ehcache在springboot中的使用

# 一、springboot中的简单使用

## 1、引入jar

<!-- ehchache -->

<dependency>

<groupId>org.springframework.boot</groupId>

<artifactId>spring-boot-starter-cache</artifactId>

</dependency>

<dependency>

<groupId>net.sf.ehcache</groupId>

<artifactId>ehcache</artifactId>

</dependency>

## 2、ehcache.xml

<?xml version="1.0" encoding="UTF-8"?>

<ehcache xmlns:xsi="http://www.w3.org/2001/XMLSchema-instance"

xsi:noNamespaceSchemaLocation="http://ehcache.org/ehcache.xsd"

updateCheck="false">

<diskStore path="java.io.tmpdir/Tmp\_EhCache"/>

<defaultCache eternal="false" maxElementsInMemory="1000"

overflowToDisk="false" diskPersistent="false" timeToIdleSeconds="0"

timeToLiveSeconds="600" memoryStoreEvictionPolicy="LRU"/>

<cache name="user" eternal="false" maxElementsInMemory="10000"

overflowToDisk="false" diskPersistent="false"

timeToIdleSeconds="0" timeToLiveSeconds="0"

memoryStoreEvictionPolicy="LFU"/>

</ehcache>

## 3、application.yml

spring:

cache:

type: ehcache

ehcache:

config: classpath:config/ehcache.xml

## 4、将@EnableCaching 放置于BootdoApplication类（启动类）上

## 5、业务层方法上加入注解

@Override

@Cacheable(value="user",key="0")

public List<MessageDO> gonggao() {

System.out.println("没有去缓存得地方");

return messageDao.gonggao();

}

# 二 、ehcache和redis的差别

1、ehcache直接在jvm虚拟机中缓存，与java程序捆绑在一起，别的独立程序拿不到数据，速度快，效率高；但是缓存共享麻烦，集群分布式应用不方便。

2、redis是独立运行的程序，通过socket访问到缓存服务，效率比ecache低，比数据库要快很多，处理集群和分布式缓存方便，有成熟的方案。

使用场景：

如果是单个应用或者对缓存访问要求很高的应用，用ehcache。

如果是大型系统，存在缓存共享、分布式部署、缓存内容很大的，建议用redis。

# 三、详解注解

@Cacheable

几个属性：

cachename/value:指定缓存组件得名字；将方法返回的结果放在那个缓存中，是数组得方式，可以指定多个缓存。（value={“user”,“message”}）

key:缓存数据使用的key；可以用来指定。默认是使用方法参数的值

编写SpEL；#id；#a0 等价 #root.args[0] 第一个参数

keyGenerator:key的生成器；可以自己指定key的生成器组件id

key/keyGenerator:二选一使用

cacheManager:指定缓存管理器；或者cacheResolver指定获取解析器

condition：指定符合条件的情况下才缓存；

condition="#id>0"

unless:否定缓存；

unless = “#result == null”

sync：是否使用异步模式

自定义key

keyGenerator=“myKeyGenerator”

@Configuration

public class MyCacheConfig{

@Bean('myKeyGenerator')

public KeyGenerator keyGenerator(){

return new KeyGenerator(){

@Override

public Object generate(Object target,Method method,Object...params){

return method.getName()+"["+Arrays.asList(params).toString()+"]";

}

}

}

}

运行流程：

1、方法运行之前，先去查询Cache（缓存组件），按照cacheNames指定的名字获取；

（组件目前是在cache.xml中配置，理论上CacheManager先获取相应的缓存，第一次如果没有的话就会自动创建，但是目前xml中不配置就会找不到报错）

2、去Cache中查找缓存的内容，使用一个key，默认的是方法的参数；

key是按照某种策略生成的，SimpleKeyGenerator生成策略:没有参数 key=new SimpleKey（）；一个参数则 key=参数的值，多个参数的话key=new SimpleKey（params);

3、没查到就会调用目标方法。

4、将返回的结果放到缓存中

@CachePut:既调用方法，又更新缓存

修改了数据库的某个数据，同时更新缓存

运行时机：

1、先调用目标方法

2、将目标方法的结果缓存起来

@CachePut(value=“uer”,key="#result.id")

其中@Cacheable不能用#result.id

@CacheEvict:缓存清除

key:指定药清除的数据

allEntries=true;指定清除这个缓存中所有的数据

beforeInvocation=false；缓存的清除在方法之后执行

默认代表是在方法之后执行，如果出现异常，就不会清除

beforeInvocation=ture；在方法之前执行，无论是否异常，都会执行。

@Caching 定义复杂的缓存规则

@Caching(

cacheable = {

@Cacheable(value="user",key="#name")

},

put = {

@CachePut(value="user",key="#result.id"),

@CachePut(value="user",key="#result.id")

}

)

# 四、mybatis-ehcache



# 五、附件

