# Redis集群的主从复制模式

在实际应用中，使用单台Redis服务器可能会出现服务器故障停机、容量瓶颈、QPS（每秒查询）瓶颈等问题，进而影响系统及网站的正常服务，从而造成必须要的经济损失，为了避免使用单个服务器出现的问题，进入了Redsi集群的主从复制模式。

## 主从复制模式简介

### 什么是Redis主从复制

1. 在Redis中，通过执行slaveof命令，或者通过配置文件设置slaveof参数，就可以让一台服务器去复制另外一台服务器，其中，被复制的服务器叫作主服务器（master），而对主服务器进行复制的服务器叫作从服务器（slave），从而实现当主服务器中的数据更新后，根据配置和策略自动同步到从服务器上。其中master以写为主，slave以读为主。
2. 可以使用info命令查看replcation部分来查看当前服务器的主从复制信息。命令如下：

**info replication**

1. 主从复制的作用如下：

·为一个数据提供多个副本，使得高可用，分布式成为可能；

·扩展Redis的读性能，可以实现读写分离；

### 关于主从服务复制功能的说明

1. 一个master（主服务器）可以由多个slave（从服务器），但一个slave（从服务器）只能有一个master（主服务器）；
2. 主服务器可以有从服务器之外，从服务器也可以有自己的从服务器，多个从服务器之间可以构成一个网状结构，它们之间具有传递关系；
3. 自Redis2.8版本以后，Redis采用异步复制功能，从服务器会以每秒一次的频率向主服务器报告复制流的处理速度；
4. 在进行复制的时候，复制功能不会阻塞主服务器，即使有多个从服务器正在进行初次同步时，主服务器也可以继续执行来自客户端的命令请求；同理，复制功能也不会阻塞从服务器，只要在配置文件中进行了配置，即使从服务器正在运行初次同步，主服务器也可以使用旧版本的数据来处理命令请求。当从服务器删除旧版本数据集并加载新版本数据集时，连接请求会被阻塞，直到加载完毕；
5. 如果一个主节点B成为另一个主节点A的从节点，那么主节点B之前保存的数据将会被清除，去同步主节点A的数据，与主节点A的数据保持一致；

## 实现主从复制

Redis要想实现主从复制功能，可以通过命令或者修改配置文件来实现。

### slaveof

在一台Redis服务器下开启一个客户端，执行slaveof命令来复制主服务器，使得该客户端所处Redis服务器成为从服务器。命令如下：

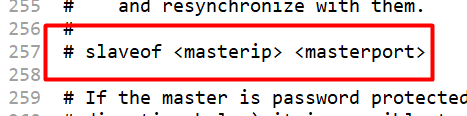
**slaveof ip port**

ip表示要复制的主服务器的ip地址；port表示主服务器端口号；

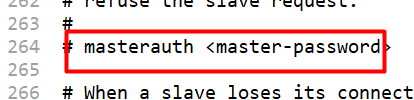
1. 在一般情况下，主、从服务器不建议在一台服务器上，因为这样做没有任何意义。
2. 如果从服务器不希望再成为主服务器的一个从节点，而是想变回主节点，则可以使用slaveof no one命令取消复制；取消复制后，从服务器变为主服务器，但之前的数据并不会丢失，只是在取消复制后，数据将不再同步。

### 修改配置文件

1. slaveof：设置主服务器的IP地址与端口号。如下：

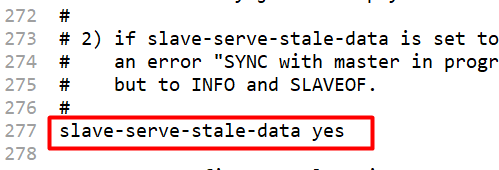


1. masterauth：设置连接主服务器时的密码；如果主服务器设置了密码，那么从服务器需要通过该参数来设置密码，密码与主服务器的requirepass对应。如下：

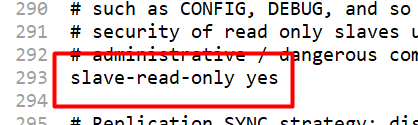


1. slave-server-stale-data：当lsave与master断开连接或者正在进行复制时，当值为yes时，表示服务器会继续处理来自客户端的命令请求；当值为no时，表示服务器对除了info命令和slaveof命令以外的任何命令请求都会返回一个错误。默认值为yes。

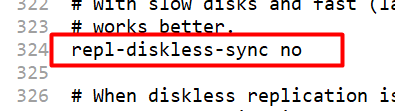
如下：



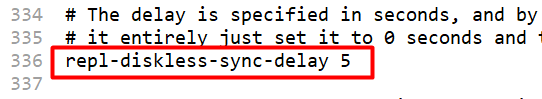
1. slave-read-only：slave（从服务器）是否为只读，默认为yes表示是；不建议设置为no。如下：



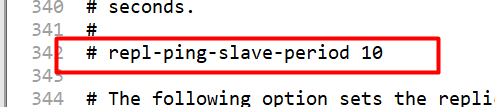
1. repl-diskless-sync：表示是否使用socket方式复制数据。Redis提供了两种复制方式：disk和socket，默认为no，即使用diskf方式复制数据。如下：



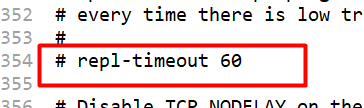
1. repl-diskless-delay：复制的延迟时间，默认为5s；不建议设置为0，因为复制一旦开始，节点不会再接收新slave的复制请求，直到下一个RDB传输。如下：



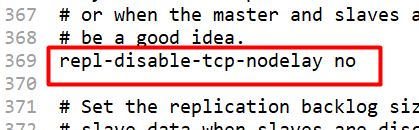
1. repl-ping-slave-period：指定ping请求的发送间隔。slave根据指定的时间间隔向master发起ping请求。默认为10s。如下：



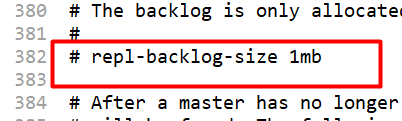
1. repl-timeout：复制连接的超时时间，默认为60s。master和slave都有超时时间的设置，如果master检测到slave上次发送的时间超过repl-timeout所设置的值，就会认为slave已经处于离线状态，并清除该slave的信息。相反，如果slave检测到上次和master交互的时间超过了repl-timeout所设置的值，就会认为master离线。但需要注意的是：repl-timeout设置的值一定要比repl-ping-slave-period所设置的值大，否则会常常出现超时。如下：



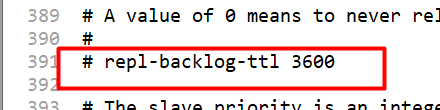
1. repl-disable-tcp-nodelay：是否禁止复制TCP链接的tcp nodelay参数，默认为no，表示允许使用tcp nodelay；如果master将该参数值设置为yes，以此来禁止使用tcp nodelay，那么在把数据复制给slave时，会减少包的数量和网络带宽，同时也可能带来数据的延迟。当传输量比较大的时候，推荐设置为yes。如下：



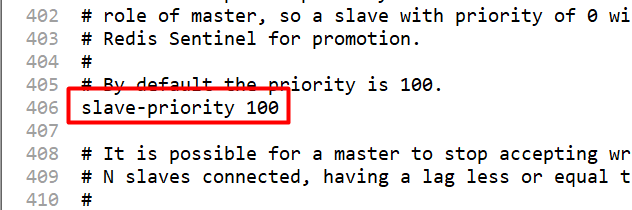
1. repl-backlog-size：用于设置复制缓冲区的大小，默认为1MB。如下：



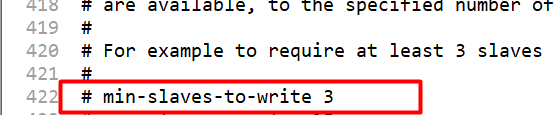
1. repl-backlog-ttl：设置一段时间，这段时间内，master和slave断开连接后会释放复制缓冲区的内存。默认为3600s。如下：



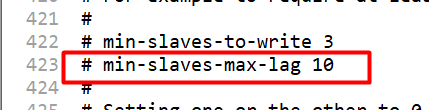
1. slave-priority：设置slave的优先级，默认为100。当master出现故障（宕机）不能使用时，会根据lsave的优先级选举一个新的master。参数值越小，就越可能被选称为master。参数值为0时，表示此slave永远都不会成为为master。



1. min-slaves-to-write：Redis提供了可以让master停止写入的方式。如果配置了min-slaves-to-write参数，当健康的lsave小于该参数值时，master就会禁止写入。该参数值默认为3，表示master最少要有3个健康的slave存活才能执行写命令。如下：



1. min-slaves-max-lag：延迟时间小于min-slaves-max-lag秒的slave将被认为是健康的slave，默认为10s，表示延迟小于10秒的slave才是健康的slave。如下：



当修改了redis.conf文件，记得重启Redis服务器使配置生效。

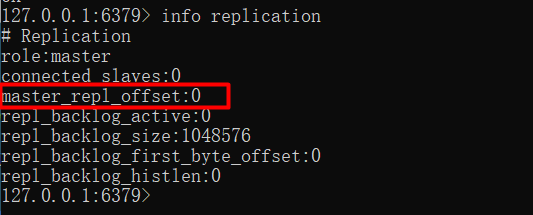
## 复制的原理与流程

### 原理

1. **全量同步与部分同步**
2. Redis主从服务器间的数据同步分为全量同步（也称全量复制）和部分同步（也称部分复制）。
3. 全量同步用于处理第一次复制的情况，通过让主服务器创建RDB文件并发送给从服务器，以及向从服务器发送保存在缓冲区中的写命令来进行数据的同步。
4. 部分同步用于处理从服务器离线后重新连接的复制情况，当从服务器在离线后重新连接上主服务器时，如果条件允许，那么主服务器可以将在从服务器断开连接期间执行的最新命令发送给从服务器，从服务器接收到这些命令并执行它们，从而达到主从服务器数据一致性。

**2、复制偏移量**

1. 在执行复制操作的时候，主从服务器都有一个复制偏移量，当主服务器每次向从服务器发送N字节的数据时，就会将自己复制偏移量的值加上N。而当从服务器每次接收到主服务器发送过来的N字节数据时，就会将自己的复制偏移量加上N。
2. 我们可以执行info命令查看replication部分来查看当前Redis服务器的复制偏移量。如下：



1. 通过查看主从服务器的复制偏移量，可以确认主从服务器的数据是否处于一致状态。如果主从服务器的复制偏移量保持一致，则说明主从服务器两者的数据一致；如果主从服务器的复制偏移量不相等，则说明主从服务器的数据不一致。

**3、运行ID**

1. 每台Redis服务器都有自己的运行ID，运行ID在服务器启动时会自动生成，运行ID由40个十六进制字符随机组成，如1dbe7f0c16dcd87c7596351c1afbfa3ab1dc5eca。
2. 当从服务器对主服务器进行初次复制同步时，主服务器会将自己的运行ID发送给从服务器，从服务器接收到主服务器的运行ID后，会将它保存起来。如果从服务器离线后重新连接上了主服务器，从服务器将向主服务器发送之前保存的运行ID，进行身份验证。

·如果保存的ID与当前所连接主服务器的运行ID相同，则执行部分同步。

·如果保存的ID与当前所连接主服务器的运行ID不同，则执行全量同步。

**4、复制缓冲区**

1. 复制缓冲区是由主服务器维护的一个固定长度、先进先出的队列。在默认情况下，复制缓冲区的大小为1MB。
2. 主服务器在执行命令后，会将写命令发送给所有与它连接的从服务器，同时将这些命令入队到复制缓冲区中。从服务器离线后重新连接上主服务器时，主服务器会根据从服务器的复制偏移量来决定对从服务器执行全量同步还是部分同步操作。

·如果从服务器的复制偏移量之后的数据存在于复制缓冲区中，那么主服务器会对这台从服务器执行部分同步。

·如果从服务器的复制偏移量之后的数据再复制缓冲区中不存在，那么主服务器会对这台从服务器执行全量同步。

### 过程

复制功能的实现步骤如下：

1. 通过slaveof命令设置主服务器的IP地址和端口号；
2. 执行slaveof命令之后，从服务器会根据主服务器的IP地址和端口号建立套接字连接；
3. 如果套接字连接建立成功，从服务器会向主服务器发送ping命令，用于检查套接字的读写状态是否正常，以及检查主服务器是否可以正常处理命令请求；
4. 如果主服务器和从服务器连接正常，则会返回pong，接下来将根据从服务器的配置信息来决定是否进行身份验证，即检查masterauth属性值与主服务器的requirepass是否一致；
5. 如果身份验证成功，从服务器会想主服务器发送端口信息并执行同步操作；
6. 完成同步功能之后，主从服务器进行命令传播阶段，主服务器将自己执行的写命令及时发送给从服务器，从服务器接收来自主服务器的写命令并执行，这样就保证了主从服务器的数据一致性。

## 读写分离与心跳机制

### Redis的读写分离

在Redis的主从复制模式中，实现了读写分离，从而将读流量分摊到每个从节点。读写分离是一个很好的策略，它将master上的数据同步给slave，让slave分摊去执行相关的业务，比如，一个业务只需要读数据，那么我们直接去读从节点就可以了。在通常情况下，master负责写操作，slave负责读操作，这样一来可以减轻master的压力，提高了性能；另一方面扩展了Redis读的能力，适用于读多写少的业务场景。

但Redis的读写分离也存在一定问题：

1. 复制数据延迟

在大多数情况下，我们在同步master上的数据时，会做一个异步执行的同步数据操作，将数据复制给slave，这期间就会有一个顶的时间差。而且当slave发送阻塞时，它会延迟接收主服务器发送过来的写命令，就有可能发生读写不一致的情况。如果担心发送读写不一致的情况，则可以对它的复制偏移量进行监控。

1. 读到一些过期的数据

Redis使用两种策略来判断某个key是否过期，进而删除这个key。一是使用惰性策略，只有当去操作某个key时，惰性策略才会去判断该key是否过期，如过期，则删除，返回空给客户端；另一种策略时定期策略，定期检查有没有过期的键，如果过期则删除，但当定期策略在采样过期的key时，如果存在大量的过期key，导致采样速度慢于过期的key的产生速度，就会造成许多过期的key没有被及时删除。另外，Redis中，master和slave之间有一个约定，也就是slave不能删除数据，这就会造成slave节点在读取master时可能读到脏数据。

1. 在读写分离的过程中，发送从节点故障（宕机）。

### Redis的心跳机制

在主从服务器之间进行命令传播的时候，从服务器默认会以每秒一次的频率发送replconf ack命令到服务器，该命令有两个作用：

1. 检测主从服务器之间的网络连接情况

从服务器通过该向主服务器发送replconf ack命令来检测主从服务器之间的网络连接情况，判断主从服务器之间的网络连接有没有出现问题。

1. 检测在命令传播过程中是否命令丢失

在主从服务器进行命令传播的过程中，可能会因为网络的故障而导致传输的写命令在途中丢失，这样就会导致主从服务器的复制偏移量不一致。当从服务器向主服务器发送replconf ack命令时，如果主服务器发现从服务器的复制偏移量与自己的偏移量不一致，就会根据从服务器的复制偏移量，在复制缓冲区找到发送过程中所丢失的写命令，然后补发给从服务器，实现同步功能。

## 主从复制模式的常见问题

1. 有一个一主二从的服务器集群，在某一时刻，3个客户端都执行某一条写命令，这3个客户端会产生什么效果？

只有主服务器可以执行成功，两台从服务器将会报错，提示不能执行写命令，这也可以体现出主从复制的读写分离。

1. 有一个一主二从的服务器集群，当master宕机之后，另外两个slave会原地待命还是夺权篡位？

如果master宕机，另外两个slave会原地待命，不会夺权篡位，它们与master的连接状态变为master\_link\_status：down

1. 如果master复活了，它还是master吗？还是会因为它宕机了一次，主从复制会被打乱？

如果master复活了，那么它依然是master，不会因为一次宕机而打乱主从复制结构。

1. 如果某个slave宕机了，那么在它复活之后，身份还是slave吗？数据能够恢复回来？

如果一个slave宕机了，在它复活之后，身份将不再是slave，而是一个新的master，它不能恢复回来，如果想要恢复回来，可以执行slaveof命令将其再设置为slave。

# Redis集群的高可用哨兵模式

## 为**什么要引入高可用哨兵模式**

通过学习主从复制模式，可以得知主从复制有两个主要的功能：一是可以为主服务器提供备份，当主服务器发送故障时，在这个备份中会有一份完整的数据；二是可以对主服务器实现分流，比如实现读写分离，将大部分的写操作放到主节点上，将大部分的读操作放到从节点上，以此来减轻主节点的压力，提升性能。

但是主从复制模式存在一个问题：如果主节点出现故障，需要手动处理故障，为了解决主从复制故障转移的问题，引入了Redis集群的高可用哨兵模式。

## 什么是高可用哨兵模式

1. 高可用哨兵模式是由一个或多个哨兵组成的哨兵系统，主要用于监控任意多台主服务器是否发生故障，以及监控这些主服务器的从服务器。当主服务器发生故障时，它会通过投票选举的方式从主服务器下属的从服务器中选举出一台新的主服务器，让这台新的主服务器替代之前的主服务器继续处理命令请求以及完成相关工作，从而实现了故障的自动转移，而无需手动操作，达到了高可用、热部署的目的。
2. 哨兵模式不会存储数据，它的作用是对Redis主从复制的节点进行监控，对其故障进行判断，进行故障转移的处理，以及通知相关的客户端。
3. Redis Sentinel（哨兵）可以监控多套主从复制模式，这样做的目的是节省资源。

## 哨兵的作用

1. 监控：哨兵可以管理多台Redis服务器，不断的检查其监控的主从服务器，判断其是否发生故障；
2. 通知：当被监控的某台Redis服务器发送故障时，哨兵会向管理员或者相关应用程序发送通知；
3. 自动故障转移：当一台服务器出现故障，无法正常工作时，哨兵模式会开始一次自动故障转移，将出现故障的主服务器下属的某台从服务器升级为新的主服务器，并让出现故障的主服务器下属的其他服务器改为复制新的主服务器地址，使得集群可以使用新的主服务代替出现故障的主服务器，这样就完成了故障的转移操作。

## 配置哨兵模式

关于配置哨兵模式的教程请参考学习书。

# Redis集群搭建

## 认识Redis集群

Redis集群（Redis Cluster）是一个分布式、容错的Redis实现，它由多个Redis节点组成，在多个Redis节点之间进行数据共享。集群可以使用的功能是普通单机Redis所能使用的功能的一个子集，它提供了复制和故障转移功能。

Redis集群中不存在中心节点或代理节点，而且不支持那些需要同时处理多个键的Redis命令，因为执行这些命令，需要在多个Redis节点之间移动数据，在高负载的情况下，执行这些命令会降低Redis集群的性能，并出现不可预料的问题。

Redis集群的设计目的是达到线性可扩展性。同时，Redis集群为了保证数据的一致性，而牺牲了一部分容错性：Redis系统会在保证对网络断线的节点失效具有有限抵抗力的前提下，尽可能地保证数据的一致性。Redis集群的容错功能是通过主从复制失效的，主节点宕机之后，从节点可以代替它完成相关工作。

Redis集群通过分区来提供一定程度的可用性，当集群中有部分节点失效或者无法提供服务时，集群也可以继续完成相关的命令请求。

Redis集群实现了单机Redis中所有处理单个数据库键的命令，不支持那些需要同时处理多个键的Redis命令。并且，Redis集群也不支持多数据库功能，它默认使用0号数据库，并且不能使用SELECT命令。

## Redis集群的好处

使用Redis集群的好处有：

1. 可以实现将数据自动切分到多个节点；
2. 当集群中有部分节点失效或者无法提供服务器的时候，它仍然可以继续完成相关的命令请求。
3. Redis集群的使用可以解决高并发、大数据量的问题。

## 集群中的节点

一个Redis集群通常由多个节点（Node）组成，一个节点都是一台Redis服务器。在没有搭建Redis集群之前，每个节点都是相互独立的，都分别表示着一台单机Redis，彼此之间没有任何联系。只有将多个独立的节点连接在一起，才能组建一个可以工作的集群。通常使用CLUSTER MEET命令来连接各个独立的节点。命令格式如下：

CLUSTER MEET <ip> <port>

通过执行该命令可以让该节点与ip和port所指定的节点进行握手，当握手成功时，就表示该节点已经成功加入集群中。在集群模式下，每个节点都是一台Redis服务器。在启动服务器的时候，会通过读取配置文件中cluster-enabled选项的值是否为yes来决定是否开启服务器的集群模式。当cluster-enabled选项的值为yes时，表示开启服务器集群模式成为集群中的一个节点。当cluster-enabled选项的值为no时，表示开启服务器的单机模式，也就是开启一台普通的Redis服务器。进入集群模式的节点会继续使用所在单机模式中所使用的服务器组件，也就是说，集群中每个节点所能使用的服务器组件不管是在集群模式下还是在单机模式中都是一样的，不受影响。节点所能使用的服务器组件有文件事件处理器、时间事件处理器，以及每个接地那都可以使用数据库来保存键值对数据、使用复制功能完成数据复制、使用RED和AOF持久化来完成数据的恢复、使用订阅发布功能、使用Lua脚本环境执行Lua脚本等，同时每一个节点也依然会继续使用redisServer和redisClient结构体来分别保存服务器的状态和客户端的状态。

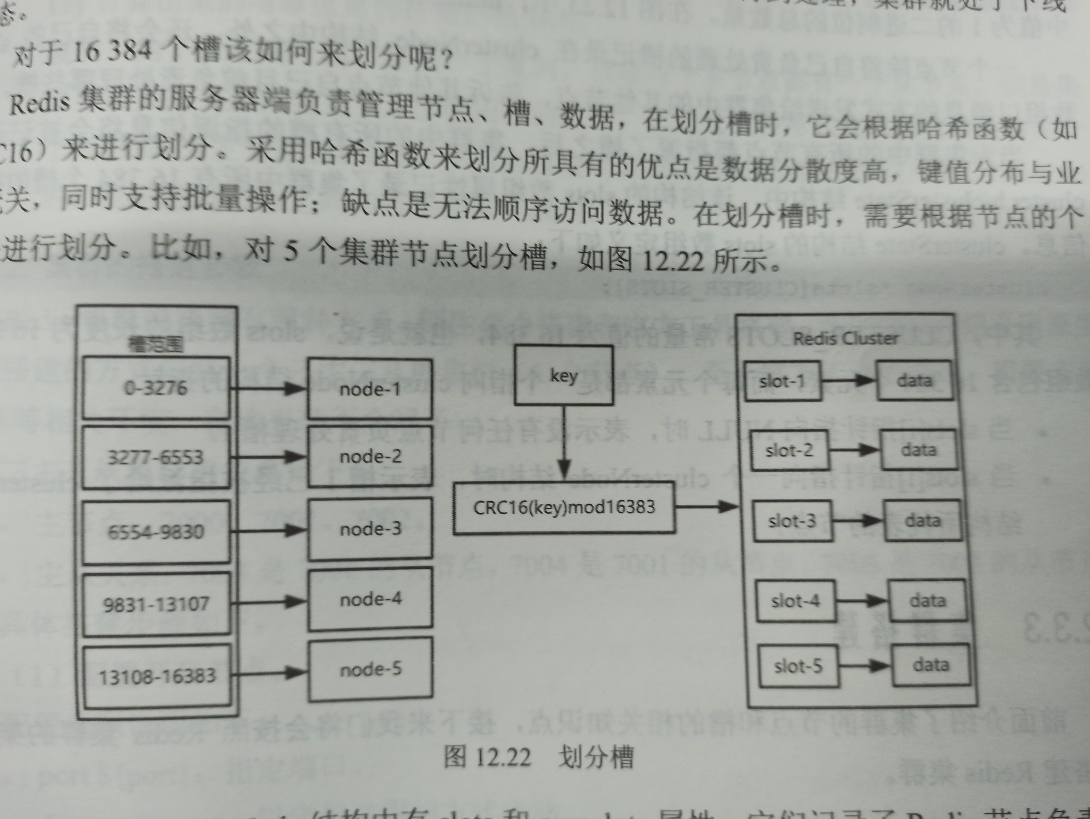
开启集群模式时，服务器会创建一个clusterState类型的结构体来保存当前节点视角下的集群状态，（clusterState结构体源码请参考教材书）；在集群模式中，每个节点都会使用cluster.h/clusterNode结构体来保存自己的当前状态，如创建时间、名字、IP地址及端口号等信息（cluster.h/clusterNode结构体源码请参考教材书）；同时也会为集群中的其他节点创建一个clusterNode结构来保存其他节点的状态信息（clusterNode结构体源码请参考教材书）；在clusterNode结构体中包含了一个link属性的结构clusterLink，该结构体中保存了其他连接节点的相关信息，如节点的创建时间、缓冲区信息等（clusterLink结构体源码请参考教材书）。

## 集群中的槽

Redis集群为了能够存储大量的数据信息，采用分区的方式将大量数据保存在数据库中，这个数据库被划分为16384个槽（Slot ）。这里所说的槽是虚拟槽，可以把这个槽理解为一个数字，槽是有一定范围的，在Redis中的范围时0~16383,。每个槽映射一个大数据子集，一般比节点数大。比如，有10万个数据，16284个槽，按照一定的哈希规则，对每个数字做一个哈希，然后对16363进行取余，如果这个数字在某个槽的范围内，就证明这个数字就是这个槽要管理的数据。

在集群的数据库中，每个键都存储在16384个槽的其中一个槽中，集群汇总的每个节点都可以处理0个或者最多16384个槽。当数据库中的16383个槽中都有节点在处理时，集群处于上线状态；而如果数据库中有任何一个槽没有得到处理，集群就处于下线状态。

对于16384个槽如何来划分呢？Redis集群的服务器负责管理节点、槽、数据，在划分槽时，它会根据哈希函数（如CRC16）来进行划分。采用哈希函数来划分所具有的优点是数据库分散度高，键值分布与业务无关，同时支持批量操作；缺点是无法顺序访问数据。在划分槽时，需要根据节点的个数来进行划分。比如，对5个集群节点划分槽，如下图所示：

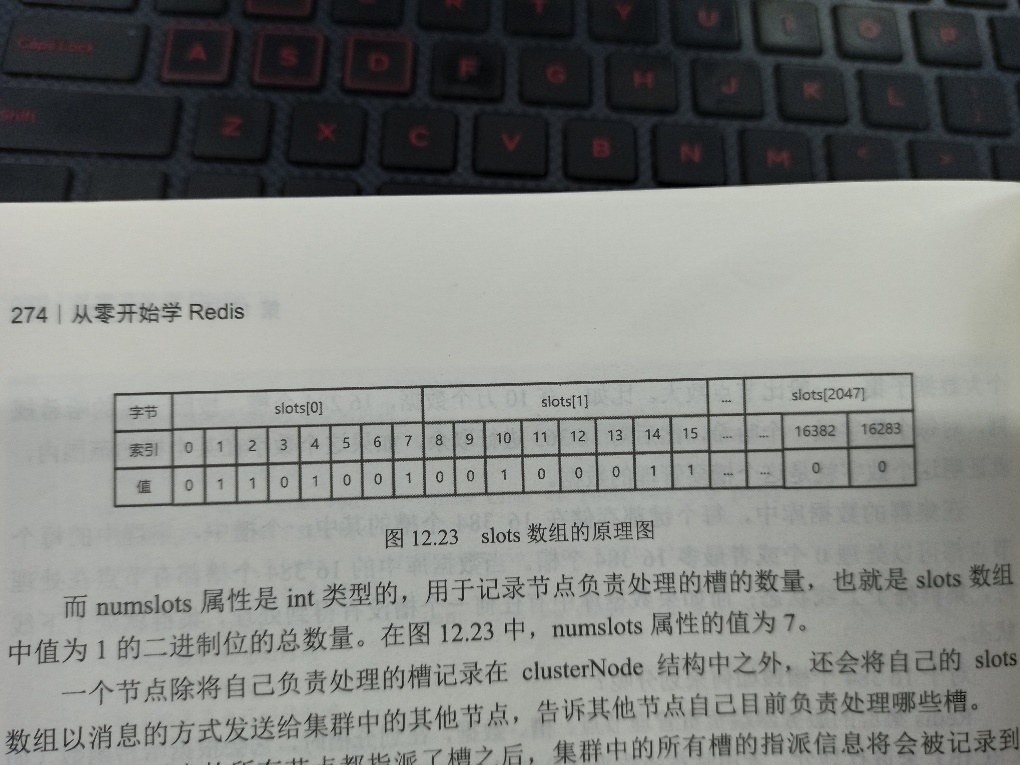


在cluster.h/clusterNode结构体中有slots和munslots属性，它们记录了Redis节点负责创建哪些槽，其中,slots属性是一个无符号二进制数组。这个slots数组的下标从0开始，到16383结束。对slots数组中的16384个二进制位进行编号，Redis会根据这个slots数组的索引i来判断节点是否负责处理槽i。

·当slots数组在索引i上的二进制位的值为1时，表示节点负责槽i

·当slots数组在索引i上的二进制位的值为0时，表示节点不负责处理槽i

假设节点负责处理索引为1、2、4、7，就表示slots数字索引为1、2、4、7的槽对应的二进制位的值为1。原理图如下：



而numslots属性是int类型的，用于记录节点负责处理的槽的数量，也就是slots数组中值为1的二进制位的总数量。

一个节点除将自己负责处理的槽记录在clusterNode结构体中之外，还会将自己的slots数组以消息的方式发送给集群的其他节点，告诉其他节点自己负责处理哪些槽。当为集群中所有节点都指派了槽之后，集群中的所有槽的指派信息将会被记录到cluster.h/clusterState结构体中，该结构体的slots数组属性记录了集群中所有16384个槽的指派信息。也就是说，slots数组的长度为16384数组包含16384个元素，而每一个元素都是一个指向clusterNode结构体的指针。

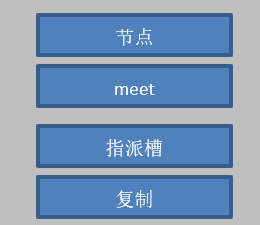
·当slots[i]指针指向NULL时，表示没有任何节点负责处理槽i

·当slots[i]指针指向一个custerNode结构时，表示槽i已经被指派给了clusterNode

结构体所代表的节点

## 集群架构说明与搭建思路

在搭建Redis集群前，先认识一下集群架构，按照Redis集群的架构图来搭建Redis集群。Redis Cluster架构图如下所示：



·节点：Redis集群中有一堆节点，节点时相互通信的，每个节点都负责读和写数据。

·meet：mee操作就是完成节点相互通信的过程。

·指派槽：只要给节点指派了对应的槽，节点才可以进行正常的读/写。当你启动了一

个节点，并为它指定了cluster模式后，它不会进行正常的读/写，还需要为

它指派槽。当有数据访问的时候，它会去查看自己的槽有没有对应的信息，

也就是传递过来的key计算出来的哈希值是否在槽的范围内。

·复制：为了保证高可用，需要一个复制，就是每个主节点都一个从节点。但是集群有

很多主节点，当主节点出现问题的时候，它通过某种形式也可以实现主备的一个高可用。当主节点宕机之后，从节点就会代替它，它内部的监控没有依赖于Sentinel的，而是通过节点之间相互监控来完成的。

## 集群搭建步骤

在搭建集群之前，先参考上一节的集群架构说明。Redis集群的搭建有两种方式：原生命令搭建和官方工具搭建。在这里只介绍采用原生命令搭建的方式安装一个三主三从的集群（6个节点）。至于官方工具的搭建，需要安装Ruby等相关依赖，在这里就不介绍了。三主三从集群的关系如下：

·主节点：7000、7001、7002

·从节点：7003是7000的从节点，7004是7001的从节点，7005是7002的

从节点。

1. 配置开启节点

配置选项说明如下：

·port ${port}：指定端口

·daemonize yes：以守护进程的方式启动

·dir “/home/redis/data”：数据目录

·dbfilename “dump-${port}.rdb”：指定RDB文件

·logfile “redis-cluster-${port}.log”：指定日志文件

·cluster-enabled yes：开启集群模式，表示该节点是一个clsuter节点

·cluster-config-file nodes-${port}.conf：为cluster节点指定配置文件

cluster节点主要配置说明如下：

·cluster-enabled yes：开启集群模式，表示该节点是一个cluster节点

·cluster-node-timeout 15000：表示故障转移的时间或节点超时的时间，15秒

·cluster-config-file “nides.conf”：集群节点的配置

·cluster-require-full-coverage yes：是否需要集群的所有节点都提供服务，才

会认为这个集群是正常运行的。加入集群中

有一个节点宕机了，它就不对外提供服务了。

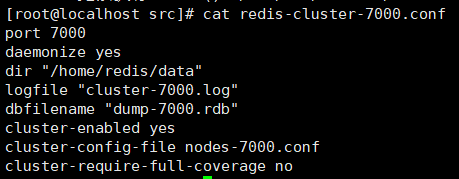
这个配置默认是yes，在实际生产过程中，

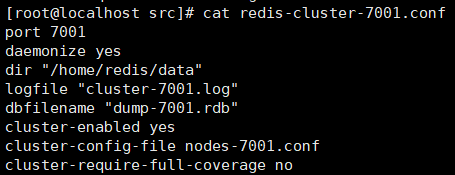
这个配置是不合理的，因为集群的一个节点

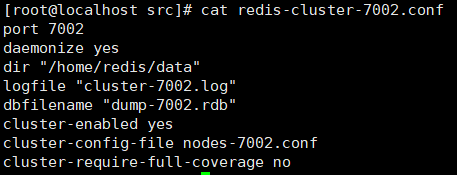
宕机而停止所有的服务，这样做是不可取的，

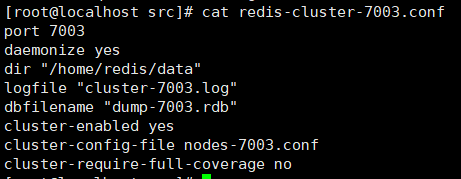
实际业务也不允许，所以建议设置为no。

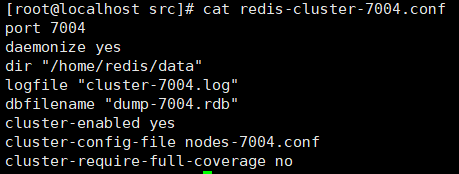
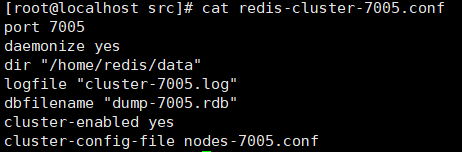
1. 在redis安装目录下的src目录下准备6个配置文件（7000，7001，7002，7003，7004，7005），文件名为redis-cluster-${post}.conf



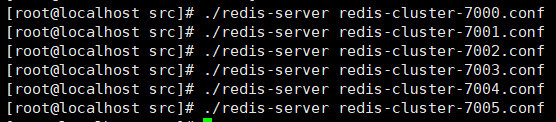


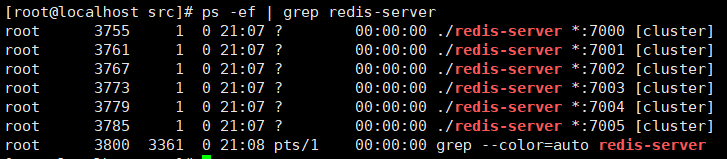






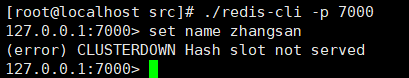
1. 配置完成后，开启启动节点，命令如下：



 然后使用ps -ef | grep redis-server查看进程验证节点是否开启成功：

可以看出，节点全部启动成功。

1. 使用redis-cli -p 7000连接7000节点（任意节点都可），并执行一条set命令，它会返回错误信息，如下：

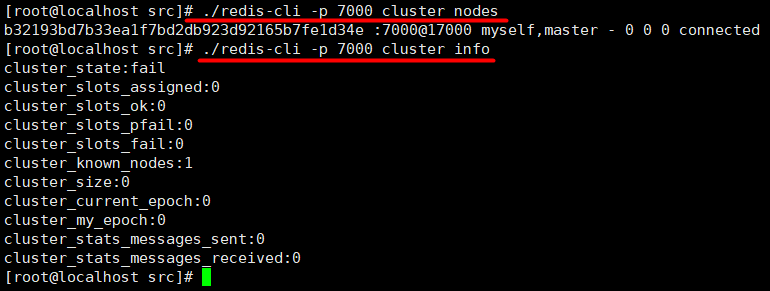


错误提示信息中CLUSTERDOWN说明集群处于下线状态，集群不可用。前面我们提到过，在集群模式下，只有当为每个节点都指派了槽，而且对16384个槽都进行指派时，这个集群节点才可用，才能对外提供服务。所以接下来的工作就是为集群节点执行meet操作及指派槽。我们退出redis客户单，使用命令查看一下7000节点的配置文件，如下：



从中可以看出7000节点时一个孤立的节点，它的node-id是b32193bd7b33ea1f7bd2db923d92165b7fe1d34e，myself表示它自己，角色是master，后面的相关信息是它的连接信息以及配置纪元等。

使用命令redis-cli -p 7000 cluster nodes查看集群节点信息，使用命令redis-cli -p 7000 cluster info查看集群信息。如下：



Redis集群中的节点负有以下责任：

·保存客户端发送过来的键值对数据

·记录集群的状态，以及某个键到其所对应的节点映射

·自动发现其他节点，监控其他节点，当某个节点出现故障时，进行故障转移

集群之间是互相连接的，组成一幅连通图，它们之间的网络连接时TCP连接，使用二进制协议（Gossip协议）进行通信。集群节点之间使用Gossip协议来完成以下工作：

·在节点之间互相传播集群信息，发现新节点

·时常监控其他节点，向其他节点发送PING数据包，监控其他节点是否正常

运行

·在特定事件发送时，发送集群消息等

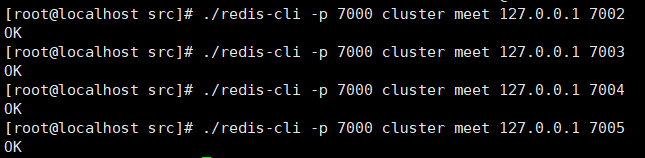
1. 进行meet操作：执行命令redis-cli -p 7000 cluster meet 127.0.0.1.7001，7000与7001节点就完成了握手关系的建立。：



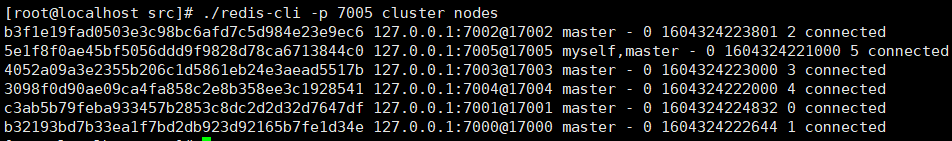
执行命令redis-cli -p 7000 cluster nodes，查看节点关系。如下：



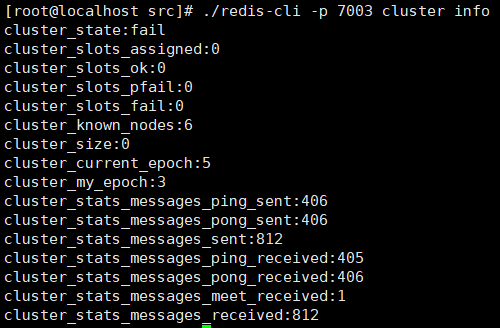
可以看出，7000与7001建立了关系。依上，分别建立7000与7002、7003、7004、7005节点的握手关系。如下：



之后集群的6个节点之间就会相互建立关系，最终形成一幅有6个节点的连通图。执行命令redis-cli -p 7005 cluster nodes查看节点之间的关系。如下：



执行redis-cli -p 7003 cluster info查看集群信息，如下所示：

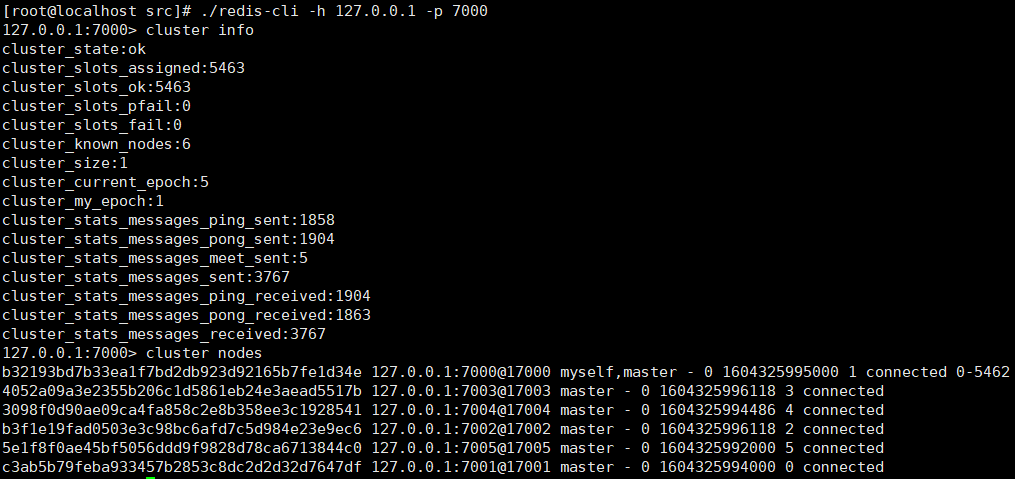


其中，cluster\_known\_nodes：6表示6个节点已经建立了连接。至此，节点之间的meet操作就完成了。Cluster\_state:fail表示集群处于下线状态。这是因为还没有为节点指派槽，集群任然不能对外提供服务，此时进入某个节点的客户端执行命令，仍会返回错误。

1. 指派槽：使用redis-cli -p 7000 cluster addslots {0...5462} 为7000节点指派0~5461范围内的槽，命令如下：



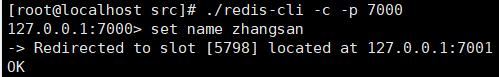
执行该命令后，建立了与7000节点客户端的连接；再执行cluster info和cluster nodes命令，查看集群信息与节点信息。如下：



从返回的信息中可以看出，已经成功为7000节点指派了5462个槽。依此然后为7001节点指派7001节点指派5463~10922范围内的槽。如下：

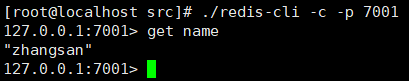


在执行该命令后，建立了与7000节点客户端的连接，然后在执行操作，如下：

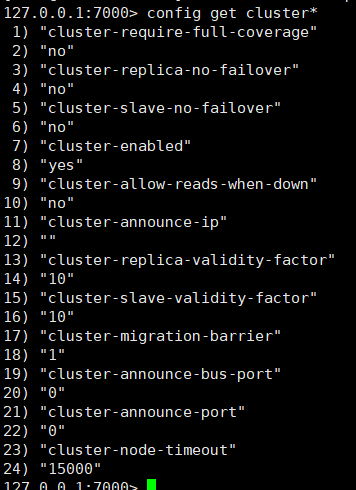


可以发现执行成功了，并派送到了7001节点，此时如果使用7001节点获取，如

下：



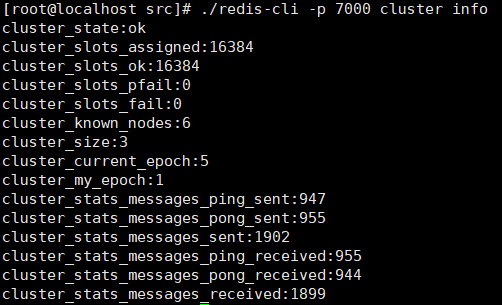
获取成功，表名7000和7001节点现在是互通的。顺便可以查看集群相关配置项。如下：



即使现在还没有为7002节点指派槽，7000节点和7001节点也能对外服务。这是因为我们设置cluster-require-full-coverage选项值为no，即集群中的某个节点宕机了，其他节点也能对外提供服务。最后为7002节点指派10923~16383范围内的槽，如下：

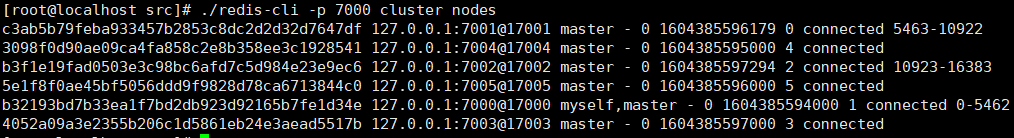


执行命令redis-cli -p 7000 cluster info查看集群信息。如下：



其中cluster\_state:ok表示集群处于上线状态；cluster\_slots\_assigned表示槽的总个数；cluster\_slots\_ok:16384表示槽的状态是OK的数目；cluster\_size:3表示指派槽的节点个数。

执行命令redis-cli -p 7000 cluster nodes查看集群节点信息。如下：



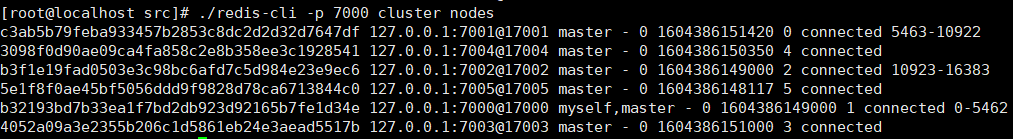
到此，主节点的槽就已经指派完成了。

1. 主从分配：主从分配关系：7003是7000的从节点，7004是7001的从节点，7005是7002的从节点。主从分配命令格式如下：

redis-cli -p <从节点端口> cluster replicate <主节点node-id>

要分配主从关系，先要获得主节点的node-id。执行如下命令查看节点node-id：

redis-cli -p 7000 cluster nodes

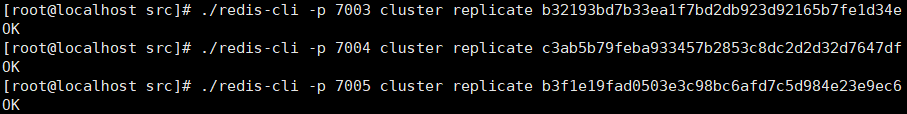


分别执行如下命令分配主从关系：

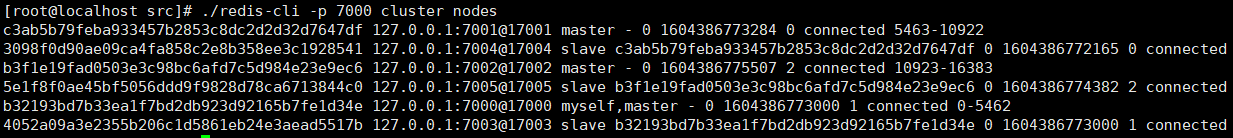
redis-cli -p 7003 cluster replicate b32193bd7b33ea1f7bd2db923d92165b7fe1d34e

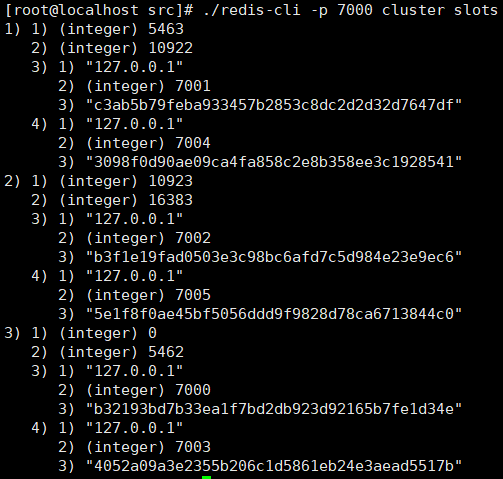
redis-cli -p 7004 cluster replicate c3ab5b79feba933457b2853c8dc2d2d32d7647df

redis-cli -p 7005 cluster replicate b3f1e19fad0503e3c98bc6afd7c5d984e23e9ec6



再次执行redis-cli -p 7000 cluster nodes命令，就能看到三主三从的主从关系了。如下：



 执行redis-cli -p 7000 cluster slots查看集群槽的指派信息，同时也能看到主从关系：

至此，Redis键搭建就已经完成了，以上案例是在同一台机器上完成的。在实际应用中，需要配置6台或3台Redis服务器，才能实现真正的高可用集群模式。如果是3台服务器，两台之间可互为主从，则也可以搭建一个集群模式。

Redis集群中的主从复制，其中主节点主要用于处理槽，而从节点则用于复制其对应的主节点。当某个主节点发生故障时，其从节点就会代替这个故障了的主节点，完成故障转移操作。执行命令cluster replicate <node\_id>来设置一个从节点，其中node\_id表示主节点的节点ID。该命令成功执行之后，从节点就会复制主节点的数据，完成主从复制功能。关于集群的主从复制和故障转移功能在此就不再多说了，其原理在前面的小节中已经说过，只是会有略微的变化。

## 使用Redis集群

在对集群节点指派槽之后，集群就属于上限状态，就可以对外提供服务了，也就是客户端可以向集群节点发送命令请求了。当客户端向集群中的某个节点发送与数据库键有关的命令，节点在接收在命令后，会根据这条命令计算出要处理的数据库键属于哪个槽，并判断这个槽是否在自己的槽的范围内，换句话说，就是判断这个槽是否由当前节点复制处理。

·如果这条命令的键所在的槽正好在当前节点槽的范围内，也就是键所在的槽指派

给了当前节点，那么这个节点就会成功执行这条命令。

·如果这条命令的键所在的槽不在当前节点槽的范围内，那么在执行这条命令后，

节点将会向客户端返回一个MOVED错误信息，并给出这条命令的键所在的槽及节点的IP地址和端口信息提示。如下：



从返回的信息中可以看出，键name所在的槽是5798，这条命令应该在127.0.0.1:7001节点中执行。

如何计算一个给定的键属于哪个槽呢？Redis集群节点采用CRC16算法来计算给定的键属于哪个槽。

HASH\_SLOT=CRC16(key) mod 16383

其中，CRC16(key)语句用于计算给定键的CRC16校验和；mod是取余操作，用于计算出0~16338之间的整数作为给定键所在槽号。使用命令cluster keyslot <key>来获取指定键属于哪个槽，也就是获取键所在的的槽号。命令操作如下：



在执行cluster keyslot <key>命令后，底层会调用CRC16(key)算法来计算出给定键所在的槽号，并返回给客户端。根据CRC(key)算法计算出槽号后，如何判断这个槽号是否在自己（当前节点）槽的范围内呢？根据CRC16(key)算法计算出给定键所在的槽号i后，节点会根据clusterState.clost[i]（其中i是slots数组的下标）是否等于clusterState.myself来判断键所在的槽是否由自己负责。

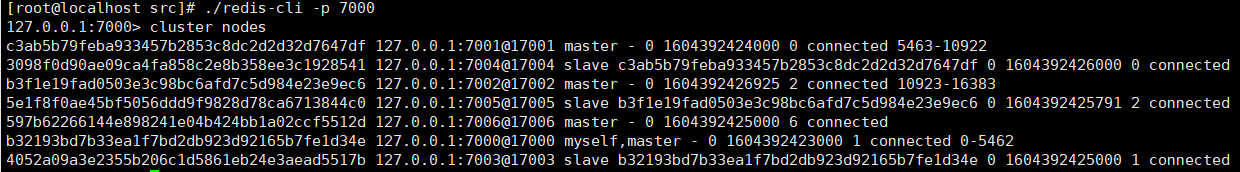
当clusterState.slots[i]等于clusterState.myself时，表示槽i在当前节点槽的范围内，

槽i由当前节点负责，这条命令可以在该节点上成功执行；

当clusterState.slots[i]不等于clusterState.myself时，表示槽i不在当前节点槽的范围内，当前节点不负责处理槽i。在执行这条命令后，节点会根据clusterState.slots[i]指向的clusterNode结构所记录的节点IP地址和端口信息，向客户端返回MOVED错误，并返回该条命令的键所在的槽号，以及所在节点的IP地址和端口信息

## 重新分片

Redis集群的重新分片操作可以将任意数量已经指派给某个（源节点）的槽重新指派给新节点（目标节点），重新分片之后，相关槽所属的键值对也会从源节点转移到目标节点。集群重新分区的过程可以在线进行，集群不需要下线处理，此时源节点和目标节点都能继续处理其他命令请求。下面为之前搭建的集群再添加一个节点127.0.0.1:7006，来进行重新分片操作。这个过程与前面讲解的搭建集群的操作步骤相同，在这里不在详细讲解。我们先新建并编辑新节点的配置文件redis-sluter-7006.conf，然后启动新节点。查看添加新节点后的集群节点消息



之后利用集群管理软件redis-trib完成重新分片操作，具体过程如下：

1. redis-trib对目标节点发送cluster setslot <slot> imporeing <source\_id>命令，来告知目标节点将会有新的键值对所对应的槽被导入，其中source\_id是目标节点的节点ID
2. 同时，redis-trib会发送命令cluster setslot<slot> migrating <target\_id>到源节点，来告知源节点属于槽的键值对将会被转移到目标节点，其中target\_id是源节点的节点ID
3. redis-trib向源节点发送命令cluster getkeysinslot<slot> <count>，来从源节点中获取count个属于槽的键值对的键名。
4. redis-trib向源节点发送migrate <target\_ip> <target\_port> <key\_name> 0 <timeout>命令，将被选中的键名及其值转移到目标节点中；一直循环执行该命令知道所有被选中的键名及其值被转移完为止。
5. 转移完之后，redis-trib会向集群中的任意一个节点发送cluster setslot <slot> node <target\_id>命令，来通知集群中的所有节点，它们以及将源节点中的部分槽指派给了目标节点，最终集群中的所有节点都会知道槽以及被指派给了目标节点。

以上这个过程就是重新分片的过程。至于提到的redis-trib工具，请读者自行学习相关资料。

## 集群中的错误

1. **MOVED错误**

集群搭建完成后，当客户端向其中一个节点发送数据库键命令时，如果这条命令的键所对应的槽号不在该节点指派槽的范围内，就会返回一个MOVED错误信息。如下：



MOVED错误的格式如下：

MOVED <slot> <ip>:<port>

这个错误信息包含键所属的槽号，以及负责处理这个槽的节点的IP地址和端口号信息，当客户端接收到节点返回的MOVED错误时，会根据错误信息中的IP地址和端口信息，专转向负责处理指定键的槽所属的节点，并向该节点重新发送这条命令请求。一个集群客户端通常会与集群中的其他节点创建套接字连接，而前面所说的节点转向实际上就是换一个套接字来发送命令。假如这个客户端还没有与想要转向的节点创建套接字连接，就会先根据MOVED错误提供的IP地址和端口信息来进行连接，在进行节点转向操作。

1. **ASK错误**

在进行重分片的时候，在源节点向目标节点转移一个槽的过程中，可能会产生这样的情况：被转移的槽的数据有可能部分存在于源节点中，另一部分随转移而进入目标节点中。此时，当客户端向源节点发送与数据库键相关命令请求时，而恰好要处理的键就在被重新指派到目标节点的槽中时，源节点在接收到这条命令请求时，会到自己的数据库中查找这个需要处理的键，如果顺利找到这个键，就成功执行这条命令请求；如果没有找到这个需要处理的键，就说明这个键所对应的槽已经被重新指派到新节点中，此时源节点会向客户端返回一个ASK错误，提供客户转向目标节点，然后重新执行这条命令请求。发送ASK错误的过程如下图所示：

ASK错误格式如下：

ASK <slot> <ip>:<port>

ASK错误格式与MOVED错误格式类似。比如返回的ASK错误信息为ASK 14531 127.0.0.1:7002，则说明这个键所对应的槽号是14531，并提示到127.0.0.1:7002节点执行该命令。接收到ASK错误信息的客户端会根据其中的IP地址和端口信息转向正在指派槽的目标节点，然后向目标节点重新发送这条命令请求，在这之前，它还会向目标节点发送一条ASKING命令。向目标节点发送ASKKING命令的目的在于打开发送该命令的客户端的REDIS\_ASKING标识，用于判断是否要执行这条命令请求。在遇到ASK错误时，就会发送ASK转向操作。在这个过程中，如果不发送ASKING命令来打开REDIS\_ASKING标识，目标节点就会拒绝执行这条命令请求。

如果客户端向某个集群节点发送一条关于槽i的命令，却没有为这个槽i指派节点，那么，在执行这条命令后，将会返回一个MOVED错误；而如果节点的的clusteState.importing slots\_from[i]显示节点正在指派槽，并且发送命令的客户端具有REDIS\_ASKING标识，那么这个节点将会执行这条关于槽i的命令。REDIS\_ASKING标识是一个一次性标识。当节点执行了一条带有REDIS\_ASKING标识的命令之后，就会立即删除这个标识。

## 集群的消息

Redis集群的多个节点之间互相连通，构成一幅连通图，它们之间通过发送信息、接收信息来进行通信，实现集群之间的数据共享，以达到分布式、高可用的目的。其中，发送信息的节点被称为发送者，而接收信息的节点被称为接收者。消息由消息头和消息正文组成。节点发送的消息类型有以下几类：

·PING消息：集群节点之间通过PING消息，来检测某个节点是否在线。在默认情

况下，集群中的每个节点每隔 1s就会从已知节点列表中随机选出5

个节点，然后对这5个节点中最长时间没有发送PING消息的节点发

送PING消息，来判断这个节点是否在线。

·PONG消息：当消息接收者接收到其他节点发送过来的PING消息或者MEET消

息时，为了表示已经成功接收到PING消息或MEET消息，会向消

息发送者返回PONG消息，以确保成功接收到消息发送者发送过来

的信息。另外，一个节点也可以向其他节点广播自己的PONG消息，

来提示其他节点立即刷新关于这个节点的认识，进行确认自己的在

线状态。

·MEET消息：当消息发送者接收到客户端发送过来的CLUSTER MEET命令时，会

向消息接收者发送MEET消息，请求信息接收者加入消息发送者的

集群中，称为集群中的新节点。

·FALL消息：当一个主节点（master1）判断另一个主节点（master2）已经进入 FALL

状态时，主节点（master1）会向集群中的其他节点发送一条关于主

节点（master2）的FALL消息，来表示主节点（master2）已经下线。当其他节点收到主节点（master1）发送过来的FALL消息时，就会立即将主节点（master2）标记位下线状态，然后主节点 （master1）从属的从节点就会进行master选举，从节点代替这个主节点，完成故障转移等相关操作。

·PUBLISH消息：当集群中的某个节点接收到一条PUBLISH命令时，该节点就会执

行这条命令，同时向集群中的其他节点广播这条PUBLISH消息，让所有接收到这条PUBLISH消息的节点都执行这条PUBLISH命令。

至此，我们已经全面讲解了Redis集群的相关知识。要想了解更多的集群相关执行，请读者自行查阅相关资料学习。