**Redis Remote Dictionary Server**

**（远程字典服务器）**

**一、应用**

分布式锁、延时队列、位图、HyperLogLog、布隆过滤器、限流、GeoHash、Scan

**二、原理**

IO现场模型、通信协议、持久化、管道、事务、PubSub、对象压缩、主从同步

**三、集群**

Sentinel,Codis,Cluster

**四、拓展知识**

Stream、Info命令、分布式锁、过期策略、LRU、懒删除、Jedis、

**五、剖析**

字符串、字典、压缩列表，快速列表，跳跃列表，紧凑列表，基数树

**Redis （内存数据库）**

BSD 开源（BSD协议）

支持数据结构类型丰富，字符串（Strings），散列（hashes）列表（lists）集合（sets）

有序集合（sorted sets）与范围查询，（bitmaps hyperloglogs ）和 地里空间（geospatial）索引半径查询。

用途：缓存（StackOverFlow）、数据库（微博）、消息中间件（微博）

[队列先进先出]

KV存储 Redis / Memcashed

文档 MongoDB/CouchDB

列存储 HBase /Cassandra

图存储 Neo4J/FlockDB

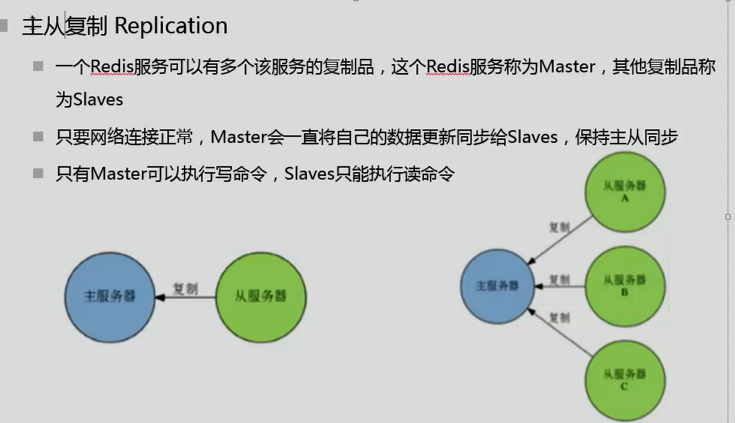
对象存储 Db4o/Versant

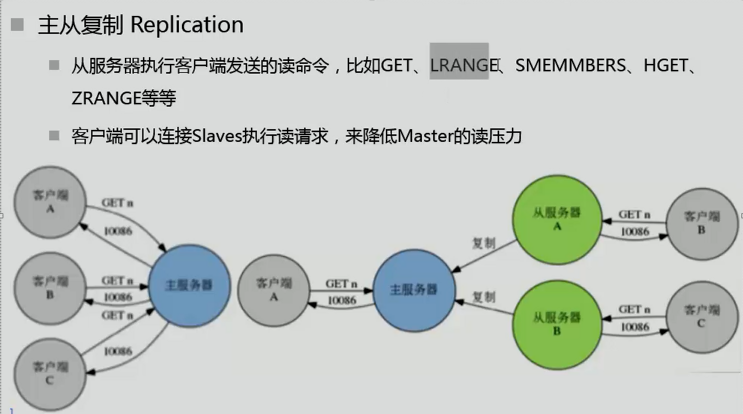
XML数据库 Berkeley DB XML /BaseX

希腊人 MySQL 意大利人

稳定版都是偶数 （索引基于内存的数据库）

主从复制（Replication）





(分布式和数据库的不一样)

分布式的主是去调度的，

从是干活的，

数据库 从是复制

多个节点

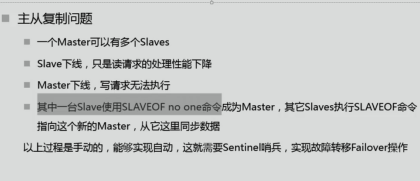
从分担了读的压力



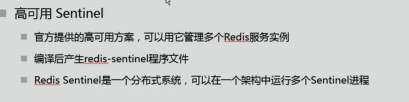
从主变为从

Slaveof <masterip>

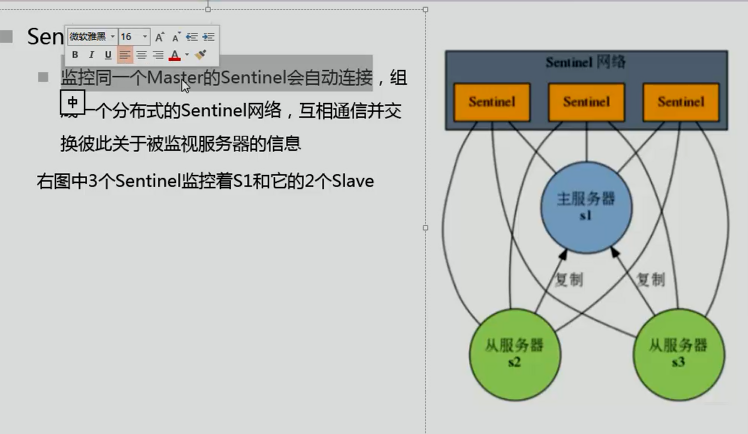
Slaveof 127.0.0.1 6379



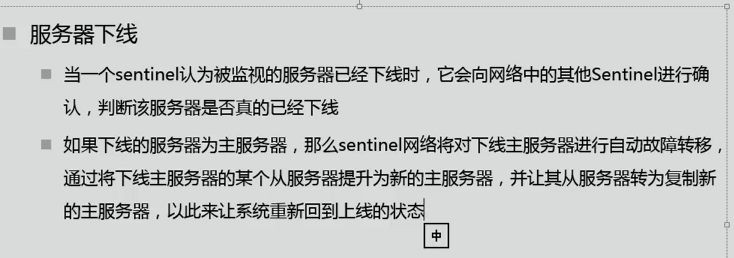
哨兵 master挂了自动变主 高可用（Sentinel热备）

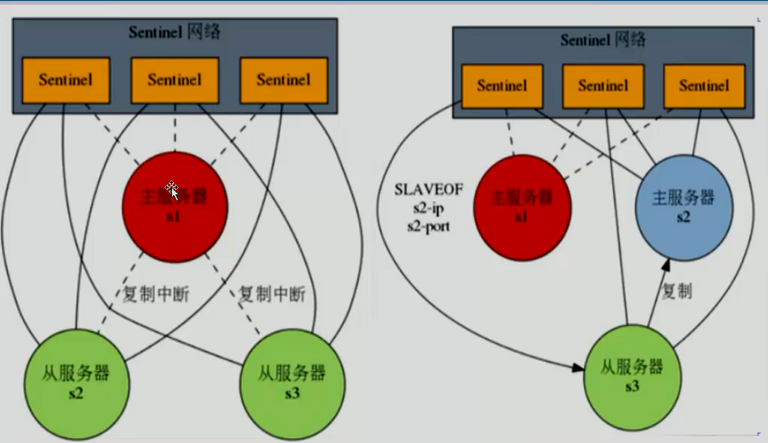


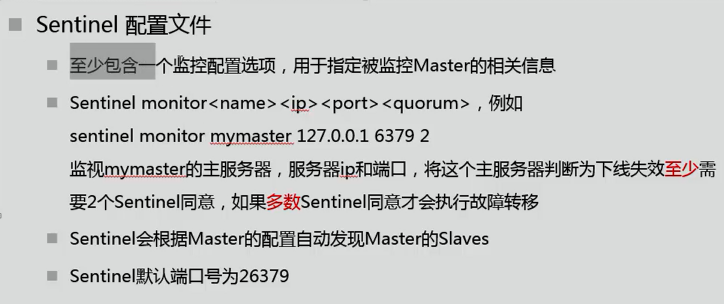




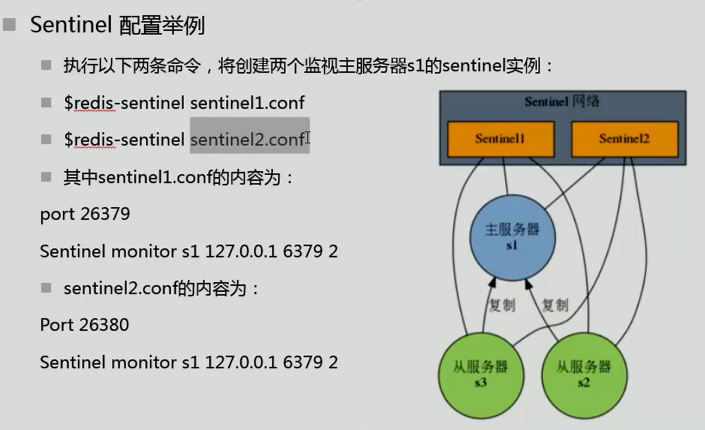
搭建奇数个Sentinel 防脑裂







Quorum 至少（过半数字）认为主挂



主从读写分离

哨兵高可用

Redis 的过期策略和内存淘汰机制

Redis 采用的是定时（期）删除和惰性删除策略

不采用定时的原因：消耗CPU

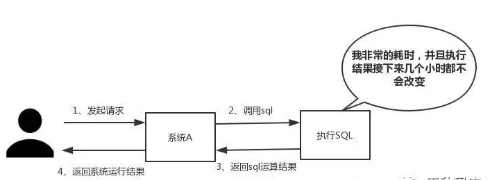
1、为什么使用Redis？

在项目中使用 Redis，主要考虑两个角度：性能和并发。如果只是为了分布式锁这些其他功能，还有其他中间件 Zookpeer 等代替，并非一定要使用 Redis。

**性能**

如下图所示，我们在碰到需要执行耗时特别久，且结果不频繁变动的 SQL，就特别适合将运行结果放入缓存。这样，后面的请求就去缓存中读取，使得请求能够迅速响应。

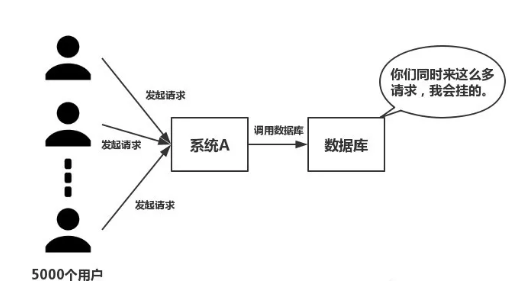
特别是在秒杀系统，在同一时间，几乎所有人都在点，都在下单。。。执行的是同一操作———向数据库查数据。



根据交互效果的不同，响应时间没有固定标准。在理想状态下，我们的页面跳转需要在瞬间解决，对于页内操作则需要在刹那间解决。

**并发**

如下图所示，在大并发的情况下，所有的请求直接访问数据库，数据库会出现连接异常。这个时候，就需要使用 Redis 做一个缓冲操作，让请求先访问到 Redis，而不是直接访问数据库。



使用 Redis 的常见问题

缓存和数据库双写一致性问题

缓存雪崩问题

缓存击穿问题

缓存的并发竞争问题

2、单线程的Redis为什么这么快？

这个问题是对 Redis 内部机制的一个考察。很多人都不知道 Redis 是单线程工作模型。

原因主要是以下三点：

纯内存操作

单线程操作，避免了频繁的上下文切换

采用了非阻塞 I/O 多路复用机制

仔细说一说 I/O 多路复用机制，打一个比方：小明在 A 城开了一家快餐店店，负责同城快餐服务。小明因为资金限制，雇佣了一批配送员，然后小明发现资金不够了，只够买一辆车送快递。

经营方式一

客户每下一份订单，小明就让一个配送员盯着，然后让人开车去送。慢慢的小明就发现了这种经营方式存在下述问题：

时间都花在了抢车上了，大部分配送员都处在闲置状态，抢到车才能去送。

随着下单的增多，配送员也越来越多，小明发现快递店里越来越挤，没办法雇佣新的配送员了。

配送员之间的协调很花时间。

综合上述缺点，小明痛定思痛，提出了经营方式二。

经营方式二

小明只雇佣一个配送员。当客户下单，小明按送达地点标注好，依次放在一个地方。最后，让配送员依次开着车去送，送好了就回来拿下一个。上述两种经营方式对比，很明显第二种效率更高。

在上述比喻中：

每个配送员→每个线程

每个订单→每个 Socket(I/O 流)

订单的送达地点→Socket 的不同状态

客户送餐请求→来自客户端的请求

小明的经营方式→服务端运行的代码

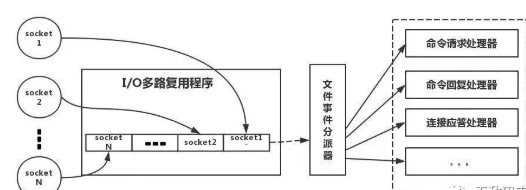
一辆车→CPU 的核数

于是有了如下结论：

经营方式一就是传统的并发模型，每个 I/O 流(订单)都有一个新的线程(配送员)管理。

经营方式二就是 I/O 多路复用。只有单个线程(一个配送员)，通过跟踪每个 I/O 流的状态(每个配送员的送达地点)，来管理多个 I/O 流。

下面类比到真实的 Redis 线程模型，如图所示：



Redis-client 在操作的时候，会产生具有不同事件类型的 Socket。在服务端，有一段 I/O 多路复用程序，将其置入队列之中。然后，文件事件分派器，依次去队列中取，转发到不同的事件处理器中。

### 3、Redis 的数据类型及使用场景

String :

最常规的 set/get 操作，Value 可以是 String 也可以是数字。一般做一些复杂的计数功能的缓存。

Hash:

这里Value 存放的是结构话的对象，比较方便的是操作某个字段。我在单点登陆的时候，就是用这种数据结构存储用户信息，以Cookie作为Key,设置30分钟过期时间，能很好的模拟出类似Session的效果。

List：

使用List 的数据接口，可以做简单的消息队列功能，另外可以利用lrange命令做基于Redis的分页功能，性能好，用户体验好。

Set：

因为Set对方的是一堆不重复的集合。所以可以全局去重的功能。我们的系统一般都是集群部署的，使用JVM自带的Set比较麻烦，另外就是利用交集，并集，差集等操作，可以计算共同喜好，全部的洗好，自己独有的洗好等功能。

Sorted Set

Sorted Set 多了一个权重参数Score ，集合中的元素能够Score进行排序，可以做排行榜应用，取Top N 操作，Sorted Set可以用来做延时任务。

### 4、Redis的过期策略和内存淘汰机制

Redis 是否用到家，从这就能看出来。比如你 Redis 只能存 5G 数据，可是你写了 10G，那会删 5G 的数据。怎么删的，这个问题思考过么？

正解：Redis 采用的是定期删除+惰性删除策略。

**为什么不用定时删除策略**

定时删除，用一个定时器来负责监视 Key，过期则自动删除。虽然内存及时释放，但是十分消耗 CPU 资源。在大并发请求下，CPU 要将时间应用在处理请求，而不是删除 Key，因此没有采用这一策略。

**定期删除+惰性删除如何工作+内存淘汰机制**  
定期删除，Redis 默认每个 100ms 检查，有过期 Key 则删除。需要说明的是，Redis 不是每隔100ms 将所有的 Key 检查一次，而是随机抽取进行检查。如果只采用定期删除策略，会导致很多 Key 到时间没有删除。于是，惰性删除派上用场。

采用定期删除+惰性删除就没其他问题了么

不是的，如果定期删除没删除掉 Key。并且你也没及时去请求 Key，也就是说惰性删除也没生效。这样，Redis 的内存会越来越高。那么就应该采用内存淘汰机制。

在 redis.conf 中有一行配置：

# maxmemory-policy volatile-lru [配置反复无常]

该配置就是配内存淘汰策略的：

noeviction：当内存不足以容纳新写入数据时，新写入操作会报错。

* allkeys-lru：当内存不足以容纳新写入数据时，在键空间中，移除最近最少使用的 Key。（推荐使用，目前项目在用这种）(最近最久使用算法)
* allkeys-random：当内存不足以容纳新写入数据时，在键空间中，随机移除某个 Key。（应该也没人用吧，你不删最少使用 Key，去随机删）
* volatile-lru：当内存不足以容纳新写入数据时，在设置了过期时间的键空间中，移除最近最少使用的 Key。这种情况一般是把 Redis 既当缓存，又做持久化存储的时候才用。（不推荐）
* volatile-random：当内存不足以容纳新写入数据时，在设置了过期时间的键空间中，随机移除某个 Key。（依然不推荐）
* volatile-ttl：当内存不足以容纳新写入数据时，在设置了过期时间的键空间中，有更早过期时间的 Key 优先移除。（不推荐）

### 4、Redis和数据库双写一致性问题

一致性问题还可以再分为最终一致性和强一致性。数据库和缓存双写，就必然会存在不一致的问题。前提是如果对数据有强一致性要求，不能放缓存。我们所做的一切，只能保证最终一致性。

另外，我们所做的方案从根本上来说，只能降低不一致发生的概率。因此，有强一致性要求的数据，不能放缓存。首先，采取正确更新策略，先更新数据库，再删缓存。其次，因为可能存在删除缓存失败的问题，提供一个补偿措施即可，例如利用消息队列。

### 5、如何应对缓存穿透和缓存雪崩问题

这两个问题，一般中小型传统软件企业很难碰到。如果有大并发的项目，流量有几百万左右，这两个问题一定要深刻考虑。缓存穿透，即黑客故意去请求缓存中不存在的数据，导致所有的请求都怼到数据库上，从而数据库连接异常。

**缓存穿透解决方案：**

* 利用互斥锁，缓存失效的时候，先去获得锁，得到锁了，再去请求数据库。没得到锁，则休眠一段时间重试。
* 采用异步更新策略，无论 Key 是否取到值，都直接返回。Value 值中维护一个缓存失效时间，缓存如果过期，异步起一个线程去读数据库，更新缓存。需要做缓存预热(项目启动前，先加载缓存)操作。
* 提供一个能迅速判断请求是否有效的拦截机制，比如，利用布隆过滤器，内部维护一系列合法有效的 Key。迅速判断出，请求所携带的 Key 是否合法有效。如果不合法，则直接返回。
* 缓存雪崩，即缓存同一时间大面积的失效，这个时候又来了一波请求，结果请求都怼到数据库上，从而导致数据库连接异常。

**缓存雪崩解决方案：**

* 给缓存的失效时间，加上一个随机值，避免集体失效。
* 使用互斥锁，但是该方案吞吐量明显下降了。
* 双缓存。我们有两个缓存，缓存 A 和缓存 B。缓存 A 的失效时间为 20 分钟，缓存 B 不设失效时间。自己做缓存预热操作。
* 然后细分以下几个小点：从缓存 A 读数据库，有则直接返回；A 没有数据，直接从 B 读数据，直接返回，并且异步启动一个更新线程，更新线程同时更新缓存 A 和缓存 B。

### 6、如何解决Redis的并发竞争Key问题

这个问题大致就是，同时有多个子系统去 Set 一个 Key。这个时候要注意什么呢？大家基本都是推荐用 Redis 事务机制。

但是我并不推荐使用 Redis 的事务机制。因为我们的生产环境，基本都是 Redis 集群环境，做了数据分片操作。你一个事务中有涉及到多个 Key 操作的时候，这多个 Key 不一定都存储在同一个 redis-server 上。因此，Redis 的事务机制，十分鸡肋。

如果对这个 Key 操作，不要求顺序

这种情况下，准备一个分布式锁，大家去抢锁，抢到锁就做 set 操作即可，比较简单。

如果对这个 Key 操作，要求顺序

假设有一个 key1，系统 A 需要将 key1 设置为 valueA，系统 B 需要将 key1 设置为 valueB，系统 C 需要将 key1 设置为 valueC。

期望按照 key1 的 value 值按照 valueA > valueB > valueC 的顺序变化。这种时候我们在数据写入数据库的时候，需要保存一个时间戳。

假设时间戳如下：

系统 A key 1 {valueA  3:00}

系统 B key 1 {valueB  3:05}

系统 C key 1 {valueC  3:10}

那么，假设系统 B 先抢到锁，将 key1 设置为{valueB 3:05}。接下来系统 A 抢到锁，发现自己的 valueA 的时间戳早于缓存中的时间戳，那就不做 set 操作了，以此类推。其他方法，比如利用队列，将 set 方法变成串行访问也可以。

**Zongjie**

Redis 在国内各大公司都能看到其身影，比如我们熟悉的新浪，阿里，腾讯，百度，美团，小米等。学习 Redis，这几方面尤其重要：Redis 客户端、Redis 高级功能、Redis 持久化和开发运维常用问题探讨、Redis 复制的原理和优化策略、Redis 分布式解决方案等。

## 在分布式数据库中CAP原理+BASE

#### 关系型数据库

关系型数据库遵循ACID规则

事务在英文中是transaction，和现实世界中的交易很类似，它有如下四个特性：

1、A (Atomicity) 原子性

原子性很容易理解，也就是说事务里的所有操作要么全部做完，要么都不做，事务成功的条件是事务里的所有操作都成功，只要有一个操作失败，整个事务就失败，需要回滚。比如银行转账，从A账户转100元至B账户，分为两个步骤：1）从A账户取100元；2）存入100元至B账户。这两步要么一起完成，要么一起不完成，如果只完成第一步，第二步失败，钱会莫名其妙少了100元。

2、C (Consistency) 一致性

一致性也比较容易理解，也就是说数据库要一直处于一致的状态，事务的运行不会改变数据库原本的一致性约束。

3、I (Isolation) 独立性

所谓的独立性是指并发的事务之间不会互相影响，如果一个事务要访问的数据正在被另外一个事务修改，只要另外一个事务未提交，它所访问的数据就不受未提交事务的影响。比如现有有个交易是从A账户转100元至B账户，在这个交易还未完成的情况下，如果此时B查询自己的账户，是看不到新增加的100元的

4、D (Durability) 持久性

持久性是指一旦事务提交后，它所做的修改将会永久的保存在数据库上，即使出现宕机也不会丢失。

#### CAP:

C:Consistency（强一致性）

A:Availability（可用性）

P:Partition tolerance（分区容错性）

 CAP理论的核心是：一个分布式系统不可能同时很好的满足一致性，可用性和分区容错性这三个需求，

最多只能同时较好的满足两个。

因此，根据 CAP 原理将 NoSQL 数据库分成了满足 CA 原则、满足 CP 原则和满足 AP 原则三 大类：

CA - 单点集群，满足一致性，可用性的系统，通常在可扩展性上不太强大。

CP - 满足一致性，分区容忍必的系统，通常性能不是特别高。

AP - 满足可用性，分区容忍性的系统，通常可能对一致性要求低一些。

#### BASE就是为了解决关系数据库强一致性引起的问题而引起的可用性降低而提出的解决方案。

BASE其实是下面三个术语的缩写：

    基本可用（Basically Available）

    软状态（Soft state）

    最终一致（Eventually consistent）

它的思想是通过让系统放松对某一时刻数据一致性的要求来换取系统整体伸缩性和性能上改观。为什么这么说呢，缘由就在于大型系统往往由于地域分布和极高性能的要求，不可能采用分布式事务来完成这些指标，要想获得这些指标，我们必须采用另外一种方式来完成，这里BASE就是解决这个问题的办法

#### 分布式系统+集群

分布式系统（distributed system）

 由多台计算机和通信的软件组件通过计算机网络连接（本地网络或广域网）组成。分布式系统是建立在网络之上的软件系统。正是因为软件的特性，所以分布式系统具有高度的内聚性和透明性。因此，网络和分布式系统之间的区别更多的在于高层软件（特别是操作系统），而不是硬件。分布式系统可以应用在在不同的平台上如：Pc、工作站、局域网和广域网上等。

**能干什么：**

内存存储和持久化：Redis支持异步将内存中的数据写到硬盘上，同时不影响继续服务

取最新N个数据的操作，如：可以将最新的10条评论的ID放在Redis的List集合里面

模拟类似于HttpSession这种需要设定过期时间的功能

发布，订阅消息系统 （MQ ,Active）

定时器，计数器

Linux 安装Redis

Tar –zxvf redis.tar.gz

chmod -R 755 redis

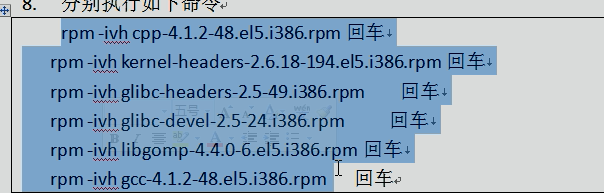
make 命令错误

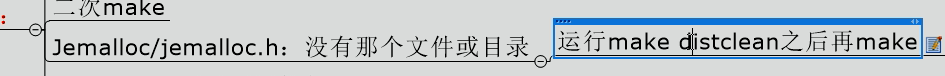
缺少 GCC

安装 Gcc

有网

yum install gcc-c++ （两个是连着的）

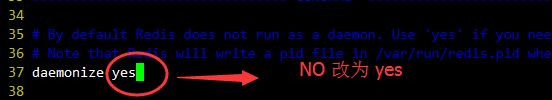




make distclean

make

cp myredis/redis.conf



打开Redis 服务端

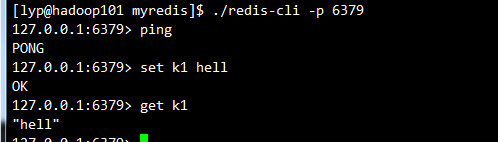
Cp redis\* ../myredis

./redis-server redis.conf

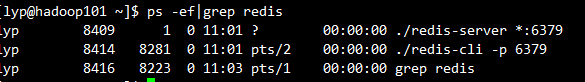
打开客户端

./redis-cli –p 6379

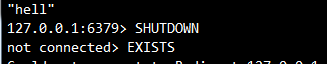
检测是否成功



检查进程



关闭

****

**查看进程**

****

**测试redis 的性能**

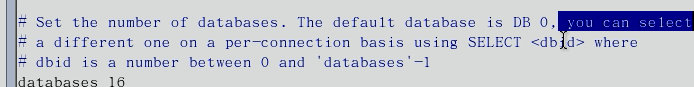
****

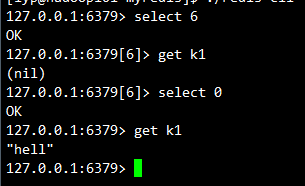
**Redis**

**单进程模式来处理客户端的请求，对读写等事件的响应**

**通过epoll函数的包装来做到的 ，redis的实际处理速度完全依赖主进程的效率**

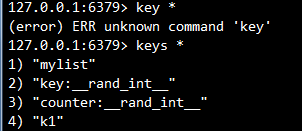
**Redis默认安装的数据库<select >**

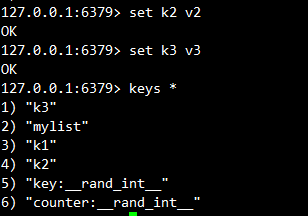
****

****

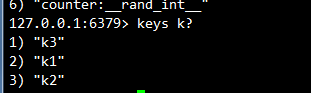
**HashSet 的底层是HashMap Key 重要**

****

****

****

**支持anter 风格**

****

**情况当前库**

****

**Redis 的五大数据类型**

**键(Key)**

**字符串(String)**

**列表(List) Object arrayList linkedList 双向列表<node>**

**哈希(Hash)**

**集合(Set)**

**有序集合(Zset)**

**一个value 最多可以是512M (格力空调晚上1度电)**

**Key (键)**

**Exists key判断某个Key 是否存在**

**Move key db 当前库就没有了，北移除了**

**Expire key 秒钟：为给定的key设置过期时间**

**Ttl key 查看还有多少秒过期 -1 表示永不过期，-2表示已过期**

**TTL(time to live) 还能活多久**

**1、DEL key 该命令用于在key存在是删除key**

**2、DUMP key 序列化给定的key 并返回被序列化的值**

**3、EXISTS key 检查key是否存在**

**4、EXPIRE key 为给定key设置过期时间**

**5、EXPIREAT key timestamp EXPIREAT的作用和EXPIRE类似都是为key设置过期时间，不同在于EXPIREAT命令接受的时间参数是UNIX时间戳**

**6、PEXPIRE key milliseconds 设置key的过期时间为亿以毫秒计**

**7、PEXPIREAT at设置key过期时间戳以毫秒计**

**8、KEYS pattern 查找所有符合给定模式（pattern）的key**

**9、MOVE key db 将当前数据库的key 移动到给定的数据库的db当中**

**10、PERSIST key 移除keyd 过期时间**

**11、PTTL key 以毫秒为单位返回key的剩余过期时间**

**12、TTL key**

**13、TYPE key 查看key是什么类型**

**String (字符串)**

**Set 、get、del、append、strlen**

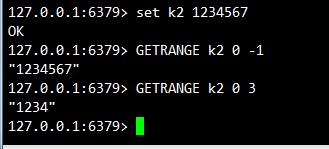
**Incr、decr、incrby、decrby 一定要是数字才能加减[对线程进行操作]**

**Setex (set with expire) 键妙值(setnx(set if not exist))**

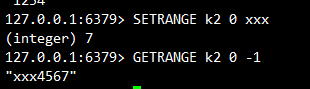
**Mset、mget、msetnx**

**Getset(先get再set)**

**GetRANGE**

****

**SETRANGET**

****

**127.0.0.1:6379> SETNX k4 20 v4**

**(error) ERR wrong number of arguments for 'setnx' command**

**127.0.0.1:6379> SETNX k11 v11**

**(integer) 1**

**127.0.0.1:6379> SETNX k4 11**

**127.0.0.1:6379> MSET k1 v1 k2 v2 k3 v3**

**OK**

**127.0.0.1:6379> MGET k1 k2 k3 k**

**1) "v1"**

**2) "v2"**

**3) "v3"**

**List (列表)**

**1、BLPOP key 1 [key2] timeout**

**移除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弹出元素为止**

**2、BRPOP key1 [key2] timeout 移除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一列表中并返回它；如果列表没有元素会阻塞列表直到等待超时或发现弹出元素为止**

**4、LINDEX key index 通过索引获取列表的元素**

**5、LINSERT key BEFOREIAFTER pivot value 在列表的元素前面或后面插入元素**

**6、LIEN key 获取列表长度**

**7、LPOP key 移动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 将一个或多个值插入到已存在 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 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)，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全部被删除**

**14、RPOP key 移除并获取列表最后一个元素**

**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**

**127.0.0.1:6379> LRANGE list0 0 -1 左进右出**

**1) "5"**

**2) "4"**

**3) "3"**

**4) "2"**

**5) "1"**

**127.0.0.1:6379> RPUSH list02 1 2 3 4 5 67**

**(integer) 6**

**127.0.0.1:6379> LRANGE list02 0 -1**

**1) "1"**

**2) "2"**

**3) "3"**

**4) "4"**

**5) "5"**

**6) "67"**

**127.0.0.1:6379>**

**127.0.0.1:6379> LRANGE list01 0 -1**

**1) "5"**

**2) "4"**

**3) "3"**

**4) "2"**

**5) "1"**

**6) "1"**

**7) "2"**

**8) "3"**

**9) "4"**

**10) "5"**

**127.0.0.1:6379> LPOP list01**

**"5" 栈顶并移除**

**127.0.0.1:6379> RPOP list01**

**"5" 栈底**

**127.0.0.1:6379> RPOP list01**

**"4"**

**127.0.0.1:6379> LRANGE list01 0 -1**

**1) "4"**

**2) "3"**

**3) "2"**

**4) "1"**

**5) "1"**

**6) "2"**

**7) "3"**

**127.0.0.1:6379> LINDEX list01 3**

**"1" 索引位置**

**127.0.0.1:6379> LINDEX list01 1**

**"3"**

**127.0.0.1:6379> LINDEX list01 0**

**"4"**

**127.0.0.1:6379> LLEN list01 长度**

**(integer) 7**

**127.0.0.1:6379> RPUSH list03 1 1 2 2 3 3 4 4 4 5 5 5**

**(integer) 12**

**127.0.0.1:6379> LREM list03 2 4 删除2个 4**

**(integer) 2**

**127.0.0.1:6379>**

**127.0.0.1:6379> LPUSH list01 1 2 3 4 5 6 7 8**

**(integer) 8**

**127.0.0.1:6379> LTRIM list01 0 3 截取 0-3 是下标**

**OK**

**127.0.0.1:6379> LRANGE list 01 0 -1**

**1) "8"**

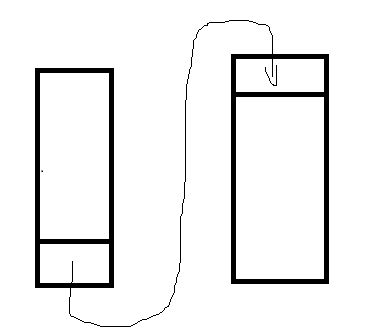
**2) "7"**

**3) "6"**

**4) "5"**

**127.0.0.1:6379>**

**RPOPLPUSH**

****

**127.0.0.1:6379> LINSERT list before x java**

**(integer) 10**

**127.0.0.1:6379> LRANGE list 0 -1**

**1) "1"**

**2) "2"**

**3) "3"**

**4) "4"**

**5) "java"**

**6) "x"**

**7) "5"**

**8) "6"**

**9) "7"**

**10) "8"**

**总结：**

**它是一个字符串链表，left right 都可以插入、添加**

**如果键不存在，创建新的链表**

**如果键已存在，新增内容**

**如果值全移除，对应的键也就消失**

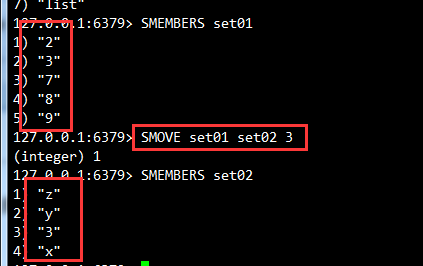
**链表的操作无论是头和尾效率极高，但如果是中间元素操作，效率会惨淡**

**Set**

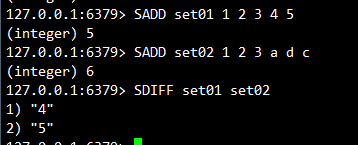
**交集：sinter 差集：sdiff 并集：sunion**

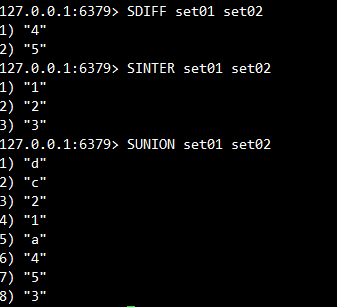
1. **SADD key member 1 [member2] 向集合添加一个或多个成员**
2. **SCADD key 获取集合的成员数**
3. **SDIFF key1 [key2] 返回给定所有集合的差集**
4. **SDIFFSTORE destination key1 [key2] 返回给定所有集合的差集并存储在destination 中**
5. **SINTER key1 [key2] 返回给定所有集合的交集**
6. **SINTERSTORE destination key1 [key2] 返回给dig所有集合的交集并存储在destination**
7. **SISMEMBER 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 出栈**
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**
15. **SSCAN key cursor [MATCH pattern] [COUNT count] 迭代集合中的元素**

**将set01 的数移动到set02中**

****

**在第一个SET里面而不在第二个set里面**

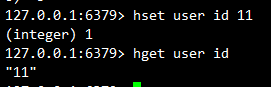
****

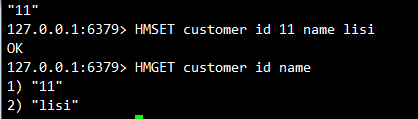
****

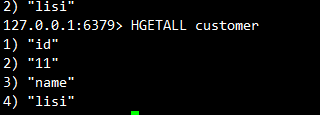
**Hash <K,V> ---map<String ,Object>**

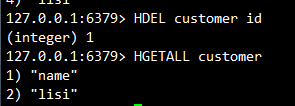
**K，V 模式不变，但V 是一个键值对**

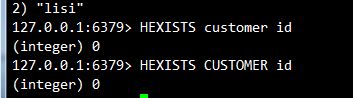
* 1. **HDEL key field2 [field2] 删除一个或多个哈希表字段**
  2. **HEXISTS key field 查看哈希表key中,指定的字段是否存在**
  3. **HGET key field 获取在哈希表中指定key的字段**
  4. **HGETALL key获取在哈希表中key 的所有字段和值**
  5. **HINCRBY key field increment 为哈希表key中的指定字段的整数值加上增量increment**
  6. **HINCRBYFLOAT key field increment 为哈希表key中的指定字段浮点值加上增量 increment**
  7. **HKEYS key 获取所有哈希表中的字段**
  8. **HLEN key 获取哈希表中的字段**
  9. **MGET key field 1 [field2] 获取所有给定字段的值**
  10. **HMSET key field1 value [field2 value2] 同时将多个filed-value（域-值）对设置到哈希表key中**
  11. **HSFT key field value 将哈希表中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**

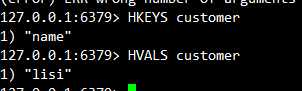
****

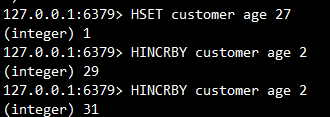
****

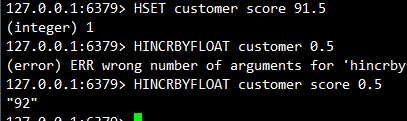
****

****

****

****

****

****

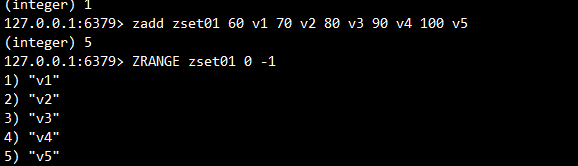
**Zset(有序的set集合)**

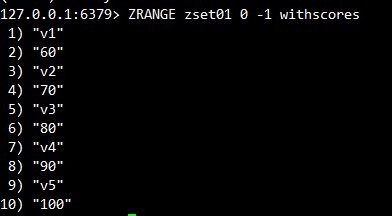
1. **ZADD key score1 member1 [score2 member2] 想有序集合中添加一个或多个成员，或者更新已存在成员的分数**
2. **ZCARD key 获取有序集合 的成员数**
3. **ZCOUNT key min max 计算在有序集合中指定区间分数的成员数**
4. **ZINCRBY key increment member有序集合中对指定成员分数加上增量increment**
5. **ZINTERSTORE destination numkeys key [key] 计算给定的一个或多个有序集合的交集并将结果存储在新的有序集合key中**
6. **ZLEXCOUNT 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指定区间内的成员**
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] 移除须有集合中的一个或多个成员**
12. **ZREMRANGEBYLEX key min max 移除所有集合中给定的字段区间的所有成员**
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] 返回有序集合中指定区间内成员，通过索引，分数从高到底**
16. **ZREVRANGEBYSCORE key max min [WITHSCORES] 返回有序集合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，有序集成员分数值从大到小排序**
18. **ZSCORE key member 返回有序集中，成员的分数值**
19. **ZUNIONSTORE destination numberkeys key [key …] 计算给定的一个或多个有序集合的并集，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）**

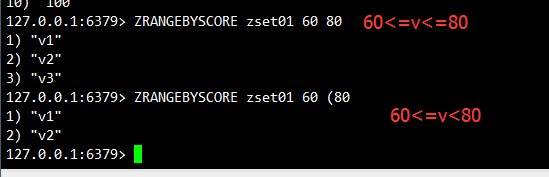
**Sorted set 和Zset 的区别 在set的基础上 加一个score**

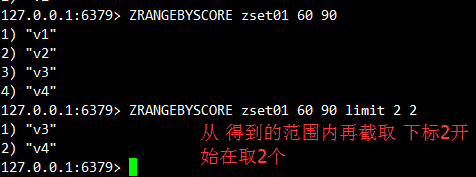
**之前set 是 k1 v1 v2 v3 ,**

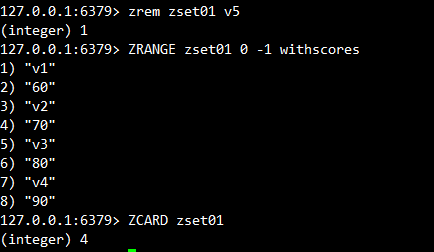
**现在是zset 是 k1 score1 v1 score2 v2**

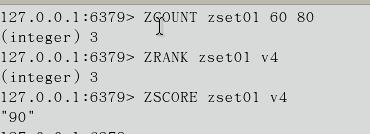
****

****

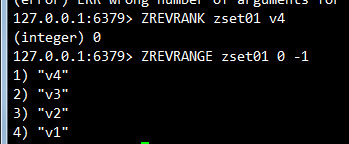
****

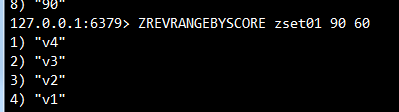
****

****

****

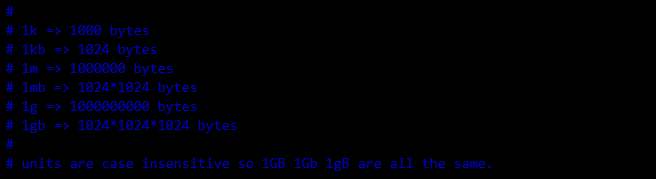
**ZREVRANK key values 值 作用是逆序获得下标值**

****

****

## Redis 配置文件（redis.conf ）

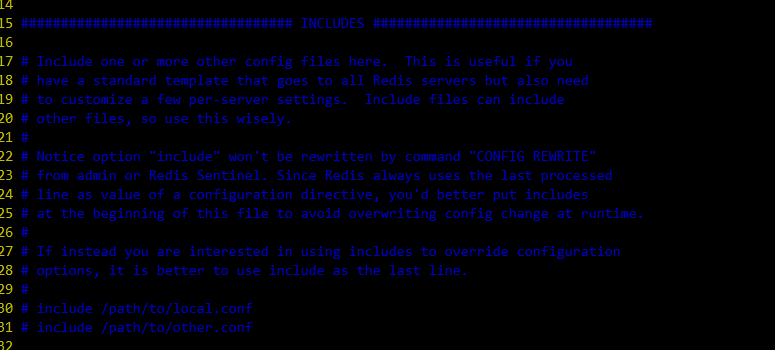
1. 放在那 reids.cnf
2. Units 单位



配置大小单位，开头定义了一些基本的度量单位，只支持bytes,不支持bit

对大小写不敏感

1. Includes 包含



包含一个或多个配置文件，includes包含，redis.conf作为总闸

1. **GENERAL 通用**

**Daemonize**

**Pidfile**

**Port**

**Tcp-backlog**

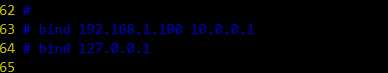
**(设置tcp的backlog，backlog其实是一连接队列，backlog队列总和=未完成三次握手队列+已经完成三次握手队列。在高并发环境下需要一个backlog值来避免慢客户端连接问题。注意Linux内核会将这个值减小到/proc/sys/net/core/sonaxconn的值，所以需要确认增大somaxconn和tcp\_max\_syn\_backlog两个值，来达到效果)**

**Timeout**

**空闲多少秒可以关，（l类似QQ 长时间不用）**

****

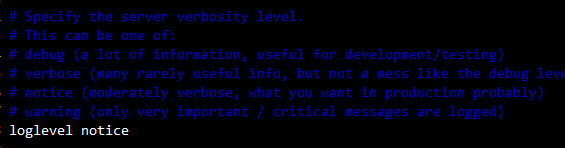
**Bind**

****

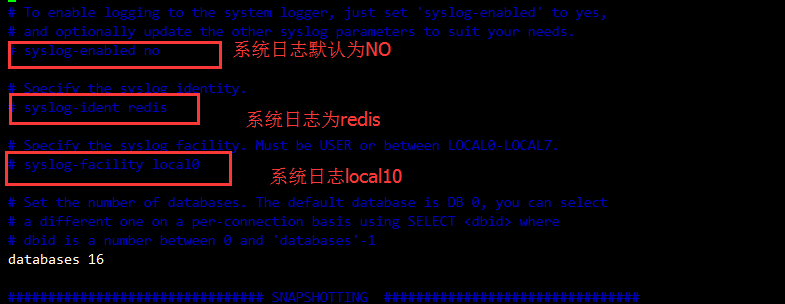
**Tcp-keepalive**

**单位为秒，如果设置为0，则不会进行Keepalive检测，建议设置成60**

**Loglevel**

**日志级别从上倒下级别增高**

**Logfile （日志的名字）**

****

**Syslog-enabled**

**是否不日志输出到syslog中**

**Syslog-ident**

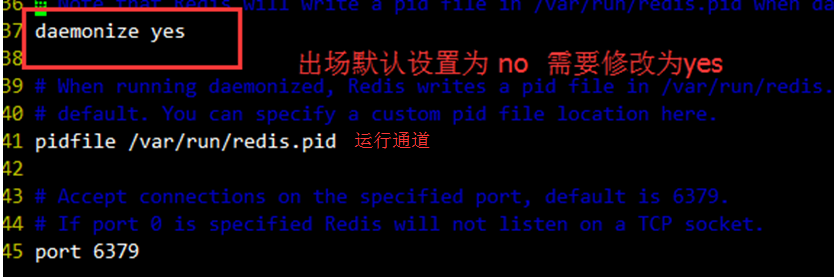
**指定syslog里的日志标志**

**Syslog-facility**

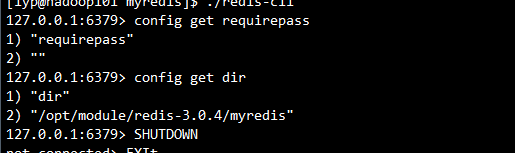
**知道syslog设备，值可以是USER或LOCAL0 – LOCAL7**

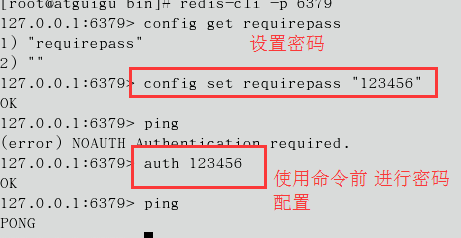
**Databases**

**默认Redis一共有16个库**

****

1. SNAPSHOTTING 快照
2. REPLICATION 复制
3. SECURITY 安全



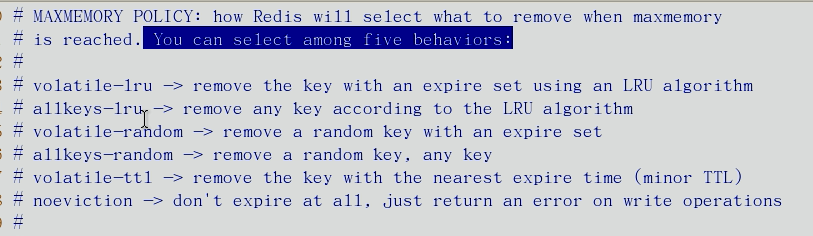


1. LIMITS 限制

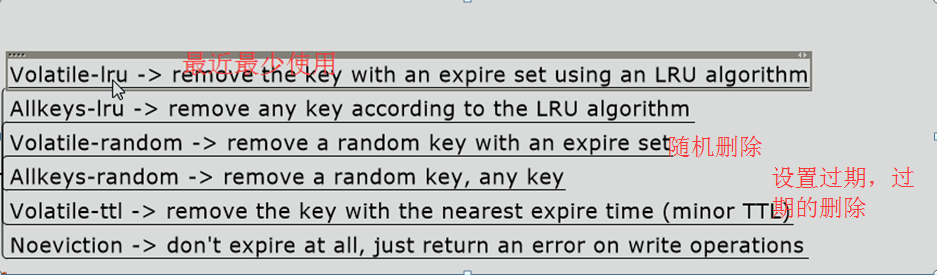


MaxClients

Maxmemory(内存过期策略)



Maxmemory-policy

11)volatitle[保护]-lru:使用LRU算法移除key,只对设置了过期的键

2)allkeys-lru：使用LRU算法移除key

3)volatitle-random:在过期集合中移除随机的key,只对过期了的时间的键

4)allkeys-random:移除随机的key

5)移除那些TTL值最小的key,即哪些最近要过期的key

6) noevition：不进行移除。针对写操作，只是返回错误信息

Maxmemory-samples

设置样本数量，LRU算法和最小TTL算法都并非是精确的算法，而是估值，所以你可以设置样本的大小，redis默认会检查这么多个key并选择其中LRU的那个

1. APPEND ONLY MODE 追加
2. 常见配置redis.conf 介绍

Redis.Conf配置想说明如下：

1.redis默认不是以守护进程的方式运行，可以通过配置项修改，使用yes启用守护进程 daemonize no

2.当Redis以守护进程方式运行时，Redis默认会把pid写入/var/run/redis.pid文件，可以通过pidfile指定pidfile/var/run/redis.pid

3.指定Redis监听端口，默认是6379，作者在自己的一篇文中解释了为什么选6379，女歌手名字

4.绑定主机地址

Bind 127.0.0.1

5.当客户端限制多长时间后关闭连接，如果指定为0，表示关闭该功能 timeout 300

6.指定日志记录级别，Redis共支持四个级别：debug、verbose、notice、warning，默认为verbose

Loglevel verbose

7.日志记录方式，默认标注输出，如果配置Redis为守护进程方式运行，而这例有配置为日志记录方式为标准输出，则日志将会发送给/dev/null

Logfile stdout

8.设置数据库的数量，默认数据库为0，可以使用SELECT <dbid>命令在连接上指定数据库id database 16

9.指定在多长时间内，有多少次更新操作，就将数据库同步到数据文件，可以抖个条件配合save<seconds> <changes>

Redis默认配置文件中提供了三个条件：

Save 900 1

Save 300 10

Save 60 10000

分别表示900秒（15分钟）内有一个更改，

300表（5分钟）内有10 个更改

10.指定存储值本地数据是否压缩数据，默认为yes,Redis采用LZF压缩，如果为了节省CPU时间，可以关闭该选项，但会导致数据库文件变的巨大 rdbcompression yes

11. 指定本地数据库文件名，默认值为dump.rdb

Dbfilename dump.rdb

12 。指定本地数据库存放目录 dir ./

13 设置当本机为slav服务时，设置master服务的IP地址及端口，在Redis启动是，它会自动从master进行数据同步slaveof <masterip> <masterport>

14 .当master 服务设置了密码保护时，slav服务连接master的密码

Masterauth<master-password>

15.设置Redis连接密码，如果配置了连接密码，客户端在连接Redis需要AUTH<passwordd>命令提供密码，默认关闭 requirepass foobared

16 设置同一时间最大客户端连接数，默认无限制，Redis可以同时打开的客户端连接数为Redis进程可以打开的最大文件描述符数，如果设置maxclients 0 ,表示不作限制，当客户端连接数到达限制时，Redis会关闭新的连接并向客户端返回max number of clients reached 错误信息 maxclient 128

17 。指定Redis 最大内存限制，Redis在启动是会把数据加载到内存后，Redis会先尝试清除已到期或即将到期的key,当此方法处理后，仍然到达最大 maxmemory <bytes>

18.指定是否在每次更新作后进行日志记录，Redsis在默认情况下是异步把数据写入磁盘，如果不开启，可能会在断电时导致一段时间内的数据丢失，因为Redis本身同步数据文件是按上面save条件类同步的，所以数据会在一段时间内只存在于内存中，默认为no

19.指定更新日志文件名，默认为appendpnly.aof

Appendfilename appendonly.aof

20 指定更新日志条件，共有3个可选值：

No：表示等操作系统进行数据缓存同步到磁盘(快)

Always：表示每次更新操作后手动调用fsync()将数据写到磁盘（慢，安全）

Everysec:表示每秒同步一次（折衷，默认值）

Appendfsync everysec

21.指定是否启用虚拟内存机制，默认值为no,简单的介绍，vm机制将数据分页存放，由Redis将访问量较少的页即冷数据swap到磁盘上，访问多的页面由磁盘自动换出到内存中 vm-enabled no

22 虚拟内存文件路径，默认值为/tmp/redis.swap 不可多个Redis实例共享

Vm-swap-file /tmp/redis.swap

23. 将所有大于vm-max-memory的数据存入虚拟内存,无论vm-max-memory设置多小,所有索引数据都是内存存储的(Redis的索引数据 就是keys),也就是说,当vm-max-memory设置为0的时候,其实是所有value都存在于磁盘。默认值为0

   vm-max-memory 0

24. Redis swap文件分成了很多的page，一个对象可以保存在多个page上面，但一个page上不能被多个对象共享，vm-page-size是要根据存储的 数据大小来设定的，作者建议如果存储很多小对象，page大小最好设置为32或者64bytes；如果存储很大大对象，则可以使用更大的page，如果不 确定，就使用默认值

   vm-page-size 32

25. 设置swap文件中的page数量，由于页表（一种表示页面空闲或使用的bitmap）是在放在内存中的，，在磁盘上每8个pages将消耗1byte的内存。

   vm-pages 134217728

26. 设置访问swap文件的线程数,最好不要超过机器的核数,如果设置为0,那么所有对swap文件的操作都是串行的，可能会造成比较长时间的延迟。默认值为4

   vm-max-threads 4

27. 设置在向客户端应答时，是否把较小的包合并为一个包发送，默认为开启

  glueoutputbuf yes

28. 指定在超过一定的数量或者最大的元素超过某一临界值时，采用一种特殊的哈希算法

  hash-max-zipmap-entries 64

  hash-max-zipmap-value 512

29. 指定是否激活重置哈希，默认为开启（后面在介绍Redis的哈希算法时具体介绍）

  activerehashing yes

30. 指定包含其它的配置文件，可以在同一主机上多个Redis实例之间使用同一份配置文件，而同时各个实例又拥有自己的特定配置文件

  include /path/to/local.conf

## 持久化 (Persistence)

#### 一、RDB(Redis Data Base)

在指定时间间隔内将内存中的数据集快照写入磁盘，也就行话讲的Snapshot快照，它回复时是将快照文件直接读到内存里

Redis 会单独创建(fork)以一个子进程进行持久化，会现将数据写入到一个临时文件中，待持久化过程都结束了，在用这个临时文件替换上次持久化好的文件。整个过程中，主进程是不进行任何IO操作的，这就确保了极高的性能，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很高，那RDB方法要比AOF更加高效，RDB的缺点是最后一次持久化的数据可能丢失

Fork的作是赋值一个与当前进程一样的进程。新进程的所有数据(变量、环境变量、程序计数器等)数值都和原来进程一致，但是是一个全新的进程，作为原进程的子进程

查看Redis 进程

如何触发RDB 配置文件中默认的快照配置，冷拷贝后重新使用

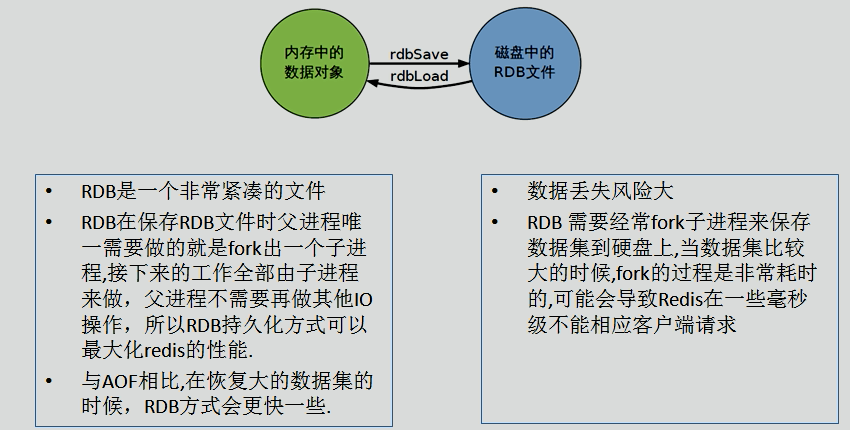
命令save（会阻塞）或者bgsave（异步不会阻塞）

优点：执行flushall命令，也会产生dump.rdb文件，但是里面是空的无意义

适合大规模的数据恢复，对大数据完整性和一致性要求不高

缺点：在一定间隔时间做一次备份，所以如果redis意外down掉的话，就会丢失最后一次快照后的所有修改，Fork的时候，内存中的数据被克隆了一份，大致2倍的膨胀性需要考虑

如何停止：redis-li config set save



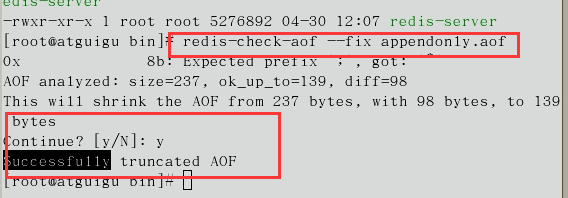
#### 二、Aof (Apend Only File)

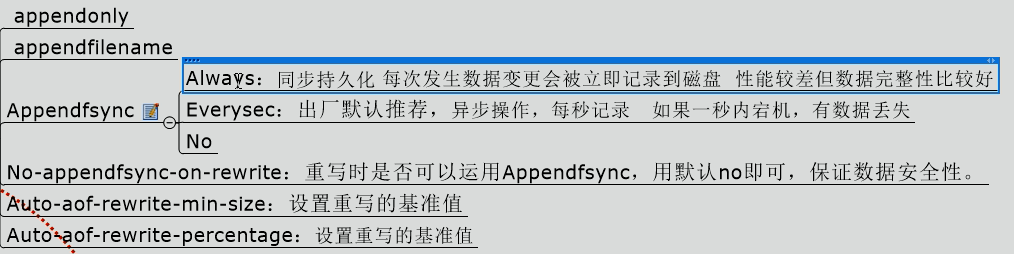
是什么：以日志的形式来记录每个写操作，将Redis执行过的所有写指令记录下来（读操作不记录），只需追加文件不可以改写文件，Redis启动支出会读取该文件重新构建数据，换句话，redis重启的话就根据日志文件的内容将写指令从前到后执行一次以完成数据的恢复工作

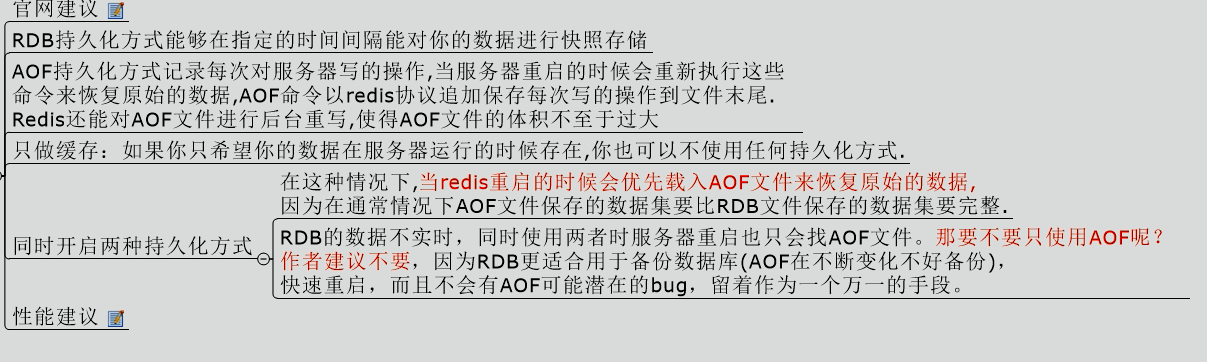
AOF:保存的是appendony.aof文件

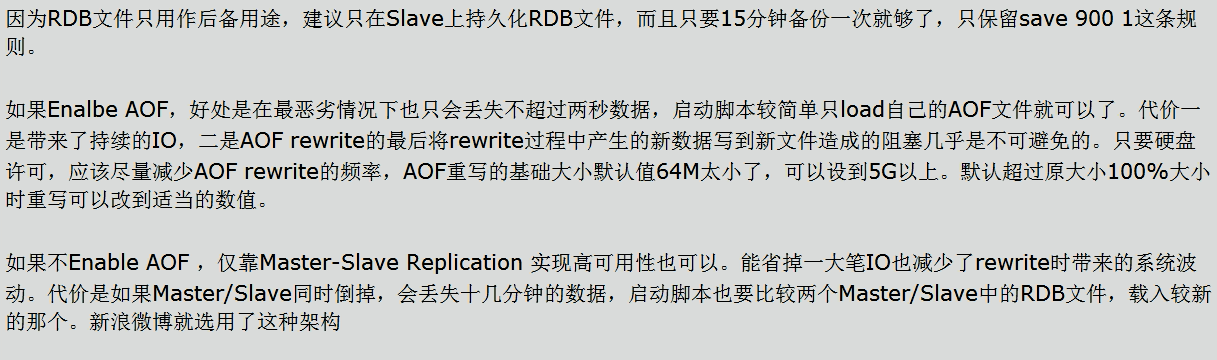
Dum 和aof文件可以共存，首先加载的是aof文件

aof文件出错修复方法









事务：

给房东转账（要么都成功，要么都失败）

## Redis的事务

是什么：可以一次执行多个命令，本质是一组命令的集合，一个事务中所有命令都会序列化，按顺序地串行化执行而不会被其他命令插入，不许加塞。

执行 监控 2.2以上支持CAS

能干什么：一个队列中，一次性顺序行，排他性的执行一系列命令

怎么玩：

MULTI（标记一个事务的开始）、EXEC（执行所有事务块内的命令）、DISCARD（取消事务，放弃执行事务块内的所有命令）、WATCH(监视一个或多个，如果事务执行之前这个[这些]key被其他命令所改动那么事务将被打断)、UNWATCH(取消watch命令对所有key的监视)

Case1:正常执行

Case2:放弃事务

Case3:全体连坐

Case4:冤头债主（对事物支持是部分支持）

Case5:watch监控 [一致性和并发性]

[表锁，涵锁]

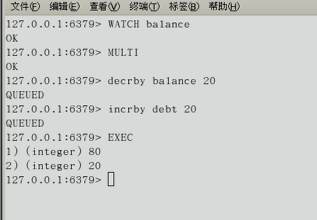
乐观锁，悲观锁，CAS （Check and Set）

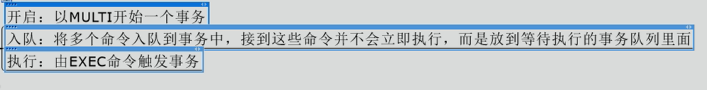
乐观锁：不锁整张表，并发性和一致性 ，在后面夹了version，乐观锁适合用于多读的应用类型，这样可以提高吞吞量，乐观锁策略：提交版本必须大于当前版本才能执行更新

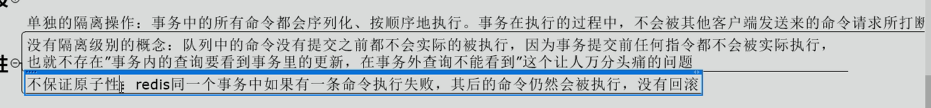
悲观锁：整张表都给锁了 每次去拿数据的时候都认为i别人会修改，所以每次在拿数据的时候都会上锁，这样别人拿这个数据就会Block直到它拿到锁，传统的关系型数据库里面就用到这种锁机制，行锁，表锁，读锁，写锁，都是在操作之前先上锁

CAS:

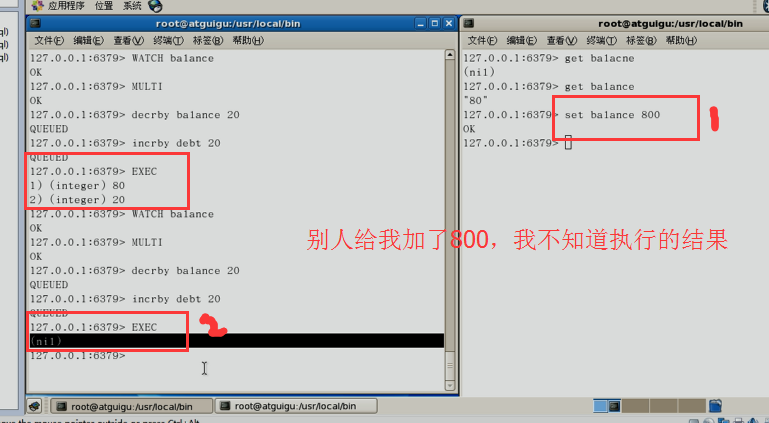
初始化信用卡可用余额和欠额







特性：

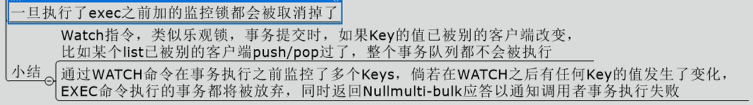


无加塞篡改，先监控再开启mutli，保证两笔金额变动在同一个事务内

有加塞篡改

Unwatch

一旦执行了exec之前加的监控锁都会被取消掉



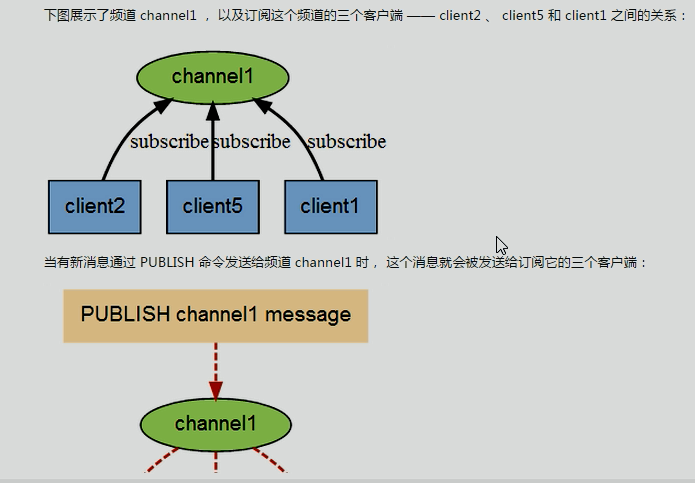
## 消息订阅和发布：

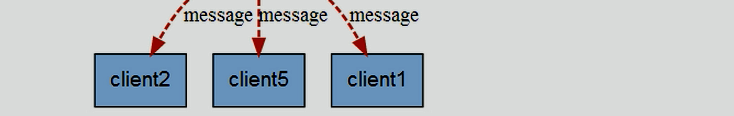
是什么：

进程间的一种消息通信模式：发送者(pub)发送消息，订阅者(sub)接收消息。

怎么做：

订阅/发布消息图





命令：

1、PSUBSCRIBE [pattern ]订阅一个或多个给定模式的频道

2、PUBSUB subcommand [argument ]查案订阅与发布系统状态

3、PUBLISH channel message 将消息发送到指定的频道

4、PUNSUBSCRIBE pattern 退订所有给定模式的频道

5、SUBSCRIBE channel 订阅给定的一个或多个频道的信息

6、UNSUBSCRIBE channel 制定退订给定的频道

案例：

1 可以一次性订阅多个，SUBSCRIBE c1 c2 c3

2 消息发布，PUBLISH c2 hello-redis

3 订阅多个，通配符\*， PSUBSCRIBE new\*

4 收取消息， PUBLSHI new1 redis2015

## Redis 复制（Master/Slave）

是什么：也就是主从复制，主机数据更新后根据配置和策略，自动同步到备机master/slaver机制，只有Master以写为主，Slave以读为主

能干什么：读写分离，容灾恢复

#### 怎么玩：

拷贝多个redis.conf 文件

开启daemonize yes



Pid 文件名字



指定端口



Log 文件名字



Dump.rdb 名字



1.配从不配主

2.从库配置：slaveof 主库ip 主库端口

每次与master 断开后，都需要重新连接除非你配置进redis.conf文件 【情景】

Info replication

3.修改配置文件细节操作

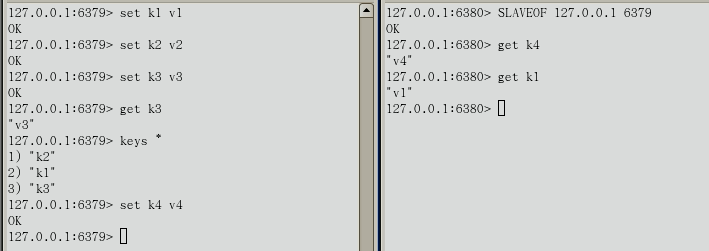
4.常用3招：

**一主二仆**

上一个Slave可以是下一个slave的Master,Slave同样可以接受其他slave的连接和同步请求，那么该slave作为连跳中下一个master可以有效减轻masterd 写压力



从机接手，从头[久]撸到尾[新]



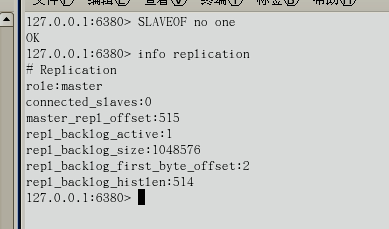
**薪火相传**

[去中心化，一对多压力大]

中途变更转向会清除之前的数据，重新建立拷贝最新的Slaveof 新主库IP新主机端口

**反客为主:**

是当前数据库停止与其他数据库的同步，转成主数据库

****

#### 复制原理：

slave启动成功连接到master后会发送一个sync命令

Master接到命令启动后太的存盘进程，同时手机所有接受到的用于修改数据集命令，

在后台进程执行完毕之后，master将传送整个数据文件到slave,以完成一次完全同步

全量复制：而slave服务在接受到数据库文件数据后，价格其存盘并加载到内存中。(首次)

增量复制：Master继续将新的所有收集到的修改命令依次传递给slave完成同步(再次)

但是只要重新连接master,一次完全同步（全量复制）将被自动执行

#### 哨兵模式：(sentinel)[反客为主 ，监控主]

反客为主的自动版，能够后台监控主机是否故障，如果故障了根据投票数自动将从库转换为主库

**怎么完(使用步骤)**

在myredis上面建立一个 sentinel.conf 文件名字不能错



一组sentine 能同时监控多个Master





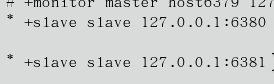


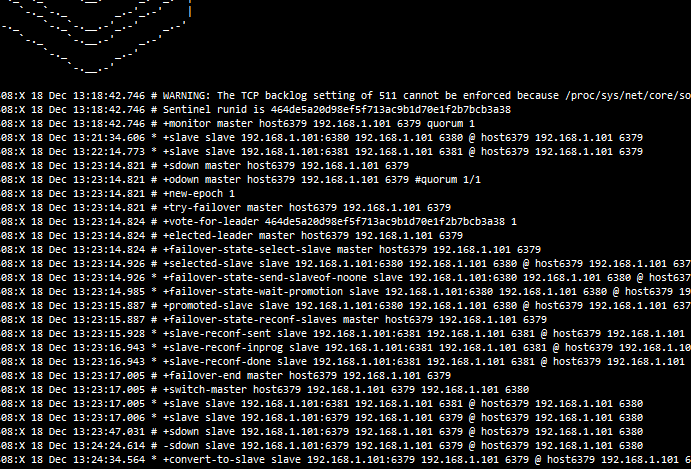
启动哨兵

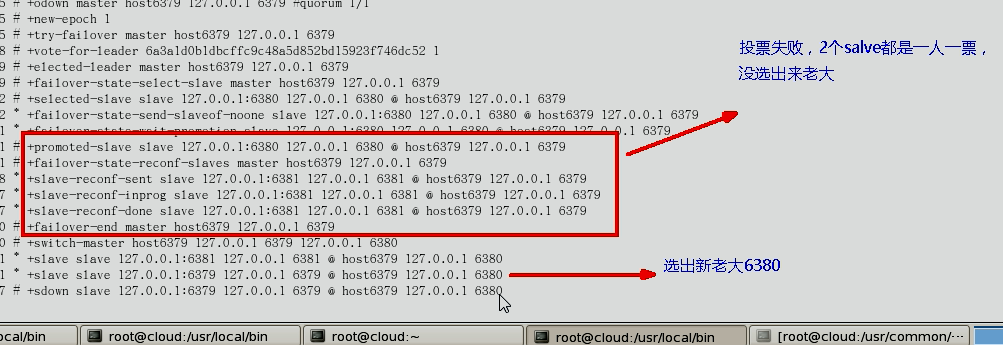
Redis-sentinel /usr/common/sentinel.conf [目录依照按照各自的实际情况配置，可能目录不同]

Pwd: 



巡逻





#### 复制的缺点：