## 1.1安装JDK

## 1.2安装Eclipse

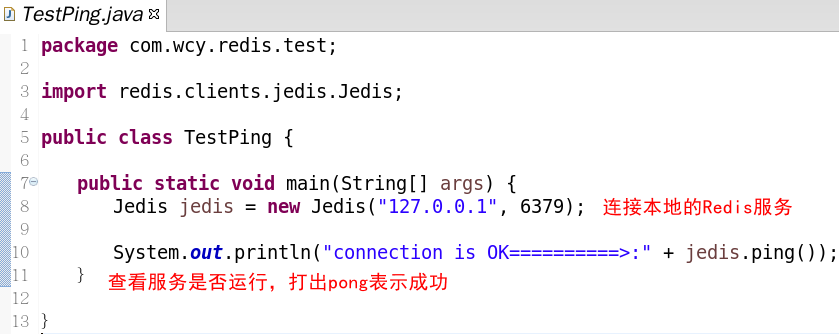
## 1.3Jedis所需要的jar包

Commons-pool-1.6.jar

Jedis-2.1.0.jar

## 1.4Jedis常用操作

1. 测试连通性



1. 常用API 5+1

一个key；

五大数据类型。

**package** com.atguigu.redis.test;

**import** java.util.\*;

**import** redis.clients.jedis.Jedis;

**public** **class** Test02

{

**public** **static** **void** main(String[] args)

{

Jedis jedis = **new** Jedis("127.0.0.1",6379);

//key

Set<String> keys = jedis.keys("\*");

**for** (Iterator iterator = keys.iterator(); iterator.hasNext();) {

String key = (String) iterator.next();

System.*out*.println(key);

}

System.*out*.println("jedis.exists====>"+jedis.exists("k2"));

System.*out*.println(jedis.ttl("k1"));

//String

//jedis.append("k1","myreids");

System.*out*.println(jedis.get("k1"));

jedis.set("k4","k4\_redis");

System.*out*.println("----------------------------------------");

jedis.mset("str1","v1","str2","v2","str3","v3");

System.*out*.println(jedis.mget("str1","str2","str3"));

//list

System.*out*.println("----------------------------------------");

//jedis.lpush("mylist","v1","v2","v3","v4","v5");

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

**for** (String element : list) {

System.*out*.println(element);

}

//set

jedis.sadd("orders","jd001");

jedis.sadd("orders","jd002");

jedis.sadd("orders","jd003");

Set<String> set1 = jedis.smembers("orders");

**for** (Iterator iterator = set1.iterator(); iterator.hasNext();) {

String string = (String) iterator.next();

System.*out*.println(string);

}

jedis.srem("orders","jd002");

System.*out*.println(jedis.smembers("orders").size());

//hash

jedis.hset("hash1","userName","lisi");

System.*out*.println(jedis.hget("hash1","userName"));

Map<String,String> map = **new** HashMap<String,String>();

map.put("telphone","13811814763");

map.put("address","atguigu");

map.put("email","abc@163.com");

jedis.hmset("hash2",map);

List<String> result = jedis.hmget("hash2", "telphone","email");

**for** (String element : result) {

System.*out*.println(element);

}

//zset

jedis.zadd("zset01",60d,"v1");

jedis.zadd("zset01",70d,"v2");

jedis.zadd("zset01",80d,"v3");

jedis.zadd("zset01",90d,"v4");

Set<String> s1 = jedis.zrange("zset01",0,-1);

**for** (Iterator iterator = s1.iterator(); iterator.hasNext();) {

String string = (String) iterator.next();

System.*out*.println(string);

}

}

}

1. 事务提交
   1. 日常：

Jedis jedis = new Jedis("127.0.0.1",6379);

Transaction transaction = jedis.multi(); //被当作一个命令进行执行

transaction.set("k4", "v44");

transaction.set("k5", "v55");

// transaction.exec();

transaction.discard();

* 1. 加锁

1. 主从复制

6379,6380启动，先各自独立。

主写

从读

## 1.5JedisPool

1. 获取Jedis实例需要从JedisPool中获取；
2. 用完Jedis实例需要返还给JedisPool；
3. 如果Jedis在使用过程中出错，则也需要还给JedisPool；
4. 案例代码：

JedisPoolUtil：

**package** com.wcy.redis.test;

**import** redis.clients.jedis.Jedis;

**import** redis.clients.jedis.JedisPool;

**import** redis.clients.jedis.JedisPoolConfig;

**public** **class** JedisPoolUtil {

**private** **static** **volatile** JedisPool *jedisPool* = **null**;// 被volatile修饰的变量不会被本地线程缓存，对该变量的读写都是直接操作共享内存。

**private** JedisPoolUtil() {

}

**public** **static** JedisPool getJedisPoolInstance() {

**if** (**null** == *jedisPool*) {

**synchronized** (JedisPoolUtil.**class**) {

**if** (**null** == *jedisPool*) {

JedisPoolConfig poolConfig = **new** JedisPoolConfig();

poolConfig.setMaxActive(1000);

poolConfig.setMaxIdle(32);

poolConfig.setMaxWait(100 \* 1000);

poolConfig.setTestOnBorrow(**true**);

*jedisPool* = **new** JedisPool(poolConfig, "127.0.0.1");

}

}

}

**return** *jedisPool*;

}

**public** **static** **void** release(JedisPool jedisPool, Jedis jedis) {

**if** (**null** != jedis) {

jedisPool.returnResourceObject(jedis);

}

}

}

**package** com.wcy.redis.test;

**import** redis.clients.jedis.Jedis;

**import** redis.clients.jedis.JedisPool;

**public** **class** TestPool {

**public** **static** **void** main(String[] args) {

JedisPool jedisPoolInstance = JedisPoolUtil.*getJedisPoolInstance*();

Jedis jedis = **null**;

**try** {

jedis = jedisPoolInstance.getResource();

jedis.set("aa", "bb");

} **catch** (Exception e) {

e.printStackTrace();

} **finally** {

JedisPoolUtil.*release*(jedisPoolInstance, jedis);

}

}

}

1. 配置总结all

JedisPool的配置参数大部分是由JedisPoolConfig的对应项来赋值的。

maxActive：控制一个pool可分配多少个jedis实例，通过pool.getResource()来获取；如果赋值为-1，则表示不限制；如果pool已经分配了maxActive个jedis实例，则此时pool的状态为exhausted。

maxIdle：控制一个pool最多有多少个状态为idle(空闲)的jedis实例；

whenExhaustedAction：表示当pool中的jedis实例都被allocated完时，pool要采取的操作；默认有三种。

WHEN\_EXHAUSTED\_FAIL --> 表示无jedis实例时，直接抛出NoSuchElementException；

WHEN\_EXHAUSTED\_BLOCK --> 则表示阻塞住，或者达到maxWait时抛出JedisConnectionException；

WHEN\_EXHAUSTED\_GROW --> 则表示新建一个jedis实例，也就说设置的maxActive无用；

maxWait：表示当borrow一个jedis实例时，最大的等待时间，如果超过等待时间，则直接抛JedisConnectionException；

testOnBorrow：获得一个jedis实例的时候是否检查连接可用性（ping()）；如果为true，则得到的jedis实例均是可用的；

testOnReturn：return 一个jedis实例给pool时，是否检查连接可用性（ping()）；

testWhileIdle：如果为true，表示有一个idle object evitor线程对idle object进行扫描，如果validate失败，此object会被从pool中drop掉；这一项只有在timeBetweenEvictionRunsMillis大于0时才有意义；

timeBetweenEvictionRunsMillis：表示idle object evitor两次扫描之间要sleep的毫秒数；

numTestsPerEvictionRun：表示idle object evitor每次扫描的最多的对象数；

minEvictableIdleTimeMillis：表示一个对象至少停留在idle状态的最短时间，然后才能被idle object evitor扫描并驱逐；这一项只有在timeBetweenEvictionRunsMillis大于0时才有意义；

softMinEvictableIdleTimeMillis：在minEvictableIdleTimeMillis基础上，加入了至少minIdle个对象已经在pool里面了。如果为-1，evicted不会根据idle time驱逐任何对象。如果minEvictableIdleTimeMillis>0，则此项设置无意义，且只有在timeBetweenEvictionRunsMillis大于0时才有意义；

lifo：borrowObject返回对象时，是采用DEFAULT\_LIFO（last in first out，即类似cache的最频繁使用队列），如果为False，则表示FIFO队列；

==================================================================================================================

其中JedisPoolConfig对一些参数的默认设置如下：

testWhileIdle=true

minEvictableIdleTimeMills=60000

timeBetweenEvictionRunsMillis=30000

numTestsPerEvictionRun=-1