1. redis的API
2. redis介绍

redis是一种基于键值对（key-value）数据库，其中value可以为string、hash、list、set、zset等多种数据结构，可以满足很多应用场景。还提供了键过期，发布订阅，事务，流水线，等附加功能,

流水线: Redis 的流水线功能允许客户端一次将多个命令请求发送给服务器， 并将被执行的多个命令请求的结果在一个命令回复中全部返回给客户端， 使用这个功能可以有效地减少客户端在执行多个命令时需要与服务器进行通信的次数。

1. 特性：

1〉速度快，数据放在内存中，官方给出的读写性能10万/S，与机器性能也有关

a,数据放内存中是速度快的主要原因

b,C语言实现，与操作系统距离近

c,使用了单线程架构，预防多线程可能产生的竞争问题

2〉键值对的数据结构服务器

3〉丰富的功能：见上功能

4〉简单稳定：单线程

5〉持久化：发生断电或机器故障，数据可能会丢失，持久化到硬盘

6〉主从复制：实现多个相同数据的redis副本

8〉高可用和分布式：哨兵机制实现高可用，保证redis节点故障发现和自动转移

9〉客户端语言多：java php python c c++ nodejs等

1. 使用场景：
2. 缓存：合理使用缓存加快数据访问速度，降低后端数据源压力
3. 排行榜：按照热度排名，按照发布时间排行，主要用到列表和有序集合
4. 计数器应用：视频网站播放数，网站浏览数，使用redis计数
5. 社交网络：赞、踩、粉丝、下拉刷新
6. 消息队列：发布和订阅
7. 正确安装与启动
8. linux上安装，windows也能装，但我们以linux环境为主

－－－－－－－－－－－－－－－－－－－－－－－－－－－－－－－

1. 配置、启动、操作、关闭

|  |  |
| --- | --- |
| 可执行文件 | 作用 |
| redis-server | 启动redis |
| redis-cli | redis命令行客户端 |
| redis-benchmark | 基准测试工具 |
| redis-check-aof | AOF持久化文件检测和修复工具 |
| redis-check-dump | RDB持久化文件检测和修复工具 |
| redis-sentinel | 启动哨兵 |

1. redis-server启动：
2. 默认配置：redis-server, 日志输出版本信息，端口6379
3. 运行启动：redis-server --port 6380 不建议
4. 配置文件启动： redis-server /opt/redis/redis.conf，灵活，生产环境使用这种

4,redis-cli 启动

1. >交互式：redis-cli -h {host} -p {prot}连接到redis服务，没有h默认连127.0

redis-cli -h 127.0.0.1 -p 6379 //没有p 默认连6379

1. >命令式：redis-cli -h 127.0.0.1 -p 6379 get hello //取key=hello的value
2. >停止redis服务： redis-cli shutdown

注意: a，关闭时：断开连接，持久化文件生成，相对安全

b，还可以用kill关闭，此方式不会做持久化，还会造成缓冲区非法关闭，可能会造成AOF和丢失数据

c，关闭前生成持久化文件：

使用redis-cli -a 123456 登录进去，再shutdown nosave|save

4，>重大版本：

1，版本号第二位为奇数，为非稳定版本（2.7、2.9、3.1）

2，第二为偶数，为稳定版本（2.6、2.8、3.0）

3，当前奇数版本是下一个稳定版本的开发版本，如2.9是3.0的开发版本

1. 重要的指令使用:

1>全局命令

1，查看所有键：keys \* set school dongnao set hello world

2，键总数 dbsize //2个键，如果存在大量键，线上禁止使用此指令

3，检查键是否存在：exists key //存在返回1，不存在返回0

4，删除键：del key //del hello school, 返回删除键个数，删除不存在键返回0

5，键过期：expire key seconds //set name test expire name 10 //10秒过期

ttl 查看剩余的过期时间

6，键的数据结构类型：type key //type hello //返回string,键不存在返回none

1. 单线程架构

列举例子：三个客户端同时执行命令

客户端1：set name test

客户端2：incr num

客户端3：incr num

执行过程：发送指令－〉执行命令－〉返回结果

执行命令：单线程执行，所有命令进入队列，按顺序执行，使用I/O多路复用解决I/O问题，后面有介绍(通过select/poll/epoll/kqueue这些I/O多路复用函数库，我们解决了一个线程处理多个连接的问题)

单线程快原因：纯内存访问， 非阻塞I/O（使用多路复用），单线程避免线程切换和竞争产生资源消耗

问题：如果某个命令执行，会造成其它命令的阻塞

1. **字符串<String>**

3.1，字符串类型：实际上可以是字符串（包括XML JSON），还有数字（整形 浮点数），二进制（图片 音频 视频），最大不能超过512MB

3.2，设值命令：set age 23 ex 10 //10秒后过期 px 10000 毫秒过期

setnx name test //不存在键name时，返回1设置成功；存在的话失败0

set age 25 xx //存在键age时，返回1成功

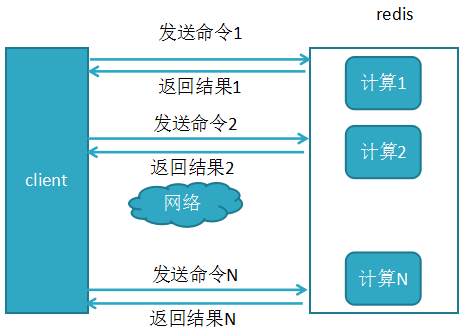
场景：如果有多客户同时执行setnx,只有一个能设置成功，可做分布式锁

获值命令：get age //存在则返回value, 不存在返回nil

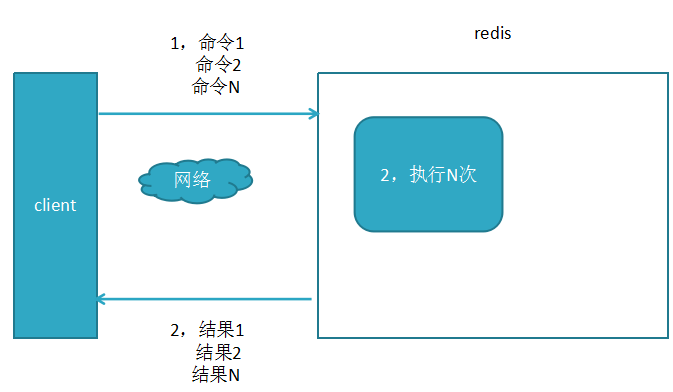
批量设值：mset country china city beijing

批量获取：mget country city address //返回china beigjin, address为nil

若没有mget命令，则要执行n次get命令



使用mget＝1次网络请求+redis内部n次查询



3.3，计数：incr age //必须为整数自加1，非整数返回错误，无age键从0自增返回1

decr age //整数age减1

incrby age 2 //整数age+2

decrby age 2//整数age -2

incrbyfloat score 1.1 //浮点型score+1.1

3.4，append追加指令：set name hello; append name world //追加后成helloworld

3.5，字符串长度：set hello “世界”；strlen hello//结果6，每个中文占3个字节

3.6，截取字符串：set name helloworld ; getrange name 2 4//返回 llo

3.7，内部编码：int:8字节长整理//set age 100; object encoding age //返回int

embstr:小于等于39字节串set name bejin; object encodeing name//embstr

raw:大于39字节的字符串set a fsdfwerwerweffffffffffdfs//返回raw

3.8，应用场景：

1. 键值设计：业务名:对象名:id:[属性]

数据库为order, 用户表user，对应的键可为 order:user:1 或order:user:1:name

注意：redis目前处于受保护模式，不允许非本地客户端链接，可以通过给redis设置密码，然后客户端链接的时候，写上密码就可以了

127.0.0.1:6379> config set requirepass 123456 临时生效

或者修改redis.conf requirepass 123456,启动时./redis-server redis.conf指定conf

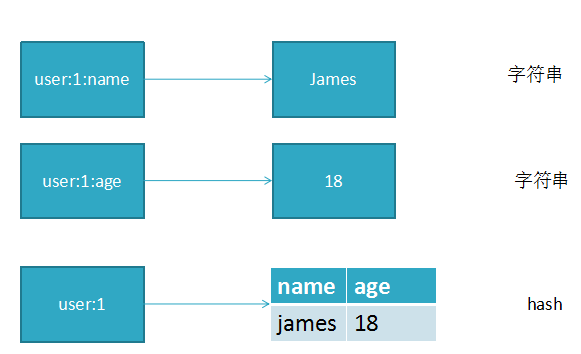
./redis-cli -p 6379 -a 12345678 //需要加入密码才能访问

切换数据库:select 2

**场景实践cache-demo**

JedisAllCommandTest OrderListHashTest SiteVisitNumTest

1. **哈希hash：是一个string类型的field和value的映射表，hash特适合用于存储对象。**



4.1命令 hset key field value

设值：hset user:1 name james //成功返回1，失败返回0

取值：hget user:1 name //返回james

删值：hdel user:1 age //返回删除的个数

计算个数：hset user:1 name james; hset user:1 age 23;

hlen user:1 //返回2，user:1有两个属性值

批量设值：hmset user:2 name james age 23 sex boy //返回OK

批量取值：hmget user:2 name age sex //返回三行：james 23 boy

判断field是否存在：hexists user:2 name //若存在返回1，不存在返回0

获取所有field: hkeys user:2 // 返回name age sex三个field

获取user:2所有value：hvals user:2 // 返回james 23 boy

获取user:2所有field与value：hgetall user:2 //name age sex james 23 boy值

增加1：hincrby user:2 age 1 //age+1

hincrbyfloat user:2 age 2 //浮点型加2

4.2内部编码：ziplist<压缩列表>和hashtable<哈希表>

当field个数少且没有大的value时，内部编码为ziplist

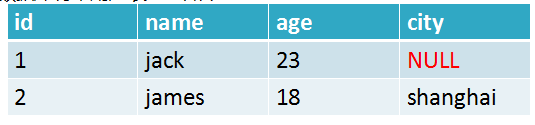
如：hmset user:3 name james age 24; object encoding user:3 //返回ziplist

当value大于64字节，内部编码由ziplist变成hashtable

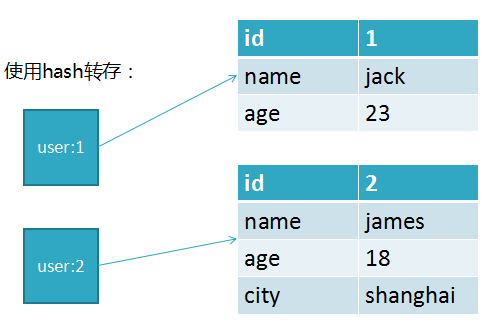
如：hset user:4 address “fsgst64字节”; object encoding user:3 //返回hashtable

4.3应用场景：

比如将关系型数据表转成redis存储：



使用hash后的存储方式为：



如果有值为NULL，那么如下：

HASH类型是稀疏，每个键可以有不同的filed, 若用redis模拟做关系复杂查询开发因难，维护成本高

4.4三种方案实现用户信息存储优缺点：

1，原生：set user:1:name james;

set user:1:age 23;

set user:1:sex boy;

优点：简单直观，每个键对应一个值

缺点：键数过多，占用内存多，用户信息过于分散，不用于生产环境

1. 将对象序列化存入redis

set user:1 serialize(userInfo);

优点：编程简单，若使用序列化合理内存使用率高

缺点：序列化与反序列化有一定开销，更新属性时需要把userInfo全取出来进行反序列化，更新后再序列化到redis

1. 使用hash类型：

hmset user:1 name james age 23 sex boy

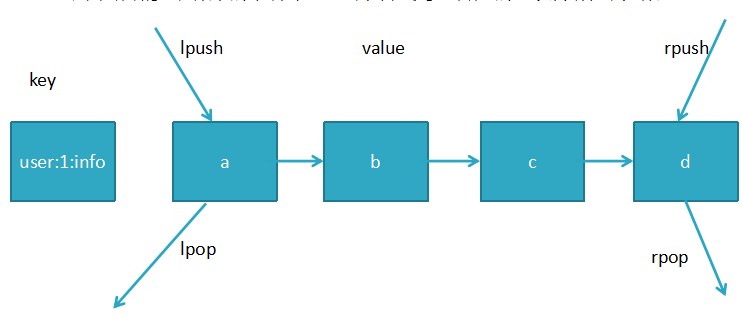
优点：简单直观，使用合理可减少内存空间消耗

缺点：要控制ziplist与hashtable两种编码转换，且hashtable会消耗更多内存

**总结：**对于更新不多的情况下，可以使用序列化，对于VALUE值不大于64字节可以使用hash类型

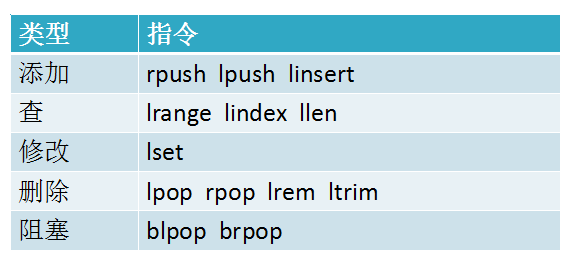
1. **列表<list>**

5.1用来存储多个有序的字符串，一个列表最多可存2的32次方减1个元素



因为有序，可以通过索引下标获取元素或某个范围内元素列表，列表元素可以重复

5.2列表命令：



添加命令：rpush james c b a //从右向左插入cba, 返回值3

lrange james 0 -1 //从左到右获取列表所有元素 返回 c b a

lpush key c b a //从左向右插入cba

linsert james before b teacher //在b之前插入teacher, after为之后，使 用lrange james 0 -1 查看：c teacher b a

查找命令： lrange key start end //索引下标特点：从左到右为0到N-1

lindex james -1 //返回最右末尾a，-2返回b

llen james //返回当前列表长度

lpop james //把最左边的第一个元素c删除

rpop james //把最右边的元素a删除

lrem key count value//删除指定元素

如：lpush test b b b b b j x z //键test放入z x j b b b b b

lrange test 0 -1 //查询结果为 z x j b b b b b

lrem test 4 b //从左右开始删除b的元素,删除4个，

若lrem test 8 b, 删除8个b, 但只有5个全部删除

lrange test 0 -1 //删除后的结果为 b j x z

lrem test 0 b //检索所有b全部删除 j x z

lpush user b b b b b j x z //键user从左到右放入 z x j b b b b b

ltrim user 1 3 //只保留从第2到第4的元素，其它全删

lrange user 0 -1 //查询结果为 x j b, 其它已全被删掉

lpush user01 z y x //键user01从左到右放入x y z

lset user01 2 java // 把第3个元素z替换成java

lrange user01 0 -1 //查询结果为 x y java

**应用场景设计： cacheListHashApplicationTest用例**

每个用户有多个订单key为 order:1 order:2 order:3, 结合hmset

1, hmset order:1 orderId 1 money 36.6 time 2018-01-01

hmset order:2 orderId 2 money 38.6 time 2018-01-01

hmset order:3 orderId 3 money 39.6 time 2018-01-01

把订单信息的key放到队列

lpush user:1:order order:1 order:2 order:3

每新产生一个订单，

hmset order:4 orderId 4 money 40.6 time 2018-01-01

追加一个order:4放入队列第一个位置

lpush user:1:order order:4

当需要查询用户订单记录时：

List orderKeys = lrange user:1 0 -1 //查询user:1 的所有订单key值

for(Order order: orderKeys){

hmget order:1

}

5.3列表内部编码：

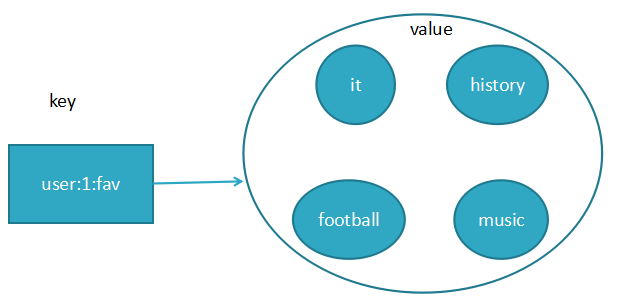
老师之前做的实验是在家里笔记本装的redis,版本是3.1，一直也没更换

经公司讲课的服务器上版本是4.0，从redis的官网查阅了相关资料，在3.2版本以后，redis提供了quicklist内部编码，它结合了ziplist和linkedlist两者的优势，之前的ziplist是存在BUG的，使用quicklist内部编码效率更高，所以我们现在3.2以后看不到这两个编码，只看到quicklist, 英语好的同学可以看一下https://matt.sh/redis-quicklist国外的这篇博客有重点提到

感兴趣的同学可以下一下3.1之前的版本

1. **集合<SET> 用户标签，社交，查询有共同兴趣爱好的人,智能推荐**

保存多元素，与列表不一样的是不允许有重复元素，且集合是无序，一个集合最多可存2的32次方减1个元素，除了支持增删改查，还支持集合交集、并集、差集；



6.1命令：

exists user //检查user键值是否存在

sadd user a b c//向user插入3个元素，返回3

sadd user a b //若再加入相同的元素，则重复无效，返回0

smember user //获取user的所有元素,返回结果无序

srem user a //返回1，删除a元素

scard user //返回2，计算元素个数

sismember user a //判断元素是否在集合存在，存在返回1，不存在0

srandmember user 2 //随机返回2个元素，2为元素个数

spop user 2 //随机返回2个元素a b,并将a b从集合中删除

smember user //此时已没有a b, 只有c

集合的交集：

sadd user:1 zhangsan 24 girl

sadd user:2 james 24 boy//初始化两个集合

sinter user:1 user:2 //求两集合交集， 此时返回24

sadd user:3 wang 24 girl //新增第三个元素

sinter user:1 user:2 user:3 //求三个集合的交集，此时返回24

集合的并集（集合合并去重）：

sunion user:1 user:2 user:3 //三集合合并(并集)，去重24

sdiff user:1 user:2//1和2差集,(zhangsan 24 girl)-(james 24 boy)=zhangsan girl

将交集(jj)、并集(bj)、差集(cj)的结果保存：

sinterstore user\_jj user:1 user:2 //将user:1 user:2的交集保存到user\_jj

sunionstore user\_bj user:1 user:2 //将user:1 user:2的(并)合集保存user\_bj

sdiffstore user\_cj user:1 user:2 //将user:1-user:2的差集保存user\_cj

smemebers user\_cj // 返回zhangsan girl

6.1内部编码：

sadd user 1 2 3 4 //当元素个数少(小于512个)且都为整数，redis使用intset减少内存的使用

sadd user 1 2...513 //当超过512个或不为整数（比如a b）时，编码为hashtable

object encoding user //hashtables

6.2使用场景：

标签，社交，查询有共同兴趣爱好的人,智能推荐

使用方式：

给用户添加标签：

sadd user:1:fav basball fball pq

sadd user:2:fav basball fball

............

或给标签添加用户

sadd basball:users user:1 user:3

sadd fball:users user:1 user:2 user:3

........

计算出共同感兴趣的人：

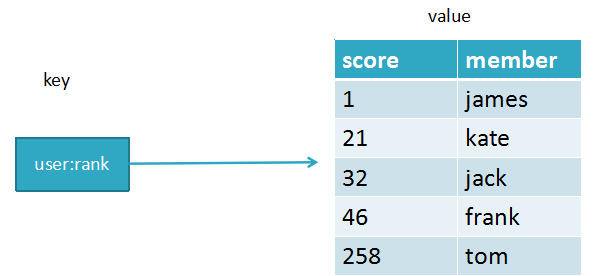
sinter user:1:fav user2:fav

规则：sadd (常用于标签) spop/srandmember(随机，比如抽奖)

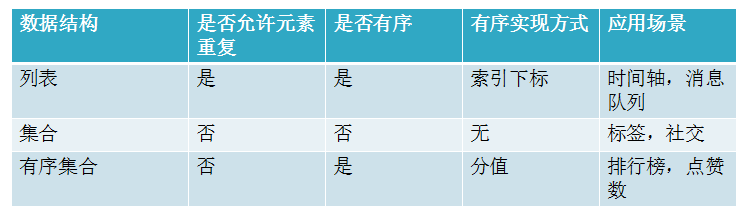
sadd+sinter (用于社交，查询共同爱好的人，匹配)

1. **有序集合: 常用于排行榜，如视频网站需要对用户上传视频做排行榜，或点赞数**

与集合有联系，不能有重复的成员



与LIST SET 对比



7.1命令

zadd key score member [score member......]

zadd user:zan 200 james //james的点赞数1, 返回操作成功的条数1

zadd user:zan 200 james 120 mike 100 lee// 返回3

zadd test:1 nx 100 james //键test:1必须不存在，主用于添加

zadd test:1 xx incr 200 james //键test:1必须存在，主用于修改,此时为300

zadd test:1 xx ch incr -299 james //返回操作结果1，300-299=1

zrange test:1 0 -1 withscores //查看点赞（分数）与成员名

zcard test:1 //计算成员个数， 返回1

查点赞数

zadd test:2 nx 100 james //新增一个集合

zscore test:2 james //查看james的点赞数（分数），返回100

排名：

zadd user:3 200 james 120 mike 100 lee//先插入数据

zrange user:3 0 -1 withscores //查看分数与成员

|  |  |  |
| --- | --- | --- |
| lee | mike | james |
| 100 | 120 | 200 |

zrank user:3 james //返回名次：第3名返回2，从0开始到2，共3名

zrevrank user:3 james //返回0， 反排序，点赞数越高，排名越前

删除成员：

zrem user:3 jame mike //返回成功删除2个成员，还剩lee

增加分数：

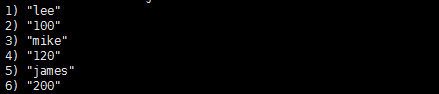
zincrby user:3 10 lee //成员lee的分数加10

zadd user:3 xx incr 10 lee //和上面效果一样

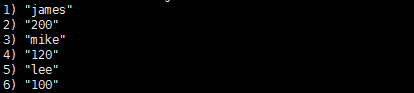
返回指定排名范围的分数与成员

zadd user:4 200 james 120 mike 100 lee//先插入数据

zrange user:4 0 -1 withscores //返回结果如下图



zrevrange user:4 0 -1 withscores //倒序，结果如下图



返回指定分数范围的成员

zrangebyscore user:4 110 300 withscores //返回120 lee ,200 James, 由低到高

zrevrangebyscore user:4 300 110 withscores //返回200james 120lee,由高到低

zrangebyscore user:4 (110 +inf withscores//110到无限大，120mike 200james

zrevrangebyscore user:4 (110 -inf withscores//无限小到110，返回100 lee

返回指定分数范围的成员个数：

zcount user:4 110 300 //返回2，由mike120和james200两条数据

删除指定排名内的升序元素：

zremrangebyrank user:4 0 1 //分数升序排列，删除第0个与第1个，只剩james

删除指定分数范围的成员

zadd user:5 200 james 120 mike 100 lee//先插入测试数据

zremrangebyscore user:5 100 300 //删除分数在100与300范围的成员

zremrangebyscore user:5 (100 +inf //删除分数大于100(不包括100),还剩lee

有序集合交集：

格式：**zinterstore** destination numkeys key ... [WEIGHTS weight] [AGGREGATE SUM|MIN|MAX]

destination:交集产生新的元素存储键名称

numkeys: 要做交集计算的键个数

key :元素键值

weights:每个被选中的键对应值乘weight, 默认为1

初始化数据：

zadd user:7 1 james 2 mike 4 jack 5 kate //初始化user:7数据

zadd user:8 3 james 4 mike 4 lucy 2 lee 6 jim //初始化user:8数据

交集例子：

zinterstore user\_jj 2 user:7 user:8 aggregate sum //2代表键合并个数，

//aggregate sum可加也不可加上，因为默认是sum

//结果user\_jj：4james(1+3), 6mike(2+4)

zinterstore user\_jjmax 2 user:7 user:8 aggregate max 或min

//取交集最大的分数，返回结果 3james 4mike, min取最小

weights:

zinterstore user\_jjweight 2 user:7 user:8 weights 8 4 aggregate max

//1,取两个成员相同的交集，user:7->1 james 2 mike; user:8->3 james 4 mike

//2,将user:7->james 1\*8=8, user:7->mike 2\*8 =16,

最后user:7结果 8 james 16 mike;

//3,将user:8-> james 3\*4=12, user:8->mike 4\*4=16

最后user:8结果12 james 16 mike

//4,最终相乘后的结果，取最大值为 12 james 16mike

//5, zrange user\_jjweight 0 -1 withscores 查询结果为 12 james 16mike

总结：将user:7成员值乘8，将user:8成员值乘4，取交集，取最大

有序集合并集（合并去重）：

格式：**zunionstore** destination numkeys key ... [WEIGHTS weight] [AGGREGATE SUM|MIN|MAX]

destination:交集产生新的元素存储键名称

numkeys: 要做交集计算的键个数

key :元素键值

weights:每个被选中的键对应值乘weight, 默认为1

zunionstore user\_jjweight2 2 user:7 user:8 weights 8 4 aggregate max

//与以上zinterstore一样，只是取并集，指令一样

7.1有序集合内部编码

1. ziplist: zadd user:9 20 james 30 mike 40 lee

object encoding user:init

//当元素个数少（小于128个），元素值小于64字节时，使用ziplist编码，可有效减少内存的使用

2，skiplist: zadd user:10 20 james......

//大于128个元素或元素值大于64字节时为skiplist编码

7.2使用场景：

排行榜系统，如视频网站需要对用户上传的视频做排行榜

点赞数：zadd user:1:20180106 3 mike //mike获得3个赞

再获一赞：zincrby user:1:20180106 1 mike //在3的基础上加1

用户作弊，将用户从排行榜删掉：zrem user:1:20180106 mike

展示赞数最多的5个用户：

zadd user:4:20160101 9 jack 10 jj 11 dd 3 james 4 lee 6 mark 7 kate

zrevrangebylex user:4:20160101 + - limit 0 5

查看用户赞数与排名：

zscore user:1:20180106 mike zrank user:1:20180106 mike

1. **redis键管理**

8.1 键重命名 rename oldKey newkey //格式

rename oldKey newKey //若oldKey之前存在则被覆盖

set name james ；set name1 mike //数据初始化

renamenx name name1 //重命名失败，只有当name1不存在才能改名

8.2 返回随机键 dbsize //redis有16个库，查看当前库的键值对总数

randomkey //返回随机键

8.3 键过期：expire name:03 20 //键name:03 在10秒后过期

ttl name:03 //查看过期按秒到计时，当返回-2说明已删除

pttl name:03 //查看过期按毫秒到时计

set name:05 james //初始化数据

pexpire name:05 20000 //20000毫秒（20S）后过期

expire name:06 -2 //直接过期，和del一样

expireat name:04 1516971599 //设置在2018/01/26 20:59:59过期

时间转时间戳：网址http://tool.chinaz.com/Tools/unixtime.aspx

hset user:01 name james //初始化数据

expire user:01 60 //设置60S右过期

ttl user:01 //查看过期剩余时间

persist user:01 //去掉过期

ttl user:1 //返回-1 可以永久查询不失效

注意：对于字符串重设值后，expire无效，

set name james

expire name 50

ttl name

set name james1 //此时expire取消

ttl name //返回-1， 长期有效

8.4 键的迁移：把部分数据迁移到另一台redis服务器

1, move key db //reids有16个库， 编号为0－15

set name james1; move name 5 //迁移到第6个库

select 5 ;//数据库切换到第6个库， get name 可以取到james1

这种模式不建议在生产环境使用，在同一个reids里可以玩

2, dump key;

restore key ttl value//实现不同redis实例的键迁移，ttl=0代表没有过期时间

例子：在A服务器上 192.168.1.111

set name james;

dump name; // 得到"\x00\x05james\b\x001\x82;f\"DhJ"

在B服务器上：192.168.1.118

restore name 0 "\x00\x05james\b\x001\x82;f\"DhJ"

get name //返回james

3,migrate指令迁移到其它实例redis，在1.111服务器上将test移到118

|  |  |  |  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- | --- | --- | --- |
| migrate | 192.168.1.118 | 6379 | test | 0 | 1000 | copy | replace | keys |
| 指令 | 要迁移的目标IP | 端口 | 迁移键值 | 目标库 | 超时时间 | 迁移后不删除原键 | 不管目标库是不存在test键都迁移成功 | 迁移多个键 |

8.5键的遍历

redis提供了两个命令来遍历所有的键

1，键全量遍历：

mset country china city bj name james //设置3个字符串键值对

keys \* //返回所有的键, \*匹配任意字符多个字符

keys \*y //以结尾的键，

keys n\*e //以n开头以e结尾，返回name

keys n?me // ?问号代表只匹配一个字符 返回name,全局匹配

keys n?m\* //返回name

keys [j,l]\* //返回以j l开头的所有键 keys [j]ames 全量匹配james

考虑到是单线程， 在生产环境不建议使用，如果键多可能会阻塞

如果键少，可以

2，渐进式遍历

mset a a b b c c d d e e f f g g h h i i j j k k l l m m n n o o p p q q r r s s t t u u v v w w x x y y z z //初始化26个字母键值对

字符串类型：

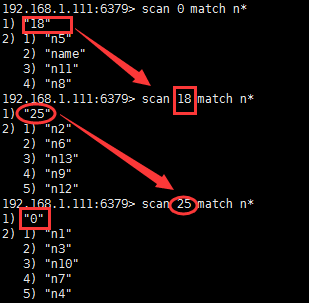
SCAN命令用于迭代当前数据库中的数据库键。

返回结果为：用于下一次迭代的新游标4096；所有被迭代的元素

如果数据集合不是以哈希表作为底层实现的话，则**scan类命令无视count**选项，直接返回数据集合中的所有元素

mset n1 1 n2 2 n3 3 n4 4 n5 5 n6 6 n7 7 n8 8 n9 9 n10 10 n11 11 n12 12 n13 13

scan 0 match n\* //匹配以n开头的键，最大是取10条，第一次scan 0开始



第二次从游标4096开始取20个以n开头的键，相当于一页一页的取

当最后返回0时，键被取完

比如将old:user开头的元素全删掉

**注：可有效地解决keys命令可能产生的阻塞问题**

* 除scan字符串外：还有以下
* [SCAN](http://doc.redisfans.com/key/scan.html" \l "scan" \t "http://blog.csdn.net/qq_27623337/article/details/_blank) 命令用于迭代当前数据库中的数据库键。
* [SSCAN](http://doc.redisfans.com/set/sscan.html" \l "sscan" \t "http://blog.csdn.net/qq_27623337/article/details/_blank) 命令用于迭代集合键中的元素。
* [HSCAN](http://doc.redisfans.com/hash/hscan.html" \l "hscan" \t "http://blog.csdn.net/qq_27623337/article/details/_blank) 命令用于迭代哈希键中的键值对。
* [ZSCAN](http://doc.redisfans.com/sorted_set/zscan.html" \l "zscan" \t "http://blog.csdn.net/qq_27623337/article/details/_blank) 命令用于迭代有序集合中的元素（包括元素成员和元素分值）。

用法和scan一样

1. **redis数据库管理**

select 0 //共16个库， 0 －－15， select切换数据库

set name james

select 1

get name //隔离了，取不到，和mysql不同库一样

其中redis3.0以后的版本慢慢弱化了这个功能，如在redis cluster中只允许0数据库

原因：1，redis单线程，如果用多个库，这些库使用同一个CPU,彼此会有影响

2，多数据库，调试与运维麻烦，若有一个慢查询，会影响其它库查询速度

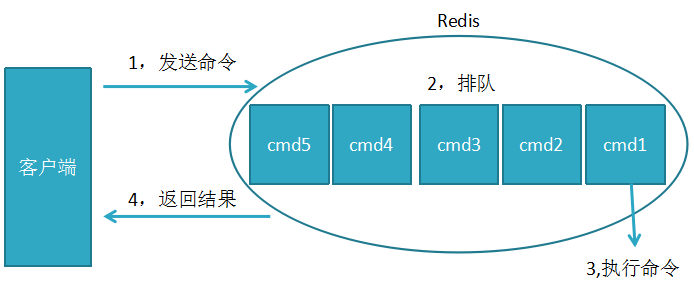
3，来回切换，容易混乱

flushdb: 只清空当前数据库的键值对 dbsiz 0

flushall: 清空所有库的键值对 （这两个指令慎用！！！！）

1. redis功能细解

2.1 慢查询原因分析:与mysql一样:当执行时间超过阀值，会将发生时间 耗时 命令记录



redis命令生命周期：发送 排队 执行 返回

慢查询只统计第3个执行步骤的时间

预设阀值：两种方式，默认为10毫秒

1. 动态设置6379:> config set slowlog-log-slower-than 10000 //10毫秒10000微秒

使用config set完后,若想将配置持久化保存到redis.conf，要执行config rewrite

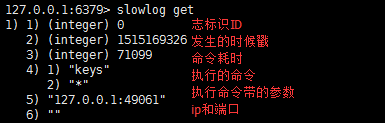
2，redis.conf修改：找到slowlog-log-slower-than 10000 ，修改保存即可

注意：slowlog-log-slower-than =0记录所有命令 -1命令都不记录

原理：慢查询记录也是存在队列里的，slow-max-len 存放的记录最大条数，比如设置的slow-max-len＝10，当有第11条慢查询命令插入时，队列的第一条命令就会出列，第11条入列到慢查询队列中， 可以config set动态设置，也可以修改redis.conf完成配置。

2.2命令：

获取队列里慢查询的命令：slowlog get 查询返回的结构如下



获取慢查询列表当前的长度：slowlog len //以上只有1条慢查询，返回1

对慢查询列表清理（重置）：slowlog reset //再查slowlog len 此时返回0 清空

对于线上slow-max-len配置的建议：线上可加大slow-max-len的值，记录慢查询存长命令时redis会做截断，不会占用大量内存，线上可设置1000以上

对于线上slowlog-log-slower-than配置的建议：默认为10毫秒，根据redis并发量来调整，对于高并发比建议为1毫秒

注意：1，慢查询只记录命令在redis的执行时间，不包括排队、网络传输时间

1. 慢查询是先进先出的队列，访问日志记录出列丢失，需定期执行slow get,将结果存储到其它设备中（如mysql）

2.3 redis-cli详解

./redis-cli -r 3 -h 192.168.1.111 -a 12345678 ping //返回pong表示127.0.0.1:6379能通，**r代表次数**

./redis-cli -r 100 -i 1 info |grep used\_memory\_human //每秒输出内存使用量,输100次，**i代表执行的时间间隔**

./redis-cli -p 6379 -h 192.168.1.111 -a 12345678

对于我们来说，这些常用指令以上可满足，但如果要了解更多

执行redis-cli --help, 可百度

2.4 redis-server详解

./redis-server ./redis.conf & //指定配置文件启动

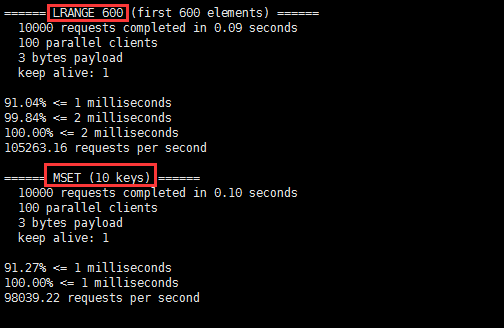
./redis-server --test-memory 1024 //检测操作系统能否提供1G内存给redis, 常用于测试，想快速占满机器内存做极端条件的测试，可使用这个指令

redis上线前，做一次测试

2.5 redis-benchmark：基准性测试，测试redis的性能

redis-benchmark -h 192.168.1.111 -c 100 -n 10000 //100个客户端同时请求redis,共执行10000次

会对各类数据结构的命令进行测试



测试命令事例：

1、redis-benchmark -h 192.168.1.111 -p 6379 -c 100 -n 100000

100个并发连接，100000个请求，检测host为localhost 端口为6379的redis服务器性能

2、redis-benchmark -h 192.168.1.111 -p 6379 -q -d 100

测试存取大小为100字节的数据包的性能

3、redis-benchmark -h 192.168.1.111 -t set,lpush -n 100000 -q

只测试 set,lpush操作的性能，-q只显示每秒钟能处理多少请求数结果

4、redis-benchmark -h 192.168.1.111 -n 100000 -q script load "redis.call('set','foo','bar')"

只测试某些数值存取的性能, 比如说我在慢查询中发现，大部分为set语句比较慢，我们自己可以测一下Set是不是真的慢，

2.6 Pipeline:

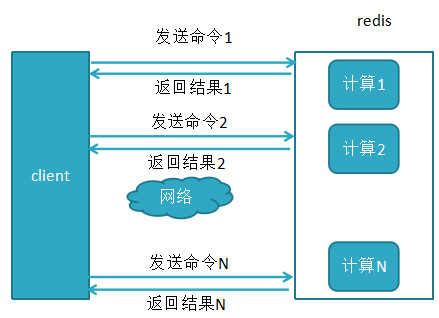
pipeline出现的背景：

redis客户端执行一条命令分4个过程：

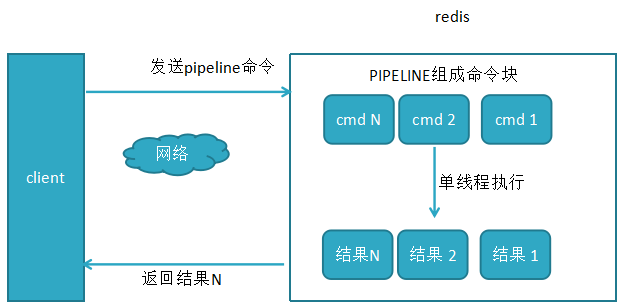
发送命令－〉命令排队－〉命令执行－〉返回结果

这个过程称为Round trip time(简称RTT, 往返时间)，mget mset有效节约了RTT，但大部分命令（如hgetall，并没有mhgetall）不支持批量操作，需要消耗N次RTT ，这个时候需要pipeline来解决这个问题

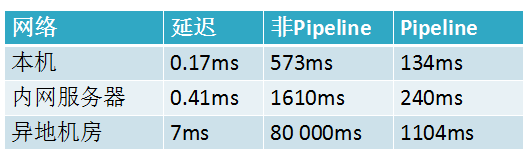
1，未使用pipeline执行N条命令



2，使用pipeline执行N条命令，后面会讲到JAVA的jedis如何使用pipeline功能



3，使用pipeline和未使用pipeline的性能对比：



小总结：使用pipeline执行速度比逐条执行要快，客户端与服务端的网络延迟越大，性能体现越明显

4，原生的批命令（mset, mget等）与pipeline的对比：

one,原生批命令是原子性，pipeline是非原子性， (原子性概念:一个事务是一个不可分割的最小工作单位,要么都成功要么都失败。原子操作是指你的一个业务逻辑必须是不可拆分的. 处理一件事情要么都成功要么都失败，其实也引用了生物里概念，分子－〉原子，原子不可拆分)

two,原生批命令一命令多个key, 但pipeline支持多命令（存在事务），非原子性

three: 原生批命令是服务端实现，而pipeline需要服务端与客户端共同完成

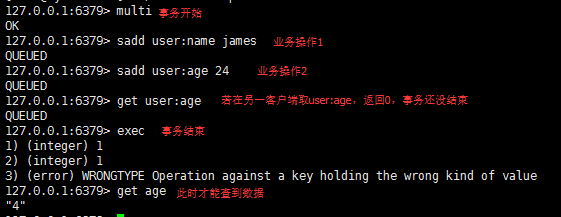
5，pipeline正确使用方式：

使用pipeline组装的命令个数不能太多，不然数据量过大，增加客户端的等待时间，还可能造成网络阻塞，可以将大量命令的拆分多个小的pipeline命令完成

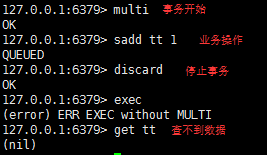
2.6事务

刚大家知道，pipeline是多条命令的组合，为了保证它的原子性，redis提供了简单的事务，什么是事务？事务是指一组动作的执行，这一组动作要么成功，要么失败。

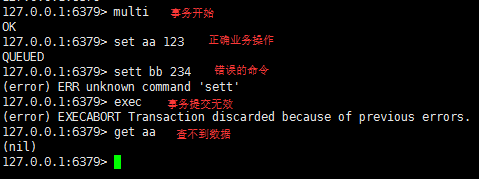
1. redis的简单事务，将一组需要一起执行的命令放到multi和exec两个命令之间，其中multi代表事务开始，exec代表事务结束



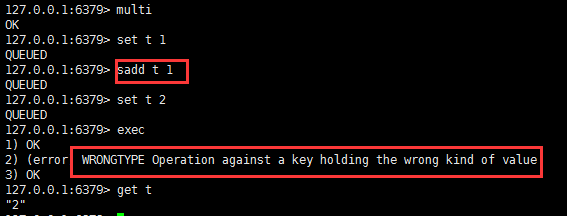
1. 停止事务discard



1. 命令错误，语法不正确，导致事务不能正常结束

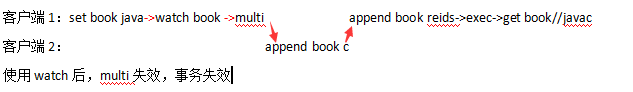


4，运行错误，语法正确，但类型错误，事务可以正常结束



可以看到redis不支持回滚功能

1. watch命令，



redis提供了简单事务，之所以说简单，不支持事务回滚

**2.7LUA语言与redis**

**LUA脚本语言是C开发的，类似存储过程**

使用脚本的好处如下:

•1.减少网络开销：本来5次网络请求的操作，可以用一个请求完成，原先5次请求的逻辑放在redis服务器上完成。使用脚本，减少了网络往返时延。

•2.原子操作：Redis会将整个脚本作为一个整体执行，中间不会被其他命令插入。

•3.复用：客户端发送的脚本会永久存储在Redis中，意味着其他客户端可以复用这一脚本而不需要使用代码完成同样的逻辑。

6379>eval "return redis.call('get',KEYS[1])" 1 name //eval+脚本+KEYS[1]+键个数+键

eval script numkeys key [key ...]

语法1：

local int sum = 0

local int i =0

while i <= 100

do sum = sum+i

i = i+1

end

print(sum)

语法2：

local tables myArray={“james”,”java”,false,34} //定义

local int sum = 0

print(myArray[3]) //返回false

for i = 1,100

do

sum = sum+1

end

print(sum)

for j = 1,#myArray //遍历数组

do

print(myArray[j])

if myArray[j] == “james”

then

print(“true”)

break

else

print(“false”)

end

end

案例-实现访问频率限制: 实现访问者 $ip 127.0.0.1在一定的时间 $time 20S内只能访问 $limit 10次.使用JAVA语言实现：

private boolean accessLimit(String ip, int limit,

int time, Jedis jedis) {

boolean result = true;

String key = "rate.limit:" + ip;

if (jedis.exists(key)) {

long afterValue = jedis.incr(key);

if (afterValue > limit) {

result = false;

}

} else {

Transaction transaction = jedis.multi();

transaction.incr(key);

transaction.expire(key, time);

transaction.exec();

}

return result;

}

以上代码有两点缺陷

可能会出现竞态条件: 解决方法是用 WATCH 监控 rate.limit:$IP 的变动, 但较为麻烦;

以上代码在不使用 pipeline 的情况下最多需要向Redis请求5条指令, 传输过多.

使用lua脚本来处理，包括了原子性：如下

../redis-cli -p 6379 -a 12345678 --eval ipCount.lua 192.168.1.111, 10 20

ttl rate.limit:127.0.0.1

其中 keys[1] = rate.limit:127.0.0.1 argv[1]=10次, argv[2]=20S失效

ipCount.lua内容：

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

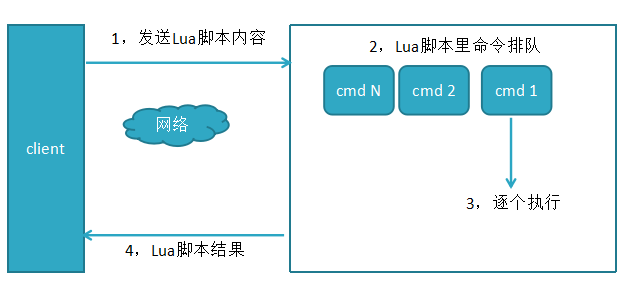
redis.call("SET", key, 1)

redis.call("EXPIRE", key, expire\_time)

return 1

end

执行逻辑：使用redis-cli --eavl时，客户端把lua脚本字符串发给redis服务端，将结果返回客户端，如下图



redis对lua脚本的管理：

1,redis-cli -h 192.168.1.111 -a 12345678 script load "$(cat random.lua)" //将LUA脚本内容加载到redis， 得到 返回的sha1值：afe90689cdeec602e374ebad421e3911022f47c0



redis-cli -h localhost -p 6379 EVALSHA afe90689cdeec602e374ebad421e3911022f47c0 0

2, 6379〉script exists afe90689cdeec602e374ebad421e3911022f47c0 //检查sha1值的LUA脚本是否加载到redis中， 返回1 已加载成功



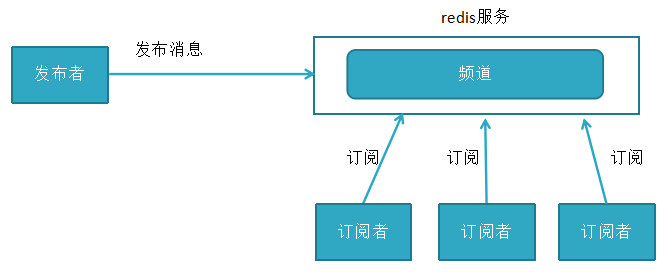
3，6379〉script flush //清空加载的lua脚本内容

4，6379〉script kill //杀掉正在执行的LUA脚本，比如LUA比较耗时阻塞，杀掉



2.8发布与订阅

redis提供了“发布、订阅”模式的消息机制，其中消息订阅者与发布者不直接通信，发布者向指定的频道（channel）发布消息，订阅该频道的每个客户端都可以接收到消息



redis主要提供发布消息、订阅频道、取消订阅以及按照模式订阅和取消订阅

1. 发布消息

publish channel:test "hello world"

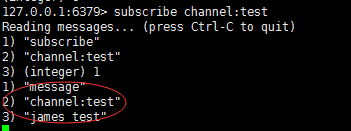


1. 订阅消息

subscrible channel:test

此时另一个客户端发布一个消息:publish channel:test "james test"

当前订阅者客户端会收到如下消息：



和很多专业的消息队列（kafka rabbitmq）,redis的发布订阅显得很lower, 比如无法实现消息规程和回溯， 但就是简单，如果能满足应用场景，用这个也可以

3，查看订阅数：

pubsub numsub channel:test // 频道channel:test的订阅数

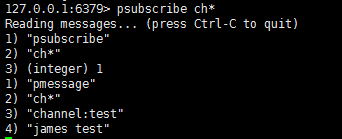
4，取消订阅

unsubscribe channel:test

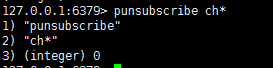
客户端可以通过unsubscribe命令取消对指定频道的订阅，取消后，不会再收到该频道的消息

5，按模式订阅和取消订阅

psubscribe ch\* //订阅以ch开头的所有频道



punsubscribe ch\* //取消以ch开头的所有频道



6，应用场景：

1、今日头条订阅号、微信订阅公众号、新浪微博关注、邮件订阅系统

2、即使通信系统

3、群聊部落系统（微信群）

**测试实践**：微信班级群 class：20170101

1、学生C订阅一个主题叫 ：class:20170101

>subscribe class:20170101

2、学生A针对class:20170101主体发送消息，那么所有订阅该主题的用户都能够接收到该数据。

>publish class:20170101 "hello world! I am A"

3、学生B针对class:20170101主体发送消息，那么所有订阅该主题的用户都能够接收到该数据。

>publish class:20170101 "hello world! I am B"

展示学生C接受到的A\B同学发送过来的消息信息

1) "subscribe"

2) "class:20170101"

3) (integer) 1

1) "message"

2) "class:20170101"

3) "hello world! I am A"

1) "message"

2) "class:20170101"

3) "hello word! I am B"

**代码例子见：redis-publish-subscribe工程**

1. 客户端细解

4.1 Jedis基本用法：

连接池建立－〉访问密码设置－〉连接超时等参数设置

详看 cacheDemo的JedisUtils工具类

pom.xml引入以下依赖即可

<dependency>

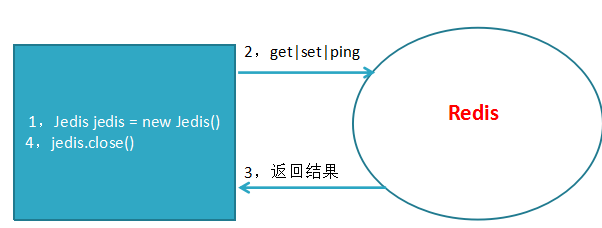
<groupId>redis.clients</groupId>

<artifactId>jedis</artifactId>

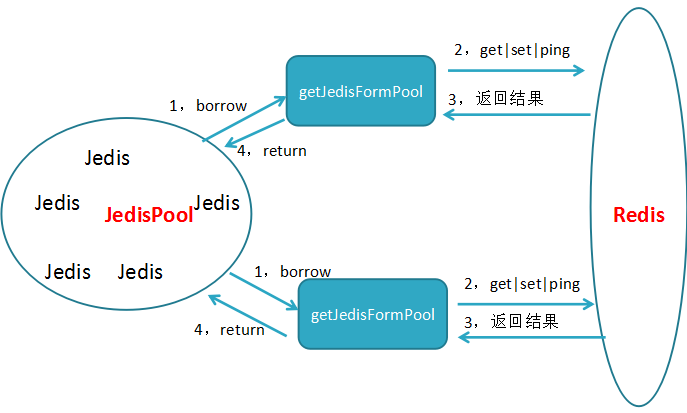
<version>2.9.0</version>

</dependency>

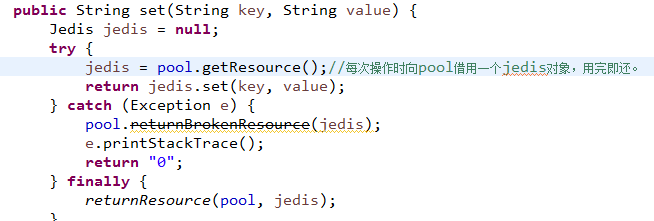
1， jedis直接连接redis如下：



2，使用连接池方式



生产环境一般使用连接池进行操作，jedis连接redis对象放在连接池里，每次用的时候去POOL借用，用完后归还



4.2序列化与反序列化

指把结构化的对象变成无结构的字节流，便于存储、传输，保持一个类在传递数据的有序性，使接收到的数据更具有保证，而反序列化是利用类成员变量反射成为一个类

（即把对象序列化后存redis, 从redis取值后反序列化为JAVA对象）

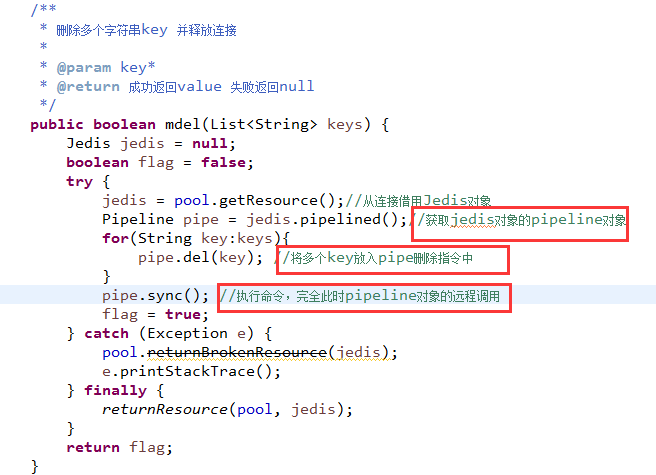
序列化的工具依赖包

|  |
| --- |
| <**dependency**>      <**groupId**>com.dyuproject.protostuff</**groupId**>      <**artifactId**>protostuff-runtime</**artifactId**>      <**version**>1.1.3</**version**>  </**dependency**>  <**dependency**>      <**groupId**>com.dyuproject.protostuff</**groupId**>      <**artifactId**>protostuff-core</**artifactId**>      <**version**>1.1.3</**version**>  </**dependency**> |

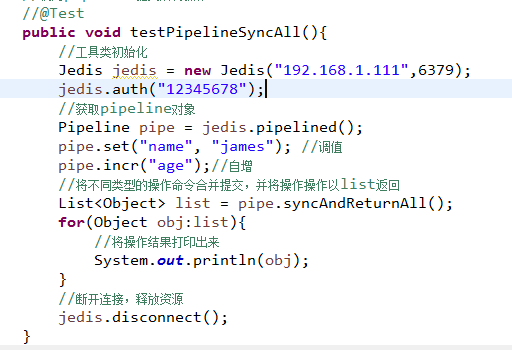
实例：见cache-demo的SerializerTest测试用例

4.3Jedis中的pipeline使用方式

大家知道redis提供了mset、mget方法，但没有提供mdel方法，如果想实现，可以借助pipeline实现，详见cache-demo, 看JedisAllCommandTest用例testPipelineMdel方法



下面代码可将set和incr做一次pipiline操作，看JedisAllCommandTest用例testPipelineSyncAll方法

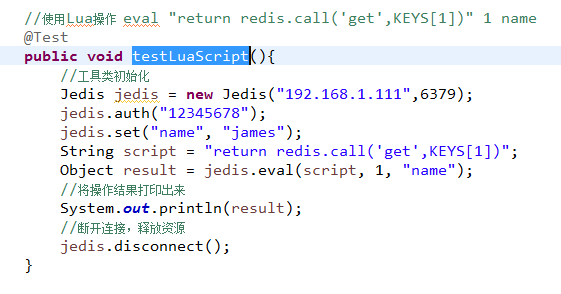


4.4Jedis中的lua脚本

1. set name james

6379>eval "return redis.call('get',KEYS[1])" 1 name //1个键,键名为name,返回james

可以看**JedisAllCommandTest**用例testLuaScript方法



如何执行lua文件呢?

**例子请看cache-demo的testLuaFile测试用例方法**

1. redis持久化

redis支持RDB和AOF两种持久化机制，持久化可以避免因进程退出而造成数据丢失；

5.1RDB持久化把当前进程数据生成快照（.rdb）文件保存到硬盘的过程，有手动触发和自动触发

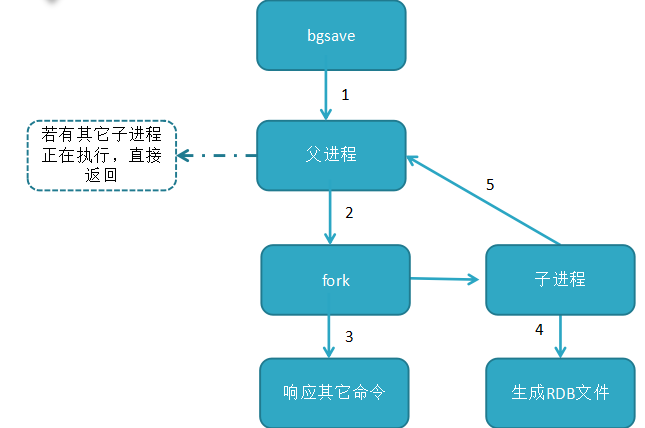
手动触发有save和bgsave两命令

save命令：阻塞当前Redis，直到RDB持久化过程完成为止，若内存实例比较大会造成长时间阻塞，线上环境不建议用它

bgsave命令：redis进程执行fork操作创建子线程，由子线程完成持久化，阻塞时间很短（微秒级），是save的优化,在执行redis-cli shutdown关闭redis服务时，如果没有开启AOF持久化，自动执行bgsave;

显然bgsave是对save的优化。

bgsave运行流程



5.2RDB文件的操作

命令：config set dir /usr/local //设置rdb文件保存路径

备份：bgsave //将dump.rdb保存到usr/local下

恢复：将dump.rdb放到redis安装目录与redis.conf同级目录，重启redis即可

优点：1，压缩后的二进制文，适用于备份、全量复制，用于灾难恢复

2，加载RDB恢复数据远快于AOF方式

缺点：1，无法做到实时持久化，每次都要创建子进程，频繁操作成本过高

2，保存后的二进制文件，存在老版本不兼容新版本rdb文件的问题

5.3AOF持久化

针对RDB不适合实时持久化，redis提供了AOF持久化方式来解决

开启：redis.conf设置：appendonly yes (默认不开启，为no)

默认文件名：appendfilename "appendonly.aof"

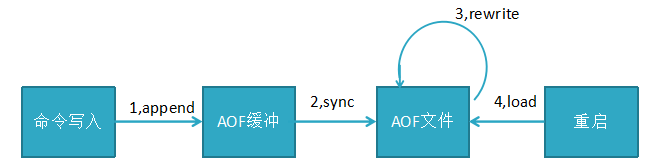
流程说明：1，所有的写入命令(set hset)会append追加到aof\_buf缓冲区中

2，AOF缓冲区向硬盘做sync同步

3，随着AOF文件越来越大，需定期对AOF文件rewrite重写，达到压缩

4，当redis服务重启，可load加载AOF文件进行恢复

AOF持久化流程：命令写入(append),文件同步(sync),文件重写(rewrite),重启加载(load)



redis的AOF配置详解：

appendonly yes //启用aof持久化方式

# appendfsync always //每收到写命令就立即强制写入磁盘，最慢的，但是保证完全的持久化，不推荐使用

appendfsync everysec //每秒强制写入磁盘一次，性能和持久化方面做了折中，推荐

# appendfsync no //完全依赖os，性能最好,持久化没保证（操作系统自身的同步）

no-appendfsync-on-rewrite yes //正在导出rdb快照的过程中,要不要停止同步aof

auto-aof-rewrite-percentage 100 //aof文件大小比起上次重写时的大小,增长率100%时,重写

auto-aof-rewrite-min-size 64mb //aof文件,至少超过64M时,重写

如何从AOF恢复？

1. 设置appendonly yes；

2. 将appendonly.aof放到dir参数指定的目录；

3. 启动Redis，Redis会自动加载appendonly.aof文件。

redis重启时恢复加载AOF与RDB顺序及流程：

1，当AOF和RDB文件同时存在时，优先加载

2，若关闭了AOF，加载RDB文件

3，加载AOF/RDB成功，redis重启成功

4，AOF/RDB存在错误，redis启动失败并打印错误信息

1. 复制:
2. 主从复制

**a,**方式一、新增redis6380.conf, 加入 slaveof 192.168.1.111 6379, 在6379启动完后再启6380，完成配置；

**b,**方式二、redis-server --slaveof 192.168.1.111 6379

**c,**查看状态：info replication

**d,**断开主从复制：在slave节点，执行6380:>slaveof no one

**e,**断开后再变成主从复制：6380:> slaveof 192.168.1.111 6379

**f,**数据较重要的节点，主从复制时使用密码验证： requirepass

**e,**从节点建议用只读模式slave-read-only=yes, 若从节点修改数据，主从数据不一致

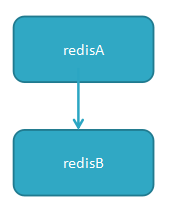
**h,**传输延迟：主从一般部署在不同机器上，复制时存在网络延时问题，redis提供repl-disable-tcp-nodelay参数决定是否关闭TCP\_NODELAY,默认为关闭

参数关闭时：无论大小都会及时发布到从节点，占带宽，适用于主从网络好的场景，

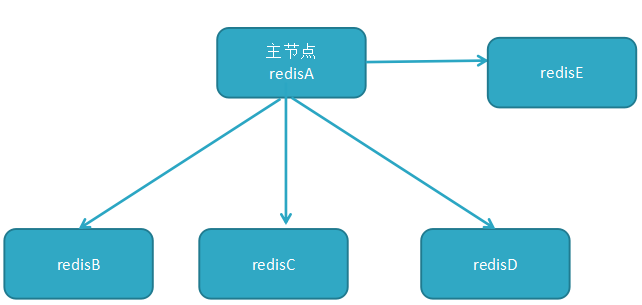
参数启用时：主节点合并所有数据成TCP包节省带宽，默认为40毫秒发一次，取决于内核，主从的同步延迟40毫秒，适用于网络环境复杂或带宽紧张，如跨机房

2，主从拓扑：支持单层或多层

A,一主一从：用于主节点故障转移从节点，当主节点的“写”命令并发高且需要持久化，可以只在从节点开启AOF（主节点不需要），这样即保证了数据的安全性，也避免持久化对主节点的影响

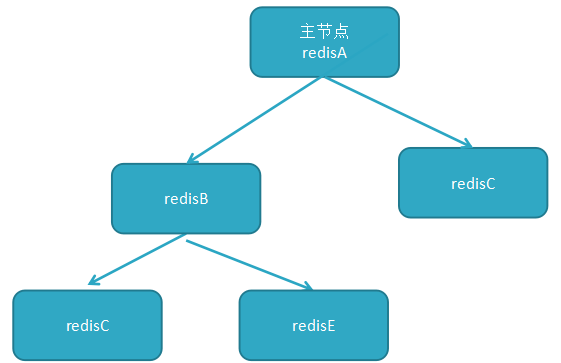


B, 一主多从：针对“读”较多的场景，“读”由多个从节点来分担，但节点越多，主节点同步到多节点的次数也越多，影响带宽，也加重主节点的稳定



C,树状主从：一主多从的缺点（主节点推送次数多压力大）可用些方案解决，

主节点只推送一次数据到从节点1，再由从节点2推送到11，减轻主节点推送的压力。



1. 复制原理

执行slave master port后，

与主节点连接，同步主节点的数据,6380:>info replication：查看主从及同步信息

master:6379

（配置完slave of 127.0.0.1 6379）

slave 6380启动

1，保存主节点信息

2，主从建立socket连接

3，发送ping命令

4，权限验证

5，同步数据集

6，命令持续复制

1. 数据同步：

redis 2.8版本以上使用psync命令完成同步，过程分“全量”与“部分”复制

全量复制：一般用于初次复制场景（第一次建立SLAVE后全量）

部分复制：网络出现问题，从节占再次连主时，主节点补发缺少的数据，每次 数据增加同步

心跳：主从有长连接心跳，主节点默认每10S向从节点发ping命令，repl-ping-slave-period控制发送频率

1. 哨兵机制:

1，为什么要讲哨兵机制？

A，我们学习了redis的主从复制，但如果说主节点出现问题不能提供服务，需要人工重新把从节点设为主节点，还要通知我们的应用程序更新了主节点的地址，这种处理方式不是科学的，耗时费事

B，同时主节点的写能力是单机的，能力能限

C，而且主节点是单机的，存储能力也有限

其中2，3的问题在后面redis集群课会讲，第1个问题我们用哨兵机制来解决

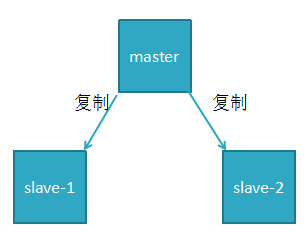
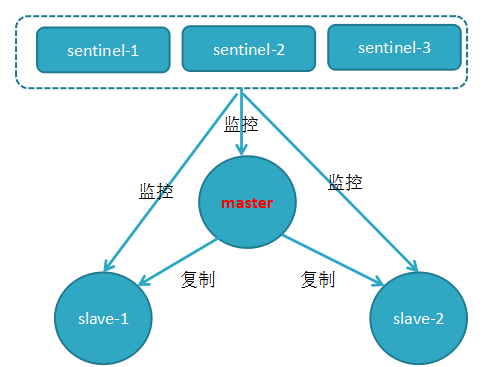
2,主从故障如何故障转移(不满足高可用)：

A，主节点(master)故障，从节点slave-1端执行 slaveof no one后变成新主节点

B，其它的节点成为新主节点的从节点，并从新节点复制数据

3，哨兵机制(sentinel)的高可用：

A，原理：当主节点出现故障时，由redis sentinel自动完成故障发现和转移，并通知应用方，实现高可用性。

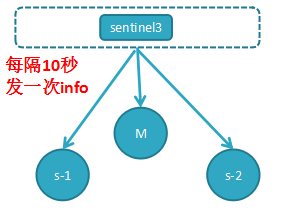
 

主从复制与redis sentinel拓扑结构图

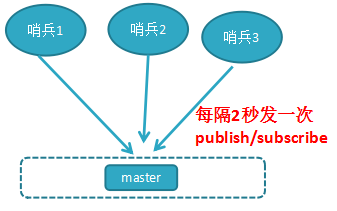
其实整个过程只需要一个哨兵节点来完成，首先使用Raft算法（感兴趣的同学可以查一下，其实就是个选举算法）实现选举机制，选出一个哨兵节点来完成转移和通知

哨兵有三个定时监控任务完成对各节点的发现和监控：

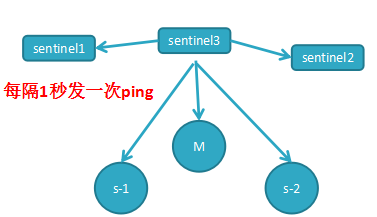
任务1，每个哨兵节点每10秒会向主节点和从节点发送info命令获取最拓扑结构图，哨兵配置时只要配置对主节点的监控即可，通过向主节点发送info，获取从节点的信息，并当有新的从节点加入时可以马上感知到



任务2，每个哨兵节点每隔2秒会向redis数据节点的指定频道上发送该哨兵节点对于主节点的判断以及当前哨兵节点的信息，同时每个哨兵节点也会订阅该频道，来了解其它哨兵节点的信息及对主节点的判断，其实就是通过消息publish和subscribe来完成的；

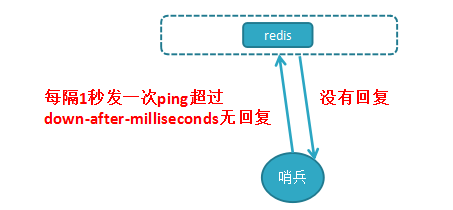


任务3，每隔1秒每个哨兵会向主节点、从节点及其余哨兵节点发送一次ping命令做一次心跳检测，这个也是哨兵用来判断节点是否正常的重要依据

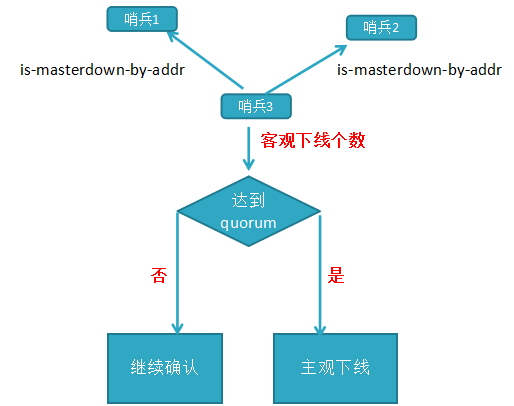


主观下线和客观下线：

主观下线：刚我知道知道哨兵节点每隔1秒对主节点和从节点、其它哨兵节点发送ping做心跳检测，当这些心跳检测时间超过down-after-milliseconds时，哨兵节点则认为该节点错误或下线，这叫主观下线；这可能会存在错误的判断。



客观下线：当主观下线的节点是主节点时，此时该哨兵3节点会通过指令sentinel is-masterdown-by-addr寻求其它哨兵节点对主节点的判断，当超过quorum（法定人数）个数，此时哨兵节点则认为该主节点确实有问题，这样就客观下线了，大部分哨兵节点都同意下线操作，也就说是客观下线

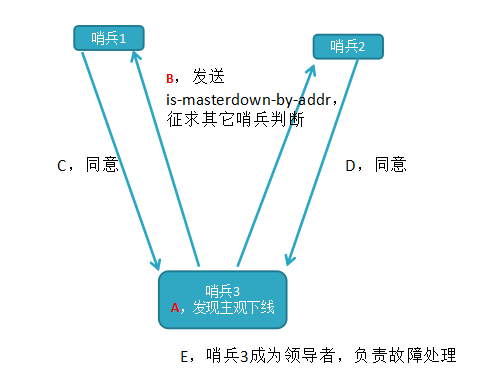


领导者哨兵选举流程：

a,每个在线的哨兵节点都可以成为领导者，当它确认（比如哨兵3）主节点下线时，会向其它哨兵发is-master-down-by-addr命令，征求判断并要求将自己设置为领导者，由领导者处理故障转移；

b,当其它哨兵收到此命令时，可以同意或者拒绝它成为领导者；

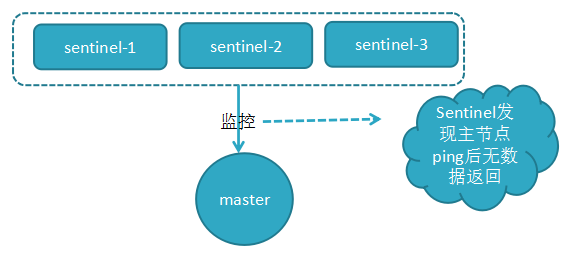
c,如果哨兵3发现自己在选举的票数大于等于num(sentinels)/2+1时，将成为领导者，如果没有超过，继续选举…………



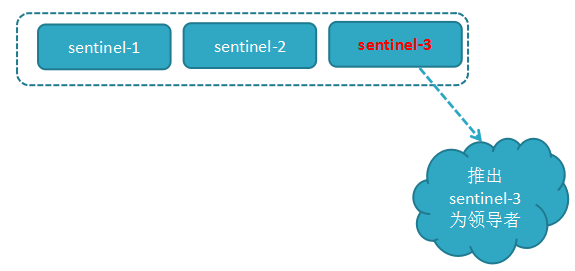
4，故障转移机制

A，由Sentinel节点定期监控发现主节点是否出现了故障

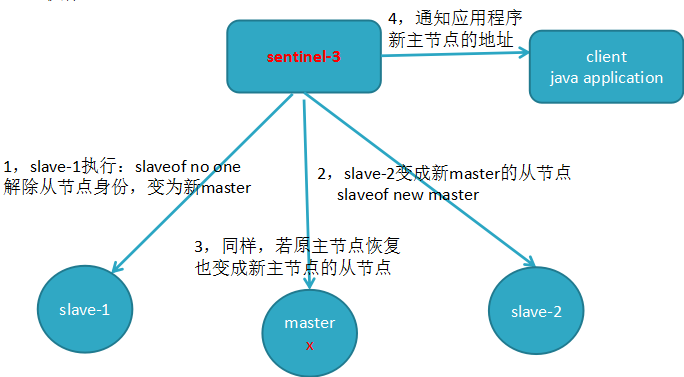
sentinel会向master发送心跳PING来确认master是否存活，如果master在“一定时间范围”内不回应PONG 或者是回复了一个错误消息，那么这个sentinel会主观地(单方面地)认为这个master已经不可用了



1. 当主节点出现故障，此时3个Sentinel节点共同选举了Sentinel3节点为领导，负载处理主节点的故障转移,



1. 由Sentinel3领导者节点执行故障转移，过程和主从复制一样，但是自动执行



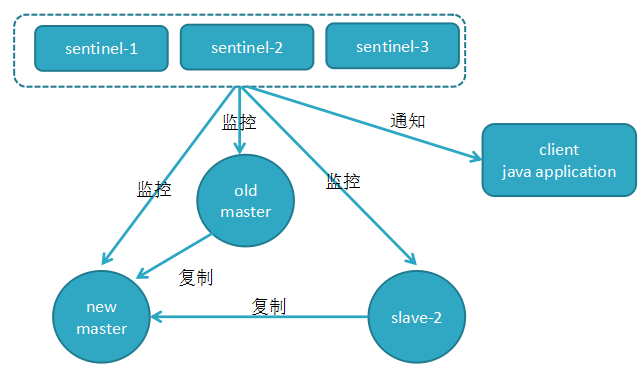
流程： 1，将slave-1脱离原从节点，升级主节点，

2，将从节点slave-2指向新的主节点

3，通知客户端主节点已更换

4，将原主节点（oldMaster）变成从节点，指向新的主节点

D，故障转移后的redis sentinel的拓扑结构图

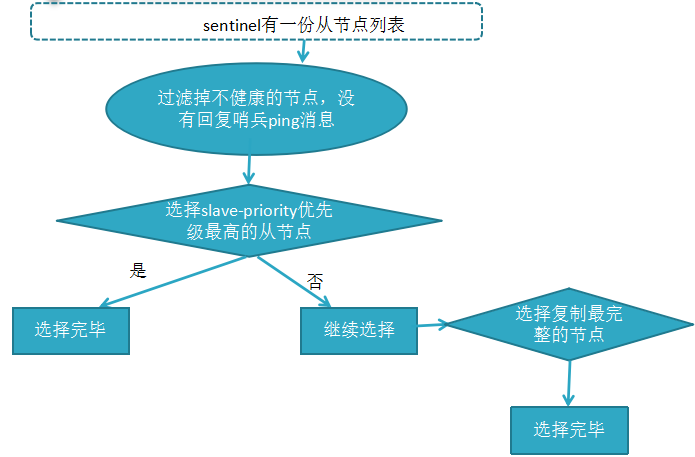


5，哨兵机制－故障转移详细流程

A，过滤掉不健康的（下线或断线），没有回复过哨兵ping响应的从节点

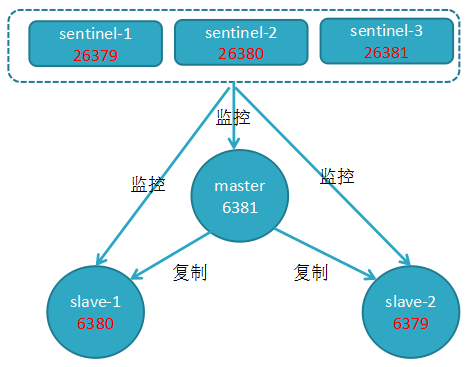
B，选择salve-priority从节点优先级最高（redis.conf）

C，选择复制偏移量最大，指复制最完整的从节点



5，如何安装和部署Reids Sentinel?

我们以3个Sentinel节点、2个从节点、1个主节点为例进行安装部署



**1，前提：**先搭好一主两从redis的主从复制，和之前复制搭建一样，搭建方式如下：

A主节点6379节点（/usr/local/bin/conf/redis6379.conf）：

修改 requirepass 12345678，注释掉#bind 127.0.0.1

B从节点redis6380.conf和redis6381.conf:

修改 requirepass 12345678 ,注释掉#bind 127.0.0.1,

加上masterauth 12345678 ,加上slaveof 127.0.0.1 6379

**注意：**当主从起来后，主节点可读写，从节点只可读不可写

**2，redis sentinel哨兵机制核心配置**(也是3个节点)：

/usr/local/bin/conf/sentinel\_26379.conf

/usr/local/bin/conf/sentinel\_26380.conf

/usr/local/bin/conf/sentinel\_26381.conf

将三个文件的端口改成: 26379 26380 26381

然后：sentinel monitor mymaster 190.168.1.111 **6379** 2 //监听主节点6379

sentinel auth-pass mymaster 12345678 //连接主节点时的密码

三个配置除端口外，其它一样。

**3，哨兵其它的配置**：只要修改每个sentinel.conf的这段配置即可：

sentinel monitor mymaster 192.168.1.10 6379 2

//监控主节点的IP地址端口，sentinel监控的master的名字叫做mymaster

2代表，当集群中有2个sentinel认为master死了时，才能真正认为该master已经不可用了

sentinel auth-pass mymaster 12345678 //sentinel连主节点的密码

sentinel config-epoch mymaster 2 //故障转移时最多可以有2从节点同时对新主节点进行数据同步

sentinel leader-epoch mymaster 2

sentinel failover-timeout mymasterA **180000** //故障转移超时时间180s，

a,如果转移超时失败，下次转移时时间为之前的2倍；

b,从节点变主节点时，从节点执行slaveof no one命令一直失败的话，当时间超过**180S**时，则故障转移失败

c,从节点复制新主节点时间超过**180S**转移失败

sentinel down-after-milliseconds mymasterA **300000**//sentinel节点定期向主节点ping命令，当超过了**300S**时间后没有回复，可能就认定为此主节点出现故障了……

sentinel parallel-syncs mymasterA **1** //故障转移后，**1**代表每个从节点按顺序排队一个一个复制主节点数据，如果为3，指3个从节点同时并发复制主节点数据，不会影响阻塞，但存在网络和IO开销

**4，启动sentinel服务:**

./redis-sentinel conf/sentinel\_26379.conf &

./redis-sentinel conf/sentinel\_26380.conf &

./redis-sentinel conf/sentinel\_26381.conf &

关闭：./redis-cli -h 192.168.1.111 -p 26379 shutdown

**5，测试：**kill -9 6379 杀掉6379的redis服务

看日志是分配6380 还是6381做为主节点，当6379服务再启动时，已变成从节点

假设6380升级为主节点:进入6380>info replication 可以看到role:master

打开sentinel\_26379.conf等三个配置，sentinel monitor mymaster 127.0.0.1 **6380** 2

打开redis6379.conf等三个配置, slaveof 192.168.1.111 6380,也变成了6380

注意：生产环境建议让redis Sentinel部署到不同的物理机上。

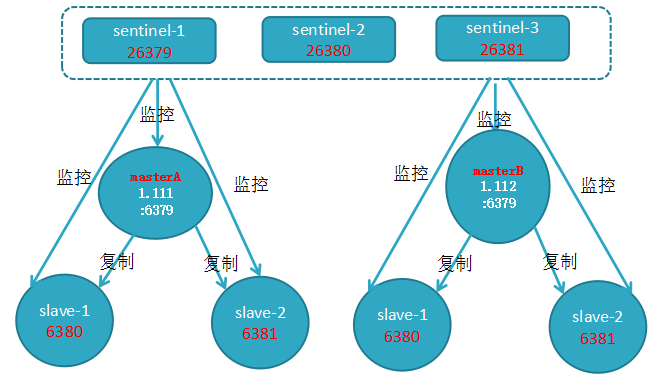
重要：sentinel monitor mymaster **192.168.1.111** 6379 2 //**切记将IP不要写成127.0.0.1**

**不然使用JedisSentinelPool取jedis连接的时候会变成取127.0.0.1 6379的错误地址**

注：我们稍后要启动四个redis实例，其中端口为6379 的redis设为master，其他两个设为slave 。所以mymaster 后跟的是master的ip和端口，最后一个’2’代表只要有2个sentinel认为master下线，就认为该master客观下线，选举产生新的master。通常最后一个参数不能多于启动的sentinel实例数。

哨兵sentinel个数为奇数，选举嘛，奇数哨兵个才能选举成功，一般建议3个

**6，RedisSentinel如何监控2个redis主节点呢？**



sentinel monitor mymasterB 192.168.1.20 6379 2

……与上面一样…………。

7，部署建议：

a，sentinel节点应部署在多台物理机（线上环境）

b，至少三个且奇数个sentinel节点

c，通过以上我们知道，3个sentinel可同时监控一个主节点或多个主节点

监听N个主节点较多时，如果sentinel出现异常，会对多个主节点有影响，同时还会造成sentinel节点产生过多的网络连接，

一般线上建议还是， 3个sentinel监听一个主节点

8，sentinel哨兵的API

命令：redis-cli -p 26379 //进入哨兵的命令模式，使用redis-cli进入

26379>sentinel masters或sentinel master mymaster //查看redis主节点相关信息

26379>sentinel slaves mymaster //查看从节点状态与相关信息

26379>sentinel sentinels mymaster //查sentinel节点集合信息(不包括当前26379)

26379>sentinel failover mymaster //对主节点强制故障转移，没和其它节点协商

9，客户端连接（redis-sentinel例子工程）

远程客户端连接时，要打开protected-mode no

./redis-cli -p 26380 shutdown //关闭

在使用工程redis-sentinel，调用jedis查询的流程如下：

1，将三个sentinel的IP和地址加入JedisSentinelPool

2，根据IP和地址创建JedisSentinelPool池对象

3，在这个对象创建完后，此时该对象已把redis的主节点

（此时sentinel monitor mymaster 必须写成**192.168.1.111** 6379 2，不能为127.0.0.1，不然查询出来的主节点的IP在客户端就变成了127.0.0.1，拿不到连接了）查询出来了，当客户准备发起查询请求时，调用pool.getResource()借用一个jedis对象，内容包括主节点的IP和端口；

4，将得到jedis对象后，可执行jedis.get(“age”)指令了……。

1. redis集群:

RedisCluster是redis的分布式解决方案，在3.0版本后推出的方案，有效地解决了Redis分布式的需求，当遇到单机内存、并发等瓶颈时，可使用此方案来解决这些问题

**8.1分布式数据库概念：**

1，分布式数据库把整个数据按分区规则映射到多个节点，即把数据划分到多个节点上，每个节点负责整体数据的一个子集

比如我们库有900条用户数据，有3个redis节点，将900条分成3份，分别存入到3个redis节点

900条数据

分区规则

redis-1

redis-2

redis-3

2，分区规则：

常见的分区规则哈希分区和顺序分区，redis集群使用了哈希分区，顺序分区暂用不到，不做具体说明；

rediscluster采用了哈希分区的“虚拟槽分区”方式（哈希分区分节点取余、一致性哈希分区和虚拟槽分区），其它两种也不做介绍，有兴趣可以百度了解一下。

3，虚拟槽分区(槽：slot)

RedisCluster采用此分区，所有的键根据哈希函数(CRC16[key]&16383)映射到0－16383槽内，共16384个槽位，每个节点维护部分槽及槽所映射的键值数据

哈希函数: Hash()=CRC16[key]&16383 按位与

槽与节点的关系如下

0~5460

5461~10921

10922~16383

data

data

data

RedisCluster

keys

hash(key)

用hash函数将键映射到槽，再由槽指向数据

0~5461

5462~10923

10923~16383

reids-1节点

redis-2节点

redis-3节点

槽范围

槽与节点的关系

1092310923

redis用虚拟槽分区原因：1，解耦数据与节点关系，节点自身维护槽映射关系，分布式存储

4，redisCluster的缺陷：

a，键的批量操作支持有限，比如mset, mget，如果多个键映射在不同的槽，就不支持了

b，键事务支持有限，当多个key分布在不同节点时无法使用事务，同一节点是支持事务

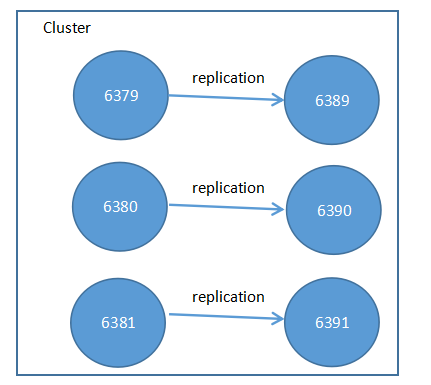
c，键是数据分区的最小粒度，不能将一个很大的键值对映射到不同的节点

d，不支持多数据库，只有0，select 0

e，复制结构只支持单层结构，不支持树型结构。

**8.2集群环境搭建－手动篇**：

1，在/usr/local/bin/clusterconf目录,



6389为6379的从节点，6390为6380的从节点，6391为6381的从节点

2，分别修改6379、 6380、 7381、 6389、 6390、 6391配置文件

port 6379 //节点端口

cluster-enabled yes //开启集群模式

cluster-node-timeout 15000 //节点超时时间（接收pong消息回复的时间）

cluster-config-file /usrlocalbin/cluster/data/nodes-6379.conf 集群内部配置文件

其它节点的配置和这个一致，改端口即可

3，配置完后，**启动6个redis服务**

命令：cd /usr/local/bin/clusterconf/data

cat nodes-6379.conf //查看6379节点ID值



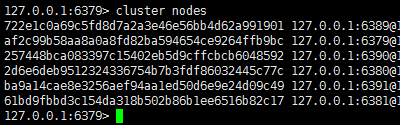
也可以这样查看 6379>cluster nodes



1. 各节点启动后，使用cluster meet ip port与各节点握手，是集群通信的第一步



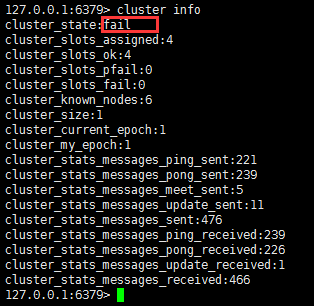
5，握手成功后，使用cluster nodes可以看到各节点都可以互相查询到



6，节点握手成功后，此时集群处理下线状态，所有读写都被禁止



7，使用cluster info命令获取集群当前状态



8，redis集群有16384个哈希槽，要把所有数据映射到16384槽，需要批量设置槽

redis-cli -h 127.0.0.1 -p 6379 cluster addslots {0...5461}

但我的虚拟机按范围分配有问题，同学们回去试一下看有没有问题

错误为： (error) ERR Invalid or out of range slot

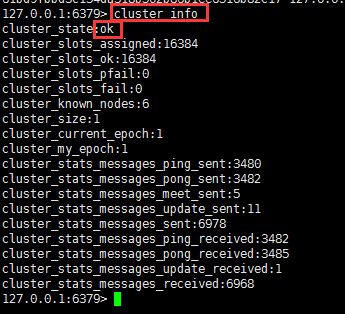
批量不行，单个是可以的

redis-cli -h 127.0.0.1 -p 6379 cluster addslots 1 2 3 4

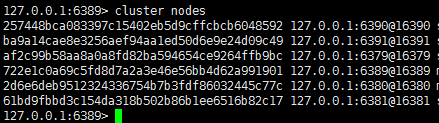
因此，我写一个脚本/usr/local/bin/addSlots.sh, 详情可见些脚本……

执行这个脚本可分配好槽位……

9，分配完槽后，可查看集群状态



1. 然后再查看cluster nodes，查看每个节点的ID



1. 将6389，6390，6391与 6379,6380,6381做主从映射

127.0.0.1:6389> cluster replicate af2c99b58aa8a0a8fd82ba594654ce9264ffb9bc

127.0.0.1:6390> cluster replicate 2d6e6deb9512324336754b7b3fdf86032445c77c

127.0.0.1:6391> cluster replicate 61bd9fbbd3c154da318b502b86b1ee6516b82c17

12，**注：**这是手动模式，在生产环境我们一般采用以下自动模式安装

**---------------------------------------------------------------------------------------------------------------**

13，**自动安装模式：**

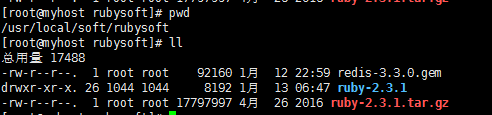
在/usr/local新建目录：ruby



下载链接：https://pan.baidu.com/s/1kWsf3Rh 密码：n3pc

从这个链接下载 ruby-2.3.1.tar.gz 和 redis-3.3.0.gem

tar -zxvf ruby-2.3.1.tar.gz



a, cd ruby-2.3.1

b, ./configure -prefix=/usr/local/ruby

c, make && make install //过程会有点慢，大概5－10分钟

d, 然后gem install -l redis-3.3.0.gem //没有gem需要安装yum install gem

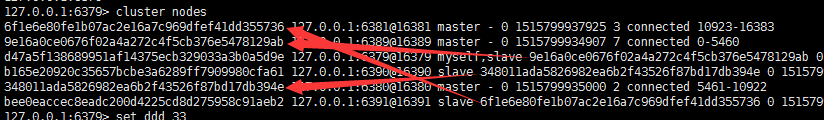
e,准备好6个节点，（注意不要设置requirepass）,将/usr/local/bin/clusterconf/data的config-file删除；依次启动6个节点：./redis-server clusterconf/redis6379.conf

如果之前redis有数据存在，flushall清空；(**坑:不需要cluster meet ..**)

f, 进入cd /usr/local/bin, 执行以下：1代表从节点的个数

./redis-trib.rb create --replicas 1 192.168.1.111:6379 192.168.1.111:6380 192.168.1.111:6381 192.168.1.111:6389 192.168.1.111:6390 192.168.1.111:6391

主从分配，6379是6389的从节点



貌似只有主节点可读写，从节点不可以

主节点死后，从节点变成主节点

e,集群健康检测：

redis-trib.rb check 192.168.1.111:6379 (注：redis先去注释掉requirepass，不然连不上)



如此出现了这个问题，6379的5798槽位号被打开了

解决如下：



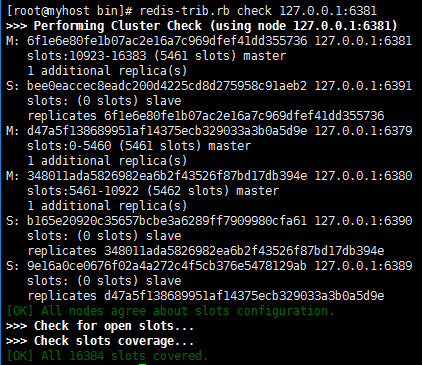
6379，6380，6381的有部分槽位被打开了，分别进入这几个节点，执行

6380:>cluster setslot 1180 stable

cluster setslot 2998 stable

cluster setslot 11212 stable

其它也一样，分别执行修复完后：



此时修复后的健康正常；

当停掉6379后，过会6389变成主节点

注意：使用客户端工具查询时要加-c

./redis-cli -h 192.168.1.111 -p 6379 **-c**

mset aa bb cc dd,批设置对应在不同的solt上，缺点



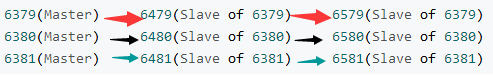
14,集群正常启动后，在每个redis.conf里加上

masterauth “12345678”

requiredpass “12345678”

当主节点下线时，从节点会变成主节点，用户和密码是很有必要的，设置成一致

15,这上面是一主一从，那能不能一主多从呢？



./redis-trib.rb create --replicas 2

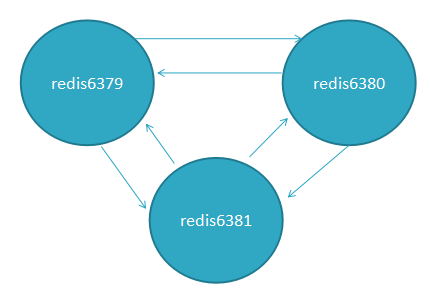
192.168.1.111:6379 192.168.1.111:6380 192.168.1.111:6381

192.168.1.111:6479 192.168.1.111:6480 192.168.1.111:6481

192.168.1.111:6579 192.168.1.111:6580 192.168.1.111:6581

**8.3节点之间的通信**

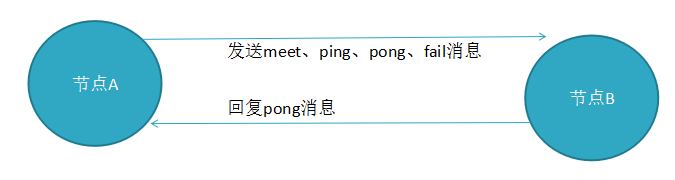
1，节点之间采用Gossip协议进行通信，Gossip协议就是指节点彼此之间不断通信交换信息



当主从角色变化或新增节点，彼此通过ping/pong进行通信知道全部节点的最新状态并达到集群同步

2,Gossip协议

Gossip协议的主要职责就是信息交换，信息交换的载体就是节点之间彼此发送的Gossip消息，常用的Gossip消息有ping消息、pong消息、meet消息、fail消息



meet消息：用于通知新节点加入，消息发送者通知接收者加入到当前集群，meet消息通信完后，接收节点会加入到集群中，并进行周期性ping pong交换

ping消息：集群内交换最频繁的消息，集群内每个节点每秒向其它节点发ping消息，用于检测节点是在在线和状态信息，ping消息发送封装自身节点和其他节点的状态数据；

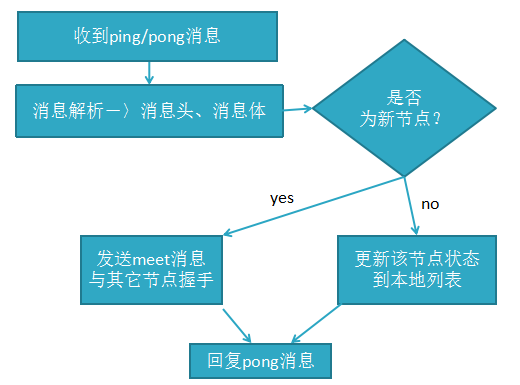
pong消息，当接收到ping meet消息时，作为响应消息返回给发送方，用来确认正常通信，pong消息也封闭了自身状态数据；

fail消息：当节点判定集群内的另一节点下线时，会向集群内广播一个fail消息，后面会讲到。……

**3,**消息解析流程

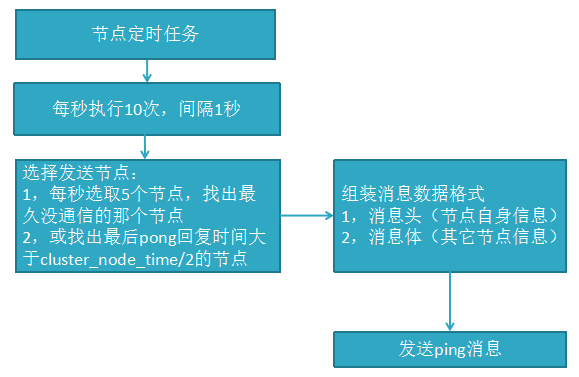
所有消息格式为：消息头、消息体，消息头包含发送节点自身状态数据（比如节点ID、槽映射、节点角色、是否下线等），接收节点根据消息头可以获取到发送节点的相关数据。

消息解析流程：



4,选择节点并发送ping消息：

Gossip协议信息的交换机制具有天然的分布式特性，但ping pong发送的频率很高，可以实时得到其它节点的状态数据，但频率高会加重带宽和计算能力，因此每次都会有目的性地选择一些节点； 但是节点选择过少又会影响故障判断的速度，redis集群的Gossip协议兼顾了这两者的优缺点，看下图：



不难看出：节点选择的流程可以看出消息交换成本主要体现在发送消息的节点数量和每个消息携带的数据量

流程说明：

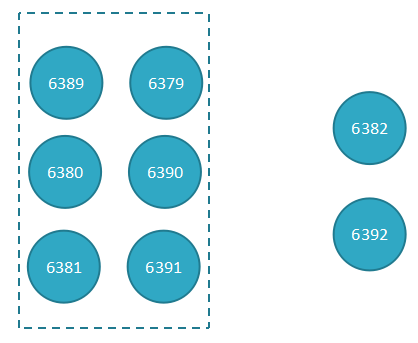
1. 选择发送消息的节点数量：集群内每个节点维护定时任务默认为每秒执行10次，每秒会随机选取5个节点，找出最久没有通信的节点发送ping消息，用来保证信息交换的随机性，每100毫秒都会扫描本地节点列表，如果发现节点最近一次接受pong消息的时间大于cluster-node-timeout/2 则立刻发送ping消息，这样做目的是防止该节点信息太长时间没更新，当我们宽带资源紧张时，在可redis.conf将cluster-node-timeout 15000 改成30秒，但不能过度加大
2. 消息数据：节点自身信息和其他节点信息

5，集群扩容

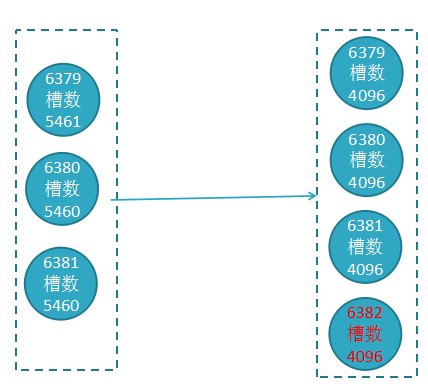
这也是分布式存储最常见的需求，当我们存储不够用时，要考虑扩容

扩容步骤如下：

1. 准备好新节点



1. 加入集群,迁移槽和数据



1),同目录下新增redis6382.conf、redis6392.conf两

启动两个新redis节点

./redis-server clusterconf/redis6382.conf & （新主节点）

./redis-server clusterconf/redis6392.conf & （新从节点）

2),新增主节点

./redis-trib.rb add-node 192.168.1.111:6382 192.168.1.111:6379

6379是原存在的主节点，6382是新的主节点

3),添加从节点

redis-trib.rb add-node --slave --master-id 03ccad2ba5dd1e062464bc7590400441fafb63f2 192.168.1.111:6392 192.168.1.111:6379

--slave，表示添加的是从节点

--master-id 03ccad2ba5dd1e062464bc7590400441fafb63f2表示主节点6382的master\_id

192.168.1.111:6392,新从节点

192.168.1.111:6379集群原存在的旧节点

4)，redis-trib.rb reshard 192.168.1.111:6382 //为新主节点重新分配solt

How many slots do you want to move (from 1 to 16384)? 1000 //设置slot数1000

What is the receiving node ID? 464bc7590400441fafb63f2 //新节点node id

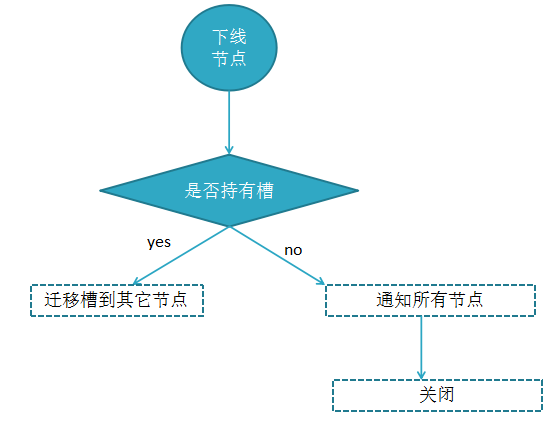
Source node #1:all //表示全部节点重新洗牌

新增完毕！

3，集群减缩节点：

集群同时也支持节点下线掉

下线的流程如下：



流程说明：

1. 确定下线节点是否存在槽slot,如果有，需要先把槽迁移到其他节点，保证整个集群槽节点映射的完整性；
2. 当下线的节点没有槽或本身是从节点时，就可以通知集群内其它节点（或者叫忘记节点），当下线节点被忘记后正常关闭。

删除节点也分两种：

一种是主节点6382，一种是从节点6392。

在从节点6392中，没有分配哈希槽，执行

./redis-trib.rb del-node 192.168.1.111:6392 7668541151b4c37d2d9 有两个参数ip：port 和节点的id。 从节点6392从集群中删除了。

主节点6382删除步骤：

1，./redis-trib.rb reshard 192.168.1.111:6382

问我们有多少个哈希槽要移走，因为我们这个节点上刚分配了1000 个所以我们这里输入1000



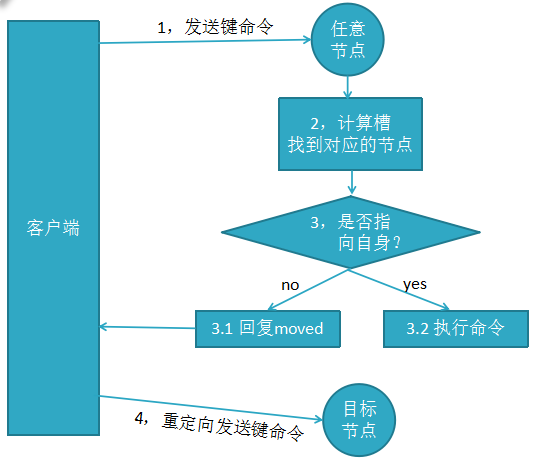
2，最后

./redis-trib.rb del-node 192.168.1.111:6382 3e50c6398c75e0088a41f908071c2c2eda1dc900

此时节点下线完成……

请求路由重定向

我们知道，在redis集群模式下，redis接收的任何键相关命令首先是计算这个键CRC值，通过CRC找到对应的槽位，再根据槽找到所对应的redis节点，如果该节点是本身，则直接处理键命令；如果不是，则回复键重定向到其它节点，这个过程叫做MOVED重定向

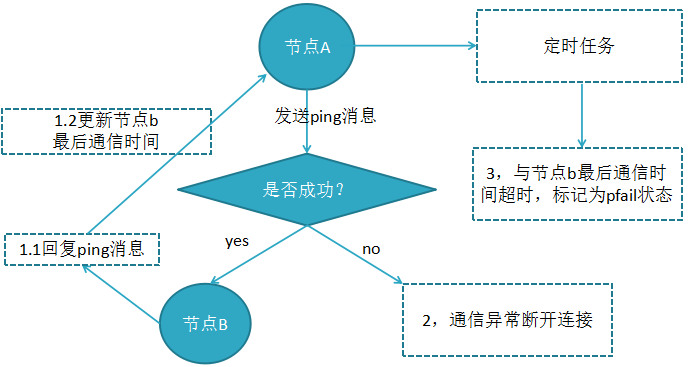


故障转移：

redis集群实现了高可用，当集群内少量节点出现故障时，通过故障转移可以保证集群正常对外提供服务。

当集群里某个节点出现了问题，redis集群内的节点通过ping pong消息发现节点是否健康，是否有故障，其实主要环节也包括了 主观下线和客观下线；

主观下线：指某个节点认为另一个节点不可用，即下线状态，当然这个状态不是最终的故障判定，只能代表这个节点自身的意见，也有可能存在误判；



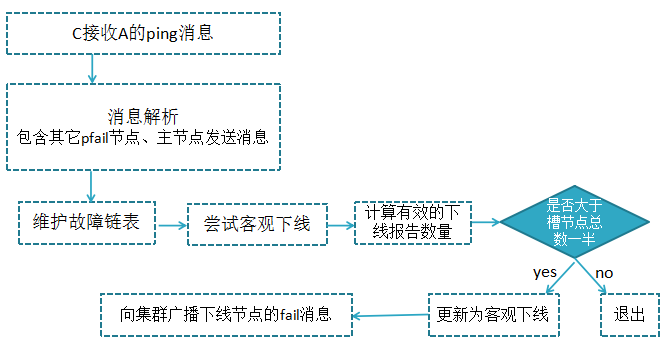
下线流程：

1. 节点a发送ping消息给节点b ,如果通信正常将接收到pong消息，节点a更新最近一次与节点b的通信时间；
2. 如果节点a与节点b通信出现问题则断开连接，下次会进行重连，如果一直通信失败，则它们的最后通信时间将无法更新；
3. 节点a内的定时任务检测到与节点b最后通信时间超过cluster\_note-timeout时，更新本地对节点b的状态为主观下线（pfail）

客观下线：指真正的下线，集群内多个节点都认为该节点不可用，达成共识，将它下线，如果下线的节点为主节点，还要对它进行故障转移

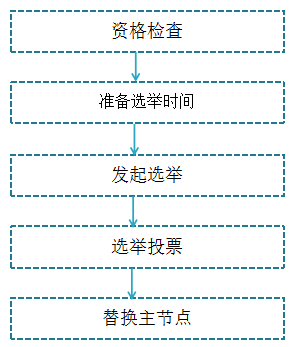
假如节点a标记节点b为主观下线，一段时间后节点a通过消息把节点b的状态发到其它节点，当节点c接受到消息并解析出消息体时，会发现节点b的pfail状态时，会触发客观下线流程；

当下线为主节点时，此时redis集群为统计持有槽的主节点投票数是否达到一半，当下线报告统计数大于一半时，被标记为客观下线状态。



故障恢复：

故障主节点下线后，如果下线节点的是主节点，则需要在它的从节点中选一个替换它，保证集群的高可用；转移过程如下：



1. 资格检查：检查该从节点是否有资格替换故障主节点，如果此从节点与主节点断开过通信，那么当前从节点不具体故障转移；
2. 准备选举时间：当从节点符合故障转移资格后，更新触发故障选举时间，只有到达该时间后才能执行后续流程；
3. 发起选举：当到达故障选举时间时，进行选举；
4. 选举投票：只有持有槽的主节点才有票，会处理故障选举消息，投票过程其实是一个领导者选举（选举**从节点为领导者**）的过程，每个主节点只能投一张票给从节点，

当从节点收集到足够的选票（大于N/2+1）后，触发替换主节点操作，撤销原故障主节点的槽，委派给自己，并广播自己的委派消息，通知集群内所有节点。