自定义缓存策略，以及删除缓存

## 1、接上文代码 (如果使用key=的话，可以删掉自定义的缓冲策略，但是序列号方式必须有的)

帮助文档：<http://www.vxzsk.com/749.html>

## Cacheable 支持如下几个参数：

### value：缓存位置名称，不能为空，如果使用EHCache，就是ehcache.xml中声明的cache的name

### key：缓存的key，默认为空，既表示使用方法的参数类型及参数值作为key，支持SpEL。

#### 1、#加变量名字，就可以表示变量了， 比如 #id

#### 2、使用对象中的变量 #user.id 或者#user.getId() 都可以

### condition：触发条件，只有满足条件的情况才会加入缓存，默认为空，既表示全部都加入缓存，支持SpEL

#### 以下有一个例子：

|  |  |
| --- | --- |
| //将缓存保存进andCache，并使用参数中的userId加上一个字符串(这里使用方法名称)作为缓存的key  @Cacheable(value="HealerJeanCache",key="#userId + 'findById'")  public SystemUser findById(String userId) {       SystemUser user = (SystemUser) dao.findById(SystemUser.class, userId);       return user ;    }    //将缓存保存进andCache，并当参数userId的长度小于32时才保存进缓存，默认使用参数值及类型作为缓存的key    @Cacheable(value=" HealerJeanCache",condition="#userId.length < 32")    public boolean isReserved(String userId) {      System.out.println("hello andCache"+userId);      return false;  } |  |

## @CacheEvict 支持如下几个参数：

### value：缓存位置名称，不能为空，同上

### key：缓存的key，默认为空，同上

### condition：触发条件，只有满足条件的情况才会清除缓存，默认为空，支持SpEL

### allEntries：true表示清除value中的全部缓存，默认为false

#### 以下是一个小例子：

|  |
| --- |
| //清除掉指定key的缓存    @CacheEvict(value="andCache",key="#user.userId + 'findById'")    public void modifyUserRole(SystemUser user) {             System.out.println("hello andCache delete"+user.getUserId());    }      //清除掉全部缓存    @CacheEvict(value="andCache",allEntries=true)   public void setReservedUsers() {      System.out.println("hello andCache deleteall");  } |

## @CachePut 注释，这个注释可以确保方法被执行，同时方法的返回值也被记录到缓存中，实现缓存与数据库的同步更新，理解为update语句。

# 2、实战 注意使用 单引号

## 1、编辑personServiceImpl ,实现一个添加和删除缓存的方法

//有省略的其他方法  
@Service  
public class PersonServiceImpl implements PersonService {  
   
 @Resource  
 private PersonRepository personRepository;  
//缓存位置名称为 HealerJeanCache key为 变量id + 字符串方法名

@Cacheable(value="HealerJeanCache",key="#id + 'findById'")  
 @Override  
 public Person findById(long id) {  
 System.***err***.println("PersonServiceImpl.findById()=========从数据库中进行获取的....id="+id);  
 return personRepository.findOne(id);  
 }  
  
  
 //缓存位置名称为 HealerJeanCache ，变量person中的id + 之前设置好的字符串方法名，也就是保证key值一样 'findById'  
 @CacheEvict(value="HealerJeanCache",key="#person.id + 'findById'")  
 @Override  
 public void deleteFromCacheForPerson(Person person) {  
 System.***out***.println("PersonServiceImpl.delete().从缓存中删除."+person.getId());  
 }  
  
}

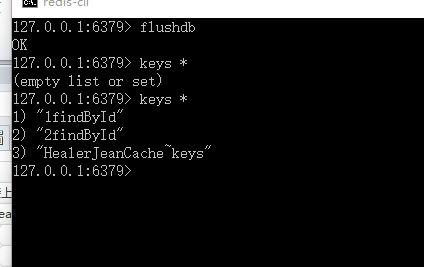
## 2、controler 开始准备添加和删除

//代码有省略哦  
@Controller  
public class HomeController {  
   
  
 @Autowired  
 PersonService personService;  
  
 @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("/deleteForPerson")  
 public @ResponseBody String deleteForPerson(long id){  
 Person person = new Person();  
 person.setId(id);  
 personService.deleteFromCacheForPerson(person);  
 return "ok";  
 }  
}

## 3、开始测试查看，缓存

### 1、浏览器中打开 <http://localhost:8080/Person>

### 2、观察redis缓存 ，可以看到有两个id的key值，那么说明这里覆盖了我们之前缓存配置类RedisCacheConfig自定的key的生成策略（public KeyGenerator keyGenerator() ）



## 4、测试删除，浏览器打开 <http://localhost:8080/deleteForPerson?id=1> 准备删除id为1的对象，删除成功

