# Redis In Action

redis 是一个远程内存数据库。

# 初识Redis

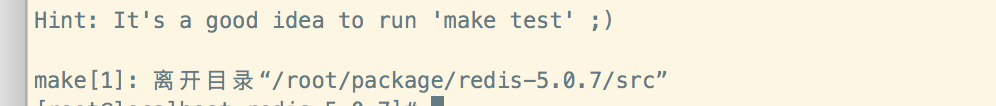
## 1.1 安装

wget <http://download.redis.io/releases/redis-5.0.7.tar.gz>

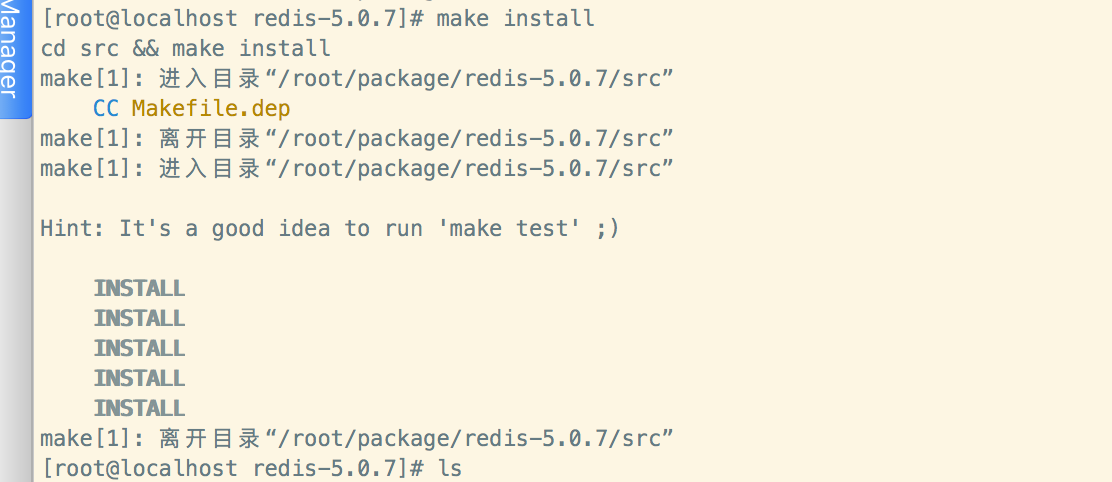
tar -zxvf redis-5.0.7.tar.gz

cd redis-5.0.7

make(成功如下)



make install

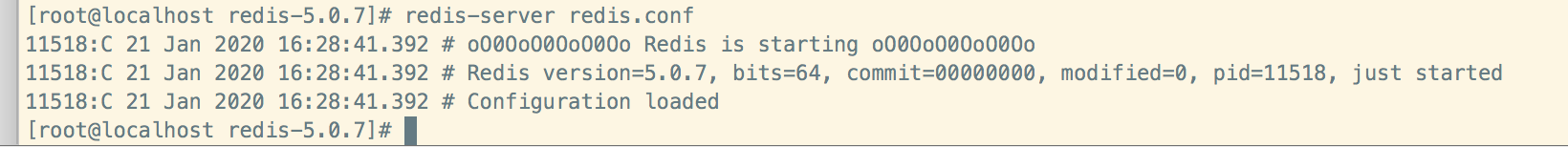


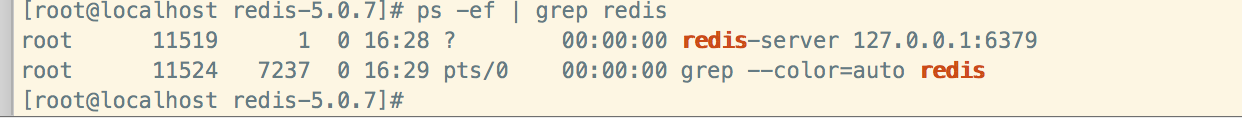
cp redis.conf redis.conf.bak 备份配置文件

vim redis.conf

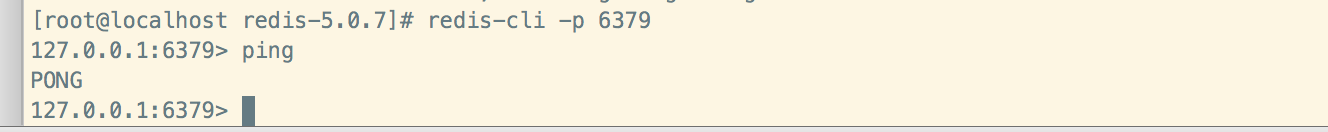
修改daemonize的值为yes，意思后台线程模式运行

redis-server redis.conf 启动



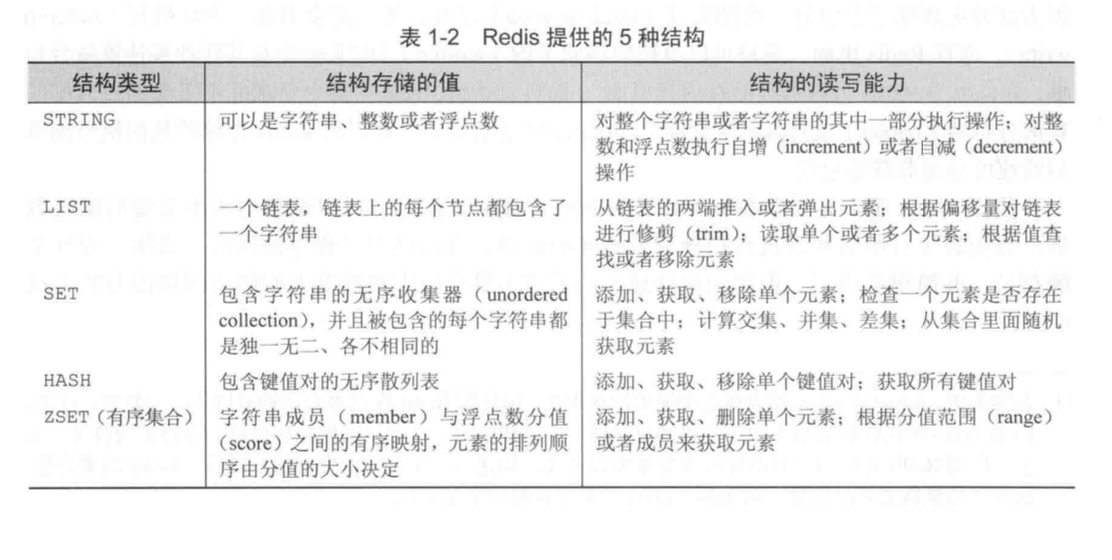


redis-cli -p 6379

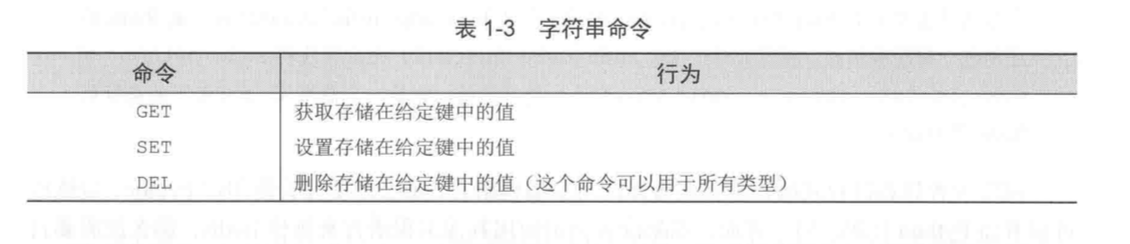


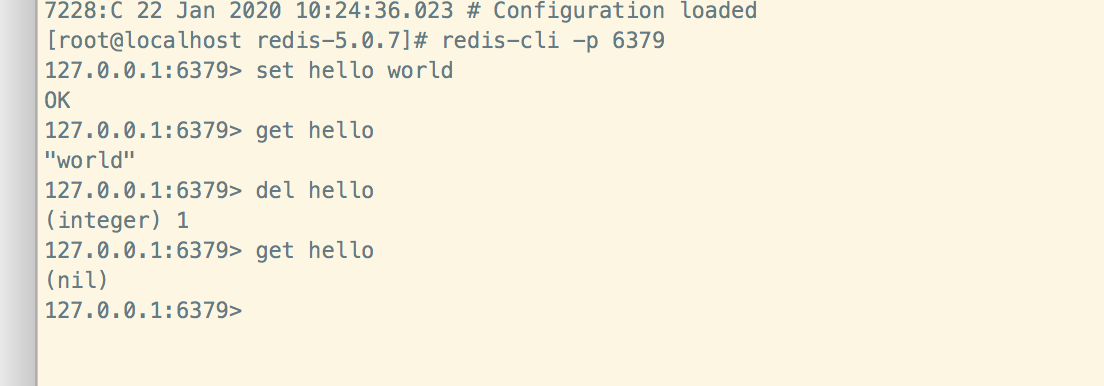
## 1.2 Redis数据结构简介

5中数据结构：STRING（字符串）、LIST（列表）、SET（集合）、HASH（散列）和ZSET（有序集合）。

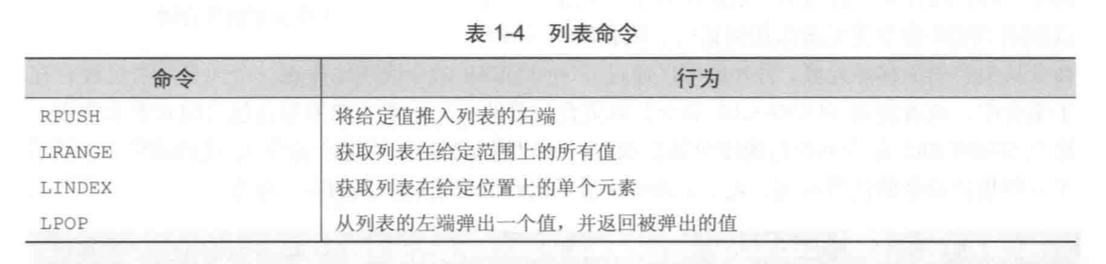


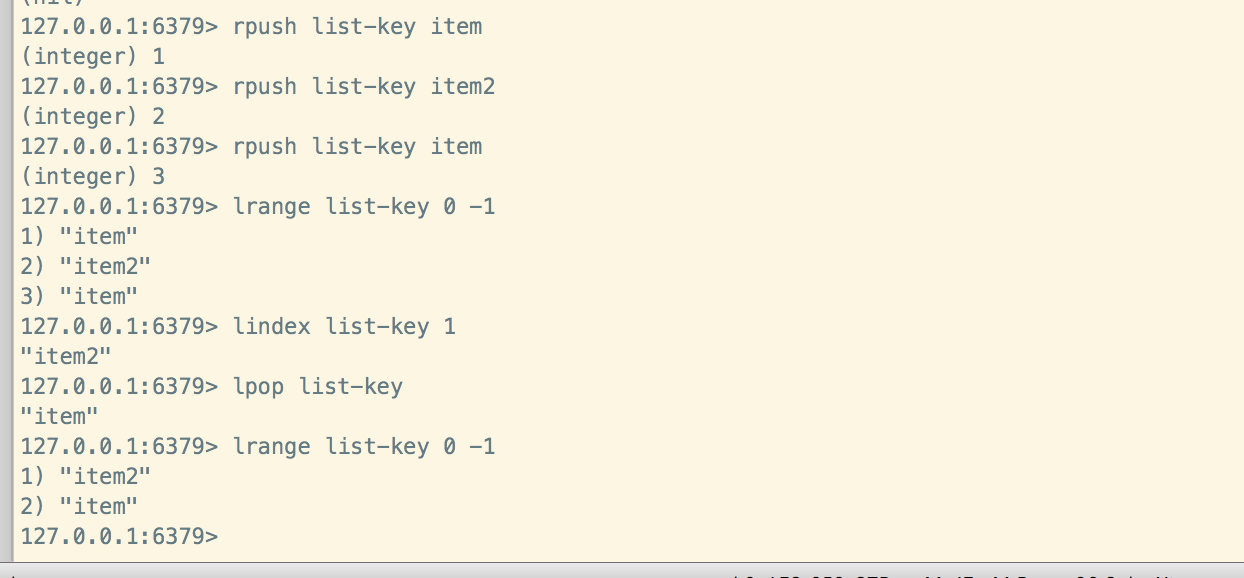
### 1.2.1 Redis中的字符串





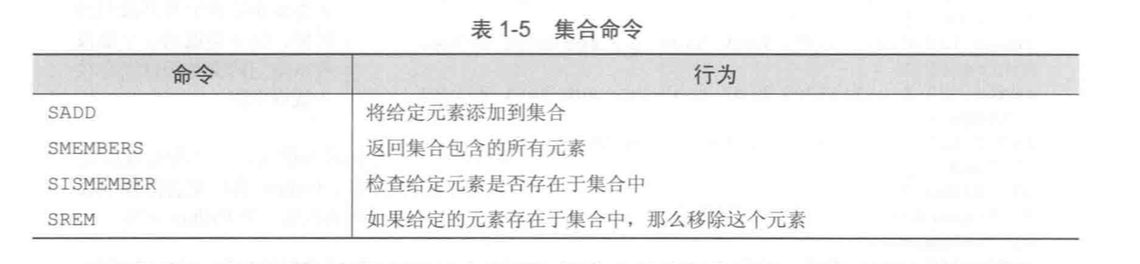
### 1.2.2 Redis中的列表





### 1.2.3 Redis的集合

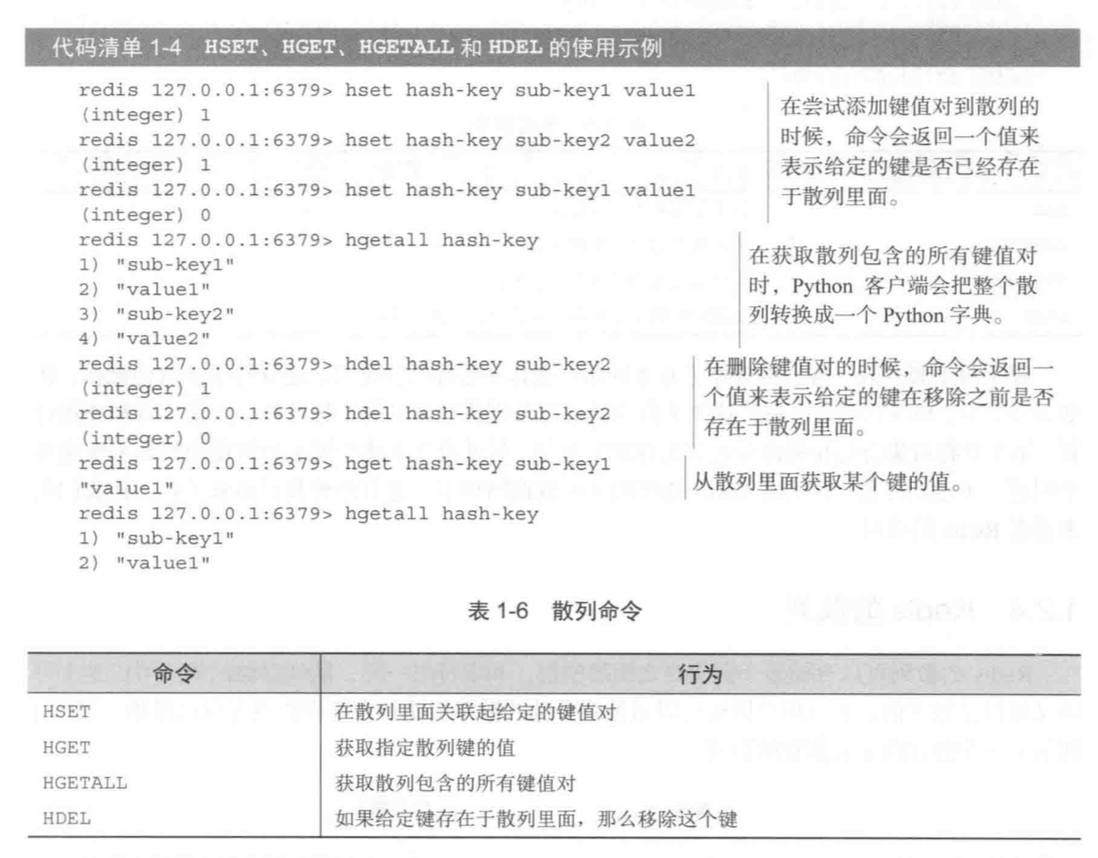
列表可以存储相同的字符串，而集合则通过使用散列表来保证自己存储的每个字符串都是各不相同的（这些散列表只有键，但没有与键相关联的值）





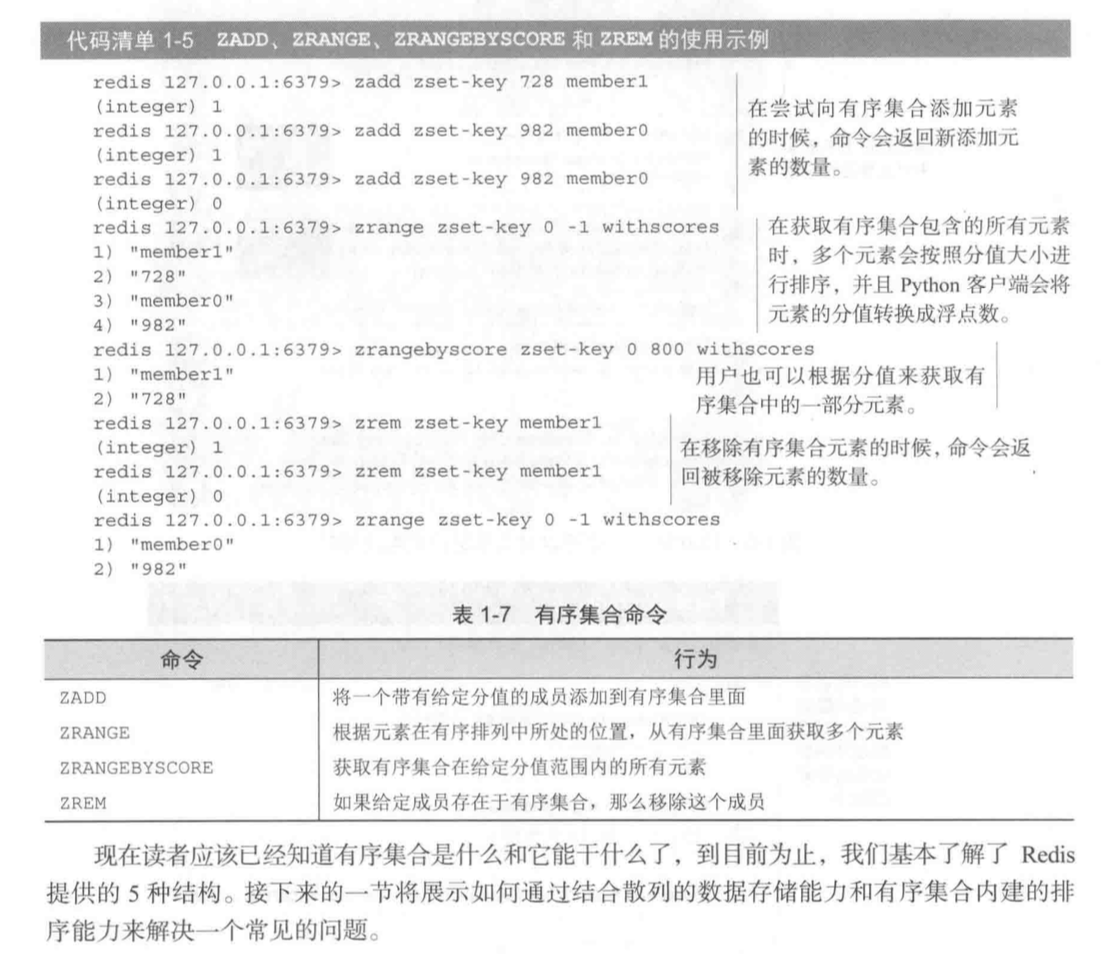
### 1.2.4 Redis的散列

散列可以存储多个键值对之间的映射。和字符串一样，散列存储的值既可以是字符串又可以是数字值。并且用户同样可以对散列存储的数字值执行自增操作或者自减操作。



### 1.2.5 Redis的有序集合

有序集合和散列一样，都用于存储键值对：有序集合的键被称为成员(member)，每个成员都是各不相同的；而有序集合的值则被称为分值(score)，分值必须为浮点数。有序集合是Redis里面唯一一个既可以根据成员访问元素(这一点和散列一样)，又可以根据分值以及分值的排列顺序来访问元素的结构。



# 使用Redis构建Web应用

## 2.1 登陆和cookie缓存

有两种常见的方法可以将登陆信息存储在cookie里面：一种是签名(signed)cookie，另一种是令牌(token) cookie。

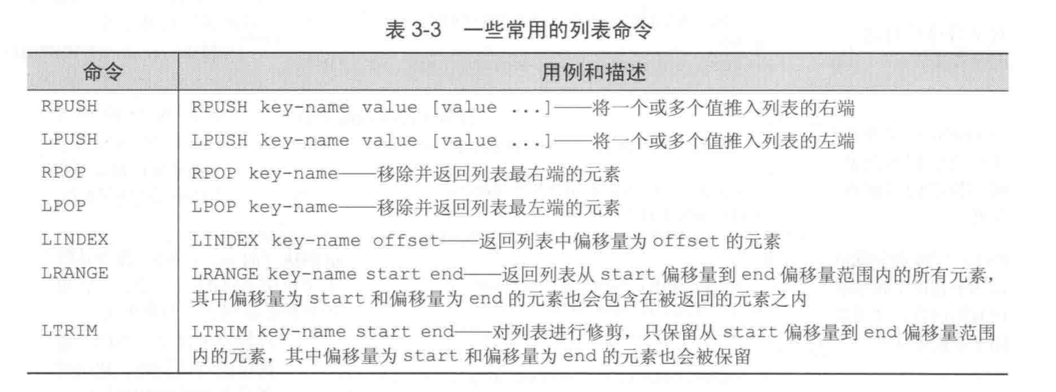
# Redis 命令

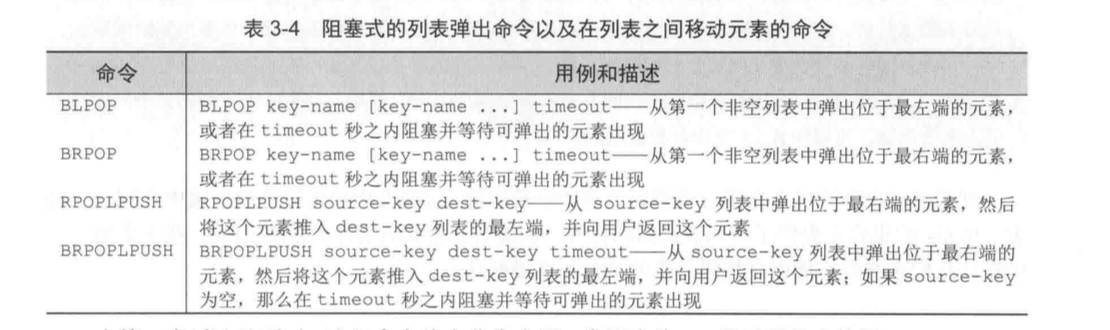
## 3.1 字符串

字符串可以存储：字符串，整数，浮点数。

|  |
| --- |
| private static void testString() {  Jedis conn = new Jedis("172.16.144.145", 6379);  conn.select(15);   String key = conn.get("key");  System.*out*.printf("key=%s\n", key);  // 自增加1,返回加1后的结果  Long incr = conn.incr("key");  System.*out*.printf("自增加%d\n", incr);   Long decr = conn.decr("key");  System.*out*.printf("自动减%d\n", decr);  // 对字符串进行追加  Long hello = conn.append("new-string-key", "hello");  System.*out*.printf("append=%d\n", hello);   System.*out*.printf("new-string-key=%s\n", conn.get("new-string-key"));   // 截取字符串  String substr = conn.substr("new-string-key", 3, 7);  System.*out*.printf("substr=%s\n", substr);   // 设置指定位置的值  Long setrange = conn.setrange("new-string-key", 0, "H");  System.*out*.println("setrange=" + setrange);  System.*out*.printf("new-string-key=%s\n", conn.get("new-string-key"));   } |

## 3.2 列表





|  |
| --- |
| private static void testList() {  Jedis conn = new Jedis("172.16.144.145", 6379);  conn.select(15);   // 将一个值推入列表的右端 // Long last = conn.rpush("list-key", "last");  // 将一个值推入列表的左端 // Long lpush = conn.lpush("list-key", "new last");  // 移除并返回最左端的元素 // String lpop = conn.lpop("list-key"); // System.out.println("lpop=" + lpop);   // 对列表进行修剪，保留从start到end之间的元素 // String ltrim = conn.ltrim("list-key", 1, -1);  // System.out.println("ltrim=" + ltrim);   // 返回列表从start偏移量到end偏移量范围内的所有元素，其中偏移量为start和偏移量为end的元素也会包含在被返回的元素之内 // List<String> list = conn.lrange("list-key", 0, -1);    conn.rpush("list","item1");  conn.rpush("list","item2");  conn.rpush("list2","item3");  // 阻塞地将一个列表中的元素移到另一个列表中  conn.brpoplpush("list2","list",1);   System.*out*.println(Arrays.*toString*(conn.lrange("list",0,-1).toArray()));    } |

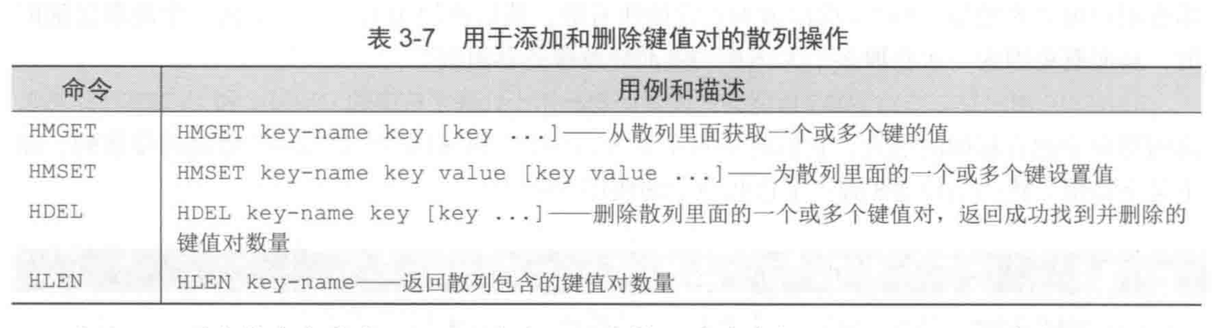
## 3.3 集合

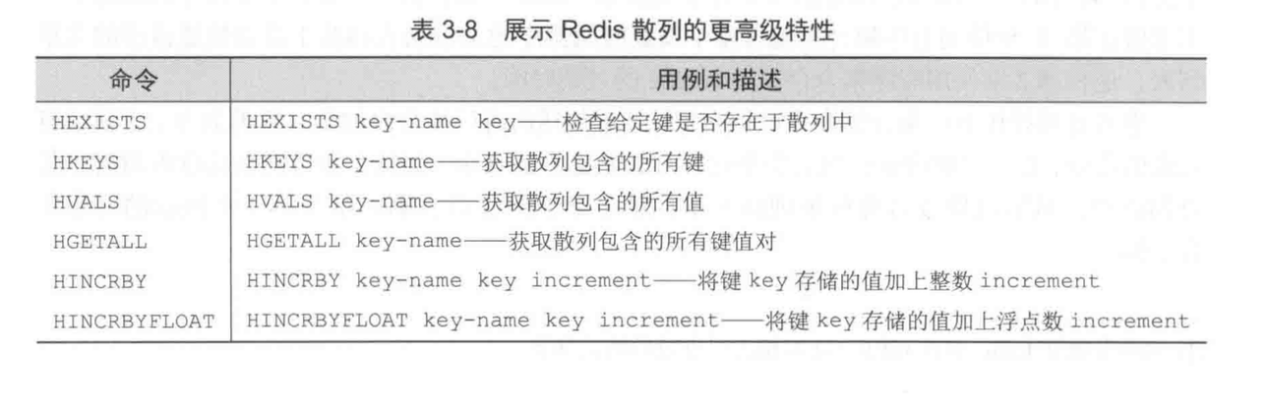




|  |
| --- |
| private static void testSet() {  Jedis conn = new Jedis("172.16.144.145", 6379);  conn.select(15);   // 将多个元素添加到集合中 // Long sadd = conn.sadd("set-key", "a", "b", "c"); // // // 删除元素，返回1，代表有元素被删除，返回0，代表没有删除任何元素 // System.out.println(conn.srem("set-key","c","d")); // System.out.println(conn.srem("set-key","c","d"));  // // 获取集合中所有元素 // Set<String> smembers = conn.smembers("set-key"); // System.out.println(Arrays.toString(smembers.toArray())); // // // 从集合set-key中移除元素a，并添加到set-key2集合中 // conn.smove("set-key", "set-key2", "a"); // // // System.out.println(Arrays.toString(conn.smembers("set-key").toArray())); // System.out.println(Arrays.toString(conn.smembers("set-key2").toArray()));    conn.sadd("skey1", "a", "b", "c", "d");  conn.sadd("skey2", "c", "d", "e", "f");   // 返回存在于第一个集合，不存在与第二个集合的元素：差集  Set<String> sdiff = conn.sdiff("skey1", "skey2");  System.*out*.println(Arrays.*toString*(sdiff.toArray()));   // 返回两个集合同时存在的元素：交集  Set<String> sinter = conn.sinter("skey1", "skey2");  System.*out*.println(Arrays.*toString*(sinter.toArray()));   // 并集运算  Set<String> sunion = conn.sunion("skey1", "skey2");  System.*out*.println(Arrays.*toString*(sunion.toArray()));   } |

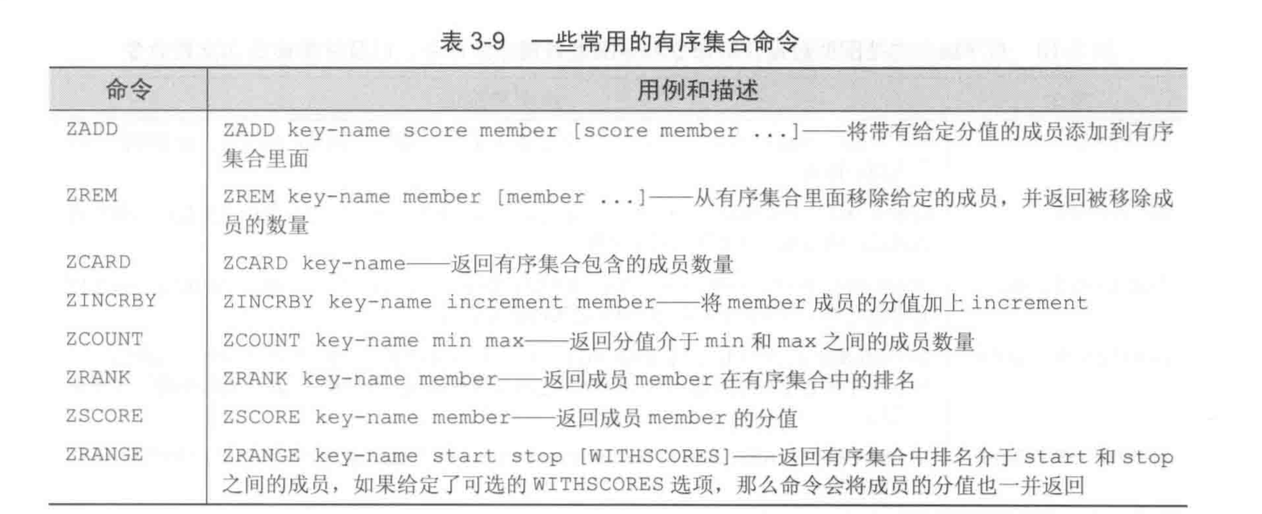
## 3.4 散列

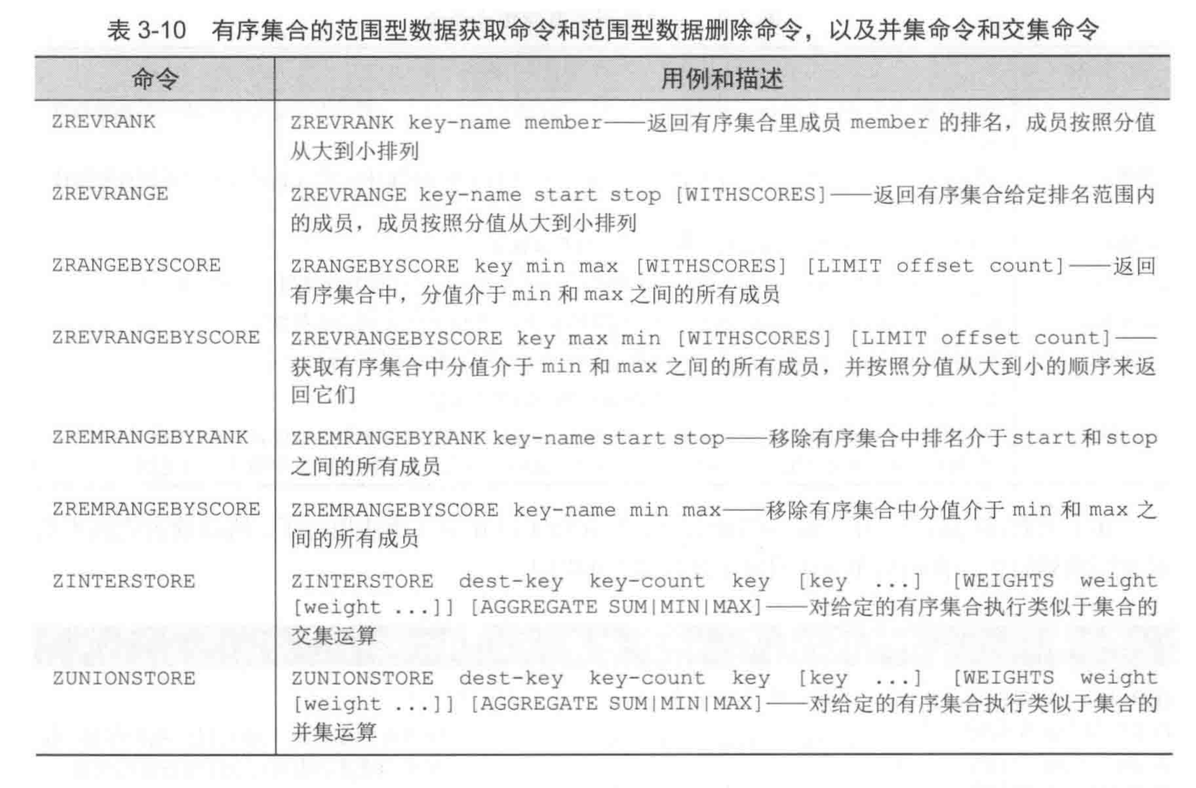




|  |
| --- |
| private static void testHash() {  Jedis conn = new Jedis("172.16.144.145", 6379);  conn.select(15);  // HashMap<String, String> hashMap = new HashMap<String, String>(); // hashMap.put("k1", "v1"); // hashMap.put("k2", "v2"); // hashMap.put("k3", "v3"); // hashMap.put("k4", "v4"); // // // 设置值 // String hmset = conn.hmset("hash-key", hashMap); // // // 从散列中获取一个或多个key的值 // List<String> hmget = conn.hmget("hash-key", "k1", "k2"); // // System.out.println(Arrays.toString(hmget.toArray())); // // // 返回散列包含的键值对数量 // System.out.println(conn.hlen("hash-key")); // // // 移除散列中的键值对 // conn.hdel("hash-key", "k1", "k2");    //    HashMap<String, String> map = new HashMap<String, String>();  map.put("short", "hello");  map.put("long", "1000\*1");   // 设置值  String hmset = conn.hmset("hash-key2", map);   // 获得散列的所有key  Set<String> hkeys = conn.hkeys("hash-key2");  System.*out*.println(Arrays.*toString*(hkeys.toArray()));   // 判断散列中是否有某个key  System.*out*.println(conn.hexists("hash-key2","num"));   // 将散列上num的值加上3  Long num = conn.hincrBy("hash-key2", "num", 3);  System.*out*.println(num);    } |

## 3.5 有序集合





## 3.6 发布与订阅

## 3.7 其他命令

### 3.7.2 基本的Redis事务

Redis有5个命令可以让用户在不被打断(interruption)的情况下对多个键执行操作，他们而分别是WATCH、MULTI、EXEC、UNWATCH和DISCARD

要在Redis里里面执行事务，我们首先需要执行MULTI命令，然后输入哪些我们想要的事务里面执行的命令，最后再执行EXEC命令。

### 3.7.3 键的过期时间



### 3.7.4 停止Redis服务

（1）通过redis-cli连接服务器后执行shutdown命令，则执行停止redis服务操作。

（2）可以使用shutdown命令关闭redis服务器外，还可以使用kill+进程号的方式关闭redis服务。

（3）不要使用Kill 9方式关闭redis进程，这样redis不会进行持久化操作，除此之外，还会造成缓冲区等资源不能优雅关闭，极端情况下会造成AOF和复制丢失数据的情况

（4）shutdown还有一个参数，代表关闭redis服务前是否生产持久化文件

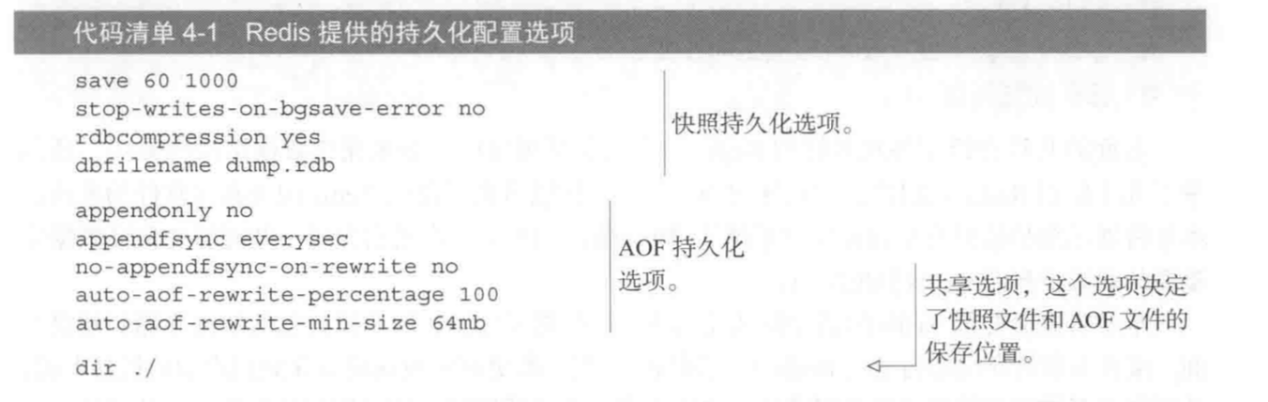
     shutdown  save|nosave

# 数据安全与性能保障

## 4.1 持久化选项

Redis提供了两种不同的持久化方法来将数据存储到硬盘里面。一种叫快照(snapshoting)，他可以将存在于某一时刻的所有数据都写入硬盘里面。另一种方法叫只追加文件(AOF)

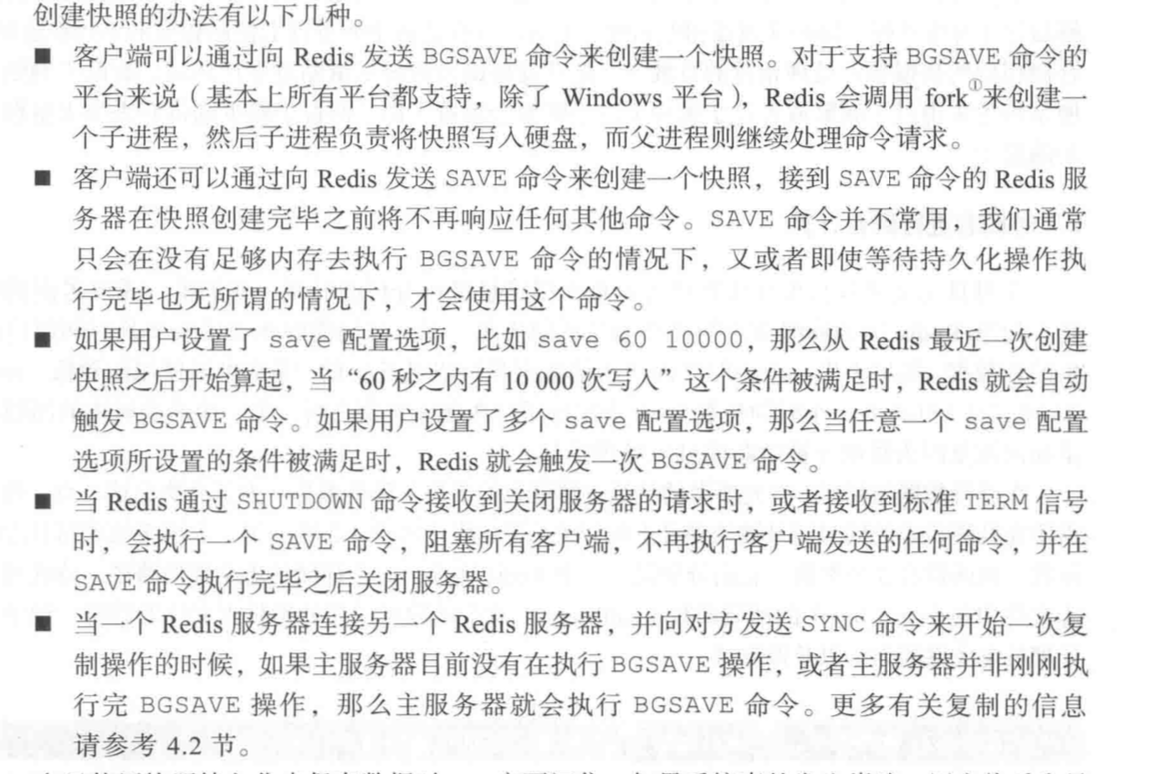
,它会将执行写命令时，将被执行的写命令复制到硬盘里。



### 4.1.1 快照持久化

根据配置，快照江北写入dbfilename选项指定的文件里面，并存储在dir选项指定的路径上面。

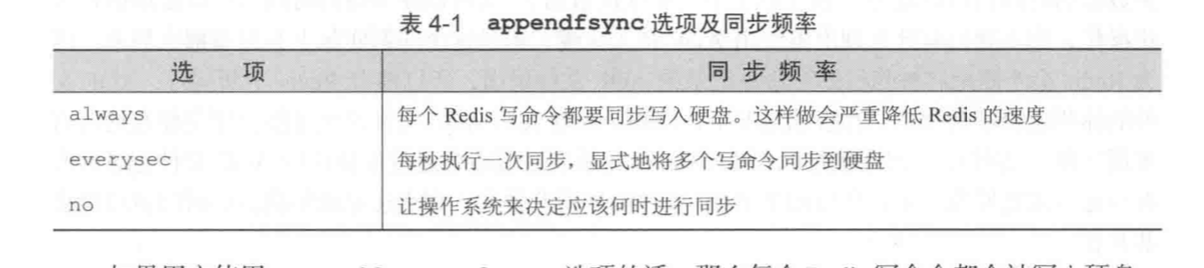
创建快照的办法有一下几种。



### 4.1.2 AOF持久化

AOF持久化会将被执行的写命令写到AOF文件的末尾，以此来记录数据发生变化。

appendonly yes 配置选项来打开。



### 4.1.3 重写/压缩AOF文件

为了解决AOF文件体积不断增大的问题，用户可以向Redis发送BGREWRITEAOF命令，这个命令会通过移除AOF文件中冗余的命令来重写AOF文件。

## 4.2 复制

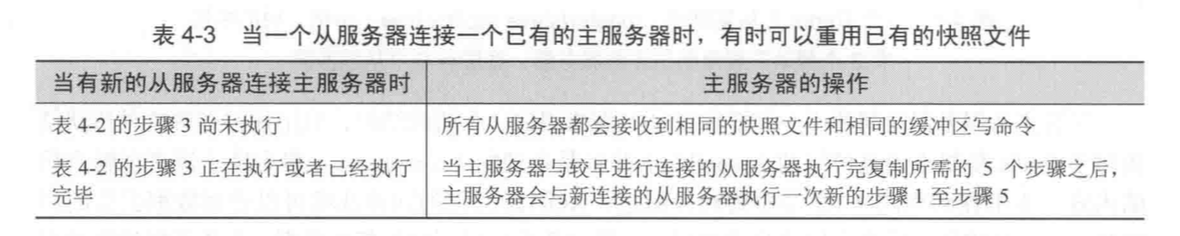
### 4.2.1 对Redis的复制相关选项进行配置

### 4.2.2 Redis复制的启动过程

|  |  |  |
| --- | --- | --- |
| 步骤 | 主服务器操作 | 从服务器操作 |
| 1 | （等待命令进入） | 连接(或者重连接)主服务器，发送SYNC命令 |
| 2 | 开始执行BGSAVE，并使用缓冲区记录BGSAVE之后执行的所有写命令 | 根据配置选项来决定是继续使用现有的数据（如果有的话）来处理客户端命令请求吧，还是向发送请求的客户端返回错误 |
| 3 | BGSAVE 执行完毕，向从服务器发送快照文件，并在发送期间继续使用缓冲区记录被执行的写命令 | 丢弃所有旧数据（如果有的话），开始载入主服务器发来的快照文件 |
| 4 | 快照文件发送完毕，开始向从服务器发送存储在缓冲区里里里里面的写命令 | 完成对快照文件的解释操作，像往常一样开始接受命令请求 |
| 5 | 缓冲区存储的写命令发送完毕乐从现在开始，每执行一个写命令，就向从服务器发送相同的写命令 | 执行主服务器发来的所有存储在缓冲区里面的写命令，并从现在开始，接受并执行主服务器传来的每个写命令 |

用户可以通过配置选项SLAVEOF host port来将一个Redis服务器设置为从服务器，又可以通过向运行中的Redis服务器发送SLAVEOF命令来将其设置为从服务器。如果用户使用的是SLAVEOF配置选项，那么Redis在启动时首先会载入当前可用的任何快照文件或者AOF文件，然后连接主服务器并执行复制步骤。

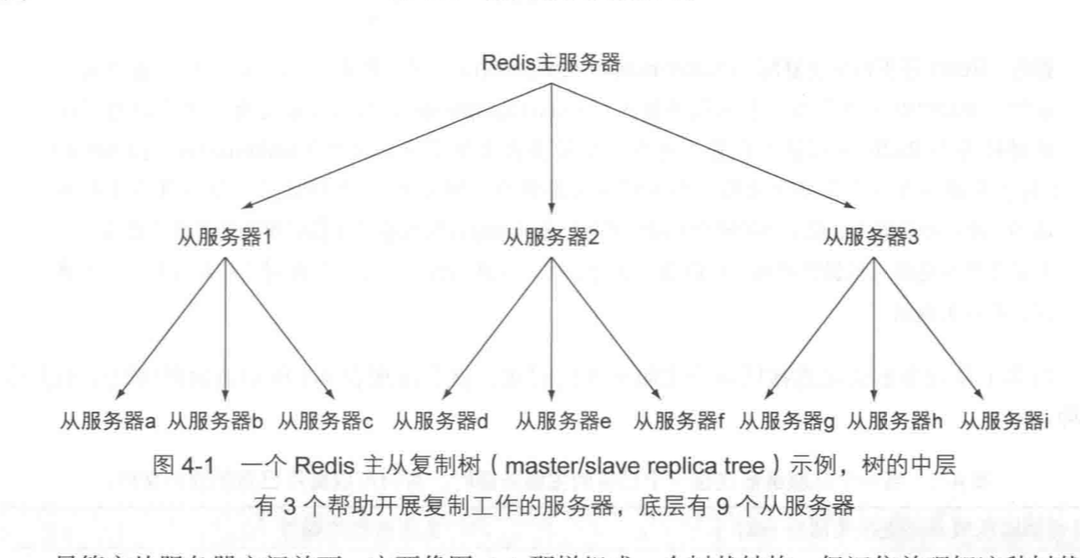
Redis不支持主主复制



### 4.2.3 主从链

从服务器也可以拥有自己的从服务器，由此形成主从链。

从服务器对从服务器进行赋值在操作上和从服务器对主服务器进行复制的区别：如果从服务器X拥有从服务器Y，那么当从服务器X在执行步骤4时，它将断开与从服务器Y的连接，导致从服务器Y需要重新连接并重新同步(resync)



### 4.2.4 检验硬盘写入

为了验证主服务器是否已经将写数据发送至从服务器，用户需要在向主服务器写入真正的数据之后，再向主服务器写入一个唯一的虚构值(unique dummy value)，然后通过检查虚构值是否存在于从服务器来判断写数据是否已经到达从服务器。

判断数据是否已经被保存到硬盘里面则还要困难的多。检查INFO命令输出结果中aof\_pending\_bio\_fsync属性的值是否为0.如果是的话，那么就表示服务器已经将一致的所有数据都保存到硬盘里面了。

## 4.3 处理故障系统

### 4.3.1 验证快照文件和AOF文件

redis-check-aof --fix 命令会对AOF文件进行修复。寻找不正确或者不完整的命令，当发现第一一个出现错误命令的时候，会删除出错以及位于出错命令之后的所有宁陵。

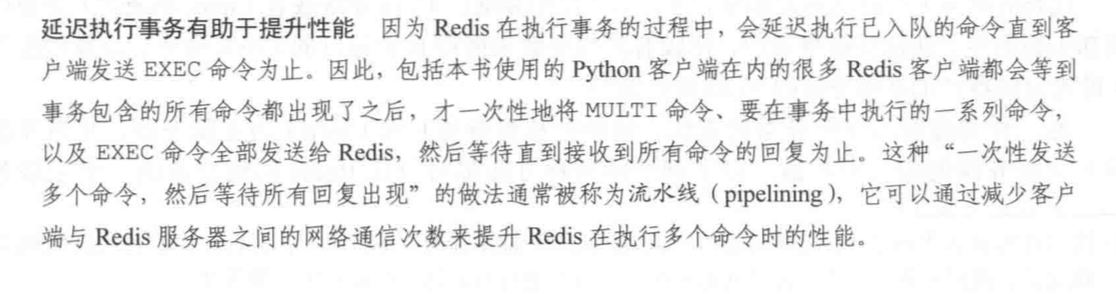
目前没有办法可以修复错误的快照文件。

### 4.3.2 更换故障主服务器

1. B两台机器，A主服务器，B从服务器，A出问题，需要更换为C

更换步骤：首先向机器B发送一个SAVE命令，让它创建一个新的快照文件，接着将这个快照文件发送给C，并在C上启动redis。最后，让B称为机器C的从服务器。

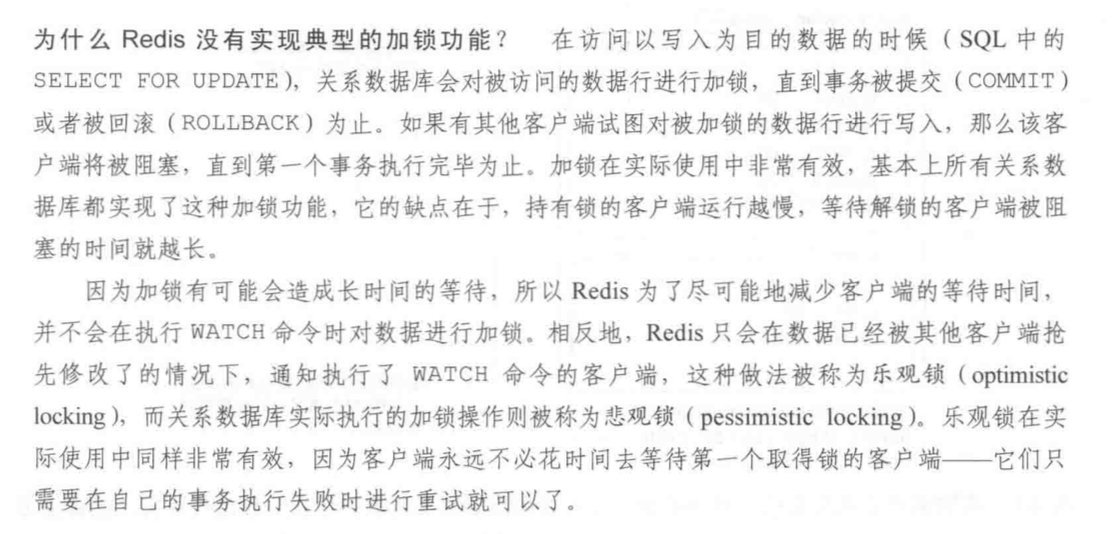
## 4.4 Redis事务



### 4.4.1 购买商品

### 4.4.2 购买商品

### 4.4.3 购买商品

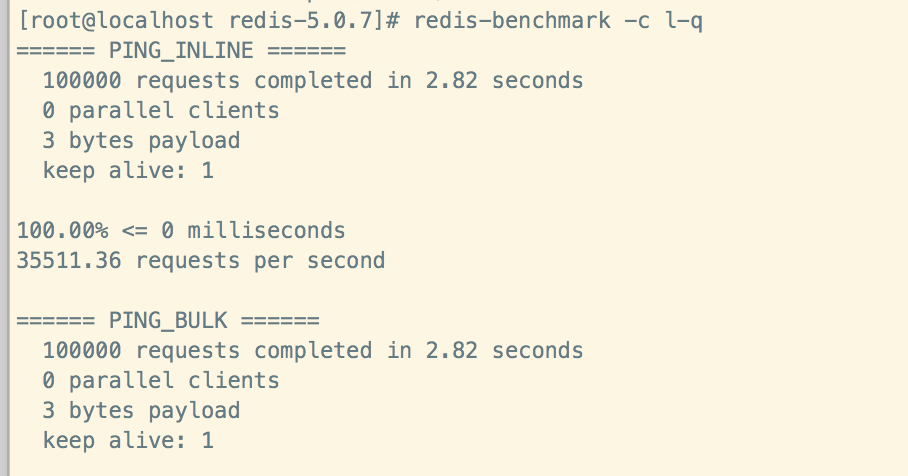


## 4.5 非持物型流水线

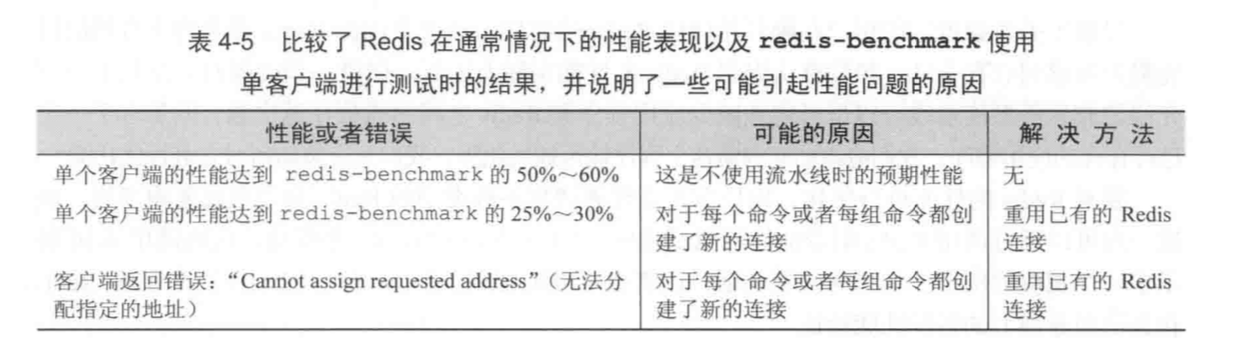
## 4.6 关于性能方面的注意事项

需要弄清楚各种类型的Redis命令到底能跑对快，而这一点可以通过调用Redis附带的性能测试程序redis-benchmark来得知。

redis-benchmark -c 1 -q （-q简化输出，-c 1程序只是用一个客户端来进行测试），



如果redis-benchmark不适用参数，默认是50个客户端进行测试



# 使用redis构建支持程序

## 5.1 记录日志

## 5.2 计数器和统计数据

## 5.3 查找IP所述城市以及国家

## 5.4 服务的发现与配置

# 使用redis构建应用程序组件

## 6.2 分布式锁

Redis使用WATCH命令来代替对数据进行加锁，因为WATCH只会在数据被其他客户端抢先修改了的情况下通知执行了这个命令的客户端，而不会阻止其他客户端数据进行修改，所以这个名年龄被称为乐观锁(optimistic locking).

### 6.2.2 简易锁