一、Redis Client 介绍

1.1、简介

Jedis Client 是 Redis 官网推荐的一个面向 java 客户端，库文件实现了对各类 API 进行封装调用。Jedis 源码工程地址：https://github.com/xetorthio/jedis

1.2、使用

Redis Client 最好选用与服务端对应的版本,本例使用 Redis 2.8.19客户端使用 jedis -2.6.3，Maven工程添加如下引用即可。

```
<dependency>
    <groupId>redis.clients</groupId>
    <artifactId>jedis</artifactId>
    <version>2.6.3</version>
    <type>jar</type>
    <scope>compile</scope>
</dependency>
```

1.3、注意事项

Redis Client拥有众多对接版本，本项目目前使用Jedis为官方推荐Java对接客户端，是基于其对 Redis良好的版本支持和 API 对接，另外编码中尽量避免使用废弃接口。

Redis目前正在新版过渡期，3.0版本暂未稳定，但是由于3.0版本提供了最新的集群功能，可能在日后稳定版发布以后升级到 3.0，目前使用的Jedis 支持 3.0 的目前版本API。

二、Redis Client常用API

2.1、环境要求

语言：Java / JDK：1.7 / Redis : 2.8.19（稳定版）

2.2、系统使用

2.2.1、建立连接

普通连接

```
Jedis jedis = new Jedis("localhost");
jedis.set("foo", "bar");
String value = jedis.get("foo");
System.out.println(value);
```

设置连接池配置

该方法用于得到redis 连接池连接使用的连接池配置，该连接池配置也可以通过spring注入的方式来进行相对应的配置，连接池采用的是平时比较常用的org.apache.commons.pool2.impl.GenericObjectPoolConfig来进行的连接池管理

配置文件如下:

```
#redis 服务器 ip #
  redis.ip=172.30.5.117
#redis服务器端口号#
  redis.port=6379
###jedis##pool##config###
#jedis的最大分配对象#
  jedis.pool.maxActive=1024
#jedis最大保存idel状态对象数 #
  jedis.pool.maxIdle=200
#jedis池没有对象返回时，最大等待时间 #
  jedis.pool.maxWait=1000
#jedis调用 borrowObject方法时，是否进行有效检查#
  jedis.pool.testOnBorrow=true
#jedis调用 returnObject 方法时，是否进行有效检查 #
  jedis.pool.testOnReturn=true
```

连接池配置实例化代码（也可通过 spring 注入进行配置）：

```
 /**
  * 获取化连接池配置
  *@return JedisPoolConfig
  */
  private JedisPoolConfig getPoolConfig(){
     if(config == null){
     config = new JedisPoolConfig();
  //最大连接数
     config.setMaxTotal(Integer.valueOf(getResourceBundle().getString("redis.pool.ma xTotal")));
  //最大空闲连接数
     config.setMaxIdle(Integer.valueOf(getResourceBundle().getString("redis.pool.max Idle")));
  //获取连接时的最大等待毫秒数(如果设置为阻塞时BlockWhenExhausted),如果超时就抛异常, 小于零:阻塞不
确定的时间,  默认-1
     config.setMaxWaitMillis(Long.valueOf(getResourceBundle()
             .getString("redis.pool. maxWaitMillis")));
  //在获取连接的时候检查有效性, 默认false
     config.setTestOnBorrow(Boolean.valueOf(getResourceBundle()
             .getString("redis.pool.testOnBorrow")));
  //在获取返回结果的时候检查有效性, 默认 false
     config.setTestOnReturn(Boolean.valueOf(getResourceBundle()
             .getString("redis.pool.testOnReturn")));
     }
     return config;
  }
```

普通连接池连接

这里展示的是普通的连接池方式链接redis的方案，跟普通的数据库连接池的操作方式类似；

```
  /**
   *初始化 JedisPool
   */
   private void initJedisPool(){
      if(pool == null){
   //获取服务器IP地址
         String ipStr = getResourceBundle().getString("redis.ip");
   //获取服务器端口
         int portStr = Integer.valueOf(getResourceBundle().getString("redis.port"));
   //初始化连接池
         pool = new JedisPool(getPoolConfig(), ipStr,portStr);
      }
   }
```

Sentinel连接池连接

该连接池用于应对 Redis的Sentinel的主从切换机制，能够正确在服务器宕机导致服务器切换时得到正确的服务器连接，当服务器采用该部署策略的时候推荐使用该连接池进行操作；

```
   private void initJedisSentinelPool(){
      if(sentinelpool == null){
      //监听器列表
         Set<String> sentinels = new HashSet<String>(
      //监听器1
         sentinels.add(new HostAndPort("192.168.50.236", 26379).toString());
      //监听器2
         sentinels.add(new HostAndPort("192.168.50.237",26379).toString());
      //实际使用的时候在properties 里配置即可：
      //redis.sentinel.hostandports = 192.168.50.236:26379,192.168.50.237:26379
         getResourceBundle().getString("redis.sentinel.hostandports")
      //mastername是服务器上的master的名字，在master服务器的sentinel.conf 中配置:
      //[sentinel monitor  server-1M  192.168.50.236 6379 2]
      //中间的server-1M即为这里的masterName
         String masterName = getResourceBundle().getString("redis.sentinel.masterName");
      //初始化连接池
         sentinelpool = new JedisSentinelPool(masterName,shardPool = new
                  ShardedJedisPool(getPoolConfig(),serverlist);
      }
   }
```

ShardedJedisPool连接池分片连接

```
/**
 * 初始化ShardedJedisPool
 * Redis在容灾处理方面可以通过服务器端配置Master-Slave模式来实现。
 * 而在分布式集群方面目前只能通过客户端工具来实现一致性哈希分布存储，即key分片存储。
 * Redis可能会在3.0 版本支持服务器端的分布存储
 */
private void initShardedJedisPool() {
   if (shardPool == null) {
   // 创建多个redis 共享服务
      String redis1Ip = getResourceBundle().getString("redis1.ip");
      int redis1Port = Integer.valueOf(bundle.getString("redis.port"));
      JedisShardInfo jedisShardInfo1 = new JedisShardInfo(redis1Ip,redis1Port);
      String redis2Ip = getResourceBundle().getString("redis2.ip");
      int redis2Port = Integer.valueOf(bundle.getString("redis.port"));
      JedisShardInfo jedisShardInfo2 = new JedisShardInfo(redis2Ip,redis2Port);
      List<JedisShardInfo> serverlist = new LinkedList<JedisShardInfo>();
      serverlist.add(jedisShardInfo1);
      serverlist.add(jedisShardInfo2);
   // 初始化连接池
      shardPool = new ShardedJedisPool(getPoolConfig(),serverlist);
   }
}
```

读写删除操作

```
// 从池中获取一个Jedis对象
Jedis jedis = sentinelpool.getSentinelpoolResource();
String keys = "name";
// 删除key-value对象，如果key不存在则忽略此操作
jedis.del(keys);
// 存数据
jedis.set(keys, "snowolf");
// 判断 key 是否存在，不存在返回 false 存在返回true
jedis.exists(keys);
// 取数据
String value = jedis.get(keys);
// 释放对象池（3.0 将抛弃该方法）
sentinelpool.returnSentinelpoolResource(jedis);
```

三、示例代码

1.String的简单追加

```
// 从池中获取一个Jedis对象
JedisUtil.getInstance().STRINGS.append(key, value);
```

2.价格时间排序（前提是已经存储了价格，时间的SortSet）

```
//执行 2级排序操作（） String stPriceSet = “stPriceSet”;
//stPriceSet 价格的 sortset列表名
String stTimeSet = “stTimeSet”;
// stTimeSet时间的 sortset列表名
Set<Tuple> sumSet = JedisUtilEx.getInstance()
          .getSortSetByPirceUpAndTimeDown(stPriceSet, stTimeSet);
//排序以后可以重复获取上次排序结果(缓存时间10 分钟)
Set<Tuple> sumSet = JedisUtilEx.getInstance()
          .getLastPirceUpAndTimeDownSet();
```

3.价格时间排序（前提是已经存储了价格，时间的SortSet）

```
//执行 2级排序操作
String stPriceSet = “stPriceSet”;
//stPriceSet 价格的 sortset列表名
String stTimeSet = “stTimeSet”;
// stTimeSet时间的 sortset列表名
Set<Tuple> sumSet = JedisUtilEx.getInstance()
          .getSortSetByPirceDownAndTimeDown (stPriceSet, stTimeSet);
//排序以后可以重复获取上次排序结果(缓存时间10分钟)
Set<Tuple> sumSet = JedisUtilEx.getInstance()
          .getLastPirceDownAndTimeDownSet ();
```

4.保存 JavaBean 到 hash 表中

```
// bean 继承至 RedisBean
JedisUtilEx.getInstance().setBeanToHash(bean);
```

5.从 hash 表中读取 JavaBean

```
//uuid 为业务制定的唯一标识符规则（相当于主键）
String uuid = “1”;
//该 ID 是我们提前就知道的
//T 继承至 RedisBean;
JedisUtilEx.getInstance().getBeanFromHash (uuid,Class<T> cls);
```

6.将 JavaBean 列表装入 hash 中

```
//list 中的 bean 继承至 RedisBean
List<T> beanList = …;
JedisUtilEx.getInstance().setBeanListToHash(beanList);
//异步版本的存储列表到 hash
JedisUtilEx.getInstance().setBeanListToHashSyn(beanList);
```

7.普通的操作流程示例

```
//获取 jedis 引用
Jedis jedis = JedisUtil.getInstance().getJedis();
//执行业务以及调用 jedis 提供的接口功能
…
jedis.hset(…);
…
//执行完成以后务必释放资源
JedisUtil.getInstance().returnJedis(jedis);
//若以后不会使用JEDIS，需要关闭所有链接池
RedisConnetcion.destroyAllPools();
```

8.事务执行流程

```
//获取连接资源
Jedis jd = JedisUtil.getInstance().getJedis();
//开启事务
Transaction ts = jd.multi();
//执行业务以及调用jedis提供的接口功能
…
jedis.hset(…);
…
//执行事务
List<Object> list = ts.exec();
//释放资源
JedisUtil.getInstance().returnJedis(jd);
```

9.异步执行

```
Jedis jedis = JedisUtil.getInstance().getJedis();   //获取连接资源
Pipeline pipeline = jedis.pipelined();  //获取管
道 //执行业务以及调用jedis提供的接口功能
…
jedis.hset(…);
…
pipeline.syncAndReturnAll();  //提交并释放管道
//释放资源
JedisUtil.getInstance().returnJedis(jedis);
```

10.如何获取 Jedis 命名规则的合成 KEY

```
//获取类的唯一键值key,例如:User:1(User为class,1为uuid)其中user继承于Reidsbean
JedisUtilEx.getInstance().getBeanKey(user);
//另一种获取类的唯一键值 key 的方法
JedisUtilEx.getInstance().getBeanKey(String uuid,Class<T> cls);
//获取 bean 对应的 KEY(对应列的唯一键值 key)
JedisUtilEx.getInstance().getBeanKey(String uuid,Class<T> cls, String... fileds);
//获取 bean 对应的 KEY(集群 key)
JedisUtilEx.getInstance().getBeanKey(Class<T> cls, String... fileds);
```

四、jedis 操作命令：

1.对value操作的命令

exists(key)：确认一个key是否存在
del(key)：删除一个key
type(key)：返回值的类型
keys(pattern)：返回满足给定pattern的所有 key
randomkey：随机返回key空间的一个key
rename(oldname, newname)：将 key 由oldname重命名为newname，
若 newname存在则删除newname表示的 key
dbsize：返回当前数据库中key 的数目
expire：设定一个key的活动时间（s）
ttl：获得一个key的活动时间
select(index)：按索引查询
move(key, dbindex)：将当前数据库中的key 转移到有dbindex索引的数据库 flushdb：
删除当前选择数据库中的所有key
flushall：删除所有数据库中的所有key

2.对String操作的命令

set(key, value)：给数据库中名称为key的string赋予值 value
get(key)：返回数据库中名称为 key的string 的value getset(key, value)：
给名称为key的string 赋予上一次的value
mget(key1,key2,…, key N)：返回库中多个string（它们的名称为key1，key2…）的 value
setnx(key,value)：如果不存在名称为key的string，则向库中添加string名称为 key值为 value
setex(key,time, value)：向库中添加string（名称为key，值为 value）同时,设定过期时间time
mset(key1,value1, key2, value2,…key N, value N)：同时给多个string 赋值，
名称为key i的string赋值value i
msetnx(key1, value1, key2, value2,…key N, value N):如果所有名称为key i的string都不存在，
则向库中添加string，名称key i赋值为 value i
incr(key)：名称为key的string增1操作incrby(key, integer)：名称为 key的string增加integer
decr(key)：名称为key的string减1操作decrby(key, integer)：名称为 key的string减少integer
append(key, value)：名称为key的string的值附加 value
substr(key, start, end)：返回名称为key的string的value的子串

3.对List操作的命令

rpush(key, value)：在名称为key的list 尾添加一个值为value 的元素
lpush(key, value)：在名称为key的list 头添加一个值为value 的 元素
llen(key)：返回名称为key的list的长度
lrange(key, start, end)：返回名称为key的list中start至end 之间的元素（下标从0开始，下同）
ltrim(key, start, end)：截取名称为key 的list，保留start 至end之间的元素
lindex(key, index)：返回名称为key的list中index 位置的元素

lset(key, index, value)：给名称为key的list中index位置的元素赋值为value

lrem(key, count, value)：删除count个名称为key的list中值为 value的元素。count为0，删除所有值为value的元素，count> 0 从头至尾删除count个值为 value 的元素，count< 0从尾到头删除|count|个值为value的元素。

lpop(key)：返回并删除名称为key的list中的首元素

rpop(key)：返回并删除名称为key的list中的尾元素

blpop(key1, key2,… key N, timeout)：lpop 命令的block版本.即当timeout为0时,若遇到名称为key i的list不存在或该list为空，则命令结束。如果 timeout>0,则遇到上述情况时,等待timeout秒，如果问题没有解决，则对key i+1 开始的list 执行pop操作。

brpop(key1, key2,… key N, timeout)：rpop的block版本。参考上一命令。

rpoplpush(srckey, dstkey)：返回并删除名称为srckey的list 的尾元素，并将该元素添加到名称为dstkey的list 的头部

4.对Set操作的命令

sadd(key, member)：向名称为key的set中添加元素member

srem(key, member) ：删除名称为key的set中的元素member

spop(key) ：随机返回并删除名称为key的set 中一个元素

smove(srckey, dstkey, member) ：将 member元素从名称为srckey的集合移到名称为dstkey的集合

scard(key) ：返回名称为key的set的基数

sismember(key, member) ：测试member是否是名称为key的set 的元素

sinter(key1, key2,…key N) ：求交集

sinterstore(dstkey, key1, key2,…key N) ：求交集并将交集保存到dstkey的集合

sunion(key1, key2,…key N) ：求并集

sunionstore(dstkey, key1, key2,…key N) ：求并集并将并集保存到dstkey的集合

sdiff(key1, key2,…key N) ：求差集

sdiffstore(dstkey, key1, key2,…key N) ：求差集并将差集保存到dstkey 的集合

smembers(key) ：返回名称为key的set 的所有元素

srandmember(key) ：随机返回名称为key的set的一个元素

5.对zset（sorted set）操作的命令

zadd(key, score, member)：向名称为key的zset 中添加元素member，score用于排序。如果该元素已经存在，则根据 score更新该元素的顺序。

zrem(key, member) ：删除名称为key的zset中的元素 member

zincrby(key, increment, member) ：如果在名称为key的zset中已经存在元素member，则该元素的score增加increment；否则向集合中添加该元素，其score 的值为increment

zrank(key, member) ：返回名称为key 的zset（元素已按score从小到大排序）中 member元素的rank（即index，从0开始），若没有member元素，返回“nil”

zrevrank(key, member) ：返回名称为key的zset（元素已按score从大到小排序）中member元素的rank（即 index，从 0开始），若没有member 元素，返回“nil”

zrange(key, start, end)：返回名称为key的zset（元素已按score从小到大排序）中的index从start到end的所有元素

zrevrange(key, start, end)：返回名称为key的zset（元素已按score从大到小排序）中的 index从start到end的所有元素

zrangebyscore(key, min, max)：返回名称为key 的zset中score >= min且score <= max 的所有元素

zcard(key)：返回名称为key的zset的基数

zscore(key, element)：返回名称为key的zset中元素 element的score

zremrangebyrank(key, min, max)：删除名称为key的zset中rank >= min且rank <= max 的所有元素

zremrangebyscore(key, min, max) ：删除名称为key的zset中score >= min且score <= max 的所有元素

zunionstore / zinterstore(dstkeyN, key1,…,keyN, WEIGHTS w1,… wN, AGGREGATE SUM|MIN|MAX)：对N个zset求并集和交集，并将最后的集合保存在dstkeyN 中。对于集合中每一个元素的score，在进行AGGREGATE运算前，都要乘以对于的WEIGHT 参数。如果没有提供WEIGHT，默认为1。默认的 AGGREGATE是SUM，即结果集合中元素的score是所有集合对应元素进行 SUM 运算的值，而MIN和MAX是指，结果集合中元素的score 是所有集合对应元素中最小值和最大值。

6.对 Hash 操作的命令

hset(key, field, value)：向名称为key的hash中添加元素field<—>value

hget(key, field)：返回名称为key的hash中field对应的 value

hmget(key, field1, …,field N)：返回名称为key的hash中field i对应的value

hmset(key, field1, value1,…,field N, value N):向名称为key的hash中添加元素field i<—>value i

hincrby(key, field, integer)：将名称为key的hash中 field的value增加integer

hexists(key, field)：名称为key的hash中是否存在键为field的域 hdel(key, field)：删除名称为key的hash 中键为field 的域

hlen(key)：返回名称为key的hash中元素个数

hkeys(key)：返回名称为key的hash中所有键

hvals(key)：返回名称为key的hash中所有键对应的value

hgetall(key)：返回名称为key 的hash中所有的键（field）及其对应的value

五、Redis命名规则

由于Redis所有数据为键值对，即所有数据均只能通过键值（Key）来进行管理，故需要规范命名规则，jedis客户端包装了有专门的命名规则生产函数，调用即可！代码参考实例代码：

六、参考资料

官方API：http://redisdoc.com