作者:Wilson

Spring+Redis缓存

一、缓存概念

基本机制：先从缓存中读取数据，如果没有再从慢速设备上读取实际数据（数据也会存入缓存）

需要被缓存的数据：

- 经常读取且不经常修改的数据
- 昂贵（CPU/IO）的且对于相同的请求有相同的计算结果的数据

缓存命中率

- 命中率 = 从缓存中读取次数 / (总读取次数[从缓存中读取次数 + 从慢速设备上读取的次数])
- Miss率 = 没有从缓存中读取的次数 / (总读取次数[从缓存中读取次数 + 从慢速设备上读取的次数])

TTL（Time To Live）

存活期，即从缓存中创建时间点开始直到它到期的一个时间段（不管在这个时间段内有没有访问都将过期

TTI（Time To Idle）

空闲期，即一个数据多久没被访问将从缓存中移除的时间。

Eviction policy

移除策略，即如果缓存满了，从缓存中移除数据的策略 常见：

- FIFO（First In First Out）：先进先出算法，即先放入缓存的先被移除；
- LRU（Least Recently Used）：最久未使用算法，使用时间距离现在最久的那个被移除；
- LFU（Least Frequently Used）：最近最少使用算法，一定时间段内使用次数（频率）最少的那个被移除；
- 带过期时间

二、 Spring Cache

自Spring 3.1起，提供了类似于@Transactional注解事务的注解Cache支持，且提供了Cache抽象，在此之前一般通过AOP实现。

> 使用Spring Cache的好处：
>
> - 提供基本的Cache抽象，方便切换各种底层Cache；
> - 通过注解Cache可以实现类似于事务一样，缓存逻辑透明的应用到我们的业务代码上，且只需要更少的代码就可以完成；
> - 提供事务回滚时也自动回滚缓存；
> - 支持比较复杂的缓存逻辑；

常见注解

> 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。

@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**，(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:触发缓存写入。
- @CacheEvict:触发缓存清除。
- @CachePut:更新缓存(不会影响到方法的运行)。
- @Caching:重新组合要应用于方法的多个缓存操作。
- @CacheConfig:设置类级别上共享的一些常见缓存设置。

基本概念

名称	解释
Cache	缓存接口，定义缓存操作。实现有：RedisCache、EhCacheCache、ConcurrentMapCache等
CacheManager	缓存管理器，管理各种缓存（cache）组件
@Cacheable	主要针对方法配置，能够根据方法的请求参数对其进行缓存
@CacheEvict	驱逐缓存
@CachePut	保证方法被调用，又希望结果被缓存。 与@Cacheable区别在于是否每次都调用方法，常用于更新
@EnableCaching	开启基于注解的缓存
keyGenerator	缓存数据时key生成策略
serialize	缓存数据时value序列化策略
@CacheConfig	统一配置本类的缓存注解的属性

@Cacheable/@CachePut/@CacheEvict 主要的参数

名称	解释
value	缓存的名称，在 spring 配置文件中定义，必须指定至少一个 例如：@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 例如： @Cacheable(value=”testcache”,key=”#id”)
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/清除缓存 例如：@Cacheable(value=”testcache”,condition=”#userName.length()>2”)
unless	否定缓存。当条件结果为TRUE时，就不会缓存。 @Cacheable(value=”testcache”,unless=”#userName.length()>2”)
allEntries (@CacheEvict)	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 例如： @CachEvict(value=”testcache”,allEntries=true)
beforeInvocation (@CacheEvict)	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 例如： @CachEvict(value=”testcache”， beforeInvocation=true)

SpEL上下文数据

SpEL **(Spring Expression Language),即Spring表达式语言,是比JSP的EL更强大的一种表达式语言。**

Spring Cache提供了一些供我们使用的SpEL上下文数据，下表直接摘自Spring官方文档：

名称	位置	描述	示例
methodName	root对象	当前被调用的方法名	`#root.methodname`
method	root对象	当前被调用的方法	`#root.method.name`
target	root对象	当前被调用的目标对象实例	`#root.target`
targetClass	root对象	当前被调用的目标对象的类	`#root.targetClass`
args	root对象	当前被调用的方法的参数列表	`#root.args[0]`
caches	root对象	当前方法调用使用的缓存列表	`#root.caches[0].name`
Argument Name	执行上下文	当前被调用的方法的参数如findArtisan(Artisan artisan),可以通过#artsian.id获得参数	`#artsian.id`
result	执行上下文	方法执行后的返回值（当方法执行后的判断有效，如 unless cacheEvict的beforeInvocation=false）	`#result`

SpEL提供了多种运算符

类型	运算符
关系	<，>，<=，>=，==，!=，lt，gt，le，ge，eq，ne
算术	+，-，*，/，%，^
逻辑	&&，||，!，and，or，not，between，instanceof
条件	?: (ternary)，?: (elvis)
正则表达式	matches
其他类型	?.，?[...]，![...]，^[...]，$[...]

三、SpringBoot实战Cache

整合Redis

导入依赖:

```
<!--默认只需导入依赖依赖即可 SpringBoot的CacheManager就会使用RedisCache
-->
<dependency>
        <groupId>org.springframework.boot</groupId>
        <artifactId>spring-boot-starter-data-redis</artifactId>
</dependency>
```

导入application.yml:

```
spring:
redis:
 port: 6379
 password: 123456
 host: 192.168.1.153
 lettuce:
   pool:
     max-active: 8
     max-idle: 8
     min-idle: 0
     max-wait: 1000
   shutdown-timeout: 100
 database: 12
```

启动入口类上方加入注解@EnableCaching

```
package com.qfjy;

import org.springframework.boot.SpringApplication;
import org.springframework.boot.autoconfigure.SpringBootApplication;
import org.springframework.cache.annotation.EnableCaching;

@EnableCaching  //启动缓存配置  使用@EnableCaching启用Cache注解支持
@SpringBootApplication
public class Ch3BootRedisCacheApplication
```

四、Cache注解实战

缓存名称描述

1、最简单的使用方式,注解名称=缓存名称

```
//缓存名称=注解名称，默认是 user::id
@Cacheable("user")
public User selectById1(String id){
    return  userMapper.selectById(id);
}
```

2、一个方法可以对应两个缓存名称，如下

```
@Cacheable({"user","user1"})
public User selectById1(String id){
    return  userMapper.selectById(id);
}
```

3 、@Cacheable的缓存名称是可以配置动态参数的

比如选择传入的参数,如下: (以下示例是使用SpEL声明,如果您不熟悉SpEL，可以阅读Spring Expression Language)

SpEL上下文数据（参考）

@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**，(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)

```
// 注解方法应用
public @interface Cacheable {
 String[] value();              //缓存的名称
 String key() default "";       //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
 String condition() default "";//缓存的条件(入参)，可以为空，使用 SpEL 编写，返回 true 或者 false，只有为 true 才进行缓存
 String unless() default ""; //函数返回值符合表达式条件的，（否决）、不缓存
```

案例1:

```
// 应用到读取数据的方法上，即可缓存的方法，如查找方法：先从缓存中读取，如果没有再调用方法获取数据，然后把数据添加到缓存中
@Cacheable(value = "user",key = "#id")
public User selectById(String id){
 return  userMapper.selectById(id);
}
```

案例2：@Cacheable还可以设置根据条件判断是否需要缓存

- condition:取决于给定的参数是否满足条件
- unless:取决于返回值是否满足条件
- unless参数的作用是：函数返回值符合表达式条件的，（否决）、不缓存

```
//根据条件来判断  多个参数组合
// condition 长度小于5 进行缓存
@Cacheable(value = "user",key = "#id",condition = "#id.length()<5")
public User selectById3(String id){
    return  userMapper.selectById(id);
}
```

案例3：函数返回值符合表达式条件的，（否决）、不缓存

```
/**
     key:users :: id
     condition: 长度小于5 进行缓存
     unless:返回值中age=20 不进行缓存
*/
@Cacheable(value = "users",key = "#id",condition = "#id.length()<5",unless = "#result.age==20")
public User selectById5(String id){
 return  userMapper.selectById(id);
}
```

@CachePut更新缓存

```
public @interface CachePut {
 String[] value();              //缓存的名字，可以把数据写到多个缓存
 String key() default "";       //缓存key，如果不指定将使用默认的KeyGenerator生成
 String condition() default "";//满足缓存条件的数据才会放入缓存，condition在调用方法之前和之后都会判断
 String unless() default "";  //用于否决缓存更新的，不像condition，该表达只在方法执行之后判断，此时可以拿到返回值result进行判断了
}
```

应用到写数据的方法上，如新增/修改方法，调用方法时会自动把相应的数据放入缓存

```
@CachePut(value = "user",key = "#user.id")
 public int insertUser(User user){
    return userMapper.insertUser(user);
 }
```

@CacheEvict 触发缓存清除

@CacheEvict:删除缓存的注解,这对删除旧的数据和无用的数据是非常有用的。这里还多了一个参数(allEntries),设置allEntries=true时，可以对整个条目进行批量删除。

```
// 注解 驱逐缓存
public @interface CacheEvict {
 String[] value();                          //请参考@CachePut
 String key() default "";                   //请参考@CachePut
 String condition() default "";             //请参考@CachePut
 boolean allEntries() default false;        //是否移除所有数据
 boolean beforeInvocation() default false;//是调用方法之前移除/还是调用之后移除
}
```

案例1：移除指定key的数据

```
//驱逐缓存
 @CacheEvict(value = "user",key="#id")
 public int deleteById(String id){
     return userMapper.deleteById(id);
 }
```

案例2：移除所有数据

```
 //删除所有数据 和清空 user的所有缓存
 @CacheEvict(value = "user",allEntries = true)
 public int deleteAll(){
     return userMapper.deleteAll();
 }
```

@CacheConfig注解

@CacheConfig:缓存提供了许多的注解选项，但是有一些公用的操作，我们可以使用@CacheConfig在类上进行全局设置。

```java
/**
 * @ClassName UserServiceImpl
 * @Description TODO
 * @Author guoweixin
 * @Date 2018/7/18
 * @Version 1.0
 */
@Service
@CacheConfig(cacheNames = "user")
public class UserServiceImpl implements UserService {
```