Red Hat Data Grid 8.5

Data Grid REST API

Data Grid RESTAPI を設定して操作する

Last Updated: 2024-07-11

Red Hat Data Grid 8.5 Data Grid REST API

Data Grid RESTAPI を設定して操作する

法律上の通知

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

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。

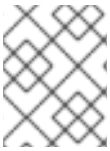
DATA GRID のドキュメント

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 8.4 ドキュメント
- Data Grid 8.4 コンポーネントの詳細
- Data Grid 7.3 でサポートされる構成
- Data Grid 8 機能のサポート
- Data Grid で非推奨の機能

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。

多様性を受け入れるオープンソースの強化

Red Hat では、コード、ドキュメント、Web プロパティーにおける配慮に欠ける用語の置き換えに取り組んでいます。まずは、マスター (master)、スレーブ (slave)、ブラックリスト (blacklist)、ホワイトリスト (whitelist) の 4 つの用語の置き換えから始めます。この取り組みは膨大な作業を要するため、用語の置き換えは、今後の複数のリリースにわたって段階的に実施されます。詳細は、Red Hat CTO である Chris Wright のメッセージ をご覧ください。

第1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。

1.1. REST 認証

Data Grid コマンドラインインターフェイス (CLI) と**user**コマンドを使用して、REST エンドポイントへの認証を設定します。CLI を使用すると、REST エンドポイントにアクセスするためのユーザー、パスワード、および承認ロールを作成および管理できます。

参照

- Data Grid ユーザーの作成
- エンドポイント認証メカニズムの設定

1.2. 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。

1.3. 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。

MediaTypes と Data Grid を使用したデータのエンコードの詳細については、キャッシュのエンコードとマーシャリング を参照してください。

次の例では、エントリーのストレージ形式を設定します。

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

1.3.1. 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。

1.3.2. 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。

1.3.3. 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。

1.3.4. 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。

1.3.5. 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、Protobuf 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** を起動して **___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。

Person.proto

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

Person.json

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

1.4. 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。

1.4.1. 一部のオリジンに対してすべての CORS パーミッションを許可する

VM プロパティーの **infinispan.server.rest.cors-allow** は、サーバーの起動時に使用して、1 つまたは複数のオリジンにすべてのパーミッションを許可することができます。以下に例を示します。

```
./bin/server.sh -Dinfinispan.server.rest.cors-
allow=http://192.168.1.78:11222,http://host.mydomain.com
```

この方法を使用して指定されたすべてのオリジンは、設定されたルールよりも優先されます。

第2章 DATA GRID REST API との連携

Data Grid REST API は、Data Grid のデプロイメントを監視、維持、管理し、データへのアクセスを提供します。

注記

デフォルトでは、Data Grid REST API の操作は、成功すると **200 (OK)** を返します。ただし、一部の操作が正常に処理されると、 **200** ではなく **204** や **202** などの HTTP ステータスコードが返されます。

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}
```

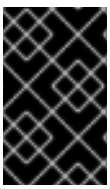
表2.1 ヘッダー

Header	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

2.1.1.1. キャッシュ設定

XML、JSON、および YAML 形式で宣言型キャッシュ設定を作成できます。

すべての宣言型キャッシュは Data Grid スキーマに準拠する必要があります。JSON 形式の設定は XML 設定の構造に従う必要があります。要素がオブジェクトに対応し、属性はフィールドに対応します。



重要

Data Grid では、キャッシュ名またはキャッシュテンプレート名の文字数を最大 **255** 文字に制限しています。この文字制限を超えると、Data Grid は例外を出力します。簡潔なキャッシュ名とキャッシュテンプレート名を記述します。

重要

ファイルシステムによってファイル名の長さに制限が設定される場合があるため、キャッシュの名前がこの制限を超えないようにしてください。キャッシュ名がファイルシステムの命名制限を超えると、そのキャッシュに対する一般的な操作または初期化操作が失敗する可能性があります。簡潔なファイル名を書きます。

分散キャッシュ

XML

```
<distributed-cache owners="2"
                   segments="256"
                   capacity-factor="1.0"
                   l1-lifespan="5000"
                   mode="SYNC"
                   statistics="true">
  <encoding media-type="application/x-protostream"/>
  <locking isolation="REPEATABLE_READ"/>
  <transaction mode="FULL_XA"
               locking="OPTIMISTIC"/>
  <expiration lifespan="5000"
              max-idle="1000" />
  <memory max-count="1000000"
          when-full="REMOVE"/>
  <indexing enabled="true"
            storage="local-heap">
    <index-reader refresh-interval="1000"/>
    <indexed-entities>
      <indexed-entity>org.infinispan.Person</indexed-entity>
    </indexed-entities>
  </indexing>
  <partition-handling when-split="ALLOW_READ_WRITES"
                      merge-policy="PREFERRED_NON_NULL"/>
  <persistence passivation="false">
    <!-- Persistent storage configuration. -->
  </persistence>
</distributed-cache>
```

JSON

```
{
  "distributed-cache": {
    "mode": "SYNC",
    "owners": "2",
    "segments": "256",
    "capacity-factor": "1.0",
    "l1-lifespan": "5000",
    "statistics": "true",
    "encoding": {
      "media-type": "application/x-protostream"
    },
    "locking": {
      "isolation": "REPEATABLE_READ"
```

```
    },
    "transaction": {
      "mode": "FULL_XA",
      "locking": "OPTIMISTIC"
    },
    "expiration" : {
      "lifespan" : "5000",
      "max-idle" : "1000"
    },
    "memory": {
      "max-count": "1000000",
      "when-full": "REMOVE"
    },
    "indexing" : {
          "enabled" : true,
          "storage" : "local-heap",
          "index-reader" : {
            "refresh-interval" : "1000"
          },
          "indexed-entities": [
            "org.infinispan.Person"
          ]
    },
    "partition-handling" : {
      "when-split" : "ALLOW_READ_WRITES",
      "merge-policy" : "PREFERRED_NON_NULL"
    },
    "persistence" : {
      "passivation" : false
    }
  }
}
```

YAML

```
distributedCache:
  mode: "SYNC"
  owners: "2"
  segments: "256"
  capacityFactor: "1.0"
  l1Lifespan: "5000"
  statistics: "true"
  encoding:
    mediaType: "application/x-protostream"
  locking:
    isolation: "REPEATABLE_READ"
  transaction:
    mode: "FULL_XA"
    locking: "OPTIMISTIC"
  expiration:
    lifespan: "5000"
    maxIdle: "1000"
  memory:
    maxCount: "1000000"
    whenFull: "REMOVE"
```

```
  indexing:
    enabled: "true"
    storage: "local-heap"
    indexReader:
      refreshInterval: "1000"
    indexedEntities:
      - "org.infinispan.Person"
  partitionHandling:
    whenSplit: "ALLOW_READ_WRITES"
    mergePolicy: "PREFERRED_NON_NULL"
  persistence:
    passivation: "false"
    # Persistent storage configuration.
```

レプリケートされたキャッシュ

XML

```
<replicated-cache segments="256"
                  mode="SYNC"
                  statistics="true">
  <encoding media-type="application/x-protostream"/>
  <locking isolation="REPEATABLE_READ"/>
  <transaction mode="FULL_XA"
               locking="OPTIMISTIC"/>
  <expiration lifespan="5000"
              max-idle="1000" />
  <memory max-count="1000000"
          when-full="REMOVE"/>
  <indexing enabled="true"
            storage="local-heap">
    <index-reader refresh-interval="1000"/>
    <indexed-entities>
      <indexed-entity>org.infinispan.Person</indexed-entity>
    </indexed-entities>
  </indexing>
  <partition-handling when-split="ALLOW_READ_WRITES"
                      merge-policy="PREFERRED_NON_NULL"/>
  <persistence passivation="false">
    <!-- Persistent storage configuration. -->
  </persistence>
</replicated-cache>
```

JSON

```
{
  "replicated-cache": {
    "mode": "SYNC",
    "segments": "256",
    "statistics": "true",
    "encoding": {
      "media-type": "application/x-protostream"
    },
    "locking": {
```

```
      "isolation": "REPEATABLE_READ"
    },
    "transaction": {
      "mode": "FULL_XA",
      "locking": "OPTIMISTIC"
    },
    "expiration" : {
      "lifespan" : "5000",
      "max-idle" : "1000"
    },
    "memory": {
      "max-count": "1000000",
      "when-full": "REMOVE"
    },
    "indexing" : {
      "enabled" : true,
      "storage" : "local-heap",
      "index-reader" : {
        "refresh-interval" : "1000"
        },
      "indexed-entities": [
        "org.infinispan.Person"
      ]
    },
    "partition-handling" : {
      "when-split" : "ALLOW_READ_WRITES",
      "merge-policy" : "PREFERRED_NON_NULL"
    },
    "persistence" : {
      "passivation" : false
    }
  }
}
```

YAML

```
replicatedCache:
  mode: "SYNC"
  segments: "256"
  statistics: "true"
  encoding:
    mediaType: "application/x-protostream"
  locking:
    isolation: "REPEATABLE_READ"
  transaction:
    mode: "FULL_XA"
    locking: "OPTIMISTIC"
  expiration:
    lifespan: "5000"
    maxIdle: "1000"
  memory:
    maxCount: "1000000"
    whenFull: "REMOVE"
  indexing:
    enabled: "true"
```

```
    storage: "local-heap"
    indexReader:
      refreshInterval: "1000"
    indexedEntities:
      - "org.infinispan.Person"
  partitionHandling:
    whenSplit: "ALLOW_READ_WRITES"
    mergePolicy: "PREFERRED_NON_NULL"
  persistence:
    passivation: "false"
    # Persistent storage configuration.
```

複数のキャッシュ

XML

```
<infinispan
      xmlns:xsi="http://www.w3.org/2001/XMLSchema-instance"
      xsi:schemaLocation="urn:infinispan:config:14.0 https://infinispan.org/schemas/infinispan-config-
14.0.xsd
                          urn:infinispan:server:14.0 https://infinispan.org/schemas/infinispan-server-14.0.xsd"
      xmlns="urn:infinispan:config:14.0"
      xmlns:server="urn:infinispan:server:14.0">
  <cache-container name="default"
                   statistics="true">
    <distributed-cache name="mycacheone"
                       mode="ASYNC"
                       statistics="true">
      <encoding media-type="application/x-protostream"/>
      <expiration lifespan="300000"/>
      <memory max-size="400MB"
              when-full="REMOVE"/>
    </distributed-cache>
    <distributed-cache name="mycachetwo"
                       mode="SYNC"
                       statistics="true">
      <encoding media-type="application/x-protostream"/>
      <expiration lifespan="300000"/>
      <memory max-size="400MB"
              when-full="REMOVE"/>
    </distributed-cache>
  </cache-container>
</infinispan>
```

JSON

```
{
  "infinispan" : {
    "cache-container" : {
      "name" : "default",
      "statistics" : "true",
      "caches" : {
        "mycacheone" : {
          "distributed-cache" : {
```

```
            "mode": "ASYNC",
            "statistics": "true",
            "encoding": {
              "media-type": "application/x-protostream"
            },
            "expiration" : {
              "lifespan" : "300000"
            },
            "memory": {
              "max-size": "400MB",
              "when-full": "REMOVE"
            }
          }
        },
        "mycachetwo" : {
          "distributed-cache" : {
            "mode": "SYNC",
            "statistics": "true",
            "encoding": {
              "media-type": "application/x-protostream"
            },
            "expiration" : {
              "lifespan" : "300000"
            },
            "memory": {
              "max-size": "400MB",
              "when-full": "REMOVE"
            }
          }
        }
      }
    }
  }
}
```

YAML

```
infinispan:
  cacheContainer:
    name: "default"
    statistics: "true"
    caches:
      mycacheone:
       distributedCache:
          mode: "ASYNC"
          statistics: "true"
          encoding:
            mediaType: "application/x-protostream"
          expiration:
            lifespan: "300000"
          memory:
            maxSize: "400MB"
            whenFull: "REMOVE"
      mycachetwo:
        distributedCache:
```

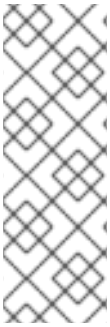
```
    mode: "SYNC"
    statistics: "true"
    encoding:
      mediaType: "application/x-protostream"
    expiration:
      lifespan: "300000"
    memory:
      maxSize: "400MB"
      whenFull: "REMOVE"
```

関連情報

- Data Grid 設定スキーマ参照
- infinispan-config-14.0.xsd

2.1.2. キャッシュの変更

ペイロードに XML または JSON 設定を含む **PUT** リクエストを使用して、Data Grid クラスター全体のキャッシュ設定の属性を変更します。



注記

変更が既存の設定と互換性がある場合にのみ、キャッシュを変更できます。

たとえば、レプリケートされたキャッシュ設定を使用して分散キャッシュを変更することはできません。同様に、特定の属性を使用してキャッシュ設定を作成する場合、代わりに別の属性を使用するように設定を変更することはできません。たとえば、**max-count** 属性の値を指定してキャッシュ設定を変更しようとすると、**max-size** がすでに設定されている場合、無効な設定になります。

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

表2.2 ヘッダー

Header	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

2.1.3. キャッシュの検証

HEAD リクエストを持つ Data Grid クラスターにキャッシュが存在するかどうかを確認します。

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

GET リクエストを使用してキャッシュの正常性を取得します。

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

2.1.4. 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リスト表示する を参照してください。

2.1.5. 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2.3 ヘッダー

Header	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。

2.1.6. XML、JSON、YAML 間のキャッシュ設定の変換

有効な設定と **?action=convert** パラメーターを使用して **POST** リクエストを呼び出します。Data Grid は、**Accept** ヘッダで指定されたタイプの設定の同等の表現で応答します。

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

キャッシュ設定を変換するには、**Content-Type** ヘッダーで設定の入力形式を指定し、**Accept** ヘッダーで目的の出力形式を指定する必要があります。たとえば、次のコマンドは、レプリケートされたキャッシュ設定を XML から YAML に変換します。

```
curl localhost:11222/rest/v2/caches?action=convert \
--digest -u username:password \
-X POST -H "Accept: application/yaml" -H "Content-Type: application/xml" \
-d '<replicated-cache mode="SYNC" statistics="false"><encoding media-type="application/x-protostream"/><expiration lifespan="300000" /><memory max-size="400MB" when-full="REMOVE"/></replicated-cache>'
```

2.1.7. キャッシュ設定の比較

2 つのキャッシュ設定と **?action=compare** パラメーターを含む **multipart/form-data** 本体で **POST** 要求を呼び出します。

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

ヒント

比較で変更可能な属性を無視するには、**ignoreMutable=true** パラメーターを追加します。

Data Grid は、設定が等しい場合は **204 (No Content)** で応答し、設定が異なる場合は **409 (Conflict)** で応答します。

2.1.8. 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}?action=stats
```

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,
  "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,
```

```
  "rebalancing_enabled": true,
  "bounded": false,
  "indexed": false,
  "persistent": false,
  "transactional": false,
  "secured": false,
  "has_remote_backup": false,
  "indexing_in_progress": false,
  "statistics": false,
  "mode" : "DIST_SYNC",
  "storage_type": "HEAP",
  "max_size": "",
  "max_size_bytes" : -1
}
```


- **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。
- **rebalancing_enabled** は、リバランシングが有効な場合は true。このプロパティーの取得は、サーバーで失敗する可能性があります。その場合、プロパティーはペイロードに存在しません。
- **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。
- **key_storage** キャッシュキーのメディアタイプです。
- **value_storage** キャッシュの値のメディアタイプです。

注記

key_storage と **value_storage** は、キャッシュのエンコーディング設定と一致します。エンコーディングなしのサーバーキャッシュの場合、Data Grid はキャッシュがインデックス化されている場合は **application/x-protostream** を、それ以外の場合は **application/unknown** を想定しています。

2.1.9. すべてのキャッシュ統計のリセット

POST 要求を呼び出して、Data Grid キャッシュのすべての統計をリセットします。

```
POST /rest/v2/caches/{name}?action=stats-reset
```

2.1.10. キャッシュのデータ分散の取得

GET 要求を呼び出して、Data Grid キャッシュのデータ配布に関するすべての詳細を取得します。

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

Data Grid は、以下のような JSON レスポンスを提供します。

```
[
  {
    "node_name": "NodeA",
    "node_addresses": [
      "127.0.0.1:44175"
    ],
    "memory_entries": 0,
    "total_entries": 0,
    "memory_used": 528512
  },
  {
    "node_name":"NodeB",
    "node_addresses": [
      "127.0.0.1:44187"
    ],
    "memory_entries": 0,
    "total_entries": 0,
    "memory_used": 528512
  }
]
```

リスト内の各要素はノードを表します。プロパティーは次のとおりです。

- **node_name** はノード名です
- **node_addresses** は、すべてのノードの物理アドレスのリストです。
- **memory_entries** ノードがキャッシュに属するメモリーに保持するエントリーの数。
- **total_entries** ノードがキャッシュに属するメモリーとディスクに持つエントリーの数。
- **memory_used** は、エビクションアルゴリズムがキャッシュの占有量を推定するバイト単位の値です。エヴィクションが有効になっていない場合は -1 を返します。

2.1.11. すべての可変キャッシュ設定属性の取得

GET 要求を呼び出して、Data Grid キャッシュのすべての可変キャッシュ設定属性を取得します。

```
GET /rest/v2/caches/{name}?action=get-mutable-attributes
```

Data Grid は、以下のような JSON レスポンスを提供します。

```
[
  "jmx-statistics.statistics",
```

```
  "locking.acquire-timeout",
  "transaction.single-phase-auto-commit",
  "expiration.max-idle",
  "transaction.stop-timeout",
  "clustering.remote-timeout",
  "expiration.lifespan",
  "expiration.interval",
  "memory.max-count",
  "memory.max-size"
]
```

値や型の情報を得るには、**full** パラメーターを追加します。

```
GET /rest/v2/caches/mycache?action=get-mutable-attributes&full=true
```

Data Grid は、以下のような JSON レスポンスを提供します。

```
{
  "jmx-statistics.statistics": {
    "value": true,
    "type": "boolean"
  },
  "locking.acquire-timeout": {
    "value": 15000,
    "type": "long"
  },
  "transaction.single-phase-auto-commit": {
    "value": false,
    "type": "boolean"
  },
  "expiration.max-idle": {
    "value": -1,
    "type": "long"
  },
  "transaction.stop-timeout": {
    "value": 30000,
    "type": "long"
  },
  "clustering.remote-timeout": {
    "value": 17500,
    "type": "long"
  },
  "expiration.lifespan": {
    "value": -1,
    "type": "long"
  },
  "expiration.interval": {
    "value": 60000,
    "type": "long"
  },
  "memory.max-count": {
    "value": -1,
    "type": "long"
  },
  "memory.max-size": {
```

```
    "value": null,
    "type": "string"
  }
}
```

enum 型の属性の場合、追加の **universe** プロパティーには、可能な値のセットが含まれます。

2.1.12. キャッシュ設定属性の更新

変更可能なキャッシュ設定属性を変更するために、**POST** リクエストを呼び出します。

```
POST /rest/v2/caches/{name}?action=set-mutable-attributes&attribute-name=
{attributeName}&attribute-value={attributeValue}
```

2.1.13. 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。

エントリーが正常に作成されると、サービスは **204 (No Content)** を返します。

指定されたキーにすでに値が存在する場合、**POST** リクエストは **409 (Conflict)** を返し、値の変更は行いません。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。

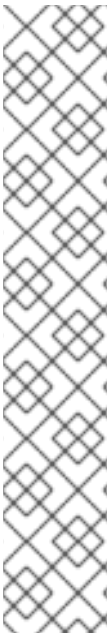
表2.4 ヘッダー

Header	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。

Header	必須またはオプション	パラメーター
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。

2.1.14. 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、**409 (Conflict)** を返す **POST** リクエストを使用してください。値の追加 を参照してください。

2.1.15. 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2.5 ヘッダー

Header	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:

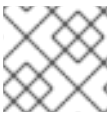
- **Cluster-Primary-Owner** は、キーのプライマリー所有者であるノード名を返します。
- **Cluster-Node-Name** は、要求を処理したサーバーの JGroups ノード名を返します。
- **Cluster-Physical-Address** は、要求を処理したサーバーの物理 JGroups アドレスを返します。

2.1.16. 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2.6 ヘッダー

Header	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。

2.1.17. 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2.7 ヘッダー

Header	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。

2.1.18. キャッシュエントリーの分散の確認

このエンドポイントを呼び出して、Data Grid キャッシュエントリーのデータディストリビューションの詳細を取得します。

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

Data Grid は、以下のような JSON レスポンスを提供します。

```
{
  "contains_key": true,
  "owners": [
    {
      "node_name": "NodeA",
      "primary": true,
      "node_addresses": [
        "127.0.0.1:39492"
      ]
    },
    {
      "node_name": "NodeB",
      "primary": false,
      "node_addresses": [
        "127.0.0.1:38195"
      ]
    }
  ]
}
```

- キャッシュにキーが含まれる場合、**contains_key** は **true** を返します。
- **所有者** は、キーを含むノードの一覧を提供します

所有者の一覧には、以下のプロパティーが含まれます。

- **node_name** はノードの名前を示します
- **プライマリー** はプライマリー所有者であるノードを識別します
- **node_addresses** は、ノードにアクセスできる IP アドレスとポートを示します。

2.1.19. 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}
```

2.1.20. 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2.8 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
limit	オプション	InputStream を使用して取得するキーの最大数を指定します。負の値はすべてのキーを取得します。デフォルト値は **-1** です。

パラメーター	必須またはオプション	値
batch	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。

2.1.21. キャッシュからのすべてのエントリーの取得

GET リクエストを呼び出し、キャッシュ内のすべてのエントリーを JSON 形式で取得します。

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

表2.9 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
metadata	オプション	応答の各エントリーのメタデータが含まれます。デフォルト値は **false** です。
limit	オプション	応答に含めるキーの最大数を指定します。負の値はすべてのキーを取得します。デフォルト値は **-1** です。
batch	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。
content-negotiation	オプション	**true** の場合、キーと値を読み取り可能な形式に変換します。テキストエンコーディングのキャッシュ (text/plain、xml、json など) では、サーバーはキーと値をプレーンテキストで返します。バイナリーエンコーディングのキャッシュの場合、変換がサポートされている場合、サーバーはエントリーを JSON として返します。サポートされていない場合は、テキストの 16 進形式 (**0xA123CF98** など) で返します。content-negotiation が使用される場合、応答には **key-content-type** と **value-content-type** の 2 つのヘッダが含まれ、ネゴシエートされたフォーマットが記述されます。

Data Grid は、以下のような JSON レスポンスを提供します。

```
[
   {
      "key": 1,
      "value": "value1",
      "timeToLiveSeconds": -1,
      "maxIdleTimeSeconds": -1,
      "created": -1,
      "lastUsed": -1,
      "expireTime": -1
   },
   {
      "key": 2,
      "value": "value2",
      "timeToLiveSeconds": 10,
      "maxIdleTimeSeconds": 45,
      "created": 1607966017944,
      "lastUsed": 1607966017944,
      "expireTime": 1607966027944,
      "version": 7
   },
   {
      "key": 3,
      "value": "value2",
      "timeToLiveSeconds": 10,
      "maxIdleTimeSeconds": 45,
      "created": 1607966017944,
      "lastUsed": 1607966017944,
      "expireTime": 1607966027944,
      "version": 7,
      "topologyId": 9
   }
]
```

- **key** エントリーのキー。
- **value** エントリーの値。
- **timeToLiveSeconds** エントリーの有効期間に基づきますが、秒単位です。エントリーが期限切れにならない場合は **-1** です。metadata = "true"を設定しない限り、返されません。
- **maxIdleTimeSeconds** 最大アイドル時間 (秒単位)。エントリーが期限切れにならない場合は **-1**。metadata = "true"を設定しない限り、返されません。
- 作成済みエントリーが **created** された時刻、または不滅のエントリーの場合は **-1**。metadata = "true"を設定しない限り、返されません。
- **lastUsed** エントリーに対して操作が行われた最後の時刻、または不滅のエントリーでは **-1**。metadata = "true"を設定しない限り、返されません。
- **expireTime** エントリーが期限切れになる時間、または不死身のエントリーの場合は **-1**。metadata = "true"を設定しない限り、返されません。
- **version** キャッシュエントリーに関連するメタデータバージョン。値が存在する場合にのみます。

- **topologyId** クラスター化されたバージョンメタデータのトポロジー ID。値が存在する場合にのみます。

2.1.22. 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、**POST** リクエストに **?action=clear** パラメーターを付けて実行します。

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

操作が正常に完了すると、サービスは **204 (No Content)** を返します。

2.1.23. 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
```

2.1.24. 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
```

2.1.25. キャッシュのリスト表示

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リスト表示します。

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

2.1.26. 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/caches?action=detailed
```

Data Grid は、次の例のように、使用可能な各キャッシュをリスト表示して説明する 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",
  "rebalancing_enabled": true
}, {
  "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",
  "rebalancing_enabled": false
}]
```

表2.10 要求パラメーター

パラメーター	必須またはオプション	説明
pretty	オプション	**true** の場合は、スペースや行区切りが追加された、フォーマット済みのコンテンツが返されます。これは、読みやすくはなりますが、パイロードのサイズが増えます。デフォルトは **false** です。

2.1.27. ロールのアクセス可能なキャッシュの一覧表示

セキュリティーが有効な場合は、ロールのアクセス可能なすべてのキャッシュのリストを取得します。この操作には ADMIN 権限が必要です。

```
GET /rest/v2/caches?action=role-accessible&role=observer
```

Data Grid は、次の例のように JSON で応答します。

```
{
  "secured" : ["securedCache1", "securedCache2"],
  "non-secured" : ["cache1", "cache2", "cache3"]
}
```

表2.11 要求パラメーター

パラメーター	必須またはオプション	説明

パラメーター	必須またはオプション	説明
pretty	オプション	**true** の場合は、スペースや行区切りが追加された、フォーマット済みのコンテンツが返されます。これは、読みやすくはなりますが、パイロードのサイズが増えます。デフォルトは **false** です。

2.1.28. キャッシュイベントをリッスンする

Server-Sent Events でキャッシュイベントを受信する。**event** 値は、**cache-entry-created**、**cache-entry-removed**、**cache-entry-updated**、**cache-entry-expired** のいずれかになります。**data** 値には、**Accept** ヘッダーで設定された形式でイベントを発生させたエントリーのキーが含まれます。

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

表2.12 ヘッダー

Header	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。サポートされている形式は、 **text/plain** および **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。

2.1.29. リバランスの有効化

特定のキャッシュの自動リバランスをオンにします。

```
POST /rest/v2/caches/{cacheName}?action=enable-rebalancing
```

2.1.30. リバランスの無効化

特定のキャッシュの自動リバランスをオフにします。

```
POST /rest/v2/caches/{cacheName}?action=disable-rebalancing
```

2.1.31. キャッシュの可用性の取得

キャッシュの可用性を取得します。

```
GET /rest/v2/caches/{cacheName}?action=get-availability
```

注記

内部キャッシュの可用性を取得できますが、これは将来の Data Grid バージョンで変更される可能性があります。

2.1.32. キャッシュの可用性の設定

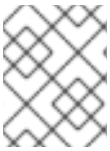
DENY_READ_WRITES または ALLOW_READS パーティション処理戦略のいずれかを使用する場合は、クラスター化されたキャッシュの可用性を変更します。

```
POST /rest/v2/caches/{cacheName}?action=set-availability&availability={AVAILABILITY}
```

表2.13 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
availability	必須	AVAILABLE または DEGRADED_MODE

- **AVAILABLE** は、ネットワークパーティション内のすべてのノードでキャッシュを利用できるようにします。
- **DEGRADED_MODE** は、ネットワークパーティションが発生したときにキャッシュの読み取りおよび書き込み操作を防止します。



注記

内部キャッシュの可用性を設定できますが、これは将来の Data Grid バージョンで変更される可能性があります。

2.1.33. stable トポロジーの設定

デフォルトでは、クラスターのシャットダウン後、Data Grid はすべてのノードがクラスターに参加してトポロジーを復元するまで待機します。ただし、REST 操作を使用して、特定のキャッシュに対して現在のクラストポロジーを安定していると定義することは可能です。

```
POST /rest/v2/caches/{cacheName}?action=initialize&force={FORCE}
```

表2.14 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
force	オプション	true または false

- 現在のトポロジーで欠落しているノードの数が所有者の数以上である場合は、**force** が必要です。

重要

トポロジーを手動でインストールすると、データが失われる可能性があります。この操作は、初期トポロジーを再作成できない場合にのみ実行してください。

2.1.34. RESTAPI を使用したインデックス作成とクエリー

HTTP クライアントから **GET** リクエストと **?action=search&query** パラメーターを使用して、リモートキャッシュにクエリーを実行します。

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

Data Grid の応答

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

- **hit_count** は、クエリーの結果の合計数を示します。
- **hit_count_exact** は **true** で、**hit_count** が正確であることを意味します。**false** の場合、ヒットカウントの値が下限であることを意味します。
- **hits** は、クエリーからの個々の一致の配列を表します。
- **hit** は、クエリー内の一致に対応する各オブジェクトを参照します。

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2.15 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
query	必須	クエリー文字列を指定します。
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。

パラメーター	必須またはオプション	値
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。
hit_count_accuracy	オプション	インデックス付きクエリーのヒット数の必要な精度に上限を設定します。デフォルトは / です。**query.hit-count-accuracy** キャッシュプロパティーを設定することで、デフォルトの制限を変更できます。
local	オプション	**true** の場合、クエリーは、リクエストを処理するノードに存在するデータに制限されます。デフォルトは **false** 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
```

リクエスト本文のクエリー

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

2.1.34.1. インデックスの再構築

フィールドを削除するか、インデックスフィールド定義を変更する場合は、インデックスを再構築して、インデックスがキャッシュ内のデータと一致していることを確認する必要があります。

注記

REST、CLI、Data Grid Console、またはリモートクライアントを使用して Protobuf スキーマを再構築すると、不整合が発生する可能性があります。リモートクライアントには Protostream エンティティーのバージョンが異なる場合があり、信頼性の低い動作が発生する可能性があります。

POST リクエストと **?action=reindex** 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。

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

表2.16 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
mode	オプション	**mode** パラメーターの値は以下の通りです。 **sync** は、インデックス再作成の操作が完了した後にのみ、**204 (No Content)** を返します。 **async** はすぐに **204 (No Content)** を返し、再インデックス化処理はクラスター内で継続して実行されます。Index StatisticsREST コールで状態を確認できます。
local	オプション	**true** の場合、リクエストを処理するノードからのデータのみが再インデックス付けされます。デフォルトは **false** で、クラスター全体のデータが再インデックス化されます。

2.1.34.2. インデックススキーマの更新

インデックススキーマの更新操作を使用すると、最小限のダウンタイムでスキーマの変更を追加できます。Data Grid は、以前にインデックス化されたデータを削除してインデックススキーマを再作成する代わりに、新しいフィールドを既存のスキーマに追加します。

POST リクエストと **?action=updateSchema** パラメーターを使用して、キャッシュ内の値のインデックススキーマを更新します。

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

2.1.34.3. インデックスのパージ

POST 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。

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

操作が正常に完了すると、サービスは **204 (No Content)** を返します。

2.1.34.4. インデックスのメタモデル取得

このキャッシュで定義されたすべてのインデックスの完全なインデックススキーマメタモデルを提示します。

```
GET /rest/v2/caches/{cacheName}/search/indexes/metamodel
```

Data Grid の応答

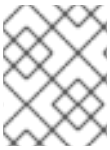
```
[{
    "entity-name": "org.infinispan.query.test.Book",
    "java-class": "org.infinispan.query.test.Book",
    "index-name": "org.infinispan.query.test.Book",
    "value-fields": {
        "description": {
            "multi-valued": false,
            "multi-valued-in-root": false,
            "type": "java.lang.String",
            "projection-type": "java.lang.String",
            "argument-type": "java.lang.String",
            "searchable": true,
            "sortable": false,
            "projectable": false,
            "aggregable": false,
            "analyzer": "standard"
        },
        "name": {
            "multi-valued": false,
            "multi-valued-in-root": true,
            "type": "java.lang.String",
            "projection-type": "java.lang.String",
            "argument-type": "java.lang.String",
            "searchable": true,
            "sortable": false,
            "projectable": false,
            "aggregable": false,
            "analyzer": "standard"
        },
        "surname": {
            "multi-valued": false,
            "multi-valued-in-root": true,
            "type": "java.lang.String",
            "projection-type": "java.lang.String",
            "argument-type": "java.lang.String",
            "searchable": true,
            "sortable": false,
            "projectable": false,
            "aggregable": false
        },
        "title": {
            "multi-valued": false,
            "multi-valued-in-root": false,
            "type": "java.lang.String",
            "projection-type": "java.lang.String",
            "argument-type": "java.lang.String",
            "searchable": true,
            "sortable": false,
            "projectable": false,
            "aggregable": false
        }
    },
    "object-fields": {
        "authors": {
            "multi-valued": true,
            "multi-valued-in-root": true,
```

```
            "nested": true,
            "value-fields": {
               "name": {
                  "multi-valued": false,
                  "multi-valued-in-root": true,
                  "type": "java.lang.String",
                  "projection-type": "java.lang.String",
                  "argument-type": "java.lang.String",
                  "searchable": true,
                  "sortable": false,
                  "projectable": false,
                  "aggregable": false,
                  "analyzer": "standard"
               },
               "surname": {
                  "multi-valued": false,
                  "multi-valued-in-root": true,
                  "type": "java.lang.String",
                  "projection-type": "java.lang.String",
                  "argument-type": "java.lang.String",
                  "searchable": true,
                  "sortable": false,
                  "projectable": false,
                  "aggregable": false
               }
            }
         }
      }
   }, {
      "entity-name": "org.infinispan.query.test.Author",
      "java-class": "org.infinispan.query.test.Author",
      "index-name": "org.infinispan.query.test.Author",
      "value-fields": {
         "surname": {
            "multi-valued": false,
            "multi-valued-in-root": false,
            "type": "java.lang.String",
            "projection-type": "java.lang.String",
            "argument-type": "java.lang.String",
            "searchable": true,
            "sortable": false,
            "projectable": false,
            "aggregable": false
         },
         "name": {
            "multi-valued": false,
            "multi-valued-in-root": false,
            "type": "java.lang.String",
            "projection-type": "java.lang.String",
            "argument-type": "java.lang.String",
            "searchable": true,
            "sortable": false,
            "projectable": false,
            "aggregable": false,
            "analyzer": "standard"
```

```
      }
    }
}]
```

2.1.34.5. クエリーおよびインデックス統計の取得

GET 要求を使用して、キャッシュでクエリーとインデックスに関する情報を取得します。



注記

キャッシュ設定で統計を有効にする必要があります。有効にしないと、結果が空になります。

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

表2.17 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
scope	オプション	クラスターの全メンバーの連結統計情報を取得するには、**cluster** を使用します。省略すると、Data Grid はローカルのクエリーとインデックスの統計情報を返します。

Data Grid の応答

```
{
  "query": {
    "indexed_local": {
      "count": 1,
      "average": 12344.2,
      "max": 122324,
      "slowest": "FROM Entity WHERE field > 4"
    },
    "indexed_distributed": {
      "count": 0,
      "average": 0.0,
      "max": -1,
      "slowest": "FROM Entity WHERE field > 4"
    },
    "hybrid": {
      "count": 0,
      "average": 0.0,
      "max": -1,
      "slowest": "FROM Entity WHERE field > 4 AND desc = 'value'"
    },
    "non_indexed": {
      "count": 0,
      "average": 0.0,
      "max": -1,
      "slowest": "FROM Entity WHERE desc = 'value'"
```

```
    },
    "entity_load": {
      "count": 123,
      "average": 10.0,
      "max": 120
    }
  },
  "index": {
    "types": {
      "org.infinispan.same.test.Entity": {
        "count": 5660001,
        "size": 0
      },
      "org.infinispan.same.test.AnotherEntity": {
        "count": 40,
        "size": 345560
      }
    },
    "reindexing": false
  }
}
```


query のセクション

- **indexed_local** インデックス付きのクエリーの詳細を提供します。
- **indexed_distributed** 分散したインデックス付きクエリーの詳細を提供します。
- **hybrid** インデックスを部分的にしか使用していないクエリーの詳細を提供します。
- **non_indexed** インデックスを使用しなかったクエリーの詳細を提供します。
- **entity_load** インデックス付きクエリーの実行後にオブジェクトをフェッチするためのキャッシュ操作に関する詳細を提供します。

注記

時間は常にナノ秒単位で測定されます。

index のセクション

- **types** キャッシュに設定されている各インデックスタイプ (クラス名や protobuf メッセージ) の詳細を提供する。
  - **count** そのタイプにインデックスされているエンティティーの数。
  - **size** 型の使用量 (バイト)。
- **reindexing** 値が **true** の場合、**Indexer** はキャッシュで動作しています。

2.1.34.6. クエリー統計のクリア

POST 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。

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

Data Grid は、ローカルノードのクエリー実行時間のみをリセットします。この操作では、インデックス統計はクリアされません。

2.1.34.7. インデックス統計の取得 (非推奨)

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

Data Grid の応答

```
{
   "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。

2.1.34.8. クエリー統計の取得 (非推奨)

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

Data Grid の応答

```
{
   "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。

2.1.34.9. クエリー統計情報のクリア (非推奨)

POST 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。

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

2.1.35. 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。

2.1.35.1. 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 /rest/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": {
    "status": "online"
  },
  "LON": {
    "status": "mixed",
    "online": [
      "NodeA"
```

```
    ],
    "offline": [
      "NodeB"
    ]
  }
}
```

表2.18 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。

2.1.35.2. 特定のバックアップの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 /rest/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2.19 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。

2.1.35.3. バックアップの場所をオフラインにする

POST 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。

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

2.1.35.4. バックアップの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。

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

2.1.35.5. 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。

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

2.1.35.6. 状態遷移のキャンセル

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

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

2.1.35.7. 状態遷移ステータスの取得

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

```
GET /rest/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2.20 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遷移が進行中です。
OK	状態の転送が正常に完了しました。
ERROR	状態遷移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。

2.1.35.8. 状態遷移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遷移ステータスをクリアします。

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

2.1.35.9. オフラインにする条件の変更

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

   操作が正常に完了すると、サービスは **204 (No Content)** を返します。

2.1.35.10. 受信サイトからの状態遷移のキャンセル

2 つのバックアップ場所間の接続が切断された場合は、プッシュを受信しているサイトでの状態遷移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。

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

2.1.36. 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

2.1.36.1. ソースクラスターの接続

ターゲット・クラスターとソース・クラスターの接続は

```
POST /rest/v2/caches/{cacheName}/rolling-upgrade/source-connection
```

本文として JSON 形式の **remote-store** 定義を提供する必要があります。

JSON

```
{
  "remote-store": {
    "cache": "my-cache",
    "shared": true,
    "raw-values": true,
    "socket-timeout": 60000,
    "protocol-version": "2.9",
    "remote-server": [
      {
        "host": "127.0.0.2",
        "port": 12222
      }
    ],
    "connection-pool": {
      "max-active": 110,
      "exhausted-action": "CREATE_NEW"
    },
    "async-executor": {
      "properties": {
        "name": 4
      }
    },
    "security": {
      "authentication": {
        "server-name": "servername",
        "digest": {
          "username": "username",
          "password": "password",
          "realm": "realm",
          "sasl-mechanism": "DIGEST-MD5"
        }
      },
      "encryption": {
        "protocol": "TLSv1.2",
        "sni-hostname": "snihostname",
        "keystore": {
          "filename": "/path/to/keystore_client.jks",
          "password": "secret",
          "certificate-password": "secret",
          "key-alias": "hotrod",
          "type": "JKS"
        },
        "truststore": {
          "filename": "/path/to/gca.jks",
          "password": "secret",
          "type": "JKS"
        }
      }
    }
  }
}
```

security 、 **async-executor** 、 **connection-pool** など、いくつかの要素はオプションです。この設定には、最低限、キャッシュ名、**false** に設定された **raw-values**、ソースクラスター内の単一ポートのホスト/IP が含まれていなければなりません。**remote-store** 設定の詳細については、XSD Schema を参照してください。

操作が正常に完了した場合、サービスは 204(No Content) を返します。ターゲット・クラスターがソース・クラスターにすでに接続されている場合は、ステータス 304(Not Modified) を返します。

2.1.36.2. ソースクラスター接続の詳細の取得

キャッシュの **remote-store** 定義を取得するには、 **GET** リクエストを使用します。

```
GET /rest/v2/caches/{cacheName}/rolling-upgrade/source-connection
```

キャッシュが以前に接続されていた場合は、関連付けられた **remote-store** の設定を JSON 形式とステータス 200(OK) で返します。それ以外の場合は、404(見つかりません) ステータスを返します。

注記

これはクラスター全体の操作ではなく、REST の呼び出しが処理されたノードのキャッシュのリモートストアのみを返します。

2.1.36.3. キャッシュが接続されているかどうかの確認

キャッシュがリモートクラスターに接続されているかどうかを確認するには、**HEAD** リクエストを使用します。

```
HEAD /rest/v2/caches/{cacheName}/rolling-upgrade/source-connection
```

クラスターのすべてのノードにおいて、**cacheName** に 1 つのリモートストアが設定されている場合は 200(OK)、そうでない場合は 404(NOT_FOUND) を返します。

2.1.36.4. 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、**POST** リクエストと **?action=sync-data** パラメーターを使用します。

```
POST /rest/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。

2.1.36.5. ソースクラスターの接続

ターゲット・クラスターにデータを同期させた後、**DELETE** リクエストでソース・クラスターから切断します。

```
DELETE /rest/v2/caches/{cacheName}/rolling-upgrade/source-connection
```

操作が正常に完了すると、サービスは **204 (No Content)** を返します。ソースが接続されていない場合、コード 304(変更なし) を返します。

2.2. カウンターの作成と管理

REST API を使用し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2.21 ヘッダー

Header	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。

2.2.5. カウンターのリセット

POST リクエストと **?action=reset** パラメーターを使用せずに、カウンターの初期値を復元することができます。

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

操作が正常に完了すると、サービスは **204 (No Content)** を返します。

2.2.6. カウンターのインクリメント

POST 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。

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

注記

WEAK カウンターは操作後に応答せず、**204 (No Content)** を返します。

STRONG カウンターは、各操作後に **200 (OK)** と現在値を返します。

2.2.7. 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 **POST** リクエストで、カウンターに任意の値を追加します。

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

注記

WEAK カウンターは操作後に応答せず、**204 (No Content)** を返します。

STRONG カウンターは、各操作後に **200 (OK)** と現在値を返します。

2.2.8. 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、**POST** リクエストと **?action=decrement** パラメーターを使用します。

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

注記

WEAK カウンターは操作後に応答せず、**204 (No Content)** を返します。

STRONG カウンターは、各操作後に **200 (OK)** と現在値を返します。

2.2.9. 強力なカウンターでの getAndSet アトミック操作の実行

POST リクエストと **getAndSet** パラメーターを使用して、強力なカウンターの値をアトミックに設定します。

```
POST /rest/v2/counters/{counterName}?action=getAndSet&value={value}
```

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。

2.2.10. 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** パラメーターを使用して、強力なカウンターの値をアトミックに設定します。

```
POST /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。

```
POST /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. カウンターのリスト表示

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. PROTOBUF スキーマの操作

Data Grid REST API を利用して、Protobuf スキーマ **.proto** を作成・管理することができます。

2.3.1. Protobuf スキーマの作成

ペイロードに protobuf ファイルのコンテンツを含む **POST** リクエストで、Data Grid クラスター全体に Protobuf スキーマを作成します。

```
POST /rest/v2/schemas/{schemaName}
```

スキーマがすでに存在する場合、Data Grid は HTTP**409 (Conflict)** を返します。構文エラーのため、またはその依存関係の一部が欠落しているためにスキーマが有効でない場合、Data Grid はスキーマを格納し、応答本文にエラーを返します。

Data Grid は、スキーマ名とエラーで応答します。

```
{
  "name" : "users.proto",
  "error" : {
    "message": "Schema users.proto has errors",
    "cause": "java.lang.IllegalStateException:Syntax error in error.proto at 3:8: unexpected label:
messoge"
  }
}
```

- **name** は Protobuf スキーマの名前です。
- **error** は、有効な Protobuf スキーマでは **null** です。Data Grid がスキーマを正常に検証できない場合、エラーを返します。

操作が正常に完了すると、サービスは **201 (Created)** を返します。

2.3.2. Protobuf スキーマの読み取り

GET リクエストで Data Grid から Protobuf スキーマを取得します。

```
GET /rest/v2/schemas/{schemaName}
```

2.3.3. Protobuf スキーマの更新

ペイロードに protobuf ファイルのコンテンツを含む **PUT** リクエストで Protobuf スキーマを変更する。

重要

既存の Protobuf スキーマ定義を変更する場合は、インデックススキーマを更新または再構築する必要があります。変更に既存のフィールドの変更が含まれる場合は、インデックスを再構築する必要があります。既存のスキーマを変更せずに新しいフィールドを追加する場合は、再構築の代わりにインデックススキーマを更新できます。

```
PUT /rest/v2/schemas/{schemaName}
```

構文エラーのため、またはその依存関係の一部が欠落しているためにスキーマが有効でない場合、Data Grid はスキーマを更新し、応答本文にエラーを返します。

```
{
  "name" : "users.proto",
  "error" : {
    "message": "Schema users.proto has errors",
    "cause": "java.lang.IllegalStateException:Syntax error in error.proto at 3:8: unexpected label:
```

```
messoge"
  }
}
```

- **name** は Protobuf スキーマの名前です。
- **error** は、有効な Protobuf スキーマでは **null** です。Data Grid がスキーマを正常に検証できない場合、エラーを返します。

2.3.4. Protobuf スキーマの削除

DELETE リクエストで Data Grid クラスターから Protobuf スキーマを削除します。

```
DELETE /rest/v2/schemas/{schemaName}
```

操作が正常に完了すると、サービスは **204 (No Content)** を返します。

2.3.5. Protobuf スキーマのリスト表示

GET リクエストで利用可能なすべての Protobuf スキーマをリストアップします。

```
GET /rest/v2/schemas/
```

Data Grid は、クラスターで使用可能なすべてのスキーマのリストで応答します。

```
[ {
  "name" : "users.proto",
  "error" : {
    "message": "Schema users.proto has errors",
    "cause": "java.lang.IllegalStateException:Syntax error in error.proto at 3:8: unexpected label:
messoge"
  }
}, {
  "name" : "people.proto",
  "error" : null
}]
```

- **name** は Protobuf スキーマの名前です。
- **error** は、有効な Protobuf スキーマでは **null** です。Data Grid がスキーマを正常に検証できない場合、エラーを返します。

2.3.6. Protobuf タイプの一覧表示

GET 要求で使用可能なすべての Protobuf タイプを一覧表示します。

```
GET /rest/v2/schemas?action=types
```

Data Grid は、クラスターで使用可能なすべてのタイプのリストで応答します。

```
["org.infinispan.Person", "org.infinispan.Phone"]
```

2.4. 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。

2.4.1. 基本的なコンテナー情報の取得

GET リクエストでキャッシュマネージャーの情報を取得します。

```
GET /rest/v2/container
```

Data Grid は、次の例のように、JSON 形式の情報で応答します。

注記

セキュリティー承認のあるキャッシュに関する情報は、特定のロールとパーミッションが割り当てられているユーザーのみが利用できます。


```
{
   "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
        }
    ],
    "local_site": "LON",
    "relay_node": true,
    "relay_nodes_address": [
      "CacheManagerResourceTest-NodeA-49696"
    ],
    "sites_view": [
        "LON",
        "NYC"
    ],
    "rebalancing_enabled": 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 Cache Manager の名前が含まれます。
- **coordinator** は、Cache Manager 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、現在のユーザーがアクセスできる、Cache Manager 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、Cache Manager 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** Cache Manager 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、Cache Manager 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** Cache Manager 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。
- **local_site** ローカルサイトの名前。
  クロスサイトレプリケーションが設定されていない場合、Data Grid は "local "を返します。

- **relay_node** は、クラスター間の RELAY メッセージを処理するノードであれば true。
- **relay_nodes_address** は、リレーノードの論理アドレスの配列です。
- **sites_view** クロスサイトレプリケーションに参加しているサイトのリスト。
  クロスサイトレプリケーションが設定されていない場合、 Data Grid は空のリストを返します。
- **rebalancing_enabled** は、リバランシングが有効な場合は true。このプロパティーの取得は、サーバーで失敗する可能性があります。その場合、プロパティーはペイロードに存在しません。

2.4.2. 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/container/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。
- **FAILED** は、指定された設定でキャッシュを開始できなかったことを示します。

- **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** は HEALTHY、DEGRADED、HEALTHY_REBALANCING または FAILED です。
  - **cache_name** 設定で定義されているキャッシュの名前。

2.4.3. コンテナ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、Data Grid コンテナーの正常性ステータスを取得します。

```
GET /rest/v2/container/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**
- **FAILED**

2.4.4. 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/container/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。

2.4.5. グローバル設定の取得

GET リクエストを使用してデータコンテナーのグローバル設定を取得します。

```
GET /rest/v2/container/config
```

表2.22 ヘッダー

Header	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。

表2.23 要求パラメーター

パラメーター	必須またはオプション	説明
pretty	オプション	**true** の場合は、スペースや行区切りが追加された、フォーマット済みのコンテンツが返されます。これは、読みやすくはなりますが、パイロードのサイズが増えます。デフォルトは **false** です。

参照

GlobalConfiguration

2.4.6. 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/container/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"
         }
      }
   }
}
]
```

表2.24 要求パラメーター

パラメーター	必須またはオプション	説明
pretty	オプション	**true** の場合は、スペースや行区切りが追加された、フォーマット済みのコンテンツが返されます。これは、読みやすくはなりますが、パイロードのサイズが増えます。デフォルトは **false** です。

2.4.7. 利用可能なキャッシュテンプレートのリスト表示

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-configs/templates
```

ヒント

テンプレートを使用したキャッシュの作成 を参照してください。

表2.25 要求パラメーター

パラメーター	必須またはオプション	説明
pretty	オプション	**true** の場合は、スペースや行区切りが追加された、フォーマット済みのコンテンツが返されます。これは、読みやすくはなりますが、パイロードのサイズが増えます。デフォルトは **false** です。

2.4.8. コンテナー統計の取得

GET リクエストを使用してコンテナーの統計を取得します。

```
GET /rest/v2/container/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,
  "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。
- **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。

2.4.9. コンテナー統計のリセット

POST リクエストで統計をリセットします。

```
POST /rest/v2/container/stats?action=reset
```

2.4.10. すべてのコンテナーキャッシュをシャットダウンします

POST リクエストを使用して、サーバー上の Data Grid コンテナーをシャットダウンします。

```
POST /rest/v2/container?action=shutdown
```

Data Grid は **204 (No Content)** と応答し、コンテナー内のすべてのキャッシュをシャットダウンします。サーバーはアクティブなエンドポイントとクラスタリングで実行されたままですが、コンテナーリソースへの REST 呼び出しは、503 Service Unavailable 応答になります。

注記

このメソッドは、主に Data Grid オペレーターによる使用を目的としています。このエンドポイントが呼び出された直後に、サーバープロセスが手動で終了することが期待されます。このメソッドが呼び出されると、コンテナーの状態を再開することはできません。

2.4.11. すべてのキャッシュのリバランスを有効にする

すべてのキャッシュの自動リバランスをオンにします。

```
POST /rest/v2/container?action=enable-rebalancing
```

2.4.12. すべてのキャッシュのリバランスを無効にする

すべてのキャッシュの自動リバランスをオフにします。

```
POST /rest/v2/container?action=disable-rebalancing
```

2.4.13. Data Grid のバックアップ

現在 Cache Manager に保存されているリソース (キャッシュ、キャッシュテンプレート、カウンター、Protobuf スキーマ、サーバータスクなど) を含むバックアップアーカイブ (**application/zip**) を作成します。

```
POST /rest/v2/container/backups/{backupName}
```

同じ名前のバックアップがすでに存在する場合、サービスは **409 (Conflict)** 応答します。**directory** パラメーターが無効な場合、サービスは **400 (Bad Request)** 返します。**202** 応答は、バックアップ要求が処理のために受け入れられたことを示します。

オプションで、次のように、バックアップ操作のパラメーターを含む JSON ペイロードをリクエストに含めます。

表2.26 JSON パラメーター

キー	必須またはオプション	値
directory	オプション	バックアップアーカイブを作成および保存するサーバー上の場所を指定します。
resources	オプション	バックアップするリソースを JSON 形式で指定します。デフォルトでは、すべてのリソースがバックアップされます。1 つまたは複数のリソースを指定した場合、Data Grid はそれらのリソースのみをバックアップします。詳細については、**リソースパラメーター**の表を参照してください。

表2.27 リソースパラメーター

キー	必須またはオプション	値
caches	オプション	バックアップするキャッシュ名の配列を指定するか、すべてのキャッシュを対象とする * を指定します。
cache-configs	オプション	バックアップするキャッシュテンプレートの配列、またはすべてのテンプレートの * を指定します。
counters	オプション	バックアップするカウンター名の配列、またはすべてのカウンターの * を定義します。

キー	必須またはオプション	値
proto-schemas	オプション	バックアップする Protobuf スキーマ名の配列、またはすべてのスキーマの * を定義します。
process	オプション	バックアップするサーバータスクの配列、またはすべてのタスクの * を指定します。

次の例では、指定されたディレクトリーに **[cache1,cache2]** という名前のすべてのカウンターとキャッシュを含むバックアップアーカイブを作成します。

```
{
  "directory": "/path/accessible/to/the/server",
  "resources": {
    "caches": ["cache1", "cache2"],
    "counters": ["*"]
  }
}
```

2.4.14. バックアップのリスト表示

進行中、完了、または失敗したすべてのバックアップ操作の名前を取得します。

```
GET /rest/v2/container/backups
```

Data Grid は、以下の例のように、すべてのバックアップ名の配列で応答します。

```
["backup1", "backup2"]
```

2.4.15. バックアップの可用性の確認

バックアップ操作が完了していることを確認します。

```
HEAD /rest/v2/container/backups/{backupName}
```

200 のレスポンスは、バックアップアーカイブが利用可能であることを示します。**202** の応答は、バックアップ操作が進行中であることを示します。

2.4.16. バックアップアーカイブのダウンロード

サーバーからバックアップアーカイブをダウンロードします。

```
GET /rest/v2/container/backups/{backupName}
```

200 のレスポンスは、バックアップアーカイブが利用可能であることを示します。**202** の応答は、バックアップ操作が進行中であることを示します。

2.4.17. バックアップアーカイブの削除

サーバーからバックアップアーカイブを削除します。

```
DELETE /rest/v2/container/backups/{backupName}
```

204 応答は、バックアップアーカイブが削除されたことを示します。**202** 応答は、バックアップ操作が進行中であるが、操作が完了すると削除されることを示します。

2.4.18. バックアップアーカイブからの Data Grid リソースの復元

バックアップアーカイブから Data Grid リソースを復元します。提供されている **{restoreName}** は、復元の進行状況を追跡するためのものであり、復元されるバックアップファイルの名前とは無関係です。

```
POST /rest/v2/container/restores/{restoreName}
```

202 応答は、復元要求が処理のために受け入れられたことを示します。

2.4.18.1. Data Grid サーバー上のバックアップアーカイブからの復元

サーバー上のアーカイブからバックアップする場合は、POST リクエストに **application/json** コンテンツタイプを使用します。

表2.28 JSON パラメーター

キー	必須またはオプション	値
location	必須	復元するバックアップアーカイブのパスを指定します。
resources	オプション	復元するリソースを JSON 形式で指定します。デフォルトでは、すべてのリソースを復元します。1 つまたは複数のリソースを指定した場合、Data Grid はそれらのリソースのみをリストアします。詳細については、**リソースパラメーター**の表を参照してください。

表2.29 リソースパラメーター

キー	必須またはオプション	値
caches	オプション	バックアップするキャッシュ名の配列を指定するか、すべてのキャッシュを対象とする * を指定します。

キー	必須またはオプション	値
cache-configs	オプション	バックアップするキャッシュテンプレートの配列、またはすべてのテンプレートの * を指定します。
counters	オプション	バックアップするカウンター名の配列、またはすべてのカウンターの * を定義します。
proto-schemas	オプション	バックアップする Protobuf スキーマ名の配列、またはすべてのスキーマの * を定義します。
process	オプション	バックアップするサーバータスクの配列、またはすべてのタスクの * を指定します。

次の例では、サーバー上のバックアップアーカイブからすべてのカウンターを復元します。

```
{
  "location": "/path/accessible/to/the/server/backup-to-restore.zip",
  "resources": {
    "counters": ["*"]
  }
}
```

2.4.18.2. ローカルバックアップアーカイブからの復元

ローカルのバックアップアーカイブをサーバーにアップロードするには、POST リクエストに **multipart/form-data** コンテンツタイプを使用します。

表2.30 フォームデータ

パラメーター	Content-Type	必須またはオプション	値
backup	**application/zip**	必須	復元するバックアップアーカイブのバイトを指定します。
resources	**application/json**, **text/plain**	オプション	リクエストパラメーターの JSON オブジェクトを定義します。

要求の例

```
Content-Type: multipart/form-data; boundary=5ec9bc07-f069-4662-a535-46069afeda32
Content-Length: 7721
```

```
--5ec9bc07-f069-4662-a535-46069afeda32
Content-Disposition: form-data; name="resources"
Content-Length: 23

{"scripts":["test.js"]}
--5ec9bc07-f069-4662-a535-46069afeda32
Content-Disposition: form-data; name="backup"; filename="testManagerRestoreParameters.zip"
Content-Type: application/zip
Content-Length: 7353

<zip-bytes>
--5ec9bc07-f069-4662-a535-46069afeda32--
```

2.4.19. リストの復元

進行中、完了、または失敗したすべての復元要求の名前を取得します。

```
GET /rest/v2/container/restores
```

Data Grid は、次の例のように、すべての復元名の配列で応答します。

```
["restore1", "restore2"]
```

2.4.20. 復元の進行状況を確認する

復元操作が完了したことを確認します。

```
HEAD /rest/v2/container/restores/{restoreName}
```

201 (Created) 応答は、リストア操作が完了したことを示します。**202 (Accepted)** 応答は、バックアップ操作が進行中であることを示します。

2.4.21. 復元メタデータの削除

サーバーから復元要求のメタデータを削除します。このアクションにより、復元要求に関連付けられているすべてのメタデータが削除されますが、復元されたコンテンツは削除されません。リクエストのメタデータを削除すると、リクエスト名を使用して後続の復元操作を実行できます。

```
DELETE /rest/v2/container/restores/{restoreName}
```

204 (No Content) 応答は、復元メタデータが削除されたことを示します。**202 (Accepted)** 応答は、復元操作が進行中であり、操作が完了すると削除されることを示します。

2.4.22. コンテナー設定イベントのリスニング

Server-Sent Events を使用して、設定変更に関するイベントを受信します。**event** 値は、**create-cache**、**remove-cache**、**update-cache**、**create-template**、**remove-template**、または **update-template** のいずれかになります。**data** 値には、作成されたエンティティーの宣言型設定が含まれます。削除イベントには、削除されたエンティティーの名前のみが含まれます。

```
GET /rest/v2/container/config?action=listen
```

表2.31 ヘッダー

Header	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。サポートされている形式は、**application/yaml**、**application/json**、**application/xml** です。デフォルトは **application/yaml** です。詳細については、Accept を参照してください。

表2.32 要求パラメーター

パラメーター	必須またはオプション	説明
includeCurrentState	オプション	**true** の場合、結果には変更に加えて既存の設定の状態が含まれます。**false** に設定すると、リクエストは変更のみを返します。デフォルト値は **false** です。
pretty	オプション	**true** の場合は、スペースや行区切りが追加された、フォーマット済みのコンテンツが返されます。これは、読みやすくはなりますが、パイロードのサイズが増えます。デフォルトは **false** です。

2.4.23. コンテナーイベントのリッスン

Server-Sent Events を使用してコンテナーからイベントを受け取ります。発行されたイベントはログに記録された情報に基づいているため、各イベントにはメッセージに関連付けられた識別子が含まれています。**event** 値は **lifecycle-event** になります。**data** には、**message**、**category**、**level**、**timestamp**、**owner**、**context**、および **scope** を含む、ログに記録された情報があり、その一部は空である可能性があります。現在、**LIFECYCLE** イベントのみを公開しています。

```
GET /rest/v2/container?action=listen
```

表2.33 ヘッダー

Header	必須またはオプション	パラメーター

Header	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。サポートされている形式は、 **application/yaml** 、 **application/json** 、 **application/xml** です。デフォルトは **application/yaml** です。詳細については、 Accept を参照してください。

表2.34 要求パラメーター

パラメーター	必須またはオプション	説明
includeCurrentState	オプション	**true** の場合、結果には変更に加えて既存の設定の状態が含まれます。 **false** に設定すると、リクエストは変更のみを返します。デフォルト値は **false** です。
pretty	オプション	**true** の場合は、スペースや行区切りが追加された、フォーマット済みのコンテンツが返されます。これは、読みやすくはなりますが、パイロードのサイズが増えます。デフォルトは **false** です。

2.4.24. 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。

2.4.24.1. バックアップの場所のステータスの取得

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 /rest/v2/container/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"
      ],
      "mixed": [
        "CACHE_3"
      ]
    }
  }
```

表2.35 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。

```
GET /rest/v2/container/x-site/backups/{site}
```

1 つのバックアップの場所のステータスを返します。

2.4.24.2. バックアップの場所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。

```
POST /rest/v2/container/x-site/backups/{siteName}?action=take-offline
```

2.4.24.3. バックアップの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。

```
POST /rest/v2/container/x-site/backups/{siteName}?action=bring-online
```

2.4.24.4. 状態遷移モードの取得

GET リクエストで状態遷移モードを確認してください。

```
GET /rest/v2/caches/{cacheName}/x-site/backups/{site}/state-transfer-mode
```

2.4.24.5. 状態遷移モードの設定

?action=set パラメーターを使用して状態遷移モードを設定します。

```
POST /rest/v2/caches/{cacheName}/x-site/backups/{site}/state-transfer-mode?action=set&mode={mode}
```

2.4.24.6. 状態遷移の開始

?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。

```
POST /rest/v2/container/x-site/backups/{siteName}?action=start-push-state
```

2.4.24.7. 状態遷移のキャンセル

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

```
POST /rest/v2/container/x-site/backups/{siteName}?action=cancel-push-state
```

2.5. 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。

2.5.1. 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"
}
```

2.5.2. 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 Cache Manager 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 Cache Manager 名の配列で応答します。

注記

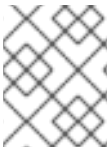
Data Grid は現在、サーバーごとに 1 つの Cache Manager のみをサポートしています。

2.5.3. 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。Cache Manager 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 /rest/v2/server/ignored-caches/{cache}
```

Data Grid は、キャッシュが無視リストに正常に追加された場合は **204 (No Content)**、キャッシュや Cache Manager が見つからない場合は **404 (Not Found)** で応答します。



注記

Data Grid は現在、サーバーごとに 1 つの Cache Manager 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 Cache Manager 名を指定する必要があります。

2.5.4. 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 /rest/v2/server/ignored-caches/{cache}
```

Data Grid は、キャッシュが無視リストから正常に削除された場合は **204 (No Content)**、キャッシュや Cache Manager が見つからない場合は **404 (Not Found)** で応答します。

2.5.5. 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 /rest/v2/server/ignored-caches/
```

2.5.6. 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"
        }
    }
  }
}
```

2.5.7. 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
```

2.5.8. 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。

2.5.9. JVM ヒープダンプの取得

POST 要求で Data Grid サーバーの JVM ヒープダンプを生成します。

```
POST /rest/v2/server/memory?action=heap-dump[&live=true|false]
```

Data Grid は、サーバーのデータディレクトリーに HPROF 形式のヒープダンプファイルを生成し、JSON 形式のファイルのフルパスで応答します。

2.5.10. 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。

2.5.11. Data Grid Server の診断レポートの取得

GET リクエストで Data Grid サーバーの集約されたレポートを取得します。要求されたサーバーのレポートを取得するには、以下を実行します。

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

クラスター内の別のサーバーのレポートを取得するには、名前でノードを参照します。

```
GET /rest/v2/server/report/{nodeName}
```

Data Grid は、Data Grid サーバーとホストの両方に関する診断情報を含む集約されたレポートを含む **tar.gz** アーカイブで応答します。レポートは、設定ファイルやログファイルに加えて、CPU、メモリー、オープンファイル、ネットワークソケットとルーティング、スレッドに関する詳細を提供します。

2.5.12. Data Grid サーバーの停止

POST リクエストで Data Grid サーバーを停止する。

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

Data Grid は **204 (No Content)** と応答し、実行を停止します。

2.5.13. クライアント接続情報の取得

GET リクエストで Data Grid サーバーに接続されているクライアントの情報を一覧表示します。

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

Data Grid は、次の例のように、すべてのアクティブなクライアント接続に関する詳細を JSON 形式で応答します。


```
[
  {
    "id": 2,
    "name": "flower",
    "created": "2023-05-18T14:54:37.882566188Z",
    "principal": "admin",
    "local-address": "/127.0.0.1:11222",
    "remote-address": "/127.0.0.1:58230",
    "protocol-version": "RESP3",
    "client-library": null,
    "client-version": null,
    "ssl-application-protocol": "http/1.1",
    "ssl-cipher-suite": "TLS_AES_256_GCM_SHA384",
    "ssl-protocol": "TLSv1.3"
  },
  {
    "id": 0,
```

```
    "name": null,
    "created": "2023-05-18T14:54:07.727775875Z",
    "principal": "admin",
    "local-address": "/127.0.0.1:11222",
    "remote-address": "/127.0.0.1:35716",
    "protocol-version": "HTTP/1.1",
    "client-library": "Infinispan CLI 15.0.0-SNAPSHOT",
    "client-version": null,
    "ssl-application-protocol": "http/1.1",
    "ssl-cipher-suite": "TLS_AES_256_GCM_SHA384",
    "ssl-protocol": "TLSv1.3"
  }
 ]
```


表2.36 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
global	オプション	**true**: クラスターのすべてのサーバーから接続を収集します。

2.5.14. キャッシュ設定のデフォルト値の取得

GET リクエストを使用してキャッシュ設定のデフォルト値を取得します。

```
POST /rest/v2/server/caches/defaults
```

Data Grid は、JSON 形式のキャッシュ設定のデフォルト値で応答します。

2.6. 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。

2.6.1. Data Grid クラスターの停止

POST 要求を使用して Data Grid クラスター全体をシャットダウンします。

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

Data Grid は **204 (No Content)** と応答し、クラスター全体の秩序あるシャットダウンを実行します。

2.6.2. 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。

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

Data Grid は **204 (No Content)** と応答します。

2.6.3. Data Grid クラスターのバックアップ

クラスターのキャッシュコンテナーに現在保存されているリソース (キャッシュ、テンプレート、カウンター、Protobuf スキーマ、サーバータスクなど) を含むバックアップアーカイブ、 **application/zip** を作成します。

```
POST /rest/v2/cluster/backups/{backupName}
```

オプションで、次のように、バックアップ操作のパラメーターを含む JSON ペイロードをリクエストに含めます。

表2.37 JSON パラメーター

キー	必須またはオプション	値
directory	オプション	バックアップアーカイブを作成および保存するサーバー上の場所を指定します。

バックアップ操作が正常に完了すると、サービスは **202 (Accepted)** を返します。同じ名前のバックアップがすでに存在する場合、サービスは **409 (Conflict)** 返します。**directory** パラメーターが無効な場合、サービスは **400 (Bad Request)** 返します。

2.6.4. バックアップのリスト表示

進行中、完了、または失敗したすべてのバックアップ操作の名前を取得します。

```
GET /rest/v2/cluster/backups
```

Data Grid は、以下の例のように、すべてのバックアップ名の配列で応答します。

```
["backup1", "backup2"]
```

2.6.5. バックアップの可用性の確認

バックアップ操作が完了していることを確認します。**200** のレスポンスは、バックアップアーカイブが利用可能であることを示します。**202** の応答は、バックアップ操作が進行中であることを示します。

```
HEAD /rest/v2/cluster/backups/{backupName}
```

2.6.6. バックアップアーカイブのダウンロード

サーバーからバックアップアーカイブをダウンロードします。**200** のレスポンスは、バックアップアーカイブが利用可能であることを示します。**202** の応答は、バックアップ操作が進行中であることを示します。

```
GET /rest/v2/cluster/backups/{backupName}
```

2.6.7. バックアップアーカイブの削除

サーバーからバックアップアーカイブを削除します。**204** 応答は、バックアップアーカイブが削除されたことを示します。**202** 応答は、バックアップ操作が進行中であるが、操作が完了すると削除されることを示します。

```
DELETE /rest/v2/cluster/backups/{backupName}
```

2.6.8. Data Grid クラスターリソースの復元

バックアップアーカイブ内のリソースを適用して、Data Grid クラスターを復元します。提供されている **{restoreName}** は、復元の進行状況を追跡するためのものであり、復元されるバックアップファイルの名前とは無関係です。

重要

バックアップアーカイブ内のコンテナー名がクラスターのコンテナー名と一致する場合にのみ、リソースを復元できます。

```
POST /rest/v2/cluster/restores/{restoreName}
```

202 応答は、復元要求が処理のために受け入れられたことを示します。

2.6.8.1. Data Grid サーバー上のバックアップアーカイブからの復元

サーバー上のアーカイブからバックアップする場合は、POST リクエストに **application/json** コンテンツタイプを使用します。

表2.38 JSON パラメーター

キー	必須またはオプション	値
location	必須	復元するバックアップアーカイブのパスを指定します。
resources	オプション	復元するリソースを JSON 形式で指定します。デフォルトでは、すべてのリソースを復元します。1 つまたは複数のリソースを指定した場合、Data Grid はそれらのリソースのみをリストアします。詳細については、**リソースパラメーター**の表を参照してください。

表2.39 リソースパラメーター

キー	必須またはオプション	値
caches	オプション	バックアップするキャッシュ名の配列を指定するか、すべてのキャッシュを対象とする * を指定します。

キー	必須またはオプション	値
cache-configs	オプション	バックアップするキャッシュテンプレートの配列、またはすべてのテンプレートの * を指定します。
counters	オプション	バックアップするカウンター名の配列、またはすべてのカウンターの * を定義します。
proto-schemas	オプション	バックアップする Protobuf スキーマ名の配列、またはすべてのスキーマの * を定義します。
process	オプション	バックアップするサーバータスクの配列、またはすべてのタスクの * を指定します。

次の例では、サーバー上のバックアップアーカイブからすべてのカウンターを復元します。

```
{
  "location": "/path/accessible/to/the/server/backup-to-restore.zip",
  "resources": {
    "counters": ["*"]
  }
}
```

2.6.8.2. ローカルバックアップアーカイブからの復元

ローカルのバックアップアーカイブをサーバーにアップロードするには、POST リクエストに **multipart/form-data** コンテンツタイプを使用します。

表2.40 フォームデータ

パラメーター	Content-Type	必須またはオプション	値
backup	**application/zip**	必須	復元するバックアップアーカイブのバイトを指定します。

要求の例

```
Content-Type: multipart/form-data; boundary=5ec9bc07-f069-4662-a535-46069afeda32
Content-Length: 7798

--5ec9bc07-f069-4662-a535-46069afeda32
Content-Disposition: form-data; name="backup"; filename="testManagerRestoreParameters.zip"
Content-Type: application/zip
Content-Length: 7353
```

```
<zip-bytes>
--5ec9bc07-f069-4662-a535-46069afeda32--
```

2.6.9. リストの復元

進行中、完了、または失敗したすべての復元要求の名前を取得します。

```
GET /rest/v2/cluster/restores
```

Data Grid は、次の例のように、すべての復元名の配列で応答します。

```
["restore1", "restore2"]
```

2.6.10. 復元の進行状況を確認する

復元操作が完了したことを確認します。

```
HEAD /rest/v2/cluster/restores/{restoreName}
```

201 (Created) 応答は、リストア操作が完了したことを示します。**202** の応答は、バックアップ操作が進行中であることを示します。

2.6.11. 復元メタデータの削除

サーバーから復元要求のメタデータを削除します。このアクションにより、復元要求に関連付けられているすべてのメタデータが削除されますが、復元されたコンテンツは削除されません。リクエストのメタデータを削除すると、リクエスト名を使用して後続の復元操作を実行できます。

```
DELETE /rest/v2/cluster/restores/{restoreName}
```

204 応答は、復元メタデータが削除されたことを示します。**202** 応答は、復元操作が進行中であり、操作が完了すると削除されることを示します。

2.6.12. クラスター分布の確認

Data Grid クラスター内のすべてのサーバーに関するディストリビューションの詳細を取得します。

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

クラスター内の各 Data Grid サーバー統計の JSON 配列を次の形式で返します。

```
[
  {
    "node_name": "NodeA",
    "node_addresses": [
      "127.0.0.1:39313"
    ],
    "memory_available": 466180016,
    "memory_used": 56010832
  },
  {
```

```
    "node_name": "NodeB",
    "node_addresses": [
      "127.0.0.1:47477"
    ],
    "memory_available": 467548568,
    "memory_used": 54642280
  }
]
```

配列内の各要素は、Data Grid ノードを表します。統計収集が無効になっている場合、メモリー使用量の値に関する情報は -1 です。プロパティーは次のとおりです。

- **node_name** はノード名です。
- **node_addresses** は、すべてのノードの物理アドレスのリストです。
- **memory_available** ノードで使用可能なメモリー (バイト単位)。
- **memory_used** ノードの使用メモリー (バイト単位)。

2.7. DATA GRID サーバーのログ設定

実行時に Data Grid クラスターのログ設定を表示および変更します。

2.7.1. ロギングアペンダーのリスト表示

GET リクエストで設定されたすべてのアペンダーのリストを表示します。

```
GET /rest/v2/logging/appenders
```

Data Grid は、次の例のように、JSON 形式のアペンダーのリストで応答します。

```
{
  "STDOUT" : {
    "name" : "STDOUT"
  },
  "JSON-FILE" : {
    "name" : "JSON-FILE"
  },
  "HR-ACCESS-FILE" : {
    "name" : "HR-ACCESS-FILE"
  },
  "FILE" : {
    "name" : "FILE"
  },
  "REST-ACCESS-FILE" : {
    "name" : "REST-ACCESS-FILE"
  }
}
```

2.7.2. ロガーのリスト表示

GET リクエストで設定されたすべてのロガーのリストを表示します。

```
GET /rest/v2/logging/loggers
```

Data Grid は、次の例のように、JSON 形式のロガーのリストで応答します。

```
[ {
  "name" : "",
  "level" : "INFO",
  "appenders" : [ "STDOUT", "FILE" ]
}, {
  "name" : "org.infinispan.HOTROD_ACCESS_LOG",
  "level" : "INFO",
  "appenders" : [ "HR-ACCESS-FILE" ]
}, {
  "name" : "com.arjuna",
  "level" : "WARN",
  "appenders" : [ ]
}, {
  "name" : "org.infinispan.REST_ACCESS_LOG",
  "level" : "INFO",
  "appenders" : [ "REST-ACCESS-FILE" ]
} ]
```

2.7.3. ロガーの作成/変更

新しいロガーを作成するか、 **PUT** リクエストで既存のロガーを変更します。

```
PUT /rest/v2/logging/loggers/{loggerName}?level={level}&appender={appender}&appender=
{appender}...
```

Data Grid は、**{loggerName}** で識別されるロガーのレベルを **{level}** に設定します。オプションで、ロガーに 1 つ以上のアペンダーを設定できます。アペンダーが指定されていない場合は、ルートロガーで指定されたアペンダーが使用されます。

操作が正常に完了すると、サービスは **204 (No Content)** を返します。

2.7.4. ロガーの削除

DELETE リクエストで既存のロガーを削除します。

```
DELETE /rest/v2/logging/loggers/{loggerName}
```

Data Grid は、**{loggerName}** で識別されるロガーを削除し、root ロガー設定の使用に効果的に戻します。

操作が正常に処理された場合、サービスは応答コード **204 (No Content)** を返します。

2.8. 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。

2.8.1. 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2.41 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"
  }
]
```

2.8.2. タスクの実行

タスク名、オプションのキャッシュ名、**param** で始まる必須パラメーターを含む **POST** リクエストでタスクを実行します。

```
POST /rest/v2/tasks/SimpleTask?action=exec&cache=mycache&param.p1=v1&param.p2=v2
```

Data Grid はタスクの結果で応答します。

2.8.3. 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
```

2.8.4. スクリプトタスクのダウンロード

GET 要求でスクリプトタスクをダウンロードします。

```
GET /rest/v2/tasks/taskName?action=script
```

2.9. DATA GRID セキュリティーの操作

セキュリティー情報を表示および変更します。

2.9.1. ユーザーの ACL の取得

ユーザーのプリンシパルとアクセス制御リストに関する情報を表示します。

```
GET /rest/v2/security/user/acl
```

Data Grid は、リクエストを実行したユーザーに関する情報で応答します。このリストには、ユーザーのプリンシパル、リソースのリストとユーザーがアクセスする際のパーミッションが含まれています。

```
{
  "subject": [
    {
      "name": "deployer",
      "type": "NamePrincipal"
    }
  ],
  "global": [
    "READ", "WRITE", "EXEC", "LISTEN", "BULK_READ", "BULK_WRITE", "CREATE", "MONITOR",
"ALL_READ", "ALL_WRITE"  ],
  "caches": {
    "___protobuf_metadata": [
      "READ", "WRITE", "EXEC", "LISTEN", "BULK_READ", "BULK_WRITE", "CREATE", "MONITOR",
"ALL_READ", "ALL_WRITE"
    ],
    "mycache": [
      "LIFECYCLE", "READ", "WRITE", "EXEC", "LISTEN", "BULK_READ", "BULK_WRITE", "ADMIN",
"CREATE", "MONITOR", "ALL_READ", "ALL_WRITE"
    ],
    "___script_cache": [
      "READ", "WRITE", "EXEC", "LISTEN", "BULK_READ", "BULK_WRITE", "CREATE", "MONITOR",
"ALL_READ", "ALL_WRITE"
    ]
  }
}
```

2.9.2. セキュリティーキャッシュのフラッシュ

クラスター全体でセキュリティーキャッシュをフラッシュします。

```
POST /rest/v2/security/cache?action=flush
```

2.9.3. 利用可能なロールの取得

サーバーに定義されている利用可能なすべてのロールを表示します。

```
GET /rest/v2/security/roles
```

Data Grid は、利用可能なロールのリストで応答します。認証が有効な場合、**ADMIN** 権限を持つユーザーのみがこの API を呼び出すことができます。

```
["observer","application","admin","monitor","deployer"]
```

2.9.4. 利用可能なロールの詳細取得

サーバーに定義されている利用可能なすべてのロールを詳細で表示します。

```
GET /rest/v2/security/roles?action=detailed
```

Data Grid は、利用可能なロールのリストとその詳細で応答します。認証が有効な場合、**ADMIN** 権限を持つユーザーのみがこの API を呼び出すことができます。

```
{
  "observer": {
    "inheritable": true,
    "permissions": [
      "MONITOR",
      "ALL_READ"
    ],
    "implicit": true,
    "description": "..."
  },
  "application": {
    "inheritable": true,
    "permissions": [
      "MONITOR",
      "ALL_WRITE",
      "EXEC",
      "LISTEN",
      "ALL_READ"
    ],
    "implicit": true,
    "description": "..."
  },
  "admin": {
    "inheritable": true,
    "permissions": [
      "ALL"
    ],
```

```
    "implicit": true,
    "description": "..."
  },
  "monitor": {
    "inheritable": true,
    "permissions": [
      "MONITOR"
    ],
    "implicit": true,
    "description": "..."
  },
  "deployer": {
    "inheritable": true,
    "permissions": [
      "CREATE",
      "MONITOR",
      "ALL_WRITE",
      "EXEC",
      "LISTEN",
      "ALL_READ"
    ],
    "implicit": true,
    "description": "..."
  }
}
```

2.9.5. プリンシパルのロールの取得

プリンシパルにマップするすべてのロールを表示します。

```
GET /rest/v2/security/roles/some_principal
```

Data Grid は、指定されたプリンシパルで使用可能なロールのリストで応答します。プリンシパルは、使用中の領域に存在する必要はありません。

```
["observer"]
```

2.9.6. プリンシパルにロールを付与する

プリンシパルに 1 つ以上の新しいロールを付与します。

```
PUT /rest/v2/security/roles/some_principal?action=grant&role=role1&role=role2
```

表2.42 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
role	必須	ロールの名前

2.9.7. 校長へのロールの拒否

以前にプリンシパルに付与された 1 つ以上のロールを削除します。

```
PUT /rest/v2/security/roles/some_principal?action=deny&role=role1&role=role2
```

表2.43 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
role	必須	ロールの名前

2.9.8. プリンシパルの一覧表示

ユーザーを列挙できるすべてのセキュリティーレルムのプリンシパル名を一覧表示します(**プロパティー**、**ldap**)。

```
GET /rest/v2/security/principals
```

Data Grid は、各レルムにキー指定されたプリンシパルのリストで応答します

```
{"default:properties":["admin","user1","user2"]}
```

2.9.9. ロールの作成

名前、パーミッション、および要求本文で任意の説明を定義して、ロールを作成します。

```
POST /rest/v2/security/permissions/somerole?permission=permission1&permission=permission2
```

表2.44 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
permission	必須	パーミッションの名前

2.9.10. ロールの更新

既存のロールパーミッションや説明を更新します。

```
POST /rest/v2/security/permissions/somerole?permission=permission1&permission=permission2
```

表2.45 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
permission	必須	パーミッションの名前

2.9.11. ロールの削除

既存のロールを削除します。

```
DELETE /rest/v2/security/permissions/somerole
```

2.9.12. ロールのパーミッションの取得

ロールのすべてのパーミッションを表示します。

```
GET /rest/v2/security/permissions/somerole
```

Data Grid は、指定されたプリンシパルで使用可能なロールのリストで応答します。プリンシパルは、使用中の領域に存在する必要はありません。

```
{
  "name" : "application",
  "permissions" : [ "LISTEN","ALL_READ","MONITOR","ALL_WRITE","EXEC" ],
  "inheritable": true,
  "implicit": true,
  "description": "..."
}
```

2.10. トレース伝播の有効化

Data Grid Server と REST API を使用したトレースにより、リクエストのフローを監視および分析し、さまざまなコンポーネント間の実行パスを追跡できます。

2.10.1. Data Grid Server と REST API 間のトレース伝播の有効化

Data Grid サーバーと REST API の間でトレース伝播を有効にする場合は、クライアント側とサーバー側の両方でトレースを設定する必要があります。

OpenTelemetry トレーススパンを Data Grid スパンに伝播するには、REST 呼び出しごとにトレースコンテキストを設定する必要があります。

前提条件

- Data Grid サーバーとリモートクライアント側でトレースを有効にしている。

手順

1. **io.opentelemetry.api.trace.propagation.W3CTraceContextPropagator** を使用して、現在のトレースコンテキストを抽出します。
   抽出により、トレースコンテキスト情報を格納するコンテキストマップが生成されます。
2. トレースコンテキストが確実に保持されるように、REST 呼び出しのヘッダーでコンテキストマップを渡します。

```
HashMap<String, String> contextMap = new HashMap<>();

// Inject the request with the *current* Context, which contains our current Span.
W3CTraceContextPropagator.getInstance().inject(Context.current(), contextMap,
(carrier, key, value) -> carrier.put(key, value));

// Pass the context map in the header
RestCacheClient client = restClient.cache(CACHE_NAME);
client.put("aaa",
MediaType.TEXT_PLAIN.toString(),RestEntity.create(MediaType.TEXT_PLAIN, "bbb"),
contextMap);
```

クライアントアプリケーションが生成するトレーススパンは、Data Grid サーバーによって生成される依存スパンと関連付けられます。

関連情報

- Data Grid トレースの有効化
- Hot Rod クライアントトレース伝播