http://www.egovframe.go.kr/wiki/doku.php?id=egovframework:rte2:fdl:cache_abstraction

Cache Abstraction

Outline

From Spring 3.1 on, the cache service offers @Cacheable, a cache abstraction method intended for any Java Method. Just like transaction works in Spring, cache abstraction allows you to work the proxy while controlling code changes to the minimum. By providing you not with the cache implementor but the with cache abstraction, the cache data repository supports both EhCache and ConcurrentMap.

- Cache Configuration : You can define which cache data repository you' going to use. (EhCache/ConcurrentMap)
- Cache Manager : You can access the cache regardless of configuration, via CacheManager
- Cache Annotation : You can save / remove cache data, via cache annotation

Description

In this Section, we are going to learn how caches can be accessed via Cache Manager and how @cacheable works for caching of Java Methods.

Cache Configuration

EhCache

- See EhCache for more information.

Spring offers EhCacheCacheManager, a CacheManager supporting EhCache. <EhCache configuration>

```
<cache:annotation-driven cache-manager="cacheManager" />
<!-- Cache Manager using EhCache as a repository -->
  <bean id="cacheManager" class="org.springframework.cache.ehcache.EhCacheCacheManager">
<property name="cacheManager" ref="ehcache"></property>
</bean>
<!-- Ehcache library setup -->
<bean id="ehcache"
class="org.springframework.cache.ehcache.EhCacheManagerFactoryBean"
p:config-location="classpath:springframework/cache/ehcache/ehcache.xml" />
```

You need to configure defaultCache repository and additional Cache repository in Ehcache.xml. Use of EhCache is highly recommended over ConcurrentMap despite poor speed, as you can take advantage of EhCache.

- The developer of the project needs to guide use of caches. You thus need to separate cache repositories by type of business / service and provide guidances of the cases applied.

<ehcache.xml>

```
<ehcache…>
<defaultCache
                maxElementsInMemory="10000"
                eternal="false"
                timeToIdleSeconds="120"
                timeToLiveSeconds="120"
                overflowToDisk="true"
                diskSpoolBufferSizeMB="30"
                maxElementsOnDisk="10000000"
                diskPersistent="false"
                diskExpiryThreadIntervalSeconds="120"
```

```
                memoryStoreEvictionPolicy="LRU"
                statistics="false"
                />
<cache name="ehcache"
               …>
```

ConcurrentMap

Spring provides you with the SimpleCacheManager that are compatible with ConcurrentMap.

<Configuring CacheManager>

```
<cache:annotation-driven cache-manager="cacheManager"/>
  <!-- Cache Manager using ConcurrentMap as a repository -->
<bean id="cacheManager" class="org.springframework.cache.support.SimpleCacheManager">
    <property name="caches">
        <set>
            <bean class="org.springframework.cache.concurrent.ConcurrentMapCacheFactoryBean"
p:name="default"/>
            <bean class="org.springframework.cache.concurrent.ConcurrentMapCacheFactoryBean"
p:name="books"/>
        </set>
    </property>
</bean>
```

Configuring Annotation-driven for Cache Annotation

In order to apply caches via Cache Annotation, you need to add the Namespace <cache:annotation-driven> to allow configurations regarding caching using AOP. You can work the similar way to <tx:annotation-driven> for transaction.

Property	Default	Description
cache-manager	cacheManager	You need to configure this when the name and Bean ID does not match those of CacheManager.
mode	proxy	Configuration for Spring AOP. You can also use “aspect” alternatively.
proxy-target-class	false	If configured “false” you need to use the proxy based upon JDK Interface. Configure “true” if you want to use class-based proxy.
order	Ordered.LOWEST_PRECEDENCE	Sequence of cache advice of @Cacheable/@CacheEvict Method.

- <cache:annotation-driven/> looks for @Cacheable and @CacheEvict in beans of the ApplicationContext where it is defined. When <cache:annotation-driven> is declared in WebApplicationContext for DispatcherServlet, both @Cacheable and @CacheEvict are identified inside the controller.

Accessing Caches via CacheManager

You can have beans injected into the CacheManager Interfaces featuring the identical codes, regardless of configurations made in ehCache and concurrentMap.

<How to use CacheManager>

```
import org.springframework.cache.CacheManager;
@Autowired
private CacheManager cacheManager;
Cache cache = cacheManager.getCache("cache name");
```

@Cacheable

You can work caching in the JAVA Method by configuring @Cacheable. When the target method is called, you need to check for the concerned method existing in the form of the identical factor. If not, you no longer need to call the method and cause the pre-cached result returned out of the proxy.

See the following for the cache annotations breakdown applicable to Java Method:

- @Cacheable : You can generate the method data in Cache.
- @CacheEvict : You can remove the method data out of Cache.

How to use @Cacheable

All you need to do is to record the name of caches for the unconditionally called factors. In the following code the cache repository for caches entitled “books” is assumed. You can check out the cache data in “books” and check if it has been previously executed. If you can find the relevant data in “books”, you’re going to return the corresponding value.

Basic Operations

How to use @Cacheable
public Book findBook(ISBN isbn, boolean checkWarehouse, boolean includeUsed)

You may define multiple repositories for caching. Refer to the following codes for how to call the Method findBook and reposit cache data in both “books” and “isbns”.

<How to cache in multiple repositories>

@Cacheable({ "books", "isbns" })
public Book findBook(ISBN isbn) {...}

Generating Cache Abstraction Key

Caches are reposited in the form of key that is loaded when the cached method is called. With the separate custom keys left undefined, the caches work KeyGenerator to generate keys as follows:

- When no parameters are available, “0” is returned.
- When a single parameter is available, the concerned instance is returned.
- When plural parameters are available, the key is returned calculated by hash of parameters.

You may also generate the Interface “org.springframework.cache.KeyGenerator” to generate other abstraction key.

Adding Custom Key

You can define keys multiple method factors using @Cacheable in the form of SpEL.

@Cacheable(value="books", key="#isbn"
public Book findBook(ISBN isbn, boolean checkWarehouse, boolean includeUsed)

@Cacheable(value="books", key="#isbn.rawNumber")
public Book findBook(ISBN isbn, boolean checkWarehouse, boolean includeUsed)

@Cacheable(value="books", key="T(someType).hash(#isbn)")
public Book findBook(ISBN isbn, boolean checkWarehouse, boolean includeUsed)

Adding Conditional

You can add Conditional and configure for true / false for caching / skip. The method is thus executed every time you call for caching. You can also use SpEL to define either Condition or Unless. When you configure for Unless, the result is referred when the result of method is returned to determine implementation of caching.

@Cacheable(value="book", condition="#name.length < 32")
public Book findBook(String name)
@Cacheable(value="book", condition="#name.length < 32", unless="#result.hardback")
public Book findBook(String name)

Refer to the following description on SpEL for Conditional:

Title	Location	Description	Example
methodName	Root object	Title of method called	#root.methodName
method	Root object	Method called	#root.method.name
target	Root object	Target objects called	#root.target
targetClass	Root object	Target class called	#root.targetClass
args	Root object	Factors(Arrays) used when calling the target	#root.args[0]
caches	Root object	A set of caches where the method is being executed.	#root.caches[0].name
argument name	Evaluation context	You can replace the name of method factor into a<#arg> that refers to the index of factor that counts from 0.	iban or a0 (or p<#arg>)
result	Evaluation context	Output of a method called. Valid only in Unless and Cache Evict.	#result

@CacheEvict

@CacheEvict is an opposite to @Cacheable as it “evicts” the caches reposited to secure free space. Declared to the method for eviction of caches, @CacheEvict supports declaration of plural caches capable of using both key and condition. If you work the attribute allEntries, you’ll see the cache entries are evicted in their entirety, in which case the duly notified keys are ignored to evict the entries in their entirety.

@CacheEvict(value = "books", allEntries=true)
public void loadBooks(InputStream batch)

@CachePut

You need to use @CachePut to reposit or update the data in cash without interrupting flow of the method. While the result of the executed method is reposited in the cache, @CachePut provides the same configuration options with @Cacheable and shall preferably be used to optimize the flow of method.

It is not recommended, however, to use @CachePut along with @Cacheable.

@Caching

It is advised to use @Caching when a multitude of cache annotation is required. Note @Caching supports @Cacheable, @CacheEvic and @CachePut.

@Caching(evict = { @CacheEvict("primary"), @CacheEvict(value = "secondary", key = "#p0") })
public Book importBooks(String deposit, Date date)

Configuration of XML-based Caching

Instead of adding annotation to Java Method, you can define a method to be cached via XML. While the following example omits configuration of CacheManager beans, you can use “cacheable” with the Method findBook Java as a sub-method of the Package BookService. Also notable is cacheEvict applied to the Method loadBooks.

```
<!-- the service we want to make cacheable -->
<bean id="bookService" class="x.y.service.DefaultBookService"/>

<!-- cache definitions -->
<cache:advice id="cacheAdvice" cache-manager="cacheManager">
	<cache:caching cache="books">
		<cache:cacheable method="findBook" key="#isbn"/>
		<cache:cache-evict method="loadBooks" all-entries="true"/>
	</cache:caching>
</cache:advice>

<!-- apply the cacheable behavior to all BookService interfaces -->
<aop:config>
	<aop:advisor advice-ref="cacheAdvice" pointcut="execution(* x.y.BookService.*(..))"/>
</aop:config>
...
// Configuration for CacheManager omitted
```

References

- Spring Framework - Reference Document / cache
- Spring Team Blog - Caching