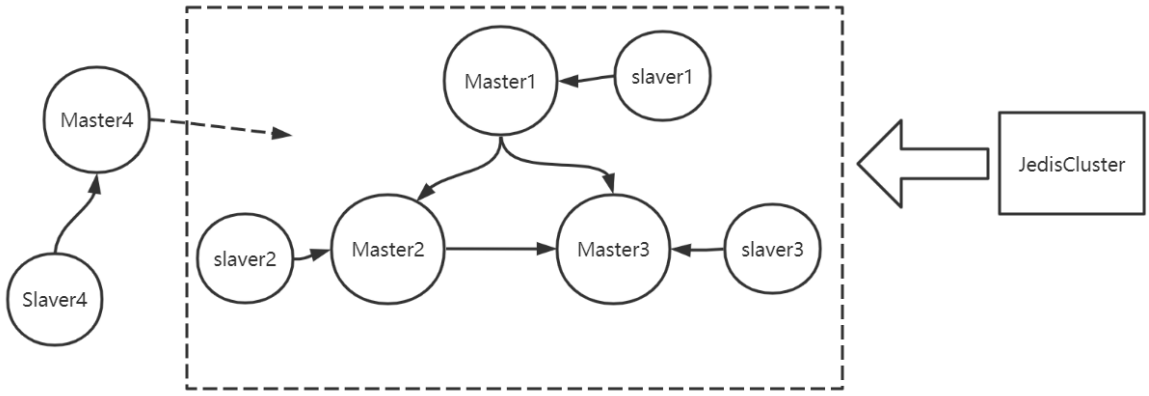
作业：RedisCluster的安装、部署、扩容和Java客户端调用



如图

（1）搭建Redis5.0集群，要求三主三从，记录下安装步骤

（2）能够添加一主一从（Master4和Slaver4），记录下安装步骤

（3）能够通过JedisCluster向RedisCluster添加数据和取出数据

Redis提问

4班

先写缓存，缓存更新数据库为何比先写数据库然后更缓存效率高?

先写缓存，缓存更新数据库，不需要应用服务器处理，应用服务器压力会降低些

但需要缓存支持直读或直写

redis的7种基本数据类型如何选择用什么？插入数据的时候是自动选择数据类型吗

根据业务需求选择，具体详见Redis数据类型操作

3班

1、发现一主一从的情况下直接点挂掉，从节点没有接管，一主一从不能保证高可用，需要至少两从?

这个问题在我这没有发现。

```
[root@localhost bin]# cat nodes.conf
9a66709978d93a9514743d1db113dc1b0f955f89 192.168.72.128:7005@17005 slave
6e0653801dbdb52f38ca705c3cd7c32d90ebfc94 0 1600056111000 5 connected
0208af3f81f997740e1b29590b8c7123cb6a62c9 192.168.72.128:7001@17001 myself,slave
8712fb5f596307bc8a184f8aa5f5544ad40f0b6b 0 1600056109000 1 connected
8712fb5f596307bc8a184f8aa5f5544ad40f0b6b 192.168.72.128:7006@17006 master - 0
1600056111090 7 connected 0-5460
6e0653801dbdb52f38ca705c3cd7c32d90ebfc94 192.168.72.128:7003@17003 master - 0
1600056110000 3 connected 10923-16383
70a60c1f64e53dbff5bf0ffd8442e7e3ed2fe837 192.168.72.128:7002@17002 master,fail -
1600056034389 1600056030450 2 disconnected
0adfdb139d7fb2847e15f3871e5766a2365d01ce 192.168.72.128:7004@17004 master - 0
1600056112230 9 connected 5461-10922
# 重新启动7002
[root@localhost bin]# cat nodes.conf
9a66709978d93a9514743d1db113dc1b0f955f89 192.168.72.128:7005@17005 slave
6e0653801dbdb52f38ca705c3cd7c32d90ebfc94 0 1600056345000 5 connected
0208af3f81f997740e1b29590b8c7123cb6a62c9 192.168.72.128:7001@17001 myself,slave
8712fb5f596307bc8a184f8aa5f5544ad40f0b6b 0 1600056345000 1 connected
8712fb5f596307bc8a184f8aa5f5544ad40f0b6b 192.168.72.128:7006@17006 master - 0
1600056345178 7 connected 0-5460
6e0653801dbdb52f38ca705c3cd7c32d90ebfc94 192.168.72.128:7003@17003 master - 0
1600056345000 3 connected 10923-16383
70a60c1f64e53dbff5bf0ffd8442e7e3ed2fe837 192.168.72.128:7002@17002 slave
0adfdb139d7fb2847e15f3871e5766a2365d01ce 0 1600056345984 9 connected
0adfdb139d7fb2847e15f3871e5766a2365d01ce 192.168.72.128:7004@17004 master - 0
1600056341000 9 connected 5461-10922
vars currentEpoch 9 lastVoteEpoch 0
```

2、可以详细说一下redis-cli下 —cluster help下的命令吗?

```
[root@localhost bin]# ./redis-cli --cluster help

Cluster Manager Commands:
  create         host1:port1 ... hostN:portN   #创建集群
                 --cluster-replicas <arg>      #从节点个数
  check          host:port                     #检查集群
                 --cluster-search-multiple-owners #检查是否有槽同时被分配给了多个节点
  info           host:port                     #查看集群状态
  fix            host:port                     #修复集群
                 --cluster-search-multiple-owners #修复槽的重复分配问题
  reshard        host:port                     #指定集群的任意一节点进行迁移slot，重新分slots
                 --cluster-from <arg>          #需要从哪些源节点上迁移slot，可从多个源节点完成迁移，以逗号隔开，传递的是节点的node id，还可以直接传递--from all，这样源节点就是集群的所有节点，不传递该参数的话，则会在迁移过程中提示用户输入
                 --cluster-to <arg>            #slot需要迁移的目的节点的node id，目的节点只能填写一个，不传递该参数的话，则会在迁移过程中提示用户输入
                 --cluster-slots <arg>         #需要迁移的slot数量，不传递该参数的话，则会在迁移过程中提示用户输入。
                 --cluster-yes                 #指定迁移时的确认输入
                 --cluster-timeout <arg>       #设置migrate命令的超时时间
                 --cluster-pipeline <arg>      #定义cluster getkeysinslot命令一次取出的key数量，不传的话使用默认值为10
```

```
                  --cluster-replace               #是否直接replace到目标节点
  rebalance       host:port                                   #指定集群的任意一节
点进行平衡集群节点slot数量
                  --cluster-weight <node1=w1...nodeN=wN>      #指定集群节点的权重
                  --cluster-use-empty-masters                 #设置可以让没有分配
slot的主节点参与，默认不允许
                  --cluster-timeout <arg>                     #设置migrate命令的
超时时间
                  --cluster-simulate                          #模拟rebalance操
作，不会真正执行迁移操作
                  --cluster-pipeline <arg>                    #定义cluster
getkeysinslot命令一次取出的key数量，默认值为10
                  --cluster-threshold <arg>                   #迁移的slot阈值超
过threshold，执行rebalance操作
                  --cluster-replace                           #是否直接replace到
目标节点
  add-node        new_host:new_port existing_host:existing_port  #添加节点，把新节点
加入到指定的集群，默认添加主节点
                  --cluster-slave                             #新节点作为从节点，
默认随机一个主节点
                  --cluster-master-id <arg>                   #给新节点指定主节点
  del-node        host:port node_id                           #删除给定的一个节
点，成功后关闭该节点服务
  call            host:port command arg arg .. arg            #在集群的所有节点执
行相关命令
  set-timeout     host:port milliseconds                      #设置cluster-
node-timeout
  import          host:port                                   #将外部redis数据导
入集群
                  --cluster-from <arg>                        #将指定实例的数据导
入到集群
                  --cluster-copy                              #migrate时指定
copy
                  --cluster-replace                           #migrate时指定
replace
  help

For check, fix, reshard, del-node, set-timeout you can specify the host and port
of any working node in the cluster.
```

3、添加节点时需要指定一台已经在集群中的节点，是用于确认把新节点添加到哪个集群吗?

Gossip协议是一个通信协议，一种传播消息的方式。

sender向receiver发出，请求receiver加入sender的集群

./redis-cli --cluster add-node 192.168.127.128:7007 192.168.127.128:7001

2班

1、哨兵是怎样实现修改redis配置文件的?

sentinel将选择出来的新的主机，修改redis.conf 去掉repliaof或slaveof

sentinel自己的sentinel.conf 修改sentinel monitor mymaster 新的主机IP和端口 2

如果原来的主机重新启动，则sentinel修改它的redis.conf加上repliaof 新的主机和端口或slaveof新的主机和端口

2、集群模式下需要做服务器之间的ssh免密登陆吗?

RedisCluster做密码设置：

redis cluster设置密码有两种方式 1.在集群创建时，配置文件中添加如下两行

```
masterauth passwd
requirepass passwd
```

2.如果集群已经创建好，也可以动态设置密码 在集群的所有实例（包含主节点和从节点）中执行

```
config set masterauth 123456
config set requirepass 123456
auth 123456
config rewrite
```

方法二的效果和方法一是一样的，会在redis的配置文件中写入下面两行配置，并且配置立即生效，不需要重启redis。

```
masterauth 123456
requirepass 123456
```

在访问时，添加密码:

```
[root@localhost bin]# ./redis-cli -p 7001 -a '123456' -c
Warning: Using a password with '-a' or '-u' option on the command line interface
may not be safe.
127.0.0.1:7001> set a2 111
-> Redirected to slot [11786] located at 192.168.72.128:7003
OK
192.168.72.128:7003> get a2
"111"
```

ssh免密登陆一般不使用，因为不安全，早期可以利用Redis的漏洞实现SSH免密登录，Redis5已经补上这个漏洞了。

3、redis集群中总线端口如何修改

每个Redis集群中的节点都需要打开两个TCP连接。一个连接用于正常的给Client提供服务，比如7001，还有一个额外的端口（通过在这个端口号上加10000）作为数据端口，比如17001。第二个端口（17001）用于集群总线，这是一个用二进制协议的点对点通信信道。这个集群总线（Cluster bus）用于节点的失败侦测、配置更新、故障转移授权，等等。客户端从来都不应该尝试和这些集群总线端口通信，它们只应该和正常的Redis命令端口进行通信。注意，确保在你的防火墙中开放着两个端口，否则，Redis集群节点之间将无法通信。

命令端口和集群总线端口的偏移量总是10000。

```
tcp        0      0 0.0.0.0:17001           0.0.0.0:*               LISTEN
tcp        0      0 0.0.0.0:17002           0.0.0.0:*               LISTEN
tcp        0      0 0.0.0.0:17003           0.0.0.0:*               LISTEN
tcp        0      0 0.0.0.0:17004           0.0.0.0:*               LISTEN
tcp        0      0 0.0.0.0:17005           0.0.0.0:*               LISTEN
tcp        0      0 0.0.0.0:17006           0.0.0.0:*               LISTEN
```

4、zookeeper的分布式锁实现源码能提供一下吗?

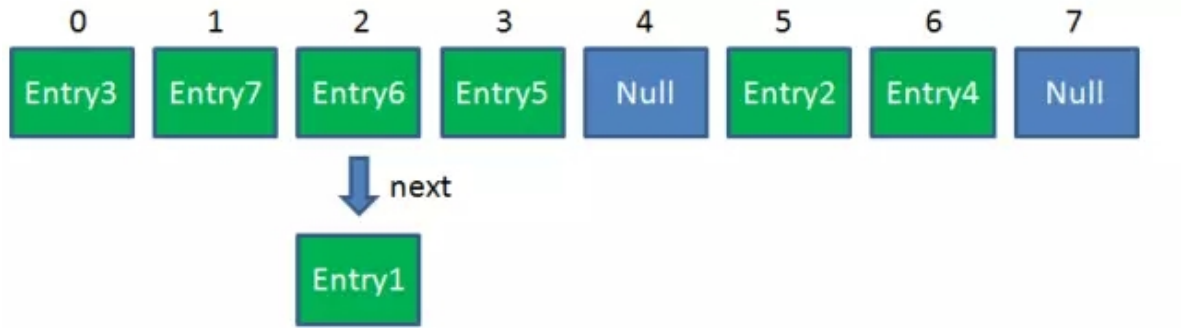
请找讲zk的老师

5、课程中讲本地缓存固定数据时说：“也可以使用JDK的CurrentHashMap，需要自行实现”。想问：既然是固定数据，只读也不能使用HashMap么?

HashMap在高并发下引起的死循环（JDK1.8以前）

HashMap的容量是有限的。当经过多次元素插入，使得HashMap达到一定饱和度时，Key映射位置发生冲突的几率会逐渐提高。

这时候，HashMap需要扩展它的长度，也就是进行Resize。



影响发生Resize的因素有两个：

- **Capacity**

HashMap的当前长度。HashMap的长度是2的幂。

- **LoadFactor**

HashMap负载因子，默认值为0.75f。

衡量HashMap是否进行Resize的条件如下：

HashMap.Size >= Capacity * LoadFactor

- **扩容**

创建一个新的Entry空数组，长度是原数组的2倍。

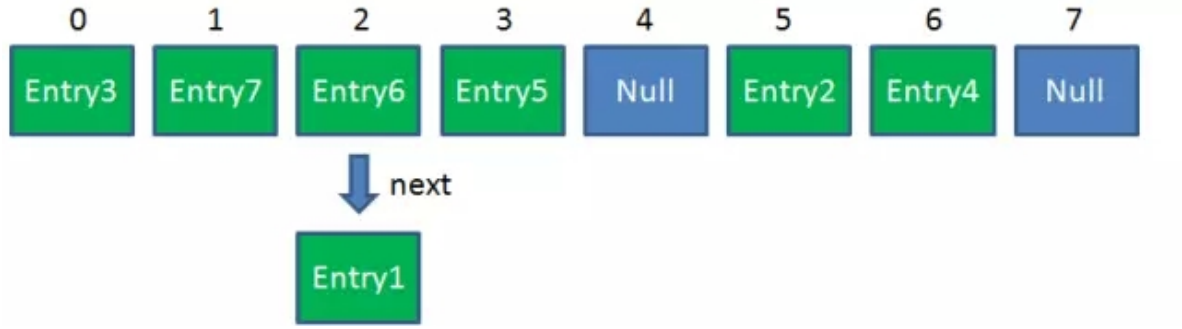
- **ReHash**

遍历原Entry数组，把所有的Entry重新Hash到新数组。为什么要重新Hash呢？因为长度扩大以后，Hash的规则也随之改变。

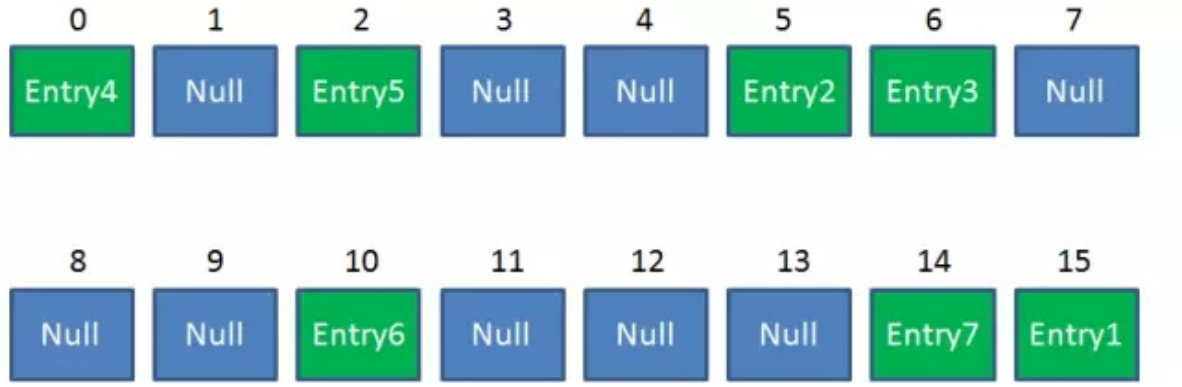
让我们回顾一下Hash公式：

index = HashCode（Key） & （Length - 1）

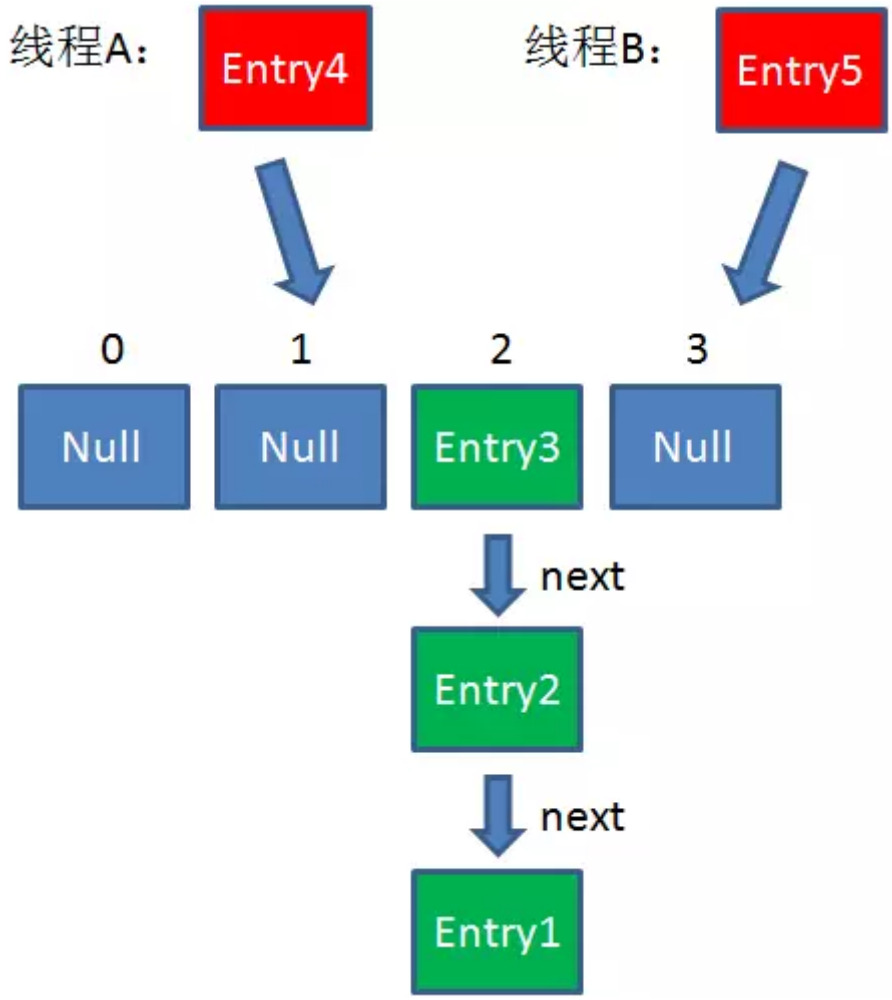
Resize前的HashMap：

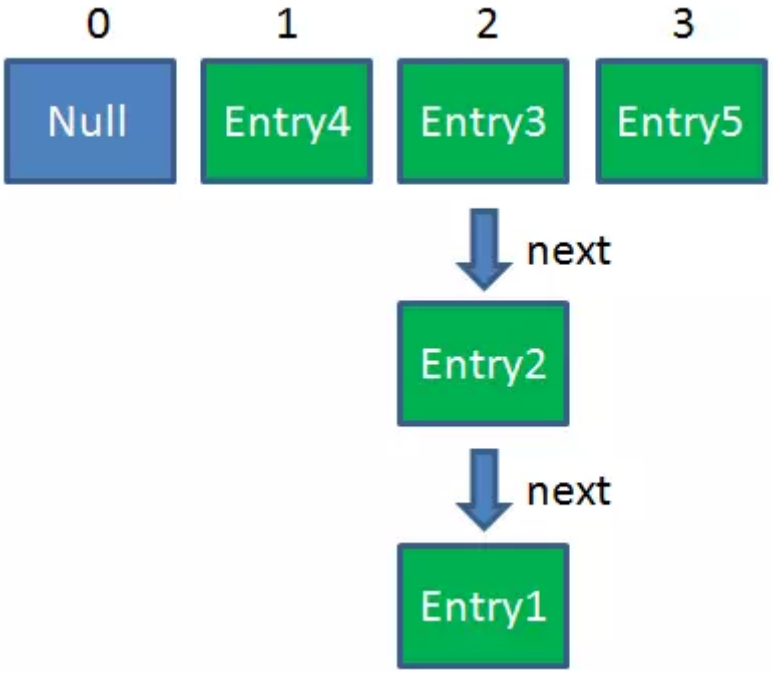


Resize后的HashMap：

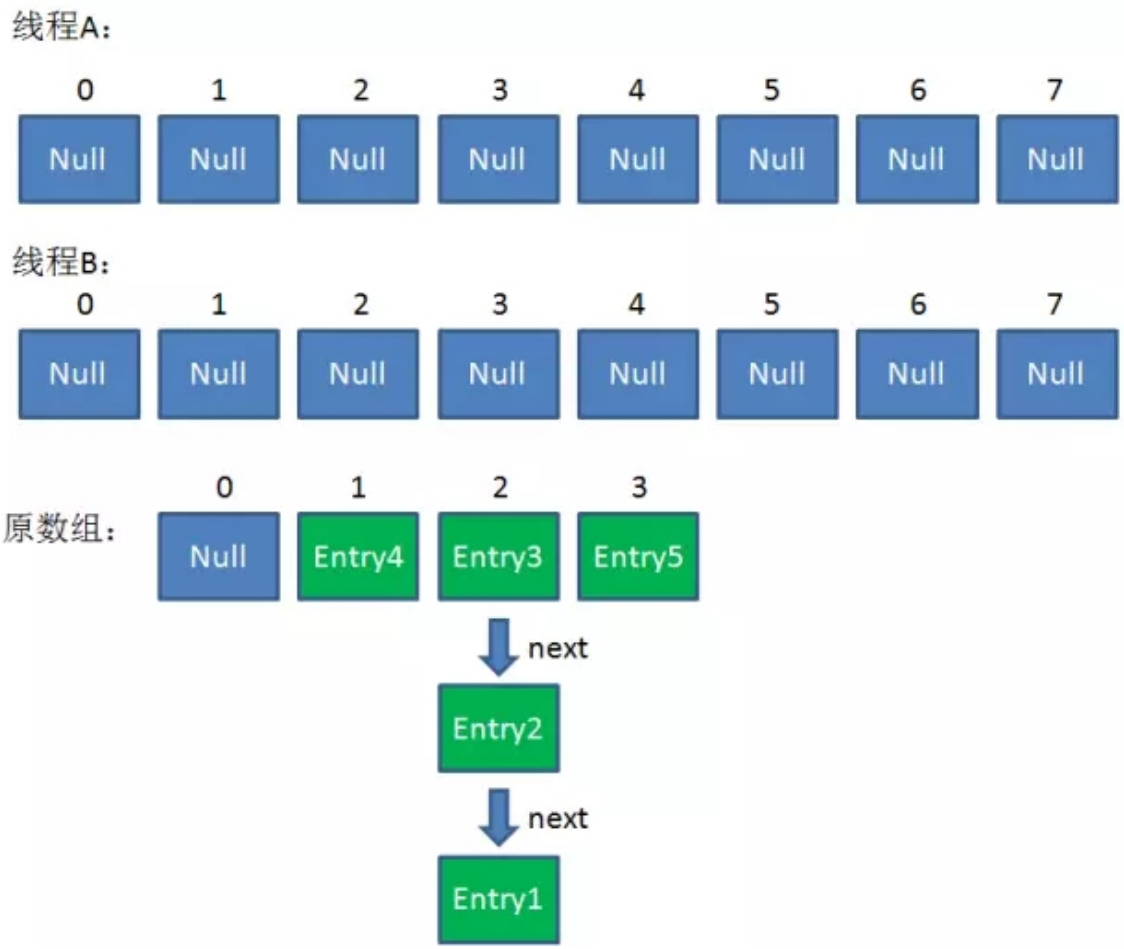


假设一个HashMap已经到了Resize的临界点。此时有两个线程A和B，在同一时刻对HashMap进行Put操作：





此时达到Resize条件，两个线程各自进行Rezie的第一步，也就是扩容：



这时候，两个线程都走到了ReHash的步骤。让我们回顾一下ReHash的代码：

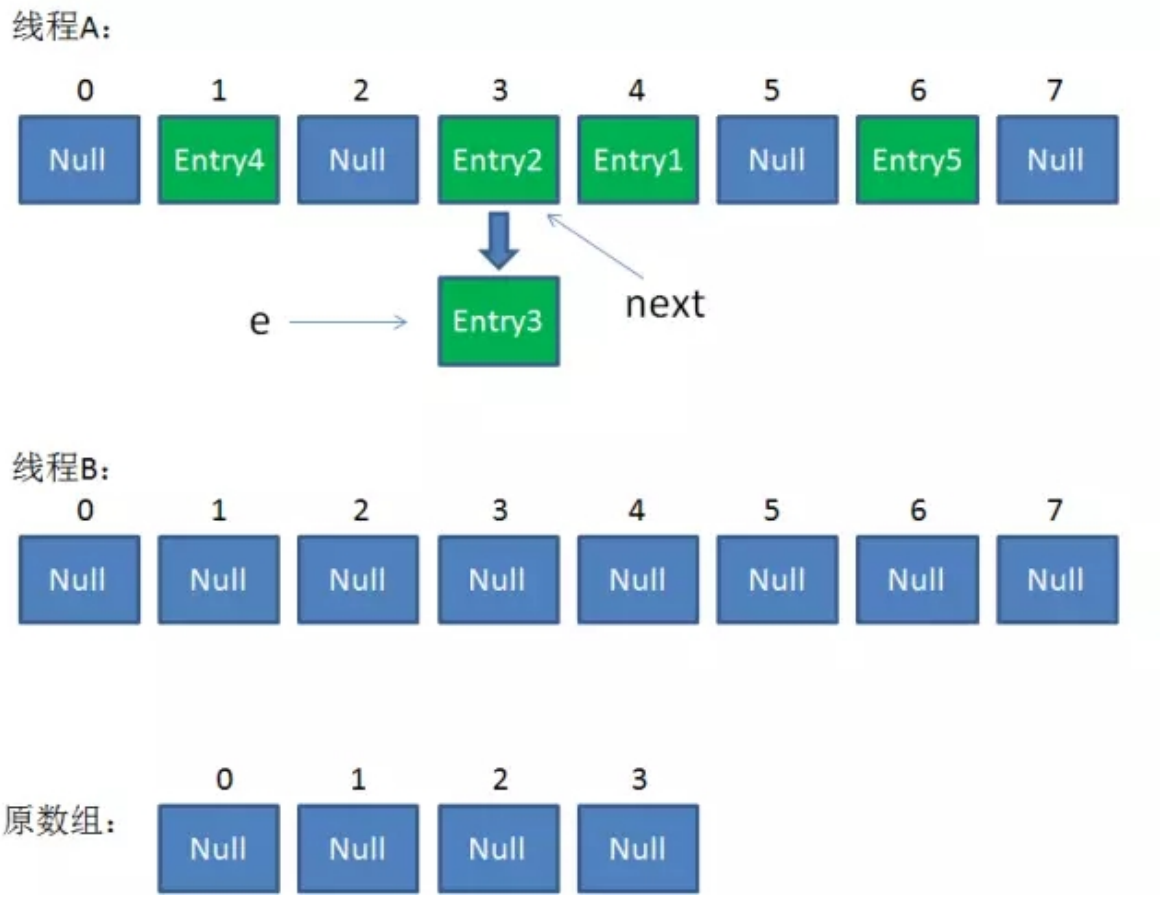
```
void transfer(Entry[] newTable, boolean rehash) {
    int newCapacity = newTable.length;
    for (Entry<K,V> e : table) {
        while(null != e) {
            Entry<K,V> next = e.next;
            if (rehash) {
                e.hash = null == e.key ? 0 : hash(e.key);
            }
            int i = indexFor(e.hash, newCapacity);
            e.next = newTable[i];
            newTable[i] = e;
            e = next;
        }
    }
}
```

假如此时线程B遍历到Entry3对象，刚执行完红框里的这行代码，线程就被挂起。对于线程B来说：

e = Entry3

next = Entry2

这时候线程A畅通无阻地进行着Rehash，当ReHash完成后，结果如下（图中的e和next，代表线程B的两个引用）：



直到这一步，看起来没什么毛病。接下来线程B恢复，继续执行属于它自己的ReHash。线程B刚才的状态是：

e = Entry3

next = Entry2

```
void transfer(Entry[] newTable, boolean rehash) {
    int newCapacity = newTable.length;
    for (Entry<K,V> e : table) {
        while(null != e) {
            Entry<K,V> next = e.next;
            if (rehash) {
                e.hash = null == e.key ? 0 : hash(e.key);
            }
            int i = indexFor(e.hash, newCapacity);
            e.next = newTable[i];
            newTable[i] = e;
            e = next;
        }
    }
}
```

当执行到上面这一行时，显然 i = 3，因为刚才线程A对于Entry3的hash结果也是3。

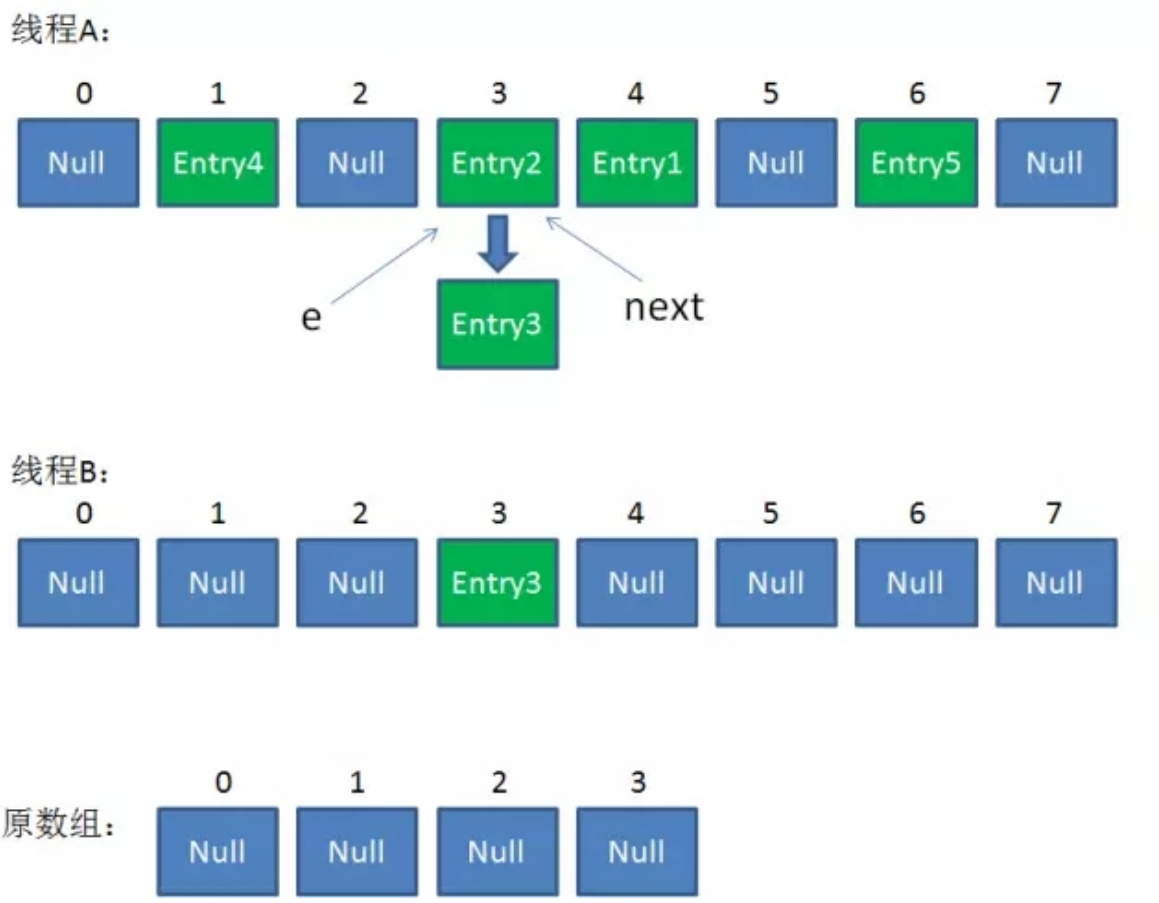
```
void transfer(Entry[] newTable, boolean rehash) {
    int newCapacity = newTable.length;
    for (Entry<K,V> e : table) {
        while(null != e) {
            Entry<K,V> next = e.next;
            if (rehash) {
                e.hash = null == e.key ? 0 : hash(e.key);
            }
            int i = indexFor(e.hash, newCapacity);
            e.next = newTable[i];
            newTable[i] = e;
            e = next;
        }
    }
}
```

我们继续执行到这两行，Entry3放入了线程B的数组下标为3的位置，并且e指向了Entry2。此时e和next的指向如下：

e = Entry2

next = Entry2

整体情况如图所示：



接着是新一轮循环，又执行到红框内的代码行：

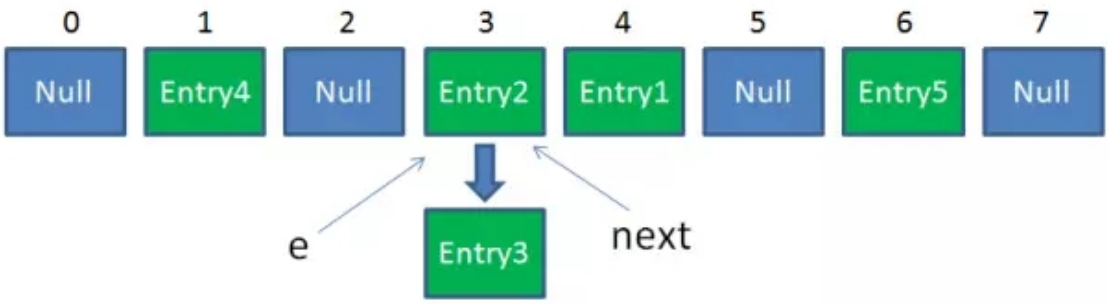
```
void transfer(Entry[] newTable, boolean rehash) {
    int newCapacity = newTable.length;
    for (Entry<K,V> e : table) {
        while(null != e) {
            Entry<K,V> next = e.next;
            if (rehash) {
                e.hash = null == e.key ? 0 : hash(e.key);
            }
            int i = indexFor(e.hash, newCapacity);
            e.next = newTable[i];
            newTable[i] = e;
            e = next;
        }
    }
}
```

e = Entry2

next = Entry3

整体情况如图所示：

线程A：



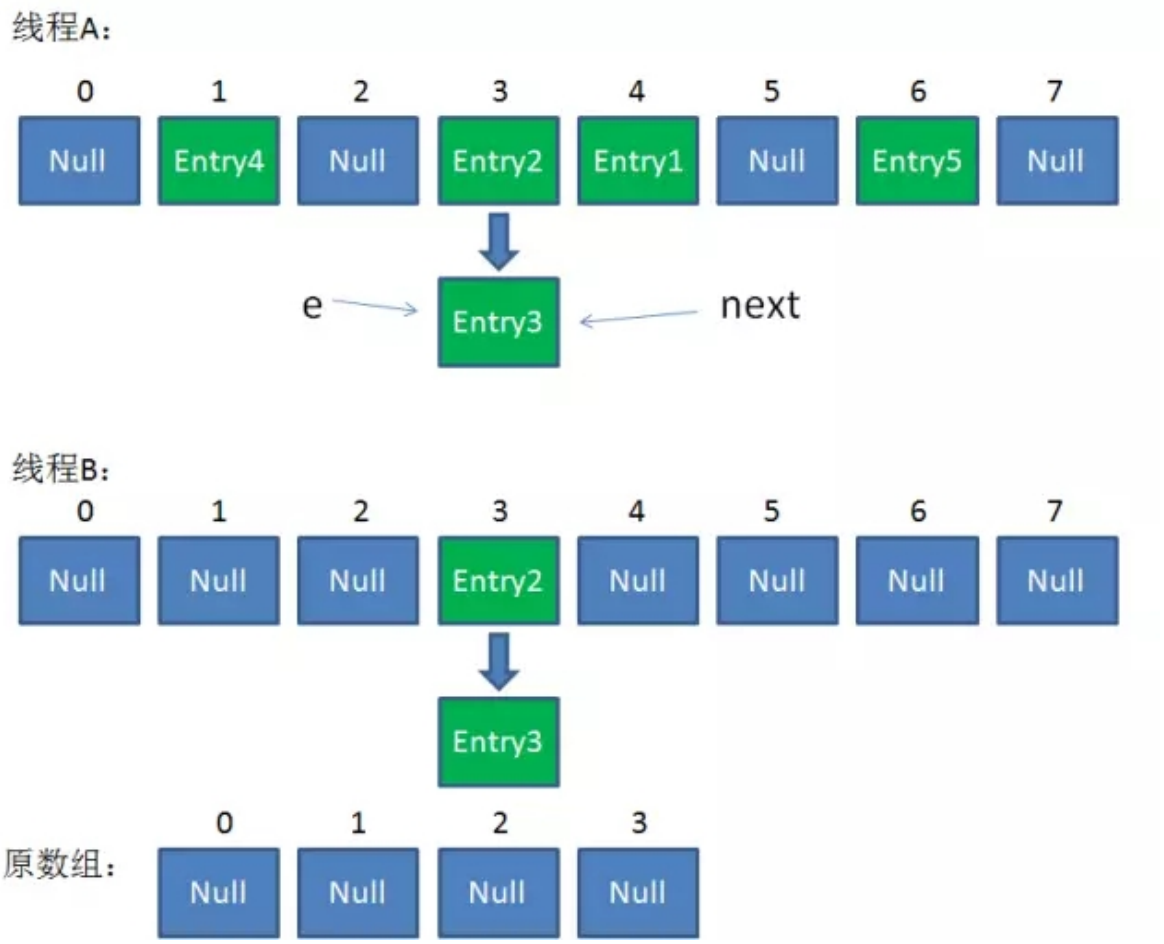
线程B：

原数组：

接下来执行下面的三行，用头插入法把Entry2插入到了线程B的数组的头结点：

```
void transfer(Entry[] newTable, boolean rehash) {
    int newCapacity = newTable.length;
    for (Entry<K,V> e : table) {
        while(null != e) {
            Entry<K,V> next = e.next;
            if (rehash) {
                e.hash = null == e.key ? 0 : hash(e.key);
            }
            int i = indexFor(e.hash, newCapacity);
            e.next = newTable[i];
            newTable[i] = e;
            e = next;
        }
    }
}
```

整体情况如图所示：



第三次循环开始，又执行到红框的代码：

```
void transfer(Entry[] newTable, boolean rehash) {
    int newCapacity = newTable.length;
    for (Entry<K,V> e : table) {
        while(null != e) {
            Entry<K,V> next = e.next;
            if (rehash) {
                e.hash = null == e.key ? 0 : hash(e.key);
            }
            int i = indexFor(e.hash, newCapacity);
            e.next = newTable[i];
            newTable[i] = e;
            e = next;
        }
    }
}
```

e = Entry3

next = Entry3.next = null

最后一步，当我们执行下面这一行的时候，见证奇迹的时刻来临了：

```
void transfer(Entry[] newTable, boolean rehash) {
    int newCapacity = newTable.length;
    for (Entry<K,V> e : table) {
        while(null != e) {
            Entry<K,V> next = e.next;
            if (rehash) {
                e.hash = null == e.key ? 0 : hash(e.key);
            }
            int i = indexFor(e.hash, newCapacity);
            e.next = newTable[i];
            newTable[i] = e;
            e = next;
        }
    }
}
```

newTable[i] = Entry2

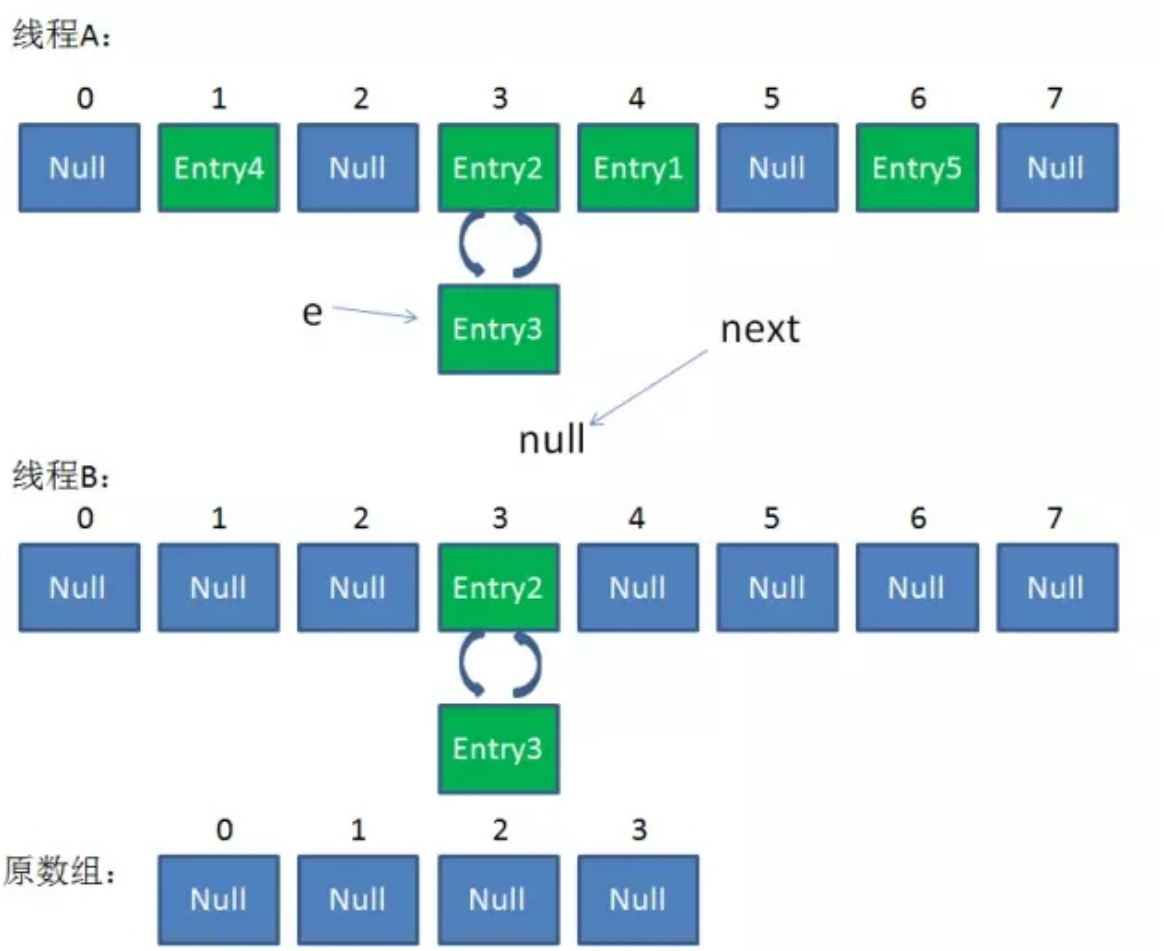
e = Entry3

Entry2.next = Entry3

Entry3.next = Entry2

链表出现了环形!

整体情况如图所示:



此时，问题还没有直接产生。当调用Get查找一个不存在的Key，而这个Key的Hash结果恰好等于3的时候，由于位置3带有环形链表，所以程序将会进入死循环！

解决方案：currenthashmap

6、RedLock的原理及存在的问题

RedLock（Redis Distributed Lock）

使用场景：多个服务间保证同一时刻同一时间段内同一用户只能有一个请求

RedLock算法思想，意思是不能只在一个redis实例上创建锁，应该是在多个redis实例上创建锁，n / 2 + 1，必须在大多数redis节点上都成功创建锁，才能算这个整体的RedLock加锁成功，避免说仅仅在一个redis实例上加锁而带来的问题

官网介绍：http://redis.cn/topics/distlock.html

存在的问题:

1、故障重启后带来的锁的安全性的问题

我们一共有 A、B、C 这三个节点。

1. 客户端 1 在 A，B 上加锁成功。C 上加锁失败。
2. 这时节点 B 崩溃重启了，但是由于持久化策略导致客户端 1 在 B 上的锁没有持久化下来。
3. 客户端 2 发起申请同一把锁的操作，在 B，C 上加锁成功。
4. 这个时候就又出现同一把锁，同时被客户端 1 和客户端 2 所持有了。

Redis 的作者又提出了延迟重启（delayed restarts）的概念

一个节点崩溃后，不要立即重启它，而是等待一定的时间后再重启。等待的时间应该大于锁的过期时间（TTL）。这样做的目的是保证这个节点在重启前所参与的锁都过期。相当于把以前的帐勾销之后才能参与后面的加锁操作。

神仙打架：

神仙一：Redis 的作者 antirez 。

神仙二：分布式领域专家 Martin Kleppmann

他认为 Redlock 是一个严重依赖系统时钟的分布式锁

他举了一个例子：

1. 客户端 1 从 Redis 节点 A, B, C 成功获取了锁。由于网络问题，无法访问 D 和 E。
2. 节点 C 上的时钟发生了向前跳跃，导致它上面维护的锁过期了。
3. 客户端 2 从 Redis 节点 C, D, E 成功获取了同一个资源的锁。由于网络问题，无法访问 A 和 B。
4. 现在，客户端 1 和客户端 2 都认为自己持有了锁。

这样的场景是可能出现的，因为 Redlock 严重依赖系统时钟，所以一旦系统的时间变得不准确了，那么该算法的安全性也就得不到保障了

长发哥认为，应该考虑类似 Zookeeper、etcd的方案，或者支持事务的数据库。

7、redis锁怎么保证在事务提交后执行，如果锁释放了，事务还没提交怎么办

先获得分布式锁再操作，操作成功后释放锁

```
//加锁并设置有效期
if(redis.lock("RDL",200)){
    //判断库存
    if (orderNum<getCount()){
    //加锁成功 ,可以下单
    order(5);
    //释放锁
    redis,unlock("RDL");
    }
}
```

8、及时聊天系统用发布订阅或者stream实现靠谱?

可以做

stream是Redis5.0后新增的数据结构，用于可持久化的消息队列。

几乎满足了消息队列具备的全部内容，包括:

- 消息ID的序列化生成
- 消息遍历
- 消息的阻塞和非阻塞读取
- 消息的分组消费
- 未完成消息的处理
- 消息队列监控

stream提供对消息历史记录的查询，所以也可以实现群聊

9、我在生成环境中有次被DDOS攻击，Redis积累了数据特别太多导致系统卡顿，但是明明都加了过期时间，不知道问题出在哪儿

1、过期策略的设置

2、Redis积累了数据特别太多在RDB时会fork时间过长，会阻塞主进程

3、DDOS是网络攻击，一般不会对数据产生影响

4、加一个prometheus对redis的监控，确定一下redis现在的状态

5、增加aof/rdb的执行周期或暂时关闭rdb/aof

10、Redis发布订阅和stream应用场景

发布订阅:

1、构建消息系统

2、公众号、博客等

stream应用：

原则上能够使用MQ的地方都可以使用

1班

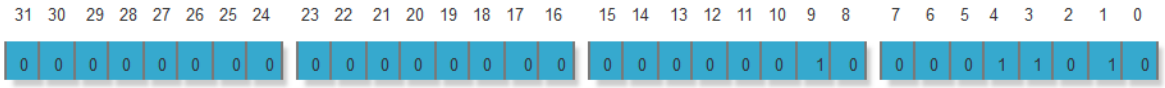
1、Mysql驱动commit方法是否同步，如果同步，客户端先commit再更新缓存，是否不会出现缓存脏数据问题 -- 缓存的读写模式

只要在高并发场景下就会出现脏读问题，因为本质为不同的数据源，所以只能通过延时双删让一致性的事件尽量的短。

2、redis的bitmap模式实现位图的原理是什么，为什么位数很大也可以存储

Bitmap 的基本原理就是用一个 bit 来标记某个元素对应的 Value，而 Key 即是该元素。由于采用一个 bit 来存储一个数据，因此可以大大的节省空间。

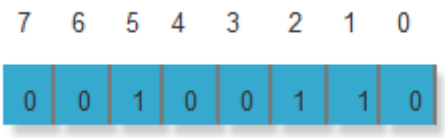
Java 中 int 类型占用 4 个字节，即 4 byte，又 1 byte = 8 bit，所以 一个 int 数字的表示大概如下，



试想以下，如果有一个很大的 int 数组，如 10000000，数组中每一个数值都要占用 4 个字节，则一共需要占用 10000000 * 4 = 40000000 个字节，即 40000000 / 1024.0 / 1024.0 = 38 M

如果使用 bit 来存放上述 10000000 个元素，只需要 10000000 个 bit 即可， 10000000 / 8.0 / 1024.0 / 1024.0 = 1.19 M 左右，可以看到 bitmap 可以大大的节约内存。

使用 bit 来表示数组 [1, 2, 5] 如下所示，可以看到只用 1 字节即可表示：



3、目前在互联网公司中哨兵和Redis集群哪个用的比较多?

都有使用，RedisCluster在5.0以后趋近于完美，很多公司使用的是Redis3的版本

所以还是用哨兵多些，对于新开发的产品选择RedisCluster的比较多

4、rediscluster迁移过程中，如果moved到a节点，a节点正在迁移，但是get的数据尚未迁移完成，是否会在a节点直接返回get的数据

是的，当一个槽被设置为 MIGRATING 状态时， 原来持有这个槽的节点仍然会继续接受关于这个槽的命令请求， 但只有命令所处理的键仍然存在于节点时， 节点才会处理这个命令请求。如果命令所使用的键不存在与该节点， 那么节点将向客户端返回一个 -ASK 转向（redirection）错误， 告知客户端， 要将命令请求发送到槽的迁移目标节点。

5、rediscluster的副本漂移功能是自动生效的么，是否需要进行配置

不需要，Redis通过周期性调度函数ClusterCorn来完成副本漂移

6、自动化缓存预热怎么做?

- 数据量不大的情况下系统启动时加载
- 定时刷新
- 手动刷新

7、redis客户端分区，如果使用一致性hash环，添加节点后，数据迁移怎么做

redis客户端分区是Redis客户端自行实现分片算法的，所以需要手动迁移数据

可以直接拷贝rdb文件，这样数据会有冗余

可以使用move命令写脚本或数据迁移工具（redis-dump）

比较麻烦，所以用一致性hash环会初始化尽量多的节点。