spring boot redis注解缓存

类别

# 1、创建springboot项目，最重要的就是引入依赖

## 准备工作， 建立数据库表，和依赖关系

## 1、pom依赖如下

|  |
| --- |
| *<?*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>sinoRedisSpringBootId</groupId>  <artifactId>sinoredis\_springboot</artifactId>  <version>0.0.1-SNAPSHOT</version>  <packaging>jar</packaging>   <name>sinoredis\_springboot</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>    <!-- mysql 数据库驱动. -->  <dependency>  <groupId>mysql</groupId>  <artifactId>mysql-connector-java</artifactId>  </dependency>   <!-- 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>   <!-- 添加redis支持-->  <dependency>  <groupId>org.springframework.boot</groupId>  <artifactId>spring-boot-starter-redis</artifactId>  <version>1.4.7.RELEASE</version>  </dependency>   </dependencies>   <build>  <plugins>  <plugin>  <groupId>org.springframework.boot</groupId>  <artifactId>spring-boot-maven-plugin</artifactId>  </plugin>  </plugins>  </build>   </project> |

## 2、创建数据库表DemoInfo

@Entity  
public class DemoInfo implements Serializable{  
 private static final long ***serialVersionUID*** = 1L;  
 @Id @GeneratedValue  
 private long id;  
 private String name;  
 private String pwd;  
 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;  
 }  
   
 @Override  
 public String toString() {  
 return "DemoInfo [id=" + id + ", name=" + name + ", pwd=" + pwd + "]";  
 }  
}

## 3、配置springboot配置文件

########################################################  
###datasource 配置MySQL数据源；  
########################################################  
spring.datasource.url = jdbc:mysql://localhost:3306/redisdb  
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  
  
  
  
########################################################  
###REDIS (RedisProperties) redis基本配置；  
########################################################  
# database name  
spring.redis.database=0  
# server host1  
spring.redis.host=127.0.0.1   
# server password  
#spring.redis.password=  
#connection port  
spring.redis.port=6379  
# pool settings ...  
spring.redis.pool.max-idle=8  
spring.redis.pool.min-idle=0  
spring.redis.pool.max-active=8  
spring.redis.pool.max-wait=-1  
# name of Redis server  
#spring.redis.sentinel.master=  
# comma-separated list of host:port pairs  
#spring.redis.sentinel.nodes=  
  
### Java Persistence Api 自动进行建表  
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

# 2、编辑redis缓存配置类

## 解释：

### 1、其一就是CacheManager缓存管理器；

### 2、其二就是具体操作实现类；RedisTemplate

### 3、其三就是CacheManager工厂类（这个可以使用配置文件配置的进行注入，也可以通过编码的方式进行实现）；

### 4、其四就是缓存key生产策略（这里先使用Spring自带生成策略，在Redis客户端进行查看的话是系列化的key,对于我们肉眼来说就是感觉是乱码了）。

## 1、RedisCacheConfig 开始启用缓存 @EnableCaching

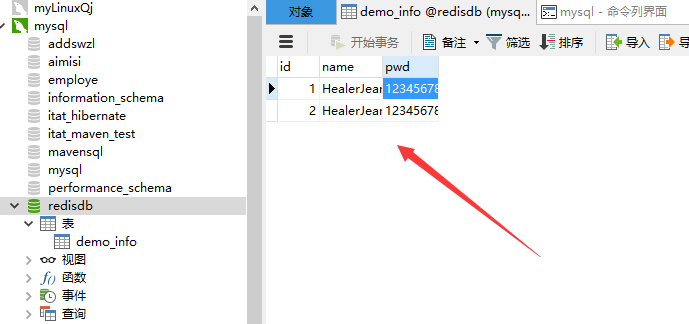
Configuration  
@EnableCaching //启用缓存，这个注解很重要；  
public class RedisCacheConfig {  
  
 /\*\*  
 \* 缓存管理器.  
 \* @param redisTemplate  
 \* @return  
 \*/  
 @Bean  
 public CacheManager cacheManager(RedisTemplate<?,?> redisTemplate) {  
 CacheManager cacheManager = new RedisCacheManager(redisTemplate);  
 return cacheManager;  
 }  
  
  
 /\*\*  
 \* RedisTemplate缓存操作类,类似于jdbcTemplate的一个类;  
 \* @param factory : 通过Spring进行注入，参数在application.properties进行配置；  
 \*/  
 @Bean  
 public RedisTemplate<String, String> redisTemplate(RedisConnectionFactory factory) {  
 RedisTemplate<String, String> redisTemplate = new RedisTemplate<String, String>();  
 redisTemplate.setConnectionFactory(factory);  
  
 return redisTemplate;  
 }  
}

# 3、编辑control类

## 1、测试添加缓存、删除缓存

@Controller  
public class DemoInfoController {  
   
 @Autowired  
 DemoInfoService demoInfoService;  
  
 /\*\*  
 \* 测试添加缓存  
 \* @return  
 \*/  
 @RequestMapping("/test")  
 public @ResponseBody String test(){  
 DemoInfo loaded = demoInfoService.findById(1);  
 System.***out***.println("loaded="+loaded);  
 DemoInfo cached = demoInfoService.findById(1);  
 System.***out***.println("cached="+cached);  
 loaded = demoInfoService.findById(2);  
 System.***out***.println("loaded2="+loaded);  
 return "ok";  
 }  
  
 /\*\*  
 \* 测试删除缓存  
 \* @param id  
 \* @return  
 \*/  
 @RequestMapping("/delete")  
 public @ResponseBody String delete(long id){  
 demoInfoService.deleteFromCache(id);  
 return "ok";  
 }   
   
  
}

# 4、在数据库中的表导入两条数据



# 5、编辑repository类

public interface DemoInfoRepository extends CrudRepository<DemoInfo,Long> {  
  
}

# 6、编辑service接口和实现类，开始利用缓存注解

## 1、接口

public interface DemoInfoService {  
   
 public DemoInfo findById(long id);  
   
 public void deleteFromCache(long id);  
  
  
}

## 2、重点，impl service value为缓存位置名称，不能为空的

### 1、@Cacheable(value="")

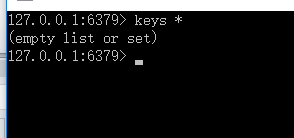
### 2、@ CacheEvict(value="")

@Service  
public class DemoInfoServiceImpl implements DemoInfoService {  
   
 @Resource  
 private DemoInfoRepository demoInfoRepository;  
   
 @Cacheable(value="HealerJean") //缓存,这里没有指定key.  
 @Override  
 public DemoInfo findById(long id) {  
 System.***err***.println("DemoInfoServiceImpl.findById()=========从数据库中进行获取的....id="+id);  
 return demoInfoRepository.findOne(id);  
 }  
   
 @CacheEvict(value=" Zhangyujin")  
 @Override  
 public void deleteFromCache(long id) {

//因为是模拟，所以我没有删除数据库中的内容，只删除了缓存  
 System.***out***.println("DemoInfoServiceImpl.delete().从缓存中删除."+id);  
 }  
   
}

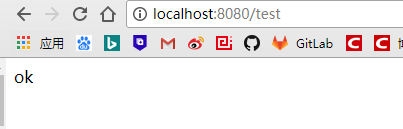
### 3、二者必须给定value值，且相同，这里的value值，一般默认是实体对象的名字，但我这里随意都写写的，都是HealerJean。

#### 1、观察redis，可以看到没有缓存，开始测试

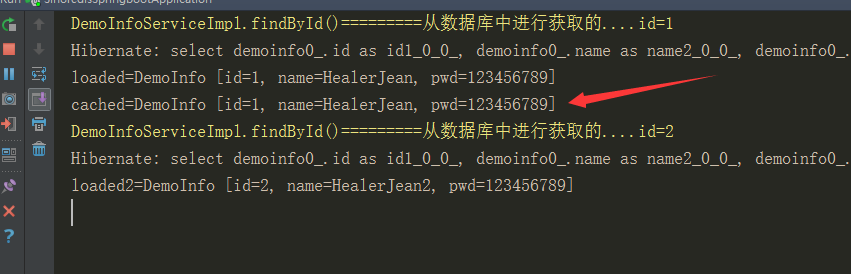


#### 2、测试查看，进入缓存，浏览器访问<http://localhost:8080/test>

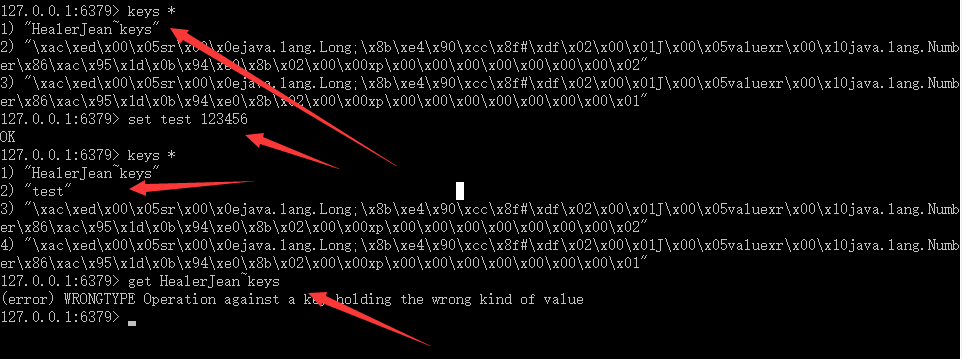
##### 1、浏览起得到ok



##### 2、控制台 （有时候，看到的控制台顺序不一致，不过不用担心，只不过有点像异步而已）可以看到当controler第二次读取id为1时走的缓存

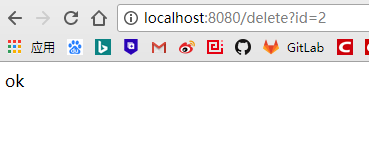
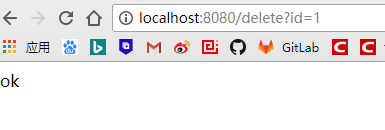


##### 3、观察redis缓存中，我这里为了向看看这个HealerJean`key是啥意思，所以set了一个变量看了看

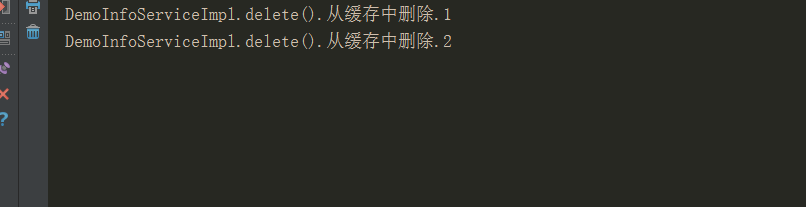


#### 3、测试删除，分别访问<http://localhost:8080/delete?id=1> <http://localhost:8080/delete?id=2>

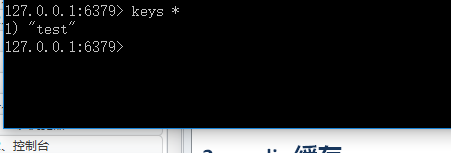
##### 1、浏览器



##### 2、控制台



##### 3、redis缓存，确实已经删除了，而且删除非常干净



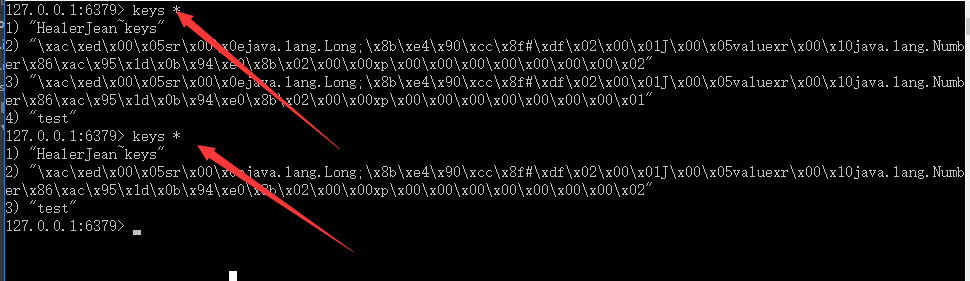
## 4、当value值不一样的时候，@cacheable 为HealerJean @ CacheEvict为zhangyujin（因为这种方式完全是依靠id来进行存储和查询的，所以才没有报错。具体看8标题1）

### 1、测试查看，和上面的其实是一样的

### 2、测试删除，

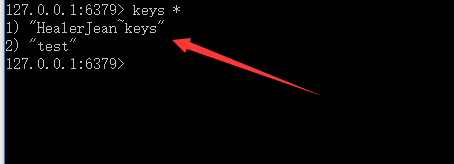
#### 1、浏览器访问删除id为1 <http://localhost:8080/delete?id=1>

##### 1、观察redis，会发现少了一条



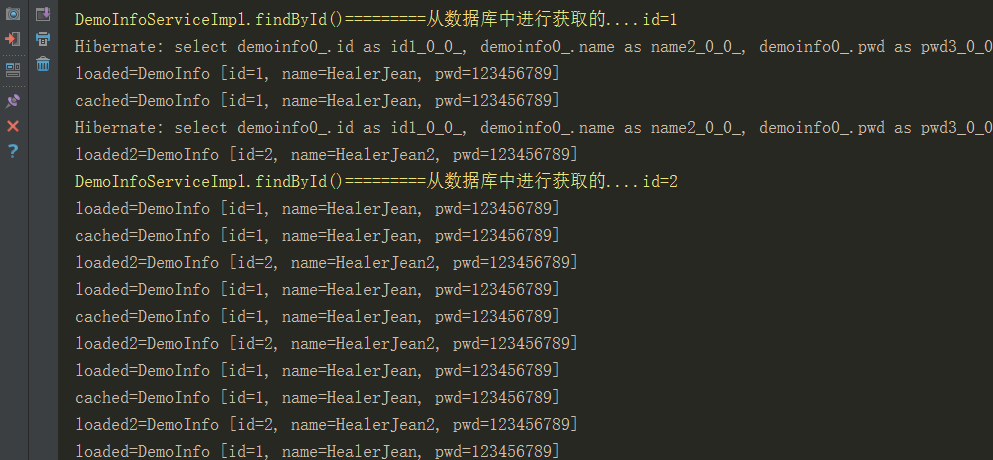
#### 2、浏览器访问删除id为2 <http://localhost:8080/delete?id=2>

##### 1、观察redis ，会发现又少了一条，但是没有删除干净



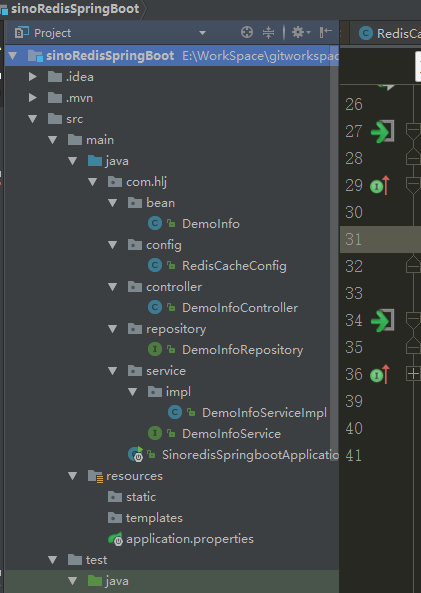
## 5、二者给定正常值demoInfo，也就是实体的名字，开始测试

### 1、多次刷新浏览器 <http://localhost:8080/test>，会发现除了2次hql语句其他都是从缓存中读取的



### 2、不测试了，肯定是删除成功，记得删除redis缓存哦

## 6、代码位置



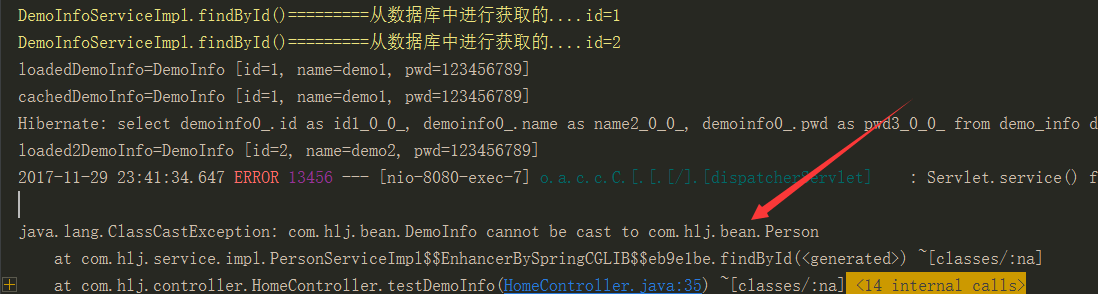
# 7、此时如果再添加一个person实体，如果两个获取对象的service方法都在一个control中，这种方式就会出现问题，因为，这里的缓存完全是依靠id来的。

## 1、测试control中添加person，@cacheable，@ CacheEvict都分别为DemoInfo和person

/\*\*  
 \* 测试添加缓存  
 \* @return  
 \*/  
@RequestMapping("/DemoInfo")  
public @ResponseBody String testDemoInfo(){  
 DemoInfo loadedDemoInfo = demoInfoService.findById(1);  
 System.***out***.println("loadedDemoInfo="+loadedDemoInfo);  
 DemoInfo cachedDemoInfo = demoInfoService.findById(1);  
 System.***out***.println("cachedDemoInfo="+cachedDemoInfo);  
 loadedDemoInfo = demoInfoService.findById(2);  
 System.***out***.println("loaded2DemoInfo="+loadedDemoInfo);  
  
 Person person = personService.findById(1);  
 System.***out***.println("loadedPerson="+person);  
  
 return "DemoInfo\_ok";  
  
  
}

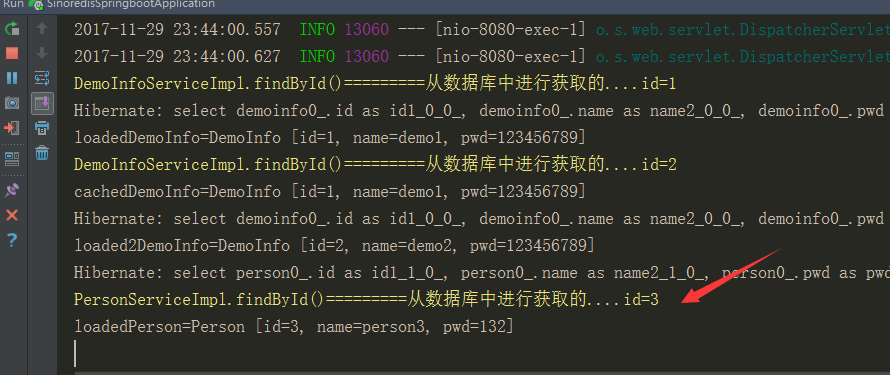
## 2、浏览器访问，控制台报错，提示不能够转化对象，很明显，这里person我给的id为1，系统就默认从缓存id为1中读取了

##### <http://localhost:8080/DemoInfo>



## 3、为了验证2中的猜想，这里讲person id设置为3 不和demoinfo重复，出现了查询，说明2中猜想正确。

##### <http://localhost:8080/DemoInfo>



## 4、代码位置

# 自定义的缓存策略

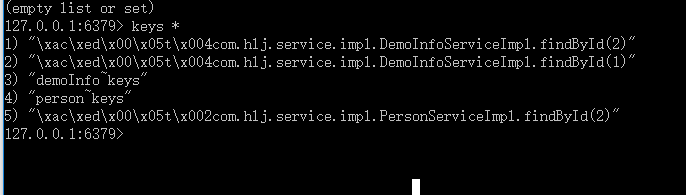
## 1、缓存配置类RedisCacheConfig继承CachingConfigurerSupport，开始重写 keyGenerato

/\*\*  
 \* 自定义key.  
 \* 此方法将会根据类名+方法名+所有参数 的值生成唯一的一个key,  
 \* 即使@Cacheable中的value属性一样，key也会不一样。  
 \*/  
@Override  
public KeyGenerator keyGenerator() {  
 System.***out***.println("RedisCacheConfig.keyGenerator()");  
 return new KeyGenerator() {  
 @Override  
 public Object generate(Object o, Method method, Object... objects) {  
 // This will generate a unique key of the class name, the method name  
 //and all method parameters appended.  
 StringBuilder sb = new StringBuilder();  
 sb.append(o.getClass().getName());  
 sb.append("."+method.getName());  
 for (Object obj : objects) {  
 sb.append("("+obj.toString()+")");  
 }  
 System.***out***.println("keyGenerator=" + sb.toString());  
 return sb.toString();  
 }  
 };  
}

## 2、缓存操作类，实现key的序列化方式，否则即使有了key的生成策略，但是没有这种序列化方式就是还是向上面的一样，是乱码

@Bean  
public RedisTemplate<String, String> redisTemplate(RedisConnectionFactory factory) {  
 RedisTemplate<String, String> redisTemplate = new RedisTemplate<String, String>();  
 redisTemplate.setConnectionFactory(factory);  
  
 //key序列化方式;（不然会出现乱码;）,但是如果方法上有Long等非String类型的话，会报类型转换错误；  
 //所以在没有自己定义key生成策略的时候，可以不配置,我就在使用spring自带的的key的生成策略时候。没有配置  
 RedisSerializer<String> redisSerializer = new StringRedisSerializer();//Long类型不可以会出现异常信息;  
 redisTemplate.setKeySerializer(redisSerializer);  
 redisTemplate.setHashKeySerializer(redisSerializer);  
  
 return redisTemplate;  
}

### 1、如果没有上面的序列化方式，在我测试完本例之后，注释掉上面的，看到的。



## 3、开始测试查看，缓存

### 1、control 中添加3个访问的地址,2个查看，一个删除

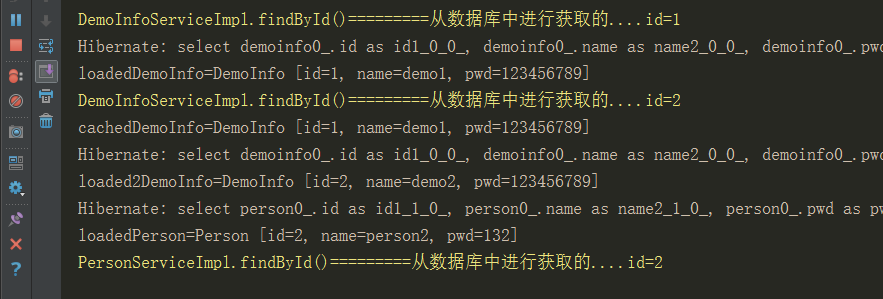
@Controller  
public class HomeController {  
   
 @Autowired  
 DemoInfoService demoInfoService;  
  
 @Autowired  
 PersonService personService;  
 /\*\*  
 \* 测试添加缓存  
 \* @return  
 \*/  
 @RequestMapping("/DemoInfo")  
 public @ResponseBody String testDemoInfo(){  
 DemoInfo loadedDemoInfo = demoInfoService.findById(1);  
 System.***out***.println("loadedDemoInfo="+loadedDemoInfo);  
 DemoInfo cachedDemoInfo = demoInfoService.findById(1);  
 System.***out***.println("cachedDemoInfo="+cachedDemoInfo);  
 loadedDemoInfo = demoInfoService.findById(2);  
 System.***out***.println("loaded2DemoInfo="+loadedDemoInfo);  
  
 Person person = personService.findById(1);  
 System.***out***.println("loadedPerson="+person);  
  
 return "DemoInfo\_ok";  
  
  
 }  
 @RequestMapping("/Person")  
 public @ResponseBody String testPerson() {  
 //person 开始  
 Person person = personService.findById(1);  
 System.***out***.println("loadedPerson="+person);  
 Person cachedPerson = personService.findById(1);  
 System.***out***.println("cachedPerson="+cachedPerson);  
 person = personService.findById(2);  
 System.***out***.println("loaded2Person="+person);  
 return "Person\_ok";  
 }  
 /\*\*  
 \* 测试删除缓存  
 \* @param id  
 \* @return  
 \*/  
 @RequestMapping("/delete")  
 public @ResponseBody String delete(long id){  
 demoInfoService.deleteFromCache(id);  
 return "ok";  
 }   
   
  
}

### 2、@cacheable，@ CacheEvict都分别为DemoInfo和person

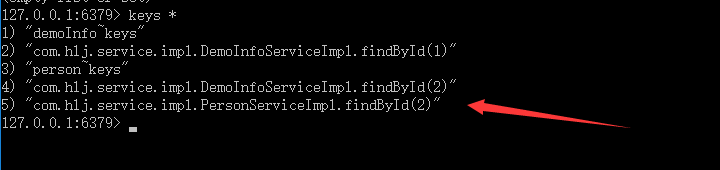
### 3、浏览器访问 <http://localhost:8080/DemoInfo> 在自定义key的时候，打上断点，可以观察到，最后的key是一个类+方法名+变量的名字



### 4、运行，成功，id同时为2的情况下，没有报错哦。哈哈，因为这里使用的key的生成策略是自己的。



### 5、redis缓存中为



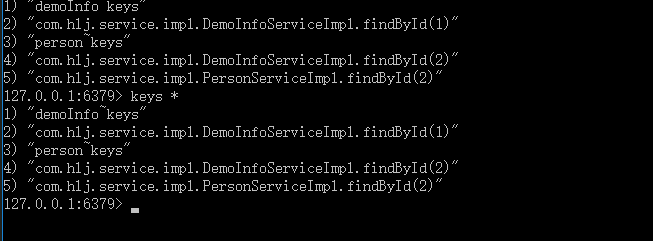
## 4、紧接着测试删除（发现不能删除。因为有两个实体了。具体不清楚哦）

### 1、service中

@CacheEvict(value="demoInfo")  
@Override  
public void deleteFromCache(long id) {  
 System.***out***.println("DemoInfoServiceImpl.delete().从缓存中删除."+id);  
}

### 2、浏览器访问，会发现什么都没有删除，哈哈，这里告诉你原因，因为删除必须知道 key的值，那么这里的生成策略key为 "com.hlj.service.impl.DemoInfoServiceImpl.findById(1) ，你没有给这个key值，怎么可能删除呢，哈哈。后面讲Cacheable属性的时候会知道的，那么你想问为什么spring自带的生成策略可以删，因为spring的生成策略中id是最关键的，它只识别id。其他的都不看。所以才会有两个对象存在报错的情况。

#### <http://localhost:8080/delete?id=1>



### 3、不信，再访问浏览器，查看缓存，没有通过hql语句查看数据库，全是从缓存中拿到的

#### <http://localhost:8080/DemoInfo>

