# Redis高级

## Redis持久化

### 问题描述

Redis的数据都保存在内存中,如果遇到宕机或者断电,则内存数据清空.数据丢失.但是在生产环境下不允许出现数据丢失的问题.

问如何解决???

### 持久化原则

redis内部根据特定的规则(1.RDB 2.AOF模式),定期将数据进行持久化操作(将内存数据保存到本地磁盘中).这时如果遇到宕机,则服务重新启动读取指定的持久化问题,用来恢复数据.

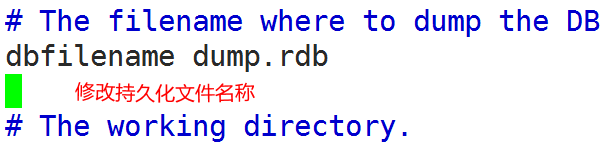
### RDB模式

#### RDB模式介绍

1. RDB模式是redis的默认的持久化策略.
2. RDB模式的持久化效率是最高的
3. RDB模式是定期备份 风险:可能会丢失数据
4. RDB模式做的是内存数据的**快照**,持久化文件占用空间较小.

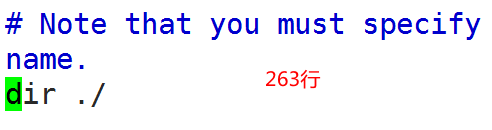
#### RDB模式配置策略

1. save 表示主动进行持久化操作. 影响:如果执行save操作,则不允许用户执行set等更新操作.用户陷入阻塞.
2. bgsave 表示后台运行持久化操作(异步操作),这时不会影响用户set等更新操作.但是持久化什么时候完成不能确定时间. (gc)
3. 修改持久化文件名称



1. 持久化文件路径

要求持久化文件在当前目录生成.



#### Redis持久化策略

save 900 1

save 300 10

save 60 10000

1. 如果用户在900秒内执行1次更新操作.则持久化一次.
2. 如果用户在300秒内执行10次更新操作,则持久化一次.
3. 60秒内持久化一次

用户操作越频繁则持久化周期越短.

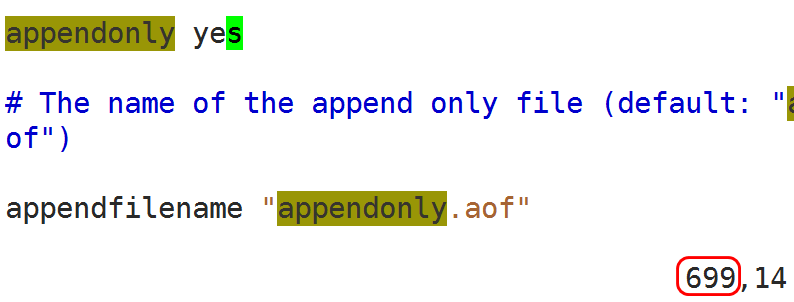
### AOF模式

#### 模式说明

1. AOF模式可以实现数据的**实时**持久化操作.
2. AOF模式默认是关闭的.需要手动开启.
3. AOF模式记录的是用户的**执行过程**,持久化文件占用空间大.持久化效率低.
4. AOF模式持久化是异步操作.

#### AOF模式配置策略

1.开启AOF模式



#### 持久化策略

#appendfsync always 只要用户执行更新操作 则持久化一次

appendfsync everysec redis1秒持久化一次

#appendfsync no 将持久化时间交给操作系统决定(20分钟)

### 关于AOF与RDB总结

1. RDB模式与AOF模式可以同时存在.
2. 默认条件下以AOF为主.
3. 当程序执行save执行时才会执行RDB模式操作.

## Redis内存策略

### 问题说明

如果redis内存被占满如何处理???

### 内存策略

1. volatile-lru 将设定超时时间的数据采用LRU算法删除.

2. allkeys-lru 将所有的数据采用LRU算法删除.

3. volatile-lfu 将设定超时时间的数据采用LFU算法删除

4. allkeys-lfu 将所有数据采用LFU算法删除

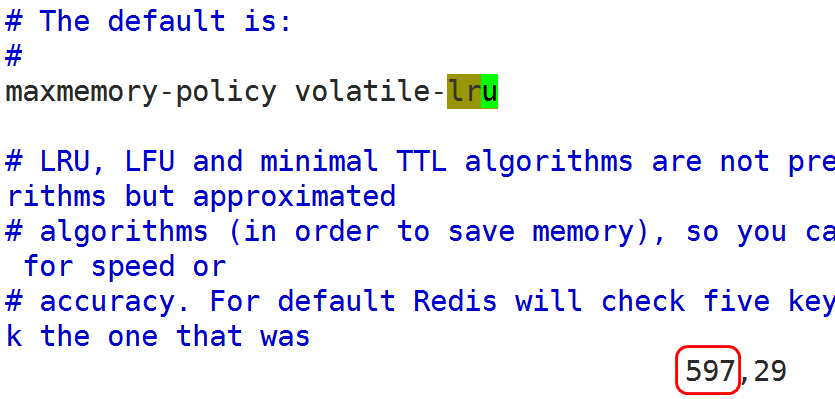
5. volatile-random 将设定超时时间的数据 随机删除.

6. allkeys-random 所有的数据都随机删除

7. volatile-ttl 将设定超时时间的数据 按照剩余时间排序删除.

8. noeviction 表示不删除数据,如果内存占满 报错返回.

### 内存策略修改



### LRU算法

LRU是Least Recently Used的缩写，即最近最少使用，是一种常用的[页面置换算法](https://baike.baidu.com/item/%E9%A1%B5%E9%9D%A2%E7%BD%AE%E6%8D%A2%E7%AE%97%E6%B3%95/7626091)，选择**最近最久未使用的页面予以淘汰**。该算法赋予每个[页面](https://baike.baidu.com/item/%E9%A1%B5%E9%9D%A2/5544813)一个访问字段，用来记录一个页面自上次被访问以来所经历的时间 t，当须淘汰一个页面时，选择现有页面中其 t 值最大的，即最近最久未使用的页面予以淘汰。

### LFU算法

常识:Redis5.0之后才有LFU算法!!!!

LFU（least frequently used (LFU) page-replacement algorithm）。即最不经常使用页置换算法，要求在页置换时置换引用计数最小的页，因为经常使用的页应该有一个较大的**引用次数**。但是有些页在开始时使用次数很多，但以后就不再使用，这类页将会长时间留在内存中，因此可以将引用计数寄存器定时右移一位，形成指数衰减的平均使用次数。

least frequently used (LFU) page-replacement algorithm

即最不经常使用页置换算法，要求在页置换时置换引用计数最小的页，因为经常使用的页应该有一个较大的引用次数。但是有些页在开始时使用次数很多，但以后就不再使用，这类页将会长时间留在内存中，因此可以将引用计数寄存器定时右移一位，形成指数衰减的平均使用次数。

# Redis集群

## redis集群业务说明

### Redis分片和哨兵特点

优点:

1. 分片可以实现redis内存的动态扩容.
2. 哨兵可以实现redis的高可用.

缺点:

1. 分片:如果一个节点出现宕机则整个分片无法运行.
2. 哨兵:虽然哨兵可以实现Redis高可用,但是哨兵本身没有高可用效果. 如果哨兵宕机则整个服务宕机.

### Redis集群说明

Redis集群相当于整合redis分片机制(内存扩容),redis哨兵机制(高可用).Redis集群由所有的主节点负责监控和选举.从而在不依赖第三方的基础之后实现了redis集群的高可用.

细节:集群中的所有的节点都能互相通信.(PING-PONG)

## Redis集群搭建步骤

### 为什么要搭建集群

通常，为了提高网站响应速度，总是把热点数据保存在内存中而不是直接从后端数据库中读取。

Redis是一个很好的Cache工具。大型网站应用，热点数据量往往巨大，几十G上百G是很正常的事儿。

由于内存大小的限制，使用一台 Redis 实例显然无法满足需求，这时就需要使用多台 Redis作为缓存数据库。但是如何保证数据存储的一致性呢,这时就需要搭建redis集群.采用合理的机制,保证用户的正常的访问需求.

采用redis集群,可以保证数据分散存储,同时保证数据存储的一致性.并且在内部实现高可用的机制.实现了服务故障的自动迁移.

### 集群搭建计划

主从划分:

3台主机 3台从机共6台 端口划分7000-7005

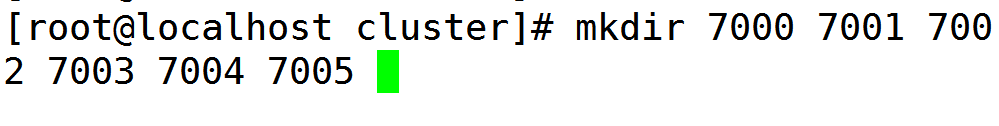
## 集群搭建

### 准备集群文件夹

1.准备集群文件夹

Mkdir cluster

2.在cluster文件夹中分别创建7000-7005文件夹



### 复制配置文件

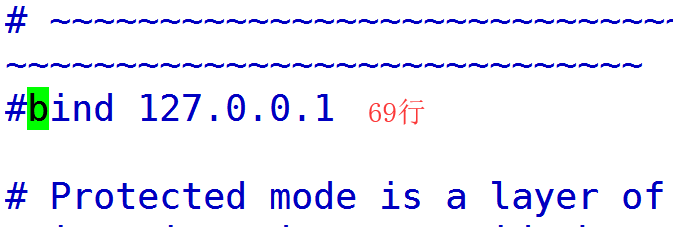
说明:

将redis根目录中的redis.conf文件复制到cluster/7000/ 并以原名保存

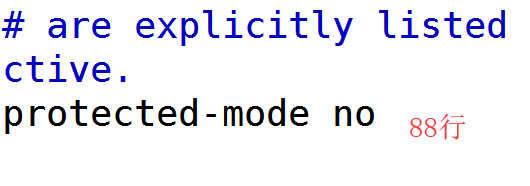
cp redis.conf cluster/7000/

### 编辑配置文件

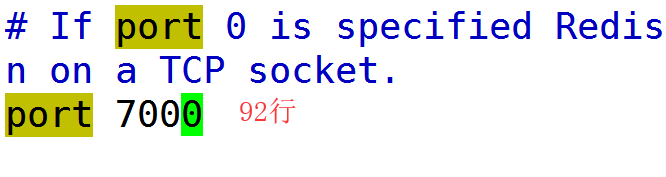
1. 注释本地绑定IP地址



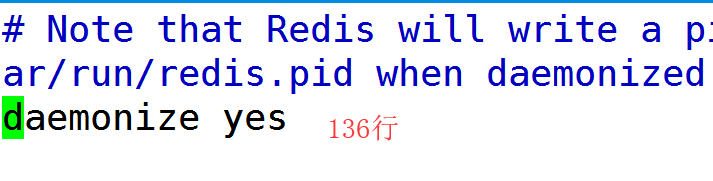
1. 关闭保护模式



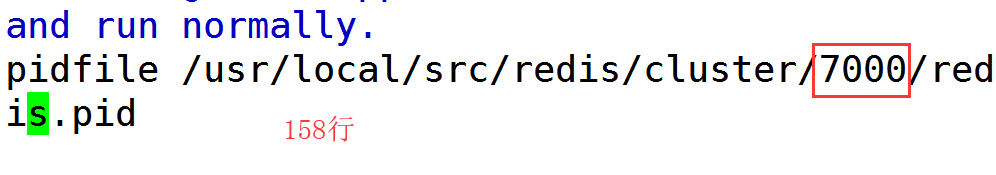
1. 修改端口号



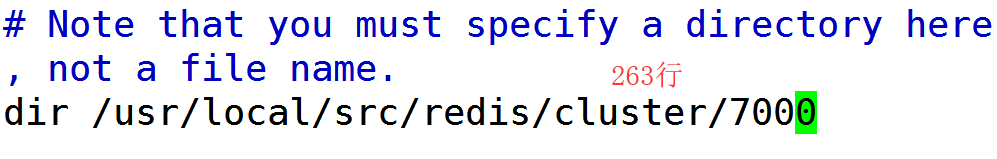
1. 启动后台启动



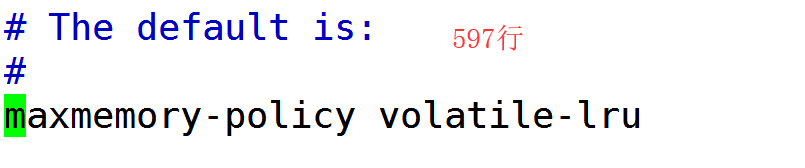
1. 修改pid文件



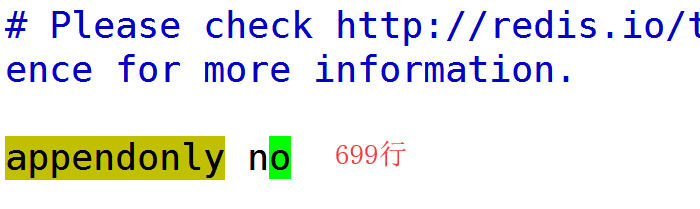
1. 修改持久化文件路径



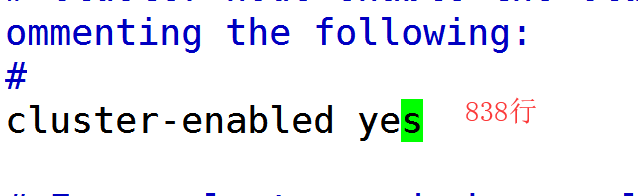
1. 设定内存优化策略



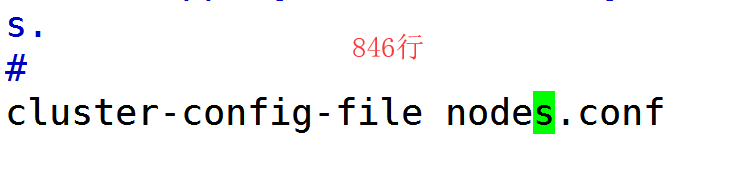
1. 关闭AOF模式



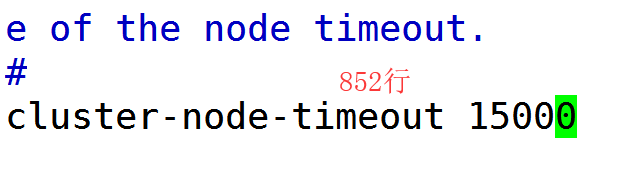
1. 开启集群配置



1. 开启集群配置文件



1. 修改集群超时时间



### 复制修改后的配置文件

说明:将7000文件夹下的redis.conf文件分别复制到7001-7005中

[root@localhost cluster]# cp 7000/redis.conf 7001/

[root@localhost cluster]# cp 7000/redis.conf 7002/

[root@localhost cluster]# cp 7000/redis.conf 7003/

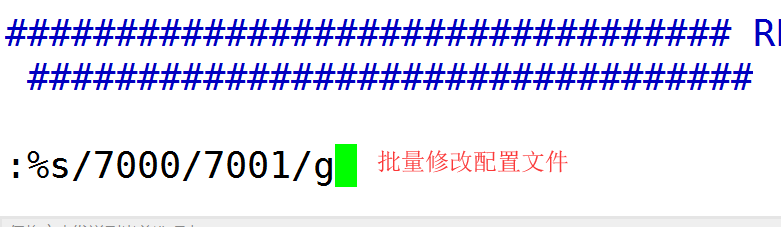
[root@localhost cluster]# cp 7000/redis.conf 7004/

[root@localhost cluster]# cp 7000/redis.conf 7005/

### 批量修改

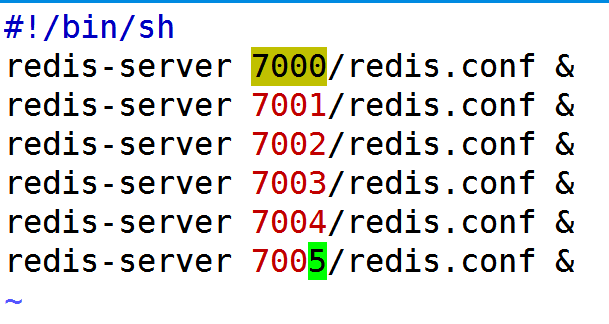
说明:分别将7001-7005文件中的7000改为对应的端口号的名称,

修改时注意方向键的使用

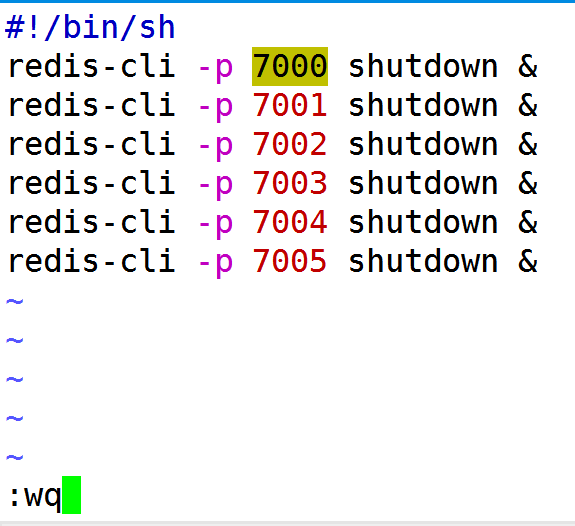


### 通过脚本编辑启动/关闭指令

1. 创建启动脚本 vim start.sh



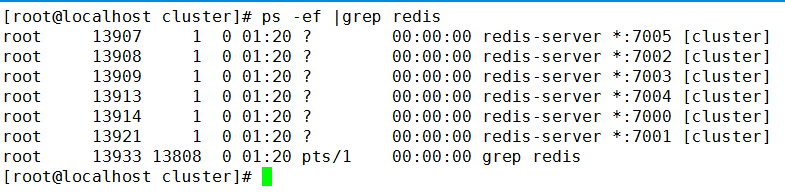
1. 编辑关闭的脚本 vim shutdown.sh



1. 启动redis节点

sh start.sh

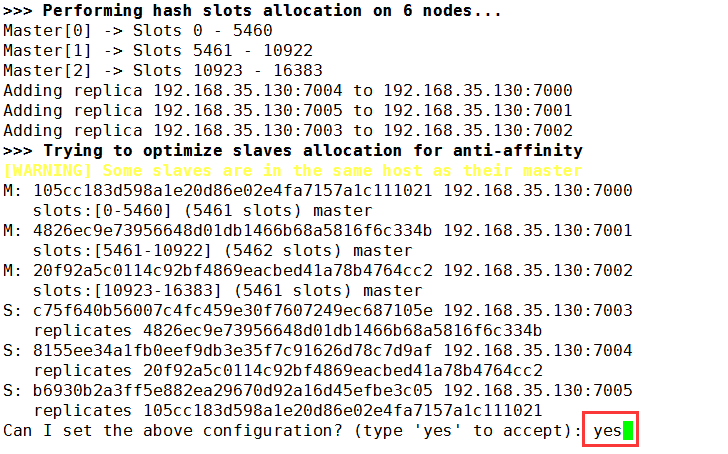
1. 检查redis节点启动是否正常

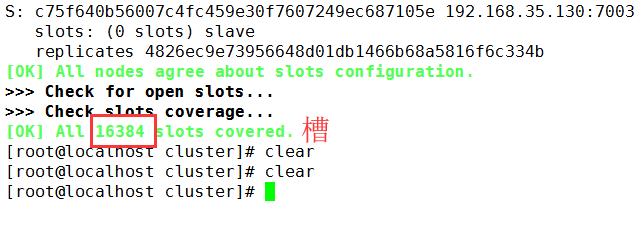


### 创建redis集群

#5.0版本执行 使用C语言内部管理集群

redis-cli --cluster create --cluster-replicas 1 192.168.35.130:7000 192.168.35.130:7001 192.168.35.130:7002 192.168.35.130:7003 192.168.35.130:7004 192.168.35.130:7005



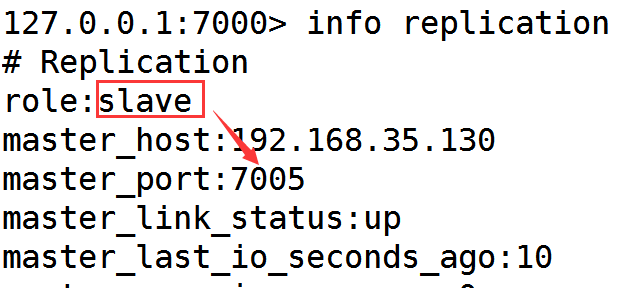


### Redis集群高可用测试

1. 关闭redis主机.检查是否自动实现故障迁移.
2. 再次启动关闭的主机.检查是否能够实现自动的挂载.

一般情况下 能够实现主从挂载

个别情况: 宕机后的节点重启,可能挂载到其他主节点中(7001-7002) 正确的



## Redis集群原理

### Redis集群高可用推选原理

如图-24所示

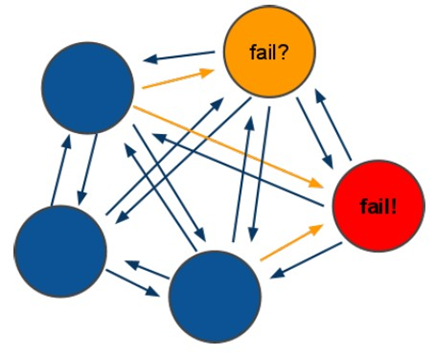


图- 24

原理说明:

Redis的所有节点都会保存当前redis集群中的全部主从状态信息.并且每个节点都能够相互通信.当一个节点发生宕机现象.则集群中的其他节点通过PING-PONG检测机制检查Redis节点是否宕机.当有半数以上的节点认为宕机.则认为主节点宕机.同时由**Redis剩余的主节点**进入选举机制.投票选举链接宕机的主节点的从机.实现故障迁移.

### Redis集群宕机条件

特点:集群中如果主机宕机,那么从机可以继续提供服务,

当主机中没有从机时,则向其它主机借用多余的从机.继续提供服务.如果主机宕机时没有从机可用,则集群崩溃.

答案:9个redis节点,节点宕机5-7次时集群才崩溃.

如图-25所示:

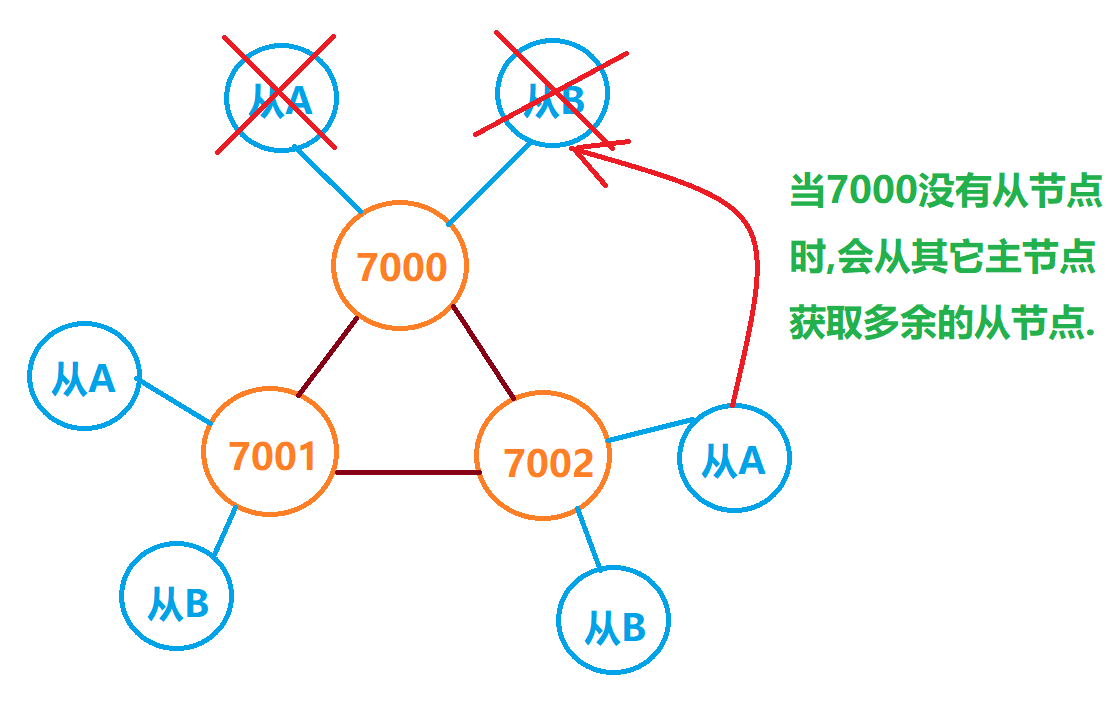


图- 25

### 集群主机为什么是基数

**集群存活节点数量 > n/2**

3 > 1.5 3台可以搭建集群 宕机1台

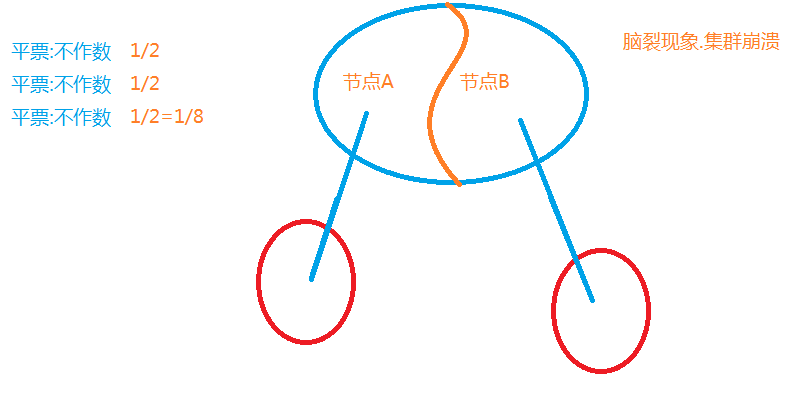
4 > 2 4台可以搭建集群 宕机1台

通过上述证明.集群搭建的主机台数是基数时 解是最优的.

一般情况下的集群是基数的要求. zookeeper Eureka集群

### 如果选举平票如何处理?

规定投票进行3次.如果连续平票的概率是1/8.概率偏低.如果适当增加主节点的数量则可以有效的降低平票导致脑裂的风险.



### Redis hash槽存储数据原理

说明: RedisCluster采用此分区，所有的键根据哈希函数(**CRC16**[key]&16383)映射到0－16384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.根据主节点的个数,均衡划分区间.

 算法:哈希函数: Hash()=CRC16[key]&16384按位与

如图-26所示

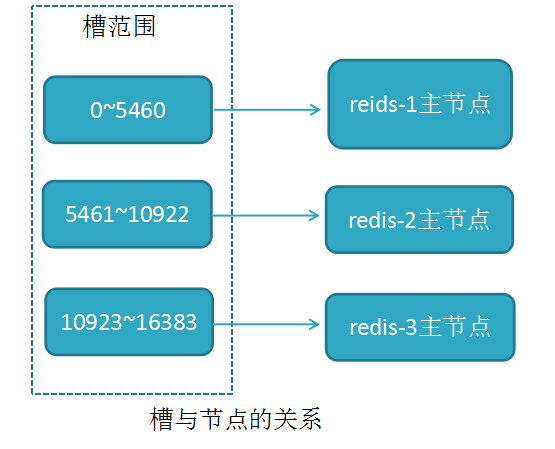


图- 26

当向redis集群中插入数据时,首先将key进行计算.之后将计算结果匹配到具体的某一个槽的区间内,之后再将数据set到管理该槽的节点中.

如图-27所示

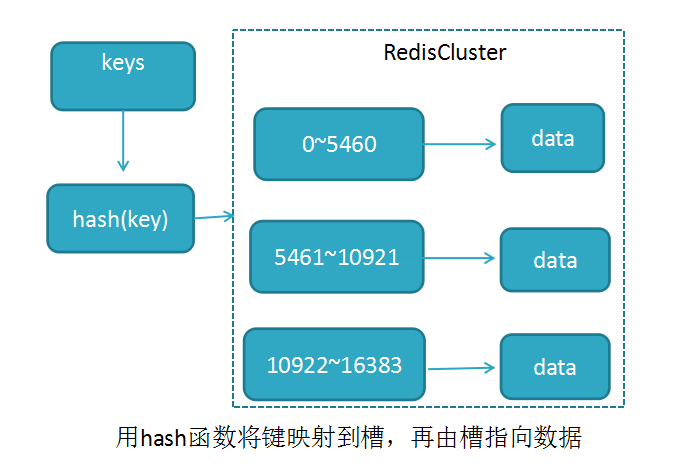


图- 27

### 关于分片与集群总结

说明:

1.分片的运算发生在tomcat服务器中.数据直接计算完成之后保存到了具体的redis节点中.

2.redis集群是将key在redis内部进行计算.之后保存数据,性能浪费在计算上.

## SpringBoot整合Redis集群

### 入门案例

/\*\*

\* 测试redis集群

\*/

@Test

**public** **void** testCluster() {

Set<HostAndPort> nodes = **new** HashSet<HostAndPort>();

nodes.add(**new** HostAndPort("192.168.182.129",7000));

nodes.add(**new** HostAndPort("192.168.182.129",7001));

nodes.add(**new** HostAndPort("192.168.182.129",7002));

nodes.add(**new** HostAndPort("192.168.182.129",7003));

nodes.add(**new** HostAndPort("192.168.182.129",7004));

nodes.add(**new** HostAndPort("192.168.182.129",7005));

JedisCluster jedisCluster = **new** JedisCluster(nodes);

jedisCluster.set("1903","集群搭建完成!!!!!");

System.***out***.println(jedisCluster.get("1903"));

}

### 编辑Properties文件

#redis集群

redis.nodes=192.168.182.129:7000,192.168.182.129:7001,192.168.182.129:7002,192.168.182.129:7003,192.168.182.129:7004,192.168.182.129:7005

### 编辑redis配置类

@Configuration //标识我是一个配置类

@PropertySource("classpath:/properties/redis.properties")

**public** **class** RedisConfig {

@Value("${redis.nodes}")

**private** String nodes; //node,node,....

/\*\*

\* 搭建redis集群

\*/

@Bean

**public** JedisCluster jedisCluster() {

Set<HostAndPort> nodes = getNodes();

**return** **new** JedisCluster(nodes);

}

//表示不要有重复数据

**private** Set<HostAndPort> getNodes() {

Set<HostAndPort> nodesSets = **new** HashSet<>();

String[] strNode = nodes.split(",");

**for** (String redisNode : strNode) {

String host = redisNode.split(":")[0];

**int** port = Integer.*parseInt*

(redisNode.split(":")[1]);

HostAndPort hostAndPort =

**new** HostAndPort(host, port);

nodesSets.add(hostAndPort);

}

**return** nodesSets;

}

}

### 作业:AOP实现redis缓存