Red Hat Data Grid 8.3

Using Data Grid with Spring

Add Data Grid to Spring applications

Last Updated: 2023-11-24

Red Hat Data Grid 8.3 Using Data Grid with Spring

Add Data Grid to Spring applications

Legal Notice

Abstract

Add Data Grid caching capabilities to Spring-based applications.

Table of Contents

RED HAT DATA GRID

Data Grid is a high-performance, distributed in-memory data store.

Schemaless data structure

Flexibility to store different objects as key-value pairs.

Grid-based data storage

Designed to distribute and replicate data across clusters.

Elastic scaling

Dynamically adjust the number of nodes to meet demand without service disruption.

Data interoperability

Store, retrieve, and query data in the grid from different endpoints.

DATA GRID DOCUMENTATION

Documentation for Data Grid is available on the Red Hat customer portal.

- Data Grid 8.3 Documentation
- Data Grid 8.3 Component Details
- Supported Configurations for Data Grid 8.3
- Data Grid 8 Feature Support
- Data Grid Deprecated Features and Functionality

DATA GRID DOWNLOADS

Access the Data Grid Software Downloads on the Red Hat customer portal.

NOTE

You must have a Red Hat account to access and download Data Grid software.

MAKING OPEN SOURCE MORE INCLUSIVE

Red Hat is committed to replacing problematic language in our code, documentation, and web properties. We are beginning with these four terms: master, slave, blacklist, and whitelist. Because of the enormity of this endeavor, these changes will be implemented gradually over several upcoming releases. For more details, see our CTO Chris Wright's message.

CHAPTER 1. USING DATA GRID AS A SPRING CACHE PROVIDER

Add Data Grid dependencies to your application and use Spring Cache annotations to store data in embedded or remote caches.

1.1. SETTING UP SPRING CACHING WITH DATA GRID

Add the Data Grid dependencies to your Spring application project. If you use remote caches in a Data Grid Server deployment, you should also configure your Hot Rod client properties.

Procedure

1. Add Data Grid and the Spring integration module to your **pom.xml**.
   - Remote caches: **infinispan-spring5-remote**
   - Embedded caches: **infinispan-spring5-embedded**

   TIP

   Spring Boot users can add the **infinispan-spring-boot-starter-embedded** instead of the **infinispan-spring5-embedded** artifact.

2. Configure your Hot Rod client to connect to your Data Grid Server deployment in the **hotrod-client.properties** file.

```
infinispan.client.hotrod.server_list = 127.0.0.1:11222
infinispan.client.hotrod.auth_username=admin
infinispan.client.hotrod.auth_password=changeme
```

Spring Cache dependencies

Remote caches

```
<dependencies>
   <dependency>
      <groupId>org.infinispan</groupId>
      <artifactId>infinispan-spring5-remote</artifactId>
   </dependency>
   <dependency>
      <groupId>org.springframework</groupId>
      <artifactId>spring-context</artifactId>
      <version>${version.spring}</version>
   </dependency>
</dependencies>
```

Embedded caches

```
<dependencies>
   <dependency>
      <groupId>org.infinispan</groupId>
```

```
      <artifactId>infinispan-spring5-embedded</artifactId>
    </dependency>
    <dependency>
      <groupId>org.springframework</groupId>
      <artifactId>spring-context</artifactId>
      <version>${version.spring}</version>
    </dependency>
  </dependencies>
```

Additional resources

- Configuring Hot Rod Client connections

1.2. USING DATA GRID AS A SPRING CACHE PROVIDER

Add the **@EnableCaching** annotation to one of your configuration classes and then add the **@Cacheable** and **@CacheEvict** annotations to use remote or embedded caches.

Prerequisites

- Add the Data Grid dependencies to your application project.
- Create the required remote caches and configure Hot Rod client properties if you use a Data Grid Server deployment.

Procedure

1. Enable cache annotations in your application context in one of the following ways:

   Declarative

   ```
   <beans xmlns="http://www.springframework.org/schema/beans"
      xmlns:xsi="http://www.w3.org/2001/XMLSchema-instance"
      xmlns:cache="http://www.springframework.org/schema/cache"
      xmlns:p="http://www.springframework.org/schema/p"
      xsi:schemaLocation="
          http://www.springframework.org/schema/beans
   http://www.springframework.org/schema/beans/spring-beans.xsd
          http://www.springframework.org/schema/cache
   http://www.springframework.org/schema/cache/spring-cache.xsd">

          <cache:annotation-driven />

   </beans>
   ```

   Programmatic

   ```
   @EnableCaching @Configuration
   public class Config {
   }
   ```

2. Annotate methods with **@Cacheable** to cache return values.

TIP

To reference entries in the cache directly, you must include the **key** attribute.

3. Annotate methods with **@CacheEvict** to remove old entries from the cache.

Additional resources

- Spring Framework - Default Key Generation

1.3. SPRING CACHE ANNOTATIONS

The **@Cacheable** and **@CacheEvict** annotations add cache capabilities to methods.

@Cacheable

Stores return values in a cache.

@CacheEvict

Controls cache size by removing old entries.

@Cacheable

Taking **Book** objects as an example, if you want to cache each instance after loading it from a database with a method such as **BookDao#findBook(Integer bookId)**, you could add the **@Cacheable** annotation as follows:

```
@Transactional
@Cacheable(value = "books", key = "#bookId")
public Book findBook(Integer bookId) {...}
```

With the preceding example, when **findBook(Integer bookId)** returns a **Book** instance it gets stored in the cache named **books**.

@CacheEvict

With the **@CacheEvict** annotation, you can specify if you want to evict the entire **books** cache or only the entries that match a specific #bookId.

Entire cache eviction

Annotate the **deleteAllBookEntries()** method with **@CacheEvict** and add the **allEntries** parameter as follows:

```
@Transactional
@CacheEvict (value="books", key = "#bookId", allEntries = true)
public void deleteAllBookEntries() {...}
```

Entry based eviction

Annotate the **deleteBook(Integer bookId)** method with **@CacheEvict** and specify the key associated to the entry as follows:

```
@Transactional
@CacheEvict (value="books", key = "#bookId")
public void deleteBook(Integer bookId) {...}
```

1.4. CONFIGURING TIMEOUTS FOR CACHE OPERATIONS

The Data Grid Spring Cache provider defaults to blocking behaviour when performing read and write operations. Cache operations are synchronous and do not time out.

If necessary you can configure a maximum time to wait for operations to complete before they time out.

Procedure

- Configure the following timeout properties in the context XML for your application on either **SpringEmbeddedCacheManagerFactoryBean** or **SpringRemoteCacheManagerFactoryBean**.
  For remote caches, you can also add these properties to the **hotrod-client.properties** file.

Property	Description
infinispan.spring.operation.read.timeout	Specifies the time, in milliseconds, to wait for read operations to complete. The default is **0** which means unlimited wait time.
infinispan.spring.operation.write.timeout	Specifies the time, in milliseconds, to wait for write operations to complete. The default is **0** which means unlimited wait time.

The following example shows the timeout properties in the context XML for **SpringRemoteCacheManagerFactoryBean**:

```
<bean id="springRemoteCacheManagerConfiguredUsingConfigurationProperties"
      class="org.infinispan.spring.remote.provider.SpringRemoteCacheManagerFactoryBean">
   <property name="configurationProperties">
      <props>
         <prop key="infinispan.spring.operation.read.timeout">500</prop>
         <prop key="infinispan.spring.operation.write.timeout">700</prop>
      </props>
   </property>
</bean>
```

CHAPTER 2. EXTERNALIZING SESSIONS WITH SPRING SESSION

Store session data for Spring applications in Data Grid caches and independently of the container.

2.1. EXTERNALIZING SESSIONS WITH SPRING SESSION

Use the Spring Session API to externalize session data to Data Grid.

Procedure

1. Add dependencies to your **pom.xml**.
   - Embedded caches: **infinispan-spring5-embedded**
   - Remote caches: **infinispan-spring5-remote**
     The following example is for remote caches:

```
<dependencies>
  <dependency>
    <groupId>org.infinispan</groupId>
    <artifactId>infinispan-core</artifactId>
  </dependency>
  <dependency>
    <groupId>org.infinispan</groupId>
    <artifactId>infinispan-spring5-remote</artifactId>
  </dependency>
  <dependency>
    <groupId>org.springframework</groupId>
    <artifactId>spring-context</artifactId>
    <version>${version.spring}</version>
  </dependency>
  <dependency>
    <groupId>org.springframework.session</groupId>
    <artifactId>spring-session-core</artifactId>
    <version>${version.spring}</version>
  </dependency>
  <dependency>
    <groupId>org.springframework</groupId>
    <artifactId>spring-web</artifactId>
    <version>${version.spring}</version>
  </dependency>
</dependencies>
```

2. Specify the appropriate **FactoryBean** to expose a **CacheManager** instance.
   - Embedded caches: **SpringEmbeddedCacheManagerFactoryBean**
   - Remote caches: **SpringRemoteCacheManagerFactoryBean**
3. Enable Spring Session with the appropriate annotation.
   - Embedded caches: **@EnableInfinispanEmbeddedHttpSession**
   - Remote caches: **@EnableInfinispanRemoteHttpSession**

These annotations have optional parameters:

- **maxInactiveIntervalInSeconds** sets session expiration time in seconds. The default is **1800**.
- **cacheName** specifies the name of the cache that stores sessions. The default is **sessions**.

The following example shows a complete, annotation-based configuration:

```
@EnableInfinispanEmbeddedHttpSession
@Configuration
public class Config {

  @Bean
  public SpringEmbeddedCacheManagerFactoryBean springCacheManager() {
    return new SpringEmbeddedCacheManagerFactoryBean();
  }

  //An optional configuration bean responsible for replacing the default
  //cookie that obtains configuration.
  //For more information refer to the Spring Session documentation.
  @Bean
  public HttpSessionIdResolver httpSessionIdResolver() {
     return HeaderHttpSessionIdResolver.xAuthToken();
  }
}
```