Red Hat Data Grid 8.0

Data Grid REST API

Data Grid のドキュメント

Last Updated: 2023-03-12

Red Hat Data Grid 8.0 Data Grid REST API

Data Grid のドキュメント

Enter your first name here. Enter your surname here.
Enter your organisation's name here. Enter your organisational division here.
Enter your email address here.

法律上の通知

Copyright © 2023 | You need to change the HOLDER entity in the en-US/Data_Grid_REST_API.ent file |.

The text of and illustrations in this document are licensed by Red Hat under a Creative Commons Attribution–Share Alike 3.0 Unported license ("CC-BY-SA"). An explanation of CC-BY-SA is available at
http://creativecommons.org/licenses/by-sa/3.0/
. In accordance with CC-BY-SA, if you distribute this document or an adaptation of it, you must provide the URL for the original version.

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, the Red Hat 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 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

データへのアクセス、クラスターの監視と保守、Data Grid RESTAPI を介した管理操作の実行。

目次

第1章 RED HAT DATA GRID 4
1.1. DATA GRID のドキュメント 4
1.2. DATA GRID のダウンロード 4

第2章 DATA GRID REST API との連携 5
2.1. キャッシュの作成と管理 5
2.1.1. キャッシュの作成 5
2.1.1.1. XML の設定 5
2.1.1.2. JSON 設定 6
2.1.2. キャッシュの検証 6
2.1.3. テンプレートを使用したキャッシュの作成 6
2.1.4. キャッシュ設定の取得 6
2.1.5. キャッシュ設定を JSON に変換する 7
2.1.6. すべてのキャッシュの詳細を取得する 7
2.1.7. エントリーの追加 8
2.1.8. エントリーの置き換え 10
2.1.9. キーによるデータの取得 10
2.1.10. エントリーが存在するかどうかの確認 11
2.1.11. エントリーの削除 12
2.1.12. キャッシュの削除 12
2.1.13. キャッシュからのすべてのキーの取得 12
2.1.14. キャッシュのクリア 12
2.1.15. キャッシュサイズの取得 13
2.1.16. キャッシュ統計の取得 13
2.1.17. キャッシュのクエリー 13
2.1.18. データのインデックスの再作成 14
2.1.19. インデックスのパージ 15
2.1.20. インデックス統計情報の取得 15
2.1.21. クエリー統計情報の取得 15
2.1.22. クエリー統計のクリア 16
2.1.23. キャッシュの一覧表示 16
2.1.24. キャッシュを利用したクロスサイト・オペレーション 16
2.1.24.1. すべてのバックアップロケーションのステータス取得 16
2.1.24.2. 特定のバックアップ場所のステータスの取得 17
2.1.24.3. バックアップ先をオフラインにする 17
2.1.24.4. バックアップ場所をオンラインにする 18
2.1.24.5. バックアップ場所への状態のプッシュ 18
2.1.24.6. 状態転送のキャンセル 18
2.1.24.7. 状態転送ステータスの取得 18
2.1.24.8. 状態転送ステータスのクリア 19
2.1.24.9. オフラインテイク条件の変更 19
2.1.24.10. 受信サイトからの状態転送のキャンセル 19
2.1.25. ローリングアップグレード 19
2.1.25.1. データの同期 19
2.1.25.2. ソースクラスターの接続 20
2.2. カウンターの作成と管理 20
2.2.1. カウンターの作成 20
2.2.2. カウンターの削除 20
2.2.3. カウンター設定の取得 20
2.2.4. カウンターへの値の追加 21
2.2.5. カウンター値の取得 21

2.2.6. カウンターのリセット 21
2.2.7. カウンターのインクリメント 21
2.2.8. カウンターへのデルタの追加 22
2.2.9. カウンター値のデクリメント 22
2.2.10. ストロングカウンターでの compareAndSet オペレーションの実行 22
2.2.11. 強力なカウンターでの compareAndSwap 操作の実行 22
2.2.12. カウンターの一覧表示 23
2.3. キャッシュマネージャーの操作 23
2.3.1. 基本的なキャッシュマネージャー情報の取得 23
2.3.2. クラスターヘルスの取得 24
2.3.3. キャッシュマネージャーのヘルスステータスの取得 26
2.3.4. REST エンドポイントの可用性の確認 26
2.3.5. キャッシュマネージャーのグローバル設定の取得 26
2.3.6. すべてのキャッシュの設定を取得する 26
2.3.7. 利用可能なキャッシュテンプレートの一覧表示 27
2.3.8. (実験的) キャッシュのステータスと情報の取得 27
2.3.9. キャッシュマネージャー統計の取得 28
2.3.10. キャッシュ・マネージャーによるクロスサイト操作 29
2.3.10.1. バックアップ場所のステータスの取得 30
2.3.10.2. バックアップ先をオフラインにする 30
2.3.10.3. バックアップ場所をオンラインにする 30
2.3.10.4. 状態転送の開始 31
2.3.10.5. 状態転送のキャンセル 31
2.4. DATA GRID サーバーの操作 31
2.4.1. 基本的なサーバー情報の取得 31
2.4.2. キャッシュマネージャーの取得 31
2.4.3. 無視リストへのキャッシュの追加 31
2.4.4. 無視リストからのキャッシュの削除 32
2.4.5. 無視されたキャッシュの確認 32
2.4.6. サーバー設定の取得 32
2.4.7. 環境変数の取得 33
2.4.8. JVM メモリーの詳細の取得 33
2.4.9. JVM スレッドダンプの取得 33
2.4.10. Data Grid サーバーの停止 33
2.5. DATA GRID クラスターの操作 33
2.5.1. Data Grid クラスターの停止 34
2.5.2. クラスター内の特定の Data Grid サーバーの停止 34
2.6. サーバータスクの使用 34
2.6.1. サーバータスク情報の取得 34
2.6.2. タスクの実行 35
2.6.3. スクリプトタスクのアップロード 35

第3章 DATA GRID REST エンドポイント 36
3.1. REST 認証 36
3.2. サポートされているプロトコル 36
3.3. データ形式と RESTAPI 36
3.3.1. サポート対象の形式 36
3.3.2. ヘッダーを受け入れる 37
3.3.3. 特殊文字を含む名前 37
3.3.4. Key-Content-Type ヘッダー 37
3.3.5. JSON /プロトストリーム変換 38
3.4. クロスオリジンリソースシェアリング (CORS) リクエスト 39

第1章 RED HAT DATA GRID

Data Grid は、高性能の分散型インメモリーデータストアです。

スキーマレスデータ構造
さまざまなオブジェクトをキーと値のペアとして格納する柔軟性があります。

グリッドベースのデータストレージ
クラスター間でデータを分散および複製するように設計されています。

エラスティックスケーリング
サービスを中断することなく、ノードの数を動的に調整して要件を満たします。

データの相互運用性
さまざまなエンドポイントからグリッド内のデータを保存、取得、およびクエリーします。

1.1. DATA GRID のドキュメント

Data Grid のドキュメントは、Red Hat カスタマーポータルで入手できます。

- Data Grid 8.0 ドキュメント
- Data Grid 8.0 コンポーネントの詳細
- Data Grid 8.0 でサポートされる設定

1.2. DATA GRID のダウンロード

Red Hat カスタマーポータルで Data Grid Software Downloads にアクセスします。

注記

Data Grid ソフトウェアにアクセスしてダウンロードするには、Red Hat アカウントが必要です。

第2章 DATA GRID REST API との連携

Data Grid REST API は、Data Grid の展開を監視、維持、管理し、データへのアクセスを提供します。

2.1. キャッシュの作成と管理

Data Grid のキャッシュを作成・管理し、データに対する操作を行うことができます。

2.1.1. キャッシュの作成

XML または JSON 設定をペイロードに含む **POST** リクエストで、Data Grid クラスター全体に名前付きキャッシュを作成します。

```
POST /rest/v2/caches/{cacheName}
```

表2.1 ヘッダー

ヘッダー	必須またはオプション	パラメーター
Content-Type	必須	Data Grid 設定のペイロードの MediaType を設定します (**application/xml** または **application/json** のいずれか)。
Flags	オプション	AdminFlags を設定するために使用されます

References

- Data Grid XML 設定
- Data Grid JSON 設定

2.1.1.1. XML の設定

XML 形式の Data Grid 設定はスキーマに準拠し、以下を含める必要があります。

- **<infinispan>** ルート要素。
- **<cache-container>** 定義。

XML 設定のサンプル

```
<infinispan>
   <cache-container>
      <distributed-cache name="cacheName" mode="SYNC">
         <memory>
            <object size="20"/>
         </memory>
      </distributed-cache>
   </cache-container>
</infinispan>
```

2.1.1.2. JSON 設定

JSON 形式の Data Grid 設定:

- キャッシュ定義のみが必要です。
- XML 設定の構造に従う必要があります。
  - XML 要素は JSON オブジェクトになります。
  - XML 属性は JSON フィールドになります。

JSON 設定の例

```
{
  "distributed-cache": {
    "mode": "SYNC",
    "memory": {
      "object": {
        "size": 20
      }
    }
  }
}
```

2.1.2. キャッシュの検証

HEAD リクエストで Data Grid クラスターでキャッシュが利用可能かどうかを確認します。

```
HEAD /rest/v2/caches/{cacheName}
```

2.1.3. テンプレートを使用したキャッシュの作成

POST リクエストと **?template=** パラメーターを使用して、Data Grid テンプレートからキャッシュを作成します。

```
POST /rest/v2/caches/{cacheName}?template={templateName}
```

ヒント

使用可能なキャッシュテンプレートを一覧表示する を参照してください。

2.1.4. キャッシュ設定の取得

GET リクエストで Data Grid のキャッシュ設定を取得します。

```
GET /rest/v2/caches/{name}?action=config
```

表2.2 ヘッダー

ヘッダー	必須またはオプション	パラメーター
Accept	オプション	コンテンツを返すために必要な形式を設定します。対応フォーマットは、**application/xml** と **application/json** です。デフォルトは **application/json** です。詳細については、 Accept を参照してください。

2.1.5. キャッシュ設定を JSON に変換する

有効な XML 設定と **?action=toJSON** パラメーターを使用して **POST** 要求を呼び出します。Data Grid は、設定の同等の JSON 表現で応答します。

```
POST /rest/v2/caches?action=toJSON
```

2.1.6. すべてのキャッシュの詳細を取得する

GET リクエストを呼び出し、Data Grid キャッシュのすべての詳細を取得します。

```
GET /rest/v2/caches/{name}
```

Data Grid は、以下のような JSON レスポンスを提供します。

```
{
  "stats": {
    "time_since_start": -1,
    "time_since_reset": -1,
    "hits": -1,
    "current_number_of_entries": -1,
    "current_number_of_entries_in_memory": -1,
    "total_number_of_entries": -1,
    "stores": -1,
    "off_heap_memory_used": -1,
    "data_memory_used": -1,
    "retrievals": -1,
    "misses": -1,
    "remove_hits": -1,
    "remove_misses": -1,
    "evictions": -1,
    "average_read_time": -1,
    "average_read_time_nanos": -1,
    "average_write_time": -1,
    "average_write_time_nanos": -1,
    "average_remove_time": -1,
    "average_remove_time_nanos": -1,
    "required_minimum_number_of_nodes": -1
  },
  "size": 0,
  "configuration": {
```

```
    "distributed-cache": {
      "mode": "SYNC",
      "transaction": {
        "stop-timeout": 0,
        "mode": "NONE"
      }
    }
  },
  "rehash_in_progress": false,
  "bounded": false,
  "indexed": false,
  "persistent": false,
  "transactional": false,
  "secured": false,
  "has_remote_backup": false,
  "indexing_in_progress": false,
  "statistics": false
}
```

- **stats** キャッシュの現在の状態を表示します。
- **size** キャッシュの推定サイズ。
- **configuration** キャッシュ設定。
- **rehash_in_progress** リハッシュが進行中の場合は true。
- **indexing_in_progress** インデックス作成中の場合は true。
- **bounded** 有効期限が有効になっている。
- **indexed** キャッシュがインデックス化されている場合は true。
- **persistent** キャッシュが永続化されている場合は true。
- **transactional** キャッシュがトランザクショナルである場合は true。
- **secured** キャッシュが保護されている場合は true。
- **has_remote_backup** キャッシュがリモートバックアップを持っている場合は true。

2.1.7. エントリーの追加

POST リクエストでキャッシュにエントリーを追加します。

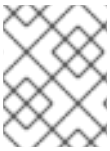
```
POST /rest/v2/caches/{cacheName}/{cacheKey}
```

前述の要求は **cacheKey** キーで **cacheName** を指定のキャッシュに、ペイロード、またはリクエストボディを配置します。リクエストは、既存のデータを置き換え、適用可能であれば、**Time-To-Live** と **Last-Modified** の値を更新します。

指定されたキーの値がすでに存在する場合、**POST** リクエストは HTTP **CONFLICT** ステータスを返し、値を変更しません。値を更新するには、**PUT** リクエストを使用する必要があります。エントリーの入れ替え をご覧ください。

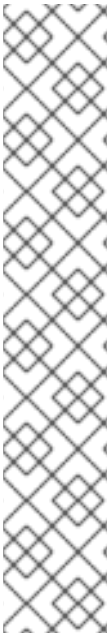
表2.3 ヘッダー

ヘッダー	必須またはオプション	パラメーター
Key-Content-Type	オプション	リクエストのキーのコンテンツタイプを設定します。詳細は、Key-Content-Type を参照してください。
Content-Type	オプション	キーの値の MediaType を設定します。
timeToLiveSeconds	オプション	エントリーが自動的に削除されるまでの秒数を設定します。このパラメーターを設定しない場合、DataGrid は設定のデフォルト値を使用します。負の値を設定すると、エントリーが削除されることはありません。
maxIdleTimeSeconds	オプション	エントリーがアイドル状態になることができる秒数を設定します。最大アイドル時間が経過してもエントリーの読み取りまたは書き込み操作が発生しない場合、エントリーは自動的に削除されます。このパラメーターを設定しない場合、DataGrid は設定のデフォルト値を使用します。負の値を設定すると、エントリーが削除されることはありません。
flags	オプション	エントリーの追加に使用されるフラグ。詳細については、フラグ を参照してください。



注記

flags ヘッダーは、キャッシュでのデータ操作を含む他のすべての操作にも適用されます。



注記

timeToLiveSeconds と **maxIdleTimeSeconds** の両方の値が **0** の場合、Data Grid は設定のデフォルトの **lifespan** と **maxIdle** の値を使用します。

only **maxIdleTimeSeconds** のみ値が **0** の場合、Data Grid が使用します。

- コンフィグレーションのデフォルトの **maxIdle** 値を使用します。
- リクエストパラメーターとして渡した **timeToLiveSeconds** の値、または値を渡さなかった場合は **-1** の値です。

timeToLiveSeconds の値**だけが 0** の場合、Data Grid は

- 設定のデフォルトの **lifespan** 値を使用。
- リクエストパラメーターとして渡された **maxIdle** の値、または値を渡さない場合は **-1** の値を使用。

2.1.8. エントリーの置き換え

キャッシュ内のエントリーを **PUT** リクエストに置き換えます。

```
PUT /rest/v2/caches/{cacheName}/{cacheKey}
```

指定されたキーの値がすでに存在する場合、 **PUT** 要求は値を更新します。既存の値を変更したくない場合は、値を変更する代わりに HTTP **CONFLICT** ステータスを返す **POST** リクエストを使用します。値の追加 を参照してください。

2.1.9. キーによるデータの取得

GET リクエストを使用して特定のキーのデータを取得します。

```
GET /rest/v2/caches/{cacheName}/{cacheKey}
```

サーバーは、応答本文の指定されたキー **cacheKey** の下にある指定されたキャッシュ **cacheName** からデータを返します。応答には、**MediaType** ネゴシエーションに対応する **Content-Type** ヘッダーが含まれています。

注記

ブラウザーは、たとえばコンテンツ配信ネットワーク (CDN) として、キャッシュに直接アクセスすることもできます。Data Grid は、各エントリーに固有のETagを、**Last-Modified** および **Expires** ヘッダーフィールドとともに返します。

これらのフィールドは、リクエストで返されるデータの状態に関する情報を提供します。ETag を使用すると、ブラウザーやその他のクライアントは、変更されたデータのみを要求できるため、帯域幅が節約されます。

表2.4 ヘッダー

ヘッダー	必須またはオプション	パラメーター
Key-Content-Type	オプション	リクエストのキーのコンテンツタイプを設定します。デフォルトは **application/x-java-object; type=java.lang.String** です。詳細は、Key-Content-Type を参照してください。
Accept	オプション	コンテンツを返すために必要な形式を設定します。詳細については、 Accept を参照してください。

ヒント

extended パラメーターをクエリー文字列に追加して、追加情報を取得します。

```
GET /rest/v2/caches/{cacheName}/{cacheKey}?extended
```

上記のリクエストはカスタムヘッダーを返します:

- **Cluster-Primary-Owner** は、キーのプライマリー所有者であるノード名を返します。
- **Cluster-Node-Name** は、要求を処理したサーバーの JGroups ノード名を返します。
- **Cluster-Physical-Address** は、要求を処理したサーバーの物理 JGroups アドレスを返します。

2.1.10. エントリーが存在するかどうかの確認

HEAD リクエストで特定のエントリーが存在することを確認します。

```
HEAD /rest/v2/caches/{cacheName}/{cacheKey}
```

上記のリクエストは、ヘッダーフィールドと、エントリーとともに保存したものと同じコンテンツのみを返します。たとえば、文字列を保存した場合、リクエストは文字列を返します。バイナリー、base64 エンコード、blob、またはシリアル化された Java オブジェクトを保存した場合、DataGrid はリクエストのコンテンツを逆シリアル化しません。

注記

HEAD リクエストは、**extended** パラメーターもサポートしています。

表2.5 ヘッダー

ヘッダー	必須またはオプション	パラメーター

ヘッダー	必須またはオプション	パラメーター
Key-Content-Type	オプション	リクエストのキーのコンテンツタイプを設定します。デフォルトは **application/x-java-object; type=java.lang.String** です。詳細は、Key-Content-Type を参照してください。

2.1.11. エントリーの削除

DELETE リクエストを使用してキャッシュからエントリーを **削除** します。

```
DELETE /rest/v2/caches/{cacheName}/{cacheKey}
```

表2.6 ヘッダー

ヘッダー	必須またはオプション	パラメーター
Key-Content-Type	オプション	リクエストのキーのコンテンツタイプを設定します。デフォルトは **application/x-java-object; type=java.lang.String** です。詳細は、Key-Content-Type を参照してください。

2.1.12. キャッシュの削除

DELETE リクエストを使用して Data Grid クラスターからキャッシュを **DELETE** 削除します。

```
DELETE /rest/v2/caches/{cacheName}
```

2.1.13. キャッシュからのすべてのキーの取得

GET リクエストを呼び出して、キャッシュ内のすべてのキーを JSON 形式で取得します。

```
GET /rest/v2/caches/{cacheName}?action=keys
```

表2.7 リクエストパラメーター

パラメーター	必須またはオプション	値
batch-size	オプション	キーを取得するときの内部バッチサイズを指定します。デフォルト値は **1000** です。

2.1.14. キャッシュのクリア

キャッシュからすべてのデータを削除するには、**GET** リクエストに **?action=clear** パラメーターを付けて実行します。

```
GET /rest/v2/caches/{cacheName}?action=clear
```

2.1.15. キャッシュサイズの取得

GET リクエストと **?action=size** パラメーターを使用して、クラスター全体のキャッシュのサイズを取得します。

```
GET /rest/v2/caches/{cacheName}?action=size
```

2.1.16. キャッシュ統計の取得

GET リクエストを使用してキャッシュの実行時統計を取得します。

```
GET /rest/v2/caches/{cacheName}?action=stats
```

2.1.17. キャッシュのクエリー

GET リクエストと **?action=search&query** パラメーターを使用して、キャッシュに対して Ickle クエリーを実行します。

```
GET /rest/v2/caches/{cacheName}?action=search&query={ickle query}
```

Data Grid は、次のようなクエリーヒットで応答します。

```
{
  "total_results" : 150,
  "hits" : [ {
    "hit" : {
      "name" : "user1",
      "age" : 35
    }
  }, {
    "hit" : {
      "name" : "user2",
      "age" : 42
    }
  }, {
    "hit" : {
      "name" : "user3",
      "age" : 12
    }
  } ]
}
```

- **total_results** は、クエリーの結果の合計数を表示します。
- **hits** は、クエリーからのマッチの配列です。
- **hit** は、クエリーにマッチするオブジェクトです。

ヒント

ヒットにはすべてのフィールドを含めることができますが、**Select** 句を使用するとフィールドのサブセットを含めることができます。

表2.8 リクエストパラメーター

パラメーター	必須またはオプション	値
query	必須	クエリー文字列を指定します。
max_results	オプション	返す結果の数を設定します。デフォルトは **10** です。
offset	オプション	返される最初の結果のインデックスを指定します。デフォルトは **0** です。
query_mode	オプション	Data Grid サーバーがクエリーを実行する方法を指定します。値は **FETCH** および **BROADCAST** です。デフォルトは **FETCH** です。

クエリーパラメーターを指定せずにリクエストの本文を使用するには、以下のように **POST** リクエストを呼び出します。

```
POST /rest/v2/caches/{cacheName}?action=search
```

次の例は、リクエスト本文のクエリーを示しています。

```
{
 "query":"from Entity where name:\"user1\"",
 "max_results":20,
 "offset":10
}
```

2.1.18. データのインデックスの再作成

GET リクエストと **?action=mass-index&mode={mode}** パラメーターを使用して、キャッシュ内のすべてのデータのインデックスを再作成します。

```
GET  /v2/caches/{cacheName}/search/indexes?action=mass-index&mode={mode}
```

mode パラメーターの値は以下の通りです。

- **sync** は、インデックス変更操作が完了した後にのみ **200** の応答を返します。
- **async** は **200** の応答を即座に返し、再インデックス化操作はクラスター内で実行を継続します。Index Statistics REST コールで状態を確認できます。

2.1.19. インデックスのパージ

GET リクエストと **?action=clear** パラメーターを使用して、キャッシュからすべてのインデックスを削除します。

```
GET  /v2/caches/{cacheName}/search/indexes?action=clear
```

2.1.20. インデックス統計情報の取得

GET リクエストでキャッシュ内のインデックスの情報を取得します。

```
GET /v2/caches/{cacheName}/search/indexes/stats
```

Data Grid は、以下のような JSON レスポンスを提供します。

```
{
   "indexed_class_names": ["org.infinispan.sample.User"],
   "indexed_entities_count": {
      "org.infinispan.sample.User": 4
   },
   "index_sizes": {
      "cacheName_protobuf": 14551
   },
   "reindexing": false
}
```

- **indexed_class_names** キャッシュに存在するインデックスのクラス名を提供します。Protobuf の場合、値は常に **org.infinispan.query.remote.impl.indexing.ProtobufValueWrapper** です。
- **indexed_entities_count** クラスごとにインデックスされているエンティティーの数を提供します。
- **index_sizes** キャッシュ内の各インデックスのサイズをバイト単位で指定します。
- **reindexing** キャッシュに対してインデックス変更操作が行われたかどうかを示す。この値が **true** の場合、**MassIndexer** はキャッシュで起動したことになります。

2.1.21. クエリー統計情報の取得

GET リクエストでキャッシュに実行されたクエリーの情報を取得します。

```
GET /v2/caches/{cacheName}/search/query/stats
```

Data Grid は、以下のような JSON レスポンスを提供します。

```
{
   "search_query_execution_count":20,
   "search_query_total_time":5,
   "search_query_execution_max_time":154,
   "search_query_execution_avg_time":2,
   "object_loading_total_time":1,
   "object_loading_execution_max_time":1,
```

```
    "object_loading_execution_avg_time":1,
    "objects_loaded_count":20,
    "search_query_execution_max_time_query_string": "FROM entity"
}
```

- **search_query_execution_count** 実行されたクエリーの数を提供します。
- **search_query_total_time** クエリーに費やされた総時間を提供します。
- **search_query_execution_max_time** クエリーにかかる最大時間を指定します。
- **search_query_execution_avg_time** クエリーの平均実行時間を提供します。
- **object_loading_total_time** クエリー実行後にキャッシュからオブジェクトをロードするのにかかった時間の合計を提供します。
- **object_loading_execution_max_time** オブジェクトのロード実行にかかる最大時間を提供します。
- **object_loading_execution_avg_time** オブジェクトのロード実行にかかる最大時間を提供します。
- **objects_loaded_count** ロードされたオブジェクトの数を提供します。
- **search_query_execution_max_time_query_string** 実行された最も遅いクエリーを提供します。

2.1.22. クエリー統計のクリア

GET リクエストと **?action=clear** パラメーターを使用して、ランタイムの統計情報をリセットすることができます。

```
GET /v2/caches/{cacheName}/search/query/stats?action=clear
```

2.1.23. キャッシュの一覧表示

GET リクエストを使用して、 Data Grid クラスターで使用可能なすべてのキャッシュを一覧表示します。

```
GET /rest/v2/caches/
```

2.1.24. キャッシュを利用したクロスサイト・オペレーション

Data Grid REST API を使用して、クロスサイトレプリケーション操作を行います。

2.1.24.1. すべてのバックアップロケーションのステータス取得

GET リクエストですべてのバックアップロケーションのステータスを取得します。

```
GET /v2/caches/{cacheName}/x-site/backups/
```

Data Grid は、以下の例のように、各バックアップロケーションのステータスを JSON 形式で応答します。

-

```
{
  "NYC": "online",
  "LON": "offline"
}
```

表2.9 リターンステータス

値	説明
online	ローカルクラスター内のすべてのノードには、バックアップの場所を含むクロスサイトビューがあります。
offline	ローカルクラスター内のノードには、バックアップの場所とのクロスサイトビューがありません。
mixed	ローカルクラスター内の一部のノードにはバックアップの場所を含むクロスサイトビューがあり、ローカルクラスター内の他のノードにはクロスサイトビューがありません。応答は、各ノードのステータスを示します。

2.1.24.2. 特定のバックアップ場所のステータスの取得

GET リクエストでバックアップロケーションのステータスを取得する。

```
GET /v2/caches/{cacheName}/x-site/backups/{siteName}
```

Data Grid は、以下の例のように、サイト内の各ノードのステータスを JSON 形式で応答します。

```
{
  "NodeA":"offline",
  "NodeB":"online"
}
```

表2.10 リターンステータス

値	説明
online	ノードはオンラインです。
offline	ノードはオフラインです。
failed	ステータスを取得できません。リモートキャッシュがシャットダウンしているか、リクエスト中にネットワークエラーが発生した可能性があります。

2.1.24.3. バックアップ先をオフラインにする

GET リクエストと **?action=take-offline** パラメーターを使用して、バックアップの場所をオフラインにします。

```
GET /v2/caches/{cacheName}/x-site/backups/{siteName}?action=take-offline
```

2.1.24.4. バックアップ場所をオンラインにする

?action=bring-online パラメーターを使用してバックアップ場所をオンラインにします。

```
GET /v2/caches/{cacheName}/x-site/backups/{siteName}?action=bring-online
```

2.1.24.5. バックアップ場所への状態のプッシュ

?action=start-push-state パラメーターを使用して、キャッシュ状態をバックアップ場所にプッシュします。

```
GET /v2/caches/{cacheName}/x-site/backups/{siteName}?action=start-push-state
```

2.1.24.6. 状態転送のキャンセル

?action=cancel-push-state パラメーターを使用して状態転送操作をキャンセルします。

```
GET /v2/caches/{cacheName}/x-site/backups/{siteName}?action=cancel-push-state
```

2.1.24.7. 状態転送ステータスの取得

?action=push-state-status パラメーターを使用して状態転送操作のステータスを取得します。

```
GET /v2/caches/{cacheName}/x-site/backups?action=push-state-status
```

Data Grid は、以下の例のように、各バックアップ拠点の状態移行の状況を JSON 形式で応答します。

```
{
  "NYC":"CANCELED",
  "LON":"OK"
}
```

表2.11 リターンステータス

値	説明
SENDING	バックアップ場所への状態転送が進行中です。
OK	状態の転送が正常に完了しました。
ERROR	状態転送でエラーが発生しました。ログファイルを確認してください。
CANCELLING	状態移行のキャンセルが進行中です。

2.1.24.8. 状態転送ステータスのクリア

?action=clear-push-state-status パラメーターを使用して送信サイトの状態転送ステータスをクリアします。

```
GET /v2/caches/{cacheName}/x-site/local?action=clear-push-state-status
```

2.1.24.9. オフラインテイク条件の変更

特定の条件が満たされると、サイトはオフラインになります。オフラインにするパラメーターを変更して、バックアップロケーションが自動的にオフラインになるタイミングを制御します。

手順

1. **GET** リクエストと **take-offline-config** パラメーターで設定されたテイクオフラインパラメーターを確認します。

   ```
   GET /v2/caches/{cacheName}/x-site/backups/{siteName}/take-offline-config
   ```

   Data Grid のレスポンスには、以下のように **after_failures** と **min_wait** フィールドがあります。

   ```
   {
     "after_failures": 2,
     "min_wait": 1000
   }
   ```

2. **PUT** リクエストの本文のオフライン取得パラメーターを変更します。

   ```
   PUT /v2/caches/{cacheName}/x-site/backups/{siteName}/take-offline-config
   ```

2.1.24.10. 受信サイトからの状態転送のキャンセル

2 つのバックアップ場所間の接続が切断された場合、プッシュを受信しているサイトでの状態転送をキャンセルできます。

?action=cancel-receive-state パラメーターで、リモートサイトからの状態転送をキャンセルし、ローカルキャッシュの現在の状態を維持する。

```
GET /v2/caches/{cacheName}/x-site/backups/{siteName}?action=cancel-receive-state
```

2.1.25. ローリングアップグレード

Data Grid クラスター間でキャッシュデータのローリングアップグレードを実行します

2.1.25.1. データの同期

ソースクラスターからターゲットクラスターへのデータの同期には、**GET** リクエストと **?action=sync-data** パラメーターを使用します。

```
GET /v2/caches/{cacheName}?action=sync-data
```

操作が完了すると、Data Grid はターゲットクラスターにコピーされたエントリーの合計数で応答します。

2.1.25.2. ソースクラスターの接続

データをターゲットクラスターに同期した後、**GET** リクエストと **?action=disconnect-source** パラメーターを使用してソースクラスターから切断します。

```
GET /v2/caches/{cacheName}?action=disconnect-source
```

2.2. カウンターの作成と管理

REST API を使ってカウンターの作成、削除、修正ができます。

2.2.1. カウンターの作成

ペイロードに設定が含まれる **POST** リクエストでカウンターを作成します。

```
POST /rest/v2/counters/{counterName}
```

弱いカウンターの例

```
{
   "weak-counter":{
      "initial-value":5,
      "storage":"PERSISTENT",
      "concurrency-level":1
   }
}
```

強力なカウンターの例

```
{
   "strong-counter":{
      "initial-value":3,
      "storage":"PERSISTENT",
      "upper-bound":5
   }
}
```

2.2.2. カウンターの削除

DELETE リクエストで特定のカウンターを削除します。

```
DELETE /rest/v2/counters/{counterName}
```

2.2.3. カウンター設定の取得

GET リクエストで特定のカウンターの設定を取得します。

```
GET /rest/v2/counters/{counterName}/config
```

–

Data Grid は、JSON 形式でカウンターの設定を応答します。

2.2.4. カウンターへの値の追加

POST リクエストを使用して、特定のカウンターに値を追加します。

重要

この方法では、**plain/text** コンテンツのみを処理します。

```
POST /rest/v2/counters/{counterName}
```

リクエストペイロードが空の場合、カウンターは 1 つずつ増えます。そうでないと、ペイロードは signed long として解釈され、カウンターに追加されます。

注記

WEAK カウンターは操作後に応答しません。

STRONG カウンターは、各操作の後に現在の値を返します。

2.2.5. カウンター値の取得

GET リクエストでカウンターの値を取得します。

```
GET /rest/v2/counters/{counterName}
```

表2.12 ヘッダー

ヘッダー	必須またはオプション	パラメーター
Accept	オプション	コンテンツを返すために必要なフォーマットです。対応フォーマットは、**application/jsonとtext/plain**です。ヘッダーが指定されていない場合、JSON が想定されます。

2.2.6. カウンターのリセット

GET リクエストと **?action=reset** パラメーターなしでカウンターの初期値を復元します。

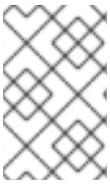
```
GET /rest/v2/counters/{counterName}?action=reset
```

2.2.7. カウンターのインクリメント

GET リクエストに **?action=increment** パラメーターを指定して、カウンターの値を増加させます。

```
GET /rest/v2/counters/{counterName}?action=increment
```

-



注記

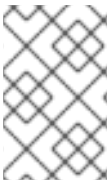
WEAK カウンターは操作後に応答しません。

STRONG カウンターは、各操作の後に現在の値を返します。

2.2.8. カウンターへのデルタの追加

?action=add および **delta** パラメーターを含む **GET** リクエストで、カウンターに任意の値を追加します。

```
GET /rest/v2/counters/{counterName}?action=add&delta={delta}
```



注記

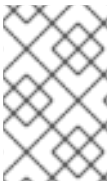
WEAK カウンターは操作後に応答しません。

STRONG カウンターは、各操作の後に現在の値を返します。

2.2.9. カウンター値のデクリメント

カウンターの値を減少させるには、**GET** リクエストと **?action=decrement** パラメーターを使用します。

```
GET /rest/v2/counters/{counterName}?action=decrement
```



注記

WEAK カウンターは操作後に応答しません。

STRONG カウンターは、各操作の後に現在の値を返します。

2.2.10. ストロングカウンターでの compareAndSet オペレーションの実行

GET リクエストと **compareAndSet** パラメーターを使って、強力なカウンターの値をアトミックに設定します。

```
GET /rest/v2/counters/{counterName}?action=compareAndSet&expect={expect}&update={update}
```

Data Grid は、現在の値が **{expect}** の場合、**{update}** にアトミックに値を設定します。操作が成功した場合、Data Grid は **true**.

2.2.11. 強力なカウンターでの compareAndSwap 操作の実行

GET リクエストと **compareAndSwap** パラメーターで強力なカウンターの値をアトミックに設定します。

```
GET /rest/v2/counters/{counterName}?action=compareAndSwap&expect={expect}&update={update}
```

Data Grid は、現在の値が **{expect}** の場合、**{update}** にアトミックに値を設定します。操作が成功すると、 Data Grid はペイロードの前の値を返します。

2.2.12. カウンターの一覧表示

GET リクエストで Data Grid クラスターのカウンターのリストを取得します。

```
GET /rest/v2/counters/
```

2.3. キャッシュマネージャーの操作

Data Grid キャッシュマネージャーと対話して、クラスターと使用状況の統計を取得します。

2.3.1. 基本的なキャッシュマネージャー情報の取得

GET リクエストで Cache Manager の情報を取得します。

```
GET /rest/v2/cache-managers/{cacheManagerName}
```

Data Grid は、次の例のように、JSON 形式の情報で応答します。


```
{
    "version":"xx.x.x-FINAL",
    "name":"default",
    "coordinator":true,
    "cache_configuration_names":[
        "___protobuf_metadata",
        "cache2",
        "CacheManagerResourceTest",
        "cache1"
    ],
    "cluster_name":"ISPN",
    "physical_addresses":"[127.0.0.1:35770]",
    "coordinator_address":"CacheManagerResourceTest-NodeA-49696",
    "cache_manager_status":"RUNNING",
    "created_cache_count":"3",
    "running_cache_count":"3",
    "node_address":"CacheManagerResourceTest-NodeA-49696",
    "cluster_members":[
        "CacheManagerResourceTest-NodeA-49696",
        "CacheManagerResourceTest-NodeB-28120"
    ],
    "cluster_members_physical_addresses":[
        "127.0.0.1:35770",
        "127.0.0.1:60031"
    ],
    "cluster_size":2,
    "defined_caches":[
        {
            "name":"CacheManagerResourceTest",
            "started":true
        },
        {
            "name":"cache1",
            "started":true
        },
        {
```

```
            "name":"___protobuf_metadata",
            "started":true
        },
        {
            "name":"cache2",
            "started":true
        }
    ]

}
```

- **version** は、 Data Grid バージョンが含まれています
- **name** には、コンフィギュレーションで定義されたキャッシュマネージャーの名前が含まれます。
- **coordinator** は、キャッシュ・マネージャーがクラスターのコーディネーターである場合には真となります。
- **cache_configuration_names** には、キャッシュマネージャーで定義されたすべてのキャッシュ設定の配列が含まれます
- **cluster_name** には、設定で定義されたクラスターの名前が含まれます。
- **physical_addresses** は、キャッシュマネージャーに関連する物理ネットワークアドレスを含みます。
- **coordinator_address** には、クラスターのコーディネーターの物理ネットワークアドレスが含まれます
- **cache_manager_status** キャッシュマネージャーのライフサイクルの状態です。可能な値については、 **org.infinispan.lifecycle.ComponentStatus** ドキュメントを確認してください
- **created_cache_count** 作成されたキャッシュの数、すべての内部およびプライベートキャッシュを除く
- **running_cache_count** 実行中の作成されたキャッシュの数
- **node_address** には、キャッシュマネージャーの論理アドレスが含まれます
- **cluster_members** および **cluster_members_physical_addresses** は、クラスターのメンバーの論理アドレスと物理アドレスの配列です。
- **cluster_size** クラスター内のメンバーの数
- **defined_caches** キャッシュマネージャーで定義されているすべてのキャッシュのリスト。プライベートキャッシュは除きますが、アクセス可能な内部キャッシュは含まれます。

2.3.2. クラスターヘルスの取得

GET リクエストを使用して Data Grid クラスターのヘルス情報を取得します。

```
GET /rest/v2/cache-managers/{cacheManagerName}/health
```

Data Grid は、次の例のように、JSON 形式のクラスターヘルス情報で応答します。

```
{
  "cluster_health":{
    "cluster_name":"ISPN",
    "health_status":"HEALTHY",
    "number_of_nodes":2,
    "node_names":[
      "NodeA-36229",
      "NodeB-28703"
    ]
  },
  "cache_health":[
    {
      "status":"HEALTHY",
      "cache_name":"___protobuf_metadata"
    },
    {
      "status":"HEALTHY",
      "cache_name":"cache2"
    },
    {
      "status":"HEALTHY",
      "cache_name":"mycache"
    },
    {
      "status":"HEALTHY",
      "cache_name":"cache1"
    }
  ]

}
```

- **cluster_health** には、クラスターのヘルスが含まれます
  - **cluster_name** は、設定で定義されているクラスターの名前を指定します。
  - **health_status** は、次のいずれかを提供します。
    - **DEGRADED** は、キャッシュの少なくとも 1 つが劣化モードにあることを示します。
    - **HEALTHY_REBALANCING** は、少なくとも 1 つのキャッシュがリバランス状態にあることを示します。
    - **HEALTHY** は、クラスター内のすべてのキャッシュインスタンスが期待どおりに動作していることを示します。
  - **number_of_nodes** は、クラスターメンバーの総数を表示します。非クラスター化 (スタンドアロン) サーバーの場合は値 **0** を返します。
  - **node_names** は、すべてのクラスターメンバーの配列です。スタンドアロンサーバーの場合は空です。
- **cache_health** には、キャッシュごとのヘルス情報が含まれています
  - **status** HEALTHY、DEGRADED または HEALTHY_REBALANCING
  - **cache_name** 設定で定義されているキャッシュの名前。

2.3.3. キャッシュマネージャーのヘルスステータスの取得

認証を必要としない **GET** リクエストを使用してキャッシュマネージャーのヘルスステータスを取得します。

```
GET /rest/v2/cache-managers/{cacheManagerName}/health/status
```

Data Grid は以下のいずれかを **text/plain** 形式で応答します。

- **HEALTHY**
- **HEALTHY_REBALANCING**
- **DEGRADED**

2.3.4. REST エンドポイントの可用性の確認

HEAD リクエストを使用して Data Grid サーバーの REST エンドポイントの可用性を確認します。

```
HEAD /rest/v2/cache-managers/{cacheManagerName}/health
```

正常な応答コードを受信した場合、Data Grid REST サーバーが実行され、要求を処理しています。

2.3.5. キャッシュマネージャーのグローバル設定の取得

GET 要求を使用してキャッシュマネージャーのグローバル設定を取得します。

```
GET /rest/v2/cache-managers/{cacheManagerName}/config
```

表2.13 ヘッダー

ヘッダー	必須またはオプション	パラメーター
Accept	オプション	コンテンツを返すために必要なフォーマットです。対応フォーマットは、**application/jsonとapplication/xml**です。ヘッダーが指定されていない場合、JSON が想定されます。

参照資料

GlobalConfiguration

2.3.6. すべてのキャッシュの設定を取得する

GET リクエストを使用してすべてのキャッシュの設定を取得します。

```
GET /rest/v2/cache-managers/{cacheManagerName}/cache-configs
```

Data Grid は、以下の例のように、各キャッシュとキャッシュ設定を含む **JSON** 配列で応答します。

```
[
  {
     "name":"cache1",
     "configuration":{
        "distributed-cache":{
           "mode":"SYNC",
           "partition-handling":{
              "when-split":"DENY_READ_WRITES"
           },
           "statistics":true
        }
     }
  },
  {
     "name":"cache2",
     "configuration":{
        "distributed-cache":{
           "mode":"SYNC",
           "transaction":{
              "mode":"NONE"
           }
        }
     }
  }
]
```

2.3.7. 利用可能なキャッシュテンプレートの一覧表示

GET リクエストで、利用可能なすべての Data Grid キャッシュテンプレートを取得します。

```
GET /rest/v2/cache-managers/{cacheManagerName}/cache-configs/templates
```

ヒント

テンプレートを使ったキャッシュの作成 を参照してください。

2.3.8. (実験的) キャッシュのステータスと情報の取得

Cache Manager で利用可能なすべてのキャッシュのリストを、キャッシュ・ステータスおよび詳細とともに、**GET** 要求で取得します。

```
GET /rest/v2/cache-managers/{cacheManagerName}/caches
```

Data Grid は、次の例のように、使用可能な各キャッシュを一覧表示して説明する JSON 配列で応答します。

```
[ {
  "status" : "RUNNING",
  "name" : "cache1",
  "type" : "local-cache",
  "simple_cache" : false,
  "transactional" : false,
  "persistent" : false,
```

```
  "bounded": false,
  "secured": false,
  "indexed": true,
  "has_remote_backup": true,
  "health":"HEALTHY"
}, {
  "status" : "RUNNING",
  "name" : "cache2",
  "type" : "distributed-cache",
  "simple_cache" : false,
  "transactional" : true,
  "persistent" : false,
  "bounded": false,
  "secured": false,
  "indexed": true,
  "has_remote_backup": true,
  "health":"HEALTHY"
}]
```

2.3.9. キャッシュマネージャー統計の取得

GET リクエストを使用してキャッシュマネージャーの統計を取得します。

```
GET /rest/v2/cache-managers/{cacheManagerName}/stats
```

Data Grid は、次の例のように、JSON 形式のキャッシュマネージャー統計で応答します。

```
{
   "statistics_enabled":true,
   "read_write_ratio":0.0,
   "time_since_start":1,
   "time_since_reset":1,
   "number_of_entries":0,
   "total_number_of_entries":0,
   "off_heap_memory_used":0,
   "data_memory_used":0,
   "misses":0,
   "remove_hits":0,
   "remove_misses":0,
   "evictions":0,
   "average_read_time":0,
   "average_read_time_nanos":0,
   "average_write_time":0,
   "average_write_time_nanos":0,
   "average_remove_time":0,
   "average_remove_time_nanos":0,
   "required_minimum_number_of_nodes":1,
   "hits":0,
   "stores":0,
   "current_number_of_entries_in_memory":0,
   "hit_ratio":0.0,
   "retrievals":0
}
```

- **statistics_enabled** は、Cache Manager で統計情報の収集が有効になっている場合に **true** になります。
- **read_write_ratio** は、すべてのキャッシュにわたる読み取り/書き込み比率を表示します。
- **time_since_start** は、キャッシュマネージャーが開始されてからの時間を秒単位で示します。
- **time_since_reset** は、キャッシュマネージャーの統計が最後にリセットされてからの秒数を示します。
- **number_of_entries** は、キャッシュマネージャーから現在すべてのキャッシュにあるエントリーの総数を示します。この統計は、ローカルキャッシュインスタンスのエントリーのみを返します。
- **total_number_of_entries** は、キャッシュマネージャーのすべてのキャッシュで実行されたストア操作の数を示します。
- **off_heap_memory_used** は、このキャッシュコンテナーが使用しているオフヒープメモリーの量を **bytes[]** 単位で示します。
- **data_memory_used** は、現在の退避アルゴリズムが全キャッシュのデータに使用されていると推定している量を **bytes[]** 単位で示します。エヴィクションが有効になっていない場合は **0** を返します。
- **misses** は、すべてのキャッシュにおける **get()** のミスの数を示しています。
- **remove_hits** は、すべてのキャッシュにわたる削除ヒットの数を示します。
- **remove_misses** は、すべてのキャッシュにわたる削除ミスの数を示します。
- **evictions** は、すべてのキャッシュにおける エヴィクション の数を示しています。
- **average_read_time** は、すべてのキャッシュで **get()** 操作にかかったミリ秒数の平均値を示します。
- **average_read_time_nanos** は **average_read_time** と同じですが、単位はナノ秒です。
- **average_remove_time** は、すべてのキャッシュにおける **remove()** 操作の平均ミリ秒数を示します。
- **average_remove_time_nanos** は **average_remove_time** と同じですが、単位はナノ秒です。
- **required_minimum_number_of_nodes** は、データの一貫性を保証するために必要な最小のノード数を示します。
- **hits** は、すべてのキャッシュにおける **get()** のヒット数を示します。
- **stores** は、すべてのキャッシュにおける **put()** 操作の回数を提供します。
- **current_number_of_entries_in_memory** は、パッシベーションされたエントリーを除く、現在すべてのキャッシュにあるエントリーの総数を示します。
- **hit_ratio** は、すべてのキャッシュの合計ヒット率/(ヒット+ミス) 比率を提供します。
- **retrievals** は、**get()** 操作の総数を示しています。

2.3.10. キャッシュ・マネージャーによるクロスサイト操作

Cache Managers でクロスサイト操作を行うと、すべてのキャッシュに操作が適用されます。

2.3.10.1. バックアップ場所のステータスの取得

GET 要求により、キャッシュ・マネージャーからすべてのバックアップ・ロケーションのステータスを取得します。

```
GET /rest/v2/cache-managers/{cacheManagerName}/x-site/backups/
```

Data Grid は、以下の例のように JSON 形式でステータスを応答します。

```
{
  "SFO-3":{
    "status":"online"
  },
  "NYC-2":{
    "status":"mixed",
    "online":[
      "CACHE_1"
    ],
    "offline":[
      "CACHE_2"
    ]
  }
}
```

表2.14 リターンステータス

値	説明
online	ローカルクラスター内のすべてのノードには、バックアップの場所を含むクロスサイトビューがあります。
offline	ローカルクラスター内のノードには、バックアップの場所とのクロスサイトビューがありません。
mixed	ローカルクラスター内の一部のノードにはバックアップの場所を含むクロスサイトビューがあり、ローカルクラスター内の他のノードにはクロスサイトビューがありません。応答は、各ノードのステータスを示します。

2.3.10.2. バックアップ先をオフラインにする

?action=take-offline パラメーターで、バックアップロケーションをオフラインにします。

```
GET /rest/v2/cache-managers/{cacheManagerName}/x-site/backups/{siteName}?action=take-offline
```

2.3.10.3. バックアップ場所をオンラインにする

?action=bring-online パラメーターを使用してバックアップ場所をオンラインにします。

```
GET /rest/v2/cache-managers/{cacheManagerName}/x-site/backups/{siteName}?action=bring-online
```

2.3.10.4. 状態転送の開始

?action=start-push-state パラメーターを使用して、すべてのキャッシュの状態をリモートサイトにプッシュします。

```
GET /rest/v2/cache-managers/{cacheManagerName}/x-site/backups/{siteName}?action=start-push-state
```

2.3.10.5. 状態転送のキャンセル

?action=cancel-push-state パラメーターを使用して、進行中の状態転送操作をキャンセルします。

```
GET /rest/v2/cache-managers/{cacheManagerName}/x-site/backups/{siteName}?action=cancel-push-state
```

2.4. DATA GRID サーバーの操作

Data Grid サーバーインスタンスを監視および管理します。

2.4.1. 基本的なサーバー情報の取得

GET リクエストを使用して Data Grid サーバーに関する基本情報を表示します。

```
GET /rest/v2/server
```

Data Grid は、次の例のように、サーバー名、コードネーム、およびバージョンを JSON 形式で応答します。

```
{
  "version":"Infinispan 'Codename' xx.x.x.Final"
}
```

2.4.2. キャッシュマネージャーの取得

GET リクエストで Data Grid サーバーのキャッシュマネージャーのリストを取得します。

```
GET /rest/v2/server/cache-managers
```

Data Grid は、サーバー用に設定されたキャッシュマネージャー名の配列で応答します。

2.4.3. 無視リストへのキャッシュの追加

特定のキャッシュをクライアントの要求から一時的に除外するように Data Grid を設定します。キャッシュマネージャー名とキャッシュの名前を含む **POST** リクエストを送信します。

```
POST /v2/server/ignored-caches/{cache-manager}/{cache}
```

Data Grid は、REST クライアントリクエストのサービスが利用できないステータス (**503**) と、Hot Rod クライアントリクエストのサーバーエラー (**code 0x85**) を返します。

注記

Data Grid は現在、サーバーごとに 1 つのキャッシュマネージャーのみをサポートしています。将来の互換性のために、リクエストでキャッシュマネージャー名を指定する必要があります。

2.4.4. 無視リストからのキャッシュの削除

DELETE リクエストでキャッシュを無視リストから削除します。

```
DELETE /v2/server/ignored-caches/{cache-manager}/{cache}
```

2.4.5. 無視されたキャッシュの確認

GET リクエストでキャッシュが無視されることを確認します。

```
GET /v2/server/ignored-caches/{cache-manager}
```

2.4.6. サーバー設定の取得

GET リクエストで Data Grid サーバーの設定を取得します。

```
GET /rest/v2/server/config
```

Data Grid は、以下のように JSON 形式で設定を応答します。

```
{
   "server":{
      "interfaces":{
         "interface":{
            "name":"public",
            "inet-address":{
               "value":"127.0.0.1"
            }
         }
      },
      "socket-bindings":{
         "port-offset":0,
         "default-interface":"public",
         "socket-binding":[
            {
               "name":"memcached",
               "port":11221,
               "interface":"memcached"
            }
         ]
      },
      "security":{
         "security-realms":{
            "security-realm":{
```

```
                "name":"default"
            }
        }
    },
    "endpoints":{
        "socket-binding":"default",
        "security-realm":"default",
        "hotrod-connector":{
            "name":"hotrod"
        },
        "rest-connector":{
            "name":"rest"
        }
    }
  }
}
```

2.4.7. 環境変数の取得

GET リクエストで Data Grid サーバーのすべての環境変数を取得します。

```
GET /rest/v2/server/env
```

2.4.8. JVM メモリーの詳細の取得

GET リクエストで Data Grid サーバーの JVM メモリー使用量情報を取得します。

```
GET /rest/v2/server/memory
```

Data Grid は、ヒープと非ヒープのメモリー統計、直接のメモリー使用量、メモリープールとガベージコレクションに関する情報を JSON 形式で応答します。

2.4.9. JVM スレッドダンプの取得

GET リクエストで、JVM の現在のスレッドダンプを取得します。

```
GET /rest/v2/server/threads
```

Data Grid は現在のスレッドダンプを **text/plain** 形式で応答します。

2.4.10. Data Grid サーバーの停止

GET リクエストで Data Grid サーバーを停止します。

```
GET /rest/v2/server?action=stop
```

Data Grid は **200(OK)** で応答し、実行を停止します。

2.5. DATA GRID クラスターの操作

Data Grid クラスターの管理タスクを監視および実行します。

2.5.1. Data Grid クラスターの停止

GET リクエストを使用して Data Grid クラスター全体をシャットダウンします。

```
GET /rest/v2/cluster?action=stop
```

Data Grid は **200(OK)** で応答し、クラスター全体を順番にシャットダウンします。

2.5.2. クラスター内の特定の Data Grid サーバーの停止

GET リクエストに **?action=stop&server** パラメーターを指定して、Data Grid クラスター内の 1 つまたは複数の特定のサーバーをシャットダウンすることができます。

```
GET /rest/v2/cluster?action=stop&server={server1_host}&server={server2_host}
```

Data Grid は **200(OK)** で応答します。

2.6. サーバータスクの使用

Data Grid サーバータスクを取得、実行、およびアップロードします。

2.6.1. サーバータスク情報の取得

GET リクエストで利用可能なサーバータスクに関する情報を表示します。

```
GET /rest/v2/tasks
```

表2.15 リクエストパラメーター

パラメーター	必須またはオプション	値
type	オプション	**user**: 内部 (管理者) のタスクを結果から除外します。

Data Grid は、利用可能なタスクのリストで応答します。リストには、次の例のように、タスクの名前、タスクを処理するエンジン、タスクの名前付きパラメーター、タスクの実行モード (**ONE_NODE** または **ALL_NODES**)、許可されるセキュリティーロールが **JSON** 形式で記載されています。

```
[
  {
    "name": "SimpleTask",
    "type": "TaskEngine",
    "parameters": [
      "p1",
      "p2"
    ],
    "execution_mode": "ONE_NODE",
    "allowed_role": null
  },
  {
    "name": "RunOnAllNodesTask",
    "type": "TaskEngine",
```

```
    "parameters": [
      "p1"
    ],
    "execution_mode": "ALL_NODES",
    "allowed_role": null
  },
  {
    "name": "SecurityAwareTask",
    "type": "TaskEngine",
    "parameters": [],
    "execution_mode": "ONE_NODE",
    "allowed_role": "MyRole"
  }
]
```

2.6.2. タスクの実行

タスクの実行は、タスク名と **param** を先頭につけた必須パラメーターを含む **GET** リクエストで行います。

```
GET /rest/v2/tasks/myTask?action=exec&param.p1=v1&param.p2=v2
```

Data Grid はタスクの結果で応答します。

2.6.3. スクリプトタスクのアップロード

PUT または **POST** リクエストでスクリプトタスクをアップロードします。

リクエストのコンテンツペイロードとしてスクリプトを提供します。Data Grid がスクリプトをアップロードした後、**GET** リクエストでスクリプトを実行することができます。

```
POST /rest/v2/tasks/taskName
```

第3章 DATA GRID REST エンドポイント

Data Grid サーバーは、Netty 上に構築された REST エンドポイントを介してデータへの RESTful HTTP アクセスを提供します。

3.1. REST 認証

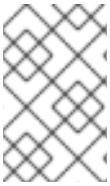
Data Grid サーバーディストリビューションの **bin/user-tool.sh** スクリプトを使用して、REST エンドポイントへの認証を設定します。

3.2. サポートされているプロトコル

Data Grid REST エンドポイントは、**HTTP/1.1** および **HTTP/2** のプロトコルをサポートしています。

HTTP/2 を使用するには、以下のいずれかの方法があります。

- HTTP /1.1 アップグレードを 実行します。
- TLS/ALPN 拡張 を使用して通信プロトコルを交渉する。



注記

JDK8 を使用する TLS / ALPN には、追加のクライアント設定が必要です。REST クライアントの適切なドキュメントを参照してください。ほとんどの場合、Jetty ALPNAgent または OpenSSL バインディングのいずれかを使用する必要があります。

3.3. データ形式と RESTAPI

Data Grid キャッシュは、MediaType で定義できる形式でデータを保存します。

次の例では、エントリーのストレージ形式を設定します。

```
<cache>
  <encoding>
    <key media-type="application/x-java-object; type=java.lang.Integer"/>
    <value media-type="application/xml; charset=UTF-8"/>
  </encoding>
</cache>
```

MediaType を設定しない場合、DataGrid はデフォルトでキーと値の両方に対して **application/octet-stream** になります。しかし、キャッシュがインデックス化されている場合、Data Grid のデフォルトは **application/x-protostream** です。

3.3.1. サポート対象の形式

さまざまな形式でデータの書き込みと読み取りを行うことができ、DataGrid は必要に応じてそれらの形式間で変換できます。

次の標準フォーマットは交換可能です。

- application/x-java-object
- application/octet-stream

- **application/x-www-form-urlencoded**
- **text/plain**

上記のデータ形式を次の形式に変換することもできます。

- **application/xml**
- **application/json**
- **application/x-jboss-marshalling**
- **application/x-protostream**
- **application/x-java-serialized**

Data Grid では、**application/x-protostream** と **application/json** の間で変換することもできます。

REST API へのすべての呼び出しは、書き込まれたコンテンツまたは読み取るときに必要なコンテンツの形式を説明するヘッダーを提供できます。Data Grid は、値に適用される標準的な HTTP/1.1 ヘッダーの "Content-Type "と "Accept "に加えて、キーに同様の効果を持つ "Key-Content-Type "をサポートしています。

3.3.2. ヘッダーを受け入れる

Data Grid REST エンドポイントは RFC-2616 Accept ヘッダーに準拠しており、サポートされている変換に基づいて正しい MediaType をネゴシエートします。

例えば、データ読み込み時に次のようなヘッダーを送信します。

```
Accept: text/plain;q=0.7, application/json;q=0.8, */*;q=0.6
```

上記のヘッダーにより、Data Grid は最初にコンテンツを JSON 形式 (優先度 0.8) で返します。保存形式を JSON に変換できない場合、Data Grid は次の形式である **text/plain** を試みます (2 番目に高い優先度 0.7)。最後に、Data Grid は***/***にフォールバックし、キャッシュの設定に基づいて適切なフォーマットを選択します。

3.3.3. 特殊文字を含む名前

REST リソースの作成には、URL の一部となる名前が必要です。この名前に RFC 3986 仕様のセクション 2.2 で定義されている特殊文字が含まれている場合には、Percent encoding メカニズムでエンコードする必要があります。

3.3.4. Key-Content-Type ヘッダー

ほとんどの REST API コールでは、URL に Key が含まれています。Data Grid は、これらの呼び出しを処理する際に、Key が**java.lang.String**であることを前提としていますが、異なるフォーマットのキーに対しては、特定のヘッダー**Key-Content-Type**を使用することができます。

Key-Content-Type ヘッダーの例

- byte[] Key を Base64 文字列で指定する

API 呼び出し:

```
`PUT /my-cache/AQIDBDM=`
```

ヘッダー:

Key-Content-Type: application/octet-stream

- byte[] Key を 16 進数の文字列で指定する。

API 呼び出し:

GET /my-cache/0x01CA03042F

ヘッダー:

```
Key-Content-Type: application/octet-stream; encoding=hex
```

- ダブルキーの指定:

API 呼び出し:

POST /my-cache/3.141456

ヘッダー:

```
Key-Content-Type: application/x-java-object;type=java.lang.Double
```

application/x-java-object の **type** パラメーターは次のように制限されています。

- Primitive wrapper types
- java.lang.String
- バイトで、**application/x-java-object;type=Bytes** が **application/octet-stream;encoding=hex** と同等になります

3.3.5. JSON / プロトストリーム変換

キャッシュがインデックス化されている場合や、**application/x-protostream** を保存するように特別に設定されている場合、Protostream との間で自動的に変換された JSON ドキュメントを送受信することができます。

変換を機能させるには、protobuf スキーマを登録する必要があります。

REST 経由で protobuf スキーマを登録するには、次の例のように、POST または PUT を呼び出して **___protobuf_metadata** キャッシュを起動します。

```
curl -u user:password -X POST --data-binary @./schema.proto
http://127.0.0.1:11222/rest/v2/caches/___protobuf_metadata/schema.proto
```

JSON ドキュメントを記述する際には、ドキュメントに対応する Protobuf Message を識別するための特別なフィールド **_type** は、ドキュメントに対応する protobuf **Message**を識別するために、ドキュメント内に存在する必要があります。

たとえば、以下のスキーマについて考えてみましょう。

```
message Person {
  required string name = 1;
  required int32 age = 2;
}
```

対応する JSON ドキュメントは以下のとおりです。

```
{
  "_type": "Person",
  "name": "user1",
  "age": 32
}
```

3.4. クロスオリジンリソースシェアリング (CORS) リクエスト

Data Grid REST コネクターは、プリフライトやリクエストの発信元に基づくルールなど、 CORS をサポートします。

以下に、CORS ルールを使用した REST コネクター設定の例を示します。

```
<rest-connector name="rest1" socket-binding="rest" cache-container="default">
  <cors-rules>
    <cors-rule name="restrict host1"
               allow-credentials="false">
      <allowed-origins>http://host1,https://host1</allowed-origins>
      <allowed-methods>GET</allowed-methods>
    </cors-rule>
    <cors-rule name="allow ALL"
               allow-credentials="true"
               max-age-seconds="2000">
      <allowed-origins>*</allowed-origins>
      <allowed-methods>GET,OPTIONS,POST,PUT,DELETE</allowed-methods>
      <allowed-headers>Key-Content-Type</allowed-headers>
    </cors-rule>
  </cors-rules>
</rest-connector>
```

Data Grid は、ブラウザーが設定した "Origin "ヘッダに基づいて CORS ルールを順次評価します。

前述の例では、オリジンが http://host1 または https://host1 のいずれかであれば、ルール restrict host1 が適用されます。オリジンが異なる場合は、次のルールがテストされます。

"allow ALL"ルールはすべてのオリジンを許可するため、"http://host1" または"https://host1" 以外のオリジンを持つスクリプトは、許可されたメソッドを実行し、提供されたヘッダーを使用することができます。

CORS ルールの設定については、Data Grid サーバー設定スキーマ を参照してください。