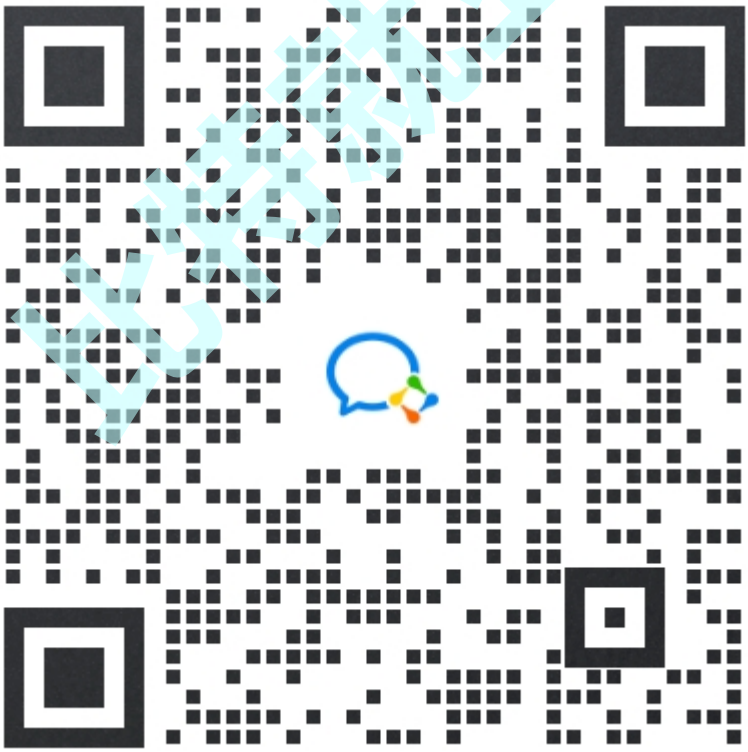
Redis Java使用 样例列表

版权说明

对比特课程感兴趣，可以联系这个微信。



代码 & 板书链接

https://gitee.com/HGtz2222/bitproject/tree/master/redis

引入依赖

Java 操作 redis 的客户端有很多. 其中最知名的是 jedis.

创建 maven 项目, 把 jedis 的依赖拷贝到 pom.xml 中.

```
<!-- https://mvnrepository.com/artifact/redis.clients/jedis -->
<dependency>
    <groupId>redis.clients</groupId>
    <artifactId>jedis</artifactId>
    <version>4.3.2</version>
</dependency>
```

> 版本选择一个相对较新的版本即可.

配置端口转发

Redis 服务器安装在云服务器上, 而我们编写的代码则是在本地主机.

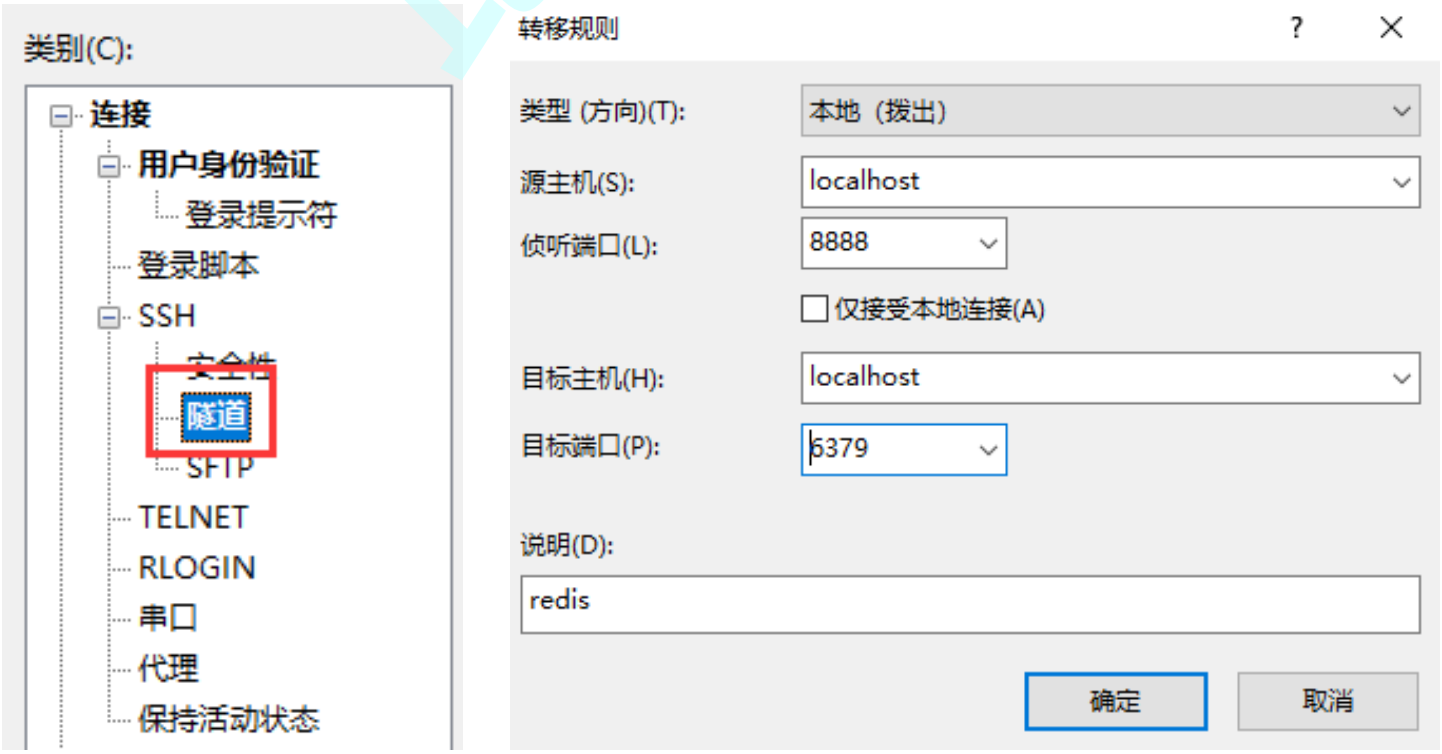
要想让本地主机能访问 redis, 需要把 redis 的端口通过云服务器后台页面的 "防火墙" / "安全组" 放开端口到公网上. 但是这个操作非常危险(黑客会顺着 redis 端口进来).

因此我们可以使用端口转发的方式, 直接把服务器的 redis 端口映射到本地.

在 xshell 中, 进行如下配置:

1) 右键云服务器的会话, 选择属性.

2) 找到隧道 -> 配置转移规则.



3) 使用该会话连接服务器.

此时, 访问本地的 8888, 就相当于访问对应服务器的 6379

> 注意, xshell 和服务器必须处在连接状态, 这样的映射才是有效的.

连接 Redis Server

- 使用 JedisPool 描述 Redis 服务器的位置. 使用 url 来表示.
- 使用 `getResource` 和服务器建立连接.
- 连接使用完毕需要 close 关闭. 也可以直接使用 try 自动关闭.
- 通过 ping 方法可以检测连接是否正确建立.

```
1 JedisPool jedisPool = new JedisPool("tcp://127.0.0.1:8888");
2 try (Jedis jedis = jedisPool.getResource()) {
3     String ping = jedis.ping();
4     System.out.println(ping);
5 }
```

执行结果

```
1 PONG
```

> **注意:**
>
> ```
> SLF4J: Failed to load class "org.slf4j.impl.StaticLoggerBinder".
> SLF4J: Defaulting to no-operation (NOP) logger implementation
> SLF4J: See http://www.slf4j.org/codes.html#StaticLoggerBinder for further details.
> ```
>
> 这个报错没有关系, 不影响使用, 忽略即可.

基础操作

set 和 get

- key 不存在时, 得到的 value 为 null.

```
public static void main(String[] args) {
    JedisPool jedisPool = new JedisPool("tcp://127.0.0.1:8888");
    try (Jedis jedis = jedisPool.getResource()) {
        // testPing(jedis);
        testGetSet(jedis);
        jedis.flushDB();
    }
}

private static void testGetSet(Jedis jedis) {
    jedis.set("key1", "value1");
    jedis.set("key2", "value2");

    String value1 = jedis.get("key1");
    System.out.println(value1);

    String value2 = jedis.get("key2");
    System.out.println(value2);

    String valueNull = jedis.get("noSuchKey");
    System.out.println(valueNull);
}
```

执行结果

```
value1
value2
null
```


exists 和 del

- del 可以删除多个 key, 以变长参数列表的方式体现. 返回值是实际删除的 key 的个数.

```
private static void testExistsAndDel(Jedis jedis) {
    jedis.set("key1", "value");
    jedis.set("key2", "value");
    jedis.set("key3", "value");
    boolean ret = jedis.exists("key1");
    System.out.println(ret);

    long n = jedis.del("key1");
```

```
    System.out.println(n);

    ret = jedis.exists("key1");
    System.out.println(ret);

    n = jedis.del("key2", "key3");
    System.out.println(n);
    ret = jedis.exists("key2");
    System.out.println(ret);
}
```

执行结果

```
true
1
false
2
false
```

keys

```
private static void testKeys(Jedis jedis) {
    jedis.set("key1", "value1");
    jedis.set("key2", "value2");
    jedis.set("key3", "value3");
    jedis.set("myKey", "value4");

    Set<String> keys = jedis.keys("*");
    System.out.println(keys);

    keys = jedis.keys("key?");
    System.out.println(keys);
}
```

执行结果

```
[key1, key2, key3, myKey]
[key1, key2, key3]
```

expire 和 ttl

- 使用 setex 直接设置 key 的同时指定过期时间. 单位为秒.

```
private static void testExpireAndTTL(Jedis jedis) {
    jedis.setex("key", 60, "value");
    long ttl = jedis.ttl("key");
    System.out.println(ttl);
}
```

执行结果

```
60
```

type

```
private static void testType(Jedis jedis) {
    jedis.set("key1", "value");
    System.out.println(jedis.type("key1"));

    jedis.lpush("key2", "a", "b", "c");
    System.out.println(jedis.type("key2"));

    jedis.hset("key3", "name", "zhangsan");
    System.out.println(jedis.type("key3"));

    jedis.sadd("key4", "111", "222", "333");
    System.out.println(jedis.type("key4"));

    jedis.zadd("key5", 1, "aaa");
    System.out.println(jedis.type("key5"));
}
```

执行结果

```
string
```

```
list
hash
set
zset
```

字符串操作

`mget` 和 `mset`

```
private static void testMSetAndMGet(Jedis jedis) {
    jedis.mset("key1", "value1", "key2", "value2", "key3", "value3");
    List<String> values = jedis.mget("key1", "key2", "key3");
    System.out.println(values);
}
```

执行结果

```
[value1, value2, value3]
```

`append`

```
private static void testAppend(Jedis jedis) {
    jedis.append("key", "aaa");
    String value = jedis.get("key");
    System.out.println(value);
    jedis.append("key", "bbb");
    value = jedis.get("key");
    System.out.println(value);
}
```

执行结果

```
aaa
aaabbb
```

`getrange` 和 `setrange`

- 注意 getrange 的区间是闭区间.

```
1 private static void testGetRangeAndSetRange(Jedis jedis) {
2     jedis.set("key", "abcdefg");
3     String value = jedis.getrange("key", 1, 4);
4     System.out.println(value);
5 
6     jedis.setrange("key", 0, "xyz");
7     value = jedis.get("key");
8     System.out.println(value);
9 }
```

执行结果

```
1 bcde
2 xyzdefg
```

`setnx`

```
1 private static void testSetnx(Jedis jedis) {
2     long n = jedis.setnx("key", "value");
3     System.out.println(n);
4     String value = jedis.get("key");
5     System.out.println(value);
6 
7     n = jedis.setnx("key", "value2");
8     System.out.println(n);
9     value = jedis.get("key");
10    System.out.println(value);
11 }
```

执行结果

```
1
value
0
value
```

`psetex`

- 获取到的结果不一定刚好 1000. `pttl` 本身也是有时间开销的.

```
private static void testPsetexAndPttl(Jedis jedis) {
    jedis.psetex("key", 1000, "value");
    long ttl = jedis.pttl("key");
    System.out.println(ttl);
}
```

执行结果

```
998
```

`incr` 和 `decr`

```
private static void testIncrAndDecr(Jedis jedis) {
    jedis.set("key", "0");
    jedis.incr("key");
    System.out.println(jedis.get("key"));

    jedis.decr("key");
    System.out.println(jedis.get("key"));
}
```

执行结果

```
1
0
```

`incrby` 和 `decrby`

```
1 private static void testIncrByAndDecrBy(Jedis jedis) {
2     jedis.set("key", "0");
3     jedis.incrBy("key", 10);
4     System.out.println(jedis.get("key"));
5 
6     jedis.decrBy("key", 5);
7     System.out.println(jedis.get("key"));
8 }
```

执行结果

```
1 10
2 5
```

列表操作

`lpush` 和 `lpop`

```
1  private static void testLpushAndLpop(Jedis jedis) {
2      long n = jedis.lpush("key", "1", "2", "3", "4");
3      System.out.println(n);
4  
5      String value = jedis.lpop("key");
6      System.out.println(value);
7  
8      value = jedis.lpop("key");
9      System.out.println(value);
10 
11     value = jedis.lpop("key");
12     System.out.println(value);
13 
14     value = jedis.lpop("key");
15     System.out.println(value);
16 
```

```
    value = jedis.lpop("key");
    System.out.println(value);
}
```

执行结果

```
4
4
3
2
1
null
```

`rpush` 和 `rpop`

```
private static void testRpushAndRpop(Jedis jedis) {
    long n = jedis.rpush("key", "1", "2", "3", "4");
    System.out.println(n);

    String value = jedis.rpop("key");
    System.out.println(value);

    value = jedis.rpop("key");
    System.out.println(value);

    value = jedis.rpop("key");
    System.out.println(value);

    value = jedis.rpop("key");
    System.out.println(value);

    value = jedis.rpop("key");
    System.out.println(value);
}
```

执行结果

```
4
4
```

```
3
2
1
null
```

lrange

- `lrange` 填写的区间为闭区间.

```java
private static void testLrange(Jedis jedis) {
    jedis.rpush("key", "1", "2", "3", "4");
    List<String> values = jedis.lrange("key", 1, 3);
    System.out.println(values);
}
```

执行结果

```
[2, 3, 4]
```

blpop

- 返回值 List 是个二元组. [0] 表示 key, [1] 表示 value
- 超时时间设为 0 表示死等.
- 在执行同时, 起一个 redis-cli, 插入数据, 即可看到 blpop 的返回结果.

> **注意:** 在代码中另起一个线程, 直接通过当前 jedis 这个连接插入数据是不行的. 必须另起一个 jedis 连接.

```java
private static void testBLpop(Jedis jedis) {
    while (true) {
        List<String> values = jedis.blpop(0, "key");
        System.out.println(values);
    }
}
```

执行结果

```
# 客户端执行 rpush key 1 2 3

[key, 1]
[key, 2]
[key, 3]
```

brpop

- 使用方式和 blpop 类似

```
private static void testBRpop(Jedis jedis) {
    System.out.println("开始调用 brpop");
    while (true) {
        List<String> values = jedis.brpop(0, "key");
        System.out.println(values);
    }
}
```

执行结果

```
[key, 3]
[key, 2]
[key, 1]
```

lindex

```
private static void testLindex(Jedis jedis) {
    jedis.rpush("key", "1", "2", "3", "4");
    String value = jedis.lindex("key", 2);
    System.out.println(value);
}
```

执行结果

```
1 3
```

linsert

- 通过 `ListPosition.BEFORE` 和 `ListPosition.AFTER` 标识插入位置.

```
1 private static void testLinsert(Jedis jedis) {
2     jedis.rpush("key", "a", "b", "c", "d");
3     jedis.linsert("key", ListPosition.BEFORE, "c", "100");
4     List<String> values = jedis.lrange("key", 0, -1);
5     System.out.println(values);
6 }
```

执行结果

```
1 [a, b, 100, c, d]
```

llen

```
1 private static void testLlen(Jedis jedis) {
2     jedis.rpush("key", "a", "b", "c", "d");
3     long n = jedis.llen("key");
4     System.out.println(n);
5 }
```

执行结果

```
1 4
```

哈希表操作

`hset` 和 `hget`

```
1 private static void testHsetAndHget(Jedis jedis) {
2     jedis.hset("key", "name", "zhangsan");
3     jedis.hset("key", "age", "20");
4     String name = jedis.hget("key", "name");
5     System.out.println(name);
6     String age = jedis.hget("key", "age");
7     System.out.println(age);
8 }
```

执行结果

```
1 zhangsan
2 20
```

`hexists` 和 `hdel`

```
1 private static void testHexistsAndHdel(Jedis jedis) {
2     jedis.hset("key", "name", "zhangsan");
3     boolean ok = jedis.hexists("key", "name");
4     System.out.println(ok);
5 
6     jedis.hdel("key", "name");
7     ok = jedis.hexists("key", "name");
8     System.out.println(ok);
9 }
```

执行结果

```
1 true
2 false
```

`hkeys` 和 `hvals`

```
1  private static void testHkeysAndHvalues(Jedis jedis) {
2      jedis.hset("key", "name", "zhangsan");
3      jedis.hset("key", "age", "20");
4
5      Set<String> keys = jedis.hkeys("key");
6      System.out.println(keys);
7
8      List<String> values = jedis.hvals("key");
9      System.out.println(values);
10 }
```

执行结果

```
1 [name, age]
2 [zhangsan, 20]
```

`hmget`

```
1 private static void testHmget(Jedis jedis) {
2     jedis.hset("key", "name", "zhangsan");
3     jedis.hset("key", "age", "20");
4     List<String> values = jedis.hmget("key", "name", "age");
5     System.out.println(values);
6 }
```

执行结果

```
1 [zhangsan, 20]
```

`hlen`

```java
private static void testHlen(Jedis jedis) {
    jedis.hset("key", "name", "zhangsan");
    jedis.hset("key", "age", "20");
    long n = jedis.hlen("key");
    System.out.println(n);
}
```

执行结果

```
2
```

`hincrby` 和 `hincrbyfloat`

```java
private static void testHIncrByAndIncrByFloat(Jedis jedis) {
    jedis.hset("key", "age", "20");
    long n = jedis.hincrBy("key", "age", 10);
    System.out.println(n);
    String value = jedis.hget("key", "age");
    System.out.println(value);

    double dn = jedis.hincrByFloat("key", "age", 0.5);
    System.out.println(dn);
    value = jedis.hget("key", "age");
    System.out.println(value);
}
```

执行结果

```
30
30
30.5
30.5
```

集合操作

`sadd` 和 `smembers`

```
1 private static void testSaddAndSmembers(Jedis jedis) {
2     jedis.sadd("key", "aaa", "bbb", "ccc");
3     Set<String> members = jedis.smembers("key");
4     System.out.println(members);
5 }
```

执行结果

```
1 [aaa, ccc, bbb]
```

`srem` 和 `sismember`

```
1  private static void testSremAndSismember(Jedis jedis) {
2      jedis.sadd("key", "aaa", "bbb", "ccc");
3      boolean ok = jedis.sismember("key", "aaa");
4      System.out.println(ok);
5      long n = jedis.srem("key", "aaa", "bbb");
6      System.out.println(n);
7
8      ok = jedis.sismember("key", "aaa");
9      System.out.println(ok);
10 }
```

执行结果

```
1 true
2 2
3 false
```

`scard`

```java
private static void testScard(Jedis jedis) {
    jedis.sadd("key", "aaa", "bbb", "ccc");
    long n = jedis.scard("key");
    System.out.println(n);
}
```

执行结果

```
3
```

sinter

```java
private static void testSinter(Jedis jedis) {
    jedis.sadd("key1", "aaa", "bbb", "ccc");
    jedis.sadd("key2", "aaa", "bbb", "ddd");

    Set<String> results = jedis.sinter("key1", "key2");
    System.out.println(results);
}
```

执行结果

```
[aaa, bbb]
```

sunion

```java
private static void testSunion(Jedis jedis) {
    jedis.sadd("key1", "aaa", "bbb", "ccc");
    jedis.sadd("key2", "aaa", "bbb", "ddd");

    Set<String> results = jedis.sunion("key1", "key2");
    System.out.println(results);
}
```

执行结果

```
[aaa, ccc, bbb, ddd]
```

`sdiff`

```
private static void testSdiff(Jedis jedis) {
    jedis.sadd("key1", "aaa", "bbb", "ccc");
    jedis.sadd("key2", "aaa", "bbb", "ddd");

    Set<String> results = jedis.sdiff("key1", "key2");
    System.out.println(results);
}
```

执行结果

```
[ccc]
```

有序集合操作

`zadd` 和 `zrange`

- `zrange` 通过下标获取元素. 闭区间.

```
private static void testZaddAndZrange(Jedis jedis) {
    jedis.zadd("key", 100, "吕布");
    jedis.zadd("key", 98, "赵云");
    jedis.zadd("key", 95, "典韦");
    jedis.zadd("key", 92, "关羽");
    jedis.zadd("key", 70, "刘备");

    List<String> members = jedis.zrange("key", 0, 4);
    System.out.println(members);


```

```
    List<Tuple> membersWithScore = jedis.zrangeWithScores("key", 0, 4);
    System.out.println(membersWithScore);
}
```

执行结果

```
[刘备, 关羽, 典韦, 赵云, 吕布]
[[刘备,70.0], [关羽,92.0], [典韦,95.0], [赵云,98.0], [吕布,100.0]]
```

zrem 和 zcard

```
private static void testZremAndZcard(Jedis jedis) {
    jedis.zadd("key", 100, "吕布");
    jedis.zadd("key", 98, "赵云");
    jedis.zadd("key", 95, "典韦");
    jedis.zadd("key", 92, "关羽");
    jedis.zadd("key", 70, "刘备");

    long n = jedis.zcard("key");
    System.out.println(n);

    n = jedis.zrem("key", "吕布", "赵云");
    System.out.println(n);
    n = jedis.zcard("key");
    System.out.println(n);
}
```

执行结果

```
5
2
3
```

zcount

- 获取指定分数区间中的元素个数. 闭区间.

```java
private static void testZcount(Jedis jedis) {
    jedis.zadd("key", 100, "吕布");
    jedis.zadd("key", 98, "赵云");
    jedis.zadd("key", 95, "典韦");
    jedis.zadd("key", 92, "关羽");
    jedis.zadd("key", 70, "刘备");

    long n = jedis.zcount("key", 92, 98);
    System.out.println(n);
}
```

执行结果

```
3
```

`zpopmax` 和 `zpopmin`

```java
private static void testZpopmaxAndZpopmin(Jedis jedis) {
    jedis.zadd("key", 100, "吕布");
    jedis.zadd("key", 98, "赵云");
    jedis.zadd("key", 95, "典韦");
    jedis.zadd("key", 92, "关羽");
    jedis.zadd("key", 70, "刘备");

    Tuple tuple = jedis.zpopmax("key");
    System.out.println(tuple);

    tuple = jedis.zpopmin("key");
    System.out.println(tuple);
}
```

执行结果

```
[吕布,100.0]
[刘备,70.0]
```

zrank

- 获取指定 member 的下标.

```
1  private static void testZrank(Jedis jedis) {
2      jedis.zadd("key", 100, "吕布");
3      jedis.zadd("key", 98, "赵云");
4      jedis.zadd("key", 95, "典韦");
5      jedis.zadd("key", 92, "关羽");
6      jedis.zadd("key", 70, "刘备");
7
8      long n = jedis.zrank("key", "赵云");
9      System.out.println(n);
10
11     n = jedis.zrevrank("key", "赵云");
12     System.out.println(n);
13 }
```

执行结果

```
1 3
2 1
```


zscore

```
1  private static void testZscore(Jedis jedis) {
2      jedis.zadd("key", 100, "吕布");
3      jedis.zadd("key", 98, "赵云");
4      jedis.zadd("key", 95, "典韦");
5      jedis.zadd("key", 92, "关羽");
6      jedis.zadd("key", 70, "刘备");
7
8      double score = jedis.zscore("key", "赵云");
9      System.out.println(score);
10 }
```

执行结果

```
98.0
```

zincrby

```
private static void testZincrby(Jedis jedis) {
    jedis.zadd("key", 100, "吕布");

    double n = jedis.zincrby("key", 10, "吕布");
    System.out.println(n);

    n = jedis.zincrby("key", -20, "吕布");
    System.out.println(n);
}
```

执行结果

```
110.0
90.0
```

zinterstore

```
private static void testZinterstore(Jedis jedis) {
    jedis.zadd("key1", 100, "吕布");
    jedis.zadd("key1", 98, "赵云");
    jedis.zadd("key1", 95, "典韦");

    jedis.zadd("key2", 100, "吕布");
    jedis.zadd("key2", 98, "赵云");
    jedis.zadd("key2", 92, "关羽");

    long n = jedis.zinterstore("key3", "key1", "key2");
    System.out.println(n);
    List<Tuple> tuples = jedis.zrangeWithScores("key3", 0, -1);
    System.out.println(tuples);
}
```

执行结果

```
2
[[赵云,196.0], [吕布,200.0]]
```

zunionstore

```
private static void testZunionstore(Jedis jedis) {
    jedis.zadd("key1", 100, "吕布");
    jedis.zadd("key1", 98, "赵云");
    jedis.zadd("key1", 95, "典韦");

    jedis.zadd("key2", 100, "吕布");
    jedis.zadd("key2", 98, "赵云");
    jedis.zadd("key2", 92, "关羽");

    long n = jedis.zunionstore("key3", "key1", "key2");
    System.out.println(n);
    List<Tuple> tuples = jedis.zrangeWithScores("key3", 0, -1);
    System.out.println(tuples);
}
```

执行结果

```
4
[[关羽,92.0], [典韦,95.0], [赵云,196.0], [吕布,200.0]]
```


访问集群

使用 `JedisCluster` 类代替 `Jedis` 类即可.

需要在创建实例的时候, 把多个节点的地址, 都设置进去.

`JedisCluster` 提供的方法和 `Jedis` 基本一致. 都和 Redis 命令是对应的. 具体细节我们不再演示了.

```java
public static void main(String[] args) {
    Set<HostAndPort> nodes = new HashSet<>();
    nodes.add(new HostAndPort("172.30.0.101", 6379));
    nodes.add(new HostAndPort("172.30.0.102", 6379));
    nodes.add(new HostAndPort("172.30.0.103", 6379));
    nodes.add(new HostAndPort("172.30.0.104", 6379));
    nodes.add(new HostAndPort("172.30.0.105", 6379));
    nodes.add(new HostAndPort("172.30.0.106", 6379));
    nodes.add(new HostAndPort("172.30.0.107", 6379));
    nodes.add(new HostAndPort("172.30.0.108", 6379));
    nodes.add(new HostAndPort("172.30.0.109", 6379));

    try (JedisCluster jedisCluster = new JedisCluster(nodes)) {
        jedisCluster.set("k1", "111");
        String value = jedisCluster.get("k1");
        System.out.println(value);
    }
}
```

注意! 由于此处我们的代码是需要访问整个 redis 集群, 因此不能直接在 windows 上运行程序了. (上述 docker 容器的 ip 都是 windows 主机上无法直接访问的).

需要把整个程序打成 jar 包, 上传到 Linux 中, 并通过下列方式运行.

```
java -jar [jar包名]
```