Using Infinispan with Spring

Table of Contents

Infinispan integrates with the Spring Framework to make it easy to add caching capabilities to your applications.

Chapter 1. Adding Infinispan to Spring applications

Infinispan implements the Spring SPI to offer high-performance, in-memory caching capabilities for Spring applications. You can use Infinispan as a Spring Cache provider and with the Spring Sessions API.

1.1. Adding Spring Cache support

To add caching support to your application, with the following annotations:

- `@Cacheable` adds entries to the cache.
- `@CacheEvict` removes entries from the cache.

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. Add Infinispan and the Spring integration module to your `pom.xml`.
   - Embedded caches: `infinispan-spring5-embedded`
   - Remote caches: `infinispan-spring5-remote`

   The following example is for embedded caches:

```
<dependencies>
    <dependency>
        <groupId>org.infinispan</groupId>
        <artifactId>infinispan-spring5-embedded</artifactId>
    </dependency>
    <!-- Tip: Use the Spring Boot starter
    instead of the spring-boot artifact. -->
    <dependency>
        <groupId>org.springframework</groupId>
        <artifactId>spring-context</artifactId>
        <version>${version.spring}</version>
    </dependency>
</dependencies>
```

1.2. Spring Cache annotations

Annotation application code with the `@Cacheable` and `@CacheEvict` annotations.

`@Cacheable`

The `@Cacheable` annotation adds returned values to a defined cache.

For instance, you have a data access object (DAO) for books. You want book instances to be cached after they have been loaded from the underlying database using `BookDao#findBook(Integer bookId)`.

Annotate the `findBook(Integer bookId)` method with `@Cacheable` as follows:

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

Any `Book` instances returned from `findBook(Integer bookId)` are stored in a cache named `books`, using `bookId` as the key.

Note that "#bookId" is an expression in the Spring Expression Language that evaluates the `bookId` argument.

If your application needs to reference entries in the cache directly, you should include the `key` attribute. For more information, see Default Key Generation in the Spring documentation.

`@CacheEvict`

The `@CacheEvict` annotation deletes entries from a defined cache.

Annotate the `deleteBook(Integer bookId)` method with `@CacheEvict` as follows:

```
// Evict all entries in the "books" cache
@Transactional
@CacheEvict (value="books", key = "#bookId", allEntries = true)
public void deleteBookAllEntries() {...}

// Evict entries in the "books" cache that match #bookId
@Transactional
@CacheEvict (value="books", key = "#bookId")
public void deleteBook(Integer bookId) {...]}
```

1.3. Configuring Timeouts for Cache Operations

The Infinispan 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>
```

1.4. Externalizing Sessions with Spring Session

Use the Spring Session API to externalize session data to Infinispan.

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>${version.spring}</version>
    </dependency>
    <dependency>
       <groupId>org.springframework</groupId>
       <artifactId>spring-session-core</artifactId>
       <version>${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 HttpSessionStrategy httpSessionStrategy() {
      return new HeaderHttpSessionStrategy();
   }
}
```