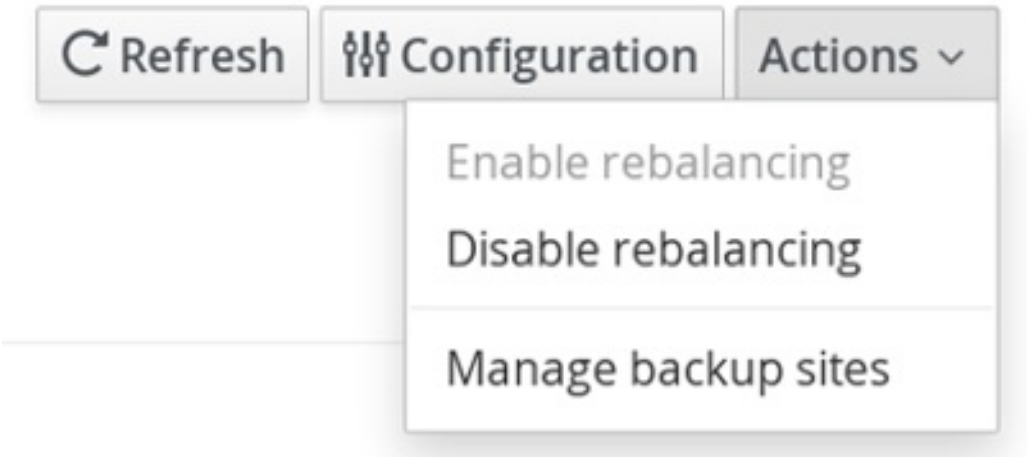


図24.75

4. 確認ダイアログボックスが表示されます。「**Accept**」をクリックします。

図24.76

5. クラスターの再調整が正常に無効にされます。

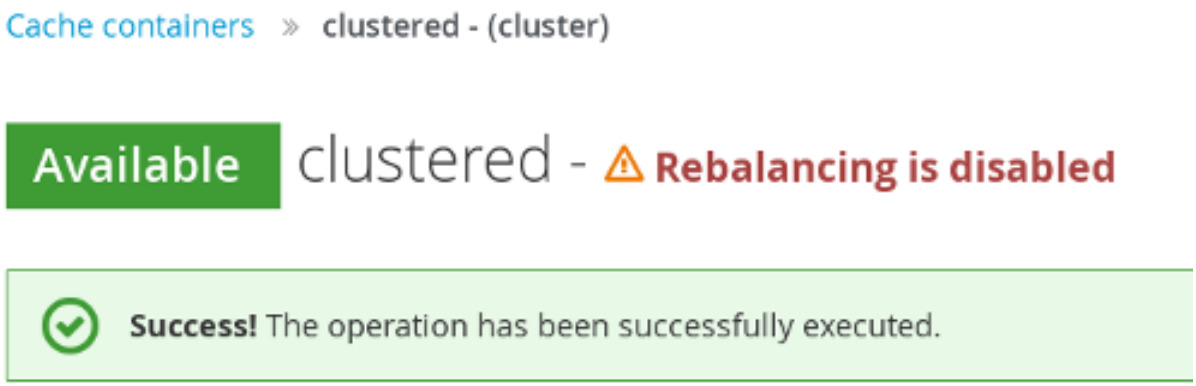


図24.77

6. 再調整を有効にするには、**「Actions」 > 「Enable Rebalancing」**をクリックします。

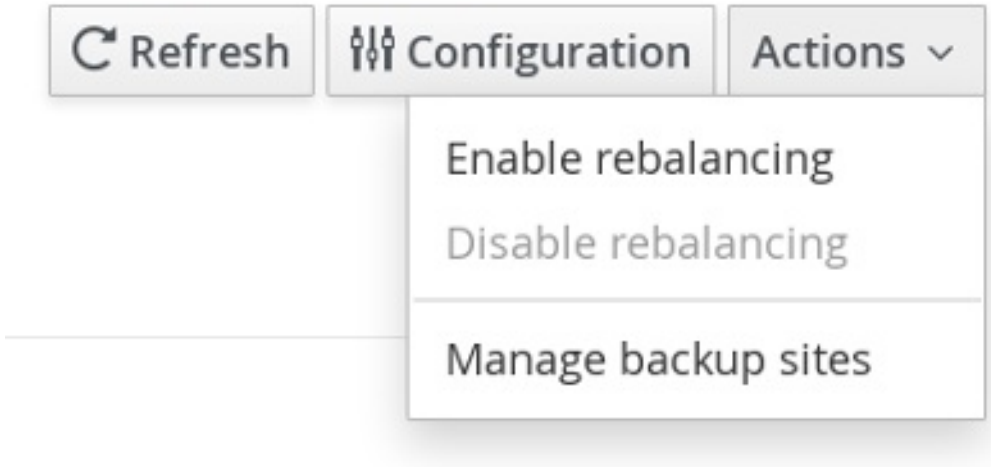


図24.78

7. 確認ダイアログボックスが表示されます。「**Accept**」をクリックします。

図24.79

再調整が正常に有効にされます。

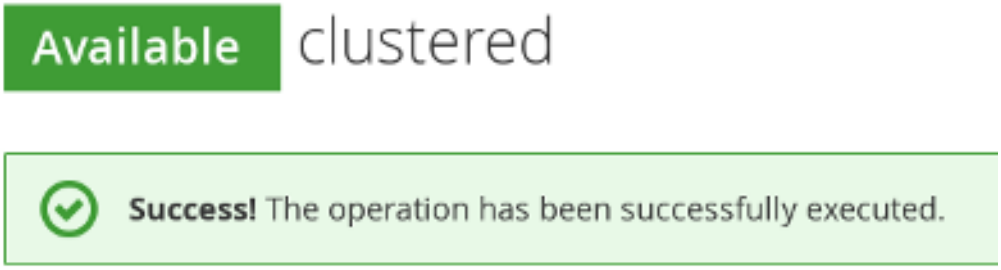


図24.80

以下の手順は、クラスターの再調整をキャッシュレベルで有効および無効にする手順を簡単に説明しています。

手順24.15 再調整の有効化および無効化

1. キャッシュコンテナービューで、キャッシュコンテナーの名前をクリックします。
2. キャッシュビューで、特定のキャッシュをクリックします。
3. キャッシュ統計ページが表示されます。右側にある「**`Actions`**」をクリックします。

図24.81

4. コールアウトメニューから「**`Disable Rebalance`**」をクリックします。

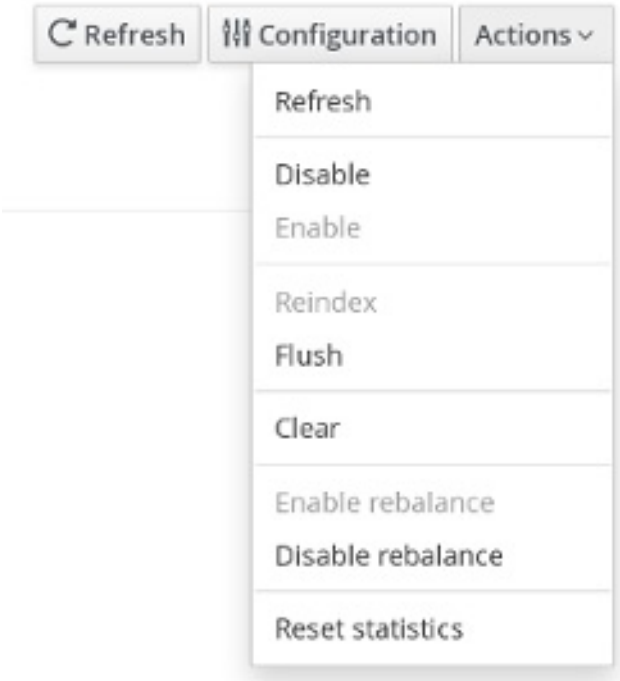


図24.82

5. 確認ダイアログボックスが表示されます。「**`Disable Rebalance`**」をクリックします。

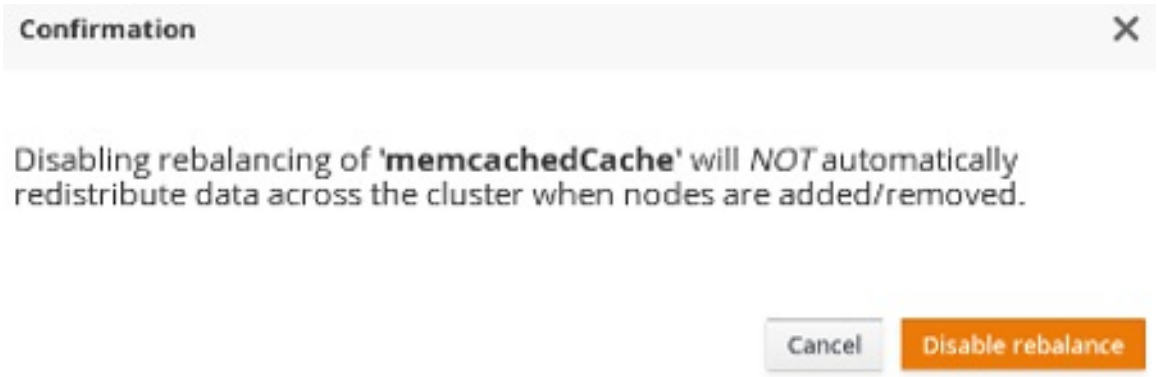


図24.83

6. キャッシュの再調整が正常に無効にされます。

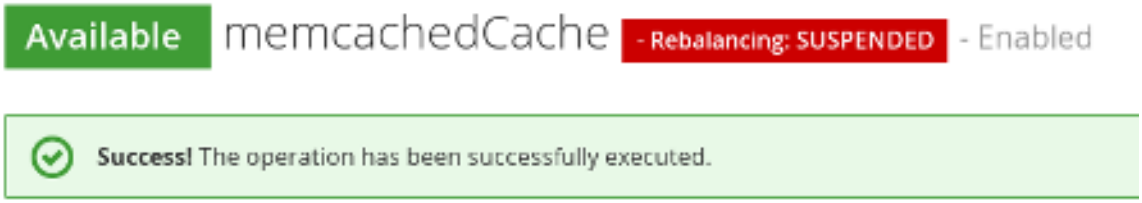


図24.84

7. キャッシュレベルの再調整を有効にするには、「**Actions**」メニューから「**Enable rebalance**」をクリックします。

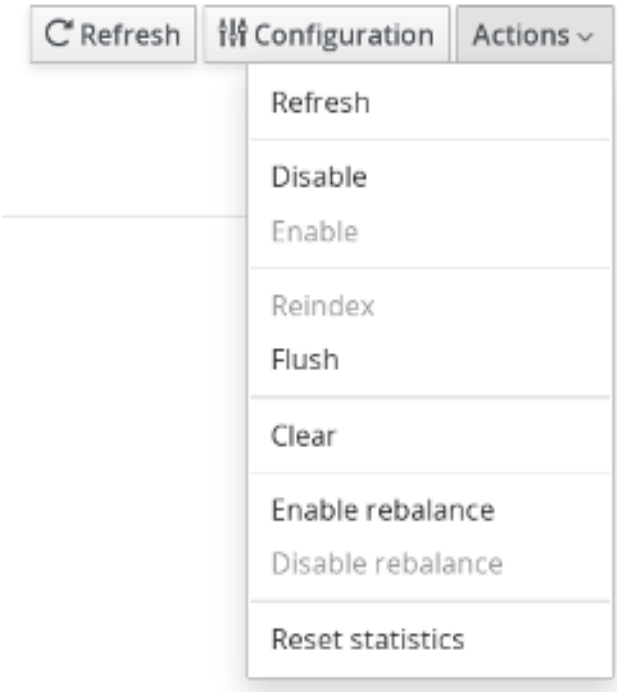


図24.85

8. 確認ダイアログボックスが表示されます。「**Enable rebalance**」をクリックします。

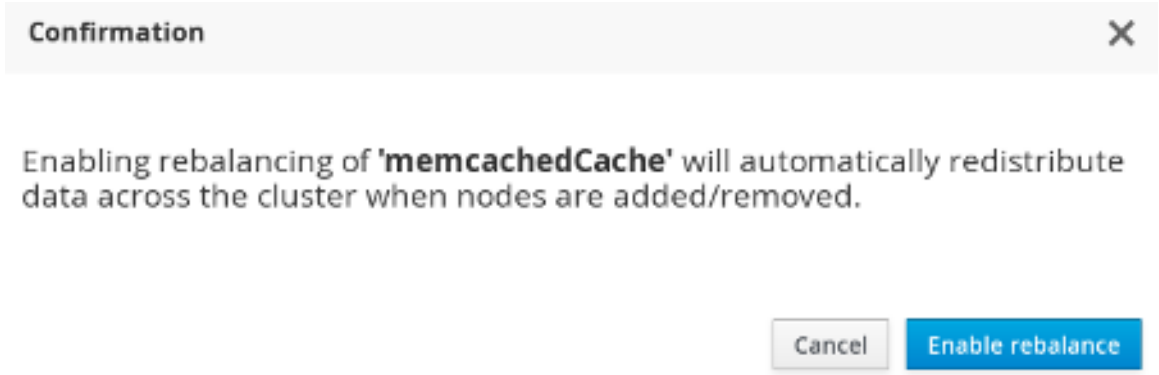


図24.86

キャッシュの再調整が正常に有効にされます。

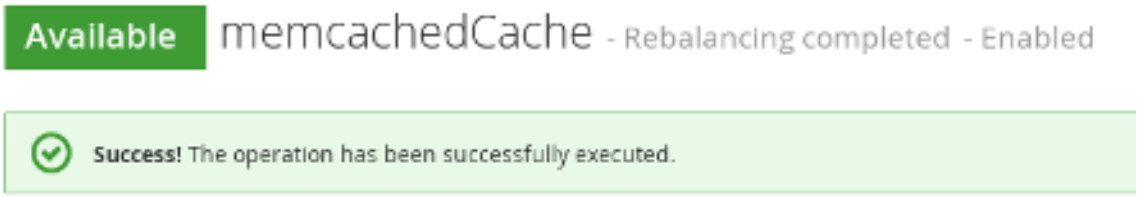


図24.87

バグを報告する

24.7.4. クラスターパーティションの処理

JBoss Data Grid 管理コンソールは、クラスターの状態が「**DEGRADED**」に変更される際に警告をユーザーに表示します。「**DEGRADED**」クラスターの想定される原因には、ネットワークパーティション、到達できないノードまたは予期しない追加ノードの発生が含まれます。

警告は「**Clusters**」ビューに表示されます。「Clusters」ビューにアクセスするには、ダッシュボードから「**Clusters**」タブに移動し、クラスターの名前をクリックします。以下の画面では、「**DEGRADED**」という警告がクラスター名「**JDG Cluster #1**」の横に表示されます。

図24.88 ネットワークパーティションの警告

「**DEGRADED**」クラスターの警告がコンソールのクラスター、キャッシュコンテナーおよびキャッシュの各レベルに表示されます。

バグを報告する

24.7.5. クラスターイベント

JBoss Data Grid 管理コンソールは、統合セクションに cluster-splitおよび cluster-merge イベントなど

のクラスター全体のイベントを表示します。コンソールは、クラスターイベントのほかにも、関連付けられたイベントのタイムスタンプを表示します。クラスターイベントはキャッシュコンテナーのページ、クラスタービューのページおよびダッシュボードの「Status Events」タブに表示できます。

キャッシュコンテナーページでクラスターイベントを表示するには、コンソールへのログイン後のデフォルトのランディングインターフェースであるデフォルトのキャッシュコンテナービューに移動します。クラスターのイベントは、統合セクションの右側にある「**Latest Grid Events**」というタイトルの下に表示されます。

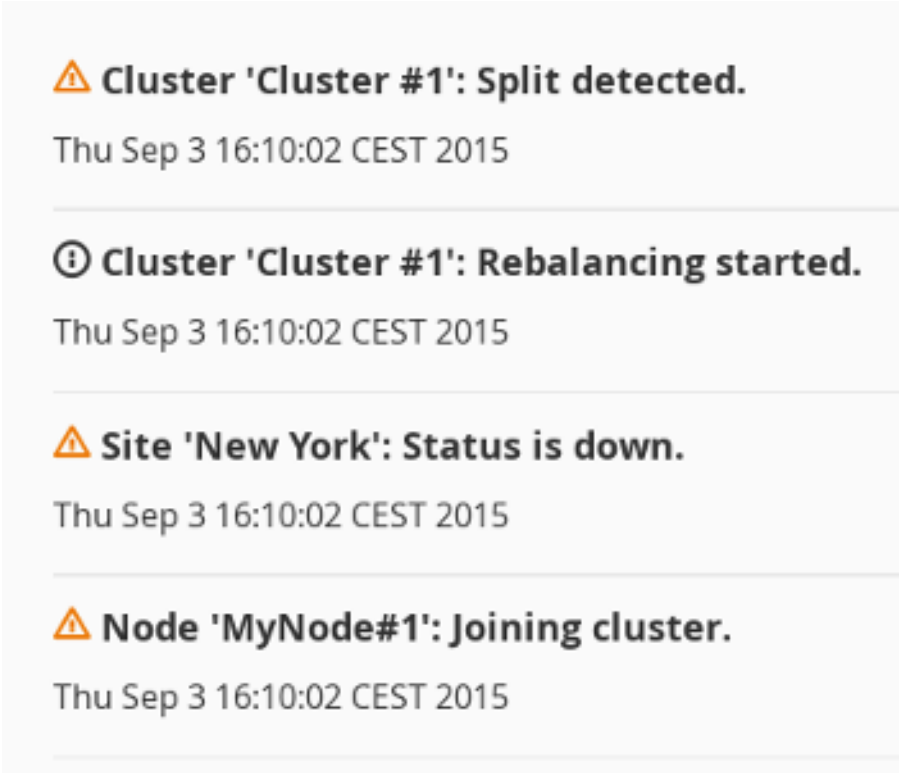


図24.89

クラスタービューページでクラスターイベントを表示するには、「Clusters」タブをクリックしてクラスタービューに移動します。クラスターイベントは、統合セクションの右側にある「**Latest status Events**」というタイトルの下に表示されます。

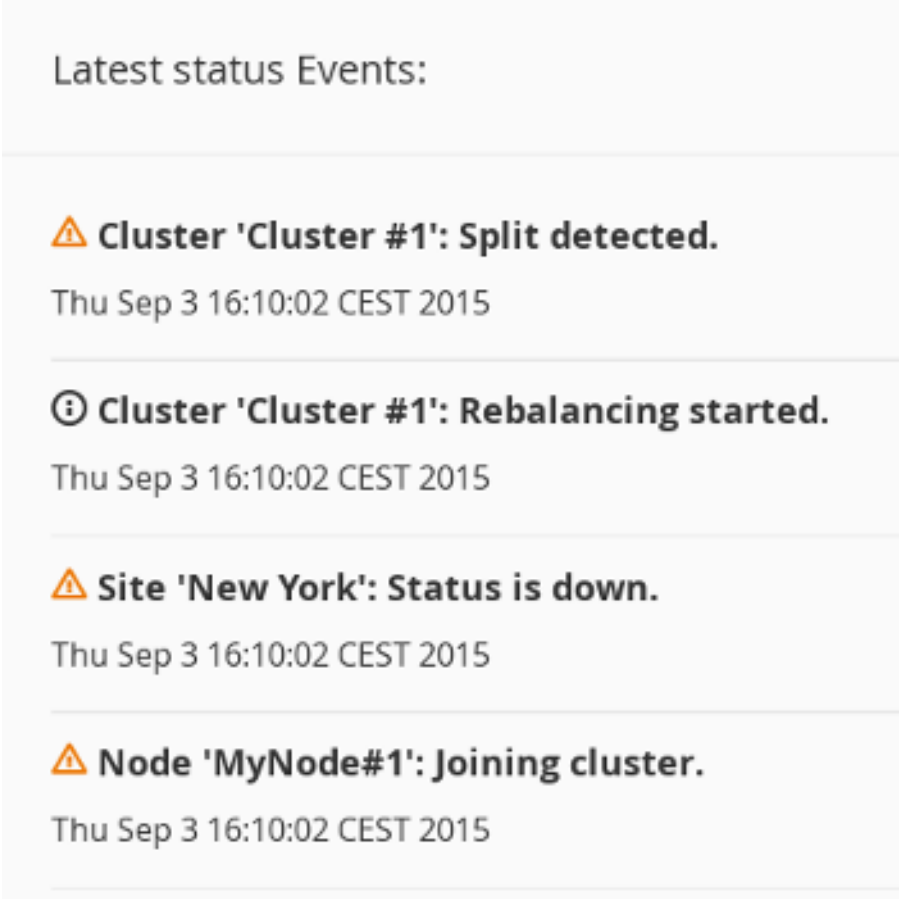


図24.90

バグを報告する

24.7.6. ノードの追加

JBoss Data Grid 管理コンソールを使用すると、管理者は新規ノードを設定できます。

以下の手順は、新規ノードを追加する方法を簡単に説明しています。

手順24.16 新規ノードの追加

1. ダッシュボードビューで、「**Clusters**」タブをクリックします。

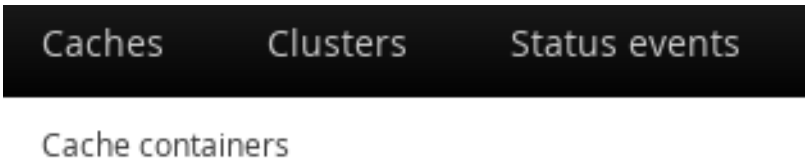


図24.91 「Clusters」タブ

2. 新規ノードを追加する必要のあるクラスターの名前をクリックします。

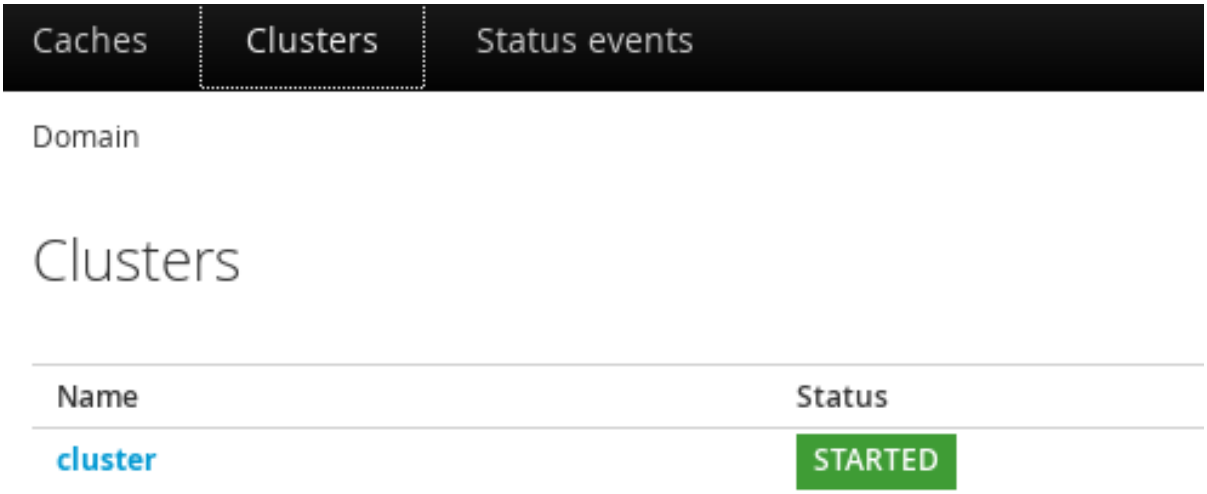


図24.92 クラスターの選択

3. 「**Add Node**」をクリックします。

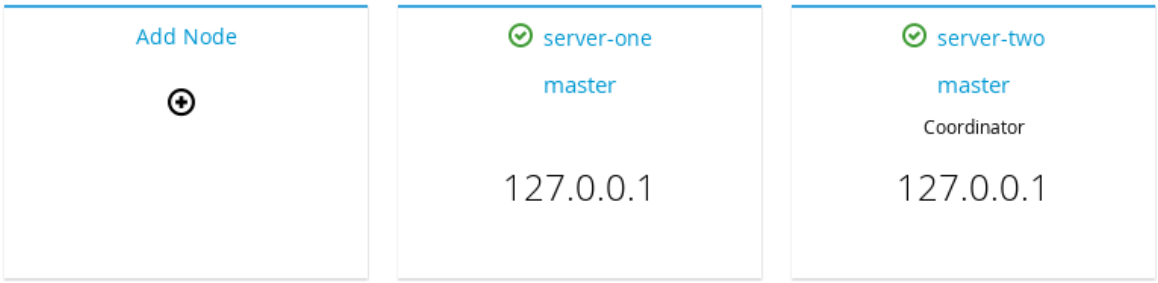


図24.93 作成されたノードの追加

4. ノード設定ウィンドウが開かれます。ノードプロパティーをそれぞれのフィールドに入力して「**Create**」をクリックします。

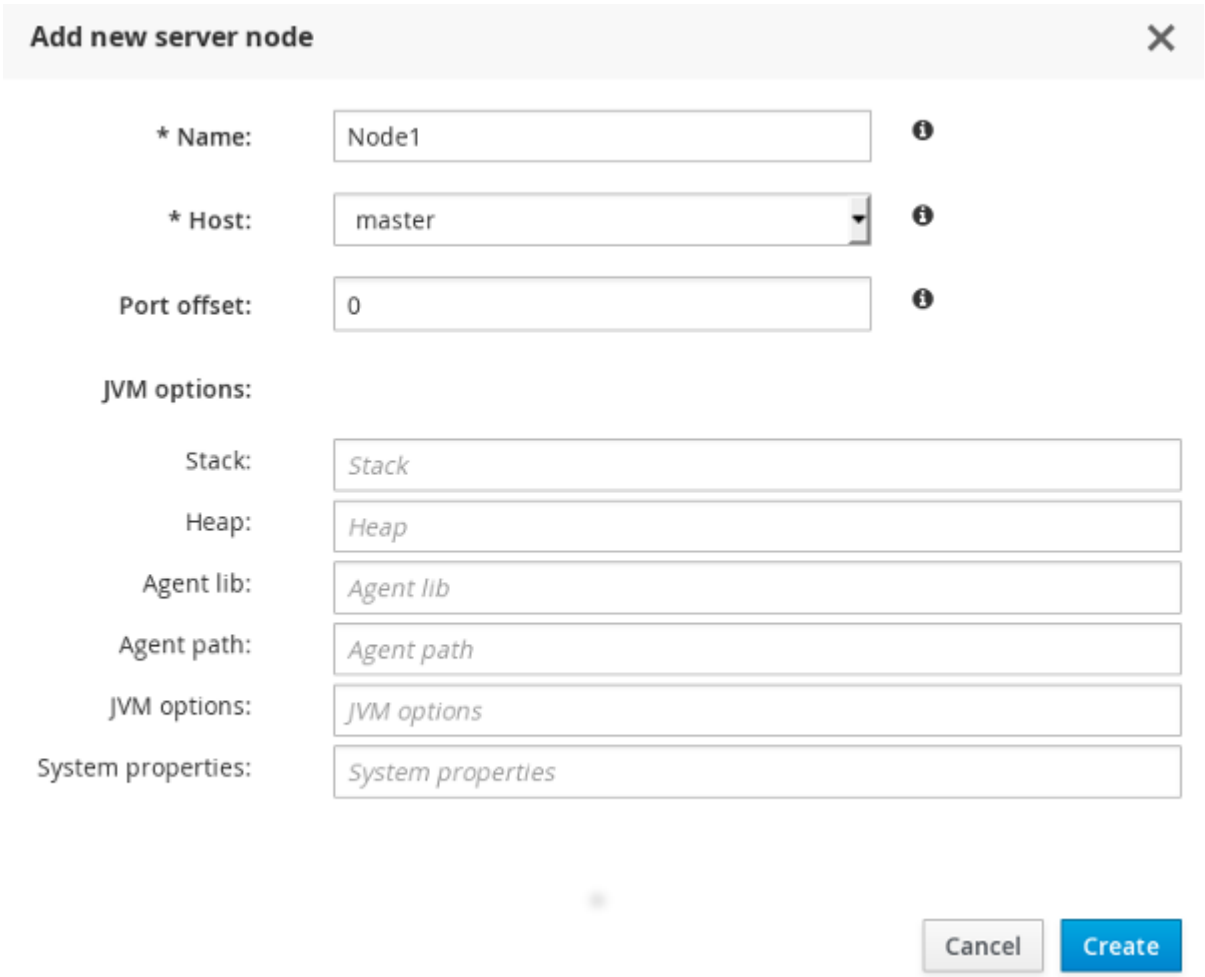


図24.94 ノードプロパティー

5. システムが起動します。

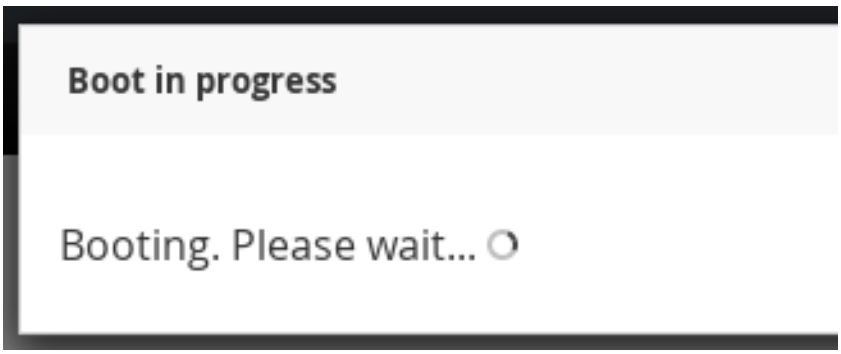


図24.95 システムの起動

6. 新規ノードが正常に作成されます。

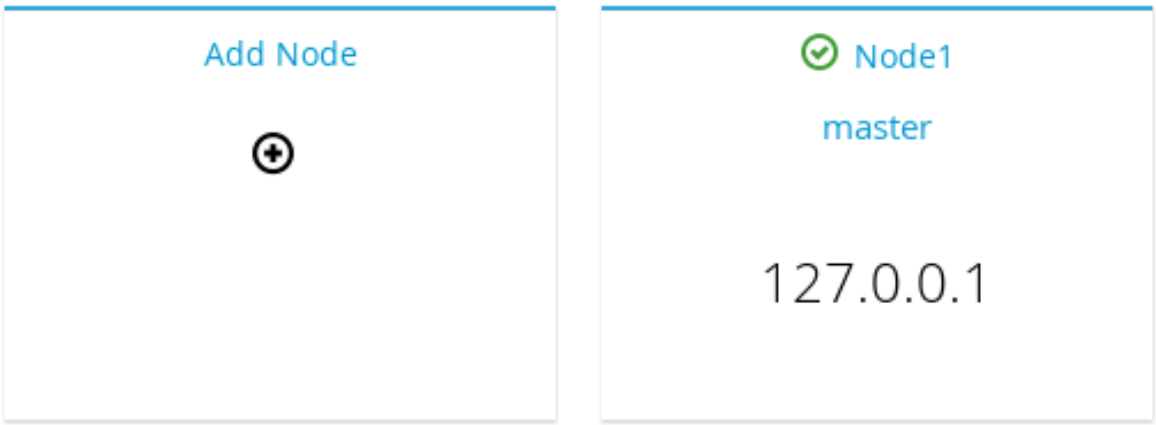


図24.96 新規ノード

バグを報告する

24.7.7. ノード統計およびプロパティービュー

JBoss Data Grid 管理コンソールにより、ユーザーは読み取りの平均時間、書き込みの平均時間、エントリーの合計数、読み取りの合計数、失敗した読み取りの合計数、書き込みの合計数およびその他のデータを表示できます。

ノード統計を表示するには、JBoss Data Grid 管理コンソールの「Clusters」タブでノードの名前をクリックします。

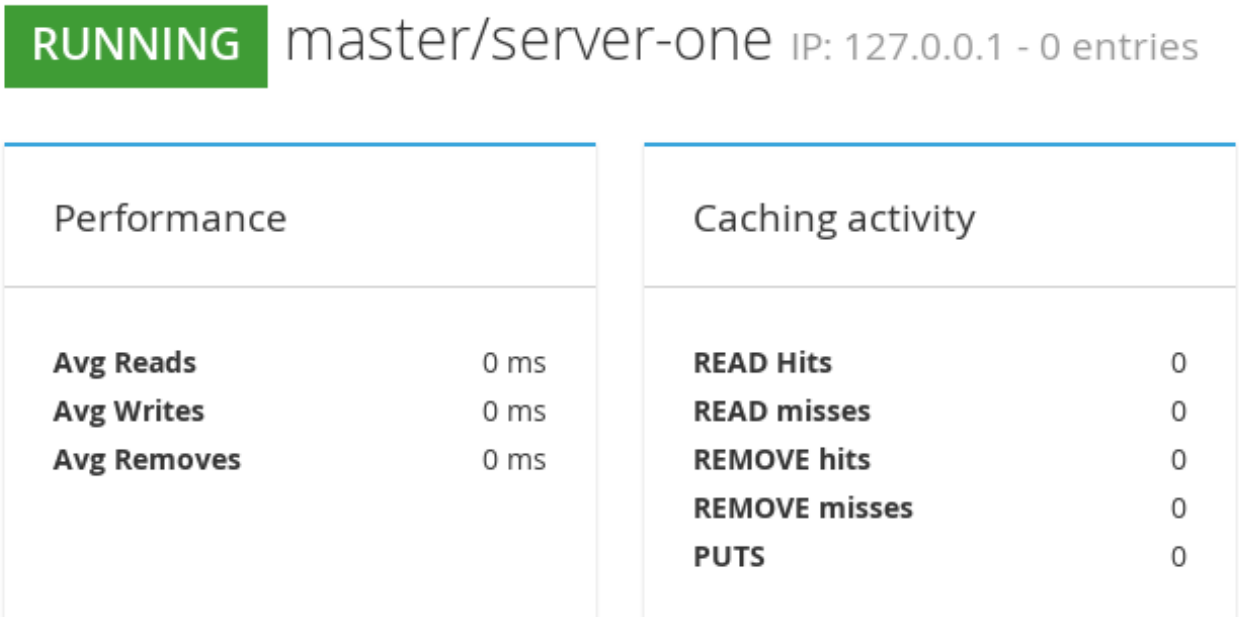


図24.97 ノード統計

バグを報告する

24.7.8. ノードパフォーマンスメトリックスビュー

ノードパフォーマンスメトリックスを表示するには、JBoss Data Grid 管理コンソールの「Clusters」タブでノードの名前をクリックします。

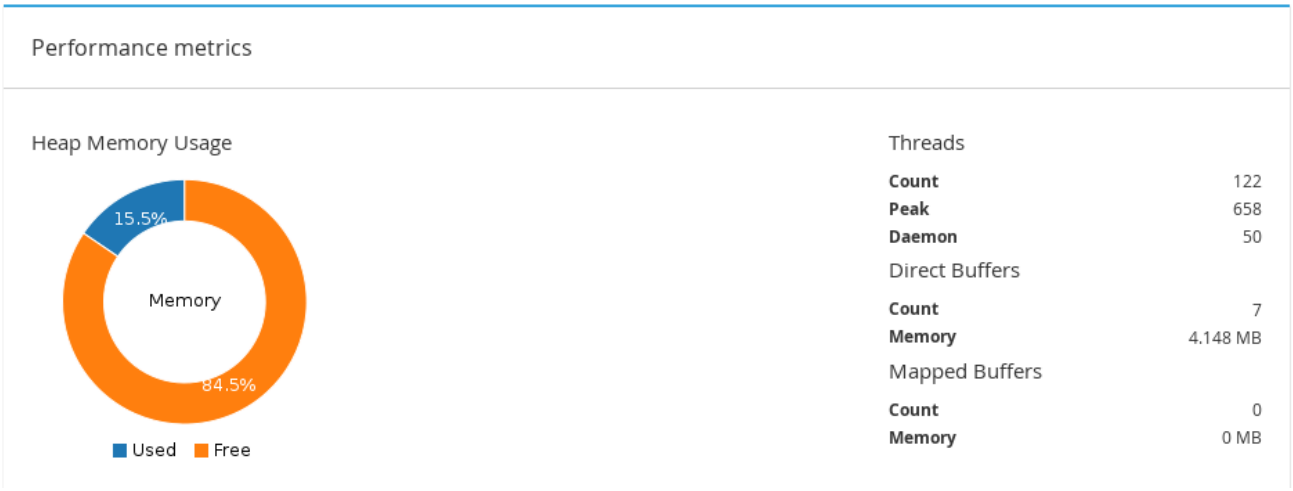


図24.98 ノードパフォーマンスメトリックス

バグを報告する

24.7.9. ノードの無効化

JBoss Data Grid 管理コンソールを使用すると、管理者はノードを無効にすることができます。

クラスターのノードを無効にするには、以下の手順を実行します。

手順24.17 新規ノードの追加

1. JBoss Data Grid 管理コンソールのクラスタービューでクラスターの名前をクリックします。
2. ノードビューで、無効にするノードをクリックします。

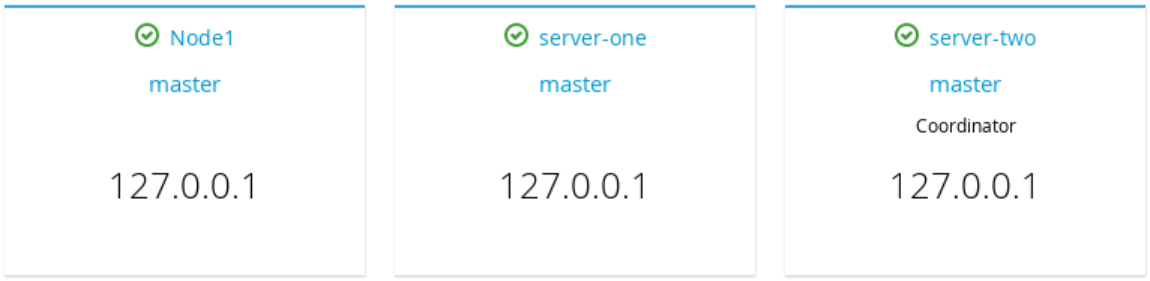


図24.99 ノードビュー

3. ノード統計ビューが開かれます。ページ右上にある「**Actions**」タブをクリックしてから「**Stop**」をクリックします。

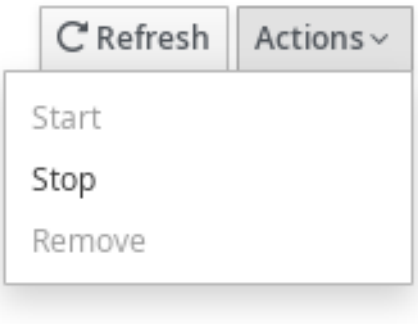


図24.100 ノードの停止

4. 確認するためのボックスが表示されます。「**Stop**」をクリックしてクラスターからノードを削除します。

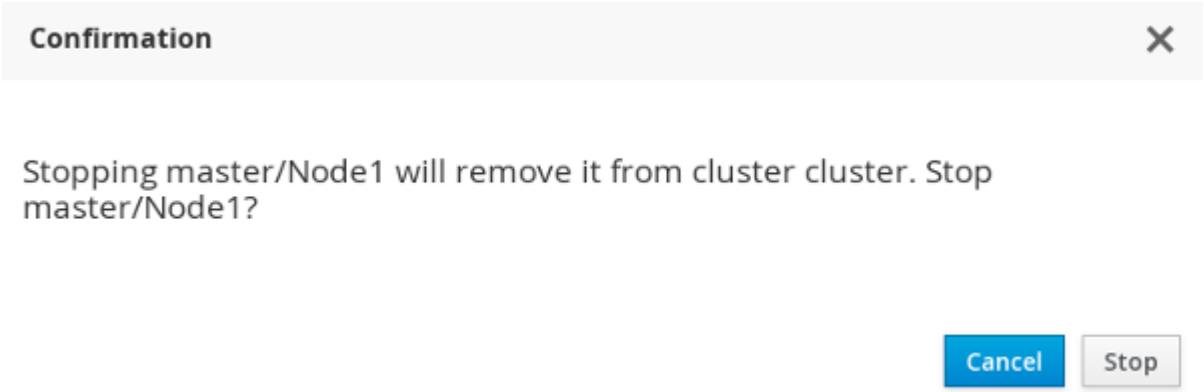


図24.101 確認ボックス

バグを報告する

24.7.10. クラスターのシャットダウンおよび再起動

バグを報告する

24.7.10.1. クラスターのシャットダウン

JBoss Data Grid 管理コンソールにより、JBoss Data Grid クラスターをシャットダウンするための保守用の便利かつ制御された方法を利用できます。キャッシュストアが設定されているキャッシュの場合、データは損失することなく永続化します。キャッシュストアが設定されていないキャッシュの場合、データはクラスターのシャットダウン後に失われます。

クラスターをシャットダウンまたは停止するには、以下の手順を実行します。

手順24.18 クラスターのシャットダウン

1. JBoss Data Grid 管理コンソールのクラスタービューに移動して、クラスターの名前をクリックします。

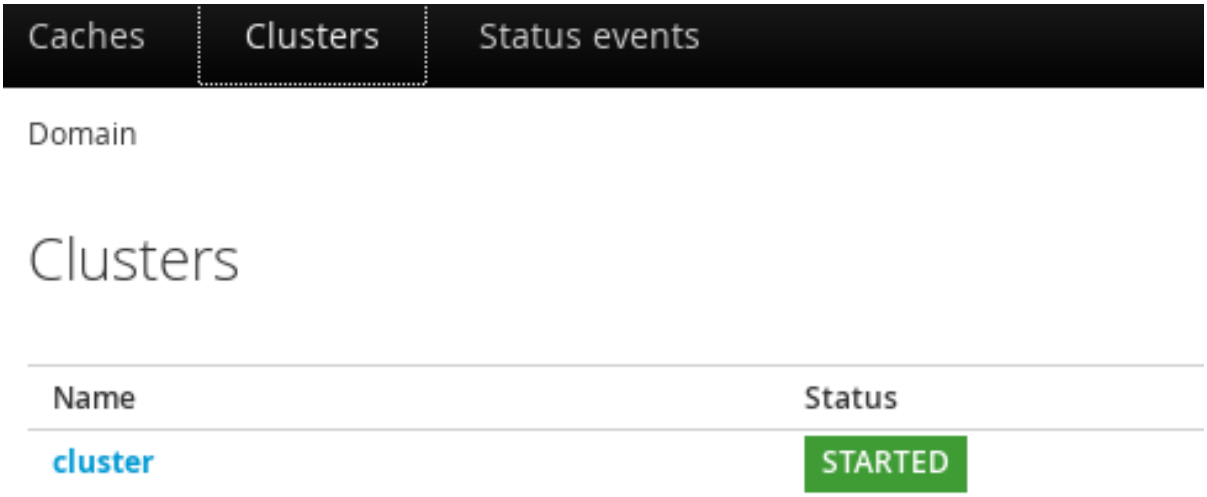


図24.102 クラスタービュー

2. ノードビューのページのインターフェースの右上に「**Actions**」タブがあります。「**Actions**」タブをクリックしてから「**Stop**」をクリックします。

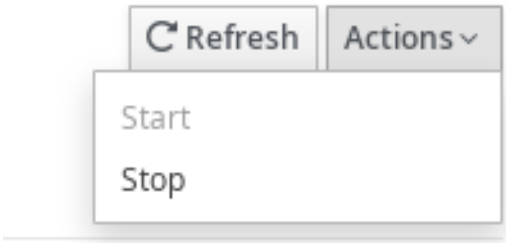


図24.103 クラスターの停止

3. 確認のためのボックスが表示されます。確認するには、「**Stop**」をクリックします。

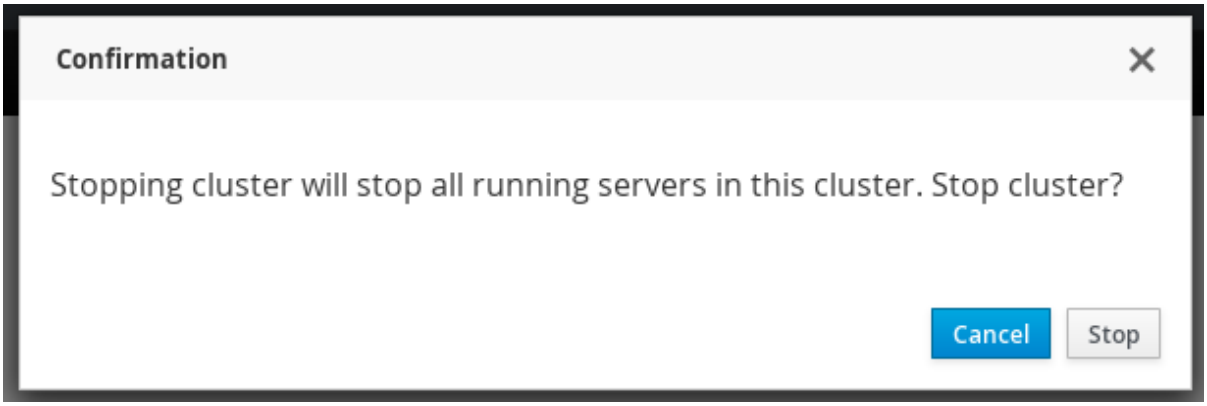


図24.104 確認ボックス

バグを報告する

24.7.10.2. クラスターの起動

JBoss Data Grid 管理コンソールにより、停止したクラスターを再起動できます。キャッシュデータは、キャッシュストアが設定されているキャッシュの場合にはデータが失われることなくプリロード

されます。キャッシュストアが設定されていないキャッシュには初期の段階ではデータが含まれません。

プリロードは、これがキャッシュストアで有効にされている場合にのみ実行されます。ノードのいずれかのローカルキャッシュの状態が破損している場合、キャッシュは開始されず、手動の操作が必要になります。

クラスターに対して、以下の手順を実行します。

手順24.19 クラスターの起動

1. JBoss Data Grid 管理コンソールの「Clusters」ビューに移動し、クラスターの名前をクリックします。
2. ノードビューのページのインターフェース右上に「Actions」タグがあります。「Actions」タグをクリックしてから「`Start`」をクリックします。

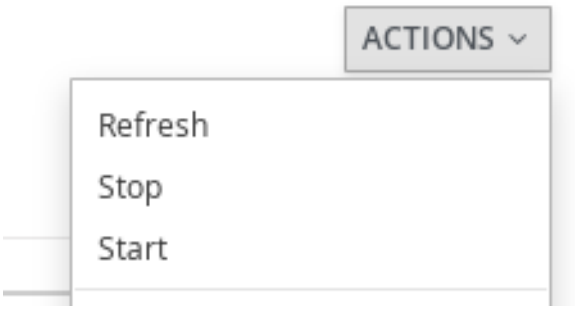


図24.105 クラスターの起動

3. 確認のためのボックスが表示されます。「Start」をクリックしてクラスターを起動します。

バグを報告する

パート XII. RED HAT JBOSS DATA GRID におけるデータのセキュア化

Red Hat JBoss Data Grid では、データセキュリティーを以下の方法で実装できます。

ロールベースアクセス制御

JBoss Data Grid には、指定のセキュア化されたキャッシュ上の操作に対応するロールベースのアクセス制御機能が含まれます。ロールは、キャッシュおよびキャッシュマネージャー操作のパーミッションにマップされた状態で、アプリケーションにアクセスするユーザーに割り当てられます。認証されたユーザーのみがこれらのロールで許可されている操作を実行できます。

ライブラリーモードでは、データは、認証がコンテナーまたはアプリケーションに委譲された状態で CacheManager および Cache 用のロールベースのアクセス制御によってセキュア化されます。リモートクライアントサーバーモードでは、JBoss Data Grid は、Identity トークンを Hot Rod クライアントから、Cache および CacheManager のサーバーおよびロールベースのアクセス制御に渡すことによってセキュア化されます。

ノードの認証および承認

ノードレベルのセキュリティーには、新規ノードやマージされるパーティションがクラスターに参加する前に認証される必要があります。クラスターに参加することが許可される認証されたノードのみがこれを実行できます。これにより、許可されたサーバーがデータを保存することを防ぎ、データが保護されます。

クラスター内の暗号化通信

JBoss Data Grid では、JCA (Java Cryptography Architecture、Java 暗号化アーキテクチャー) によってサポートされるユーザー指定の暗号化アルゴリズム使用して、ノード間での通信の暗号化がサポートされるようになり、データのセキュリティーが強化されました。

また、JBoss Data Grid は操作の監査ログ、および Transport Layer Security (TLS/SSL) を使用した Hot Rod クライアントとサーバー間の通信を暗号化する機能も提供します。

バグを報告する

第25章 RED HAT JBOSS DATA GRID セキュリティー: 認証および承認

25.1. RED HAT JBOSS DATA GRID セキュリティー: 認証および承認

Red Hat JBoss Data Grid は、CacheManager および Cache での承認を実行できます。JBoss Data Grid 承認は、JAAS および SecurityManager などの JDK で利用できる標準的なセキュリティー機能に基づいています。

アプリケーションがセキュリティーが保護された CacheManager および Cache との対話を試行する場合、JBoss Data Grid のセキュリティーレイヤーが必須ロールおよびパーミッションのセットに対して検証できるアイデンティティーを指定する必要があります。検証後にはクライアントに後続の操作のためのトークンが発行されます。アクセスが拒否されると、セキュリティー違反を示す例外がスローされます。

キャッシュの承認が設定される場合、キャッシュが取得されると **SecureCache** のインスタンスが返されます。**SecureCache** はキャッシュに関する単純なラッパーであり、「現行ユーザー」が操作の実行に必要なパーミッションを持つかどうかをチェックします。「現行ユーザー」は **AccessControlContext** に関連付けられた「Subject」になります。

JBoss Data Grid はプリンシパル名を、1 つ以上のパーミッションを表すロールにマップします。以下の図はこれらの関係を示しています。

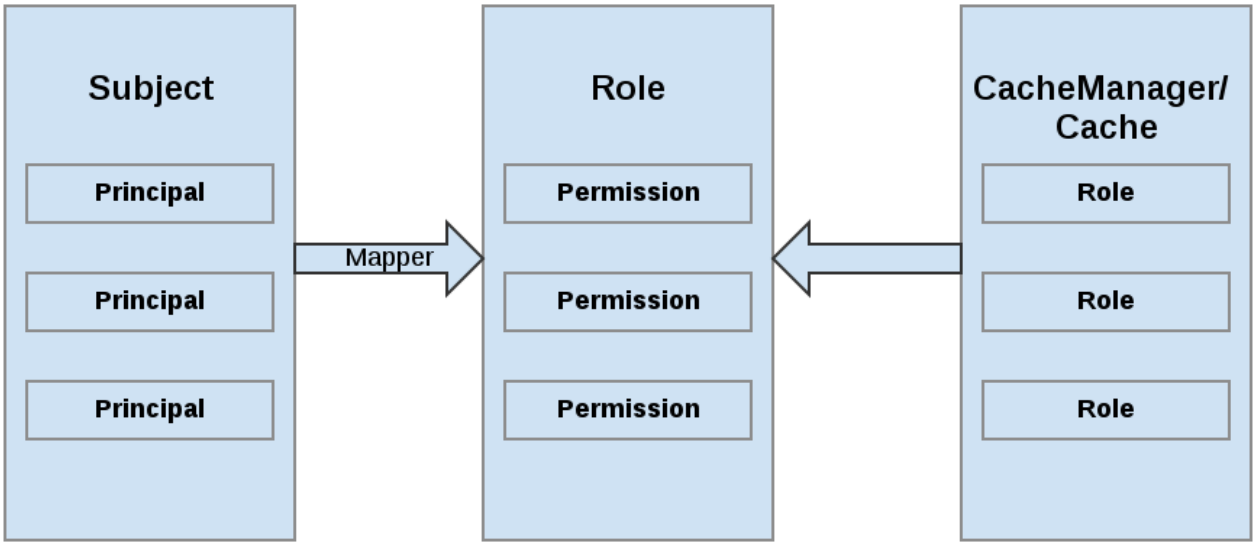


図25.1 ロールとパーミッションのマッピング

バグを報告する

25.2. パーミッション

CacheManager または Cache へのアクセスは必要なパーミッションセットを使用して制御されます。パーミッションは、操作されているデータのタイプではなく、CacheManager または Cache で実行されるアクションのタイプを制御します。これらのパーミッションの一部は、名前付きキャッシュなどの名前エンティティーにとくに適用されます。エンティティーによって使用できるタイプのパーミッションが異なります。

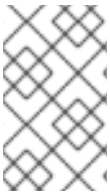
表25.1 CacheManager パーミッション

パーミッション	関数	説明
CONFIGURATION	defineConfiguration	新規キャッシュ設定を定義できるかどうか。
LISTEN	addListener	リスナーをキャッシュマネージャーに対して登録できるかどうか。
LIFECYCLE	stop, start	キャッシュマネージャーを停止または開始できるかどうか。
ALL		上記すべてを含む便利なパーミッション。

表25.2 キャッシュパーミッション

パーミッション	関数	説明
READ	get, contains	エントリーをキャッシュから取得できるかどうか。
WRITE	put, putIfAbsent, replace, remove, evict	データをキャッシュから書き込み、置き換え、エビクトできるかどうか。
EXEC	distexec, mapreduce	コード実行をキャッシュに対して実行できるかどうか。
LISTEN	addListener	リスナーをキャッシュに対して登録できるかどうか。
BULK_READ	keySet, values, entrySet,query	一括取得操作を実行できるかどうか。
BULK_WRITE	g	一括書き込み操作を実行できるかどうか。
LIFECYCLE	start, stop	キャッシュを開始または停止できるかどうか。

パーミッション	関数	説明
ADMIN	getVersion, addInterceptor*, removeInterceptor, getInterceptorChain, getEvictionManager, getComponentRegistry, getDistributionManager, getAuthorizationManager, evict, getRpcManager, getCacheConfiguration, getCacheManager, getInvocationContextContainer, setAvailability, getDataContainer, getStats, getXAResource	基礎となるコンポーネントまたは内部構造へのアクセスが可能かどうか。
ALL		上記すべてを含む便利なパーミッション。
ALL_READ		READ と BULK_READ の組み合わせ。
ALL_WRITE		WRITE と BULK_WRITE の組み合わせ。



注記

使いやすさを強化するために一部のパーミッションを他のパーミッションに組み合わせることが必要になる場合があります。たとえば、EXEC を READ または WRITE と組み合わせることがあります。

バグを報告する

25.3. ロールマッピング

「Subject」内のプリンシパルを承認で使用される一連のロールに変換するには、**`PrincipalRoleMapper`** をグローバル設定に指定する必要があります。Red Hat JBoss Data Grid では 3 つのマッパーが同梱されており、さらにカスタムマッパーを使用することもできます。

表25.3 マッパー

マッパー名	Java	XML	説明
IdentityRoleMapper	org.infinispan.security.impl.IdentityRoleMapper	<identity-role-mapper />	ロール名としてプリンシパル名を使用します。

マッパー名	Java	XML	説明
CommonNameRoleMapper	org.infinispan.security.impl.CommonRoleMapper	<common-name-role-mapper />	プリンシパル名が識別名 (DN) の場合、このマッパーは共通名 (CN) を抽出し、これをロール名として使用します。たとえば、DN **cn=managers,ou=people,dc=example,dc=com** はロールの **managers** にマップされます。
ClusterRoleMapper	org.infinispan.security.impl.ClusterRoleMapper	<cluster-role-mapper />	**ClusterRegistry** を使用してプリンシパルをロールマッピングに保存します。これにより、CLI の **GRANT** および **DENY** コマンドを使用してロールのプリンシパルへの追加または削除を実行できます。
Custom Role Mapper		<custom-role-mapper class="a.b.c" />	**org.infinispan.security.impl.PrincipalRoleMapper** の実装の完全修飾クラス名を指定します。

バグを報告する

25.4. ログインモジュールを使用した認証およびロールマッピングの設定

LDAP からロールのクエリーを実行するために認証 ***login-module*** を使用する場合、カスタムクラスが使用されているため、プリンシパルからロールへの独自のマッピングを実装する必要があります。この変換の実装例は the JBoss Data Grid の 『Developer Guide』 を参照してください。以下は、宣言的な設定例です。

例25.1 LDAP ログインモジュールの設定例

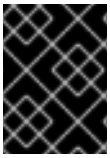
```
 <security-domain name="ispn-secure" cache-type="default">
    <authentication>
        <login-module
code="org.jboss.security.auth.spi.LdapLoginModule" flag="required">
            <module-option name="java.naming.factory.initial"
value="com.sun.jndi.ldap.LdapCtxFactory"/>
            <module-option name="java.naming.provider.url"
value="ldap://localhost:389"/>
            <module-option name="java.naming.security.authentication"
value="simple"/>
            <module-option name="principalDNPrefix" value="uid="/>
            <module-option name="principalDNSuffix"
```

```
value=",ou=People,dc=infinispan,dc=org"/>
            <module-option name="rolesCtxDN"
value="ou=Roles,dc=infinispan,dc=org"/>
            <module-option name="uidAttributeID" value="member"/>
            <module-option name="matchOnUserDN" value="true"/>
            <module-option name="roleAttributeID" value="cn"/>
            <module-option name="roleAttributeIsDN" value="false"/>
            <module-option name="searchScope" value="ONELEVEL_SCOPE"/>
        </login-module>
    </authentication>
</security-domain>
```

例25.2 ログインモジュールの設定例

```
<security-domain name="krb-admin" cache-type="default">
    <authentication>
        <login-module code="Kerberos" flag="required">
            <module-option name="useKeyTab" value="true"/>
            <module-option name="principal"
value="admin@INFINISPAN.ORG"/>
            <module-option name="keyTab"
value="${basedir}/keytab/admin.keytab"/>
        </login-module>
    </authentication>
</security-domain>
```

GSSAPI 認証を使用する場合、通常 JBoss Data Grid Server は LDAP サーバーに対して GSSAPI 経由で認証され、ロールマッピングには LDAP が使用されます。この認証の Active Directory サーバーに対する設定例については、「Kerberos を使用した Active Directory 認証 (GSSAPI)」 を参照してください。



重要

LDAP サーバーの設定または LDAP サーバーでのユーザーとロールの指定についての詳細は、Red Hat Directory Server の 『Administration Guide』 を参照してください。

バグを報告する

25.5. RED HAT JBOSS DATA GRID の承認の設定

承認はキャッシュコンテナー (CacheManager) と単一キャッシュの 2 つのレベルで設定されます。

CacheManager

以下は、CacheManager レベルの承認の設定例です。

例25.3 CacheManager 承認 (宣言的な設定)

```
<cache-container name="local" default-cache="default">
        <security>
           <authorization>
             <identity-role-mapper />
```

```
            <role name="admin" permissions="ALL"/>
            <role name="reader" permissions="READ"/>
            <role name="writer" permissions="WRITE"/>
            <role name="supervisor" permissions="ALL_READ ALL_WRITE"/>
         </authorization>
      </security>
</cache-container>
```

各キャッシュコンテナーは以下を決定します。

- 承認を使用するかどうか。
- プリンシプルをルールセットにマップするクラス。
- 名前付きロールのセットとそれらが表すパーミッション。

コンテナーレベルで定義されたロールのサブセットのみを使用することを選択できます。

ロール

ロールは、以下のようにキャッシュコンテナーのレベルで定義されたロールを使用して、キャッシュごとに適用できます。

例25.4 ロールの定義

```
<local-cache name="secured">
  <security>
    <authorization roles="admin reader writer supervisor"/>
  </security>
</local-cache>
```

重要

認証が必要とされるキャッシュには、ロールの一覧が定義されている必要があります。そうでない場合、キャッシュの承認では **no-anonymous** (非匿名) ポリシーは定義されないため、認証は施行されません。

重要

REST プロトコルは、承認で使用するためにサポートされておらず、承認を有効にした状態でのキャッシュへのアクセスを試行すると **SecurityException** が生じます。

バグを報告する

25.6. SECURITYMANAGER を使用した承認

Red Hat JBoss Data Grid のリモートクライアントサーバーモードでは、承認は基本的なキャッシュ操作の場合に **SecurityManager** がなくても機能します。ライブラリーモードでは、**SecurityManager** を使用して、とくに **distexec** およびクエリーなどのより複雑なタスクの一部を実行することもできます。

アクセス制限を施行するには、以下のメソッドにいずれかを使用して JVM で **SecurityManager** を有効にします。

コマンドライン

```
java -Djava.security.manager ...
```

プログラミングの使用

```
System.setSecurityManager(new SecurityManager());
```

JDK のデフォルト実装を使用する必要はありません。ただし、適切なポリシーファイルを使用する必要があります。ポリシーファイルは、アプリケーションがアクションを実行する際に **SecurityManager** が調査するパーミッションのセットを定義します。アクションがポリシーファイルで許可される場合、**SecurityManager** がアクションの実行を許可しますが、アクションがポリシーで許可されない場合は **SecurityManager** はそのアクションを拒否します。

必要な構文を示すポリシーファイルの例は以下のとおりです。

```
// If the code is signed by "admin", grant it read/write access to all
files
grant signedBy "admin" {
    permission java.io.FilePermission "/*", "read,write";
};

// Grant everyone read permissions on specific environment variables:
grant {
    permission java.util.PropertyPermission "java.home", "read";
    permission java.util.PropertyPermission "java.class.path", "read";
    permission java.util.PropertyPermission "java.vendor", "read";
};

// Grant a specific codebase, example.jar, read and write access to
"/tmp/*"
grant codeBase "file:///path/to/example.jar" {
    permission java.io.FilePermission "/tmp/*", "read,write";
};
```

バグを報告する

25.7. JAVA のセキュリティーマネージャー

25.7.1. Java Security Manager

Java Security Manager

> Java Security Manager は、Java 仮想マシン (JVM) サンドボックスの外部の境界を管理するクラスで、JVM 内で実行しているコードと JVM 外のリソースとの連携方法を制御します。Java Security Manager がアクティブになると、Java API は安全でない多様な動作を実行する前に Security Manager と承認を確認します。

Java Security Manager は、セキュリティーポリシーを使用して、特定のアクションが許可または拒否されるかどうかを決定します。

バグを報告する

25.7.2. Java Security Manager のポリシー

セキュリティーポリシー

> コードの様々なクラスに対して定義されたパーミッションのセットです。Java Security Manager はセキュリティーポリシーとアプリケーションから要求されたアクションを比較します。ポリシーがアクションを許可している場合は、Java Security Manager はそのアクションが行われることを許可します。ポリシーがアクションを許可していない場合は、Java Security Manager はそのアクションを拒否します。セキュリティポリシーは、コードの場所またはコードのシグニチャー、またはサブジェクトのプリンシパルに基づいてパーミッションを定義できます。

使用する Java Security Manager とセキュリティーポリシーは、Java 仮想マシンのオプション **`java.security.manager`** や **`java.security.policy`** を使用して設定されます。

基本情報

セキュリティーポリシーのエントリーは、**`policytool`** に関係のある以下の設定要素から構成されています。

CodeBase

コードの元の URL の場所 (ホストとドメインの情報以外)。オプションのパラメーターです。

SignedBy

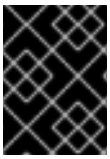
コードを署名するためにプライベートキーが使用された署名者を参照するキーストアで使用されたエイリアス。これは、単一値またはカンマ区切りの値リストになります。オプションのパラメーターです。省略された場合は、署名の有無に関わらず Java Security Manager に影響はありません。

Principals

`principal_type` と ***`principal_name`*** のペアのリスト。これは、実行スレッドのプリンシパルセット内に存在する必要があります。Principals エントリーは任意です。このエントリーを省略すると、実行スレッドのプリンシパルによる Java Security Manager への影響はありません。

Permissions

パーミッションは、コードに与えられるアクセス権です。多くのパーミッションは、Java Enterprise Edition 6 (Java EE 6) 仕様の一部として提供されます。本書では、JBoss EAP 6 で提供される追加のパーミッションについてのみ説明します。



重要

実装によって設定が異なる可能性があるため、セキュリティーポリシーの設定方法についてコンテナーのドキュメントを参照してください。

バグを報告する

25.7.3. Java Security Manager ポリシーの記述

はじめに

ほとんどの JDK および JRE ディストリビューションには、Java Security Manager セキュリティーポ

リシーを作成および編集するための **policytool** という名前のアプリケーションが含まれます。**policytool** の詳細については、http://docs.oracle.com/javase/6/docs/technotes/tools/ を参照してください。

手順25.1 新しい Java Security Manager ポリシーの設定

1. **policytool を起動します。**
   policytool ツールを次のいずれかの方法で起動します。
   - **Red Hat Enterprise Linux**
     GUI またはコマンドプロンプトで、**/usr/bin/policytool** を実行します。
   - **Microsoft Windows Server**
     スタートメニューまたは Java インストールの **bin** から、**policytool.exe** を実行します。場所は異なることがあります。
2. **ポリシーを作成します。**
   ポリシーを作成するには、**Add Policy Entry** を選択します。必要なパラメーターを追加し、**Done** をクリックします。
3. **既存のポリシーを編集します。**
   既存のポリシーのリストからポリシーを選択し、**Edit Policy Entry** ボタンを選択します。必要に応じてパラメーターを編集します。
4. **既存のポリシーを削除します。**
   既存のポリシーのリストからポリシーを選択し、**Remove Policy Entry** ボタンを選択します。

バグを報告する

25.7.4. Java Security Manager 内での Red Hat JBoss Data Grid Server の実行

Java Security Manager ポリシーを指定するには、ブートストラッププロセス中にサーバーインスタンスに渡す Java オプションを編集する必要があります。このため、**standalone.sh** スクリプトにパラメーターをオプションとして渡すことはできません。次の手順を実行すると、インスタンスが Java Security Manager ポリシー内で実行されるように設定できます。

前提条件

- この手順を実行する前に、Java Development Kit (JDK) に含まれる **policytool** コマンドを使用してセキュリティーポリシーを記述する必要があります。この手順では、ポリシーが ***JDG_HOME*/bin/server.policy** にあることを前提としています。この代わりに、テキストエディターを使用してセキュリティーポリシーを書き、***JDG_HOME*/bin/server.policy** として手作業で保存することもできます。
- 設定ファイルを編集する前に、JBoss Data Grid サーバーを完全に停止する必要があります。

環境内の各物理ホストまたはインスタンスに対して次の手順を実行してください。

手順25.2 JBoss Data Grid Server の Security Manager の設定

1. **設定ファイルを開きます。**

編集のために設定ファイルを開きます。このファイルの場所は OS で以下のように一覧表示されます。これはサーバーを起動するために使用される実行可能ファイルではなく、ランタイムパラメーターを含む設定ファイルであることに注意してください。

- Linux の場合: ***JDG_HOME*/bin/standalone.conf**
- Windows の場合: ***JDG_HOME*\bin\standalone.conf.bat**

2. **ファイルに Java オプションを追加します。**
確実に Java オプションが使用されるようにするため、以下で始まるコードブロックに Java オプションを追加します。

```
if [ "x$JAVA_OPTS" = "x" ]; then
```

-Djava.security.policy の値を編集して、セキュリティーポリシーの場所を指定できます。改行を入れずに 1 行で指定する必要があります。**-Djava.security.policy** プロパティーを設定するときに **==** を使用して指定すると、セキュリティーマネージャーは指定されたポリシーファイルのみを使用します。**=** を使用して指定すると、セキュリティーマネージャーは指定されたポリシーと ***JAVA_HOME*/lib/security/java.security** の **policy.url** セクションに指定されたポリシーを一緒に使用します。

重要

JBoss Enterprise Application Platform 6.2.2 およびそれ以降のリリースでは、システムプロパティー **jboss.modules.policy-permissions** を **true** に設定する必要があります。

例25.5 standalone.conf

```
JAVA_OPTS="$JAVA_OPTS -Djava.security.manager -
Djava.security.policy==$PWD/server.policy -
Djboss.home.dir=$JBOSS_HOME -Djboss.modules.policy-
permissions=true"
```

例25.6 standalone.conf.bat

```
set "JAVA_OPTS=%JAVA_OPTS% -Djava.security.manager -
Djava.security.policy==\path\to\server.policy -
Djboss.home.dir=%JBOSS_HOME% -Djboss.modules.policy-
permissions=true"
```

3. **サーバーを起動します。**
サーバーを通常どおりに起動します。

バグを報告する

25.8. リモートクライアントサーバーモードのデータセキュリティー

25.8.1. セキュリティーレルム

セキュリティーレルム はユーザーとパスワード間、およびユーザーとロール間のマッピングです。セキュリティーレルムは EJB や Web アプリケーションに認証や承認を追加するメカニズムです。Red Hat JBoss Data Grid サーバーはデフォルトで次の 2 つのセキュリティーレルムを提供します。

- **ManagementRealm** は、管理 CLI や Web ベースの管理コンソールに機能を提供する管理 API の認証情報を保存します。これは、JBoss Data Grid Server を管理するための認証システムを提供します。管理 API に使用する同じビジネスルールでアプリケーションを認証する必要がある場合には、**ManagementRealm** を使用することもできます。
- **ApplicationRealm** は Web アプリケーションと EJB のユーザー、パスワード、およびロール情報を保存します。

各レルムはファイルシステム上の 2 つのファイルに保存されます。

- ***REALM*-users.properties** はユーザー名とハッシュ化されたパスワードを保存します。
- ***REALM*-roles.properties** はユーザー対ロールのマッピングを保存します。
- **mgmt-groups.properties** は **ManagementRealm** のユーザー対ロールのマッピングを保存します。

プロパティーファイルは **standalone/configuration/** ディレクトリーに保存されます。ファイルは **add-user.sh** や **add-user.bat** コマンドによって同時に書き込まれます。コマンドの実行時、新しいユーザーをどのレルムに追加するかを最初に決定します。

バグを報告する

25.8.2. 新しいセキュリティーレルムの追加

1. **管理 CLI を実行します。**
   cli.sh または **cli.bat** コマンドを開始し、サーバーに接続します。

2. **新しいセキュリティーレルムを作成します。**
   次のコマンドを実行し、ドメインコントローラーまたはスタンドアロンサーバー上で **MyDomainRealm** という名前の新しいセキュリティーレルムを作成します。

```
/host=master/core-service=management/security-
realm=MyDomainRealm:add()
```

3. **新しいレルムのユーザーの情報を保存するプロパティーファイルへの参照を作成します。**
   以下のコマンドを実行して、新規セキュリティーレルムのプロパティーファイルの場所を定義します。このファイルには、このセキュリティーレルムのユーザーについての情報が含まれます。以下のコマンドは、**jboss.server.config.dir** 内の **myfile.properties** という名前のファイルを参照します。

注記

新たに作成されたプロパティーファイルは、含まれる **add-user.sh** および **add-user.bat** スクリプトによって管理されません。そのため、外部から管理する必要があります。

```
/host=master/core-service=management/security-
realm=MyDomainRealm/authentication=properties:add(path="myfile.prope
rties",relative-to="jboss.server.config.dir")
```

4. **サーバーの再ロード**
   変更を反映するためにサーバーをリロードします。

```
:reload
```

結果

新しいセキュリティーレルムが作成されます。新たに作成されたこのレルムにユーザーやロールを追加すると、デフォルトのセキュリティーレルムとは別のファイルに情報が保存されます。新規ファイルはご使用のアプリケーションやプロシージャーを使用して管理できます。

バグを報告する

25.8.3. セキュリティーレルムへのユーザーの追加

1. **`add-user.sh` または `add-user.bat` コマンドを実行します。**
   ターミナルを開き、***`JDG_HOME`*`/bin/`** ディレクトリーへ移動します。Red Hat Enterprise Linux や他の UNIX 系のオペレーティングシステムを稼働している場合は、**`add-user.sh`** を実行します。Microsoft Windows Server を稼働している場合は **`add-user.bat`** を実行します。

2. **管理ユーザーかアプリケーションユーザーのどちらを追加するか選択します。**
   この手順では **b** を入力し、アプリケーションユーザーを追加します。

3. **ユーザーが追加されるレルムを選択します。**
   デフォルトでは、**`ManagementRealm`** および **`ApplicationRealm`** のみが選択可能です。ただし、カスタムレルムが追加されている場合はその名前を入力します。

4. **入力を促されたらユーザー名、パスワード、ロールを入力します。**
   入力を促されたら希望のユーザー名、パスワード、任意のロールを入力します。**yes** を入力して選択を確認するか、**no** を入力して変更をキャンセルします。変更はセキュリティーレルムの各プロパティーファイルに書き込まれます。

バグを報告する

25.8.4. セキュリティーレルムの宣言的な有効化

リモートクライアントサーバーモードでは、Hot Rod エンドポイントはセキュリティーレルムを指定する必要があります。

セキュリティーレルムは、***`authentication`*** および ***`authorization`*** セクションを宣言します。

例25.7 セキュリティーレルムの宣言的な有効化

```
<security-realms>
            <security-realm name="ManagementRealm">
                <authentication>
                    <local default-user="$local" skip-group-
loading="true"/>
                    <properties path="mgmt-users.properties" relative-
to="jboss.server.config.dir"/>
                </authentication>
                <authorization map-groups-to-roles="false">
                    <properties path="mgmt-groups.properties" relative-
```

```
to="jboss.server.config.dir"/>
                </authorization>
            </security-realm>
            <security-realm name="ApplicationRealm">
                <authentication>
                    <local default-user="$local" allowed-users="*"
skip-group-loading="true"/>
                    <properties path="application-users.properties"
relative-to="jboss.server.config.dir"/>
                </authentication>
                <authorization>
                    <properties path="application-roles.properties"
relative-to="jboss.server.config.dir"/>
                </authorization>
            </security-realm>
        </security-realms>
```

server-identities パラメーターも証明書を指定するために使用できます。

バグを報告する

25.8.5. 承認のための LDAP からのロールのロード (リモートクライアントサーバーモード)

LDAP ディレクトリーには、属性によって相互参照されるユーザーアカウントとグループのエントリーが含まれます。LDAP サーバーの設定によっては、**memberOf** 属性を用いてユーザーエンティティーがユーザーが属するグループをマップしたり、 **uniqueMember** 属性を用いてグループエンティティーが属するユーザーをマップしたりすることがあります。また、両方のマッピングが LDAP サーバーによって維持されることもあります。

通常、ユーザーは簡単なユーザー名を使用してサーバーに対して認証を行います。グループメンバーシップ情報を検索する場合、使用中のディレクトリーサーバーに応じて、検索がこの単純名を使用して実行されたり、ディレクトリーのユーザーエントリーの識別名を使用して実行されたりします。

必ず最初に、サーバーへ接続するユーザーを認証する手順が実行されます。ユーザーの認証に成功した後、サーバーはサーバーグループをロードします。認証手順と承認手順はそれぞれ LDAP サーバーへの接続が必要になります。レルムはグループをロードする手順の認証接続を再使用して、このプロセスを最適化します。以下の設定手順のとおり、承認セクション内でルールを定義し、ユーザーの単純名を識別名に変換できます。認証中に行われる「ユーザー名から識別名へのマッピング」検索の結果はキャッシュされ、**force** が false に設定されている場合は承認クエリー中に再使用されます。**force** が true の場合は、承認中 (グループのロード中) に検索が再実行されます。通常、これは認証と承認が異なるサーバーによって実行される場合に行われます。

```
<authorization>
    <ldap connection="...">
     <!-- OPTIONAL -->
       <username-to-dn force="true">
           <!-- Only one of the following. -->
           <username-is-dn />
           <username-filter base-dn="..." recursive="..." user-dn-
attribute="..." attribute="..." />
           <advanced-filter base-dn="..." recursive="..." user-dn-
attribute="..." filter="..." />
        </username-to-dn>
```

```
       <group-search group-name="..." iterative="..." group-dn-
attribute="..." group-name-attribute="..." >
           <!-- One of the following -->
           <group-to-principal base-dn="..." recursive="..." search-
by="...">
               <membership-filter principal-attribute="..." />
           </group-to-principal>
           <principal-to-group group-attribute="..." />
       </group-search>
    </ldap>
</authorization>
```

重要

これらの例では、実証目的で一部の属性をデフォルト値に指定します。デフォルト値を指定する属性は、サーバーによって永続化されると設定から削除されます。例外は **force** 属性で、デフォルト値の **false** に設定されていても必要となります。

username-to-dn

username-to-dn 要素は、ユーザー名をエントリーの識別名へマップする方法を指定します。この要素は、以下の両方の条件に見合う場合のみ必要となります。

- 認証および承認手順は異なる LDAP サーバーに対するものである。
- グループ検索が識別名を使用する。

1:1 username-to-dn

リモートユーザーによって入力されたユーザー名はユーザーの識別名であると指定します。

```
<username-to-dn force="false">
   <username-is-dn />
</username-to-dn>
```

これは 1:1 マッピングを定義し、追加の設定はありません。

username-filter

次のオプションは、前述の認証手順の簡単なオプションと似ています。指定のユーザー名に対して一致する指定の属性が検索されます。

```
<username-to-dn force="true">
    <username-filter base-dn="dc=people,dc=harold,dc=example,dc=com"
recursive="false" attribute="sn" user-dn-attribute="dn" />
</username-to-dn>
```

設定可能な属性は次のとおりです。

- **base-dn:** 検索を開始するコンテキストの識別名。
- **recursive:** サブコンテキストが検索対象となるかどうか。デフォルトは**false** です。

- **attribute:** 指定のユーザー名に対して一致されるユーザーエントリーの属性。デフォルトは **uid** です。
- **user-dn-attribute:** ユーザーの識別名を取得するために読み取る属性。デフォルトは**dn** です。

advanced-filter

詳細フィルターを指定するオプションです。認証セクションでは、カスタムフィルターを使用してユーザーの識別名を見つけられます。

```
<username-to-dn force="true">
    <advanced-filter base-dn="dc=people,dc=harold,dc=example,dc=com"
recursive="false" filter="sAMAccountName={0}" user-dn-attribute="dn" />
</username-to-dn>
```

username-filter の例で一致する属性は、その意味とデフォルト値が同じです。新たな属性は以下の 1 つのみです。

- **filter:** ユーザー名が **{0}** プレースホルダーで置き換えられる、ユーザーのエントリーの検索に使用されるカスタムフィルター。

重要

フィルターの定義後も XML が有効である必要があるため、**&** などの特殊文字が使用される場合は、適切な形式が使用されるようにしてください。たとえば、**&** 文字には **&** を使用します。

グループ検索

グループメンバーシップ情報の検索に 2 つのスタイルを使用できます。1 つ目のスタイルは、ユーザーがメンバーであるグループを参照する属性が含まれるユーザーのエントリーで、2 つ目のスタイルは、ユーザーエントリーを参照する属性が含まれるグループです。

使用するスタイルを選択できる場合、Red Hat はグループを参照するユーザーのエントリーの設定を使用することを推奨します。検索を実行せずに既知の識別名の属性を読み取り、グループ情報をロードできるためです。別の方法には、ユーザーを参照するグループを特定するための広範な検索が必要となります。

設定を記述する前に、LDIF の例を見てみましょう。

例25.8 LDIF の例: プリンシプルからグループ

この例では、ユーザー **TestUserOne** は **GroupOne** のメンバーで、**GroupOne** は **GroupFive** のメンバーです。グループメンバーシップは、**memberOf** 属性を使用して表されます。**memberOf** 属性は、ユーザー (またはグループ) がメンバーであるグループの識別名に設定されます。

ここには示されていませんが、複数の **memberOf** 属性を設定することも可能です (ユーザーが直接メンバーであるグループごとに 1 つ)。

```
dn: uid=TestUserOne,ou=users,dc=principal-to-group,dc=example,dc=org
objectClass: extensibleObject
objectClass: top
```

```
objectClass: groupMember
objectClass: inetOrgPerson
objectClass: uidObject
objectClass: person
objectClass: organizationalPerson
cn: Test User One
sn: Test User One
uid: TestUserOne
distinguishedName: uid=TestUserOne,ou=users,dc=principal-to-
group,dc=example,dc=org
memberOf: uid=GroupOne,ou=groups,dc=principal-to-group,dc=example,dc=org
memberOf: uid=Slashy/Group,ou=groups,dc=principal-to-
group,dc=example,dc=org
userPassword::
e1NTSEF9WFpURzhLVjc4WVZBQUJNbEI3Ym96UVAva0RTNlFNWUpLOTdTMUE9PQ==

dn: uid=GroupOne,ou=groups,dc=principal-to-group,dc=example,dc=org
objectClass: extensibleObject
objectClass: top
objectClass: groupMember
objectClass: group
objectClass: uidObject
uid: GroupOne
distinguishedName: uid=GroupOne,ou=groups,dc=principal-to-
group,dc=example,dc=org
memberOf: uid=GroupFive,ou=subgroups,ou=groups,dc=principal-to-
group,dc=example,dc=org

dn: uid=GroupFive,ou=subgroups,ou=groups,dc=principal-to-
group,dc=example,dc=org
objectClass: extensibleObject
objectClass: top
objectClass: groupMember
objectClass: group
objectClass: uidObject
uid: GroupFive
distinguishedName: uid=GroupFive,ou=subgroups,ou=groups,dc=principal-to-
group,dc=example,dc=org
```

例25.9 LDIF の例: グループからプリンシパル

この例でも、ユーザー **TestUserOne** は **GroupOne** のメンバーであり、**GroupOne** は **GroupFive** のメンバーですが、相互参照にはグループからユーザーへ属性 **uniqueMember** が使用されます。

グループメンバーシップの相互参照に使用される属性は繰り返しが可能で、**GroupFive** を確認すると、別のユーザー **TestUserFive** への参照があります (ここでは示されていません)。

```
dn: uid=TestUserOne,ou=users,dc=group-to-principal,dc=example,dc=org
objectClass: top
objectClass: inetOrgPerson
objectClass: uidObject
objectClass: person
objectClass: organizationalPerson
cn: Test User One
```

```
sn: Test User One
uid: TestUserOne
userPassword::
e1NTSEF9SjR0OTRDR1ltaHc1VVZQOEJvbXhUYjl1dkFVd1lQTmRLSEdzaWc9PQ==

dn: uid=GroupOne,ou=groups,dc=group-to-principal,dc=example,dc=org
objectClass: top
objectClass: groupOfUniqueNames
objectClass: uidObject
cn: Group One
uid: GroupOne
uniqueMember: uid=TestUserOne,ou=users,dc=group-to-
principal,dc=example,dc=org

dn: uid=GroupFive,ou=subgroups,ou=groups,dc=group-to-
principal,dc=example,dc=org
objectClass: top
objectClass: groupOfUniqueNames
objectClass: uidObject
cn: Group Five
uid: GroupFive
uniqueMember: uid=TestUserFive,ou=users,dc=group-to-
principal,dc=example,dc=org
uniqueMember: uid=GroupOne,ou=groups,dc=group-to-
principal,dc=example,dc=org
```

一般的なグループ検索

前述の 2 つの方法の例を確認する前に、両方の方法に共通する属性を定義する必要があります。

```
<group-search group-name="..." iterative="..." group-dn-attribute="..."
group-name-attribute="..." >
    ...
</group-search>
```

- **group-name:** この属性は、ユーザーがメンバーであるグループのリストとして返されるグループ名に使用される形式を指定するために使用されます。これは、単純なグループ名またはグループの識別名になります。識別名が必要な場合は、この属性を **DISTINGUISHED_NAME** に設定できます。デフォルトは **SIMPLE** です。

- **iterative:** ユーザーがメンバーであるグループを特定した後に、グループがメンバーであるグループを特定するため、グループを基に反復検索するかどうかを指定するために使用される属性です。反復検索が有効であると、他のグループのメンバーでないグループが見つかるか、サイクルが検出されるまで検索を続行します。デフォルトは **false** です。

巡回のグループメンバーシップは問題ではありません。検索済みグループの再検索を防ぐため、各検索の記録が保存されます。

重要

反復検索が正しく実行されるようにするには、グループエントリーがユーザーエントリーと同じに表示される必要があります。ユーザーがメンバーであるグループを特定するために使用された方法で、グループがメンバーであるグループを特定します。グループからグループへのメンバーシップで相互参照に使用される属性の名前が変更されたり、参照の方向が変更されたりする場合には、これを実行することはできません。

- **group-dn-attribute:** 属性が識別名であるグループのエントリーです。デフォルトは**dn** です。
- **group-name-attribute:** 属性が単純名であるグループのエントリーです。デフォルトは**uid** です。

例25.10 プリンシパルからグループへの設定例

前述の LDIF の例を基にした、ユーザーグループを反復的にロードする設定の例は次のとおりです。相互参照に使用される属性はユーザーの **memberOf** 属性です。

```
<authorization>
    <ldap connection="LocalLdap">
        <username-to-dn>
            <username-filter base-dn="ou=users,dc=principal-to-
group,dc=example,dc=org" recursive="false" attribute="uid" user-dn-
attribute="dn" />
        </username-to-dn>
        <group-search group-name="SIMPLE" iterative="true" group-dn-
attribute="dn" group-name-attribute="uid">
            <principal-to-group group-attribute="memberOf" />
        </group-search>
    </ldap>
</authorization>
```

この設定で最も重要なことは、**principal-to-group** 要素が単一の属性で追加されていることです。

- **group-attribute:** ユーザーがメンバーであるグループの識別名と一致する、ユーザーエントリー上の属性名。デフォルトは **memberOf** です。

例25.11 グループからプリンシパルへの設定例

この例は、前述のグループからプリンシパルへの LDIF の例に対する反復検索を示しています。

```
<authorization>
      <ldap connection="LocalLdap">
          <username-to-dn>
              <username-filter base-dn="ou=users,dc=group-to-
principal,dc=example,dc=org" recursive="false" attribute="uid" user-dn-
attribute="dn" />
          </username-to-dn>
          <group-search group-name="SIMPLE" iterative="true" group-dn-
attribute="dn" group-name-attribute="uid">
              <group-to-principal base-dn="ou=groups,dc=group-to-
principal,dc=example,dc=org" recursive="true" search-
```

```
by="DISTINGUISHED_NAME">
                    <membership-filter principal-attribute="uniqueMember"
/>
                </group-to-principal>
            </group-search>
        </ldap>
    </authorization>
```

ここでは、**group-to-principal** 要素が追加されています。この要素は、ユーザーエントリーを参照するグループの検索がどのように実行されるかを定義するために使用されます。以下の属性が設定されます。

- **base-dn**: 検索を開始するために使用するコンテキストの識別名。
- **recursive**: サブコンテキストも検索されるかどうか。デフォルトは**false** です。
- **search-by**: 検索で使用されるロール名の形式です。有効な値は**SIMPLE** および **DISTINGUISHED_NAME** です。デフォルトは **DISTINGUISHED_NAME** です。

group-to-principal 要素内に、相互参照を定義する **membership-filter** 要素があります。

- **principal-attribute**: ユーザーエントリーを参照する、グループエントリー上の属性名。デフォルトは **member** です。

バグを報告する

25.9. インターフェースのセキュア化

25.9.1. Hot Rod インターフェースセキュリティー

25.9.1.1. Hot Rod エンドポイントをパブリックインターフェースとして公開

Red Hat JBoss Data Grid の Hot Rod サーバーはデフォルトで管理インターフェースとして動作します。操作をパブリックインターフェースに拡張するには、以下のように、**management** の **socket-binding** 要素の ***interface*** パラメーターの値を **public** に変更します。

```
<socket-binding name="hotrod" interface="public" port="11222" />
```

バグを報告する

25.9.1.2. Hot Rod サーバーと Hot Rod クライアント間の通信の暗号化

Hot Rod は TLS/SSL を使用して暗号化でき、証明書ベースのクライアント認証を必要とするオプションを使用できます。

以下の手順を使用し、SSL を使用して Hot Rod コネクターのセキュリティーを保護します。

手順25.3 SSL/TLS を使用したセキュアな Hot Rod

1. **キーストアを生成します。**

JDK と共に配信されるキーツールアプリケーションを使用して Java キーストアを作成し、証明書をこれに追加します。証明書は、セキュリティーポリシーに応じて自己署名型を使用するか、または信頼された CA から取得できます。

2. **キーストアを設定ディレクトリーに配置します。**
キーストアを、**~/JDG_HOME/docs/examples/configs** ディレクトリーからの **standalone-hotrod-ssl.xml** ファイルとと共に **~/JDG_HOME/standalone/configuration** ディレクトリーに配置します。

3. **SSL サーバー ID を宣言します。**
設定ファイルの管理セクションのセキュリティーレルム内で SSL サーバー ID を宣言します。SSL サーバー ID ではキーストアへのパスとそのシークレットキーを指定する必要があります。

```
<server-identities>
  <ssl protocol="...">
    <keystore path="..." relative-to="..." keystore-
password="${VAULT::VAULT_BLOCK::ATTRIBUTE_NAME::ENCRYPTED_VALUE}" />
  </ssl>
  <secret value="..." />
</server-identities>
```

これらのパラメーターについての詳細は、「Hot Rod 認証の設定 (X.509) 」 を参照してください。

4. **セキュリティー要素を追加します。**
以下のようにセキュリティー要素を Hot Rod コネクターに追加します。

```
<hotrod-connector socket-binding="hotrod" cache-container="local">
    <encryption ssl="true" security-realm="ApplicationRealm"
require-ssl-client-auth="false" />
</hotrod-connector>
```

a. **証明書のサーバー認証**
サーバーがクライアント証明書の認証を実行できるようにするには、有効なクライアント証明書を含むトラストストアを作成し、***require-ssl-client-auth*** 属性を **true** に設定します。

5. **サーバーを起動します。**
以下を使用してサーバーを起動します。

```
bin/standalone.sh -c standalone-hotrod-ssl.xml
```

これにより、Hot Rod エンドポイントがポート 11222 にある状態でサーバーが起動します。このエンドポイントは SSL 接続のみを受け入れます。

重要

プレーンテキストのパスワードが設定またはソースコードに表示されないようにするには、プレーンテキストのパスワードを Vault パスワードに変更する必要があります。Vault パスワードのセットアップ方法についての詳細は、『Red Hat Enterprise Application Platform Security Guide』 を参照してください。

バグを報告する

25.9.1.3. SSL を使用した Hot Rod のセキュア化

SSL を有効にして LDAP に接続する際に、適切な証明書が含まれるトラストストアまたはキーストアを指定する必要がある場合があります。

「Hot Rod サーバーと Hot Rod クライアント間の通信の暗号化」 は、Hot Rod クライアント/サーバー間通信のために SSH をセットアップする方法について説明しています。これは、たとえば **PLAIN** ユーザー名/パスワードを使ったセキュアな Hot Rod クライアントの認証に使用できます。ユーザー名/パスワードが LDAP のクレデンシャルに対してチェックされる際、Hot Rod サーバーから LDAP サーバーへのセキュアな接続も必要になります。SSL 経由で Hot Rod サーバーから LDAP への接続を有効にするには、セキュリティーレルムを以下のように定義する必要があります。

例25.12 Hot Rod クライアントの LDAP サーバーに対する認証

```
<management>
        <security-realms>
            <security-realm name="LdapSSLRealm">
                <authentication>
                    <truststore  path="ldap.truststore" relative-
to="jboss.server.config.dir" keystore-
password=${VAULT::VAULT_BLOCK::ATTRIBUTE_NAME::ENCRYPTED_VALUE} />
                </authentication>
            </security-realm>
        </security-realms>
        <outbound-connections>
            <ldap name="LocalLdap" url="ldaps://localhost:10389"
search-dn="uid=wildfly,dc=simple,dc=wildfly,dc=org" search-
credential="secret" security-realm="LdapSSLRealm" />
        </outbound-connections>
    </management>
```

重要

プレーンテキストのパスワードが設定またはソースコードに表示されないようにするには、プレーンテキストのパスワードを Vault パスワードに変更する必要があります。Vault パスワードのセットアップ方法についての詳細は、『 Red Hat Enterprise Application Platform Security Guide』 を参照してください。

バグを報告する

25.9.1.4. SASL を使用した Hot Rod でのユーザー認証

Hot Rod でのユーザー認証は、以下の SASL (Simple Authentication and Security Layer) メカニズムを使用して実装できます。

- **PLAIN** は、クレデンシャルがプレーンテキスト形式でトランスポートされるために最も安全性の低いメカニズムになります。ただし、実装が最も単純なメカニズムでもあります。このメカニズムは、セキュリティーを強化するために暗号化 (SSL) と併用できます。
- **DIGEST-MD5** は、クレデンシャルをトランスポートする前にハッシュ化するメカニズムです。その結果、**PLAIN** メカニズムよりも安全性が高くなります。

- **GSSAPI** は、Kerberos チケットを使用するメカニズムです。その結果、正しく設定された Kerberos Domain Controller (例: Microsoft Active Directory) が必要になります。
- **EXTERNAL** は、基礎となるトランスポート (例:**X.509** クライアント証明書) から必要なクレデンシャルを取得するメカニズムであるため、正常に機能するにはクライアント証明書の暗号化が必要です。

バグを報告する

25.9.1.4.1. Hot Rod 認証の設定 (GSSAPI/Kerberos)

以下の手順を使用し、SASL GSSAPI/Kerberos メカニズムを使用して Hot Rod 認証をセットアップします。

手順25.4 SASL GSSAPI/Kerberos 認証の設定: サーバー側の設定

1. セキュリティードメインサブシステムを使用して Kerberos のセキュリティーログインモジュールを定義します。

```
<system-properties>
    <property name="java.security.krb5.conf"
value="/tmp/infinispan/krb5.conf"/>
    <property name="java.security.krb5.debug" value="true"/>
    <property name="jboss.security.disable.secdomain.option"
value="true"/>
</system-properties>

<security-domain name="infinispan-server" cache-type="default">
    <authentication>
        <login-module code="Kerberos" flag="required">
            <module-option name="debug" value="true"/>
            <module-option name="storeKey" value="true"/>
            <module-option name="refreshKrb5Config" value="true"/>
            <module-option name="useKeyTab" value="true"/>
            <module-option name="doNotPrompt" value="true"/>
            <module-option name="keyTab"
value="/tmp/infinispan/infinispan.keytab"/>
            <module-option name="principal"
value="HOTROD/localhost@INFINISPAN.ORG"/>
        </login-module>
    </authentication>
</security-domain>
```

2. キャッシュコンテナーに承認ロールが定義されており、これらのロールが「Red Hat JBoss Data Grid の承認の設定」に示されるようにキャッシュの承認ブロックで適用されていることを確認します。
3. Hot Rod コネクターを以下のように設定します。

```
<hotrod-connector socket-binding="hotrod"
    cache-container="default">
 <authentication security-realm="ApplicationRealm">
  <sasl server-name="node0"
        mechanisms="{mechanism_name}"
        qop="{qop_name}"
```

```
        strength="{value}">
    <policy>
     <no-anonymous value="true" />
    </policy>
    <property
name="com.sun.security.sasl.digest.utf8">true</property>
   </sasl>
   </authentication>
</hotrod-connector>
```

- ***server-name*** 属性は、サーバーが着信クライアントに対して宣言する名前を指定します。クライアント設定にも同じサーバー名の値が含まれる必要があります。
- ***server-context-name*** 属性は、特定の SASL メカニズムのサーバーサブジェクトを取得するために使用されるログインコンテキストの名前を指定します (例: GSSAPI)。
- ***mechanisms*** 属性は、使用される認証メカニズムを指定します。サポートされるメカニズムの一覧は、「SASL を使用した Hot Rod でのユーザー認証」 を参照してください。
- ***qop*** 属性は、設定についての本番環境の値の SASL の品質を指定します。この属性のサポートされる値は、**auth** (認証)、**auth-int** (改ざんを検出するためにメッセージがチェックサムに対して検証されることを意味する認証と整合性)、および **auth-conf** (メッセージの暗号化も行われることを意味する認証、整合性および機密性) です。**auth-int auth-conf** などのように複数の値を指定できます。順番は優先順位を示し、クライアントとサーバーの優先順位の両方に一致する最初の値が選択されます。
- ***strength*** 属性は、SASL の暗号強度を指定します。有効な値は **low**、**medium**、および **high** です。
- **policy** 要素内の **no-anonymous** 要素は、匿名ログインを受け入れるメカニズムを許可するかどうかを指定します。許可するには、この値を **false** にし、拒否するには **true** に設定します。

4. 各クライアントでクライアント側の設定を実行します。Hot Rod クライアントをプログラムを用いて設定する際に、この設定の情報については、JBoss Data Grid の 『Developer Guide』 で確認できます。

25.9.1.4.2. Hot Rod 認証の設定 (MD5)

以下の手順を使用し、MD5 メカニズムを使用した SASL による Hot Rod 認証をセットアップします。

手順25.5 Hot Rod 認証の設定 (MD5)

1. **sasl** 要素を **authentication** 要素に追加し、以下のように Hot Rod コネクター設定をセットアップします (**authentication** 要素の詳細は、「セキュリティーレルムの宣言的な有効化」 を参照してください)。

```
<hotrod-connector socket-binding="hotrod"
                            cache-container="default">
    <authentication security-realm="ApplicationRealm">
        <sasl server-name="myhotrodserver"
                mechanisms="DIGEST-MD5"
                qop="auth" />
    </authentication>
</hotrod-connector>
```

- ***server-name*** 属性は、サーバーが着信クライアントに対して宣言する名前を指定します。クライアント設定にも同じサーバー名の値が含まれる必要があります。
- ***mechanisms*** 属性は、使用される認証メカニズムを指定します。サポートされるメカニズムの一覧は、「SASL を使用した Hot Rod でのユーザー認証」 を参照してください。
- ***qop*** 属性は、設定についての本番環境の値の SASL の品質を指定します。この属性のサポートされる値は、**auth**、**auth-int**、および **auth-conf** です。

2. 各クライアントが Hot Rod コネクターに接続されるように設定します。この手順をプログラムを用いて実行する際に、その方法については JBoss Data Grid の 『Developer Guide』 で確認できます。

25.9.1.4.3. LDAP/Active Directory を使用した Hot Rod の設定

以下を使用し、LDAP または Microsoft Active Directory を使用して Hot Rod で認証を設定します。

```
<security-realms>
 <security-realm name="ApplicationRealm">
  <authentication>
   <ldap connection="ldap_connection"
         recursive="true"
         base-dn="cn=users,dc=infinispan,dc=org">
    <username-filter attribute="cn" />
   </ldap>
  </authentication>
 </security-realm>
</security-realms>
<outbound-connections>
 <ldap name="ldap_connection"
       url="ldap://my_ldap_server"
       search-dn="CN=test,CN=Users,DC=infinispan,DC=org"
       search-credential="Test_password"/>
</outbound-connections>
```

以下は、この設定で使用される要素およびパラメーターについての詳細です。

- **security-realm** 要素の ***name*** パラメーターは、接続の確立時に使用するために参照するセキュリティーレルムを指定します。
- **authentication** 要素には認証の詳細情報が含まれます。
- **ldap** 要素は、ユーザーを認証するために LDAP 検索を使用する方法を指定します。最初に、LDAP への接続が確立され、ユーザーの識別名を特定するために指定されるユーザー名を使用して検索が実行されます。その後のサーバーへの接続は、ユーザーが指定するパスワードを使って確立されます。2 番目の接続が成功すると認証が行われます。
  - ***connection*** パラメーターは、LDAP に接続するために使用する接続の名前を指定します。
  - (オプション) ***recursive*** パラメーターは、フィルターが再帰的に実行されるかどうかを指定します。このパラメーターのデフォルト値は **false** です。
  - ***base-dn*** パラメーターは、検索の開始に使用するコンテキストの識別名を指定します。
  - (オプション) ***user-dn*** パラメーターは、ユーザーの検索後にユーザーの識別名を読み取る

ために使用する属性を指定します。このパラメーターのデフォルト値は **dn** です。

- **outbound-connections** 要素は LDAP ディレクトリーに接続するために使用される接続名を指定します。
- **ldap** 要素は送信 LDAP 接続のプロパティーを指定します。
  - ***name*** パラメーターは、この接続を参照するために使用される固有名を指定します。
  - ***url*** パラメーターは、LDAP 接続を確立するために使用される URL を指定します。
  - ***search-dn*** パラメーターは、認証対象で、検索を実行するためのユーザーの識別名を指定します。
  - ***search-credential*** パラメーターは、LDAP に ***search-dn*** として接続することが必要なパスワードを指定します。
  - (オプション) ***initial-context-factory*** パラメーターは、初期のコンテキストファクトリーのオーバーライドを許可します。このパラメーターのデフォルト値は **com.sun.jndi.ldap.LdapCtxFactory** です。

バグを報告する

25.9.1.4.4. Hot Rod 認証の設定 (X.509)

X.509 証明書をノードでインストールでき、かつ受信および送信 SSL 接続の認証のためにこれを他のノードで利用できるようにすることができます。これは、サーバーの外部アプリケーションへの表示方法を定義するセキュリティーレルム定義の ***<server-identities/>*** 要素を使用して有効にされます。この要素は、リモート接続の確立時、および **X.509** キーのロード時に使用されるパスワードを設定するために使用できます。

以下の例は、**X.509** 証明書をノードにインストールする方法を示しています。

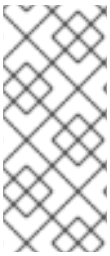
```
<security-realm name="ApplicationRealm">
  <server-identities>
    <ssl protocol="...">
      <keystore path="..." relative-to="..." keystore-password="..."
alias="..." key-password="..." />
    </ssl>
  </server-identities>

  [... authentication/authorization ...]

 </security-realms>
```

この例では、SSL 要素に **<keystore/>** 要素が含まれます。これは、キーをファイルベースのキーストアからロードする方法を定義するために使用されます。以下のパラメーターはこの要素に利用できます。

表25.4 <server-identities/> オプション

パラメーター	必須/オプション	説明

パラメーター	必須/オプション	説明
path	必須	これはキーストアへのパスです。絶対パスを使用することも、次の属性の相対パスを使用することもできます。
relative-to	任意	キーストアが相対するパスを表すサービスの名前。
keystore-password	必須	キーストアを開くために必要なパスワード。
alias	任意	キーストアから使用するエントリーのエイリアス。複数のエントリーが使用されるキーストアの場合、最初に使用できるエントリーが使用されますが、これに依存するのではなく、使用されるエントリーを保証できるようにエイリアスを設定する必要があります。
key-password	任意	キーエントリーをロードするためのパスワード。省略されている場合、**keystore-password** が代わりに使用されます。



注記

以下のエラーが発生する場合、1 つのキーのみがロードされていることを確認するために、***key-password*** および ***alias*** を指定します。

```
UnrecoverableKeyException: Cannot recover key
```

バグを報告する

25.9.2. REST インターフェースセキュリティー

25.9.2.1. REST エンドポイントをパブリックインターフェースとして公開

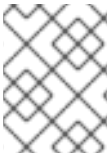
Red Hat JBoss Data Grid の REST サーバーはデフォルトで管理インターフェースとして動作します。操作をパブリックインターフェースに拡張するには、以下のように、**management** の **socket-binding** 要素の ***interface*** パラメーターの値を **public** に変更します。

```
<socket-binding name="http"
  interface="public"
  port="8080"/>
```

バグを報告する

25.9.2.2. REST エンドポイントのセキュリティーの有効化

以下の手順を使用して、Red Hat JBoss Data Grid で REST エンドポイントのセキュリティーを有効にします。



注記

REST エンドポイントは、JBoss Enterprise Application Platform のセキュリティーサブシステムプロバイダーのいずれかをサポートします。

手順25.6 REST エンドポイントのセキュリティーの有効化

REST インターフェースを使用する場合に JBoss Data Grid のセキュリティーを有効にするには、**`standalone.xml`** に以下の変更を行います。

1. **セキュリティーパラメーターの指定**
   rest エンドポイントで ***`security-domain`*** パラメーターおよび ***`auth-method`*** パラメーターの有効な値を指定するようにします。これらのパラメーターの推奨設定は以下のとおりです。

```
<subsystem xmlns="urn:infinispan:server:endpoint:8.0">
            <rest-connector virtual-server="default-host"
                            cache-container="local"
                            security-domain="other"
                            auth-method="BASIC"/>
</subsystem>
```

2. **セキュリティードメイン宣言のチェック**
   セキュリティーサブシステムに、対応するセキュリティードメイン宣言が含まれるようにします。セキュリティードメイン宣言の設定の詳細については、JBoss Enterprise Application Platform 7 ドキュメンテーションを参照してください。

3. **アプリケーションユーザーの追加**
   該当するスクリプトを実行し、アプリケーションユーザーを追加する設定を入力します。

   a. **`adduser.sh`** スクリプト (**`$JDG_HOME/bin`** に存在) を実行します。

      - Windows システムでは、**`adduser.bat`** ファイル (**`$JDG_HOME/bin`** に存在) を代わりに実行します。

   b. 追加するユーザーのタイプについて尋ねられたら、**b** を入力して **`Application User (application-users.properties)`** を選択します。

   c. リターンキーを押して、レルム (**`ApplicationRealm`**) のデフォルト値を使用します。

   d. ユーザー名とパスワードを指定します。

   e. グループを尋ねられたら、**REST** と入力します。

   f. プロンプトが表示されたら、ユーザー名とアプリケーションレルム情報が正しいことを確認し、**"yes"** と入力して作業を続行します。

4. **作成されたアプリケーションユーザーの確認**
   作成されたアプリケーションユーザーが正しく設定されていることを確認します。

a. **application-users.properties** ファイル (**$JDG_HOME/standalone/configuration/** に存在) にリストされた設定を確認します。以下は、このファイルの正しい設定の例です。

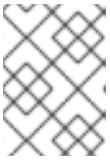
```
user1=2dc3eacfed8cf95a4a31159167b936fc
```

b. **application-roles.properties** ファイル (**$JDG_HOME/standalone/configuration/** に存在) にリストされた設定を確認します。以下は、このファイルの正しい設定の例です。

```
user1=REST
```

5. **サーバーのテスト**
サーバーを起動し、ブラウザーウィンドウに以下のリンクを入力して REST エンドポイントにアクセスします。

```
http://localhost:8080/rest/namedCache
```



注記

GET 要求の使用をテストする場合は、**405** 応答コードが期待され、サーバーが正常に認証されたことが示されます。

バグを報告する

25.9.3. Memcached インターフェースセキュリティー

25.9.3.1. Memcached エンドポイントをパブリックインターフェースとして公開

Red Hat JBoss Data Grid の memcached サーバーは、デフォルトで管理インターフェースとして機能します。memcached 操作をパブリックインターフェースに拡張することはできますが、このインターフェースに使用できる追加のセキュリティーはありません。セキュリティーに関連する懸念点がある場合には、このインターフェースを、分離された内部ネットワークに留めるか、または REST または Hot Rod インターフェースのいずれかを使用することをお勧めします。

memcached インターフェースをパブリックインターフェースとして設定するには、**socket-binding** 要素の ***interface*** パラメーターの値を、以下のように**management** から **public** に変更します。

```
<socket-binding name="memcached"
  interface="public"
  port="11211" />
```

バグを報告する

25.10. ACTIVE DIRECTORY 認証 (KERBEROS 以外)

Kerberos 以外の Active Directory 認証の設定例については、例25.1「LDAP ログインモジュールの設定例」 を参照してください。

バグを報告する

25.11. KERBEROS を使用した ACTIVE DIRECTORY 認証 (GSSAPI)

Red Hat JBoss Data Grid を Microsoft Active Directory と共に使用する場合、データセキュリティーは Kerberos 認証で有効にできます。Microsoft Active Directory の Kerberos 認証を設定するには、以下の手順を使用します。

手順25.7 Active Directory の Kerberos 認証の設定 (ライブラリーモード)

1. JBoss EAP サーバーが Kerberos で認証できるように設定します。これは、以下のような専用セキュリティードメインを設定して実行できます。

```
<security-domain name="ldap-service" cache-type="default">
    <authentication>
        <login-module code="Kerberos" flag="required">
            <module-option name="storeKey" value="true"/>
            <module-option name="useKeyTab" value="true"/>
            <module-option name="refreshKrb5Config" value="true"/>
            <module-option name="principal"
value="ldap/localhost@INFINISPAN.ORG"/>
            <module-option name="keyTab"
value="${basedir}/keytab/ldap.keytab"/>
            <module-option name="doNotPrompt" value="true"/>
        </login-module>
    </authentication>
</security-domain>
```

2. 認証用のセキュリティードメインは JBoss EAP について正常に設定される必要があり、アプリケーションには有効な Kerberos チケットがなければなりません。Kerberos チケットを初期化するには、以下を使用して別のセキュリティードメインを参照する必要があります。

```
<module-option name="usernamePasswordDomain" value="krb-admin"/>
```

. これは手順 3 で説明される標準の Kerberos ログインモジュールを参照します。

```
<security-domain name="ispn-admin" cache-type="default">
    <authentication>
        <login-module code="SPNEGO" flag="requisite">
            <module-option name="password-stacking"
value="useFirstPass"/>
            <module-option name="serverSecurityDomain" value="ldap-
service"/>
            <module-option name="usernamePasswordDomain" value="krb-
admin"/>
        </login-module>
        <login-module code="AdvancedAdLdap" flag="required">
            <module-option name="password-stacking"
value="useFirstPass"/>
            <module-option name="bindAuthentication"
value="GSSAPI"/>
            <module-option name="jaasSecurityDomain" value="ldap-
service"/>
            <module-option name="java.naming.provider.url"
value="ldap://localhost:389"/>
            <module-option name="baseCtxDN"
value="ou=People,dc=infinispan,dc=org"/>
```

```
            <module-option name="baseFilter" value="
(krb5PrincipalName={0})"/>
            <module-option name="rolesCtxDN"
value="ou=Roles,dc=infinispan,dc=org"/>
            <module-option name="roleFilter" value="(member={1})"/>
            <module-option name="roleAttributeID" value="cn"/>
        </login-module>
    </authentication>
</security-domain>
```

3. 直前の手順で説明されているセキュリティードメインの認証設定は、以下の標準 Kerberos ログインモジュールを参照します。

```
<security-domain name="krb-admin" cache-type="default">
    <authentication>
        <login-module code="Kerberos" flag="required">
            <module-option name="useKeyTab" value="true"/>
            <module-option name="principal"
value="admin@INFINISPAN.ORG"/>
            <module-option name="keyTab"
value="${basedir}/keytab/admin.keytab"/>
        </login-module>
    </authentication>
</security-domain>
```

バグを報告する

25.12. セキュリティー監査ロガー

Red Hat JBoss Data Grid には、キャッシュのセキュリティーログを監査するためのロガーが含まれます。これは、キャッシュまたはキャッシュマネージャーの操作が各種操作について許可または拒否されるかどうかにとくに関連するものです。

デフォルトの監査ロガーは **org.infinispan.security.impl.DefaultAuditLogger** です。このロガーは、利用可能なロギングフレームワーク (JBoss ロギングなど) を使用して監査ログを出力し、**TRACE** レベルおよび **AUDIT** カテゴリーで結果を提供します。

AUDIT カテゴリーを、ログファイル、JMS キュー、またはデータベースのいずれかに送信するには、適切なログアペンダーを使用します。

バグを報告する

25.12.1. セキュリティー監査ロガーの設定 (ライブラリーモード)

以下を使用して Red Hat JBoss Data Grid の監査ロガーを設定します。

```
<infinispan>
  ...
 <global-security>
  <authorization audit-logger =
"org.infinispan.security.impl.DefaultAuditLogger">
     ...
  </authorization>
```

```
 </global-security>
 ...
</infinispan>
```

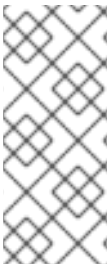
バグを報告する

25.12.2. セキュリティー監査ロガーの設定 (リモートクライアントサーバーモード)

以下のコードを使用し、Red Hat JBoss Data Grid のリモートクライアントサーバーモードで監査ロガーを設定します。

異なる監査ロガーを使用するには、これを ***<authorization>*** 要素に指定します。***<authorization>*** 要素は、Infinispan サブシステムの ***<cache-container>*** 要素内になければなりません (**standalone.xml** 設定ファイル内)。

```
<cache-container name="local" default-cache="default">
    <security>
        <authorization audit-
logger="org.infinispan.security.impl.DefaultAuditLogger">
            <identity-role-mapper/>
            <role name="admin" permissions="ALL"/>
            <role name="reader" permissions="READ"/>
            <role name="writer" permissions="WRITE"/>
            <role name="supervisor" permissions="ALL_READ ALL_WRITE"/>
        </authorization>
    </security>
    <local-cache name="default" start="EAGER">
        <locking isolation="NONE" acquire-timeout="30000" concurrency-
level="1000" striping="false"/>
        <transaction mode="NONE"/>
        <security>
            <authorization roles="admin reader writer supervisor"/>
        </security>
    </local-cache>
    [...]
</cache-container>
```



注記

サーバーモードのデフォルト監査ロガーは、ログメッセージをサーバー監査ログに送信する **org.jboss.as.clustering.infinispan.subsystem.ServerAuditLogger** です。詳細は、JBoss Enterprise Application Platform の 『Administration and Configuration Guide』 の 『Management Interface Audit Logging』 の章を参照してください。

バグを報告する

25.12.3. カスタム監査ロガー

ユーザーは、カスタム監査ロガーを Red Hat JBoss Data Grid の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実装できます。カスタムロガーは **org.infinispan.security.AuditLogger** インターフェースを実装する必要があります。カスタムロガーが指定されない場合、デフォルトのロガー (**DefaultAuditLogger**) が使用されます。

バグを報告する

第26章 クラスタートラフィックのセキュリティー

26.1. ノードの認証および承認 (リモートクライアントサーバーモード)

セキュリティーは SASL プロトコル経由でノードレベルで有効にすることができます。これにより、セキュリティーレルムに対するノードの認証が可能になります。この場合、ノードがクラスターに参加またはマージする際に相互に認証する必要があります。セキュリティーレルムについての詳細は、「セキュリティーレルム」 を参照してください。

以下の例は、**<sasl />** 要素について説明しています。これは SASL プロトコルを利用します。現在、**DIGEST-MD5** および **GSSAPI** の両方のメカニズムがサポートされています。

例26.1 SASL 認証の設定

```
<management>
    <security-realms>
        <!-- Additional configuration information here -->
        <security-realm name="ClusterRealm">
            <authentication>
                <properties path="cluster-users.properties" relative-
to="jboss.server.config.dir"/>
                </authentication>
                <authorization>
                    <properties path="cluster-roles.properties"
relative-to="jboss.server.config.dir"/>
                </authorization>
            </security-realm>
        </security-realms>
        <!-- Additional configuration information here -->
    </security-realms>
</management>

<stack name="udp">
    <!-- Additional configuration information here -->
    <sasl mech="DIGEST-MD5" security-realm="ClusterRealm" cluster-
role="cluster">
        <property name="client_name">node1</property>
        <property name="client_password">password</property>
    </sasl>
    <!-- Additional configuration information here -->
</stack>
```

この例では、ノードは **DIGEST-MD5** メカニズムを使用して **ClusterRealm** に対して認証しています。参加するノードには ***cluster*** ロールがなければなりません。

cluster-role 属性は、クラスターで **JOIN** または **MERGE** を実行するために、セキュリティーレルムですべてのノードが属する必要のあるロールを決定します。これが指定されない場合は、***cluster-role*** 属性は、デフォルトでクラスター化された ***<cache-container>*** の名前になります。各ノードは、***client-name*** プロパティーを使用して各自を識別します。何も指定されていない場合は、サーバーが実行されているホスト名が使用されます。

この名前は、コマンドラインで上書きできる **jboss.node.name** システムプロパティーを指定して上書きすることもできます。以下が例になります。

```
$ standalone.sh -Djboss.node.name=node001
```

注記

JGroups AUTH プロトコルはセキュリティーレルムに統合されておらず、その使用については Red Hat JBoss Data Grid で推奨されていません。

バグを報告する

26.1.1. クラスターセキュリティーのノード認証の設定 (DIGEST-MD5)

以下の例では、クラスターノードの専用レルムで、プロパティーベースのセキュリティーレルムを使って **DIGEST-MD5** を使用する方法を示しています。

例26.2 DIGEST-MD5 メカニズムの使用

```
<management>
         <security-realms>
             <security-realm name="ClusterRealm">
                 <authentication>
                     <properties path="cluster-users.properties"
relative-to="jboss.server.config.dir"/>
                 </authentication>
                 <authorization>
                     <properties path="cluster-roles.properties"
relative-to="jboss.server.config.dir"/>
                 </authorization>
             </security-realm>
         </security-realms>
</management>
<subsystem xmlns="urn:infinispan:server:jgroups:8.0" default-
stack="${jboss.default.jgroups.stack:udp}">
     <stack name="udp">
         <transport type="UDP" socket-binding="jgroups-udp"/>
         <protocol type="PING"/>
         <protocol type="MERGE2"/>
         <protocol type="FD_SOCK" socket-binding="jgroups-udp-fd"/>
         <protocol type="FD_ALL"/>
         <protocol type="pbcast.NAKACK"/>
         <protocol type="UNICAST2"/>
         <protocol type="pbcast.STABLE"/>
         <protocol type="pbcast.GMS"/>
         <protocol type="UFC"/>
         <protocol type="MFC"/>
         <protocol type="FRAG2"/>
         <protocol type="RSVP"/>
         <sasl security-realm="ClusterRealm" mech="DIGEST-MD5">
             <property name="client_password>...</property>
         </sasl>
     </stack>
</subsystem>
<subsystem xmlns="urn:infinispan:server:core:8.3" default-cache-
container="clustered">
     <cache-container name="clustered" default-cache="default">
```

```
            <transport executor="infinispan-transport" lock-
timeout="60000" stack="udp"/>
            <!-- various clustered cache definitions here -->
        </cache-container>
</subsystem>
```

この例では、各種ノードのホスト名が **node001**、**node002**、**node003** であると想定した場合に ***cluster-users.properties*** には以下が含まれます。

- **node001=/<node001passwordhash>/**
- **node002=/<node002passwordhash>/**
- **node003=/<node003passwordhash>/**

cluster-roles.properties には以下が含まれます。

- node001=clustered
- node002=clustered
- node003=clustered

これらの値を生成するには、以下の **add-users.sh** スクリプトを使用できます。

```
$ add-user.sh -up cluster-users.properties -gp cluster-roles.properties -r
ClusterRealm -u node001 -g clustered -p <password>
```

ノードの **MD5** パスワードハッシュも **<sasl/>** 要素の "***client_password***" プロパティーに置かれる必要があります。

```
<property name="client_password>...</property>
```

注記

セキュリティーのレベルを上げるには、このパスワードを Vault を使って保管することをお勧めします。vault 式についての詳細は、『Red Hat Enterprise Application Platform Security Guide』 を参照してください。

ノードのセキュリティーが説明されている通りにセットアップされていると、クラスターコーディネーターは、ノードがクラスタービューの一部になる前に、それぞれの **JOIN** および **MERGE** を実行するノードのクレデンシャルをレルムに対して検証します。

バグを報告する

26.1.2. クラスターセキュリティーのノード認証の設定 (GSSAPI/Kerberos)

GSSAPI メカニズムを使用する場合、***client_name*** は、セキュリティードメインサブシステム内で定義される Kerberos で有効にされるログインモジュールの名前として使用されます。この実行方法についての詳細は、「Hot Rod 認証の設定 (GSSAPI/Kerberos)」 を参照してください。

例26.3 Kerberos ログインモードの使用

```
<security-domain name="krb-node0" cache-type="default">
    <authentication>
        <login-module code="Kerberos" flag="required">
            <module-option name="storeKey" value="true"/>
            <module-option name="useKeyTab" value="true"/>
            <module-option name="refreshKrb5Config" value="true"/>
            <module-option name="principal"
value="jgroups/node0/clustered@INFINISPAN.ORG"/>
            <module-option name="keyTab"
value="${jboss.server.config.dir}/keytabs/jgroups_node0_clustered.keytab
"/>
            <module-option name="doNotPrompt" value="true"/>
        </login-module>
    </authentication>
</security-domain>
```

以下のプロパティーは、参照できるように **<sasl/>** 要素に設定する必要があります。

```
<sasl <!-- Additional configuration information here --> >
     <property name="login_module_name">
       <!-- Additional configuration information here -->
     </property>
</sasl>
```

結果として、ノードが Kerberos Domain Controller に対して検証されるため、セキュリティーレルムの **authentication** セクションは無視されます。**authorization** 設定は、ノードのプリンシパルが必要なクラスターロールに属するため、依然として必要になります。

いずれの場合も、管理を単純にするために、LDAP などの共有承認データベースをノードのメンバーシップを検証するために使用することをお勧めします。

デフォルトで、参加するノードのプリンシパルは以下の形式でなければなりません。

```
jgroups/$NODE_NAME/$CACHE_CONTAINER_NAME@REALM
```

バグを報告する

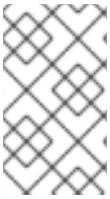
26.2. ノードセキュリティーの設定 (ライブラリーモード)

ライブラリーモードでは、ノード認証は JGroups 設定で直接設定されます。JGroups を設定してノードがクラスターへの参加またはマージの際に相互に認証できるようにします。認証は SASL を使用し、**SASL** プロトコルを JGroups XML 設定に追加して有効にされます。

SASL は、認証ハンドシェイクに必要な特定の情報を取得するために、**CallbackHandlers** などの JAAS の概念に基づいています。ユーザーはクライアントとサーバーの両サイドに独自の **CallbackHandlers** を指定する必要があります。

重要

JAAS API はユーザー認証および承認の設定時にのみ利用でき、ノードのセキュリティー用には利用できません。



注記

この例では、**CallbackHandler** クラスは例示する目的でのみ使用されており、Red Hat JBoss Data Grid リリースには含まれていません。ユーザーは特定の LDAP 実装に適した **CallbackHandler** クラスを提供する必要があります。

例26.4 JGroups での SASL 認証のセットアップ

```
<SASL mech="DIGEST-MD5"
    client_name="node_user"
    client_password="node_password"

server_callback_handler_class="org.example.infinispan.security.JGroupsSa
slServerCallbackHandler"

client_callback_handler_class="org.example.infinispan.security.JGroupsSa
slClientCallbackHandler"
    sasl_props="com.sun.security.sasl.digest.realm=test_realm" />
```

上記の例は、**DIGEST-MD5** メカニズムを使用します。各ノードは、クラスターへの参加時に使用するユーザーとパスワードを宣言する必要があります。

重要

認証を有効にするには、SASL プロトコルを GMS プロトコルの前に配置する必要があります。

バグを報告する

26.2.1. 単純な承認コールバックハンドラー

複雑度の高い Kerberos や LDAP アプローチが不要な場合、**SimpleAuthorizingCallbackHandler** クラスを使用できます。これを有効にするには、以下の例にあるように ***server_callback_handler*** と ***client_callback_handler*** の両方を **org.jgroups.auth.sasl.SimpleAuthorizingCallbackHandler** に設定します。

```
<SASL mech="DIGEST-MD5"
  client_name="node_user"
  client_password="node_password"

server_callback_handler_class="org.jgroups.auth.sasl.SimpleAuthorizingCall
backHandler"

client_callback_handler_class="org.jgroups.auth.sasl.SimpleAuthorizingCall
backHandler"
  sasl_props="com.sun.security.sasl.digest.realm=test_realm" />
```

SimpleAuthorizingCallbackHandler は、**java.util.Properties** のインスタンスにコンストラクターを渡すプログラムを用いた方法によるか、または **-DpropertyName=propertyValue** 表記を使用してコマンドラインに設定される標準的な Java システムプロパティーを使用して設定できます。以下のプロパティーを使用できます。

- **sasl.credentials.properties** - principal=password として表されるプリンシパル/クレデンシャルのマッピングが含まれるプロパティーファイルへのパス。
- **sasl.local.principal** - ローカルノードを識別するために使用されるプリンシパルの名前。これは sasl.credentials.properties ファイルに存在している必要があります。
- **sasl.roles.properties** - (オプション) principal=role1,role2,role3 として表されるプリンシパル/ロールのマッピングが含まれるプロパティーファイルへのパス。
- **sasl.role** - (オプション) プリンシパルがある場合にのみノードの参加を承認します (ある場合)。
- **sasl.realm** - (オプション) SASL メカニズムが要求し、使用するレルムの名前。

バグを報告する

26.2.2. ライブラリーモードのノード認証の設定 (DIGEST-MD5)

ノードの動作は、これがコーディネーターノードか他のノードかに応じて異なります。コーディネーターは SASL サーバーとして機能し、参加またはマージするノードは SASL クライアントとして機能します。ライブラリーモードで DIGEST-MD5 メカニズムを使用する場合、サーバーとクライアントのコールバックを指定し、サーバーとクライアントがクレデンシャルの取得方法を認識できるようにする必要があります。そのため、2 つの **CallbackHandler** が必要になります。

- **server_callback_handler_class** はコーディネーターによって使用されます。
- **client_callback_handler_class** は他のノードによって使用されます。

以下の例は、3 つの **CallbackHandler** を示しています。

例26.5 コールバックハンドラー

```
<SASL mech="DIGEST-MD5"
      client_name="node_name"
      client_password="node_password"

client_callback_handler_class="${CLIENT_CALLBACK_HANDLER_IN_CLASSPATH}"

server_callback_handler_class="${SERVER_CALLBACK_HANDLER_IN_CLASSPATH}"
      sasl_props="com.sun.security.sasl.digest.realm=test_realm"
/>
```

JGroups は、すべてのノードがクライアントの動作によってコーディネーターまたはクライアントとして機能できるように設計されています。そのため、現在のコーディネーターノードがダウンする場合、後続チェーンの次のノードがコーディネーターになります。この動作のために、サーバーとクライアントのコールバックハンドラーはどちらも Red Hat JBoss Data Grid 実装の SASL 内で指定される必要があります。

バグを報告する

26.2.3. ライブラリーノードのノード認証の設定 (GSSAPI)

GSSAPI メカニズムを使用してライブラリーモードでノードの認証を実行する際に、***login_module_name*** を ***callback*** の代わりに指定する必要があります。

このログインモジュールは、サーバーに対してクライアントを認証するために使用される、有効な Kerberos チケットを取得するために使用されます。また、クライアントプリンシパルも **jgroups/$server_name@REALM** として構成されるため、***server_name*** を指定する必要もあります。

例26.6 コーディネーターノードでのログインモジュールおよびサーバーの指定

```
<SASL mech="GSSAPI"
         server_name="node0/clustered"
         login_module_name="krb-node0"

server_callback_handler_class="org.infinispan.test.integration.security.
utils.SaslPropCallbackHandler" />
```

コーディネーターノードでは、***server_callback_handler_class*** をノードの承認用に指定する必要があります。これにより、認証された参加ノードがクラスターに参加するパーミッションを持つかどうかが決まります。

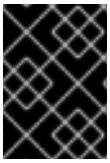
注記

サーバーのプリンシパルは常に **jgroups/server_name** として構成されるため、Kerberos のサーバープリンシパルも **jgroups/server_name** である必要があります。たとえば、Kerberos のサーバー名が **jgroups/node1/mycache** の場合、サーバー名は **node1/mycache** である必要があります。

バグを報告する

26.3. JGROUPS の暗号化

JGroups には、クラスタートラフィックの暗号化を提供するための **SYM_ENCRYPT** および **ASYM_ENCRYPT** プロトコルが含まれます。



重要

ENCRYPT プロトコルが非推奨になったため、本番環境で使用することはできません。**SYM_ENCRYPT** または **ASYM_ENCRYPT** のいずれかを使用することをお勧めします。

デフォルトでは、これらのプロトコルはいずれもメッセージ本体のみを暗号化し、メッセージヘッダーは暗号化しません。宛先および送信元アドレスなどのすべてのヘッダーを含むメッセージ全体を暗号化するには、プロパティー ***encrypt_entire_message*** が **true** でなければなりません。これらのプロトコルを定義する際、**NAKACK2** の下に直接配置する必要があります。

どちらのプロトコルも、JGroups で通信を暗号化および暗号の解除を行うために使用でき、以下の方法で使用されます。

- **SYM_ENCRYPT: JCEKS** ストアタイプを使用してキーストアのシークレットキーで設定されます。
- **ASYM_ENCRYPT:** アルゴリズムおよびキーサイズで設定されます。このシナリオでは、シークレットキーはキーストアから取得されませんが、コーディネーターによって生成され、新規メ

ンバーに配信されます。メンバーがクラスターに参加すると、それらがシークレットキーの要求をコーディネーターに送信し、コーディネーターはメンバーの公開鍵で暗号化された新規メンバーにシークレットキーと共に応答します。

各メッセージは、暗号化ヘッダーを示す特定の暗号化ヘッダーとメッセージの暗号化および暗号化を解除するために使用するキーのバージョンを示す MD5 ダイジェストで暗号化済みとして識別されます。

バグを報告する

26.3.1. JGroups 暗号化プロトコルの設定

JGroups 暗号化プロトコルは JGroups 設定ファイルに置かれます。JBoss Data Grid が使用される方法に応じてこのファイルを組み込むメソッドとして 3 つの方法があります。

- 標準 Java プロパティーを設定で使用することもでき、起動時に JGroups 設定へのパスを **-D** オプションを使用して渡すことができます。
- デフォルトの事前に設定された JGroups ファイルは **infinispan-embedded.jar** にパッケージ化されます。または、独自の設定ファイルを作成できます。ライブラリーモードでカスタム JGroup 設定を使用するために JBoss Data Grid をセットアップする方法については、「JGroups の設定 (ライブラリーモード)」 を参照してください。
- リモートクライアントサーバーモードでは、JGroups 設定はメインサーバーの設置ファイルの一部になります。

SYM_ENCRYPT と **ASYM_ENCRYPT** の両方のプロトコルを定義する際に、それらを設定ファイルの **NAKACK2** の下に直接配置します。

バグを報告する

26.3.2. SYM_ENCRYPT: キーストアの使用

SYM_ENCRYPT はストアタイプ JCEKS を使用します。JCEKS と互換性のあるキーストアを生成するには、以下の keytool のコマンドラインオプションを使用します。

```
$ keytool -genseckey -alias myKey -keypass changeit -storepass changeit -
keyalg Blowfish -keysize 56 -keystore defaultStore.keystore -storetype
JCEKS
```

SYM_ENCRYPT は、以下の情報をアプリケーションで使用される JGroups ファイルに追加して設定できます。

```
<SYM_ENCRYPT sym_algorithm="AES"
             encrypt_entire_message="true"
             keystore_name="defaultStore.keystore"
             store_password="changeit"
             alias="myKey"/>
```

注記

defaultStore.keystore はクラスパスに置かれる必要があります。

バグを報告する

26.3.3. ASYM_ENCRYPT: アルゴリズムとキーサイズによる設定

この暗号化モードでは、コーディネーターが secretKey を選択し、これをすべてのピアに配信します。キーストアはなく、キーは公開/秘密鍵の交換を使用して配信されます。暗号化は以下のように行われます。

1. シークレットキーがコーディネーターにより生成され、配信される。
2. ビューが変更されると、ピアは独自の公開鍵でキーの要求を送信し、シークレットキーを要求する。
3. コーディネーターは公開鍵を使ってシークレットキーを暗号化し、これをピアに送信し戻す。
4. ピアが独自のシークレットキーとしてキーの暗号を解除し、これをインストールする。
5. 追加の通信はシークレットキーを使用して暗号化および暗号化が解除される。

例26.7 ASYM_ENCRYPT の例

```
...
<VERIFY_SUSPECT/>
<ASYM_ENCRYPT encrypt_entire_message="true"
         sym_keylength="128"
         sym_algorithm="AES/ECB/PKCS5Padding"
         asym_keylength="512"
         asym_algorithm="RSA"/>

<pbcast.NAKACK2/>
<UNICAST3/>
<pbcast.STABLE/>
<FRAG2/>
<AUTH auth_class="org.jgroups.auth.MD5Token"
      auth_value="chris"
      token_hash="MD5"/>
<pbcast.GMS join_timeout="2000" />
```

この例では、**ASYM_ENCRYPT** は **NAKACK2** の直下に置かれ、**encrypt_entire_message** が有効にされています。これはメッセージヘッダーがメッセージ本体と共に暗号化されることを示します。つまり、**NAKACK2** および **UNICAST3** プロトコルも暗号化されます。さらに、**AUTH** は設定の一部として組み込まれるので、認証されたノードのみがコーディネーターからシークレットキーを要求できます。

新規コントローラーを特定するビューの変更により、新規のシークレットキーが生成され、すべてのピアに配信されます。これにより、ピアの大幅な変更と共にアプリケーションの大きなオーバーヘッドが生じます。オプションとして、新規のシークレットキーは、**change_key_on_leave** を true に設定することによりクラスターメンバーの退出時に生成することもできます。

メッセージ全体を暗号化する際に、メッセージは暗号化する前にバイトバッファーにマーシャルする必要があります。これによりパフォーマンスが低下します。

バグを報告する

26.3.4. JGroups 暗号化設定パラメーター

以下の表は、**SYM_ENCRYPT** および **ASYM_ENCRYPT** が共に拡張する **ENCRYPT** JGroups プロトコルの設定パラメーターを示しています。

表26.1 ENCRYPT 設定パラメーター

名前	説明
asym_algorithm	非対称アルゴリズムの暗号化エンジンの変換。デフォルトは RSA です。
asym_keylength	初期の公開/秘密鍵の長さ。デフォルトは 512 です。
asym_provider	暗号化サービスプロバイダー。デフォルトは Bouncy Castle Provider です。
encrypt_entire_message	デフォルトでは、メッセージ本体のみが暗号化されます。**encrypt_entire_message** を有効にすると、すべてのヘッダー、宛先および送信元アドレス、およびメッセージ本体が暗号化されます。
sym_algorithm	対称アルゴリズムの暗号化エンジンの変換。デフォルトは AES です。
sym_keylength	一致する対称アルゴリズムの初期のキーの長さ。デフォルトは 128 です。
sym_provider	暗号化サービスプロバイダー。デフォルトは Bouncy Castle Provider です。

以下の表では、**SYM_ENCRYPT** プロトコルパラメーターの一覧を示しています。

表26.2 SYM_ENCRYPT 設定パラメーター

名前	説明
alias	キーの回復に使用されるエイリアス。デフォルトを変更します。
key_password	キーを回復するためのパスワード。デフォルトを変更します。
keystore_name	キーストアリポジトリーが含まれるクラスパス上のファイル。
store_password	整合性のチェックまたはキーストアのロック解除に使用されるパスワード。デフォルトを変更します。

以下の表は、**ASYM_ENCRYPT** プロトコルパラメーターの一覧を示しています。

表26.3 ASYM_ENCRYPT 設定パラメーター

名前	説明
change_key_on_leave	メンバーがビューから出る際にシークレットキーが変更されるため、古いメンバーによる傍受 (eavesdrop)を防ぐことができます。

バグを報告する

パート XIII. コマンドラインツール

Red Hat JBoss Data Grid には、データグリッド内のキャッシュとの対話に使用する 2 つのコマンドラインツールが含まれます。

- JBoss Data Grid Library CLI。詳細については、「Red Hat JBoss Data Grid ライブラリーモード CLI」を参照してください。
- JBoss Data Grid Server CLI。詳細については、「Red Hat Data Grid Server CLI」 を参照してください。

バグを報告する

第27章 RED HAT JBOSS DATA GRID CLI

Red Hat JBoss Data Grid には、ライブラリーモード CLI (詳細については「Red Hat JBoss Data Grid ライブラリーモード CLI」を参照) とサーバーモード CLI (詳細については「Red Hat Data Grid Server CLI」を参照) の 2 つのコマンドラインインターフェースが含まれます。

バグを報告する

27.1. RED HAT JBOSS DATA GRID ライブラリーモード CLI

Red Hat JBoss Data Grid には、(トランザクション、データセンター間のレプリケーションサイト、およびローリングアップグレードなどの) キャッシュや内部コンポーネント内のデータを検査し、変更するために使用する Red Hat JBoss Data Grid ライブラリーモードコマンドラインインターフェース (CLI) が含まれます。JBoss Data Grid ライブラリーモード CLI は、トランザクションなどのさらに高度な操作に使用することもできます。

バグを報告する

27.1.1. ライブラリーモード CLI (サーバー) の起動

Red Hat JBoss Data Grid CLI のサーバー側のモジュールを、**standalone** および **domain** ファイルを使って起動します。Linux の場合、**standlaone.sh** または **domain.sh** スクリプトを使用し、Windows の場合、**standalone.bat** または **domain.bat** ファイルを使用します。

バグを報告する

27.1.2. ライブラリーモード CLI (クライアント) の起動

bin ディレクトリーにある **cli** ファイルを使用して JBoss Data Grid CLI クライアントを起動します。Linux の場合は、**bin/cli.sh** を実行し、Windows の場合は、**bin\cli.bat** を実行します。

CLI クライアントを起動する際、特定のコマンドラインスイッチを使用して起動をカスタマイズすることができます。

バグを報告する

27.1.3. CLI クライアントのコマンドラインスイッチ

リストされているコマンドラインスイッチは、Red Hat JBoss Data Grid CLI コマンドの起動時にコマンドラインに追加されます。

表27.1 CLI クライアントのコマンドラインスイッチ

短縮表記	全表記	説明

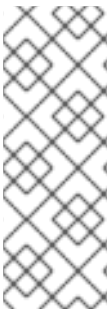
短縮表記	全表記	説明
-c	--connect=${URL}	実行中の Red Hat JBoss Data Grid インスタンスに接続します。たとえば、JMX over RMI の場合、**jmx://[username[:password]]@host:port[/container[/cache]]** を使用し、JMX over JBoss Remoting の場合は **remoting://[username[:password]]@host:port[/container[/cache]]** を使用します。
-f	--file=${FILE}	インタラクティブモードではなく、指定されたファイルからの入力を読み込みます。値が **-** に設定されると、***stdin*** が入力として使用されます。
-h	--help	ヘルプ情報を表示します。
-v	--version	CLI のバージョン情報を表示します。

バグを報告する

27.1.4. アプリケーションへの接続

以下のコマンドを用いて、CLI によりアプリケーションに接続します。

```
[disconnected//]> connect jmx://localhost:12000
[jmx://localhost:12000/MyCacheManager/>
```



注記

ポートの値 **12000** は、JVM が開始される際の値によって変わります。たとえば、JVM を **-Dcom.sun.management.jmxremote.port=12000** コマンドラインパラメーターを使用して開始する場合はこのポートが使用されますが、それ以外の場合はランダムポートが選択されます。リモートプロトコル (**remoting://localhost:9999**) が使用される場合、Red Hat JBoss Data Grid サーバー管理ポートが使用されます (デフォルトはポート **9999**)。

コマンドラインプロンプトは、現在選択されている **CacheManager** を含む、アクティブな接続情報を表示します。

キャッシュ操作を実行する前に、**cache** コマンドを使用してキャッシュを選択します。CLI はタブ補完をサポートしているため、**cache** を使用してタブボタンを押すと、アクティブなキャッシュのリストが表示されます。

```
[[jmx://localhost:12000/MyCacheManager/> cache
___defaultcache  namedCache
[jmx://localhost:12000/MyCacheManager/]> cache ___defaultcache
[jmx://localhost:12000/MyCacheManager/___defaultcache]>
```

さらに、タブを押すと CLI の有効なコマンドのリストが表示されます。

バグを報告する

27.2. RED HAT DATA GRID SERVER CLI

Red Hat JBoss Data Grid には、新しいリモートクライアントサーバーモード CLI が含まれます。この CLI は、サーバーサブシステムの操作などの以下の特定のユースケースにのみ使用できます。

- 設定
- 管理
- メトリックスの取得

バグを報告する

27.2.1. サーバーモード CLI の起動

以下のコマンドを使用してコマンドラインから JBoss Data Grid Server CLI を実行します。

Linux の場合:

```
$ JDG_HOME/bin/cli.sh
```

Windows の場合:

```
C:\>JDG_HOME\bin\cli.bat
```

バグを報告する

27.3. CLI コマンド

特に指定されていない限り、リストされたすべての JBoss Data Grid CLI 用のコマンドは、ライブラリーモードおよびサーバーモード CLI で使用できます。ただし、**deny** (「deny コマンド」を参照)、**grant** (「grant コマンド」を参照)、および**roles** (「roles コマンド」を参照) コマンドは、サーバーモード CLI でのみ利用可能です。

バグを報告する

27.3.1. abort コマンド

abort コマンドは、**start** コマンドを使用して開始された実行中のバッチを中止します。バッチ処理は指定したキャッシュに対して有効にされている必要があります。以下は使用例です。

```
[jmx://localhost:12000/MyCacheManager/namedCache]> start
[jmx://localhost:12000/MyCacheManager/namedCache]> put a a
[jmx://localhost:12000/MyCacheManager/namedCache]> abort
```

```
[jmx://localhost:12000/MyCacheManager/namedCache]> get a
null
```

バグを報告する

27.3.2. begin コマンド

begin コマンドはトランザクションを開始します。このコマンドでは、対象とするキャッシュに対してトランザクションを有効にする必要があります。このコマンドの使用例は次のとおりです。

```
[jmx://localhost:12000/MyCacheManager/namedCache]> begin
[jmx://localhost:12000/MyCacheManager/namedCache]> put a a
[jmx://localhost:12000/MyCacheManager/namedCache]> put b b
[jmx://localhost:12000/MyCacheManager/namedCache]> commit
```

バグを報告する

27.3.3. cache コマンド

cache コマンドは、すべての後続の操作に使用されるデフォルトキャッシュを指定します。パラメーターを指定せずに呼び出されると、現在選択されているキャッシュを表示します。この使用例は次のとおりです。

```
[jmx://localhost:12000/MyCacheManager/namedCache]> cache ___defaultcache
[jmx://localhost:12000/MyCacheManager/___defaultcache]> cache
___defaultcache
[jmx://localhost:12000/MyCacheManager/___defaultcache]>
```

バグを報告する

27.3.4. clearcache コマンド

clearcache コマンドは、キャッシュからすべてのコンテンツをクリアします。この使用例は次のとおりです。

```
[jmx://localhost:12000/MyCacheManager/namedCache]> put a a
[jmx://localhost:12000/MyCacheManager/namedCache]> clearcache
[jmx://localhost:12000/MyCacheManager/namedCache]> get a
null
```

バグを報告する

27.3.5. commit コマンド

commit コマンドは、進行中のトランザクションへの変更をコミットします。この使用例は次のとおりです。

```
[jmx://localhost:12000/MyCacheManager/namedCache]> begin
[jmx://localhost:12000/MyCacheManager/namedCache]> put a a
[jmx://localhost:12000/MyCacheManager/namedCache]> put b b
[jmx://localhost:12000/MyCacheManager/namedCache]> commit
```

バグを報告する

27.3.6. container コマンド

container コマンドはデフォルトのキャッシュコンテナー (キャッシュマネージャー) を選択します。パラメーターを指定せずに呼び出されると、利用可能なすべてのコンテナーをリストします。この使用例は次のとおりです。

```
[jmx://localhost:12000/MyCacheManager/namedCache]> container
MyCacheManager OtherCacheManager
[jmx://localhost:12000/MyCacheManager/namedCache]> container
OtherCacheManager
[jmx://localhost:12000/OtherCacheManager/]>
```

バグを報告する

27.3.7. create コマンド

create コマンドは、既存のキャッシュ定義に基づいて新規のキャッシュを作成します。この使用例は次のとおりです。

```
[jmx://localhost:12000/MyCacheManager/namedCache]> create newCache like
namedCache
[jmx://localhost:12000/MyCacheManager/namedCache]> cache newCache
[jmx://localhost:12000/MyCacheManager/newCache]>
```

バグを報告する

27.3.8. deny コマンド

承認が有効であり、ロールマッパーが ClusterRoleMapper と設定された場合は、ロールマッピングに対するプリンシパルはクラスターレジストリー (すべてのノードで利用可能なレプリケートされたキャッシュ) 内に格納されます。**deny** コマンドは、以前にプリンシパルに割り当てられたロールを拒否するために使用できます。

```
[remoting://localhost:9999]> deny supervisor to user1
```

注記

deny コマンドは JBoss Data Grid サーバーモード CLI でのみ利用できます。

バグを報告する

27.3.9. disconnect コマンド

disconnect コマンドは、現在アクティブな接続を解除します。これにより、CLI は別のインスタンスに接続できます。この使用例は次のとおりです。

```
[jmx://localhost:12000/MyCacheManager/namedCache]> disconnect
[disconnected//]
```

バグを報告する

27.3.10. encoding コマンド

encoding コマンドは、キャッシュから/へのエントリーの読み書きを行う際に使用するデフォルトのコーデックを設定します。引数なしで呼び出される場合、現在選択されているコーデックが表示されます。この使用例は次のとおりです。

```
[jmx://localhost:12000/MyCacheManager/namedCache]> encoding
none
[jmx://localhost:12000/MyCacheManager/namedCache]> encoding --list
memcached
hotrod
none
rest
[jmx://localhost:12000/MyCacheManager/namedCache]> encoding hotrod
```

バグを報告する

27.3.11. end コマンド

end コマンドは、 **start** コマンドを使用して開始された実行中のバッチを終了します。この使用例は次のとおりです。

```
[jmx://localhost:12000/MyCacheManager/namedCache]> start
[jmx://localhost:12000/MyCacheManager/namedCache]> put a a
[jmx://localhost:12000/MyCacheManager/namedCache]> end
[jmx://localhost:12000/MyCacheManager/namedCache]> get a
a
```

バグを報告する

27.3.12. evict コマンド

evict コマンドは、キャッシュから特定のキーに関連付けられたエントリーをエビクトします。この使用例は次のとおりです。

```
[jmx://localhost:12000/MyCacheManager/namedCache]> put a a
[jmx://localhost:12000/MyCacheManager/namedCache]> evict a
```

バグを報告する

27.3.13. get コマンド

get コマンドは、指定されたキーと関連付けられている値を表示します。プリミティブ型および 文字列の場合に、**get** コマンドはデフォルト表現のみを表示します。他のオブジェクトの場合、オブジェクトの JSON 表現が表示されます。この使用例は次のとおりです。

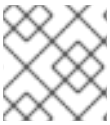
```
[jmx://localhost:12000/MyCacheManager/namedCache]> put a a
[jmx://localhost:12000/MyCacheManager/namedCache]> get a
a
```

バグを報告する

27.3.14. grant コマンド

承認が有効であり、ロールマッパーが **ClusterRoleMapper** と設定された場合は、ロールマッピングに対するプリンシパルはクラスターレジストリー (すべてのノードで利用可能なレプリケートされたキャッシュ) 内に格納されます。**grant** コマンドは、以下のようにプリンシパルに新しいロールを与えるために使用できます。

```
[remoting://localhost:9999]> grant supervisor to user1
```



注記

grant コマンドは JBoss Data Grid サーバーモード CLI でのみ利用できます。

バグを報告する

27.3.15. info コマンド

info コマンドは選択されたキャッシュまたはコンテナーの設定を表示します。この使用例は次のとおりです。

```
[jmx://localhost:12000/MyCacheManager/namedCache]> info
GlobalConfiguration{asyncListenerExecutor=ExecutorFactoryConfiguration{fac
tory=org.infinispan.executors.DefaultExecutorFactory@98add58},
asyncTransportExecutor=ExecutorFactoryConfiguration{factory=org.infinispan
.executors.DefaultExecutorFactory@7bc9c14c},
evictionScheduledExecutor=ScheduledExecutorFactoryConfiguration{factory=or
g.infinispan.executors.DefaultScheduledExecutorFactory@7ab1a411},
replicationQueueScheduledExecutor=ScheduledExecutorFactoryConfiguration{fa
ctory=org.infinispan.executors.DefaultScheduledExecutorFactory@248a9705},
globalJmxStatistics=GlobalJmxStatisticsConfiguration{allowDuplicateDomains
=true, enabled=true, jmxDomain='jboss.infinispan',
mBeanServerLookup=org.jboss.as.clustering.infinispan.MBeanServerProvider@6
c0dc01, cacheManagerName='local', properties={}},
transport=TransportConfiguration{clusterName='ISPN', machineId='null',
rackId='null', siteId='null', strictPeerToPeer=false,
distributedSyncTimeout=240000, transport=null, nodeName='null',
properties={}},
serialization=SerializationConfiguration{advancedExternalizers=
{1100=org.infinispan.server.core.CacheValue$Externalizer@5fabc91d,
1101=org.infinispan.server.memcached.MemcachedValue$Externalizer@720bffd,
1104=org.infinispan.server.hotrod.ServerAddress$Externalizer@771c7eb2},
marshaller=org.infinispan.marshall.VersionAwareMarshaller@6fc21535,
version=52,
classResolver=org.jboss.marshalling.ModularClassResolver@2efe83e5},
shutdown=ShutdownConfiguration{hookBehavior=DONT_REGISTER}, modules={},
site=SiteConfiguration{localSite='null'}}
```

バグを報告する

27.3.16. locate コマンド

locate コマンドは、分散クラスター内の指定されたエントリーの物理的な場所を表示します。この使用例は次のとおりです。

```
[jmx://localhost:12000/MyCacheManager/namedCache]> locate a
[host/node1,host/node2]
```

バグを報告する

27.3.17. put コマンド

put コマンドはエントリーをキャッシュに挿入します。キーに対するマッピングが存在する場合、**put** コマンドは古い値を上書きします。CLI により、キーと値を保存するために使用されるデータのタイプに対して制御が可能になります。この使用例は次のとおりです。

```
[jmx://localhost:12000/MyCacheManager/namedCache]> put a a
[jmx://localhost:12000/MyCacheManager/namedCache]> put b 100
[jmx://localhost:12000/MyCacheManager/namedCache]> put c 4139l
[jmx://localhost:12000/MyCacheManager/namedCache]> put d true
[jmx://localhost:12000/MyCacheManager/namedCache]> put e {
"package.MyClass": {"i": 5, "x": null, "b": true } }
```

オプションとして、**put** は次のようにライフスパンと最大アイドル時間の値を指定することができます。

```
[jmx://localhost:12000/MyCacheManager/namedCache]> put a a expires 10s
[jmx://localhost:12000/MyCacheManager/namedCache]> put a a expires 10m
maxidle 1m
```

バグを報告する

27.3.18. replace コマンド

replace コマンドはキャッシュ内の既存のエントリーを指定した新しい値に置き換えます。この使用例は次のとおりです。

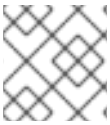
```
[jmx://localhost:12000/MyCacheManager/namedCache]> put a a
[jmx://localhost:12000/MyCacheManager/namedCache]> replace a b
[jmx://localhost:12000/MyCacheManager/namedCache]> get a
b
[jmx://localhost:12000/MyCacheManager/namedCache]> replace a b c
[jmx://localhost:12000/MyCacheManager/namedCache]> get a
c
[jmx://localhost:12000/MyCacheManager/namedCache]> replace a b d
[jmx://localhost:12000/MyCacheManager/namedCache]> get a
c
```

バグを報告する

27.3.19. roles コマンド

承認が有効であり、ロールマッパーが **ClusterRoleMapper** と設定された場合は、ロールマッピングに対するプリンシパルはクラスターレジストリー (すべてのノードで利用可能なレプリケートされたキャッシュ) 内に格納されます。**roles** コマンドは、特定のユーザーまたはすべてのユーザー (ユーザーが指定されていない場合) に関連付けられたロールをリストするために使用できます。

```
[remoting://localhost:9999]> roles user1
[supervisor, reader]
```



注記

roles コマンドは JBoss Data Grid サーバーモード CLI でのみ利用できます。

バグを報告する

27.3.20. rollback コマンド

rollback コマンドは、進行中のトランザクションによる変更をロールバックします。この使用例は次のとおりです。

```
[jmx://localhost:12000/MyCacheManager/namedCache]> begin
[jmx://localhost:12000/MyCacheManager/namedCache]> put a a
[jmx://localhost:12000/MyCacheManager/namedCache]> put b b
[jmx://localhost:12000/MyCacheManager/namedCache]> rollback
```

バグを報告する

27.3.21. site コマンド

site コマンドは、データセンター間レプリケーションに関連する管理タスクを実行します。また、このコマンドはサイトの状態についての情報を取得し、サイトの状態を切り替えます。この使用例は次のとおりです。

```
[jmx://localhost:12000/MyCacheManager/namedCache]> site --status NYC
online
[jmx://localhost:12000/MyCacheManager/namedCache]> site --offline NYC
ok
[jmx://localhost:12000/MyCacheManager/namedCache]> site --status NYC
offline
[jmx://localhost:12000/MyCacheManager/namedCache]> site --online NYC
```

バグを報告する

27.3.22. start コマンド

start コマンドは、操作のバッチを開始します。この使用例は次のとおりです。

```
[jmx://localhost:12000/MyCacheManager/namedCache]> start
[jmx://localhost:12000/MyCacheManager/namedCache]> put a a
[jmx://localhost:12000/MyCacheManager/namedCache]> put b b
[jmx://localhost:12000/MyCacheManager/namedCache]> end
```

バグを報告する

27.3.23. stats コマンド

stats コマンドはキャッシュの統計を表示します。この使用例は次のとおりです。

```
[jmx://localhost:12000/MyCacheManager/namedCache]> stats
Statistics: {
  averageWriteTime: 143
  evictions: 10
  misses: 5
  hitRatio: 1.0
  readWriteRatio: 10.0
  removeMisses: 0
  timeSinceReset: 2123
  statisticsEnabled: true
  stores: 100
  elapsedTime: 93
  averageReadTime: 14
  removeHits: 0
  numberOfEntries: 100
  hits: 1000
}
LockManager: {
  concurrencyLevel: 1000
  numberOfLocksAvailable: 0
  numberOfLocksHeld: 0
}
```

バグを報告する

27.3.24. upgrade コマンド

upgrade コマンドは、ローリングアップグレードの手順を実装します。ローリングアップグレードの詳細については、36章*ローリングアップグレード*を参照してください。

upgrade コマンドの使用例は次のとおりです。

```
[jmx://localhost:12000/MyCacheManager/namedCache]> upgrade --
synchronize=hotrod --all
[jmx://localhost:12000/MyCacheManager/namedCache]> upgrade --
disconnectsource=hotrod --all
```

バグを報告する

27.3.25. version コマンド

version コマンドは、CLI クライアントおよびサーバーのバージョン情報を表示します。この使用例は次のとおりです。

```
[jmx://localhost:12000/MyCacheManager/namedCache]> version
Client Version 5.2.1.Final
Server Version 5.2.1.Final
```

バグを報告する

パート XIV. 他の RED HAT JBOSS DATA GRID 機能

第28章 1 次キャッシュのセットアップ

28.1. 1 次キャッシュについて

1 次 (L1) キャッシュは、最初にアクセスされた後にリモートキャッシュエントリーを格納するため、同じエントリーがその後使用される時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。1 次キャッシュは、Red Hat JBoss Data Grid のキャッシュモードがディストリビューションに設定されている場合にのみ利用できます。他のキャッシュモードでは、1 次キャッシュに関連するいずれの設定も無視されます。

キャッシュがディストリビューションモードで設定される場合、エントリーがすべてのクラスター化されたキャッシュ間で均等に分散されます。それぞれのエントリーは、必要な数の所有者にコピーされ、その数はキャッシュの合計数より小さくなる可能性があります。その結果、システムのスケーラビリティーが改善されるだけでなく、いくつかのエントリーがすべてのノードで利用できなくなり、それらの所有者ノードから取り込まれる必要があります。この状態では、後続のユーザーに対して繰り返される取り込みを避けるために、キャッシュコンポーネントを、所有しないエントリーを一時的に保存するための 1 次キャッシュを使用できるように設定します。

キーが更新されるたびに、インバリデーションメッセージが生成されます。このメッセージは、現在の 1 次キャッシュエントリーに対応するデータを含むそれぞれのノードのマルチキャストです。インバリデーションメッセージにより、それらの各ノードは、関連するエントリーを無効なものとしてマークします。さらに、エントリーのロケーションがクラスター内で変更されると、対応する 1 次キャッシュエントリーは無効にされ、古くなったキャッシュエントリーが発生しないようにします。

バグを報告する

28.2. 1 次キャッシュの設定

28.2.1. 1 次キャッシュの設定 (ライブラリーモード)

次の設定例は、Red Hat JBoss Data Grid のライブラリーモードにおける 1 次キャッシュのデフォルト値を示しています。

例28.1 ライブラリーモードの 1 次キャッシュ設定

```
<distributed-cache name="distributed_cache"
        owners="2"
        l1-lifespan="0"
        l1-cleanup-interval="60000"/>
```

以下の属性は 1 次キャッシュの動作を制御します。

- ***l1-lifespan*** 属性は、1 次キャッシュに配置されるエントリーの最大ライフスパンをミリ秒単位で示します。これは非分散キャッシュでは許可されません。L1 のデフォルトでは、この値は 1 次キャッシュが無効にされていることを示す 0 になり、正の値が定義される場合にのみ有効にされます。
- ***li-cleanup-interval*** パラメーターは、1 次トラッキングデータを除去するクリーンアップタスクが実行される頻度をミリ秒単位で制御します。これはデフォルトで 10 分に定義されます。

バグを報告する

28.2.2.1 1次キャッシュの設定 (リモートクライアントサーバーモード)

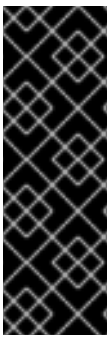
次の設定例は、1次キャッシュのデフォルト値 **0** を示しています。この値は、このキャッシュが Red Hat JBoss Data Grid のリモートクライアントサーバーモードで無効にされていることを示しています。

例28.2 リモートクライアントサーバーモード用1次キャッシュの設定

```
<distributed-cache l1-lifespan="0">
 <!-- Additional configuration information here -->
</distributed-cache>
```

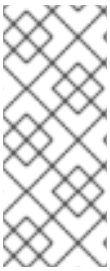
l1-lifespan 要素は、1次キャッシュを有効にし、キャッシュの1次キャッシュエントリーについてのライフスパンを設定するために **distributed-cache** 要素に追加されます。この要素は、分散キャッシュにのみ有効です。

l1-lifespan が **0** または負の数値 **(-1)** に設定される場合、1次キャッシュは無効になります。1次キャッシュは、**l1-lifespan** の値が **0** より大きくなる場合に有効になります。



重要

キャッシュが Hot Rod プロトコル経由でリモートでアクセスされる場合、クライアントは所有者ノードに直接アクセスします。したがって、Hot Rod プロトコル経由で1次キャッシュを使用することは推奨されません。代わりに、JBoss Data Grid の『Developer Guide』の「Near Caching」のセクションを参照してください。他のリモートクライアント (Memcached、REST) は所有者をターゲットにしないため、1次キャッシュを使用するとパフォーマンスが向上します (同時に高いメモリー使用量が発生します)。



注記

リモートクライアントサーバーモードで、1次キャッシュは ***l1-lifespan*** 属性が設定されていない場合であっても分散キャッシュが使用されたときにデフォルトで有効になります。デフォルトのライフスパン値は 10 分です。JBoss Data Grid 6.3 以降、デフォルトのライフスパンは 0 であり 1次キャッシュが無効になります。***l1-lifespan*** パラメーターにゼロ以外の値を設定して1次キャッシュを有効にします。

バグを報告する

第29章 トランザクションのセットアップ

29.1. トランザクション

トランザクションは、相互に依存しているか、または関連のある操作またはタスクのコレクションで構成されています。単一トランザクション内のすべての操作が成功しないと、トランザクションの全体の成功にはつながりません。トランザクション内のいずれかの操作が失敗すると、トランザクションは全体として失敗し、すべての変更をロールバックします。トランザクションは、大規模な操作の一部として一連の変更を処理する場合にとくに役立ちます。

Red Hat JBoss Data Grid では、トランザクションはライブラリーモードでのみ利用可能です。

バグを報告する

29.1.1. トランザクションマネージャーについて

Red Hat JBoss Data Grid では、トランザクションマネージャーは、単一または複数のリソースにまたがってトランザクションを調整します。トランザクションマネージャーの役割には以下が含まれます。

- トランザクションの開始および終了
- 各トランザクションについての情報の管理
- トランザクションが複数リソースにまたがって動作する際のトランザクションの調整
- 変更のロールバックによる、失敗したトランザクションからのリカバリー

バグを報告する

29.1.2. XA リソースおよび同期

XA リソースは独立したトランザクション要素です。準備段階で (詳細については、「2 相コミット (2PC) について」を参照)、XA リソースは、値が **OK** または **ABORT** のいずれかの投票を返します。トランザクションマネージャーがすべての XA リソースから **OK** 投票を受信する場合、トランザクションはコミットされ、それ以外の場合にはロールバックされます。

同期は、トランザクションのライフサイクルにつながるイベントについての通知を受信するリスナーの 1 つのタイプです。同期は、操作の終了前後にイベントを受信します。

リカバリーが必要ではない場合、完全な XA リソースとして登録する必要はありません。同期の利点には、同期により、トランザクションマネージャーが、その他の 1 つのリソースのみがそのトランザクションでリストされる 1 相コミット (1PC) で 2 相コミット (2PC) を最適化できる点があります (最終リソースコミット最適化)。これにより、同期をより効率的にすることができます。

ただし、Red Hat JBoss Data Grid 内の準備段階で操作が失敗する場合、トランサクションはロールバックされず、トランザクションにより多くの参加者が存在する場合、それらはその失敗を無視し、コミットできます。さらに、コミット段階で発生するエラーは、トランザクションをコミットするアプリケーションコードに伝搬されません。

デフォルトで、JBoss Data Grid は同期としてトランザクションに登録されます。

バグを報告する

29.1.3. 楽観的トランザクションと悲観的トランザクション

悲観的トランザクションは、キー上で最初の書き込み操作が実行される際にロックを取得します。キーがロックされた後は、このトランザクションがコミットされるか、またはロールバックされるまでその他のトランザクションはキーを変更することができません。デッドロックを回避するために正しい順番でロックを取得できるかどうかは、アプリケーションコードによります。

楽観的トランザクションの場合、ロックはトランザクションの準備時間に取得され、トランザクションがコミット (またはロールバック) するまで保持されます。さらに、Red Hat JBoss Data Grid は、トランザクション内の変更されたすべてのエントリーについてキーを自動的にソートし、ロックされているキーの順序が正しくないために生じるデッドロックを回避します。この結果は以下のようになります。

- トランザクションの実行時に送信されるメッセージが少なくなる
- ロックの保持期間が短くなる
- スループットが改善する

注記

読み取り操作ではロックを一切取得しません。オンデマンドで読み取り操作のロックを取得することは、この操作と共に **FORCE_WRITE_LOCK** フラグを使用した場合の悲観的トランザクションでのみ可能になります。

バグを報告する

29.1.4. 書き込みスキューのチェック

エントリーの共通のユースケースとして、エントリーはまず読み取られ、その後トランザクションで書き込みが行なわれます。ただし、3 つ目のトランザクションが、これら 2 つの操作の間にエントリーを変更する可能性があります。このような状況を検出し、トランザクションをロールバックするために、Red Hat JBoss Data Grid は、エントリーのバージョン管理と書き込みスキューのチェックを行います。変更されたバージョンがトランザクション時に最後に読み取られたバージョンと同じでない場合、書き込みスキューチェックは例外をスローし、トランザクションはロールバックされます。

書き込みスキューのチェックを有効にするには、**REPEATABLE_READ** の分離レベルが必要です。さらに、クラスターモード (ディストリビューションモードまたはレプリケーションモード) で、エントリーのバージョン管理をセットアップします。ローカルモードの場合、エントリーのバージョン管理は不要です。

重要

楽観的トランザクションの場合、書き込みスキューのチェックは、(アトミックな) 条件操作で必要になります。

バグを報告する

29.1.5. 複数のキャッシュインスタンス間でのトランザクション

各キャッシュは個別のスタンドアロン Java Transaction API (JTA) リソースとして動作します。ただし、コンポーネントは最適化のために Red Hat JBoss Data Grid で内部的に共有できますが、この共有は、キャッシュの Java Transaction API (JTA) Manager との対話方法には影響を与えません。

バグを報告する

29.2. トランザクションの設定

29.2.1. トランザクションの設定 (ライブラリーモード)

Red Hat JBoss Data Grid では、ライブラリーモードのトランザクションは、同期化およびトランザクションリカバリーと共に設定できます。トランザクションは全体として (同期化およびトランザクションリカバリーを含む)、リモートクライアントサーバーモードで使用することはできません。

キャッシュ操作を実行するため、キャッシュには環境のトランザクションマネージャーへの参照が必要となります。**TransactionManagerLookup** インターフェースの実装に属するクラス名を用いて、キャッシュを設定します。キャッシュが初期化されると、指定クラスのインスタンスが作成され、**getTransactionManager()** メソッドを呼び出してトランザクションマネージャーへの参照を見つけ、これを返します。

ライブラリーモードでは、トランザクションは以下のように設定されます。

手順29.1 ライブラリーモードでのトランザクションの設定 (XML 設定)

```
<local-cache name="default" <!-- Additional configuration information here
-->>
 <transaction mode="BATCH"
     stop-timeout="60000"
     auto-commit="true"
     protocol="DEFAULT"
     recovery-cache="recoveryCache">
 <locking <!-- Additional configuration information here --> >
 <versioning versioningScheme="SIMPLE"/>
  <!-- Additional configuration information here -->
</local-cache>
```

1. ***mode*** を定義して ***transactions*** を有効にします。デフォルトでモードは **NONE** であるため、トランザクションは無効になります。有効なトランザクションのモードは **BATCH**、**NON_XA**、**NON_DURABLE_XA**、**FULL_XA** です。

2. ***stop-timeout*** を定義し、キャッシュの停止時に継続しているトランザクションがある場合にインスタンスはその継続しているトランザクションが終了するのを待機できるようにします。デフォルトは **30000** ミリ秒に設定されます。

3. ***auto-commit*** を有効にし、単一操作のトランザクションを手動で開始しなくても済むようにします。デフォルトは **true** に設定されます。

4. 使用されているコミットの ***protocol*** を定義します。有効なコミットプロトコルは **DEFAULT** および **TOTAL_ORDER** です。

5. リカバリー関連情報が保持される **recovery-cache** の名前を定義します。デフォルトは **__recoveryInfoCacheName__** に設定されます。

6. ***versioningScheme*** 属性を **SIMPLE** として定義してエントリーの ***versioning*** を有効にします。デフォルトは **NONE** に設定され、これはバージョン管理が無効にされていることを示します。

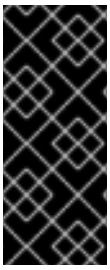
バグを報告する

29.2.2. トランザクションの設定 (リモートクライアントサーバーモード)

Red Hat JBoss Data Grid は、リモートクライアントサーバーモードでトランザクションを提供しません。デフォルトで、唯一サポートされている設定は、以下のようなトランザクションではない設定です。

例29.1 リモートクライアントサーバーモードでのトランザクション設定

```
<cache>
 <!-- Additional configuration elements here -->
  <transaction mode="NONE" />
 <!-- Additional configuration elements here -->
</cache>
```



重要

リモートクライアントサーバーモードでは、JBoss Data Grid が互換モードで使用され、クラスターに JBoss Data Grid サーバーインスタンスとライブラリーインスタンスの両方が含まれない限り、トランザクション要素が **NONE** に設定されます。このときにトランザクションがライブラリーモードインスタンスで設定される場合は、サーバーインスタンスでもトランザクションを設定する必要があります。

バグを報告する

29.3. トランザクションリカバリー

トランザクションマネージャーはリカバリー処理を調整し、操作の完了に手作業の介入が必要となるトランザクションを判断するため Red Hat JBoss Data Grid と共に動作します。この処理はトランザクションリカバリーと呼ばれます。

JBoss Data Grid は、トランザクションのコミットまたはロールバックを明示的に強制する操作に対して JMX を使用します。これらのメソッドは関連するトランザクションに関連付けられた数の代わりに XID を定義するバイトアレイを受け取ります。

システム管理者はこのような JMX 操作を使用して、手動介入が必要なトランザクションに対して自動的にジョブを完了できます。このプロセスでは、トランザクションマネージャーのトランザクションリカバリープロセスを使用し、トランザクションマネージャーの XID オブジェクトにアクセスできます。

バグを報告する

29.3.1. トランザクションリカバリーのプロセス

次のプロセスは、Red Hat JBoss Data Grid のトランザクションリカバリーのプロセスを簡単に説明しています。

手順29.2 トランザクションリカバリーのプロセス

1. トランザクションマネージャーは介入が必要なトランザクションの一覧を作成します。
2. 電子メールまたはログを使用して、JMX を使用して JBoss Data Grid に接続するシステム管理者へトランザクション (トランザクション ID を含む) の一覧を提供します。各トランザクショ

ンの状態は **COMMITTED** か **PREPARED** になります。**COMMITTED** と **PREPARED** の両方の状態であるトランザクションがある場合、トランザクションがあるノード上でコミットされている一方、他のノードで準備状態のトランザクションがあることを示しています。

3. システム管理者は、トランザクションマネージャーより受信した XID を JBoss Data Grid の内部 ID へ視覚的にマッピングします。XID (バイトアレイ) を JMX ツールに渡し、このマッピングがない状態で JBoss Data Grid によって再アセンブルすることはできないため、この手順が必要となります。

4. マッピングされた内部 ID を基に、システム管理者はトランザクションに対してコミットまたはロールバックプロセスを強制します。

バグを報告する

29.3.2. トランザクションリカバリーの例

以下の例は、現金がデータベースに格納された口座から Red Hat JBoss Data Grid に格納された口座に転送される状況でどのようにトランザクションが使用されるかを示しています。

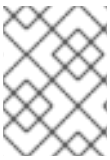
例29.2 データベースに格納された口座から JBoss Data Grid 内の口座への送金

1. 送信元 (データベース) と送信先 (JBoss Data Grid) リソース間の 2 フェーズコミットプロトコルを実行するために、`TransactionManager.commit()` メソッドが呼び出されます。

2. `TransactionManager` が、準備フェーズ (2 フェーズコミットの最初のフェーズ) を開始するようデータベースと JBoss Data Grid に指示します。

コミットフェーズ中、データベースは変更を適用しますが、JBoss Data Grid はトランザクションマネージャーのコミット要求を受け取る前に失敗します。結果として、不完全なトランザクションのためにシステムは不整合な状態になります。とくに送金される金額はデータベースから引かれますが、準備された変更を適用できないため、JBoss Data Grid に表示されません。

ここでは、トランザクションリカバリーはデータベースと JBoss Data Grid エントリー間の不整合を調整するために使用されます。



注記

JMX を使用してトランザクションリカバリーを管理するには、JMX サポートを明示的に有効にする必要があります。

バグを報告する

29.4. デッドロックの検出

デッドロックは、複数のプロセスまたはタスクが他のプロセスまたはタスクが相互に必要なリソースを解放するまで待つときに発生します。デッドロックにより、特に複数のトランザクションが 1 つのキーセットに対して動作する場合にシステムのスループットが大幅に短縮されることがあります。

Red Hat JBoss Data Grid は、このようなデッドロックを識別するデッドロック検出を提供します。デッドロック検出は、デフォルトで有効にされます。

バグを報告する

29.4.1. デッドロック検出を有効にする

Red Hat JBoss Data Grid でのデッドロック検出はデフォルトで有効にされ、以下のように ***cache*** 設定要素の ***deadlock-detection-spin*** 属性を調整して設定できます。

```
<local-cache [...] deadlock-detection-spin="1000"/>
```

deadlock-detection-spin 属性は、特定ロックの取得が許可される最大時間 (ミリ秒単位) 内にロックの取得を試行する頻度を定義します。この値はデフォルトで **100** ミリ秒になり、負の値はデッドロック検出を無効にします。

デッドロック検出は個別のキャッシュにのみ適用できます。JBoss Data Grid は複数のキャッシュに適用されたデッドロックを検出できません。

バグを報告する

第30章 JGROUPS の設定

JGroups は、Red Hat JBoss Data Grid インスタンスに接続するために使用される基礎となるグループ通信ライブラリーです。JBoss Data Grid でサポートされる JGroups プロトコルの完全なリストについては、「サポート対象 JGroups プロトコル」を参照してください。

バグを報告する

30.1. RED HAT JBOSS DATA GRID インターフェースバインディングの設定 (リモートクライアントサーバーモード)

30.1.1. インターフェース

Red Hat JBoss Data Grid では、ユーザーは特定の (不明の) IP アドレスではなくインターフェースタイプを指定することができます。

- ***link-local***: **169.*x*.*x*.*x*** または **254.*x*.*x*.*x*** アドレスを使用します。これは、1 つのボックス内のトラフィックに適しています。

```
<interfaces>
    <interface name="link-local">
        <link-local-address/>
    </interface>
    <!-- Additional configuration elements here -->
</interfaces>
```

- ***site-local***: たとえば **192.168.*x*.*x*** などのプライベート IP アドレスを使用します。これにより、GoGrid や同様のプロバイダーから追加の帯域幅についてチャージされることを避けられます。

```
<interfaces>
    <interface name="site-local">
        <site-local-address/>
    </interface>
    <!-- Additional configuration elements here -->
</interfaces>
```

- ***global***: パブリック IP アドレスを選択します。これは、レプリケーショントラフィックの場合は避ける必要があります。

```
<interfaces>
    <interface name="global">
        <any-address/>
    </interface>
    <!-- Additional configuration elements here -->
</interfaces>
```

- ***non-loopback***: **127.*x*.*x*.*x*** アドレスではないアクティブなインターフェースにある最初のアドレスを使用します。

```
<interfaces>
    <interface name="non-loopback">
```

```
        <not>
      <loopback />
  </not>
     </interface>
</interfaces>
```

バグを報告する

30.1.2. ソケットのバインディグ

ソケットのバインディングにより、インターフェースとポートの名前付きの組み合わせが提供されます。ソケットは、個別にまたはソケットのバインディンググループを使用するかのいずれかにより、インターフェースにバインドできます。

バグを報告する

30.1.2.1. 単一のソケットをバインドする例

以下は、JGroups インターフェースのソケットバインディングを使用し、***socket-binding*** 要素を用いて個別のソケットをバインドする例を表しています。

例30.1 ソケットバインディング (Socket Binding)

```
<socket-binding name="jgroups-udp" <!-- Additional configuration
elements here --> interface="site-local"/>
```

バグを報告する

30.1.2.2. ソケットのグループをバインドする例

以下は、JGroups インターフェースのソケットバインディングを使用し、***socket-binding-group*** 要素を用いてグループをバインドする例を表しています。

例30.2 グループのバインド

```
<socket-binding-group name="ha-sockets" default-interface="global">
 <!-- Additional configuration elements here -->
 <socket-binding name="jgroups-tcp" port="7600"/>
 <socket-binding name="jgroups-tcp-fd" port="57600"/>
 <!-- Additional configuration elements here -->
</socket-binding-group>
```

この例の 2 つのサンプルのソケットバインディングは、同じ ***default-interface*** (**global**) にバインドされます。そのため、インターフェース属性を指定する必要はありません。

バグを報告する

30.1.3. JGroups ソケットバインディングの設定

JGroups サブシステムで設定されるそれぞれの JGroups スタックでは、特定のソケットバインディングを使用します。以下のようにソケットバインディングをセットアップします。

例30.3 JGroups UDP ソケットバインディング設定

以下の例では、クラスターから UDP を自動的に使用します。この例では、**jgroups-udp** ソケットバインディングがトランスポートに定義されます。

```
<subsystem xmlns="urn:jboss:domain:jgroups:3.0" default-stack="udp">
    <stack name="udp">
        <transport type="UDP" socket-binding="jgroups-udp">
            <!-- Additional configuration elements here -->
        </transport>
        <protocol type="PING"/>
        <protocol type="MERGE3"/>
        <protocol type="FD_SOCK" socket-binding="jgroups-udp-fd"/>
        <protocol type="FD_ALL"/>
        <protocol type="VERIFY_SUSPECT"/>
        <protocol type="pbcast.NAKACK2"/>
        <protocol type="UNICAST3"/>
        <protocol type="pbcast.STABLE"/>
        <protocol type="pbcast.GMS"/>
        <protocol type="UFC"/>
        <protocol type="MFC"/>
        <protocol type="FRAG2"/>
    </stack>
</subsystem>
```

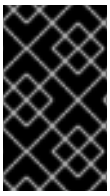
例30.4 JGroups TCP ソケットバインディング設定

以下の例では、2 つのクラスターノード間の直接的な通信を確立するために TCP を使用します。以下の例では、node1 は 192.168.1.2:7600 にあり、node2 は 192.168.1.3:7600 にあります。使用されているポートは、**socket-binding** セクションの jgroups-tcp プロパティーで定義されます。

```
<subsystem xmlns="urn:infinispan:server:jgroups:8.0" default-
stack="tcp">
    <stack name="tcp">
        <transport type="TCP" socket-binding="jgroups-tcp"/>
        <protocol type="TCPPING">
            <property
name="initial_hosts">192.168.1.2[7600],192.168.1.3[7600]</property>
            <property name="num_initial_members">2</property>
            <property name="port_range">0</property>
            <property name="timeout">2000</property>
        </protocol>
        <protocol type="MERGE3"/>
        <protocol type="FD_SOCK" socket-binding="jgroups-tcp-fd"/>
        <protocol type="FD_ALL"/>
        <protocol type="VERIFY_SUSPECT"/>
        <protocol type="pbcast.NAKACK2">
            <property name="use_mcast_xmit">false</property>
        </protocol>
        <protocol type="UNICAST3"/>
        <protocol type="pbcast.STABLE"/>
        <protocol type="pbcast.GMS"/>
        <protocol type="MFC"/>
```

```
        <protocol type="FRAG2"/>
    </stack>
</subsystem>
```

UDP または TCP のどちらを使用するかはそれぞれの環境で決定されます。デフォルトで、JGroups は UDP を使用します。UDP はクラスター化されたメンバーの動的な検出を可能にし、またネットワークフットプリントが比較的小さいため大型クラスターで効果的に拡張できるためです。さらに UDP を使用する場合は、マルチキャストパケットがマルチキャストアドレスへのすべてのサブスクライバーによって受信されるため、クラスターごとに 1 つのパケットのみが必要になります。ただし、マルチキャストトラフィックが阻まれる環境か、またはクラスターメンバーが別の VLAN に置かれている場合など、UDP トラフィックがリモートクラスターノードに達しない場合は、クラスターを作成するために TCP トラフィックを使用できます。



重要

UDP を JGroups トランスポートとして使用する場合、ソケットバインディングは、通常の (ユニキャスト) ポート、マルチキャストアドレス、およびマルチキャストポートを指定する必要があります。

バグを報告する

30.2. JGROUPS の設定 (ライブラリーモード)

Red Hat JBoss Data Grid がクラスターモードで動作するには、適切な JGroups 設定が必要になります。

例30.5 JGroups XML 設定

```
<infinispan xmlns="urn:infinispan:config:8.3">
    <jgroups>
        <stack-file name="jgroupsStack"
path="/path/to/jgroups/xml/jgroups.xml}"/>
    </jgroups>
    <cache-container name="default" default-cache="default">
        <transport stack="jgroupsStack" lock-timeout="600000"
cluster="default" />
    </cache-container>
</infinispan>
```

まず JBoss Data Grid はクラスパスで **`jgroups.xml`** を検索します。インスタンスがクラスパスに見つからない場合、絶対パス名を探します。

バグを報告する

30.2.1. JGroups トランスポートプロトコル

トランスポートプロトコルは、プロトコルスタックの底辺にあるプロトコルです。トランスポートプロトコルは、ネットワークとのメッセージの送受信を行います。

Red Hat JBoss Data Grid には、UDP と TCP トランスポートプロトコルの両方が同梱されています。

バグを報告する

30.2.1.1. UDP トランスポートプロトコル

UDP は、以下を使用するトランスポートプロトコルです。

- クラスターのすべてのメンバーにメッセージを送信する IP マルチキャスト。
- 単一メンバーに送信されるユニキャストメッセージの UDP データグラム。

UDP トランスポートが開始されると、ユニキャストソケットとマルチキャストソケットが開きます。ユニキャストソケットは、ユニキャストメッセージの送受信に使用され、マルチキャストソケットは、マルチキャストソケットの送受信を行います。チャネルの物理アドレスは、ユニキャストソケットのアドレスおよびポート番号と同じです。

バグを報告する

30.2.1.2. TCP トランスポートプロトコル

TCP/IP は、IP マルチキャストが使用できなくなる状況で使用できる UDP の代替トランスポートです。このような状況には、ルーターが IP マルチキャストパケットを破棄する可能性のある WAN 上の操作が実行される場合などが含まれます。

TCP は、ユニキャストおよびマルチキャストメッセージを送信するために使用されるトランスポートプロトコルです。

- マルチキャストメッセージを送信する場合に、TCP は複数のユニキャストメッセージを送信します。

IP マルチキャストは初期メンバーを検出するために使用することができないため、初期メンバーを見つけるには別のメカニズムを使用する必要があります。

バグを報告する

30.2.1.3. TCPPing プロトコルの使用

一部のネットワークでは、TCP のみを使用できます。事前に設定された **`default-configs/default-jgroups-tcp.xml`** には **`MPING`** プロトコルが含まれ、検出に **`UDP`** マルチキャストが使用されます。**`UDP`** マルチキャストが利用できない場合は、**`MPING`** プロトコルを別のメカニズムで置き換える必要があります。推奨される別の方法は、**`TCPPING`** プロトコルを使用することです。**`TCPPING`** 設定には、ノード検出のために接続される IP アドレスの静的なリストが含まれます。

例30.6 JGroups サブシステムでの TCPPING の使用の設定

```
<TCP bind_port="7800" />
<TCPPING
initial_hosts="${jgroups.tcpping.initial_hosts:HostA[7800],HostB[7801]}"
         port_range="1" />
```

バグを報告する

30.2.2. 事前設定された JGroups ファイル

Red Hat JBoss Data Grid では、事前設定された複数の JGroups ファイルが **infinispan-embedded.jar** にパッケージ化され、デフォルトでクラスパス上にて使用可能です。これらのファイルの 1 つを使用するには、**jgroups.xml** を使用する代わりにこれらのいずれかのファイルの名前を指定します。

JBoss Data Grid に含まれる JGroups 設定ファイルは、プロジェクトの基礎として使用することを目的としています。通常 JGroups では、ネットワークのパフォーマンスを最適化するのに細かな調整が必要となります。

利用可能な設定は以下のとおりです。

- **default-configs/default-jgroups-udp.xml**
- **default-configs/default-jgroups-tcp.xml**
- **default-configs/default-jgroups-ec2.xml**

バグを報告する

30.2.2.1. default-jgroups-udp.xml

default-configs/default-jgroups-udp.xml ファイルは、Red Hat JBoss Data Grid の事前設定済み JGroups 設定です。**default-jgroups-udp.xml** 設定には、以下のことが該当します。

- UDP をトランスポートとして使用し、UDP マルチキャストをディスカバリーに使用します。
- 大型のクラスター (9 以上のノード) に適しています。
- インバリデーションまたはレプリケーションモードを使用する場合に適しています。

起動時に特定のシステムプロパティーを JVM に追加すると、一部の設定の挙動を変更することが可能です。変更可能な設定は以下の表のとおりです。

表30.1 default-jgroups-udp.xml システムプロパティー

システムプロパティー	説明	デフォルト	必要性
jgroups.udp.mcast_addr	マルチキャスト (通信とディスカバリーの両方) に使用する IP アドレス。IP マルチキャストに適した有効なクラス D IPアドレスでなければなりません。	228.6.7.8	いいえ
jgroups.udp.mcast_port	マルチキャストに使用するポート。	46655	いいえ
jgroups.udp.ip_ttl	IP マルチキャストパケットの TTL (有効期間) を指定します。この値は、パケットがドロップされる前に許可されるネットワークホップの数になります。	2	いいえ

バグを報告する

30.2.2.2. default-jgroups-tcp.xml

default-configs/default-jgroups-tcp.xml ファイルは、Red Hat JBoss Data Grid の事前設定済み JGroups 設定です。**default-jgroups-tcp.xml** 設定には、以下のことが該当します。

- TCP をトランスポートとして使用し、UDP マルチキャストをディスカバリーに使用します。
- 通常は、マルチキャスト UDP がオプションではない場合にのみ使用されます。
- 8 つ以上のノードから構成されるクラスターの場合、TCP は UDP ほどパフォーマンスがよくありません。4 つ以下のノードで構成されるクラスターの場合、UDP と TCP のパフォーマンスはほとんど同じレベルになります。

他の事前設定された JGroups ファイルと同様に、起動時に特定のシステムプロパティーを JVM に追加すると一部の設定の挙動を変更することが可能です。変更可能な設定は以下の表のとおりです。

表30.2 default-jgroups-tcp.xml システムプロパティー

システムプロパティー	説明	デフォルト	必要性
jgroups.tcp.address	TCP トランスポートに使用する IP アドレス	127.0.0.1	いいえ
jgroups.tcp.port	TCP ソケットに使用するポート。	7800	いいえ
jgroups.mping.mcast_addr	マルチキャスト (ディスカバリー) に使用する IP アドレス。IP マルチキャストに適した有効なクラス D IP アドレスでなければなりません。	228.6.7.8	いいえ
jgroups.mping.mcast_port	マルチキャストに使用するポート。	46655	いいえ
jgroups.udp.ip_ttl	IP マルチキャストパケットの TTL (有効期間) を指定します。この値は、パケットがドロップされる前に許可されるネットワークホップの数になります。	2	いいえ

バグを報告する

30.2.2.3. default-jgroups-ec2.xml

default-configs/default-jgroups-ec2.xml ファイルは、Red Hat JBoss Data Grid の事前設定済み JGroups 設定です。**default-jgroups-ec2.xml** 設定には、以下のことが該当します。

- TCP をトランスポートとして使用し、ディスカバリーに S3_PING を使用します。
- UDP マルチキャストが使用できない Amazon EC2 ノードに適しています。

他の事前設定された JGroups ファイルと同様に、起動時に特定のシステムプロパティーを JVM に追加すると一部の設定の挙動を変更することが可能です。変更可能な設定は以下の表のとおりです。

表30.3 default-jgroups-ec2.xml システムプロパティー

システムプロパティー	説明	デフォルト	必要性
jgroups.tcp.address	TCP トランスポートに使用する IP アドレス。	127.0.0.1	いいえ
jgroups.tcp.port	TCP ソケットに使用するポート。	7800	いいえ
jgroups.s3.access_key	S3 バケットのアクセスに使用される Amazon S3 アクセスキー。		はい
jgroups.s3.secret_access_key	S3 バケットのアクセスに使用される Amazon S3 の秘密キー。		はい
jgroups.s3.bucket	使用する Amazon S3 バケットの名前。一意の名前で、すでに存在している必要があります。		はい
jgroups.s3.pre_signed_delete_url	DELETE 操作に使用する事前署名付き URL。		はい
jgroups.s3.pre_signed_put_url	PUT 操作に使用する事前署名付き URL。		はい
jgroups.s3.prefix	設定された場合、S3_PING はそのプレフィックス値で始まる名前のバケットを検索します。		いいえ

バグを報告する

30.2.2.4. default-jgroups-google.xml

`default-configs/default-jgroups-google.xml` ファイルは Red Hat JBoss Data Grid の事前に定義された JGroups 設定です。**`default-jgroups-google.xml`** 設定には以下のことが該当します。

- TCP をトランスポートとして使用し、GOOGLE_PING をディスカバリーに使用します。
- UDP マルチキャストが使用できない Google Compute Engine ノードに適しています。

他の事前に設定された JGroups ファイルと同様に、起動時に特定のシステムプロパティーを JVM に追加すると一部の設定の動作を変更することが可能です。変更可能な設定は以下の表のとおりです。

表30.4 default-jgroups-google.xml System Properties

システムプロパティー	説明	デフォルト	必要性
jgroups.tcp.address	TCP トランスポートに使用する IP アドレス	127.0.0.1	いいえ

システムプロパティー	説明	デフォルト	必要性
jgroups.tcp.port	TCP ソケットに使用するポート。	7800	いいえ
jgroups.google.access_key	バケットのアクセスに使用される Google Compute Engine ユーザーのアクセスキー。		はい
jgroups.google.secret_access_key	バケットのアクセスに使用される Google Compute Engine ユーザーのシークレットアクセスキー。		はい
jgroups.google.bucket	使用する Google Compute Engine バケットの名前。一意の名前で、すでに存在している必要があります。		はい

バグを報告する

30.3. JGROUPS を使用したマルチキャストのテスト

システムがクラスター内でマルチキャストを正しく設定していることを確認する方法について学習します。

バグを報告する

30.3.1. 異なる Red Hat JBoss Data Grid バージョンを使用したテスト

以下の表は、このマルチキャストテストと互換性のある Red Hat JBoss Data Grid バージョンの詳細を示しています。

注記

${infinispan.version} は、JBoss Data Grid の特定のリリースに組み込まれた Infinispan のバージョンに対応します。これは、メジャーバージョン、マイナーバージョンおよびリビジョンを含む x.y.z 形式で表示されます。

表30.5 異なる JBoss Data Grid バージョンを使用したテスト

バージョン	テストケース	詳細

バージョン	テストケース	詳細
JBoss Data Grid 7.0.0	利用可能	テストクラスの場所はディストリビューションによって異なります。 • ライブラリーモードでは、**infinispan-embedded-${infinispan.version}.Final-redhat-#** JAR ファイルに含まれます。 • リモートクライアントサーバーモードでは、***${JDG_HOME}*/modules/system/layers/base/org/jgroups/main/** ディレクトリーの JGroups JAR ファイルに含まれます。
JBoss Data Grid 6.6.0	利用可能	テストクラスの場所はディストリビューションによって異なります。 • ライブラリーモードでは、**infinispan-embedded-${infinispan.version}.Final-redhat-#** JAR ファイルに含まれます。 • リモートクライアントサーバーモードでは、***${JDG_HOME}*/modules/system/layers/base/org/jgroups/main/** ディレクトリーの JGroups JAR ファイルに含まれます。

バージョン	テストケース	詳細
JBoss Data Grid 6.5.1	利用可能	テストクラスの場所はディストーションによって異なります。 • ライブラリーモードでは、**infinispan-embedded-${infinispan.version}.Final-redhat-#** JAR ファイルに含まれます。 • リモートクライアントサーバーモードでは、***${JDG_HOME}*/modules/system/layers/base/org/jgroups/main/** ディレクトリーの JGroups JAR ファイルに含まれます。
JBoss Data Grid 6.5.0	利用可能	テストクラスの場所はディストーションによって異なります。 • ライブラリーモードでは、**infinispan-embedded-${infinispan.version}.Final-redhat-#** JAR ファイルに含まれます。 • リモートクライアントサーバーモードでは、***${JDG_HOME}*/modules/system/layers/base/org/jgroups/main/** ディレクトリーの JGroups JAR ファイルに含まれます。

バージョン	テストケース	詳細
JBoss Data Grid 6.4.0	利用可能	テストクラスの場所はディストリビューションによって異なります。 • ライブラリーモードでは、**infinispan-embedded-${infinispan.version}.Final-redhat-#** JAR ファイルに含まれます。 • リモートクライアントサーバーモードでは、***${JDG_HOME}*/modules/system/layers/base/org/jgroups/main/** ディレクトリーの JGroups JAR ファイルに含まれます。
JBoss Data Grid 6.3.0	利用可能	テストクラスの場所はディストリビューションによって異なります。 • ライブラリーモードでは、lib ディレクトリーの JGroups JAR ファイルに含まれます。 • リモートクライアントサーバーモードでは、***${JDG_HOME}*/modules/system/layers/base/org/jgroups/main** の JGroups JAR ファイルに含まれます。
JBoss Data Grid 6.2.1	利用可能	テストクラスの場所はディストリビューションによって異なります。 • ライブラリーモードでは、lib ディレクトリーの JGroups JAR ファイルに含まれます。 • リモートクライアントサーバーモードでは、***${JDG_HOME}*/modules/system/layers/base/org/jgroups/main** の JGroups JAR ファイルに含まれます。

バージョン	テストケース	詳細
JBoss Data Grid 6.2.0	利用可能	テストクラスの場所はディストリビューションによって異なります。 • ライブラリーモードでは、lib ディレクトリーの JGroups JAR ファイルに含まれます。 • リモートクライアントサーバーモードでは、***${JDG_HOME}*/modules/system/layers/base/org/jgroups/main** の JGroups JAR ファイルに含まれます。
JBoss Data Grid 6.1.0	利用可能	テストクラスの場所はディストリビューションによって異なります。 • ライブラリーモードでは、**lib** ディレクトリーの JGroups JAR ファイルに含まれます。 • リモートクライアントサーバーモードでは、***${JDG_HOME}*/modules/org/jgroups/main/** ディレクトリーの JGroups JAR ファイルに含まれます。
JBoss Data Grid 6.0.1	使用不可	JBoss Data Grid のこのバージョンには、このテストで使用されるテストクラスが含まれない JBoss Enterprise Application 6.0 をベースにしています。
JBoss Data Grid 6.0.0	使用不可	JBoss Data Grid のこのバージョンには、このテストで使用されるテストクラスが含まれない JBoss Enterprise Application Server 6.0 をベースにしています。

バグを報告する

30.3.2. JGroups を使用したマルチキャストのテスト

以下の手順は、Red Hat JBoss Data Grid を使用している場合に JGroups を使用したマルチキャストのテストを実行するための手順を詳細に示しています。

前提条件

テストの手順を開始する前に以下の要件を満たしていることを確認してください。

1. ***bind_addr*** 値をインスタンスの適切な IP アドレスに設定します。
2. 精度を高めるために、クラスター通信の値と同じ ***mcast_addr*** と ***port*** の値を設定します。
3. 2 つのコマンドラインターミナルウィンドウを起動します。最初のターミナルで 2 つのノードの内の 1 つの JGroups JAR ファイルのロケーションと、2 番目のターミナルで 2 つ目のノードの同じロケーションにナビゲートします。

手順30.1 JGroups を使用したマルチキャストのテスト

1. **1 つ目のノードでマルチキャストサーバーを実行します。**
   最初のノードについて、コマンドラインターミナルで以下のコマンドを実行します (ライブラリーモードの場合は **jgroups.jar** を **infinispan-embedded.jar** に置き換えます)。

   ```
   java -cp jgroups.jar org.jgroups.tests.McastReceiverTest -mcast_addr
   230.1.2.3 -port 5555 -bind_addr $YOUR_BIND_ADDRESS
   ```

2. **2 つ目のノードでマルチキャストサーバーを実行します。**
   2 番目のノードについて、コマンドラインターミナルで以下のコマンドを実行します (ライブラリーモードの場合は **jgroups.jar** を **infinispan-embedded.jar** に置き換えます)。

   ```
   java -cp jgroups.jar org.jgroups.tests.McastSenderTest -mcast_addr
   230.1.2.3 -port 5555 -bind_addr $YOUR_BIND_ADDRESS
   ```

3. **情報パケットを送信します。**
   2 つ目のノード (パケットを送信するノード) のインスタンスに情報を入力し、Enter を押して情報を送信します。
4. **受信情報パケットを表示します。**
   1 番目のノードのインスタンスで受信された情報を表示します。直前の手順で入力した情報がここに表示されます。
5. **情報転送を確認します。**
   パケットがドロップされずに、送信情報がすべて受信されていることを確認するために、手順 3 と 4 を繰り返します。
6. **他のインスタンスのテストを繰り返します。**
   送信者と受信者のそれぞれを確認するために、手順 1 から 4 を繰り返します。テストを繰り返すことにより、誤って設定された他のインスタンスが特定されます。

結果

送信者ノードから送信されるすべての情報パケットは、受信者ノードに表示される必要があります。送信された情報が予想どおりに表示されない場合、マルチキャストがオペレーティングシステムまたはネットワークで誤って設定されていることになります。

バグを報告する

第31章 RED HAT DATA GRID の AMAZON WEB サービスとの使用

31.1. S3_PING JGROUPS ディスカバリープロトコル

Amazon の Elastic Compute Cloud (EC2) ではマルチキャストがサポートされず、**MPING** が許可されないため、**S3_PING** は EC2 と一緒に使用するのに最適なディスカバリープロトコルです。

それぞれの EC2 インスタンスは小さいファイルをバケットとして知られる S3 データコンテナーに追加します。その後、各インスタンスはバケット内のファイルを読み込み、クラスターの他のメンバーを検出します。

バグを報告する

31.2. S3_PING 設定オプション

Red Hat JBoss Data Grid は Amazon Web Services と 2 通りの方法で連動します。

- ライブラリーモードでは、JGroups の **default-configs/default-jgroups-ec2.xml** ファイル (詳細については、「default-jgroups-ec2.xml」を参照) または **S3_PING** プロトコルを使用します。
- リモートクライアントサーバーモードでは、JGroups の **S3_PING** プロトコルを使用します。

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では、Amazon AWS で機能するようにクラスター化するために **S3_PING** プロトコルを設定する方法として 3 つの方法があります。

- プライベート S3 バケットを使用します。これらのバケットは Amazon AWS のクレデンシャルを使用します。
- 事前署名付き URL を使用します。これらの事前に割り当てられる URL は、プライベートの書き込みアクセスとパブリックの読み取りアクセスを持つバケットに割り当てられます。
- パブリック S3 バケットを使用します。これらのバケットにはクレデンシャルがありません。

バグを報告する

31.2.1. プライベート S3 バケットの使用

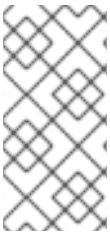
この設定には、適切な AWS クレデンシャルによってのみアクセスできるプライベートバケットへのアクセスが必要です。適切な権限が利用できることを確認するには、ユーザーにバケットの以下の権限があることを確認してください。

- List
- Upload/Delete
- View Permissions
- Edit Permissions

S3_PING 設定には以下のプロパティーが含まれることを確認してください。

- バケットのある ***location***。

- AWS ユーザーには ***access_key*** と ***secret_access_key*** プロパティー。



注記

この設定の使用時に **403** エラーが表示される場合、プロパティーに正しい値があることを検証してください。問題が存続する場合、EC2 ノードのシステム時間が正しいことを確認してください。Amazon S3 は、セキュリティー上の理由により、サーバーの時間よりも **15** 分を超える遅れがあるタイムスタンプを持つ要求を拒否します。

例31.1 プライベートバケットの使用による Red Hat JBoss Data Grid Server の起動

サーバーディレクトリーの上位レベルから次のコマンドを実行して、プライベート S3 バケットを使用して Red Hat JBoss Data Grid サーバーを起動します。

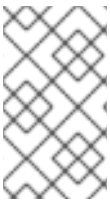
```
bin/standalone.sh
  -c cloud.xml
  -Djboss.node.name={node_name}
  -Djboss.socket.binding.port-offset={port_offset}
  -Djboss.default.jgroups.stack=s3-private
  -Djgroups.s3.bucket={s3_bucket_name}
  -Djgroups.s3.access_key={access_key}
  -Djgroups.s3.secret_access_key={secret_access_key}
```

1. *{node_name}* をサーバーの必要なノード名に置き換えます。
2. *{port_offset}* をポートオフセットに置き換えます。デフォルトのポートを使用するには、これを **0** に指定します。
3. *{s3_bucket_name}* を適切なバケット名に置き換えます。
4. *{access_key}* をユーザーのアクセスキーに置き換えます。
5. *{secret_access_key}* をユーザーの秘密アクセスキーに置き換えます。

バグを報告する

31.2.2. 事前署名付き URL の使用

この設定では、**List** 権限を **Everyone** に設定し、パブリックの読み取りアクセスを許可することにより、S3 の一般に読み取り可能なバケットを作成します。クラスター内の各ノードは、**S3_PING** プロトコルで要求される場合に、put および delete 操作用に事前署名付き URL を生成します。この URL は固有のファイルを参照し、バケット内のフォルダーパスを組み込むことができます。



注記

パスが長くなると、**S3_PING** にエラーが発生します。たとえば、**my_bucket/DemoCluster/node1** のようなパスは機能しますが、**my_bucket/Demo/Cluster/node1** のようにパスが長くなると機能しません。

バグを報告する

31.2.2.1. 事前署名付き URL の生成

JGroups の **S3_PING** クラスには、事前署名付き URL を生成するためのユーティリティーメソッドが含まれます。このメソッドの最後の引数は、Unix epoch (1970 年 1 月 1 日) からの秒数で表される URL の有効期間です。

事前署名付き URL を生成する構文は次のようになります。

```
String Url = S3_PING.generatePreSignedUrl("{access_key}",
"{secret_access_key}", "{operation}", "{bucket_name}", "{path}",
{seconds});
```

1. *{operation}* を **PUT** または **DELETE** のいずれかに置き換えます。
2. *{access_key}* をユーザーのアクセスキーに置き換えます。
3. *{secret_access_key}* をユーザーの秘密アクセスキーに置き換えます。
4. *{bucket_name}* をバケットの名前に置き換えます。
5. *{path}* をバケット内のファイルへの必要なパスに置き換えます。
6. *{seconds}* を、Unix epoch (1970 年 1 月 1 日) からのパスの有効期間を表す秒数に置き換えます。

例31.2 事前署名付き URL の生成

```
String putUrl = S3_PING.generatePreSignedUrl("access_key",
"secret_access_key", "put", "my_bucket", "DemoCluster/jgroups.list",
1234567890);
```

S3_PING 設定に、**S3_PING.generatePreSignedUrl()** の呼び出しで生成された ***pre_signed_put_url*** および ***pre_signed_delete_url*** プロパティーが含まれていることを確認してください。この設定の場合、AWS クレデンシャルがクラスター内の各ノードに保存されないため、プライベート S3 バケットを使用する設定よりも安全度が高くなります。

注記

事前署名付き URL が XML ファイルに入力される場合、URL 内の **&** 文字をその XML エンティティー **(&)** に置き換える必要があります。

バグを報告する

31.2.2.2. コマンドラインを使用した事前署名付き URL の設定

コマンドラインを使用して事前署名付き URL を設定するには、以下のガイドラインを使用します。

- URL を二重引用符 **("")** で囲みます。
- URL では、アンパーサンド **(&)** 文字の各出現箇所はバックスラッシュ **(\)** でエスケープする必要があります。

例31.3 事前署名付き URL による JBoss Data Grid Server の起動

```
bin/standalone.sh
  -c cloud.xml
  -Djboss.node.name={node_name}
  -Djboss.socket.binding.port-offset={port_offset}
  -Djboss.default.jgroups.stack=s3-presigned
  -
Djgroups.s3.pre_signed_delete_url="http://{s3_bucket_name}.s3.amazonaws.
com/jgroups.list?AWSAccessKeyId={access_key}\&amp;Expires=
{expiration_time}\&amp;Signature={signature}"
  -
Djgroups.s3.pre_signed_put_url="http://{s3_bucket_name}.s3.amazonaws.com
/jgroups.list?AWSAccessKeyId={access_key}\&amp;Expires=
{expiration_time}\&amp;Signature={signature}"
```

1. *{node_name}* をサーバーの必要なノード名に置き換えます。
2. *{port_offset}* をポートオフセットに置き換えます。デフォルトのポートを使用するには、これを **0** に指定します。
3. *{s3_bucket_name}* を適切なバケット名に置き換えます。
4. *{access_key}* をユーザーのアクセスキーに置き換えます。
5. *{expiration_time}* を、**S3_PING.generatePreSignedUrl()** メソッドに渡された URL の値に置き換えます。
6. *{signature}* を、**S3_PING.generatePreSignedUrl()** メソッドで生成される値に置き換えます。

バグを報告する

31.2.3. パブリック S3 バケットの使用

この設定には、パブリックの読み書き権限のある S3 バケットが関係します。つまり、**Everyone** には、バケットの **List**、 **Upload/Delete** の権限、**View Permissions**、および **Edit Permissions** が設定されます。

location プロパティーは、この設定のバケット名で指定する必要があります。この設定メソッドは、バケットの名前を知っているいずれのユーザーもバケットにデータをアップロードしたり、保存したりでき、バケット作成者のアカウントはこのデータについてチャージされるため、安全度は最も低くなります。

Red Hat JBoss Data Grid サーバーを起動するには、以下のコマンドを使用します。

```
bin/standalone.sh
  -c cloud.xml
  -Djboss.node.name={node_name}
  -Djboss.socket.binding.port-offset={port_offset}
  -Djboss.default.jgroups.stack=s3-public
  -Djgroups.s3.bucket={s3_bucket_name}
```

1. *{node_name}* をサーバーの必要なノード名に置き換えます。

2. *{port_offset}* をポートオフセットに置き換えます。デフォルトのポートを使用するには、これを **0** に指定します。

3. *{s3_bucket_name}* を適切なバケット名に置き換えます。

バグを報告する

31.3. ELASTIC IP アドレスの使用

クラスター内の各ノードが S3_PING プロトコルを使用して他のノードを検出できる一方で、すべてのネットワークトラフィックは内部のプライベートネットワーク上にあります。単一ノードに Elastic IP または静的 IP を設定し、起動時に管理コンソールなどでクラスターを設定するために一貫性のあるアドレスを設定できるようにします。Elastic IP が設定されない場合、各インスタンスには、起動時には常にパブリックネットワークのランダム化された IPアドレスが含まれます。

Elastic IP アドレスの設定方法についての詳細は、Amazon の Getting Started Guide を参照してください。

バグを報告する

第32章 GOOGLE COMPUTE ENGINE での RED HAT JBOSS DATA GRID の使用

32.1. GOOGLE_PING プロトコル

GOOGLE_PING は、クラスターの作成時に JGroups によって使用されるディスカバリープロトコルです。Google Compute Engine (GCE) で使用し、個別のクラスターメンバーについての情報を格納するために Google Cloud Storage を使用できることが望ましいでしょう。

バグを報告する

32.2. GOOGLE_PING 設定

Red Hat JBoss Data Grid は以下の方法で Google Compute Engine と連動します。

- ライブラリーモードでは、JGroups の設定ファイル **default-configs/default-jgroups-google.xml** を使用するか、または既存の設定ファイルの**GOOGLE_PING** プロトコルを使用します。
- リモートクライアントサーバーモードでは、サーバーを起動して JGroups Google スタックを使用する際にコマンドラインでプロパティーを定義します (「Google Compute Engine でのサーバーの起動」 の例を参照してください)。

GOOGLE_PING プロトコルがライブラリーモードおよびリモートクライアントサーバーモードの Google Compute Engine で機能するように設定するには、以下を実行します。

- JGroups バケットを使用します。これらのバケットは Google Compute Engine クレデンシャルを使用します。
- アクセスキーを使用します。
- シークレットアクセスキーを使用します。

注記

マルチキャストが許可されないため、**TCP** プロトコルのみが Google Compute Engine でサポートされます。

バグを報告する

32.2.1. Google Compute Engine でのサーバーの起動

この設定には、適切な Google Compute Engine クレデンシャルでのみアクセスできるバケットへのアクセスが必要です。

GOOGLE_PING 設定には以下のプロパティーが含まれることを確認してください。

- Google Compute Engine ユーザーの ***access_key*** と ***secret_access_key*** プロパティー。

> **例32.1 バケットを使用した Red Hat JBoss Data Grid サーバーの起動**
>
> サーバーディレクトリーの上位レベルから次のコマンドを実行して、バケットを使用して Red Hat JBoss Data Grid サーバーを起動します。

```
bin/standalone.sh
  -c cloud.xml
  -Djboss.node.name={node_name}
  -Djboss.socket.binding.port-offset={port_offset}
  -Djboss.default.jgroups.stack=google
  -Djgroups.google.bucket={google_bucket_name}
  -Djgroups.google.access_key={access_key}
  -Djgroups.google.secret_access_key={secret_access_key}
```

1. *{node_name}* をサーバーの必要なノード名に置き換えます。
2. *{port_offset}* をポートオフセットに置き換えます。デフォルトのポートを使用するには、これを **0** に指定します。
3. *{google_bucket_name}* を適切なバケット名に置き換えます。
4. *{access_key}* をユーザーのアクセスキーに置き換えます。
5. *{secret_access_key}* をユーザーのシークレットアクセスキーに置き換えます。

バグを報告する

32.3. 静的 IP アドレスの使用

クラスター内の各ノードが GOOGLE_PING プロトコルを使用して他のノードを検出できる一方で、すべてのネットワークトラフィックは内部のプライベートネットワーク上にあります。単一ノードに外部の静的 IP アドレスを設定し、起動時に管理コンソールなどでクラスターを設定するために一貫性のあるアドレスを設定できるようにします。静的アドレスが設定されない場合、各インスタンスには、起動時には常にパブリックネットワークのランダム化された IPアドレスが含まれます。

外部の静的 IP アドレスの設定方法についての詳細は、Google の「Configuring an Instance's IP Address」ドキュメントを参照してください。

バグを報告する

第33章 SPRING FRAMEWORK との統合

JBoss Data Grid により、ユーザーは Spring キャッシュプロバイダーを定義でき、アプリケーションにキャッシュサポートを簡単に追加できる方法を提供し、ユーザーが JBoss Data Grid でキャッシュを実行する方法としての Spring のプログラミングモデルに精通できます。

以下の手順および例は、管理者が Spring サポート用に JBoss Data Grid ノードを設定するために使用できる方法を示しています。Spring のアノテーションをアプリケーションに組み込む方法などの追加情報は、JBoss Data Grid の 『Developer Guide』 を参照してください。

バグを報告する

33.1. SPRING キャッシュサポートの宣言的な有効化 (ライブラリーモード)

Spring のキャッシュサポートは以下の手順を実行して xml ファイルで有効にできます。

1. **<cache:annotation-driven/>** を xml ファイルに追加します。この行は、標準的な spring の注釈をアプリケーションで使用できるようにします。

2. **<infinispan:embedded-cache-manager ... />** を使用してキャッシュマネージャーを定義します。

以下の例はこれらの変更を示しています。

例33.1 宣言的な設定例

```
<?xml version="1.0" encoding="UTF-8"?>
<beans xmlns="http://www.springframework.org/schema/beans"
       xmlns:xsi="http://www.w3.org/2001/XMLSchema-instance"
       xmlns:infinispan="http://www.infinispan.org/schemas/spring"
       xmlns:cache="http://www.springframework.org/schema/cache"
       xsi:schemaLocation="http://www.springframework.org/schema/beans
http://www.springframework.org/schema/beans/spring-beans.xsd
       http://www.springframework.org/schema/cache
http://www.springframework.org/schema/cache/spring-cache.xsd
       http://www.infinispan.org/schemas/spring
http://www.infinispan.org/schemas/infinispan-spring.xsd">
[...]
<cache:annotation-driven/>

<infinispan:embedded-cache-manager
            configuration="classpath:/path/to/cache-config.xml"/>
[...]
```

バグを報告する

33.2. SPRING キャッシュサポートの宣言的な有効化 (リモートクライアントサーバーモード)

Spring のキャッシュサポートは以下の手順を実行して宣言的に有効にできます。

1. **<cache:annotation-driven/>** を xml ファイルに追加します。この行は、標準的な spring の注釈をアプリケーションで使用できるようにします。

2. **<infinispan:remote-cache-manager ... />** を使用して HotRod クライアントのプロパティーを定義します。

以下の例はこれらの変更を示しています。

例33.2 宣言的な設定例

```
<?xml version="1.0" encoding="UTF-8"?>
<beans xmlns="http://www.springframework.org/schema/beans"
       xmlns:xsi="http://www.w3.org/2001/XMLSchema-instance"
       xmlns:infinispan="http://www.infinispan.org/schemas/spring"
       xmlns:cache="http://www.springframework.org/schema/cache"
       xsi:schemaLocation="http://www.springframework.org/schema/beans
http://www.springframework.org/schema/beans/spring-beans.xsd
       http://www.springframework.org/schema/cache
http://www.springframework.org/schema/cache/spring-cache.xsd
       http://www.infinispan.org/schemas/spring
http://www.infinispan.org/schemas/infinispan-spring.xsd">
[...]
<cache:annotation-driven/>

<infinispan:remote-cache-manager
            configuration="classpath:/path/to/hotrod-
client.properties"/>
[...]
```

バグを報告する

第34章 サーバーヒンティングを用いた高可用性

Red Hat JBoss Data Grid では、サーバーヒンティングによってデータのバックアップコピーが元データと同じ物理サーバー、ラック、またはデータセンター上に保存されないようにします。サーバーヒンティングは、完全レプリケーションによってすべてのサーバー、ラック、およびデータセンター上で完全なレプリカが作成されるため、完全レプリケーションには適用されません。

複数のノードにまたがるデータ分散が一貫したハッシュメカニズムによって制御されます。
JBoss Data Grid は、一貫したハッシュアルゴリズムを指定するためのプラグ可能なポリシーを提供します。このポリシーの設定についてさらに詳しくは、JBoss Data Grid 『Developer Guide』の**ConsistentHashFactories** のセクションを参照してください。

machineId、***rackId***、または ***siteId*** を transport 設定に設定することにより、**TopologyAwareConsistentHashFactory** の使用がトリガーされます。これは、サーバーヒンティングが有効にされた状態の **DefaultConsistentHashFactory** に相当します。

サーバーヒンティングは、JBoss Data Grid 実装の高可用性を確実に実現する場合に特に重要になります。

バグを報告する

34.1. JGROUPS によるサーバーヒンティングの設定

Red Hat JBoss Data Grid でクラスター化環境を設定する場合、JGroups の設定時にサーバーヒンティングが設定されます。

JBoss Data Grid には、クラスターモード向けに事前設定された複数の JGroups ファイルが含まれています。これらのファイルは、JBoss Data Grid でサーバーヒンティングを設定する場合の土台として使用することができます。

関連トピック:

- 「事前設定された JGroups ファイル」

バグを報告する

34.2. サーバーヒンティングの設定 (リモートクライアントサーバーモード)

Red Hat JBoss Data Grid のリモートクライアントサーバーモードでは、サーバーヒンティングは、次のようにデフォルトスタックについて **transport** 要素の JGroups サブシステムで設定されます。

手順34.1 リモートクライアントサーバーモードでのサーバーヒンティングの設定

```
<subsystem xmlns="urn:jboss:domain:jgroups:3.0"
    default-stack="${jboss.default.jgroups.stack:udp}">
 <stack name="udp">
  <transport type="UDP"
      socket-binding="jgroups-udp"
      site="${jboss.jgroups.transport.site:s1}"
      rack="${jboss.jgroups.transport.rack:r1}"
      machine="${jboss.jgroups.transport.machine:m1}">
      <!-- Additional configuration elements here -->
```

```
   </transport>
  </stack>
</subsystem>
```

1. JGroups サブシステム設定を見つけます。
2. **transport** 要素でサーバーヒンティングを有効にします。
   a. ***site*** パラメーターを使用してサイト ID を設定します。
   b. ***rack*** パラメーターを使用してラック ID を設定します。
   c. ***machine*** パラメーターを使用してマシン ID を設定します。

バグを報告する

34.3. サーバーヒンティングの設定 (ライブラリーモード)

Red Hat JBoss Data Grid のライブラリーモードでは、トランスポートレベルでサーバーヒンティングが設定されます。以下はサーバーヒンティングの設定例になります。

手順34.2 ライブラリーモードでのサーバーヒンティングの設定

次の設定属性を使用して、JBoss Data Grid にサーバーヒンティングを設定します。

```
<transport cluster = "MyCluster"
           machine = "LinuxServer01"
           rack = "Rack01"
           site = "US-WestCoast" />
```

1. ***cluster*** 属性はクラスターに割り当てられた名前を指定します。
2. ***machine*** 属性は、元データを格納する JVM インスタンスを指定します。これは、複数の JVM があるノードや複数の仮想ホストを持つ物理ホストに対してとくに有用な属性です。
3. ***rack*** 属性は、元データが含まれるラックを指定します。これにより、別のラックがバックアップに使用されるようにします。
4. ***siteId*** 属性は、相互にレプリケーションを行っている異なるデータセンターのノードを区別します。

リストされているパラメーターは、JBoss Data Grid 設定ではオプションです。

machine、***rack***、または ***site*** が設定に含まれている場合、**TopologyAwareConsistentHashFactory** が自動的に選択され、サーバーヒンティングを有効にします。ただし、サーバーヒンティングが設定されていない場合、JBoss Data Grid の分散アルゴリズムにより元データと同じ物理マシン/ラック/データセンターにレプリケーションを保存することができます。

バグを報告する

第35章 データセンター間のレプリケーションのセットアップ

Red Hat JBoss Data Grid では、データセンター間レプリケーションにより、管理者は複数のクラスターでデータバックアップを作成することができます。3 つのクラスターを同じ物理ロケーションまたは異なるロケーションに置くことができます。JBoss Data Grid のサイト間レプリケーションの実装は JGroups の **RELAY2** プロトコルをベースとします。

データセンター間レプリケーションにより、複数のクラスター間のデータ冗長性が保証されます。理想的には、これらのクラスターをそれぞれ他のロケーションとは異なる物理サイトに置く必要があります。

バグを報告する

35.1. データセンター間レプリケーションの操作

Red Hat JBoss Data Grid のデータセンター間レプリケーションの操作は、以下のような例を使用して説明できます。

例35.1 データセンター間レプリケーションの例

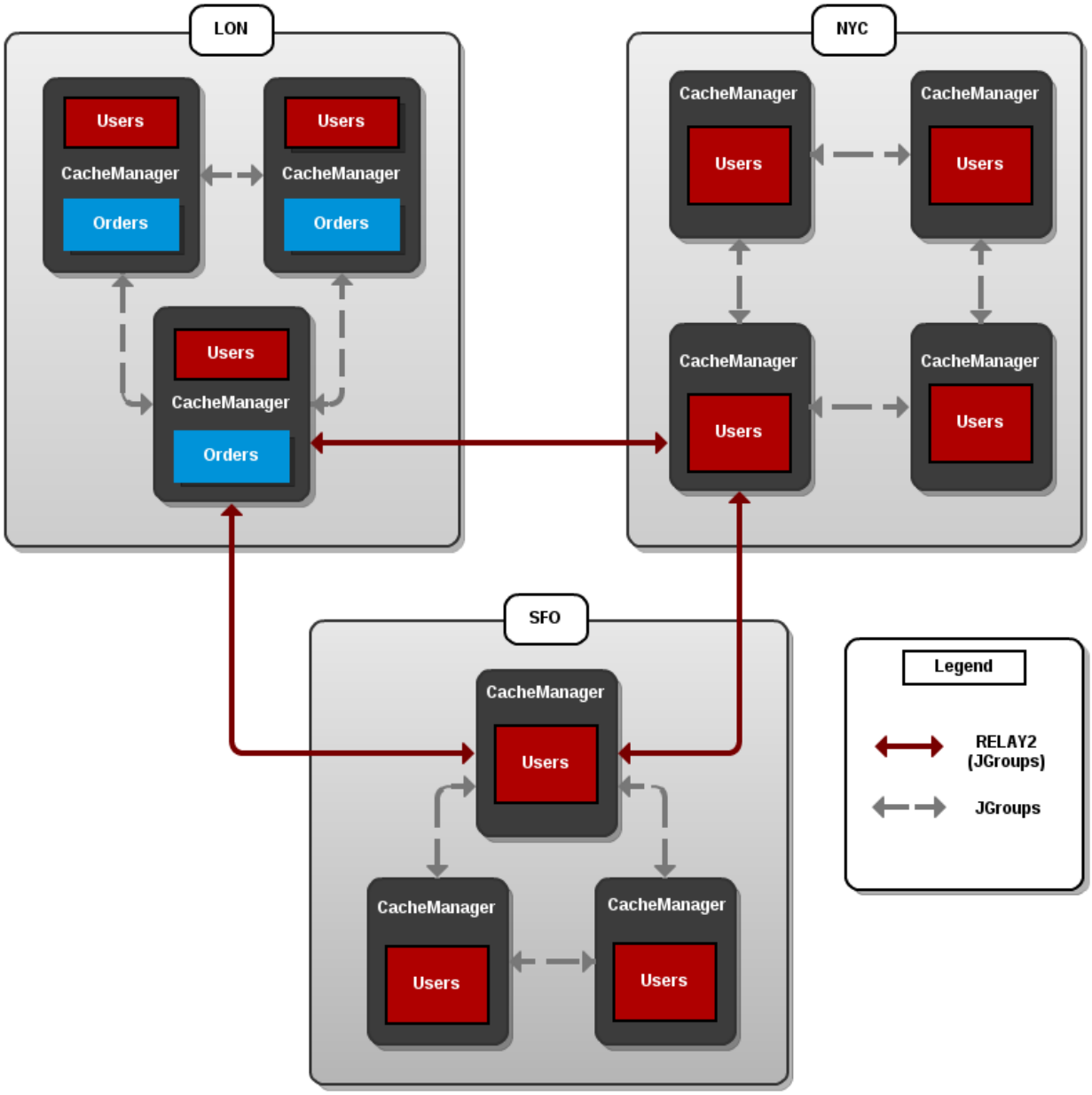


図35.1 データセンター間レプリケーションの例

この例では、**LON**、**NYC** および **SFO** の 3 つのサイトが設定されます。それぞれのサイトは、3 つから 4 つの物理ノードから構成される実行中の JBoss Data Grid クラスターをホストします。

Users キャッシュは、**LON**、**NYC** および **SFO** の 3 つのすべてのサイトでアクティブです。これらのサイトのいずれかの **Users** キャッシュへの変更は、キャッシュが設定で他の 2 つのサイトをバックアップとして定義している限り、それらの他の 2 つのサイトにレプリケートされます。ただし、**Orders** キャッシュは、他のサイトにレプリケートされないため **LON** サイトでのみ利用できます。

Users キャッシュは各サイトで異なるレプリケーションメカニズムを使用できます。たとえば、データのバックアップを **SFO** に同期的に、**NYC** と **LON** に非同期的に行います。

さらに **Users** キャッシュにはあるサイトから別のサイトへとそれぞれ異なる設定を持たせることもできます。たとえば、これを **LON** サイトで ***owners*** を **2** に設定した分散キャッシュとして、**NYC** サイトではレプリケートされたキャッシュとして、さらに **SFO** サイトでは ***owners*** を **1** に設定した分散キャッシュとして設定することができます。

JGroups はサイト間通信と共に各サイト内の通信のために使用されます。たとえば、**RELAY2** という JGroups プロトコルは、サイト間の通信を容易にします。さらに詳しくは、「RELAY2 について」を参照してください。

バグを報告する

35.2. データセンター間レプリケーションの設定

35.2.1. データセンター間レプリケーションの設定 (リモートクライアントサーバーモード)

Red Hat JBoss Data Grid のリモートクライアントサーバーモードでは、データセンター間のレプリケーションは以下のようにセットアップされます。

手順35.1 データセンター間のレプリケーションのセットアップ

1. **RELAY のセットアップ**
   RELAY をセットアップするには、以下の設定を **standalone.xml** ファイルに追加します。

```
<subsystem xmlns="urn:infinispan:server:jgroups:8.0">
    <channels default="cluster">
        <channel name="cluster"/>
        <channel name="xsite" stack="tcp"/>
    </channels>
    <stacks default="udp">
        <stack name="udp">
            <transport type="UDP" socket-binding="jgroups-udp"/>
            <...other protocols...>
            <relay site="LON">
                <remote-site name="NYC" channel="xsite"/>
                <remote-site name="SFO" channel="xsite"/>
            </relay>
        </stack>
    </stacks>
</subsystem>{
```

   RELAY プロトコルは、リモートサイトと通信するために追加のスタック (既存の**UDP** スタックと並行して実行される) を作成します。**TCP** ベースのスタックがローカルクラスターに使用される場合、2 つの **TCP** ベースのスタック設定が必要になります。1 つはローカル通信用で、もう 1 つはリモートサイトへの接続用になります。さらに詳しい説明は、「データセンター間レプリケーションの操作」を参照してください。

2. **サイトのセットアップ**
   クラスター内のそれぞれの分散キャッシュに対してサイトをセットアップするには、**standalone.xml** ファイルで以下の設定を使用します。

```
<distributed-cache name="namedCache">
     <!-- Additional configuration elements here -->
     <backups>
        <backup site="{FIRSTSITENAME}" strategy="{SYNC/ASYNC}" />
        <backup site="{SECONDSITENAME}" strategy="{SYNC/ASYNC}" />
     </backups>
</distributed-cache>
```

3. **ローカルサイトトランスポートの設定**
   トランスポートを設定するには、**transport** 要素にローカルサイトの名前を追加します。

```
<transport executor="infinispan-transport"
           lock-timeout="60000"
           cluster="LON"
           stack="udp"/>
```

データセンター間の設定例は **$JDG_SERVER/docs/examples/configs/clustered-xsite.xml** で確認できます。

バグを報告する

35.2.2. データセンター間レプリケーションの設定 (ライブラリーモード)

35.2.2.1. データセンター間レプリケーションを宣言的に設定する

データセンター間のレプリケーションを設定する際、**relay.RELAY2** プロトコルは、リモートサイトと通信するために追加のスタック (既存の **TCP** スタックと並行して実行される) を作成します。**TCP** ベースのスタックがローカルクラスターに使用される場合、2 つの **TCP** ベースのスタック設定が必要になります。1 つはローカル通信用で、もう 1 つはリモートサイトへの接続用です。

Red Hat JBoss Data Grid のライブラリーモードでは、データセンター間のレプリケーションは以下のようにセットアップされます。

手順35.2 データセンター間のレプリケーションのセットアップ

1. **ローカルサイトを設定します。**

```
<infinispan
      xmlns:xsi="http://www.w3.org/2001/XMLSchema-instance"
      xsi:schemaLocation="urn:infinispan:config:8.0
http://www.infinispan.org/schemas/infinispan-config-8.0.xsd"
      xmlns="urn:infinispan:config:8.0">

  <jgroups>
    <stack-file name="udp" path="jgroups-with-relay.xml"/>
  </jgroups>

  <cache-container default-cache="default">
    <transport cluster="infinispan-cluster" lock-timeout="50000"
               stack="udp" node-name="node1"
               machine="machine1" rack="rack1" site="LON"/>
      <local-cache name="default">
        <backups>
          <backup site="NYC" strategy="SYNC" failure-policy="IGNORE"
timeout="12003"/>
          <backup site="SFO" strategy="ASYNC"/>
        </backups>
      </local-cache>

   <!-- Additional configuration information here -->
</infinispan>
```

a. **site** 属性を **transport** 要素に追加してローカルサイト (この例では、ローカルサイトの名前は **LON**) を定義します。

b. サイト間のレプリケーションには、デフォルト以外の JGroups 設定が必要です。**jgroups** 要素を定義し、カスタム **stack-file** を定義して参照するファイルの名前と場所をこのカスタム設定に渡します。この例では、JGroups 設定ファイルの名前は **jgroups-with-relay.xml** です。

c. **NYC** および **SFO** サイトにバックアップするには、**LON** サイトでキャッシュを設定します。

d. バックアップキャッシュを設定します。

i. **LON** からバックアップデータを受信するには、**NYC** サイトでキャッシュを設定します。

```
        <local-cache name="backupNYC">
          <backups/>
          <backup-for remote-cache="default" remote-site="LON"/>
        </local-cache>
```

ii. **LON** からバックアップデータを受信するには、**SFO** サイトでキャッシュを設定します。

```
        <local-cache name="backupSFO">
          <backups/>
          <backup-for remote-cache="default" remote-site="LON"/>
        </local-cache>
```

2. **設定ファイルの内容を追加します。**
デフォルトで、Red Hat JBoss Data Grid には、**infinispan-embedded-*{VERSION}*.jar** パッケージ内の **default-configs/default-jgroups-tcp.xml** および **default-configs/default-jgroups-udp.xml** などの JGroups 設定ファイルが含まれます。

JGroups 設定を新規ファイル (この例では、ファイル名は **jgroups-with-relay.xml**) にコピーし、指定される設定情報をこのファイルに追加します。**relay.RELAY2** プロトコル設定は、設定スタックの最新のプロトコルである必要があります。

```
<config>
    ...
    <relay.RELAY2 site="LON"
              config="relay.xml"
              relay_multicasts="false" />
</config>
```

3. **relay.xml ファイルを設定します。**
relay.xml ファイルで **relay.RELAY2** 設定をセットアップします。このファイルは、グローバルクラスター設定について説明するものです。

```
<RelayConfiguration>
    <sites>
        <site name="LON"
              id="0">
            <bridges>
```

```
                <bridge config="jgroups-global.xml"
                        name="global"/>
                </bridges>
        </site>
        <site name="NYC"
              id="1">
            <bridges>
                <bridge config="jgroups-global.xml"
                        name="global"/>
                </bridges>
        </site>
        <site name="SFO"
              id="2">
            <bridges>
                <bridge config="jgroups-global.xml"
                        name="global"/>
            </bridges>
        </site>
    </sites>
</RelayConfiguration>
```

4. **グローバルクラスターを設定します。**
   relay.xml で参照されるファイル **jgroups-global.xml** には、グローバルクラスター、つまりサイト間の通信で使用される別の JGroups 設定が含まれます。

   グローバルクラスター設定は、通常は **TCP** ベースであり、**TCPPING** プロトコル (**PING** または **MPING** ではない) を使用してメンバーを検出します。**default-configs/default-jgroups-tcp.xml** の内容を **jgroups-global.xml** にコピーし、**TCPPING** を設定するために以下の設定を追加します。

```
<config>
    <TCP bind_port="7800" ... />
    <TCPPING
initial_hosts="lon.hostname[7800],nyc.hostname[7800],sfo.hostname[78
00]"
             ergonomics="false" />
          <!-- Rest of the protocols -->
</config>
```

   TCPPING.initial_hosts のホスト名 (または IP アドレス) をサイトマスターに使用されるものに置き換えます。ポート (この場合は **7800**) は ***TCP.bind_port*** に一致している必要があります。

   TCPPING プロトコルの詳細は、「TCPPing プロトコルの使用」 を参照してください。

バグを報告する

35.3. サイトをオフラインにする

Red Hat JBoss Data Grid のデータセンター間レプリケーション設定では、一定の期間内にあるサイトへのバックアップが複数回失敗すると、そのサイトをオフラインとして自動的にマークできます。この機能により、サイトをオフラインとマークする管理者の手動の介入は不要になります。

指定される条件を満たす場合にサイトを自動的に停止させたり、管理者がサイトを手動で停止させたりできるように JBoss Data Grid を設定することができます。

- サイトの自動的なオフライン設定:
  - リモートクライアントサーバーモードで宣言的に実行。
  - ライブラリーモードで宣言的に実行。
  - プログラムを用いたメソッドの使用。
- サイトの手動によるオフライン設定:
  - JBoss Operations Network (JON) の使用。
  - JBoss Data Grid コマンドラインインターフェース (CLI) の使用。

バグを報告する

35.3.1. サイトをオフラインにする

Red Hat JBoss Data Grid のいずれかのモードでサイトをオフラインにするには、**take-offline** 要素を **backup** 要素に追加します。これにより、サイトが自動的にオフラインにするように設定されます。

例35.2 サイトをオフラインに設定する (リモートクライアントサーバーモード)

```
<backup>
 <take-offline after-failures="${NUMBER}"
        min-wait="${PERIOD}" />
</backup>
```

take-offline 要素は、いつサイトをオフラインにするかを設定するために以下のパラメーターを使用します。

- ***after-failures*** パラメーターは、サイトがオフラインになる前にサイトへの接続の試行を失敗できる回数を指定します。
- ***min-wait*** パラメーターは、応答しないサイトをオフラインとしてマークするために待機する時間 (秒数単位) を指定します。このサイトは、***min-wait*** 期間が最初の試行後に経過した時や ***after-failures*** パラメーターで指定される試行の失敗回数に達した時にオフラインになります。

バグを報告する

35.3.2. JBoss Operations Network (JON) 経由でサイトをオフラインにする

JBoss Operations Network の操作を使用して、Red Hat JBoss Data Grid でサイトをオフラインにすることができます。メトリックスの一覧については、「JBoss Operations Network プラグイン操作」を参照してください。

バグを報告する

35.3.3. CLI でサイトをオフラインにする

サイトが応答しない場合、Red Hat JBoss Data Grid のコマンドラインインターフェース (CLI) で ***site*** コマンドを使用して、データセンター間のレプリケーション設定からサイトを手動でシャットダウンします。

site コマンドを使用して、以下のようにサイトの状態を確認することができます。

```
[jmx://localhost:12000/MyCacheManager/namedCache]> site --status
${SITENAME}
```

このコマンドの結果は、名前付きサイトの現在の状態に応じて、**online** または **offline** のいずれかになります。

このコマンドは、以下のように名前を使用してサイトをオンラインまたはオフラインにするように使用できます。

```
[jmx://localhost:12000/MyCacheManager/namedCache]> site --offline
${SITENAME}
```

```
[jmx://localhost:12000/MyCacheManager/namedCache]> site --online
${SITENAME}
```

コマンドが正常に実行されると、コマンドの後に出力 **ok** が表示されます。また、JMX を使用してサイトをオンラインに復帰させることもできます (詳細については「サイトをオンラインに戻す」を参照)。

JBoss Data Grid CLI とそのコマンドについてさらに詳しくは、『Developer Guide』 の JBoss Data Grid コマンドラインインターフェース (CLI) についての章を参照してください。

バグを報告する

35.3.4. サイトをオンラインに戻す

サイトがオフラインになった後に、サイトは、JMX コンソールを使用して **XSiteAdmin** MBean 上で **bringSiteOnline(*siteName*)** 操作を呼び出すか (詳細については「XSiteAdmin」を参照)、CLI を使用する (詳細については「CLI でサイトをオフラインにする」を参照) ことによりオンラインに復帰させることができます。

バグを報告する

35.4. サイト間の状態転送

オフラインのマスターサイトがオンラインに復帰した場合に、バックアップサイトから最新のデータでマスターサイトの状態を同期する必要があります。状態の転送により、あるサイトから別のサイトに状態を転送できるため、マスターサイトがバックアップサイトと同期され、整合性が確保されます。同様に、バックアップサイトが利用可能になった場合に、状態の転送を使用してバックアップサイトとマスターサイトの整合性を確保できます。

マスターサイト A とバックアップサイト B の 2 つのサイトがあるとします。クライアントは最初にマスターサイト A のみにアクセスでき、バックアップサイト B は隠れたバックアップとして動作します。サイト間の状態転送は、双方向でプッシュできます。新しいバックアップサイト B がオンラインになると、マスターサイト A と状態を同期するために、状態の転送を開始してマスターサイト A からバックアップサイト B に状態をプッシュできます。

同様に、マスターサイト A がオンラインに復帰すると、バックアップサイト B と状態を同期するために、状態の転送を開始してバックアップサイト B からマスターサイト A に状態をプッシュできます。

このユースケースは Active-Passive 状態転送と Active-Active 状態転送の両方に適用されます。違いは、Active-Active 状態転送ではキャッシュ操作をサイトで実行できることを前提とします (状態が消費されます)。

システム管理者または権限があるユーザーは JMX を使用して手動で状態転送を開始します。システム管理者は、**`XSiteAdminOperations`** MBean で利用可能な **`pushState(SiteName String)`** 操作を呼び出します。

以下のインターフェースは、JConsole の **`pushState(SiteName String)`** 操作を示します。

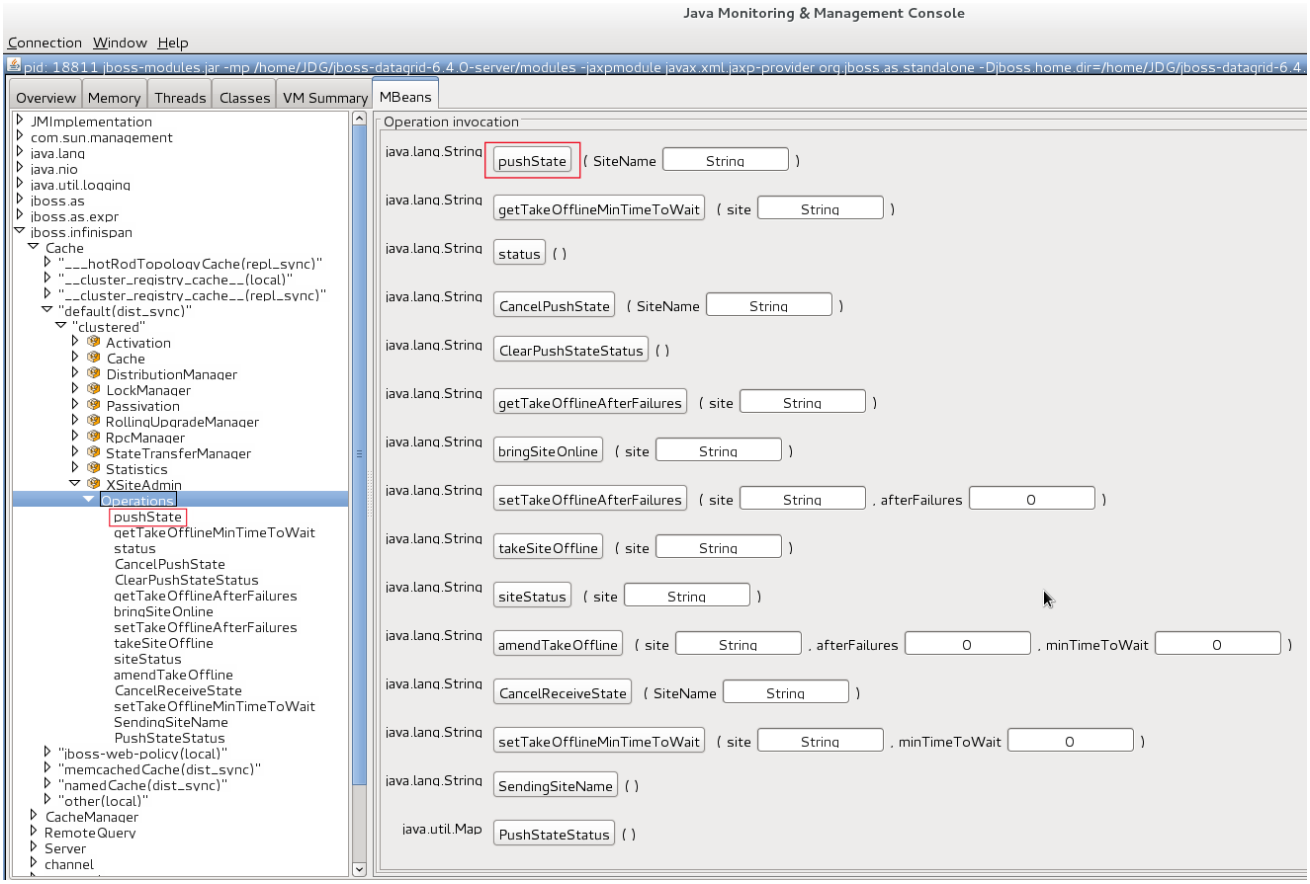
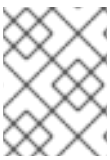


図35.2 PushState 操作

また、状態転送はコマンドラインインターフェース (CLI) を使用して **`site push sitename`** コマンドによって呼び出されます。たとえば、マスターサイトがオンラインに復帰すると、システム管理者は状態を受け取るマスターサイトの名前を指定してバックアップサイトで状態転送操作を呼び出します。

マスターサイトは、プッシュ操作の実行時にオフラインにできます。状態転送が正常に行われた場合に、両方のサイトに共通の状態データがマスターサイトで上書きされます。たとえば、キー A がバックアップサイトではなくマスターサイトに存在する場合、キー A はマスターサイトから削除されません。キー B がバックアップサイトとマスターサイトに存在する場合は、キー B がマスターサイトで上書きされます。



注記

状態転送の開始後に実行されたキーのアップグレードは、受け取る状態転送によって上書きされません。

サイト間状態転送は、トランザクション対応であり、1PC および 2PC トランザクションオプションをサポートします。1PC および 2PC オプションは、トランザクション内部で変更されたデータを 1 または 2 フェーズでリモートサイトにバックアップするかどうかを定義します。2PC にはトランザクショ

ンが正常に準備されたことをバックアップサイトが確認する準備フェーズが含まれます。両方のオプションがサポートされます。

バグを報告する

35.4.1. Active-Passive 状態転送

Active-Passive 状態転送は、サイト間レプリケーションを使用してマスターサイトをバックアップする場合に使用されます。マスターサイトはすべての要求を処理しますが、オフラインになった場合にバックアップサイトがそれらの要求を処理し始めます。マスターサイトがオンラインに復帰すると、マスターサイトはバックアップサイトから状態を受け取り、クライアント要求を処理し始めます。Active-Passive 状態転送モードでは、状態を送信するサイトの状態転送と同時にトランザクション書き込みが行われます。

Active-Passive 状態転送モードでは、クライアントの読み書き要求がバックアップサイトでのみ発生します。マスターサイトは、状態転送の完了時にクライアントの要求がマスターサイトに切り替わるまで隠れたバックアップとして機能します。Active-Passive 状態転送モードは、データセンター間レプリケーションで完全にサポートされます。

Active-Passive 状態転送がネットワーク障害により中断された場合は、システム管理者が JMX 操作を呼び出し、状態転送を手動で再開します。状態を (たとえば、マスターサイト A からバックアップサイト B に) 転送するには、マスターサイト A で JMX 操作を呼び出します。同様に、バックアップサイト B からマスターサイト A に状態を転送するには、バックアップサイト B で JMX 操作を呼び出します。

JMX 操作は、状態を同期するためにオンラインである他のサイトに状態を転送するサイトで呼び出されます。

たとえば、稼働しているバックアップサイトがあり、システム管理者がマスターサイトをオンラインに復帰させたいとします。この場合は、Active-Passive 状態転送を使用するために、システム管理者が以下の手順を実行します。

- マスターサイトで Red Hat JBoss Data Grid クラスターを起動します。
- バックアップサイトがマスターサイトに状態をプッシュするようにします。
- 状態転送が完了するまで待機します。
- マスターサイトが要求を処理できることをクライアントに通知します。

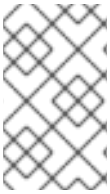
バグを報告する

35.4.2. Active-Active 状態転送

Active-Active 状態転送モードでは、状態転送の実行中にクライアントの要求が両方のサイトで同時に発生します。現在の実装では、状態転送の実行中に新しいサイトで要求を処理できます (これにより、データの整合性が確保されなくなることがあります)。

警告

Active-Active 状態転送モードは、完全にサポートされません。データの不整合が発生することがあります。



注記

Active-Active 状態転送モードでは、マスターサイトとバックアップサイトの両方で同じ役割を共有します。Active-Active 状態転送モードでは、マスターサイトとバックアップが明確に区別されません。

たとえば、稼働しているサイトがあり、システム管理者が新しいサイトをオンラインに復帰させたいとします。この場合は、Active-Passive 状態転送を使用するために、システム管理者が以下の手順を実行する必要があります。

- 新しいサイトで Red Hat JBoss Data Grid クラスターを起動します。
- 稼働しているサイトが新しいサイトに状態をプッシュするようにします。
- 新しいサイトが要求を処理できることをクライアントに通知します。

バグを報告する

35.4.3. 状態転送の設定

サイト間の状態転送は、有効でないか無効です。ただし、一部のパラメーターをチューニングできます。この設定は、状態転送中または状態転送後にマスターサイトに要求を切り替えるようロードバランサーを設定するときにシステム管理者のみが行えます。実装では、状態を受け取る前にキーがクライアントによって更新されるケースが処理されます (いつ送信するかは無視されます)。

デフォルトのパラメーター値は以下のとおりです。

```
<backups>
  <backup site="NYC"
   strategy="SYNC"
   failure-policy="FAIL">
    <state-transfer chunk-size="512"
      timeout="1200000"
      max-retries="30"
      wait-time="2000" />
 </backup>
</backups>
```

バグを報告する

35.5. 複数サイトマスターの設定

標準的な Red Hat JBoss Data Grid のデータセンター間のレプリケーション設定には、サイトごとに 1 つのマスターノードが含まれます。マスターノードは、他のサイトのマスターノードと通信するための他のノードのゲートウェイです。

この標準設定は、単純なデータセンター間のレプリケーション設定の場合に機能します。ただし、サイト間のトラフィック量が増えると、単一マスターノードを経由して渡されるトラフィックによりボトルネックが発生する可能性があり、その場合はノード間の通信が遅くなります。

JBoss Data Grid では、複数サイト間のトラフィックを最適化するために、各サイトに対して複数のマスターノードを設定します。

バグを報告する

35.5.1. 複数サイトマスターの操作

複数サイトマスターが有効な状態で設定されている場合、各サイトのマスターノードは、グローバルクラスター (複数サイトのメンバーであるノードが含まれる) と共に、ローカルクラスター (つまりローカルサイト) に加わります。

各ノードは、サイトマスターとして機能し、ターゲットサイトとサイトマスターのリストから構成されるルーティングテーブルを維持します。メッセージが到着すると、送信先サイトのランダムなマスターノードが選択されます。次にメッセージがランダムなマスターノードに転送され、そこで送信先ノードに送信されます (ランダムに選択されたノードが送信先ではない場合)。

バグを報告する

35.5.2. 複数サイトマスターの設定 (リモートクライアントサーバーモード)

前提条件

Red Hat JBoss Data Grid のリモートクライアントサーバーモード用にデータセンター間レプリケーションを設定します。

手順35.3 リモートクライアントサーバーモードで複数のサイトマスターを設定します。

```
<relay site="LON">
 <remote-site name="NYC" stack="tcp" cluster="global"/>
 <remote-site name="SFO" stack="tcp" cluster="global"/>
 <property name="relay_multicasts">false</property>
 <property name="max_site_masters">16</property>
 <property name="can_become_site_master">true</property>
</relay>
```

1. **ターゲット設定の特定**
   clustered-xsite.xml のサンプル設定ファイルでターゲットサイトの設定を見つけます。この設定例は、上記の例のようになります。

2. **最大サイトの設定**
   サイト内のマスターノードの最大数を決定するには ***max_site_masters*** プロパティーを使用します。すべてのノードをマスターにするために、この値をサイト内のノード数に設定します。

3. **サイトマスターの設定**
   ノードをサイトマスターにすることを許可するには、***can_become_site_master*** プロパティーを使用します。このフラグはデフォルトで **true** に設定されます。このフラグを **false** に設定することにより、ノードがサイトマスターになることを防ぐことができます。これは、ノードに外部ネットワークに接続されたネットワークインターフェースがない場合に必要になります。

バグを報告する

35.5.3. 複数サイトマスターの設定 (ライブラリーモード)

Red Hat JBoss Data Grid のライブラリーモードで複数サイトマスターを設定するには、以下を実行します。

手順35.4 複数サイトマスターの設定 (ライブラリーモード)

1. **データセンター間レプリケーションの設定**
   JBoss Data Grid でデータセンター間レプリケーションを設定します。XML 設定については「データセンター間レプリケーションを宣言的に設定する」の手順を使用し、プログラミングによる設定の場合は、JBoss Data Grid 『Developer Guide』 を参照します。

2. **設定ファイルの内容を追加します。**
   以下のように ***can_become_site_master*** および ***max_site_masters*** パラメーターを設定に追加します。

```
<config>
    <!-- Additional configuration information here -->
    <relay.RELAY2 site="LON"
                config="relay.xml"
                relay_multicasts="false"
                can_become_site_master="true"
                max_site_masters="16"/>
</config>
```

   すべてのノードをマスターにするために、***max_site_masters*** 値をクラスター内のノード数に設定します。

バグを報告する

第36章 ローリングアップグレード

Red Hat JBoss Data Grid では、ローリングアップグレードは、クラスターの 1 つのバージョンから新バージョンへのダウンタイムなしのアップグレードを可能にします。これにより、ノードのアップグレードを、アプリケーションの再起動なしに、またはデータ損失のリスクを伴わずに実行することができます。

JBoss Data Grid では、ローリングアップグレードはリモートクライアントサーバーモードでのみ実行できます。

重要

ローリングアップグレードを実行する際には、データの不整合が生じる可能性があるため、ソースクラスター内のキャッシュエントリーは更新しないことをお勧めします。

バグを報告する

36.1. HOT ROD を使用したローリングアップグレード

以下のプロセスは、Hot Rod を使用してリモートクライアントサーバーで実行されている Red Hat JBoss Data Grid でローリングアップグレードを実行するために使用されます。

重要

お使いの JBoss Data Grid バージョンで Hot Rod プロトコルの正しいバージョンが使用されていることを確認してください。バージョンは、プログラムを用いてクライアントに指定する必要があります。またこれを定義する方法については、JBoss Data Grid の『Developer Guide』 を参照してください。各リリースの Hot Rod プロトコルバージョンは以下の通りです。

- JBoss Data Grid 7.0 の場合、Hot Rod プロトコルバージョン 2.5 を使用
- JBoss Data Grid 6.6 の場合、Hot Rod プロトコルバージョン 2.3 を使用
- JBoss Data Grid 6.5 の場合、Hot Rod プロトコルバージョン 2.0 を使用
- JBoss Data Grid 6.4 の場合、Hot Rod プロトコルバージョン 2.0 を使用
- JBoss Data Grid 6.3 の場合、Hot Rod プロトコルバージョン 2.0 を使用
- JBoss Data Grid 6.2 の場合、Hot Rod プロトコルバージョン 1.3 を使用
- JBoss Data Grid 6.1 の場合、Hot Rod プロトコルバージョン 1.2 を使用

前提条件

この手順では、クラスターがすでに設定されており、実行中であること、およびクラスターが JBoss Data Grid の古いバージョンを使用していることを想定しています。このクラスターは以下ではソースクラスターとして言及されており、ターゲットクラスターはデータの移行先となる新規クラスターを指します。

1. **ターゲットクラスターの設定**
   ソースクラスターとは別にターゲットクラスター (新規 JBoss Data Grid のあるノードで構成) を設定するには、別のネットワーク設定または別の JGroups クラスター名のいずれかを使用します。それぞれのキャッシュについては、以下の設定で `RemoteCacheStore` を設定します。

a. ***remote-server*** がソースクラスターを参照するようにします。

b. キャッシュ名がソースクラスターのキャッシュの名前と一致するようにします。

c. ***hotrod-wrapping*** を有効に (**true** に設定) します。

d. ***purge*** を無効に (**false** に設定) します。

e. ***passivation*** を無効に (**false** に設定) します。

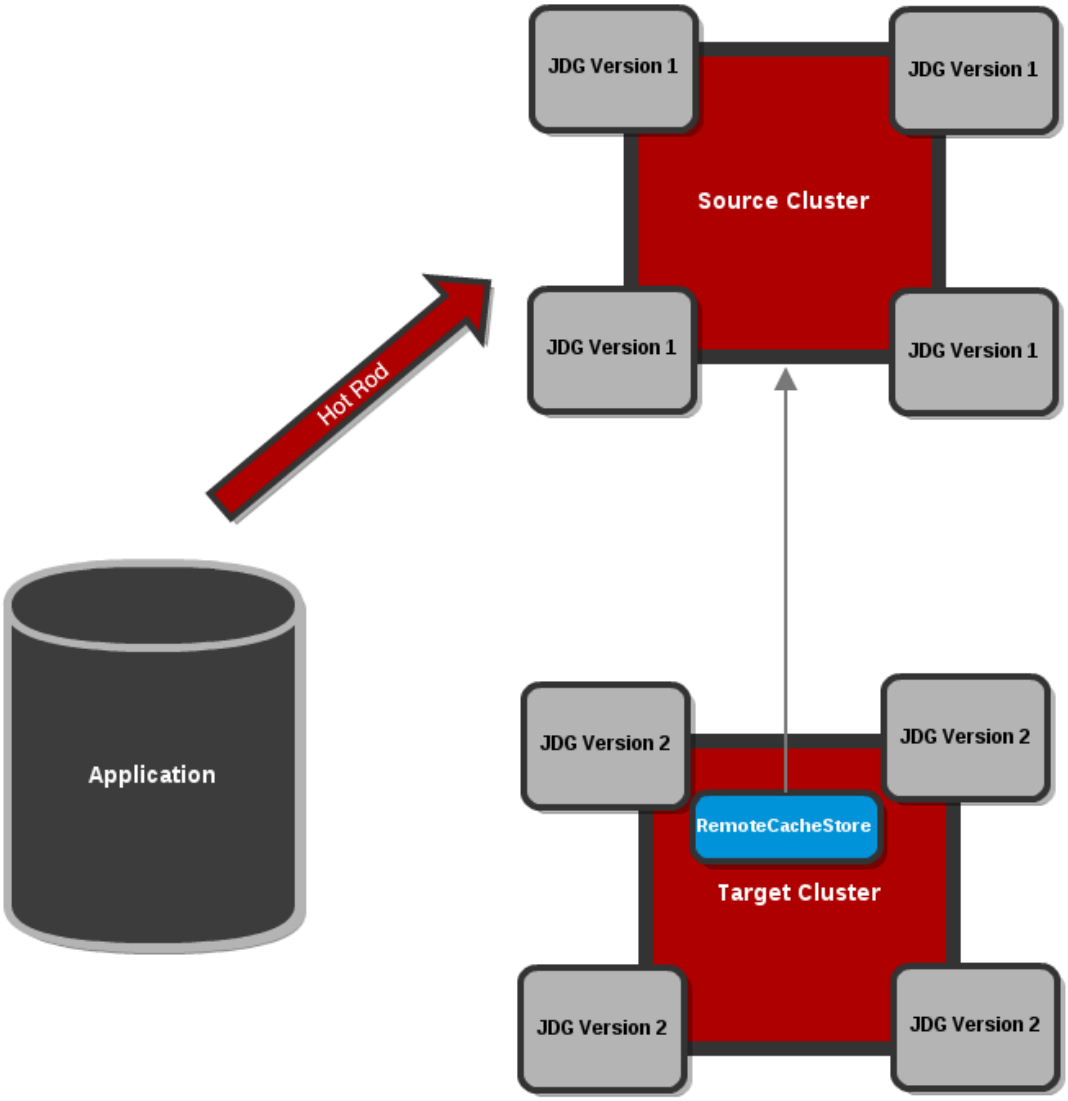


図36.1 RemoteCacheStore によるターゲットクラスターの設定

注記

ローリングアップグレードを実行する際のターゲットクラスター設定の詳細の例については、**$JDG_HOME/docs/examples/configs/standalone-hotrod-rolling-upgrade.xml** ファイルを参照してください。

2. **ターゲットクラスターの起動**
ターゲットクラスターのノードを起動します。ソースクラスターではなく、ターゲットクラス

ターを指すように各クライアントを設定します。最終的に、ターゲットクラスターはソースクラスターの代わりにすべての要求を処理します。その後ターゲットクラスターは **RemoteCacheStore** を使用してソースクラスターのデータをオンデマンドでロードします。

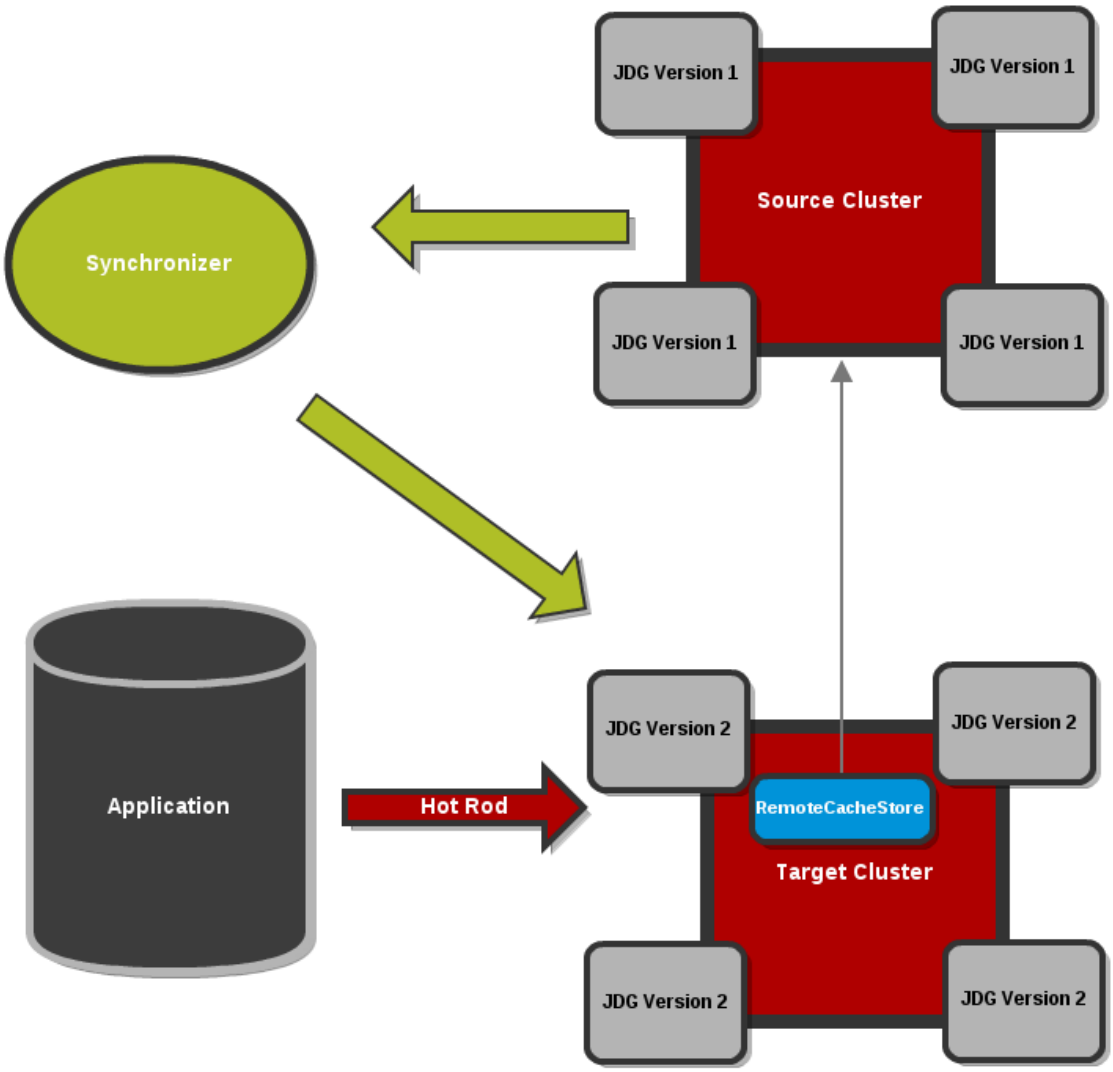


図36.2 ソースクラスターをターゲットクラスターのRemoteCacheStore としてターゲットクラスターを指すクラスター。

3. **ソースクラスターのキーセットのダンプ**
   すべての接続がターゲットクラスターを使用している場合、ソースクラスターのキーセットをダンプする必要があります。これは JMX または CLI のいずれかを使用して実行できます。

   - **JMX**
     移行する必要のあるすべてのキャッシュについて、ソースクラスターの **RollingUpgradeManager** MBean で ***recordKnownGlobalKeyset*** 操作を起動します。

   - **CLI**
     移行する必要のあるすべてのキャッシュについて、ソースクラスターで **upgrade --dumpkeys** コマンドを起動するか、または **--all** スイッチを使用してクラスターのすべてのキャッシュをダンプします。

4. **ソースクラスターからの残りのデータの取り込み**
   ターゲットクラスターはソースクラスターから残りのすべてのデータを取り込みます。これも JMX または CLI のいずれかを使用して実行できます。

- **JMX**
  synchronizeData 操作を起動し、移行する必要のあるすべてのキャッシュについて、ターゲットクラスターの **RollingUpgradeManager** MBean で ***recordKnownGlobalKeyset*** を指定します。

- **CLI**
  移行する必要のあるすべてのキャッシュについて、ターゲットクラスターで **upgrade --synchronize=hotrod** コマンドを起動するか、または **--all** スイッチを使用してクラスター内のすべてのキャッシュを同期します。

5. **RemoteCacheStore の無効化**
   ターゲットクラスターがソースクラスターからすべてのデータを取得した後に、ターゲットクラスターの **RemoteCacheStore** を無効にする必要があります。これは以下のように実行できます。

   - **JMX**
     ターゲットクラスターの **RollingUpgradeManager** MBean に ***hotrod*** パラメーターを指定して **disconnectSource** 操作を起動します。

   - **CLI**
     ターゲットクラスターで **upgrade --disconnectsource=hotrod** コマンドを起動します。

6. **ソースクラスターの使用の停止**
   最終手順としてソースクラスターの使用を停止します。

バグを報告する

36.2. REST を使用したローリングアップグレード

以下の手順では、REST プロトコルを使用して Red Hat JBoss Data Grid インストールをリモートグリッドとして使用する方法を簡単に説明します。この手順は、クライアントアプリケーションではなく、グリッドのローリングアップグレードに適用されます。

手順36.1 REST を使用したローリングアップグレードの実施

この手順では、ソースクラスターは、現在使用中の古いクラスターを指し、ターゲットクラスターはデータの宛先クラスターを指します。

1. **ターゲットクラスターの設定**
   ソースクラスターとは別にターゲットクラスター (新規 JBoss Data Grid のノードで構成) を設定するには、別のネットワーク設定または別の JGroups クラスター名のいずれかを使用します。それぞれのキャッシュについては、以下の設定を考慮に入れて **RestCacheStore** を設定します。

   a. ホストおよびポートの値がソースクラスターを参照するようにする。

   b. パスの値がソースクラスターの REST エンドポイントを参照するようにする。

2. **ターゲットクラスターの開始**
   ターゲットクラスターのノードを起動します。ソースクラスターではなく、ターゲットクラスターを参照するように各クライアントを設定します。最終的に、ターゲットクラスターはソースクラスターの代わりにすべての要求を処理します。次にターゲットクラスターは **RestCacheStore** を使用してソースクラスターのデータをオンデマンドでレイジーにロードします。

3. **REST ローリングアップグレード時にはキーセットをダンプしない**
   REST ローリングアップグレードのユースケースは、**recordKnownGlobalKeyset** 操作を使用せずにすべてのデータをソースクラスターから取得することが意図されています。

警告

REST ローリングアップグレードの **recordKnownGlobalKeyset** 操作は起動しないでください。この操作を起動すると、データの破損が生じ、REST ローリングアップグレードは正常に完了しません。

4. **残りのデータの取得**
   ターゲットクラスターはソースクラスターから残りのデータすべてを取得する必要があります。これは、以下のように JMX または CLI を使用して実行されます。

   a. **JMX の使用**
      すべてのキャッシュが移行されるように、**rest** パラメーターをターゲットクラスターの **RollingUpgradeManager** MBean に指定して **synchronizeData** 操作を起動します。

   b. **CLI の使用**
      すべてのキャッシュが移行されるように **upgrade --synchronize=rest** をターゲットクラスターで実行します。オプションで、***--all*** スイッチを使用して、クラスター内のすべてのキャッシュを同期します。

5. **RestCacheStore の無効化**
   JMX または CLI のいずれかを使用して、以下のようにターゲットクラスターで **RestCacheStore** を無効にします。

   a. **JMX の使用**
      rest パラメーターをターゲットクラスターの**RollingUpgradeManager** MBean に指定して **disconnectSource** 操作を起動します。

   b. **CLI の使用**
      ターゲットクラスターで **upgrade --disconnectsource=rest** コマンドを実行します。オプションで、***--all*** スイッチを使用して、クラスター内のすべてのキャッシュの接続を解除します。

結果

ターゲットクラスターへの移行は完了しました。ソースクラスターの使用を停止できます。

バグを報告する

36.3. ROLLINGUPGRADEMANAGER 操作

RollingUpgradeManager Mbean は、ローリングアップグレードの実施時に Red Hat JBoss Data Grid の 1 つのバージョンから別のバージョンへのデータ移行を可能にする操作を処理します。**RollingUpgradeManager** 操作には以下が含まれます。

- ***recordKnownGlobalKeyset*** は、JBoss Data Grid の古いバージョンで実行されるクラスターからキーセット全体を取得します。

- ***synchronizeData*** は、ソースクラスターから JBoss Data Grid の新しいバージョンを実行しているターゲットクラスターへのデータ移行を実行します。
- ***disconnectSource*** は、ターゲットクラスターへのデータ移行が完了すると、JBoss Data Grid の古いバージョンであるソースクラスターを無効にします。

バグを報告する

36.4. ローリングアップグレードの REMOTECACHESTORE パラメーター

36.4.1. rawValues および RemoteCacheStore

デフォルトで、RemoteCacheStore ストアの値は InternalCacheEntry にラップされます。***rawValues*** パラメーターを有効にすると、RemoteCacheManager による直接のアクセスとの相互運用性を図るために生の値を代わりに格納することができます。

rawValues は、RemoteCacheStore および RemoteCacheManager の両方で Hot Rod キャッシュとの対話を可能にするために有効にされる必要があります。

バグを報告する

36.4.2. hotRodWrapping

hotRodWrapping パラメーターは、ローリングアップグレードを実行するために rawValues を有効にし、適切なマーシャラーおよびエントリーラッパーを設定するショートカットです。

バグを報告する

第37章 カスタムインターセプター

カスタムインターセプターは、宣言的に、またはプログラムを用いて Red Hat JBoss Data Grid に追加できます。カスタムインターセプターは、キャッシュの変更に影響を与えたり、キャッシュの変更に応答したりすることにより JBoss Data Grid を拡張します。キャッシュの変更例には、要素またはトランザクションの追加、削除、または更新などがあります。

警告

カスタムインターセプターのサポートは JBoss Data Grid 7.0 では非推奨になります。カスタムインターセプターを実行する新たな方法が JBoss Data Grid 7.1 で導入される予定です。さらに、インターセプタースタックは JBoss Data Grid の内部 API の一部であり、リリースごとに変更される可能性があります。このため、ご使用のアプリケーションからカスタムインターセプターを直接使用することはお勧めしません。

バグを報告する

37.1. カスタムインターセプター設計

Red Hat JBoss Data Grid でカスタムインターセプターを設計するには、以下のガイドラインに従ってください。

- カスタムインターセプターは **CommandInterceptor** を拡張しなければなりません。
- カスタムインターセプターはインスタンス化を可能にするために、パブリックの空のコンストラクターを宣言しなければなりません。
- カスタムインターセプターでは、**property** 要素を介して定義されたプロパティーに対して JavaBean スタイルセッターを定義する必要があります。

バグを報告する

37.2. カスタムインターセプターを宣言して追加

Red Hat JBoss Data Grid のそれぞれの名前付きキャッシュは独自のインターセプタースタックを持ちます。結果として、カスタムインターセプターは名前付きキャッシュごとに追加できます。

カスタムインターセプターは、XML を使用して追加できます。以下の手順に従ってカスタムインターセプターを追加します。

手順37.1 カスタムインターセプターの追加

```
<local-cache name="cacheWithCustomInterceptors">
   <custom-interceptors>
      <interceptor position="FIRST"
class="com.mycompany.CustomInterceptor1">
        <property name="attributeOne" value="value1" />
        <property name="attributeTwo" value="value2" />
```

```
      </interceptor>
      <interceptor position="LAST"
class="com.mycompany.CustomInterceptor2"/>
      <interceptor index="3" class="com.mycompany.CustomInterceptor1"/>
      <interceptor before="org.infinispan.interceptors.CallInterceptor"
class="com.mycompany.CustomInterceptor2"/>
      <interceptor after="org.infinispan.interceptors.CallInterceptor"
class="com.mycompany.CustomInterceptor1"/>
   </customInterceptors>
</local-cache>
```

1. **カスタムインターセプターの定義**
   すべてのカスタムインターセプターは **`org.infinispan.interceptors.base.BaseCustomInterceptor`** を拡張する必要があります。

2. **新規カスタムインターセプターの位置の定義**
   インターセプターの位置を定義しておく必要があります。オプションは相互に排他的であり、インターセプターは position 属性と index 属性を両方持つことができません。有効なオプションは以下の通りです。

   - **Position 属性の使用**
     - **FIRST:** 新規インターセプターがチェーンの最初に置かれるように指定します。
     - **LAST:** 新規インターセプターがチェーンの最後に置かれるように指定します。
     - **OTHER_THAN_FIRST_OR_LAST:** 新規インターセプターをチェーンの最初と最後以外の任意の場所に配置できるように指定します。
   - **Index 属性の使用**
     - **`index`** は、このインターセプターのチェーン内の位置を特定します。この index は、チェーン内の最初の位置として 0 から始まり、所定の設定内のインターセプターの数まで続きます。
   - **Before または After 属性の使用**
     - **`after`** 属性は、完全修飾クラス名で指定された、名前付きインターセプターのインスタンスの後に新規インターセプターを直接配置します。
     - **`before`** 属性は、完全修飾クラス名で指定された、名前付きインターセプターのインスタンスの前に新規インターセプターを直接配置します。
   - **インターセプタープロパティーの定義**
     特定のインターセプタープロパティーを定義します。

3. **他のカスタムインターセプターの適用**
   この例では、次のカスタムインターセプターは CustomInterceptor2 と呼ばれます。

注記

カスタムインターセプターに Position の **`OTHER_THAN_FIRST_OR_LAST`** が指定されると、CacheManager が失敗する可能性があります。

注記

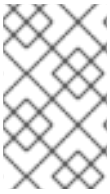
この設定は JBoss Data Grid のライブラリーモードでのみ有効です。

バグを報告する

第38章 セッションの外部化

38.1. JBOSS EAP から JBOSS DATA GRID への HTTP セッションの外部化

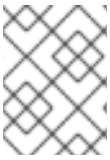
Red Hat JBoss Data Grid は、HTTP セッションなどの、JBoss Enterprise Application Platform (EAP) のアプリケーション固有データの外部キャッシュコンテナーとして使用できます。これにより、アプリケーションとは独立したデータレイヤーのスケーリングが可能になり、さまざまなドメインに存在する可能性がある異なる EAP クラスターが同じ JBoss Data Grid クラスターからデータにアクセスできるようになります。また、他のアプリケーションは Red Hat JBoss Data Grid により提供されたキャッシュと対話できます。



注記

以下の手順は、JBoss EAP 7.0 および JBoss Data Grid 7.0 で機能するようテストされ、確認されています。HTTP セッションを JBoss Data Grid 7.x で外部化する場合、これらのみを使用するか、または各製品の各種バージョンを後で使用します。

以下の手順は、EAP のスタンドアロンモードとドメインモードの両方に適用されます。ただし、ドメインモードでは、各サーバーグループで一意のリモートキャッシュが設定されている必要があります。複数のサーバーグループは同じ Red Hat JBoss Data Grid クラスターを使用できますが、各リモートキャッシュは EAP サーバーグループに対して一意になります。



注記

分散可能なアプリケーションごとに完全に新しいキャッシュを作成する必要があります。新しいキャッシュは Web などの既存のキャッシュコンテナーに作成できます。

手順38.1 HTTP セッションの外部化

1. リモートキャッシュコンテナーが EAP の **infinispan** サブシステムで定義されるようにします。以下の例では、**remote-store** 要素の ***cache*** 属性によって、リモート JBoss Data Grid サーバーのキャッシュ名を定義します。

```
<subsystem xmlns="urn:jboss:domain:infinispan:4.0">
  [...]
  <cache-container name="cacheContainer" default-cache="default-
cache" module="org.jboss.as.clustering.web.infinispan" statistics-
enabled="true">
    <transport lock-timeout="60000"/>
    <replicated-cache name="default-cache" mode="SYNC">
      <locking isolation="REPEATABLE_READ"/>
      <transaction mode="BATCH"/>
      <remote-store remote-servers="remote-jdg-server1 remote-jdg-
server2"
                    cache="default" socket-timeout="60000"
                    preload="true" passivation="false"
purge="false" shared="true"/>
    </replicated-cache>
  </cache-container>
</subsystem>
```

2. ネットワーク情報を **socket-binding-group** に追加することにより、リモート Red Hat JBoss Data Grid サーバーの場所を定義します。

```
<socket-binding-group ...>
  <outbound-socket-binding name="remote-jdg-server1">
    <remote-destination host="JDGHostName1" port="11222"/>
  </outbound-socket-binding>
  <outbound-socket-binding name="remote-jdg-server2">
    <remote-destination host="JDGHostName2" port="11222"/>
  </outbound-socket-binding>
</socket-binding-group>
```

3. 各キャッシュコンテナーと各 Red Hat JBoss Data Grid サーバーに対して上記の手順を繰り返します。定義された各サーバーでは、個別の **<outbound-socket-binding>** 要素が定義されている必要があります。

4. パッシベーションとキャッシュ情報をアプリケーションの **jboss-web.xml** に追加します。以下の例では、**cacheContainer** はキャッシュコンテナーの名前であり、**default-cache** はこのコンテナーにあるデフォルトのキャッシュの名前です。サンプルファイルを以下に示します。

```
<?xml version="1.0" encoding="UTF-8"?>
<jboss-web xmlns="http://www.jboss.com/xml/ns/javaee"
           xmlns:xsi="http://www.w3.org/2001/XMLSchema-instance"
           xsi:schemaLocation="http://www.jboss.com/xml/ns/javaee
http://www.jboss.org/j2ee/schema/jboss-web_10_0.xsd"
           version="10.0">

    <replication-config>
        <replication-granularity>SESSION</replication-granularity>
        <cache-name>cacheContainer.default-cache</cache-name>
    </replication-config>

</jboss-web>
```

注記

上記のパッシベーションタイムアウトは、通常のセッションが 15 分以内に破棄され、JBoss EAP のデフォルトの HTTP セッションタイムアウトが 30 分であることを前提として提供されています。これらの値は、各アプリケーションのワークロードに 基づいて調整する必要がある場合があります。

バグを報告する

第39章 データの相互運用性

39.1. ライブラリーとリモートクライアントサーバーエンドポイント間の相互運用性

Red Hat JBoss Data Grid は、以下のようなデータと対話するための複数の方法を提供します。

- ローカル (組み込み型) の方法でデータを保存し、取得する。
- 各種のエンドポイントを使用してデータを保存し、取得する。

以前のバージョンの JBoss Data Grid では、これらのメソッドのいずれかを選択すると、データを保存するために使用したメソッドと同じメソッドがデータの取得に必要でした。たとえば、組み込みモードでデータを保存すると、REST エンドポイントを使用するなどの別のメソッドではなく、組み込みモードでデータを取得する必要がありました。

JBoss Data Grid は、データの保存に使用したメソッドの種類を問わず、ユーザーが任意のユーザーインターフェースでデータにアクセスできる互換性モードを提供するようになりました。互換性モードでは、データ形式は各エンドポイント用に自動的に変換されるため、情報の保存や取得に異なるインターフェースを使用できます。

JBoss Data Grid の互換性モードは、データについての以下を仮定の基づいています。互換性モードが最もよく使用されるのは Java オブジェクトを保存する場合であるため、互換性モードは、有効にされるとデータの保存時にデータをマーシャル解除し、シリアライズを解除します。そのため、組み込みキャッシュと共に Java オブジェクトを使用する際に効率性が高まります。

互換性モードのパフォーマンスコストは、非互換性モードよりも高くなります。そのため、この機能は JBoss Data Grid ではデフォルトで無効にされます。

バグを報告する

39.2. 互換性モードの使用

Red Hat JBoss Data Grid の互換性モードでは、以下が予定通りに機能する必要があります。

- すべてのエンドポイント設定が同じキャッシュマネージャーを指定する
- すべてのエンドポイントが同じターゲットキャッシュと対話する

JBoss Data Grid のリモートクライアントサーバーディストリビューションを使用してこれらの要件に基づいてデフォルトで設定されていることを確認します。

バグを報告する

39.3. プロトコルの相互運用性

Red Hat JBoss Data Grid のプロトコルの相互運用性により、raw バイト形式のデータへの、 C++ または Java などの各種プログラミング言語で書かれた REST、Memcacherd、ライブラリー、および Hot Rod などの各種のプロトコルを使った読み取り/書き込みアクセスを可能にします。

デフォルトで、各プロトコルはそのプロトコルの最も効率的な形式でデータを保存し、エントリーの取得時に変換が必要とならないようにします。このデータを複数のプロトコルからアクセスする必要がある場合、互換性モードは、共有されるキャッシュで有効にされる必要があります。

互換性モードの宣言的な有効化

compatibility 要素の ***marshaller*** パラメーターをカスタムマーシャラーに設定して互換性の変換を有効にします。この例は以下のようになります。

例39.1 互換性モードの有効化

```
<cache-container name="local" default-cache="default" statistics="true">
    <local-cache name="default" start="EAGER" statistics="true">
        <compatibility marshaller="com.example.CustomMarshaller"/>
    </local-cache>
</cache-container>
```

バグを報告する

第40章 ネットワークパーティションの処理 (スプリットブレイン)

ネットワークパーティションは、クラスターが 2 つ以上のパーティションに分割される場合に作成されます。結果として、各パーティションのノードは他のパーティションのノードを見つけたり、そのノードと通信したりできません。この結果、意図せずにネットワークが分割されます。

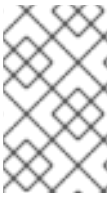
Red Hat JBoss Data Grid などの分散システムでネットワークパーティションが発生した場合は、CAP (ブリュワーの) 定理を使用します。CAP 定理によると、ネットワークパーティション (P) の発生時に、分散システムがデータの整合性 (C) または利用可能性 (A) のいずれかを提供できます (ただし、両方は提供できません)。

デフォルトでは、JBoss Data Grid でパーティション処理は無効になっています。ネットワークパーティションの発生時に、パーティションは整合性 (C) を犠牲にして利用可能性 (A) を維持します。

ただし、パーティション処理が有効な場合は、JBoss Data Grid により利用可能性 (A) よりも整合性 (C) が優先されます。

Red Hat JBoss Data Grid は、分割されたネットワークを修復するためにプライマリーパーティションストラテジーを提供します。ネットワークパーティションが発生し、キャッシュが 2 つ以上のパーティションに分割されると、最大 1 つのパーティションがプライマリーパーティションになり (利用可能な状態)、他のパーティションがセカンダリーパーティションとして指定されます (劣化モードになります)。パーティションがマージされ 1 つのキャッシュに戻ると、プライマリーパーティションはすべてのセカンダリーパーティションの参照として使用されます。セカンダリーパーティションのすべてのメンバーは現在の状態の情報を削除し、プライマリーパーティションのメンバーからの最新の状態情報で置き換える必要があります。分割時にプライマリーパーティションがない場合は、各ノードの状態が正しいと見なされます。

JBoss Data Grid では、キャッシュは数多くのノードに格納されたデータから構成されます。ノードで障害が発生した場合にデータ損失を防ぐために、JBoss Data Grid は複数のノードでデータアイテムをレプリケートします。分散モードでは、この冗長性は ***owners*** 設定属性を使用して設定され、キャッシュ内の各キャッシュエントリーのレプリカの数が指定されます。結果として、障害が発生したノードの数が ***owners*** の値よりも小さい限り、JBoss Data Grid は損失データのコピーを保持し、復元できます。



注記

ただし、JBoss Data Grid のレプリケーションモードでは、各ノードにキャッシュ内の各データアイテムのコピーが含まれるため、***owners*** は常にキャッシュ内のノードの数と同じになります。

場合によっては、***owners*** の値よりも大きいノードの数がキャッシュからなくなることがあります。この一般的な理由は、以下の 2 つのです。

- スプリットブレイン: 通常は、ルーターのクラッシュの結果、キャッシュが 2 つ以上のパーティションに分割されます。各パーティションは他のパーティションに関係なく動作し、各パーティションには同じデータの別のバージョンが含まれることがあります。
- 連続クラッシュノード: ***owners*** の値よりも大きいノードの数が何らかの理由で連続してクラッシュします。JBoss Data Grid は、クラッシュの間に状態を適切に分散できず、結果として部分的なデータ損失が発生します。

バグを報告する

40.1. スプリットブレインの検出および回復

スプリットブレインがデータグリッドで発生した場合、各ネットワークパーティションには、削除された他のパーティションからノードと共に独自の JGroups ビューがインストールされます。パーティションはお互いを認識しないため、ネットワークが分割されたパーティションの数を判断することはできません。Red Hat JBoss Data Grid では、1 つまたは複数のノードが明示的なメッセージを送信せずに JGroups キャッシュから消失した場合に (実際の原因は物理的 (スイッチのクラッシュやケーブルの障害など) または仮想的 (stop-the-world ガーベッジコレクション))、キャシュが予期せずに分割されることを前提とします。

新しく分割された各パーティションは独立して稼働し、同じデータエントリーの競合アップデートを保存することがあるため、この状態は危険です。

パーティション処理モードが有効であり (手順については「パーティション処理の設定」を参照)、JBoss Data Grid が 1 つまたは複数のノードにアクセスできないことを疑う場合は、各パーティションで再調整がすぐに開始されませんが、代わりに劣化モードに切り替えるかどうかがチェックされます。劣化モードに切り替えるには、以下の以下のいずれかの条件が満たされている必要があります。

- 少なくとも 1 つのセグメントがすべての所有者を失います。つまり、***owners*** の値以上の数のノードが JGroups ビューから脱退します。
- パーティションには、最新の安定したトポロジーの過半数のノード (半分超) が含まれます。安定したトポロジーは、再調整操作が正常に行われ、コーディネーターが追加の再調整が必要ないことを確認するたびに更新されます。

いずれの条件も満たさない場合は、パーティションが引き続き正常に稼働し、JBoss Data Grid がノードを再調整しようとします。これらの条件に基づき、最大 1 つのパーティションを利用可能モードのままにすることができます。他のパーティションは劣化モードになります。

パーティションが劣化モードになると、エントリーのすべての所有者 (コピー) が同じパーティション内のノードに存在するエントリーの読み取り/書き込みアクセスのみが許可されます。1 人または複数の所有者がプラットフォームから消失したノードに存在するエントリーの読み取りおよび書き込み要求が **`AvailabilityException`** で拒否されます。

注記

制限としては、2 つのパーティションが分離パーティションとして開始され、マージしない場合、これらのパーティションは不整合なデータを読み取り、書き込むことができます。JBoss Data Grid はこのようなパーティションを分割パーティションと認識しません。

警告

データ整合性は、キャッシュが物理的に分割されたとき (t1) から JBoss Data Grid が接続の変更を検出し、パーティションの状態を変更するとき (t2) までの間で損なわれることがあります。

- 物理的な分割が行われたとき (t1) に処理中だったトランザクション書き込みが一部の所有者でロールバックされることがあります。また、これにより、このような書き込みで影響を受けたエントリーのコピー (パーティションの再参加後) 間で不整合が発生することがあります。ただし、t1 後に開始されたトランザクション書き込みは予想どおりに失敗します。
- 書き込みが非トランザクションである場合は、この時間枠の間に、(物理的に分割され、パーティションがまだ劣化状態ではないため) マイナーパーティションにのみ書き込まれた値がパーティションの再参加時に失われることがあります (このマイナーパーティションは再参加時にプライマリー (利用可能な) パーティションから状態を受け取ります)。パーティションが再参加時に状態を受け取らない場合 (つまり、すべてのパーティションが劣化状態)、値は失われませんが、不整合が維持されることがあります。
- マイナーパーティションが劣化状態になるまでエントリーが引き続き利用可能になるため、この移行期間中にマイナーパーティションで無効な読み取りが発生することもあります。

ネットワークパーティションの実行後にパーティションがマージされる場合は、以下の状況が発生します。

- ネットワークパーティション中にいずれかのパーティションが利用可能な場合は、参加パーティションが消去され、利用可能な (パーティション) パーティションから参加ノードへの状態転送が実行されます。
- すべての参加パーティションがスプリットブレインの間に劣化状態にある場合は、マージ中に状態転送が実行されません。結合されたキャッシュは、メージパーティションに最新の安定したトポロジー (トポロジー ID が最大のトポロジー) の過半数のメンバーが含まれ、各セグメントに対して少なくとも 1 の所有者がいる (つまり、キーは失われません) 場合にのみ利用可能になります。

警告

パーティションがマージを開始したとき (t1) からマージが完了したとき (t2) の間に、ノードは一連のマージイベントを介して再接続されます。この時間枠の間に、ノードがクラスターを一時的に脱退したとして報告されることがあります。トランザクションキャッシュの場合、t1 から t2 の時間枠にこのようなノードが他のノードにまたがるトランザクションを実行すると、このトランザクションがリモートノードで実行されないことがあります (ただし、元のノードでは正常に実行されます)。結果として、このトランザクションをコミットしなかったノードで影響を受けたエントリーの値が無効になることがあります。

t2 の後に、すべてのノードでマージが完了すると、この状況は以降のトランザクションで発生しません。ただし、t1 と t2 の間の時間枠に実行中のトランザクションによって影響を受けたエントリーで発生した不整合は、これらのエントリーが結果的に更新または削除されるまで解決されません。それまでは、影響を受けたこのようなエントリーの読み取りによって無効な値が返されることがあります。

バグを報告する

40.2. スプリットブレインのタイミング: スプリットの検出

FD_ALL プロトコルを使用すると、以下のミリ秒の経過後に特定のノードが疑われます。

```
FD_ALL.timeout + FD_ALL.interval + VERIFY_SUSPECT.timeout +
GMS.view_ack_collection_timeout
```

重要

上記の式で使用される時間は、JBoss Data Grid が脱退なしのクラスタービューのインストールに要する時間です。ただし、JVM の過剰なガーベッジコレクション (GC) 時間内で実行される JBoss Data Grid は、上記の障害検出時間を超える時間を設定できます。JBoss Data Grid はこれらの GC 時間を制御しないため、コーディネーターの過剰な GC により、検出は GC 時間に等しい時間分遅延する可能性があります。

バグを報告する

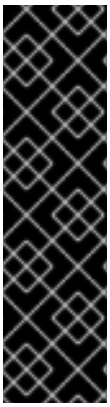
40.3. スプリットブレインのタイミング: スプリットからの回復

スプリットが発生すると、JBoss Data Grid はパーティションをマージし戻します。ネットワークパーティションの自動修復後のマージを検出するための最大時間は以下のようになります。

```
3.1 * MERGE3.max_interval
```

スプリット後に複数のマージが発生し、クラスターにすべての利用可能なパーティションが含まれる場合があります。この場合、複数のマージが発生すると、これらすべてが完了するまでの時間を確保する必要があります。また 3 つのマージが順番に発生する可能性もあるため、遅延時間の合計は以下を下回っている必要があります。

```
10 * MERGE3.max_interval
```



重要

上記の式で使用される時間は、JBoss Data Grid が脱退なしのクラスタービューのインストールに要する時間です。ただし、JVM の過剰なガーベッジコレクション (GC) 時間内で実行される JBoss Data Grid は、上記の障害検出時間を超える時間を設定できます。JBoss Data Grid はこれらの GC 時間を制御しないため、コーディネーターの過剰な GC により、検出は GC 時間に等しい時間分遅延する可能性があります。

さらにクラスタービューのマージの実行時に、JBoss Data Grid はすべてのメンバーがいることを確認しようとします。ただし、これらの応答の待ち時間に上限はなく、クラスタービューのマージはネットワークの問題により遅れる可能性があります。

バグを報告する

40.4. 連続してクラッシュしたノードの検出および回復

Red Hat JBoss Data Grid は、プロセスまたはマシンのクラッシュのため、またはネットワーク障害のため、ノードがクラスターを脱退したかどうかを区別できません。

1 つのノードがクラスターを脱退し、***owners*** の値が 1 よりも大きい場合は、クラスターが引き続き利用可能になり、JBoss Data Grid が損失データの新しいレプリカを作成しようとします。ただし、この再調整プロセス中に追加のノードがクラッシュした場合は、いくつかのエントリーに対して、データのすべてのコピーがノードに存在しないため、回復できないことがあります。

連続してクラッシュしたノードからデータグリッドを保護するために推奨される方法は、パーティションの処理を有効にし (手順については「パーティション処理の設定」を参照)、大量のノードがクラスターを連続して脱退しても、JBoss Data Grid がノードを再調整して損失データを回復できるように ***owners*** の適切な値を設定することです。

バグを報告する

40.5. ネットワークパーティションリカバリーの例

以下の例は、ネットワークパーティションが Red Hat JBoss Data Grid クラスターでどのように実行および処理され、結果的にマージされるかを示しています。次のシナリオ例は、詳細に説明されています。

1. ***owners*** が 3 に設定された分散 4 ノードのクラスター (「Owners が 3 の分散 4 ノードキャッシュの例」 を参照)
2. ***owners*** が 2 に設定された分散 4 ノードクラスター (「Owners が 2 の分散 4 ノードキャッシュの例」 を参照)
3. ***owners*** が 3 に設定された分散 5 ノードクラスター (「Owners が 3 の分散 5 ノードキャッシュの例」 を参照)
4. ***owners*** が 4 に設定されたレプリケート 4 ノードクラスター (「Owners が 4 のレプリケーション 4 ノードキャッシュの例」 を参照)
5. ***owners*** が 5 に設定されたレプリケート 5 ノードクラスター (「Owners が 5 のレプリケーション 5 ノードキャッシュの例」 を参照)

6. ***owners*** が 8 に設定されたレプリケート 8 ノードクラスター (「Owners が 8 のレプリケーション 8 ノードキャッシュの例」 を参照)

バグを報告する

40.5.1. Owners が 3 の分散 4 ノードキャッシュの例

最初のシナリオ例には、4 つのデータエントリー (**k1**、**k2**、**k3**、および **k4**) を含む 4 ノード分散キャッシュが含まれます。このキャッシュでは、***owners*** が 3 と等しくなります。つまり、各データエントリーごとに、キャッシュ内のさまざまなノードの 3 つのコピーが必要です。

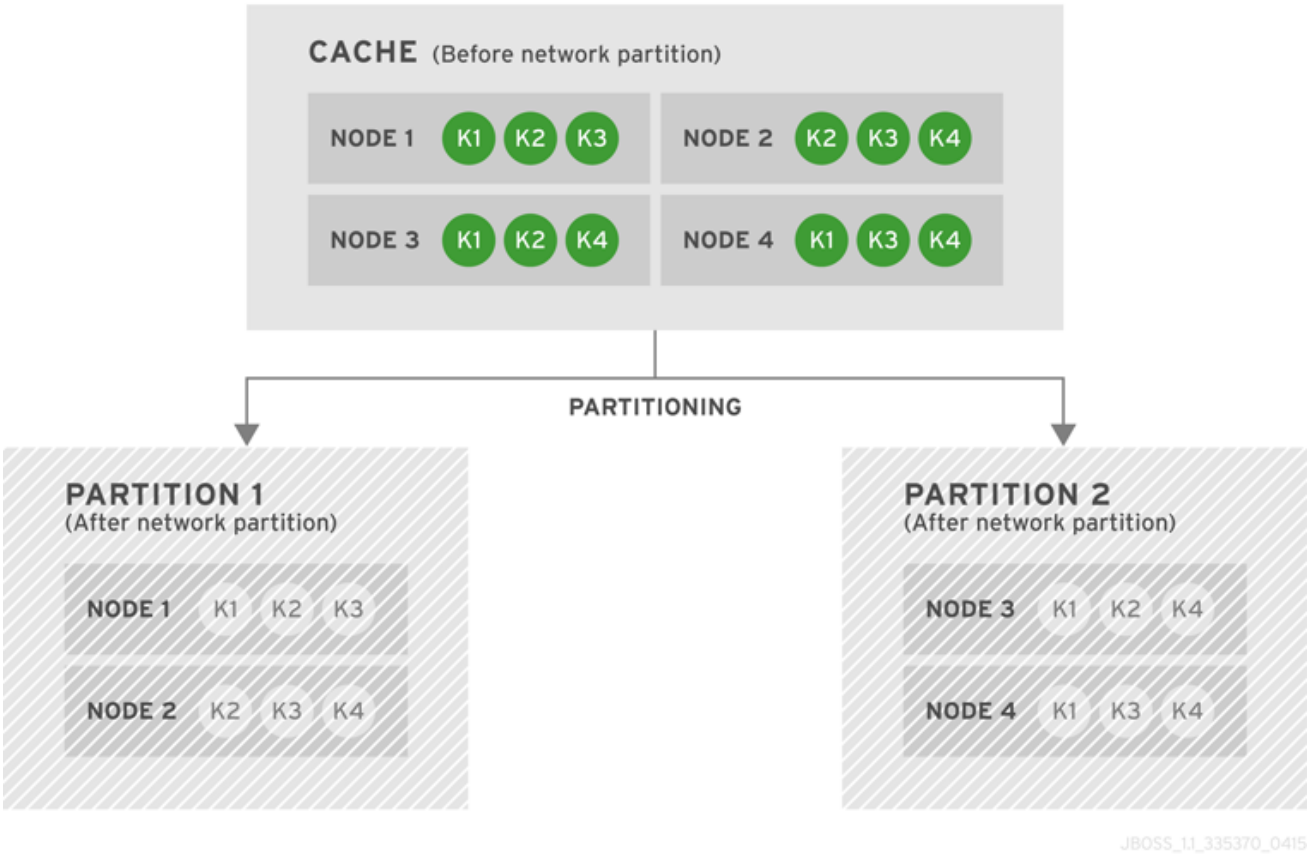


図40.1 ネットワークパーティション実行前とネットワークパーティション実行後のキャッシュ

図で示されたように、ネットワークパーティションの実行後は、ノード 1 とノード 2 がパーティション 1 を形成し、ノード 3 とノード 4 がパーティション 2 を形成します。分割後に、2 つのパーティションは劣化モードになります (図では網かけ表示されたノードにより表されます)。これは、どちらのパーティションでも、最後の安定したビューから 3 つ (***owners*** の値) 以上のノードが残っていないためです。結果として、4 つのエントリー (**k1**、**k2**、**k3**、および **k4**) は読み書きできません。新しいエントリーはいずれかの劣化パーティションで書き込みできます (どちらのパーティションでもエントリーの 3 つのコピーを格納できません)。

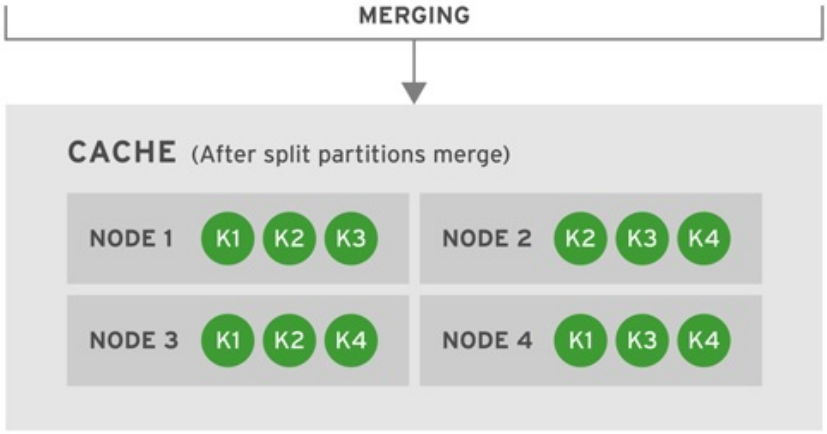


図40.2 パーティションのマージ後のキャッシュ

JBoss Data Grid は結果的に、分割された 2 つのパーティションをマージします。状態転送は不要であり、マージされた新しいキャッシュは利用可能モードになります (4 つのノードと 4 つのデータエントリー (**k1**、**k2**、**k3**、および **k4**) から構成されます)。

バグを報告する

40.5.2. Owners が 2 の分散 4 ノードキャッシュの例

2 つ目のシナリオ例には、4 つのノードから構成される分散キャッシュが含まれます。このシナリオでは、***owners*** が 2 となり、4 つのデータエントリー (**k1**、**k2**、**k3**、および **k4**) それぞれがキャッシュ内に 2 つのコピーを持ちます。

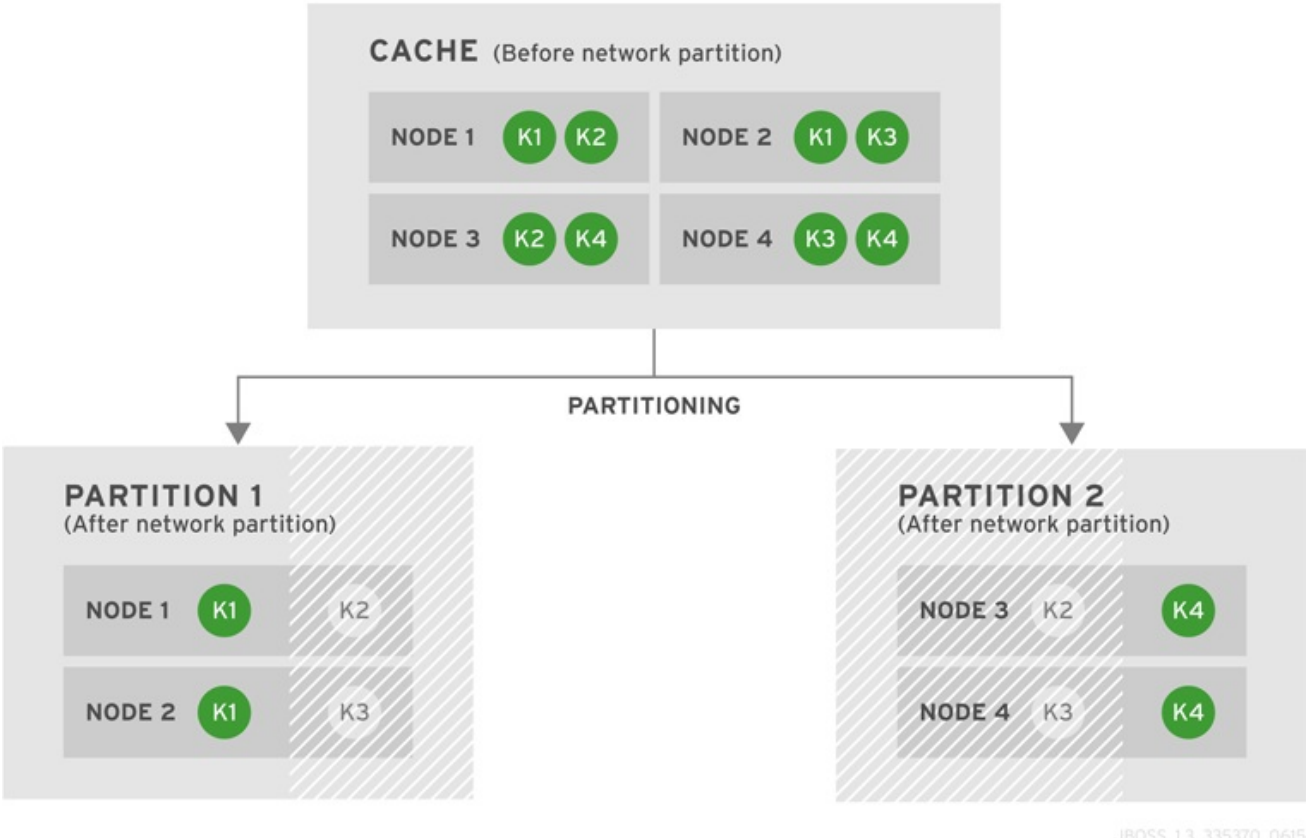


図40.3 ネットワークパーティション実行前とネットワークパーティション実行後のキャッシュ

ネットワークパーティションの実行時に、パーティション 1 および 2 は劣化モードになります (図では網かけ表示されたノードとして表されます)。各パーティション内では、エントリーは読み取るか、または書き込みことができます (両方のコピーが同じパーティション内にある場合)。パーティション 1 では、データエントリー **k1** は読み書きできます (***owners*** が 2 であり、エントリーの両方のコピーがパーティション 1 に保持されるため)。パーティション 2 では、同じ理由により **k4** は読み書きできます。エントリー **k2** および **k3** は両方のパーティションで利用できません (いずれのパーティションにもこれらのエントリーのすべてのコピーが含まれないため)。新しいエントリー **k5** は、**k5** の両方のコピーを所有するパーティションにのみ書き込むことができます。

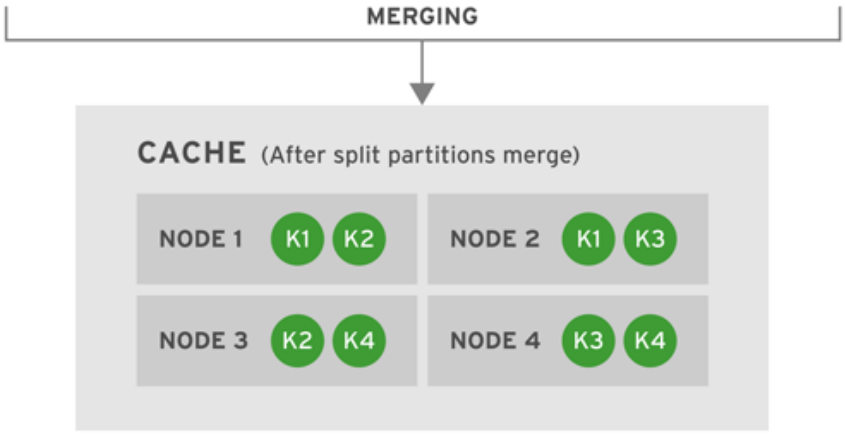


図40.4 パーティションのマージ後のキャッシュ

結果として JBoss Data Grid は分割された 2 つのパーティションを 1 つのキャッシュにマージします。状態転送は不要であり、キャッシュは利用可能モードに戻ります (4 つのノードと 4 つのデータエントリー (**k1**、**k2**、**k3**、および **k4**) から構成されます)。

バグを報告する

40.5.3. Owners が 3 の分散 5 ノードキャッシュの例

3 つ目のシナリオ例には、5 つのノードがあり、***owners*** が 3 に等しい分散キャッシュが含まれます。

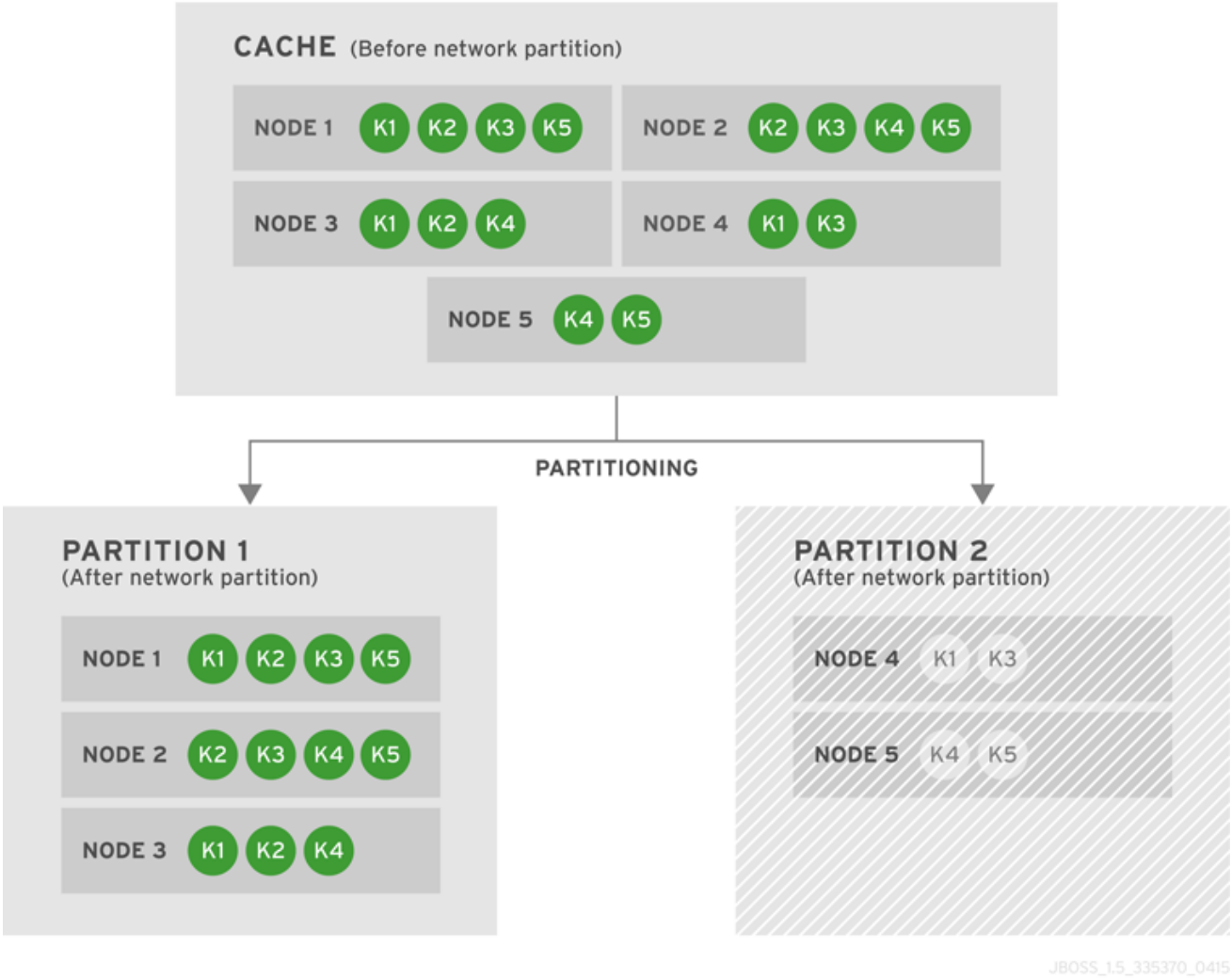


図40.5 ネットワークパーティション実行前とネットワークパーティション実行後のキャッシュ

ネットワークパーティションの実行後に、キャッシュは 2 つのパーティションに分割されます。パーティション 1 にはノード 1、ノード 2、およびノード Node 3 が含まれ、パーティション 2 にはノード 4 とノード 5 が含まれます。パーティション 2 は、キャッシュ内のノードの合計の過半数を含まないため劣化モードになります。パーティション 1 は、ノードの過半数を含み、 ***owners*** ノードよりも少ない数を失ったため、利用可能モードのままになります。

このパーティションは劣化モードであり、データのすべてのコピーを所有できないため、新しいエントリーをパーティション 2 に追加することはできません。

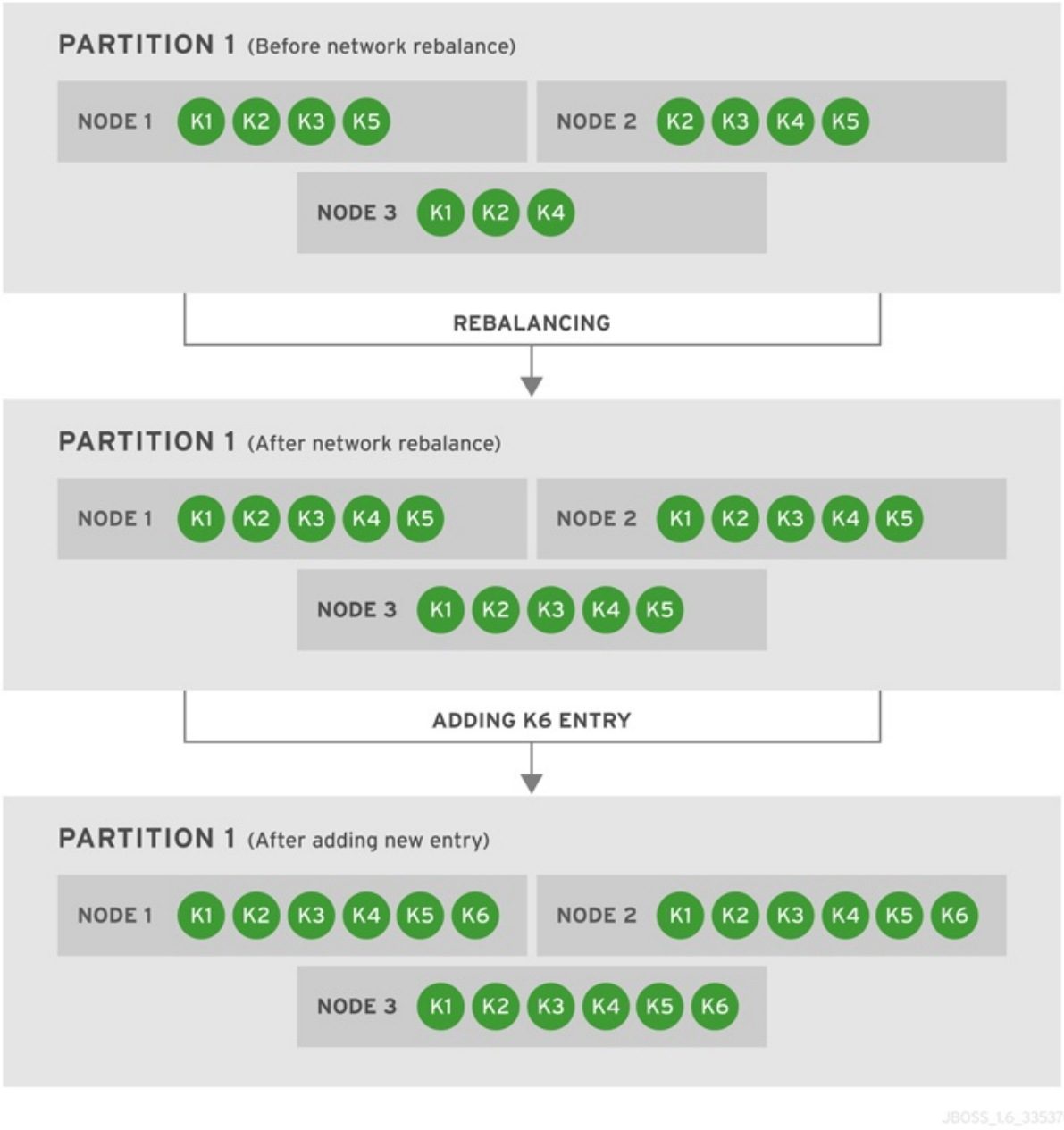


図40.6 パーティション 1 の再調整と別のエントリーの追加

パーティションの分割後に、パーティション 1 はノードの過半数を保持するため、不明なエントリーを置き換えるコピーを作成することによりパーティション自体を再調整できます。上記の図に示されたように、再調整を行うと、キャッシュ内の各エントリーの 3 つのコピーを確保できます (***numOwners*** は 3)。結果として、3 つのそれぞれのノードにはキャッシュ内の各エントリーのコピーが含まれます。次に、新しいエントリー (k6) をキャッシュに追加します。***owners*** 値は引き続き 3 であり、パーティション 1 には 3 つのノードがあるため、各ノードには **k6** のコピーが含まれます。

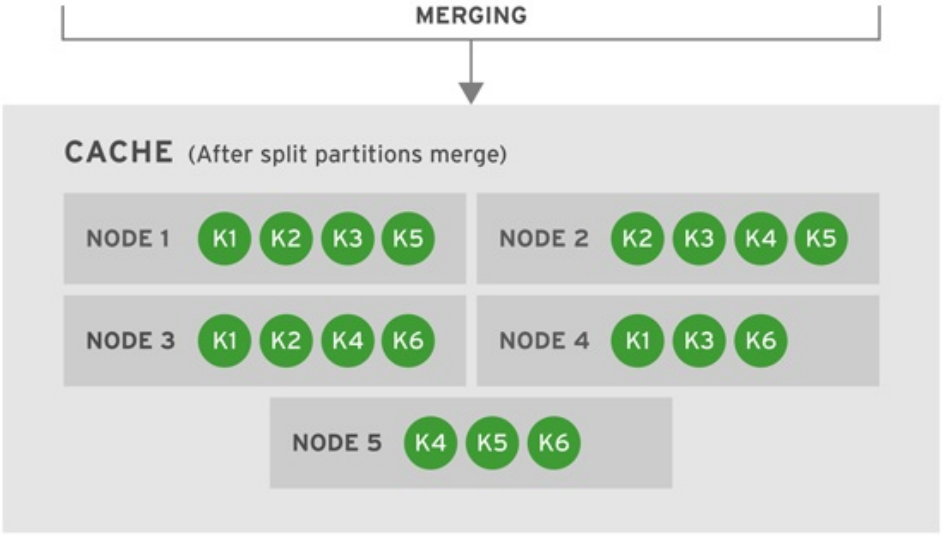


図40.7 パーティションのマージ後のキャッシュ

結果的に、パーティション 1 および 2 が 1 つのキャッシュにマージされます。各データエントリーには 3 つのコピーしか必要ないため (***owners***=3)、JBoss Data Grid はノードを再調整し、データエントリーがキャッシュ内の 4 つのノード間で分散されるようにします。マージされた新しいキャッシュは完全に利用可能になります。

バグを報告する

40.5.4. Owners が 4 のレプリケーション 4 ノードキャッシュの例

4 つ目のシナリオ例には、4 つのノードがあり、***owners*** が 4 に等しいレプリケートキャッシュが含まれます。

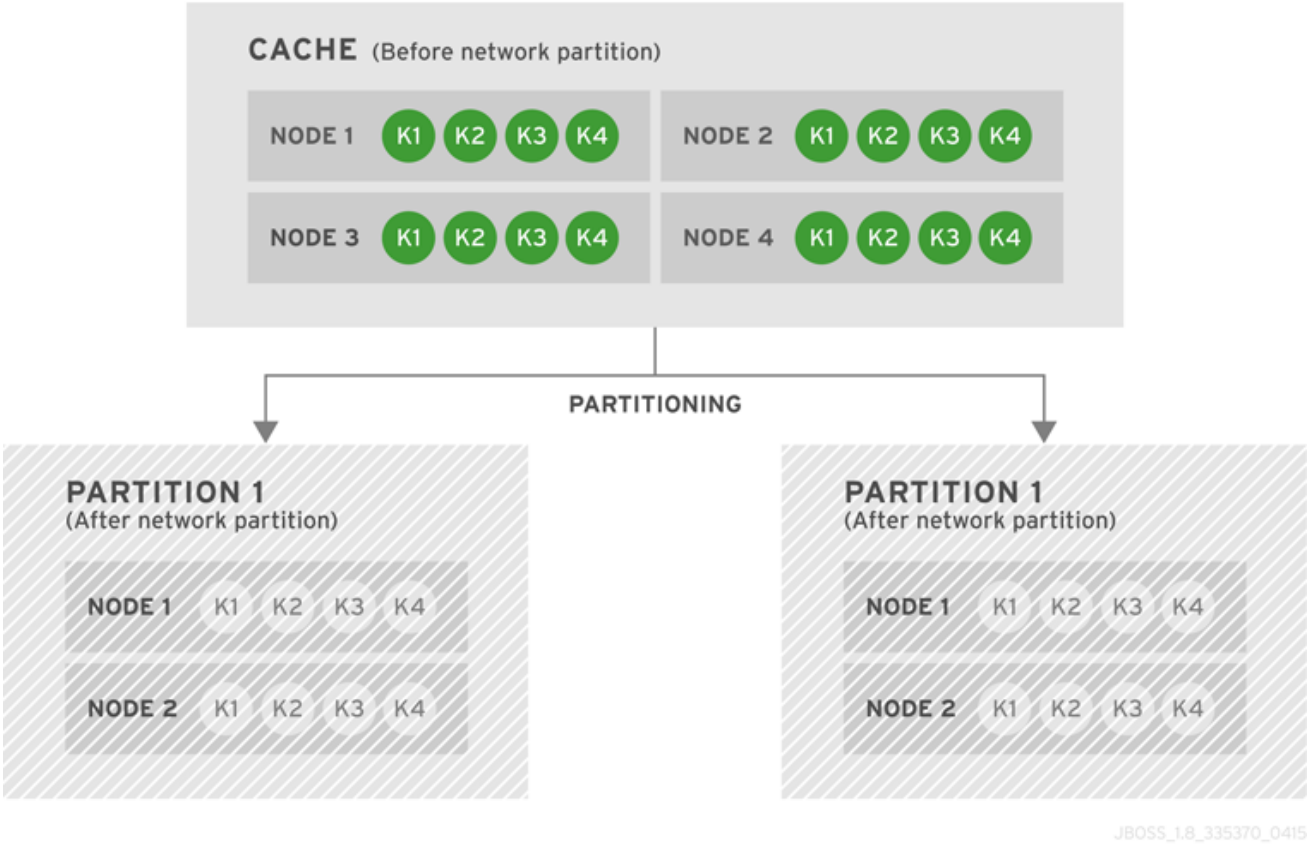


図40.8 ネットワークパーティション実行前とネットワークパーティション実行後のキャッシュ

ネットワークパーティションの実行後に、パーティション 1 にはノード 1 とノード 2 が含まれ、ノード 3 とノード 4 はパーティション 2 に含まれます。両方のパーティションは、過半数のノードを含まないため、劣化モードになります。すべての 4 つのキー **(k1**、**k2**、**k3**、および **k4)** は読み書きできません。これは、2 つのパーティションのどちらも 4 つのキーのすべてのコピーを所有しないためです。

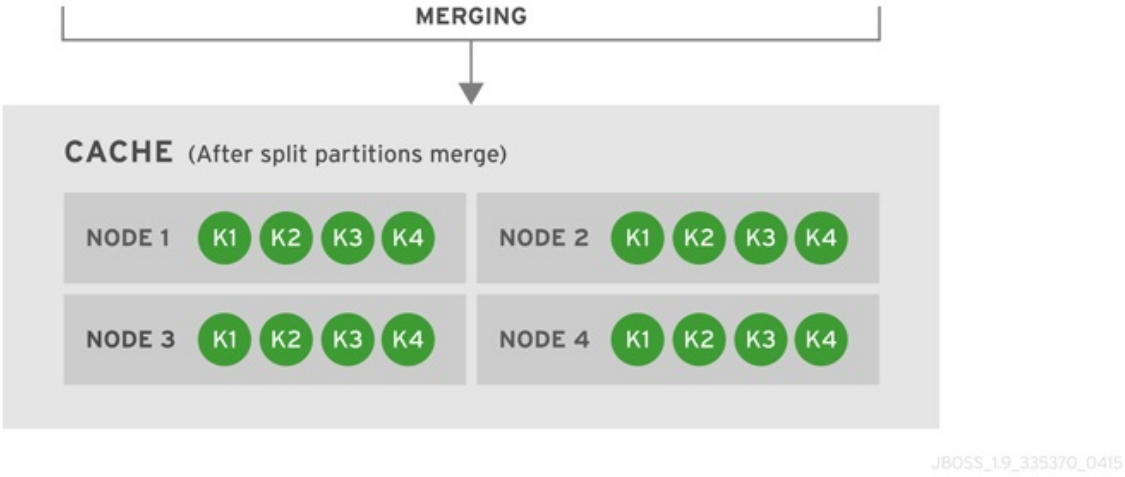


図40.9 パーティションのマージ後のキャッシュ

JBoss Data Grid は結果的に、分割された 2 つのパーティションを 1 つのキャッシュにマージします。状態転送は不要であり、キャッシュは利用可能モードで元の状態に戻ります **(4** つのノードと 4 つのデータエントリー **(k1**、**k2**、**k3**、および **k4)** から構成されます**)**。

バグを報告する

40.5.5. Owners が 5 のレプリケーション 5 ノードキャッシュの例

5 つ目のシナリオ例には、5 つのノードがあり、***owners*** が 5 に等しいレプリケーションキャッシュが含まれます。

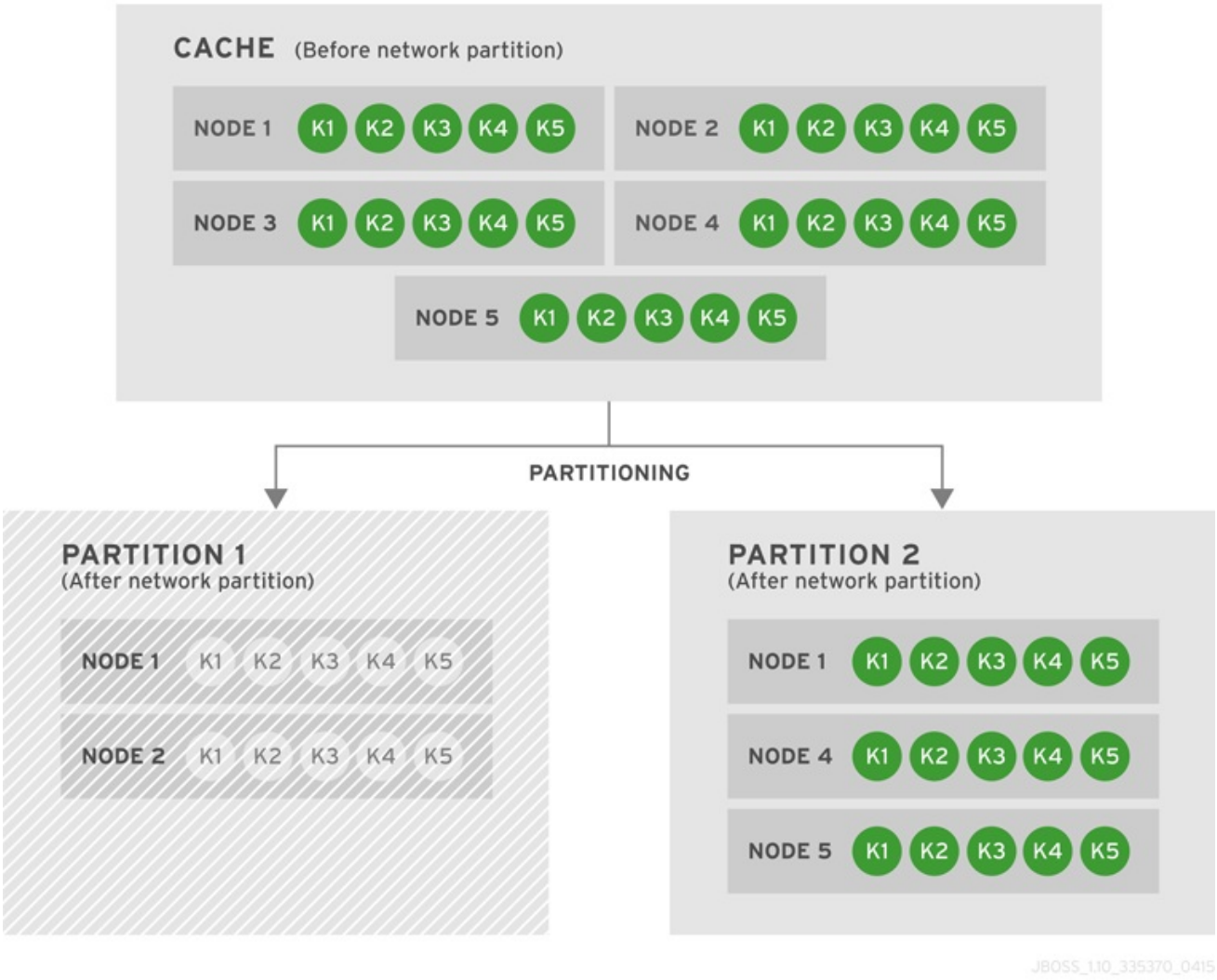


図40.10 ネットワークパーティション実行前とネットワークパーティション実行後のキャッシュ

ネットワークパーティションの実行後に、キャッシュは 2 つのパーティションに分割されます。パーティション 1 にはノード 1 とノード 2 が含まれ、パーティション 2 にはノード 3、ノード 4、およびノード 5 が含まれます。パーティション 1 は、ノードの過半数を含まないため劣化モードになります (網かけ表示されたノードで表されます)。ただし、パーティション 2 は利用可能なままになります。

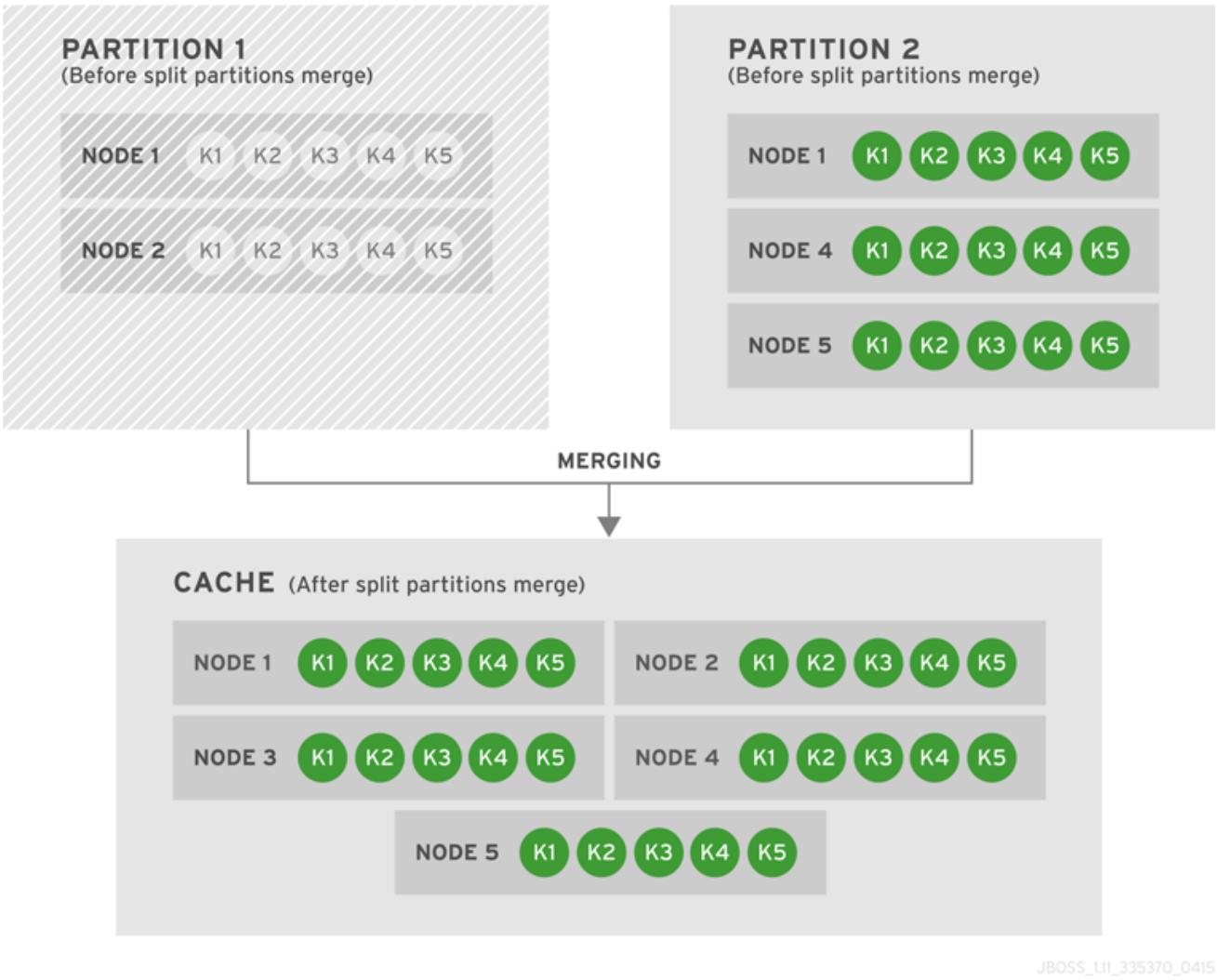


図40.11 両方のパーティションが 1 つのキャッシュにマージされる

この例では、JBoss Data Grid がパーティションをマージすると、パーティション 2 (完全に利用可能) がプライマリーパーティションと見なされます。状態はパーティション 1 とパーティション 2 から転送されます。マージされたキャッシュは完全に利用可能になります。

バグを報告する

40.5.6. Owners が 8 のレプリケーション 8 ノードキャッシュの例

6 つ目のシナリオは、8 つのノードがあり、***owners*** が 8 に等しいレプリケートキャッシュを対象としています。

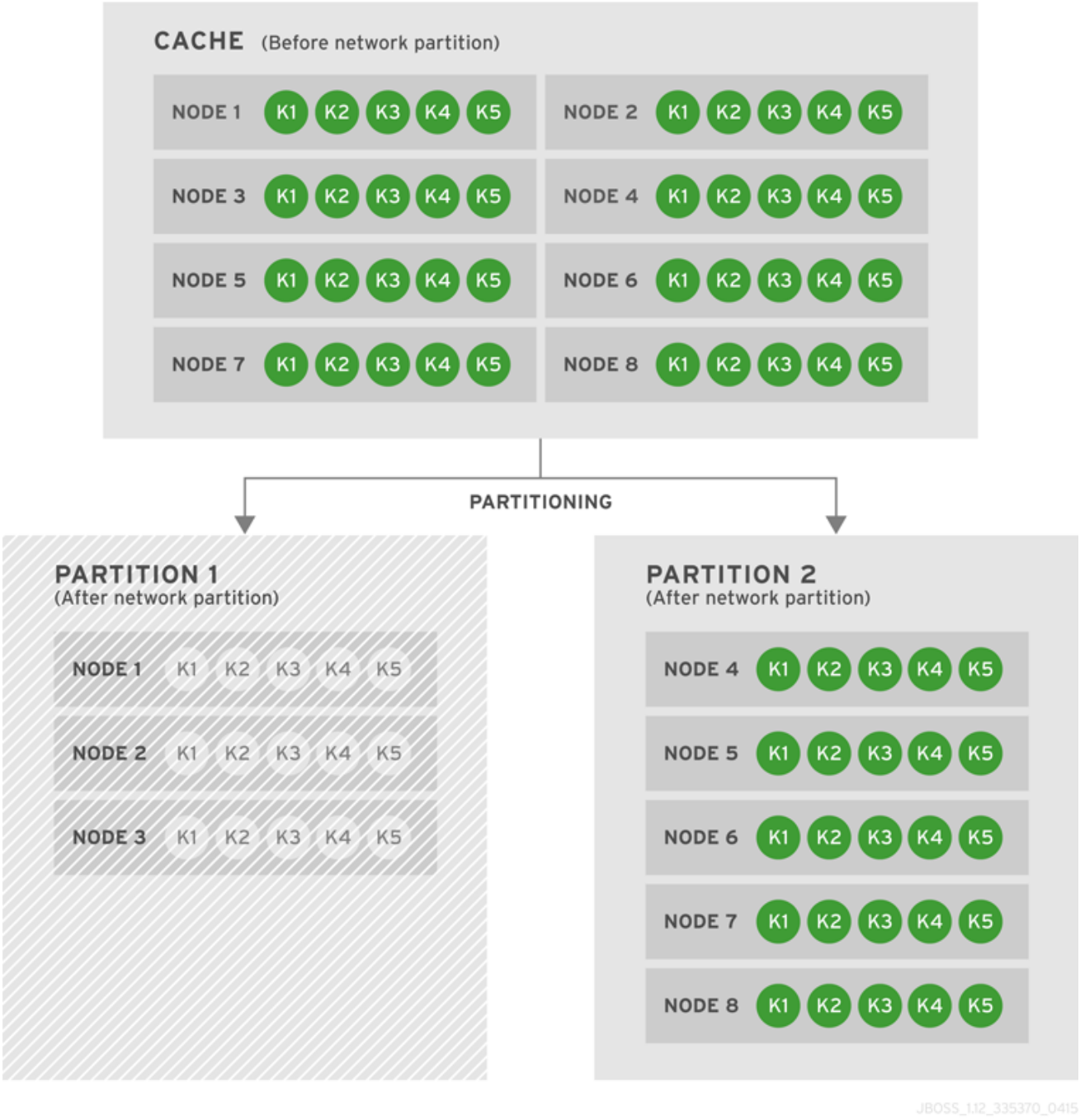


図40.12 ネットワークパーティション実行前とネットワークパーティション実行後のキャッシュ

ネットワークパーティションは、クラスターを 3 つのノードから構成されるパーティション 1 と 5 つのノードから構成されるパーティション 2 に分割します。パーティション 1 は劣化状態になりますが、パーティション 2 は利用可能なままです。

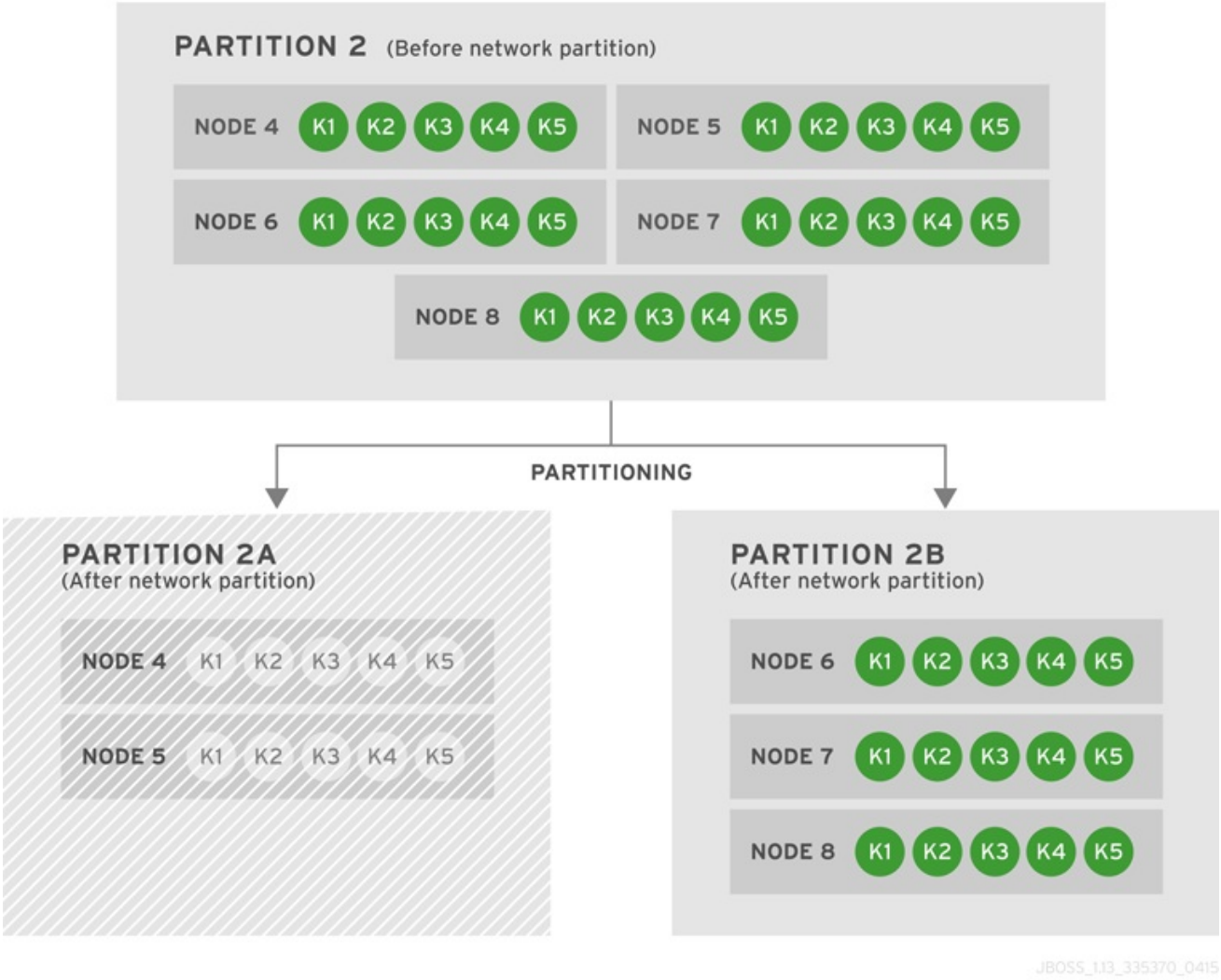


図40.13 パーティション 2 はさらにパーティション 2A と 2B に分割される

この時点で、別のネットワークパーティションはパーティション 2 に影響を与え、パーティション 2 はさらにパーティション 2A と 2B に分割されます。パーティション 2A にはノード 4 とノード 5 が含まれ、パーティション 2B にはノード 6、ノード 7、およびノード 8 が含まれます。パーティション 2A は、ノードの過半数を含まないため劣化モードになります。ただし、パーティション 2B は利用可能なままになります。

想定できる解決シナリオ

このシナリオでは 4 つのキャッシュの解決法があります。

- ケース 1: パーティション 2A と 2B のマージ
- ケース 2: パーティション 1 と 2A のマージ
- ケース 3: パーティション 1 と 2B のマージ
- ケース 4: パーティション 1、パーティション 2A、およびパーティション 2B のマージ

図40.14 ケース 1: パーティション 2A と 2B のマージ

パーティション分割されたネットワークの最初の解決法では、パーティション 2B の状態の情報をパーティション 2A にコピーします。結果としてパーティション 2 が作成され、ノード 5、 ノード 6、ノード 7、およびノード 8 が含まれます。新しくマージされたパーティションは利用可能になります。

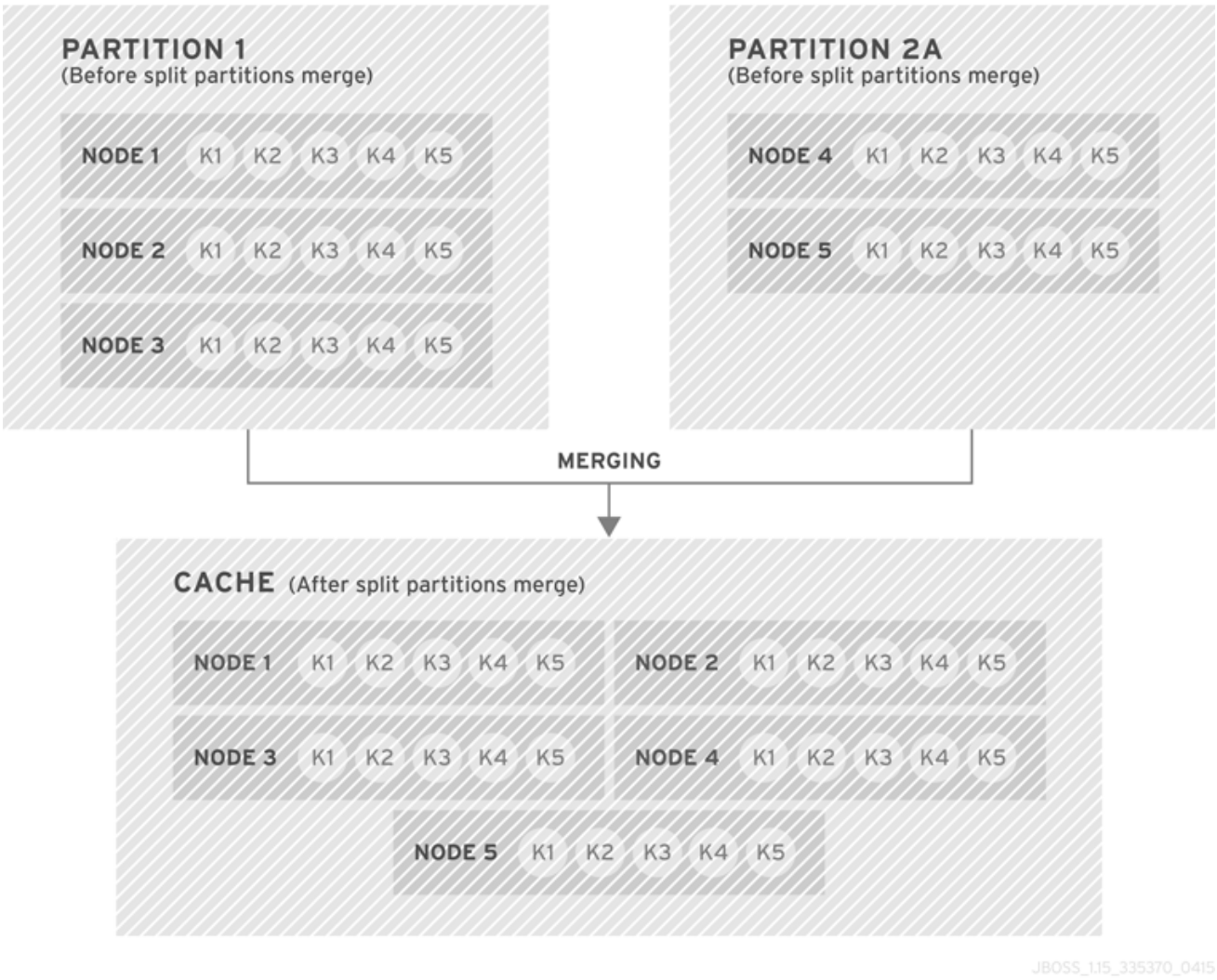


図40.15 ケース 2: パーティション 1 と 2A のマージ

パーティション分割されたネットワークの 2 つ目の解決法では、パーティション 1 とパーティション 2A をマージします。マージされたパーティションには、ノード 1、ノード 2、ノード 3、ノード 4、およびノード 5 が含まれます。どちらのパーティションも最新の安定したトポロジーを含まないため、結果としてマージされたパーティションは劣化モードのままになります。

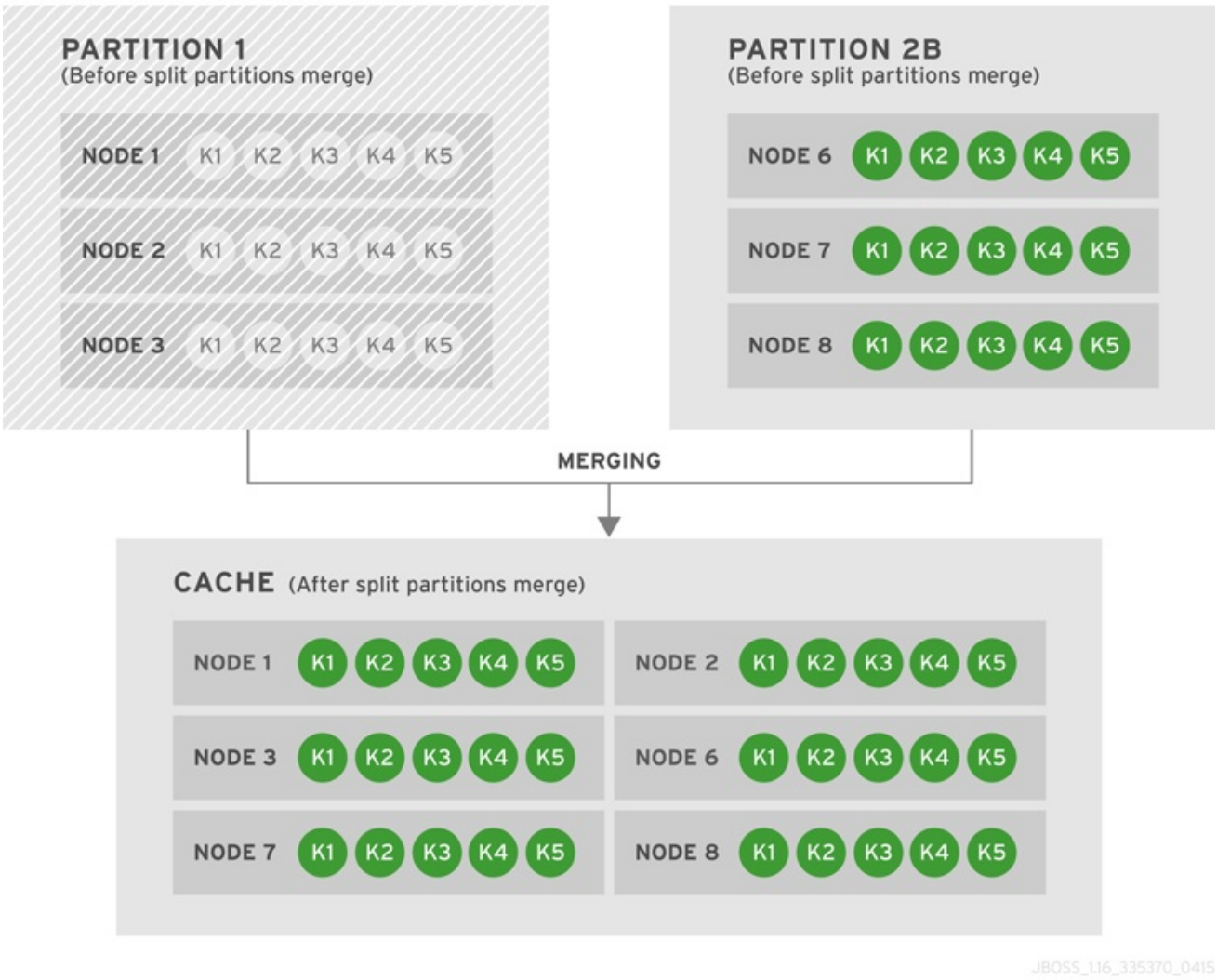


図40.16 ケース 3: パーティション 1 と 2B のマージ

パーティション分割されたネットワークの 3 つ目の解決法では、パーティション 1 とパーティション 2B をマージします。パーティション 1 は、パーティション 2B から状態情報を受け取り、マージされたパーティションは利用可能になります。

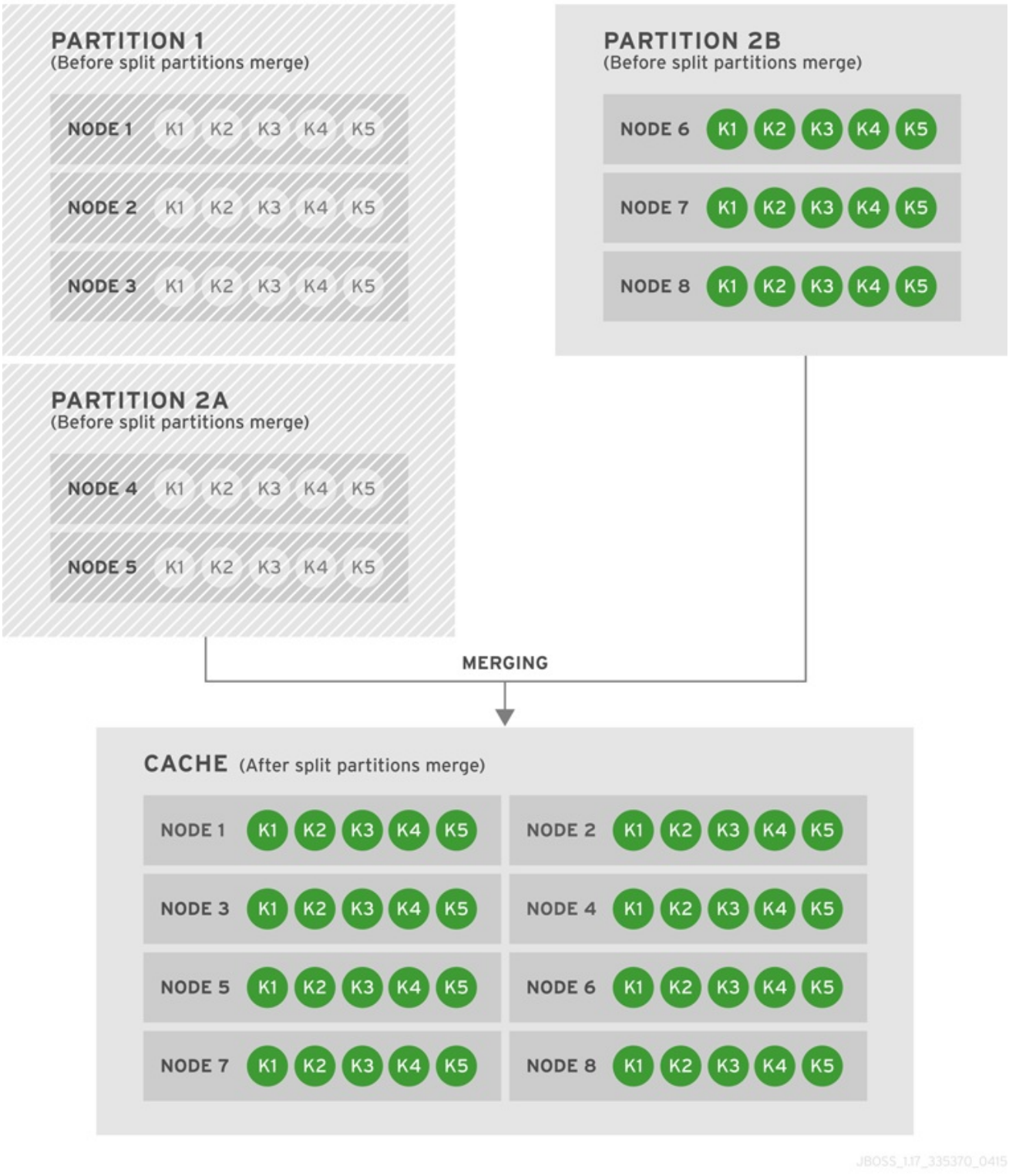


図40.17 ケース 4: パーティション 1、パーティション 2A、およびパーティション 2B のマージ

パーティション分割されたネットワークの 4 つ目の最後の解決法では、パーティション 1、パーティション 2A、およびパーティション 2B をマージしてパーティション 1 を形成します。状態はパーティション 2B からパーティション 1 および 2A の両方のパーティションに転送されます。結果的に、キャッシュには 8 つのノード (ノード 1、ノード 2、ノード 3、ノード 4、ノード 5、ノード 6、ノード 7、およびノード 8) が含まれ、キャッシュは利用可能になります。

バグを報告する

40.6. パーティション処理の設定

Red Hat JBoss Data Grid では、パーティション処理はデフォルトで無効になります。

宣言による設定 (ライブラリーモード)

パーティション処理を以下のように宣言します。

```
<distributed-cache name="distributed_cache"
        owners="2"
        l1-lifespan="20000">
    <partition-handling enabled="true"/>
</distributed-cache>
```

宣言による設定 (リモートクライアントサーバーモード)

以下の設定を使用して、パーティション処理をリモートクライアントサーバーモードで宣言的に有効にします。

```
<subsystem xmlns="urn:infinispan:server:core:8.3" default-cache-
container="clustered">
    <cache-container name="clustered" default-cache="default"
statistics="true">
        <distributed-cache name="default" mode="SYNC" segments="20"
owners="2"
                           remote-timeout="30000" start="EAGER">
            <partition-handling enabled="true" />
            <locking isolation="READ_COMMITTED" acquire-timeout="30000"
                     concurrency-level="1000" striping="false"/>
            <transaction mode="NONE"/>
        </distributed-cache>
    </cache-container>
</subsystem>
```

バグを報告する

付録A JBOSS DATA GRID 向けの推奨 JGROUPS 値

A.1. サポート対象 JGROUPS プロトコル

以下の表には、JBoss Data Grid でサポートされる JGroups プロトコルのリストが含まれます。

表A.1 サポート対象 JGroups プロトコル

プロトコル	詳細
TCP	TCP/IP は、IP マルチキャストが使用できなくなる状況で使用できる UDP の代替トランスポートです。このような状況には、ルーターが IP マルチキャストパケットを破棄する可能性のある WAN 上の操作が実行される場合などが含まれます。 TCP は、ユニキャストおよびマルチキャストメッセージを送信するために使用されるトランスポートプロトコルです。 • マルチキャストメッセージを送信する場合に、TCP は複数のユニキャストメッセージを送信します。 • TCP を使用する場合に、すべてのクラスターメンバーに対するそれぞれのメッセージが複数のユニキャストメッセージとして送信されるか、または各メンバーに対して 1 つのメッセージが送信されます。 IP マルチキャストは初期メンバーを検出するために使用することができないため、初期メンバーを見つけるには別のメカニズムを使用する必要があります。 Red Hat JBoss Data Grid の Hot Rod はカスタム TCP クライアント/サーバープロトコルです。
UDP	UDP は、以下を使用するトランスポートプロトコルです。 • クラスターのすべてのメンバーにメッセージを送信する IP マルチキャスト。 • 単一メンバーに送信されるユニキャストメッセージの UDP データグラム。 UDP トランスポートが開始すると、トランスポートはユニキャストソケットとマルチキャストソケットを開きます。ユニキャストソケットは、ユニキャストメッセージの送受信に使用され、マルチキャストソケットは、マルチキャストソケットの送受信を行います。チャネルの物理アドレスは、ユニキャストソケットのアドレスおよびポート番号と同じです。

プロトコル	詳細
PING	PING プロトコルは、メンバーの内部検出に使用されます。PING 要求を IP マルチキャストアドレスにマルチキャストすることにより、最も古いメンバーであるコーディネーターを検出するために使用されます。 各メンバーはコーディネーターのアドレスと自身のアドレスを含むパケットで ping に応答します。指定されたミリ秒数 (N) または応答数 (M) のあとに、参加者が応答からコーディネーターを決定し、コーディネーターに JOIN 要求 (GMS により処理されます) を送信します。応答がない場合、参加者はグループの最初のメンバーと見なされます。 PING は、動的検出を使用するため、TCPPING とは異なります。つまり、メンバーは他のクラスターメンバーの場所を事前に知る必要がありません。PING はトランスポートの IP マルチキャスト機能を使用して検出要求をクラスターに送信します。結果として PING はトランスポートとして UDP を必要とします。
TCPPING	TCCPING プロトコルは、既知の一連のメンバーを使用して、検出のために ping を送信します。このプロトコルは静的設定です。
MPING	MPING (マルチキャスト PING) プロトコルは IP マルチキャストを使用して初期メンバーシップを検出します。すべてのトランスポートで使用できますが、通常は TCP と共に使用されます。
S3_PING	S3_PING は、Amazon の Elastic Compute Cloud (EC2) で使用するのに理想的なディスカバリープロトコルです。これは、EC2 ではマルチキャストが許可されず、MPING も許可されないためです。 それぞれの EC2 インスタンスは小さいファイルをバケットとして知られる S3 データコンテナーに追加します。その後、各インスタンスはバケット内のファイルを読み込み、クラスターの他のメンバーを検出します。
JDBC_PING	JDBC_PING はクラスター内のノードに関する情報を保存するために共有データベースを使用するディスカバリープロトコルです。
TCPGOSSIP	TCPGOSSIP は 1 つ以上の設定された GossipRouter プロセスを使用してクラスター内のノードについての情報を保存するディスカバリープロトコルです。

プロトコル	詳細
MERGE3	MERGE3 プロトコルは JGroups 3.1 以降で利用可能です。MERGE2 とは異なり、MERGE3 では、すべてのメンバーがアドレス (UUID)、論理名、物理アドレス、およびビュー ID を使用して周期的に INFO メッセージを送信します。周期的に、各コーディネーターは不整合が発生しないように INFO 詳細情報を確認します。
FD_ALL	障害検出に使用される FD_ALL は単純なハートビートプロトコルを使用します。各メンバーは他のすべてのメンバー (メンバー自身を除く) のテーブルを維持し定期的にハートビートをマルチキャストします。たとえば、P からデータまたはハートビートを受け取ると、P のタイムスタンプが現在の時刻に設定されます。周期的に、失効したメンバーはタイムスタンプ値を使用して識別されます。
FD_SOCK	FD_SOCK は、クラスターメンバー間で作成された TCP ソケットのリングに基づいた障害検出プロトコルです。各クラスターメンバーは近接メンバーに接続し (最後のメンバーは最初のメンバーに接続します)、リングが形成されます。メンバー B は、近接メンバー A が TCP ソケットの異常な終了 (通常はノード B のクラッシュのため) を検出したときに疑われます。ただし、メンバー B が正常に脱退した場合、メンバー B はメンバー A に通知し、脱退しても疑われません。
FD_HOST	FD_HOST は、すべてのホストのクラッシュまたはハングを検出し、ICMP ping メッセージまたはカスタムコマンドを介して該当するホストのすべてのクラスターメンバーを疑う障害検出プロトコルです。FD_HOST は、ローカルホストの 1 つのメンバーのクラッシュまたはハングを検出しませんが、クラスター内の他のすべてのホストがライブ状態であり利用可能であるかどうかのみチェックします。したがって、FD_HOST は、FD_ALL や FD_SOCK などの他の障害検出プロトコルと共に使用されます。このプロトコルは通常、複数のクラスターメンバーが同じ物理的なボックス上で実行されている場合に使用されます。 FD_HOST プロトコルは、JBoss Data Grid 向けの Windows でサポートされます。**cmd** パラメーターを **ping.exe** に設定し、ping 数を指定する必要があります。
VERIFY_SUSPECT	VERIFY_SUSPECT プロトコルは、メンバーを除外する前にメンバーに ping を送信することにより、疑われたメンバーがダウンしているかどうかを確認します。メンバーが応答した場合は、疑いに関するメッセージが破棄されます。

プロトコル	詳細
NAKACK2	NAKACK2 プロトコルは、NAKACK プロトコルの後継プロトコルであり、JGroups 3.1 で導入されました。 NACKACK2 プロトコルはマルチキャストメッセージに使用され、NAK を使用します。各メッセージは、シーケンス番号でタグ付けされます。受信側はシーケンス番号を追跡し、メッセージを順番に配信します。シーケンス番号のギャップが検出されると、受信側は不明なメッセージを再送信するよう送信側に要求します。
UNICAST3	UNICAST3 プロトコルは、安定した配信を提供し (送信側が送信したメッセージは番号付けされたシーケンスで送信されるため、失われません)、送信側と受信側間のポイントツーポイントメッセージに FIFO (First In First Out) プロパティーを使用します。 UNICAST3 は、再送信にポジティブ ack を使用します。たとえば、送信側 A は、受信側 B がメッセージ M を受診するまでメッセージ M を送信し続け、正常な送信を示すために ack を返します。送信側 A は、B から ack を受信するまで、B がクラスターを脱退するまで、または A がクラッシュするまでメッセージ M を再送信し続けます。
STABLE	STABLE プロトコルは、クラスター内のすべてのメンバーによって参照されたメッセージのガーベッジコレクターです。再送信が必要なことがあるため、各メンバーはすべてのメッセージを格納します。メッセージは、すべてのメンバーがメッセージを参照したときにのみ再送信バッファーから削除できます。STABLE プロトコルは、参照された最大値のメッセージと最小値のメッセージを定期的に拡散します。最小値は、min (すべてのメンバーに対して最小のシーケンス番号すべて) を計算するために使用され、min 値よりも小さいシーケンス番号のメッセージは破棄できます。
GMS	GMS プロトコルは、グループメンバーシッププロトコルです。このプロトコルは、参加/脱退/クラッシュ (疑い) を処理し、新しいビューを適切に生成します。
MFC	MFC は、フロー制御プロトコルのマルチキャストバージョンです。
UFC	UFC は、フロー制御プロトコルのユニキャストバージョンです。

プロトコル	詳細
FRAG2	FRAG2 プロトコルは、大きいメッセージを小さいメッセージに断片化し、小さいメッセージを送信します。受信側では、小さい断片が、大きく完全なメッセージに再び組み立てられ、アプリケーションに配信されます。FRAG2 はマルチキャストメッセージとユニキャストメッセージの両方に使用されます。
SYM_ENCRYPT	JGroups には、クラスターのトラフィック向けの暗号化を提供する SYM_ENCRYPT プロトコルが含まれます。デフォルトでは、暗号化によりメッセージ本文のみが暗号化され、メッセージヘッダーは暗号化されません。すべてのヘッダーを含むメッセージ全体と宛先アドレスおよびソースアドレスを暗号化するには、プロパティー **`encrypt_entire_message`** が true である必要があります。このプロトコルを定義する際、これは **NAKACK2** の下に直接配置する必要があります。 SYM_ENCRYPT レイヤーは、キーストアでシークレットを定義することで JGroups で通信を暗号化および復号化するために使用されます。 各メッセージは、暗号化ヘッダーを示す特定の暗号化ヘッダーとメッセージを暗号化および復号化するために使用するキーのバージョンを示す MD5 ダイジェストで暗号化済みとして識別されます。
ASYM_ENCRYPT	JGroups には、クラスターのトラフィック向けの暗号化を提供する ASYM_ENCRYPT プロトコルが含まれます。デフォルトでは、暗号化によりメッセージ本文のみが暗号化され、メッセージヘッダーは暗号化されません。すべてのヘッダーを含むメッセージ全体と宛先アドレスおよびソースアドレスを暗号化するには、プロパティー **`encrypt_entire_message`** が true である必要があります。このプロトコルを定義する際、これは **NAKACK2** の下に直接配置する必要があります。 ASYM_ENCRYPT レイヤーは、定義済みのアルゴリズムとキーサイズを使用してコーディネーターにシークレットキーを生成させることにより、JGroups の通信の暗号化および復号化するために使用されます。 各メッセージは、暗号化ヘッダーを示す特定の暗号化ヘッダーとメッセージを暗号化および復号化するために使用するキーのバージョンを示す MD5 ダイジェストで暗号化済みとして識別されます。

プロトコル	詳細
SASL	SASL (Simple Authentication and Security Layer) プロトコルは、置き換え可能なメカニズムを使用した接続指向プロトコルで認証およびデータセキュリティーサービスを提供するフレームワークです。また、SASL はプロトコルとメカニズム間で構造化インターフェースを提供します。
RELAY2	RELAY プロトコルは、各サイトの 1 つのノード間の接続を作成することによって 2 つのリモートクラスターをブリッジします。これにより、1 つのサイトに送信されたマルチキャストメッセージを他のサイトにリレーし、他のサイトからそのサイトにもリレーすることができます。 JGroups には、Red Hat JBoss Data Grid のサイト間レプリケーションにおけるサイト間の通信に使用される RELAY2 プロトコルが含まれます。 RELAY2 プロトコルは RELAY のように動作しますが、若干の違いがあります。RELAY とは異なり、RELAY2 プロトコルは以下のことを行います。 • 3 つ以上のサイトを接続します。 • 自律的に機能し、相互に認識しないサイトに接続します。 • サイト間でユニキャストルーティングとマルチキャストルーティングの両方を提供します。

バグを報告する

A.2. TCP のデフォルト値と推奨値

JGroups と TCP および UDP の使用の詳細については、「JGroups の設定 (ライブラリーモード)」を参照してください。

注記

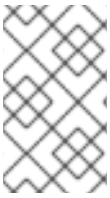
- **`JGroups Default Value`** の値は、JGroups に対して内部的に設定される値を示しますが、カスタム設定ファイルや JBoss Data Grid に同梱される JGroups 設定ファイルで上書きできます。
- **`JBoss Data Grid Configured Values`** の値は、JBoss Data Grid に同梱される JGroups の設定ファイルのいずれかを使用する場合にデフォルトで使用される値を示します。JGroups のカスタム設定ファイルが JBoss Data Grid と併用される場合にこれらの値を使用することが推奨されます。

JBoss Data Grid に組み込まれる設定ファイルの詳細情報は、「JBoss Data Grid JGroups 設定ファイル」を参照してください。

表A.2 TCP の推奨値とデフォルト値

パラメーター	JGroups のデフォルト値	JBoss Data Grid の設定値
bind_addr	任意の非ループバック	特定のインターフェースにアドレスを設定
bind_port	任意の空きポート	特定のポートを設定
loopback	true	デフォルト値と同じ
port_range	50	必要なポート範囲に 基づいて設定
recv_buf_size	150,000	デフォルト値と同じ
send_buf_size	150,000	640,000
use_send_queues	true	デフォルト値と同じ
sock_conn_timeout	2,000	300
max_bundle_size	64,000	64,000
enable_diagnostics	true	false
thread_pool.enabled	true	デフォルト値と同じ
thread_pool.min_threads	2	これは、ノードの数と同じである必要があります。
thread_pool.max_threads	30	これは、***thread_pool.min_threads*** よりも大きい必要があります。たとえば、小さいグリッド (2〜10 ノード) の場合は、この値をノードの数の 2 倍に設定しますが、大きいグリッド (20 以上のノード) は、比率を小さくする必要があります。たとえば、グリッドに 20 ノードが含まれる場合はこの値を 25 に設定し、グリッドに 100 ノードが含まれる場合はこの値を 110 に設定します。
thread_pool.keep_alive_time	30,000	60,000
thread_pool.queue_enabled	true	false
thread_pool.queue_max_size	500	なし。キューを無効にする必要があります。

パラメーター	JGroups のデフォルト値	JBoss Data Grid の設定値
thread_pool.rejection_policy	Discard	デフォルト値と同じ
internal_thread_pool.enabled	true	デフォルト値と同じ
internal_thread_pool.min_threads	2	5
internal_thread_pool.max_threads	4	20
internal_thread_pool.keep_alive_time	30,000	60,000
internal_thread_pool.queue_enabled	true	false
internal_thread_pool.rejection_policy	Discard	停止
oob_thread_pool.enabled	true	デフォルト値と同じ
oob_thread_pool.min_threads	2	20 以上
oob_thread_pool.max_threads	10	200 以上 (負荷に基づく)
oob_thread_pool.keep_alive_time	30,000	60,000
oob_thread_pool.queue_enabled	true	false
oob_thread_pool.queue_max_size	500	なし。キューを無効にする必要があります。
oob_thread_pool.rejection_policy	Discard	デフォルト値と同じ



注記

Red Hat JBoss Data Grid 7.0 は JGroups 3.6.9 を使用します。JGroups では、TCPPING タイムアウト値が削除され、***pbcast.GMS join_timeout*** 値が代わりにタイムアウト期間を示します。

S3_PING の推奨値

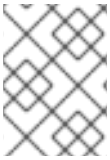
JBoss Data Grid 向けの S3_PING の設定の詳細については、「S3_PING 設定オプション」を参照してください。

TCPGOSSIP の推奨値

JBoss Data Grid 向けの TCPGOSSIP の設定の詳細については、「TCPGOSSIP 設定オプション」 を参照してください。

表A.3 MPING の推奨値

パラメーター	JGroups のデフォルト値	JBoss Data Grid の設定値
bind_addr	任意の非ループバック	特定のインターフェースにアドレスを設定
break_on_coord_rsp	true	デフォルト値と同じ
mcast_addr	230.5.6.7	デフォルト値と同じ
mcast_port	7555	デフォルト値と同じ
ip_ttl	8	2



注記

JGroups 3.6.1 では、MPING タイムアウト値が削除され、***pbcast.GMS join_timeout*** 値が代わりにタイムアウト期間を示します。

表A.4 MERGE3 の推奨値

パラメーター	JGroups のデフォルト値	JBoss Data Grid の設定値
min_interval	1,000	10,000
max_interval	10,000	30,000

表A.5 FD_SOCK の推奨値

パラメーター	JGroups のデフォルト値	JBoss Data Grid の設定値
client_bind_por	0 (ポートがランダムに選択され、使用されます)	デフォルト値と同じ
get_cache_timeout	1000 ミリ秒	デフォルト値と同じ
keep_alive	true	デフォルト値と同じ
num_tries	3	デフォルト値と同じ
start_port	0 (ポートがランダムに選択され、使用されます)	デフォルト値と同じ
suspect_msg_interval	5000 ミリ秒	デフォルト値と同じ

表A.6 FD_ALL の推奨値

パラメーター	JGroups のデフォルト値	JBoss Data Grid の設定値
timeout	40,000	60,000。FD_ALL タイムアウト値は、CMS ガーベッジコレクターでの stop the world ガーベッジコレクション一時停止の最大時間の 2 倍に設定されます。適切にチューニングされた JVM では、一時停止の最大時間はヒープサイズに応じて決まり、ヒープの 1 GB あたり 1 秒を超えないようにする必要があります。たとえば、8GB のヒープでは、一時停止時間が 8 秒を超えないようにし、FD_ALL タイムアウト値を 16 秒に設定する必要があります。長いガーベッジコレクション一時停止が使用された場合は、ノードで false 障害検出を回避するためにこのタイムアウト値を増やす必要があります。
間隔	8,000	15,000。FD_ALL ***interval*** 値は、FD_ALL の ***timeout*** 値に設定された値よりも 4 分 1 以下である必要があります。
timeout_check_interval	2,000	5,000

表A.7 FD_HOST の推奨値

パラメーター	JGroups のデフォルト値	JBoss Data Grid の設定値
check_timeout	3,000	5,000
cmd	InetAddress.isReachable() (ICMP ping)	-
間隔	20,000	15,000。FD_HOST の ***interval*** 値は、FD_HOST の ***timeout*** 値の 4 分 1 である必要があります。
timeout	60,000	60,000

表A.8 VERIFY_SUSPECT の推奨値

パラメーター	JGroups のデフォルト値	JBoss Data Grid の設定値
timeout	2,000	5,000

表A.9 pbcast.NAKACK2 の推奨値

パラメーター	JGroups のデフォルト値	JBoss Data Grid の設定値
use_mcast_xmit	true	false
xmit_interval	1,000	デフォルト値と同じ
xmit_table_num_rows	50	50
xmit_table_msgs_per_row	10,000	1,024
xmit_table_max_compaction_time	10,000	30,000
max_msg_batch_size	100	デフォルト値と同じ
resend_last_seqno	false	true

表A.10 UNICAST3 の推奨値

パラメーター	JGroups のデフォルト値	JBoss Data Grid の設定値
xmit_interval	500	デフォルト値と同じ
xmit_table_num_rows	100	50
xmit_table_msgs_per_row	1,0000	1,024
xmit_table_max_compaction_time	600,000	30,000
max_msg_batch_size	500	100
conn_close_timeout	60,000	推奨値なし。
conn_expiry_timeout	120,000	0

表A.11 pbcast.STABLE の推奨値

パラメーター	JGroups のデフォルト値	JBoss Data Grid の設定値
stability_delay	6,000	500
desired_avg_gossip	20,000	5,000
max_bytes	2,000,000	1,000,000

表A.12 pbcast.GMS の推奨値

パラメーター	JGroups のデフォルト値	JBoss Data Grid の設定値
print_local_addr	true	false
join_timeout	5,000	15,000
view_bundling	true	デフォルト値と同じ

表A.13 MFC の推奨値

パラメーター	JGroups のデフォルト値	JBoss Data Grid の設定値
max_credits	500,000	2,000,000
min_threshold	0.40	デフォルト値と同じ

表A.14 FRAG2 の推奨値

パラメーター	JGroups のデフォルト値	JBoss Data Grid の設定値
frag_size	60,000	デフォルト値と同じ

表A.15 SYM_ENCRYPT の推奨値

パラメーター	JGroups のデフォルト値	JBoss Data Grid の設定値
sym_algorithm	AES	-
sym_keylength	128	-
sym_provider	Bouncy Castle Provider	-

表A.16 ASYM_ENCRYPT の推奨値

パラメーター	JGroups のデフォルト値	JBoss Data Grid の設定値
asym_algorithm	RSA	-
asym_keylength	512	-
asym_provider	Bouncy Castle Provider	-
change_keys_on_leave	false	-

SASL の推奨値

詳細については、『Red Hat JBoss Data Grid Developer Guide 』の「User Authentication over Hot Rod Using SASL」セクションを参照してください。

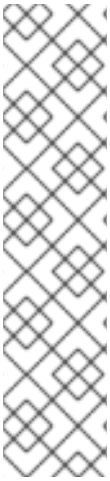
RELAY2 の推奨値

詳細については、35章*データセンター間のレプリケーションのセットアップ*を参照してください。

バグを報告する

A.3. UDP のデフォルト値と推奨値

JGroups と TCP および UDP の使用の詳細については、「JGroups の設定 (ライブラリーモード) 」を参照してください。



注記

- **`JGroups Default Value`** の値は、JGroups に対して内部的に設定される値を示しますが、カスタム設定ファイルや JBoss Data Grid に同梱される JGroups 設定ファイルで上書きできます。
- **`JBoss Data Grid Configured Values`** の値は、JBoss Data Grid に同梱される JGroups の設定ファイルのいずれかを使用する場合にデフォルトで使用される値を示します。JGroups のカスタム設定ファイルが JBoss Data Grid と併用される場合にこれらの値を使用することが推奨されます。

JBoss Data Grid に組み込まれる設定ファイルの詳細情報は、「JBoss Data Grid JGroups 設定ファイル」 を参照してください。

表A.17 UDP の推奨値

パラメーター	JGroups のデフォルト値	JBoss Data Grid の設定値
bind_addr	任意の非ループバック	特定のインターフェースにアドレスを設定
bind_port	任意の空きポート	特定のポートを設定
loopback	true	true

パラメーター	JGroups のデフォルト値	JBoss Data Grid の設定値
port_range	50	必要なポート範囲に 基づいて設定
mcast_addr	228.8.8.8	デフォルト値と同じ
mcast_port	7600	デフォルト値と同じ
tos	8	デフォルト値と同じ
ucast_recv_buf_size	64,000	20,000,000
ucast_send_buf_size	100,000	1,000,000
mcast_recv_buf_size	500,000	25,000,000
mcast_send_buf_size	100,000	1,000,000
ip_ttl	8	2
thread_naming_pattern	cl	pl
max_bundle_size	64,000	デフォルト値と同じ
enable_diagnostics	true	false
thread_pool.enabled	true	デフォルト値と同じ
thread_pool.min_threads	2	これは、ノードの数と同じである必要があります。
thread_pool.max_threads	30	これは、thread_pool.min_threads よりも大きい必要があります。たとえば、小さいグリッド (2〜10 ノード) の場合は、この値をノードの数の 2 倍に設定しますが、大きいグリッド (20 以上のノード) は、比率を小さくする必要があります。たとえば、グリッドに 20 ノードが含まれる場合はこの値を 25 に設定し、グリッドに 100 ノードが含まれる場合はこの値を 110 に設定します。
thread_pool.keep_alive_time	30,000	60,000
thread_pool.queue_enabled	true	false

パラメーター	JGroups のデフォルト値	JBoss Data Grid の設定値
thread_pool.queue_max_size	500	なし。キューを無効にする必要がある
thread_pool.rejection_policy	Discard	デフォルト値と同じ
internal_thread_pool.enabled	true	デフォルト値と同じ
internal_thread_pool.min_threads	2	5
internal_thread_pool.max_threads	4	20
internal_thread_pool.keep_alive_time	30,000	60,000
internal_thread_pool.queue_enabled	true	false
internal_thread_pool.rejection_policy	Discard	停止
oob_thread_pool.enabled	true	デフォルト値と同じ
oob_thread_pool.min_threads	2	20 以上
oob_thread_pool.max_threads	10	200 以上 (負荷に基づく)
oob_thread_pool.keep_alive_time	30,000	60,000
oob_thread_pool.queue_enabled	true	false
oob_thread_pool.queue_max_size	500	なし。キューを無効にする必要があります。
oob_thread_pool.rejection_policy	Discard	デフォルト値と同じ

注記

JGroups 3.5 では、PING タイムアウト値が削除され、pbcast.GMS ***join_timeout*** 値が代わりにタイムアウト期間を示します。

表A.18 MERGE3 の推奨値

パラメーター	JGroups のデフォルト値	JBoss Data Grid の設定値
min_interval	1,000	10,000
max_interval	10,000	30,000

表A.19 FD_SOCK の推奨値

パラメーター	JGroups のデフォルト値	JBoss Data Grid の設定値
client_bind_por	0 (ポートがランダムに選択され、使用されます)	デフォルト値と同じ
get_cache_timeout	1000 ミリ秒	デフォルト値と同じ
keep_alive	true	デフォルト値と同じ
num_tries	3	デフォルト値と同じ
start_port	0 (ポートがランダムに選択され、使用されます)	デフォルト値と同じ
suspect_msg_interval	5000 ミリ秒	デフォルト値と同じ

表A.20 FD_ALL の推奨値

パラメーター	JGroups のデフォルト値	JBoss Data Grid の設定値
timeout	40,000	60,000。FD_ALL タイムアウト値は、CMS ガーベッジコレクターでの stop the world ガーベッジコレクション一時停止の最大時間の 2 倍に設定されます。適切にチューニングされた JVM では、一時停止の最大時間はヒープサイズに応じて決まり、ヒープの 1 GB あたり 1 秒を超えないようにする必要があります。たとえば、8GB のヒープでは、一時停止時間が 8 秒を超えないようにし、FD_ALL タイムアウト値を 16 秒に設定する必要があります。長いガーベッジコレクション一時停止が使用された場合は、ノードで false 障害検出を回避するためにこのタイムアウト値を増やす必要があります。

パラメーター	JGroups のデフォルト値	JBoss Data Grid の設定値
間隔	8,000	15,000。FD_ALL ***interval*** 値は、FD_ALL の ***timeout*** 値に設定された値よりも 4 分 1 以下である必要があります。
timeout_check_interval	2,000	5,000

表A.21 FD_HOST の推奨値

パラメーター	JGroups のデフォルト値	JBoss Data Grid の設定値
check_timeout	3,000	5,000
cmd	InetAddress.isReachable() (ICMP ping)	-
間隔	20,000	15,000。FD_HOST の ***interval*** 値は、FD_HOST の ***timeout*** 値の 4 分 1 である必要があります。
timeout	-	60,000

表A.22 VERIFY_SUSPECT の推奨値

パラメーター	JGroups のデフォルト値	JBoss Data Grid の設定値
timeout	2,000	5,000

表A.23 pbcast.NAKACK2 の推奨値

パラメーター	JGroups のデフォルト値	JBoss Data Grid の設定値
use_mcast_xmit	true	false
xmit_interval	1,000	デフォルト値と同じ
xmit_table_num_rows	50	50
xmit_table_msgs_per_row	10,000	1,024
xmit_table_max_compaction_time	10,000	30,000
max_msg_batch_size	100	デフォルト値と同じ

パラメーター	JGroups のデフォルト値	JBoss Data Grid の設定値
resend_last_seqno	false	true

表A.24 UNICAST3 の推奨値

パラメーター	JGroups のデフォルト値	JBoss Data Grid の設定値
xmit_interval	500	デフォルト値と同じ
xmit_table_num_rows	100	50
xmit_table_msgs_per_row	1,0000	1,024
xmit_table_max_compaction_time	600,000	30,000
max_msg_batch_size	500	100
conn_close_timeout	60,000	推奨値なし
conn_expiry_timeout	120,000	0

表A.25 pbcast.STABLE の推奨値

パラメーター	JGroups のデフォルト値	JBoss Data Grid の設定値
stability_delay	6,000	500
desired_avg_gossip	20,000	5,000
max_bytes	2,000,000	1,000,000

表A.26 pbcast.GMS の推奨値

パラメーター	JGroups のデフォルト値	JBoss Data Grid の設定値
print_local_addr	true	false
join_timeout	5,000	15,000
view_bundling	true	デフォルト値と同じ

表A.27 UFC の推奨値

パラメーター	JGroups のデフォルト値	JBoss Data Grid の設定値
max_credits	500,000	2,000,000
min_threshold	0.40	デフォルト値と同じ

表A.28 MFC の推奨値

パラメーター	JGroups のデフォルト値	JBoss Data Grid の設定値
max_credits	500,000	2,000,000
min_threshold	0.40	デフォルト値と同じ

表A.29 FRAG2 の推奨値

パラメーター	JGroups のデフォルト値	JBoss Data Grid の設定値
frag_size	60,000	デフォルト値と同じ

表A.30 SYM_ENCRYPT の推奨値

パラメーター	JGroups のデフォルト値	JBoss Data Grid の設定値
sym_algorithm	AES	-
sym_keylength	128	-
sym_provider	Bouncy Castle Provider	-

表A.31 ASYM_ENCRYPT の推奨値

パラメーター	JGroups のデフォルト値	JBoss Data Grid の設定値
asym_algorithm	RSA	-
asym_keylength	512	-
asym_provider	Bouncy Castle Provider	-
change_keys_on_leave	false	-

SASL の推奨値

詳細については、『Red Hat JBoss Data Grid Developer Guide 』の「User Authentication over Hot Rod Using SASL」セクションを参照してください。

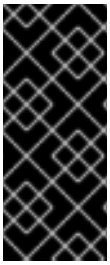
RELAY2 の推奨値

詳細については、35章*データセンター間のレプリケーションのセットアップ*を参照してください。

バグを報告する

A.4. TCPGOSSIP JGROUPS プロトコル

TCPGOSSIP ディスカバリープロトコルは 1 つ以上の設定された GossipRouter プロセスを使用してクラスターのノードについての情報を保存します。



重要

GossipRouter プロセスがクラスター内のすべてのノードで一貫して利用できることは重要になります。このプロセスがないと、ノードを追加することができないためです。このため、このプロセスは利用可能度の高いメソッドでこのプロセスをデプロイすることを強くお勧めします。たとえば、複数の仮想マシンが対象になる Availability Set を使用することができます。

GossipRouter の実行

GossipRouter は JGroups jar ファイルに含まれており、ノードが起動する前に実行中である必要があります。このプロセスは、JBoss Data Grid に含まれている JGroups jar ファイルの **GossipRouter** クラスを参照して開始できます。

```
java -classpath jgroups-${jgroups.version}.jar
org.jgroups.stack.GossipRouter -bindaddress IP_ADDRESS -port PORT
```

複数の GossipRouters が利用可能で指定されている場合、ノードは常にすべての指定された GossipRouters に登録されます。ただし、これは最初に利用可能な GossipRouter からのみ情報を取得します。GossipRouter が利用できない場合、失敗としてマークされ、一覧から削除されます。その際、バックグラウンドスレッドが失敗した GossipRouter への再接続を定期的に試行します。スレッドが GossipRouter に正常に再接続すると、GossipRouter が一覧に再度挿入されます。

JBoss Data Grid で TCPGOSSIP を使用するように設定する (ライブラリーモード)

ライブラリーモードでは、JGroups xml ファイルを使用して **TCPGOSSIP** を設定する必要がありますが、デフォルトでは **TCPGOSSIP** 設定は含まれていません。「事前設定された JGroups ファイル」で指定されている既存ファイルのいずれかを使用し、**TCPGOSSIP** を組み込むように設定を調整することをお勧めします。たとえば、**default-configs/default-jgroups-ec2.xml** を選択してから **S3_PING** プロトコルを削除し、以下のブロックを代わりに追加することができます。

```
<TCPGOSSIP initial_hosts="IP_ADDRESS_0[PORT_0],IP_ADDRESS_1[PORT_1]" />
```

JBoss Data Grid で TCPGOSSIP を使用できるように設定する (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は、**stack** をサーバーの設定ファイルの **jgroups** サブシステムで **TCPGOSSIP** について定義できます。以下の設定スニペットにこの例が含まれます。

```
<subsystem xmlns="urn:infinispan:server:jgroups:8.0" default-
stack="${jboss.default.jgroups.stack:tcpgossip}">
[...]
    <stack name="jdbc_ping">
        <transport type="TCP" socket-binding="jgroups-tcp"/>
        <protocol type="TCPGOSSIP">
```

```
            <property
name="initial_hosts">IP_ADDRESS_0[PORT_0],IP_ADDRESS_1[PORT_1]</property>
        </protocol>
        <protocol type="MERGE3"/>
        <protocol type="FD_SOCK" socket-binding="jgroups-tcp-fd"/>
        <protocol type="FD_ALL"/>
        <protocol type="VERIFY_SUSPECT"/>
        <protocol type="pbcast.NAKACK2">
            <property name="use_mcast_xmit">false</property>
        </protocol>
        <protocol type="UNICAST3"/>
        <protocol type="pbcast.STABLE"/>
        <protocol type="pbcast.GMS"/>
        <protocol type="MFC"/>
        <protocol type="FRAG2"/>
    </stack>
[...]
</subsystem>
```

バグを報告する

A.5. TCPGOSSIP 設定オプション

以下の **TCPGOSSIP** 固有のプロパティーを設定できます。

- **initial_hosts** - 初期のメンバーシップについて接続されるホストのカンマ区切りのリストです。
- **reconnect_interval** - 接続が切断されたノードが Gossip Router への再接続を試行する間隔 (ミリ秒単位)。
- **sock_conn_timeout** - ソケット作成に許可される最大時間 (ミリ秒単位)。デフォルトは **1000** に設定されます。
- **sock_read_timeout** - 読み取りがブロックされる最大時間 (ミリ秒単位)。**0** を値として指定すると無制限にブロックされます。

バグを報告する

A.6. JBOSS DATA GRID JGROUPS 設定ファイル

以下の設定ファイルは JBoss Data Grid に含まれており、これには JGroups の推奨される値が含まれます。以下のファイルはすべて、ライブラリーモードのディストリビューションにある **infinispan-embedded-${infinispan.version}.jar** にあります。

- **default-configs/default-jgroups-ec2.xml**
- **default-configs/default-jgroups-google.xml**
- **default-configs/default-jgroups-tcp.xml**
- **default-configs/default-jgroups-udp.xml**

バグを報告する

付録B JCONSOLE による接続

B.1. JCONSOLE 経由での JDG への接続

JConsole は JMX GUI であり、これによりユーザーは、JVM、その MBeans を監視し、各種操作を実行するために、ローカルまたはリモートで JVM に接続できます。

手順B.1 管理ユーザーの JBoss Data Grid への追加

リモートの JBoss Data Grid インスタンスに接続できるようにするには、ユーザーが作成されている必要があります。ユーザーはリモートインスタンスで以下の手順を実行します。

1. **bin** ディレクトリーに移動します。

```
cd $JDG_HOME/bin
```

2. **add-user.sh** スクリプトを実行します。

```
./add-user.sh
```

3. Return を押して **ManagementUser** のデフォルトオプションを受け入れます。
4. Return を押して **ManagementRealm** のデフォルトオプションを受け入れます。
5. 必要なユーザー名を入力します。この例では **jmxadmin** が使用されます。
6. パスワードを入力し、確認します。
7. Return を押して no groups のデフォルトオプションを受け入れます。
8. **yes** を入力して、必要なユーザーが **ManagementRealm** に追加されることを確認します。
9. このユーザーはプロセス間の接続で使用されないため、**no** を入力します。
10. 以下の画像は、実行例を示しています。

```
What type of user do you wish to add?
 a) Management User (mgmt-users.properties)
 b) Application User (application-users.properties)
(a):

Enter the details of the new user to add.
Realm (ManagementRealm) :
Username : jmxadmin
Password requirements are listed below. To modify these restrictions edit the add-user.properties configuration file.
 - The password must not be one of the following restricted values {root, admin, administrator}
 - The password must contain at least 8 characters, 1 alphabetic character(s), 1 digit(s), 1 non-alphanumeric symbol(s)
 - The password must be different from the username
Password :
Re-enter Password :
What groups do you want this user to belong to? (Please enter a comma separated list, or leave blank for none)[  ]:
About to add user 'jmxadmin' for realm 'ManagementRealm'
Is this correct yes/no? yes
Added user 'jmxadmin' to file '/opt/jboss/jboss-datagrid-6.5.0-server/standalone/configuration/mgmt-users.properties'
Added user 'jmxadmin' with groups  to file '/opt/jboss/jboss-datagrid-6.5.0-server/standalone/configuration/mgmt-groups.properties'
Is this new user going to be used for one AS process to connect to another AS process?
e.g. for a slave host controller connecting to the master or for a Remoting connection for server to server EJB calls.
yes/no? no
```

図B.1 add-user.sh の実行

管理インターフェースのバインディング

デフォルトでは、JBoss Data Grid は 127.0.0.1 への管理インターフェースのバインディングから開始します。リモート接続を実行するために、このインターフェースはネットワークで表示できる IP アドレスにバインドされている必要があります。以下のいずれかのオプションでこれを実行できます。

- **オプション 1: ランタイム -** 起動時に **`jboss.bind.address.management`** プロパティーを調整することにより、新規の IP アドレスを指定できます。以下の例では、JBoss Data Grid は起動時に 192.168.122.5 にバインドされています。

```
./standalone.sh ... -Djboss.bind.address.management=192.168.122.5
```

- **オプション 2: 設定 -** 設定ファイルで **`jboss.bind.address.management`** を調整します。これは **`interfaces`** サブシステムにあります。IP が 192.168.122.5 に調整された設定ファイルのスニペットは以下のようになります。

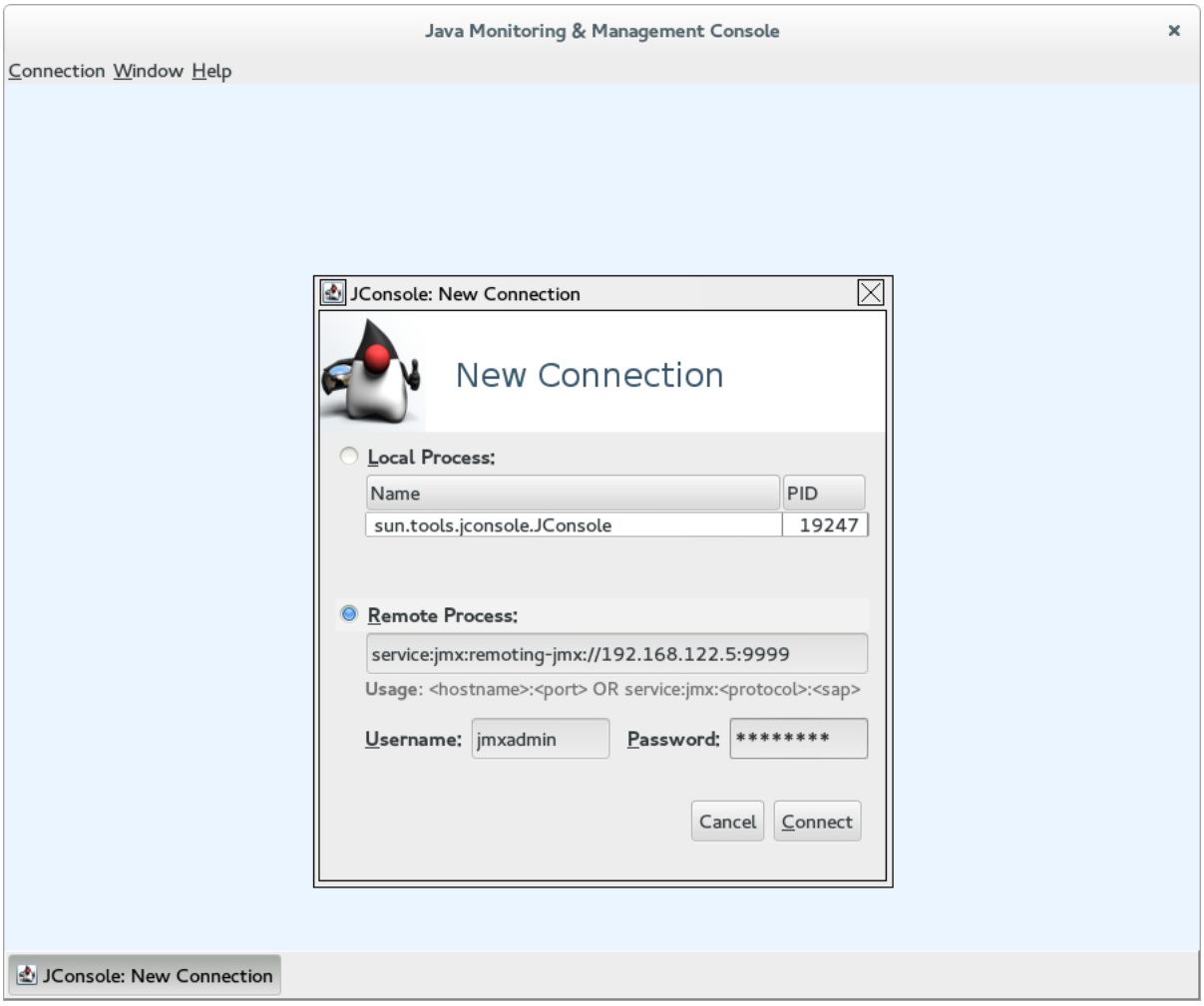
```
<interfaces>
    <interface name="management">
        <inet-address
value="${jboss.bind.address.management:192.168.122.5}"/>
    </interface>
    [...]
</interface>
```

JConsole の実行

`jconsole.sh` スクリプトは **`$JDG_HOME/bin`** ディレクトリーで提供されます。このスクリプトを実行すると、JConsole が起動します。

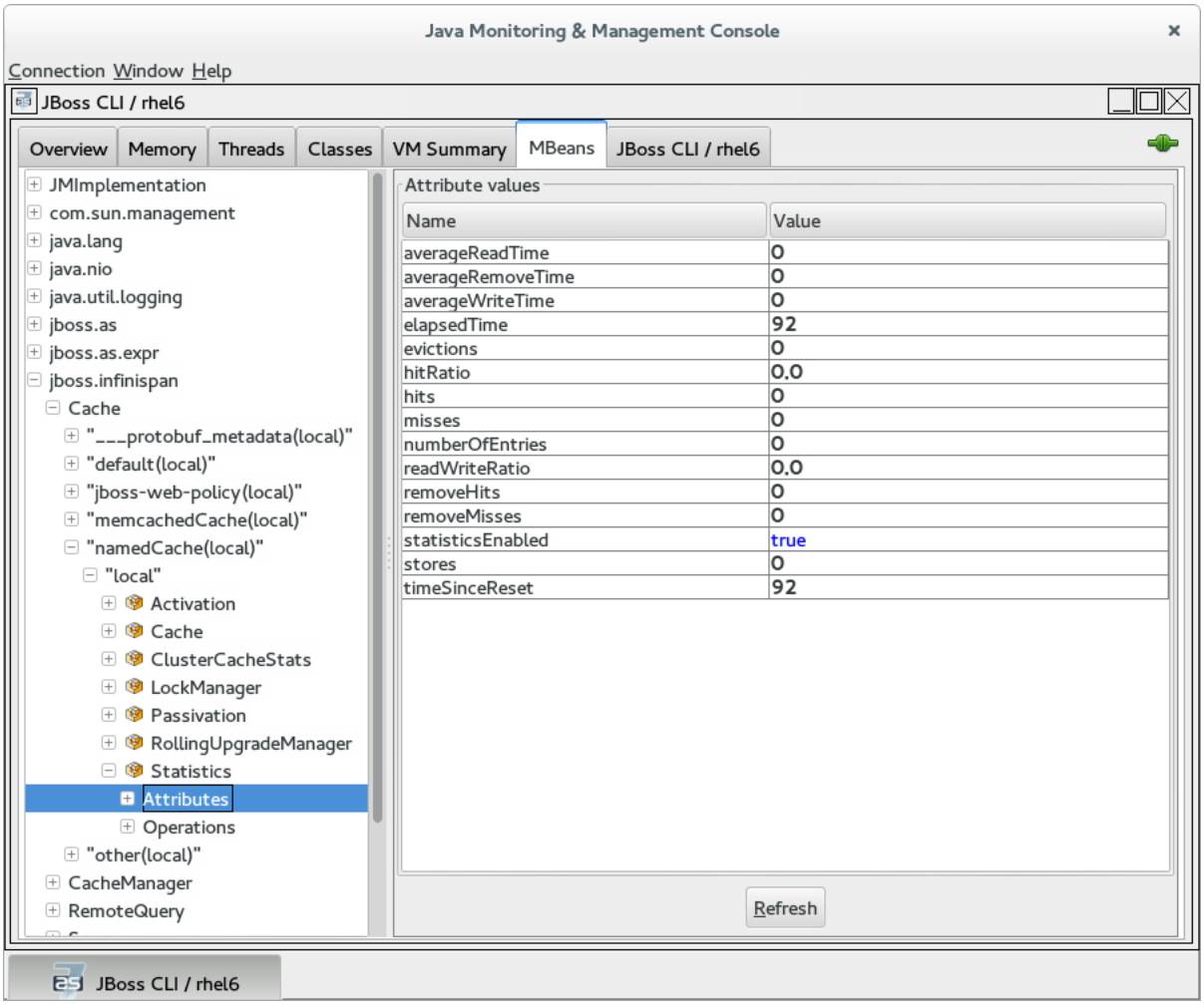
手順B.2 JConsole を使用したリモート JBoss Data Grid インスタンスへの接続

1. **`$JDG_HOME/bin/jconsole.sh`** スクリプトを実行します。これにより、以下を表示するウィンドウが表示されます。



図B.2 JConsole

2. **`Remote Process`** を選択します。
3. テキスト領域に **`service:jmx:remoting-jmx://$IP:9999`** と入力します。
4. **`add-user.sh`** スクリプトで作成されるユーザー名およびパスワードを入力します。
5. **`Connect`** をクリックして接続を開始します。
6. いったん接続されたら、キャッシュ関連のノードが表示されることを確認します。以下のスクリーンショットはこのノード例を示しています。



図B.3 JConsole: キャッシュの表示

バグを報告する

付録C RED HAT JBOSS DATA GRID における JMX MBEANS

C.1. ACTIVATION

`org.infinispan.eviction.ActivationManagerImpl`

エントリーをメモリーにロードすることにより、CacheStore にパッシベートされたエントリーをアクティベートします。

表C.1 属性

名前	説明	タイプ	書き込み可能
activations	アクティベーションイベントの数です。	文字列	いいえ
statisticsEnabled	このコンポーネントにより、統計の収集を有効または無効にします。	ブール値	はい

表C.2 操作

名前	説明	署名
resetStatistics	このコンポーネントによって収集される統計をリセットします。	void resetStatistics()

バグを報告する

C.2. CACHE

`org.infinispan.CacheImpl`

Cache コンポーネントは、個別のキャッシュインスタンスを表します。

表C.3 属性

名前	説明	タイプ	書き込み可能
cacheName	キャッシュ名を返します。	文字列	いいえ
cacheStatus	キャッシュの状態を返します。	文字列	いいえ
configurationAsProperties	プロパティーの形式でキャッシュの設定を返します。	プロパティー	いいえ

名前	説明	タイプ	書き込み可能
version	Infinispan のバージョンを返します。	文字列	いいえ
cacheAvailability	キャッシュの利用可能性を返す	文字列	はい

表C.4 操作

名前	説明	署名
start	キャッシュを起動します。	void start()
stop	キャッシュを停止します。	void stop()
clear	キャッシュをクリアにします。	void clear()

バグを報告する

C.3. CACHECONTAINERSTATS

`org.infinispan.stats.impl.CacheContainerStatsImpl`

CacheContainerStats コンポーネントには、タイミング、ヒット/ミス比率、稼働情報などの統計が含まれます。

表C.5 属性

名前	説明	タイプ	書き込み可能
averageReadTime	このキャッシュコンテナー内のすべての読み取り操作に対するキャッシュコンテナー合計平均時間 (ミリ秒単位)。	long	いいえ
averageRemoveTime	このキャッシュコンテナー内のすべての削除操作に対するキャッシュコンテナー合計平均時間 (ミリ秒単位)。	long	いいえ
averageWriteTime	このキャッシュコンテナー内のすべての書き込み操作に対するキャッシュコンテナー合計平均時間 (ミリ秒単位)。	long	いいえ

名前	説明	タイプ	書き込み可能
evictions	キャッシュエビクション操作のキャッシュコンテナー合計数。	long	いいえ
hitRatio	このキャッシュに対するキャッシュコンテナー総ヒット比率 (ヒット数/(ヒット数 + ミス数))。	double	いいえ
hits	キャッシュ属性ヒットのキャッシュコンテナー合計数。	long	いいえ
misses	キャッシュ属性ミスのキャッシュコンテナー合計数。	long	いいえ
numberOfEntries	このキャッシュコンテナーのすべてのキャッシュに現在存在するエントリーのキャッシュコンテナー合計数。	整数	いいえ
readWriteRatio	このキャッシュコンテナーのすべてのキャッシュのキャッシュコンテナー読み取り/書き込み比率。	double	いいえ
removeHits	削除ヒットのキャッシュコンテナー合計数。	double	いいえ
removeMisses	キーが見つからなかった場合のキャッシュ削除のキャッシュコンテナー合計数。	long	いいえ
statisticsEnabled	このコンポーネントにより、統計の収集を有効または無効にします。	ブール値	はい
stores	キャッシュ属性 put 操作のキャッシュコンテナー合計数。	long	いいえ

バグを報告する

C.4. CACHELOADER

`org.infinispan.interceptors.CacheLoaderInterceptor`

このコンポーネントは、エントリーを CacheStore からメモリーにロードします。

表C.6 属性

名前	説明	タイプ	書き込み可能
cacheLoaderLoads	キャッシュストアからロードされるエントリーの数です。	long	いいえ
cacheLoaderMisses	キャッシュストアに存在しなかったエントリーの数です。	long	いいえ
stores	設定済みの有効にされているキャッシュローダーのコレクションを返します。	コレクション	いいえ
statisticsEnabled	このコンポーネントにより、統計の収集を有効または無効にします。	ブール値	はい

表C.7 操作

名前	説明	署名
disableStore	指定されるタイプのすべてのキャッシュローダーを無効にします。このタイプは、無効にするキャッシュローダーの完全修飾クラス名です。	void disableStore(String storeType)
resetStatistics	このコンポーネントによって収集される統計をリセットします。	void resetStatistics()

バグを報告する

C.5. CACHEMANAGER

`org.infinispan.manager.DefaultCacheManager`

CacheManager コンポーネントは、システム内のキャッシュのマネージャー、ファクトリー、およびコンテナーとして動作します。

表C.8 属性

名前	説明	タイプ	書き込み可能
cacheManagerStatus	キャッシュマネージャーのインスタンスの状態です。	文字列	いいえ
clusterMembers	クラスターのメンバーを一覧表示します。	文字列	いいえ
clusterName	クラスター名です。	文字列	いいえ
clusterSize	ノードの数で表されるクラスターのサイズです。	整数	いいえ
createdCacheCount	デフォルトキャッシュを含む、作成されたキャッシュの合計数です。	文字列	いいえ
definedCacheCount	デフォルトキャッシュを除く、定義されたキャッシュの合計数です。	文字列	いいえ
definedCacheNames	定義されたキャッシュ名とそれらのキャッシュの状態です。デフォルトのキャッシュはこの表示には含まれません。	文字列	いいえ
name	このキャッシュマネージャーの名前です。	文字列	いいえ
nodeAddress	このインスタンスに関連付けられたネットワークアドレスです。	文字列	いいえ
physicalAddresses	このインスタンスに関連付けられた物理ネットワークアドレスです。	文字列	いいえ
runningCacheCount	デフォルトキャッシュを含む、実行中のキャッシュの合計数です。	文字列	いいえ
version	Infinispan のバージョンです。	文字列	いいえ

名前	説明	タイプ	書き込み可能
globalConfigurationAsProperties	グローバル設定プロパティーです。	プロパティー	いいえ

表C.9 操作

名前	説明	署名
startCache	キャッシュマネージャーに関連付けられたデフォルトのキャッシュを起動します。	void startCache()
startCache	このキャッシュマネージャーから名前付きキャッシュを起動します。	void startCache (String p0)

バグを報告する

C.6. CACHESTORE

`org.infinispan.interceptors.CacheWriterInterceptor`

CacheStore コンポーネントは、エントリーをメモリーから CacheStore に保存します。

表C.10 属性

名前	説明	タイプ	書き込み可能
writesToTheStores	ストアへの書き込み回数です。	long	いいえ
statisticsEnabled	このコンポーネントにより、統計の収集を有効または無効にします。	ブール値	はい

表C.11 操作

名前	説明	署名
resetStatistics	このコンポーネントによって収集される統計をリセットします。	void resetStatistics()

バグを報告する

C.7. CLUSTERCACHESTATS

`org.infinispan.stats.impl.ClusterCacheStatsImpl`

ClusterCacheStats コンポーネントには、クラスター全体のタイミング、ヒット/ミス比率、稼働情報などの統計が含まれます。

表C.12 属性

名前	説明	タイプ	書き込み可能
activations	クラスター内のアクティベーションの合計数。	long	いいえ
averageReadTime	キャッシュの読み取り操作にかかるクラスター全体での総平均時間 (ミリ秒単位)。	long	いいえ
averageRemoveTime	キャッシュの削除操作にかかるクラスター全体での総平均時間 (ミリ秒単位)。	long	いいえ
averageWriteTime	キャッシュの書き込み操作にかかるクラスター全体での平均時間 (ミリ秒単位)。	long	いいえ
cacheLoaderLoads	クラスター内のキャッシュローダーロード操作の合計数。	long	いいえ
cacheLoaderMisses	クラスター内のキャッシュローダーロードミスの合計数。	long	いいえ
evictions	キャッシュエビクション操作のクラスター全体での合計数。	long	いいえ
hitRatio	このキャッシュに対するクラスター全体での総ヒット比率 (ヒット数/(ヒット数 + ミス数))。	double	いいえ
hits	クラスター全体でのキャッシュヒット合計数。	long	いいえ
invalidations	クラスター内のインバリデーションの合計数。	long	いいえ

名前	説明	タイプ	書き込み可能
misses	クラスター全体でのキャッシュ属性ミスの合計数。	long	いいえ
numberOfEntries	現在キャッシュにあるエントリーのクラスター全体での合計数。	整数	いいえ
numberOfLocksAvailable	クラスターで利用可能な排他ロックの合計数。	整数	いいえ
numberOfLocksHeld	クラスターで保持されたロックの合計数。	整数	いいえ
passivations	クラスター内のパッシベーションの合計数。	long	いいえ
readWriteRatio	キャッシュのクラスター全体での読み取り/書き込み比率。	double	いいえ
removeHits	クラスター全体でのキャッシュ削除ヒットの合計数。	double	いいえ
removeMisses	キーが見つからなかった場合のキャッシュ削除のクラスター全体での合計数。	long	いいえ
statisticsEnabled	このコンポーネントにより、統計の収集を有効または無効にします。	ブール値	はい
storeWrites	クラスター内のキャッシュストア格納操作の合計数。	long	いいえ
stores	キャッシュ属性 put 操作のクラスター全体での合計数。	long	いいえ
timeSinceStart	最初のキャッシュノードが開始された以降の時間 (秒単位)。	long	いいえ

表C.13 操作

名前	説明	署名
setStaleStatsTreshold	クラスター全体での統計更新のしきい値 (ミリ秒単位) を設定します。	void setStaleStatsTreshold(long staleStatsThreshold)
resetStatistics	このコンポーネントによって収集される統計をリセットします。	void resetStatistics()

バグを報告する

C.8. DEADLOCKDETECTINGLOCKMANAGER

`org.infinispan.util.concurrent.locks.DeadlockDetectingLockManager`

このコンポーネントは、検出されたデッドロックの数についての情報を提供します。

表C.14 属性

名前	説明	タイプ	書き込み可能
detectedLocalDeadlocks	デッドロックによりロールバックされたローカルトランザクションの数です。	long	いいえ
detectedRemoteDeadlocks	デッドロックによりロールバックされたリモートトランザクションの数です。	long	いいえ
overlapWithNotDeadlockAwareLockOwners	デッドロックの判別を試行しているが、他のロックを所有するのがトランザクションでは「ない」状況の数です。このシナリオでは、デッドロック検出メカニズムを実行することはできません。	long	いいえ
totalNumberOfDetectedDeadlocks	検出されたローカルデッドロックの合計数です。	long	いいえ
concurrencyLevel	MVCC ロックマネージャーについて設定された平行性レベルです。	int	いいえ
numberOfLocksAvailable	利用可能な排他ロックの数です。	int	いいえ

名前	説明	タイプ	書き込み可能
numberOfLocksHeld	保持されている排他ロックの数です。	int	いいえ

表C.15 操作

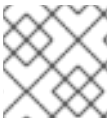
名前	説明	署名
resetStatistics	このコンポーネントによって収集される統計をリセットします。	void resetStatistics()

バグを報告する

C.9. DISTRIBUTIONMANAGER

`org.infinispan.distribution.DistributionManagerImpl`

DistributionManager コンポーネントは、クラスター間のコンテンツの分散を処理します。



注記

DistrubutionManager コンポーネントはクラスターモードのみで利用可能です。

表C.16 操作

名前	説明	署名
isAffectedByRehash	指定されたキーが進行中のリハッシュによって影響を受けるかどうかを決定します。	boolean isAffectedByRehash(Object p0)
isLocatedLocally	指定されたキーがキャッシュのこのインスタンスに対してローカルであるかどうかを示します。文字列キーでのみ機能します。	boolean isLocatedLocally(String p0)
locateKey	クラスター内のオブジェクトを見つけます。文字列キーでのみ機能します。	List locateKey(String p0)

バグを報告する

C.10. INTERPRETER

`org.infinispan.cli.interpreter.Interpreter`

Interpreter コンポーネントは、コマンドラインインターフェース (CLI 操作) を実行します。

表C.17 属性

名前	説明	タイプ	書き込み可能
cacheNames	キャッシュマネージャーのキャッシュのリストを取得します。	String[]	いいえ

表C.18 操作

名前	説明	署名
createSessionId	新規のインタープリターセッションを作成します。	String createSessionId(String cacheName)
execute	IspnCliQL ステートメントを解析し、実行します。	String execute(String p0, String p1)

バグを報告する

C.11. INVALIDATION

`org.infinispan.interceptors.InvalidationInterceptor`

Invalidation コンポーネントは、エントリーがローカルに作成される場合にリモートキャッシュのエントリーを無効にします。

表C.19 属性

名前	説明	タイプ	書き込み可能
invalidations	インバリデーションの数です。	long	いいえ
statisticsEnabled	このコンポーネントにより、統計の収集を有効または無効にします。	ブール値	はい

表C.20 操作

名前	説明	署名
resetStatistics	このコンポーネントによって収集される統計をリセットします。	void resetStatistics()

バグを報告する

C.12. LOCKMANAGER

`org.infinispan.util.concurrent.locks.LockManagerImpl`

LockManager コンポーネントは、エントリーの MVCC ロックを処理します。

表C.21 属性

名前	説明	タイプ	書き込み可能
concurrencyLevel	MVCC ロックマネージャーが設定される平行性レベル。	整数	いいえ
numberOfLocksAvailable	利用可能な排他ロックの数。	整数	いいえ
numberOfLocksHeld	保留にされている排他ロックの数。	整数	いいえ

バグを報告する

C.13. LOCALTOPOLOGYMANAGER

`org.infinispan.topology.LocalTopologyManagerImpl`

LocalTopologyManager コンポーネントは、Red Hat JBoss Data Grid におけるキャッシュのメンバーシップと状態の転送を制御します。

注記

LocalTopologyManager コンポーネントは、クラスターモードでのみ利用可能です。

表C.22 属性

Name	説明	タイプ	書き込み可能

Name	説明	タイプ	書き込み可能
rebalancingEnabled	false の場合、新規に起動したノードは既存のクラスターに参加せず、状態もそれらに転送されません。現在のクラスターメンバーのいずれかが再調整が無効にされている状態で停止した場合、ノードがそのクラスターを離れ、状態の再調整は残りのノード間で行なわれません。これにより、再調整が再び有効にされるまで、コピーの数は **owners** 属性で指定される数よりも少なくなります。	ブール値	はい
clusterAvailability	**AVAILABLE** の場合は、ノードが現在正常に動作しています。**DEGRADED** の場合は、ネットワークの分割または連続したノードの脱退のため、データに安全にアクセスできません。	文字列	いいえ

バグを報告する

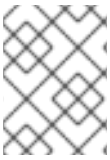
C.14. MASSINDEXER

`org.infinispan.query.MassIndexer`

MassIndexer コンポーネントは、キャッシュされたデータを使用してインデックスを再構築します。

表C.23 操作

Name	説明	署名
start	インデックスの再構築を開始します。	void start()



注記

この操作は、インデックス化を有効にしたキャッシュの場合にのみ利用できます。さらに詳しくは、Red Hat JBoss Data Grid の 『Developer Guide』 を参照してください。

バグを報告する

C.15. PASSIVATION

`org.infinispan.eviction.PassivationManager`

パッシベーションコンポーネントは、エビクションの CacheStore へのエントリーのパッシベーションを処理します。

表C.24 属性

名前	説明	タイプ	書き込み可能
passivations	パッシベーションイベントの数です。	文字列	いいえ
statisticsEnabled	このコンポーネントにより、統計の収集を有効または無効にします。	ブール値	はい

表C.25 操作

名前	説明	署名
resetStatistics	このコンポーネントによって収集される統計をリセットします。	void resetStatistics()

バグを報告する

C.16. RECOVERYADMIN

`org.infinispan.transaction.xa.recovery.RecoveryAdminOperations`

RecoveryAdmin コンポーネントは、トランザクションリカバリーを処理するためのツールを公開します。

表C.26 操作

名前	説明	署名
forceCommit	不明なトランザクションのコミットを強制します。	String forceCommit(long p0)
forceCommit	不明なトランザクションのコミットを強制します。	String forceCommit(int p0, byte[] p1, byte[] p2)
forceRollback	不明なトランザクションのロールバックを強制します。	String forceRollback(long p0)
forceRollback	不明なトランザクションのロールバックを強制します。	String forceRollback(int p0, byte[] p1, byte[] p2)

名前	説明	署名
forget	指定されるトランザクションのリカバリー情報を削除します。	String forget(long p0)
forget	指定されるトランザクションのリカバリー情報を削除します。	String forget(int p0, byte[] p1, byte[] p2)
showInDoubtTransactions	元のノードがクラッシュする準備されたトランザクションをすべて表示します。	String showInDoubtTransactions()

バグを報告する

C.17. ROLLINGUPGRADEMANAGER

`org.infinispan.upgrade.RollingUpgradeManager`

RollingUpgradeManager コンポーネントは、Red Hat JBoss Data Grid の 1 つのバージョンから別のバージョンへのデータ移行を行うために制御フックを処理します。

表C.27 操作

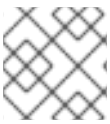
名前	説明	署名
disconnectSource	指定される移行プログラムに従って、ターゲットクラスターをソースクラスターから切り離します。	void disconnectSource(String p0)
recordKnownGlobalKeyset	アップグレードプロセスで取得するために、グローバルな既知のキーセットを既知のキーにダンプします。	void recordKnownGlobalKeyset()
synchronizeData	指定された移行プログラムを使用して、古いクラスターのデータをこれに同期します。	long synchronizeData(String p0)

バグを報告する

C.18. RPCMANAGER

`org.infinispan.remoting.rpc.RpcManagerImpl`

RpcManager コンポーネントは、クラスター内のリモートキャッシュインスタンスへのリモート呼び出しをすべて管理します。



注記

RpcManager コンポーネントは、クラスターモードでのみ利用可能です。

表C.28 属性

名前	説明	タイプ	書き込み可能
averageReplicationTime	トランスポート層で費やされた平均時間 (ミリ秒単位) です。	long	いいえ
committedViewAsString	コミット済みのビューを取得します。	文字列	いいえ
pendingViewAsString	保留中のビューを取得します。	文字列	いいえ
replicationCount	正常なレプリケーションの数です。	long	いいえ
replicationFailures	失敗したレプリケーションの数です。	long	いいえ
successRatio	レプリケーションの合計数に対する正常なレプリケーションの比率です。	文字列	いいえ
successRatioFloatingPoint	数値 (double) 形式でのレプリケーションの合計数に対する正常なレプリケーションの比率です。	double	いいえ
statisticsEnabled	このコンポーネントにより、統計の収集を有効または無効にします。	ブール値	はい

表C.29 操作

名前	説明	署名
resetStatistics	このコンポーネントによって収集される統計をリセットします。	void resetStatistics()
setStatisticsEnabled	統計を有効または無効にするかを設定します (true/false)。	void setStatisticsEnabled(boolean enabled)

バグを報告する

C.19. STATETRANSFERMANAGER

`org.infinispan.statetransfer.StateTransferManager`

StateTransferManager コンポーネントは Red Hat JBoss Data Grid における状態の転送を処理します。

注記

StateTransferManager コンポーネントは、クラスターモードでのみ利用可能です。

表C.30 属性

名前	説明	タイプ	書き込み可能
joinComplete	true の場合、ノードはグリッドに正常に加わっており、保留状態であると見なされます。false の場合、join プロセスは依然として進行中です。	ブール値	いいえ
stateTransferInProgress	このクラスターメンバーに保留中のインバウンドの状態転送があるかどうかをチェックします。	ブール値	いいえ

バグを報告する

C.20. STATISTICS

`org.infinispan.interceptors.CacheMgmtInterceptor`

このコンポーネントは、タイミング、読み取り/書き込み比率などの一般的な統計を処理します。

表C.31 属性

名前	説明	タイプ	書き込み可能
averageReadTime	キャッシュの読み取り操作にかかる平均のミリ秒数です。	long	いいえ
averageWriteTime	キャッシュの書き込み操作にかかる平均のミリ秒数です。	long	いいえ
elapsedTime	キャッシュの開始以降の秒数です。	long	いいえ
evictions	キャッシュエビクション操作の数です。	long	いいえ

名前	説明	タイプ	書き込み可能
hitRatio	キャッシュのヒット/(ヒット+ミス) 比率 (パーセンテージ) です。	double	いいえ
hits	キャッシュ属性のヒット数です。	long	いいえ
misses	キャッシュ属性の失敗回数です。	long	いいえ
numberOfEntries	キャッシュ内の現在のエントリーの数です。	整数	いいえ
readWriteRatio	キャッシュの読み取り/書き込み比率です。	double	いいえ
removeHits	キャッシュ除去のヒット数です。	long	いいえ
removeMisses	キーが見つからなかった場合のキャッシュ除去の回数です。	long	いいえ
stores	キャッシュ属性 PUT 操作の数です。	long	いいえ
timeSinceReset	キャッシュ統計が最後にリセットされてからの秒数です。	long	いいえ
averageRemoveTime	キャッシュの削除操作にかかる平均のミリ秒数です。	long	いいえ

表C.32 操作

名前	説明	署名
resetStatistics	このコンポーネントによって収集される統計をリセットします。	void resetStatistics()

バグを報告する

C.21. TRANSACTIONS

`org.infinispan.interceptors.TxInterceptor`

Transactions コンポーネントは、JTA トランザクションでのキャッシュの参加を管理します。

表C.33 属性

名前	説明	タイプ	書き込み可能
commits	最終リセット時から実行されるトランザクションのコミット数です。	long	いいえ
prepares	最終リセット時から実行されるトランザクションの準備回数です。	long	いいえ
rollbacks	最終リセット時から実行されるトランザクションのロールバック回数です。	long	いいえ
statisticsEnabled	このコンポーネントにより、統計の収集を有効または無効にします。	ブール値	はい

表C.34 操作

名前	説明	署名
resetStatistics	このコンポーネントによって収集される統計をリセットします。	void resetStatistics()

バグを報告する

C.22. TRANSPORT

`org.infinispan.server.core.transport.NettyTransport`

Transport コンポーネントは、サーバーへの/からの読み取りおよび書き込み操作を管理します。

表C.35 属性

名前	説明	タイプ	書き込み可能
hostName	トランスポートがバインドされるホストを返します。	文字列	いいえ
idleTimeout	アイドル状態のタイムアウトを返します。	文字列	いいえ

名前	説明	タイプ	書き込み可能
numberOfGlobalConnections	クラスター内のアクティブな接続の数を返します。この操作は、結果を集約するためのリモート呼び出しを行うため、待ち時間がこの属性の計算スピードに影響を与える場合があります。	整数	false
numberOfLocalConnections	このサーバーのアクティブな接続の数を返します。	整数	いいえ
numberWorkerThreads	ワーカースレッドの数を返します。	文字列	いいえ
port	トランスポートがバインドされるポートを返します。	文字列	
receiveBufferSize	受信バッファーサイズを返します。	文字列	いいえ
sendBufferSize	送信バッファーサイズを返します。	文字列	いいえ
totalBytesRead	プロトコルおよびユーザー情報の両方を含む、サーバーがクライアントから読み取るバイト数の合計を返します。	文字列	いいえ
totalBytesWritten	プロトコルおよびユーザー情報の両方を含む、サーバーがクライアントに書き戻すバイト数の合計を返します。	文字列	いいえ
tcpNoDelay	TCP no delay (TCP 遅延なし) が設定されているかどうかの情報を返します。	文字列	いいえ

バグを報告する

C.23. XSITEADMIN

`org.infinispan.xsite.XSiteAdminOperations`

XSiteAdmin コンポーネントは、データをリモートサイトにバックアップするためのツールを公開します。

表C.36 操作

名前	説明	署名
bringSiteOnline	すべてのクラスターで指定されたサイトをオンラインに戻します。	String bringSiteOnline(String p0)
amendTakeOffline	クラスター内のすべてのノードで 'TakeOffline' 機能の値を修正します。	String amendTakeOffline(String p0, int p1, long p2)
getTakeOfflineAfterFailures	'TakeOffline' 機能に対して 'afterFailures' の値を返します。	String getTakeOfflineAfterFailures(String p0)
getTakeOfflineMinTimeToWait	'TakeOffline' 機能に対して 'minTimeToWait' の値を返します。	String getTakeOfflineMinTimeToWait(String p0)
setTakeOfflineAfterFailures	クラスター内のすべてのノードで 'TakeOffline' 機能について 'afterFailures' の値を修正します。	String setTakeOfflineAfterFailures(String p0, int p1)
setTakeOfflineMinTimeToWait	クラスター内のすべてのノードで 'TakeOffline' 機能について 'minTimeToWait' の値を修正します。	String setTakeOfflineMinTimeToWait(String p0, long p1)
siteStatus	指定されるバックアップサイトがオフラインかどうかをチェックします。	String siteStatus(String p0)
status	設定されたすべてのバックアップサイトについて、ステータス (オフライン/オンライン) を返します。	String status()
takeSiteOffline	クラスター内のすべてのノードでこのノードをオフラインにします。	String takeSiteOffline(String p0)
pushState	指定されたサイト名へのサイト間ステータス転送を開始します。	String pushState(String p0)
cancelPushState	指定されたサイト名へのサイト間状態転送をキャンセルします。	String cancelPushState(String p0)

名前	説明	署名
getSendingSiteName	このサイトに状態をプッシュしているサイト名を返します。	String getSendingSiteName()
cancelReceiveState	サイトを通常の状態に復元します。状態の転送中にサイト間のリンクが壊れた場合に使用されます。	String cancelReceiveState(String p0)
getPushStateStatus	完了したサイト間状態転送と実行中のサイト間状態転送のステータスを返します。	String getPushStateStatus()
clearPushStateStatus	完了したサイト間状態転送のステータスをクリアします。	String clearPushStateStatus()

バグを報告する

付録D 推奨される設定

D.1. タイムアウト値

表D.1 JBoss Data Grid に推奨されるタイムアウト値

タイムアウト値	親要素	デフォルト値	推奨値
distributedSyncTimeout	transport	240,000 (4 分)	デフォルト値と同じ
lockAcquisitionTimeout	locking	10,000 (10 秒)	デフォルト値と同じ
cacheStopTimeout	transaction	30,000 (30 秒)	デフォルト値と同じ
completedTxTimeout	transaction	60,000 (60 秒)	デフォルト値と同じ
replTimeout	sync	15,000 (15 秒)	デフォルト値と同じ
timeout	stateTransfer	240,000 (4 分)	デフォルト値と同じ
timeout	backup	10,000 (10 秒)	デフォルト値と同じ
flushLockTimeout	async	1 (1 ミリ秒)	デフォルト値と同じです。この値は非同期キャッシュストアに適用されますが、非同期キャッシュには適用されないことに注意してください。
shutdownTimeout	async	25,000 (25 秒)	デフォルト値と同じです。この値は非同期キャッシュストアに適用されますが、非同期キャッシュには適用されないことに注意してください。
pushStateTimeout	singletonStore	10,000 (10 秒)	デフォルト値と同じ。
backup	replicationTimeout	10,000 (10 秒)	
remoteCallTimeout	clusterLoader	0	ほとんどの要件については、デフォルト値と同じです。この値は、通常 ***sync.replTimeout*** 値と同じ値に設定されます。

バグを報告する

付録E パフォーマンスに関する推奨事項

E.1. 大規模なクラスターの同時起動

大量のキャッシュを同時に管理する大量のインスタンスを起動する場合は、各ノードがクラスターに参加するときに再調整によりデータを均等に分散しようとするため、しばらく時間がかかることがあります。クラスターの初期起動中に行われる再調整の試行回数を制限するには、以下の手順に従って再調整を一時的に無効にします。

1. クラスターの最初のノードを起動します。
2. 「LocalTopologyManager」に示されたように、JMX 属性 ***jboss.infinispan/CacheManager/"clustered"/LocalTopologyManager/rebalancingEnabled*** を **false** に設定します。
3. クラスターの残りのノードを起動します。
4. 「LocalTopologyManager」に示されたように、この値を**true** に設定し直すことにより、JMX 属性 ***jboss.infinispan/CacheManager/"clustered"/LocalTopologyManager/rebalancingEnabled*** を再び有効にします。

バグを報告する

付録F 参考資料

F.1. 一貫性について

一貫性とは、あるノード上のデータ記録が他のノード上の同じデータ記録と異なることが可能であるかどうかを記述するポリシーのことです。

たとえば、ネットワークの速度が原因で、マスターノード上で実行された書き込み操作がストアの他のノードでは実行されていない場合があります。強い一貫性保証があると、完全にレプリケートされていないデータがアプリケーションに返されることはありません。

バグを報告する

F.2. 一貫性保証について

全所有者ではなく単一所有者のロックであるにも関わらず、Red Hat JBoss Data Grid の一貫性保証は維持されます。一貫性とは次のようなことを言います。

1. キー **K** はノード **{A, B}** へハッシュ化されます。トランザクション **TX1** は、たとえばノード **A** 上の **K** ロックを取得します。

2. 他のキャッシュへのアクセスが、ノード **B** または他のノードで発生すると、**TX2** が **K** をロックしようとしますが、トランザクション **TX1** がすでに **K** のロックを保持しているため、タイムアウトが発生してこのアクセスの試行は失敗します。

キー **K** のロックはトランザクションの発生元に関係なく、常にクラスターの同じノード上で取得されるため、このロックの取得は常に失敗します。

バグを報告する

F.3. JBOSS CACHE について

Red Hat JBoss Cache は、ツリー構造のクラスター化されたトランザクションキャッシュで、スタンドアロンのクラスター化されていない環境でも使用できます。頻繁にアクセスされるデータをインメモリーでキャッシュし、Java Transactional API (JTA) との互換、エビクション、および永続性などのエンタープライズ機能の提供中に、データの読み出しや計算のボトルネックが発生しないようにします。

JBoss Cache は Infinispan と Red Hat JBoss Data Grid の前身です。

バグを報告する

F.4. RELAY2 について

RELAY プロトコルは、各サイトの 1 つのノード間の接続を作成することによって 2 つのリモートクラスターをブリッジします。これにより、1 つのサイトに送信されたマルチキャストメッセージが他のサイトにリレーされ、他のサイトからそのサイトにもリレーさせることができます。

JGroups には、Red Hat JBoss Data Grid のサイト間レプリケーションにおけるサイト間の通信に使用される **RELAY2** プロトコルが含まれます。

RELAY2 プロトコルは **RELAY** と同様に機能しますが、若干の相違点があります。**RELAY** とは異なり、**RELAY2** プロトコルは以下を実行します。

- 3 つ以上のサイトを接続します。

- 自律的に機能し、相互に認識しないサイトに接続します。
- サイト間のユニキャストとマルチキャストのルーティングの両方を提供します。

バグを報告する

F.5. 戻り値について

キャッシュ操作によって返される値は戻り値と呼ばれます。Red Hat JBoss Data Grid では、使用されるキャッシュモードや同期または非同期通信が使用されるかどうかに関係なく、これらの戻り値は信頼性を維持します。

バグを報告する

F.6. 実行可能インターフェースについて

実行可能インターフェースは、クラスのコードのアクティブな部分を実行する単一の **run()** メソッドを宣言します。実行可能オブジェクトは、スレッドコンストラクターに渡された後、独自のスレッドでの実行が可能です。

バグを報告する

F.7. 2 相コミット (2PC) について

2 相コミットプロトコル (2PC) は、分散トランザクションをアトミックにコミットまたはロールバックするために使用される合意プロトコルです。ネットワークノードや通信の障害など一時的なシステム障害が発生しても正常に動作するため、幅広く使用されています。

バグを報告する

F.8. 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

F.9. 完全なバイトアレイの要求

バイトアレイの部分的な内容ではなく、完全なバイトアレイを Red Hat JBoss Data Grid が返すように要求する方法はあるのでしょうか。

デフォルトでは、必要のない巨大なバイトアレイを出力しないように、JBoss Data Grid はバイトアレイの一部のみをログに出力します。以下の場合にバイトアレイがログに出力されます。

- JBoss Data Grid のキャッシュにレイジーデシリアライゼーションが設定されている場合。レイジーデシリアライゼーションは JBoss Data Grid のリモートクライアントサーバーモードでは使用できません。
- **Memcached** または **Hot Rod** サーバーが実行されている場合。

このような場合、最初から 10 ポジションまでのバイトアレイがログに表示されます。バイトアレイの全内容をログに表示するには、起動時に ***-Dinfinispan.arrays.debug=true*** システムプロパティーを渡します。

例F.1 部分的なバイトアレイのログ

```
2010-04-14 15:46:09,342 TRACE [ReadCommittedEntry] (HotRodWorker-1-1)
Updating entry
(key=CacheKey{data=ByteArray{size=19, hashCode=1b3278a,
array=[107, 45, 116, 101, 115, 116, 82, 101, 112, 108, ..]}}
removed=false valid=true changed=true created=true
value=CacheValue{data=ByteArray{size=19,
array=[118, 45, 116, 101, 115, 116, 82, 101, 112, 108, ..]},
version=281483566645249}]
And here's a log message where the full byte array is shown:
2010-04-14 15:45:00,723 TRACE [ReadCommittedEntry] (Incoming-
2,Infinispan-Cluster,eq-6834) Updating entry
(key=CacheKey{data=ByteArray{size=19, hashCode=6cc2a4,
array=[107, 45, 116, 101, 115, 116, 82, 101, 112, 108, 105, 99, 97, 116,
101, 100, 80, 117, 116]}}
removed=false valid=true changed=true created=true
value=CacheValue{data=ByteArray{size=19,
array=[118, 45, 116, 101, 115, 116, 82, 101, 112, 108, 105, 99, 97, 116,
101, 100, 80, 117, 116]},
version=281483566645249}]
```

バグを報告する

付録G 改訂履歴

改訂 7.0.0-4.1 **Tue Mar 07 2017** **Terry Chuang**
Translation Completed

改訂 7.0.0-4 **Mon 18 Jul 2016** **Rakesh Ghatvisave, Christian Huffman**
JDG-26: GA 向けに JDG 管理コンソールの章を更新。
JGroups SYM_ENCRYPT および ASYM_ENCRYPT プロトコルを含むように更新。

改訂 7.0.0-3 **Wed 27 Apr 2016** **Christian Huffman**
JDG-306: キャッシュストアスキーマを更新。

改訂 7.0.0-2 **Wed 27 Apr 2016** **Christian Huffman**
タイトルを修正。

改訂 7.0.0-1 **Wed 27 Apr 2016** **Rakesh Ghatvisave**
JBoss Data Grid 管理コンソールについての新たな章を追加。

改訂 7.0.0-0 **Tue 19 Apr 2016** **Christian Huffman**
7.0.0 の初回ドラフト。
ガイドのリファクタリングを完了。
Cassandra キャッシュストアを追加。