



Red Hat Data Grid 8.1

Data Grid Spring Boot スターター

SpringBoot プロジェクトでの Data Grid の使用

Red Hat Data Grid 8.1 Data Grid Spring Boot スターター

SpringBoot プロジェクトでの Data Grid の使用

Enter your first name here. Enter your surname here.

Enter your organisation's name here. Enter your organisational division here.

Enter your email address here.

法律上の通知

Copyright © 2022 | You need to change the HOLDER entity in the en-US/Data_Grid_Spring_Boot_Starter.ent file |.

The text of and illustrations in this document are licensed by Red Hat under a Creative Commons Attribution–Share Alike 3.0 Unported license ("CC-BY-SA"). An explanation of CC-BY-SA is available at

<http://creativecommons.org/licenses/by-sa/3.0/>

. In accordance with CC-BY-SA, if you distribute this document or an adaptation of it, you must provide the URL for the original version.

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

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

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

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

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

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

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

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

All other trademarks are the property of their respective owners.

概要

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のものを含む、管理対象の推移的依存関係のセットを使用して、Spring Boot プロジェクトをすばやく稼働させます。Data Grid Spring Boot スターターを使用すると、Spring Boot を使い始めると便利ですが、任意となります。Spring Boot で Data Grid を使用するには、必要な依存関係を追加だけです。

目次

第1章 RED HAT DATA GRID	3
1.1. DATA GRID のドキュメント	3
1.2. DATA GRID のダウンロード	3
1.3. 多様性を受け入れるオープンソースの強化	3
第2章 プロジェクトの設定	4
2.1. DATA GRID バージョンの適用	4
2.2. 使用モードの依存関係の追加	4
第3章 組み込みモードでの実行	6
3.1. EMBEDDED CACHEMANAGER BEAN の追加	6
3.2. キャッシュマネージャーの設定 BEAN	6
3.3. SPRING キャッシュサポートの有効化	7
第4章 サーバーモードでの実行	8
4.1. REMOTE CACHEMANAGER の設定	8
4.2. マーシャリングの構成	8
4.3. キャッシュマネージャーの設定 BEAN	9
4.4. SPRING キャッシュサポートの有効化	9
4.5. DATA GRID 統計の公開	10
第5章 SPRING SESSION の使用	12
5.1. SPRING SESSION サポートの有効化	12
第6章 アプリケーションのプロパティ	13

第1章 RED HAT DATA GRID

Data Grid は、高性能の分散型インメモリデータストアです。

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ルリング

サービスを中断することなく、ノードの数を動的に調整して要件を満たします。

データの相互運用性

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

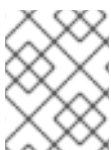
1.1. DATA GRID のドキュメント

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

- [Data Grid 8.1 ドキュメント](#)
- [Data Grid 8.1 コンポーネントの詳細](#)
- [Data Grid 8.1 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](#)

1.2. DATA GRID のダウンロード

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



注記

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

1.3. 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。これは大規模な取り組みであるため、これらの変更は今後の複数のリリースで段階的に実施されます。詳細は、[弊社 の CTO、Chris Wright のメッセージ](#)を参照してください。

第2章 プロジェクトの設定

Data Grid Spring Boot スターターの依存関係をプロジェクトに追加します。

2.1. DATA GRID バージョンの適用

このスターターは、高レベルの API を使用して、Data Grid のメジャーバージョン間の互換性を確保します。ただし、**infinispan-bom** モジュールを使用して特定のバージョンの Data Grid を適用できます。

以下のように、スターターの依存関係の前に **infinispan-bom** を **pom.xml** ファイルに追加します。

```
<dependencyManagement>
  <dependencies>
    <dependency>
      <groupId>org.infinispan</groupId>
      <artifactId>infinispan-bom</artifactId>
      <version>${version.infinispan}</version>
      <type>pom</type>
      <scope>import</scope>
    </dependency>
    <dependency>
      <groupId>org.springframework.boot</groupId>
      <artifactId>spring-boot-starter-parent</artifactId>
      <version>${version.spring.boot}</version>
      <type>pom</type>
      <scope>import</scope>
    </dependency>
    <dependency>
      <groupId>org.infinispan</groupId>
      <artifactId>infinispan-spring-boot-starter</artifactId>
      <version>2.3.6.Final-redhat-00001</version>
    </dependency>
  </dependencies>
</dependencyManagement>
```

重要

Data Grid Spring Boot スターターは、Red Hat OpenShift Application Runtimes などの他のプロジェクトとは異なる Spring Boot バージョンを使用します。他のプロジェクトと互換性を確保するために特定の Spring Boot バージョンを使用する場合は、プロジェクトに正しい依存関係を追加する必要があります。

2.2. 使用モードの依存関係の追加

Data Grid は、使用モードごとに異なる依存関係を提供します。次のいずれかを **pom.xml** ファイルに追加します。

組み込みモード

```
<dependency>
  <groupId>org.infinispan</groupId>
  <artifactId>infinispan-spring-boot-starter-embedded</artifactId>
```



```
<version>2.3.6.Final-redhat-00001</version>  
</dependency>
```

リモートクライアント/サーバーモード

```
<dependency>  
  <groupId>org.infinispan</groupId>  
  <artifactId>infinispan-spring-boot-starter-remote</artifactId>  
  <version>2.3.6.Final-redhat-00001</version>  
</dependency>
```

第3章 組み込みモードでの実行

インメモリーデータストレージ用に、プロジェクトに Data Grid ライブラリーを埋め込みます。

3.1. EMBEDDEDCACHEMANAGER BEAN の追加

1. **infinispan-spring-boot-starter-embedded** をプロジェクトのクラスパスに追加して、組み込みモードを有効にします。
このスターターは、デフォルトでクラスパス上の **infinispan-spring-boot-starter-remote** で、リモートクライアント/サーバーモードで動作します。
2. 次の例のように、Spring **@Autowired** アノテーションを使用して、Java 構成クラスに **EmbeddedCacheManagerBean** を含めます。

```
private final EmbeddedCacheManager cacheManager;

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

これで、組み込みモードで Data Grid を使用できるようになりました。以下は簡単な例です。

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

3.2. キャッシュマネージャーの設定 BEAN

次の設定 Bean を使用してキャッシュマネージャーをカスタマイズできます。

- **InfinispanGlobalConfigurer**
- **InfinispanCacheConfigurer**
- 設定
- **InfinispanConfigurationCustomizer**
- **InfinispanGlobalConfigurationCustomizer**



注記

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

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);
}

```

3.3. SPRING キャッシュサポートの有効化

@EnableCaching アノテーションをアプリケーションに追加し、Spring Cache のサポートを有効にします。

このスターターが **EmbeddedCacheManager** Bean を検出すると、[Spring Cache](#) の実装を提供する新しい **SpringEmbeddedCacheManager** をインスタンス化します。

第4章 サーバーモードでの実行

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。

4.1. REMOTECACHEMANAGER の設定

1. スターターが **RemoteCacheManager** Bean を作成できるように Data Grid サーバーの場所を指定します。
スターターは、まず **hotrod-client.properties** でサーバーの場所を見つけようとし、次に **application.properties** からサーバーの場所を見つけようとします。
2. Spring **@Autowired** アノテーションを使用して、独自のカスタムキャッシュマネージャー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;
}
```

Hot Rod クライアントプロパティ

以下のように、クラスパスの **hotrod-client.properties** でクライアント設定を指定します。

```
# List Infinispan or Data Grid servers by IP address or hostname at port 11222.
infinispan.client.hotrod.server_list=127.0.0.1:6667
```

詳細は、[org.infinispan.client.hotrod.configuration](https://www.infinispan.org/documentation/8.1/en/reference/appendixes/hotrod-client-configuration.html) を参照してください。

アプリケーションプロパティ

application.properties でプロジェクトを設定します。詳細は、「[アプリケーションプロパティ](#)」を参照してください。

4.2. マーシャリングの構成

Java シリアライゼーションを使用してオブジェクトをマーシャリングするように Data Grid サーバーを設定します。

デフォルトでは、Data Grid サーバーは ProtoStream シリアライゼーションライブラリーをデフォルトのマーシャラーとして使用します。ただし、ProtoStream マーシャラーは Spring インテグレーションではサポートされません。このため、Java Serialization Marshaller を使用する必要があります。

- **application.properties** に以下のプロパティを指定します。

```
infinispan.remote.marshaller=org.infinispan.commons.marshall.JavaSerializationMarshaller
1
infinispan.remote.java-serial-whitelist=your_marshalled_beans_package.* 2
```

1 Java Serialization Marshaller を使用します。

2 クラスをシリアライズホワイトリストに追加し、Data Grid がオブジェクトをマーシャリングできるようにします。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会するための正規表現

るようになります。元々修飾子「protected」は、パッケージ内またはパッケージで派生するクラスのみにアクセス可能を指定できます。

4.3. キャッシュマネージャーの設定 BEAN

次の設定 Bean を使用してキャッシュマネージャーをカスタマイズします。

- **InfinispanRemoteConfigurer**
- 設定
- **InfinispanRemoteCacheCustomizer**



注記

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

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。

4.4. SPRING キャッシュサポートの有効化

@EnableCaching アノテーションをアプリケーションに追加し、Spring Cache のサポートを有効にします。

Data Grid スターターが **RemoteCacheManager** Bean を検出すると、[Spring Cache](#) の実装を提供する新しい **SpringRemoteCacheManager** をインスタンス化します。

4.5. 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クスとして公開します。

Actuator を使用するには、以下を **pom.xml** ファイルに追加します。

```
<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>
```

続いて、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する必要があります。

プログラムで

```
@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 name="my-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。

```
@Autowired
CacheMetricsRegistrar cacheMetricsRegistrar;

@Autowired
```

```
CacheManager cacheManager;
```

```
...
```

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

第5章 SPRINGSESSION の使用

5.1. 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。このスターターは、デフォルトでこれらの Bean を生成します。

プロジェクトで Spring Sessionを使用するには、以下を実行します。

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**

第6章 アプリケーションのプロパティ

application.properties または **application.yaml** を使用してプロジェクトを構成します。

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

#
# 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.serverList =

# 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 whitelist.
#infinispan.remote.java-serial-whitelist=
```

