**Redis使用规范**

**编写：高广金**

**评审：高广金**

**批准：高广金**

**日期：2015-05-17**

目录

[Redis使用规范 1](#_Toc419732489)

[1 引言 3](#_Toc419732490)

[1.1 编写目的 3](#_Toc419732491)

[1.2 编写原则 3](#_Toc419732492)

[1.3 读者对象 3](#_Toc419732493)

[1.4 文档贡献 3](#_Toc419732494)

[1.5 术语定义 3](#_Toc419732495)

[1.5.1 名词释义 3](#_Toc419732496)

[2 开发规则 5](#_Toc419732497)

[2.1 命名 5](#_Toc419732498)

[2.2 命名规则意义 6](#_Toc419732499)

[3 开发实例 9](#_Toc419732500)

[3.1 初始化从数据库将表数据缓存到redis 9](#_Toc419732501)

[3.2 测试调用 10](#_Toc419732502)

[3.3 简单封装的公共api 15](#_Toc419732503)

# 引言

## 编写目的

本文档制定使用Redis规范和使用要求，制定规范命名和流程化的使用规则，从而降低后期维护、扩容、管理的成本。

## 编写原则

根据内部项目管理规范则，从业务人员的角度对需求的阐述，以及对需求描述的细化程度。此文档将在开发过程随着需求的变更而更改和修订。

## 读者对象

项目经理、开发人员、业务人员、测试人员。

## 文档贡献

大宗交易开发团队。

## 术语定义

### 名词释义

| 术语 | 解释 |
| --- | --- |
| Redis | 可基于内存亦可持久化的日志型、Key-Value数据库。 |
| TableName | 实体类名称,例如bi\_menu表对应的实体类是BiMenu |
| PrimaryKeyValue | 某个表的主键值 |
| Column | 实体类里面的字段名称。例如bi\_menu表对应的实体类字段名称name |
| ColumnValue | 列的值 |
| index | 字符串index代表此key是用来记录索引的 |
| sort | 字符串sort代表此key是用来排序标记的 |

# 开发规则

## 命名

key命名类型:

1. 存放表里面所有字段信息。TableName: PrimaryKeyValue:Column
2. 存放表里面某一列的信息 TableName：Column：ColumnValue
3. 存放表里面某一行object信息。TableName: PrimaryKeyValue
4. 存放表里面的索引。TableName:index:Column
5. 存放表里面的字段排序。TableName:sort:Column

实例：某数据库按钮表：bi\_menu

|  |  |  |  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- | --- | --- | --- |
| bi\_menu (菜单按钮) | | | | | | | | |
| 是否主键 | 字段名 | 字段描述 | 数据类型 | 长度 | 可空 | 约束 | 缺省值 | 备注 |
| 是 | id | 主键 | VARCHAR(36) | 36 |  |  |  |  |
| 否 | name | 名称 | VARCHAR(200) | 200 | 是 |  | NULL |  |
| 否 | sequence | 序列 | INT(11) | 11 | 是 |  | NULL |  |
| 否 | description | 描述 | VARCHAR(200) | 200 | 是 |  | NULL |  |
| 否 | logo | 图标 | VARCHAR(200) | 200 | 是 |  | NULL |  |
| 否 | url | 访问地址 | VARCHAR(200) | 200 | 是 |  | NULL | 点击访问地址 |
| .. | … | …. | … | .. | .. |  | .. | …. |

类型1(value就是具体的值)

|  |  |
| --- | --- |
| Key | Vaule |
| BiMenu:1:id | 1 |
| BiMenu:1:name | 系统应用 |
| BiMenu:1:sequence | 1 |
| BiMenu:1:description | 这是系统应用的按钮 |
| BiMenu:1:logo | 图片地址 |
| BiMenu:1:url | http://localhost:8080/aa.do |
| ……….. |  |

类型2(value就是改行的主键值PrimaryKeyValue)

|  |  |
| --- | --- |
| Key | Vaule |
| BiMenu:name:系统应用 | 1 |
| BiMenu:url:http://www.baidu.com | 1 |
| ……….. |  |

类型3(value是改行数据的jason格式)

|  |  |
| --- | --- |
| Key | Vaule |
| BiMenu:1 | {  "id": "1",  "name": "系统应用",  " sequence ": "1",  " description ": "这是系统应用的按钮"  } |
| BiMenu:2 | …….. |
| ……….. |  |

类型4(value是改行数据的主键值PrimaryKeyValue)

|  |  |
| --- | --- |
| Key | Vaule |
| BiMenu:index:id | BiMenu:1 |
| BiMenu:index:id | BiMenu:2 |
| BiMenu:index:id | BiMenu:3 |

类型5(value是改行数据的主键值PrimaryKeyValue)

|  |  |
| --- | --- |
| Key | Vaule |
| BiMenu:sort:sequence | BiMenu:1 |
| BiMenu:sort:sequence | BiMenu:2 |
| BiMenu:sort:sequence | BiMenu:3 |

## 命名规则意义

1. 类型1

场景：比如我们登陆一个系统，首先是输入登录名和登陆密码,假设username唯一，后台有类似的sql.

|  |
| --- |
| select \* from tc\_user where user\_name=? and user\_pass=? |

Redis替代

|  |
| --- |
| 1、Sort<String> setKey=keys(“User:\*:userName”)//获取所有带User:\*:userName的key  2、验证用户名是否存在  3、若存在查询到指定userName的 id，继续第四步，若不存在，返回。  4、验证密码是否一致  String pass=get(“user:id:pass”);//得到缓存的pass  If(inputpass.equal(pass)//略 |

1. 类型2

场景：比如我们查询数据的时候，需要返回一个结果集，数据库sql是这样的。

Sql:

|  |
| --- |
| 1、select \* from tcnote where note\_id =?//知道id查询  2、select \* from tcnote where note\_name=’ redis学习’and author\_name=’ gao’//交  3、select \* from tcnote where note\_name in(‘gao’,’admin’);//并  4、select \* from tcnote where author\_name in(‘gao’,’admin’) and flag=0// 先并再交 |

Redis替代

|  |
| --- |
| 1、String json=get(“Note:111”);//可以将json字符串转换成实体类bean  2、Sort<String> keySort= sinter (“Note:noteName:redis学习”,”Note:authorName:gao”)。//获取到对应列的主键值集合。我们根据主键再调用第一步获取实体类  3、Sort<String> keySort=sunion(“Note: authorName:admin”,”Note:authorName:gao”)// 获取到对应列的主键值集合。我们根据主键再调用第一步获取实体类  4、组合查询我们可以细分多步。有多少个in(a,b,c,d).就细化成多少个 步骤三  ①sunion(“Note: authorName: gao’”,”Note: flag:0”)  ②sunion(“Note: authorName:admin”,”Note: flag:0”)  ③然后把两个查询结果得到的list 合并一下 |

1. 类型3

场景：我们常用的就是从数据库查询数据，然后得到结果放到List<Bean>里面。

Sql:

|  |
| --- |
| Session session = sessionFactory.getCurrentSession();  Query query = session.createQuery("from Note");  List<Note> noteList = query.list();//noteList就是我们想要的结果 |

Redis:

|  |
| --- |
| //单个这样，多个key就直接来个循环，查询到之后再add到noteList里面  List<Note> noteList=new ArrayList<Note>();  String json=get(“Note:111”);  Note note=JSON.parseObject(json, Note.class)  noteList.add(note); |

1. 类型四

场景：查找数据库里面某个表的所有主键值

Sql:

|  |
| --- |
| select note\_id from tcnote |

Redis:

|  |
| --- |
| 最快捷的：  Set<String> keySet=smembers(“Note:index:noteId”);//获取到 note表里面所有的主键  或者这种也可以：  Keys(“Note:\*:noteId”)//获取主键id |

1. 类型五

场景：我们常用sql的排序，分页

Sql:

|  |
| --- |
| select \* from tcnote order by noteId |

Redis:

|  |
| --- |
| Set<String> keySet= zrange (“Note:sort:noteId”,”0”,”-1”);//从小到大查询到主键的id  Set<String> keySet= zrevrange (“Note:sort:noteId”,”0”,”-1”);//从大到小查询到主键的id |

# 开发实例

## 初始化从数据库将表数据缓存到redis

完整git地址：

<https://github.com/ggj2010/TeamStudy/blob/user/gaoguangjin/main/src/main/java/com/team/gaoguangjin/%E7%BC%93%E5%AD%98/redis/util/NoteDao.java>

代码：

|  |
| --- |
| @Test  @Transactional  **public** **void** testStore() **throws** Exception {  // 事物的控制交给当前线程，如果当前方法执行完毕就自动提交事物关闭session  **final** Session session = sessionFactory.getCurrentSession();  Query query = session.createQuery("from Note");  List<Note> noteList = query.list();  System.*out*.println("总大小：" + noteList.size());  // 截取留50个  noteList = noteList.subList(0, 50);  *jedis*.flushDB();// 清空测试  **long** time = System.*currentTimeMillis*();    insertToredis(noteList);  **long** end = System.*currentTimeMillis*();  *log*.info("耗时：" + (end - time));  }    **private** **void** insertToredis(List<Note> noteList) **throws** Exception {  Field[] fieldList = Thread.*currentThread*().getContextClassLoader().loadClass(Note.**class**.getName()).getDeclaredFields();  **for** (Note note : noteList) {  note.setContents(note.ClobToString(note.getContent()));  note.setContent(**null**);  **for** (Field field : fieldList) {  field.setAccessible(**true**);  String fieldName = field.getName();  Object fieldValue = field.get(note);  **if** (!"serialVersionUID".equals(fieldName) && !"log".equals(fieldName) && **null** != fieldValue) {  // 1、类型一k/v  setTable("Note", "" + note.getNoteId(), fieldName, fieldValue.toString());  // 对有注解的进行sadd kv存储  **if** (field.isAnnotationPresent(NeedSelect.**class**)) {  // 2、类型二 k/v  saddColumn("Note", fieldName, fieldValue.toString(), note.getNoteId() + "");  }  }  }  Object object = JSON.*toJSON*(note);  // 3、 存放映射bean key-jsonValue  setJSON("Note", "" + note.getNoteId(), JSON.*toJSON*(note).toString());  // 4、id值放到set里面，id加索引  zaddIndex("Note", "noteId", note.getNoteId() + "");    // 5、按照id大小排序  zaddSort("Note", "noteId", note.getNoteId() + "", note.getNoteId() + "");  }  } |

## 测试调用

完整git地址：

<https://github.com/ggj2010/TeamStudy/blob/user/gaoguangjin/main/src/main/java/com/team/gaoguangjin/%E7%BC%93%E5%AD%98/redis/util/NoteDao.java>

|  |
| --- |
| **public** **static** **void** main(String[] args) {  **try** {  NoteDao noteDao = **new** NoteDao();  // noteDao.select(noteDao);// 查询  // noteDao.insert();// 插入  noteDao.update(noteDao);// 更新  // noteDao.delete( noteDao);  // 异步更新到数据库  *dataBaseUpdate*(noteDao);  } **catch** (Exception e) {  *log*.error("redis查询错误！" + e.getLocalizedMessage());  }  }    **private** **static** **void** dataBaseUpdate(NoteDao noteDao) {  **try** {  List<Object> object = noteDao.updateToDataBase();  **for** (Object object2 : object) {  *log*.info(object2.toString());  String className = object2.getClass().getName();  Field[] fieldList = Thread.*currentThread*().getContextClassLoader().loadClass(className).getDeclaredFields();  **for** (Field field : fieldList) {  field.setAccessible(**true**);  **if** (field.getName().equals("flag")) {  **if** (field.get(object2).toString().equals("1")) {  // hibernate对数据库进行删除操作  *log*.info("数据库删除动作");  } **else** {  // 比如用hibernate sessionFactory.getCurrentSession().save(object2)。就直接新增和更新了  *log*.info("数据库更新动作");  }  // 更新成功之后删除set k/v里面的值  // 目前没有写具体判断细节。  *jedis*.srem(className, JSON.*toJSON*(object2).toString());  }  }  }  } **catch** (Exception e) {  e.printStackTrace();  }  }    **private** **void** insert() **throws** Exception {  List<Note> noteList = **new** ArrayList<Note>();  Note note = **new** Note();  note.setNoteId(20151314);  note.setAuthorName("高广金");  note.setFromUrl("www.baidu.com");  noteList.add(note);  insertToredis(noteList);  *log*.info("插入id为 20151314 成功！Note:20151314");  }    /\*\*  \* **@Description**:select \* from table where url=? and ...  \* **@param** noteDao  \* **@return**:void  \*/  **private** **void** select(NoteDao noteDao) {  String fromUrl = "http://www.tuicool.com/articles/zu6Nrm";  String authorName1 = "admin";  String authorName2 = "gao";  String flag = "0";  // 查询根据id  // selectById(noteDao);  // /\* 查询方式一 根据set/get \*/  // selectTypeOne(noteDao, fromUrl);  // /\* 查询方式二 \* 根据 sadd/smembers \*/  // selectTypeTwo(noteDao, fromUrl);  /\* 查询方式三 and \*/  selectTypeThree(noteDao, authorName1, fromUrl);  /\* 查询方式四 or \*/  selectTypeFour(noteDao, authorName1, authorName2);  /\* 查询方式五 and和or 同时存在 \*/  selectTypeFour(noteDao, authorName1, authorName2, flag);    }    /\*\*  \* **@Description**: 查询并交集合 select \* from tableName where authorName in("gao","admin") and flag=0  \* **@param** noteDao  \* **@param** authorName1  \* **@param** authorName2  \* **@param** fromUrl  \* **@return**:void  \*/  **private** **void** selectTypeFour(NoteDao noteDao, String authorName1, String authorName2, String flag) {  // 并交等于。给自己的交在加到一起  *log*.info("=================并交================");  **long** time = System.*currentTimeMillis*();  Set<String> sortKey1 = *jedis*.sinter("Note:authorName:" + authorName1, "Note:flag:" + flag);  Set<String> sortKey2 = *jedis*.sinter("Note:authorName:" + authorName2, "Note:flag:" + flag);  *display*(sortKey1, noteDao);  *display*(sortKey2, noteDao);  **long** endTime = System.*currentTimeMillis*();  *log*.info("sadd/smembers sinter两次查询交集合 方式执行select \* from tableName where authorName in(?,?) and flag=0 耗时：" + (endTime - time) + "ms");  }    /\*\*  \* **@Description**:查询并集合 sunion select\* from tableName where authorName in("gao","admin")  \* **@param** noteDao  \* **@param** url  \* **@param** authorName1  \* **@param** authorName2  \* **@return**:void  \*/  **private** **void** selectTypeFour(NoteDao noteDao, String authorName1, String authorName2) {  *log*.info("=================================");  **long** time = System.*currentTimeMillis*();  Set<String> sortKey = *jedis*.sunion("Note:authorName:" + authorName1, "Note:authorName:" + authorName2);  *display*(sortKey, noteDao);  **long** endTime = System.*currentTimeMillis*();  *log*.info("sadd/smembers sunion查询并结合 select\* from tableName where authorName in(gao,admin) 耗时：" + (endTime - time) + "ms");  }    /\*\*  \* **@Description**: 查询交集合 sinter select\* from tableName where authorName=? and url=?  \* **@param** noteDao  \* **@param** url  \* **@param** authorName admin  \* **@param** url  \* **@return**:void  \*/  **private** **void** selectTypeThree(NoteDao noteDao, String authorName1, String fromUrl) {  *log*.info("=================================");  **long** time = System.*currentTimeMillis*();  Set<String> sortKey = *jedis*.sinter("Note:authorName:" + authorName1, "Note:fromUrl:" + fromUrl);  *display*(sortKey, noteDao);  **long** endTime = System.*currentTimeMillis*();  *log*.info("sadd/smembers sinter查询交集合 方式执行 select\* from tableName where authorName=? and url=?耗时：" + (endTime - time) + "ms");  }    **private** **void** selectTypeTwo(NoteDao noteDao, String url) {  **long** time = System.*currentTimeMillis*();  Set<String> sortKey = *jedis*.smembers("Note:fromUrl:" + url);  *display*(sortKey, noteDao);  **long** endTime = System.*currentTimeMillis*();  *log*.info("sadd/smembers set方式执行 select \* from table where url=?耗时：" + (endTime - time) + "ms");  }    **private** **static** **void** selectTypeOne(NoteDao noteDao, String url) {  **long** time = System.*currentTimeMillis*();  Set<String> sort = *jedis*.keys("Note:\*:fromUrl");  // 符合条件的key值集合  Set<String> sortKey = **new** HashSet<String>();  **for** (String key : sort) {  **if** (url.equals(*jedis*.get(key))) {  StringTokenizer st = **new** StringTokenizer(key, ":");  st.nextToken();  sortKey.add("Note:" + st.nextToken());  }  }  *display*(sortKey, noteDao);  **long** endTime = System.*currentTimeMillis*();  *log*.info("set/get方式执行 select \* from table where url=?耗时：" + (endTime - time) + "ms");  }    **private** **static** **void** display(Set<String> sortKey, NoteDao noteDao) {  // 查询所有key的结果  List<Note> noteList = noteDao.getListBean(sortKey);  // 打印检查  **for** (Note note2 : noteList) {  *log*.info(note2.toString());  }  }    **private** **static** **void** selectById(NoteDao noteDao) {  Note note = noteDao.getBean("Note:29453");  *log*.info("获取redis服务器上面的值。select \* from table where id=?");  *log*.info(note.toString());  }    **private** **void** delete(NoteDao noteDao) {    }    **private** **void** update(NoteDao noteDao) {  Note note = noteDao.getBean("Note:29892");  // 修改了url  String newFromUrl = "www.baidu.com";  String oldFromUrl = note.getFromUrl();  String id = "29892";  String jsonString = "";  note.setFromUrl(newFromUrl);    // 如果删除的话就用到flag标记  // note.setFlag(1);    // 记录日志  **if** (**null** != note) {  String className = note.getClass().getName();  jsonString = JSON.*toJSON*(note).toString();  noteDao.log(className, jsonString);  }    // 更新缓存类型一  setTable("Note", id, "fromUrl", newFromUrl);    // 更新缓存类型二、先去除原来key里面的，然后再重新创建一个  *jedis*.srem("Note:fromUrl:" + oldFromUrl, id);  saddColumn("Note", "fromUrl", newFromUrl, id);  // 更新缓存类型三  noteDao.setJSON("Note", id, jsonString);    } |

## 简单封装的公共api

Git地址：

<https://github.com/ggj2010/TeamStudy/blob/user/gaoguangjin/main/src/main/java/com/team/gaoguangjin/%E7%BC%93%E5%AD%98/redis/util/RedisDao.java>

|  |
| --- |
| package com.team.gaoguangjin.缓存.redis.util;  import java.lang.reflect.ParameterizedType;  import java.lang.reflect.Type;  import java.util.ArrayList;  import java.util.List;  import java.util.Set;  import redis.clients.jedis.Jedis;  import com.alibaba.fastjson.JSON;  /\*\*  \* @ClassName:RedisDao.java  \* @Description:redisDao公共API  \* @author gaoguangjin  \* @Date 2015-5-17 下午10:24:33  \*/  public abstract class RedisDao<T> {  public final static String SPLIT\_MARK = ":";  public final static String SORT = "sort";  public final static String INDEX = "index";  public final static String UPDATE\_LOG = "updatelog";  public static Jedis jedis;  protected Class<T> classs;    static {  // 连接redis linux服务器  jedis = new Jedis("123.56.118.135", 6379);  // 权限认证  jedis.auth("gaoguangjin");// 密码最好越长越好，防止暴力破解  }    /\*\*  \* 构造器 自动获取class  \*/  public RedisDao() {  Class clazz = getClass();  while (clazz != Object.class) {  Type t = clazz.getGenericSuperclass();  if (t instanceof ParameterizedType) {  Type[] args = ((ParameterizedType) t).getActualTypeArguments();  if (args[0] instanceof Class) {  this.classs = (Class<T>) args[0];  break;  }  }  classs = clazz.getSuperclass();  }  }    /\*\*  \* @Description比较key存储的value 和传入的value是否相等  \* @param key  \* @return:Boolean  \*/  public Boolean existValueByKey(String key, String value) {  return jedis.get(key) == value ? true : false;  }    /\*\*  \* @Description:初始化将表中数据放到redis,存放格式为tableName:id:column.  \* @param tableName 表映射的类名  \* @param id 主键ID值  \* @param column 表映射的列信息  \* @param value 列对应的值  \* @return:String  \*/  public String setTable(String tableName, String id, String column, String value) {  return jedis.set(tableName + SPLIT\_MARK + id + SPLIT\_MARK + column, value);  }    /\*\*  \* @Description:初始化存放表中所有字段数据，存放格式为tableName:column:columnValue  \* @param tableName  \* @param column  \* @param value  \* @param id  \* @return:Long  \*/  public Long saddColumn(String tableName, String column, String columnValue, String id) {  return jedis.sadd(tableName + SPLIT\_MARK + column + SPLIT\_MARK + columnValue, tableName + SPLIT\_MARK + id);  }    /\*\*  \* @Description:初始化将表中数据放到redis,将bean转换成json格式，存放格式为tableName:id。  \* @param tableName 表映射的类名  \* @param id 主键id值  \* @param json 改id值对应的json字符串  \* @return:String  \*/  public String setJSON(String tableName, String id, String json) {  return jedis.set(tableName + SPLIT\_MARK + id, json);  }    /\*\*  \* @Description: 根据jsonKey，获取相应的json字符串，转换成实体类List  \* @param key  \* @return  \* @return:List<T>  \*/  public List<T> getListBean(Set<String> sortKey) {  List<T> list = new ArrayList<T>();  for (String key : sortKey) {  list.add(getBean(key));  }  return list;  }    /\*\*  \* @Description:根据jsonKey，获取相应的json字符串，转换成实体类  \* @param key  \* @return:T dao层泛型的实体类  \*/  public T getBean(String key) {  return JSON.parseObject(jedis.get(key), classs);  }    /\*\*  \* @Description:对指定值加排序  \* @param tableName  \* @param column  \* @param value  \* @param id  \* @return:void  \*/  public Long zaddSort(String tableName, String column, String value, String id) {  return jedis.zadd(tableName + SPLIT\_MARK + SORT + SPLIT\_MARK + column, Double.parseDouble(value), id);  }    /\*\*  \* @Description:对表主键加索引  \* @param tableName  \* @param column  \* @param value  \* @return:Long  \*/  public Long zaddIndex(String tableName, String column, String value) {  return jedis.sadd(tableName + SPLIT\_MARK + INDEX + SPLIT\_MARK + column, value);  }    /\*\*  \* @Description: 对redis缓存操作的日志，以对象json格式保存，然后队列方式更新到数据库  \* @param value  \* @return:Long  \*/  public Long log(String className, String value) {  // set里面放类名+json list里面放类名  jedis.sadd(className, value);  return jedis.lpush(UPDATE\_LOG, className);  }    /\*\*  \* @Description:定时任务自动将更新数据同步到数据库  \* @return:void  \* @throws ClassNotFoundException  \*/  public List<Object> updateToDataBase() throws ClassNotFoundException {  List<Object> ob = new ArrayList<Object>();  String className = jedis.lpop(UPDATE\_LOG);  Set<String> keySet = jedis.smembers(className);  if (keySet.size() > 0) {  Class<?> object = classs.forName(className);  for (String json : keySet) {  Object bean = JSON.parseObject(json, object);  ob.add(bean);  }  }  return ob;  }  } |