fashflying



# Spring缓存注解@Cacheable、@CacheEvict、@CachePut使用

从3.1开始，Spring引入了对Cache的支持。其使用方法和原理都类似于Spring对事务管理的支持。Spring Cache是作用在方法上的，其核心思想是这样的：当我们在调用一个缓存方法时会把该方法参数和返回结果作为一个键值对存放在缓存中，等到下次利用同样的参数来调用该方法时将不再执行该方法，而是直接从缓存中获取结果进行返回。所以在使用Spring Cache的时候我们要保证我们缓存的方法对于相同的方法参数要有相同的返回结果。

使用Spring Cache需要我们做两方面的事：

n 声明某些方法使用缓存

n 配置Spring对Cache的支持

和Spring对事务管理的支持一样，Spring对Cache的支持也有基于注解和基于XML配置两种方式。下面我们先来看看基于注解的方式。

## 1 基于注解的支持

Spring为我们提供了几个注解来支持Spring Cache。其核心主要是@Cacheable和@CacheEvict。使用@Cacheable标记的方法在执行后Spring Cache将缓存其返回结果，而使用@CacheEvict标记的方法会在方法执行前或者执行后移除Spring Cache中的某些元素。下面我们将来详细介绍一下Spring基于注解对Cache的支持所提供的几个注解。

### 1.1 @Cacheable

@Cacheable可以标记在一个方法上，也可以标记在一个类上。当标记在一个方法上时表示该方法是支持缓存的，当标记在一个类上时则表示该类所有的方法都是支持缓存的。对于一个支持缓存的方法，Spring会在其被调用后将其返回值缓存起来，以保证下次利用同样的参数来执行该方法时可以直接从缓存中获取结果，而不需要再次执行该方法。Spring在缓存方法的返回值时是以键值对进行缓存的，值就是方法的返回结果，至于键的话，Spring又支持两种策略，默认策略和自定义策略，这个稍后会进行说明。需要注意的是当一个支持缓存的方法在对象内部被调用时是不会触发缓存功能的。@Cacheable可以指定三个属性，value、key和condition。

#### 1.1.1 value属性指定Cache名称

value属性是必须指定的，其表示当前方法的返回值是会被缓存在哪个Cache上的，对应Cache的名称。其可以是一个Cache也可以是多个Cache，当需要指定多个Cache时其是一个数组。

```
@Cacheable("cache1")//Cache是发生在cache1上的
public User find(Integer id) {
    returnnull;
}

@Cacheable({"cache1", "cache2"})//Cache是发生在cache1和cache2上的
public User find(Integer id) {
    returnnull;
}
```

公告

昵称：fashflying
园龄：2年3个月
粉丝：3
关注：2
+加关注

< 2018年4

| 日 | 一 | 二 | 三 |
|---|---|---|---|
| 25 | 26 | 27 | 28 |
| 1 | 2 | 3 | 4 |
| 8 | 9 | 10 | 11 |
| 15 | 16 | 17 | 18 |
| 22 | 23 | 24 | 25 |
| 29 | 30 | 1 | 2 |

搜索

随笔档案

2018年4月 (1)
2017年10月 (1)
2017年6月 (2)
2017年5月 (10)
2017年4月 (4)
2017年3月 (3)
2017年2月 (1)
2017年1月 (3)

### 1.1.2 使用key属性自定义key

key属性是用来指定Spring缓存方法的返回结果时对应的key的。该属性支持SpringEL表达式。当我们没有指定该属性时，Spring将使用默认策略生成key。我们这里先来看看自定义策略，至于默认策略会在后文单独介绍。

自定义策略是指我们可以通过Spring的EL表达式来指定我们的key。这里的EL表达式可以使用方法参数及它们对应的属性。使用方法参数时我们可以直接使用“#参数名”或者“#p参数index”。下面是几个使用参数作为key的示例。

```
@Cacheable(value="users", key="#id")
public User find(Integer id) {
  returnnull;
}

@Cacheable(value="users", key="#p0")
public User find(Integer id) {
  returnnull;
}

@Cacheable(value="users", key="#user.id")
public User find(User user) {
  returnnull;
}

@Cacheable(value="users", key="#p0.id")
public User find(User user) {
  returnnull;
}
```

除了上述使用方法参数作为key之外，Spring还为我们提供了一个root对象可以用来生成key。通过该root对象我们可以获取到以下信息。

| 属性名称 | 描述 | 示例 |
| --- | --- | --- |
| methodName | 当前方法名 | #root.methodName |
| method | 当前方法 | #root.method.name |
| target | 当前被调用的对象 | #root.target |
| targetClass | 当前被调用的对象的class | #root.targetClass |
| args | 当前方法参数组成的数组 | #root.args[0] |
| caches | 当前被调用的方法使用的Cache | #root.caches[0].name |

当我们要使用root对象的属性作为key时我们也可以将“#root”省略，因为Spring默认使用的就是root对象的属性。如：

```
@Cacheable(value={"users", "xxx"}, key="caches[1].name")
public User find(User user) {
  returnnull;
}
```

### 1.1.3 condition属性指定发生的条件

2016年11月 (2)
2016年10月 (2)
2016年9月 (2)
2016年8月 (1)
2016年7月 (2)
2016年6月 (5)
2016年5月 (2)
2016年4月 (2)
2016年2月 (2)
2016年1月 (12)

引用

orcale case
spring data jpa 自定义
spring data jpa 自定义
data jpa 定义全局接口
SpringMVC接收复杂集
SpringMVC接收复杂集
SQL update select结合
use_hash
http://www.dba-oracle.com/t_use_ha
考
xml基础
使用jQuery.form.js/sp
现文件上传功能
jquery.form 实现文件_
数据库死锁解决办法
用nohup命令让Linux下
执行

有的时候我们可能并不希望缓存一个方法所有的返回结果。通过condition属性可以实现这一功能。condition属性默认为空，表示将缓存所有的调用情形。其值是通过SpringEL表达式来指定的，当为true时表示进行缓存处理；当为false时表示不进行缓存处理，即每次调用该方法时该方法都会执行一次。如下示例表示只有当user的id为偶数时才会进行缓存。

```
@Cacheable(value={"users"}, key="#user.id", condition="#user.id%2==0")
public User find(User user) {
    System.out.println("find user by user " + user);
    return user;
}
```

## 1.2 @CachePut

在支持Spring Cache的环境下，对于使用@Cacheable标注的方法，Spring在每次执行前都会检查Cache中是否存在相同key的缓存元素，如果存在就不再执行该方法，而是直接从缓存中获取结果进行返回，否则才会执行并将返回结果存入指定的缓存中。
@CachePut也可以声明一个方法支持缓存功能。与@Cacheable不同的是使用@CachePut标注的方法在执行前不会去检查缓存中是否存在之前执行过的结果，而是每次都会执行该方法，并将执行结果以键值对的形式存入指定的缓存中。

@CachePut也可以标注在类上和方法上。使用@CachePut时我们可以指定的属性跟@Cacheable是一样的。

```
@CachePut("users")//每次都会执行方法，并将结果存入指定的缓存中
public User find(Integer id) {
    returnnull;
}
```

## 1.3 @CacheEvict

@CacheEvict是用来标注在需要清除缓存元素的方法或类上的。当标记在一个类上时表示其中所有的方法的执行都会触发缓存的清除操作。@CacheEvict可以指定的属性有value、key、condition、allEntries和beforeInvocation。其中value、key和condition的语义与@Cacheable对应的属性类似。即value表示清除操作是发生在哪些Cache上的（对应Cache的名称）；key表示需要清除的是哪个key，如未指定则会使用默认策略生成的key；condition表示清除操作发生的条件。下面我们来介绍一下新出现的两个属性allEntries和beforeInvocation。

### 1.3.1 allEntries属性

allEntries是boolean类型，表示是否需要清除缓存中的所有元素。默认为false，表示不需要。当指定了allEntries为true时，Spring Cache将忽略指定的key。有的时候我们需要Cache一下清除所有的元素，这比一个一个清除元素更有效率。

```
@CacheEvict(value="users", allEntries=true)
public void delete(Integer id) {
    System.out.println("delete user by id: " + id);
}
```

### 1.3.2 beforeInvocation属性

清除操作默认是在对应方法成功执行之后触发的，即方法如果因为抛出异常而未能成功返回时也不会触发清除操作。使用beforeInvocation可以改变触发清除操作的时间，当我们指定该属性值为true时，Spring会在调用该方法之前清除缓存中的指定元素。

```
@CacheEvict(value="users", beforeInvocation=true)
public void delete(Integer id) {
    System.out.println("delete user by id: " + id);
}
```

其实除了使用@CacheEvict清除缓存元素外，当我们使用Ehcache作为实现时，我们也可以配置Ehcache自身的驱除策略，其是通过Ehcache的配置文件来指定的。由于Ehcache不是本文描述的重点，这里就不多赘述了，想了解更多关于Ehcache的信息，请查看我关于Ehcache的专栏。

## 1.4 @Caching

@Caching注解可以让我们在一个方法或者类上同时指定多个Spring Cache相关的注解。其拥有三个属性：cacheable、put和evict，分别用于指定@Cacheable、@CachePut和@CacheEvict。

```
@Caching(cacheable = @Cacheable("users"), evict = { @CacheEvict("cache2"),
    @CacheEvict(value = "cache3", allEntries = true) })
public User find(Integer id) {
  returnnull;
}
```

### 1.5 使用自定义注解

Spring允许我们在配置可缓存的方法时使用自定义的注解，前提是自定义的注解上必须使用对应的注解进行标注。如我们有如下这么一个使用@Cacheable进行标注的自定义注解。

```
@Target({ElementType.TYPE, ElementType.METHOD})
@Retention(RetentionPolicy.RUNTIME)
@Cacheable(value="users")
public @interface MyCacheable {

}
```

那么在我们需要缓存的方法上使用@MyCacheable进行标注也可以达到同样的效果。

```
@MyCacheable
public User findById(Integer id) {
  System.out.println("find user by id: " + id);
  User user = new User();
  user.setId(id);
  user.setName("Name" + id);
  return user;
}
```

## 2 配置Spring对Cache的支持

### 2.1 声明对Cache的支持

#### 2.1.1 基于注解

配置Spring对基于注解的Cache的支持，首先我们需要在Spring的配置文件中引入cache命名空间，其次通过<cache:annotation-driven />就可以启用Spring对基于注解的Cache的支持。

```
<?xml version="1.0" encoding="UTF-8"?>
<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"
  xsi:schemaLocation="http://www.springframework.org/schema/beans
    http://www.springframework.org/schema/beans/spring-beans-3.0.xsd
    http://www.springframework.org/schema/cache
    http://www.springframework.org/schema/cache/spring-cache.xsd">

  <cache:annotation-driven/>

</beans>
```

<cache:annotation-driven/>有一个cache-manager属性用来指定当前所使用的CacheManager对应的bean的名称，默认是cacheManager，所以当我们的CacheManager的id为cacheManager时我们可以不指定该参数，否则就需要我们指定了。

<cache:annotation-driven/>还可以指定一个mode属性，可选值有proxy和aspectj。默认是使用proxy。当mode为proxy时，只有缓存方法在外部被调用的时候Spring Cache才会发生作用，这也就意味着如果一个缓存方法在其声明对象内部被调用时Spring Cache是不会发生作用的。而mode为aspectj时就不会有这种问题。另外使用proxy时，只有public方法上的@Cacheable等标注才会起作用，如果需要非public方法上的方法也可以使用Spring Cache时把mode设置为aspectj。

此外，<cache:annotation-driven/>还可以指定一个proxy-target-class属性，表示是否要代理class，默认为false。我们前面提到的@Cacheable、@cacheEvict等也可以标注在接口上，这对于基于接口的代理来说是没有什么问题的，但是需要注意的是当我们设置proxy-target-class为true或者mode为aspectj时，是直接基于class进行操作的，定义在接口上的@Cacheable等Cache注解不会被识别到，那对应的Spring Cache也不会起作用了。

需要注意的是<cache:annotation-driven/>只会去寻找定义在同一个ApplicationContext下的@Cacheable等缓存注解。

### 2.1.2 基于XML配置

除了使用注解来声明对Cache的支持外，Spring还支持使用XML来声明对Cache的支持。这主要是通过类似于aop:advice的cache:advice来进行的。在cache命名空间下定义了一个cache:advice元素用来定义一个对于Cache的advice。其需要指定一个cache-manager属性，默认为cacheManager。cache:advice下面可以指定多个cache:caching元素，其有点类似于使用注解时的@Caching注解。cache:caching元素下又可以指定cache:cacheable、cache:cache-put和cache:cache-evict元素，它们类似于使用注解时的@Cacheable、@CachePut和@CacheEvict。下面来看一个示例：

```
<cache:advice id="cacheAdvice" cache-manager="cacheManager">
  <cache:caching cache="users">
    <cache:cacheable method="findById" key="#p0"/>
    <cache:cacheable method="find" key="#user.id"/>
    <cache:cache-evict method="deleteAll" all-entries="true"/>
  </cache:caching>
</cache:advice>
```

上面配置定义了一个名为cacheAdvice的cache:advice，其中指定了将缓存findById方法和find方法到名为users的缓存中。这里的方法还可以使用通配符"*"，比如"find*"表示任何以"find"开始的方法。

有了cache:advice之后，我们还需要引入aop命名空间，然后通过aop:config指定定义好的cacheAdvice要应用在哪些pointcut上。如：

```
<aop:config proxy-target-class="false">
  <aop:advisor advice-ref="cacheAdvice" pointcut="execution(* com.xxx.UserService.*(..))"/>
</aop:config>
```

上面的配置表示在调用com.xxx.UserService中任意公共方法时将使用cacheAdvice对应的cache:advice来进行Spring Cache处理。更多关于Spring Aop的内容不在本文讨论范畴内。

## 2.2 配置CacheManager

CacheManager是Spring定义的一个用来管理Cache的接口。Spring自身已经为我们提供了两种CacheManager的实现，一种是基于Java API的ConcurrentMap，另一种是基于第三方Cache实现——Ehcache，如果我们需要使用其它类型的缓存时，我们可以自己来实现Spring的CacheManager接口或AbstractCacheManager抽象类。下面分别来看看Spring已经为我们实现好了的两种CacheManager的配置示例。

### 2.2.1 基于ConcurrentMap的配置

```
<bean id="cacheManager" class="org.springframework.cache.support.SimpleCacheManager">
  <property name="caches">
    <set>
      <bean class="org.springframework.cache.concurrent.ConcurrentMapCacheFactoryBean" p:name="xxx"/>
    </set>
  </property>
```

```
</bean>
```

上面的配置使用的是一个SimpleCacheManager，其中包含一个名为“xxx”的ConcurrentMapCache。

#### 2.2.2 基于Ehcache的配置

```
<!-- Ehcache实现 -->
<bean id="cacheManager" class="org.springframework.cache.ehcache.EhCacheCacheManager" p:cache-manager-ref="ehcacheManager"/>
<bean id="ehcacheManager" class="org.springframework.cache.ehcache.EhCacheManagerFactoryBean" p:config-location="ehcache-spring.xml"/>
```

上面的配置使用了一个Spring提供的EhCacheCacheManager来生成一个Spring的CacheManager，其接收一个Ehcache的CacheManager，因为真正用来存入缓存数据的还是Ehcache。Ehcache的CacheManager是通过Spring提供的EhCacheManagerFactoryBean来生成的，其可以通过指定ehcache的配置文件位置来生成一个Ehcache的CacheManager。若未指定则将按照Ehcache的默认规则取classpath根路径下的ehcache.xml文件，若该文件也不存在，则获取Ehcache对应jar包中的ehcache-failsafe.xml文件作为配置文件。更多关于Ehcache的内容这里就不多说了，它不属于本文讨论的内容，欲了解更多关于Ehcache的内容可以参考我之前发布的Ehcache系列文章，也可以参考官方文档等。

# 3 键的生成策略

键的生成策略有两种，一种是默认策略，一种是自定义策略。

## 3.1 默认策略

默认的key生成策略是通过KeyGenerator生成的，其默认策略如下：

- 如果方法没有参数，则使用0作为key。
- 如果只有一个参数的话则使用该参数作为key。
- 如果参数多余一个的话则使用所有参数的hashCode作为key。

如果我们需要指定自己的默认策略的话，那么我们可以实现自己的KeyGenerator，然后指定我们的Spring Cache使用的KeyGenerator为我们自己定义的KeyGenerator。

使用基于注解的配置时是通过cache:annotation-driven指定的.

```
<cache:annotation-driven key-generator="userKeyGenerator"/>

<bean id="userKeyGenerator" class="com.xxx.cache.UserKeyGenerator"/>
```

而使用基于XML配置时是通过cache:advice来指定的。

```
<cache:advice id="cacheAdvice" cache-manager="cacheManager" key-generator="userKeyGenerator">
</cache:advice>
```

需要注意的是此时我们所有的Cache使用的Key的默认生成策略都是同一个KeyGenerator。

## 3.2 自定义策略

自定义策略是指我们可以通过Spring的EL表达式来指定我们的key。这里的EL表达式可以使用方法参数及它们对应的属性。使用方法参数时我们可以直接使用“#参数名”或者“#p参数index”。下面是几个使用参数作为key的示例。

```
@Cacheable(value="users", key="#id")
public User find(Integer id) {
  returnnull;
}

@Cacheable(value="users", key="#p0")
```

```
public User find(Integer id) {
  returnnull;
}

@Cacheable(value="users", key="#user.id")
public User find(User user) {
  returnnull;
}

@Cacheable(value="users", key="#p0.id")
public User find(User user) {
  returnnull;
}
```

除了上述使用方法参数作为key之外，Spring还为我们提供了一个root对象可以用来生成key。通过该root对象我们可以获取到以下信息。

| 属性名称 | 描述 | 示例 |
| --- | --- | --- |
| methodName | 当前方法名 | #root.methodName |
| method | 当前方法 | #root.method.name |
| target | 当前被调用的对象 | #root.target |
| targetClass | 当前被调用的对象的class | #root.targetClass |
| args | 当前方法参数组成的数组 | #root.args[0] |
| caches | 当前被调用的方法使用的Cache | #root.caches[0].name |

当我们要使用root对象的属性作为key时我们也可以将“#root”省略，因为Spring默认使用的就是root对象的属性。如：

```
@Cacheable(value={"users", "xxx"}, key="caches[1].name")
public User find(User user) {
  returnnull;
}
```

# 4 Spring单独使用Ehcache

前面介绍的内容是Spring内置的对Cache的支持，其实我们也可以通过Spring自己单独的使用Ehcache的CacheManager或Ehcache对象。通过在Application Context中配置EhCacheManagerFactoryBean和EhCacheFactoryBean，我们就可以把对应的EhCache的CacheManager和Ehcache对象注入到其它的Spring bean对象中进行使用。

## 4.1 EhCacheManagerFactoryBean

EhCacheManagerFactoryBean是Spring内置的一个可以产生Ehcache的CacheManager对象的FactoryBean。其可以通过属性configLocation指定用于创建CacheManager的Ehcache配置文件的路径，通常是ehcache.xml文件的路径。如果没有指定configLocation，则将使用默认位置的配置文件创建CacheManager，这是属于Ehcache自身的逻辑，即如果在classpath根路径下存在ehcache.xml文件，则直接使用该文件作为Ehcache的配置文件，否则将使用ehcache-xxx.jar中的ehcache-failsafe.xml文件作为配置文件来创建Ehcache的CacheManager。此外，如果不希望创建的CacheManager使用默认的名称（在ehcache.xml文件中定义的，

或者是由CacheManager内部定义的），则可以通过cacheManagerName属性进行指定。下面是一个配置EhCacheManagerFactoryBean的示例。

```
<!-- 定义CacheManager -->
<bean id="cacheManager" class="org.springframework.cache.ehcache.EhCacheManagerFactoryBean">
    <!-- 指定配置文件的位置 -->
    <property name="configLocation" value="/WEB-INF/config/ehcache.xml"/>
    <!-- 指定新建的CacheManager的名称 -->
    <property name="cacheManagerName" value="cacheManagerName"/>
</bean>
```

## 4.2 EhCacheFactoryBean

EhCacheFactoryBean是用来产生Ehcache的Ehcache对象的FactoryBean。定义EhcacheFactoryBean时有两个很重要的属性我们可以来指定。一个是cacheManager属性，其可以指定将用来获取或创建Ehcache的CacheManager对象，若未指定则将通过CacheManager.create()获取或创建默认的CacheManager。另一个重要属性是cacheName，其表示当前EhCacheFactoryBean对应的是CacheManager中的哪一个Ehcache对象，若未指定默认使用beanName作为cacheName。若CacheManager中不存在对应cacheName的Ehcache对象，则将使用CacheManager创建一个名为cacheName的Cache对象。此外我们还可以通过EhCacheFactoryBean的timeToIdle、timeToLive等属性指定要创建的Cache的对应属性，注意这些属性只对CacheManager中不存在对应Cache时新建的Cache才起作用，对已经存在的Cache将不起作用，更多属性设置请参考Spring的API文档。此外还有几个属性是对不管是已经存在还是新创建的Cache都起作用的属性：statisticsEnabled、sampledStatisticsEnabled、disabled、blocking和cacheEventListeners，其中前四个默认都是false，最后一个表示为当前Cache指定CacheEventListener。下面是一个定义EhCacheFactoryBean的示例。

```
<!-- 定义CacheManager -->
<bean id="cacheManager" class="org.springframework.cache.ehcache.EhCacheManagerFactoryBean">
    <!-- 指定配置文件的位置 -->
    <property name="configLocation" value="/WEB-INF/config/ehcache.xml"/>
    <!-- 指定新建的CacheManager的名称 -->
    <property name="cacheManagerName" value="cacheManagerName"/>
</bean>

<!-- 定义一个Ehcache -->
<bean id="userCache" class="org.springframework.cache.ehcache.EhCacheFactoryBean">
    <property name="cacheName" value="user"/>
    <property name="cacheManager" ref="cacheManager"/>
</bean>
```



fashflying
关注 - 2
粉丝 - 3
+加关注

5 0

« 上一篇：linux Shell脚本编码格式
» 下一篇：linux 删除指定日期之前的文件

posted @ 2017-05-26 13:08 fashflying 阅读(25283) 评论(0) 编辑 收藏