# 1.添加依赖

*<!-- 集成ehcache需要的依赖-->*<**dependency**>  
 <**groupId**>net.sf.ehcache</**groupId**>  
 <**artifactId**>ehcache</**artifactId**>  
</**dependency**>

# 2.配置数据库和数据源jpa（自己设置）

# 3.缓存配置文件

*/\*\*  
 \* 缓存配置.  
 \** ***@author*** *Angel(QQ*:*412887952)  
 \** ***@version*** *v.0.1  
 \*/*@Configuration  
@EnableCaching *//标注启动缓存.***public class** CacheConfiguration {  
   
 */\*\*  
 \* ehcache 主要的管理器  
 \** ***@param bean*** *\** ***@return*** *\*/* @Bean  
 **public** EhCacheCacheManager ehCacheCacheManager(EhCacheManagerFactoryBean bean){  
 System.***out***.println(**"CacheConfiguration.ehCacheCacheManager()"**);   
 **return new** EhCacheCacheManager(bean.getObject());  
 }  
   
 */\*  
 \* 据shared与否的设置,  
 \* Spring分别通过CacheManager.create()  
 \* 或new CacheManager()方式来创建一个ehcache基地.  
 \*   
 \* 也说是说通过这个来设置cache的基地是这里的Spring独用,还是跟别的(如hibernate的Ehcache共享)  
 \*   
 \*/* @Bean  
 **public** EhCacheManagerFactoryBean ehCacheManagerFactoryBean(){  
 System.***out***.println(**"CacheConfiguration.ehCacheManagerFactoryBean()"**);  
 EhCacheManagerFactoryBean cacheManagerFactoryBean = **new** EhCacheManagerFactoryBean ();  
 cacheManagerFactoryBean.setConfigLocation (**new** ClassPathResource(**"conf/ehcache.xml"**));  
 cacheManagerFactoryBean.setShared(**true**);  
 **return** cacheManagerFactoryBean;  
 }  
   
}

# 4. 编写ehcache.xml配置文件

*<?***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：为缓存路径，ehcache分为内存和磁盘两级，此属性定义磁盘的缓存位置。参数解释如下：  
 user.home – 用户主目录  
 user.dir – 用户当前工作目录  
 java.io.tmpdir – 默认临时文件路径  
   
 -->* <**diskStore path="java.io.tmpdir/Tmp\_EhCache"** />  
   
 *<!--   
 defaultCache： 默认缓存策略，当ehcache找不到定义的缓存时，则使用这个缓存策略。只能定义一个。  
 -->  
   
 <!--   
 name:缓存名称。   
 maxElementsInMemory:缓存最大数目   
 maxElementsOnDisk：硬盘最大缓存个数。   
 eternal:对象是否永久有效，一但设置了，timeout将不起作用。   
 overflowToDisk:是否保存到磁盘，当系统当机时  
 timeToIdleSeconds:设置对象在失效前的允许闲置时间（单位：秒）。仅当eternal=false对象不是永久有效时使用，可选属性，默认值是0，也就是可闲置时间无穷大。   
 timeToLiveSeconds:设置对象在失效前允许存活时间（单位：秒）。最大时间介于创建时间和失效时间之间。仅当eternal=false对象不是永久有效时使用，默认是0.，也就是对象存活时间无穷大。  
 diskPersistent：是否缓存虚拟机重启期数据 Whether the disk store persists between restarts of the Virtual Machine. The default value is false.   
 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。   
 diskExpiryThreadIntervalSeconds：磁盘失效线程运行时间间隔，默认是120秒。   
 memoryStoreEvictionPolicy：当达到maxElementsInMemory限制时，Ehcache将会根据指定的策略去清理内存。默认策略是LRU（最近最少使用）。你可以设置为FIFO（先进先出）或是LFU（较少使用）。   
 clearOnFlush：内存数量最大时是否清除。   
 memoryStoreEvictionPolicy:可选策略有：LRU（最近最少使用，默认策略）、FIFO（先进先出）、LFU（最少访问次数）。   
 FIFO，first in first out，这个是大家最熟的，先进先出。   
 LFU， Less Frequently Used，就是上面例子中使用的策略，直白一点就是讲一直以来最少被使用的。如上面所讲，缓存的元素有一个hit属性，hit值最小的将会被清出缓存。  
 LRU，Least Recently Used，最近最少使用的，缓存的元素有一个时间戳，当缓存容量满了，而又需要腾出地方来缓存新的元素的时候，那么现有缓存元素中时间戳离当前时间最远的元素将被清出缓存。  
 -->* <**defaultCache   
 eternal="false"   
 maxElementsInMemory="1000"  
 overflowToDisk="false"   
 diskPersistent="false"   
 timeToIdleSeconds="0"  
 timeToLiveSeconds="600"   
 memoryStoreEvictionPolicy="LRU"** />  
  
 <**cache   
 name="demo"   
 eternal="false"   
 maxElementsInMemory="100"  
 overflowToDisk="false"   
 diskPersistent="false"   
 timeToIdleSeconds="0"  
 timeToLiveSeconds="300"   
 memoryStoreEvictionPolicy="LRU"** />  
   
</**ehcache**>

# 5 服务层

@Service

**public class** DemoInfoServiceImpl **implements** DemoInfoService {

    //这里的单引号不能少，否则会报错，被识别是一个对象;

**public static final** String ***CACHE\_KEY*** = "'demoInfo'";

     @Resource

**private** DemoInfoRepository demoInfoRepository;

    /\*\*

     \* value属性表示使用哪个缓存策略，缓存策略在ehcache.xml

    \*/

**public static final** String ***DEMO\_CACHE\_NAME*** = "demo";

    /\*\*

     \* 保存数据.

     \* **@param** demoInfo

     \*/

    @CacheEvict(value=***DEMO\_CACHE\_NAME***,key=***CACHE\_KEY***)

    @Override

**public** DemoInfo save(DemoInfo demoInfo){

**return**demoInfoRepository.save(demoInfo);

    }

    /\*\*

     \* 查询数据.

     \* **@param** id

     \* **@return**

     \*/

    @Cacheable(value=***DEMO\_CACHE\_NAME***,key="'demoInfo\_'+#id")

    @Override

**public** DemoInfo findById(Long id){

       System.***err***.println("没有走缓存！"+id);

**return**demoInfoRepository.findOne(id);

    }

    /\*\*

     \* http://www.mincoder.com/article/2096.shtml:

     \*

     \* 修改数据.

     \*

     \* 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。

     \*

     \* **@param** updated

     \* **@return**

     \*

     \* **@throws** NotFoundException

     \*/

    @CachePut(value = ***DEMO\_CACHE\_NAME***,key = "'demoInfo\_'+#updated.getId()")

    //@CacheEvict(value = DEMO\_CACHE\_NAME,key = "'demoInfo\_'+#updated.getId()")//这是清除缓存.

    @Override

**public** DemoInfo update(DemoInfo updated) **throws** NotFoundException{

       DemoInfo demoInfo = demoInfoRepository.findOne(updated.getId());

**if**(demoInfo == **null**){

**thrownew** NotFoundException("No find");

       }

       demoInfo.setName(updated.getName());

       demoInfo.setPwd(updated.getPwd());

**return**demoInfo;

    }

    /\*\*

     \* 删除数据.

     \* **@param** id

     \*/

    @CacheEvict(value = ***DEMO\_CACHE\_NAME***,key = "'demoInfo\_'+#id")//这是清除缓存.

    @Override

**public void** delete(Long id){

       demoInfoRepository.delete(id);

    }

}

# 6.测试controller

@RestController

**public class** DemoInfoController {

    @Resource

**private** DemoInfoService demoInfoService;

    @RequestMapping("/test")

**public** String test(){

       //存入两条数据.

        DemoInfo demoInfo = **new** DemoInfo();

        demoInfo.setName("张三");

        demoInfo.setPwd("123456");

        DemoInfo demoInfo2 = demoInfoService.save(demoInfo);

        //不走缓存.

        System.***out***.println(demoInfoService.findById(demoInfo2.getId()));

        //走缓存.

        System.***out***.println(demoInfoService.findById(demoInfo2.getId()));

        demoInfo = **new** DemoInfo();

        demoInfo.setName("李四");

        demoInfo.setPwd("123456");

        DemoInfo demoInfo3 = demoInfoService.save(demoInfo);

        //不走缓存.

        System.***out***.println(demoInfoService.findById(demoInfo3.getId()));

        //走缓存.

        System.***out***.println(demoInfoService.findById(demoInfo3.getId()));

        System.***out***.println("============修改数据=====================");

        //修改数据.

        DemoInfo updated = **new** DemoInfo();

        updated.setName("李四-updated");

        updated.setPwd("123456");

        updated.setId(demoInfo3.getId());

**try** {

           System.***out***.println(demoInfoService.update(updated));

       } **catch** (NotFoundException e) {

           e.printStackTrace();

       }

        //不走缓存.

        System.***out***.println(demoInfoService.findById(updated.getId()));

**return**"ok";

    }

}

# 7。测试成功

Hibernate: insert into demo\_info (name, pwd, state) values (?, ?, ?)

没有走缓存！52

DemoInfo [id=52, name=张三, pwd=123456, state=0]

DemoInfo [id=52, name=张三, pwd=123456, state=0]

Hibernate: insert into demo\_info (name, pwd, state) values (?, ?, ?)

没有走缓存！53

DemoInfo [id=53, name=李四, pwd=123456, state=0]

DemoInfo [id=53, name=李四, pwd=123456, state=0]

============修改数据=====================

DemoInfo [id=53, name=李四-updated, pwd=123456, state=0]

DemoInfo [id=53, name=李四-updated, pwd=123456, state=0]

C:\Users\ADMINI~1.ANG\AppData\Local\Temp\

Hibernate: insert into demo\_info (name, pwd, state) values (?, ?, ?)

没有走缓存！54

DemoInfo [id=54, name=张三, pwd=123456, state=0]

DemoInfo [id=54, name=张三, pwd=123456, state=0]

Hibernate: insert into demo\_info (name, pwd, state) values (?, ?, ?)

没有走缓存！55

DemoInfo [id=55, name=李四, pwd=123456, state=0]

DemoInfo [id=55, name=李四, pwd=123456, state=0]

============修改数据=====================

DemoInfo [id=55, name=李四-updated, pwd=123456, state=0]

DemoInfo [id=55, name=李四-updated, pwd=123456, state=0]