文档修订历史

|  |  |  |  |  |
| --- | --- | --- | --- | --- |
| **版本** | **作者** | **工作描述** | **修订历史** | **修改日期** |
|  |  |  |  |  |
|  |  |  |  |  |
|  |  |  |  |  |
|  |  |  |  |  |
|  |  |  |  |  |

目录

[文档修订历史 1](#_Toc522698973)

[【文档标题名称】 4](#_Toc522698974)

[1 范围 **错误!未定义书签。**](#_Toc522698975)

[1.1 内容范围 **错误!未定义书签。**](#_Toc522698976)

[1.2 适用范围 **错误!未定义书签。**](#_Toc522698977)

[2 业务描述 **错误!未定义书签。**](#_Toc522698978)

[3 角色说明 **错误!未定义书签。**](#_Toc522698979)

[3.1 分销代理商 **错误!未定义书签。**](#_Toc522698980)

[4 业务需求 **错误!未定义书签。**](#_Toc522698981)

[4.1 注册登录 **错误!未定义书签。**](#_Toc522698982)

[4.2 查看及修改账户信息 **错误!未定义书签。**](#_Toc522698983)

[4.3 创建客户 **错误!未定义书签。**](#_Toc522698984)

[4.4 商品信息 **错误!未定义书签。**](#_Toc522698985)

[4.5 修改客户商品价格 **错误!未定义书签。**](#_Toc522698986)

[4.6 推送价格 **错误!未定义书签。**](#_Toc522698987)

[4.7 查看客户信息 **错误!未定义书签。**](#_Toc522698988)

[4.8 代理商利润计算公式 **错误!未定义书签。**](#_Toc522698989)

[4.9 代理商利润及打款 **错误!未定义书签。**](#_Toc522698990)

[4.10 操作界面语言选项 **错误!未定义书签。**](#_Toc522698991)

[5 外围系统关系 **错误!未定义书签。**](#_Toc522698992)

[6 其它需求 **错误!未定义书签。**](#_Toc522698993)

[6.1 部署需求 **错误!未定义书签。**](#_Toc522698994)

[6.2 环境要求 **错误!未定义书签。**](#_Toc522698995)

[6.3 运维要求 **错误!未定义书签。**](#_Toc522698996)

CBSS-Redis学习摘录

Redis的读写

# 框架

## Ctrl+2

### Ctrl+3

#### Ctrl+4

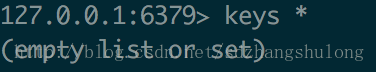
# 附录

## 附录一：springboot中关于缓存@Cacheable、@CacheEvict、@CachePut的用法理解浅析

转载：<https://blog.csdn.net/sdzhangshulong/article/details/79076372>

我们要实现的是,当查询时,先从缓存中查找数据,当数据存在时,不查数据库,当数据不存在时,查找数据库,并将值存到redis中

首先,缓存中是没有数据的,redis查询keys \* ,会发现redis中无数据

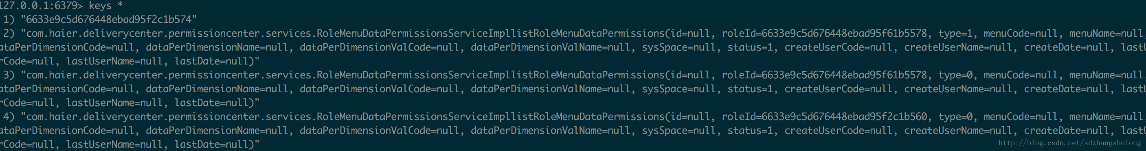


此时,调用查询方法(service层)

|  |
| --- |
| /\*\*  \* 查询所有角色菜单数据权限信息  \* @return  \*/  @Override  @Cacheable(key = "#p0.roleId")  public List<RoleMenuDataPermissions> list(RoleMenuDataPermissions roleMenuDataPermissions) {  return roleMenuDataPermissionsDao.list(roleMenuDataPermissions);  } |

会将查询到的结果存到redis缓存中,key值就是参数的roleId, key="#p0.roleId"是指方法的第一个参数的roleId属性为key,

此时,reids中会有数据如下图:



此时,要新增或者修改或者删除数据,那么,需要将redis中数据删除,当再次调用查询方法时,会发现redis中没有数据,需要从数据库中查找并存到缓存redis中,那么新增或者修改或者删除方法如下这种写法:

|  |
| --- |
| /\*\*  \* 插入  \* @param roleMenuDataPermissions  \* @return  \*/  @Override  @Transactional  @CacheEvict(key="#p0.roleId")  public int insert(RoleMenuDataPermissions roleMenuDataPermissions) {  return roleMenuDataPermissionsDao.insert(roleMenuDataPermissions);  } |

|  |
| --- |
| /\*\*  \* 根据主键更新  \* @param roleMenuDataPermissions  \* @return  \*/  @Override  @Transactional  @CacheEvict(key="#p0.roleId")  public int updateByPrimaryKey(RoleMenuDataPermissions roleMenuDataPermissions) {  return roleMenuDataPermissionsDao.updateByPrimaryKey(roleMenuDataPermissions);  } |

|  |
| --- |
| /\*\*  \* 根据主键删除（逻辑删）  \* @param roleMenuDataPermissions  \* @return  \*/  @Override  @Transactional  @CacheEvict(key="#p0.roleId")  public int deleteByPrimaryKey(RoleMenuDataPermissions roleMenuDataPermissions) {  return roleMenuDataPermissionsDao.deleteByPrimaryKey(roleMenuDataPermissions);  } |

@CacheEvict(key="#p0.roleId") 会根据方法第一个参数的roleId为key,从redis中删除数据,这样,当再次调用查询方法时,redis中根据相同的roleId是查不到数据的,需要从数据库中查找,@Cacheable会自动将查找到的redis中没有的数据存到redis中。

以上适用于批量查询,当还有查询某一条数据的需求是,修改方法可以增加一个@CachePut方法

小结:@Cacheable是想缓存中添加数据,@CacheEvict是删除缓存中数据,@CachePut是更新缓存中数据

@Cacheable：主要用作查询，如果根据key从redis中没查到就去mysql查并且把返回结果插入redis，下次直接从redis取；

@CachePut：主要用作更新，如果根据key从redis中没查到就去mysql查并且把返回结果插入redis，如果从redis查到那么也去mysql查一遍，所以不管redis中有没有都去查mysql，所以主要用作更新

@CacheEvict：主要用作删除，根据key从redis中删除数据

|  |
| --- |
| @GetMapping("/deleteById/{id}")  @CacheEvict(value = "zh",key = "'user\_' + #id")  public void deleteById(@PathVariable Integer id){  System.out.println("UserController.deleteById().从数据库中删除.");  userService.deleteById(id);  }  @PostMapping("/save")  @CachePut(value = "zh",key = "'user\_' + #user.id")  public User save(User user){  return userService.save(user);  }  @PostMapping("/updateNameById")  public void updateNameById(User user){  userService.updateNameById(user);  } |

“value”相当于命名空间，表示该数据放在哪个命名空间里。

## 附录二：SpringBoot使用Redis数据库

转载：https://www.cnblogs.com/gdpuzxs/p/7224944.html

### (1)、pom.xml文件引入jar包，如下：

|  |
| --- |
| <dependency>  <groupId>org.springframework.boot</groupId>  <artifactId>spring-boot-starter-data-redis</artifactId>  </dependency> |

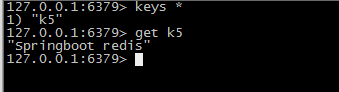
### (2)、application.properties文件中配置redis连接信息，如下：

|  |
| --- |
| # Redis数据库索引（默认为0）  spring.redis.database=0  # Redis服务器地址  spring.redis.host=172.31.19.222  # Redis服务器连接端口  spring.redis.port=6379  # Redis服务器连接密码（默认为空）  spring.redis.password=  # 连接池最大连接数（使用负值表示没有限制）  spring.redis.pool.max-active=8  # 连接池最大阻塞等待时间（使用负值表示没有限制）  spring.redis.pool.max-wait=-1  # 连接池中的最大空闲连接  spring.redis.pool.max-idle=8  # 连接池中的最小空闲连接  spring.redis.pool.min-idle=0  # 连接超时时间（毫秒）  spring.redis.timeout=0 |

### (3)、测试redis缓存

|  |
| --- |
| package springboot.web;  import org.springframework.beans.factory.annotation.Autowired;  import org.springframework.data.redis.core.StringRedisTemplate;  import org.springframework.web.bind.annotation.RequestMapping;  import org.springframework.web.bind.annotation.RestController;  @RestController  public class HelloController {    @Autowired  private StringRedisTemplate stringRedisTemplate;  @RequestMapping("/redisHandler")  public String redisHandler(){  stringRedisTemplate.opsForValue().set("k5", "Springboot redis");  return stringRedisTemplate.opsForValue().get("k5");  }  } |

### (4)、启动项目，调用reidsHandler方法，查询redis服务器信息，如下：



## 附录三：SpringBoot使用Redis缓存

### (1)、pom.xml文件引入jar包，如下：

|  |
| --- |
| <dependency>  <groupId>org.springframework.boot</groupId>  <artifactId>spring-boot-starter-data-redis</artifactId>  </dependency> |

### (2)、application.properties文件中配置redis连接信息，如下：

|  |
| --- |
| # Redis数据库索引（默认为0）  spring.redis.database=0  # Redis服务器地址  spring.redis.host=172.31.19.222  # Redis服务器连接端口  spring.redis.port=6379  # Redis服务器连接密码（默认为空）  spring.redis.password=  # 连接池最大连接数（使用负值表示没有限制）  spring.redis.pool.max-active=8  # 连接池最大阻塞等待时间（使用负值表示没有限制）  spring.redis.pool.max-wait=-1  # 连接池中的最大空闲连接  spring.redis.pool.max-idle=8  # 连接池中的最小空闲连接  spring.redis.pool.min-idle=0  # 连接超时时间（毫秒）  spring.redis.timeout=0 |

### (3)、修改项目启动类，增加注解@EnableCaching，开启缓存功能，如下：

|  |
| --- |
| package springboot;  import org.springframework.boot.SpringApplication;  import org.springframework.boot.autoconfigure.SpringBootApplication;  import org.springframework.cache.annotation.EnableCaching;  import org.springframework.scheduling.annotation.EnableScheduling;  @SpringBootApplication  @EnableScheduling  @EnableCaching  public class SpringbootApplication{  public static void main(String[] args) {  SpringApplication.run(SpringbootApplication.class, args);  }  } |

### (4)、新建Redis缓存配置类RedisConfig，如下：

|  |
| --- |
| package springboot.config;  import org.springframework.beans.factory.annotation.Value;  import org.springframework.cache.CacheManager;  import org.springframework.cache.annotation.CachingConfigurerSupport;  import org.springframework.cache.annotation.EnableCaching;  import org.springframework.context.annotation.Bean;  import org.springframework.context.annotation.Configuration;  import org.springframework.data.redis.cache.RedisCacheManager;  import org.springframework.data.redis.connection.RedisConnectionFactory;  import org.springframework.data.redis.core.RedisTemplate;  import org.springframework.data.redis.core.StringRedisTemplate;  import org.springframework.data.redis.serializer.Jackson2JsonRedisSerializer;  import com.fasterxml.jackson.annotation.JsonAutoDetect;  import com.fasterxml.jackson.annotation.PropertyAccessor;  import com.fasterxml.jackson.databind.ObjectMapper;  /\*\*  \* Redis缓存配置类  \* @author szekinwin  \*  \*/  @Configuration  @EnableCaching  public class RedisConfig extends CachingConfigurerSupport{  @Value("${spring.redis.host}")  private String host;  @Value("${spring.redis.port}")  private int port;  @Value("${spring.redis.timeout}")  private int timeout;    //自定义缓存key生成策略  // @Bean  // public KeyGenerator keyGenerator() {  // return new KeyGenerator(){  // @Override  // public Object generate(Object target, java.lang.reflect.Method method, Object... params) {  // StringBuffer sb = new StringBuffer();  // sb.append(target.getClass().getName());  // sb.append(method.getName());  // for(Object obj:params){  // sb.append(obj.toString());  // }  // return sb.toString();  // }  // };  // }  //缓存管理器  @Bean  public CacheManager cacheManager(@SuppressWarnings("rawtypes") RedisTemplate redisTemplate) {  RedisCacheManager cacheManager = new RedisCacheManager(redisTemplate);  //设置缓存过期时间  cacheManager.setDefaultExpiration(10000);  return cacheManager;  }  @Bean  public RedisTemplate<String, String> redisTemplate(RedisConnectionFactory factory){  StringRedisTemplate template = new StringRedisTemplate(factory);  setSerializer(template);//设置序列化工具  template.afterPropertiesSet();  return template;  }  private void setSerializer(StringRedisTemplate template){  @SuppressWarnings({ "rawtypes", "unchecked" })  Jackson2JsonRedisSerializer jackson2JsonRedisSerializer = new Jackson2JsonRedisSerializer(Object.class);  ObjectMapper om = new ObjectMapper();  om.setVisibility(PropertyAccessor.ALL, JsonAutoDetect.Visibility.ANY);  om.enableDefaultTyping(ObjectMapper.DefaultTyping.NON\_FINAL);  jackson2JsonRedisSerializer.setObjectMapper(om);  template.setValueSerializer(jackson2JsonRedisSerializer);  }  } |

### (5)、新建UserMapper，如下：

|  |
| --- |
| package springboot.dao;  import org.apache.ibatis.annotations.Delete;  import org.apache.ibatis.annotations.Insert;  import org.apache.ibatis.annotations.Mapper;  import org.apache.ibatis.annotations.Param;  import org.apache.ibatis.annotations.Select;  import org.apache.ibatis.annotations.Update;  import org.springframework.cache.annotation.CacheConfig;  import org.springframework.cache.annotation.CacheEvict;  import org.springframework.cache.annotation.CachePut;  import org.springframework.cache.annotation.Cacheable;  import springboot.domain.User;  @Mapper  @CacheConfig(cacheNames = "users")  public interface UserMapper {  @Insert("insert into user(name,age) values(#{name},#{age})")  int addUser(@Param("name")String name,@Param("age")String age);    @Select("select \* from user where id =#{id}")  @Cacheable(key ="#p0")  User findById(@Param("id") String id);    @CachePut(key = "#p0")  @Update("update user set name=#{name} where id=#{id}")  void updataById(@Param("id")String id,@Param("name")String name);    //如果指定为 true，则方法调用后将立即清空所有缓存  @CacheEvict(key ="#p0",allEntries=true)  @Delete("delete from user where id=#{id}")  void deleteById(@Param("id")String id);    } |

@Cacheable将查询结果缓存到redis中，（key="#p0"）指定传入的第一个参数作为redis的key。

@CachePut，指定key，将更新的结果同步到redis中

@CacheEvict，指定key，删除缓存数据，allEntries=true,方法调用后将立即清除缓存

### (6)、service层与controller层跟上一篇整合一样，启动redis服务器，redis服务器的安装与启动可以参考之前的博客，地址如下：

http://www.cnblogs.com/gdpuzxs/p/6623171.html

### (7)、配置log4j日志信息，如下：

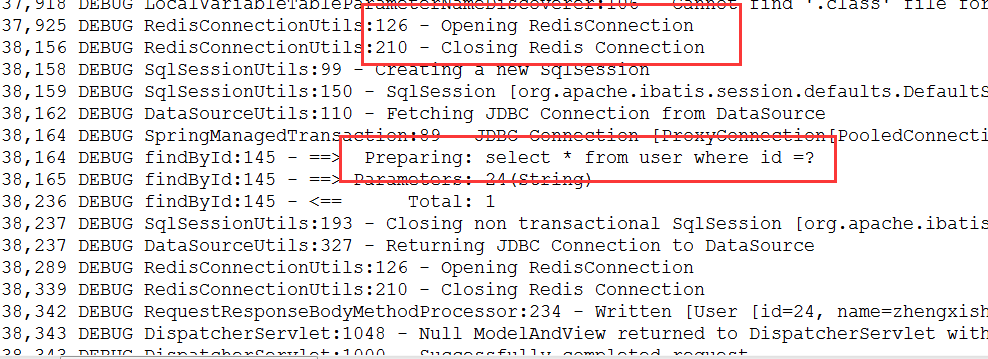
|  |
| --- |
| ## LOG4J配置  log4j.rootCategory=DEBUG,stdout  ## 控制台输出  log4j.appender.stdout=org.apache.log4j.ConsoleAppender  log4j.appender.stdout.layout=org.apache.log4j.PatternLayout  log4j.appender.stdout.layout.ConversionPattern=%d{yyyy-MM-dd HH:mm:ss,SSS} %5p %c{1}:%L - %m%n |

### (8)、验证redis缓存

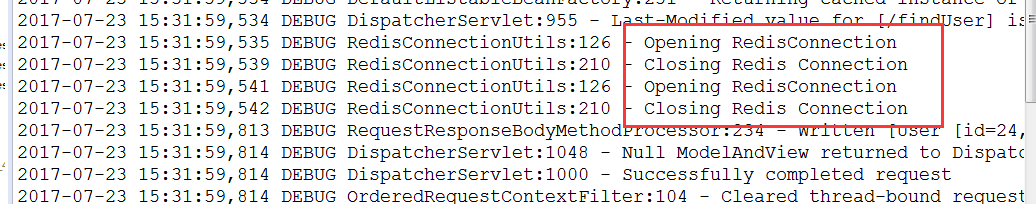
首先我们向user表总插入一条数据，数据库显示如下：

https://images2015.cnblogs.com/blog/961610/201707/961610-20170723152550049-259578829.png

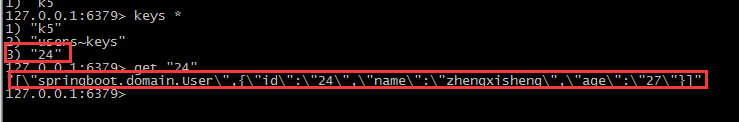
现在，我们查询一下user表中id=24的数据，观擦控制台输出的信息，如下：



通过控制台输出信息我们可以知道，这次执行了数据库查询，并开启了Redis缓存查询结果。接下来我们再次查询user表中id=24的数据，观察控制台，如下：



通过控制台输出信息我们可以知道，这次并没有执行数据库查询，而是从Redis缓存中查询，并返回查询结果。我们查看redis中的信息，如下：



方法finduser方法使用了注解@Cacheable(key="#p0")，即将id作为redis中的key值。当我们更新数据的时候，应该使用@CachePut(key="#p0")进行缓存数据的更新，否则将查询到脏数据。