Redis 基础命令 @CreateByShadow

> 1. SELECT dbIndex (0-15)
>
> 切换数据库
>
> 2. DEL key
>
> key存在时删除key
>
> 3. DUMP key
>
> 序列化给定key，并返回被序列化的值
>
> 4. EXISTS key
>
> 检查给定的key是否存在
>
> 5. EXPIRE key seconds
>
> 给指定key设置过期时间，以秒计
>
> 6. MOVE key dbIndex
>
> 将当前数据库的key移动到给定的数据库db当中
>
> 7. RENAME key newKey
>
> 修改key的名称
>
> 8. RENAMENX key newKey
>
> 仅当newKey不存在时，将key改名为newKey
>
> 9. TYPE key
>
> 返回key所存储的值的类型

Redis 连接命令

序号	命令及描述
1	AUTH password 验证密码是否正确
2	ECHO message 打印字符串
3	PING 查看服务是否运行
4	QUIT 关闭当前连接
5	SELECT index 切换到指定的数据库

Redis 数据类型

> Redis支持五种数据类型：string（字符串）、hash（哈希）、list（列表）、set（集合）及zset（sorted set：有序集合）

1、String（字符串）

> - string是Redis最基本的类型，一个key对应一个value
> - **string类型是二进制安全的**，即redis的字符串可以包含任何数据，如jpg图片或者序列化的对象
> - **string类型的值最大能存储512MB**

2、Hash（哈希）

> - Redis hash是一个键值（key=>value）对集合
> - Redis hash是一个String类型的field和value的映射表，**hash特别适用于存储对象**
> - 每个hash可以存储2^32 -1 键值对（4294967295, 每个hash可存储40多亿）

3、List（列表）

> - Redis列表是简单的字符串列表，按照插入顺序排序。可以添加一个元素到列表的**头部（左边）或者尾部（右边）**
> - 列表最多可以存储2^32 -1元素（4294967295, 每个列表可存储40多亿）

4、Set（集合）

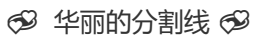
> - Redis的**Set是String类型的无序集合，不允许重复成员**，集合是通过哈希表实现的，所以添加、删除、查找的复杂度都是O（1）
> - 集合内**元素是唯一的**，已存在的元素再次插入将被忽略
> - 集合中最大的成员数为2^32 -1（4294967295, 每个集合可存储40多亿个成员）

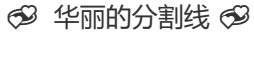
5、Zset（sorted set：有序集合）

> - Redis Zset和set一样也是String类型元素的集合，且不允许重复的成员，不同的是每个元素都会关联一个double类型的分数；Redis正是通过分数来为集合中的成员进行从小到大的排序
> - Zset的成员是唯一的，但分数（score）却可以重复

五大数据类型命令使用介绍

> 链接远程redis服务命令：redis-cli -h host -p port -a password





1、Redis 字符串（String）

常用命令：

> 1. SET key value
>
> 设置指定key的值
>
> 2. GET key
>
> 获取指定key的值

> 3. APPEND key value
>
> 如果key已经存在并且是一个字符串，APPEND命令将指定的value追加到该key原来值的末尾

命令列表：

序号	命令及描述
1	SET key value 设置指定 key 的值
2	GET key 获取指定 key 的值。
3	GETRANGE key start end 返回 key 中字符串值的子字符
4	GETSET key value 将给定 key 的值设为 value ，并返回 key 的旧值(old value)。
5	GETBIT key offset 对 key 所储存的字符串值，获取指定偏移量上的位(bit)。
6	MGET key1 [key2..] 获取所有(一个或多个)给定 key 的值。
7	SETBIT key offset value 对 key 所储存的字符串值，设置或清除指定偏移量上的位(bit)。
8	SETEX key seconds value 将值 value 关联到 key ，并将 key 的过期时间设为 seconds (以秒为单位)。
9	SETNX key value 只有在 key 不存在时设置 key 的值。
10	SETRANGE key offset value 用 value 参数覆写给定 key 所储存的字符串值，从偏移量 offset 开始。
11	STRLEN key 返回 key 所储存的字符串值的长度。
12	MSET key value [key value ...] 同时设置一个或多个 key-value 对。

13	MSETNX key value [key value ...] 同时设置一个或多个 key-value 对，当且仅当所有给定 key 都不存在。
14	PSETEX key milliseconds value 这个命令和 SETEX 命令相似，但它以毫秒为单位设置 key 的生存时间，而不是像 SETEX 命令那样，以秒为单位。
15	INCR key 将 key 中储存的数字值增一。
16	INCRBY key increment 将 key 所储存的值加上给定的增量值（increment） 。
17	INCRBYFLOAT key increment 将 key 所储存的值加上给定的浮点增量值（increment） 。
18	DECR key 将 key 中储存的数字值减一。
19	DECRBY key decrement key 所储存的值减去给定的减量值（decrement） 。
20	APPEND key value 如果 key 已经存在并且是一个字符串， APPEND 命令将指定的 value 追加到该 key 原来值（value）的末尾。

2、Redis 哈希（Hash）

常用命令：

> 1. HSET key field value
>
> 将哈希表key中的字段field的值设置为value，可以同时多个field设置 HMSET key field value field1 value1...
>
> 2. HGET key field
>
> 获取存储在哈希表中指定字段的值
>
> 3. HMGET key field1 [field2..]
>
> 获取所有给定字段的值
>
> 4. HGETALL key
>
> 获取在哈希表中指定key的所有字段和值
>
> 5. HLEN key
>
> 获取哈希表中字段的数量
>
> 6. HDEL key field1 [field2]
>
> 删除一个或多个哈希表字段
>
> 7. HVALS key
>
> 获取哈希表中所有值

```
8. HKEYS key
```

> 获取哈希表中的所有字段

命令列表：

序号	命令及描述
1	HDEL key field1 [field2] 删除一个或多个哈希表字段
2	HEXISTS key field 查看哈希表 key 中，指定的字段是否存在。
3	HGET key field 获取存储在哈希表中指定字段的值。
4	HGETALL key 获取在哈希表中指定 key 的所有字段和值
5	HINCRBY key field increment 为哈希表 key 中的指定字段的整数值加上增量 increment 。
6	HINCRBYFLOAT key field increment 为哈希表 key 中的指定字段的浮点数值加上增量 increment 。
7	HKEYS key 获取所有哈希表中的字段
8	HLEN key 获取哈希表中字段的数量
9	HMGET key field1 [field2] 获取所有给定字段的值
10	HMSET key field1 value1 [field2 value2] 同时将多个 field-value (域-值)对设置到哈希表 key 中。
11	HSET key field value 将哈希表 key 中的字段 field 的值设为 value 。
12	HSETNX key field value 只有在字段 field 不存在时，设置哈希表字段的值。
13	HVALS key 获取哈希表中所有值
14	HSCAN key cursor [MATCH pattern] [COUNT count] 迭代哈希表中的键值对。

Redis 列表（List）【左边为头部，右边为尾部】

常用命令：

> 1. LPUSH key value1 [value2]
>
> 将一个或多个值插入到列表头部
>
> 2. RPUSH key value1 [value2]
>
> 在列表中添加一个或多个值
>
> 3. LPOP key
>
> 移除并获取列表的第一个元素
>
> 4. RPOP key
>
> 移除并获取列表的最后一个元素
>
> 5. LLEN key

> 获取列表的长度
>
> 6. LSET key index value
>
> 通过索引设置列表元素的值

命令列表：

序号	命令及描述
1	BLPOP key1 [key2] timeout 移出并获取列表的第一个元素， 如果列表没有元素会阻塞列表直到等待超时或发现可弹出元素为止。
2	BRPOP key1 [key2] timeout 移出并获取列表的最后一个元素， 如果列表没有元素会阻塞列表直到等待超时或发现可弹出元素为止。
3	BRPOPLPUSH source destination timeout 从列表中弹出一个值，将弹出的元素插入到另外一个列表中并返回它； 如果列表没有元素会阻塞列表直到等待超时或发现可弹出元素为止。
4	LINDEX key index 通过索引获取列表中的元素
5	LINSERT key BEFORE|AFTER pivot value 在列表的元素前或者后插入元素
6	LLEN key 获取列表长度
7	LPOP key 移出并获取列表的第一个元素
8	LPUSH key value1 [value2] 将一个或多个值插入到列表头部
9	LPUSHX key value 将一个值插入到已存在的列表头部
10	LRANGE key start stop 获取列表指定范围内的元素
11	LREM key count value 移除列表元素
12	LSET key index value 通过索引设置列表元素的值
13	LTRIM key start stop 对一个列表进行修剪(trim)，就是说，让列表只保留指定区间内的元素，不在指定区间之内的元素都将被删除。
14	RPOP key 移除列表的最后一个元素，返回值为移除的元素。
15	RPOPLPUSH source destination 移除列表的最后一个元素，并将该元素添加到另一个列表并返回
16	RPUSH key value1 [value2] 在列表中添加一个或多个值
17	RPUSHX key value 为已存在的列表添加值

Redis 集合（Set）

常用命令：

> 1. SADD key member1 [member2]
>
> 向集合添加一个或多个成员
>
> 2. SMEMBERS key
>
> 返回集合中的所有成员
>
> 3. SPOP key
>
> 移除并返回集合中的一个随机元素

命令列表：

序号	命令及描述
1	SADD key member1 [member2] 向集合添加一个或多个成员
2	SCARD key 获取集合的成员数
3	SDIFF key1 [key2] 返回给定所有集合的差集
4	SDIFFSTORE destination key1 [key2] 返回给定所有集合的差集并存储在 destination 中
5	SINTER key1 [key2] 返回给定所有集合的交集
6	SINTERSTORE destination key1 [key2] 返回给定所有集合的交集并存储在 destination 中
7	SISMEMBER key member 判断 member 元素是否是集合 key 的成员
8	SMEMBERS key 返回集合中的所有成员
9	SMOVE source destination member 将 member 元素从 source 集合移动到 destination 集合
10	SPOP key 移除并返回集合中的一个随机元素
11	SRANDMEMBER key [count] 返回集合中一个或多个随机数
12	SREM key member1 [member2] 移除集合中一个或多个成员
13	SUNION key1 [key2] 返回所有给定集合的并集
14	SUNIONSTORE destination key1 [key2] 所有给定集合的并集存储在 destination 集合中
15	SSCAN key cursor [MATCH pattern] [COUNT count] 迭代集合中的元素

Redis 有序集合（sorted set）

常用命令：

> 1. ZADD key score1 member1 [score2 member2]
>
> 向有序集合添加一个或多个成员，或者更新已存在成员的分数（score）
>
> 2. ZCARD key
>
> 获取有序集合的成员数

命令列表：

序号	命令及描述
1	ZADD key score1 member1 [score2 member2] 向有序集合添加一个或多个成员，或者更新已存在成员的分数
2	ZCARD key 获取有序集合的成员数
3	ZCOUNT key min max 计算在有序集合中指定区间分数的成员数
4	ZINCRBY key increment member 有序集合中对指定成员的分数加上增量 increment
5	ZINTERSTORE destination numkeys key [key ...] 计算给定的一个或多个有序集的交集并将结果集存储在新的有序集合 key 中
6	ZLEXCOUNT key min max 在有序集合中计算指定字典区间内成员数量
7	ZRANGE key start stop [WITHSCORES] 通过索引区间返回有序集合成指定区间内的成员
8	ZRANGEBYLEX key min max [LIMIT offset count] 通过字典区间返回有序集合的成员
9	ZRANGEBYSCORE key min max [WITHSCORES] [LIMIT] 通过分数返回有序集合指定区间内的成员
10	ZRANK key member 返回有序集合中指定成员的索引
11	ZREM key member [member ...] 移除有序集合中的一个或多个成员
12	ZREMRANGEBYLEX key min max 移除有序集合中给定的字典区间的所有成员

13	ZREMRANGEBYRANK key start stop 移除有序集合中给定的排名区间的所有成员
14	ZREMRANGEBYSCORE key min max 移除有序集合中给定的分数区间的所有成员
15	ZREVRANGE key start stop [WITHSCORES] 返回有序集中指定区间内的成员，通过索引，分数从高到底
16	ZREVRANGEBYSCORE key max min [WITHSCORES] 返回有序集中指定分数区间内的成员，分数从高到低排序
17	ZREVRANK key member 返回有序集合中指定成员的排名，有序集成员按分数值递减(从大到小)排序
18	ZSCORE key member 返回有序集中，成员的分数值
19	ZUNIONSTORE destination numkeys key [key ...] 计算给定的一个或多个有序集的并集，并存储在新的 key 中
20	ZSCAN key cursor [MATCH pattern] [COUNT count] 迭代有序集合中的元素（包括元素成员和元素分值）

JAVA 使用Redis 的工具类：

```
public class RedisUtil {

    //服务器IP地址
    private static String ADDR = "192.168.41.65";
    //端口
    private static int PORT = 6379;
    //密码
    private static String AUTH = "123456";
    //连接实例的最大连接数
    private static int MAX_ACTIVE = 1024;
    //控制一个pool最多有多少个状态为idle(空闲的)的jedis实例，默认值也是8。
    private static int MAX_IDLE = 200;
    //等待可用连接的最大时间，单位毫秒，默认值为-1，表示永不超时。如果超过等待时间，则直接抛出
JedisConnectionException
    private static int MAX_WAIT = 10000;
    //连接超时的时间
    private static int TIMEOUT = 10000;
    // 在borrow一个jedis实例时，是否提前进行validate操作；如果为true，则得到的jedis实例均
是可用的；
    private static boolean TEST_ON_BORROW = true;

    private static JedisPool jedisPool = null;
    //数据库模式是16个数据库 0~15
    public static final int DEFAULT_DATABASE = 0;
    /**
     * 初始化Redis连接池
     */

    static {

        try {
```

```
            JedisPoolConfig config = new JedisPoolConfig();
            config.setMaxTotal(MAX_ACTIVE);
            config.setMaxIdle(MAX_IDLE);
            config.setMaxWaitMillis(MAX_WAIT);
            config.setTestOnBorrow(TEST_ON_BORROW);
            jedisPool = new JedisPool(config, ADDR, PORT,
TIMEOUT,AUTH,DEFAULT_DATABASE);

        } catch (Exception e) {

            e.printStackTrace();
        }

    }

    /**
     * 获取Jedis实例
     */

    public synchronized static Jedis getJedis() {

        try {

            if (jedisPool != null) {
                Jedis resource = jedisPool.getResource();
                System.out.println("redis--服务正在运行: "+resource.ping());
                return resource;
            } else {
                return null;
            }

        } catch (Exception e) {
            e.printStackTrace();
            return null;
        }

    }

    /***
     *
     * 释放资源
     */

    public static void returnResource(final Jedis jedis) {
            if(jedis != null) {
                jedisPool.returnResource(jedis);
            }

    }
}
```