Infinispan Spring Boot Starter

Table of Contents

1. Setting Up Your Project . . . 2
1.1. Enforcing Infinispan Versions . . . 2
1.2. Adding Dependencies for Usage Modes . . . 2
2. Using Embedded Caches . . . 4
2.1. Adding the EmbeddedCacheManager Bean . . . 4
2.2. Cache Manager Configuration Beans . . . 4
2.3. Enabling Spring Cache Support . . . 6
3. Using Remote Caches . . . 7
3.1. Setting Up the RemoteCacheManager . . . 7
3.1.1. Properties Files . . . 7
3.2. Configuring Marshalling . . . 8
3.3. Cache Manager Configuration Beans . . . 8
3.4. Enabling Spring Cache Support . . . 9
3.5. Exposing Infinispan Statistics . . . 10
4. Using Spring Session . . . 12
4.1. Enabling Spring Session Support . . . 12
5. Application Properties . . . 13

The Infinispan starter provides a set of managed transitive dependencies that include everything your Spring Boot project needs to seamlessly interact with Infinispan.

The Infinispan Spring Boot starter gives you a convenient way to get started with Spring Boot but is optional. To use Infinispan with Spring Boot you can simply add the dependencies you want.

Chapter 1. Setting Up Your Project

Add dependencies for the Infinispan Spring Boot Starter to your project.

1.1. Enforcing Infinispan Versions

This starter uses a high-level API to ensure compatibility between major versions of Infinispan. However you can enforce a specific version of Infinispan with the `infinispan-bom` module.

Procedure

- Add `infinispan-bom` to your `pom.xml` file before the starter dependencies.

```
<properties>
  <version.infinispan>12.1.10.Final</version.infinispan>
</properties>

<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>
       </dependency>
    </dependencies>
 </dependencyManagement>
```

1.2. Adding Dependencies for Usage Modes

Infinispan provides different dependencies for embedded caches and remote caches.

Procedure

- Add one of the following to your `pom.xml` file:

Embedded caches

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

Remote caches

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

Chapter 2. Using Embedded Caches

Embed Infinispan caches directly in your project for in-memory data storage.

2.1. Adding the EmbeddedCacheManager Bean

Configure your application to use embedded caches.

Procedure

1. Add `infinispan-spring-boot-starter-embedded` to your project's classpath to enable Embedded mode.
2. Use the Spring `@Autowired` annotation to include an `EmbeddedCacheManager` bean in your Java configuration classes, as in the following example:

```
private final EmbeddedCacheManager cacheManager;

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

You are now ready to use Infinispan caches directly within your application, as in the following example:

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

2.2. Cache Manager Configuration Beans

You can customize the cache manager with the following configuration beans:

- `InfinispanGlobalConfigurer`
- `InfinispanCacheConfigurer`
- `Configuration`
- `InfinispanConfigurationCustomizer`
- `InfinispanGlobalConfigurationCustomizer`

You can create one `InfinispanGlobalConfigurer` bean only. However you can create multiple configurations with the other beans.

InfinispanCacheConfigurer Bean

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

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

Configuration Bean

Link the bean name to the cache that it configures, as follows:

```
@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();
}
```

Customizer Beans

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

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

2.3. Enabling Spring Cache Support

With both embedded and remote caches, Infinispan provides an implementation of Spring Cache that you can enable.

Procedure

- Add the `@EnableCaching` annotation to your application.

If the Infinispan starter detects the:

- `EmbeddedCacheManager` bean, it instantiates a new `SpringEmbeddedCacheManager`.
- `RemoteCacheManager` bean, it instantiates a new `SpringRemoteCacheManager`.

Reference

Spring Cache Reference

Chapter 3. Using Remote Caches

Store and retrieve data from remote Infinispan clusters using Hot Rod, a custom TCP binary wire protocol.

3.1. Setting Up the RemoteCacheManager

Configure your application to use remote caches on Infinispan clusters.

1. Provide the addresses where Infinispan Server listens for client connections so the starter can create the `RemoteCacheManager` bean.
2. Use the Spring `@Autowired` annotation to include your own custom cache manager class in your application:

```
private final RemoteCacheManager cacheManager;

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

3.1.1. Properties Files

You can specify properties in either `hotrod-client.properties` or `application.properties`.

Properties can be in both properties files but the starter applies the configuration in `hotrod-client.properties` first, which means that file takes priority over `application.properties`.

`hotrod-client.properties`

Properties in this file take the format of `infinispan.client.hotrod.*`, for example:

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

- Hot Rod client configuration API

`application.properties`

Properties in this file take the format of `infinispan.remote.*`, for example:

```
# List Infinispan servers by IP address or hostname at port 11222.
infinispan.remote.server-list=127.0.0.1:6667
```

Additional resources

- Application Properties

3.2. Configuring Marshalling

Configure Infinispan to marshall Java objects into binary format so they can be transferred over the wire or stored to disk.

By default Infinispan uses a Java Serialization marshaller, which requires you to add your classes to an allow list. As an alternative you can use ProtoStream, which requires you to annotate your classes and generate a `SerializationContextInitializer` for custom Java objects.

Procedure

1. Open `hotrod-client.properties` or `application.properties` for editing.
2. Do one of the following:
   - Use ProtoStream as the marshaller.

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

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

   - Add your classes to the serialization allow list if you use Java Serialization. You can specify a comma-separated list of fully qualified class names or a regular expression to match classes.

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

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

3. Save and close your properties file.

Additional resources

- Cache Encoding and Marshalling

3.3. Cache Manager Configuration Beans

Customize the cache manager with the following configuration beans:

- `InfinispanRemoteConfigurer`
- `Configuration`
- `InfinispanRemoteCacheCustomizer`

You can create one `InfinispanRemoteConfigurer` bean only. However you can create multiple configurations with the other beans.

InfinispanRemoteConfigurer Bean

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

Configuration 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);
}
```

Use the `@Ordered` annotation to apply customizers in a specific order.

3.4. Enabling Spring Cache Support

With both embedded and remote caches, Infinispan provides an implementation of Spring Cache that you can enable.

Procedure

- Add the `@EnableCaching` annotation to your application.

If the Infinispan starter detects the:

- `EmbeddedCacheManager` bean, it instantiates a new `SpringEmbeddedCacheManager`.
- `RemoteCacheManager` bean, it instantiates a new `SpringRemoteCacheManager`.

Reference

Spring Cache Reference

3.5. Exposing Infinispan Statistics

Infinispan supports the Spring Boot Actuator to expose cache statistics as metrics.

Procedure

1. Add the following to your `pom.xml` file:

```
<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. Activate statistics for the appropriate cache instances, either programmatically or declaratively.

   Programmatically

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

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

   Declaratively

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

The Spring Boot Actuator registry binds cache instances when your application starts.

If you create caches dynamically, you should use the `CacheMetricsRegistrar` bean to bind caches to the Actuator registry, as follows:

```
@Autowire
CacheMetricsRegistrar cacheMetricsRegistrar;

@Autowire
CacheManager cacheManager;
...

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

Chapter 4. Using Spring Session

4.1. Enabling Spring Session Support

Infinispan Spring Session support is built on `SpringRemoteCacheManager` and `SpringEmbeddedCacheManager`. The Infinispan starter produces those beans by default.

Procedure

1. Add this starter to your project.
2. Add Spring Session to the classpath.
3. Add the following annotations to your configuration:
    - `@EnableCaching`
    - `@EnableInfinispanRemoteHttpSession`
    - `@EnableInfinispanEmbeddedHttpSession`

Infinispan does not provide a default cache. To use Spring Session, you must first create a Infinispan cache.

Chapter 5. Application Properties

Configure your project with `application.properties` or `application.yaml`.

```
#
# 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=
```