**Redis**

# 缓存问题

## 缓存雪崩

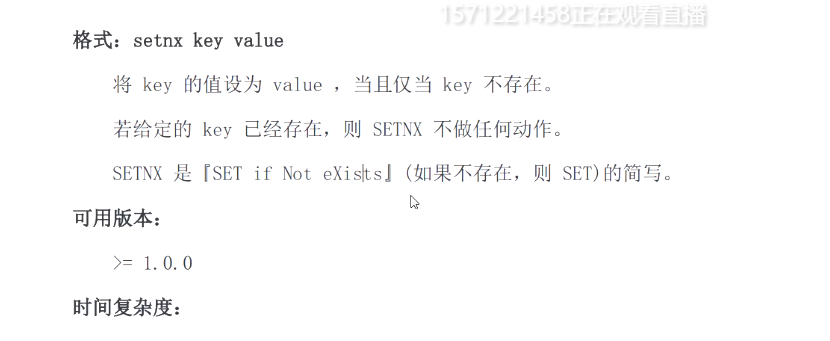
## 缓存击穿

## 缓存穿透

缓存穿透，是指查询一个数据库一定不存在的数据。正常的使用缓存流程大致是，数据查询先进行缓存查询，如果key不存在或者key已经过期，再对数据库进行查询，并把查询到的对象，放进缓存。如果数据库查询对象为空，则不放进缓存。

解決方法:布隆过滤器

# 分布式锁



* 放了防止redis宕机的时候，key一直存在，所以要设置超时时间。  
  设置了超时时间，如果过了超时时间，业务还没有执行完，那么key自动失效，就会有其他的线程来争夺锁，等到该线程执行完的时候，会释放锁，但是是释放的是其他线程加的锁，所以还是出现超卖的现象。
* 可以每隔一段时间(超时时间的三分之一，例如redission)检查锁是否失效，没有失效，就更新超时时间，为锁续命

## 分布式锁的问题

1. 由单机锁演进到分布式锁
2. 分布式锁可以由redis中setnx来实现
3. 使用这条命令，如果服务器宕机了，那么设置的值将一直存在

解决方法:可以设置超时时间

如果设置的超时时间过短，A线程的业务代码还没有执行完，锁就被redis删除了，此时B线程获取了锁，开始执行业务代码，刚好在这时B线程的锁就被A线程释放了，其他线程又可以获取锁继续执行，那么这还是会导致超卖问题

解决方法:可以写个定时任务，定时（最好是超时时间的三分之一）检查锁是否持有，如果持有，就延长锁的时间，可以使用redision

# 数据结构

Key设计的时候一般遵循：功能模块 表名 业务模块

## 存储上限

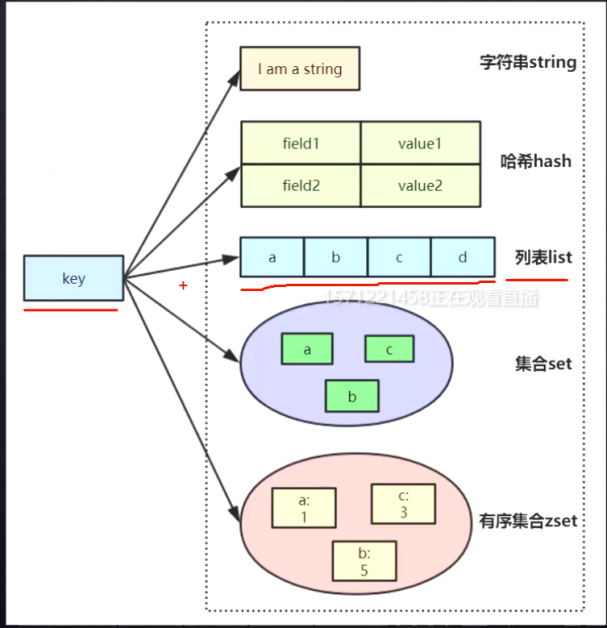
Strings类型：一个String类型的value最大可以存储512M

Lists类型：list的元素个数最多为2^32-1个，也就是4294967295个。

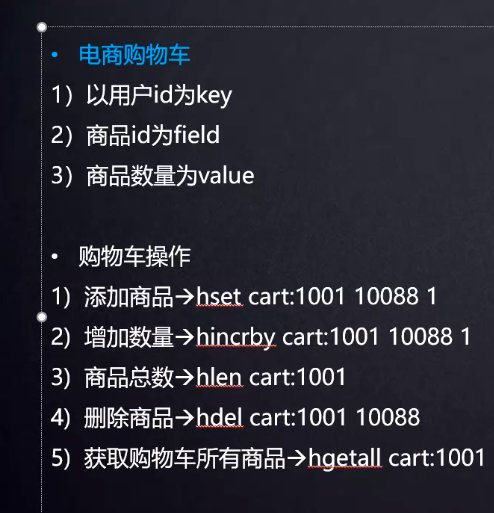
Sets类型：元素个数最多为2^32-1个，也就是4294967295个。

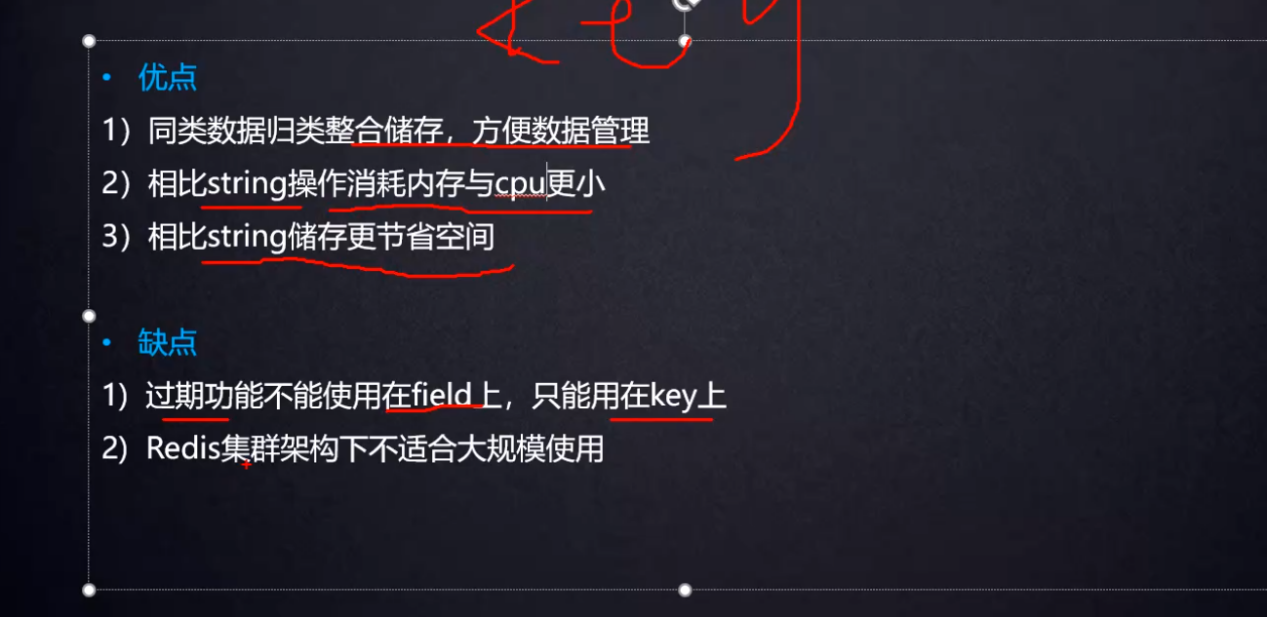
Hashes类型：键值对个数最多为2^32-1个，也就是4294967295个。

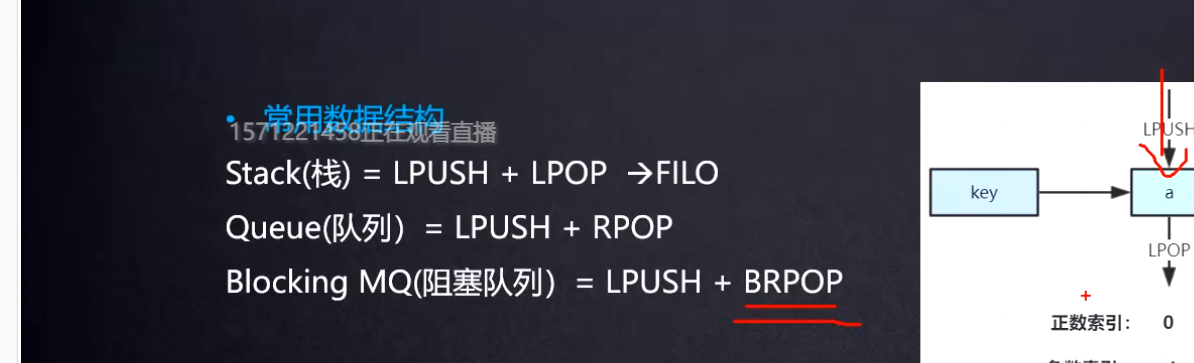
Sorted sets类型：跟Sets类型相似。

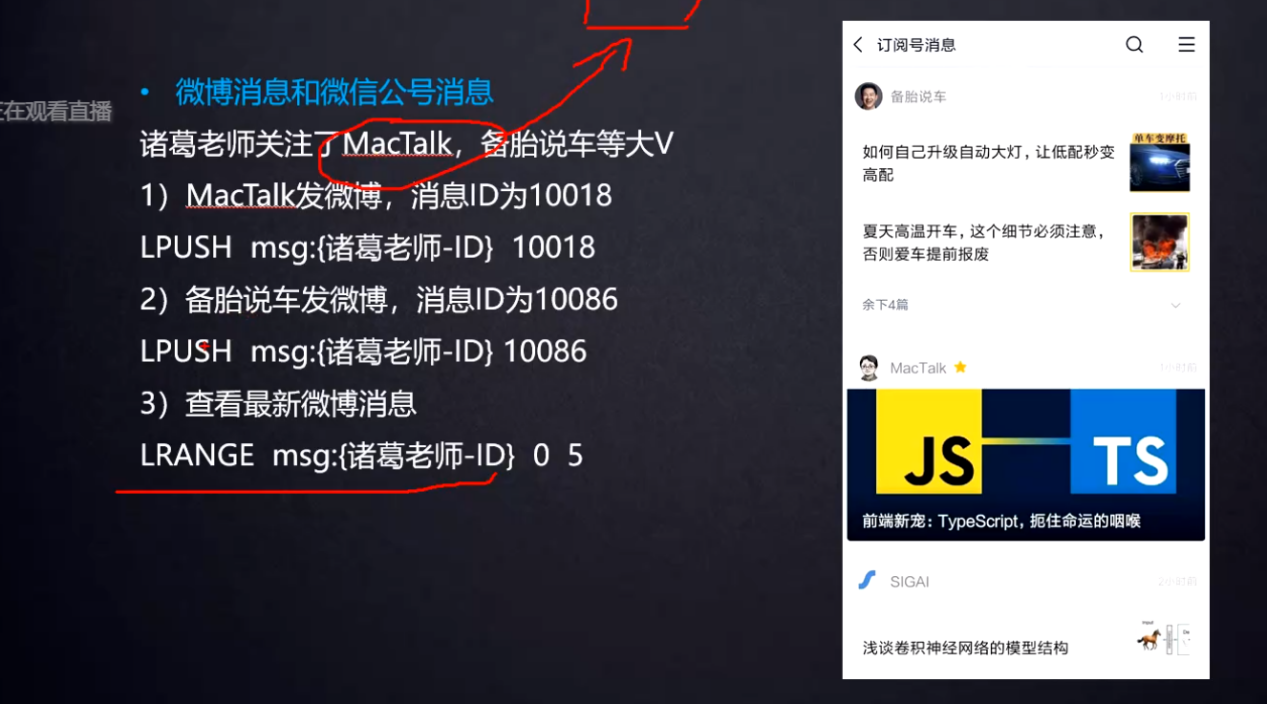


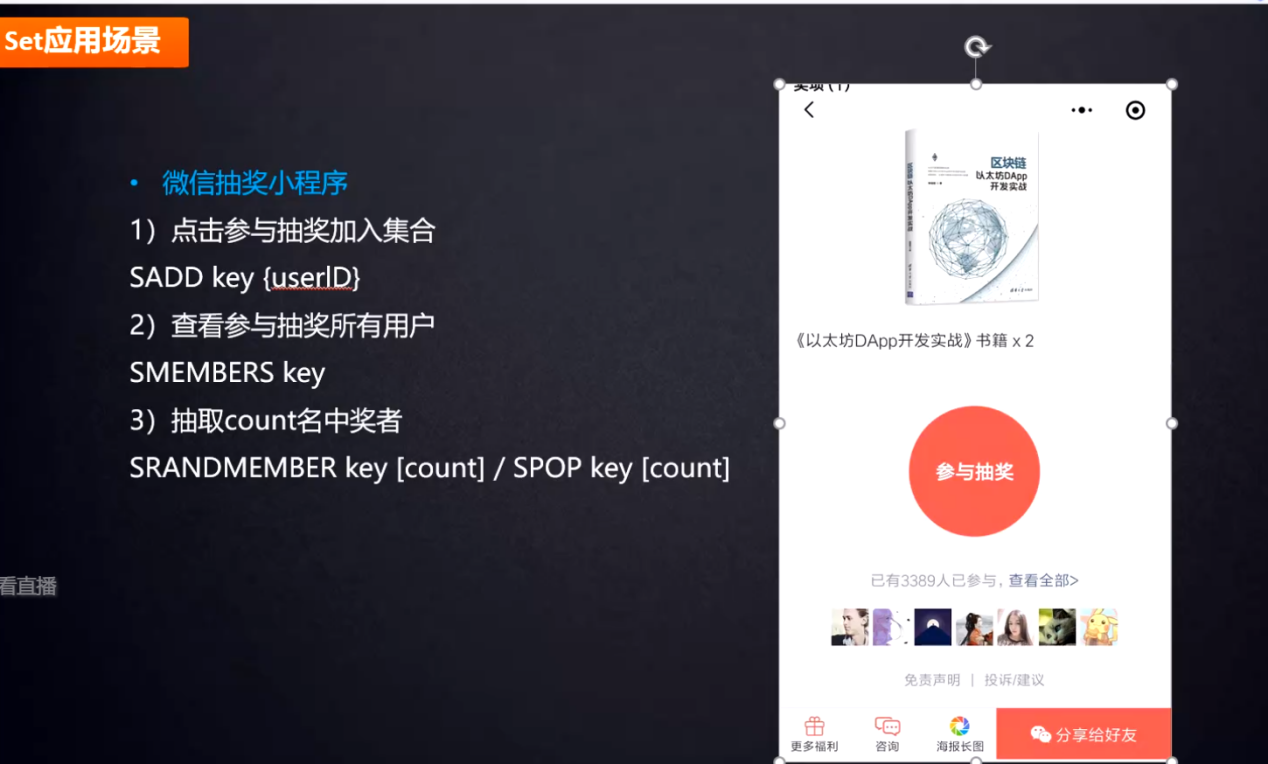
## Hash

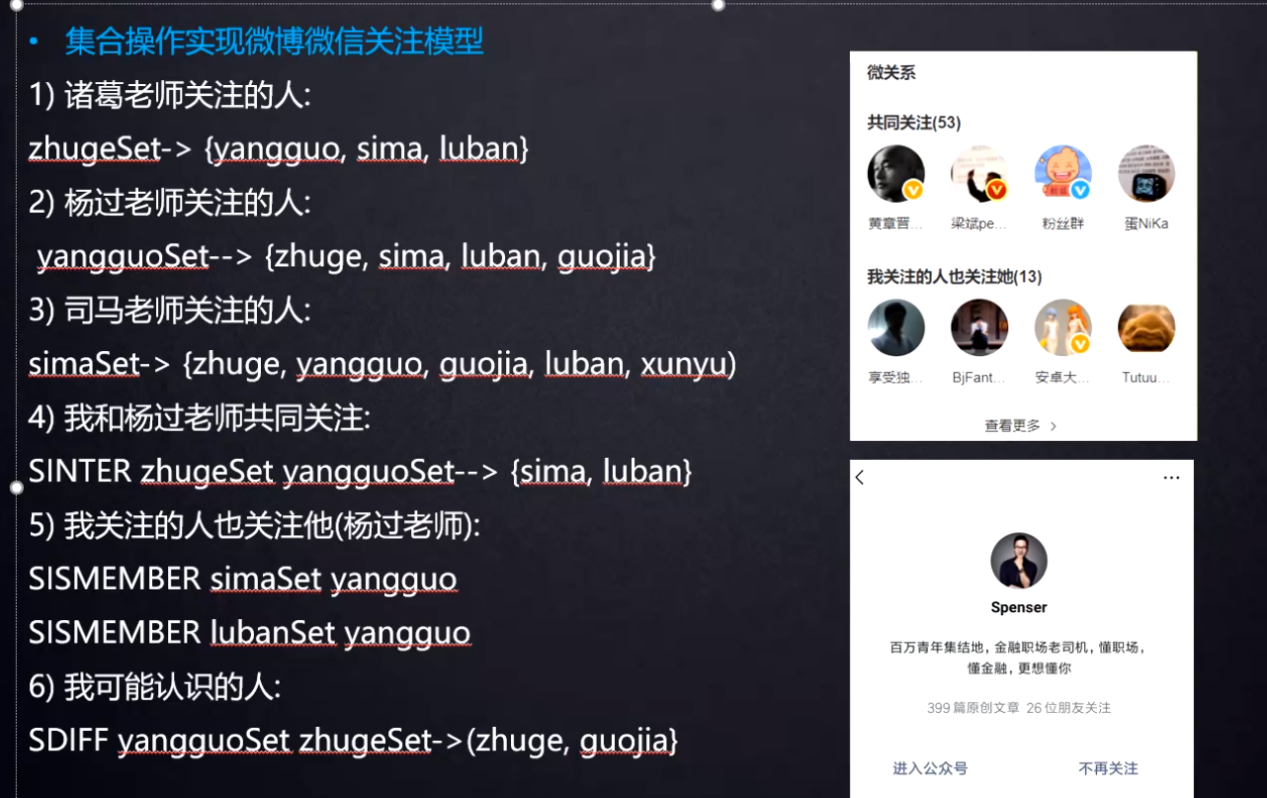


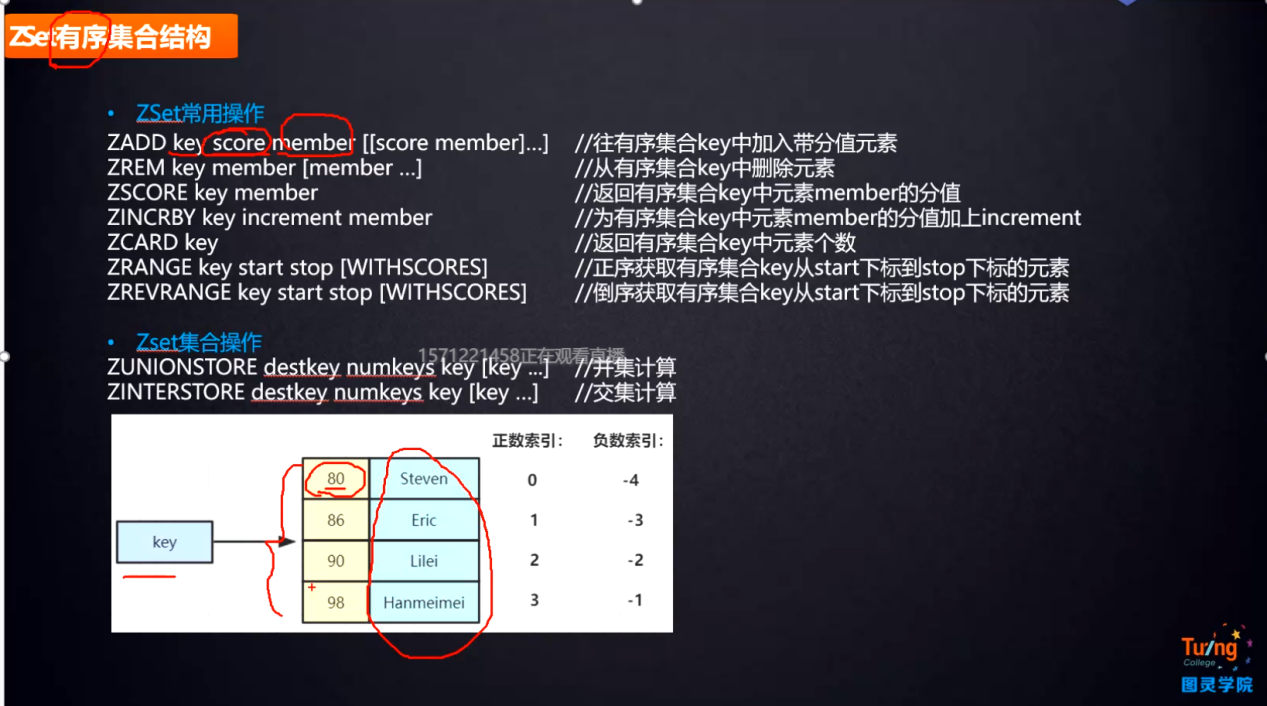






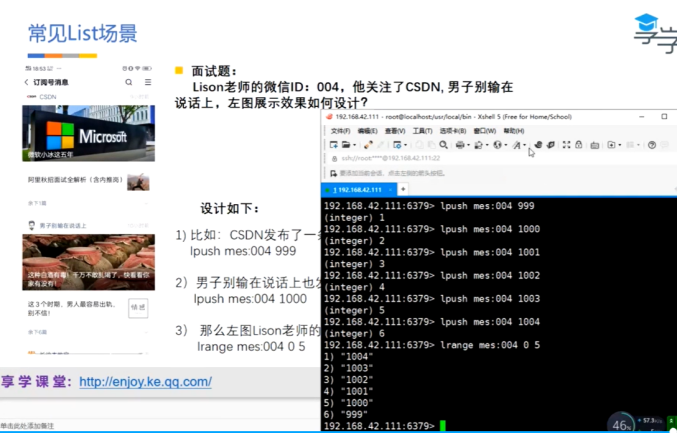






## LIst

使用场景：消息推送



## Set

抽奖:**srandmember**

## Zset

可以做热榜

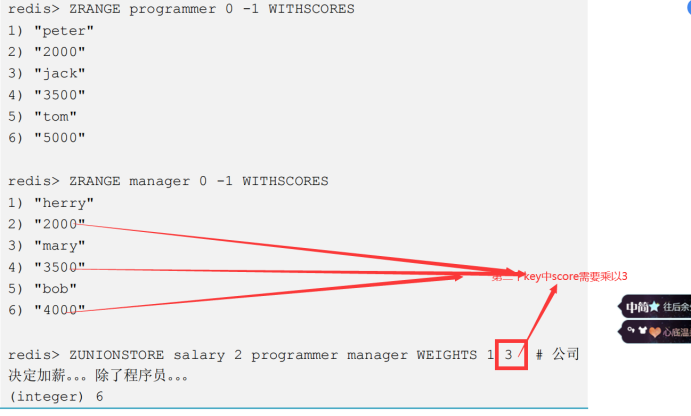
### ZUNIONSTORE

可以用来将两个key中相同的元素的score相加

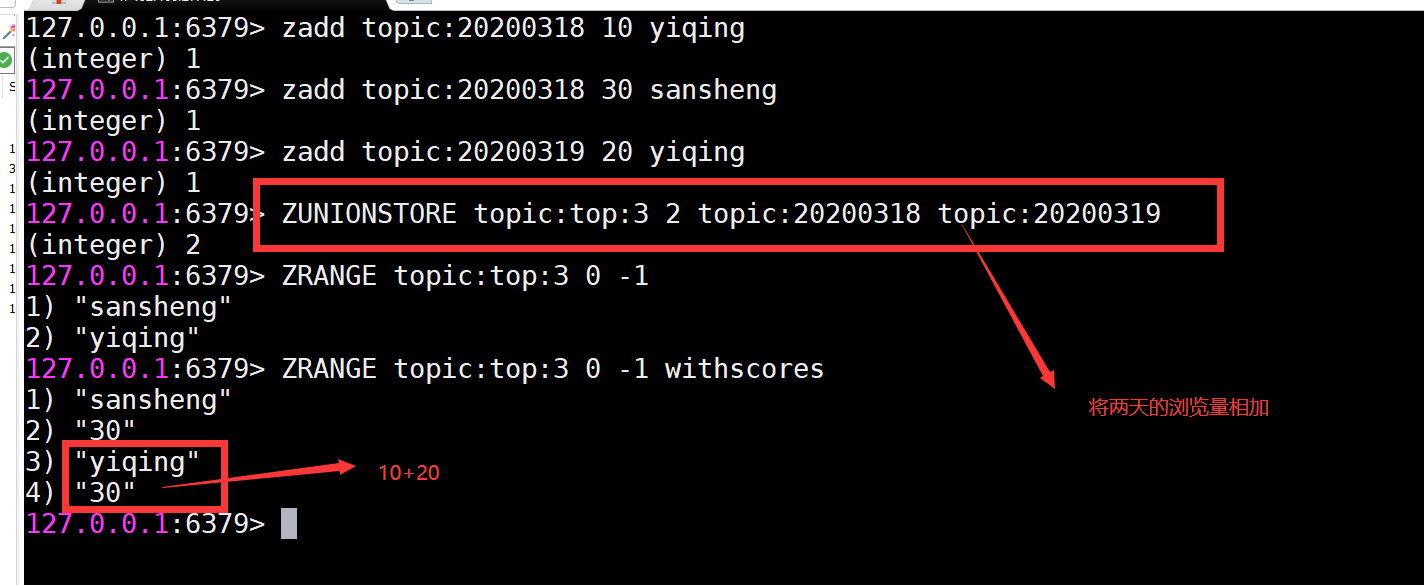
比如keya：1 one

Keyb: 2 one

用这个命令之后，就变为了3=1+2



可以用来将几天的浏览量相加

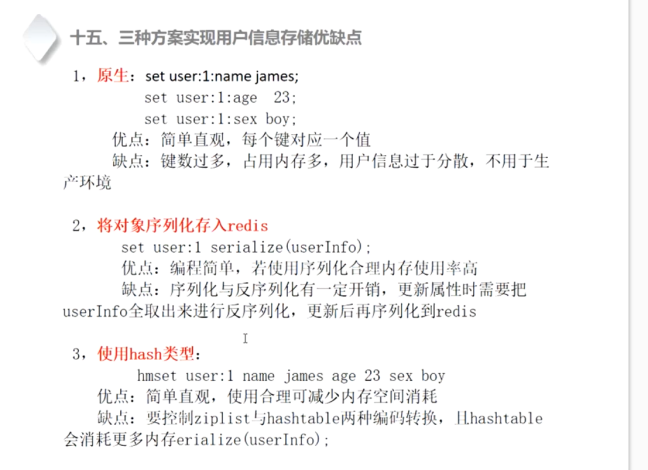


# 特性

速度快:数据存储到内存中，多路复用



# 数据存储的优缺点



# Redis持久化

## rdb

## AOF

# Lua

## 安装教程

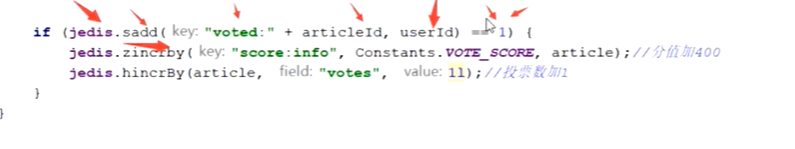
1. 到http://www.lua.org/ftp/下载包
2. 解压
3. yum install libtermcap-devel ncurses-devel libevent-devel readline-devel
4. Make linux test
5. Make install

# 应用

## 投票

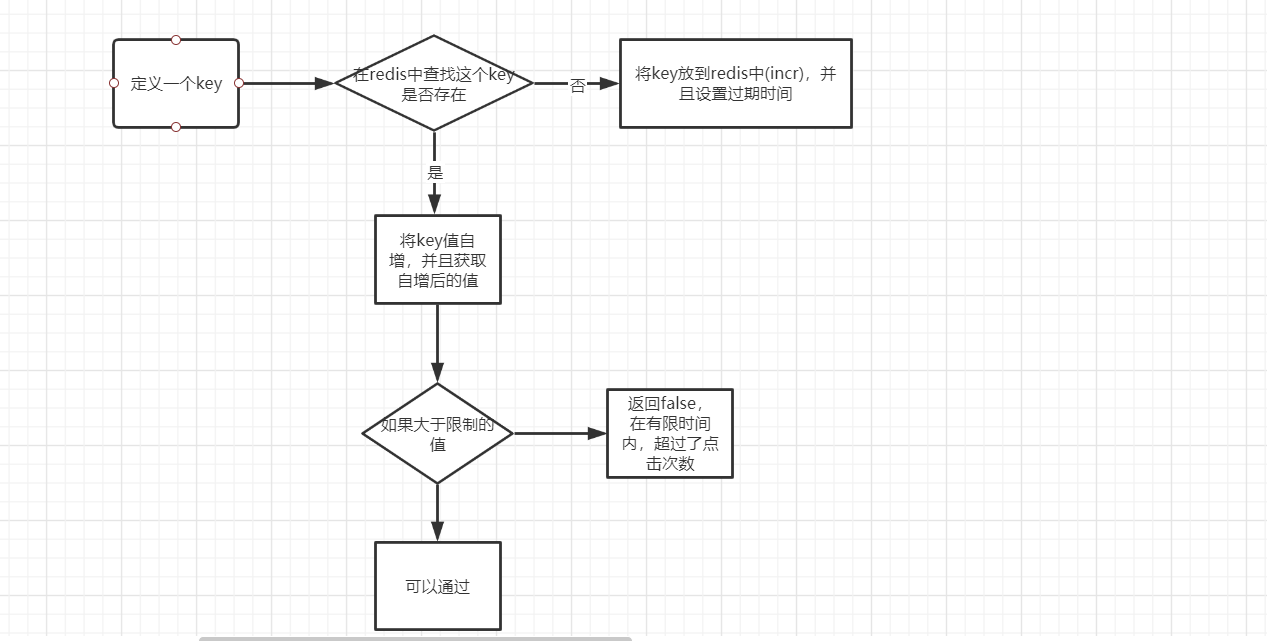
给文章投票

1. 创建文章，将文章保存到redis中
2. 给文章投票，防止一个人头多次，利用set类型来保存
3. 投一票要加分，分数需要排序，用到了zset
4. 还要记录文章的创建时间，按照时间排序，利用zset来保存



# 限流

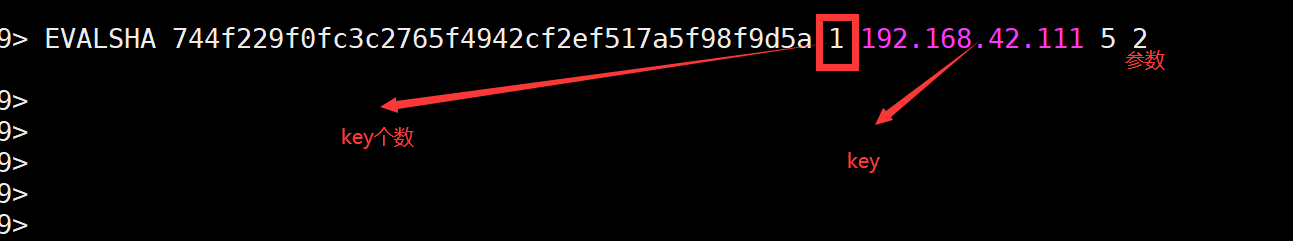
## 流程图



## 限流脚本

|  |
| --- |
| local key = KEYS[1] local limit = tonumber(ARGV[1]) local expire\_time = ARGV[2]  local is\_exists = redis.call("EXISTS", key)  if is\_exists == 1 then   if redis.call("INCR", key) > limit then  return 0  else  return 1  end  else  redis.call("SET", key, 1);  redis.call("EXPIRE", key, expire\_time)  return 1 end |

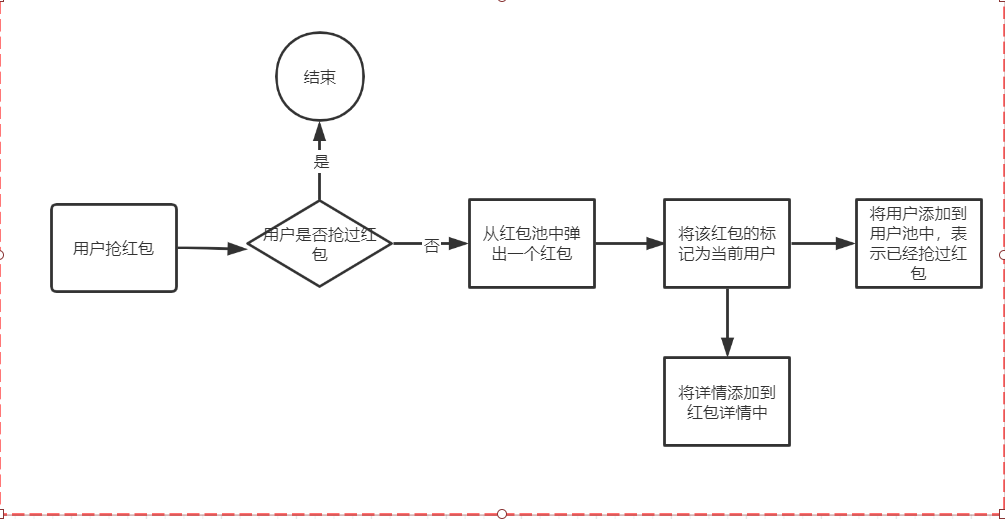
执行:EVALSHA 744f229f0fc3c2765f4942cf2ef517a5f98f9d5a 1 192.168.42.111 5 2



# 抢红包

Csdn：<https://blog.csdn.net/qq_31706095/article/details/105005493>

## 流程图



Key:hongbaoPool,hongbaoDetails,userSetForGrabbed(抢到红包的人)

参数:用户id

## Lua脚本

|  |
| --- |
| local userId = ARGV[1]  if redis.call("SISMEMBER", KEYS[3], userId) ==1 then  return nil  else  local hongbao = redis.call("RPOP", KEYS[1]);  if hongbao then  local x = cjson.decode(hongbao);  x['userId'] = userId;  local re = cjson.encode(x);  redis.call("SADD", KEYS[3], userId)  redis.call("LPUSH", KEYS[2], re)  return re  else  return nil  end  end |

## 创建红包

|  |
| --- |
| Map<String, String> map = new HashMap<>();  for (int i = 0; i < 10; i++) {  map.put("id", "rid" + i);  map.put("money", i +"");  jedisClient.lpush("hongbaoPoolKey", JSONUtils.toJSONString(map));  } |

## 抢红包

|  |
| --- |
| public class Qianghongbao {    private static JedisPool jedisPool = new JedisPool("192.168.27.128", 6379);    private static AtomicInteger atomicInteger = new AtomicInteger(0);    public static void main(String[] args) {  String script = "local userId = ARGV[1]\n" +  "if redis.call(\"SISMEMBER\", KEYS[3], userId) == 1 then\n" +  " return nil\n" +  "else\n" +  " local hongbao = redis.call(\"RPOP\", KEYS[1]);\n" +  " if hongbao then\n" +  " local x = cjson.decode(hongbao);\n" +  " x['userId'] = userId;\n" +  " local re = cjson.encode(x);\n" +  " redis.call(\"SADD\", KEYS[3], userId)\n" +  " redis.call(\"LPUSH\", KEYS[2], re)\n" +  " return re\n" +  " else\n" +  " return nil\n" +  " end\n" +  "end";  int n = 10;  // 实现一起开始抢  CyclicBarrier cyclicBarrier = new CyclicBarrier(n);  //抢完之后，统计多少人抢到了红包  CyclicBarrier cyclicBarrier2 = new CyclicBarrier(n, new Runnable() {  @Override  public void run() {  System.out.println("共" + atomicInteger.get() + "抢到了红包");  }  });  /\* System.out.println( resource.eval(script, 3, "hongbaoPoolKey", "hongbaoDetails",  "userSetForGrabbed", "1234541"));\*/  for (int i = 0; i < n; i++) {  if (i == 19) {  System.out.println();  }  new Thread(new Runnable() {  @Override  public void run() {  Jedis resource = null;  try {  cyclicBarrier.await();  // 每个线程需要获取一个连接  resource = jedisPool.getResource();  String userId = UUID.randomUUID().toString();    Object eval = resource.eval(script, 3, "hongbaoPoolKey", "hongbaoDetails",  "userSetForGrabbed", userId);  if (eval != null) {  atomicInteger.addAndGet(1);  System.out.println(userId + "抢到了红包");  } else {  System.out.println(userId + "没有抢到红包");  }  } catch (InterruptedException e) {  e.printStackTrace();  } catch (BrokenBarrierException e) {  e.printStackTrace();  } catch (Exception e) {  e.printStackTrace();  } finally {  if (resource != null) {// 连接池最多有8个连接，用完记得关闭  resource.close();  }  try {  cyclicBarrier2.await();  } catch (InterruptedException e) {  e.printStackTrace();  } catch (BrokenBarrierException e) {  e.printStackTrace();  }  }    }  }).start();  }  }  } |

# 连接关闭

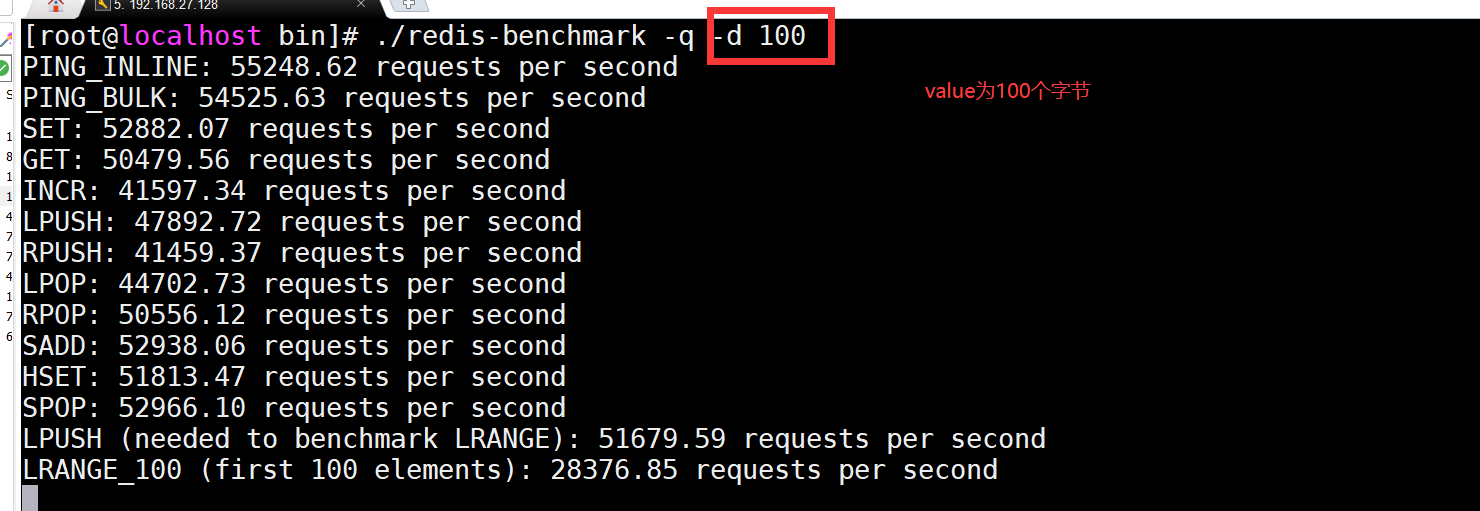
Jedis连接池默认的最大连接数为8个，用完之后记得关闭

# 测试性能

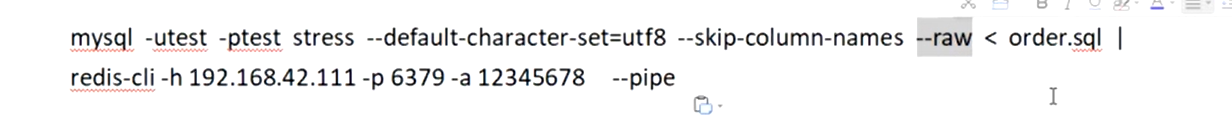
./redis-benchmark -c 100 -n 10000

100个并发连接 10000个请求

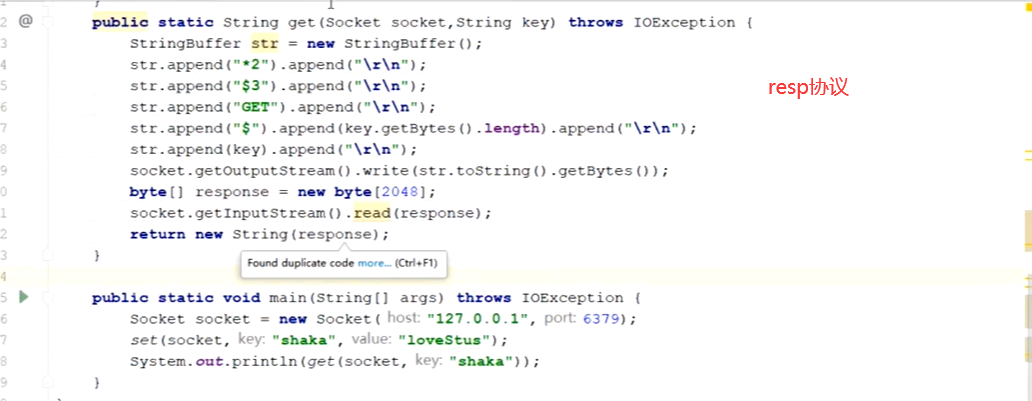
Value为100个字节的测试



# 从mysql往redis导入数据

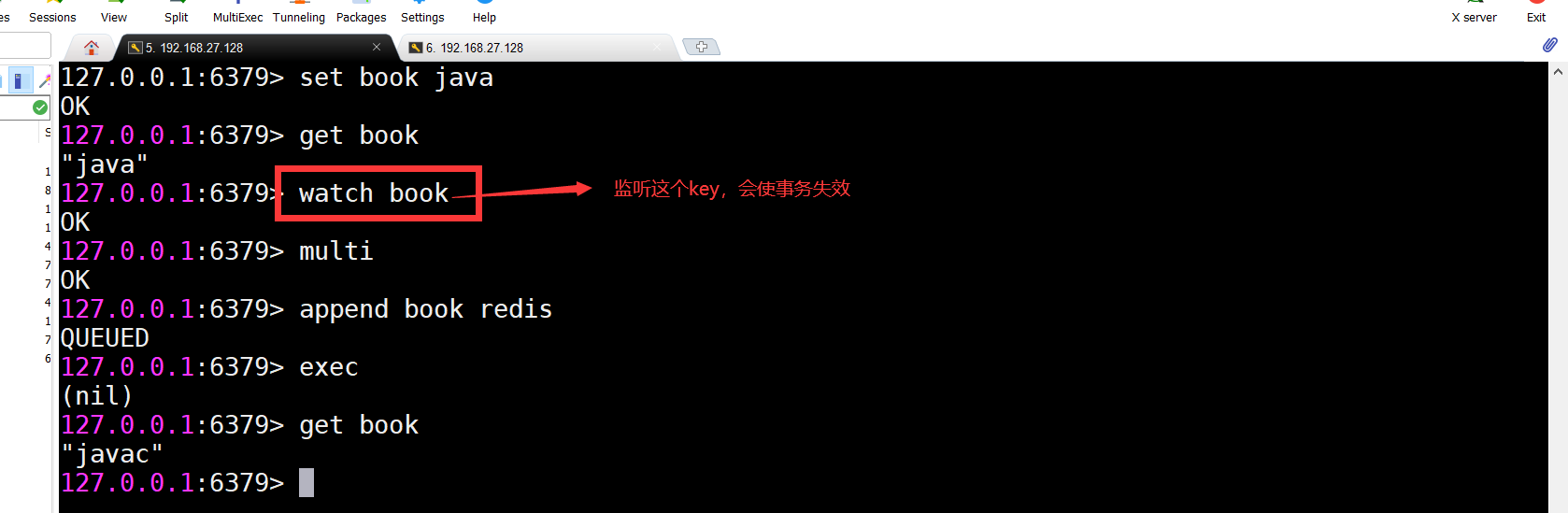


# 手写jedis

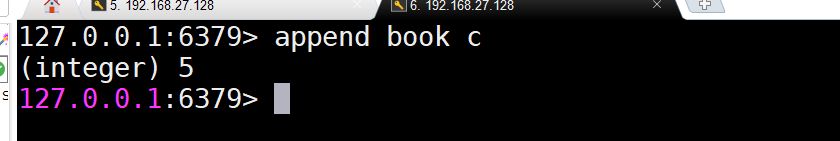


# Watch机制

会话一：监听完key之后，就去第二个会话追加



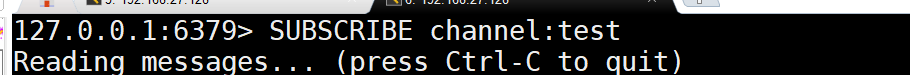
会话二



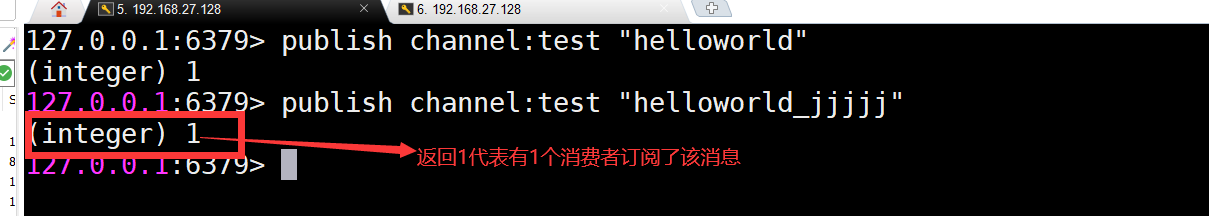
在第一个会话中查询book，发现是javac，在事务中追加的redis并没有追加进去，这是因为监听机制会使事务失效，所以没有追加进去

# 发布订阅

订阅者订阅消息

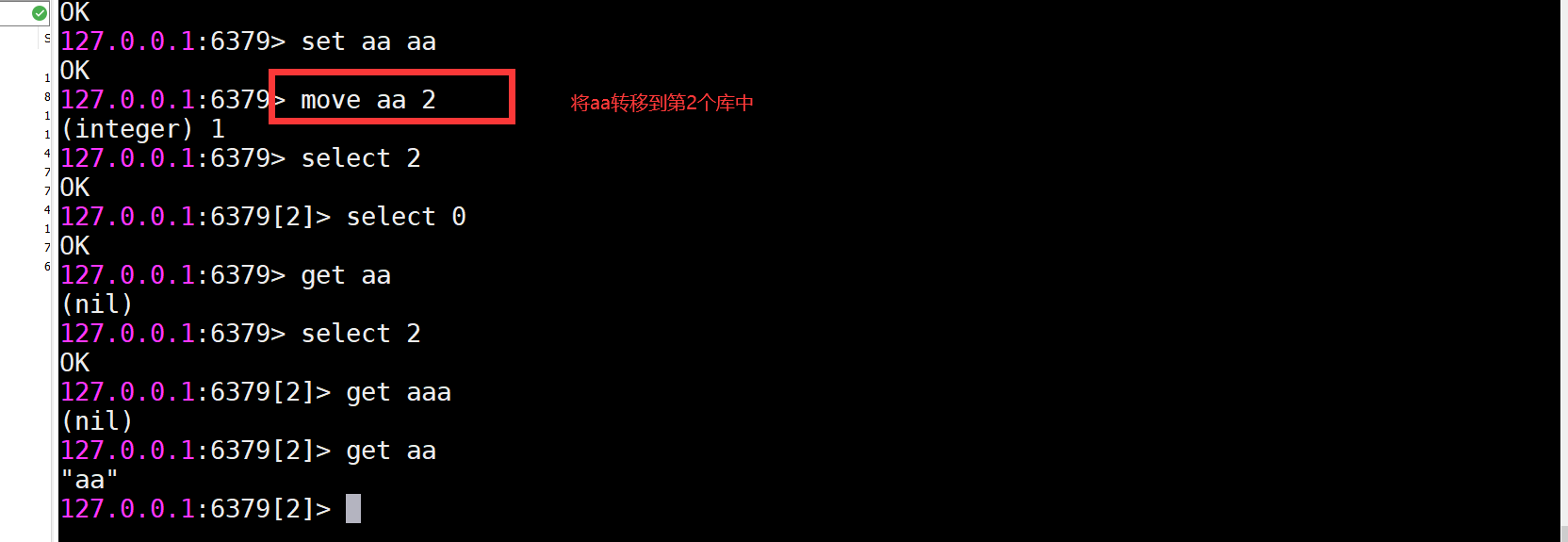


发布者发布消息

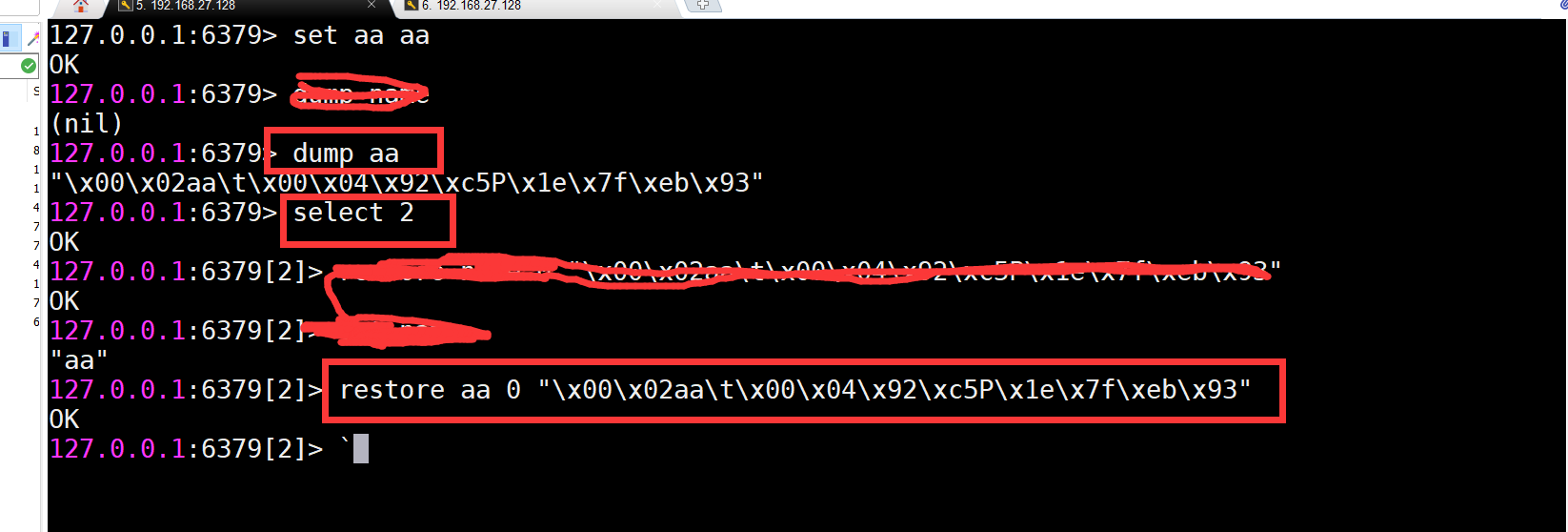


# 键的转移

1、Move key db



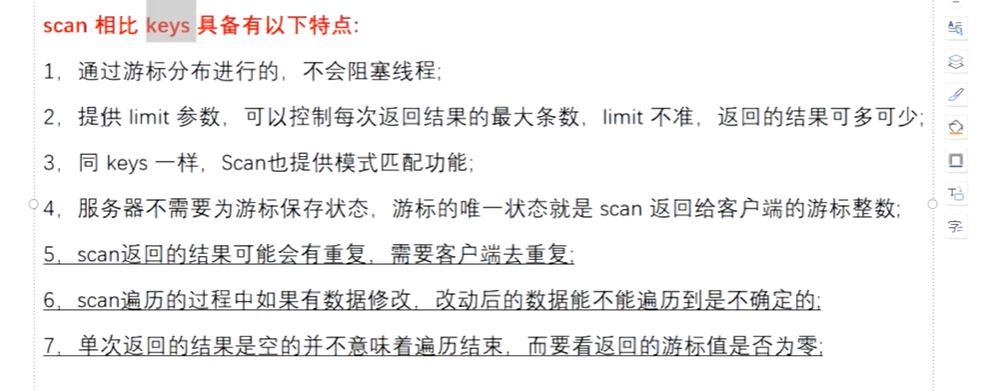
1. dump key



1. migrate



# 遍历key



# Redis高可用

## 主从拓扑

1. 一主多从

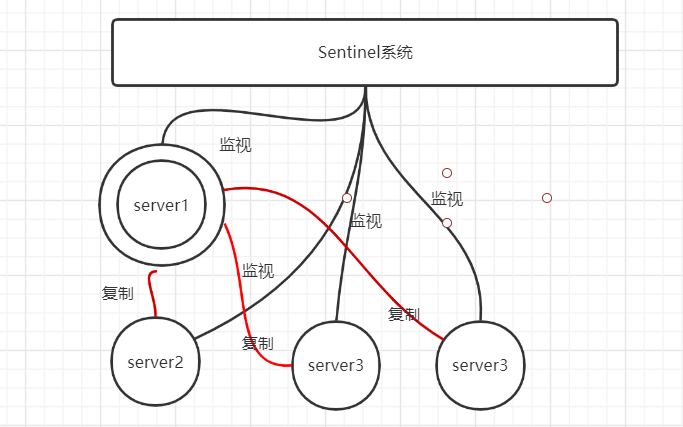
针对”读”较多的情景，”读”由较多个节点来分担，但节点越多，主节点同步到从节点的次数也越多，影响带宽，也影响主节点的稳定性。

查看主从信息

Info replication

## 哨兵

<https://www.cnblogs.com/Eugene-Jin/p/10819601.html>



### 故障转移详细流程

### 

### 作用

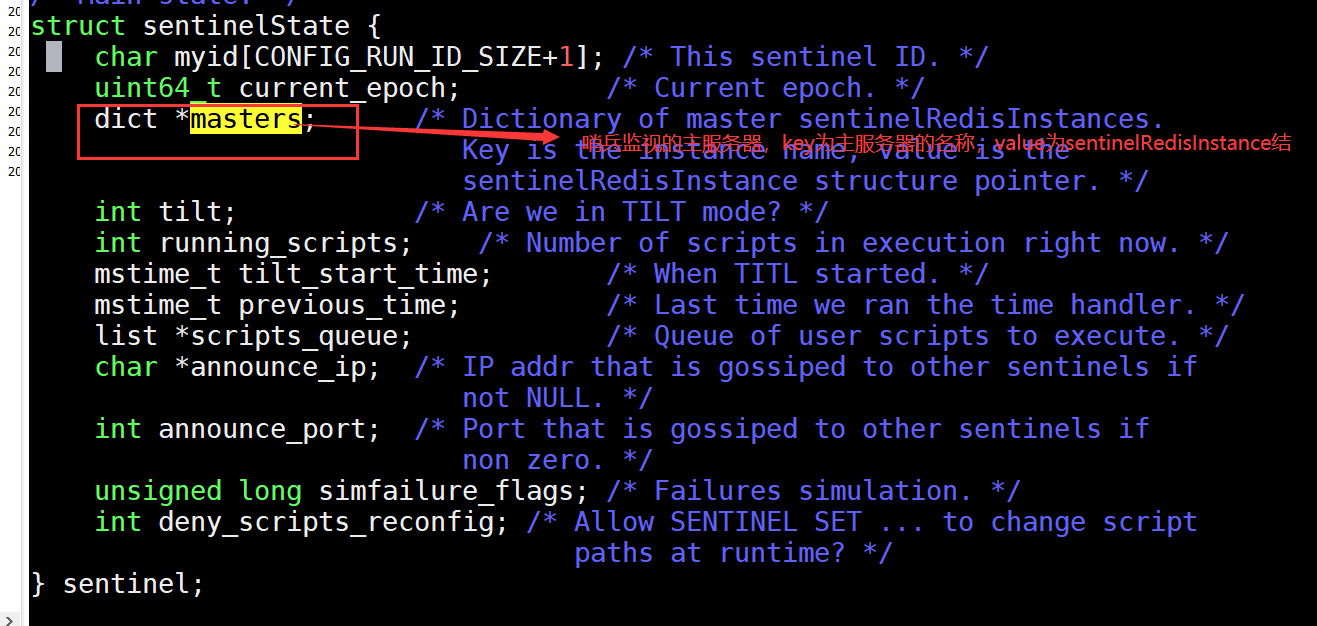
1. 可以监视任意多个主服务器，以及这些主服务器下的从服务器
2. 自动故障转移: 在主服务器下线时，自动将主服务器下的从服务升级为新的主服务器，然后由新的主服务器代替已下线的主服务器继续处理命令请求。

### 重要的属性

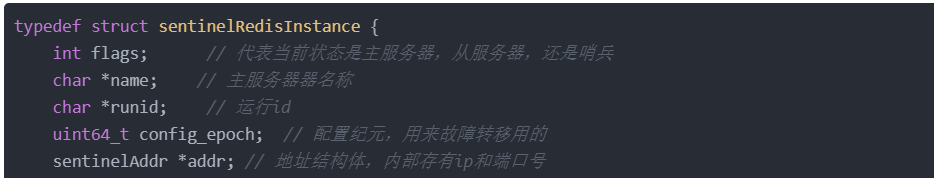
#### quorm

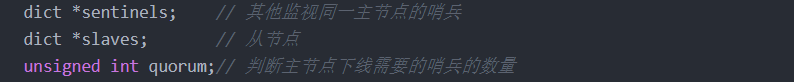
判断主服务器客观下线所需的支持投票数量

### Sentinel状态结构



sentinelRedisInstance的属性





### 如何与被监视的服务器通信

#### 创建连向主服务器的网络连接

#### 获取主服务器信息

#### 获取从服务器信息

# 分布式的解决方案

## 集群

### 缺陷

