Red Hat Data Grid 8.3

Hot Rod Node.JS クライアントガイド

Hot Rod JS クライアントの設定および使用

Last Updated: 2023-12-06

Red Hat Data Grid 8.3 Hot Rod Node.JS クライアントガイド

Hot Rod JS クライアントの設定および使用

法律上の通知

概要

Hot Rod JS クライアントは、Node.js アプリケーションの Data Grid クラスターへの非同期でイベント駆動型のアクセスを提供します。非同期操作の結果は `Promise` インスタンスで表され、クライアントは複数の呼び出しを連鎖させ、エラー処理を一元化できます。

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。

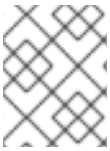
DATA GRID のドキュメント

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 8.3 ドキュメント
- Data Grid 8.3 コンポーネントの詳細
- Data Grid 8.3 でサポートされる設定
- 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、今後の複数のリリースで段階的に用語の置き換えを実施して参ります。詳細は、Red Hat CTO である Chris Wright のメッセージ を参照してください。

第1章 HOT ROD JS クライアントのインストールおよび設定

Hot Rod JS クライアントをインストールする前に、システムが要件を満たしていることを確認してください。次に、Hot Rod JS クライアントを Data Grid Server に接続し、キーと値に異なるメディアタイプを使用し、ロギングをカスタマイズできます。

1.1. HOT ROD JS クライアントのインストール

Data Grid は、NPM パッケージマネージャーからインストールできる Hot Rod JS クライアントのディストリビューションを提供します。

前提条件

- Node.js バージョン **12** または **14**。
- Data Grid Server 8.3。

手順

1. Red Hat リポジトリーを NPM 設定に追加します。
   npm config コマンドを使用するか、プロジェクトの **.npmrc** ファイルに以下を追加できます。

```
@redhat:registry=https://npm.registry.redhat.com
registry=https://registry.npmjs.org/
```

2. Hot Rod JS クライアントを次のようにインストールします。

```
npm install @redhat/infinispan
```

1.2. DATA GRID コネクションの設定

Hot Rod JS クライアントを Data Grid Server に接続するように設定します。

複数のサーバーアドレスを設定に追加する場合、Hot Rod JS クライアントは、接続できるノードを見つけるまでループします。

ただし、クライアントがクラスタートポロジー全体を受け取るには、Data Grid Server アドレスを 1 つだけ追加する必要があります。Hot Rod JS クライアントがクラスターのメンバーである単一のサーバーインスタンスに接続する場合、クライアントはすべてのノードのアドレス情報を取得します。

Hot Rod JS クライアントはトポロジーを認識するため、1 つの Data Grid Server への接続が切断されると、クライアントはクラスター内の他のノードで不完全な操作を再試行します。同様に、ある Data Grid Server に登録されているクライアントリスナーに障害が発生した場合や、クラスターを離れると、クライアントはリスナー登録をクラスター内の別のノードに透過的に移行し、イベントの受信を継続できるようにします。

前提条件

- Hot Rod JS クライアントをインストールします。
- 稼働中の Data Grid Server インスタンスが 1 つ以上ある。

手順

- クライアント設定で Data Grid Server のホスト名およびポートを指定します。

```
var infinispan = require('infinispan');

var connected = infinispan.client(
  [{port: 11322, host: '127.0.0.1'}, {port: 11222, host: '127.0.0.1'}]
  {
    // Configure client connections with authentication and encryption here.
  }
);

connected.then(function (client) {

  var members = client.getTopologyInfo().getMembers();

  // Displays all members of the Data Grid cluster.
  console.log('Connected to: ' + JSON.stringify(members));

  return client.disconnect();

}).catch(function(error) {

  console.log("Got error: " + error.message);

});
```

1.2.1. クライアント設定での Data Grid クラスターの定義

クロスサイトレプリケーションを実行するように別のデータセンターに Data Grid クラスターを設定すると、異なるサイトの接続詳細をクライアント設定に追加できます。

前提条件

- Hot Rod JS クライアントをインストールします。
- クロスサイトレプリケーション用に Data Grid を設定します。

手順

1. **clusters** 定義を設定に追加します。
2. 各 Data Grid クラスターの **name** と **servers** 定義を追加します。

```
var connected = infinispan.client({port: 11222, host: '127.0.0.1'},
  {
    clusters: [
      {
        name: 'LON',
        servers: [{port: 11222, host: 'LON-host'}]
      },
      {
        name: 'NYC',
        servers: [{port: 11222, host: 'NYC-host1'}, {port: 11322, host: 'NYC-host2'}]
```

```
    }
  ]
});
```

1.2.2. Data Grid クラスターの手動切り替え

Hot Rod JS クライアントが接続される Data Grid クラスターを変更します。

前提条件

- Hot Rod JS クライアント設定で Data Grid クラスターを定義します。

手順

1. **switchToCluster(clusterName)** メソッドを呼び出して、クライアントに、クライアント設定に定義されている Data Grid クラスターへの切り替えを強制します。
2. **switchToDefaultCluster()** メソッドを呼び出して、最初の Data Grid クラスターの使用を開始します。

```
var connected = infinispan.client({port: 11222, host: '127.0.0.1'},
 {
  clusters: [
   {
    name: 'LON',
    servers: [{port: 11222, host: 'LON-host'}]
   },
   {
    name: 'NYC',
    servers: [{port: 11222, host: 'NYC-host1'}, {port: 11322, host: 'NYC-host2'}]
   }
  ]
 });

connected.then(function (client) {

 var switchToB = client.getTopologyInfo().switchToCluster('NYC');

 switchToB.then(function(switchSucceed) {

  if (switchSucceed) {
   ...
  }

  ...

  var switchToDefault = client.getTopologyInfo().switchToDefaultCluster();

  switchToDefault.then(function(switchSucceed) {

   if (switchSucceed) {
    ...
   }

  })
```

```
  })

});
```

1.3. 認証の設定

Data Grid Server は、Hot Rod JS クライアント接続を認証するために異なる SASL メカニズムを使用します。

前提条件

- Data Grid ユーザーを作成します。
- SASL 認証メカニズムを Data Grid Server 設定の Hot Rod コネクターに追加します。

手順

1. Hot Rod JS クライアント設定を開いて編集します。
2. **enabled: true** フラグを設定する **authentication** メソッドを追加します。
3. Hot Rod コネクターの SASL 認証メカニズムに一致する **saslMechanism** パラメーターの値を指定します。
4. 必要に応じて、SASL 認証メカニズムに固有のパラメーターを設定します。

1.3.1. SASL 認証メカニズム

Hot Rod JS クライアントは、以下の SASL 認証メカニズムを使用して Data Grid Server に接続できます。

PLAIN
HTTP **BASIC** 認証と同様の方法で、ネットワーク上のプレーンテキスト (暗号化されていない) で認証情報を送信します。

重要

Data Grid の認証情報のセキュリティーを保護するには、TLS 暗号化と組み合わせた **PLAIN** 認証のみを使用する必要があります。

```
var connected = infinispan.client(
  {port: 11222, host: '127.0.0.1'},
  {
    authentication: {
      enabled: true,
      saslMechanism: 'PLAIN',
      userName: 'username',
      password: 'changeme'
    }
  }
);
```

DIGEST-MD5

nonce に加えて MD5 ハッシュアルゴリズムを使用して、認証情報を暗号化します。

```
var connected = infinispan.client(
  {port: 11222, host: '127.0.0.1'},
  {
    authentication: {
      enabled: true,
      saslMechanism: 'DIGEST-MD5',
      userName: 'username',
      password: 'changeme',
      serverName: 'infinispan'
    }
  }
);
```

SCRAM
ハッシュアルゴリズムと nonce の値に加えて salt 値を使用して認証情報を暗号化します。Hot Rod エンドポイントは、**SCRAM-SHA-1**、**SCRAM-SHA-256**、**SCRAM-SHA-384**、**SCRAM-SHA-512** ハッシュアルゴリズムを強度順にサポートします。

```
var connected = infinispan.client(
  {port: 11222, host: '127.0.0.1'},
  {
    authentication: {
      enabled: true,
      saslMechanism: 'SCRAM-SHA-1',
      userName: 'username',
      password: 'changeme'
    }
  }
);
```

関連情報

- エンドポイント認証メカニズムの設定

1.4. 暗号化の設定

Data Grid Server は、異なるタイプの SSL/TLS 暗号化を適用して、Hot Rod JS クライアント接続をセキュアにすることができます。

前提条件

- Hot Rod JS クライアントが Data Grid Server アイデンティティーを検証するのに使用できるトラストストアを作成します。
- Data Grid Server を設定してクライアント証明書を検証または認証する場合は、必要に応じてキーストアを作成します。

手順

1. Hot Rod JS クライアント設定を開いて編集します。
2. **enabled: true** フラグを設定する **ssl** メソッドを追加します。

3. 使用する暗号化タイプに固有のその他の設定を指定します。

1.4.1. 暗号化タイプ

Hot Rod JS クライアントは、異なるタイプの暗号化を使用して、Data Grid Server とセキュアな接続をネゴシエートできます。

Data Grid Server のアイデンティティー
基本的な暗号化の場合、次のように、Data Grid Server 証明書の署名証明書または CA バンドルを設定に追加できます。

注記

Data Grid Server に発行された証明書を検証するには、Hot Rod JS クライアントでは、完全な証明書チェーンまたは Root CA で始まる部分的なチェーンのいずれかが必要です。

```
var connected = infinispan.client({port: 11222, host: '127.0.0.1'},
  {
    ssl: {
      enabled: true,
      trustCerts: ['my-root-ca.crt.pem']
    }
  }
);
```

トラストストア
以下のように、**PKCS12** または **PFX** 形式でトラストストアを追加できます。

```
var connected = infinispan.client({port: 11222, host: '127.0.0.1'},
  {
    ssl: {
      enabled: true,
      cryptoStore: {
        path: 'my-truststore.p12',
        passphrase: 'secret'
      }
    }
  }
);
```

クライアント証明書認証
Data Grid Server 設定でクライアント証明書認証を有効にする場合は、以下の例のようにキーストアを追加します。

注記

クライアント証明書認証を使用する場合は、Hot Rod JS クライアントを **EXTERNAL** 認証メカニズムで設定する必要があります。

```
var connected = infinispan.client({port: 11222, host: '127.0.0.1'},
  {
    ssl: {
```

```
      enabled: true,
      trustCerts: ['my-root-ca.crt.pem'],
      clientAuth: {
        key: 'privkey.pem',
        passphrase: 'secret',
        cert:ssl 'cert.pem'
      }
    }
  }
);
```

Server Name Indication (SNI)
SNI を使用して Hot Rod JS クライアントが Data Grid Server ホスト名を要求できるようにするには、Data Grid Server 設定のホスト名と一致する **sniHostName** パラメーターの値を設定します。

注記

The **sniHostName** パラメーターのデフォルトは **localhost** です。

```
var connected = infinispan.client({port: 11222, host: '127.0.0.1'},
  {
    ssl: {
      enabled: true,
      trustCerts: ['my-root-ca.crt.pem']
      sniHostName: 'example.com'
    }
  }
);
```

ヒント

Hot Rod JS クライアントは、デフォルトで自己署名証明書を許可しません。これにより、パブリック認証局 (CA) キーが利用できない開発環境またはテスト環境で問題が発生する可能性があります。

Java keytool で署名済み証明書を作成するなど、Data Grid コードチュートリアル を参照してください。

関連情報

- ネットワークインターフェイスおよびエンドポイント
- Data Grid Server 接続の暗号化

1.5. データフォーマットの設定

Hot Rod JS クライアントは、キーおよび値をネイティブの JavaScript Object Notation (JSON) オブジェクト、または String オブジェクトとして処理できます。デフォルトでは、クライアントはエントリーを String オブジェクトとして処理します。JSON 形式で Data Grid Server にデータを送信する場合は、Hot Rod JS クライアントを設定する必要があります。

注記

スクリプト操作は、String キー/値のペアと String パラメーターのみをサポートします。

手順

1. **dataFormat** 設定をクライアントに追加します。
2. **keyType** および **valueType** パラメーターで、キーおよび値のデータ形式を随時設定します。

キーと値は異なるメディアタイプを持つことができます。JSON オブジェクトには、**application/json** を指定します。String オブジェクトの場合は、**text/plain** を指定するか、デフォルトを使用するようにパラメーターを省略します。

```
var infinispan = require('infinispan');

var connected = infinispan.client(
   {port: 11222, host: '127.0.0.1'},
   {
      dataFormat : {
         keyType: 'application/json',
         valueType: 'application/json'
      }
   }
);

connected.then(function (client) {

  var clientPut = client.put({k: 'key'}, {v: 'value'});

  var clientGet = clientPut.then(
     function() { return client.get({k: 'key'}); });

  var showGet = clientGet.then(
     function(value) { console.log("get({k: 'key'})=" + JSON.stringify(value)); });

  return showGet.finally(
     function() { return client.disconnect(); });

}).catch(function(error) {

  console.log("Got error: " + error.message);

});
```

1.6. ロギングの設定

Hot Rod JS クライアントは、JSON 形式で設定を指定して変更できる **log4js** を使用します。

手順

1. JSON 形式でロギング設定を作成します。
   たとえば、以下の JSON は、TRACE レベルのログイベントをファイルに書き込むアペンダーを設定します。

```
{
  "appenders": {
    "test": {
      "type": "fileSync",
```

```
      "filename": "my-log-file.log"
    }
  },
  "categories": {
    "default": {
      "appenders": ["test"],
      "level": "trace"
    }
  }
}
```

2. **var log4js = require('log4js')** ステートメントを Hot Rod JS クライアント設定に追加します。
3. 以下の例のように、**log4js.configure()** メソッドを使用して JSON ロギング設定へのパスを指定します。

```
var log4js = require('log4js');
log4js.configure('path/to/my-log4js.json');
```

関連情報

- log4js
- log4js-node examples

第2章 HOT ROD JS クライアントの使用

Data Grid で Hot Rod JS クライアントを使用する例を見てみましょう。

2.1. HOT ROD JS クライアントの例

Hot Rod JS クライアントをインストールして設定した後に、Data Grid とのより複雑な操作に移動する前に、一部の基本キャッシュ操作を試して使用を開始します。

2.1.1. Hello world

Data Grid Server で"myCache"という名前のキャッシュを作成し、エントリーを追加および取得します。

```
var infinispan = require('infinispan');

// Connect to Data Grid Server.
// Use an existing cache named "myCache".
var connected = infinispan.client(
  {port: 11222, host: '127.0.0.1'},
  {
    cacheName: 'myCache',
    clientIntelligence: 'BASIC',
    authentication: {
      enabled: true,
      saslMechanism: 'DIGEST-MD5',
      userName: 'username',
      password: 'changeme'
    }
  }
);

connected.then(function (client) {

  console.log('Connected to `myCache`');

  // Add an entry to the cache.
  var clientPut = client.put('hello', 'world');

  // Retrieve the entry you added.
  var clientGet = clientPut.then(
      function() { return client.get('hello'); });

  // Print the value of the entry.
  var showGet = clientGet.then(
      function(value) { console.log('get(hello)=' + value); });

  // Disconnect from Data Grid Server.
  return client.disconnect();

}).catch(function(error) {

  // Log any errors.
```

```
  console.log("Got error: " + error.message);

});
```

2.1.2. エントリーの使用およびキャッシュ統計の取得

単一エントリーを追加、取得、削除し、キャッシュの統計を表示します。

```
var infinispan = require('infinispan');

var connected = infinispan.client(
  {port: 11222, host: '127.0.0.1'},
  {
    cacheName: 'myCache',
    authentication: {
        enabled: true,
        saslMechanism: 'DIGEST-MD5',
        userName: 'username',
        password: 'changeme'
    }
  }
);

connected.then(function (client) {

  var clientPut = client.put('key', 'value');

  var clientGet = clientPut.then(
      function() { return client.get('key'); });

  var showGet = clientGet.then(
      function(value) { console.log('get(key)=' + value); });

  var clientRemove = showGet.then(
      function() { return client.remove('key'); });

  var showRemove = clientRemove.then(
      function(success) { console.log('remove(key)=' + success); });

  var clientStats = showRemove.then(
    function() { return client.stats(); });

  var showStats = clientStats.then(
    function(stats) {
      console.log('Number of stores: ' + stats.stores);
      console.log('Number of cache hits: ' + stats.hits);
      console.log('All statistics: ' + JSON.stringify(stats, null, " "));
    });

  return showStats.finally(
      function() { return client.disconnect(); });

}).catch(function(error) {
```

```
  console.log("Got error: " + error.message);

});
```

2.1.3. 複数のキャッシュエントリーの使用

単純な再帰的ループで複数のキャッシュエントリーを作成します。

```
var infinispan = require('infinispan');

var connected = infinispan.client(
  {port: 11222, host: '127.0.0.1'},
  {
    cacheName: 'myCache',
    authentication: {
        enabled: true,
        saslMechanism: 'DIGEST-MD5',
        userName: 'username',
        password: 'changeme'
    }
  }
);

connected.then(function (client) {
  var data = [
    {key: 'multi1', value: 'v1'},
    {key: 'multi2', value: 'v2'},
    {key: 'multi3', value: 'v3'}];

  var clientPutAll = client.putAll(data);

  var clientGetAll = clientPutAll.then(
    function() { return client.getAll(['multi2', 'multi3']); });

  var showGetAll = clientGetAll.then(
    function(entries) {
      console.log('getAll(multi2, multi3)=%s', JSON.stringify(entries));
    }
  );

  var clientIterator = showGetAll.then(
    function() { return client.iterator(1); });

  var showIterated = clientIterator.then(
    function(it) {
      function loop(promise, fn) {
        // Simple recursive loop over the iterator's next() call.
        return promise.then(fn).then(function (entry) {
          return entry.done
            ? it.close().then(function () { return entry.value; })
            : loop(it.next(), fn);
        });
      }

      return loop(it.next(), function (entry) {
```

```
        console.log('iterator.next()=' + JSON.stringify(entry));
        return entry;
      });
    }
  );

  var clientClear = showIterated.then(
    function() { return client.clear(); });

  return clientClear.finally(
    function() { return client.disconnect(); });

}).catch(function(error) {

  console.log("Got error: " + error.message);

});
```

2.1.4. Async および Await コンストラクトの使用

Node.js は、キャッシュ操作を簡素化できる **async** および **await** コンストラクトを提供します。

単一のキャッシュエントリー

```
const infinispan = require("infinispan");

const log4js = require('log4js');
log4js.configure('example-log4js.json');

async function test() {
  await new Promise((resolve, reject) => setTimeout(() => resolve(), 1000));
  console.log('Hello, World!');

  let client = await infinispan.client({port: 11222, host: '127.0.0.1'});
  console.log(`Connected to Infinispan dashboard data`);

  await client.put('key', 'value');

  let value = await client.get('key');
  console.log('get(key)=' + value);

  let success = await client.remove('key');
  console.log('remove(key)=' + success);

  let stats = await client.stats();
  console.log('Number of stores: ' + stats.stores);
  console.log('Number of cache hits: ' + stats.hits);
  console.log('All statistics: ' + JSON.stringify(stats, null, " "));

  await client.disconnect();
}

test();
```

複数のキャッシュエントリー

```
const infinispan = require("infinispan");

const log4js = require('log4js');
log4js.configure('example-log4js.json');

async function test() {
  let client = await infinispan.client({port: 11222, host: '127.0.0.1'});
  console.log(`Connected to Infinispan dashboard data`);

  let data = [
    {key: 'multi1', value: 'v1'},
    {key: 'multi2', value: 'v2'},
    {key: 'multi3', value: 'v3'}];

  await client.putAll(data);

  let entries = await client.getAll(['multi2', 'multi3']);
  console.log('getAll(multi2, multi3)=%s', JSON.stringify(entries));

  let iterator = await client.iterator(1);

  let entry = {done: true};

  do {
    entry = await iterator.next();
    console.log('iterator.next()=' + JSON.stringify(entry));
  } while (!entry.done);

  await iterator.close();

  await client.clear();

  await client.disconnect();
}

test();
```

2.1.5. サーバー側のスクリプトの実行

カスタムスクリプトを Data Grid Server に追加し、Hot Rod JS クライアントから実行できます。

サンプルスクリプト

```
// mode=local,language=javascript,parameters=[k, v],datatype='text/plain; charset=utf-8'
cache.put(k, v);
cache.get(k);
```

スクリプト実行

```
var infinispan = require('infinispan');
var readFile = Promise.denodeify(require('fs').readFile);

var connected = infinispan.client(
  {port: 11222, host: '127.0.0.1'}
```

```
  {
    // Configure client connections with authentication and encryption here.
  }
);

connected.then(function (client) {

  var addScriptFile = readFile('sample-script.js').then(
    function(file) {
      return client.addScript('sample-script', file.toString());
    });

  var clientExecute = addScriptFile.then(
    function() {
      return client.execute('sample-script', {k: 'exec-key', v: 'exec-value'});
    });

  var showExecute = clientExecute.then(
    function(ret) { console.log('Script execution returned: ' + ret); });

  return showExecute.finally(
    function() { return client.disconnect(); });

}).catch(function(error) {

  console.log("Got error: " + error.message);

});
```

2.1.6. イベントリスナーの登録

イベントリスナーは、エントリーの作成、変更、削除、期限切れのタイミングなど、キャッシュ更新が発生したときに Hot Rod JS クライアントに通知します。

注記

エントリー作成および変更のイベントは、クライアントにキーと値について通知します。エントリー削除および有効期限のイベントは、クライアントに対してキーのみについて通知します。

イベントリスナーの登録

```
var infinispan = require('infinispan');

var connected = infinispan.client(
  {port: 11222, host: '127.0.0.1'}
  {
    // Configure client connections with authentication and encryption here.
  }
);

connected.then(function (client) {

    var clientAddListenerCreate = client.addListener('create', onCreate);
```

```
    var clientAddListeners = clientAddListenerCreate.then(
        function(listenerId) {
            // Associate multiple callbacks with a single client-side listener.
            // To do this, register listeners with the same listener ID.
            var clientAddListenerModify =
                client.addListener('modify', onModify, {listenerId: listenerId});

            var clientAddListenerRemove =
                client.addListener('remove', onRemove, {listenerId: listenerId});

            return Promise.all([clientAddListenerModify, clientAddListenerRemove]);
        });

    var clientCreate = clientAddListeners.then(
        function() { return client.putIfAbsent('eventful', 'v0'); });

    var clientModify = clientCreate.then(
        function() { return client.replace('eventful', 'v1'); });

    var clientRemove = clientModify.then(
        function() { return client.remove('eventful'); });

    var clientRemoveListener =
        Promise.all([clientAddListenerCreate, clientRemove]).then(
            function(values) {
                var listenerId = values[0];
                return client.removeListener(listenerId);
            });

    return clientRemoveListener.finally(
        function() { return client.disconnect(); });

}).catch(function(error) {

    console.log("Got error: " + error.message);

});

function onCreate(key, version) {
    console.log('[Event] Created key: ' + key +
        ' with version: ' + JSON.stringify(version));
}

function onModify(key, version) {
    console.log('[Event] Modified key: ' + key +
        ', version after update: ' + JSON.stringify(version));
}

function onRemove(key) {
    console.log('[Event] Removed key: ' + key);
}
```

イベントリスナーからの通知を調整して、**key-value-with-previous-converter-factory** コンバーターで不要なラウンドトリップを回避します。これにより、たとえば、後で取得するのではなく、イベント内のキーに関連付けられた値を見つけることができます。

リモートイベントコンバーター

```
var infinispan = require('infinispan');

var connected = infinispan.client(
   {port: 11222, host: '127.0.0.1'}
   , {
      dataFormat : {
         keyType: 'application/json',
         valueType: 'application/json'
      }
   }
);

connected.then(function (client) {
   // Include the remote event converter to avoid unnecessary roundtrips.
   var opts = {
      converterFactory : {
         name: "key-value-with-previous-converter-factory"
      }
   };

   var clientAddListenerCreate = client.addListener('create', logEvent("Created"), opts);

   var clientAddListeners = clientAddListenerCreate.then(
      function(listenerId) {
         // Associate multiple callbacks with a single client-side listener.
         // To do this, register listeners with the same listener ID.
         var clientAddListenerModify =
            client.addListener('modify', logEvent("Modified"), {opts, listenerId: listenerId});

         var clientAddListenerRemove =
            client.addListener('remove', logEvent("Removed"), {opts, listenerId: listenerId});

         return Promise.all([clientAddListenerModify, clientAddListenerRemove]);
      });

   var clientCreate = clientAddListeners.then(
      function() { return client.putIfAbsent('converted', 'v0'); });

   var clientModify = clientCreate.then(
      function() { return client.replace('converted', 'v1'); });

   var clientRemove = clientModify.then(
      function() { return client.remove('converted'); });

   var clientRemoveListener =
      Promise.all([clientAddListenerCreate, clientRemove]).then(
         function(values) {
            var listenerId = values[0];
            return client.removeListener(listenerId);
         });

   return clientRemoveListener.finally(
      function() { return client.disconnect(); });
```

```
}).catch(function(error) {

    console.log("Got error: " + error.message);

});

function logEvent(prefix) {
    return function(event) {
        console.log(prefix + " key: " + event.key);
        console.log(prefix + " value: " + event.value);
        console.log(prefix + " previous value: " + event.prev);
    }
}
```

ヒント

カスタムコンバーターを Data Grid Server に追加できます。詳細は、Data Grid のドキュメント を参照してください。

2.1.7. 条件操作の使用

Hot Rod プロトコルは、Data Grid に値に関するメタデータを保存します。このメタデータは、特定の条件に対してキャッシュ操作を実行できる決定論的な要因を提供します。たとえば、バージョンに一致しない場合はキーの値を置き換えることができます。

getWithMetadata メソッドを使用して、キーの値に関連付けられたメタデータを取得します。

```
var infinispan = require('infinispan');

var connected = infinispan.client(
  {port: 11222, host: '127.0.0.1'}
  {
    // Configure client connections with authentication and encryption here.
  }
);

connected.then(function (client) {

  var clientPut = client.putIfAbsent('cond', 'v0');

  var showPut = clientPut.then(
     function(success) { console.log(':putIfAbsent(cond)=' + success); });

  var clientReplace = showPut.then(
     function() { return client.replace('cond', 'v1'); } );

  var showReplace = clientReplace.then(
     function(success) { console.log('replace(cond)=' + success); });

  var clientGetMetaForReplace = showReplace.then(
     function() { return client.getWithMetadata('cond'); });

  // Call the getWithMetadata method to retrieve the value and its metadata.
  var clientReplaceWithVersion = clientGetMetaForReplace.then(
     function(entry) {
```

```
      console.log('getWithMetadata(cond)=' + JSON.stringify(entry));
      return client.replaceWithVersion('cond', 'v2', entry.version);
    }
  );

  var showReplaceWithVersion = clientReplaceWithVersion.then(
    function(success) { console.log('replaceWithVersion(cond)=' + success); });

  var clientGetMetaForRemove = showReplaceWithVersion.then(
    function() { return client.getWithMetadata('cond'); });

  var clientRemoveWithVersion = clientGetMetaForRemove.then(
    function(entry) {
      console.log('getWithMetadata(cond)=' + JSON.stringify(entry));
      return client.removeWithVersion('cond', entry.version);
    }
  );

  var showRemoveWithVersion = clientRemoveWithVersion.then(
    function(success) { console.log('removeWithVersion(cond)=' + success)});

  return showRemoveWithVersion.finally(
    function() { return client.disconnect(); });

}).catch(function(error) {

  console.log("Got error: " + error.message);

});
```

2.1.8. 一時データの使用

getWithMetadata メソッドおよび **size** メソッドを使用してキャッシュエントリーを失効させます。

```
var infinispan = require('infinispan');

var connected = infinispan.client(
  {port: 11222, host: '127.0.0.1'}
  {
    // Configure client connections with authentication and encryption here.
  }
);

connected.then(function (client) {

  var clientPutExpiry = client.put('expiry', 'value', {lifespan: '1s'});

  var clientGetMetaAndSize = clientPutExpiry.then(
    function() {
      // Compute getWithMetadata and size in parallel.
      return Promise.all([client.getWithMetadata('expiry'), client.size()]);
    });

  var showGetMetaAndSize = clientGetMetaAndSize.then(
    function(values) {
```

```
      console.log('Before expiration:');
      console.log('getWithMetadata(expiry)=' + JSON.stringify(values[0]));
      console.log('size=' + values[1]);
    });

  var clientContainsAndSize = showGetMetaAndSize.then(
    function() {
      sleepFor(1100); // Sleep to force expiration.
      return Promise.all([client.containsKey('expiry'), client.size()]);
    });

  var showContainsAndSize = clientContainsAndSize.then(
    function(values) {
      console.log('After expiration:');
      console.log('containsKey(expiry)=' + values[0]);
      console.log('size=' + values[1]);
    });

  return showContainsAndSize.finally(
    function() { return client.disconnect(); });

}).catch(function(error) {

  console.log("Got error: " + error.message);

});

function sleepFor(sleepDuration){
  var now = new Date().getTime();
  while(new Date().getTime() < now + sleepDuration){ /* Do nothing. */ }
}
```