package com.wujintao.redis;

import java.util.Date;

import java.util.HashMap;

import java.util.Iterator;

import java.util.List;

import java.util.Map;

import java.util.Set;

import org.junit.Test;

import redis.clients.jedis.Jedis;

import redis.clients.jedis.Pipeline;

import redis.clients.jedis.SortingParams;

import com.wujintao.redis.util.RedisUtil;

public class TestCase {

/\*\*

\* 在不同的线程中使用相同的Jedis实例会发生奇怪的错误。但是创建太多的实现也不好因为这意味着会建立很多sokcet连接，

\* 也会导致奇怪的错误发生。单一Jedis实例不是线程安全的。为了避免这些问题，可以使用JedisPool,

\* JedisPool是一个线程安全的网络连接池。可以用JedisPool创建一些可靠Jedis实例，可以从池中拿到Jedis的实例。

\* 这种方式可以解决那些问题并且会实现高效的性能

\*/

public static void main(String[] args) {

// ...when closing your application:

RedisUtil.getPool().destroy();

}

public static void Hello() {

Jedis jedis = RedisUtil.getJedis();

try {

// 向key-->name中放入了value-->minxr

jedis.set("name", "minxr");

String ss = jedis.get("name");

System.out.println(ss);

// 很直观，类似map 将jintao append到已经有的value之后

jedis.append("name", "jintao");

ss = jedis.get("name");

System.out.println(ss);

// 2、直接覆盖原来的数据

jedis.set("name", "jintao");

System.out.println(jedis.get("jintao"));

// 删除key对应的记录

jedis.del("name");

System.out.println(jedis.get("name"));// 执行结果：null

/\*\*

\* mset相当于 jedis.set("name","minxr"); jedis.set("jarorwar","aaa");

\*/

jedis.mset("name", "minxr", "jarorwar", "aaa");

System.out.println(jedis.mget("name", "jarorwar"));

} catch (Exception e) {

e.printStackTrace();

} finally {

RedisUtil.getPool().returnResource(jedis);

}

}

private void testKey() {

Jedis jedis = RedisUtil.getJedis();

System.out.println("=============key==========================");

// 清空数据

System.out.println(jedis.flushDB());

System.out.println(jedis.echo("foo"));

// 判断key否存在

System.out.println(jedis.exists("foo"));

jedis.set("key", "values");

System.out.println(jedis.exists("key"));

}

public static void testString() {

System.out.println("==String==");

Jedis jedis = RedisUtil.getJedis();

try {

// String

jedis.set("key", "Hello World!");

String value = jedis.get("key");

System.out.println(value);

} catch (Exception e) {

e.printStackTrace();

} finally {

RedisUtil.getPool().returnResource(jedis);

}

System.out.println("=============String==========================");

// 清空数据

System.out.println(jedis.flushDB());

// 存储数据

jedis.set("foo", "bar");

System.out.println(jedis.get("foo"));

// 若key不存在，则存储

jedis.setnx("foo", "foo not exits");

System.out.println(jedis.get("foo"));

// 覆盖数据

jedis.set("foo", "foo update");

System.out.println(jedis.get("foo"));

// 追加数据

jedis.append("foo", " hello, world");

System.out.println(jedis.get("foo"));

// 设置key的有效期，并存储数据

jedis.setex("foo", 2, "foo not exits");

System.out.println(jedis.get("foo"));

try {

Thread.sleep(3000);

} catch (InterruptedException e) {

}

System.out.println(jedis.get("foo"));

// 获取并更改数据

jedis.set("foo", "foo update");

System.out.println(jedis.getSet("foo", "foo modify"));

// 截取value的值

System.out.println(jedis.getrange("foo", 1, 3));

System.out.println(jedis.mset("mset1", "mvalue1", "mset2", "mvalue2",

"mset3", "mvalue3", "mset4", "mvalue4"));

System.out.println(jedis.mget("mset1", "mset2", "mset3", "mset4"));

System.out.println(jedis.del(new String[] { "foo", "foo1", "foo3" }));

}

public static void testList() {

System.out.println("==List==");

Jedis jedis = RedisUtil.getJedis();

try {

// 开始前，先移除所有的内容

jedis.del("messages");

jedis.rpush("messages", "Hello how are you?");

jedis.rpush("messages", "Fine thanks. I'm having fun with redis.");

jedis.rpush("messages", "I should look into this NOSQL thing ASAP");

// 再取出所有数据jedis.lrange是按范围取出，

// 第一个是key，第二个是起始位置，第三个是结束位置，jedis.llen获取长度 -1表示取得所有

List<String> values = jedis.lrange("messages", 0, -1);

System.out.println(values);

} catch (Exception e) {

e.printStackTrace();

} finally {

RedisUtil.getPool().returnResource(jedis);

}

// 清空数据

System.out.println(jedis.flushDB());

// 添加数据

jedis.lpush("lists", "vector");

jedis.lpush("lists", "ArrayList");

jedis.lpush("lists", "LinkedList");

// 数组长度

System.out.println(jedis.llen("lists"));

// 排序

System.out.println(jedis.sort("lists"));

// 字串

System.out.println(jedis.lrange("lists", 0, 3));

// 修改列表中单个值

jedis.lset("lists", 0, "hello list!");

// 获取列表指定下标的值

System.out.println(jedis.lindex("lists", 1));

// 删除列表指定下标的值

System.out.println(jedis.lrem("lists", 1, "vector"));

// 删除区间以外的数据

System.out.println(jedis.ltrim("lists", 0, 1));

// 列表出栈

System.out.println(jedis.lpop("lists"));

// 整个列表值

System.out.println(jedis.lrange("lists", 0, -1));

}

public static void testSet() {

System.out.println("==Set==");

Jedis jedis = RedisUtil.getJedis();

try {

jedis.sadd("myset", "1");

jedis.sadd("myset", "2");

jedis.sadd("myset", "3");

jedis.sadd("myset", "4");

Set<String> setValues = jedis.smembers("myset");

System.out.println(setValues);

// 移除noname

jedis.srem("myset", "4");

System.out.println(jedis.smembers("myset"));// 获取所有加入的value

System.out.println(jedis.sismember("myset", "4"));// 判断 minxr

// 是否是sname集合的元素

System.out.println(jedis.scard("sname"));// 返回集合的元素个数

} catch (Exception e) {

e.printStackTrace();

} finally {

RedisUtil.getPool().returnResource(jedis);

}

// 清空数据

System.out.println(jedis.flushDB());

// 添加数据

jedis.sadd("sets", "HashSet");

jedis.sadd("sets", "SortedSet");

jedis.sadd("sets", "TreeSet");

// 判断value是否在列表中

System.out.println(jedis.sismember("sets", "TreeSet"));

;

// 整个列表值

System.out.println(jedis.smembers("sets"));

// 删除指定元素

System.out.println(jedis.srem("sets", "SortedSet"));

// 出栈

System.out.println(jedis.spop("sets"));

System.out.println(jedis.smembers("sets"));

//

jedis.sadd("sets1", "HashSet1");

jedis.sadd("sets1", "SortedSet1");

jedis.sadd("sets1", "TreeSet");

jedis.sadd("sets2", "HashSet2");

jedis.sadd("sets2", "SortedSet1");

jedis.sadd("sets2", "TreeSet1");

// 交集

System.out.println(jedis.sinter("sets1", "sets2"));

// 并集

System.out.println(jedis.sunion("sets1", "sets2"));

// 差集

System.out.println(jedis.sdiff("sets1", "sets2"));

}

public static void sortedSet() {

System.out.println("==SoretedSet==");

Jedis jedis = RedisUtil.getJedis();

try {

jedis.zadd("hackers", 1940, "Alan Kay");

jedis.zadd("hackers", 1953, "Richard Stallman");

jedis.zadd("hackers", 1965, "Yukihiro Matsumoto");

jedis.zadd("hackers", 1916, "Claude Shannon");

jedis.zadd("hackers", 1969, "Linus Torvalds");

jedis.zadd("hackers", 1912, "Alan Turing");

Set<String> setValues = jedis.zrange("hackers", 0, -1);

System.out.println(setValues);

Set<String> setValues2 = jedis.zrevrange("hackers", 0, -1);

System.out.println(setValues2);

} catch (Exception e) {

e.printStackTrace();

} finally {

RedisUtil.getPool().returnResource(jedis);

}

// 清空数据

System.out.println(jedis.flushDB());

// 添加数据

jedis.zadd("zset", 10.1, "hello");

jedis.zadd("zset", 10.0, ":");

jedis.zadd("zset", 9.0, "zset");

jedis.zadd("zset", 11.0, "zset!");

// 元素个数

System.out.println(jedis.zcard("zset"));

// 元素下标

System.out.println(jedis.zscore("zset", "zset"));

// 集合子集

System.out.println(jedis.zrange("zset", 0, -1));

// 删除元素

System.out.println(jedis.zrem("zset", "zset!"));

System.out.println(jedis.zcount("zset", 9.5, 10.5));

// 整个集合值

System.out.println(jedis.zrange("zset", 0, -1));

}

public static void testHsh() {

System.out.println("==Hash==");

Jedis jedis = RedisUtil.getJedis();

try {

Map<String, String> pairs = new HashMap<String, String>();

pairs.put("name", "Akshi");

pairs.put("age", "2");

pairs.put("sex", "Female");

jedis.hmset("kid", pairs);

List<String> name = jedis.hmget("kid", "name");// 结果是个泛型的LIST

// jedis.hdel("kid","age"); //删除map中的某个键值

System.out.println(jedis.hmget("kid", "pwd")); // 因为删除了，所以返回的是null

System.out.println(jedis.hlen("kid")); // 返回key为user的键中存放的值的个数

System.out.println(jedis.exists("kid"));// 是否存在key为user的记录

System.out.println(jedis.hkeys("kid"));// 返回map对象中的所有key

System.out.println(jedis.hvals("kid"));// 返回map对象中的所有value

Iterator<String> iter = jedis.hkeys("kid").iterator();

while (iter.hasNext()) {

String key = iter.next();

System.out.println(key + ":" + jedis.hmget("kid", key));

}

List<String> values = jedis.lrange("messages", 0, -1);

values = jedis.hmget("kid", new String[] { "name", "age", "sex" });

System.out.println(values);

Set<String> setValues = jedis.zrange("hackers", 0, -1);

setValues = jedis.hkeys("kid");

System.out.println(setValues);

values = jedis.hvals("kid");

System.out.println(values);

pairs = jedis.hgetAll("kid");

System.out.println(pairs);

} catch (Exception e) {

e.printStackTrace();

} finally {

RedisUtil.getPool().returnResource(jedis);

}

// 清空数据

System.out.println(jedis.flushDB());

// 添加数据

jedis.hset("hashs", "entryKey", "entryValue");

jedis.hset("hashs", "entryKey1", "entryValue1");

jedis.hset("hashs", "entryKey2", "entryValue2");

// 判断某个值是否存在

System.out.println(jedis.hexists("hashs", "entryKey"));

// 获取指定的值

System.out.println(jedis.hget("hashs", "entryKey")); // 批量获取指定的值

System.out.println(jedis.hmget("hashs", "entryKey", "entryKey1"));

// 删除指定的值

System.out.println(jedis.hdel("hashs", "entryKey"));

// 为key中的域 field 的值加上增量 increment

System.out.println(jedis.hincrBy("hashs", "entryKey", 123l));

// 获取所有的keys

System.out.println(jedis.hkeys("hashs"));

// 获取所有的values

System.out.println(jedis.hvals("hashs"));

}

public static void testOther() throws InterruptedException {

Jedis jedis = RedisUtil.getJedis();

try {

// keys中传入的可以用通配符

System.out.println(jedis.keys("\*")); // 返回当前库中所有的key [sose, sanme,

// name, jarorwar, foo,

// sname, java framework,

// user, braand]

System.out.println(jedis.keys("\*name"));// 返回的sname [sname, name]

System.out.println(jedis.del("sanmdde"));// 删除key为sanmdde的对象 删除成功返回1

// 删除失败（或者不存在）返回 0

System.out.println(jedis.ttl("sname"));// 返回给定key的有效时间，如果是-1则表示永远有效

jedis.setex("timekey", 10, "min");// 通过此方法，可以指定key的存活（有效时间） 时间为秒

Thread.sleep(5000);// 睡眠5秒后，剩余时间将为<=5

System.out.println(jedis.ttl("timekey")); // 输出结果为5

jedis.setex("timekey", 1, "min"); // 设为1后，下面再看剩余时间就是1了

System.out.println(jedis.ttl("timekey")); // 输出结果为1

System.out.println(jedis.exists("key"));// 检查key是否存在

System.out.println(jedis.rename("timekey", "time"));

System.out.println(jedis.get("timekey"));// 因为移除，返回为null

System.out.println(jedis.get("time")); // 因为将timekey 重命名为time

// 所以可以取得值 min

// jedis 排序

// 注意，此处的rpush和lpush是List的操作。是一个双向链表（但从表现来看的）

jedis.del("a");// 先清除数据，再加入数据进行测试

jedis.rpush("a", "1");

jedis.lpush("a", "6");

jedis.lpush("a", "3");

jedis.lpush("a", "9");

System.out.println(jedis.lrange("a", 0, -1));// [9, 3, 6, 1]

System.out.println(jedis.sort("a")); // [1, 3, 6, 9] //输入排序后结果

System.out.println(jedis.lrange("a", 0, -1));

} catch (Exception e) {

e.printStackTrace();

} finally {

RedisUtil.getPool().returnResource(jedis);

}

}

@org.junit.Test

public void testUnUsePipeline() {

long start = new Date().getTime();

Jedis jedis = RedisUtil.getJedis();

for (int i = 0; i < 10000; i++) {

jedis.set("age1" + i, i + "");

jedis.get("age1" + i);// 每个操作都发送请求给redis-server

}

long end = new Date().getTime();

System.out.println("unuse pipeline cost:" + (end - start) + "ms");

RedisUtil.getPool().returnResource(jedis);

}

@org.junit.Test

/\*\*

\* 参考:http://blog.csdn.net/freebird\_lb/article/details/7778919

\*/

public void testUsePipeline() {

long start = new Date().getTime();

Jedis jedis = RedisUtil.getJedis();

jedis.flushDB();

Pipeline p = jedis.pipelined();

for (int i = 0; i < 10000; i++) {

p.set("age2" + i, i + "");

System.out.println(p.get("age2" + i));

}

p.sync();// 这段代码获取所有的response

long end = new Date().getTime();

System.out.println("use pipeline cost:" + (end - start) + "ms");

RedisUtil.getPool().returnResource(jedis);

}

@org.junit.Test

/\*\*

\* 时间复杂度：

O(N+M\*log(M))， N 为要排序的列表或集合内的元素数量， M 为要返回的元素数量。

如果只是使用 SORT 命令的 GET 选项获取数据而没有进行排序，时间复杂度 O(N)。

\*/

public void testSort1() {

// 排序默认以数字作为对象，值被解释为双精度浮点数，然后进行比较

Jedis redis = RedisUtil.getJedis();

// 一般SORT用法 最简单的SORT使用方法是SORT key。

redis.lpush("mylist", "1");

redis.lpush("mylist", "4");

redis.lpush("mylist", "6");

redis.lpush("mylist", "3");

redis.lpush("mylist", "0");

// List<String> list = redis.sort("sort");// 默认是升序

SortingParams sortingParameters = new SortingParams();

sortingParameters.desc();

// sortingParameters.alpha();//当数据集中保存的是字符串值时，你可以用 ALPHA

// 修饰符(modifier)进行排序。

sortingParameters.limit(0, 2);// 可用于分页查询

List<String> list = redis.sort("mylist", sortingParameters);// 默认是升序

for (int i = 0; i < list.size(); i++) {

System.out.println(list.get(i));

}

redis.flushDB();

RedisUtil.closeJedis(redis);

}

@org.junit.Test

/\*\*

\* sort list

\* LIST结合hash的排序

\*/

public void testSort2() {

Jedis jedis = RedisUtil.getJedis();

jedis.del("user:66", "user:55", "user:33", "user:22", "user:11",

"userlist");

jedis.lpush("userlist", "33");

jedis.lpush("userlist", "22");

jedis.lpush("userlist", "55");

jedis.lpush("userlist", "11");

jedis.hset("user:66", "name", "66");

jedis.hset("user:55", "name", "55");

jedis.hset("user:33", "name", "33");

jedis.hset("user:22", "name", "79");

jedis.hset("user:11", "name", "24");

jedis.hset("user:11", "add", "beijing");

jedis.hset("user:22", "add", "shanghai");

jedis.hset("user:33", "add", "guangzhou");

jedis.hset("user:55", "add", "chongqing");

jedis.hset("user:66", "add", "xi'an");

SortingParams sortingParameters = new SortingParams();

// 符号 "->" 用于分割哈希表的键名(key name)和索引域(hash field)，格式为 "key->field" 。

sortingParameters.get("user:\*->name");

sortingParameters.get("user:\*->add");

// sortingParameters.by("user:\*->name");

// sortingParameters.get("#");

List<String> result = jedis.sort("userlist", sortingParameters);

for (String item : result) {

System.out.println("item...." + item);

}

/\*\*

\* 对应的redis客户端命令是：sort ml get user\*->name sort ml get user:\*->name get

\* user:\*->add

\*/

}

@org.junit.Test

/\*\*

\* sort set

\* SET结合String的排序

\*/

public void testSort3() {

Jedis jedis = RedisUtil.getJedis();

jedis.del("tom:friend:list", "score:uid:123", "score:uid:456",

"score:uid:789", "score:uid:101", "uid:123", "uid:456",

"uid:789", "uid:101");

jedis.sadd("tom:friend:list", "123"); // tom的好友列表

jedis.sadd("tom:friend:list", "456");

jedis.sadd("tom:friend:list", "789");

jedis.sadd("tom:friend:list", "101");

jedis.set("score:uid:123", "1000"); // 好友对应的成绩

jedis.set("score:uid:456", "6000");

jedis.set("score:uid:789", "100");

jedis.set("score:uid:101", "5999");

jedis.set("uid:123", "{'uid':123,'name':'lucy'}"); // 好友的详细信息

jedis.set("uid:456", "{'uid':456,'name':'jack'}");

jedis.set("uid:789", "{'uid':789,'name':'jay'}");

jedis.set("uid:101", "{'uid':101,'name':'jolin'}");

SortingParams sortingParameters = new SortingParams();

sortingParameters.desc();

// sortingParameters.limit(0, 2);

// 注意GET操作是有序的，GET user\_name\_\* GET user\_password\_\*

// 和 GET user\_password\_\* GET user\_name\_\*返回的结果位置不同

sortingParameters.get("#");// GET 还有一个特殊的规则—— "GET #"

// ，用于获取被排序对象(我们这里的例子是 user\_id )的当前元素。

sortingParameters.get("uid:\*");

sortingParameters.get("score:uid:\*");

sortingParameters.by("score:uid:\*");

// 对应的redis 命令是./redis-cli sort tom:friend:list by score:uid:\* get # get

// uid:\* get score:uid:\*

List<String> result = jedis.sort("tom:friend:list", sortingParameters);

for (String item : result) {

System.out.println("item..." + item);

}

}

/\*\*

\*

\* 只获取对象而不排序 BY 修饰符可以将一个不存在的 key 当作权重，让 SORT 跳过排序操作。

\* 该方法用于你希望获取外部对象而又不希望引起排序开销时使用。 # 确保fake\_key不存在 redis> EXISTS fake\_key

\* (integer) 0 # 以fake\_key作BY参数，不排序，只GET name 和 GET password redis> SORT

\* user\_id BY fake\_key GET # GET user\_name\_\* GET user\_password\_\* 1) "222" #

\* id 2) "hacker" # user\_name 3) "hey,im in" # password 4) "59230" 5) "jack"

\* 6) "jack201022" 7) "2" 8) "huangz" 9) "nobodyknows" 10) "1" 11) "admin"

\* 12) "a\_long\_long\_password"

\*/

public void testSort4() {

}

/\*\*

\*

保存排序结果 默认情况下， SORT 操作只是简单地返回排序结果，如果你希望保存排序结果，可以给 STORE 选项指定一个 key

\* 作为参数，排序结果将以列表的形式被保存到这个 key 上。(若指定 key 已存在，则覆盖。) redis> EXISTS

\* user\_info\_sorted\_by\_level # 确保指定key不存在 (integer) 0 redis> SORT user\_id BY

\* user\_level\_\* GET # GET user\_name\_\* GET user\_password\_\* STORE

\* user\_info\_sorted\_by\_level # 排序 (integer) 12 # 显示有12条结果被保存了 redis> LRANGE

\* user\_info\_sorted\_by\_level 0 11 # 查看排序结果 1) "59230" 2) "jack" 3)

\* "jack201022" 4) "2" 5) "huangz" 6) "nobodyknows" 7) "222" 8) "hacker" 9)

\* "hey,im in" 10) "1" 11) "admin" 12) "a\_long\_long\_password" 一个有趣的用法是将 SORT

\* 结果保存，用 EXPIRE 为结果集设置生存时间，这样结果集就成了 SORT 操作的一个缓存。 这样就不必频繁地调用 SORT

\* 操作了，只有当结果集过期时，才需要再调用一次 SORT 操作。

\* 有时候为了正确实现这一用法，你可能需要加锁以避免多个客户端同时进行缓存重建(也就是多个客户端，同一时间进行 SORT

\* 操作，并保存为结果集)，具体参见 SETNX 命令。

\*/

@Test

public void testSort5() {

// 排序默认以数字作为对象，值被解释为双精度浮点数，然后进行比较

Jedis jedis = RedisUtil.getJedis();

// 一般SORT用法 最简单的SORT使用方法是SORT key。

jedis.lpush("mylist", "1");

jedis.lpush("mylist", "4");

jedis.lpush("mylist", "6");

jedis.lpush("mylist", "3");

jedis.lpush("mylist", "0");

// List<String> list = redis.sort("sort");// 默认是升序

SortingParams sortingParameters = new SortingParams();

sortingParameters.desc();

// sortingParameters.alpha();//当数据集中保存的是字符串值时，你可以用 ALPHA

// 修饰符(modifier)进行排序。

// sortingParameters.limit(0, 2);//可用于分页查询

// 没有使用 STORE 参数，返回列表形式的排序结果. 使用 STORE 参数，返回排序结果的元素数量。

jedis.sort("mylist", sortingParameters, "mylist");// 排序后指定排序结果到一个KEY中，这里讲结果覆盖原来的KEY

List<String> list = jedis.lrange("mylist", 0, -1);

for (int i = 0; i < list.size(); i++) {

System.out.println(list.get(i));

}

jedis.sadd("tom:friend:list", "123"); // tom的好友列表

jedis.sadd("tom:friend:list", "456");

jedis.sadd("tom:friend:list", "789");

jedis.sadd("tom:friend:list", "101");

jedis.set("score:uid:123", "1000"); // 好友对应的成绩

jedis.set("score:uid:456", "6000");

jedis.set("score:uid:789", "100");

jedis.set("score:uid:101", "5999");

jedis.set("uid:123", "{'uid':123,'name':'lucy'}"); // 好友的详细信息

jedis.set("uid:456", "{'uid':456,'name':'jack'}");

jedis.set("uid:789", "{'uid':789,'name':'jay'}");

jedis.set("uid:101", "{'uid':101,'name':'jolin'}");

sortingParameters = new SortingParams();

// sortingParameters.desc();

sortingParameters.get("#");// GET 还有一个特殊的规则—— "GET #"

// ，用于获取被排序对象(我们这里的例子是 user\_id )的当前元素。

sortingParameters.by("score:uid:\*");

jedis.sort("tom:friend:list", sortingParameters, "tom:friend:list");

List<String> result = jedis.lrange("tom:friend:list", 0, -1);

for (String item : result) {

System.out.println("item..." + item);

}

jedis.flushDB();

RedisUtil.closeJedis(jedis);

}

public void testMore(){

//ZRANGE取出最新的10个项目。

//使用LPUSH + LTRIM，确保只取出最新的1000条项目。

//HINCRBY key field increment,为哈希表 key 中的域 field 的值加上增量 increment

//INCRBY,HINCRBY等等，redis有了原子递增（atomic increment），你可以放心的加上各种计数，用GETSET重置，或者是让它们过期。

// LREM greet 2 morning # 移除从表头到表尾，最先发现的两个 morning,这个可以用来删除特定评论

// zrevrank test a 查看a在sorted set中倒排序时排在第几名，查询结果按照INDEX，所以INDEX是3表示排在第四名

// zrank test a 相反，表示正排序时候的名次

// zscore test one表示one这个元素在sorted set中的score为多少

// zrevrange test 0 -1 表示sorted set倒排序,zrange test 0 -1表示正排序

//将一个或多个 member 元素及其 score 值加入到有序集 key 当中。如果某个 member 已经是有序集的成员，那么更新这个 member 的 score 值，并通过重新插入这个 member 元素，来保证该 member 在正确的位置上。

//zrem test one删除sorted set中某个元素

}

public List<String> get\_latest\_comments(int start, int num\_items){

//获取最新评论

//LPUSH latest.comments <ID>

//-我们将列表裁剪为指定长度，因此Redis只需要保存最新的5000条评论：

//LTRIM latest.comments 0 5000

//们做了限制不能超过5000个ID，因此我们的获取ID函数会一直询问Redis。只有在start/count参数超出了这个范围的时候，才需要去访问数据库。

Jedis jedis = RedisUtil.getJedis();

List<String> id\_list = jedis.lrange("latest.comments",start,start+num\_items-1) ;

if(id\_list.size()<num\_items){

//id\_list = SQL.EXECUTE("SELECT ... ORDER BY time LIMIT ...");

}

return id\_list;

}

@Test

public void testDB() {

Jedis jedis = RedisUtil.getJedis();

System.out.println(jedis.select(0));// select db-index

// 通过索引选择数据库，默认连接的数据库所有是0,默认数据库数是16个。返回1表示成功，0失败

System.out.println(jedis.dbSize());// dbsize 返回当前数据库的key数量

System.out.println(jedis.keys("\*")); // 返回匹配指定模式的所有key

System.out.println(jedis.randomKey());

jedis.flushDB();// 删除当前数据库中所有key,此方法不会失败。慎用

jedis.flushAll();// 删除所有数据库中的所有key，此方法不会失败。更加慎用

}

@Test

public void testMget() {

Jedis jedis = RedisUtil.getJedis();

jedis.flushDB();// 删除当前数据库中所有key,此方法不会失败。慎用

jedis.rpush("ids", "aa");

jedis.rpush("ids", "bb");

jedis.rpush("ids", "cc");

List<String> ids = jedis.lrange("ids", 0, -1);

jedis.set("aa", "{'name':'zhoujie','age':20}");

jedis.set("bb", "{'name':'yilin','age':28}");

jedis.set("cc", "{'name':'lucy','age':21}");

List<String> list = jedis.mget(ids.toArray(new String[ids.size()]));

System.out.println(list);

}

/\*\*

\* 可以利用lrange对list进行分页操作

\*/

@Test

public void queryPageBy() {

int pageNo = 6;

int pageSize = 6;

Jedis jedis = RedisUtil.getJedis();

jedis.del("a");

for (int i = 1; i <= 30; i++) {

jedis.rpush("a", i + "");

}

int start = pageSize \* (pageNo - 1);// 因为redis中list元素位置基数是0

int end = start + pageSize - 1;

List<String> results = jedis.lrange("a", start, end);// 从start算起，start算一个元素，到结束那个元素

for (String str : results) {

System.out.println(str);

}

}

@Test

/\*\*

\* [向Redis list压入ID而不是实际的数据]

在上面的例子里 ，我们将“对象”（此例中是简单消息）直接压入Redis list，但通常不应这么做，

由于对象可能被多次引用：例如在一个list中维护其时间顺序，在一个集合中保存它的类别，只要有必要，它还会出现在其他list中，等等。

让我们回到reddit.com的例子，将用户提交的链接（新闻）添加到list中，有更可靠的方法如下所示：

$ redis-cli incr next.news.id

(integer) 1

$ redis-cli set news:1:title "Redis is simple"

OK

$ redis-cli set news:1:url "http://code.google.com/p/redis"

OK

$ redis-cli lpush submitted.news 1

OK

我们自增一个key，很容易得到一个独一无二的自增ID，然后通过此ID创建对象–为对象的每个字段设置一个key。最后将新对象的ID压入submitted.news list。

这只是牛刀小试。在命令参考文档中可以读到所有和list有关的命令。你可以删除元素，旋转list，根据索引获取和设置元素，当然也可以用LLEN得到list的长度。

\*/

public void testListStrUsage() {

String title = "太阳能是绿色能源4";

String url = "http://javacreazyer.iteye.com";

Jedis jedis = RedisUtil.getJedis();

long adInfoId = jedis.incr("ad:adinfo:next.id");

jedis.set("ad:adinfo:" + adInfoId + ":title", title);

jedis.set("ad:adinfo:" + adInfoId + ":url", url);

jedis.lpush("ad:adinfo", String.valueOf(adInfoId));

String resultTitle = jedis.get("ad:adinfo:" + adInfoId + ":title");

String resultUrl = jedis.get("ad:adinfo:" + adInfoId + ":url");

List<String> ids = jedis.lrange("ad:adinfo", 0, -1);

System.out.println(resultTitle);

System.out.println(resultUrl);

System.out.println(ids);

/\*\*

\* dbsize返回的是所有key的数目，包括已经过期的， 而redis-cli keys "\*"查询得到的是有效的key数目

\*/

System.out.println(jedis.dbSize());

jedis.flushAll();

}

/\*\*

\* 下面是一个简单的方案：对每个想加标签的对象，用一个标签ID集合与之关联，并且对每个已有的标签，一组对象ID与之关联。 例如假设我们的新闻ID

\* 1000被加了三个标签tag 1,2,5和77，就可以设置下面两个集合： $ redis-cli sadd news:1000:tags 1

\* (integer) 1 $ redis-cli sadd news:1000:tags 2 (integer) 1 $ redis-cli

\* sadd news:1000:tags 5 (integer) 1 $ redis-cli sadd news:1000:tags 77

\* (integer) 1 $ redis-cli sadd tag:1:objects 1000 (integer) 1 $ redis-cli

\* sadd tag:2:objects 1000 (integer) 1 $ redis-cli sadd tag:5:objects 1000

\* (integer) 1 $ redis-cli sadd tag:77:objects 1000 (integer) 1

\* 要获取一个对象的所有标签，如此简单： $ redis-cli smembers news:1000:tags 1. 5 2. 1 3. 77 4.

\* 2 而有些看上去并不简单的操作仍然能使用相应的Redis命令轻松实现。例如我们也许想获得一份同时拥有标签1, 2,

\* 10和27的对象列表。这可以用SINTER命令来做，他可以在不同集合之间取出交集。因此为达目的我们只需： $ redis-cli sinter

\* tag:1:objects tag:2:objects tag:10:objects tag:27:objects ... no result

\* in our dataset composed of just one object ...

\* 在命令参考文档中可以找到和集合相关的其他命令，令人感兴趣的一抓一大把。一定要留意SORT命令，Redis集合和list都是可排序的。

\*/

@Test

public void testSetUsage() {

Jedis jedis = RedisUtil.getJedis();

jedis.sadd("zhongsou:news:1000:tags", "1");

jedis.sadd("zhongsou:news:1000:tags", "2");

jedis.sadd("zhongsou:news:1000:tags", "5");

jedis.sadd("zhongsou:news:1000:tags", "77");

jedis.sadd("zhongsou:news:2000:tags", "1");

jedis.sadd("zhongsou:news:2000:tags", "2");

jedis.sadd("zhongsou:news:2000:tags", "5");

jedis.sadd("zhongsou:news:2000:tags", "77");

jedis.sadd("zhongsou:news:3000:tags", "2");

jedis.sadd("zhongsou:news:4000:tags", "77");

jedis.sadd("zhongsou:news:5000:tags", "1");

jedis.sadd("zhongsou:news:6000:tags", "5");

jedis.sadd("zhongsou:tag:1:objects", 1000 + "");

jedis.sadd("zhongsou:tag:2:objects", 1000 + "");

jedis.sadd("zhongsou:tag:5:objects", 1000 + "");

jedis.sadd("zhongsou:tag:77:objects", 1000 + "");

jedis.sadd("zhongsou:tag:1:objects", 2000 + "");

jedis.sadd("zhongsou:tag:2:objects", 2000 + "");

jedis.sadd("zhongsou:tag:5:objects", 2000 + "");

jedis.sadd("zhongsou:tag:77:objects", 2000 + "");

Set<String> sets = jedis.sinter("zhongsou:tag:1:objects",

"zhongsou:tag:2:objects", "zhongsou:tag:5:objects",

"zhongsou:tag:77:objects");

System.out.println(sets);

jedis.flushAll();

}

@Test

public void testSortedSetUsage() {

Jedis jedis = RedisUtil.getJedis();

jedis.zadd("zhongsou:hackers", 1940, "Alan Kay");

jedis.zadd("zhongsou:hackers", 1953, "Richard Stallman");

jedis.zadd("zhongsou:hackers", 1943, "Jay");

jedis.zadd("zhongsou:hackers", 1920, "Jellon");

jedis.zadd("zhongsou:hackers", 1965, "Yukihiro Matsumoto");

jedis.zadd("zhongsou:hackers", 1916, "Claude Shannon");

jedis.zadd("zhongsou:hackers", 1969, "Linus Torvalds");

jedis.zadd("zhongsou:hackers", 1912, "Alan Turing");

Set<String> hackers = jedis.zrange("zhongsou:hackers", 0, -1);

System.out.println(hackers);

Set<String> hackers2 = jedis.zrevrange("zhongsou:hackers", 0, -1);

System.out.println(hackers2);

// 区间操作,我们请求Redis返回score介于负无穷到1920年之间的元素（两个极值也包含了）。

Set<String> hackers3 = jedis.zrangeByScore("zhongsou:hackers", "-inf",

"1920");

System.out.println(hackers3);

// ZREMRANGEBYSCORE 这个名字虽然不算好，但他却非常有用，还会返回已删除的元素数量。

long num = jedis.zremrangeByScore("zhongsou:hackers", "-inf", "1920");

System.out.println(num);

jedis.flushAll();

}

}