Red Hat Data Grid 8.5

Data Grid Spring Boot スターター

Spring Boot プロジェクトでの Data Grid の使用

Last Updated: 2024-08-12

Red Hat Data Grid 8.5 Data Grid Spring Boot スターター

Spring Boot プロジェクトでの Data Grid の使用

法律上の通知

概要

Spring Boot プロジェクトが Data Grid とシームレスに対話するために必要なすべてを含む一連の管理された推移的な依存関係を使用して、Spring Boot プロジェクトをすばやく起動して実行します。Data Grid Spring Boot スターターは、Spring Boot を開始するための便利な方法を提供しますが、オプションであることに注意してください。必要な依存関係を追加するだけで、Spring Boot で Data Grid を使用できます。

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。

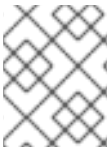
DATA GRID のドキュメント

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 8.5 ドキュメント
- Data Grid 8.5 コンポーネントの詳細
- Data Grid 8.5 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 を参照してください。

第1章 埋め込みキャッシュの使用

インメモリーデータストレージ向けに Data Grid キャッシュをプロジェクトに直接埋め込みます。

1.1. EMBEDDEDCACHEMANAGER BEAN の追加

埋め込みキャッシュを使用するようにアプリケーションを設定します。

手順

1. **infinispan-spring-boot3-starter-embedded** をプロジェクトのクラスパスに追加して、埋め込みモードを有効にします。
2. 次の例のように、Spring の **@Autowired** アノテーションを使用して、Java 設定クラスに **EmbeddedCacheManagerBean** を含めます。

```
private final EmbeddedCacheManager cacheManager;

@Autowired
public YourClassName(EmbeddedCacheManager cacheManager) {
   this.cacheManager = cacheManager;
}
```

これで、次の例のように、アプリケーション内でデータグリッドキャッシュを直接使用する準備が整いました。

```
cacheManager.getCache("testCache").put("testKey", "testValue");
System.out.println("Received value from cache: " +
cacheManager.getCache("testCache").get("testKey"));
```

1.2. REACTOR でリアクティブモードを使用する

Spring 6.1 以降では、リアクティブアプリケーション内でキャッシュを利用するためにリアクティブモードがサポートされています。**spring-boot-starter-webflux** を使用すると、アプリケーションがブロックされる可能性があります。

Data Grid リアクティブドライバーを有効にするには、**application.properties** で次のプロパティーを指定します。

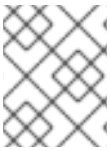
```
infinispan.embedded.reactive=true
```

1.3. キャッシュマネージャーの設定 BEAN

次の設定 Bean を使用して Cache Manager をカスタマイズできます。

- **InfinispanGlobalConfigurer**
- **InfinispanCacheConfigurer**
- **Configuration**
- **InfinispanConfigurationCustomizer**

- **InfinispanGlobalConfigurationCustomizer**



注記

InfinispanGlobalConfigurerBean は 1 つしか作成できません。ただし、他の Bean を使用して複数の設定を作成できます。

InfinispanCacheConfigurer Bean

```
@Bean
public InfinispanCacheConfigurer cacheConfigurer() {
 return manager -> {
  final Configuration ispnConfig = new ConfigurationBuilder()
                    .clustering()
                    .cacheMode(CacheMode.LOCAL)
                    .build();

  manager.defineConfiguration("local-sync-config", ispnConfig);
 };
}
```

設定 Bean

次のように、設定するキャッシュに Bean 名をリンクします。

```
@Bean(name = "small-cache")
public org.infinispan.configuration.cache.Configuration smallCache() {
    return new ConfigurationBuilder()
        .read(baseCache)
        .memory().size(1000L)
        .memory().evictionType(EvictionType.COUNT)
        .build();
}

@Bean(name = "large-cache")
public org.infinispan.configuration.cache.Configuration largeCache() {
    return new ConfigurationBuilder()
        .read(baseCache)
        .memory().size(2000L)
        .build();
}
```

カスタマイザー Bean

```
@Bean
public InfinispanGlobalConfigurationCustomizer globalCustomizer() {
   return builder -> builder.transport().clusterName(CLUSTER_NAME);
}

@Bean
public InfinispanConfigurationCustomizer configurationCustomizer() {
   return builder -> builder.memory().evictionType(EvictionType.COUNT);
}
```

1.4. SPRING キャッシュサポートの有効化

Data Grid は、埋め込みキャッシュとリモートキャッシュの両方を使用して、有効にできる Spring Cache の実装を提供します。

手順

- **@EnableCaching** アノテーションをアプリケーションに追加します。

Data Grid スターターが以下を検出した場合:

- **EmbeddedCacheManager** Bean の場合は、新しい **SpringEmbeddedCacheManager** をインスタンス化します。
- **RemoteCacheManager** Bean の場合は、新しい **SpringRemoteCacheManager** をインスタンス化します。

参考資料

Spring キャッシュリファレンス

第2章 リモートキャッシュの使用

カスタム TCP バイナリーワイヤプロトコルである Hot Rod を使用して、リモートの Data Grid クラスターからデータを保存および取得します。

2.1. REMOTECACHEMANAGER の設定

Data Grid クラスターでリモートキャッシュを使用するようにアプリケーションを設定します。

1. スターターが **RemoteCacheManagerBean** を作成できるように、Data Grid Server がクライアント接続をリッスンするアドレスを指定します。
2. Spring **@Autowired** アノテーションを使用して、独自のカスタム Cache Manager クラスをアプリケーションに含めます。

```
private final RemoteCacheManager cacheManager;

@Autowired
public YourClassName(RemoteCacheManager cacheManager) {
   this.cacheManager = cacheManager;
}
```

2.2. REACTOR でリアクティブモードを使用する

Spring 6.1 以降では、リアクティブアプリケーション内でキャッシュを利用するためにリアクティブモードがサポートされています。**spring-boot-starter-webflux** を使用すると、アプリケーションがブロックされる可能性があります。

Data Grid リアクティブドライバーを有効にするには、**application.properties** で次のプロパティーを指定します。

```
infinispan.remote.reactive=true
```

2.2.1. プロパティーファイル

hotrod-client.properties または **application.properties** のいずれかでプロパティーを指定できます。

プロパティーは、両方のプロパティーファイルに含めることができますが、スターターは最初に **hotrod-client.properties** の設定を適用するので、**application.properties** よりもファイルが優先されます。

hotrod-client.properties
このファイルのプロパティーは、**infinispan.client.hotrod.*** の形式を取ります。次に例を示します。

```
# List Data Grid servers by IP address or hostname at port localhost:11222.
infinispan.client.hotrod.server_list=127.0.0.1:11222
```

- Hot Rod クライアント設定 API

application.properties
このファイルのプロパティーは、**infinispan.remote.*** の形式を取ります。次に例を示します。

```
# List Data Grid servers by IP address or hostname at port localhost:11222.
infinispan.remote.server-list=127.0.0.1:11222
```

関連情報

- アプリケーションのプロパティー

2.3. マーシャリングの設定

Java オブジェクトをバイナリー形式にマーシャリングするように Data Grid を設定して、ネットワーク経由で転送したり、ディスクに保存したりできるようにします。

デフォルトでは、Data Grid は Java シリアル化マーシャラーを使用するため、クラスを許可リストに追加する必要があります。別の方法として、ProtoStream を使用できます。これには、クラスにアノテーションを付けて、カスタム Java オブジェクトの **SerializationContextInitializer** を生成する必要があります。

手順

1. **hotrod-client.properties** または **application.properties** を開いて編集します。
2. 次のいずれかを行います。
   - マーシャラーとして ProtoStream を使用します。

     ```
     infinispan.client.hotrod.marshaller=org.infinispan.commons.marshall.ProtoStreamMarshaller
     ```

     ```
     infinispan.remote.marshaller=org.infinispan.commons.marshall.ProtoStreamMarshaller
     ```

   - Java シリアル化を使用する場合は、使用するクラスをシリアル化許可リストに追加します。完全修飾クラス名のコンマ区切りリストまたはクラスを照合するための正規表現を指定できます。

     ```
     infinispan.client.hotrod.java_serial_allowlist=your_marshalled_beans_package.*
     ```

     ```
     infinispan.remote.java-serial-allowlist=your_marshalled_beans_package.*
     ```

3. プロパティーファイルを保存して閉じます。

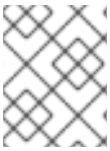
関連情報

- キャッシュのエンコードとマーシャリング

2.4. キャッシュマネージャーの設定 BEAN

次の設定 Bean を使用して Cache Manager をカスタマイズします。

- **InfinispanRemoteConfigurer**
- **Configuration**
- **InfinispanRemoteCacheCustomizer**



注記

InfinispanRemoteConfigurerBean は 1 つしか作成できません。ただし、他の Bean を使用して複数の設定を作成できます。

InfinispanRemoteConfigurer Bean

```
@Bean
public InfinispanRemoteConfigurer infinispanRemoteConfigurer() {
   return () -> new ConfigurationBuilder()
      .addServer()
      .host("127.0.0.1")
      .port(12345)
      .build();
}
```

設定 Bean

```
@Bean
public org.infinispan.client.hotrod.configuration.Configuration customConfiguration() {
   new ConfigurationBuilder()
      .addServer()
      .host("127.0.0.1")
      .port(12345)
      .build();
}
```

InfinispanRemoteCacheCustomizer Bean

```
@Bean
public InfinispanRemoteCacheCustomizer customizer() {
   return b -> b.tcpKeepAlive(false);
}
```

ヒント

@Ordered アノテーションを使用して、カスタマイザーを特定の順序で適用します。

2.5. SPRING キャッシュサポートの有効化

Data Grid は、埋め込みキャッシュとリモートキャッシュの両方を使用して、有効にできる Spring Cache の実装を提供します。

手順

- **@EnableCaching** アノテーションをアプリケーションに追加します。

Data Grid スターターが以下を検出した場合:

- **EmbeddedCacheManager** Bean の場合は、新しい **SpringEmbeddedCacheManager** をインスタンス化します。

- **RemoteCacheManager** Bean の場合は、新しい **SpringRemoteCacheManager** をインスタンス化します。

参考資料

Spring キャッシュリファレンス

2.6. DATA GRID 統計の公開

Data Grid は、Spring Boot Actuator をサポートして、キャッシュ統計をメトリックとして公開します。

手順

1. **pom.xml** ファイルに以下を追加します。

```
<dependency>
  <groupId>org.springframework.boot</groupId>
  <artifactId>spring-boot-starter-actuator</artifactId>
  <version>${version.spring.boot}</version>
 </dependency>

<dependency>
  <groupId>org.springframework.boot</groupId>
  <artifactId>spring-boot-starter-web</artifactId>
  <version>${version.spring.boot}</version>
</dependency>
```

2. プログラムまたは宣言を使用して、適切なキャッシュインスタンスの統計を有効にします。

プログラム

```
@Bean
public InfinispanCacheConfigurer cacheConfigurer() {
  return cacheManager -> {
    final org.infinispan.configuration.cache.Configuration config =
          new ConfigurationBuilder()
              .jmxStatistics().enable()
              .build();

    cacheManager.defineConfiguration("my-cache", config);
  };
}
```

宣言

```
<local-cache statistics="true"/>
```

Spring Boot Actuator レジストリーは、アプリケーションの起動時にキャッシュインスタンスをバインドします。

キャッシュを動的に作成する場合は、次のように、**CacheMetricsRegistrarBean** を使用してキャッシュを Actuator レジストリーにバインドする必要があります。

```
@Autowire
CacheMetricsRegistrar cacheMetricsRegistrar;

@Autowire
CacheManager cacheManager;
...

cacheMetricsRegistrar.bindCacheToRegistry(cacheManager.getCache("my-cache"));
```

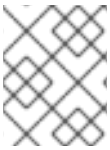
第3章 SPRING SESSION の使用

3.1. SPRING SESSION サポートの有効化

Data Grid Spring Session サポートは **SpringRemoteCacheManager** と **SpringEmbeddedCacheManager** をもとに構築されます。Data Grid スターターは、デフォルトでこれらの Bean を生成します。

手順

1. このスターターをプロジェクトに追加します。
2. Spring Session をクラスパスに追加します。
3. 次のアノテーションを設定に追加します。
    - **@EnableCaching**
    - **@EnableInfinispanRemoteHttpSession**
    - **@EnableInfinispanEmbeddedHttpSession**



注記

Data Grid にはデフォルトのキャッシュがありません。Spring Session を使用するには、最初に Data Grid キャッシュを作成する必要があります。

第4章 アプリケーションのプロパティー

application.properties または **application.yaml** を使用してプロジェクトを設定します。

```
#
# Embedded Properties - Uncomment properties to use them.
#

# Enables embedded capabilities in your application.
# Values are true (default) or false.
#infinispan.embedded.enabled =

# Sets the Spring state machine ID.
#infinispan.embedded.machineId =

# Sets the name of the embedded cluster.
#infinispan.embedded.clusterName =

# Specifies a XML configuration file that takes priority over the global
# configuration bean or any configuration customizer.
#infinispan.embedded.configXml =

#
# Server Properties - Uncomment properties to use them.
#

# Specifies a custom filename for Hot Rod client properties.
#infinispan.remote.clientProperties =

# Enables remote server connections.
# Values are true (default) or false.
#infinispan.remote.enabled =

# Defines a comma-separated list of servers in this format:
# `host1[:port],host2[:port]`.
#infinispan.remote.server-list=

# Sets a timeout value, in milliseconds, for socket connections.
#infinispan.remote.socketTimeout =

# Sets a timeout value for initializing connections with servers.
#infinispan.remote.connectTimeout =

# Sets the maximum number of attempts to connect to servers.
#infinispan.remote.maxRetries =

# Specifies the marshaller to use.
#infinispan.remote.marshaller =

# Adds your classes to the serialization allow list.
#infinispan.remote.java-serial-allowlist=
```