# EhCache

是一个存Java的进程内缓存框架，具有快速，简单的特点，Hibernate中默认就是Cacheprovider

## 与Redis的区别

应用场景：

**ehcache直接在jvm虚拟机中缓存，速度快，效率高；但是缓存共享麻烦，集群分布式应用不方便。**

**redis是通过socket访问到缓存服务，效率比ehcache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。**

**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。**

## EhCache配置

属性说明：

diskStore：指定数据在磁盘中的存储位置。

defaultCache：当借助CacheManager.add(“demoCache”)创建Cache时，EhCache便会采用指定的的管理策略

以下属性是必须的：

maxElementsInMemory - 在内存中缓存的element的最大数目

maxElementsOnDisk - 在磁盘上缓存的element的最大数目，若是0表示无穷大

eternal - 设定缓存的elements是否永远不过期。如果为true，则缓存的数据始终有效，如果为false那么还要根据timeToIdleSeconds，timeToLiveSeconds判断

overflowToDisk - 设定当内存缓存溢出的时候是否将过期的element缓存到磁盘上

以下属性是可选的：

timeToIdleSeconds - 当缓存在EhCache中的数据前后两次访问的时间超过timeToIdleSeconds的属性取值时，这些数据便会删除，默认值是0,也就是可闲置时间无穷大

timeToLiveSeconds - 缓存element的有效生命期，默认是0.,也就是element存活时间无穷大

diskSpoolBufferSizeMB 这个参数设置DiskStore(磁盘缓存)的缓存区大小.默认是30MB.每个Cache都应该有自己的一个缓冲区.

diskPersistent - 在VM重启的时候是否启用磁盘保存EhCache中的数据，默认是false。

diskExpiryThreadIntervalSeconds - 磁盘缓存的清理线程运行间隔，默认是120秒。每个120s，相应的线程会进行一次EhCache中数据的清理工作

memoryStoreEvictionPolicy - 当内存缓存达到最大，有新的element加入的时候， 移除缓存中element的策略。默认是LRU（最近最少使用），可选的有LFU（最不常使用）和FIFO（先进先出）

## Spring 整合 Ehcache

 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

### 基于注解

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。

#### 基于注解的三个解析

**1.@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可以指定三个属性，value、key和condition。

**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。而@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。  
**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。

#### 注解属性

属性一：Value 必须指定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，为ehcache.xml中的<cache name="myCache"/> 。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

属性二：Key 当没有改属性时，Spring会默认调用生成，其中有两种策略

ey的生成策略有两种：一种是默认策略，一种是自定义策略  
       ¹默认的key生成策略是通过KeyGenerator生成的，其默认策略如下：  
            1.如果方法没有参数，则使用0作为key。  
            2.如果只有一个参数的话则使用该参数作为key。  
            3.如果参数多余一个的话则使用所有参数的hashCode作为key  
        ²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

**属性三** ：condition

    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;  
              }  
**属性四** ：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);  
              }  
**属性五** ：beforeInvocation

    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);  
             }

**注意**我们也可以使用ehcache的去除策略最近使用(LRU)"策略,其它还有先入先出FIFO,最少使用LFU,较少使用LRU

#### 注解配置

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。

       <cache:annotation-driven/>有一个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。

## ehcache.xml配置

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

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

xsi:noNamespaceSchemaLocation="ehcache.xsd"

updateCheck="true" monitoring="autodetect"

dynamicConfig="true">

<diskStore path="java.io.tmpdir"/>

<defaultCache

maxEntriesLocalHeap="10000"

eternal="false"

overflowToDisk="false"

timeToIdleSeconds="120"

timeToLiveSeconds="120"

diskSpoolBufferSizeMB="30"

maxEntriesLocalDisk="10000000"

diskExpiryThreadIntervalSeconds="120"

memoryStoreEvictionPolicy="LRU">

<persistence strategy="localTempSwap"/>

</defaultCache>

<cache name="myCache"

maxEntriesLocalHeap="10000"

maxEntriesLocalDisk="1000"

eternal="false"

diskSpoolBufferSizeMB="30"

timeToIdleSeconds="300"

timeToLiveSeconds="600"

memoryStoreEvictionPolicy="LFU"

transactionalMode="off">

<persistence strategy="localTempSwap"/>

</cache>

</ehcache>

## 3.spring-ehcache.xml配置

<!-- 启用缓存注解功能，这个是必须的，否则注解不会生效，另外，该注解一定要声明在spring主配置文件中才会生效 -->

<cache:annotation-driven cache-manager="ehcacheManager"/>

<!-- cacheManager工厂类，指定ehcache.xml的位置 -->

<bean id="ehcacheManagerFactory" class="org.springframework.cache.ehcache.EhCacheManagerFactoryBean"> <property name="configLocation" value="classpath:ehcache.xml" />

</bean>   
<!--声明cheManager -->

<bean id="ehcacheManager" class="org.springframework.cache.ehcache.EhCacheCacheManager">

<property name="cacheManager" ref="ehcacheManagerFactory" />

</bean>