Spring Boot - Cache Provider

Last Updated : 23 Jul, 2025

The Spring Framework provides support for transparently adding caching to an application. The **Cache provider** gives authorization to programmers to configure cache explicitly in an application. It incorporates various cache providers such as **EhCache, Redis, Guava, Caffeine**, **etc.**

It keeps frequently accessed objects, images, and data closer to where you need them, speeding up access by not hitting the database or any third-party application multiple times for the same data and saving monetary costs. Data that does not change frequently can be cached. In Spring Boot, Cache Abstraction depends on the abstraction occurred by two interfaces i.e. **org.springframework.cache.CacheManager** interface or **org.springframework.cache.Cache** interface.

Auto-Configuration of Caching

The Spring Boot Framework facilitates and reduces complexities by implementing auto-configuration support in caching. It searches for the libraries and configuration files in the classpath and initializes the required dependency beans at the time of application startup. Spring Boot auto-configures the cache infrastructure as long as caching is enabled via the **@EnableCaching** annotation.

If we do not add any specific cache library, Spring Boot auto-configures a simple provider that uses concurrent maps in memory but it is not really recommended for production usage.

To add caching to an operation of your application we need to add **@Cacheable** annotation to its method,

```
// Annotation
@Component

// Class
public class Student
{
        // Annotation
        @Cacheable("Names")

          // Method
          public int getName(String name) {}
}
```

Now, Before invoking **getName()** method the abstraction looks for an entry in the **Names** cache that matches the **name** argument. If an entry is found, the content in the cache is immediately returned to the caller, and the method is not invoked. Otherwise, the method is invoked, and the cache is updated before returning the value.

Spring Boot Cache Providers

The cache abstraction does not provide an actual store and relies on abstraction materialized by the **org.springframework.cache.Cache** and **org.springframework.cache.CacheManager** interfaces. If we have not defined a bean of type **CacheManager** or a **CacheResolver** named **cacheResolver**, Spring Boot tries to detect the following providers:

1. **Generic**
2. **JCache (JSR-107)**
3. **EhCache 2.x**
4. **Hazelcast**
5. **Guava**
6. **Infinispan**
7. **Couchbase**
8. **Redis**
9. **Caffeine**

10. **Simple**

To quickly add basic caching dependencies, we must use the **spring-boot-starter-cache** in **pom.xml file.** If we want to add dependencies manually, we must include **spring-context-support** in our **pom.xml** file in order to use the **JCache, EhCache 2.x, or Guava** support.

```
<dependency>
<groupId>org.springframework</groupId>
<artifactId>spring-context-support</artifactId>
<version>5.2.3.RELEASE</version>
</dependency>
```

To add required libraries in the classpath, we need to add **EhCache** dependency in pom.xml as shown below:

```
<dependency>
<groupId>org.ehcache</groupId>
<artifactId>ehcache</artifactId>
</dependency>
```

Provider 1: Generic

Generic caching is used in the context that defines at least one **org.springframework.cache.Cache** bean. A CacheManager wrapping all beans of that type is created.

Provider 2: JCache

JCache starts with the presence of a **javax.cache.spi.CachingProvider** on the classpath and the **JCacheCacheManager** is provided by the **spring-boot-starter-cache** “Starter”. It might happen that more than one provider is present, in which case the provider must be explicitly specified. Even if the JSR-107 standard does not enforce a standardized way to define the location of the configuration file, Spring Boot accommodates setting a cache with implementation details as shown in below illustration as follows:

Illustration:

```
# Only necessary if more than one provider is present
spring.cache.jcache.provider=com.acme.MyCachingProvider
spring.cache.jcache.config=classpath:acme.xml
```

If the cache library provides both native implementation and **JSR** support, Spring Boot prefers **JSR** support.

Provider 3: EhCache 2.x

Ehcache 2.x is an open-source, standards-based cache that boosts performance, offloads your database, and simplifies scalability. It's the most widely-used Java-based cache because it's robust, proven, full-featured, and integrates with other popular libraries and frameworks. EhCache 2.x is used if a file named **ehcache.xml** can be found at the root of the classpath. If EhCache 2.x is found, the **EhCacheCacheManager** provided by the **spring-boot-starter-cache** "Starter" is used to bootstrap the cache manager. It scales from in-process caching, all the way to mixed in-process/out-of-process deployments with terabyte-sized caches. We can configure EhCache by using the following property:

```
spring.cache.ehcache.config=classpath:config/demo-config.xml
```

Provider 4: Hazelcast

Hazelcast is actually a streaming and memory-first application platform for fast, stateful, data-intensive workloads on-premises, at the edge or as a fully managed cloud service. Spring Boot has general support for **Hazelcast**. If a **HazelcastInstance** has been auto-configured, it is automatically wrapped in a **CacheManager** unless the

spring.cache.jcache.config property is specified. We can configure Hazelcast by using the following property:

```
spring.hazelcast.config=classpath:config/demo-hazelcast.xml
```

Provider 5: Guava

Guava is a set of core Java libraries from Google that includes new collection types (such as multimap and multiset), immutable collections, a graph library, and utilities for concurrency, I/O, hashing, caching, primitives, strings, and more. Guava is a single JAR that provides cache among many other capabilities. We can configure the **applicationConfig.xml** file by:

```
<bean id="cacheManager"
class="org.springframework.cache.guava.GuavaCacheManager"/>
```

Provider 6: Infinispan

Infinispan is an open-source in-memory data grid that offers flexible deployment options and robust capabilities for storing, managing, and processing data. It can be easily integrated with JCache, JPA Quarkus, Spring, etc. It provides a key/value data store that can hold all types of data, from Java objects to plain text. Infinispan has no default configuration file location, so it must be specified explicitly. Otherwise, the default bootstrap is used.

```
spring.cache.infinispan.config=infinispan.xml
```

Provider 7: Couchbase

CouchbaseCacheManager gets auto-configured if the **Couchbase** and the **couchbase-spring-cache** implementation are available and is configured. We can create additional caches on startup by setting the

spring.cache.cache-names property. These caches operate on the **Bucket** that was auto-configured. We can also create additional caches on another **Bucket** by using the customizer. We can create the two caches in one Bucket with configuration, as follows:

```
spring.cache.cache-names=cache1,cache2
```

Provider 8: Redis

RedisCacheManager gets auto-configured if **Redis** is available and configured. We can create additional caches on startup by setting the **spring.cache.cache-names** property and cache defaults can be configured by using the property **spring.cache.redis.*** .

The following configuration creates two caches named cache1 and cache2, which lives for 1 minute.

```
spring.cache.cache-names=cache1,cache2
spring.cache.redis.time-to-live=60000
```

Provider 9: Caffeine

CaffeineCacheManager (provided by the spring-boot-starter-cache "Starter") gets auto-configured If Caffeine is present and configured. Caffeine supersedes support for Guava. Caches can be created on startup by setting the **spring.cache.cache-names** property.

The following configuration creates cache1 and cache2 caches with a maximum size of 100 and a time to live of 1 minute:

```
spring.cache.cache-names=cache1,cache2
spring.cache.caffeine.spec=maximumSize=100,expireAfterAccess=60s
```

To use Caffeine in Spring Boot application, we need to add caffeine dependency in the pom.xml as shown below:

```
<dependency>
<groupId>com.github.ben-manes.caffeine</groupId>
<artifactId>caffeine</artifactId>
</dependency>
```

Provider 10: Simple

If none of the other providers is mentioned, a simple implementation using a **ConcurrentHashMap** as the cache-store is configured. it will be used only if no caching library is present in the application. We can restrict the list of available caches by setting the **cache-names** property. If we want only cache1 and cache2 caches, set the **cache-names** property as follows:

Dependency for simple cache provider

```
spring.cache.cache-names=cache1,cache2
```

Comment More info Advertise with us