eHcache 缓存

# 1、ehcache介绍

## EhCache 是一个纯Java的进程内缓存框架，具有快速、精干等特点，是Hibernate中默认的CacheProvider。

Ehcache是一种广泛使用的开源Java分布式缓存。主要面向通用缓存,Java EE和轻量级容器。

它具有内存和磁盘存储，缓存加载器,缓存扩展,缓存异常处理程序,一个gzip缓存servlet过滤器,

支持REST和SOAP api等特点。 Ehcache最初是由Greg Luck于2003年开始开发。2009年,该项目被Terracotta购买。

软件仍然是开源,但一些新的主要功能(例如，快速可重启性之间的一致性的)只能在商业产品中使用，

例如Enterprise EHCache and BigMemory。,维基媒体Foundationannounced目前使用的就是Ehcache技术。

## 2、主要的特性有：

1. 快速

2. 简单

3. 多种缓存策略

4. 缓存数据有两级：内存和磁盘，因此无需担心容量问题

5. 缓存数据会在虚拟机重启的过程中写入磁盘

6. 可以通过RMI、可插入API等方式进行分布式缓存 7. 具有缓存和缓存管理器的侦听接口 8. 支持多缓存管理器实例，以及一个实例的多个缓存区域 9. 提供Hibernate的缓存实现

# 2、准备工作

## 1、依赖包

*<?*xml version="1.0" encoding="UTF-8"*?>*<project xmlns="http://maven.apache.org/POM/4.0.0" xmlns:xsi="http://www.w3.org/2001/XMLSchema-instance"  
 xsi:schemaLocation="http://maven.apache.org/POM/4.0.0 http://maven.apache.org/xsd/maven-4.0.0.xsd">  
 <modelVersion>4.0.0</modelVersion>  
  
 <groupId>sinoehcachespringboot</groupId>  
 <artifactId>sinoehcachespringboot</artifactId>  
 <version>0.0.1-SNAPSHOT</version>  
 <packaging>jar</packaging>  
  
 <name>sinoehcachespringboot</name>  
 <description>Demo project for Spring Boot</description>  
  
 <!-- spring boot 父节点依赖,  
 引入这个之后相关的引入就不需要添加version配置，  
 spring boot会自动选择最合适的版本进行添加。  
 -->  
 <parent>  
 <groupId>org.springframework.boot</groupId>  
 <artifactId>spring-boot-starter-parent</artifactId>  
 <version>1.5.9.RELEASE</version>  
 <relativePath/> <!-- lookup parent from repository -->  
 </parent>  
  
 <properties>  
 <project.build.sourceEncoding>UTF-8</project.build.sourceEncoding>  
 <project.reporting.outputEncoding>UTF-8</project.reporting.outputEncoding>  
 <java.version>1.8</java.version>  
 </properties>  
  
 <dependencies>  
 <dependency>  
 <groupId>org.springframework.boot</groupId>  
 <artifactId>spring-boot-starter-data-jpa</artifactId>  
 </dependency>  
 <dependency>  
 <groupId>org.springframework.boot</groupId>  
 <artifactId>spring-boot-starter-thymeleaf</artifactId>  
 </dependency>  
  
 <!-- spring boot web支持：mvc,aop... -->  
 <dependency>  
 <groupId>org.springframework.boot</groupId>  
 <artifactId>spring-boot-starter-web</artifactId>  
 </dependency>  
  
 <dependency>  
 <groupId>mysql</groupId>  
 <artifactId>mysql-connector-java</artifactId>  
 <scope>runtime</scope>  
 </dependency>  
  
 <!-- Spring boot单元测试. -->  
 <dependency>  
 <groupId>org.springframework.boot</groupId>  
 <artifactId>spring-boot-starter-test</artifactId>  
 <scope>test</scope>  
 </dependency>  
  
 <!--  
 包含支持UI模版（Velocity，FreeMarker，JasperReports），  
 邮件服务，  
 脚本服务(JRuby)，  
 缓存Cache（EHCache），  
 任务计划Scheduling（uartz）。  
 -->  
 <dependency>  
 <groupId>org.springframework</groupId>  
 <artifactId>spring-context-support</artifactId>  
 </dependency>  
  
 <!-- 集成ehcache需要的依赖-->  
 <dependency>  
 <groupId>net.sf.ehcache</groupId>  
 <artifactId>ehcache</artifactId>  
 </dependency>  
 </dependencies>  
  
 <build>  
 <plugins>  
 <plugin>  
 <groupId>org.springframework.boot</groupId>  
 <artifactId>spring-boot-maven-plugin</artifactId>  
 </plugin>  
 </plugins>  
 </build>  
  
  
</project>

## 2、配置数据库表数据源信息

########################################################  
###datasource ,mysql数据库连接配置  
########################################################  
spring.datasource.url = jdbc:mysql://localhost:3306/EhCache  
spring.datasource.username = root  
spring.datasource.password = \*\*\*\*\*  
spring.datasource.driverClassName = com.mysql.jdbc.Driver  
spring.datasource.max-active=20  
spring.datasource.max-idle=8  
spring.datasource.min-idle=8  
spring.datasource.initial-size=10  
  
  
  
########################################################  
### Java Persistence Api ，JPA自动建表操作配置  
########################################################  
spring.jpa.database = MYSQL  
spring.jpa.show-sql = true  
spring.jpa.hibernate.ddl-auto = update  
spring.jpa.hibernate.naming-strategy = org.hibernate.cfg.ImprovedNamingStrategy  
spring.jpa.properties.hibernate.dialect = org.hibernate.dialect.MySQL5Dialect

## 3、测试实体类

/\*\*  
 \* @Description 测试实体类  
 \* @Author HealerJean  
 \* @Date 2017/11/30 14:48.  
 \*/  
@Entity  
public class DemoInfo {  
 @Id  
 @GeneratedValue  
 private long id; //主键;  
 private String name;//名称;  
 private String pwd;//密码;  
 private int state;  
  
 public DemoInfo() {  
 }  
 public long getId() {  
 return id;  
 }  
 public void setId(long id) {  
 this.id = id;  
 }  
 public String getName() {  
 return name;  
 }  
 public void setName(String name) {  
 this.name = name;  
 }  
 public String getPwd() {  
 return pwd;  
 }  
 public void setPwd(String pwd) {  
 this.pwd = pwd;  
 }  
 public int getState() {  
 return state;  
 }  
 public void setState(int state) {  
 this.state = state;  
 }  
  
  
  
 @Override  
 public String toString() {  
 return "DemoInfo [id=" + id + ", name=" + name + ", pwd=" + pwd + ", state=" + state + "]";  
 }  
}

## 4、spring data jpa操作数据库

/\*\*  
 \* @Description  
 \* @Author HealerJean  
 \* @Date 2017/11/30 15:21.  
 \*/  
public interface DemoInfoRepository extends CrudRepository<DemoInfo,Long> {  
}

# 3、编辑ehcache.xml配置文件，用来配置缓存管理器工厂bean

*<?*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。缓存位置名称，也叫缓存策略名称  
 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" />  
  
 <!--name:缓存名称。demo  
 maxElementsInMemory:缓存最大数目 100  
 overflowToDisk:是否保存到磁盘，当系统当机时 false  
 diskPersistent：是否缓存虚拟机重启期数据 false  
 timeToIdleSeconds:设置对象在失效前的允许闲置时间（单位：秒） 0  
 timeToLiveSeconds:设置对象在失效前允许存活时间（单位：秒） 300  
 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）。  
 -->  
 <cache  
 name="demo"  
 eternal="false"  
 maxElementsInMemory="100"  
 overflowToDisk="false"  
 diskPersistent="false"  
 timeToIdleSeconds="0"  
 timeToLiveSeconds="300"  
 memoryStoreEvictionPolicy="LRU" />  
  
</ehcache>

# 4、缓存配置类，配置缓存管理器和缓存管理工厂（ehcache.xml），使用上面上面的配置文件

@Configuration  
@EnableCaching//标注启动缓存.  
public class CacheConfiguration {  
  
 /\*\*  
 \* ehcache.xml 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;  
 }  
  
}

# 5、service接口和实现

public interface DemoInfoService {  
  
 DemoInfo save(DemoInfo demoInfo);  
  
 void delete(Long id);  
  
 DemoInfo update(DemoInfo updated) throws NotFoundException;  
  
 DemoInfo findById(Long id);  
  
  
}

## 2、实现类 （注意这里保存的时候就开始存放了了，存放的是一个对象，"'demoInfo'";，它的key值默认就是 demoInfor\_id）

/\*\*  
 \* @Description  
 \* @Author HealerJean  
 \* @Date 2017/11/30 15:21.  
 \*/  
@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 也叫缓存存放位置名称，不能为空  
 \*/  
 public static final String ***DEMO\_CACHE\_NAME*** = "demo";  
  
 /\*\*  
 \* 保存数据，防止是更新的操作，所以将之前缓存的删除,事实上，我也并没有很成功的实现它  
 \* @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，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){  
 throw new 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、controller测试

@RestController  
public class DemoInfoController {  
  
 @Resource  
 private DemoInfoService demoInfoService;  
  
 @RequestMapping("/save")  
 public String test(){  
  
 //存入数据.  
 DemoInfo demoInfo = new DemoInfo();  
 demoInfo.setId(18);  
 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()));  
  
 return "ok";  
 }  
  
 @RequestMapping("/update")  
 public String update(long id ){  
  
 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(id);  
 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";  
 }  
  
 @RequestMapping("/delete")  
 public String delete(long id ){  
 demoInfoService.delete(id);  
 return "ok";  
 }  
  
 @RequestMapping("/findById")  
 public String findById(long id ){  
 System.***out***.println(demoInfoService.findById(id));  
  
 return "ok";  
 }  
  
}

## 1、测试，全部成功

# 7、代码位置

