32. Caching

The Spring Framework provides support for transparently adding caching to an application. At its core, the abstraction applies caching to methods, reducing thus the number of executions based on the information available in the cache. The caching logic is applied transparently, without any interference to the invoker.

Check the relevant section of the Spring Framework reference for more details.

In a nutshell, adding caching to an operation of your service is as easy as adding the relevant annotation to its method:

```
import javax.cache.annotation.CacheResult;

import org.springframework.stereotype.Component;

@Component
public class MathService {

    @CacheResult
    public int computePiDecimal(int i) {
        // ...
    }

}
```

You can either use the standard JSR-107 (JCache) annotations or Spring's own caching annotations transparently. We strongly advise you however to not mix and match them.

It is also possible to update or evict data from the cache transparently.

32.1 Supported 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. Spring Boot auto-configures a suitable `CacheManager` according to the implementation as long as the caching support is enabled via the `@EnableCaching` annotation.

> Use the `spring-boot-starter-cache` "Starter POM" to quickly add required caching dependencies. If you are adding dependencies manually you should note that certain implementations are only provided by the `spring-context-support` jar.

Spring Boot tries to detect the following providers (in this order):

- Generic
- JCache (JSR-107)
- EhCache 2.x
- Hazelcast
- Infinispan
- Redis
- Guava
- Simple

It is also possible to *force* the cache provider to use via the `spring.cache.type` property.

If the `CacheManager` is auto-configured by Spring Boot, you can further tune its configuration before it is fully initialized by exposing a bean implementing the `CacheManagerCustomizer` interface. The following set the cache names to use.

```
@Bean
public CacheManagerCustomizer<ConcurrentMapCacheManager> cacheManagerCustomizer()
    return new CacheManagerCustomizer<ConcurrentMapCacheManager>() {
        @Override
        public void customize(ConcurrentMapCacheManager cacheManager) {
            cacheManager.setCacheNames(Arrays.asList("one", "two"));
        }
    };
}
```

=== In the example above, a `ConcurrentMapCacheManager` is expected to be configured. If that is not the case, the customizer won't be invoked at all. You can have as many customizers as you want and you can also order them as usual using `@Order` or `Ordered`. ===

32.1.1 Generic

Generic caching is used if the context defines *at least* one `org.springframework.cache.Cache` bean, a `CacheManager` wrapping them is configured.

32.1.2 JCache

JCache is bootstrapped via the presence of a `javax.cache.spi.CachingProvider` on the classpath (i.e. a JSR-107 compliant caching library). It might happen than more that 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 does its best to accommodate with implementation details.

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

Since a cache library may offer both a native implementation and JSR-107 support Spring Boot will prefer the JSR-107 support so that the same features are available if you switch to a different JSR-107 implementation.

There are several ways to customize the underlying `javax.cache.cacheManager`:

- Caches can be created on startup via the `spring.cache.cache-names` property. If a custom `javax.cache.configuration.Configuration` bean is defined, it is used to customize them.
- `org.springframework.boot.autoconfigure.cache.JCacheManagerCustomizer` beans are invoked with the reference of the `CacheManager` for full customization.

If a standard `javax.cache.CacheManager` bean is defined, it is wrapped that the abstraction expects. No further customization is applied on it.

32.1.3 EhCache 2.x

EhCache 2.x is used if a file named `ehcache.xml` can be found at the root of the classpath. If EhCache 2.x and such file is present it is used to bootstrap the cache manager. An alternate configuration file can be provide a well using:

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

32.1.4 Hazelcast

Spring Boot has a general support for Hazelcast. If a `HazelcastInstance` has been auto-configured, it is automatically wrapped in a `CacheManager`.

If for some reason you need a different `HazelcastInstance` for caching, you can request Spring Boot to create a separate one that will be only used by the `CacheManager`:

```
spring.cache.hazelcast.config=classpath:config/my-cache-hazelcast.xml
```

If a separate `HazelcastInstance` is created that way, it is not registered in the application context.

32.1.5 Infinispan

Infinispan has no default configuration file location so it must be specified explicitly (or the default bootstrap is used).

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

Caches can be created on startup via the `spring.cache.cache-names` property. If a custom `ConfigurationBuilder` bean is defined, it is used to customize them.

32.1.6 Redis

If Redis is available and configured, the `RedisCacheManager` is auto-configured. It is also

Search Documentation

possible to create additional caches on startup using the `spring.cache.cache-names` property.

32.1.7 Guava

If Guava is present, a `GuavaCacheManager` is auto-configured. Caches can be created on startup using the `spring.cache.cache-names` property and customized by one of the following (in this order):

1. A cache spec defined by `spring.cache.guava.spec`
2. A `com.google.common.cache.CacheBuilderSpec` bean is defined
3. A `com.google.common.cache.CacheBuilder` bean is defined

For instance, the following configuration creates a `foo` and `bar` caches with a maximum size of 500 and a *time to live* of 10 minutes

```
spring.cache.cache-names=foo,bar
spring.cache.guava.spec=maximumSize=500,expireAfterAccess=600s
```

Besides, if a `com.google.common.cache.CacheLoader` bean is defined, it is automatically associated to the `GuavaCacheManager`.

32.1.8 Simple

If none of these options worked out, a simple implementation using `ConcurrentHashMap` as cache store is configured. This is the default if no caching library is present in your application.