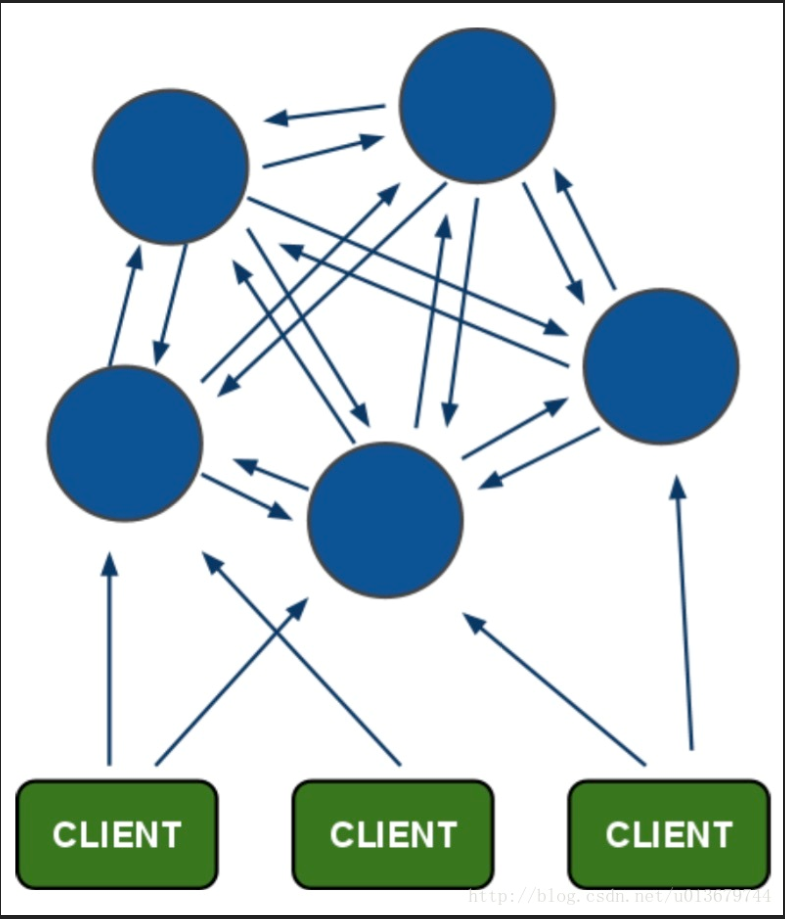
# Redis集群

Redis Cluster是由多个同时服务于一个数据集合的Redis实例组成的整体，对于用户来说，用户只关注这个数据集合，而整个数据集合的某个数据子集存储在哪个节点对于用户来说是透明的。Redis Cluster具有分布式系统的特点，也具有分布式系统如何实现高可用性与数据一致性的难点，由多个Redis实例组成的Redis Cluster结构通常如下：



部署多个节点，可以分布式存储数据，节点之间互相连接，Redis的每个节点复杂部分数据的存储，同时每个节点为了保证数据的高可用性，又会利用master-slaver结构来保证某个节点宕机后不丢失数据。

## **分区的实现原理**

Redis Cluster分区实现原理主要关注三个问题，

1）数据是如何被自动分散到不同的节点的；

2）客户端是如何能够正确找到节点的；

3）键空间迁移过程是怎么样的。

### 槽（slot）概念

Redis Cluster中有一个16384长度的槽的概念，他们的编号为0、1、2、3……16382、16383。这个槽是一个虚拟的槽，并不是真正存在的。正常工作的时候，Redis Cluster中的每个Master节点都会负责一部分的槽，**当有某个key被映射到某个Master负责的槽，那么这个Master负责为这个key提供服务**，至于哪个Master节点负责哪个槽，这是可以由用户指定的，也可以在初始化的时候自动生成（redis-trib.rb脚本）。这里值得一提的是，在Redis Cluster中，只有Master才拥有槽的所有权，如果是某个Master的slave，这个slave只负责槽的使用，但是没有所有权。

每个Master节点维护着一个（16384除8）个字节的位序列，Master节点用bit来标识对于某个槽自己是否拥有。拥有就将对应的位置1。

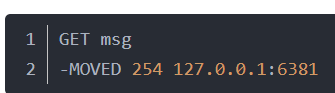
集群同时还维护着槽到集群节点的映射，是由长度为16384类型为节点的数组实现的，槽编号为数组的下标，数组内容为集群节点，这样就可以很快地通过**槽编号找到负责这个槽的master节点**。位序列这个结构很精巧，即不浪费存储空间，操作起来又很便捷。

### 键空间分布基本算法

这里讲的是Redis Cluster如何将键空间分布在不同的节点的，键空间意为Redis Cluster所拥有用户所有数据集合的键的取值范围，这个范围叫做键空间。提到空间分布，必然会想到哈希算法，没错，通过哈希算法再加上取模运算可以将一个值固定地映射到某个区间，在这里，这个区间叫做slots，区间由连续的slot组成。在Redis Cluster中，我们拥有16384个slot，这个数是固定的，我们存储在Redis Cluster中的所有的键都会被映射到这些slot中。简单来说就是和常规的hash算法一样，也是hash取模。

### 客户端访问

Redis Cluster并不会代理查询，那么如果客户端访问了一个此master节点的key并不存在的节点，这个节点是怎么处理的呢？比如我想获取key为msg的值，msg计算出来的槽编号为254，当前节点正好不负责编号为254的槽，那么就会返回客户端下面信息



表示客户端想要的254槽由运行在IP为127.0.0.1，端口为6381的Master实例服务。如果根据key计算得出的槽恰好由当前节点负责，则当期节点会立即返回结果。这里明确一下，没有代理的Redis Cluster可能会导致客户端两次连接急群中的节点才能找到正确的服务，推荐客户端缓存连接，这样最坏的情况是两次往返通信。

### 重新分片（Resharding）

重新分片意为槽到集群节点的映射关系要改变，不变的是键到槽的映射关系，因此当重新分片的时候，如果槽中有键，那么键也是要被移动到新的节点的。

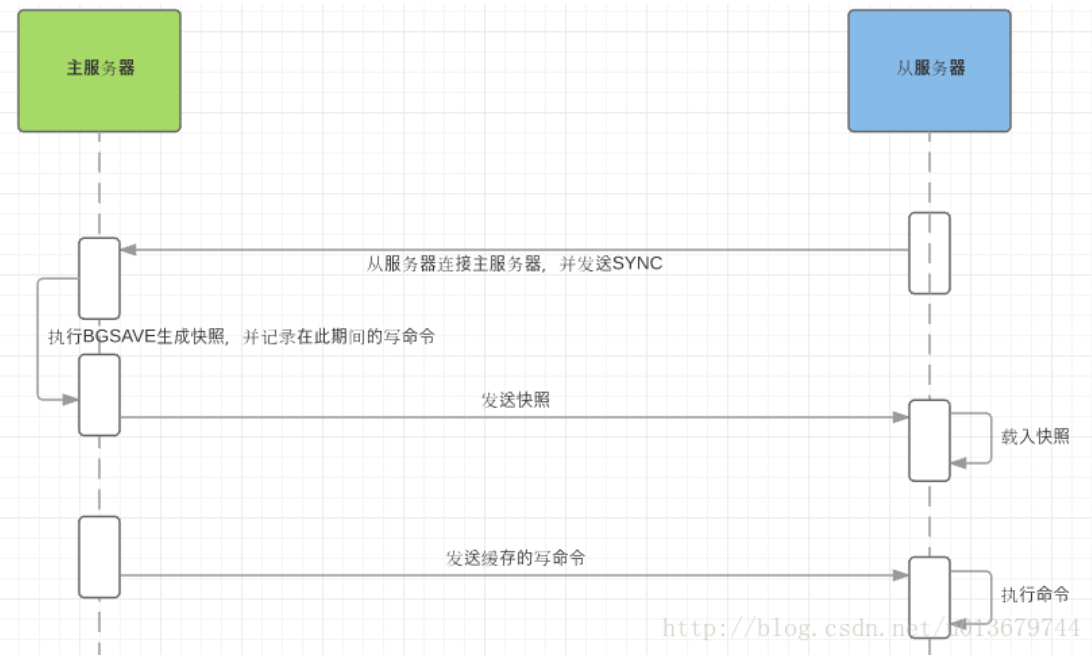
## 主从同步原理

和MySQL主从复制的原因一样，Redis虽然读取写入的速度都特别快，但是也会产生读压力特别大的情况。为了分担读压力，Redis支持主从复制，另外也是为了保证HA。Redis主从复制可以根据是否是全量分为全量同步和增量同步。

### 全量同步

Redis全量复制一般发生在Slave初始化阶段，这时Slave需要将Master上的所有数据都复制一份。具体步骤如下：

1）从服务器连接主服务器，发送SYNC命令；   
　　2）主服务器接收到SYNC命名后，开始执行BGSAVE命令生成RDB文件并使用缓冲区记录此后执行的所有写命令；   
　　3）主服务器BGSAVE执行完后，向所有从服务器发送快照文件，并在发送期间继续记录被执行的写命令；   
　　4）从服务器收到快照文件后丢弃所有旧数据，载入收到的快照；   
　　5）主服务器快照发送完毕后开始向从服务器发送缓冲区中的写命令；   
　　6）从服务器完成对快照的载入，开始接收命令请求，并执行来自主服务器缓冲区的写命令；



### 增量同步

Redis增量复制是指Slave初始化后开始正常工作时主服务器发生的写操作同步到从服务器的过程。 增量复制的过程主要是主服务器每执行一个写命令就会向从服务器发送相同的写命令，从服务器接收并执行收到的写命令。

**Redis的同步策略是：主从刚刚连接的时候，进行全量同步；全量同步结束后，进行增量同步。当然，如果有需要，slave 在任何时候都可以发起全量同步。redis 策略是，无论如何，首先会尝试进行增量同步，如不成功，要求从机进行全量同步。**

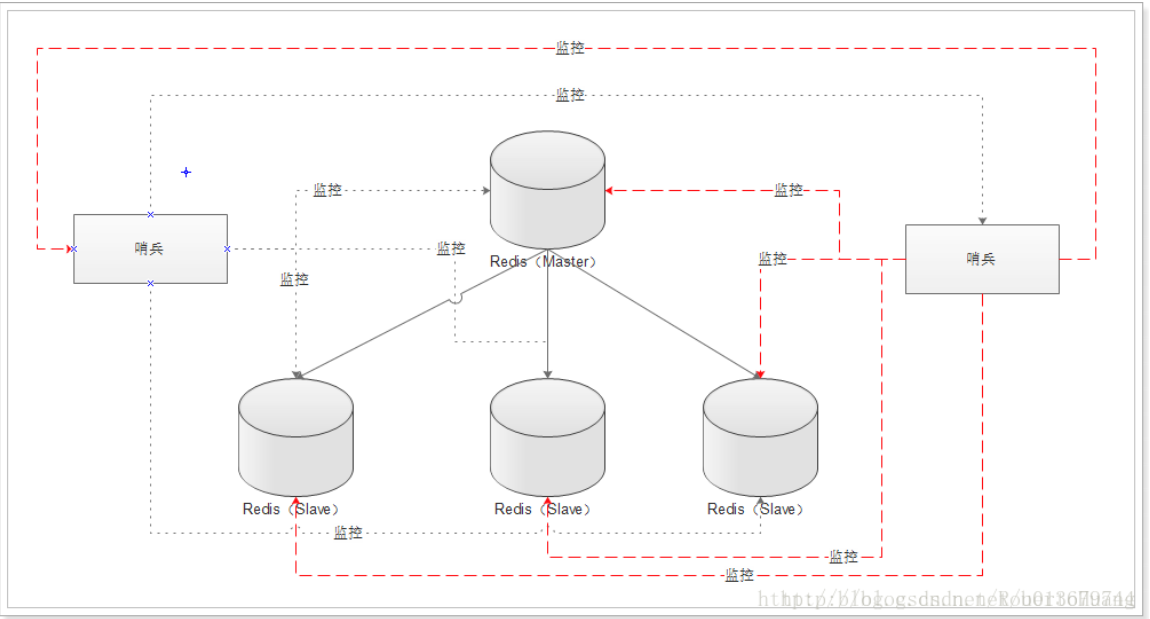
## 哨兵机制

Redis-Sentinel也就是哨兵机制，是Redis官方推荐的高可用性(HA)解决方案，当用Redis做Master-slave的高可用方案时，假如master宕机了，Redis本身(包括它的很多客户端)都没有实现自动进行主备切换，而Redis-sentinel本身也是一个独立运行的进程，它能监控多个master-slave集群，发现master宕机后能进行自动切换。

它的主要功能有以下几点：

* 不时地监控redis是否按照预期良好地运行;
* 如果发现某个redis节点运行出现状况，能够通知另外一个进程(例如它的客户端);
* 能够进行自动切换（进行主备切换）。当一个master节点不可用时，能够选举出master的多个slave(如果有超过一个slave的话)中的一个来作为新的master,其它的slave节点会将它所追随的master的地址改为被提升为master的slave的新地址。

### 哨兵机制的运行过程



如上图所示，Redis为了保证高可用性，采用Master-slave形式部署，采用AOF或RDB进行持久化，采用集群culster机制来分布式存储。多个哨兵,不仅同时监控主从数据库,而且哨兵之间互为监控，并且哨兵不会存在单点问题。哨兵会监控主从Redis服务器是否正常，当发现有问题时，会进行通知，并进行故障转移。当监控到master节点宕机后，会进行maser选举出新的master节点，并进行主从切换，并将其他节点作为新选举出的master节点的slave。哨兵Sentinel用到的分布式一致性算法是Raft分布式算法。