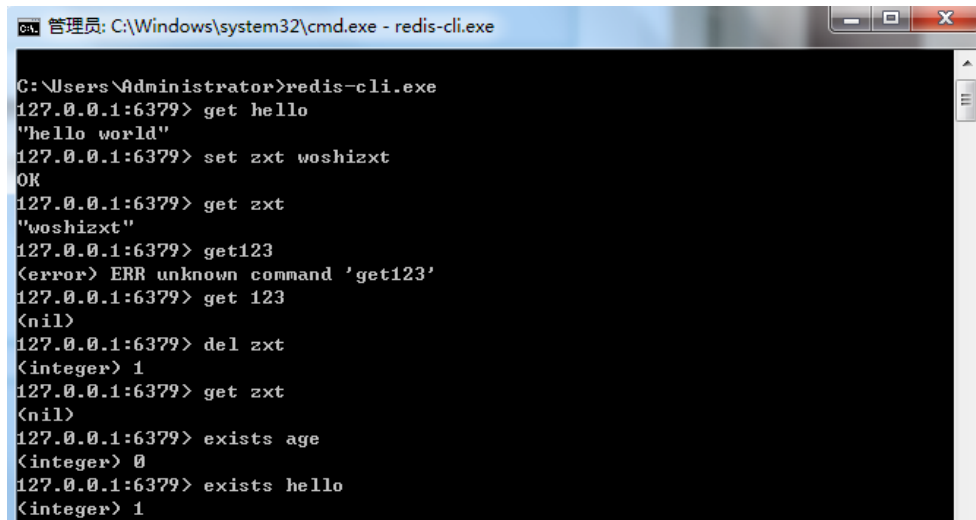


## 一、Redis 是什么

Redis 是一个开源的使用 ANSI C 语言编写、支持网络、可基于内存亦可持久化的日志型、Key-Value 数据库，并提供多种语言的 API。从 2010 年 3 月 15 日起，Redis 的开发工作由 VMware 主持。从 2013 年 5 月开始，Redis 的开发由 Pivotal 赞助。

Redis 官网只有 Linux 版本，而没有 windows 对应版本，在 Windows 系统下的安装参考：<https://blog.csdn.net/hwangfantasy/article/details/66542822>。

### 1.1、使用 redis 客户端



```
管理员: C:\Windows\system32\cmd.exe - redis-cli.exe

C:\Users\Administrator>redis-cli.exe
127.0.0.1:6379> get hello
"hello world"
127.0.0.1:6379> set zxt woshizxt
OK
127.0.0.1:6379> get zxt
"woshizxt"
127.0.0.1:6379> get123
<error> ERR unknown command 'get123'
127.0.0.1:6379> get 123
<nil>
127.0.0.1:6379> del zxt
<integer> 1
127.0.0.1:6379> get zxt
<nil>
127.0.0.1:6379> exists age
<integer> 0
127.0.0.1:6379> exists hello
<integer> 1
```

## 二、Redis 数据结构

redis 是一种高级的 key: value 存储系统，其中 value 支持五种数据类型：

- 1、字符串（strings）
- 2、字符串列表（lists）
- 3、字符串集合（sets）
- 4、有序字符串集合（sorted sets）
- 5、哈希（hashes）

而关于 key，有几个点要提醒大家：

- 1、key 不要太长，尽量不要超过 1024 字节，这不仅消耗内存，且会降低查找的效率；
- 2、key 也不要太短，太短的话，key 的可读性会降低；
- 3、在一个项目中，key 最好使用统一的命名模式，例如 user:10000:passwd。

### 2.1、strings

有人说，如果只使用 redis 中的字符串类型，且不使用 redis 的持久化功能，那么，redis 就和 memcache 非常非常的像了。这说明 strings 类型是一个很基础的数据类型，也是任何存储系统都必备的数据类型。

字符串类型的用法就是这么简单，因为是二进制安全的，所以你完全可以把一个图片文件的内容作为字符串来存储。另外，我们还可以通过字符串类型进行数值操作：在遇到数值操作时，redis 会将字符串类型转换成数值。

```
127.0.0.1:6379> set mynum 12
OK
127.0.0.1:6379> get mynum
"12"
127.0.0.1:6379> incr mynum
(integer) 13
127.0.0.1:6379> get mynum
"13"
127.0.0.1:6379> _
```

由于 INCR 等指令本身就具有原子操作的特性，所以我们完全可以利用 redis 的 INCR、INCRBY、DECR、DECRBY 等指令来实现原子计数的效果，假如，在某种场景下有 3 个客户端同时读取了 mynum 的值（值为 2），然后对其同时进行了加 1 的操作，那么，最后 mynum 的值一定是 5。不少网站都利用 redis 的这个特性来实现业务上的统计计数需求。

## 2.2、lists

redis 的另一个重要的数据结构叫做 **lists**，翻译成中文叫做“列表”。

首先要明确一点，redis 中的 **lists** 在底层实现上并不是数组，而是链表，也就是说对于一个具有上百万个元素的 **lists** 来说，在头部和尾部插入一个新元素，其时间复杂度是常数级别的，比如用 **LPUSH** 在 10 个元素的 **lists** 头部插入新元素，和在上千万元素的 **lists** 头部插入新元素的速度应该是相同的。

虽然 **lists** 有这样的优势，但同样有其弊端，那就是，链表型 **lists** 的元素定位会比较慢，而数组型 **lists** 的元素定位就会快得多。

**lists** 的常用操作包括 **LPUSH**、**RPUSH**、**LRANGE** 等。我们可以用 **LPUSH** 在 **lists** 的左侧插入一个新元素，用 **RPUSH** 在 **lists** 的右侧插入一个新元素，用 **LRANGE** 命令从 **lists** 中指定一个范围来提取元素。我们来看几个例子：

```
127.0.0.1:6379> lpush mylist 1 2
(integer) 2
127.0.0.1:6379> rpush mylist 3
(integer) 3
127.0.0.1:6379> lrange mylist 0 2
1) "2"
2) "1"
3) "3"
127.0.0.1:6379> lrange mylist 0 2
1) "2"
2) "1"
3) "3"
127.0.0.1:6379>
```

新建一个list叫做mylist，并在列表头部插入元素"1"，"2"

**lists** 的应用相当广泛，随便举几个例子：

- 1、我们可以利用 **lists** 来实现一个消息队列，而且可以确保先后顺序，不必像 MySQL 那样还需要通过 **ORDER BY** 来进行排序。
- 2、利用 **LRANGE** 还可以很方便的实现分页的功能。
- 3、在博客系统中，每篇博文的评论也可以存入一个单独的 **list** 中。

## 2.3、集合

redis 的集合，是一种无序的集合，集合中的元素没有先后顺序。集合相关的操作也很丰富，如添加新元素、删除已有元素、取交集、取并集、取差集等。

```
//向集合 myset 中加入一个新元素"one"
127.0.0.1:6379> sadd myset "one"
(integer) 1
127.0.0.1:6379> sadd myset "two"
(integer) 1
//列出集合 myset 中的所有元素
127.0.0.1:6379> smembers myset
```

```

1) "one"
2) "two"
//判断元素 1 是否在集合 myset 中, 返回 1 表示存在
127.0.0.1:6379> sismember myset "one"
(integer) 1
//判断元素 3 是否在集合 myset 中, 返回 0 表示不存在
127.0.0.1:6379> sismember myset "three"
(integer) 0
//新建一个新的集合 yourset
127.0.0.1:6379> sadd yourset "1"
(integer) 1
127.0.0.1:6379> sadd yourset "2"
(integer) 1
127.0.0.1:6379> smembers yourset
1) "1"
2) "2"
//对两个集合求并集
127.0.0.1:6379> sunion myset yourset
1) "1"
2) "one"
3) "2"
4) "two"

```

对于集合的使用，也有一些常见的方式，比如，QQ 有一个社交功能叫做“好友标签”，大家可以给你的好友贴标签，比如“大美女”、“土豪”、“欧巴”等等，这时就可以使用 redis 的集合来实现，把每一个用户的标签都存储在一个集合之中。

## 2.4、有序集合

redis 不但提供了无序集合（sets），还很体贴的提供了有序集合（sorted sets）。有序集合中的每个元素都关联一个序号（score），这便是排序的依据。

很多时候，我们都将 redis 中的有序集合叫做 zsets，这是因为在 redis 中，有序集合相关的操作指令都是以 z 开头的，比如 zrange、zadd、zrevrange、zrangebyscore 等等。老规矩，我们来看几个生动的例子：

```

//新增一个有序集合 myzset，并加入一个元素 baidu.com，给它赋予的序号是 1:
127.0.0.1:6379> zadd myzset 1 baidu.com
(integer) 1
//向 myzset 中新增一个元素 360.com，赋予它的序号是 3
127.0.0.1:6379> zadd myzset 3 360.com
(integer) 1
//向 myzset 中新增一个元素 google.com，赋予它的序号是 2

```

```

127.0.0.1:6379> zadd myzset 2 google.com
(integer) 1
//列出 myzset 的所有元素，同时列出其序号，可以看出 myzset 已经是有序的了。
127.0.0.1:6379> zrange myzset 0 -1 with scores
1) "baidu.com"
2) "1"
3) "google.com"
4) "2"
5) "360.com"
6) "3"
//只列出 myzset 的元素
127.0.0.1:6379> zrange myzset 0 -1
1) "baidu.com"
2) "google.com"
3) "360.com"

```

## 2.5、哈希

最后要给大家介绍的是 `hashes`，即哈希。哈希是从 `redis-2.0.0` 版本之后才有的数据结构。`hashes` 存的是字符串和字符串值之间的映射，比如一个用户要存储其全名、姓氏、年龄等等，就很适合使用哈希。

我们来看一个例子：

```

//建立哈希，并赋值
127.0.0.1:6379> HMSET user:001 username antirez password P1pp0 age 34
OK
//列出哈希的内容
127.0.0.1:6379> HGETALL user:001
1) "username"
2) "antirez"
3) "password"
4) "P1pp0"
5) "age"
6) "34"
//更改哈希中的某一个值
127.0.0.1:6379> HSET user:001 password 12345
(integer) 0
//再次列出哈希的内容
127.0.0.1:6379> HGETALL user:001
1) "username"
2) "antirez"
3) "password"
4) "12345"

```

5) "age"

6) "34"

## 三、Redis 持久化

redis 提供了两种持久化的方式，分别是 RDB (Redis DataBase) 和 AOF (Append Only File)。RDB，简而言之，就是在不同的时间点，将 redis 存储的数据生成快照并存储到磁盘等介质上；

AOF，则是换了一个角度来实现持久化，那就是将 redis 执行过的所有写指令记录下来，在下次 redis 重新启动时，只要把这些写指令从前到后再重复执行一遍，就可以实现数据恢复了。

其实 RDB 和 AOF 两种方式也可以同时使用，在这种情况下，如果 redis 重启的话，则会优先采用 AOF 方式来进行数据恢复，这是因为 AOF 方式的数据恢复完整度更高。

如果你没有数据持久化的需求，也完全可以关闭 RDB 和 AOF 方式，这样的话，redis 将变成一个纯内存数据库，就像 memcache 一样。

### 3.1、RDB

RDB 方式，是将 redis 某时刻的数据持久化到磁盘中，是一种快照式的持久化方法。redis 在进行数据持久化的过程中，会先将数据写入到一个临时文件中，待持久化过程都结束了，才会用这个临时文件替换上次持久化好的文件。正是这种特性，让我们可以随时来进行备份，因为快照文件总是完整可用的。

对于 RDB 方式，redis 会单独创建 (fork) 一个子进程来进行持久化，而主进程是不会进行任何 IO 操作的，这样就确保了 redis 极高的性能。

如果需要进行大规模数据的恢复，且对于数据恢复的完整性不是非常敏感，那 RDB 方式要比 AOF 方式更加的高效。

虽然 RDB 有不少优点，但它的缺点也是不容忽视的。如果你对数据的完整性非常敏感，那么 RDB 方式就不太适合你，因为即使你每 5 分钟都持久化一次，当 redis 故障时，仍然会有近 5 分钟的数据丢失。所以，redis 还提供了另一种持久化方式，那就是 AOF。

### 3.2、AOF

AOF，英文是 Append Only File，即只允许追加不允许改写的文件。

如前面介绍的，AOF 方式是将执行过的写指令记录下来，在数据恢复时按照从前到后的顺序再将指令都执行一遍，就这么简单。

我们通过配置 `redis.conf` 中的 `appendonly yes` 就可以打开 AOF 功能。如果有写操作（如 SET 等），`redis` 就会被追加到 AOF 文件的末尾。

默认的 AOF 持久化策略是每秒钟 `fsync` 一次（`fsync` 是指把缓存中的写指令记录到磁盘中），因为在这种情况下，`redis` 仍然可以保持很好的处理性能，即使 `redis` 故障，也只会丢失最近 1 秒钟的数据。

如果在追加日志时，恰好遇到磁盘空间满、inode 满或断电等情况导致日志写入不完整，也没有关系，`redis` 提供了 `redis-check-aof` 工具，可以用来进行日志修复。

因为采用了追加方式，如果不做任何处理的话，AOF 文件会变得越来越大，为此，`redis` 提供了 AOF 文件重写（`rewrite`）机制，即当 AOF 文件的大小超过所设定的阈值时，`redis` 就会启动 AOF 文件的内容压缩，只保留可以恢复数据的最小指令集。举个例子或许更形象，假如我们调用了 100 次 `INCR` 指令，在 AOF 文件中就要存储 100 条指令，但这明显是很低效的，完全可以把这 100 条指令合并成一条 `SET` 指令，这就是重写机制的原理。

在进行 AOF 重写时，仍然是采用先写临时文件，全部完成后再替换的流程，所以断电、磁盘满等问题都不会影响 AOF 文件的可用性，这点大家可以放心。

AOF 方式的另一个好处，我们通过一个“场景再现”来说明。某同学在操作 `redis` 时，不小心执行了 `FLUSHALL`，导致 `redis` 内存中的数据全部被清空了，这是很悲剧的事情。不过这也不是世界末日，只要 `redis` 配置了 AOF 持久化方式，且 AOF 文件还没有被重写（`rewrite`），我们就可以用最快的速度暂停 `redis` 并编辑 AOF 文件，将最后一行的 `FLUSHALL` 命令删除，然后重启 `redis`，就可以恢复 `redis` 的所有数据到 `FLUSHALL` 之前的状态了。是不是很神奇，这就是 AOF 持久化方式的好处之一。但是如果 AOF 文件已经被重写了，那就无法通过这种方法来恢复数据了。

虽然优点多多，但 AOF 方式也同样存在缺陷，比如在同样数据规模的情况下，AOF 文件要比 RDB 文件的体积大。而且，AOF 方式的恢复速度也要慢于 RDB 方式。

如果你直接执行 `BGREWRITEAOF` 命令，那么 `redis` 会生成一个全新的 AOF 文件，其中便包括了可以恢复现有数据的最少的命令集。

如果运气比较差，AOF 文件出现了被写坏的情况，也不必过分担忧，`redis` 并不会贸然加载这个有问题的 AOF 文件，而是报错退出。这时可以通过以下步骤来修复出错的文件：

#### 1、备份被写坏的 AOF 文件



- 2、运行 `redis-check-aof -fix` 进行修复
- 3、用 `diff -u` 来看下两个文件的差异，确认问题点
- 4、重启 `redis`，加载修复后的 AOF 文件

### 3.3、AOF 重写

AOF 重写的内部运行原理，我们有必要了解一下。

在重写即将开始之际，`redis` 会创建（`fork`）一个“重写子进程”，这个子进程会首先读取现有的 AOF 文件，并将其包含的指令进行分析压缩并写入到一个临时文件中。

与此同时，主工作进程会将新接收到的写指令一边累积到内存缓冲区中，一边继续写入到原有的 AOF 文件中，这样做是保证原有的 AOF 文件的可用性，避免在重写过程中出现意外。

当“重写子进程”完成重写工作后，它会给父进程发一个信号，父进程收到信号后就会将内存中缓存的写指令追加到新 AOF 文件中。

当追加结束后，`redis` 就会用新 AOF 文件来代替旧 AOF 文件，之后再有新的写指令，就都会追加到新的 AOF 文件中了。

### 3.4、如何选择 RDB 和 AOF

对于我们应该选择 RDB 还是 AOF，官方的建议是两个同时使用。这样可以提供更可靠的持久化方案。

## 四、主从同步

像 MySQL 一样，redis 是支持主从同步的，而且也支持一主多从以及多级从结构。

主从结构，一是为了纯粹的冗余备份，二是为了提升读性能，比如很消耗性能的 SORT 就可以由从服务器来承担。

redis 的主从同步是异步进行的，这意味着主从同步不会影响主逻辑，也不会降低 redis 的处理性能。

主从架构中，可以考虑关闭主服务器的数据持久化功能，只让从服务器进行持久化，这样可以提高主服务器的处理性能。

在主从架构中，从服务器通常被设置为只读模式，这样可以避免从服务器的数据被误修改。但是从服务器仍然可以接受 CONFIG 等指令，所以还是不应该将从服务器直接暴露到不安全的网络环境中。如果必须如此，那可以考虑给重要指令进行重命名，来避免命令被外人误执行。

### 4.1、同步原理

从服务器会向主服务器发出 SYNC 指令，当主服务器接到此命令后，就会调用 BGSAVE 指令来创建一个子进程专门进行数据持久化工作，也就是将主服务器的数据写入 RDB 文件中。在数据持久化期间，主服务器将执行的写指令都缓存在内存中。

在 BGSAVE 指令执行完成后，主服务器会将持久化好的 RDB 文件发送给从服务器，从服务器接到此文件后会将其存储到磁盘上，然后再将其读取到内存中。这个动作完成后，主服务器会将这段时间缓存的写指令再以 redis 协议的格式发送给从服务器。

另外，要说的一点是，即使有多个从服务器同时发来 SYNC 指令，主服务器也只会执行一次 BGSAVE，然后把持久化好的 RDB 文件发给多个下游。在 redis2.8 版本之前，如果从服务器与主服务器因某些原因断开连接的话，都会进行一次主从之间的全量的数据同步；而在 2.8 版本之后，redis 支持了效率更高的增量同步策略，这大大降低了连接断开的恢复成本。

主服务器会在内存中维护一个缓冲区，缓冲区中存储着将要发给从服务器的内容。从服务器在与主服务器出现网络瞬断之后，从服务器会尝试再次与主服务器连接，一旦连接成功，从服务器就会把“希望同步的主服务器 ID”和“希望请求的数据的偏移位置（replication

offset) ”发送出去。主服务器接收到这样的同步请求后，首先会验证主服务器 ID 是否和自己的 ID 匹配，其次会检查“请求的偏移位置”是否存在于自己的缓冲区中，如果两者都满足的话，主服务器就会向从服务器发送增量内容。

增量同步功能，需要服务器端支持全新的 PSYNC 指令。这个指令，只有在 redis-2.8 之后才具有。

## 五、Redis 总结

### 5.1、Redis 的优势

性能极高 - redis 能读的速度是 110000 次/s，写的速度是 81000 次/s。

丰富的数据类型 - redis 支持二进制安全的 Strings, Lists, Hashes, Sets 及 Ordered Sets 数据类型操作。

原子 - redis 的所有操作都是原子性的，同时 Redis 还支持对几个操作全并后的原子性执行。

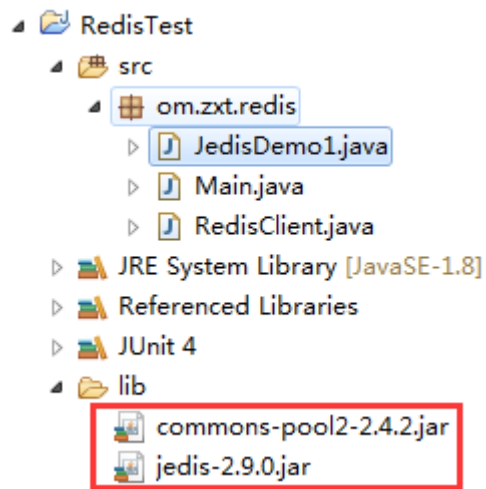
丰富的特性 - redis 还支持 publish/subscribe，通知，key 过期等等特性。

### 5.2、Redis 与其他 key-value 存储有什么不同？

redis 有着更为复杂的数据结构并且提供对他们的原子性操作，这是一个不同于其他数据库的进化路径。redis 的数据类型都是基于基本数据结构的同时对程序员透明，无需进行额外的抽象。

redis 运行在内存中但是可以持久化到磁盘，所以在对不同数据集进行高速读写时需要权衡内存，应用数据量不能大于硬件内存。在内存数据库方面的另一个优点是，相比在磁盘上相同的复杂的数据结构，在内存中操作起来非常简单，这样 Redis 可以做很多内部复杂性很强的事情。同时，在磁盘格式方面他们是紧凑的以追加的方式产生的，因为他们并不需要进行随机访问。

## 六、Java 操作 Redis（Jedis 入门）



JedisDemo1

```
package om.zxt.redis;

import org.junit.Test;

import redis.clients.jedis.Jedis;
import redis.clients.jedis.JedisPool;
import redis.clients.jedis.JedisPoolConfig;

/**
 * @ClassName: JedisDemo1.java
 * @Description: jedis的测试
 * @author zxt
 * @Date: 2018年3月28日 上午8:25:19
 */
public class JedisDemo1 {

    private JedisPool jedisPool;
    private static Jedis jedis;

    public static void main(String[] args) {
        jedis = new Jedis("localhost");
        System.out.println("连接成功");
        //查看服务是否运行
        System.out.println("服务正在运行: "+jedis.ping());
    }
}
```

```
/**
 * @Description: 单实例测试
 * @return: void
 */
@Test
public void demo1() {
    // 1、设置ip地址和端口
    Jedis jedis = new Jedis("127.0.0.1", 6379);
    // 2、保存数据
    jedis.set("name", "zxt");
    // 3、获取数据
    String value = jedis.get("name");
    System.out.println(value);
    // 4、释放资源
    jedis.close();
}

/**
 * @Description: 连接池方式连接
 * @return: void
 */
@Test
public void demo2() {
    // 1、获取连接池的配置对象
    JedisPoolConfig config = new JedisPoolConfig();
    // 设置最大连接数
    config.setMaxTotal(30);
    // 设置最大空闲连接数
    config.setMaxIdle(10);

    jedisPool = new JedisPool(config, "127.0.0.1", 6379);

    // 获取核心对象
    Jedis jedis = null;
    try {
        // 通过连接池获得连接
        jedis = jedisPool.getResource();
        // 设置数据
        jedis.set("age", "24");
        // 获取数据
        String age = jedis.get("age");
        System.out.println(age);
    } catch (Exception e) {
```

```

        e.printStackTrace();

    } finally {
        // 释放资源
        if(jedis != null) {
            jedis.close();
        }
        if(jedisPool != null) {
            jedisPool.close();
        }
    }
}
}
}

```

## RedisClient.java

```

public class RedisClient {

    private static Jedis jedis;
    private static JedisPool jedisPool;

    static {
        initialPool();
    }

    public static Jedis getMyRedis() {
        jedis = jedisPool.getResource();

        return jedis;
    }

    /**
     * @Description: 初始化Jedis的连接池
     * @return: void
     */
    private static void initialPool() {
        // 1、获取连接池的配置对象
        JedisPoolConfig config = new JedisPoolConfig();

        // 设置最大连接数
        config.setMaxTotal(30);
        // 设置最大空闲连接数
        config.setMaxIdle(10);
        // 获取连接时的最大等待毫秒数
    }
}

```

```

        config.setMaxWaitMillis(10001);

        jedisPool = new JedisPool(config, "127.0.0.1", 6379);
    }
}

```

测试:

```

public class Main {

    public static void main(String[] args) {
        Jedis jedis = RedisClient.getMyRedis();
        System.out.println("连接成功");

        // 存储数据到列表中
        jedis.lpush("site-list", "Runoob");
        jedis.lpush("site-list", "Google");
        jedis.lpush("site-list", "Taobao");
        // 获取存储的数据并输出
        List<String> list = jedis.lrange("site-list", 0, 2);
        for (int i = 0; i < list.size(); i++) {
            System.out.println("列表项为: " + list.get(i));
        }

        System.out.println("=====输出Redis中的所有key值=====");
        // 获取Redis中的所有数据并输出
        Set<String> keys = jedis.keys("*");
        Iterator<String> it=keys.iterator() ;
        while(it.hasNext()){
            String key = it.next();
            System.out.println(key);
        }

        // 释放资源
        if(jedis != null) {
            jedis.close();
        }
    }
}

```



连接成功

列表项为: Taobao

列表项为: Google

列表项为: Runoob

=====输出Redis中的所有key值=====

name

mynum

hello

age

mylist

site-list