第 4-4 课：Spring Boot 中使用 Cache 缓存的使用

我们知道绝大多数的网站/系统，最先遇到的一个性能瓶颈就是数据库，使用缓存做数据库的前置缓存，可以非常有效地降低数据库的压力，从而提升整个系统的响应效率和并发量。

以往使用缓存时，通常创建好缓存工具类，使用时将对应的工具类注入，操作工具类在前端处理缓存的逻辑。其实这种方式是低效的，大部分使用缓存的场景是基于数据库的缓存，这类缓存场景的逻辑往往是：如果缓存中存在数据，就从缓存中读取，如果缓存中不存在数据或者数据失效，就再从数据库中读取。

为了实现这样的逻辑，往往需要在业务代码中写很多的逻辑判断，那么有没有通用的代码来实现这样的逻辑呢？其实有，按照这个逻辑我们可以写一个工具类来实现，每次需要这样判断逻辑时调用工具类中的方法即可，还有没有更优雅的使用方式呢？答案是肯定的，如果我们把这种固定的逻辑使用 Java 注解来实现，每次需要使用时只需要在对应的方法或者类上写上注解即可。

Spring 也看到了这样的使用场景，于是有了**注释驱动的 Spring Cache**。它的原理是 Spring Cache 利用了 Spring AOP 的动态代理技术，在项目启动的时候动态生成它的代理类，在代理类中实现了对应的逻辑。

Spring Cache 是在 Spring 3.1 中引入的基于注释（Annotation）的缓存（Cache）技术，它本质上不是一个具体的缓存实现方案，而是一个对缓存使用的抽象，通过在既有代码中添加少量它定义的各种 Annotation，即能够达到缓存方法的返回对象的效果。

Spring 的缓存技术还具备相当的灵活性，不仅能够使用 SpEL（Spring Expression Language）来定义缓存的 key 和各种 condition，还提供了开箱即用的缓存临时存储方案，也支持和主流的专业缓存如 EHCache 集成。

> SpEL（Spring Expression Language）是一个支持运行时查询和操作对象图的强大的表达式语言，其语法类似于统一 EL，但提供了额外特性，**显式方法调用和基本字符串模板函数**。

其特点总结如下：

- 通过少量的配置 Annotation 注释即可使得既有代码支持缓存；
- 支持开箱即用 Out-Of-The-Box，即不用安装和部署额外第三方组件即可使用缓存；
- 支持 Spring Express Language，能使用对象的任何属性或者方法来定义缓存的 key 和 condition；
- 支持 AspectJ，并通过其实现任何方法的缓存支持；
- 支持自定义 key 和自定义缓存管理者，具有相当的灵活性和扩展性。

Spring Boot 中 Cache 的使用

Spring Boot 提供了非常简单的解决方案，这里给大家演示最核心的三个注解：@Cacheable、@CacheEvict、@CachePut。**spring-boot-starter-cache** 是 Spring Boot 体系内提供使用 Spring Cache 的 Starter 包。

在开始使用这三个注解之前，来介绍一个新的组件 spring-boot-starter-cache。

```
<dependency>
    <groupId>org.springframework.boot</groupId>
    <artifactId>spring-boot-starter-cache</artifactId>
</dependency>
```

spring-boot-starter-cache 是 Spring Boot 提供缓存支持的 starter 包，其会进行缓存的自动化配置和识别，Spring Boot 为 Redis 自动配置了 RedisCacheConfiguration 等信息，spring-boot-starter-cache 中的注解也主要是使用了 Spring Cache 提供的支持。

@Cacheable

@Cacheable 用来声明方法是可缓存的，将结果存储到缓存中以便后续使用相同参数调用时不需执行实际的方法，直接从缓存中取值。@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。

我们先来一个最简单的例子体验一下：

```
@RequestMapping("/hello")
@Cacheable(value="helloCache")
public String hello(String name) {
    System.out.println("没有走缓存！");
    return "hello "+name;
}
```

来测试一下，启动项目后访问网址 http://localhost:8080/hello?name=neo，输出：没有走缓存！，再次访问网址 http://localhost:8080/hello?name=neo，输出栏没有变化，说明这次没有走 hello() 这个方法，内容直接由缓存返回。

@Cacheable(value="helloCache") 这个注释的意思是，当调用这个方法时，会从一个名叫 helloCache 的缓存中查询，如果没有，则执行实际的方法（也可是查询数据库），并将执行的结果存入缓存中，否则返回缓存中的对象。这里的缓存中的 key 就是参数 name，value 就是返回的 String 值。

@Cacheable 支持如下几个参数。

- value：缓存的名称。
- key：缓存的 key，可以为空，如果指定要按照 SpEL 表达式编写；如果不指定，则缺省按照方法的所有参数进行组合。
- condition：触发条件，只有满足条件的情况才会加入缓存，默认为空，既表示全部都加入缓存，支持 SpEL。

我们把上面的方法稍微改成这样：

```
@RequestMapping("/condition")
@Cacheable(value="condition",condition="#name.length() <= 4")
public String condition(String name) {
    System.out.println("没有走缓存！");
    return "hello "+name;
}
```

启动后在浏览器中输入网址 http://localhost:8080/condition?name=neo，第一次输出栏输出：没有走缓存！再次执行无输出，表明已经走缓存。在浏览器中输入网址 http://localhost:8080/condition?name=ityouknow，浏览器执行多次仍然一直输出：没有走缓存！说明条件 condition 生效。

结合数据库的使用来做测试：

```
@RequestMapping("/getUsers")
@Cacheable(value="usersCache",key="#nickname",condition="#nickname.length() >= 6")
public List<User> getUsers(String nickname) {
    List<User> users=userRepository.findByNickname(nickname);
    System.out.println("执行了数据库操作");
    return users;
}
```

启动后在浏览器中输入网址 http://localhost:8080/getUsers?nickname=neo。

输出栏输出：

```
Hibernate: select user0_.id as id1_0_, user0_.email as email2_0_, user0_.nickname
as nickname3_0_, user0_.pass_word as pass_wor4_0_, user0_.reg_time as reg_time5_0_
, user0_.user_name as user_nam6_0_ from user user0_ where user0_.nickname=?
执行了数据库操作
```

多次执行，仍然输出上面的结果，说明每次请求都执行了数据库操作，再输入 http://localhost:8080/getUsers?nickname=ityoukonw 进行测试。只有第一次返回了上面的内容，再次执行输出栏没有变化，说明后面的请求都已经从缓存中拿取了数据。

最后总结一下：当执行到一个被 @Cacheable 注解的方法时，Spring 首先检查 condition 条件是否满足，如果不满足，执行方法，返回；如果满足，在缓存空间中查找使用 key 存储的对象，如果找到，将找到的结果返回，如果没有找到执行方法，将方法的返回值以 key-value 对象的方式存入缓存中，然后方法返回。

> 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。

@CachePut

项目运行中会对数据库的信息进行更新，如果仍然使用 @Cacheable 就会导致数据库的信息和缓存的信息不一致。在以往的项目中，我们一般更新完数据库后，再手动删除掉 Redis 中对应的缓存，以保证数据的一致性。Spring 提供了另外的一种解决方案，可以让我们以优雅的方式去更新缓存。

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。

以上面的方法为例，我们再来做一个测试：

```
@RequestMapping("/getPutUsers")
@CachePut(value="usersCache",key="#nickname")
public List<User> getPutUsers(String nickname) {
    List<User> users=userRepository.findByNickname(nickname);
    System.out.println("执行了数据库操作");
    return users;
}
```

我们新增一个 getPutUsers 方法，value、key 设置和 getUsers 方法保持一致，使用 @CachePut。同时手动在数据库插入一条 nikename 为 ityouknow 的用户数据。

```
INSERT INTO `user` VALUES ('1', 'ityouknow@126.com', 'ityouknow', '123456', '2018'
, 'keepSmile');
```

在浏览器中输入网址 http://localhost:8080/getUsers?nickname=ityouknow，并没有返回用户昵称为 ityouknow 的用户信息，再次输入网址 http://localhost:8080/getPutUsers?nickname=ityouknow 可以查看到此用户的信息，再次输入网址 http://localhost:8080/getUsers?nickname=ityouknow 就可以看到用户昵称为 ityouknow 的信息了。

说明执行在方法上声明 @CachePut 会自动执行方法，并将结果存入缓存。

@CachePut 配置方法

- value 缓存的名称。
- key 缓存的 key，可以为空，如果指定要按照 SpEL 表达式编写，如果不指定，则缺省按照方法的所有参数进行组合。
- condition 缓存的条件，可以为空，使用 SpEL 编写，返回 true 或者 false，只有为 true 才进行缓存。

可以看出 @CachePut 的参数和使用方法基本和 @Cacheable 一致。

> @CachePut 也可以标注在类上和方法上。

@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。下面来介绍一下新出现的两个

属性 allEntries 和 beforeInvocation。

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。

在上一个方法中我们使用注解：`@CachePut(value="usersCache",key="#nickname")` 来更新缓存，但如果不写 `key="#nickname"`，Spring Boot 会以默认的 key 值去更新缓存，导致最上面的 getUsers() 方法并没有获取最新的数据。但是现在我们使用 @CacheEvict 就可以解决这个问题了，它会将所有以 usersCache 为名的缓存全部清除。我们来看个例子：

```
@RequestMapping("/allEntries")
@CacheEvict(value="usersCache", allEntries=true)
public List<User> allEntries(String nickname) {
    List<User> users=userRepository.findByNickname(nickname);
    System.out.println("执行了数据库操作");
    return users;
}
```

手动修改用户表的相关信息，比如注册时间。在浏览器中输入网址 http://localhost:8080/getUsers?nickname=ityouknow 发现缓存中的数据并没有更新，再次访问地址 http://localhost:8080/getUsers?nickname=ityouknow 会发现数据已经更新，并且输出栏输出“执行了数据库操作”，这表明已经将名为 usersCache 的缓存记录清空了。

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。

```
@RequestMapping("/beforeInvocation")
@CacheEvict(value="usersCache", allEntries=true, beforeInvocation=true)
public void beforeInvocation() {
    throw new RuntimeException("test beforeInvocation");
}
```

我们来做一个测试，在方法中添加一个异常，访问网址 http://localhost:8080/beforeInvocation 查看 usersCache 的缓存是否被更新。

按照上面的实验步骤，手动修改用户表的相关信息，访问网址 http://localhost:8080/getUsers?nickname=ityouknow 发现缓存中的数据并没有更新；再访问网址 http://localhost:8080/beforeInvocation 会报错误，先不用管这里，再次访问地址 http://localhost:8080/getUsers?nickname=ityouknow 会发现数据已经更新，并且输出栏输出“执行了数据库操作”。这表明虽然在测试的过程中方法抛出了异常，但缓存中名为

usersCache 的记录都已被清空。

总结一下其作用和配置方法

@Cacheable 作用和配置方法

主要针对方法配置，能够根据方法的请求参数对其结果进行缓存：

主要参数	解释	举例
value	缓存的名称，在 spring 配置文件中定义，必须指定至少一个	如 @Cacheable(value="mycache") 或者 @Cacheable(value={"cache1"，"cache2"}
key	缓存的 key，可以为空，如果指定要按照 SpEL 表达式编写，如果不指定，则缺省按照方法的所有参数进行组合	如 @Cacheable(value="testcache"，key="#userName")
condition	缓存的条件，可以为空，使用 SpEL 编写，返回 true 或者 false，只有为 true 才进行缓存	如 @Cacheable(value="testcache"，condition="#userName.length()>2")

@CachePut 作用和配置方法

@CachePut 的作用主要针对方法配置，能够根据方法的请求参数对其结果进行缓存，和 @Cacheable 不同的是，它每次都会触发真实方法的调用。

主要参数	解释	举例
value	缓存的名称，在 spring 配置文件中定义，必须指定至少一个	如 @Cacheable(value="mycache") 或者 @Cacheable(value={"cache1"，"cache2"}
key	缓存的 key，可以为空，如果指定要按照 SpEL 表达式编写，如果不指定，则缺省按照方法的所有参数进行组合	如 @Cacheable(value="testcache"，key="#userName")
condition	缓存的条件，可以为空，使用 SpEL 编写，返回 true 或者 false，只有为 true 才进行缓存	如 @Cacheable(value="testcache"，condition="#userName.length()>2")

@CacheEvict 作用和配置方法

主要针对方法配置，能够根据一定的条件对缓存进行清空。

主要参数	解释	举例
value	缓存的名称，在 spring 配置文件中定义，必须指定至少一个	如 @CachEvict(value="mycache") 或者 @CachEvict(value={"cache1"，"cache2"}
key	缓存的 key，可以为空，如果指定要按照 SpEL 表达式编写，如果不指定，则缺省按照方法的所有参数进行组合	如 @CachEvict(value="testcache"，key="#userName")
condition	缓存的条件，可以为空，使用 SpEL 编写，返回 true 或者 false，只有为 true 才清空缓存	如 @CachEvict(value="testcache"，condition="#userName.length()>2")
allEntries	是否清空所有缓存内容，缺省为 false，如果指定为 true，则方法调用后将立即清空所有缓存	如 @CachEvict(value="testcache"，allEntries=true)
beforeInvocation	是否在方法执行前就清空，缺省为 false，如果指定为 true，则在方法还没有执行的时候就清空缓存，缺省情况下，如果方法执行抛出异常，则不会清空缓存	如 @CachEvict(value="testcache"，beforeInvocation=true)

> @Cacheable、@CacheEvict、@CachePut 三个注解非常灵活，满足了我们对数据缓存的绝大多数使用场景，并且使用起来非常的简单而又强大，在实际工作中我们可以灵活搭配使用。

总结

Spring 提供了基于注释驱动的 Spring Cache，它是一个对缓存使用的抽象，将我们常用的缓存策略都进行了高度抽象，让我们在项目中使用时只需要添加几个注解，即可完成大多数缓存策略的实现。Spring Boot Starter Cache 是 Spring Boot 提供给我们在 Spring Boot 中使用 Spring Cache 的 Starter 包，集成后方便在 Spring Boot 体系中使用缓存。

> 点击这里下载源码。