



# Red Hat JBoss Data Grid 6.5

## 管理および設定ガイド

Red Hat JBoss Data Grid 6.5 向け



# Red Hat JBoss Data Grid 6.5 管理および設定ガイド

---

Red Hat JBoss Data Grid 6.5 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

Copyright © 2015 Red Hat, Inc.

This document is licensed by Red Hat under the [Creative Commons Attribution-ShareAlike 3.0 Unported License](#). If you distribute this document, or a modified version of it, you must provide attribution to Red Hat, Inc. and provide a link to the original. If the document is modified, all Red Hat trademarks must be removed.

Red Hat, as the licensor of this document, waives the right to enforce, and agrees not to assert, Section 4d of CC-BY-SA to the fullest extent permitted by applicable law.

Red Hat, Red Hat Enterprise Linux, the Shadowman logo, JBoss, OpenShift, Fedora, the Infinity logo, and RHCE are trademarks of Red Hat, Inc., registered in the United States and other countries.

Linux ® is the registered trademark of Linus Torvalds in the United States and other countries.

Java ® is a registered trademark of Oracle and/or its affiliates.

XFS ® is a trademark of Silicon Graphics International Corp. or its subsidiaries in the United States and/or other countries.

MySQL ® is a registered trademark of MySQL AB in the United States, the European Union and other countries.

Node.js ® is an official trademark of Joyent. Red Hat Software Collections is not formally related to or endorsed by the official Joyent Node.js open source or commercial project.

The OpenStack ® Word Mark and OpenStack logo are either registered trademarks/service marks or trademarks/service marks of the OpenStack Foundation, in the United States and other countries and are used with the OpenStack Foundation's permission. We are not affiliated with, endorsed or sponsored by the OpenStack Foundation, or the OpenStack community.

All other trademarks are the property of their respective owners.

## 概要

本ガイドでは、Red Hat JBoss Data Grid 6.5 の管理および設定について説明します。

## 目次

<b>第1章 RED HAT JBOSS DATA GRID のセットアップ</b> .....	<b>8</b>
1.1. 前提条件	8
1.2. RED HAT JBOSS DATA GRID のセットアップ手順	8
<b>パート I. キャッシュマネージャーのセットアップ</b> .....	<b>11</b>
<b>第2章 キャッシュマネージャー</b> .....	<b>12</b>
2.1. キャッシュマネージャーの種類	12
2.2. CACHEMANAGERS の作成	12
2.3. 複数のキャッシュマネージャー	13
<b>パート II. JVM メモリー管理のセットアップ</b> .....	<b>15</b>
<b>第3章 エビクションのセットアップ</b> .....	<b>16</b>
3.1. エビクションについて	16
3.2. エビクションストラテジー	16
3.3. エビクションの使用	17
<b>第4章 エクスパレーションのセットアップ</b> .....	<b>20</b>
4.1. エクスパレーションについて	20
4.2. エクスパレーションの操作	20
4.3. エビクションとエクスパレーションの比較	20
4.4. キャッシュエントリーの期限切れ通知	21
4.5. エクスパレーションの設定	21
4.6. 期限つき (MORTAL) データと期限なし (IMMORTAL) データ	22
4.7. エクスパレーションのトラブルシューティング	22
<b>パート III. キャッシュのモニタリング</b> .....	<b>23</b>
<b>第5章 ロギングのセットアップ</b> .....	<b>24</b>
5.1. ロギングについて	24
5.2. 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	24
5.3. ブートロギング	25
5.4. ロギング属性	25
5.5. ロギングの設定例	31
<b>パート IV. キャッシュモードのセットアップ</b> .....	<b>37</b>
<b>第6章 キャッシュモード</b> .....	<b>38</b>
6.1. キャッシュコンテナについて	38
6.2. ローカルモード	39
6.3. クラスターモード	40
6.4. 状態の転送 (STATE TRANSFER)	42
<b>第7章 ディストリビューションモードのセットアップ</b> .....	<b>45</b>
7.1. ディストリビューションモードについて	45
7.2. 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	45
7.3. 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	45
7.4. ディストリビューションモードの戻り値	46
7.5. ディストリビューションモードの設定 (リモートクライアントサーバーモード)	46
7.6. ディストリビューションモードの設定 (ライブラリーモード)	47
7.7. 同期および非同期の分散	48
7.8. ディストリビューションモードにおける GET および PUT の使用	48
<b>第8章 レプリケーションモードのセットアップ</b> .....	<b>50</b>

8.1. レプリケーションモードについて	50
8.2. 最適化されたレプリケーションモードの使用	50
8.3. レプリケーションモードの設定 (リモートクライアントサーバーモード)	50
8.4. レプリケーションモードの設定 (ライブラリーモード)	51
8.5. 同期および非同期のレプリケーション	52
8.6. レプリケーションキュー	53
8.7. レプリケーション保証について	54
8.8. 内部ネットワークのレプリケーショントラフィック	54
<b>第9章 インバリデーションモードのセットアップ</b>	<b>56</b>
9.1. インバリデーションモードについて	56
9.2. インバリデーションモードの設定 (リモートクライアントサーバーモード)	56
9.3. インバリデーションモードの設定 (ライブラリーモード)	57
9.4. 同期的/非同期の無効化	58
9.5.1 次キャッシュと無効化	58
<b>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</b>	<b>59</b>
<b>第10章 非同期 API</b>	<b>60</b>
10.1. 非同期 API の利点	60
10.2. 非同期プロセスについて	60
10.3. 戻り値と非同期 API	61
<b>第11章 REST インターフェース</b>	<b>62</b>
11.1. RUBY クライアントコード	62
11.2. RUBY のサンプルで JSON を使用	62
11.3. PYTHON クライアントコード	63
11.4. JAVA クライアントコード	63
11.5. REST インターフェースコネクター	66
11.6. REST インターフェースの使用	67
11.7. REST インターフェースセキュリティ	72
<b>第12章 MEMCACHED インターフェース</b>	<b>75</b>
12.1. MEMCACHED サーバーについて	75
12.2. MEMCACHED 統計	75
12.3. MEMCACHED インターフェースコネクター	77
12.4. MEMCACHED インターフェースセキュリティ	78
<b>第13章 HOT ROD インターフェース</b>	<b>79</b>
13.1. HOT ROD について	79
13.2. MEMCACHED ではなく HOT ROD を使用する利点	79
13.3. HOT ROD ハッシュ機能	80
13.4. HOT ROD インターフェースコネクター	80
13.5. HOT ROD ヘッダー	82
13.6. HOT ROD 操作	88
13.7. HOT ROD 操作の値	104
13.8. PUT 要求の例	108
13.9. HOT ROD JAVA クライアント	109
13.10. HOT ROD C++ クライアント	113
13.11. HOT ROD C# クライアント	116
13.12. HOT ROD C++ と HOT ROD JAVA クライアント間の相互運用性	122
<b>パート VI. キャッシュのロックのセットアップ</b>	<b>124</b>
<b>第14章 ロック</b>	<b>125</b>

14.1. ロックの設定 (リモートクライアントサーバーモード)	125
14.2. ロックの設定 (ライブラリーモード)	125
14.3. ロックのタイプ	126
14.4. ロック操作	128
<b>第15章 ロックストライピングのセットアップ</b> .....	<b>130</b>
15.1. ロックストライピングについて	130
15.2. 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)	130
15.3. ロックストライピングの設定 (ライブラリーモード)	130
<b>第16章 分離レベルのセットアップ</b> .....	<b>132</b>
16.1. 分離レベルについて	132
16.2. READ_COMMITTED について	132
16.3. REPEATABLE_READ について	132
<b>パート VII. キャッシュストアのセットアップと設定</b> .....	<b>134</b>
<b>第17章 キャッシュストア</b> .....	<b>135</b>
17.1. キャッシュローダーとキャッシュライター	135
17.2. キャッシュストアの設定	135
17.3. 要求キャッシュストア	138
17.4. 接続ファクトリー	139
<b>第18章 キャッシュストアの実装</b> .....	<b>140</b>
18.1. キャッシュストアの比較	140
18.2. キャッシュストア設定の詳細 (ライブラリーモード)	140
18.3. 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)	145
18.4. 単一ファイルキャッシュストア	147
18.5. LEVELDB キャッシュストア	149
18.6. JDBC ベースのキャッシュストア	154
18.7. リモートキャッシュストア	165
18.8. JPA キャッシュストア	166
18.9. カスタムキャッシュストア	169
<b>パート VIII. パッシベーションのセットアップ</b> .....	<b>175</b>
<b>第19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</b> .....	<b>176</b>
19.1. パッシベーションモードの利点	176
19.2. パッシベーションの設定	176
19.3. エビクションとパッシベーション	177
<b>パート IX. キャッシュ書き込みのセットアップ</b> .....	<b>179</b>
<b>第20章 キャッシュ書き込みモード</b> .....	<b>180</b>
20.1. ライトスルーキャッシング	180
20.2. ライトビハインドキャッシング	181
<b>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</b> .....	<b>184</b>
<b>第21章 JAVA MANAGEMENT EXTENSIONS (JMX) のセットアップ</b> .....	<b>185</b>
21.1. JAVA MANAGEMENT EXTENSIONS (JMX) について	185
21.2. RED HAT JBOSS DATA GRID における JMX の使用	185
21.3. JMX 統計レベル	185
21.4. キャッシュインスタンスに対して JMX を有効にする	185
21.5. CACHEMANAGERS に対して JMX を有効にする	186
21.6. 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	186

21.7. 複数の JMX ドメイン	186
21.8. MBEANS	187
<b>第22章 JBOSS OPERATIONS NETWORK (JON) のセットアップ</b>	<b>190</b>
22.1. JBOSS OPERATIONS NETWORK (JON) について	190
22.2. JBOSS OPERATIONS NETWORK (JON) のダウンロード	190
22.3. JBOSS OPERATIONS NETWORK サーバーのインストール	192
22.4. JBOSS OPERATIONS NETWORK エージェント	192
22.5. リモートクライアントサーバーモードの JBOSS OPERATIONS NETWORK	192
22.6. JBOSS OPERATIONS NETWORK リモートクライアントサーバーのプラグイン	193
22.7. ライブラリーモードの JBOSS OPERATIONS NETWORK	200
22.8. JBOSS OPERATIONS NETWORK のプラグインクイックスタート	212
22.9. 他の管理ツールと操作	212
<b>パート XI. コマンドラインツール</b>	<b>214</b>
<b>第23章 RED HAT JBOSS DATA GRID CLI</b>	<b>215</b>
23.1. RED HAT JBOSS DATA GRID ライブラリーモード CLI	215
23.2. RED HAT DATA GRID SERVER CLI	217
23.3. CLI コマンド	217
<b>パート XII. 他の RED HAT JBOSS DATA GRID 機能</b>	<b>225</b>
<b>第24章 1次キャッシュのセットアップ</b>	<b>226</b>
24.1. 1次キャッシュについて	226
24.2. 1次キャッシュの設定	226
<b>第25章 トランザクションのセットアップ</b>	<b>228</b>
25.1. トランザクション	228
25.2. トランザクションの設定	230
25.3. トランザクションリカバリー	232
25.4. デッドロックの検出	234
<b>第26章 JGROUPS の設定</b>	<b>235</b>
26.1. 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)	235
26.2. JGROUPS の設定 (ライブラリーモード)	237
26.3. JGROUPS を使用したマルチキャストのテスト	242
<b>第27章 RED HAT DATA GRID の AMAZON WEB サービスとの使用</b>	<b>246</b>
27.1. S3_PING JGROUPS 検出プロトコル	246
27.2. S3_PING 設定オプション	246
<b>第28章 サーバーヒンティングを用いた高可用性</b>	<b>251</b>
28.1. JGROUPS によるサーバーヒンティングの設定	251
28.2. 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)	251
28.3. サーバーヒンティングの設定 (ライブラリーモード)	252
28.4. CONSISTENTHASHFACTORIES	252
28.5. キーアフィニティーサービス	254
<b>第29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</b>	<b>256</b>
29.1. データセンター間レプリケーションの操作	256
29.2. データセンター間レプリケーションの設定	258
29.3. サイトをオフラインにする	264
29.4. サイト間の状態転送	266
29.5. 複数サイトマスターの設定	269



<b>第30章 セッションの外部化</b> .....	<b>272</b>
30.1. JBOSS EAP 6.X から JBOSS DATA GRID への HTTP セッションの外部化	272
<b>第31章 ネットワークパーティションの処理 (スプリットブ레인)</b> .....	<b>274</b>
31.1. スプリットブレインの検出および回復	274
31.2. 連続してクラッシュしたノードの検出および回復	277
31.3. ネットワークパーティションリカバリの例	277
31.4. パーティション処理の設定	292
<b>付録A JBOSS DATA GRID 向けの推奨 JGROUPS 値</b> .....	<b>294</b>
A.1. サポート対象 JGROUPS プロトコル	294
A.2. TCP のデフォルト値と推奨値	299
A.3. UDP のデフォルト値と推奨値	306
<b>付録B RED HAT JBOSS DATA GRID における JMX MBEANS</b> .....	<b>313</b>
B.1. アクティベーション	313
B.2. CACHE	313
B.3. CACHECONTAINERSTATS	314
B.4. CACHELOADER	316
B.5. CACHEMANAGER	316
B.6. CACHESTORE	318
B.7. CLUSTERCACHESTATS	318
B.8. DEADLOCKDETECTINGLOCKMANAGER	321
B.9. DISTRIBUTIONMANAGER	322
B.10. インタープリター	322
B.11. インバリデーション	323
B.12. LOCKMANAGER	323
B.13. LOCALTOPOLOGYMANAGER	324
B.14. MASSINDEXER	325
B.15. パッシベーション	326
B.16. RECOVERYADMIN	326
B.17. ROLLINGUPGRADEMANAGER	327
B.18. RPCMANAGER	327
B.19. STATETRANSFERMANAGER	329
B.20. 統計	329
B.21. トランザクション	331
B.22. トランスポート	331
B.23. XSITEADMIN	333
<b>付録C 推奨される設定</b> .....	<b>335</b>
C.1. TIMEOUT VALUES	335
<b>付録D パフォーマンスに関する推奨事項</b> .....	<b>336</b>
D.1. 大規模なクラスターの同時起動	336
<b>付録E 参考資料</b> .....	<b>337</b>
E.1. 一貫性について	337
E.2. 一貫性保証について	337
E.3. JBOSS CACHE について	337
E.4. RELAY2 について	337
E.5. 戻り値について	338
E.6. 実行可能インターフェースについて	338
E.7. 2 相コミット (2PC) について	338
E.8. キーバリュエアについて	338
E.9. エクスターナライザー	338

E.10. ハッシュ領域の割り当て	340
付録F 改訂履歴 .....	342



# 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は、事前に設定し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素早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、[パートI「キャッシュマネージャーのセットアップ」](#)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) のメモリー管理をセットアップすることです。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

エントリーがどの程度頻繁に使用されているかに基づき、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きめ細やかな制御を行う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、[3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れる最長期間や、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、[4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、これを強く推奨します。JBoss Data Grid は、ユーザーが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することを可能にする JBoss ロギングを使用します。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の特定に使用することができます。さらに詳しくは、[5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IV「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[16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のにもかかわらず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 2 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、[21章Java Management Extensions \(JMX\) のセットアップ](#)を参照してください。

##### b. 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、[22章 JBoss Operations Network \(JON\) のセットアップ](#)を参照してください。

## 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。サーバーヒントリングは、トポロジー情報を JBoss Data Grid に導入する数ある方法の中の1つです。

### a. サーバーヒントリングのセットアップ

サーバーヒントリングは、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することにより高可用性を提供します。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されたキャッシュなどの場合にはオプションになります。さらに詳しくは、[28章 サーバーヒントリ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して各手順を詳細に説明します。

[バグを報告する](#)

## パート I. キャッシュマネージャーのセットアップ

## 第2章 キャッシュマネージャー

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

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

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

[バグを報告する](#)

### 2.1. キャッシュマネージャーの種類

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

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

[バグを報告する](#)

### 2.2. CACHEMANAGERS の作成

#### 2.2.1. 新しい RemoteCacheManager の作成

##### 手順2.1 新しい RemoteCacheManager の設定

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

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

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

[バグを報告する](#)



### 2.2.2. 新しい組み込みキャッシュマネージャーの作成

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

#### 手順2.2 新しい組み込みキャッシュマネージャーの作成

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

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

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

[バグを報告する](#)

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

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

#### 手順2.3 CDI を使用した新規 `EmbeddedCacheManager` の作成

1. 次のようにデフォルト設定を指定します。

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

2. デフォルトのキャッシュマネージャーを挿入します。

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

[バグを報告する](#)

## 2.3. 複数のキャッシュマネージャー

キャッシュマネージャーは、キャッシュを使用する時の開始点であり、Red Hat JBoss Data Grid で

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

[バグを報告する](#)

### 2.3.1. 単一のキャッシュマネージャーを用いた複数キャッシュの作成

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

[バグを報告する](#)

### 2.3.2. 複数のキャッシュマネージャーの使用

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

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

[バグを報告する](#)

### 2.3.3. 複数のキャッシュマネージャーの作成

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

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

[バグを報告する](#)

## パート II. JVM メモリー管理のセットアップ

## 第3章 エビクションのセットアップ

### 3.1. 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キャッシュストアと残りのクラスターに留まります。

Red Hat JBoss Data Grid は、すでにデータコンテナ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、それらを永続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](#)

### 3.2. 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3.1 エビクションストラテジー

ストラテジー名	操作	説明
<b>EvictionStrategy.NONE</b>	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。
<b>EvictionStrategy.LRU</b>	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
<b>EvictionStrategy.UNORDERED</b>	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に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。
<b>EvictionStrategy.LIRS</b>	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](#)

### 3.2.1. 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、アクセスローカルティイ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ルティイーとは、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、この場合、最も早くアクセスされるエントリーは置き換えられます。このようなケースでは、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](#)

## 3.3. 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自動的に実装されます。

- 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。
- `max-entries/maxEntries`: 指定された値がない場合、**max-entries/maxEntries** 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](#)

### 3.3.1. 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 **max-entries** 属性の値をゼロよりも大きい数に

設定します。**max-entries** 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。**max-entries** 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ため注意してください。

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

### 手順3.1 エビクションの初期化

#### 1. エン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、このステップのサンプルです。

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

#### 3. 最大エントリー数の設定

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。

- a. ライブラリーモードで、**maxEntries** パラメーターを次のように設定します。

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

- b. リモートクライアントモードで、**max-entries** を次のように設定します。

```
<eviction strategy="LRU" max-entries="200" />
```

### 結果

エン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](#)

### 3.3.2. エビクションの設定例

設定 Bean または XML ファイルを使用して Red Hat JBoss Data Grid のエビクションを設定します。エビクションの設定はキャッシュごとに行います。

- ライブラリーモードの XML 設定例は以下のようになります。

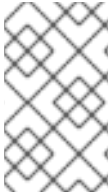
```
<eviction strategy="LRU" maxEntries="2000"/>
```

- リモートクライアントサーバーモードの XML 設定例は以下のようになります。

```
<eviction strategy="LRU" max-entries="20"/>
```

- ライブラリーモードのプログラムを用いた設定例は以下のようになります。

```
Configuration c = new  
ConfigurationBuilder().eviction().strategy(EvictionStrategy.LRU)  
    .maxEntries(2000)  
    .build();
```



### 注記

エビクションの設定では、JBoss Data Grid のライブラリモードは **maxEntries** パラメーター、リモートクライアントサーバーモードは **max-entries** パラメーターを使用します。

[バグを報告する](#)

### 3.3.3. エビクション設定のトラブルシューティング

Red Hat JBoss Data Grid では、キャッシュのサイズを **eviction** 要素の **max-entries** パラメーターに指定された値よりも大きくすることができます。これは、**max-entries** の値を 2 の累乗以外の値に設定することは可能ですが、基礎となるアルゴリズムがこの値を **V** (**max-entries** よりも大きい直近の 2 の累乗値) に変更するからです。エビクションアルゴリズムは、キャッシュコンテナのサイズが **V** の値を超えないようにします。

[バグを報告する](#)

### 3.3.4. エビクションとパッシベーション

エンタリーの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タ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。これは、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4章 エクスパレーションのセットアップ

### 4.1. 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**) として作成されます (それらは期限切れになりません)。逆に、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 **maxIdle** 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、およびクラスターより削除されます。

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、以下の点で異なります。

- 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](#)

### 4.2. 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 (ライブラリーモードでは **maxIdle**、リモートクライアントサーバーモードでは **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](#)

### 4.3. 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。エクスプレッションのライフスパン (**lifespan**) とアイドル時間 (ライブラリーモードでは **maxIdle**、リモートクライアントサーバーモードでは **max-idle**) の値は、各キャッシュエントリーと一緒にレプリケートされます。

[バグを報告する](#)

## 4.4. キャッシュエントリーの期限切れ通知

Red Hat JBoss Data Grid は、タイムアウト直後のエビクションの発生を保証しません。その代わりに、複数のメカニズムを連携して使用し、効率的なエビクション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。
- エビク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エビクト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。エントリーは結果的にエビクションスレッドにより削除されます。

[バグを報告する](#)

## 4.5. エクスプレッションの設定

Red Hat JBoss Data Grid では、エクスプレッ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4.1 エクスプレッションの設定

#### 1. エクスプレッションタグの追加

<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. エクスプレッ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。以下はこのステップの例になります。

```
<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。

- a. ライブラリーモードで、**maxIdle** パラメーターを次のように設定します。

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

- リモートクライアントサーバーモードで、**max-idle** を次のように設定します。

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

## 結果

キャッシュの実装用にエクスプレッションが設定されます。

[バグを報告する](#)

## 4.6. 期限つき (MORTAL) データと期限なし (IMMORTAL) データ

Red Hat JBoss Data Grid では、エンティティを格納するだけでなく、データに期限の情報を添付することが可能です。たとえば、標準的な **put(key, value)** を使用すると、期限なし (immortal) エントリーと呼ばれる永久に期限切れにならないエントリーが作成されます。また、**put(key, value, lifespan, timeunit)** を使用して作成されるエントリーは、指定の固定ライフスパンを持つ期限つき (mortal) エントリーで、ライフスパンの後に期限が切れます。

JBoss Data Grid は **lifespan** パラメーターの他に、エクスプレッションを判断するために使用される **maxIdle** パラメーターも提供します。**maxIdle** パラメーターと **lifespan** パラメーターをさまざまな組み合わせで使用してエントリーのライフスパンを設定することができます。

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

エントリーの期限 (つまり、期限切れの値) は、エントリーのライフスパンと最大アイドル時間の値を設定することにより設定します。設定後、これらの値は、エビクションやパッシベーションの後も保持されるようキャッシュストアで永続化する必要があります。

[バグを報告する](#)

## 4.7. エクスプレッションのトラブルシューティング

エクスプレッ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プレッションについてマークされていても削除されていないことが原因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見えます。たとえば、**number of entries** など JMX の統計を表示する場合、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含まれていることがあります。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、削除しません。このようなエントリーの削除は、以下のように行われます。

- エビクション機能を有効にすると、エビクションスレッドが期限切れエントリーを定期的に出し、これらを消去します。

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ビクションスレッドとして後で削除されます。

[バグを報告する](#)

## パート III. キャッシュのモニタリング

## 第5章 ロギングのセットアップ

### 5.1. ロギングについて

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](#)

### 5.2. 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。

- JBoss ロギングは、Red Hat JBoss Data Grid 6 に含まれています。
- [Apache Commons Logging](#)
- [Simple Logging Facade for Java \(SLF4J\)](#)
- [Apache log4j](#)
- [Java SE Logging \(java.util.logging\)](#)

[バグを報告する](#)

#### 5.2.1. JBoss ロギングについて

JBoss ロギングは JBoss Enterprise Application Platform 6 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 6 は 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がサーバーによってキャプチャーされ、サーバーの設定通りファイルに表示されたり書き込まれたりします。

[バグを報告する](#)

#### 5.2.2. 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サポート。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い作業を行います。
- 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。

[バグを報告する](#)

## 5.3. 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](#)

### 5.3.1. 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](#)

### 5.3.2. 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5.1 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
<b>boot.log</b>	<b>\$JDG_HOME/standalone/log/</b>	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。
<b>server.log</b>	<b>\$JDG_HOME/standalone/log/</b>	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](#)

## 5.4. ロギング属性

### 5.4.1. 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。特定の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、他のログレベルに対して相対的な順序を示す数値が割り当てられています。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](#)

#### 5.4.2. 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5.2 サポート対象のログレベル

ログレベル	値	説明
<b>FINEST</b>	300	-
<b>FINER</b>	400	-
<b>TRACE</b>	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。 <b>TRACE</b> レベルが有効な状態でサーバーが実行されている時に <b>TRACE</b> レベルのログメッセージがキャプチャーされます。

ログレベル	値	説明
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。 <b>DEBUG</b> レベルが有効な状態でサーバーが実行されている時に <b>DEBUG</b> 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 <b>JBoss Data Grid</b> 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

### 5.4.3. 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 사용할 수 있습니다。

[バグを報告する](#)

#### 5.4.4. 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](#)

#### 5.4.5. 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挙動を配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、特定の期間に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](#)

#### 5.4.6. 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5.3 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 ( <b>stdout</b> ) または標準エラー ( <b>stderr</b> ) 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、時間に従っ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に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。
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効です。	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。

ログハンドラーのタイプ	説明	ユースケース
カスタム (Custom)	<p>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、<code>java.util.logging.Handler</code> 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。</p>	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

### 5.4.7. 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、時間に従ってすべてのログエントリを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が一貫している必要のある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

### 5.4.8. 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](#)

## 5.5. ロギングの設定例

### 5.5.1. 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5.1 ルートロガーの設定

1. **level** プロパティを設定します。

**level** プロパティは、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:1.4">
  <root-logger>
    <level name="INFO"/>
  </root-logger>
</subsystem>
```

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:1.4">
  <root-logger>
    <level name="INFO"/>
    <handlers>
      <handler name="CONSOLE"/>
      <handler name="FILE"/>
    </handlers>
  </root-logger>
</subsystem>
```

[バグを報告する](#)

### 5.5.2. 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5.2 ログカテゴリーの設定

```
<subsystem xmlns="urn:jboss:domain:logging:1.4">
  <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** プロパティを設定します。
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

### 5.5.3. 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5.3 コンソールログハンドラーの設定

```
<subsystem xmlns="urn:jboss:domain:logging:1.4">
  <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** プロパティを設定します。

**level** プロパティは、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** プロパティを定義します。

**filter-spec** プロパティ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](#)

#### 5.5.4. 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5.4 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** プロパティを設定します。

**level** プロパティは、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が含まれます。

**relative-to** プロパティは、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** プロパティを設定します。

**append** プロパティを **"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](#)

### 5.5.5. 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5.5 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** プロパティを設定します。

**level** プロパティは、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が含まれます。

**relative-to** プロパティは、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* プロパティを設定します。

*append* プロパティを **"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

### 5.5.6. 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5.6 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* プロパティを設定します。

*level* プロパティは、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が含まれます。

*relative-to* プロパティは、ログファイルが書き込まれるディレクトリです。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/** ディレクトリを指します。

*path* プロパティは、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** プロパティを設定します。

**append** プロパティを **"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](#)

### 5.5.7. 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5.7 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 IV. キャッシュモードのセットアップ

## 第6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小型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、フォールトトレラ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](#)

### 6.1. 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6.1 キャッシュコンテナーの設定方法

```
<subsystem xmlns="urn:infinispan:server:core:6.2"
  default-cache-container="local">
  <cache-container name="local"
    default-cache="default"
    statistics="true"
    listener-executor="infinispan-listener"
    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。

- 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。
- 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。
- 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. **listener-executor** は非同期キャッシュリスナーの通知に使用されるエグゼキューターを定義します。
- e. **start** パラメーターはいつキャッシュコンテナ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 (**eagerly**)」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](#)

## 6.2. 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](#)

### 6.2.1. ローカルモードの設定 (リモートクライアントサーバーモード)

ローカルキャッシュはすべてのキャッシュコンテナ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6.2 local-cache 要素

```
<cache-container name="local"
    default-cache="default"
    statistics="true">
  <local-cache name="default"
    start="EAGER"
    batching="false"
    statistics="true">
```

```

        <indexing index="NONE">
        <property
name="default.directory_provider">ram</property>
        </indexing>
    </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 (**eagerly**)」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。
5. **indexing** パラメーターはローカルキャッシュに使用されるインデックス化のタイプを指定します。このパラメーターの有効な値は **NONE**、**LOCAL**、および **ALL** です。

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で共存できますが、コンテナに **<transport/>** がい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には **<transport/>** がいないため、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](#)

### 6.2.2. ローカルモードの設定 (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、キャッシュの **mode** パラメーターを **local** に設定することは、クラスターリングモードを指定しないことと同じではありません。後者の場合には、キャッシュのキャッシュマネージャーがトランスポートを定義する場合でも、キャッシュはデフォルトでローカルモードに設定されます。

以下のようにクラスターモードをローカルに設定します。

```

<clustering mode="local" />

```

[バグを報告する](#)

## 6.3. 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、クラスターのすべてのキャッシュインスタンスにわたって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に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、クラスターモードに追加設定することが可能です。

[バグを報告する](#)

### 6.3.1. 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](#)

### 6.3.2. キャッシュモードのトラブルシューティング

#### 6.3.2.1. 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](#)

#### 6.3.2.2. 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** プロパティが含まれ、同期通信の場合は **SYNC**、非同期通信の場合は **ASYNC** に値を設定することができます。

#### 例6.1 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](#)

### 6.3.2.3. 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](#)

## 6.4. 状態の転送 (STATE TRANSFER)

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 `numOwners` 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 (整合的な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 `numOwners` 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状態と永続状態の両方が転送されます。ただし、両方の状態は設定で無効にできます。

[バグを報告する](#)

### 6.4.1. 非ブロック状態転送

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](#)

### 6.4.2. 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](#)

### 6.4.3. `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](#)



## 第7章 ディストリビューションモードのセットアップ

### 7.1. 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レランスを実現するため、各エントリーは通常、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](#)

### 7.2. 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。従来の一貫性のあるハッシュとは少し異なりますが、この実装には一貫性のあるハッシュという用語がまだ使われています。

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レラ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（プライマリー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](#)

### 7.3. 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 **num\_copies** 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 (**num\_copies** 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](#)

## 7.4. 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](#)

## 7.5. ディストリビューションモードの設定 (リモートクライアントサーバーモード)

ディストリビューションモードは 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キャッシュコンテナに追加することができます。

### 手順7.1 distributed-cache 要素

```
<cache-container name="clustered"
  default-cache="default"
  statistics="true">
  <transport executor="infinispan-transport" lock-timeout="60000"/>
  <distributed-cache name="default"
    mode="SYNC"
    segments="20"
    start="EAGER"
    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. **statistics** がコンテナ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](#)

## 7.6. ディストリビューションモードの設定 (ライブラリーモード)

次の手順は、Red Hat JBoss Data Grid のライブラリーモードでの分散キャッシュ設定を示しています。

### 手順7.2 分散キャッシュの設定

#### 1. クラスタリング要素の設定

```
<clustering mode="distribution">
  <sync replTimeout="${TIME}" />
  <stateTransfer chunkSize="${SIZE}"
    fetchInMemoryState="{true/false}"
    awaitInitialTransfer="{true/false}"
    timeout="${TIME}" />
```

- a. **clustering** 要素の **mode** パラメーター値は、キャッシュに選択されたクラスタリングモードを決定します。
- b. **sync** 要素の **replTimeout** パラメーターは、リモート呼び出し後の確認に設定される最大の時間範囲 (ミリ秒単位) を指定します。この時間範囲が確認なしに終了する場合、例外がスローされます。
- c. **stateTransfer** 要素は、ノードがクラスターを出るか、またはクラスターに参加する際に状態がどのように転送されるかを指定します。これは以下のパラメーターを使用します。
  - i. **chunkSize** パラメーターは、1つの塊で転送されるキャッシュエントリーの数です。デフォルトの **chunkSize** 値は 512 です。 **chunkSize** は多くのパラメーター (エントリーのサイズなど) に基づき、最良の設定を探す必要があります。大きいキャッシュエントリーの値を変更するには、小さい塊が推奨されます。小さいキャッシュエントリーの場合は、 **chunkSize** を増やすと、パフォーマンスが向上します。
  - ii. **fetchInMemoryState** パラメーターは、**true** に設定された場合、起動時に近接キャッシュから状態の情報を要求します。これは、キャッシュの起動時間に影響を与えます。キャッシュの不整合を回避する共有キャッシュストアがない場合は、デフォルト値を **true** に設定します。
  - iii. **awaitInitialTransfer** パラメーターにより、参加者ノードでのメソッド **CacheManager.getCache()** の最初の呼び出しがブロックされ、参加が完了し、キャッシュが近接キャッシュからの状態の受信を完了するまで待機します (**fetchInMemoryState** が有効な場合)。このオプションは、分散キャッシュとレプリケートされたキャッシュにのみ適用され、デフォルトで有効になります。

- iv. **timeout** パラメーターは、要求された状態の近接キャッシュからの応答をキャッシュが待機する最大時間(ミリ秒単位)を指定します。**timeout** 期間内に応答が受信されない場合、起動プロセスは中止され、例外がスローされます。状態転送に失敗した結果、キャッシュはインスタンスで利用できません。

## 2. トランスポート設定の指定

```
<global>
  <transport clusterName="${NAME}"
    distributedSyncTimeout="${TIME}"
    transportClass="${CLASS}" />
</global>
```

**transport** 要素は、以下のようにキャッシュコンテナのトランスポート設定を定義します。

- a. **clusterName** パラメーターはクラスターの名前を指定します。ノードは同じ名前を共有するクラスターのみに接続できます。
- b. **distributedSyncTimeout** パラメーターは、分散ロック上でロックを取得するために待機する時間を指定します。この分散ロックにより、単一キャッシュは一度に状態を転送するか、または状態をリハッシュすることができます。
- c. **transportClass** パラメーターは、キャッシュコンテナのネットワークトランスポートを表すクラスを指定します。

[バグを報告する](#)

## 7.7. 同期および非同期の分散

ディストリビューションモードは同期通信のみをサポートします。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7.1 通信モードの例

たとえば、**A**、**B**、**C** という 3 つのキャッシュがクラスターにあり、キャッシュ **A** を **B** にマップする **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、操作が最初にキャッシュ **A** と **B** の両方に呼び出しを同期転送し、キャッシュ **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、操作が想定どおり動作しても戻り値の有用性は保証されません。

[バグを報告する](#)

## 7.8. ディストリビューションモードにおける GET および PUT の使用

ディストリビューションモードでは、**write** コマンドの前にキャッシュがリモートの **GET** コマンドを実行します。これは、**java.util.Map** コントラクトに従って指定されたキーに関連する以前の値を返すメソッド (**Cache.put()**) があるからです。これがキーを所有しないインスタンスで実行され、エントリーが 1 次キャッシュで見つからない場合、**PUT** の前にリモートの **GET** を実行することが、戻り値を取得するための信頼できる唯一の方法になります。

Red Hat JBoss Data Grid は戻り値を待たなければならないため、キャッシュが同期または非同期であるかに関わらず、**PUT** 操作の前に発生する **GET** 操作は常に同期になります。

[バグを報告する](#)

### 7.8.1. 分散された **GET** および **PUT** 操作

ディストリビューションモードでは、必要な **PUT** 操作を実行する前に、キャッシュが **GET** 操作を実行することがあります。

この操作は、リソースの面では非常にコストのかかる操作になります。リモート **GET** 操作は同期であるにも関わらず、すべての応答を待たないため、無駄になるリソースが発生します。**GET** 処理は最初に受信する有効な応答を許可するため、パフォーマンスとクラスターの大きさとの関連性はありません。

ご使用の実装で戻り値が必要でない場合、呼び出し毎の設定に対する **Flag.SKIP\_REMOTE\_LOOKUP** フラグを使用します。

このような動作は、キャッシュの操作や全パブリックメソッドの正確な機能を害することはありませんが、**java.util.Map** インターフェースコントラクトに違反します。これは、信頼できず正確でない戻り値が特定のメソッドに提供されるため、コントラクトに違反することになります。そのため、必ずこれらの戻り値が設定上重要な目的に使用されないようにしてください。

[バグを報告する](#)

## 第8章 レプリケーションモードのセットアップ

### 8.1. 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](#)

### 8.2. 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](#)

### 8.3. レプリケーションモードの設定 (リモートクライアントサーバーモード)

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キャッシュコンテナに追加することができます。

#### 手順8.1 *replicated-cache* 要素

```
<cache-container name="clustered"
  default-cache="default"
  statistics="true">
  <transport executor="infinispan-transport"
    lock-timeout="60000"/>
  <replicated-cache name="default"
    mode="SYNC"
    start="EAGER"
    statistics="true">
    <transaction mode="NONE" />
  </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。
5. **transaction** 要素は、レプリケートされたキャッシュのトランザクションモードをセットアップします。

**重要**

リモートクライアントサーバーモードでは、JBoss Data Grid が互換モードで使われ、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。

**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](#)

## 8.4. レプリケーションモードの設定 (ライブラリーモード)

以下の手順は、Red Hat JBoss Data Grid のライブラリーモードでのレプリケーションモードの設定を示しています。

### 手順8.2 レプリケーションモードの設定

#### 1. クラスタリング要素の設定

```
<clustering mode="replication">
  <sync replTimeout="${TIME}" />
  <stateTransfer chunkSize="${SIZE}"
    fetchInMemoryState="{true/false}"
    awaitInitialTransfer="{true/false}"
    timeout="${TIME}" />
```

- a. **clustering** 要素の **mode** パラメーター値は、キャッシュに選択されたクラスタリングモードを決定します。

- b. **sync** 要素の **replTimeout** パラメーターは、リモート呼び出し後の確認に設定される最大の時間範囲 (ミリ秒単位) を指定します。この時間範囲が確認なしに終了する場合、例外がスローされます。
- c. **stateTransfer** 要素は、ノードがクラスターを脱退するか、またはクラスターに参加する際に状態がどのように転送されるかを指定します。これは以下のパラメーターを使用します。
  - i. **chunkSize** パラメーターは、1つの塊で転送されるキャッシュエントリーの数です。デフォルトの **chunkSize** 値は 512 です。 **chunkSize** は多くのパラメーター (エントリーのサイズなど) に基づき、最良の設定を探す必要があります。大きいキャッシュエントリーの値を変更するには、小さい塊が推奨されます。小さいキャッシュエントリーの場合は、 **chunkSize** を増やすと、パフォーマンスが向上します。
  - ii. **fetchInMemoryState** パラメーターは、 **true** に設定された場合、起動時に近接キャッシュから状態の情報を要求します。これは、キャッシュの起動時間に影響を与えます。キャッシュの不整合を回避する共有キャッシュストアがない場合は、デフォルト値を **true** に設定します。
  - iii. **awaitInitialTransfer** パラメーターにより、参加者ノードでのメソッド **CacheManager.getCache()** の最初の呼び出しがブロックされ、参加が完了し、キャッシュが近接キャッシュからの状態の受信を完了するまで待機します (**fetchInMemoryState** が有効な場合)。このオプションは、分散キャッシュとレプリケートされたキャッシュにのみ適用され、デフォルトで有効になります。
  - iv. **timeout** パラメーターは、要求された状態の近接キャッシュからの応答をキャッシュが待機する最大時間 (ミリ秒単位) を指定します。 **timeout** 期間内に応答が受信されない場合、起動プロセスは中止され、例外がスローされます。状態転送に失敗した結果、キャッシュはインスタンスで利用できません。

## 2. トランスポート設定の指定

```
<global>
  <transport clusterName="${NAME}"
    distributedSyncTimeout="${TIME}"
    transportClass="${CLASS}" />
</global>
```

**transport** 要素は、以下のようにキャッシュコンテナのトランスポート設定を定義します。

- a. **clusterName** パラメーターはクラスターの名前を指定します。ノードは同じ名前を共有するクラスターのみに接続できます。
- b. **distributedSyncTimeout** パラメーターは、分散ロック上でロックを取得するために待機する時間を指定します。この分散ロックにより、単一キャッシュは一度に状態を転送するか、または状態をリハッシュすることができます。
- c. **transportClass** パラメーターは、キャッシュコンテナのネットワークトランスポートを表すクラスを指定します。

[バグを報告する](#)

## 8.5. 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

### 8.5.1. 非同期レプリケーションの挙動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つことがあります。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。状態転送を想定通りに動作させるには、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

### 8.6. レプリケーションキュー

レプリケーションモードでは、以下を基にして、Red Hat JBoss Data Grid は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

### 8.6.1. 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 executor** に対して **1** に設定します。**transport executor** は次のように **standalone.xml** または **clustered.xml** で定義されます。

```
<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8.1 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](#)

## 8.7. 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取得することができます。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](#)

## 8.8. 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9章 インバリデーションモードのセットアップ

### 9.1. 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、状態が必要となる度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](#)

### 9.2. インバリデーションモードの設定 (リモートクライアントサーバーモード)

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キャッシュコンテナに追加することができます。

#### 手順9.1 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](#)

### 9.3. インバリデーションモードの設定 (ライブラリーモード)

次の手順は、Red Hat JBoss Data Grid のライブラリーモードにおけるインバリデーションモードのキャッシュ設定を示しています。

#### 手順9.2 インバリデーションモードの設定

##### 1. クラスタリング要素の設定

```
<clustering mode="invalidation">
  <sync replTimeout="${TIME}" />
  <stateTransfer chunkSize="${SIZE}"
    fetchInMemoryState="{true/false}"
    awaitInitialTransfer="{true/false}"
    timeout="${TIME}" />
```

- a. **clustering** 要素の **mode** パラメーター値は、キャッシュに選択されたクラスタリングモードを決定します。
- b. **sync** 要素の **replTimeout** パラメーターは、リモート呼び出し後の確認に設定される最大の時間範囲 (ミリ秒単位) を指定します。この時間範囲が確認なしに終了する場合、例外がスローされます。
- c. **stateTransfer** 要素は、ノードがクラスターを出るか、またはクラスターに参加する際に状態がどのように転送されるかを指定します。これは以下のパラメーターを使用します。
  - i. **chunkSize** パラメーターは、転送するキャッシュエントリーの状態バッチのサイズを指定します。この値が 0 より大きい場合、設定される値は送信されるチャンクのサイズになります。値が 0 より小さい場合、すべての状態は同時に転送されます。
  - ii. **true** に設定される **fetchInMemoryState** パラメーターは、起動時に隣接したキャッシュから状態についての情報を要求します。これは、キャッシュの起動時間に影響を与えます。
  - iii. **awaitInitialTransfer** パラメーターにより、参加者ノードでのメソッド **CacheManager.getCache()** の最初の呼び出しがブロックされ、参加が完了し、キャッシュが近接キャッシュからの状態の受信を完了するまで待機します (**fetchInMemoryState** が有効な場合)。このオプションは、分散キャッシュとレプリケートされたキャッシュにのみ適用され、デフォルトで有効になります。
  - iv. **timeout** パラメーターは、要求された状態の近接キャッシュからの応答をキャッシュが待機する最大時間 (ミリ秒単位) を指定します。**timeout** の時間内に応答を受け取らない場合は、起動プロセスが中止され、例外がスローされます。

##### 2. トランスポート設定の指定

```
<global>
  <transport clusterName="${NAME}"
    distributedSyncTimeout="${TIME}"
```

```
        transportClass="${CLASS}" />  
</global>
```

**transport** 要素は、以下のようにキャッシュコンテナのトランスポート設定を定義します。

- a. **clusterName** パラメーターはクラスターの名前を指定します。ノードは同じ名前を共有するクラスターのみに接続できます。
- b. **distributedSyncTimeout** パラメーターは、分散ロック上でロックを取得するために待機する時間を指定します。この分散ロックにより、単一キャッシュは一度に状態を転送するか、または状態をリハッシュすることができます。
- c. **transportClass** パラメーターは、キャッシュコンテナのネットワークトランスポートを表すクラスを指定します。

[バグを報告する](#)

## 9.4. 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操作します。

- 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** モードで操作します。

[バグを報告する](#)

### 9.5.1 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度に生成されます。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](#)

## 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10章 非同期 API

Red Hat JBoss Data Grid は同期 API メソッドの他に、非ブロッキング方式で同じ機能を実現する非同期 API も提供します。

非同期メソッドの命名規則は、同期メソッドの命名規則と似ていますが、各メソッド名に **Async** を追加します。非同期メソッドは、操作の結果が含まれる **Future** を返します。

たとえば、**Cache(String, String)** とパラメーター化されたキャッシュでは、**Cache.put(String key, String value)** は **String** を返します。また、**Cache.putAsync(String key, String value)** は **Future(String)** を返します。

[バグを報告する](#)

### 10.1. 非同期 API の利点

非同期 API はブロックしないため、以下のような複数の利点があります。

- 同期通信が保証される (エラーと例外を処理する機能が追加される)。
- 呼び出しが完了するまでスレッドの操作をブロックする必要がない。

これらの利点により、以下のようにシステムで並列処理を向上させることができます。

#### 例10.1 非同期 API の使用

```
Set<Future<?>> futures = new HashSet<Future<?>>();
futures.add(cache.putAsync("key1", "value1"));
futures.add(cache.putAsync("key2", "value2"));
futures.add(cache.putAsync("key3", "value3"));
```

たとえば、以下の行は実行時にスレッドをブロックしません。

- **futures.add(cache.putAsync(key1, value1));**
- **futures.add(cache.putAsync(key2, value2));**
- **futures.add(cache.putAsync(key3, value3));**

これら 3 つの **put** 操作からのリモートコールは同時に実行されます。これは、分散モードで実行する場合に特に役に立ちます。

[バグを報告する](#)

### 10.2. 非同期プロセスについて

Red Hat JBoss Data Grid の一般的な書き込み操作では、以下のプロセスがクリティカルパスで失敗し、リソースが最も必要なものから必要でないものに順序付けされます。

- ネットワークコール
- マーシャリング



- キャッシュストアへの書き込み (オプション)
- ロック

JBoss Data Grid では、非同期メソッドを使用すると、クリティカルパスからネットワークコールとマーシャリングが削除されます。

[バグを報告する](#)

### 10.3. 戻り値と非同期 API

非同期 API が Red Hat JBoss Data Grid で使用された場合、クライアントコードでは以前の値を問い合わせるために非同期操作が **Future** または **NotifyingFuture** を返す必要があります。



#### 注記

**NotifyingFutures** は、JBoss Data Grid ライブラリーモードでのみ利用できます。

非同期操作の結果を取得するには、次の操作を呼び出します。この操作は呼び出されたときにスレッドをブロックします。

```
Future.get()
```

[バグを報告する](#)

## 第11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 コールを認識し、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。

[バグを報告する](#)

### 11.1. RUBY クライアントコード

以下のコードは ruby を使用して Red Hat JBoss Data Grid REST API と対話する例です。提供されたコードは特別なライブラリーを必要とせず、標準的な net/HTTP ライブラリーで十分です。

#### 例11.1 Ruby での REST API の使用

```
require 'net/http'

http = Net::HTTP.new('localhost', 8080)

#An example of how to create a new entry

http.post('/rest/MyData/MyKey', 'DATA_HERE', {"Content-Type" =>
"text/plain"})

#An example of using a GET operation to retrieve the key

puts http.get('/rest/MyData/MyKey').body

#An Example of using a PUT operation to overwrite the key

http.put('/rest/MyData/MyKey', 'MORE DATA', {"Content-Type" =>
"text/plain"})

#An example of Removing the remote copy of the key

http.delete('/rest/MyData/MyKey')

#An example of creating binary data

http.put('/rest/MyImages/Image.png',
File.read('/Users/michaelneale/logo.png'), {"Content-Type" =>
"image/png"})
```

[バグを報告する](#)

### 11.2. RUBY のサンプルで JSON を使用

前提条件

ruby で JavaScript Object Notation (JSON) を使用して Red Hat JBoss Data Grid の REST インターフェースと対話するために、JSON Ruby をインストールし (プラットフォームのパッケージマネージャーに問い合わせるか、Ruby ドキュメンテーションを参照)、以下のコードを使用して要件を宣言します。

```
require 'json'
```

### Ruby で JSON を使用

以下のコードは、Ruby で JavaScript Object Notation (JSON) と **PUT** 関数を使用して特定のデータ (この場合は、個人の名前と年齢) を送信する例です。

```
data = {:name => "michael", :age => 42 }
http.put('/rest/Users/data/0', data.to_json, {"Content-Type" =>
"application/json"})
```

[バグを報告する](#)

## 11.3. PYTHON クライアントコード

以下のコードは Python を使用して Red Hat JBoss Data Grid REST API と対話する例です。提供されたコードは、標準的な HTTP ライブラリーのみを必要とします。

### 例11.2 Python での REST API の使用

```
import urllib

#How to insert data

conn = urllib.HTTPConnection("localhost:8080")
data = "SOME DATA HERE \!" #could be string, or a file...
conn.request("POST", "/rest/Bucket/0", data, {"Content-Type":
"text/plain"})
response = conn.getresponse()
print response.status

#How to retrieve data

import urllib
conn = urllib.HTTPConnection("localhost:8080")
conn.request("GET", "/rest/Bucket/0")
response = conn.getresponse()
print response.status
print response.read()
```

[バグを報告する](#)

## 11.4. JAVA クライアントコード

以下のコードは Java を使用して Red Hat JBoss Data Grid REST API と対話する例です。

### 例11.3 インポートの定義

```
import java.io.BufferedReader;import java.io.IOException;
import java.io.InputStreamReader;import java.io.OutputStreamWriter;
import java.net.HttpURLConnection;import java.net.URL;
```

#### 例11.4 文字列値をキャッシュに追加

```
public class RestExample {

    /**
     * Method that puts a String value in cache.
     * @param urlServerAddress
     * @param value
     * @throws IOException
     */

    public void putMethod(String urlServerAddress, String value) throws
    IOException {
        System.out.println("-----");
        System.out.println("Executing PUT");
        System.out.println("-----");
        URL address = new URL(urlServerAddress);
        System.out.println("executing request " + urlServerAddress);
        HttpURLConnection connection = (HttpURLConnection)
        address.openConnection();
        System.out.println("Executing put method of value: " + value);
        connection.setRequestMethod("PUT");
        connection.setRequestProperty("Content-Type", "text/plain");
        connection.setDoOutput(true);

        OutputStreamWriter outputStreamWriter = new
        OutputStreamWriter(connection.getOutputStream());
        outputStreamWriter.write(value);

        connection.connect();
        outputStreamWriter.flush();

        System.out.println("-----");
        System.out.println(connection.getResponseCode() + " " +
        connection.getResponseMessage());
        System.out.println("-----");

        connection.disconnect();
    }
}
```

以下のコードは、JBoss Data Grid REST インターフェースと対話するために Java を使用して URL に指定された値を読み取るメソッドの例です。

#### 例11.5 キャッシュから文字列値を取得

```
/**
 * Method that gets an value by a key in url as param value.
 * @param urlServerAddress
```

```

    * @return String value
    * @throws IOException
    */
    public String getMethod(String urlServerAddress) throws IOException {
        String line = new String();
        StringBuilder stringBuilder = new StringBuilder();

        System.out.println("-----");
        System.out.println("Executing GET");
        System.out.println("-----");

        URL address = new URL(urlServerAddress);
        System.out.println("executing request " + urlServerAddress);

        HttpURLConnection connection = (HttpURLConnection)
address.openConnection();
        connection.setRequestMethod("GET");
        connection.setRequestProperty("Content-Type", "text/plain");
        connection.setDoOutput(true);

        BufferedReader bufferedReader = new BufferedReader(new
InputStreamReader(connection.getInputStream()));

        connection.connect();

        while ((line = bufferedReader.readLine()) != null) {
            stringBuilder.append(line + '\n');
        }

        System.out.println("Executing get method of value: " +
stringBuilder.toString());

        System.out.println("-----");
        System.out.println(connection.getResponseCode() + " " +
connection.getResponseMessage());
        System.out.println("-----");

        connection.disconnect();

        return stringBuilder.toString();
    }

```

### 例11.6 Java Main メソッドの使用

```

/**
 * Main method example.
 * @param args
 * @throws IOException
 */
public static void main(String[] args) throws IOException {
    //Note that the cache name is "cacheX"
    RestExample restExample = new RestExample();
    restExample.putMethod("http://localhost:8080/rest/cacheX/1", "Infinispan
REST Test");
}

```

```
restExample.getMethod("http://localhost:8080/rest/cacheX/1");
    }
}
```

[バグを報告する](#)

## 11.5. REST インターフェースコネクター

Red Hat JBoss Data Grid は、以下の 3 つのコネクタータイプをサポートします。

- Hot Rod ベースコネクターの設定を定義する **hotrod-connector** 要素。
- memcached ベースコネクターの設定を定義する **memcached-connector** 要素。
- REST インターフェースベースのコネクターの設定を定義する **rest-connector** 要素。

コネクターの宣言により、**<socket-binding-group />** 内で宣言されたソケットバインディングを使用し、**local** コンテナで宣言されたキャッシュを公開し、他のすべての設定でデフォルト値を使用して Hot Rod、Memcached、または REST サーバーが有効になります。以下の例は、Hot Rod、Memcached、および REST サーバーに接続する方法を示しています。

REST コネクターは、Web サブシステムを必要とするため、Hot Rod と Memcached 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。以下により、REST サーバーが有効になります。

```
<rest-connector virtual-server="default-host"
  cache-container="local"
  security-domain="other"
  auth-method="BASIC"/>
```

詳細については、「[REST インターフェースの使用](#)」を参照してください。

[バグを報告する](#)

### 11.5.1. 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11.1 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:6.1">
  <rest-connector virtual-server="default-host"
    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. **virtual-server** パラメーターは、REST コネクターで使用する仮想サーバーを指定します。このパラメーターのデフォルト値は **default-host** です。これはオプションパラメーターです。
2. **cache-container** パラメーターは、REST コネクターで使用するキャッシュコンテナを指定します。これは必須パラメーターです。
3. **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。
4. **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。
5. **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。
6. **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](#)

## 11.6. REST インターフェースの使用

REST インターフェースを 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して、以下の操作を実行できます。

- データの追加
- データの取得
- データの削除

[バグを報告する](#)

### 11.6.1. REST を使用したデータの追加

Red Hat JBoss Data Grid の REST インターフェースで、以下のメソッドを使用してキャッシュにデータを追加します。

- HTTP **PUT** メソッド
- HTTP **POST** メソッド

**PUT** メソッドと **POST** メソッドが使用される場合、要求の本文には、ユーザーにより追加された情報を含むこのデータが含まれます。

**PUT** メソッドと **POST** メソッドの両方には、**Content-Type** ヘッダーが必要です。

[バグを報告する](#)

### 11.6.1.1. PUT `/{{cacheName}}/{{cacheKey}}` について

提供された URL フォームからの **PUT** 要求により、提供されたキーを使用して要求本文からのペイロードがターゲットキャッシュに配置されます。このタスクが正常に完了するには、ターゲットキャッシュがサーバーに存在する必要があります。

例として、以下の URL では、値 **hr** がキャッシュ名であり、**payRo11%2F3** がキーです。値 **%2F** は、**/** がキーで使用されたことを示します。

```
http://someserver/rest/hr/payRo11%2F3
```

既存のデータは置き換えられ、更新が必要な場合は **Time-To-Live** 値と **Last-Modified** 値が更新されます。



#### 注記

以下の引数を使用してサーバーが起動された場合は、キーの **/** を表す値 **%2F** を含むキャッシュキー (提供された例を参照) を正常に実行できます。

```
-Dorg.apache.tomcat.util.buf.UDecoder.ALLOW_ENCODED_SLASH=true
```

[バグを報告する](#)

### 11.6.1.2. POST `/{{cacheName}}/{{cacheKey}}` について

提供された URL フォームからの **POST** メソッドにより、提供されたキーを使用して (要求本文からの) ペイロードがターゲットキャッシュに配置されます。ただし、**POST** メソッドでは、値がキャッシュ/キーに存在する場合に、**HTTP CONFLICT** ステータスが返され、内容が更新されません。

[バグを報告する](#)

## 11.6.2. REST を使用したデータの取得

Red Hat JBoss Data Grid の REST インターフェースで、以下のメソッドを使用してキャッシュからデータを取得します。

- **HTTP GET** メソッド。
- **HTTP HEAD** メソッド。

[バグを報告する](#)

### 11.6.2.1. GET `/{{cacheName}}/{{cacheKey}}` について

**GET** メソッドは、応答の本文として、提供された **cacheName** に存在し、関連するキーに一致するデータを返します。**Content-Type** ヘッダーは、データのタイプを提供します。ブラウザーはキャッシュに直接アクセスできます。

各エントリーに対して、要求された URL でデータの状態を示す **Last-Modified** ヘッダーとともに一意のエンティティタグ (**ETag**) が返されます。**ETag** により、ブラウザー (および他のクライアント) は、(帯域幅を節約するために) データが変更された場合のみデータを要求できます。**ETag** は、HTTP 標準の一部であり、Red Hat JBoss Data Grid によりサポートされます。



格納されたコンテンツのタイプは、返されたタイプです。例として、文字列が格納された場合は、文字列が返されます。シリアル化された形式で格納されたオブジェクトは、手動でシリアル化解除する必要があります。

[バグを報告する](#)

#### 11.6.2.2. HEAD /{cacheName}/{cacheKey} について

**HEAD** メソッドは、**GET** メソッドと同様に動作しますが、コンテンツを返しません (ヘッダーフィールドが返されます)。

[バグを報告する](#)

#### 11.6.3. REST を使用したデータの削除

REST インターフェースを使用して Red Hat JBoss Data Grid からデータを削除するには、HTTP **DELETE** メソッドを使用してキャッシュからデータを取得します。**DELETE** メソッドは以下のことを行えます。

- キャッシュエントリ/値を削除します。(DELETE /{cacheName}/{cacheKey})
- キャッシュからすべてのエントリを削除します。(DELETE /{cacheName})

[バグを報告する](#)

##### 11.6.3.1. DELETE /{cacheName}/{cacheKey} について

このコンテキスト (DELETE /{cacheName}/{cacheKey}) で使用された場合、**DELETE** メソッドは提供されたキーのキャッシュからキー/値を削除します。

[バグを報告する](#)

##### 11.6.3.2. DELETE /{cacheName} について

このコンテキスト (DELETE /{cacheName}) では、**DELETE** メソッドが名前付きキャッシュ内のすべてのエントリを削除します。正常な **DELETE** 操作後に、HTTP ステータスコード **200** が返されます。

[バグを報告する](#)

##### 11.6.3.3. バックグラウンド削除操作

**performAsync** ヘッダーの値を **true** に設定して、削除操作がバックグラウンドで続行される状態で値がすぐに返されるようにします。

[バグを報告する](#)

#### 11.6.4. REST インターフェース操作ヘッダー

以下の表は、Red Hat JBoss Data Grid REST インターフェースに含まれるヘッダーを示しています。

表11.1 ヘッダータイプ

ヘッダー	必須/オプション	値	デフォルト値	説明
Content-Type	必須	-	-	Content-Type が <b>application/x-java-serialized-object</b> に設定された場合は、Java オブジェクトとして格納されます。
performAsync	任意	true/false	-	<b>true</b> に設定された場合は、すぐに返され、独自にクラスターにデータがレプリケートされます。この機能は、大量のデータ挿入と大きいクラスターを取り扱う場合に役に立ちます。
timeToLiveSeconds	任意	数値 (正の値および負の値)	<b>-1</b> (この値により、 <b>timeToLiveSeconds</b> の直接的な結果としてエクスパレーションが回避されます。このデフォルト値よりも、他の場所で設定されたエクスパレーションの値が優先されます。)	該当するエントリーが自動的に削除されるまでの秒数を反映します。 <b>timeToLiveSeconds</b> に負の値を設定すると、デフォルト値と同じ結果が提供されます。
maxIdleTimeSeconds	任意	数値 (正の値および負の値)	<b>-1</b> (この値により、 <b>maxIdleTimeSeconds</b> の直接的な結果としてエクスパレーションが回避されます。このデフォルト値よりも、他の場所で設定されたエクスパレーションの値が優先されます。)	エントリーが自動的に削除される場合の、最後の使用時以降の秒数を含みます。負の値を渡すと、デフォルト値と同じ結果が提供されます。

**timeToLiveSeconds** ヘッダーと **maxIdleTimeSeconds** ヘッダーには以下の組み合わせを設定できます。

- **timeToLiveSeconds** ヘッダーと **maxIdleTimeSeconds** ヘッダーに値 **0** が割り当てられた場合、キャッシュは、XML を使用するか、またはプログラミングにより設定されたデフォルトの **timeToLiveSeconds** 値と **maxIdleTimeSeconds** 値を使用します。
- **maxIdleTimeSeconds** ヘッダー値のみが **0** に設定された場合は、**timeToLiveSeconds** 値をパラメーター (または、デフォルトの **-1** (パラメーターが存在しない場合)) として渡す必要があります。また、**maxIdleTimeSeconds** パラメーター値は、XML を使用するか、プログラミングにより、設定された値にデフォルトで設定されます。
- **timeToLiveSeconds** ヘッダー値のみが **0** に設定された場合は、エクスレーションが即座に発生し、**maxIdleTimeSeconds** 値がパラメーターとして渡された値に設定されます (パラメーターが提供されなかった場合はデフォルトの **-1**)。

## ETag ベースのヘッダー

各 REST インターフェースエントリーに対して、提供された URL でデータの状態を示す **Last-Modified** ヘッダーとともに、ETags (エンティティータグ) が返されます。ETags は、帯域幅を節約するためにデータが変更された場合にのみ、データを要求する HTTP 操作で使用されます。以下のヘッダーは、ETags (エンティティータグ) ベースの楽観的ロックをサポートします。

表11.2 エンティティータグ関連ヘッダー

ヘッダー	アルゴリズム	例	説明
If-Match	If-Match = "If-Match" ":" ( "*"   1#entity-tag )	-	(リソースから以前に取得された) 指定されたエンティティータグが最新であることを確認するために、関連するエンティティータグのリストとともに使用されます。
If-None-Match		-	(リソースから以前に取得された) 指定されたエンティティータグが最新でないことを確認するために、関連するエンティティータグのリストとともに使用されます。この機能により、必要なときに、最小のトランザクションオーバーヘッドで、キャッシュされた情報が効率的に更新されます。

ヘッダー	アルゴリズム	例	説明
If-Modified-Since	If-Modified-Since = "If-Modified-Since" ":" HTTP-date	If-Modified-Since: Sat, 29 Oct 1994 19:43:31 GMT	要求されたバリエーションの最終変更日時と、提供された時間および日付の値とを比較します。指定された日時以降に要求されたバリエーションが変更されなかった場合は、エンティティの代わりに <b>304 (未変更)</b> 応答がメッセージ本文なしで返されます。
If-Unmodified-Since	If-Unmodified-Since = "If-Unmodified-Since" ":" HTTP-date	If-Unmodified-Since: Sat, 29 Oct 1994 19:43:31 GMT	要求されたバリエーションの最終変更日時と、提供された時間および日付の値とを比較します。指定された日時以降に要求されたリソースが変更されなかった場合は、指定された操作が実行されます。指定された日時以降に要求されたリソースが変更された場合は、操作が実行されず、エンティティの代わりに <b>412 (事前条件失敗)</b> 応答が返されます。

[バグを報告する](#)

## 11.7. REST インターフェースセキュリティー

### 11.7.1. 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](#)

### 11.7.2. 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11.2 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:6.1">
    <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 6 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. 作成されたユーザーのロールを尋ねられたら、**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](#)

## 第12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、Memcached 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されます。

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](#)

### 12.1. 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。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](#)

### 12.2. 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12.1 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](#)

## 12.3. MEMCACHED インターフェースコネクタ

Red Hat JBoss Data Grid は、以下の 3 つのコネクタタイプをサポートします。

- Hot Rod ベースコネクタの設定を定義する **hotrod-connector** 要素。
- memcached ベースコネクタの設定を定義する **memcached-connector** 要素。
- REST インターフェースベースのコネクタの設定を定義する **rest-connector** 要素。

コネクタの宣言により、**<socket-binding-group />** 内で宣言されたソケットバインディングを使用し、**local** コンテナで宣言されたキャッシュを公開し、他のすべての設定でデフォルト値を使用して Hot Rod、Memcached、または REST サーバーが有効になります。以下の例は、Hot Rod、Memcached、および REST サーバーに接続する方法を示しています。

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** コネクタを宣言します。「[Memcached コネクタの設定](#)」を参照してください。

[バグを報告する](#)

### 12.3.1. Memcached コネクタの設定

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** コネクタを設定するために使用する属性を示しています。

#### 手順12.1 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:6.1">
  <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](#)

## 12.4. MEMCACHED インターフェースセキュリティー

### 12.4.1. 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="memcached"
  interface="public"
  port="11211" />

```

[バグを報告する](#)

## 第13章 HOT ROD インターフェース

### 13.1. 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、トポロジが変更されます。Hot Rod は、クラスターポロジ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。これを行うために、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ポロジを更新する 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](#)

### 13.2. 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トコルを提供します。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 プロトコルでは、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、クラスターリング、スケーラビリティの状態共有、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、JON、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](#)

## 13.3. 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 での整合的な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](#)

## 13.4. HOT ROD インターフェースコネクタ

Red Hat JBoss Data Grid は、以下の 3 つのコネクタタイプをサポートします。

- Hot Rod ベースコネクタの設定を定義する **hotrod-connector** 要素。
- memcached ベースコネクタの設定を定義する **memcached-connector** 要素。
- REST インターフェースベースのコネクタの設定を定義する **rest-connector** 要素。

コネクタの宣言により、`<socket-binding-group />` 内で宣言されたソケットバインディングを使用し、**local** コンテナで宣言されたキャッシュを公開し、他のすべての設定でデフォルト値を使用して Hot Rod、Memcached、または REST サーバーが有効になります。以下の例は、Hot Rod、Memcached、および REST サーバーに接続する方法を示しています。

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 コネクタは追加の設定で調整できます。Hot Rod コネクタの設定方法の詳細については、「[Hot Rod コネクタの設定](#)」を参照してください。



### 注記

Hot Rod コネクタは 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](#)

## 13.4.1. Hot Rod コネクタの設定

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、次の手順に基いて設定する必要があります。

### 手順13.1 リモートクライアントサーバーモード用 Hot Rod コネクタの設定

```
<subsystem xmlns="urn:infinispan:server:endpoint:6.1">
  <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}"
    await-initial-transfer="${TRUE/FALSE}" />
</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 コネクタで使用す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 コネクタで使用するキャッシュコンテナを指定します。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 コネクタの送信バッファ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 コネクタの受信バッファ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。
  - f. **await-initial-transfer** パラメーターは、初期状態の取得を起動時にすぐに行うかどうかを指定します。このパラメーターは、**lazy-retrieval** が **false** に設定されている場合のみ適用されます。このパラメーターのデフォルト値は **true** です。

[バグを報告する](#)

## 13.5. HOT ROD ヘッダー

### 13.5.1. Hot Rod ヘッダーデータタイプ

Red Hat JBoss Data Grid の Hot Rod で使用されたすべてのキーおよび値はバイトアレイとして格納されます。ただし、特定のヘッダー値 (REST や Memcached の値) は以下のデータタイプを使用して格納されます。

表13.1 ヘッダーデータタイプ



データタイプ	サイズ	説明
vInt	1〜5 バイト。	符号なし可変長整数値。
vLong	1〜9 バイト。	符号なし可変長ロング値。
文字列	-	文字列は常に UTF-8 エンコーディングを使用して表されます。

[バグを報告する](#)

### 13.5.2. 要求ヘッダー

Hot Rod を使用して Red Hat JBoss Data Grid にアクセスする場合、要求ヘッダーの内容は以下のものから構成されます。

表13.2 要求ヘッダーフィールド

フィールド名	データタイプ/サイズ	説明
Magic	1 バイト	ヘッダーが要求ヘッダーまたは応答ヘッダーであるかどうかを示します。
Message ID	vLong	メッセージ ID を含みます。この一意の ID は、要求に応答するときに使用されます。これにより、Hot Rod クライアントは非同期でプロトコルを実装できるようになります。
Version	1 バイト	Hot Rod サーバーバージョンを含みます。
Opcode	1 バイト	関連する操作コードを含みます。要求ヘッダー内でopcodeには要求操作コードのみを含めることができます。
Cache Name Length	vInt	キャッシュ名の長さを格納します。キャッシュ名の長さが0に設定され、キャッシュ名に値が提供されない場合、操作はデフォルトのキャッシュと対話します。
Cache Name	文字列	指定された操作のターゲットキャッシュの名前を格納します。この名前は、キャッシュ設定ファイルの事前定義済みキャッシュの名前に一致する必要があります。

フィールド名	データタイプ/サイズ	説明
Flags	vInt	システムに渡されるフラグを表す可変長の数値を含みます。さらに多くのバイトを読み取る必要があるかどうかを決定するために使用される最大ビットを除き、各ビットはフラグを表します。各フラグを表すためにビットを使用すると、フラグの組み合わせが連結された状態で表されます。
Client Intelligence	1 バイト	サーバーに対するクライアント機能を示す値を含みます。
Topology ID	vInt	クライアントの最後の既知なビュー ID を含みます。基本的なクライアントはこのフィールドに値 <b>0</b> を提供します。トポロジーまたはハッシュ情報をサポートするクライアントは、サーバーが現在のビュー ID に応答するまで値 <b>0</b> (新しいビュー ID が現在のビュー ID を置き換えるためにサーバーにより返されるまで使用されます) を提供します。
Transaction Type	1 バイト	2 つの既知のトランザクションタイプのいずれかを表す値を含みます。現時点でサポートされている値は <b>0</b> のみです。
Transaction ID	バイトアレイ	呼び出しに関連するトランザクションを一意に識別するバイトアレイを含みます。トランザクションタイプはこのバイトアレイの長さを決定します。 <b>Transaction Type</b> の値が <b>0</b> に設定された場合、トランザクション ID は存在しません。

### バグを報告する

### 13.5.3. 応答ヘッダー

Hot Rod を使用して Red Hat JBoss Data Grid にアクセスする場合、応答ヘッダーの内容は以下のものから構成されます。

表13.3 応答ヘッダーフィールド



フィールド名	データタイプ	説明
Magic	1 バイト	ヘッダーが要求または応答ヘッダーであるかどうかを示します。
Message ID	vLong	メッセージ ID を含みます。この一意の ID は、応答を元の要求とペアにするために使用されます。これにより、Hot Rod クライアントは非同期でプロトコルを実装できるようになります。
Opcode	1 バイト	関連する操作コードを含みます。応答ヘッダー内で opcode には応答操作コードのみを含めることができます。
Status	1 バイト	応答のステータスを表すコードを含みます。
Topology Change Marker	1 バイト	応答がトポロジ変更情報に含まれるかどうかを示すマーカーバイトを含みます。

[バグを報告する](#)

### 13.5.4. トポロジ変更ヘッダー

Hot Rod を使用して Red Hat JBoss Data Grid にアクセスする場合は、応答ヘッダーが、異なるトポロジまたはハッシュ配布を区別できるクライアントを探すことによりクラスターまたはビューフォーメーションの変更に応答します。Hot Rod サーバーは現在の **topology ID** と、クライアントにより送信された **topology ID** を比較し、2 つの値が異なる場合は、新しい **topology ID** を返します。

[バグを報告する](#)

#### 13.5.4.1. トポロジ変更マーカー値

以下は、応答ヘッダー内の **Topology Change Marker** フィールドの有効な値のリストです。

表13.4 Topology Change Marker フィールド値

値	説明
0	トポロジの変更情報は追加されません。
1	トポロジの変更情報が追加されます。

[バグを報告する](#)

#### 13.5.4.2. トポロジ認識クライアントのトポロジ変更ヘッダー

トポロジーの変更がサーバーにより返された場合、トポロジー認識クライアントに送信された応答ヘッダーには以下の要素が含まれます。

表13.5 トポロジー変更ヘッダーフィールド

応答ヘッダーフィールド	データタイプ/サイズ	説明
Response Header with Topology Change Marker	-	-
Topology ID	vInt	-
Num Servers in Topology	vInt	クラスターで稼働している Hot Rod サーバーの数を含みます。一部のノードのみが Hot Rod サーバーを稼働している場合に、この値は、クラスター全体のサブセットになることがあります。
mX: Host/IP Length	vInt	個別クラスターメンバーのホスト名または IP アドレスの長さを含みます。可変長により、この要素にはホスト名、IPv4、および IPv6 アドレスを含めることができます。
mX: Host/IP Address	文字列	個別クラスターメンバーのホスト名または IP アドレスを含みます。Hot Rod クライアントはこの情報を使用して個別クラスターメンバーにアクセスします。
mX: Port	符号なしショート。2 バイト	クラスターメンバーと通信するために Hot Rod クライアントが使用するポートを含みます。

トポロジー内の各サーバーに対して、接頭辞 **mX** の 3 つのエントリーが繰り返されます。トポロジーの情報フィールド内の最初のサーバーには接頭辞 **m1** が付けられ、**X** の値が **num servers in topology** フィールドで指定されたサーバーの数と等しくなるまで、各追加サーバーに対して数値が 1 つずつ増分されます。

[バグを報告する](#)

#### 13.5.4.3. ハッシュ配布認識クライアントのトポロジー変更ヘッダー

トポロジーの変更がサーバーにより返された場合、クライアントに送信された応答ヘッダーには以下の要素が含まれます。

表13.6 トポロジー変更ヘッダーフィールド

フィールド	データタイプ/サイズ	説明
Response Header with Topology Change Marker	-	-
Topology ID	vInt	-
Number Key Owners	符号なしショート、2 バイト	配布された各キーに対してグローバルに設定されたコピーの数を含みます。配布がキャッシュで設定されていない場合は、値 <b>0</b> を含みます。
Hash Function Version	1 バイト	使用中のハッシュ機能へのポインターを含みます。配布がキャッシュで設定されていない場合は、値 <b>0</b> を含みます。
Hash Space Size	vInt	ハッシュコード生成に関連するすべてのモジュール計算のために <b>JBoss Data Grid</b> により使用されるモジュールを含みます。クライアントはこの情報を使用して正しいハッシュ計算をキーに適用します。配布がキャッシュで設定されていない場合は、値 <b>0</b> を含みます。
Number servers in topology	vInt	クラスター内で稼働している <b>Hot Rod</b> サーバーの数を含みます。一部のノードのみが <b>Hot Rod</b> サーバーを稼働している場合に、この値は、クラスター全体のサブセットになることがあります。また、この値はヘッダーに含まれるホストとポートのペアの数を表します。
Number Virtual Nodes Owners	vInt	設定された仮想ノードの数を含みます。仮想ノードが設定されていない場合、または配布がキャッシュで設定されていない場合は、値 <b>0</b> を含みます。
mX: Host/IP Length	vInt	個別クラスターメンバーのホスト名または <b>IP</b> アドレスの長さを含みます。可変長により、この要素にはホスト名、 <b>IPv4</b> 、および <b>IPv6</b> アドレスを含めることができます。

フィールド	データタイプ/サイズ	説明
mX: Host/IP Address	文字列	個別クラスターメンバーのホスト名または <b>IP</b> アドレスを含みます。 <b>Hot Rod</b> クライアントはこの情報を使用して個別クラスターメンバーにアクセスします。
mX: Port	符号なしショート、2 バイト	クラスターメンバーと通信するために <b>Hot Rod</b> クライアントが使用するポートを含みます。
mX: Hashcode	4 バイト	

トポロジー内の各サーバーに対して、接頭辞 **mX** の 3 つのエントリーが繰り返されます。トポロジーの情報フィールド内の最初のサーバーには接頭辞 **m1** が付けられ、**X** の値が **num servers in topology** フィールドで指定されたサーバーの数と等しくなるまで、各追加サーバーに対して数値が 1 つずつ増分されます。

[バグを報告する](#)

## 13.6. HOT ROD 操作

以下に、Hot Rod プロトコル 1.3 を使用して Red Hat JBoss Data Grid と対話する場合に有効な操作を示します。

- BulkGetKeys
- BulkGet
- Clear
- ContainsKey
- Get
- GetWithMetadata
- Ping
- PutIfAbsent
- Put
- Query
- RemoveIfUnmodified
- Remove
- ReplaceIfUnmodified
- 次の箇所を探します。

- Stats



## 重要

RemoteCache API を使用して Hot Rod クライアントの **Put** 操作、**PutIfAbsent** 操作、**Replace** 操作、および **ReplaceWithVersion** 操作を呼び出す場合に、ライフスパンが 30 日より大きい値に設定されると、値は UNIX 時間として処理され、1970 年 1 月 1 日以降の秒数を表します。

[バグを報告する](#)

### 13.6.1. Hot Rod BulkGet 操作

Hot Rod BulkGet 操作は、以下の要求形式を使用します。

表13.7 BulkGet 操作要求形式

フィールド	データタイプ	説明
Header	-	-
エントリー数	vInt	サーバーにより返される Red Hat JBoss Data Grid エントリーの最大数が含まれます。エントリーはキーと値のペアです。

この操作の応答ヘッダーには、以下のいずれかの応答ステータスが含まれます。

表13.8 BulkGet 操作応答形式

フィールド	データタイプ	説明
Header	-	-
詳細	vInt	ストリームからエントリーをさらに読み取る必要があるかどうかを示します。 <b>More</b> は <b>1</b> に設定される一方で、 <b>More</b> の値が <b>0</b> (ストリームの最後を示します) に設定されるまで追加のエントリーが続きます。
キーサイズ	-	キーのサイズを含みます。
キー	-	キーの値を含みます。
値サイズ	-	値のサイズを含みます。
値	-	値を含みます。

要求された各エントリーに対して、**More**、**Key Size**、**Key**、**Value Size**、および**Value** エントリーが応答に追加されます。

[バグを報告する](#)

### 13.6.2. Hot Rod BulkGetKeys 操作

Hot Rod BulkGetKeys 操作は、以下の要求形式を使用します。

表13.9 BulkGetKeys 操作の要求形式

フィールド	データタイプ	説明
Header	変数	要求ヘッダー
Scope	vInt	<ul style="list-style-type: none"> <li>● <b>0</b> = デフォルトのスコープ - このスコープは <b>RemoteCache.keySet()</b> メソッドにより使用されます。リモートキャッシュが分散キャッシュである場合は、サーバーによりマップ/削減操作が実行され、すべてのノードからすべてのキーが取得されます (トポロジー認識 Hot Rod クライアントはクラスター内の任意のノードへの要求を負荷分散できます)。それ以外の場合は、要求を受信するサーバーに対してローカルのキャッシュインスタンスからキーを取得します。キーはレプリケートされたキャッシュのすべてのノードで同じである必要があります。</li> <li>● <b>1</b> = グローバルスコープ - このスコープはデフォルトスコープと同じように動作します。</li> <li>● <b>2</b> = ローカルスコープ - リモートキャッシュが分散キャッシュである状況では、サーバーはマップ/削減操作を開始してすべてのノードからキーを取得しません。代わりに、要求を受け取るサーバーに対してローカルなキャッシュインスタンスからローカルなキーのみを取得します。</li> </ul>

この操作の応答ヘッダーには、以下のいずれかの応答ステータスが含まれます。

表13.10 BulkGetKeys 操作の応答形式

フィールド	データタイプ	説明
Header	変数	応答ヘッダー
Response status	1 バイト	<b>0x00</b> = 成功 (データが続きます)。
More	1 バイト	ストリームからより多くのキーを読み取る必要があるかを表す1つのバイト。 <b>1</b> に設定されている場合はエントリーが続き、 <b>0</b> に設定されている場合はストリームの最後であり、読み取るエントリーが残っていません。
Key 1 Length	vInt	キーの長さ
Key 1	バイトアレイ	取得されたキー
More	1 バイト	-
Key 2 Length	vInt	-
Key 2	バイトアレイ	-
...etc		

[バグを報告する](#)

### 13.6.3. Hot Rod clear 操作

**clear** 操作形式には、ヘッダーのみ含まれます。

この操作に有効な応答ステータスは以下のとおりです。

表13.11 clear 操作応答

応答ステータス	説明
0x00	Red Hat JBoss Data Grid が正常に消去されました。

[バグを報告する](#)

### 13.6.4. Hot Rod ContainsKey 操作

**Hot Rod ContainsKey** 操作は、以下の要求形式を使用します。

表13.12 ContainsKey 操作要求形式

フィールド	データタイプ	説明
Header	-	-
キーの長さ	vInt	キーの長さを含みます。 <b>Integer.MAX_VALUE</b> のサイズよりも大きいサイズ (最大 <b>5</b> バイト) のため、 <b>vInt</b> データタイプが使用されます。ただし、Java では、単一アレイサイズを <b>Integer.MAX_VALUE</b> のサイズよりも大きくすることはできません。結果として、この <b>vInt</b> は <b>Integer.MAX_VALUE</b> の最大サイズに限定されます。
キー	バイトアレイ	キーを含みます (このキーの対応する値が要求されます)。

この操作の応答ヘッダーには、以下のいずれかの応答ステータスが含まれます。

表13.13 ContainsKey 操作応答形式

応答ステータス	説明
0x00	操作が成功。
0x02	キーが存在しない。

この操作の応答は空白です。

[バグを報告する](#)

### 13.6.5. Hot Rod Get 操作

Hot Rod Get 操作は、以下の要求形式を使用します。

表13.14 Get 操作要求形式

フィールド	データタイプ	説明
Header	-	-



フィールド	データタイプ	説明
Key Length	vInt	キーの長さを含みます。 <b>Integer.MAX_VALUE</b> のサイズよりも大きいサイズ(最大5 バイト)のため、 <b>vInt</b> データタイプが使用されます。ただし、Java では、単一アレイサイズを <b>Integer.MAX_VALUE</b> のサイズよりも大きくすることはできません。結果として、この <b>vInt</b> は <b>Integer.MAX_VALUE</b> の最大サイズに限定されます。
Key	バイトアレイ	キーを含みます (このキーの対応する値が要求されます)。

この操作の応答ヘッダーには、以下のいずれかの応答ステータスが含まれます。

表13.15 Get 操作応答形式

応答ステータス	説明
0x00	操作が成功。
0x02	キーが存在しない。

キーが見つかった場合の **get** 操作の応答の形式は以下のとおりです。

表13.16 Get 操作応答形式

フィールド	データタイプ	説明
Header	-	-
Value Length	vInt	値の長さを含みます。
Value	バイトアレイ	要求された値を含みます。

[バグを報告する](#)

### 13.6.6. Hot Rod GetWithMetadata 操作

Hot Rod **GetWithMetadata** 操作は以下の要求形式を使用します。

表13.17 GetWithMetadata 操作の要求形式

フィールド	データタイプ	説明
Header	変数	要求ヘッダー。
Key Length	vInt	キーの長さ。vInt のサイズは最大 5 バイトであり、理論的には <b>Integer.MAX_VALUE</b> よりも大きい数を生成できます。ただし、Java では <b>Integer.MAX_VALUE</b> よりも大きい単一アレイを作成できず、プロトコルにより vInt アレイの長さが <b>Integer.MAX_VALUE</b> に制限されます。
Key	バイトアレイ	値が要求されるキーを含むバイトアレイ。

この操作の応答ヘッダーには、以下のいずれかの応答ステータスが含まれます。

表13.18 GetWithMetadata 操作の応答要求

フィールド	データタイプ	説明
Header	変数	応答ヘッダー。
Response status	1 バイト	<b>0x00</b> = 成功 (キーが取得された場合)。  <b>0x02</b> = キーが存在しない場合。
Flag	1 バイト	応答に失効に関する情報含まれるかどうかを示すフラグ。フラグの値は、 <b>INFINITE_LIFESPAN (0x01)</b> と <b>INFINITE_MAXIDLE (0x02)</b> 間のビットごとの OR 演算として取得されます。
Created	Long	(オプション) サーバーでエントリが作成されたときのタイムスタンプを表す Long。この値は、フラグの <b>INFINITE_LIFESPAN</b> ビットが設定されていない場合にのみ返されます。

フィールド	データタイプ	説明
Lifespan	vInt	(オプション) エントリのライフスパンを表すvInt (秒単位)。この値は、フラグの <b>INFINITE_LIFESPAN</b> ビットが設定されていない場合にのみ返されます。
LastUsed	Long	(オプション) サーバーでエントリが最後にアクセスされたときのタイムスタンプを表すLong。この値は、フラグの <b>INFINITE_MAXIDLE</b> ビットが設定されていない場合にのみ返されます。
MaxIdle	vInt	(オプション) エントリの maxIdle を表すvInt (秒単位)。この値は、フラグの <b>INFINITE_MAXIDLE</b> ビットが設定されていない場合にのみ返されます。
Entry Version	8 バイト	既存のエントリー変更の一意の値。プロトコルでは <b>entry_version</b> の値はシーケンシャルであることが保証されず、キーレベルで更新ごとに一意である必要があります。
Value Length	vInt	成功した場合は、値の長さ。
Value	バイトアレイ	成功した場合は、要求された値。

[バグを報告する](#)

### 13.6.7. Hot Rod ping 操作

**ping** は、サーバーの可用性を確認するアプリケーションレベルの要求です。

この操作に有効な応答ステータスは以下のとおりです。

表13.19 ping 操作応答

応答ステータス	説明
0x00	エラーなしの正常な ping。

[バグを報告する](#)

### 13.6.8. Hot Rod Put 操作

**put** 操作要求形式には、以下のものが含まれます。

表13.20

フィールド	データタイプ	詳細
Header	-	-
Key Length	-	キーの長さを含みます。
Key	バイトアレイ	キーの値を含みます。
Lifespan	vInt	エントリーが期限切れになるまでの秒数を含みます。秒数が <b>30</b> 日を超える場合、その値はエントリーライフスパンの <b>UNIX</b> 時間 (つまり、日付 <b>1/1/1970</b> 以降の秒数) として処理されます。値が <b>0</b> に設定された場合、エントリーは期限切れになりません。
Max Idle	vInt	キャッシュからエビクトされるまでエントリーがアイドル状態のままになることが許可される秒数を含みます。このエントリーが <b>0</b> に設定された場合、エントリーは無期限でアイドル状態のままになることが許可され、 <b>max idle</b> 値のため、エビクトされません。
Value Length	vInt	値の長さを含みます。
値	バイトアレイ	要求された値。

以下は、この操作から返される応答値です。

表13.21

応答ステータス	詳細
0x00	値が正常に格納されました。

この操作では空の応答がデフォルト応答になります。ただし、**ForceReturnPreviousValue** が渡された場合は、以前の値とキーが返されます。以前のキーと値が存在しない場合は、値の長さに値 **0** が含まれます。



## 重要

RemoteCache API を使用して Hot Rod クライアントの **Put** 操作、**PutIfAbsent** 操作、**Replace** 操作、および **ReplaceWithVersion** 操作を呼び出す場合に、ライフスパンが 30 日より大きい値に設定されると、値は UNIX 時間として処理され、1970 年 1 月 1 日以降の秒数を表します。

[バグを報告する](#)

### 13.6.9. Hot Rod PutIfAbsent 操作

**putIfAbsent** 操作要求形式には、以下のものが含まれます。

表13.22 PutIfAbsent 操作要求フィールド

フィールド	データタイプ	説明
Header	-	-
Key Length	vInt	キーの長さを含みます。
Key	バイトアレイ	キーの値を含みます。
Lifespan	vInt	エントリーが期限切れになるまでの秒数を含みます。秒数が 30 日を超える場合、その値はエントリーライフスパンの UNIX 時間 (つまり、日付 <b>1/1/1970</b> 以降の秒数) として処理されます。値が <b>0</b> に設定された場合、エントリーは期限切れになりません。
Max Idle	vInt	キャッシュからエビクトされるまでエントリーがアイドル状態のままになることが許可される秒数を含みます。このエントリーが <b>0</b> に設定された場合、エントリーは無期限でアイドル状態のままになることが許可され、 <b>max idle</b> 値のため、エビクトされません。
Value Length	vInt	値の長さを含みます。
Value	バイトアレイ	要求された値を含みます。

以下は、この操作から返される応答値です。

表13.23

応答ステータス	説明
0x00	値が正常に格納されました。
0x01	キーが存在しないため、値が格納されませんでした。キーの現在の値が返されました。

この操作では空の応答がデフォルト応答になります。ただし、**ForceReturnPreviousValue** が渡された場合は、以前の値とキーが返されます。以前のキーと値が存在しない場合は、値の長さに値 **0** が含まれます。



## 重要

RemoteCache API を使用して Hot Rod クライアントの **Put** 操作、**PutIfAbsent** 操作、**Replace** 操作、および **ReplaceWithVersion** 操作を呼び出す場合に、ライフスパンが 30 日より大きい値に設定されると、値は UNIX 時間として処理され、1970 年 1 月 1 日以降の秒数を表します。

[バグを報告する](#)

## 13.6.10. Hot Rod クエリー操作

Query 操作要求形式には、以下のものが含まれます。

表13.24 クエリー操作要求フィールド

フィールド	データタイプ	詳細
Header	変数	要求ヘッダー。
Query Length	vInt	Protobuf エンコードされたクエリーオブジェクトの長さ。
Query	バイトアレイ	Protobuf エンコードされたクエリーオブジェクトを含むバイトアレイ。長さは前のフィールドにより指定されます。

以下は、この操作から返される応答値です。

表13.25 クエリー操作応答

応答ステータス	データ	詳細
Header	変数	応答ヘッダー。
Response payload Length	vInt	Protobuf エンコードされた応答オブジェクトの長さ。

応答ステータス	データ	詳細
Response payload	バイトアレイ	Protobuf エンコードされた応答オブジェクトを含むバイトアレイ。長さは前のフィールドにより指定されます。

[バグを報告する](#)

### 13.6.11. Hot Rod Remove 操作

Hot Rod Remove 操作は、以下の要求形式を使用します。

表13.26 Remove 操作要求形式

フィールド	データタイプ	説明
Header	-	-
Key Length	vInt	キーの長さを含みます。 <b>Integer.MAX_VALUE</b> のサイズよりも大きいサイズ (最大 5 バイト) のため、vInt データタイプが使用されます。ただし、Java では、単一アレイサイズを <b>Integer.MAX_VALUE</b> のサイズよりも大きくすることはできません。結果として、この vInt は <b>Integer.MAX_VALUE</b> の最大サイズに限定されます。
Key	バイトアレイ	キーを含みます (このキーの対応する値が要求されます)。

この操作の応答ヘッダーには、以下のいずれかの応答ステータスが含まれます。

表13.27 Remove 操作応答形式

応答ステータス	説明
0x00	操作が成功。
0x02	キーが存在しない。

通常、この操作の応答ヘッダーは空白です。ただし、**ForceReturnPreviousValue** が渡された場合は、応答ヘッダーに以下のいずれかが含まれます。

- 以前のキーの値および長さ。
- キーが存在しないことを示す、値の長さ 0 と応答ステータス 0x02。

**remove** 操作の応答ヘッダーには、提供されたキーの以前の値と、以前の値の長さが含まれます (**ForceReturnPreviousValue** が渡された場合)。キーが存在しない場合、または以前の値が **null** の場合、値の長さは **0** です。

[バグを報告する](#)

### 13.6.12. Hot Rod RemoveIfUnmodified 操作

**RemoveIfUnmodified** 操作要求形式には、以下のものが含まれます。

表13.28 RemoveIfUnmodified 操作要求フィールド

フィールド	データタイプ	説明
Header	-	-
Key Length	vInt	キーの長さを含みます。
Key	バイトアレイ	キーの値を含みます。
Entry Version	8 バイト	エントリーのバージョン番号。

以下は、この操作から返される応答値です。

表13.29 RemoveIfUnmodified 操作応答

応答ステータス	説明
0x00	エントリーが置換または削除された場合に返されたステータス。
0x01	キーが変更されたため、エントリーの置換または削除が失敗した場合に、ステータスを返します。
0x02	キーが存在しない場合に、ステータスを返します。

この操作では空の応答がデフォルト応答になります。ただし、**ForceReturnPreviousValue** が渡された場合は、以前の値とキーが返されます。以前のキーと値が存在しない場合は、値の長さに値 **0** が含まれます。

[バグを報告する](#)

### 13.6.13. Hot Rod replace 操作

**replace** 操作要求形式には、以下のものが含まれます。

表13.30 replace 操作要求フィールド



フィールド	データタイプ	説明
Header	-	-
Key Length	vInt	キーの長さを含みます。
キー	バイトアレイ	キーの値を含みます。
Lifespan	vInt	エントリーが期限切れになるまでの秒数を含みます。秒数が30日を超える場合、その値はエントリーライフスパンのUNIX時間(つまり、日付1/1/1970以降の秒数)として処理されます。値が0に設定された場合、エントリーは期限切れになりません。
Max Idle	vInt	キャッシュからエビクトされるまでエントリーがアイドル状態のままになることが許可される秒数を含みます。このエントリーが0に設定された場合、エントリーは無期限でアイドル状態のままになることが許可され、max idle 値のため、エビクトされません。
Value Length	vInt	値の長さを含みます。
値	バイトアレイ	要求された値を含みます。

以下は、この操作から返される応答値です。

表13.31 replace 操作応答

応答ステータス	説明
0x00	値が正常に格納されました。
0x01	キーが存在しないため、値が格納されませんでした。

この操作では空の応答がデフォルト応答になります。ただし、**ForceReturnPreviousValue** が渡された場合は、以前の値とキーが返されます。以前のキーと値が存在しない場合は、値の長さに値 0 が含まれます。



## 重要

RemoteCache API を使用して Hot Rod クライアントの **Put** 操作、**PutIfAbsent** 操作、**Replace** 操作、および **ReplaceWithVersion** 操作を呼び出す場合に、ライフスパンが 30 日より大きい値に設定されると、値は UNIX 時間として処理され、1970 年 1 月 1 日以降の秒数を表します。

[バグを報告する](#)

### 13.6.14. Hot Rod ReplaceWithVersion 操作

**ReplaceWithVersion** 操作要求形式には、以下のものが含まれます。



## 注記

RemoteCache API では、Hot Rod **ReplaceWithVersion** 操作は **ReplaceIfUnmodified** 操作を使用します。結果として、これらの 2 つの操作は JBoss Data Grid でまったく同じになります。

表13.32 ReplaceWithVersion 操作要求フィールド

フィールド	データタイプ	説明
Header	-	-
キー長	vInt	キーの長さを含みます。
Key	バイトアレイ	キーの値を含みます。
Lifespan	vInt	エントリーが期限切れになるまでの秒数を含みます。秒数が 30 日を超える場合、その値はエントリーライフスパンの UNIX 時間 (つまり、日付 <b>1/1/1970</b> 以降の秒数) として処理されます。値が <b>0</b> に設定された場合、エントリーは期限切れになりません。
Max Idle	vInt	キャッシュからエビクトされるまでエントリーがアイドル状態のままになることが許可される秒数を含みます。このエントリーが <b>0</b> に設定された場合、エントリーは無期限でアイドル状態のままになることが許可され、max idle 値のため、エビクトされません。
Entry Version	8 バイト	エントリーのバージョン番号。
Value Length	vInt	値の長さを含みます。

フィールド	データタイプ	説明
値	バイトアレイ	要求された値を含みます。

以下は、この操作から返される応答値です

**表13.33 ReplaceWithVersion 操作応答**

応答ステータス	説明
0x00	エントリーが置換または削除された場合に返されたステータス。
0x01	キーが変更されたため、エントリーの置換または削除が失敗した場合に、ステータスを返します。
0x02	キーが存在しない場合に、ステータスを返します。

この操作では空の応答がデフォルトの応答になります。ただし、**ForceReturnPreviousValue** が渡された場合は、以前の値とキーが返されます。以前のキーと値が存在しない場合は、値の長さに値 **0** が含まれます。

[バグを報告する](#)

### 13.6.15. Hot Rod 統計操作

この操作は、利用可能なすべての統計の概要を返します。返された各統計に対して、名前と値が文字列形式と UTF-8 形式の両方で返されます。

この操作では、以下の統計がサポートされます。

**表13.34 統計操作要求フィールド**

名前	説明
timeSinceStart	Hot Rod が起動した以降の秒数を含みます。
currentNumberOfEntries	Hot Rod サーバーに現在存在するエントリーの数を含みます。
totalNumberOfEntries	Hot Rod サーバーに格納されたエントリーの合計数を含みます。
stores	put 操作の試行回数を含みます。
retrievals	get 操作の試行回数を含みます。
hits	get ヒット数を含みます。

名前	説明
misses	get 失敗数を含みます。
removeHits	remove ヒット数を含みます。
removeMisses	removal 失敗数を含みます。

この操作の応答ヘッダーには以下のものが含まれます。

表13.35 統計操作応答

名前	データタイプ	説明
Header	-	-
Number of Stats	vInt	返された個別統計の数を含みます。
Name Length	vInt	名前付き統計の長さを含みます。
名前	文字列	統計の名前を含みます。
Value Length	vInt	値の長さを含みます。
値	文字列	統計値を含みます。

要求された各統計に対して、値 **Name Length**、**Name**、**Value Length**、および **Value** が繰り返されます。

[バグを報告する](#)

## 13.7. HOT ROD 操作の値

以下は、要求ヘッダーと対応する応答ヘッダー値の有効な **opcode** 値のリストです。

表13.36 opcode 要求および応答ヘッダー値

操作	要求操作コード	応答操作コード
put	0x01	0x02
get	0x03	0x04
putIfAbsent	0x05	0x06
replace	0x07	0x08

操作	要求操作コード	応答操作コード
replacelfUnmodified	0x09	0x0A
remove	0x0B	0x0C
removelfUnmodified	0x0D	0x0E
containsKey	0x0F	0x10
clear	0x13	0x14
stats	0x15	0x16
ping	0x17	0x18
bulkGet	0x19	0x1A
getWithMetadata	0x1B	0x1C
bulkKeysGet	0x1D	0x1E
query	0x1F	0x20

また、応答ヘッダーの **opcode** 値が **0x50** の場合は、エラー応答を示します。

[バグを報告する](#)

### 13.7.1. Magic 値

以下は要求および応答ヘッダー内の **Magic** フィールドの有効な値のリストです。

表13.37 Magic フィールド値

値	説明
0xA0	キャッシュ要求マーカ。
0xA1	キャッシュ応答マーカ。

[バグを報告する](#)

### 13.7.2. ステータス値

以下は、応答ヘッダー内の **Status** フィールドに対するすべての有効な値を含む表です。

表13.38 ステータス値

値	説明
0x00	エラーなし。
0x01	配置、削除、置換なし。
0x02	キーは存在しない。
0x81	無効なマジック値またはメッセージ ID。
0x82	不明なコマンド。
0x83	不明なバージョン。
0x84	要求解析エラー。
0x85	サーバーエラー。
0x86	コマンドタイムアウト。

[バグを報告する](#)

### 13.7.3. トランザクションタイプ値

以下は、要求ヘッダー内の **Transaction Type** の有効な値のリストです。

表13.39 Transaction Type フィールド値

値	説明
0	非トランザクション呼び出し、またはクライアントがトランザクションをサポートしないことを示します。使用された場合は、 <b>TX_ID</b> フィールドが省略されます。
1	X/Open XA トランザクション ID (XID) を示します。この値は現在サポートされていません。

[バグを報告する](#)

### 13.7.4. Client Intelligence 値

以下は、要求ヘッダー内の **Client Intelligence** の有効な値のリストです。

表13.40 Client Intelligence フィールド値

値	説明
0x01	クラスターまたはハッシュ情報が必要でない基本的なクライアントを示します。
0x02	トポロジーを認識し、クラスター情報が必要なクラスターを示します。
0x03	ハッシュと配布を認識し、クラスターおよびハッシュ情報が必要なクライアントを示します。

[バグを報告する](#)

### 13.7.5. フラグ値

以下は、要求ヘッダー内の有効な **flag** 値のリストです。

表13.41 フラグフィールド値

値	説明
0x0001	ForceReturnPreviousValue

[バグを報告する](#)

### 13.7.6. Hot Rod エラー処理

表13.42 応答ヘッダーフィールドを使用した Hot Rod エラー処理

フィールド	データタイプ	説明
Error Opcode	-	エラー操作コードを含みます。
Error Status Number	-	<b>error opcode</b> に対応するステータス番号を含みます。
Error Message Length	vInt	エラーメッセージの長さを含みます。
Error Message	文字列	実際のエラーメッセージを含みます。要求の解析エラーが存在することを示す <b>0x84</b> エラーコードが返された場合、このフィールドには、 <b>Hot Rod</b> サーバーでサポートされた最新バージョンが含まれます。

[バグを報告する](#)

## 13.8. PUT 要求の例

以下は、Hot Rod を使用した **put** 要求例からのコーディングされた要求です。

表13.43 put 要求の例

バイト	0	1	2	3	4	5	6	7
8	0xA0	0x09	0x41	0x01	0x07	0x4D ('M')	0x79 ('y')	0x43 ('C')
16	0x61 ('a')	0x63 ('c')	0x68 ('h')	0x65 ('e')	0x00	0x03	0x00	0x00
24	0x00	0x05	0x48 ('H')	0x65 ('e')	0x6C ('l')	0x6C ('l')	0x6F ('o')	0x00
32	0x00	0x05	0x57 ('W')	0x6F ('o')	0x72 ('r')	0x6C ('l')	0x64 ('d')	-

以下の表には、要求の例に対するすべてのヘッダーフィールドと値が含まれます。

表13.44 要求例のフィールド名と値

フィールド名	バイト	値
Magic	0	0xA0
Version	2	0x41
Cache Name Length	4	0x07
Flag	12	0x00
Topology ID	14	0x00
Transaction ID	16	0x00
Key	18-22	'Hello'
Max Idle	24	0x00
値	26-30	'World'
Message ID	1	0x09
Opcode	3	0x01



フィールド名	バイト	値
Cache Name	5-11	'MyCache'
Client Intelligence	13	0x03
Transaction Type	15	0x00
Key Field Length	17	0x05
Lifespan	23	0x00
Value Field Length	25	0x05

以下は、**put** 要求の例に対するコーディングされた応答です。

表13.45 put 要求の例のコーディングされた応答

バイト	0	1	2	3	4	5	6	7
8	0xA1	0x09	0x01	0x00	0x00	-	-	-

以下の表には、応答の例に対するすべてのヘッダーフィールドと値が含まれます。

表13.46 応答例のフィールド名および値

フィールド名	バイト	値
Magic	0	0xA1
Opcode	2	0x01
Topology Change Marker	4	0x00
Message ID	1	0x09
Status	3	0x00

[バグを報告する](#)

## 13.9. HOT ROD JAVA クライアント

Hot Rod はバイナリーの言語非依存プロトコルです。Java クライアントは、Hot Rod Java Client API を使用して Hot Rod プロトコルを介してサーバーと対話できます。

[バグを報告する](#)

### 13.9.1. Hot Rod Java クライアントのダウンロード

JBoss Data Grid Hot Rod Java クライアントをダウンロードするには、次の手順に従ってください。

#### 手順13.2 Hot Rod Java クライアントのダウンロード

1. カスタマーポータル (<https://access.redhat.com>) にログインします。
2. ページ最上部にある **ダウンロード** ボタンをクリックします。
3. **製品のダウンロード** ページで **Red Hat JBoss Data Grid** をクリックします。
4. **バージョン**: ドロップダウンメニューから適切な JBoss Data Grid バージョンを選択します。
5. **Red Hat JBoss Data Grid \${VERSION} Hot Rod Java Client** エントリーを探し、対応する **ダウンロードリンク** をクリックします。

[バグを報告する](#)

### 13.9.2. Hot Rod Java クライアントの設定

Hot Rod Java クライアントは、プログラムを使用したり、設定ファイルまたはプロパティーファイルを外部的に使用したりして設定されます。次の例は、利用可能な Java 対応 API を使用したクライアントインスタンスの作成を示しています。

#### 例13.1 クライアントインスタンスの作成

```
org.infinispan.client.hotrod.configuration.ConfigurationBuilder cb
= new org.infinispan.client.hotrod.configuration.ConfigurationBuilder();
cb.tcpNoDelay(true)
  .connectionPool()
    .numTestsPerEvictionRun(3)
    .testOnBorrow(false)
    .testOnReturn(false)
    .testWhileIdle(true)
  .addServer()
    .host("localhost")
    .port(11222);
RemoteCacheManager rmc = new RemoteCacheManager(cb.build());
```

#### プロパティーファイルを使用した Hot Rod Java クライアントの設定

Hot Rod Java クライアントを設定するには、クラスパス上の **hotrod-client.properties** ファイルを編集します。

次の例は、**hotrod-client.properties** ファイルの内容を示しています。

#### 例13.2 設定

```
infinispan.client.hotrod.transport_factory =
org.infinispan.client.hotrod.impl.transport.tcp.TcpTransportFactory

infinispan.client.hotrod.server_list = 127.0.0.1:11222
```

```

infinispan.client.hotrod.marshaller =
org.infinispan.commons.marshall.jboss.GenericJBossMarshaller

infinispan.client.hotrod.async_executor_factory =
org.infinispan.client.hotrod.impl.async.DefaultAsyncExecutorFactory

infinispan.client.hotrod.default_executor_factory.pool_size = 1

infinispan.client.hotrod.default_executor_factory.queue_size = 10000

infinispan.client.hotrod.hash_function_impl.1 =
org.infinispan.client.hotrod.impl.consistenthash.ConsistentHashV1

infinispan.client.hotrod.tcp_no_delay = true

infinispan.client.hotrod.ping_on_startup = true

infinispan.client.hotrod.request_balancing_strategy =
org.infinispan.client.hotrod.impl.transport.tcp.RoundRobinBalancingStrategy

infinispan.client.hotrod.key_size_estimate = 64

infinispan.client.hotrod.value_size_estimate = 512

infinispan.client.hotrod.force_return_values = false

infinispan.client.hotrod.tcp_keep_alive = true

## below is connection pooling config

maxActive=-1

maxTotal = -1

maxIdle = -1

whenExhaustedAction = 1

timeBetweenEvictionRunsMillis=120000

minEvictableIdleTimeMillis=300000

testWhileIdle = true

minIdle = 1

```



## 注記

**TCP KEEPALIVE** 設定は、例で示された設定プロパティー (`infinispan.client.hotrod.tcp_keep_alive = true/false`) または `org.infinispan.client.hotrod.ConfigurationBuilder.tcpKeepAlive()` メソッドを使用したプログラムによって Hot Rod Java クライアントで有効/無効になります。

Red Hat JBoss Data Grid でプロパティファイルを使用するには、次の 2 つのコンストラクターのいずれかを使用する必要があります。

1. `new RemoteCacheManager(boolean start)`
2. `new RemoteCacheManager()`

[バグを報告する](#)

### 13.9.3. Hot Rod Java クライアント基本 API

以下のコードは、クライアント API を使用して Hot Rod Java クライアントで Hot Rod サーバーから情報を保存または取得する方法を示しています。この例では、Hot Rod サーバーがデフォルトの場所 `localhost:11222` にバインドするよう起動されていることを前提とします。

#### 例13.3 基本 API

```
//API entry point, by default it connects to localhost:11222
    BasicCacheContainer cacheContainer = new RemoteCacheManager();
//obtain a handle to the remote default cache
    BasicCache<String, String> cache = cacheContainer.getCache();
//now add something to the cache and ensure it is there
    cache.put("car", "ferrari");
    assert cache.get("car").equals("ferrari");
//remove the data
    cache.remove("car");
    assert !cache.containsKey("car") : "Value must have been
removed!";
```

**RemoteCacheManager** は、**DefaultCacheManager** に対応し、両方とも **BasicCacheContainer** を実装します。

この API は、ローカルコールから Hot Rod を介したリモートコールへの移行を実現します。これは、**DefaultCacheManager** と **RemoteCacheManager** を切り替えることによって行うことができ、共通の **BasicCacheContainer** インターフェースによって単純化されます。

すべてのキーは、**keySet()** メソッドを使用してリモートキャッシュから取得できます。リモートキャッシュが分散キャッシュである場合は、サーバーによりマップ/削減ジョブが開始され、クラスターノードからすべてのキーが取得され、すべてのキーがクライアントに返されます。

キーの数が多い場合は、このメソッドを注意して使用してください。

```
Set keys = remoteCache.keySet();
```

[バグを報告する](#)

### 13.9.4. Hot Rod Java クライアントバージョン API

データの整合性を確保するために、Hot Rod は各変更を一意に識別するバージョン番号を保存します。**getVersioned** を使用して、クライアントはキーと現在のバージョンに関連付けられた値を取得できます。

Hot Rod Java クライアントを使用する場合、**RemoteCacheManager** は、リモートクラスター上の名前

付きまたはデフォルトのキャッシュにアクセスする **RemoteCache** インターフェースのインスタンスを提供します。これにより、バージョン API を含む、新しいメソッドを追加する **Cache** インターフェースが拡張されます。

#### 例13.4 バージョンメソッドの使用

```
// To use the versioned API, remote classes are specifically needed
RemoteCacheManager remoteCacheManager = new RemoteCacheManager();
RemoteCache<String, String> remoteCache = remoteCacheManager.getCache();
remoteCache.put("car", "ferrari");
VersionedValue valueBinary = remoteCache.getVersioned("car");
// removal only takes place only if the version has not been changed
// in between. (a new version is associated with 'car' key on each
change)
assert remoteCache.removeWithVersion("car", valueBinary.getVersion());
assert !remoteCache.containsKey("car");
```

#### 例13.5 置換の使用

```
remoteCache.put("car", "ferrari");
VersionedValue valueBinary = remoteCache.getVersioned("car");
assert remoteCache.replaceWithVersion("car", "lamborghini",
valueBinary.getVersion());
```

[バグを報告する](#)

## 13.10. HOT ROD C++ クライアント

Hot Rod C++ クライアントは、Hot Rod Java クライアントを含む Hot Rod クライアントファミリーに新しく追加され、C++ ランタイムアプリケーションが Red Hat JBoss Data Grid リモートサーバーに接続し、対話することを可能にします。

Hot Rod C++ クライアントにより、C++ で開発されたアプリケーションがデータを読み取ったり、データをリモートキャッシュに書き込んだりすることができるようになります。Hot Rod C++ クライアントは、すべての 3 つのレベルのクライアントインテリジェンスをサポートし、以下のプラットフォームでサポートされます。

- Red Hat Enterprise Linux 5、64 ビット
- Red Hat Enterprise Linux 6、64 ビット
- Red Hat Enterprise Linux 7、64 ビット

Hot Rod C++ クライアントは、Visual Studio 2010 がインストールされた 64 ビット Windows ではテクノロジープレビューとして利用できます。

[バグを報告する](#)

### 13.10.1. Hot Rod C++ クライアント形式

Hot Rod C++ クライアントは、以下の 2 つのライブラリー形式で利用可能です。

- 静的ライブラリー
- 共有/動的ライブラリー

### 静的ライブラリー

静的ライブラリーはアプリケーションに静的にリンクされます。これにより、最終的な実行可能ファイルのサイズは増加します。アプリケーションは自己完結型であり、別のライブラリーを提供する必要はありません。

### 共有/動的ライブラリー

共有/動的ライブラリーは、実行時にアプリケーションに動的にリンクされます。ライブラリーは別のファイルに格納され、アプリケーションを再コンパイルせずアプリケーションとは別にアップグレードできます。



#### 注記

これは、ライブラリーのメジャーバージョンがコンパイル時にアプリケーションがリンクされたものと同じである (バイナリー互換性がある) 場合にのみ可能です。

[バグを報告する](#)

## 13.10.2. Hot Rod C++ クライアントの前提条件

Hot Rod C++ クライアントを使用するには、以下のものがが必要です。

- `shared_ptr` TR1 (GCC 4.0+、Visual Studio C++ 2010) をサポートする C++ 03 コンパイラー。
- Red Hat JBoss Data Grid Server 6.1.0 以上のバージョン。

[バグを報告する](#)

## 13.10.3. Hot Rod C++ クライアントのダウンロード

Hot Rod C++ クライアントは、Red Hat カスタマーポータル (<https://access.redhat.com>) の Red Hat JBoss Data Grid バイナリー下にある個別の zip ファイル、`jboss-datagrid-<version>-hotrod-cpp-client-<platform>.zip` に含まれます。使用しているオペレーティングシステムに適切な Hot Rod C++ クライアントをダウンロードしてください。

[バグを報告する](#)

## 13.10.4. Hot Rod C++ クライアントの設定

Hot Rod C++ クライアントは `RemoteCache` API を使用してリモートの Hot Rod サーバーと対話します。特定の Hot Rod サーバーとの通信を開始するために、`RemoteCache` を設定し、Hot Rod サーバーの特定のキャッシュを選択します。

`ConfigurationBuilder` API を使用すると、以下のものを設定できます。

- 接続するサーバーの初期セット。
- 接続プール属性。
- 接続/ソケットタイムアウトおよび TCP `nodelay`。

- Hot Rod プロトコルバージョン。

### C++ 主要実行可能ファイルの設定例

以下の例は、**ConfigurationBuilder** を使用して **RemoteCacheManager** を設定する方法とデフォルトのリモートキャッシュを取得する方法を示しています。

#### 例13.6 SimpleMain.cpp

```
#include "infinispan/hotrod/ConfigurationBuilder.h"
#include "infinispan/hotrod/RemoteCacheManager.h"
#include "infinispan/hotrod/RemoteCache.h"
#include <stdlib.h>
using namespace infinispan::hotrod;
int main(int argc, char** argv) {
    ConfigurationBuilder b;
    b.addServer().host("127.0.0.1").port(11222);
    RemoteCacheManager cm(builder.build());
    RemoteCache<std::string, std::string> cache =
cm.getCache<std::string, std::string>();
    return 0;
}
```

[バグを報告する](#)

### 13.10.5. Hot Rod C++ クライアント API

**RemoteCacheManager** は、**RemoteCache** への参照を取得する開始点です。**RemoteCache API** は、リモート Hot Rod サーバーとサーバー上の特定のキャッシュと対話できます。

前の例で取得した **RemoteCache** 参照を使用すると、リモートキャッシュで値を挿入、取得、置換、および削除できます。また、すべてのキーの取得やキャッシュのクリアなどの一括操作を実行することもできます。

**RemoteCacheManager** が停止されると、使用中のすべてのリソースが解放されます。

#### 例13.7 SimpleMain.cpp

```
RemoteCache<std::string, std::string> rc = cm.getCache<std::string,
std::string>();
std::string k1("key13");
std::string v1("boron");
// put
rc.put(k1, v1);
std::auto_ptr<std::string> rv(rc.get(k1));
rc.putIfAbsent(k1, v1);
std::auto_ptr<std::string> rv2(rc.get(k1));
std::map<HR_SHARED_PTR<std::string>, HR_SHARED_PTR<std::string> > map
= rc.getBulk(0);
std::cout << "getBulk size" << map.size() << std::endl;
..
.
cm.stop();
```

[バグを報告する](#)

## 13.11. HOT ROD C# クライアント

Hot Rod C# クライアントは、Hot Rod Java クライアントと Hot Rod C++ クライアントを含む Hot Rod クライアントのリストに新しく追加されました。Hot Rod C# クライアントでは、.NET ランタイムアプリケーションが Red Hat JBoss Data Grid サーバーに接続し、対話できます。

Hot Rod C# クライアントは、クラスターポロジとハッシュスキームを認識し、Hot Rod Java クライアントと Hot Rod C++ クライアントに類似した単一のホップでサーバー上のエントリーにアクセスできます。

Hot Rod C# クライアントは、.NET Framework が Microsoft によりサポートされる 32 ビットおよび 64 ビットのオペレーティングシステムと互換性があります。.NET Framework 4.0 は、Hot Rod C# クライアントを使用するサポート対象オペレーティングシステムとともに前提条件です。

[バグを報告する](#)

### 13.11.1. Hot Rod C# クライアントのダウンロードとインストール

Hot Rod C# クライアントは、Red Hat JBoss Data Grid でダウンロード向けにパッケージされた .msi ファイル `jboss-datagrid-<version>-hotrod-dotnet-client.msi` に含まれます。Hot Rod C# クライアントをインストールするには、以下の手順を実行してください。

#### 手順13.3 Hot Rod C# クライアントのインストール

1. 管理者として、Hot Rod C# .msi ファイルがダウンロードされた場所に移動します。.msi ファイルを実行してウィンドウインストーラーを起動し、**Next** (次へ) をクリックします。



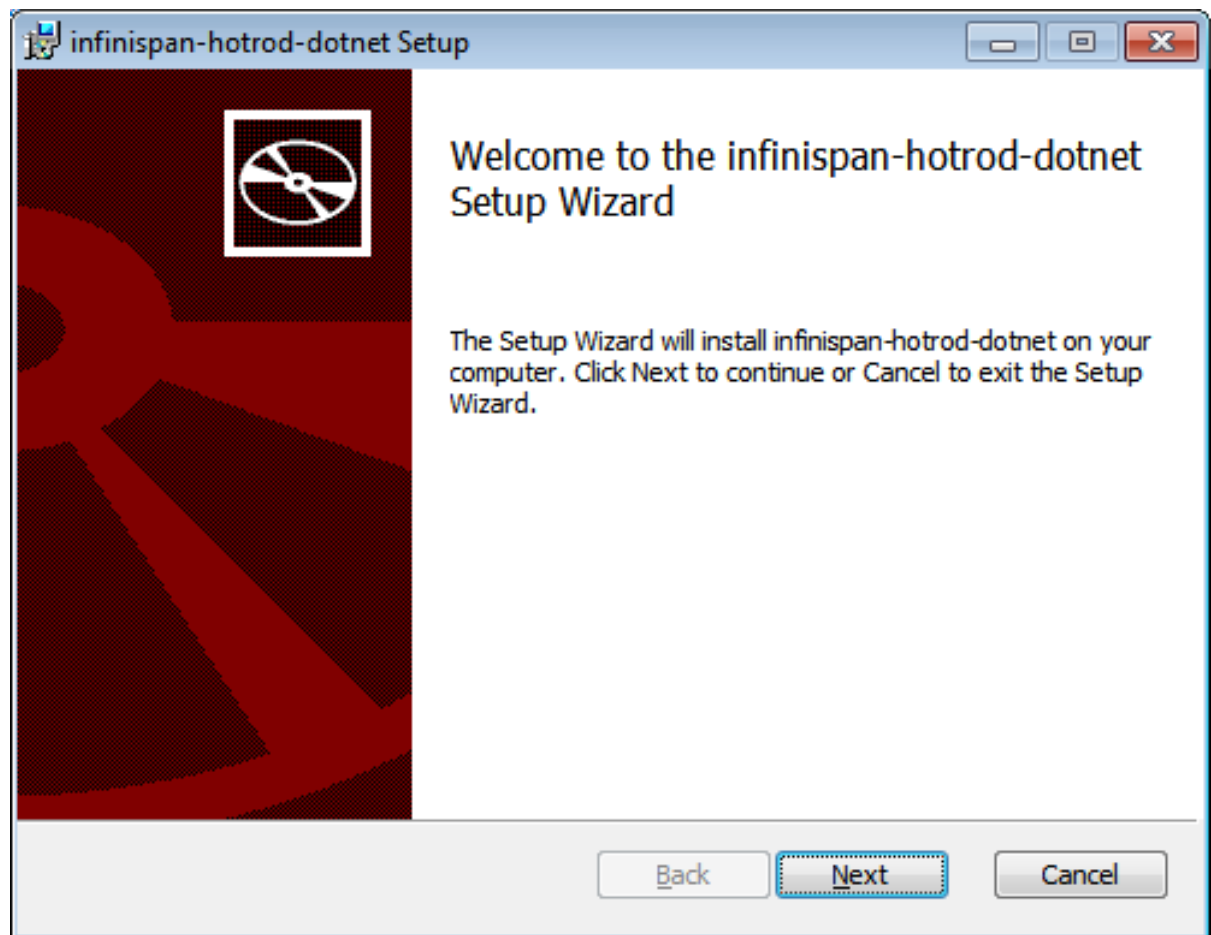


図13.1 Hot Rod C# クライアントのセットアップの開始

2. 使用許諾契約書の内容を確認します。**I accept the terms in the License Agreement** (使用許諾契約に同意します) チェックボックスを選択し、**Next** (次へ) をクリックします。

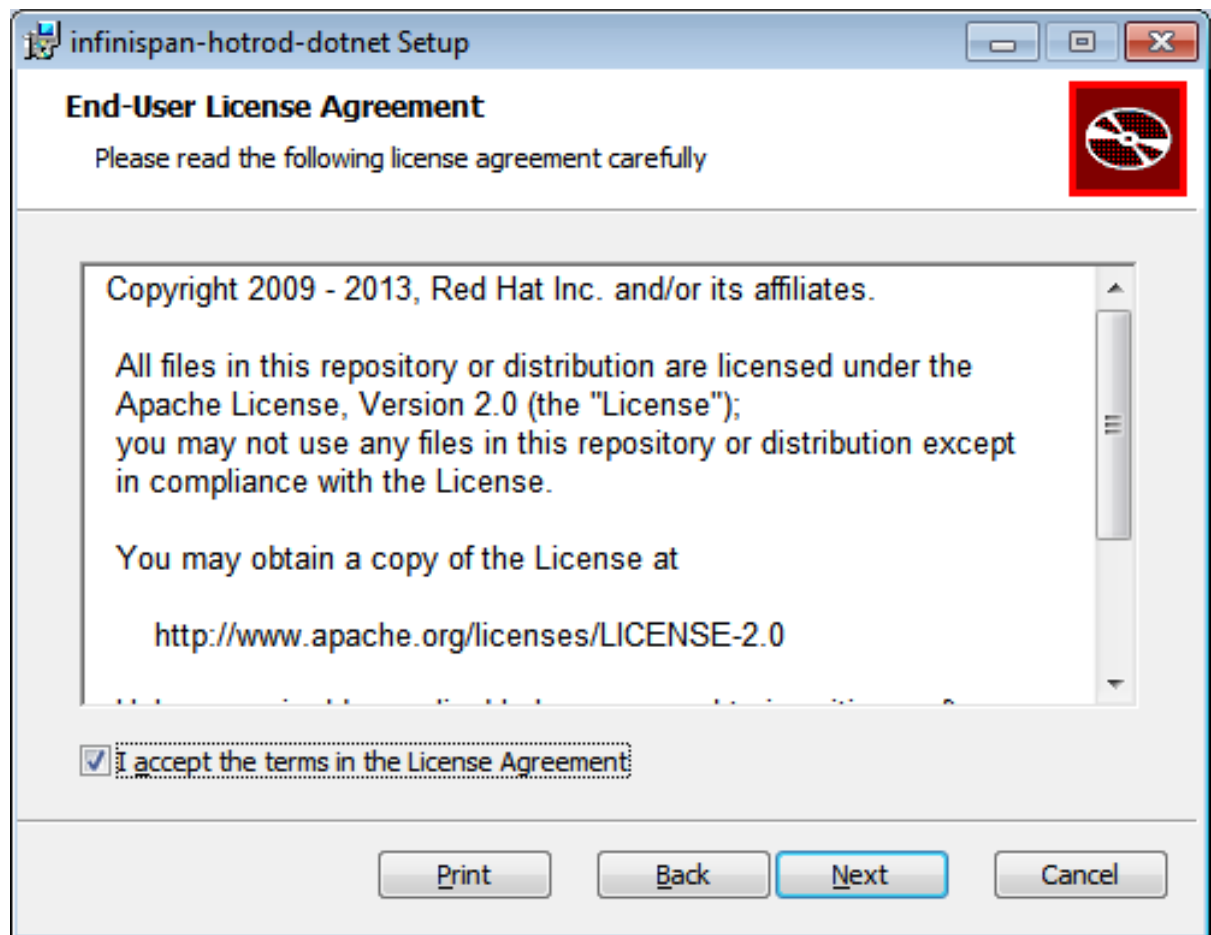


図13.2 Hot Rod C# クライアントの使用許諾契約

3. デフォルトのディレクトリーを変更するには、**Change...** (変更...) または **Next** (次へ) をクリックしてデフォルトのディレクトリーにインストールします。

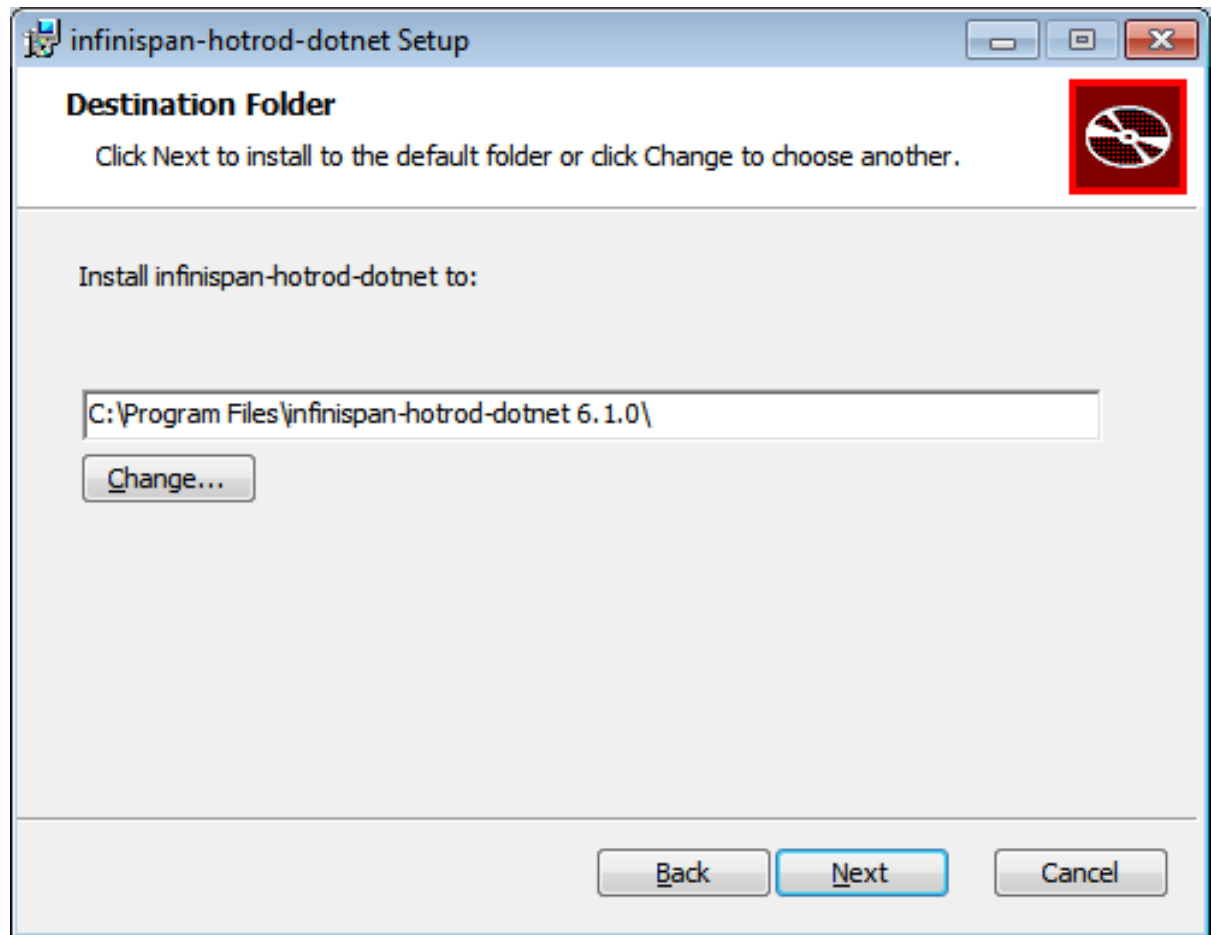


図13.3 Hot Rod C# クライアントの宛先フォルダー

4. **Finish** (完了) をクリックして Hot Rod C# クライアントのインストールを完了します。

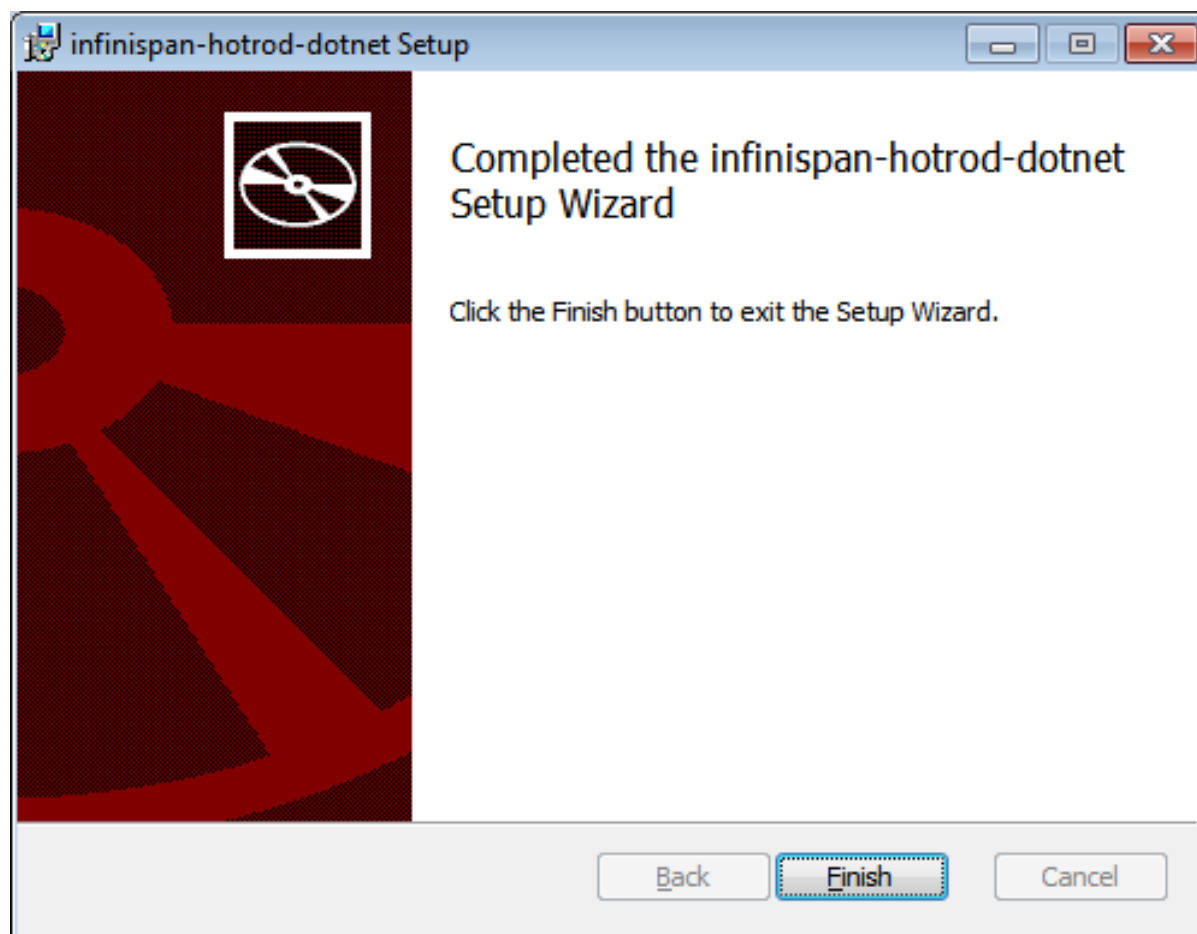


図13.4 Hot Rod C# クライアントのセットアップの完了

[バグを報告する](#)

### 13.11.2. Hot Rod C# クライアントの設定

Hot Rod C# クライアントは **ConfigurationBuilder** を使用してプログラミングにより設定されます。クライアントが接続する必要があるホストとポートを設定します。

#### C# ファイルの設定例

以下の例は、**ConfigurationBuilder** を使用して **RemoteCacheManager** を設定する方法を示しています。

#### 例13.8 C# の設定

```
using System;
using System.Collections.Generic;
using System.Linq;
using System.Text;
using System.Threading.Tasks;
using Infinispan.HotRod;
using Infinispan.HotRod.Config;
namespace simpleapp
{
    class Program
    {
        static void Main(string[] args)
        {
```

```

        ConfigurationBuilder builder = new ConfigurationBuilder();
        builder.AddServer()
            .Host(args.Length > 1 ? args[0] : "127.0.0.1")
            .Port(args.Length > 2 ? int.Parse(args[1]) : 11222);
        Configuration config = builder.Build();
        RemoteCacheManager cacheManager = new
RemoteCacheManager(config);
        [...]
    }
}
}

```

[バグを報告する](#)

### 13.11.3. Hot Rod C# クライアント API

**RemoteCacheManager** は、**RemoteCache** への参照を取得する開始点です。

以下の例は、サーバーからのデフォルトキャッシュの取得と基本的な複数の操作を示しています。

#### 例13.9

```

using System;
using System.Collections.Generic;
using System.Linq;
using System.Text;
using System.Threading.Tasks;
using Infinispan.HotRod;
using Infinispan.HotRod.Config;
namespace simpleapp
{
    class Program
    {
        static void Main(string[] args)
        {
            ConfigurationBuilder builder = new ConfigurationBuilder();
            builder.AddServer()
                .Host(args.Length > 1 ? args[0] : "127.0.0.1")
                .Port(args.Length > 2 ? int.Parse(args[1]) : 11222);
            Configuration config = builder.Build();
            RemoteCacheManager cacheManager = new
RemoteCacheManager(config);
            cacheManager.Start();
            // Retrieve a reference to the default cache.
            IRemoteCache<String, String> cache =
cacheManager.GetCache<String, String>();
            // Add entries.
            cache.Put("key1", "value1");
            cache.PutIfAbsent("key1", "anotherValue1");
            cache.PutIfAbsent("key2", "value2");
            cache.PutIfAbsent("key3", "value3");
            // Retrive entries.
            Console.WriteLine("key1 -> " + cache.Get("key1"));
            // Bulk retrieve key/value pairs.

```

```

        int limit = 10;
        IDictionary<String, String> result = cache.GetBulk(limit);
        foreach (KeyValuePair<String, String> kv in result)
        {
            Console.WriteLine(kv.Key + " -> " + kv.Value);
        }
        // Remove entries.
        cache.Remove("key2");
        Console.WriteLine("key2 -> " + cache.Get("key2"));
        cacheManager.Stop();
    }
}

```

[バグを報告する](#)

#### 13.11.4. 相互運用性を維持するための文字列マーシャラー

文字列互換性マーシャラーを使用するには、サーバー側で互換性モードを有効にします。C# クライアント側で、以下のように **CompatibilitySerializer** のインスタンスを **RemoteCacheManager** コンストラクターに渡します。

```

[...]  
RemoteCacheManager cacheManager = new RemoteCacheManager(new  
CompatibilitySerializer());  
[...]  
cache.Put("key", "value");  
String value = cache.Get("key");  
[...]  


```



#### 注記

非文字列キー/値を格納または取得しようとする、**HotRodClientException** がスローされます。

[バグを報告する](#)

### 13.12. HOT ROD C++ と HOT ROD JAVA クライアント間の相互運用性

Red Hat JBoss Data Grid は、構造化データにアクセスするために Hot Rod Java と Hot Rod C++ クライアント間の相互運用性を提供します。これは、Google の Protobuf 形式を使用してデータを構造化およびシリアル化することにより可能になります。

たとえば、言語間の相互運用性を使用すると、Hot Rod C++ クライアントが Protobuf を使用して構造化およびシリアル化された次の **Person** オブジェクトを記述し、Java C++ クライアントが Protobuf として構造化された同じ **Person** オブジェクトを読み取ることができます。

#### 例13.10 言語間の相互運用性の使用

```

package sample;
message Person {
    required int32 age = 1;
}

```

```
    required string name = 2;  
}
```

Protobuf オブジェクトをシリアル化するために Hot Rod Java クライアントが同梱された Protostream ライブラリーを使用することが推奨されます (ただし、これは強制ではなくサポートされていません)。Protobuf ライブラリーは、C++ 向けの推奨シリアル化ソリューションです (<https://code.google.com/p/protobuf/>)。

C++ と Hot Rod Java クライアント間の相互運用性は基本データタイプ、文字列、バイトアレイにおいて完全にサポートされています。Protobuf と Protostream はこれらのタイプの相互運用性のために必要ありません。

C++ クライアントが Protobuf エンコードデータを読み取る方法については、<https://github.com/jboss-developer/jboss-jdg-quickstarts/tree/master/remote-query/src/main/cpp> を参照してください。

[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14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](#)

### 14.1. 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14.1 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](#)

### 14.2. 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、各名前付きキャッシュでは、**namedCache** 要素内で設定されます。この設定の例は、以下のとおりです。

#### 手順14.2 ロックの設定 (ライブラリーモード)

```
<infinispan>
  <!-- Other configuration elements here -->
```

```
<default>
<locking concurrencyLevel="${VALUE}"
  isolationLevel="${LEVEL}"
  lockAcquisitionTimeout="${TIME}"
  useLockStriping="${TRUE/FALSE}"
  writeSkewCheck="${TRUE/FALSE}" />
```

1. **concurrencyLevel** パラメーターは、ロックコンテナ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並行スレッドの数に従ってこの値を設定します。
2. **isolationLevel** 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. **lockAcquisitionTimeout** 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. **useLockStriping** 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. **writeSkewCheck** パラメーターは、**isolationLevel** 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。**writeSkewCheck** 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](#)

## 14.3. ロックのタイプ

### 14.3.1. 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。**writeSkewCheck** 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](#)

### 14.3.2. 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](#)

### 14.3.3. 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ったと、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](#)

### 14.3.4. 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14.3 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](#)

### 14.3.5. 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14.4 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](#)

### 14.3.6. 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](#)

### 14.3.7. 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 transactionManagerLookupClass="
{TransactionManagerLookupClass}"
    transactionMode="{TRANSACTIONAL, NON_TRANSACTIONAL}"
    lockingMode="{OPTIMISTIC, PESSIMISTIC}"
    useSynchronization="true">
</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、**lockingMode** 値を **OPTIMISTIC** または **PESSIMISTIC** に設定します。

[バグを報告する](#)

## 14.4. ロック操作

### 14.4.1. 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:

- [15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](#)

#### 14.4.2. 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](#)

#### 14.4.3. 平行性レベルについて

平行性とは、データグリッドと同時に対話するスレッド数のことです。Red Hat JBoss Data Grid では、平行性レベルは、ロックコンテナ内で使用される同時スレッドの数のことです。

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15章 ロックストライピングのセットアップ

### 15.1. 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](#)

### 15.2. 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15.1 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](#)

### 15.3. 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 **useLockStriping** パラメーターを使用して行います。

### 手順15.1 ロックストライピングの設定 (ライブラリーモード)

```
<infinispan>
  <!-- Additional configuration elements here -->
  <default>

    <locking concurrencyLevel="${VALUE}"
      isolationLevel="${LEVEL}"
      lockAcquisitionTimeout="${TIME}"
      useLockStriping="${TRUE/FALSE}"
      writeSkewCheck="${TRUE/FALSE}" />
    <!-- Additional configuration elements here -->
  </default>
</infinispan>
```

1. **concurrencyLevel** 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. **isolationLevel** 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. **lockAcquisitionTimeout** 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. **useLockStriping** 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. **writeSkewCheck** は、異なるトランザクションからのエントリーへの変更によりトランザクションをロールバック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。**writeSkewCheck** および **isolation\_level** のデフォルト値はそれぞれ **FALSE** と **READ\_COMMITTED** です。**writeSkewCheck** 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16章 分離レベルのセットアップ

### 16.1. 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を見ることが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](#)

### 16.2. 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](#)

### 16.3. 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



が異なるときに発生する「反復不可能読み取り」が起こらないようにします (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であることが考えられます)。

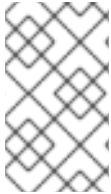
JBoss Data Grid の **REPEATABLE\_READ** 分離モードは、変更が発生する前にエントリーの値を保存します。これにより、同じエントリーの 2 番目の読み取り操作は変更された新しい値ではなく、保存された値を読み取るため、「反復不可能読み取り」の発生を防ぐことができます。そのため、読み取りの間に別のトランザクションで書き込み操作が発生しても、1 つのトランザクションで 2 つの読み取り操作によって取得される 2 つの値は常に同じになります。

[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17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の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](#)

### 17.1. 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。

`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

以前に、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 (ただし、これは引き続き使用されています)。

[バグを報告する](#)

### 17.2. キャッシュストアの設定

#### 17.2.1. 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](#)

### 17.2.2. 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">
  <singleFile shared="false"
    preload="true"
    fetchPersistentState="true"
    ignoreModifications="false"
    purgeOnStartup="false"
    location="${java.io.tmpdir}" >
  <async enabled="true"
    flushLockTimeout="15000"
    threadPoolSize="5" />
  <singleton enabled="true"
    pushStateWhenCoordinator="true"
    pushStateTimeout="20000" />
</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 \(ライブラリーモード\)](#)」を参照してください。

[バグを報告する](#)

### 17.2.3. プログラムを使用してキャッシュストアを設定

以下の例では、プログラムを使用してキャッシュストアを設定する方法を示します。

```
ConfigurationBuilder builder = new ConfigurationBuilder();
builder.persistence()
    .passivation(false)
    .addSingleFileStore()
        .shared(false)
        .preload(true)
        .fetchPersistentState(true)
        .ignoreModifications(false)
        .purgeOnStartup(false)
        .location(System.getProperty("java.io.tmpdir"))
    .async()
        .enabled(true)
        .flushLockTimeout(15000)
        .threadPoolSize(5)
    .singleton()
        .enabled(true)
        .pushStateWhenCoordinator(true)
        .pushStateTimeout(20000);
```



## 注記

この設定は単一ファイルキャッシュストア用です。**location**などの一部の属性は単一ファイルキャッシュストアに固有であり、他の種類のキャッシュストアには使用されません。

### 手順17.1 プログラムを使用してキャッシュストアを設定

1. **ConfigurationBuilder** を使用して、新規の設定オブジェクトを作成します。
2. **passivation** 要素は Red Hat JBoss Data Grid がストアと通信する方法に影響を与えます。パッシベーションは、インメモリーキャッシュからオブジェクトを削除し、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みます。パッシベーションはデフォルトで **false** です。
3. **addSingleFileStore()** 要素は、この設定用のキャッシュストアとして **SingleFileStore** を追加します。**addStore** メソッドを使用して追加できる、**JDBC** キャッシュストアなどの他のストアを作成することができます。
4. **shared** パラメーターは、キャッシュストアが異なるキャッシュインスタンスによって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。
5. **preload** 要素はデフォルトでは **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延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。**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をプリロードします。
6. **fetchPersistentState** 要素は、キャッシュの永続状態をフェッチするかどうかを決定し、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、フェッチ永続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** に設定された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。**fetchPersistentState** プロパティーはデフォルトでは **false** です。
7. **ignoreModifications** 要素は、書き込み操作を共有キャッシュストアではなく、ローカルファイルキャッシュストアに許可することで、書き込みメソッドを特定のキャッシュストアにプッシュするかどうかを決定します。場合によっては、一時的なアプリケーションデータが、インメモリーキャッシュと同じサーバー上のファイルベースのキャッシュストアにのみ存在する必要があります。たとえば、これはネットワーク内のすべてのサーバーによって使用される追加の **JDBC** ベースのキャッシュストアで適用されます。**ignoreModifications** はデフォルトでは **false** です。
8. **purgeOnStartup** 要素は、キャッシュストアの起動時にキャッシュストアをパージするかどうかを制御し、デフォルトでは **false** になります。
9. **location** 要素設定は、ストアが書き込みできるディスクの場所を設定します。

10. これらの属性は、それぞれのキャッシュストアに固有の側面を設定します。例えば、**location** 属性は、**SingleFileStore** がデータが含まれるファイルを維持する場所を指します。他のストアには、さらに複雑な設定が必要な場合があります。
11. **singletonStore** 要素を使用すると、クラスター内の1つのノードのみで変更を保存できます。このノードはコーディネーターと呼ばれます。コーディネーターは、インメモリ状態のキャッシュをディスクにプッシュします。この機能は、すべてのノードの **enabled** 属性を **true** に設定することによりアクティベートされます。**shared** パラメーターは、**singleton** を同時に有効にした状態で定義することはできません。**enabled** 属性はデフォルトでは **false** です。
12. **pushStateWhenCoordinator** 要素はデフォルトでは **true** に設定されます。**true** の場合、このプロパティにより、コーディネーターになったノードがインメモリ状態を基礎となるキャッシュストアに転送します。このパラメーターは、コーディネーターがクラッシュし、新規のコーディネーターが選択される場合に役に立ちます。

[バグを報告する](#)

#### 17.2.4. 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ブロードキャストされ、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](#)

### 17.3. 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](#)

#### 17.3.1. 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](#)

### 17.3.2. 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](#)

### 17.3.3. 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](#)

## 17.4. 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](#)

### 17.4.1. 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](#)

### 17.4.2. 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](#)

## 第18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](#)

### 18.1. 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。キーがメモリー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ラビリティ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せん。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](#)

### 18.2. キャッシュストア設定の詳細 (ライブラリーモード)

以下のリストには、JBoss Data Grid のライブラリーモード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。

#### namedCache 要素

- **name** パラメーターに名前の値を追加してキャッシュストアの名前を設定します。



## 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。

## singleFile 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。
- **fetchPersistentState** パラメーターは、キャッシュの永続状態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。**fetchPersistentState** プロパティーはデフォルトでは **false** です。
- **ignoreModifications** パラメーターは、変更メソッドが特定のキャッシュストアに適用されるかどうかを決定します。これにより、書き込み操作を共有キャッシュストアではなく、ローカルファイルキャッシュストアに適用できます。一時的なアプリケーションデータは、インメモリーキャッシュと同じサーバー上のファイルベースのキャッシュストアにのみ存在する必要がある場合があります。たとえば、これはネットワーク内のすべてのサーバーによって使用される追加の JDBC ベースのキャッシュストアで適用されます。**ignoreModifications** はデフォルトでは **false** です。
- **maxEntries** 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。
- **maxKeysInMemory** パラメーターは、データルックアップを迅速化するために使用されます。単一ファイルストアは、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、**maxKeysInMemory** パラメーターを使用してインデックスのサイズが制限されます。このパラメーターのデフォルト値は -1 です。
- **purgeOnStartup** パラメーターは、キャッシュストアの起動時にキャッシュストアがリパージされるかどうかを制御します。
- The **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。

## async 要素

**async** 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。

- **enabled** パラメーターは、ファイルストアを非同期にするかどうかを決定します。
- **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 **1** です。
- **flushLockTimeout** 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す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。
- **modificationQueueSize** 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、キューがさらに多くの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。
- **shutdownTimeout** 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 **25** 秒です。

### singleton 要素

**singleton** 要素を使用すると、クラスター内の1つのノードによってのみ変更を保存できます。このノードはコーディネーターと呼ばれます。コーディネーターは、インメモリー状態のキャッシュをディスクにプッシュします。**shared** 要素は、**singleton** を同時に有効にした状態で定義することはできません。

- **enabled** 属性は、この機能を有効にするかどうかを決定します。このパラメーターの有効な値は **true** と **false** です。**enabled** 属性は、デフォルトで **false** に設定されます。
- **pushStateWhenCoordinator** パラメーターは、デフォルトで **true** に設定されます。**true** の場合は、このプロパティにより、コーディネーターになったノードが、インメモリー状態を基礎となるキャッシュストアに転送します。このパラメーターは、コーディネーターがクラッシュし、新規のコーディネーターが選択される場合に役に立ちます。
- **pushStateWhenCoordinator** が **true** に設定された場合、**pushStateTimeout** パラメーターはインメモリー状態を基礎となるキャッシュローダーにプッシュするプロセスが完了するのにかかる最大ミリ秒数を設定します。このパラメーターのデフォルトの時間は **10** 秒です。

### remoteStore 要素

- **remoteCacheName** 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。
- **fetchPersistentState** 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ステートが取り込まれます。複数のキャッシュストアがチェーンされている場合は、1つのキャッシュストアだけでこのプロパティを **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避けられます。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** に設定する欠点は、起動時間が増えることです。この属性のデフォルト値は **false** です。

- **ignoreModifications** 属性を使用すると、キャッシュを変更する操作 (配置、削除、消去、保存など) がキャッシュストアに影響を与えることを防ぐことができます。この結果、キャッシュストアがキャッシュと同期しなくなることがあります。この属性のデフォルト値は **false** です。
- **purgeOnStartup** 属性を使用すると、キャッシュストアが起動プロセス時にパージされます。この属性のデフォルト値は **false** です。
- **tcpNoDelay** 属性を使用すると、**TCP\_NODELAY** スタックがトリガーされます。この属性のデフォルト値は **true** です。
- **pingOnStartup** 属性を使用すると、クラスタスタートポロジータを取り込むために、**ping** 要求がバックエンドサーバーに送信されます。この属性のデフォルト値は **true** です。
- **keySizeEstimate** 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。
- **valueSizeEstimate** 属性は、値をシリアルライズおよびデシリアルライズする時のバイトバッファ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。
- **forceReturnValues** 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。

### servers 要素および server 要素

複数のサーバーのサーバー情報をセットアップするために **remoteStore** 要素内に **servers** 要素を作成します。単一サーバーの情報を追加するには、一般的な **servers** 要素内に **server** 要素を追加します。

- **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われるポートを設定します。

### connectionPool 要素

- **maxActive** 属性は、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。
- **maxIdle** 属性は、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。
- **maxTotal** 属性は、組み合わされたサーバーセット内の永続的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。
- **connectionUrl** 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、**connectionUrl** 経由で接続するために使用されるユーザー名が含まれます。
- **driverClass** 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。

### levelDbStore 要素

- **location** パラメーターは、プライマリーキャッシュストアを格納する場所を指定します。ディレクトリは、存在しない場合に自動的に作成されます。

- **expiredLocation** パラメーターは、期限切れデータ用の場所を指定します。ディレクトリーには、ページされる前の期限切れデータが含まれます。ディレクトリーは、存在しない場合に自動的に作成されます。
- **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 キャッシュストアでこのパラメーターに対し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。

#### jpaStore 要素

- **persistenceUnitName** 属性は、JPA キャッシュストアの名前を指定します。
- **entityClassName** 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。
- **batchSize** (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。
- **storeMetadata** (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。

#### binaryKeyedJdbcStore 要素、stringKeyedJdbcStore 要素、および mixedKeyedJdbcStore 要素

- **fetchPersistentState** 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。**fetchPersistentState** パラメーターはデフォルトでは **false** に設定されます。
- **ignoreModifications** パラメーターは、キャッシュを変更する操作 (例: 配置、削除、消去、保存など) がキャッシュストアに影響を与えるかどうかを決定します。結果として、キャッシュストアは、キャッシュと同期しなくなります。
- **purgeOnStartup** パラメーターは、初回起動時にキャッシュがページされるかどうかを指定します。
- **key2StringMapper** 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 Key2StringMapper のクラス名を指定します。

#### binaryKeyedTable 要素および stringKeyedTable 要素

- **dropOnExit** 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。
- **createOnStart** 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。
- **prefix** パラメーターは、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。

#### idColumn 要素、dataColumn 要素、および timestampColumn 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。

#### 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

- **name** パラメーターは、プロパティの名前を指定します。
- **value** パラメーターは、プロパティの値を指定します。

#### [バグを報告する](#)

### 18.3. 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。

#### 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** プロパティの値に追加される相対パス名です。
- **maxEntries** 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。

#### 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、プロパティの名前を指定します。
- **value** パラメーターは、プロパティ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。
- **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。

#### 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任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

- **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パス内のディレクトリーを指定します。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。
- **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](#)

## 18.4. 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 **Infinispan** がキャッシュとして使用される場合にのみ機能します。**Infinispan** がこのように使用されると、**Infinispan** 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、**Infinispan** 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、**Infinispan** が正式なデータストアとして使用される場合、このユースケースでは望ましくない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](#)

#### 18.4.1. 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](#)

#### 18.4.2. 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。

```
<namedCache name="writeThroughToFile">
  <persistence passivation="false">
    <singleFile fetchPersistentState="true"
      ignoreModifications="false"
```



```

        purgeOnStartup="false"
        shared="false"
        preload="false"
        location="/tmp/Another-FileCacheStore-Location"
        maxEntries="100"
        maxKeysInMemory="100">
    <async enabled="true"
        threadPoolSize="500"
        flushLockTimeout="1"
        modificationQueueSize="1024"
        shutdownTimeout="25000"/>
</singleFile>
</persistence>
</namedCache>

```

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](#)

### 18.4.3. FileCacheStore から SingleFileCacheStore へのデータ移行

Red Hat JBoss Data Grid 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。ライブラリモードでは、JBoss Data Grid がライブラリモードで実行されるノードのインストールを指します。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利用できます。

[バグを報告する](#)

## 18.5. 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、ページの前に期限の切れたキーを保存します。

[バグを報告する](#)

### 18.5.1. 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18.1 LevelDB キャッシュストアを設定するには、以下を実行します。

- 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" >
  <expiration path="/path/to/leveldb/expires/data" />
  <implementation type="JNI" />
</leveldb-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](#)

### 18.5.2. LevelDB キャッシュストアのプログラムを使用した設定

以下は、LevelDB キャッシュストアの、プログラムを使用した設定例です。

```
Configuration cacheConfig = new ConfigurationBuilder().persistence()
    .addStore(LevelDBStoreConfigurationBuilder.class)
    .location("/tmp/leveldb/data")
    .expiredLocation("/tmp/leveldb/expired").build();
```

#### 手順18.2 LevelDB キャッシュストアのプログラムを使用した設定

1. **ConfigurationBuilder** を使用して、新規の設定オブジェクトを作成します。
2. **LevelDBCacheStoreConfigurationBuilder** クラスを使用してストアを追加して、その設定を構築します。
3. LevelDB キャッシュストアのロケーションパスを設定します。指定したパスは、主なキャッシュストアデータを保存します。ディレクトリがない場合は自動的に作成されます。
4. LevelDB スタアの **expiredLocation** パラメーターを使用して、期限切れデータのロケーションを指定します。指定されたパスは、ページされる前に期限切れデータを保存します。ディレクトリがない場合は自動的に作成されます。



#### 注記

プログラムを使用した設定は、Red Hat JBoss Data Grid ライブラリーモードのみで利用できます。

[バグを報告する](#)

### 18.5.3. 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。

```
<namedCache name="vehicleCache">
  <persistence passivation="false">
    <leveldbStore xmlns="urn:infinispan:config:store:leveldb:6.0"
                  location="/path/to/leveldb/data">
```

```

expiredLocation="/path/to/expired/data"
shared="false"
preload="true"/>
</persistence>
</namedCache>

```

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](#)

## 18.5.4. 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18.3

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。

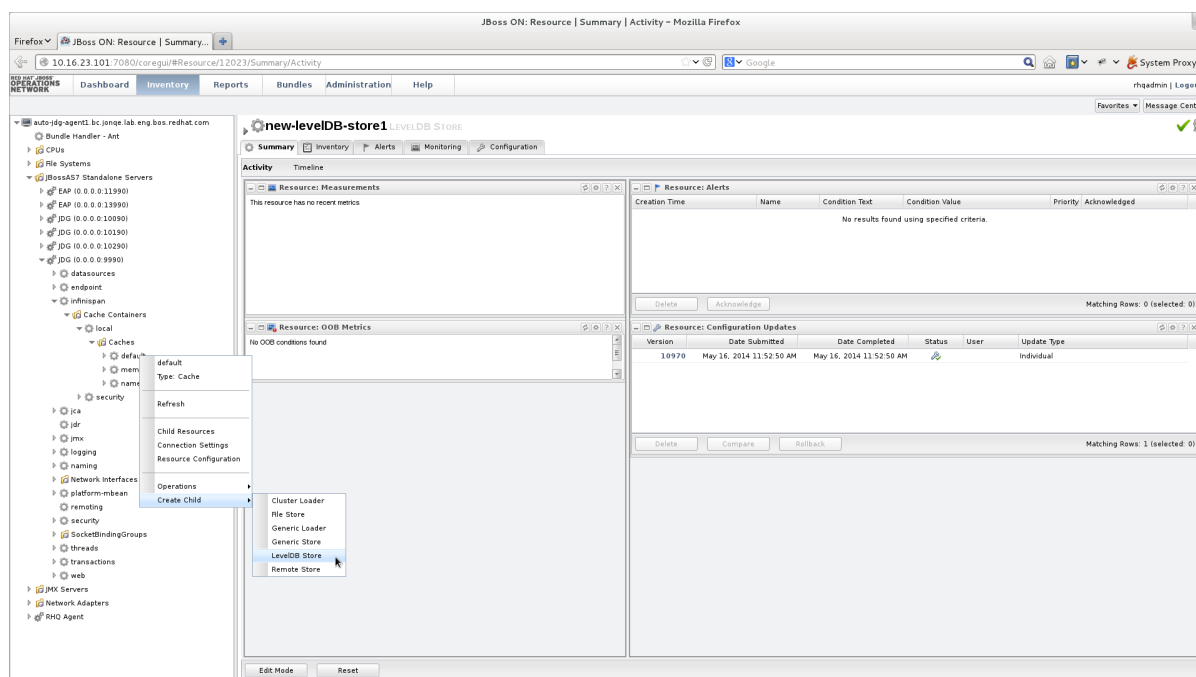


図18.1 新しい LevelDB キャッシュストアの作成

- 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。

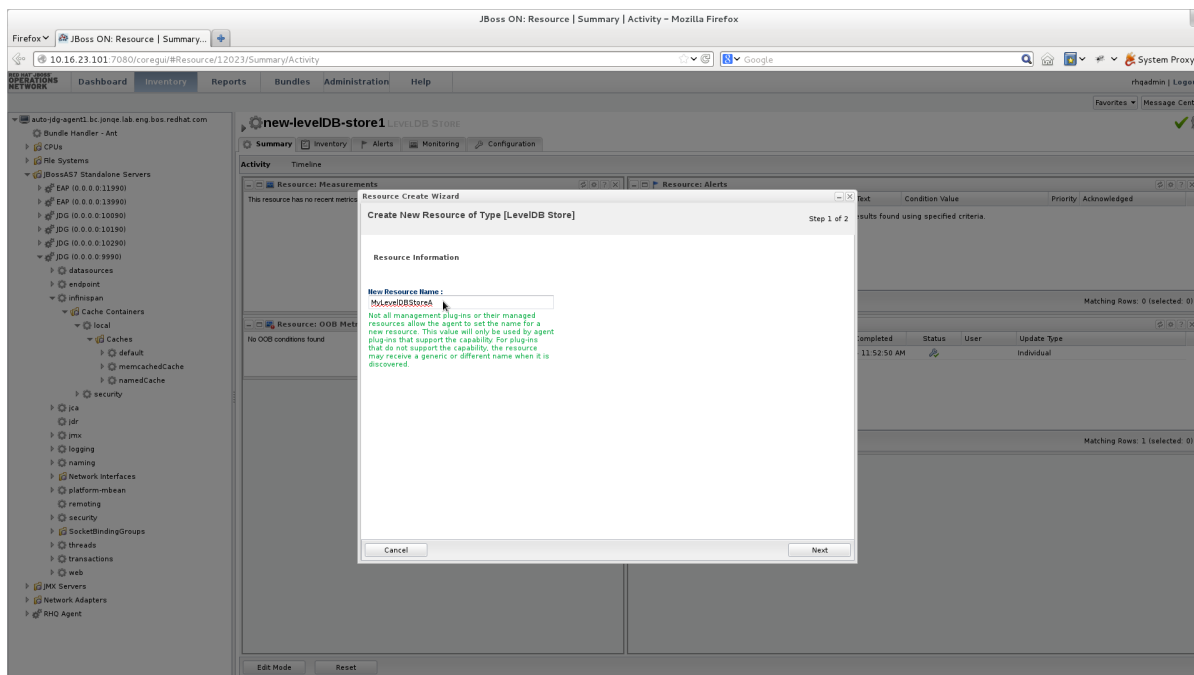


図18.2 新しい LevelDB キャッシュストアの名前指定

- 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続けます。
8. 以下のように LevelDB キャッシュストアを設定します。

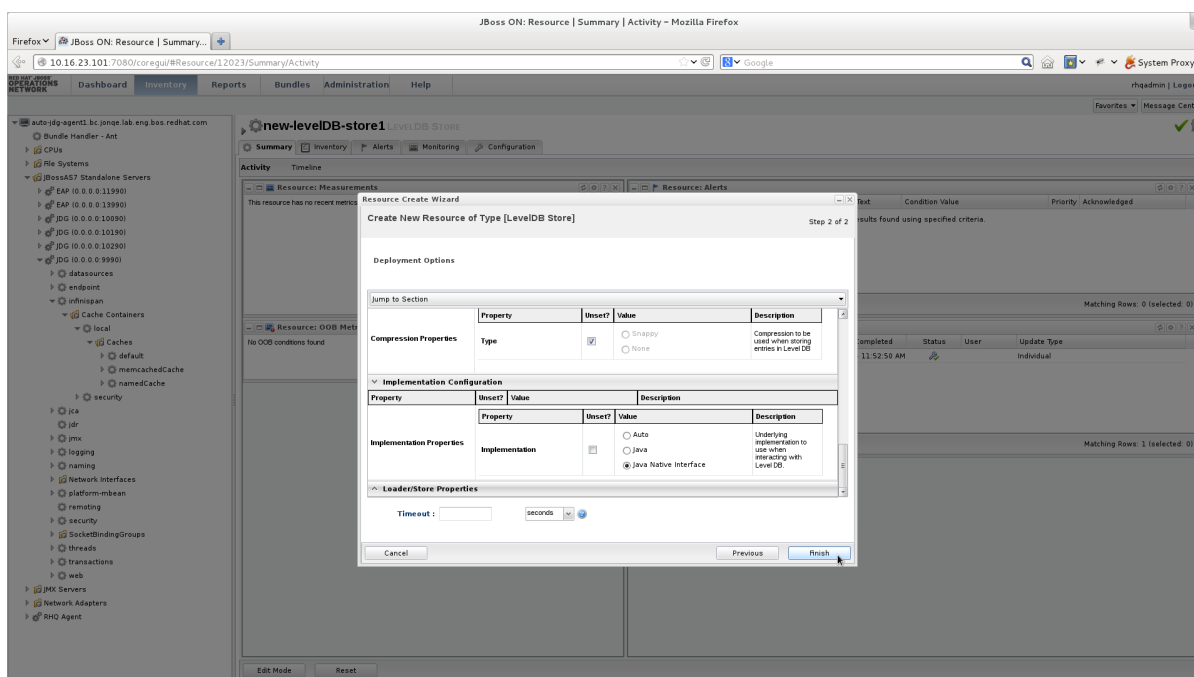


図18.3 LevelDB キャッシュストアの設定

- 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。

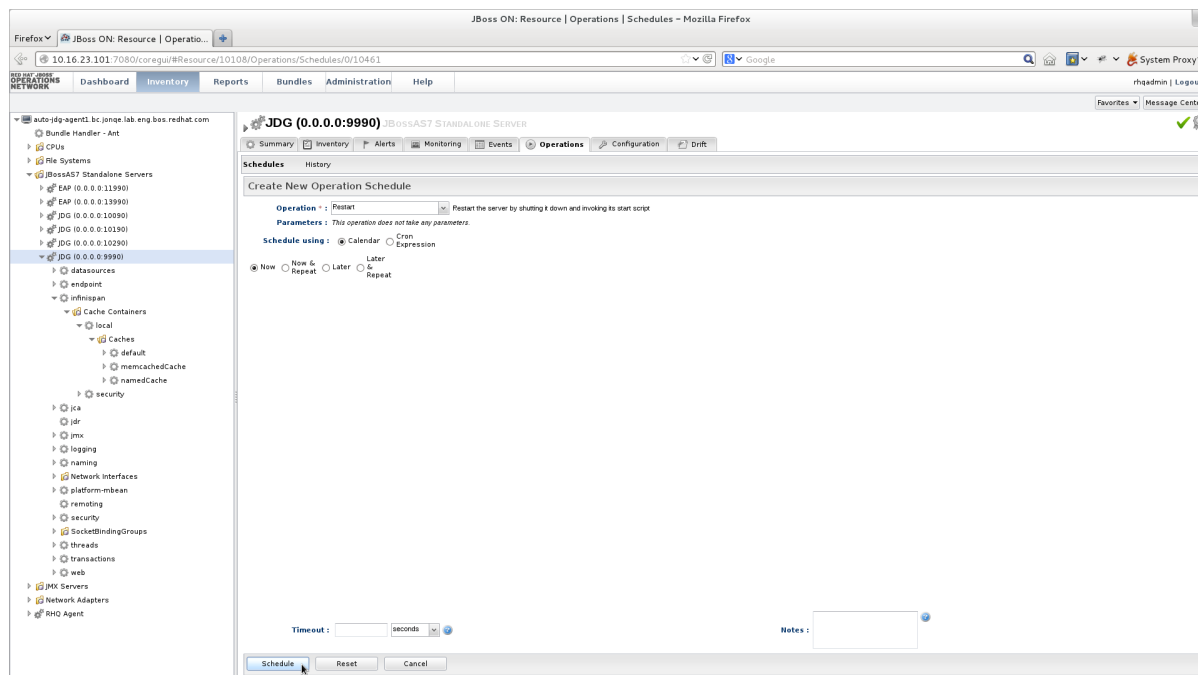


図18.4 再起動操作のスケジュール

- 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。

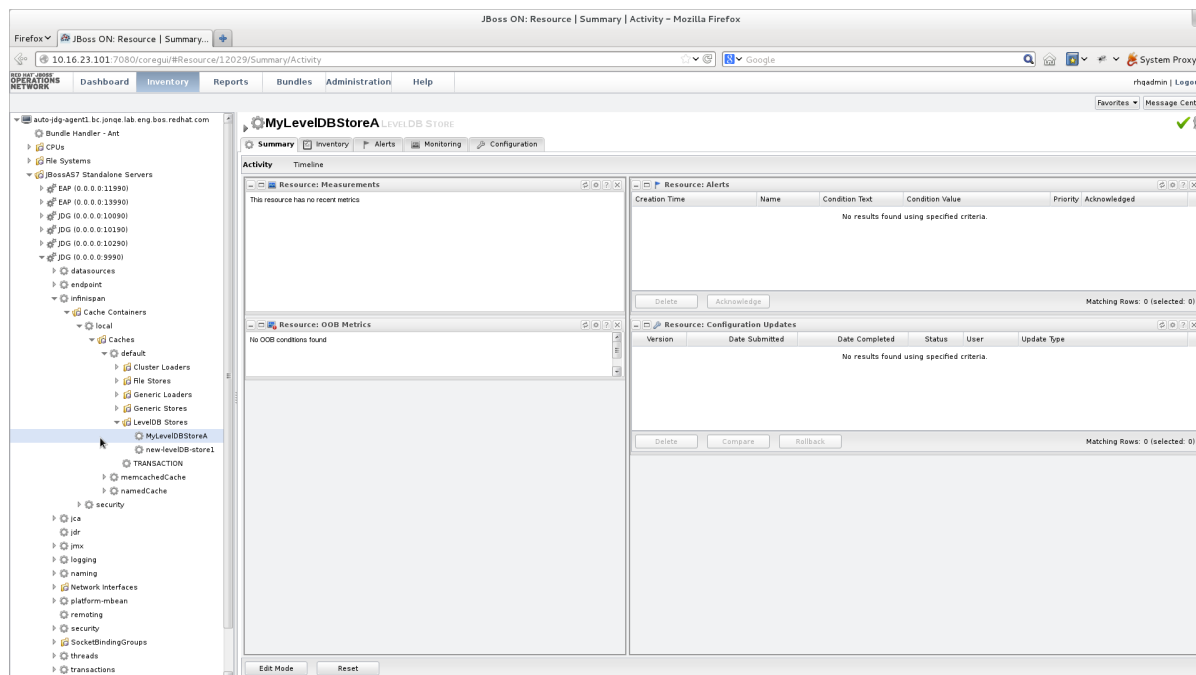


図18.5 新しい LevelDB キャッシュストアの検出

- 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 Standalone Servers** → **JDG (0.0.0.0:9990)** → **infinispan** → **Cache Containers** → **local** → **Caches** → **default** → **LevelDB Stores**
- 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

## 18.6. 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

### 18.6.1. 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、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](#)

### 18.6.1.1. 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](#)

### 18.6.1.2. 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:6.0
http://www.infinispan.org/schemas/infinispan-config-6.0.xsd
urn:infinispan:config:jdbc:6.0
http://www.infinispan.org/schemas/infinispan-cachestore-jdbc-config-6.0.xsd"
  xmlns="urn:infinispan:config:6.0">
  <!-- Additional configuration elements here -->
  <persistence>
    <binaryKeyedJdbcStore xmlns="urn:infinispan:config:jdbc:6.0"
      fetchPersistentState="false"
      ignoreModifications="false"
      purgeOnStartup="false">
      <connectionPool
        connectionUrl="jdbc:h2:mem:infinispan_binary_based;DB_CLOSE_DELAY=-1"
```

```

        username="sa"
        driverClass="org.h2.Driver"/>
<binaryKeyedTable dropOnExit="true"
    createOnStart="true"
    prefix="ISPN_BUCKET_TABLE">
    <idColumn name="ID_COLUMN"
        type="VARCHAR(255)" />
    <dataColumn name="DATA_COLUMN"
        type="BINARY" />
    <timestampColumn name="TIMESTAMP_COLUMN"
        type="BIGINT" />
</binaryKeyedTable>
</binaryKeyedJdbcStore>
</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](#)

### 18.6.1.3. JdbcBinaryStore のプログラムを用いた設定

以下は、**JdbcBinaryStore** の設定例です。

```

ConfigurationBuilder builder = new ConfigurationBuilder();
builder.persistence()
    .addStore(JdbcBinaryStoreConfigurationBuilder.class)
    .fetchPersistentState(false)
    .ignoreModifications(false)
    .purgeOnStartup(false)
    .table()
        .dropOnExit(true)
        .createOnStart(true)
        .tableNamePrefix("ISPN_BUCKET_TABLE")
        .idColumnName("ID_COLUMN").idColumnType("VARCHAR(255)")
        .dataColumnName("DATA_COLUMN").dataColumnType("BINARY")

    .timestampColumnName("TIMESTAMP_COLUMN").timestampColumnType("BIGINT")
    .connectionPool()

    .connectionUrl("jdbc:h2:mem:infinispan_binary_based;DB_CLOSE_DELAY=-1")
        .username("sa")
        .driverClass("org.h2.Driver");

```

### 手順18.4 JdbcBinaryStore のプログラムを用いた設定（ライブラリーモード）

1. **ConfigurationBuilder** を使用して、新規の設定オブジェクトを作成します。
2. このストアに関連する特定の設定を構築するには、**JdbcBinaryStore** 設定ビルダーを追加します。
3. **fetchPersistentState** 要素は、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。複数のキャッシュローダー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。**fetchPersistentState** プロパティーはデフォルトでは **false** です。

4. **ignoreModifications** 要素は、書き込み操作を共有キャッシュローダーではなく、ローカルファイルキャッシュローダーに許可することで、書き込みメソッドを特定のキャッシュローダーにプッシュするかどうかを決定します。場合によっては、一時的なアプリケーションデータが、インメモリーキャッシュと同じサーバー上のファイルベースのキャッシュローダーにのみ存在する必要があります。たとえば、これはネットワーク内のすべてのサーバーによって使用される追加の JDBC ベースのキャッシュローダーで適用されます。**ignoreModifications** はデフォルトでは **false** になります。
5. **purgeOnStartup** 要素は、初回起動時にキャッシュがパージされるかどうかを指定します。
6. テーブルを以下のように設定します。
  - a. **dropOnExit** は、キャッシュストアが停止している際にテーブルを破棄するかどうかを決定します。これは、デフォルトでは **false** に設定されます。
  - b. **createOnStart** は、現在テーブルが存在しない場合にキャッシュストアを起動すると、テーブルを作成します。このメソッドはデフォルトでは **true** です。
  - c. **tableNamePrefix** は、データが保存されるテーブルの名前にプレフィックスを設定します。
  - d. **idColumnName** プロパティーは、キャッシュキーまたはバケット ID が保存される列を定義します。
  - e. **dataColumnName** プロパティーは、キャッシュエントリまたはバケット ID が保存される列を指定します。
  - f. **timestampColumnName** 要素は、キャッシュエントリのタイムスタンプまたはバケットが保存される列を指定します。
7. **connectionPool** 要素は、次のパラメーターを使用して JDBC ドライバーの接続プールを指定します。
  - a. **connectionUrl** 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。
  - b. **username** パラメーターには、**connectionUrl** 経由で接続するために使用されるユーザー名が含まれます。
  - c. **driverClass** 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。



#### 注記

プログラムを使用した設定は、Red Hat JBoss Data Grid ライブラリーモードのみで利用できます。

[バグを報告する](#)

### 18.6.2. JdbcStringBasedStores

**JdbcStringBasedStore** は複数のエントリを各行にグループ化せずに、各エントリをテーブルの

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](#)

### 18.6.2.1. 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](#)

### 18.6.2.2. 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:6.0
    http://www.infinispan.org/schemas/infinispan-config-6.0.xsd
    urn:infinispan:config:jdbc:6.0
    http://www.infinispan.org/schemas/infinispan-cachestore-jdbc-config-6.0.xsd"
  xmlns="urn:infinispan:config:6.0">
  <!-- Additional configuration elements here -->
  <persistence>
    <stringKeyedJdbcStore xmlns="urn:infinispan:config:jdbc:6.0"
      fetchPersistentState="false"
      ignoreModifications="false"
```

```

        purgeOnStartup="false"

key2StringMapper="org.infinispan.loaders.keymappers.DefaultTwoWayKey2StringMapper">
    <connectionPool
connectionUrl="jdbc:h2:mem:infinispan_binary_based;DB_CLOSE_DELAY=-1"
    username="sa"
    driverClass="org.h2.Driver"/>
    <stringKeyedTable dropOnExit="true"
        createOnStart="true"
        prefix="ISPN_STRING_TABLE">
        <idColumn name="ID_COLUMN"
            type="VARCHAR(255)" />
        <dataColumn name="DATA_COLUMN"
            type="BINARY" />
        <timestampColumn name="TIMESTAMP_COLUMN"
            type="BIGINT" />
    </stringKeyedTable>
</stringKeyedJdbcStore>
</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](#)

### 18.6.2.3. 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:6.1" 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](#)

#### 18.6.2.4. JdbcStringBasedStore のプログラムを使用した設定

**JdbcStringBasedStore** の設定例は次のとおりです。

```

ConfigurationBuilder builder = new ConfigurationBuilder();

builder.persistence().addStore(JdbcStringBasedStoreConfigurationBuilder.class)
    .fetchPersistentState(false)
    .ignoreModifications(false)
    .purgeOnStartup(false)
    .table()
        .dropOnExit(true)
        .createOnStart(true)
        .tableNamePrefix("ISPN_STRING_TABLE")
        .idColumnName("ID_COLUMN").idColumnType("VARCHAR(255)")
        .dataColumnName("DATA_COLUMN").dataColumnType("BINARY")

    .timestampColumnName("TIMESTAMP_COLUMN").timestampColumnType("BIGINT")
    .connectionPool()

    .connectionUrl("jdbc:h2:mem:infinispan_binary_based;DB_CLOSE_DELAY=-1")
        .username("sa")
        .driverClass("org.h2.Driver");

```

#### 手順18.5 プログラムを使用した JdbcStringBasedStore の設定

1. **ConfigurationBuilder** を使用して、新規の設定オブジェクトを作成します。
2. このストアに関連する特定の設定を構築するには **JdbcStringBasedStore** 設定ビルダーを追加します。
3. **fetchPersistentState** パラメーターは、キャッシュの永続状態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。複数のキャッシュローダーでこのプロパティが **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。**fetchPersistentState** プロパティはデフォルトでは **false** です。
4. **ignoreModifications** パラメーターは、書き込み操作を共有キャッシュローダーではなく、ローカルファイルキャッシュローダーに許可することで、書き込みメソッドを特定のキャッシュローダーにプッシュするかどうかを決定します。場合によっては、一時的なアプリケーションデータが、インメモリーキャッシュと同じサーバー上のファイルベースのキャッシュローダーにのみ存在する必要があります。たとえば、これはネットワーク内のすべてのサーバーによって使用される追加の JDBC ベースのキャッシュローダーで適用されません。**ignoreModifications** はデフォルトでは **false** です。

5. **purgeOnStartup** パラメーターは、初回起動時にキャッシュがパージされるかどうかを指定します。
6. テーブルを以下のように設定します。
  - a. **dropOnExit** は、キャッシュストアが停止している際にテーブルを破棄するかどうかを決定します。これは、デフォルトでは **false** に設定されます。
  - b. **createOnStart** は、現在テーブルが存在しない場合にキャッシュストアを起動すると、テーブルを作成します。このメソッドはデフォルトでは **true** です。
  - c. **tableNamePrefix** は、データが保存されるテーブルの名前にプレフィックスを設定します。
  - d. **idColumnName** プロパティは、キャッシュキーまたはバケット ID が保存される列を定義します。
  - e. **dataColumnName** プロパティは、キャッシュエントリまたはバケット ID が保存される列を指定します。
  - f. **timestampColumnName** 要素は、キャッシュエントリのタイムスタンプまたはバケットが保存される列を指定します。
7. **connectionPool** 要素は、次のパラメーターを使用して JDBC ドライバーの接続プールを指定します。
  - a. **connectionUrl** 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。
  - b. **username** パラメーターには、**connectionUrl** 経由で接続するために使用されるユーザー名が含まれます。
  - c. **driverClass** 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。



#### 注記

プログラムを使用した設定は、Red Hat JBoss Data Grid ライブラリーモードのみで利用できます。

[バグを報告する](#)

### 18.6.3. 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](#)

#### 18.6.3.1. 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](#)

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

`mixedKeyedJdbcStore` 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:6.0
http://www.infinispan.org/schemas/infinispan-config-6.0.xsd
urn:infinispan:config:jdbc:6.0
http://www.infinispan.org/schemas/infinispan-cachestore-jdbc-config-6.0.xsd"
  xmlns="urn:infinispan:config:6.0">
  <!-- Additional configuration elements here -->
  <persistence>
    <mixedKeyedJdbcStore xmlns="urn:infinispan:config:jdbc:6.0"
      fetchPersistentState="false"
      ignoreModifications="false"
      purgeOnStartup="false"

key2StringMapper="org.infinispan.persistence.keymappers.DefaultTwoWayKey2S
tringMapper">
    <connectionPool
connectionUrl="jdbc:h2:mem:infinispan_binary_based;DB_CLOSE_DELAY=-1"
      username="sa"
      driverClass="org.h2.Driver"/>
    <binaryKeyedTable dropOnExit="true"
      createOnStart="true"
      prefix="ISPN_BUCKET_TABLE_BINARY">

```

```

<idColumn name="ID_COLUMN"
  type="VARCHAR(255)" />
<dataColumn name="DATA_COLUMN"
  type="BINARY" />
<timestampColumn name="TIMESTAMP_COLUMN"
  type="BIGINT" />
</binaryKeyedTable>
<stringKeyedTable dropOnExit="true"
  createOnStart="true"
  prefix="ISPN_BUCKET_TABLE_STRING">
  <idColumn name="ID_COLUMN"
    type="VARCHAR(255)" />
  <dataColumn name="DATA_COLUMN"
    type="BINARY" />
  <timestampColumn name="TIMESTAMP_COLUMN"
    type="BIGINT" />
</stringKeyedTable>
</mixedKeyedJdbcStore>
</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](#)

### 18.6.3.3. JdbcMixedStore のプログラムを使用した設定

以下は、**JdbcMixedStore** の設定例です。

```

ConfigurationBuilder builder = new ConfigurationBuilder();
builder.persistence().addStore(JdbcMixedStoreConfigurationBuilder.class)
  .fetchPersistentState(false)
  .ignoreModifications(false)
  .purgeOnStartup(false)
  .stringTable()
    .dropOnExit(true)
    .createOnStart(true)
    .tableNamePrefix("ISPN_MIXED_STR_TABLE")
    .idColumnName("ID_COLUMN").idColumnType("VARCHAR(255)")
    .dataColumnName("DATA_COLUMN").dataColumnType("BINARY")

  .timestampColumnName("TIMESTAMP_COLUMN").timestampColumnType("BIGINT")
    .binaryTable()
      .dropOnExit(true)
      .createOnStart(true)
      .tableNamePrefix("ISPN_MIXED_BINARY_TABLE")
      .idColumnName("ID_COLUMN").idColumnType("VARCHAR(255)")
      .dataColumnName("DATA_COLUMN").dataColumnType("BINARY")

  .timestampColumnName("TIMESTAMP_COLUMN").timestampColumnType("BIGINT")
    .connectionPool()

  .connectionUrl("jdbc:h2:mem:infinispan_binary_based;DB_CLOSE_DELAY=-1")
    .username("sa")
    .driverClass("org.h2.Driver");

```



## 手順18.6 プログラムを使用した JdbcMixedStore の設定

1. **ConfigurationBuilder** を使用して、新規の設定オブジェクトを作成します。
2. このストアに関連する特定の設定を構築するには、**JdbcMixedStore** 設定ビルダーを追加します。
3. **fetchPersistentState** 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。複数のキャッシュローダーでこのプロパティが **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。**fetchPersistentState** プロパティはデフォルトで **false** になります。
4. **ignoreModifications** パラメーターは、書き込み操作を共有キャッシュローダーではなく、ローカルファイルキャッシュローダーに許可することで、書き込みメソッドを特定のキャッシュローダーにプッシュするかどうかを決定します。場合によっては、一時的なアプリケーションデータが、インメモリーキャッシュと同じサーバー上のファイルベースのキャッシュローダーにのみ存在する必要があります。たとえば、これはネットワーク内のすべてのサーバーによって使用される追加の JDBC ベースのキャッシュローダーで適用されます。**ignoreModifications** はデフォルトでは **false** です。
5. **purgeOnStartup** パラメーターは、初回起動時にキャッシュがパージされるかどうかを指定します。
6. テーブルを以下のように設定します。
  - a. **dropOnExit** は、キャッシュストアが停止している際にテーブルを破棄するかどうかを決定します。これは、デフォルトでは **false** に設定されます。
  - b. **createOnStart** は、現在テーブルが存在しない場合にキャッシュストアを起動すると、テーブルを作成します。このメソッドはデフォルトでは **true** です。
  - c. **tableNamePrefix** は、データが保存されるテーブルの名前にプレフィックスを設定します。
  - d. **idColumnName** プロパティは、キャッシュキーまたはバケット ID が保存される列を定義します。
  - e. **dataColumnName** プロパティは、キャッシュエントリまたはバケット ID が保存される列を指定します。
  - f. **timestampColumnName** 要素は、キャッシュエントリのタイムスタンプまたはバケットが保存される列を指定します。
7. **connectionPool** 要素は、次のパラメーターを使用して JDBC ドライバーの接続プールを指定します。
  - a. **connectionUrl** 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。
  - b. **username** パラメーターには、**connectionUrl** 経由で接続するために使用されるユーザー名が含まれます。
  - c. **driverClass** 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。





## 注記

プログラムを使用した設定は、Red Hat JBoss Data Grid ライブラリーモードのみで利用できます。

[バグを報告する](#)

### 18.6.4. キャッシュストアのトラブルシューティング

#### 18.6.4.1. 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せん。

[バグを報告する](#)

### 18.7. 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レランスを提供します。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](#)

#### 18.7.1. 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](#)

#### 18.7.2. 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">
```

```

<remoteStore xmlns="urn:infinispan:config:remote:6.0"
    remoteCacheName="default"
    fetchPersistentState="false"
    shared="true"
    preload="false"
    ignoreModifications="false"
    purgeOnStartup="false"
    tcpNoDelay="true"
    pingOnStartup="true"
    keySizeEstimate="62"
    valueSizeEstimate="512"
    forceReturnValues="false">
  <servers>
    <server host="127.0.0.1"
      port="19711"/>
  </servers>
  <connectionPool maxActive="99"
    maxIdle="97"
    maxTotal="98" />
</remoteStore>
</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](#)

### 18.7.3. 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18.1 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](#)

## 18.8. 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](#)

#### 18.8.1. 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。

```
<namedCache name="users">
  <!-- Insert additional configuration elements here -->
  <persistence passivation="false">
    <jpaStore xmlns="urn:infinispan:config:jpa:6.0"
      shared="true"
      preload="true"
      persistenceUnitName="MyPersistenceUnit"

entityClassName="org.infinispan.loaders.jpa.entity.User" />
  </persistence>
</namedCache>
```

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 \(ライブラリーモード\)](#)」を参照してください。

[バグを報告する](#)

#### 18.8.2. JPA キャッシュストアのプログラミングによる設定例

**Red Hat JBoss Data Grid** で JPA キャッシュストアをプログラミングにより設定するには、以下のコードを使用します。

```
Configuration cacheConfig = new
ConfigurationBuilder().persistence().addStore(JpaStoreConfigurationBuilder
.class).persistenceUnitName("org.infinispan.loaders.jpa.configurationTest"
)
.entityClass(User.class)
.build();
```

このコード例で使用するパラメーターは以下のとおりです。

- **persistenceUnitName** パラメーターは、JPA エンティティークラスを含む設定ファイル (**persistence.xml**) の JPA キャッシュストアの名前を指定します。
- **entityClass** パラメーターは、このキャッシュに格納された JPA エンティティークラスを指定します。各設定で指定できるクラスは1つだけです。

[バグを報告する](#)

### 18.8.3. 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18.7 persistence.xml でのメタデータエンティティの設定

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 **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>
```

説明のとおり、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](#)

### 18.8.4. さまざまなコンテナでの 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。

#### 手順18.8 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:jdgc-6.5 services,
org.infinispan.persistence.jpa:jdgc-6.5 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-6.5" services="export"/>
      <module name="org.infinispan" slot="jdg-6.5"
services="export"/>
    </dependencies>
  </deployment>
</jboss-deployment-structure>
```

### 手順18.9 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-6.5"/>
      <module name="org.jgroups" slot="jdg-6.5"/>
      <module name="org.infinispan.persistence.jpa"
slot="jdg-6.5" 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 6.5 の Apache Karaf でサポートされません。

[バグを報告する](#)

## 18.9. 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 の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およびマップ/削減の反復に使用されません。

既存のキャッシュストアを新しい API に移行するか、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18.10 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、ブラウザで <https://access.redhat.com/home> に移動します。
2. ダウンロードをクリックします。
3. Red Hat JBoss Middleware というラベルのボックスで、**Download Software** (ソフトウェアのダウンロード) ボタンをクリックします。
4. Red Hat ログイン フィールドと パスワード フィールドに該当するクレデンシャルを入力し、ログインをクリックします。
5. **Software Downloads** ページで、ドロップダウン値のリストから **Data Grid** を選択します。
6. ダウンロード可能なファイルのリストから、**Red Hat JBoss Data Grid \${VERSION} Source Code** を見つけ、**Download** をクリックします。ファイルを任意の場所に保存し、解凍します。

7. `jboss-datagrid-6.5.0-sources/infinispan-6.3.0.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](#)

### 18.9.1. 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ともに新しい **Maven** プロジェクトが生成されます。

#### 手順18.11 Maven アーキタイプの生成

1. JBoss Data Grid Maven リポジトリが Red Hat JBoss Data Grid 『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-6.5.0-maven-repository/"
    archetype:generate
        -DarchetypeGroupId=org.infinispan
        -DarchetypeArtifactId=custom-cache-store-archetype
        -DarchetypeVersion=6.3.0.Final-redhat-1
```



#### 注記

読みやすさのために上記のコマンドは複数の行に分割されています。ただし、実行する場合は、この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](#)

### 18.9.2. 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18.2 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">
        <properties>
            <property name="customStoreProperty" value="10" />
        </properties>
    </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](#)

### 18.9.2.1. 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18.12 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:6.2">  
  [...]  
  <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](#)

### 18.9.2.2. 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18.13 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/jboss-cli.sh --connect=$IP:$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

### 18.9.2.3. 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18.14 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。

[バグを報告する](#)

### 18.9.3. 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18.3 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（ライブラリーモード）](#)」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ともにパッケージ化することにより実現されます。ただし、これは Red Hat JBoss Enterprise Application Platform 『管理および設定ガイド』で説明されたように、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り実現することもできます。

[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19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](#)

### 19.1. 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](#)

### 19.2. 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19.1 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 here -->
<file-store passivation="true"
  <!-- Additional configuration elements here --> />
<!-- Additional configuration elements 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19.2 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](#)

### 19.3. 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。これは、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](#)

#### 19.3.1. 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ヘ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](#)

#### 19.3.2. 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19.1 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](#)

### 19.3.3. 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19.2 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
<b>keyOne</b> の挿入	メモリー: <b>keyOne</b>	ディスク:
<b>keyTwo</b> の挿入	メモリー: <b>keyOne</b> 、 <b>keyTwo</b>	ディスク:
エビクションスレッドが実行され、 <b>keyOne</b> をエビクトする	メモリー: <b>keyTwo</b>	ディスク: <b>keyOne</b>
<b>keyOne</b> の読み取り	メモリー: <b>keyOne</b> 、 <b>keyTwo</b>	ディスク:
エビクションスレッドが実行され、 <b>keyTwo</b> をエビクトする	メモリー: <b>keyOne</b>	ディスク: <b>keyTwo</b>
<b>keyTwo</b> の削除	メモリー: <b>keyOne</b>	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20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](#)

### 20.1. 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](#)

#### 20.1.1. ライトスルーキャッシングの利点

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。ただし、キャッシュストアへのアクセスやキャッシュ操作中の更新により、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](#)

#### 20.1.2. 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20.1 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

```
<?xml version="1.0" encoding="UTF-8"?>
<infinispan xmlns:xsi="http://www.w3.org/2001/XMLSchema-instance"
            xmlns="urn:infinispan:config:6.2">
  <global />
  <default />
  <namedCache name="persistentCache">
    <persistence>
      <singleFile fetchPersistentState="true"
                  ignoreModifications="false"
                  purgeOnStartup="false"
                  shared="false"
                  location="${java.io.tmpdir}"/>
    </persistence>
  </namedCache>
</infinispan>
```



1. **name** パラメーターは、使用する **namedCache** の名前を指定します。
2. **fetchPersistentState** 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。**fetchPersistentState** パラメーターはデフォルトでは **false** に設定されます。
3. **ignoreModifications** パラメーターは、キャッシュを変更する操作 (配置、削除、消去、格納など) がキャッシュストアに影響を与えるかどうかを決定します。結果として、キャッシュストアは、キャッシュと同期が取れなくなります。
4. **purgeOnStartup** パラメーターは、初回起動時にキャッシュがパージされるかどうかを指定します。
5. **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](#)

## 20.2. 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陳腐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](#)

### 20.2.1. 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となります。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](#)

### 20.2.2. 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20.2 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

## 20.2.3. 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20.3 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21章 JAVA MANAGEMENT EXTENSIONS (JMX) のセットアップ

### 21.1. 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](#)

### 21.2. 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](#)

### 21.3. 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](#)

### 21.4. 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名前付きキャッシュに対するターゲット **<namedCache>** 要素下に、次のスニペットを追加します。

```
<jmxStatistics enabled="true"/>
```

キャッシュレベルでプログラムを用いて **JMX** を有効にする

プログラムを用いて **JMX** をキャッシュレベルで有効にするには、以下のコードを追加します。

```
Configuration configuration = new  
ConfigurationBuilder().jmxStatistics().enable().build();
```

[バグを報告する](#)

## 21.5. CACHEMANAGERS 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"/>
```

**CacheManager** レベルでプログラムを用いて **JMX** を有効にする

次のコードを追加して、プログラムを用いて **JMX** を **CacheManager** レベルで有効にします。

```
GlobalConfiguration globalConfiguration = new  
GlobalConfigurationBuilder().globalJmxStatistics().enable().build();
```

[バグを報告する](#)

## 21.6. 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](#)

## 21.7. 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"/>
```

## CacheManager の名前をプログラムを用いて設定する

次のコードを追加し、プログラムを用いて **CacheManager** の名前を設定します。

```
GlobalConfiguration globalConfiguration = new
GlobalConfigurationBuilder().globalJmxStatistics().enable().
cacheManagerName("Hibernate2LC").build();
```

[バグを報告する](#)

## 21.8. 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、およびワーク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** 下のものは Red Hat JBoss Enterprise Application Platform 向けです)。



### 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](#)

### 21.8.1. MBean を理解する

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](#)

### 21.8.2. 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"/>
```

#### 完全修飾ドメイン名をプログラムを用いて追加する

次のコードを追加します。

```
GlobalConfiguration globalConfiguration = new
GlobalConfigurationBuilder().globalJmxStatistics().enable().
mBeanServerLookup("com.acme.MyMBeanServerLookup").build();
```



[バグを報告する](#)

## 第22章 JBOSS OPERATIONS NETWORK (JON) のセットアップ

### 22.1. 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 02** 以上のパッチバージョンで JBoss Operations Network 3.2.0 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』の節「Upgrading JBoss ON」を参照してください。

[バグを報告する](#)

### 22.2. JBOSS OPERATIONS NETWORK (JON) のダウンロード

#### 22.2.1. 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。
- 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](#)

#### 22.2.2. 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22.1 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、ブラウザで <https://access.redhat.com/home> に移動します。
2. ダウンロードをクリックします。
3. Red Hat JBoss Middleware というラベルのボックスで、**Download Software** (ソフトウェアのダウンロード) ボタンをクリックします。
4. Red Hat ログイン フィールドと パスワード フィールドに該当するクレデンシャルを入力し、ログインをクリックします。
5. **Software Downloads** (ソフトウェアのダウンロード) ページで、ドロップダウン値のリストから **JBoss Operations Network** を選択します。
6. **Version** (バージョン) 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。
7. 必要なダウンロードファイルの横にある **Download** (ダウンロード) ボタンをクリックします。

[バグを報告する](#)

### 22.2.3. 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](#)

### 22.2.4. 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22.2 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 Enterprise Middleware 以下のリスト内にある **ダウンロード** 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、  
「Software Downloads」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 JDG** を選択します。

- a. **JBoss Operations Network VERSION Base Distribution** ダウンロードリンクをクリックします。
- b. Base Distribution のダウンロードを開始するには、**Download** リンクをクリックします。
- c. **JDG 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](#)

## 22.3. 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](#)

## 22.4. 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](#)

## 22.5. 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](#)

### 22.5.1. 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](#)

## 22.6. JBOSS OPERATIONS NETWORK リモートクライアントサーバーのプラグイン

### 22.6.1. JBoss Operations Network プラグインのメトリックス

表22.1 キャッシュコンテナの 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の現在のランタイム状態です。
cluster-name	Cluster Name	クラスターの名前です。
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です。
local-address	Local Address	ローカルノードのアドレスです。
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22.2 キャッシュコンテナの 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。

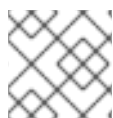
表22.3 キャッシュの JBoss Operations Network 特性

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です。
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22.4 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です。
number-of-locks-available	[LockManager] Number of locks available	現在利用可能な排他ロックの数です。
concurrency-level	[LockManager] Concurrency level	LockManager の設定済みの平行性レベルです。
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です。
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) です。
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です。
hits	[Statistics] Number of cache hits	キャッシュヒット数です。
evictions	[Statistics] Number of cache evictions	キャッシュエビクション操作の数です。
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です。
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です。
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です。
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 操作のヒット数です。
misses	[Statistics] Number of cache misses	キャッシュミス数です。

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) です。
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です。
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です。
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です。
invalidations	[Invalidation] Number of invalidations	インバリデーションの数です。
passivations	[Passivation] Number of cache passivations	パッシベーションイベントの数です。
activations	[Activations] Number of cache entries activated	アクティベーションイベントの数です。
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です。
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です。
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です。



## 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。

コネクタについての 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22.5 コネクターについての JBoss Operations Network メトリックス

メトリック名	表示名	説明
bytesRead	Bytes Read	読み込まれるバイト数です。
bytesWritten	Bytes Written	書き込まれるバイト数です。



#### 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](#)

## 22.6.2. JBoss Operations Network プラグイン操作

表22.6 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です。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、[29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22.7 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](#)

### 22.6.3. JBoss Operations Network プラグイン属性

表22.8 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 <b>240000</b> 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。

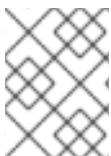
[バグを報告する](#)

#### 22.6.4. 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22.3 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

- JBoss Operations Network コンソールにログインします。
  - JBoss Operations Network コンソールにて **Inventory** をクリックします。
  - Select Servers from the Resources list on the left of the console.
- 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。
  - 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。
- 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を選択します。
  - 選択されたキャッシュコンテナを右クリックします (たとえば、**clustered**)。
  - コンテキストメニューで、**Create Child** に移動し、**Cache** を選択します。
- 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。
  - 新しいキャッシュ名を入力し、**Next** をクリックします。
  - 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](#)

## 22.7. 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](#)

### 22.7.1. 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22.4 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。

Name ^	Description	Last Updated	Enabled?	Deployed?
Abstract Augeas Plugin	An abstract plugin supporting concrete plugins that are based on Augeas, the open-source configuration library for Linux	Sep 20, 2012 1:35:12 PM	✓	✓
Abstract Database	Abstract plugin supporting concrete database plugins	Sep 20, 2012 1:35:12 PM	✓	✓
Ant Bundle Plugin		Sep 20, 2012 1:35:12 PM	✓	✓
Apache HTTP Server	Management of Apache web servers	Sep 20, 2012 1:35:12 PM	✓	✓
Generic JMX	Supports management of JMX MBean Servers via various remoting systems.	Sep 20, 2012 1:35:12 PM	✓	✓
IIS	Monitoring of Microsoft IIS Services	Sep 20, 2012 1:35:12 PM	✓	✓
Operating System Platforms	Management and monitoring of operating system level functions	Sep 20, 2012 1:35:12 PM	✓	✓
PostgreSQL Database	Management and monitoring of PostgreSQL versions 8.2 and higher	Sep 20, 2012 1:35:12 PM	✓	✓
RHQ Agent	Management and monitoring of the RHQ Agent	Sep 20, 2012 1:35:12 PM	✓	✓
Script	Manages resources that have command line executables or scripts as their management interfaces	Sep 20, 2012 1:35:12 PM	✓	✓

## 図22.1 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。

JBoss Administration Console - Administration Tab

File successfully uploaded

Name	Description	Last Updated	Enabled?	Deployed?
Abstract Augeas Plugin	An abstract plugin supporting concrete plugins that are based on Augeas, the open-source configuration library for Linux	Sep 20, 2012 1:35:12 PM	✓	✓
Abstract Database	Abstract plugin supporting concrete database plugins	Sep 20, 2012 1:35:12 PM	✓	✓
Ant Bundle Plugin		Sep 20, 2012 1:35:12 PM	✓	✓
Apache HTTP Server	Management of Apache web servers	Sep 20, 2012 1:35:12 PM	✓	✓
Generic JMX	Supports management of JMX MBean Servers via various remoting systems.	Sep 20, 2012 1:35:12 PM	✓	✓
IIS	Monitoring of Microsoft IIS Services	Sep 20, 2012 1:35:12 PM	✓	✓
Operating System Platforms	Management and monitoring of operating system level functions	Sep 20, 2012 1:35:12 PM	✓	✓
PostgreSQL Database	Management and monitoring of PostgreSQL versions 8.2 and higher	Sep 20, 2012 1:35:12 PM	✓	✓
RHQ Agent	Management and monitoring of the RHQ Agent	Sep 20, 2012 1:35:12 PM	✓	✓
Script	Manages resources that have command line executables or scripts as their management interfaces	Sep 20, 2012 1:35:12 PM	✓	✓

Buttons: Scan For Updates, Hide Deleted, Upload Plugin, Browse..., Upload, Enable, Disable, Delete, Purge, Refresh

Total Rows: 10 (selected: 0)

## 図22.2 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。

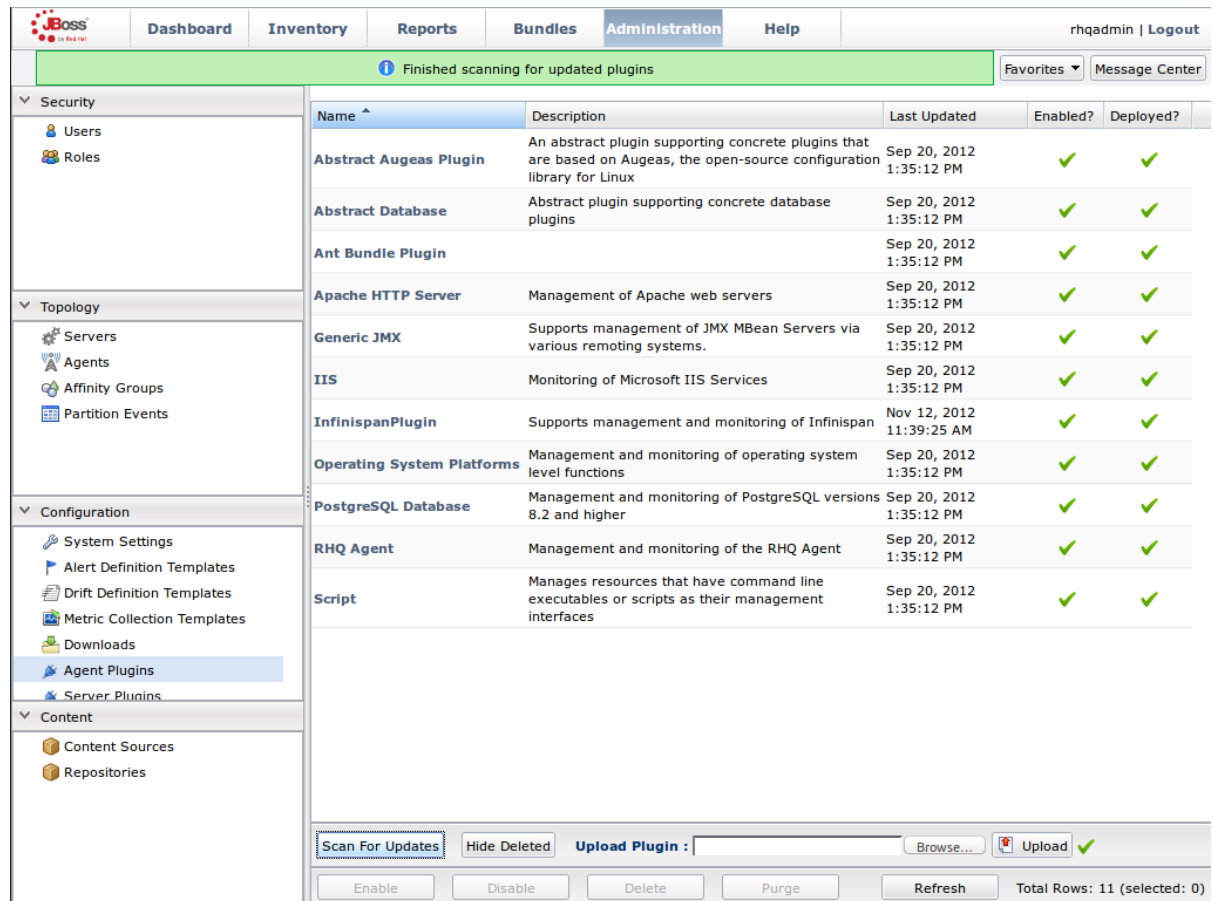


図22.3 更新済みプラグインのためのスキャン

バグを報告する

## 22.7.2. ライブラリーモードでの JBoss Data Grid インスタンスの追加

### 22.7.2.1. 前提条件

「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」の共通の前提条件を以下に示します。

- 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、この実行を避ける必要があります。

- 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ために 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のために JMX を有効にする方法については「[CacheManagers 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 の正しく追加され、設定された管理ユーザーです。

## バグを報告する

### 22.7.2.2. 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

#### 22.7.2.2.1. 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22.5 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。



The screenshot displays the JBoss Operations Network (JON) Discovery Queue. The interface is divided into a left sidebar and a main content area. The sidebar contains a tree view with 'Resources' and 'Groups' sections. The 'Resources' section is expanded, showing 'Discovery Queue', 'All Resources', 'Platforms', 'Servers - Top Level Imports', 'Servers', 'Services', and 'Unavailable Servers'. The 'Groups' section is also expanded, showing 'All Groups', 'Dynagroup Definitions', 'Compatible Groups', 'Mixed Groups', and 'Problem Groups'. The main content area displays a table titled 'Discovery Queue' with the following data:

Resource Name	Resource Key	Resource Type	Description	Inventory Status	Discovery Time
jdg	jdg	Linux	Linux Operating System	New	Nov 12, 2012 11:40:55 AM

At the bottom of the interface, there are buttons for 'Import', 'Ignore', 'Unignore', and 'Show : New'. There are also buttons for 'Select All' and 'Deselect All'.

## 図22.4 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

- jdg プラットフォームが **Platforms** リストに表示されます。
- 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。

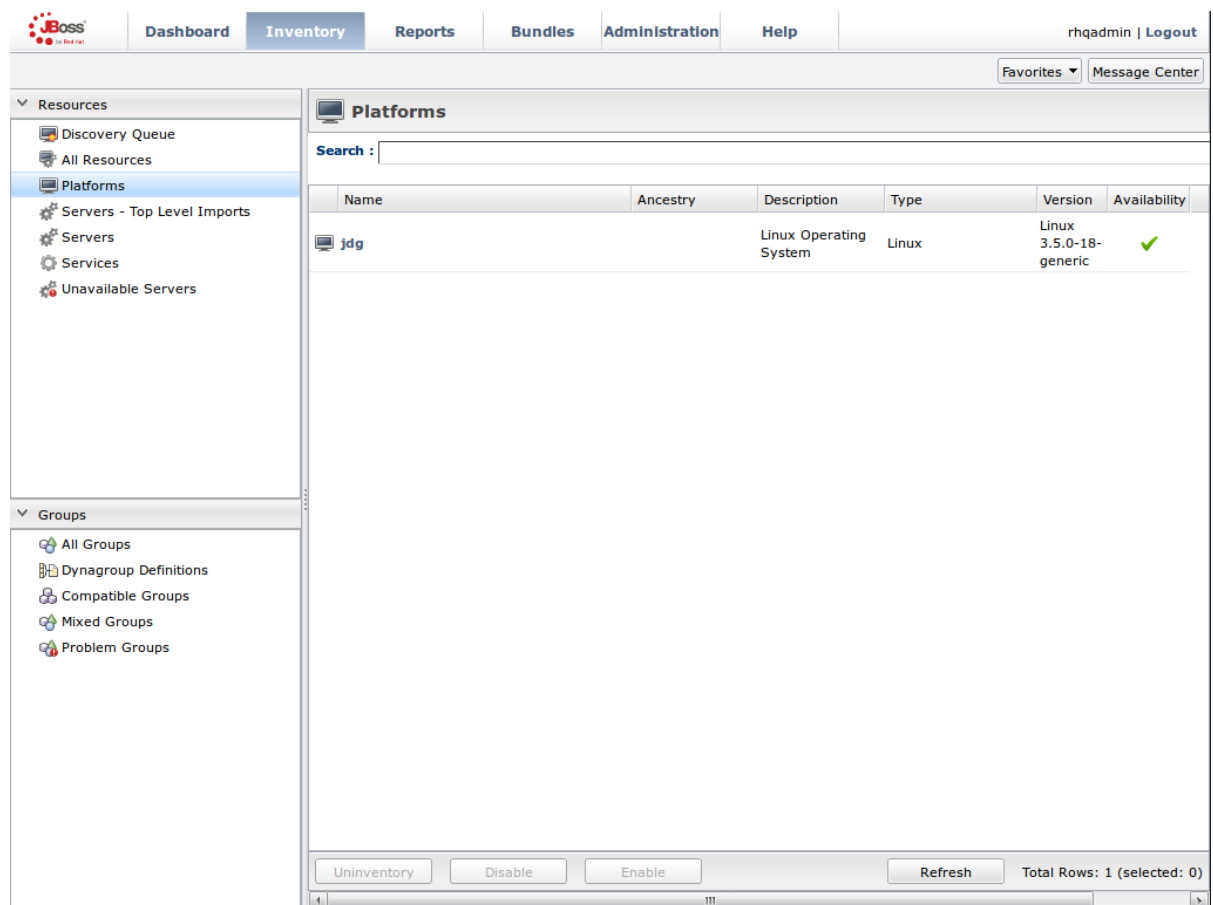


図22.5 サーバーのリストを表示するためにjdg プラットフォームを開く。

### 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。

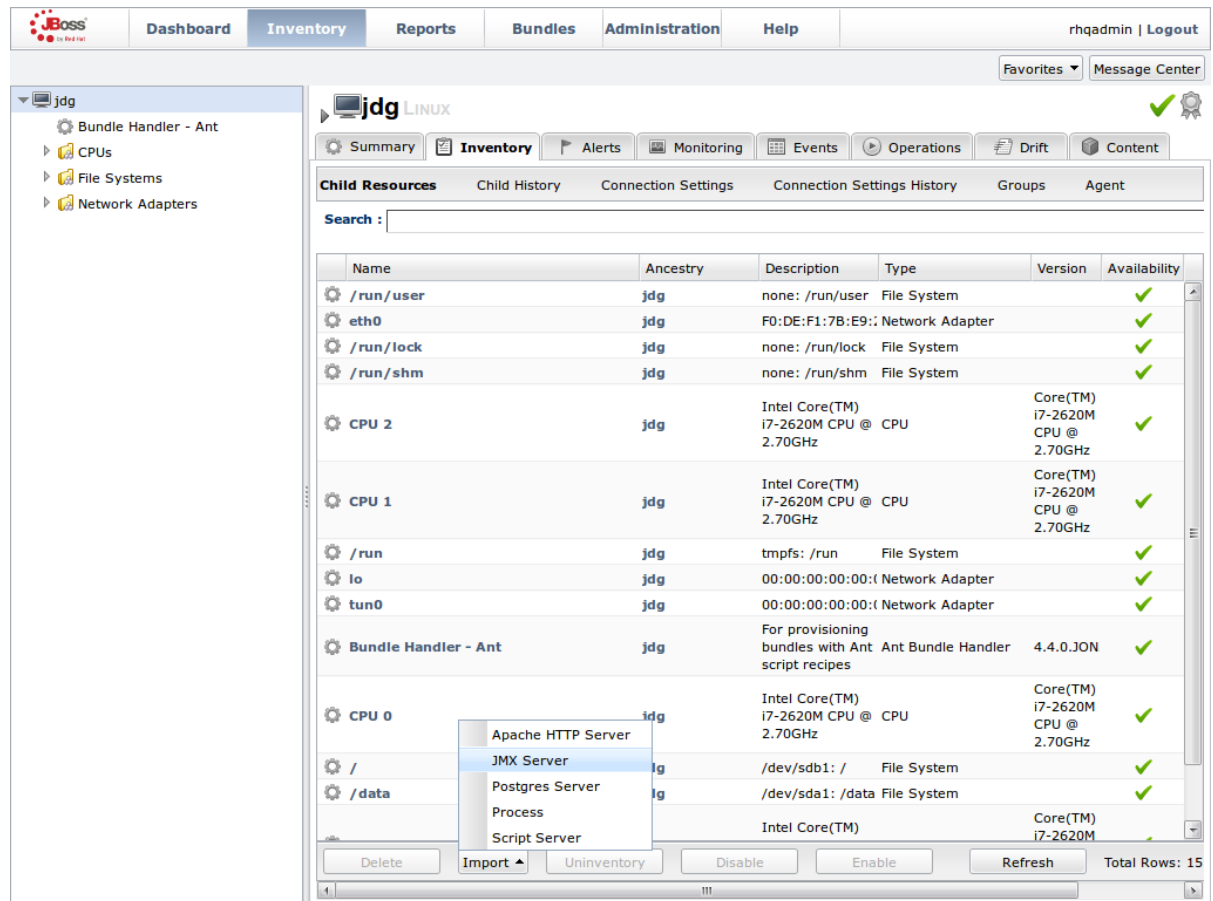


図22.6 JMX サーバーのインポート

## 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。

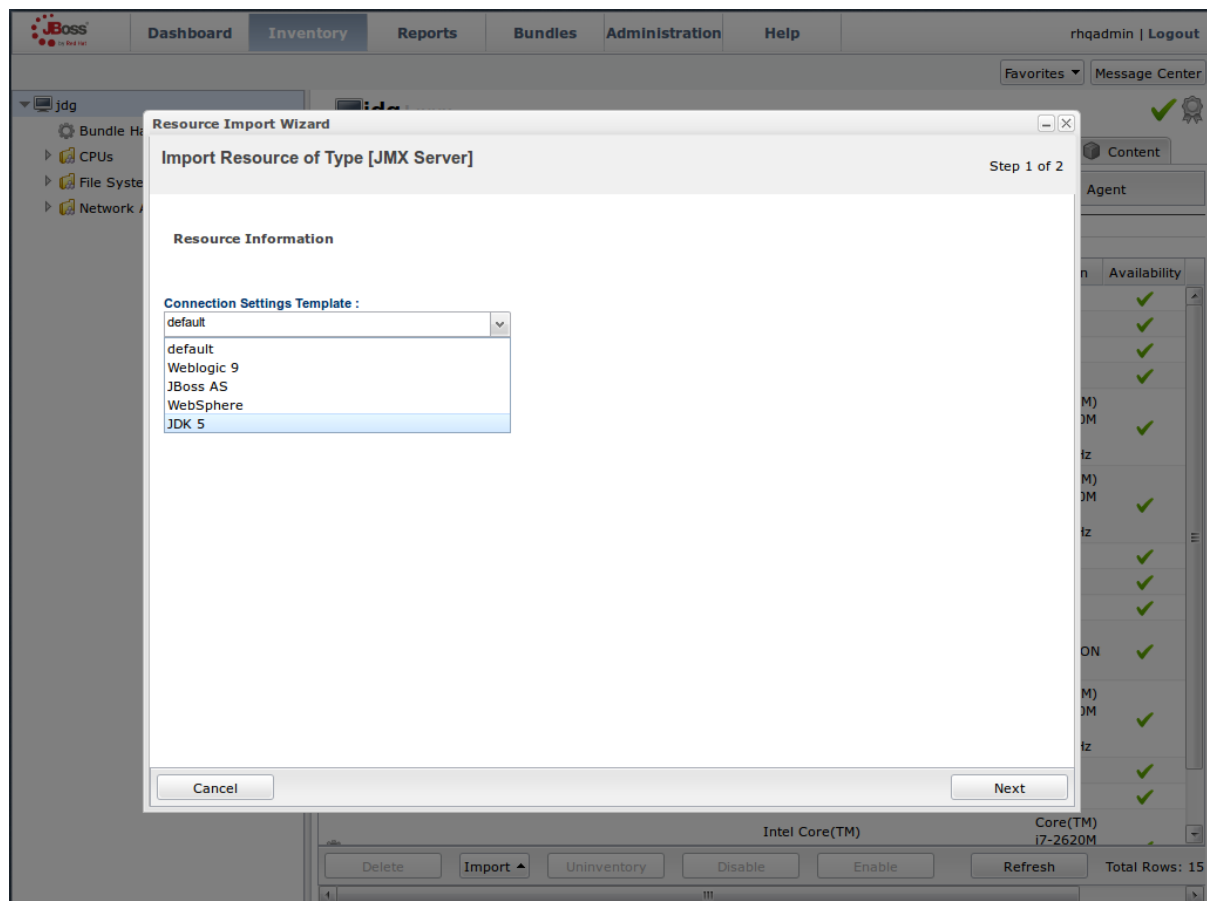


図22.7 JDK 5 テンプレートの選択

## 5. コネクタ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** を、Infinispan ライブラリーを含むプロセスのホスト名と JMX ポートで修正します。
- b. 監視する新規 JBoss Data Grid インスタンスの JMX コネクタのアドレスを入力します。以下に例を示します。

コネクタアドレス:

```
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。

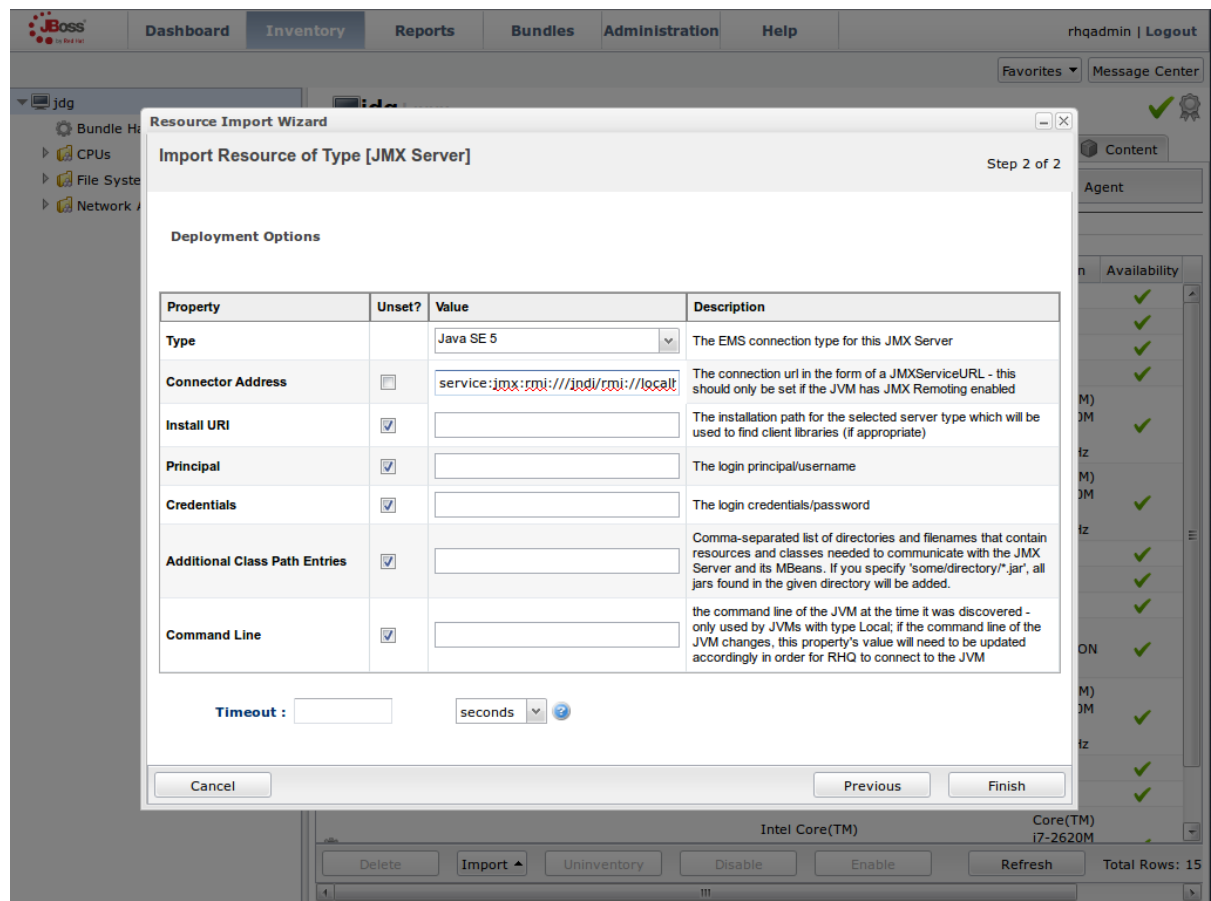


図22.8 Deployment Options 画面での値の修正

## 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、サーバー一覧を最新の情報に更新します。
- 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。利用可能な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。

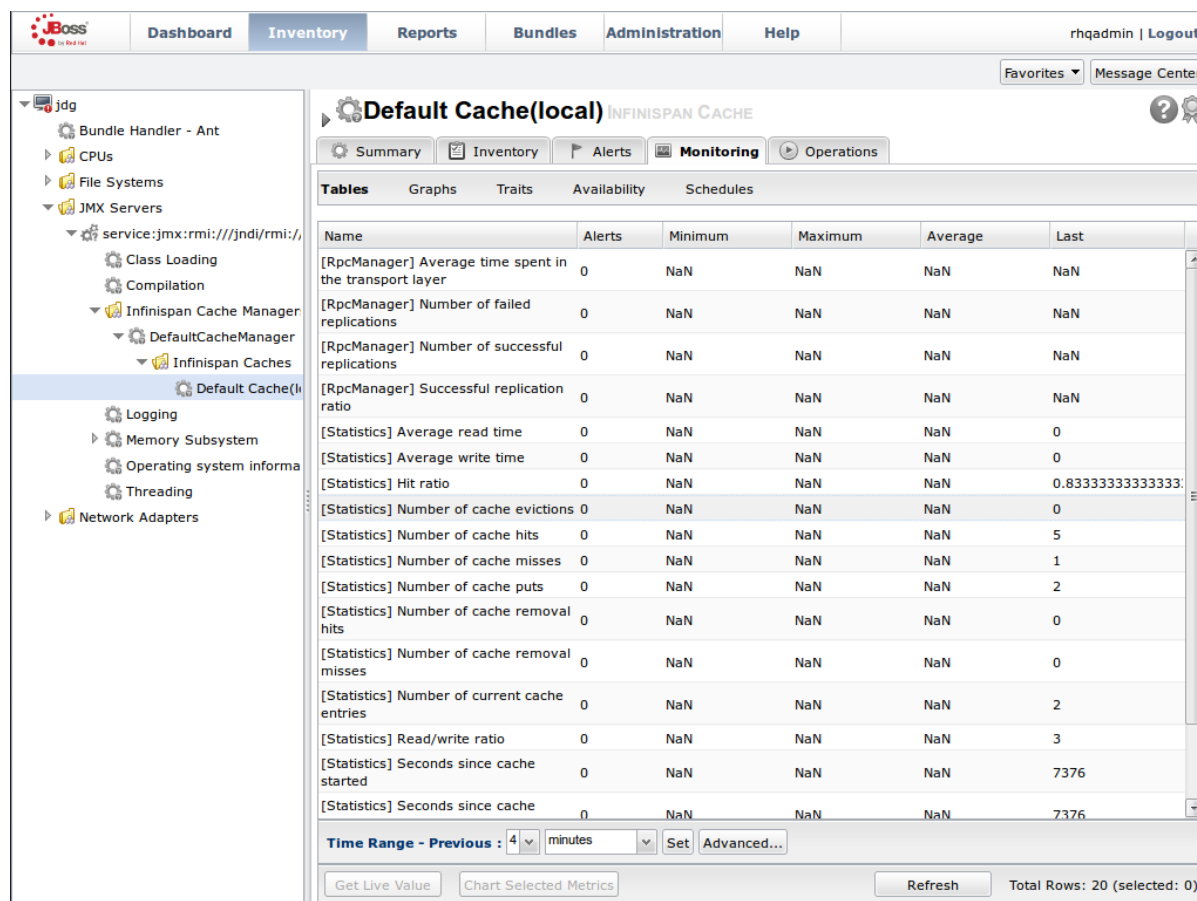


図22.9 JMX 経由でリレーされるメトリックスおよび操作データは JBoss Operations Network コンソールで利用可能

## バグを報告する

### 22.7.2.3. JBoss Enterprise Application Platform にデプロイされたアプリケーションをライブラリーモードで監視

#### 22.7.2.3.1. 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22.6 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。

```
$JBOSSE_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) を使用して監視できるようになります。

#### [バグを報告する](#)

### 22.7.2.3.2. 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22.7 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。

## 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](#)

## 22.8. JBOSS OPERATIONS NETWORK のプラグインクイックスタート

単一の JBoss Operation Network エージェント向けのテストおよびデモンストレーション目的で、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](#)

## 22.9. 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](#)

### 22.9.1. 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、「[REST インターフェースの使用](#)」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](#)

### 22.9.2. 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うことが可能です。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。さらに、前述のメソッドはローカルのキャッシュ上でのみ操作可能であり、状態のグローバルビューを提供しません。

前述のメソッドがグローバルに実行されると、パフォーマンスに大きく影響し、スケーラビリティのボトルネックが発生します。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

Red Hat JBoss Data Grid 6.3 以降、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内のエントリーが含まれます (以前は、これらのメソッドにはローカルのデータコンテナーのみが含まれていました)。基礎とな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。古い動作を保持するには **Cache.getAdvancedCache().withFlags(Flag.SKIP\_CACHE\_LOAD).values()** を使用し、パフォーマンスの低下を回避するにはキャッシュストアからロードしないようにしてください。

#### size() メソッドへの変更 (内蔵キャッシュ)

JBoss Data Grid 6.3 では、**Cache#size()** メソッドはローカルノードのエントリー数のみを返し、クラスター化キャッシュの他のノードを無視し、失効したエントリーを含みます。デフォルトの動作が JBoss Data Grid 6.4 以降で変更されなかった場合は、**infinispan.accurate.bulk.ops** システムプロパティーを **true** に設定することにより、多くの操作 (**size()** を含む) に対して正確な結果を有効にできます。操作のこのモードでは、**size()** メソッドにより返される結果は、ローカルノードに存在するエントリーの数を返すフラグ **org.infinispan.context.Flag#CACHE\_MODE\_LOCAL** と、すべてのパッシブ化エントリーを無視するフラグ **org.infinispan.context.Flag#SKIP\_CACHE\_LOAD** によって影響を受けます。

#### size() メソッドへの変更 (リモートキャッシュ)

JBoss Data Grid 6.3 では、Hot Rod **size()** メソッドは **STATS** 操作を呼び出し、返された **numberOfEntries** 統計を使用してキャッシュのサイズを取得していました。この統計は、失効済みのパッシブ化エントリーを考慮せず、操作に応答したノードにのみローカルであるため、キャッシュ内のエントリー数を正確に反映しません。別の結果として、セキュリティが有効である場合は、クライアントが適切な **BULK\_READ** の代わりに **ADMIN** パーミッションを必要とします。

JBoss Data Grid 6.4 以降では、Hot Rod プロトコルが専用の **SIZE** 操作で拡張され、クライアントが **size()** メソッドにこの操作を使用するよう更新されました。JBoss Data Grid サーバーは、サイズを正確に計算できるように **infinispan.accurate.bulk.ops** システムプロパティーを **true** に設定した状態で起動する必要があります。

[バグを報告する](#)

## パート XI. 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23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](#)

### 23.1. 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](#)

#### 23.1.1. 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 **cluster** ファイルを使って起動します。Linux の場合、**standalone.sh** または **clustered.sh** スクリプトを使用し、Windows の場合、**standalone.bat** または **clustered.bat** ファイルを使用します。

[バグを報告する](#)

#### 23.1.2. 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](#)

#### 23.1.3. 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23.1 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、 <code>jmx://[username[:password]]@host:port[/container[/cache]]</code> を使用し、JMX over JBoss Remoting の場合は <code>remoting://[username[:password]]@host:port[/container[/cache]]</code> 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、 <b>stdin</b> 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

### 23.1.4. 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](#)

## 23.2. 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](#)

### 23.2.1. 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](#)

## 23.3. 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](#)

### 23.3.1. 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](#)

### 23.3.2. 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](#)

### 23.3.3. 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](#)

### 23.3.4. clear コマンド

**clear** 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]> clear  
[jmx://localhost:12000/MyCacheManager/namedCache]> get a  
null
```

[バグを報告する](#)

### 23.3.5. 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](#)

### 23.3.6. 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 (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](#)

### 23.3.7. 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。

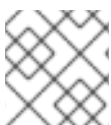
```
[jmx://localhost:12000/MyCacheManager/namedCache]> create newCache like
namedCache
[jmx://localhost:12000/MyCacheManager/namedCache]> cache newCache
[jmx://localhost:12000/MyCacheManager/newCache]>
```

[バグを報告する](#)

### 23.3.8. 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](#)

### 23.3.9. 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](#)

### 23.3.10. 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](#)

### 23.3.11. 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](#)

### 23.3.12. 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](#)

### 23.3.13. 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](#)

### 23.3.14. 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](#)

### 23.3.15. 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{factory=org.infinispan.executors.DefaultExecutorFactory@98add58},
asyncTransportExecutor=ExecutorFactoryConfiguration{factory=org.infinispan.executors.DefaultExecutorFactory@7bc9c14c},
evictionScheduledExecutor=ScheduledExecutorFactoryConfiguration{factory=org.infinispan.executors.DefaultScheduledExecutorFactory@7ab1a411},
replicationQueueScheduledExecutor=ScheduledExecutorFactoryConfiguration{factory=org.infinispan.executors.DefaultScheduledExecutorFactory@248a9705},
globalJmxStatistics=GlobalJmxStatisticsConfiguration{allowDuplicateDomains=true, enabled=true, jmxDomain='jboss.infinispan',
mBeanServerLookup=org.jboss.as.clustering.infinispan.MBeanServerProvider@6c0dc01, 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$Externalizer@5fab91d,
1101=org.infinispan.server.memcached.MemcachedValue$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](#)

### 23.3.16. 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](#)

### 23.3.17. 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 41391  
[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](#)

### 23.3.18. 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](#)

### 23.3.19. 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](#)

### 23.3.20. 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](#)

### 23.3.21. 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](#)

### 23.3.22. 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](#)

### 23.3.23. 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](#)

### 23.3.24. 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、『Red Hat JBoss Data Grid 開発者ガイド』の『ローリングアップグレード』の章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](#)

### 23.3.25. 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 XII. 他の RED HAT JBOSS DATA GRID 機能

## 第24章 1 次キャッシュのセットアップ

### 24.1.1 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くないようにします。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。

キーが更新される度に、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](#)

### 24.2.1 1 次キャッシュの設定

#### 24.2.1.1 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24.1 ライブラリーモードの L1 キャッシュ設定

```
<clustering mode="dist">
  <sync/>
  <l1 enabled="true"
      lifespan="60000" />
</clustering>
```

11 要素は、分散キャッシュインスタンスにおけるキャッシュの動作を設定します。分散されていないキャッシュと共に使用されている場合、この要素は無視されます。

- **enabled** パラメーターは 1 次キャッシュを有効にします。
- **lifespan** パラメーターは、1 次キャッシュに置かれる際に、エントリーの最大ライフスパンを設定します。

[バグを報告する](#)

#### 24.2.2.1 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)

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1次キャッシュのデフォルト値を示しています。

#### 例24.2 リモートクライアントサーバーモード用 L1 キャッシュの設定

```
<distributed-cache l1-lifespan="${VALUE}">
  <!-- 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。



#### 注記

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、この状態で1次キャッシュを使用してもパフォーマンスは改善されないため、1次キャッシュの使用は推奨されません。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、L1 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 L1 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 L1 キャッシュを有効にします。

[バグを報告する](#)

## 第25章 トランザクションのセットアップ

### 25.1. 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](#)

#### 25.1.1. 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](#)

#### 25.1.2. 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、トランザ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](#)



### 25.1.3. 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](#)

### 25.1.4. 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](#)

### 25.1.5. 複数のキャッシュインスタンスにわたるトランザクション

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](#)

## 25.2. トランザクションの設定

### 25.2.1. 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。

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25.1 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)

```
<namedCache <!-- Additional configuration information here -->>
  <transaction <!-- Additional configuration information here --> >
  <locking <!-- Additional configuration information here --> >
  <versioning enabled="{true,false}"
    versioningScheme="{NONE|SIMPLE}"/>
    <!-- Additional configuration information here -->
</namedCache>
```

1. **versioning** パラメーターの **enabled** パラメーターを **true** に設定します。
2. **versioningScheme** パラメーターを **NONE** または **SIMPLE** のいずれかに設定して、使用するバージョン管理スキームを設定します。

#### 手順25.2 ライブラリーモード (プログラムを用いた設定) でトランザクションを設定します。

```
1. Configuration config = new ConfigurationBuilder()/* ...
   */.transaction()
      .transactionMode(TransactionMode.TRANSACTIONAL)
      .transactionManagerLookup(new
GenericTransactionManagerLookup())
      .lockingMode(LockingMode.OPTIMISTIC)
      .useSynchronization(true)
      .recovery()

      .recoveryInfoCacheName("anotherRecoveryCacheName").build();
```

- a. トランザクションモードを設定します。
- b. ルックアップクラスを選択し、設定します。利用可能なルックアップクラスのリストについては、この手順の下にある表を参照してください。
- c. **lockingMode** 値は、楽観的または悲観的ロックを使用するかどうかを決定します。キャッシュが非トランザクションの場合、ロックモードは無視されます。デフォルト値は **OPTIMISTIC** です。
- d. **useSynchronization** 値は、トランザクションマネージャーを使って同期化を登録するようにキャッシュを設定するか、またはキャッシュ自体を XA リソースとして登録するようにキャッシュを設定します。デフォルト値は **true** (同期の使用) です。
- e. **recovery** パラメーターは、**true** に設定されるとキャッシュのリカバリーを有効にしま

す。

**recoveryInfoCacheName** は、リカバリー情報が保持されるキャッシュの名前を設定します。キャッシュのデフォルト名は **RecoveryConfiguration.DEFAULT\_RECOVERY\_INFO\_CACHE** によって指定されます。

## 2. 書き込みスキューチェックを設定します。

**writeSkew** チェックは、異なるトランザクションからのエントリーに対する変更によりトランザクションがロールバックされるべきかどうかを判別します。**true** に設定された書き込みスキューにより、**isolation\_level** を **REPEATABLE\_READ** に設定する必要があります。**writeSkew** および **isolation\_level** のデフォルト値はそれぞれ **false** と **READ\_COMMITTED** です。

```
Configuration config = new ConfigurationBuilder()/* ... */.locking()
    .isolationLevel(IsolationLevel.REPEATABLE_READ).writeSkewCheck(true)
    ;
```

## 3. エントリーのバージョン管理を設定します。

クラスター化されたキャッシュについては、エントリーのバージョン管理を有効にし、その値を **SIMPLE** に設定することにより書き込みスキューのチェックを有効にします。

```
Configuration config = new ConfigurationBuilder()/* ...
    *.versioning()
        .enable()
        .scheme(VersioningScheme.SIMPLE);
```

表25.1 トランザクションマネージャーのルックアップクラス

クラス名	説明
org.infinispan.transaction.lookup.DummyTransactionManagerLookup	テスト環境で主に使用されます。このテスト向けのトランザクションマネージャーは実稼働環境では使用されず、特に並列トランザクションやリカバリーなどの機能は厳しく制限されます。
org.infinispan.transaction.lookup.JBossStandaloneJTAManagerLookup	Red Hat JBoss Data Grid がスタンドアロン環境で実行される場合のデフォルトのトランザクションマネージャーです。これにより、JBoss Transactions ベースの完全に機能するトランザクションマネージャーで、 <b>DummyTransactionManager</b> の機能上の制限が解消されます。
org.infinispan.transaction.lookup.GenericTransactionManagerLookup	<b>GenericTransactionManagerLookup</b> は、トランザクションルックアップクラスが指定されていない場合にデフォルトで使用されます。このルックアップクラスは、 <b>TransactionManager</b> インターフェースを提供する Java EE 互換環境で JBoss Data Grid を使用する場合に推奨され、ほとんどの Java EE アプリケーションサーバーでトランザクションマネージャーを見つけるために使用できます。トランザクションマネージャーが見つからない場合、デフォルトは <b>DummyTransactionManager</b> になります。

クラス名	説明
org.infinispan.transaction.lookup.JBossTransactionManagerLookup	<b>JbossTransactionManagerLookup</b> は、アプリケーションサーバーで実行中の標準的なトランザクションマネージャーを見つけます。このルックアップクラスは JNDI を使用して <b>TransactionManager</b> インスタンスを検索します。これは、カスタムキャッシュが JTA トランザクションで使用されている場合に推奨されます。

[バグを報告する](#)

### 25.2.2. 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25.1 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](#)

### 25.3. 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せん。

## バグを報告する

### 25.3.1. 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25.3 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

### 25.3.2. 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25.2 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、システムは不整合な状態になります。特に、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](#)

## 25.4. 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 **disabled** に設定されます。

[バグを報告する](#)

### 25.4.1. デッドロック検出を有効にする

Red Hat JBoss Data Grid のデッドロック検出はデフォルトでは **disabled** に設定されていますが、以下を追加して **namedCache** 設定要素を使用すると、デッドロック検出を有効にし、各キャッシュに対して設定を行うことが可能です。

```
<deadlockDetection enabled="true" spinDuration="100"/>
```

**spinDuration** 属性は、特定ロックの取得が許可される時間 (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。

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26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](#)

### 26.1. 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)

#### 26.1.1. 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](#)

### 26.1.2. ソケットのバインディング

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](#)

#### 26.1.2.1. 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26.1 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](#)

#### 26.1.2.2. 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26.2 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](#)

### 26.1.3. 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26.3 JGroups ソケットバインディング設定**

```
<subsystem xmlns="urn:jboss:domain:jgroups:1.2" default-stack="udp">
  <stack name="udp">
    <transport type="UDP" socket-binding="jgroups-udp">
      <!-- Additional configuration elements here -->
    </transport>
    <!-- rest of protocols -->
  </stack>
</subsystem>
```

**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](#)

**26.2. 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26.4 JGroups プログラム可能設定**

```
GlobalConfiguration gc = new GlobalConfigurationBuilder()
    .transport()
    .defaultTransport()
    .addProperty("configurationFile", "jgroups.xml")
    .build();
```

**例26.5 JGroups XML 設定**

```
<infinispan>
  <global>
    <transport>
      <properties>
        <property name="configurationFile" value="jgroups.xml" />
      </properties>
    </transport>
  </global>
  <!-- Additional configuration elements here -->
</infinispan>
```

プログラムによる設定か、または XML 設定のいずれの場合でも、JBoss Data Grid はクラスパスに見つからない場合は絶対パス名を検索する前にクラスパスの **jgroups.xml** を検索します。

[バグを報告する](#)

## 26.2.1. 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](#)

### 26.2.1.1. 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。チャン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](#)

### 26.2.1.2. 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利用できる 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。

[バグを報告する](#)

### 26.2.1.3. 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26.6 JGroups サブシステムが TCPPING を使用するよう設定する

```
<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](#)

26.2.2. 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](#)

26.2.2.1. 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26.1 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

システムプロパティー	説明	デフォルト	必要性
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](#)

### 26.2.2.2. 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26.2 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.udp.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.udp.mcast_port	マルチキャストに使用するポート	46655	いいえ

システムプロパティー	説明	デフォルト	必要性
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

### 26.2.2.3. 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26.3 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、 <b>S3_PING</b> 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

システムプロパティ	説明	デフォルト	必要性
-----------	----	-------	-----

[バグを報告する](#)

## 26.3. 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](#)

### 26.3.1. 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。

表26.4 異なる JBoss Data Grid バージョンを使用したテスト

Version	テストケース	詳細
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。
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 Platform 6.0 をベースにしています。
JBoss Data Grid 6.1.0	利用可能	テストクラスの場合はディストリビューションによって異なります。 <ul style="list-style-type: none"> <li>ライブラリーモードでは、<b>lib</b> ディレクトリーの JGroups JAR ファイルに含まれます。</li> <li>リモートクライアントサーバーモードでは、<b><code>\${JDG_HOME}/modules/org/jgroups/main/</code></b> ディレクトリーの JGroups JAR ファイルに含まれます。</li> </ul>

Version	テストケース	詳細
JBoss Data Grid 6.2.0	利用可能	<p>テストクラスの場合はディストーションによって異なります。</p> <ul style="list-style-type: none"> <li>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。</li> <li>リモートクライアントサーバーモードでは、<b><code>\${JDG_HOME}/modules/system/layers/base/org/jgroups/main</code></b> ディレクトリーの JGroups JAR ファイルに含まれます。</li> </ul>
JBoss Data Grid 6.2.1	利用可能	<p>テストクラスの場合はディストーションによって異なります。</p> <ul style="list-style-type: none"> <li>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。</li> <li>リモートクライアントサーバーモードでは、<b><code>\${JDG_HOME}/modules/system/layers/base/org/jgroups/main</code></b> ディレクトリーの JGroups JAR ファイルに含まれます。</li> </ul>
JBoss Data Grid 6.3.0	利用可能	<p>テストクラスの場合はディストーションによって異なります。</p> <ul style="list-style-type: none"> <li>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。</li> <li>リモートクライアントサーバーモードでは、<b><code>\${JDG_HOME}/modules/system/layers/base/org/jgroups/main</code></b> ディレクトリーの JGroups JAR ファイルに含まれます。</li> </ul>

Version	テストケース	詳細
JBoss Data Grid 6.4.0	利用可能	<p>テストクラスの場合はディストーションによって異なります。</p> <ul style="list-style-type: none"> <li>ライブラリーモードでは、<b>infinispan-embedded</b> JAR ファイルに含まれます。</li> <li>リモートクライアントサーバーモードでは、<b><code>\${JDG_HOME}/modules/system/layers/base/org/jgroups/main/</code></b> ディレクトリーの JGroups JAR ファイルに含まれます。</li> </ul>
JBoss Data Grid 6.5.0	利用可能	<p>テストクラスの場合はディストーションによって異なります。</p> <ul style="list-style-type: none"> <li>ライブラリーモードでは、<b>infinispan-embedded</b> JAR ファイルに含まれます。</li> <li>リモートクライアントサーバーモードでは、<b><code>\${JDG_HOME}/modules/system/layers/base/org/jgroups/main/</code></b> ディレクトリーの JGroups JAR ファイルに含まれます。</li> </ul>

## バグを報告する

### 26.3.2. 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26.1 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 番目のノード (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27章 RED HAT DATA GRID の AMAZON WEB サービスとの使用

### 27.1. S3\_PING JGROUPS 検出プロトコル

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検出プロトコルです。

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](#)

### 27.2. S3\_PING 設定オプション

Red Hat JBoss Data Grid は Amazon Web Services と 2 とお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 の認証情報を使用します。
- 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認証情報がありません。

[バグを報告する](#)

#### 27.2.1. プライベート S3 バケットの使用

この設定には、適切な AWS 認証情報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。

- バケットを指定するための **location** または **prefix** プロパティー (両方ではない)。 **prefix**

プロパティーが設定されている場合、**S3\_PING** は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。名前の先頭がプレフィックスのバケットが見つかった場合、**S3\_PING** はそのバケットを使用します。このプレフィックスを持つバケットが見つからない場合、**S3\_PING** は、AWS クレデンシャルを使用してバケットを作成し、そのプレフィックスと **UUID** に基いて名前を指定します (名前の形式は `{prefix value}-{UUID}`)。

- 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27.1 プライベートバケットの使用による 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、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/clustered.sh -Djboss.bind.address={server_ip_address} -
Djboss.bind.address.management={server_ip_address} -
Djboss.default.jgroups.stack=s3 -Djgroups.s3.bucket={s3_bucket_name} -
Djgroups.s3.access_key={access_key} -
Djgroups.s3.secret_access_key={secret_access_key}
```

1. `{server_ip_address}` をサーバーの IP アドレスに置き換えます。
2. `{s3_bucket_name}` を適切なバケット名に置き換えます。
3. `{access_key}` をユーザーのアクセスキーに置き換えます。
4. `{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](#)

## 27.2.2. 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](#)

### 27.2.2.1. 事前署名付き URL の生成

JGroup 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27.2 事前署名付き URL の生成

```
String putUrl = S3_PING.generatePreSignedUrl("access_key",
"secret_access_key", "put", "my_bucket", "DemoCluster/node1",
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 認証情報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 (**&amp;**) に置き換える必要があります。

[バグを報告する](#)

### 27.2.2.2. 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27.3 事前署名付き URL による JBoss Data Grid サーバーの起動**

```
bin/clustered.sh -Djboss.bind.address={server_ip_address} -
Djboss.bind.address.management={server_ip_address} -
Djboss.default.jgroups.stack=s3 -
Djgroups.s3.pre_signed_put_url="http://{s3_bucket_name}.s3.amazonaws.com
/ node1?
AWSAccessKeyId={access_key}&Expires={expiration_time}&Signature={signature}"-
Djgroups.s3.pre_signed_delete_url="http://{s3_bucket_name}.s3.amazonaws.
com/ node1?
AWSAccessKeyId={access_key}&Expires={expiration_time}&Signature={signature}"
```

上記の例では、**{signatures}** 値は **S3\_PING.generatePreSignedUrl()** メソッドによって生成されます。さらに、**{expiration\_time}** 値は、**S3\_PING.generatePreSignedUrl()** メソッドに渡される URL の有効期間です。

[バグを報告する](#)

**27.2.3. 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/clustered.sh -Djboss.bind.address={server_ip_address} -
Djboss.bind.address.management={server_ip_address} -
Djboss.default.jgroups.stack=s3 -Djgroups.s3.bucket={s3_bucket_name}
```

[バグを報告する](#)

**27.2.4. S3\_PING 警告のトラブルシューティング**

使用される **S3\_PING** 設定タイプにより、以下の警告が JBoss Data Grid サーバーの起動時に表示される場合があります。

```
15:46:03,468 WARN [org.jgroups.conf.ProtocolConfiguration] (MSC service
thread 1-7) variable "${jgroups.s3.pre_signed_put_url}" in S3_PING could
not be substituted; pre_signed_put_url is removed from properties
```

```
15:46:03,469 WARN [org.jgroups.conf.ProtocolConfiguration] (MSC service
thread 1-7) variable "${jgroups.s3.prefix}" in S3_PING could not be
substituted; prefix is removed from properties
```

```
15:46:03,469 WARN [org.jgroups.conf.ProtocolConfiguration] (MSC service
thread 1-7) variable "${jgroups.s3.pre_signed_delete_url}" in S3_PING
could not be substituted; pre_signed_delete_url is removed from properties
```

それぞれの場合に、警告により不足しているものとしてリストされるプロパティーが **S3\_PING** 設定では不要であることを確認してください。

[バグを報告する](#)

## 第28章 サーバーヒントイングを用いた高可用性

Red Hat JBoss Data Grid では、サーバーヒントイ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。サーバーヒントイングは、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。さらに詳しくは、「[ConsistentHashFactories](#)」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、サーバーヒントイングが有効にされた状態の **DefaultConsistentHashFactory** に相当します。

サーバーヒントイングは、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](#)

### 28.1. JGROUPS によるサーバーヒントイングの設定

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トイ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 でサーバーヒントイ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](#)

### 28.2. サーバーヒントイングの設定 (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、サーバーヒントイングは、次のようにデフォルトスタックについて **transport** 要素の JGroup サブシステムで設定されます。

手順28.1 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トイングの設定

```
<subsystem xmlns="urn:jboss:domain:jgroups:1.1"
  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](#)

## 28.3. 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28.2 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 clusterName = "MyCluster"
            machineId = "LinuxServer01"
            rackId = "Rack01"
            siteId = "US-WestCoast" />
```

1. **clusterName** 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. **machineId** 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特に有用な属性です。
3. **rackId** パラメーターは、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** パラメーターは、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。

**machineId**、**rackId**、または **siteId** 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](#)

## 28.4. CONSISTENTHASHFACTORIES

Red Hat JBoss Data Grid は、一貫したハッシュアルゴリズムを選択するためのプラグ可能なメカニズムを提供します。これは 4 つの実装に同梱されていますが、カスタム実装を使用することもできます。

JBoss Data Grid には次の 4 つの **ConsistentHashFactory** 実装が同梱されています。

- **DefaultConsistentHashFactory** - すべてのノード間でセグメントが均等に分散されるよ



うにします。ただし、キーマッピングは、各キャッシュの履歴に依存するため、複数のキャッシュ間で同じであることは保証されません。**consistentHashFactory** が指定されない場合は、このクラスが使用されます。

- **SyncConsistentHashFactory** - 現在のメンバーシップが同じである場合、キーマッピングが各キャッシュについて同一であることを保証します。ただし、この欠点として、キャッシュに参加するノードが既存ノードによるセグメントの交換を生じさせる可能性があり、その結果、追加の状態転送のトラフィックが発生するか、またはデータ分散が少なくなるかのいずれか、またはそれらの両方が生じる場合があります。
- **TopologyAwareConsistentHashFactory** - **DefaultConsistentHashFactory** と同等ですが、設定にサーバーヒンテイングが含まれる場合に自動的に選択されます。
- **TopologyAwareSyncConsistentHashFactory** - **SyncConsistentHashFactory** と同等ですが、設定にサーバーヒンテイングが含まれる場合に自動的に選択されます。

一貫したハッシュ実装をハッシュ設定で選択できます。

```
<hash
consistentHashFactory="org.infinispan.distribution.ch.SyncConsistentHashFactory"/>
```

この設定は、同じメンバーを持つキャッシュに同一の一貫したハッシュがあることを保証し、さらに **machineId**、**rackId**、または **siteId** 属性がトランスポート設定で指定される場合、バックアップコピーを複数の物理マシン/ラック/データセンターにまたがって分散させます。

想定される欠点として、この設定により、バランスの再調整時に必要とする以上の数のセグメントが移動する可能性があります。ただし、この影響は、多数のセグメントを使用することによって軽減できます。

もう1つの想定される欠点として、セグメントが均等に分散されないことや、さらには多数のセグメントを実際に使用することによりセグメントの分散状況が悪化することなどがあります。

上述の欠点がありますが、多くの場合、**SyncConsistentHashFactory** と **TopologyAwareSyncConsistentHashFactory** を使用すると、ノードがクラスターに参加した順序に基いてハッシュが計算されないため、クラスター化環境でオーバーヘッドが減少します。また、これらのクラスは通常、各ノードに割り当てられるセグメントの数に大きな違いがあっても許容されるため、デフォルトのアルゴリズムよりも高速になります。

[バグを報告する](#)

### 28.4.1. ConsistentHashFactory の実装

カスタム **ConsistentHashFactory** は、以下のメソッド (これらすべては **org.infinispan.distribution.ch.ConsistentHash** の実装を返します) を使って、**org.infinispan.distribution.ch.ConsistentHashFactory** インターフェースを実装する必要があります。

#### 例28.1 ConsistentHashFactory メソッド

```
create(Hash hashFunction, int numOwners, int numSegments, List<Address>
members, Map<Address, Float> capacityFactors)
updateMembers(ConsistentHash baseCH, List<Address> newMembers,
Map<Address,
```

```
Float> capacityFactors)
rebalance(ConsistentHash baseCH)
union(ConsistentHash ch1, ConsistentHash ch2)
```

現在のところ、カスタムパラメーターを **ConsistentHashFactory** 実装に渡すことはできません。

[バグを報告する](#)

## 28.5. キーアフィニティーサービス

キーアフィニティーサービスを使用すると、分散された **Red Hat JBoss Data Grid** クラスターの特定のノードに値を配置できます。サービスは、識別する指定済みクラスターアドレスに基いて、ハッシュ化されたキーを特定のノードに返します。

キーアフィニティーサービスにより返されたキーは、ユーザー名などの意味を持つことができません。これらは、このレコードのためにアプリケーション全体で使用される無作為の ID にすぎません。提供されたキージェネレーターは、このサービスにより返されるキーが一意であることを保証しません。カスタムキー形式について、**KeyGenerator** の独自の実装を渡すことができます。

このサービスの参照を取得し、使用方法の例を以下に示します。

### 例28.2 キーアフィニティーサービス

```
EmbeddedCacheManager cacheManager = getCacheManager();
Cache cache = cacheManager.getCache();
KeyAffinityService keyAffinityService =
    KeyAffinityServiceFactory.newLocalKeyAffinityService(
        cache,
        new RndKeyGenerator(),
        Executors.newSingleThreadExecutor(),
        100);
Object localKey =
    keyAffinityService.getKeyForAddress(cacheManager.getAddress());
cache.put(localKey, "yourValue");
```

以下の手順は、提供された例の説明です。

### 手順28.3 キーアフィニティーサービスの使用

1. キャッシュマネージャーおよびキャッシュの参照を取得します。
2. これにより、サービスが起動され、キーを生成し、キューに格納するために、提供された **Executor** が使用されます。
3. ローカルノードにマップされるサービスからキーを取得します (**cacheManager.getAddress()** はローカルアドレスを返します)。
4. **KeyAffinityService** から取得されたキーを持つエントリは常に、提供されたアドレスを持つノードに格納されます。この場合は、ローカルノードになります。

[バグを報告する](#)

### 28.5.1. ライフサイクル

**KeyAffinityService** は、**Lifecycle** を拡張します。これにより、キーアフィニティーサービスを停止、起動、および再起動することが可能になります。

#### 例28.3 キーアフィニティーサービスライフサイクルパラメーター

```
public interface Lifecycle {
    void start();
    void stop();
}
```

サービスは、**KeyAffinityServiceFactory** を介してインスタンス化されます。すべてのファクトリーメソッドは非同期キー生成に使用される **Executor** を持つため、これは呼び出し元のスレッドで使用されません。ユーザーはこの **Executor** のシャットダウンを制御します。

**KeyAffinityService** は、必要なくなったときに明示的に停止する必要があります。これにより、バックグラウンドキー生成が停止され、保持された他のリソースがリリースされます。**KeyAffinityService** は、登録したキャッシュマネージャーがシャットダウンされた場合のみ停止します。

[バグを報告する](#)

### 28.5.2. トポロジーの変更

**KeyAffinityService** キーの所有権は、トポロジーが変更されると、変わることがあります。キーアフィニティーサービスは、トポロジーの変更と更新を監視し、古いキーまたは指定されたものと異なるノードにマップされるキーを返さないようにします。ただし、キーが使用されたときにノードアフィニティーが変更されないことは保証されません。以下に例を示します。

1. スレッド (**T1**) は、ノード (**A**) にマップされるキー (**K1**) を読み取ります。
2. トポロジーが変更され、**K1** がノード **B** にマップされます。
3. **T1** は **K1** を使用してキャッシュにデータを追加します。この時点で **K1** は読み取り時に要求されたものと異なるノードである **B** にマップされます。

上記のシナリオは理想的ではありませんが、クラスターの変更中にすでに使用中のキーを移動できるため、アプリケーションのサポートされた動作です。**KeyAffinityService** は安定したクラスターに対してアクセス近接の最適化を提供します。トポロジーの変更が安定的でないときには適用されません。

[バグを報告する](#)

## 第29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べきです。

[バグを報告する](#)

### 29.1. 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29.1 データセンター間レプリケーションの例

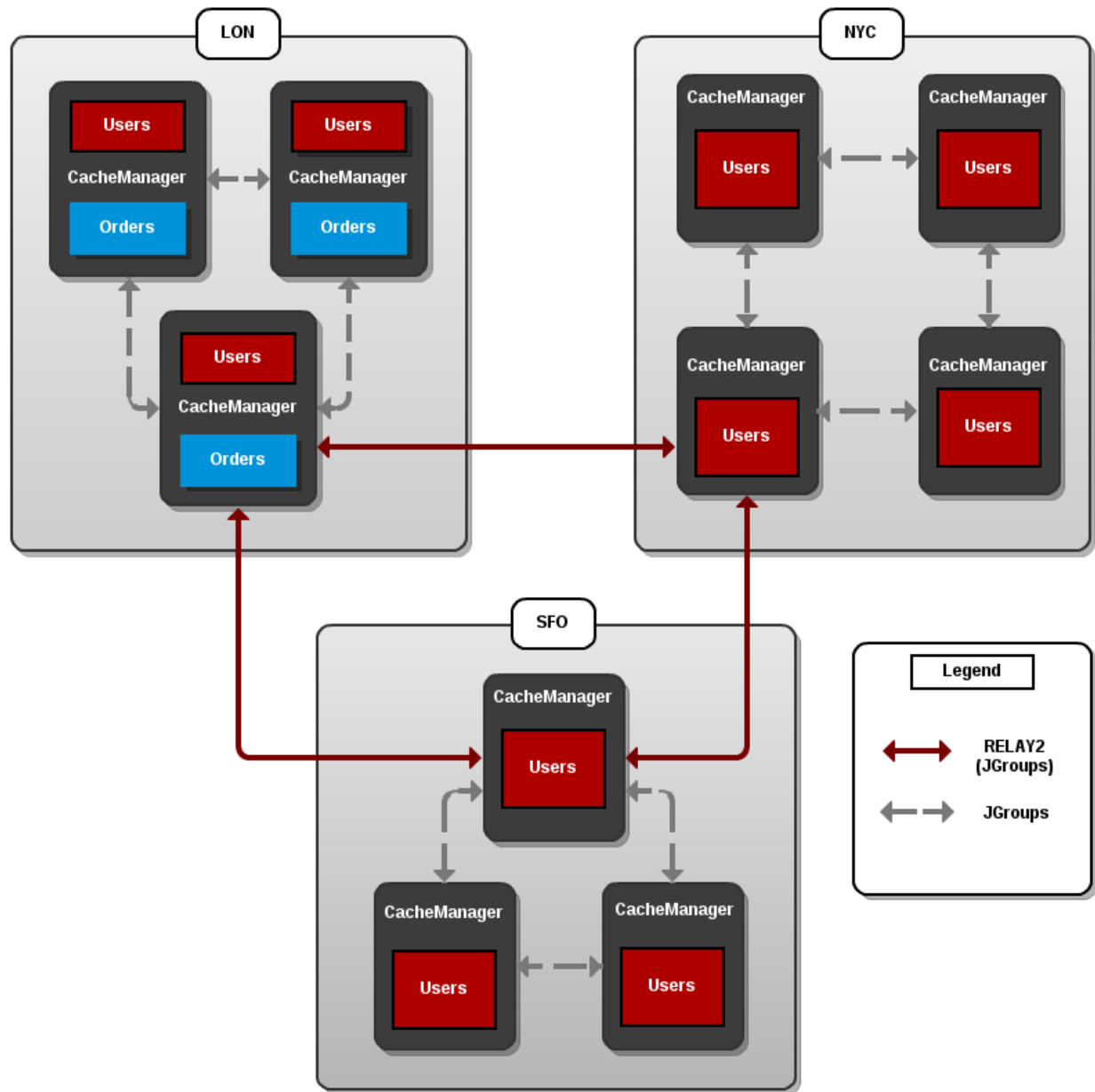


図29.1 データセンター間レプリケーションの例

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 クラスタをホストします。

**Users** キャッシュは、3 つのすべてのサイトでアクティブです。**LON** サイトでの **Users** キャッシュへの変更は、他の 2 つのサイトで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** サイトのローカルでのみ利用可能です。

**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 **numOwners** 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 **numOwners** 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](#)

## 29.2. データセンター間レプリケーションの設定

### 29.2.1. 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次のようにセットアップされます。

#### 手順29.1 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:jboss:domain:jgroups:1.2"
  default-stack="udp">
  <stack name="udp">
    <transport type="UDP"
      socket-binding="jgroups-udp"/>
    <!-- Additional configuration elements here -->
    <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>
    </relay>
  </stack>
</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 **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>
  <!-- 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"/>
```

バグを報告する

## 29.2.2. データセンター間レプリケーション (ライブラリーモード) の設定

### 29.2.2.1. 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。

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次のようにセットアップされます。

### 手順29.2 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>
  <global>
    <site local="SFO" />
    <transport clusterName="default">
      <properties>
        <property name="configurationFile" value="jgroups-
with-relay.xml"/>
      </properties>
    </transport>
    <!-- Additional configuration information here -->
  </global>
  <!-- Additional configuration information here -->
  <namedCache name="lonBackup">
    <sites>
      <backupFor remoteSite="LON"
remoteCache="lon" />
    </sites>
  </namedCache>
</infinispan>
```

- a. **site** 要素を **global** 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** です) を追加します。
- 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。**transport** 要素を追加し、設定ファイルへのパスを **configurationFile** プロパティとしてセットアップします。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取得するには、**NYC** サイトでキャッシュを設定します。
- ii. **LON** からバックアップデータを取得するには、**SFO** サイトでキャッシュを設定します。

## 2. 設定ファイルの内容を追加します。

デフォルトでは、Red Hat JBoss Data Grid には **infinispan-embedded-*{VERSION}*.jar** パッケージ内の **default-configs/default-jgroups-tcp.xml** や **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する別の 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[7800]"
    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** プロトコルについてさらに詳しくは、「[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

### 29.2.2.2. データセンター間レプリケーションをプログラムを用いて設定する

Red Hat JBoss Data Grid でのデータセンター間のレプリケーションを設定するプログラムを用いた方法を以下に示します。

#### 手順29.3 データセンター間レプリケーションをプログラムを用いて設定する

1. ノードの場所を特定します。  
ノードが所属するサイトを宣言します。

```
globalConfiguration.site().localSite("LON");
```

2. JGroups を設定します。  
**RELAY** プロトコルを使用するように JGroups を設定します。

```
globalConfiguration.transport().addProperty("configurationFile",
jgroups-with-relay.xml);
```

3. リモートサイトをセットアップします。  
リモートサイトにレプリケートするために JBoss Data Grid キャッシュをセットアップします。

```
ConfigurationBuilder lon = new ConfigurationBuilder();
lon.sites().addBackup()
    .site("NYC")
    .backupFailurePolicy(BackupFailurePolicy.WARN)
    .strategy(BackupConfiguration.BackupStrategy.SYNC)
```

```

        .replicationTimeout(12000)
        .sites().addInUseBackupSite("NYC")
    .sites().addBackup()
        .site("SFO")
        .backupFailurePolicy(BackupFailurePolicy.IGNORE)
        .strategy(BackupConfiguration.BackupStrategy.ASYNC)
        .sites().addInUseBackupSite("SFO")

```

#### 4. オプション: バックアップクラッシュを設定します。

JBoss Data Grid は、リモートサイトと同じ名前を使って、データをキャッシュに暗黙的にレプリケートします。リモートサイトのバックアップキャッシュが異なる名前を持つ場合、ユーザーは、データが正しいキャッシュにレプリケートされていることを確認するために **backupFor** キャッシュを指定する必要があります。



#### 注記

この手順は任意であり、リモートサイトのキャッシュの名前が元のキャッシュとは異なるものに設定される場合にのみ必要になります。

- a. **LON** からのバックアップデータを受信できるようにサイト **NYC** のキャッシュを設定します。

```

ConfigurationBuilder NYCbackupOfLon = new ConfigurationBuilder();
NYCbackupOfLon.sites().backupFor().remoteCache("lon").remoteSite(
    "LON");

```

- b. **LON** からバックアップデータを受信できるようにサイト **SFO** のキャッシュを設定します。

```

ConfigurationBuilder SFObackupOfLon = new ConfigurationBuilder();
SFObackupOfLon.sites().backupFor().remoteCache("lon").remoteSite(
    "LON");

```

#### 5. 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>
    <!-- Additional configuration information here -->
    <relay.RELAY2 site="LON"
        config="relay.xml"
        relay_multicasts="false" />
</config>

```

#### 6. 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>

```

#### 7. 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する別の 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発見します。**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" <!-- Additional configuration information
  here --> />
  <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** プロトコルについてさらに詳しくは、「[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](#)

## 29.3. 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](#)

### 29.3.1. 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)

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、サイトが自動的にオフラインにする設定を行うために、**take-offline** 要素が **backup** 要素に追加されます。

#### 例29.2 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](#)

### 29.3.2. サイトをオフラインにする (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、**backups** 要素内ですべてのバックアップサイトを定義した後に **backupFor** 要素を使用します。

**例29.3 サイトをオフラインに設定する (ライブラリーモード)**

```
<backup>
  <takeOffline afterFailures="${NUM}"
               minTimeToWait="${PERIOD}"/>
</backup>
```

サイトを自動的にオフラインにする設定を行うには、**takeOffline** 要素を **backup** 要素に追加します。

- **afterFailures** 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を失敗できる回数を指定します。デフォルト値 (**0**) は、**minTimeToWait** が **0** より小さい値の場合に失敗の回数を無限にすることを許可します。**minTimeToWait** が **0** より小さい値でない場合、**afterFailures** は、その値が負の値であるかのように動作します。つまり、このパラメーターが負の値である場合、**minTimeToWait** で指定される時間が経過した後にサイトがオフラインになることを示します。
- **minTimeToWait** 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。サイトは、**afterFailures** パラメーターで指定される試行回数に達し、最初の失敗後に **minTimeToWait** で指定される時間が経過した後にオフラインになります。このパラメーターが **0** 以下の値に設定される場合、このパラメーターは無視され、サイトは **afterFailures** パラメーターのみに基づいてオフラインになります。

[バグを報告する](#)

**29.3.3. サイトをオフラインにする (プログラムを用いた場合)**

Red Hat JBoss Data Grid において、プログラムを用いてデータセンター間のレプリケーションサイトを自動的にオフラインに設定するには、以下のようにします。

**例29.4 サイトをオフラインにする (プログラムを使用)**

```
lon.sites().addBackup()
    .site("NYC")
    .backupFailurePolicy(BackupFailurePolicy.FAIL)
    .strategy(BackupConfiguration.BackupStrategy.SYNC)
    .takeOffline()
    .afterFailures(500)
    .minTimeToWait(10000);
```

[バグを報告する](#)

**29.3.4. 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](#)

### 29.3.5. 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、『開発者ガイド』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](#)

### 29.3.6. 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](#)

## 29.4. 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。

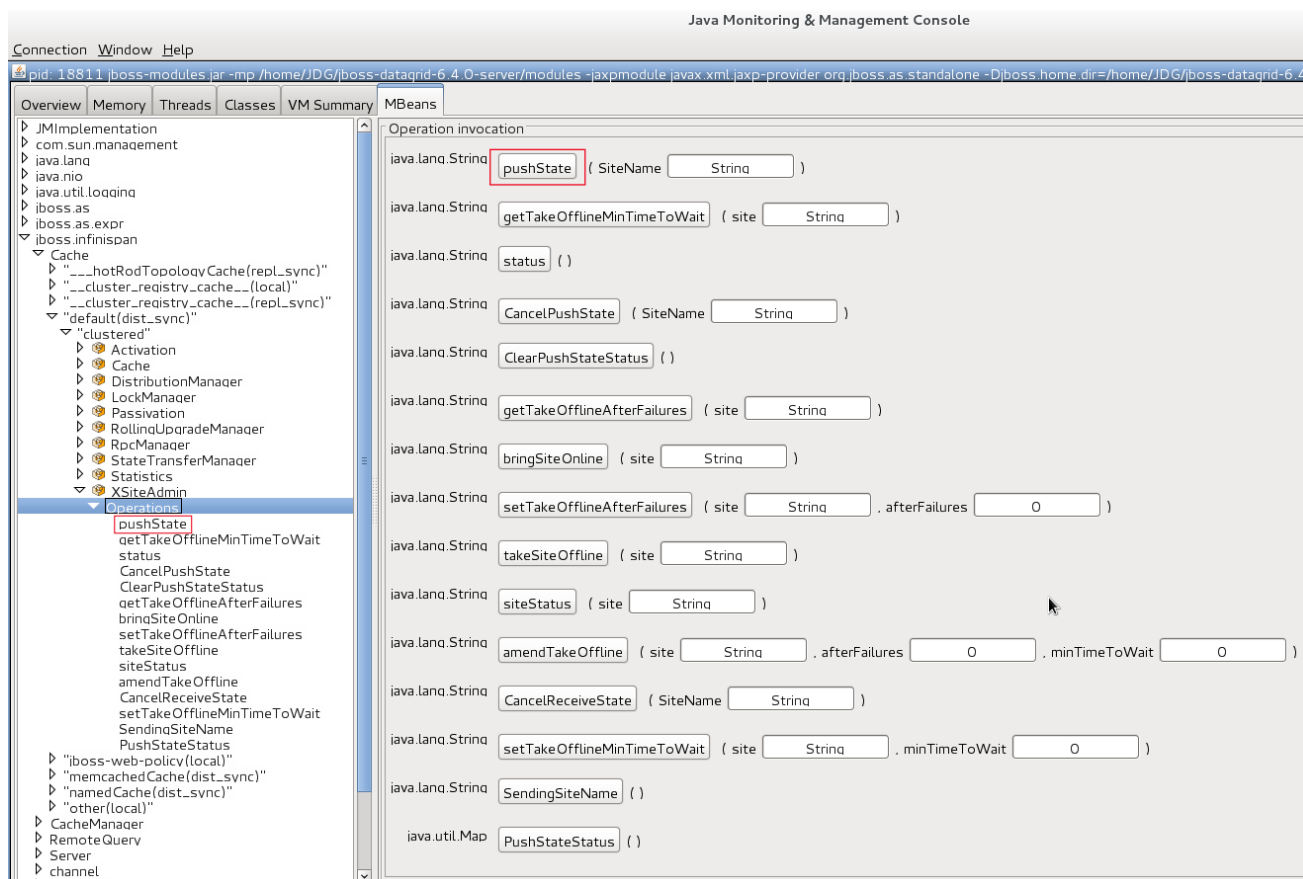
マスターサイト A とバックアップサイト B の 2 つのサイトがあるとします。クライアントは最初にマスターサイト A のみにアクセスでき、バックアップサイト B は隠れたバックアップとして動作します。サイト間の状態転送は、双方向でプッシュできます。新しいバックアップサイト B がオンラインになると、マスターサイト A と状態を同期するために、状態の転送を開始してマスターサイト A からバックアップサイト B に状態をプッシュできます。

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29.2 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](#)

### 29.4.1. 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](#)

### 29.4.2. 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](#)

### 29.4.3. 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](#)

## 29.5. 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](#)

### 29.5.1. 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](#)

### 29.5.2. 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29.4** 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、ノードが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を持たない状況で必要になります。

[バグを報告する](#)

### 29.5.3. 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29.5 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を設定します。

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「[データセンター間レプリケーションをプログラムを用いて設定する](#)」の手順を使用します。

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30章 セッションの外部化

### 30.1. JBOSS EAP 6.X 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。

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的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。

#### 手順30.1 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:1.5">
  [...]
  <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" batching="true">
      <remote-store cache="default" socket-timeout="60000" preload="true" passivation="false" purge="false" shared="true">
        <remote-server outbound-socket-binding="remote-jdg-server1"/>
        <remote-server outbound-socket-binding="remote-jdg-server2"/>
      </remote-store>
    </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の名前であり、**default-cache** はこのコンテナ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 version="6.0"
  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_6_0.xsd">

  <replication-config>
    <replication-trigger>SET</replication-trigger>
    <replication-granularity>SESSION</replication-granularity>
    <cache-name>jboss.infinispan.cacheContainer.default-
cache</cache-name>
  </replication-config>

  <passivation-config>
    <use-session-passivation>true</use-session-passivation>
    <passivation-min-idle-time>900</passivation-min-idle-time>
    <passivation-max-idle-time>1800</passivation-max-idle-time>
  </passivation-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31章 ネットワークパーティションの処理 (スプリットブ레인)

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戦略を提供します。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、キャッシュは複数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 **numOwners** 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指定されます。結果として、障害が発生したノードの数が **numOwners** 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、**numOwners** 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、**numOwners** 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레인**: 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**: **numOwners** 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](#)

### 31.1. スプリットブレインの検出および回復

スプリットプレー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ともに独自の 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))、キャッ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、`numOwners` 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ー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জ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](#)

## 31.2. 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、**numOwners** 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 **numOwners** の適切な値を設定することです。

[バグを報告する](#)

## 31.3. ネットワークパーティションリカバリの例

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。

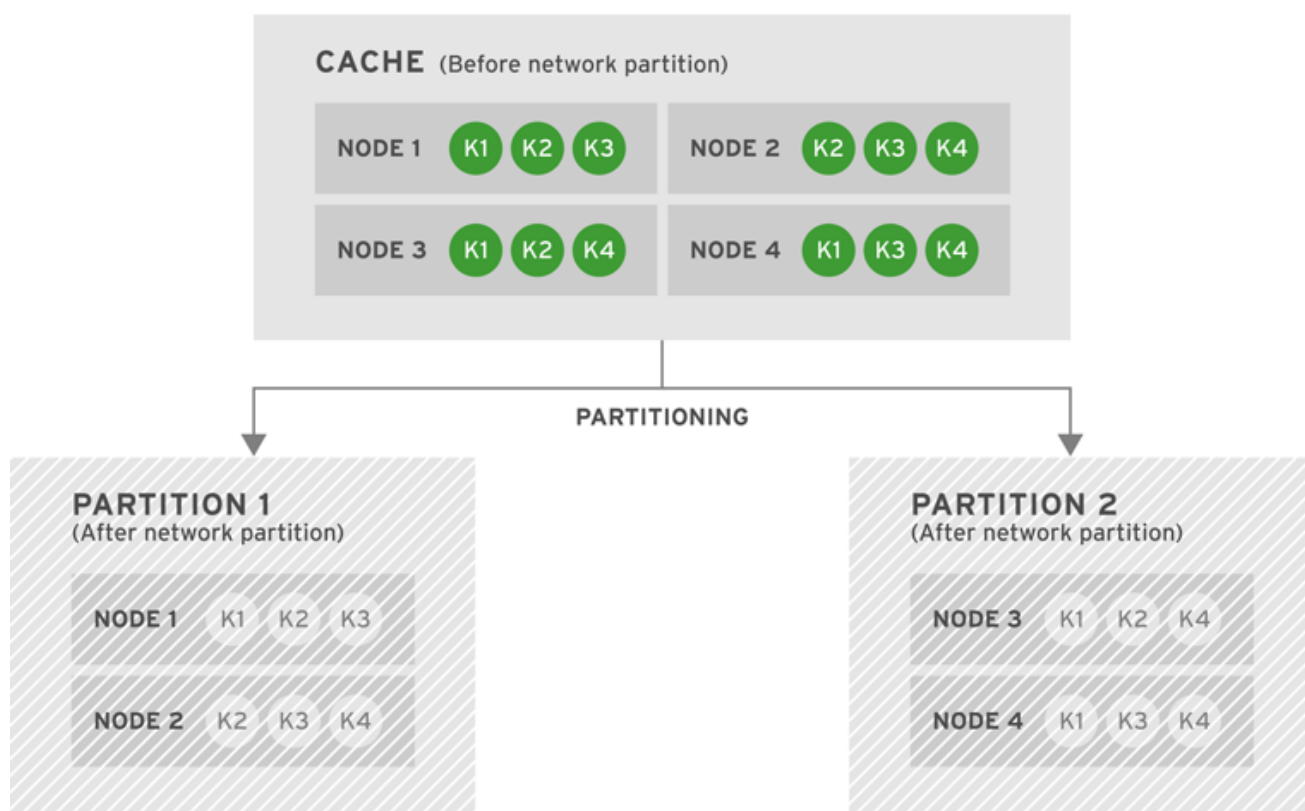
1. **numOwners** が 3 に設定された分散 4 ノードクラスター (「[NumOwners が 3 の分散 4 ノード キャッシュの例](#)」を参照)
2. **numOwners** が 2 に設定された分散 4 ノードクラスター (「[NumOwners が 2 の分散 4 ノード キャッシュの例](#)」を参照)
3. **numOwners** が 3 に設定された分散 5 ノードクラスター (「[NumOwners が 3 の分散 5 ノード キャッシュの例](#)」を参照)

4. **numOwners** が 4 に設定されたレプリケート 4 ノードクラスター (「[NumOwners 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. **numOwners** が 5 に設定されたレプリケート 5 ノードクラスター (「[NumOwners 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. **numOwners** が 8 に設定されたレプリケート 8 ノードクラスター (「[NumOwners 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](#)

### 31.3.1. NumOwners が 3 の分散 4 ノードキャッシュの例

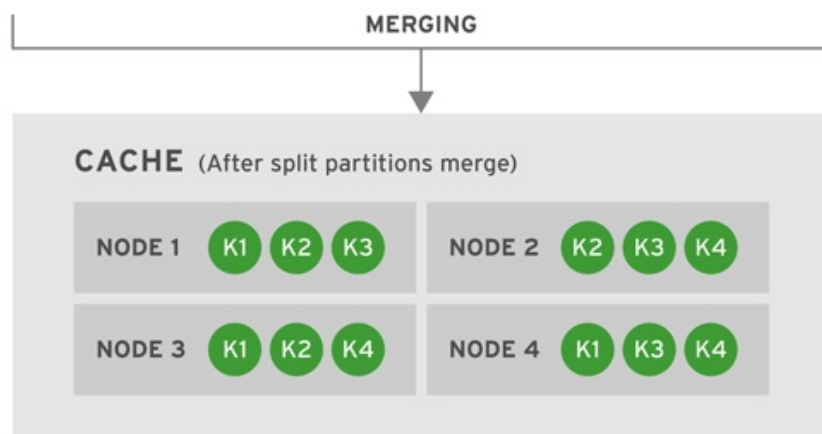
最初のシナリオ例には、4 つのデータエントリー (**k1**、**k2**、**k3**、および **k4**) を含む 4 ノード分散キャッシュが含まれます。このキャッシュでは、**numOwners** が 3 と等しくなります。つまり、各データエントリーごとに、キャッシュ内のさまざまなノードの 3 つのコピーが必要です。



JBOSS\_11\_335370\_0415

図31.1 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 (**numOwners** 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)。



JBoss\_12\_335370\_0415

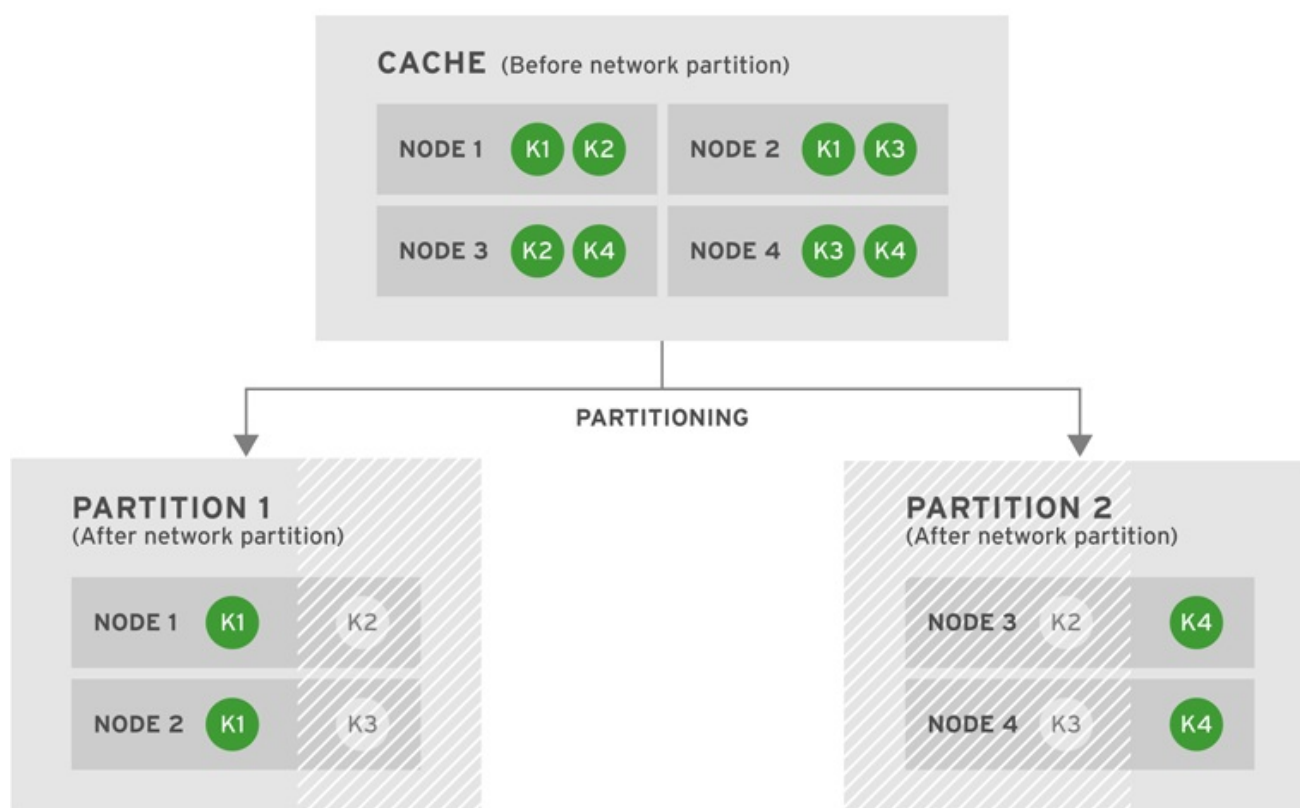
図31.2 パーティションのマージ後のキャッシュ

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](#)

### 31.3.2. NumOwners が 2 の分散 4 ノードキャッシュの例

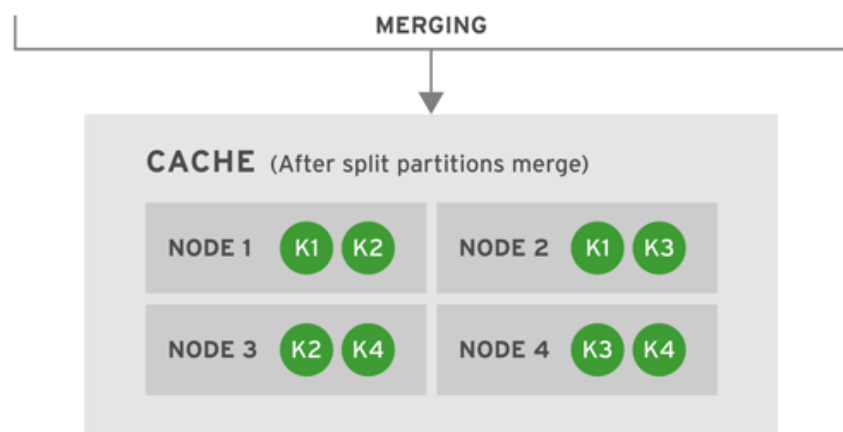
2 つ目のシナリオ例には、4 つのノードから構成される分散キャッシュが含まれます。このシナリオでは、**numOwners** が 2 となり、4 つのデータエントリー (k1、k2、k3、および k4) それぞれがキャッシュ内に 2 つのコピーを持ちます。



JBoss\_13\_335370\_0615

図31.3 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 (**numOwners** 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。



JBoss\_14\_335370\_0415

図31.4 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](#)

### 31.3.3. NumOwners 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、**numOwners** が 3 に等しい分散キャッシュが含まれます。

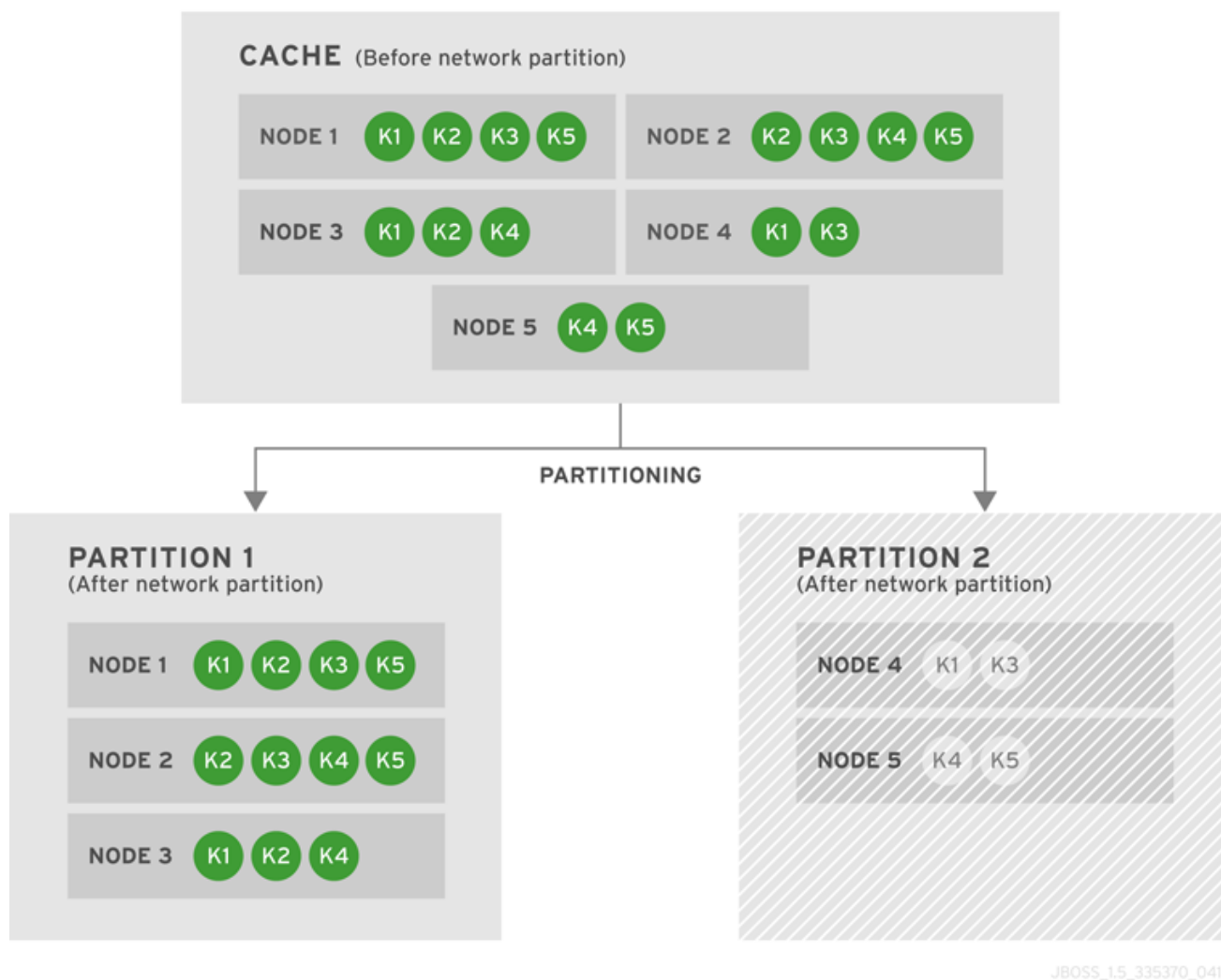
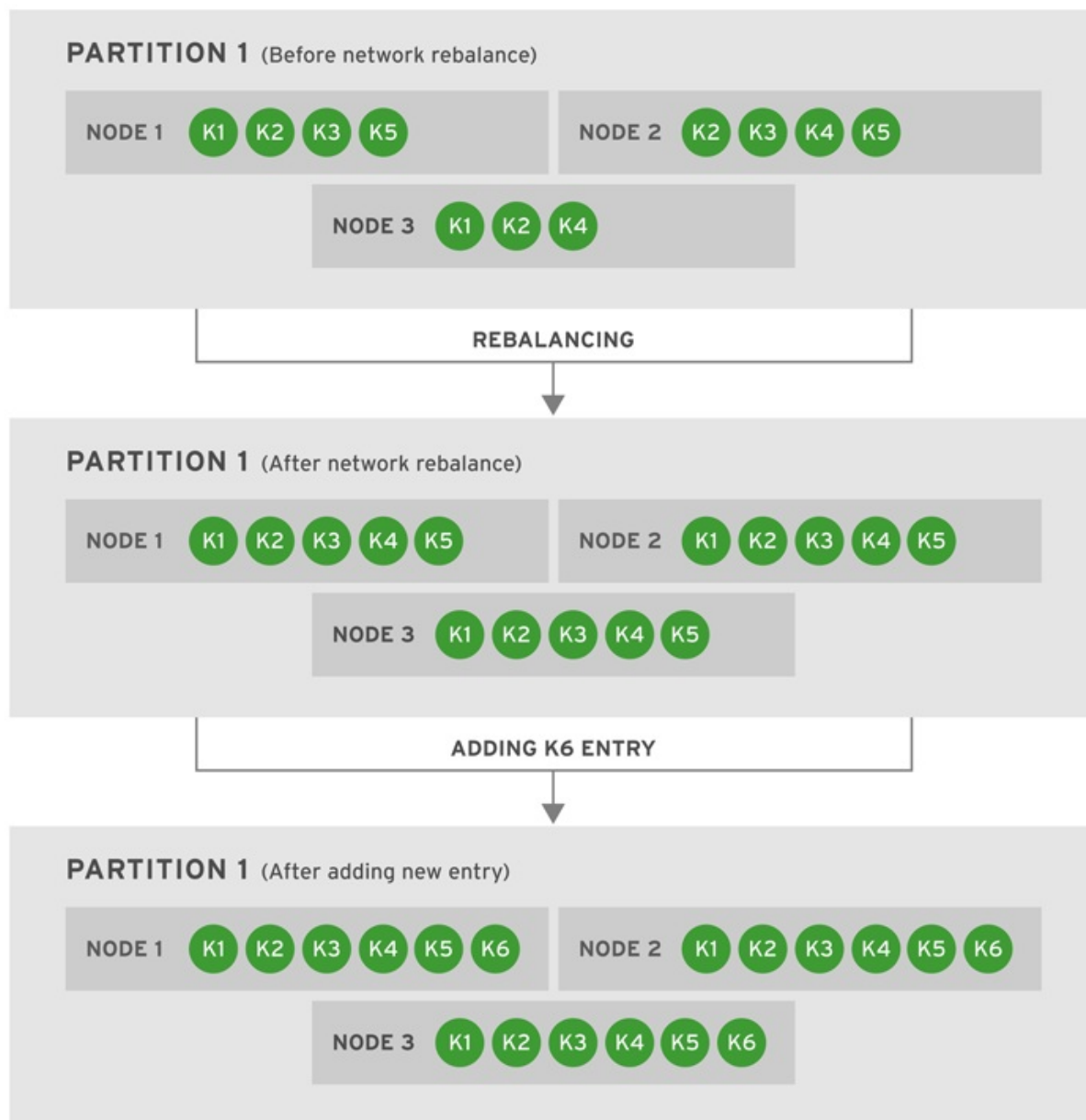


図31.5 ネットワークパーティション実行前とネットワークパーティション実行後のキャッシュ

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、**numOwners** 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、新しいエントリをパーティション2に追加することはできません。

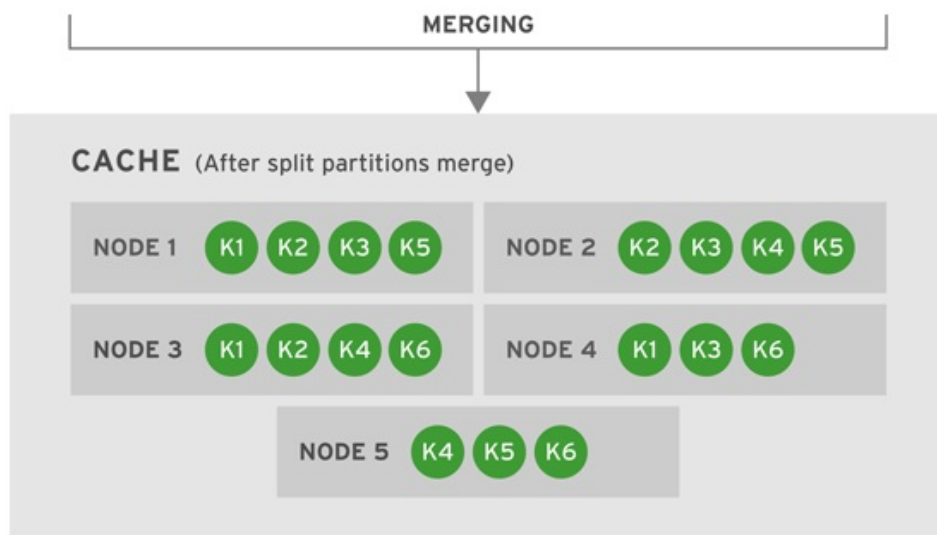


JBOSSE\_1.6\_335370\_0415

図31.6 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。**numOwners** 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。





JBoss\_17\_335370\_0415

図31.7 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 (**numOwners=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](#)

#### 31.3.4. NumOwners 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、**numOwners**が4に等しいレプリケートキャッシュが含まれます。

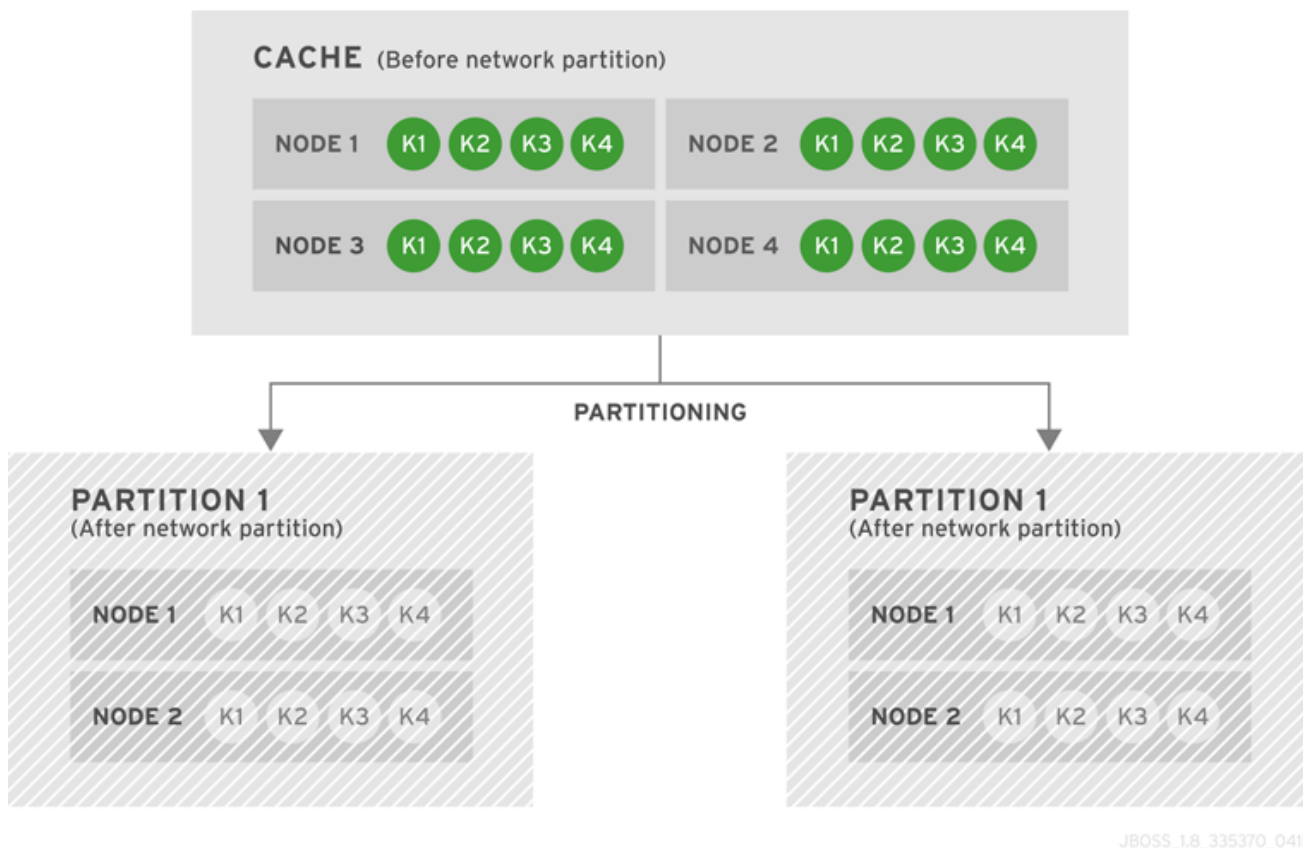


図31.8 ネットワークパーティション実行前とネットワークパーティション実行後のキャッシュ

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。

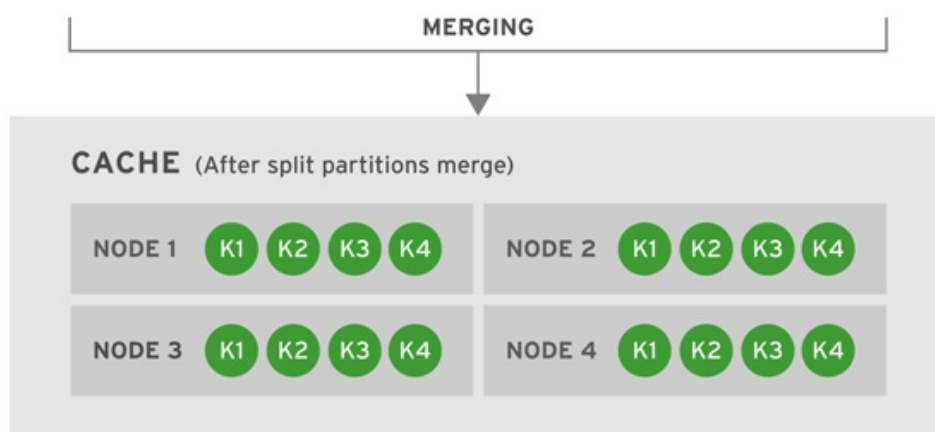


図31.9 パーティションのマージ後のキャッシュ

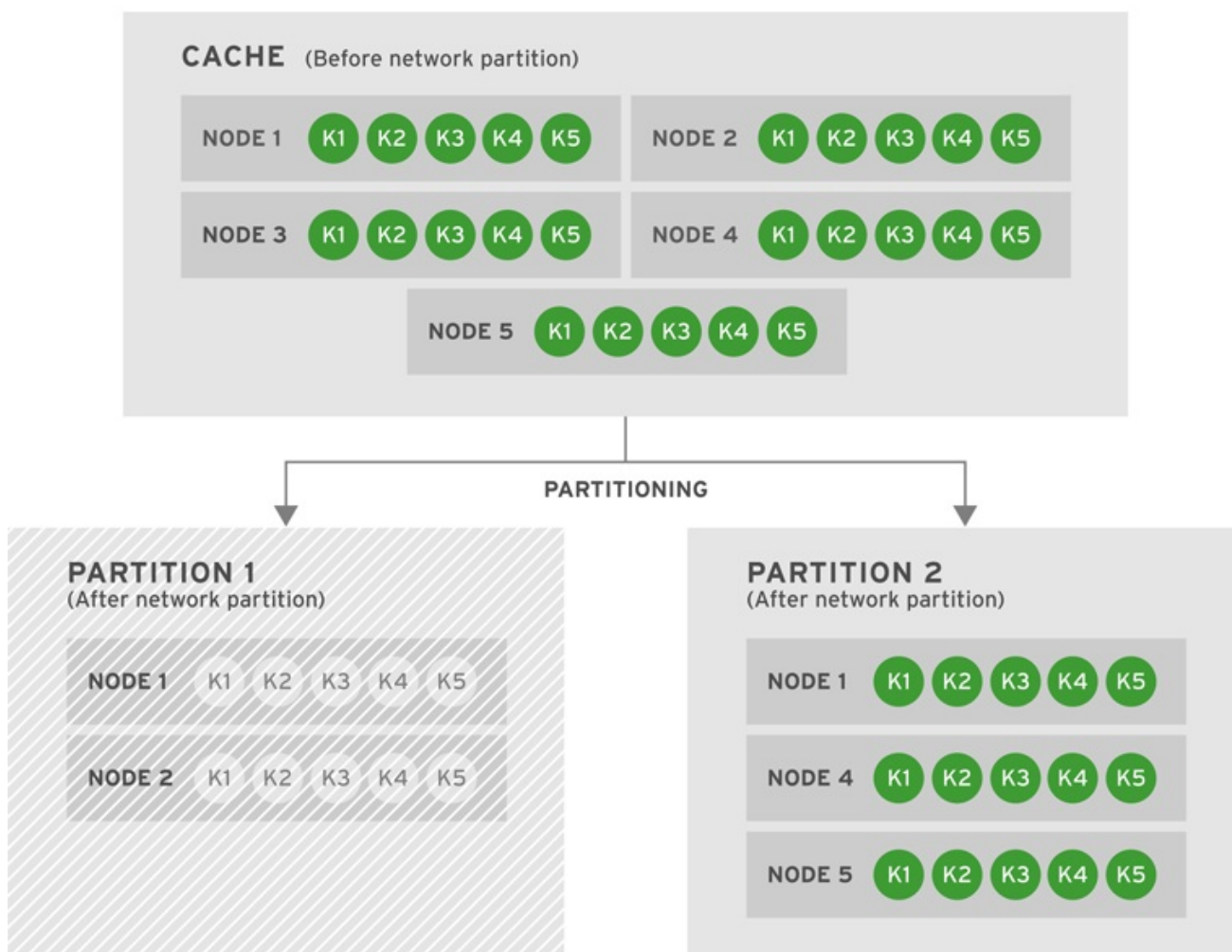
JBoss Data Grid は結果的に、分割された2つのパーティションを1つのキャッシュにマージします。状態転送は不要であり、キャッシュは利用可能モードで元の状態に戻ります (4つのノードと4つのデータエントリ (**k1**、**k2**、**k3**、および **k4**) から構成されます)。

[バグを報告する](#)



### 31.3.5. NumOwners 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、*numOwners* 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。



JBoss\_110\_335370\_0415

図31.10 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。

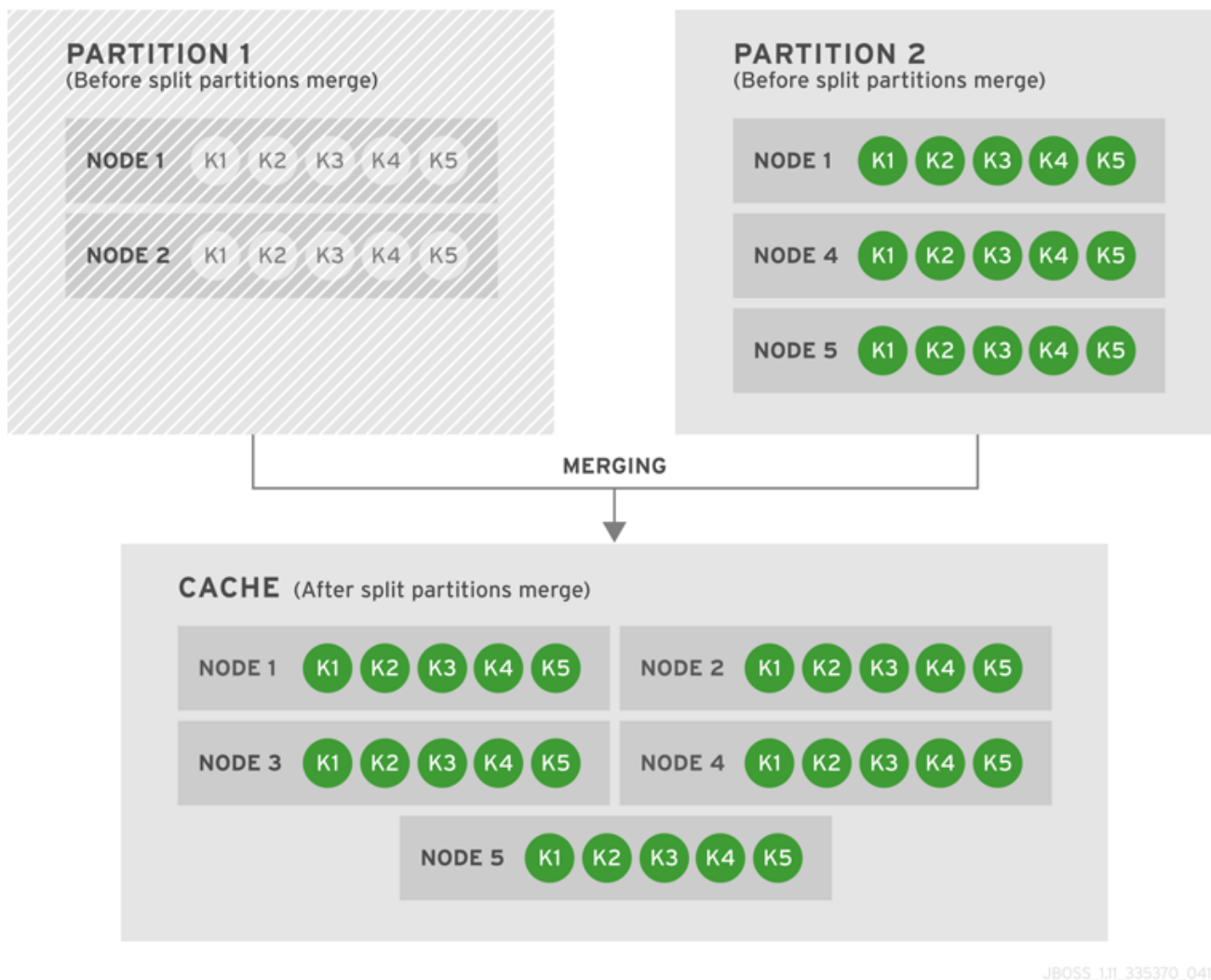


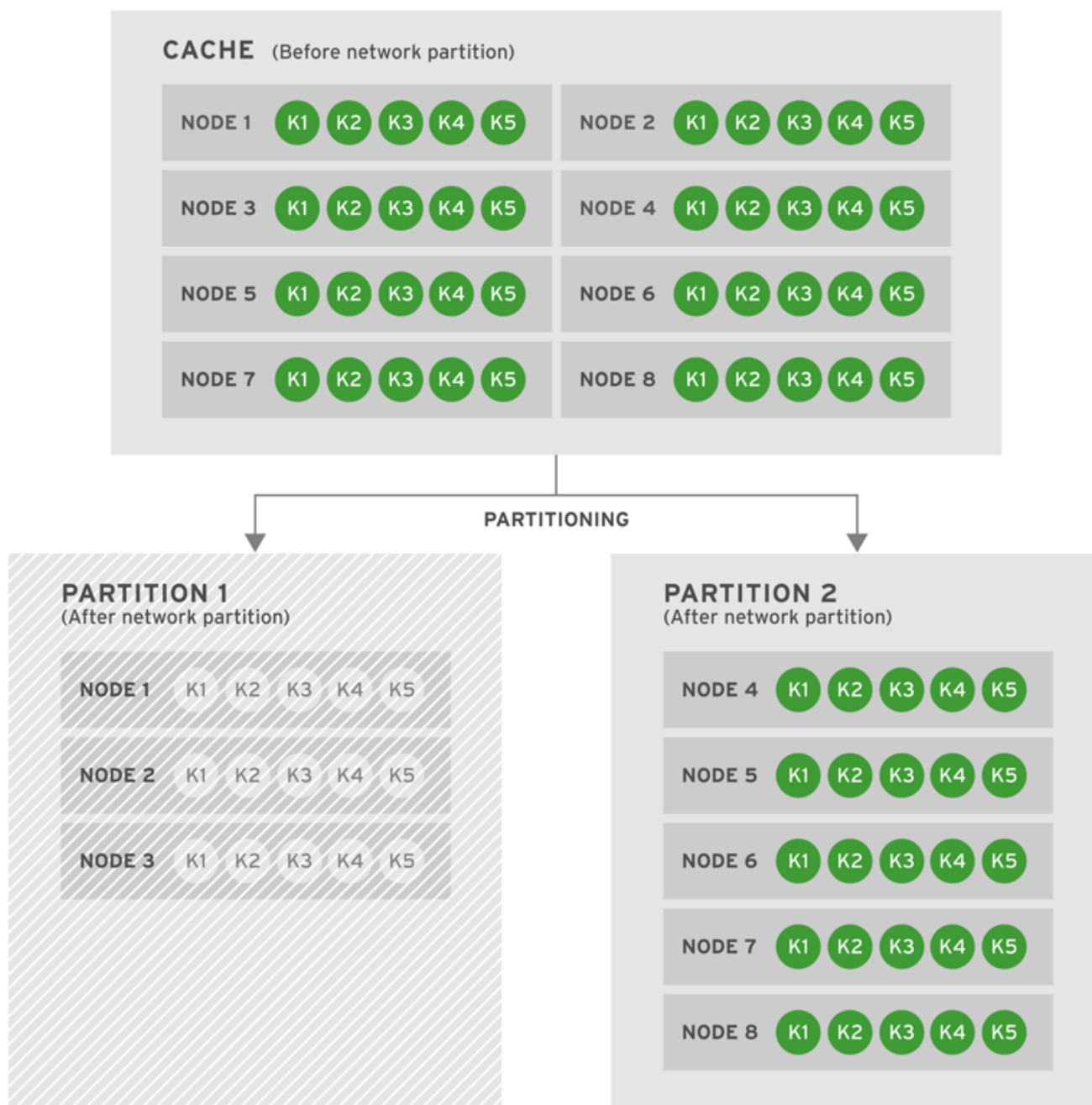
図31.11 両方のパーティションが1つのキャッシュにマージされる

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](#)

### 31.3.6. NumOwners 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、**numOwners** 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。



JBoss\_112\_335370\_0415

図31.12 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。

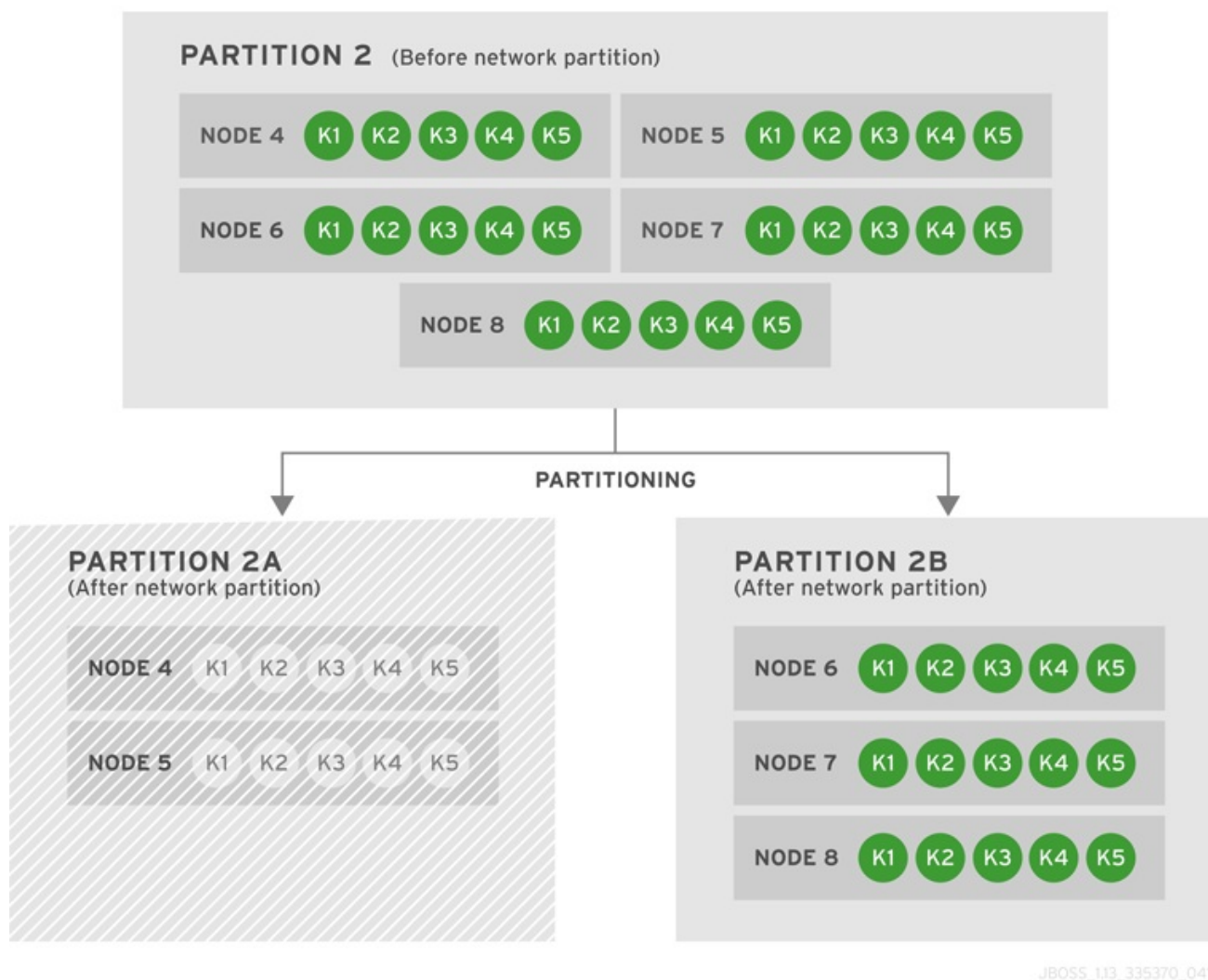


図31.13 パーティション 2 はさらにパーティション 2A と 2B に分割される

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。

#### 考えられる解決シナリオ

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

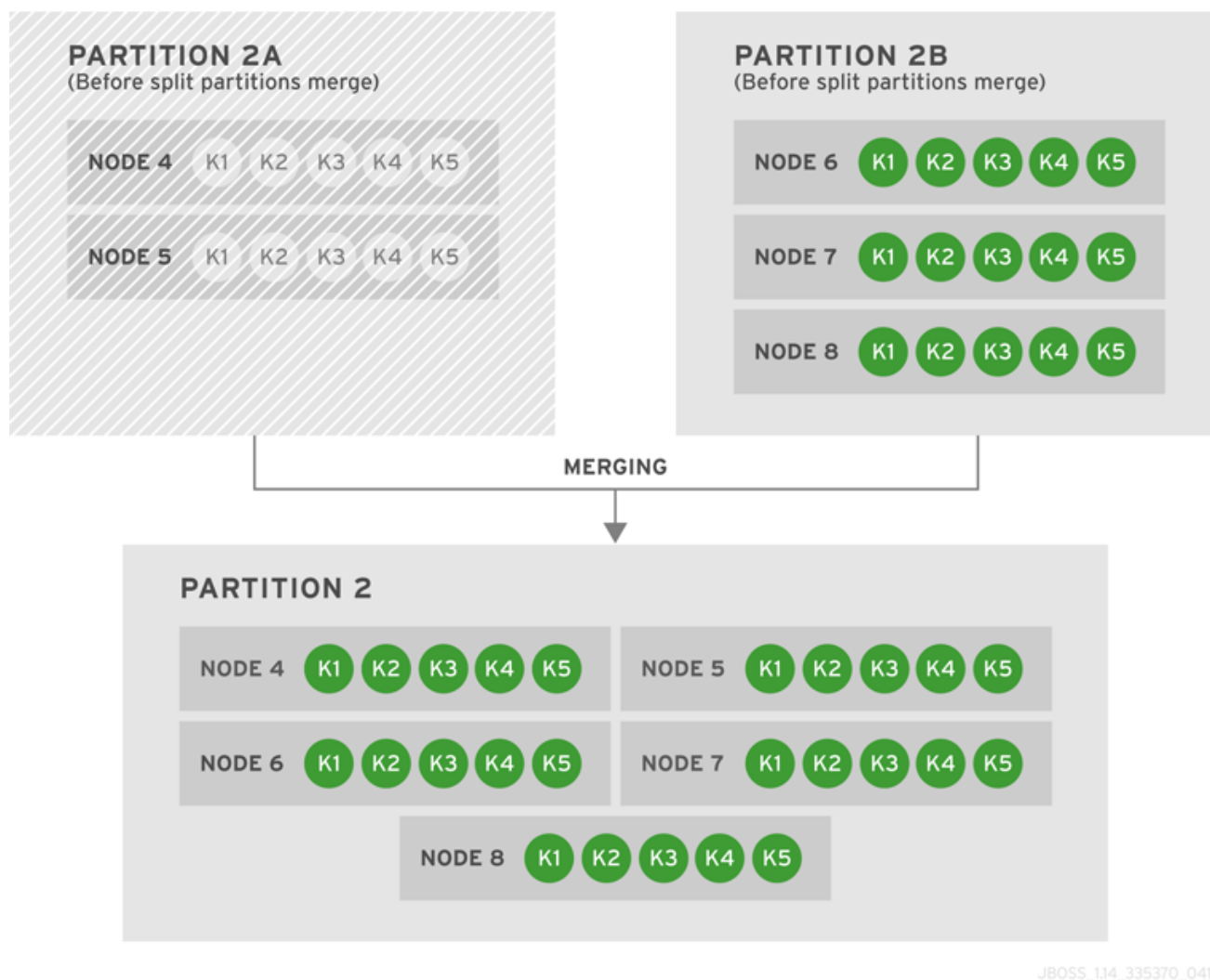


図31.14 ケース 1: パーティション 2A と 2B のマージ

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。

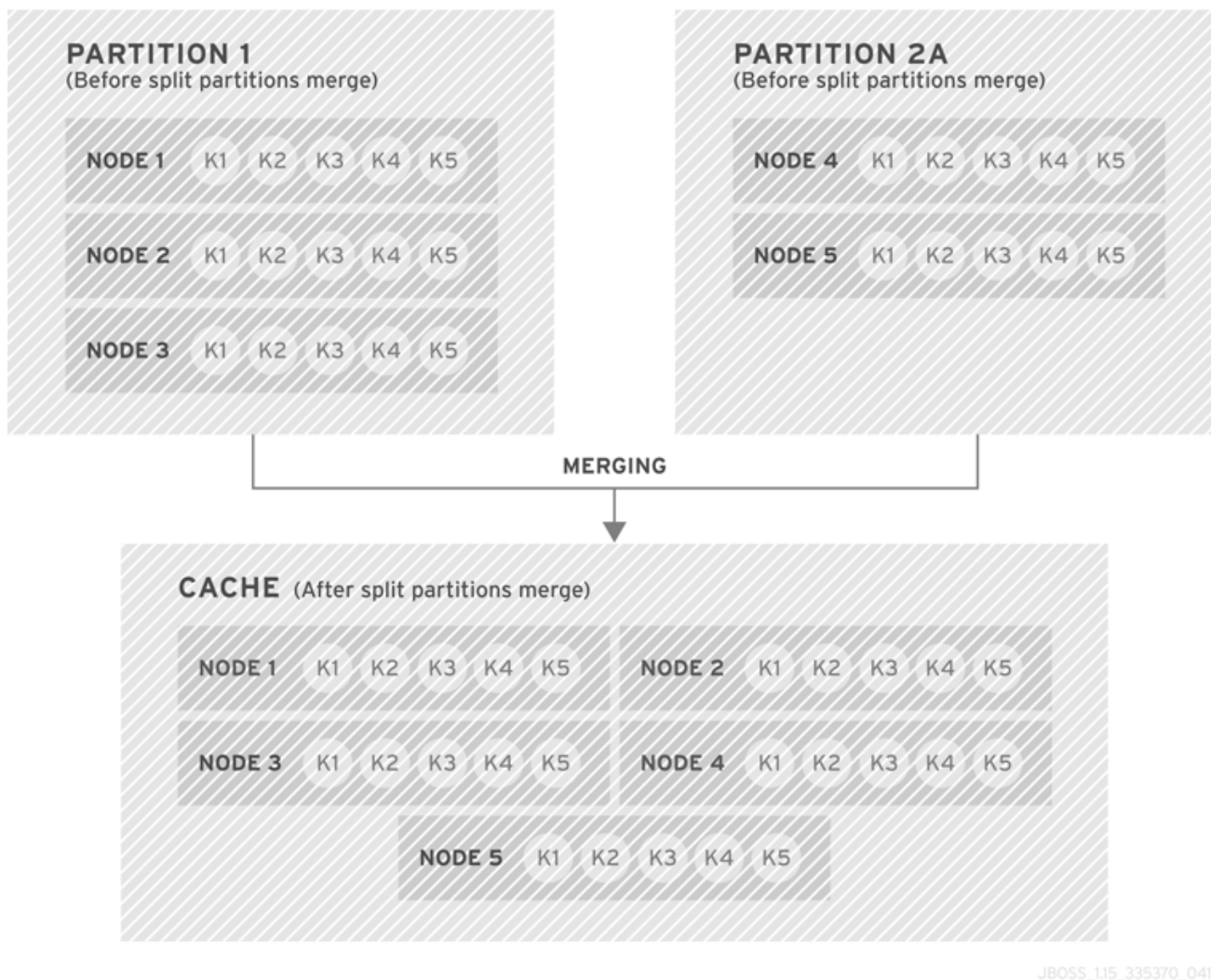


図31.15 ケース 2: パーティション 1 と 2A のマージ

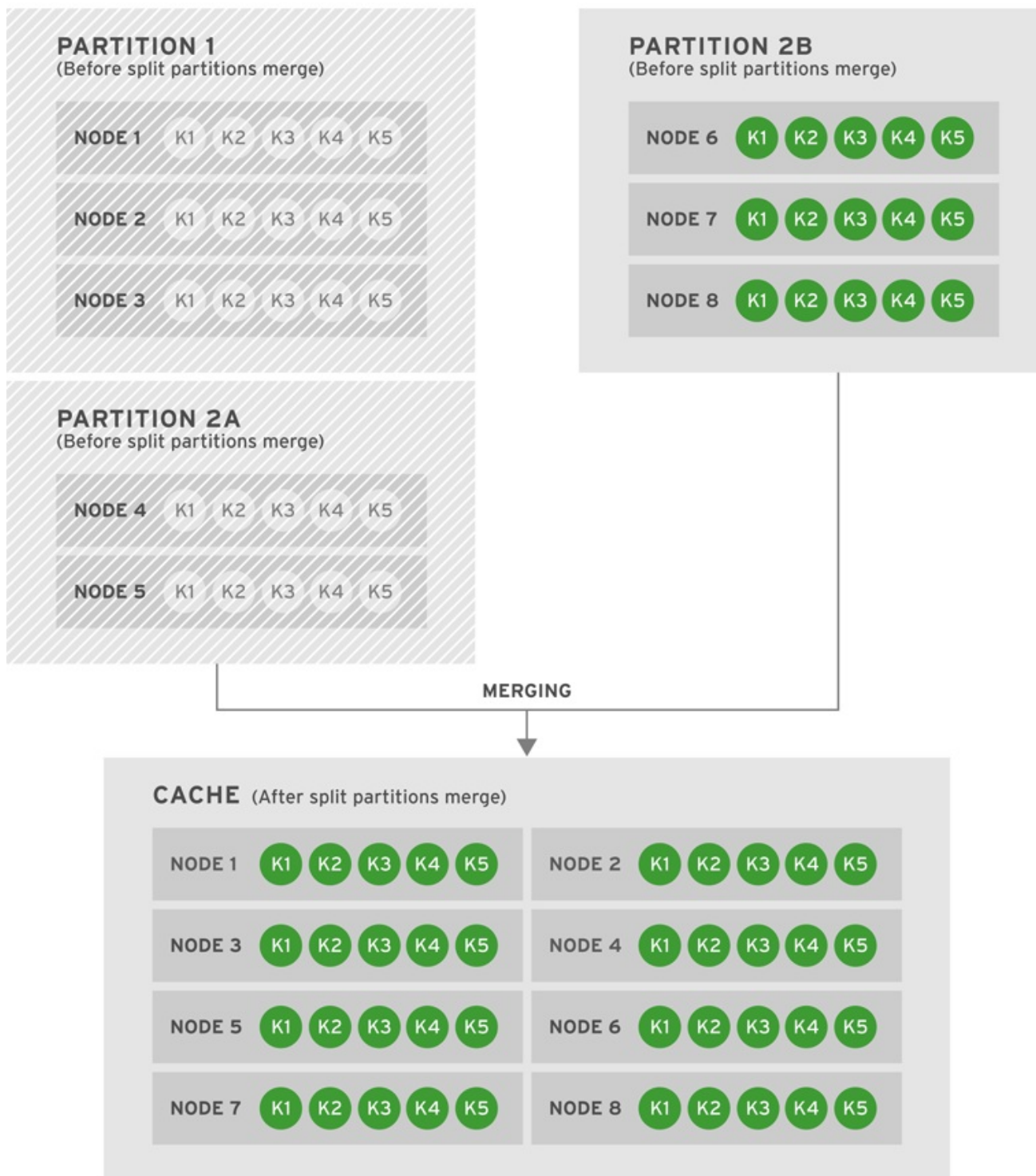
パーティション分割されたネットワークの 2 つ目の解決法では、パーティション 1 とパーティション 2A をマージします。マージされたパーティションには、ノード 1、ノード 2、ノード 3、ノード 4、およびノード 5 が含まれます。どちらのパーティションも最新の安定したトポロジーを含まないため、結果としてマージされたパーティションは劣化モードのままになります。





図31.16 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。



JBoss\_117\_335370\_0415

図31.17 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](#)

## 31.4. 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

パーティション処理を以下のように宣言します。

```
<namedCache name="myCache">
  <clustering mode="distribution">
    <partitionHandling enabled="true" />
  </clustering>
</namedCache>
```

### プログラミングによる設定

パーティション処理を以下のようにプログラミングします。

```
ConfigurationBuilder dcc = new ConfigurationBuilder();
dcc.clustering().partitionHandling().enabled(true);
```

[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 (Protocol)	詳細
TCP	<p>TCP/IP は、IP マルチキャストが使用できなくなる状況で利用できる 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。</p> <p>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。</p> <ul style="list-style-type: none"> <li>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。</li> <li>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。</li> </ul> <p>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。</p> <p>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。</p>
UDP	<p>UDP は、以下を使用するトランスポートプロトコルです。</p> <ul style="list-style-type: none"> <li>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。</li> <li>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。</li> </ul> <p>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。チャン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。</p>

プロトコル (Protocol)	詳細
PING	<p>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。</p> <p>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。</p> <p>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。</p>
TCPPING	<p>TCPPING 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。</p>
MPING	<p>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。</p>
S3_PING	<p>S3_PING は、Amazon の Elastic Compute Cloud (EC2) で使用するのに理想的な検出プロトコルです。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。</p> <p>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。</p>
MERGE3	<p>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。</p>

プロトコル (Protocol)	詳細
FD_ALL	<p>障害検出に使用される <b>FD_ALL</b> 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、<b>P</b> からデータまたはハートビートを受け取ると、<b>P</b> 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。</p>
FD SOCK	<p><b>FD SOCK</b> 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>B</b> は、近接メンバー <b>A</b> が TCP ソケットの異常な終了 (通常はノード <b>B</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>B</b> が正常に脱退した場合、メンバー <b>B</b> はメンバー <b>A</b> に通知し、脱退しても疑われません。</p>
FD_HOST	<p><b>FD_HOST</b> は、すべてのホストのクラッシュまたはハングを検出し、<b>ICMP ping</b> 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。  <b>FD_HOST</b> 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、<b>FD_HOST</b> は、<b>FD_ALL</b> や <b>FD SOCK</b> などの他の障害検出プロトコルとともに使用されます。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。</p> <p><b>FD_HOST</b> プロトコルは、JBoss Data Grid 向けの Windows でサポートされます。<b>cmd</b> パラメーターを <b>ping.exe</b> に設定し、<b>ping</b> 数を指定する必要があります。</p>
VERIFY_SUSPECT	<p><b>VERIFY_SUSPECT</b> プロトコルは、メンバーを除外する前にメンバーに <b>ping</b> 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。</p>

プロトコル (Protocol)	詳細
NAKACK2	<p>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。</p> <p>NAKACK2 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。</p>
UNICAST3	<p>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) プロパティを使用します。</p> <p>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 を受信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。</p>
STABLE	<p>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。</p>
GMS	<p>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。</p>
MFC	<p>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。</p>
UFC	<p>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。</p>

プロトコル (Protocol)	詳細
FRAG2	<p>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。</p>
ENCRYPT	<p>JGroups には、クラスタートラフィック向けの暗号化を提供する 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 <code>encrypt_entire_message</code> が <code>true</code> である必要があります。また、ENCRYPT は、暗号化する必要があるヘッダーではすべてのプロトコル以下である必要があります。</p> <p>ENCRYPT レイヤーは、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。JGroups ENCRYPT プロトコルは、以下の 2 つの方法で使用できます。</p> <ul style="list-style-type: none"> <li>• キーストアの <code>secretKey</code> で設定されます。</li> <li>• アルゴリズムとキーサイズで設定されます。</li> </ul> <p>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。</p>
SASL	<p>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。</p>

プロトコル (Protocol)	詳細
RELAY2	<p>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。</p> <p>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。</p> <p>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。</p> <ul style="list-style-type: none"> <li>• 3つ以上のサイトを接続します。</li> <li>• 自律的に機能し、相互に認識しないサイトに接続します。</li> <li>• 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。</li> </ul>

[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。

表A.2 TCPの推奨値とデフォルト値

パラメーター	デフォルト値 (JGroups)	推奨値 (JBoss Data Grid)
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

パラメーター	デフォルト値 (JGroups)	推奨値 (JBoss Data Grid)
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、 <b><i>thread_pool.min_threads</i></b> 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。
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



パラメーター	デフォルト値 (JGroups)	推奨値 (JBoss Data Grid)
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 6.5 は JGroups 3.6.3 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*** 値は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。

表A.3 MPING の推奨値

パラメーター	デフォルト値 (JGroups)	推奨値 (JBoss Data Grid)
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*** 値は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	デフォルト値 (JGroups)	推奨値 (JBoss Data Grid)
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	デフォルト値 (JGroups)	推奨値 (JBoss Data Grid)
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	デフォルト値 (JGroups)	推奨値 (JBoss Data Grid)
--------	------------------	-----------------------

パラメーター	デフォルト値 (JGroups)	推奨値 (JBoss Data Grid)
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 <b>stop the world</b> 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1GB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 <b>false</b> 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 <b>interval</b> 値は、FD_ALL の <b>timeout</b> 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	デフォルト値 (JGroups)	推奨値 (JBoss Data Grid)
check_timeout	3,000	5,000
cmd	InetAddress.isReachable() (ICMP ping)	-
間隔	20,000	15,000。FD_HOST の <b>interval</b> 値は、FD_HOST の <b>timeout</b> 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	デフォルト値 (JGroups)	推奨値 (JBoss Data Grid)
timeout	2,000	5,000

表A.9 pbcast.NAKACK2の推奨値

パラメーター	デフォルト値 (JGroups)	推奨値 (JBoss Data Grid)
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	デフォルト値 (JGroups)	推奨値 (JBoss Data Grid)
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	デフォルト値 (JGroups)	推奨値 (JBoss Data Grid)
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	デフォルト値 (JGroups)	推奨値 (JBoss Data Grid)
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	デフォルト値 (JGroups)	推奨値 (JBoss Data Grid)
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	デフォルト値 (JGroups)	推奨値 (JBoss Data Grid)
frag_size	60,000	デフォルト値と同じ

表A.15 ENCRYPT の推奨値

パラメーター	デフォルト値 (JGroups)	推奨値 (JBoss Data Grid)
asymAlgorithm	RSA	-
asymInit	512	-
asymProvider	Bouncy Castle Provider	-
changeKeysOnViewChange	false	-
symAlgorithm	AES	-
symInit	128	-
symProvider	Bouncy Castle Provider	-



## 注記

詳細については、『Red Hat JBoss Data Grid Developer Guide』の項「JGroups ENCRYPT」を参照してください。

## SASLの推奨値

詳細については、『Red Hat JBoss Data Grid Developer Guide』の項「User Authentication over Hot Rod Using SASL」を参照してください。

## RELAY2の推奨値

詳細については、[29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。

表A.16 UDP の推奨値

パラメーター	デフォルト値 (JGroups)	推奨値 (JBoss Data Grid)
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
port_range	50	必要なポート範囲に基いて設定
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

パラメーター	デフォルト値 (JGroups)	推奨値 (JBoss Data Grid)
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	Abort
oob_thread_pool.enabled	true	デフォルト値と同じ
oob_thread_pool.min_threads	2	20 以上

パラメーター	デフォルト値 (JGroups)	推奨値 (JBoss Data Grid)
<code>oob_thread_pool.max_threads</code>	10	200 以上 (負荷に基づく)
<code>oob_thread_pool.keep_alive_time</code>	30,000	60,000
<code>oob_thread_pool.queue_enabled</code>	true	false
<code>oob_thread_pool.queue_max_size</code>	500	なし。キューを無効にする必要がある
<code>oob_thread_pool.rejection_policy</code>	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` 値は代わりにタイムアウト期間を示します。

表A.17 MERGE3 の推奨値

パラメーター	デフォルト値 (JGroups)	推奨値 (JBoss Data Grid)
<code>min_interval</code>	1,000	10,000
<code>max_interval</code>	10,000	30,000

表A.18 FD SOCK の推奨値

パラメーター	デフォルト値 (JGroups)	推奨値 (JBoss Data Grid)
<code>client_bind_port</code>	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
<code>get_cache_timeout</code>	1000 ミリ秒	デフォルト値と同じ
<code>keep_alive</code>	true	デフォルト値と同じ
<code>num_tries</code>	3	デフォルト値と同じ
<code>start_port</code>	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
<code>suspect_msg_interval</code>	5000 ミリ秒。	デフォルト値と同じ

表A.19 FD\_ALL の推奨値



パラメーター	デフォルト値 (JGroups)	推奨値 (JBoss Data Grid)
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 <b>stop the world</b> 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1GB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 <b>false</b> 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 <b>interval</b> 値は、FD_ALL の <b>timeout</b> 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.20 FD\_HOST の推奨値

パラメーター	デフォルト値 (JGroups)	推奨値 (JBoss Data Grid)
check_timeout	3,000	5,000
cmd	InetAddress.isReachable() (ICMP ping)	-
間隔	20,000	15,000。FD_HOST の <b>interval</b> 値は、FD_HOST の <b>timeout</b> 値の4分1である必要があります。
timeout	-	60,000

表A.21 VERIFY\_SUSPECT の推奨値

パラメーター	デフォルト値 (JGroups)	推奨値 (JBoss Data Grid)
timeout	2,000	5,000

表A.22 pbcast.NAKACK2の推奨値

パラメーター	デフォルト値 (JGroups)	推奨値 (JBoss Data Grid)
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.23 UNICAST3の推奨値

パラメーター	デフォルト値 (JGroups)	推奨値 (JBoss Data Grid)
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.24 pbcast.STABLEの推奨値

パラメーター	デフォルト値 (JGroups)	推奨値 (JBoss Data Grid)
stability_delay	6,000	500
desired_avg_gossip	20,000	5,000
max_bytes	2,000,000	1,000,000

表A.25 pbcaster.GMS の推奨値

パラメーター	デフォルト値 (JGroups)	推奨値 (JBoss Data Grid)
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.26 UFC の推奨値

パラメーター	デフォルト値 (JGroups)	推奨値 (JBoss Data Grid)
max_credits	500,000	2,000,000
min_threshold	0.40	デフォルト値と同じ

表A.27 MFC の推奨値

パラメーター	デフォルト値 (JGroups)	推奨値 (JBoss Data Grid)
max_credits	500,000	2,000,000
min_threshold	0.40	デフォルト値と同じ

表A.28 FRAG2 の推奨値

パラメーター	デフォルト値 (JGroups)	推奨値 (JBoss Data Grid)
frag_size	60,000	デフォルト値と同じ

表A.29 ENCRYPT の推奨値

パラメーター	デフォルト値 (JGroups)	推奨値 (JBoss Data Grid)
asymAlgorithm	RSA	-
asymInit	512	-
asymProvider	Bouncy Castle Provider	-
changeKeysOnViewChange	false	-
symAlgorithm	AES	-

パラメーター	デフォルト値 (JGroups)	推奨値 (JBoss Data Grid)
symInnit	128	-
symProvider	Bouncy Castle Provider	-



### 注記

詳細については、『Red Hat JBoss Data Grid Developer Guide』の項「JGroups ENCRYPT」を参照してください。

### SASLの推奨値

詳細については、『Red Hat JBoss Data Grid Developer Guide』の項「User Authentication over Hot Rod Using SASL」を参照してください。

### RELAY2の推奨値

詳細については、[29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](#)

## 付録B RED HAT JBOSS DATA GRID における JMX MBEANS

### B.1. アクティベーション

`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B.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B.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。	<code>void resetStatistics()</code>

[バグを報告する](#)

### B.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B.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B.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](#)

## B.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B.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内のすべての削除操作に対するキャッシュコンテナ合計平均時間 (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](#)

## B.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B.6 属性

名前	説明	タイプ	書き込み可能
<code>cacheLoaderLoads</code>	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。	<code>long</code>	いいえ
<code>cacheLoaderMisses</code>	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。	<code>long</code>	いいえ
<code>stores</code>	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
<code>statisticsEnabled</code>	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B.7 操作

名前	説明	署名
<code>disableStore</code>	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。	<code>void disableStore(String storeType)</code>
<code>resetStatistics</code>	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。	<code>void resetStatistics()</code>

[バグを報告する](#)

## B.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、およびコンテナとして動作します。

表B.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
名前	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です。	プロパティ	いいえ

表B.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](#)

## B.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B.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B.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](#)

## B.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B.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B.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](#)

## B.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B.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B.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](#)

## B.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B.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](#)

## B.10. インタープリター

`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B.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のキャッシュのリストを取得します。	String[]	いいえ

表B.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	IsbnCliQL 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](#)

## B.11. インバリデーション

### 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B.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B.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](#)

## B.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B.21 属性

名前	説明	タイプ	書き込み可能
<code>concurrencyLevel</code>	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
<code>numberOfLocksAvailable</code>	利用可能な排他ロックの数。	整数	いいえ
<code>numberOfLocksHeld</code>	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](#)

## B.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B.22 属性

名前	説明	タイプ	書き込み可能
----	----	-----	--------



名前	説明	タイプ	書き込み可能
rebalancingEnabled	<b>false</b> 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 <b>numOwners</b> 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	<b>AVAILABLE</b> の場合は、ノードが現在正常に動作しています。 <b>DEGRADED</b> 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](#)

## B.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B.23 操作

名前	説明	署名
start	インデックスの再構築を開始します。	<code>void start()</code>



### 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 『Infinispan Query Guide』を参照してください。

[バグを報告する](#)

## B.15. パッシベーション

`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B.24 属性

名前	説明	タイプ	書き込み可能
<code>passivations</code>	パッシベーションイベントの数です。	文字列	いいえ
<code>statisticsEnabled</code>	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B.25 操作

名前	説明	署名
<code>resetStatistics</code>	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。	<code>void resetStatistics()</code>

[バグを報告する](#)

## B.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B.26 操作

名前	説明	署名
<code>forceCommit</code>	不明なトランザクションのコミットを強制します。	<code>String forceCommit(long p0)</code>
<code>forceCommit</code>	不明なトランザクションのコミットを強制します。	<code>String forceCommit(int p0, byte[] p1, byte[] p2)</code>
<code>forceRollback</code>	不明なトランザクションのロールバックを強制します。	<code>String forceRollback(long p0)</code>

名前	説明	署名
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)
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](#)

## B.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B.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](#)

## B.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B.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B.29 操作

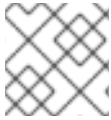
名前	説明	署名
resetStatistics	このコンポーネントによって収集される統計をリセットします。	void resetStatistics()
setStatisticsEnabled	統計を有効または無効にするかを設定します (true/false)。	void setStatisticsEnabled(boolean enabled)

[バグを報告する](#)

## B.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B.30 属性

名前	説明	タイプ	書き込み可能
<code>joinComplete</code>	<code>true</code> 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。 <code>false</code> の場合、 <code>join</code> プロセスは依然として進行中です。	ブール値	いいえ
<code>stateTransferInProgress</code>	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](#)

## B.20. 統計

`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B.31 属性

名前	説明	タイプ	書き込み可能
<code>averageReadTime</code>	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。	<code>long</code>	いいえ
<code>averageWriteTime</code>	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。	<code>long</code>	いいえ
<code>elapsedTime</code>	キャッシュの開始以降の秒数です。	<code>long</code>	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数です。	long	いいえ
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B.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](#)

## B.21. トランザクション

`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B.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B.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](#)

## B.22. トランスポート

`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B.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

名前	説明	タイプ	書き込み可能
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
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	受信バッファサイズを返します。	文字列	いいえ
sendBufferSize	送信バッファサイズを返します。	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](#)



## B.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B.36 操作

名前	説明	署名
<code>bringSiteOnline</code>	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。	<code>String bringSiteOnline(String p0)</code>
<code>amendTakeOffline</code>	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。	<code>String amendTakeOffline(String p0, int p1, long p2)</code>
<code>getTakeOfflineAfterFailures</code>	'TakeOffline' 機能に対して 'afterFailures' の値を返します。	<code>String getTakeOfflineAfterFailures(String p0)</code>
<code>getTakeOfflineMinTimeToWait</code>	'TakeOffline' 機能に対して 'minTimeToWait' の値を返します。	<code>String getTakeOfflineMinTimeToWait(String p0)</code>
<code>setTakeOfflineAfterFailures</code>	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。	<code>String setTakeOfflineAfterFailures(String p0, int p1)</code>
<code>setTakeOfflineMinTimeToWait</code>	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。	<code>String setTakeOfflineMinTimeToWait(String p0, long p1)</code>
<code>siteStatus</code>	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。	<code>String siteStatus(String p0)</code>
<code>status</code>	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。	<code>String status()</code>
<code>takeSiteOffline</code>	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。	<code>String takeSiteOffline(String p0)</code>
<code>pushState</code>	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。	<code>String pushState(String p0)</code>

名前	説明	署名
<code>cancelPushState</code>	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。	<code>String cancelPushState(String p0)</code>
<code>getSendingSiteName</code>	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。	<code>String getSendingSiteName()</code>
<code>cancelReceiveState</code>	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。	<code>String cancelReceiveState(String p0)</code>
<code>getPushStateStatus</code>	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。	<code>String getPushStateStatus()</code>
<code>clearPushStateStatus</code>	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。	<code>String clearPushStateStatus()</code>

[バグを報告する](#)

## 付録C 推奨される設定

### C.1. TIMEOUT VALUES

表C.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 <b><i>sync.replTimeout</i></b> 値と同じ値に設定されます。

## 付録D パフォーマンスに関する推奨事項

### D.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E 参考資料

### E.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](#)

### E.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せんが、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](#)

### E.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 の前身となるものです。

[バグを報告する](#)

### E.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](#)

## E.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](#)

## E.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](#)

## E.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](#)

## E.8. キーバリュールペアについて

キーバリュールペア (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。
- バリュール (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](#)

## E.9. エクスターナライザー

### E.9.1. エクスターナライザーについて

**Externalizer** クラスは、以下のことを行えるクラスです。

- 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をオブジェクトタイプのインスタンスにマーシャリング解除します。

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。

[バグを報告する](#)

### E.9.2. 内部エクスターナライザー実装アクセス

Externalizable オブジェクトは Red Hat JBoss Data Grids エクスターナライザー実装にアクセスしないようにする必要があります。間違った使用法の例を以下に示します。

```
public static class ABCMarshallingExternalizer implements
AdvancedExternalizer<ABCMarshalling> {
    @Override
    public void writeObject(ObjectOutput output, ABCMarshalling object)
throws IOException {
        MapExternalizer ma = new MapExternalizer();
        ma.writeObject(output, object.getMap());
    }

    @Override
    public ABCMarshalling readObject(ObjectInput input) throws IOException,
ClassNotFoundException {
        ABCMarshalling hi = new ABCMarshalling();
        MapExternalizer ma = new MapExternalizer();
        hi.setMap((ConcurrentHashMap<Long, Long>) ma.readObject(input));
        return hi;
    }
    <!-- Additional configuration information here -->
}
```

エンドユーザーエクスターナライザーは内部のエクスターナライザークラスと対話する必要がありません。正しい使用法の例を以下に示します。

```
public static class ABCMarshallingExternalizer implements
AdvancedExternalizer<ABCMarshalling> {
    @Override
    public void writeObject(ObjectOutput output, ABCMarshalling object)
throws IOException {
        output.writeObject(object.getMap());
    }

    @Override
    public ABCMarshalling readObject(ObjectInput input) throws IOException,
ClassNotFoundException {
        ABCMarshalling hi = new ABCMarshalling();
        hi.setMap((ConcurrentHashMap<Long, Long>) input.readObject());
        return hi;
    }
    <!-- Additional configuration information here -->
}
```

[バグを報告する](#)

## E.10. ハッシュ領域の割り当て

### E.10.1. ハッシュ領域の割り当てについて

Red Hat JBoss Data Grid は、使用可能なハッシュ領域全体の一部を各ノードに割り当てる役割があります。エントリーを格納する必要がある後続の操作中、JBoss Data Grid は関連するキーのハッシュを作成し、その部分のハッシュ領域を所有するノード上にエントリーを格納します。

[バグを報告する](#)

### E.10.2. ハッシュ領域におけるキーの検索

Red Hat JBoss Data Grid は常にアルゴリズムを使用してハッシュ領域のキーを見つけます。そのため、キーを格納するノードを手動で指定することはありません。このスキームにより、キーの所有者情報を配信しなくても、すべてのノードが特定のキーを所有するノードを判断することができます。このスキームによりオーバーヘッドの量が削減されます。さらに重要なことに、ノードの障害時に所有者情報をレプリケートする必要がないため、冗長性が向上します。

[バグを報告する](#)

### E.10.3. 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を渡します。

#### 例E.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F 改訂履歴

<b>改訂 6.5.0-7.1</b> バージョン 6.1 を日本語に翻訳	<b>Mon Aug 17 2015</b>	<b>Takuro Nagamoto</b>
<b>改訂 6.5.0-7</b> BZ-1188924: 大量のキャッシュの同時起動に関する情報が追加されました。 BZ-1188922: ClusterCacheStats と CacheContainerStats の JMX メトリックスが追加されました。 BZ-1188906: Hot Rod C++ クライアントの項が更新されました。 BZ-1227899: EAP からの HTTP セッションの外部化に関する項が追加されました。 BZ-1228226: カスタムキャッシュストア向けの maven aアーキタイプを生成する手順が追加されました。	<b>Wed Jun 3 2015</b>	<b>Christian Huffman</b>
<b>改訂 6.5.0-6</b> BZ-1203175: サイト間の状態に関するトピックが更新されました。	<b>Wed May 27 2015</b>	<b>Rakesh Ghatvisave</b>
<b>改訂 6.5.0-5</b> BZ-1188906: Hot Rod C++ クライアントのサポートプラットフォームが更新されました。	<b>Mon May 18 2015</b>	<b>Rakesh Ghatvisave</b>
<b>改訂 6.5.0-4</b> BZ-1200763: LocalTopologyManager MBean 内の clusterAvailability が追加されました。	<b>Fri May 08 2015</b>	<b>Christian Huffman</b>
<b>改訂 6.5.0-3</b> カスタムキャッシュストアの追加に関する記述の構成が変更されました。	<b>Wed 06 May 2015</b>	<b>Christian Huffman</b>
<b>改訂 6.5.0-2</b> BZ-1188928: JON を使用したキャッシュの作成に関するトピックが追加されました。 BZ-1188921: C# 文字列マーシャラーに関するトピックが追加されました。	<b>Mon 04 May 2015</b>	<b>Rakesh Ghatvisave</b>
<b>改訂 6.5.0-1</b> BZ-1188905: ニアキャッシングに関するトピックが追加されました。	<b>Mon 27 Apr 2015</b>	<b>Rakesh Ghatvisave</b>
<b>改訂 6.5.0-0</b> BZ-1188921: C# 技術プレビューモードに関する警告の記述が削除されました。	<b>Mon 13 Apr 2015</b>	<b>Rakesh Ghatvisave</b>