# 哈希一致性

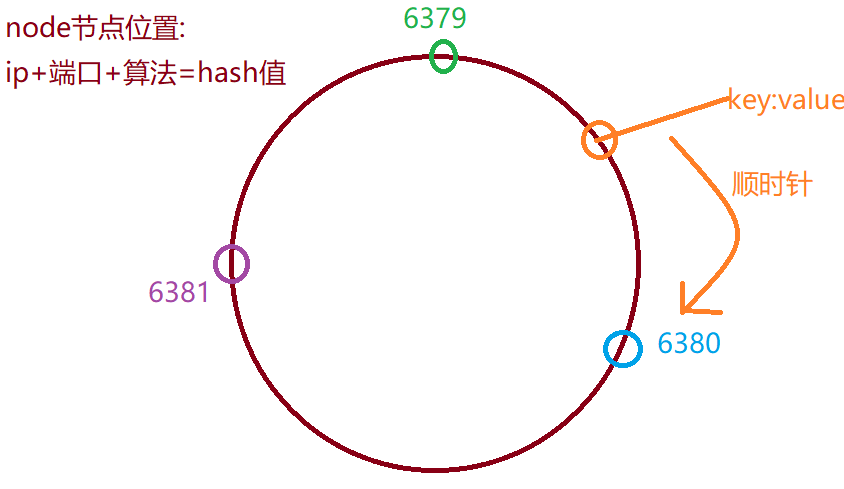
## 哈希一致性算法

说明:redis分片时采用哈希一致性算法,实现数据动态的绑定.

术语:

1.节点(node) 代表真实的redis服务器

2.key 用户保存/读取关键字



### 均衡性

问题:

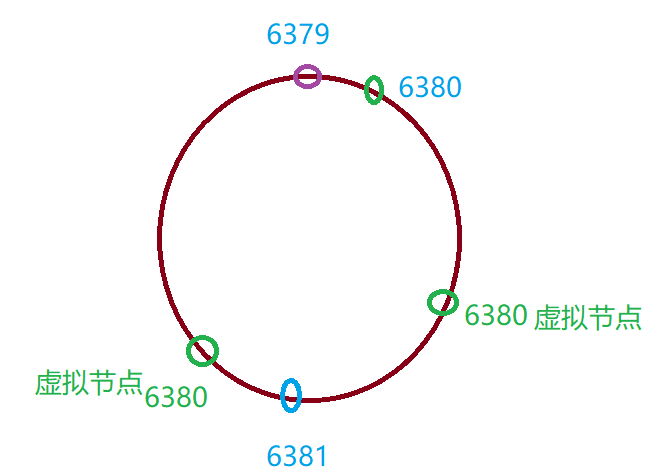
由于node节点是经过hash一致性算法计算的结果.那么可能会出现计算的节点所分配的内存不均.如果内存不均,会导致某些节点内存负载过高.

特点说明:

要求全部的数据,尽可能的均匀分配到不同的节点中,每个节点中保存的数据尽可能接近1/n

实现步骤:

如果遇到节点负载不均时,会自动的启动虚拟节点,进行数据的平衡



### 单调性

说明:如果节点的个数**增加**,原有的节点的挂载会自动的发生变化.将满足条件的数据自动的挂载到新的节点中.

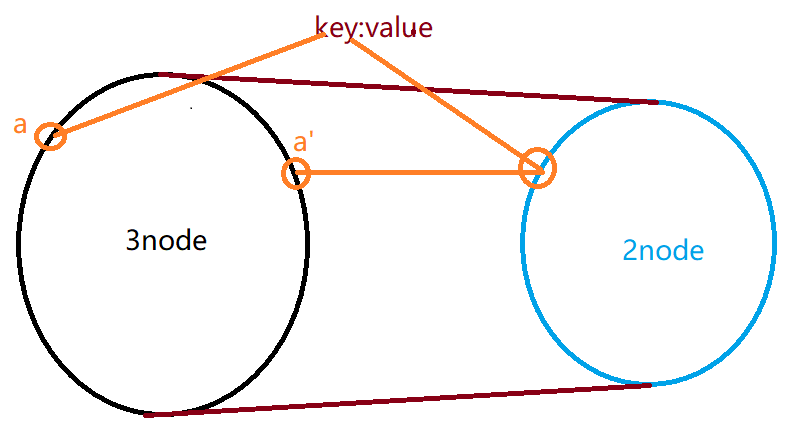
原则:尽可能保证原有的数据不变

总结:节点增加,数据动态发生迁移

### 分散性

说明:由于分布式的项目部署,导致项目不能全部获取node节点.

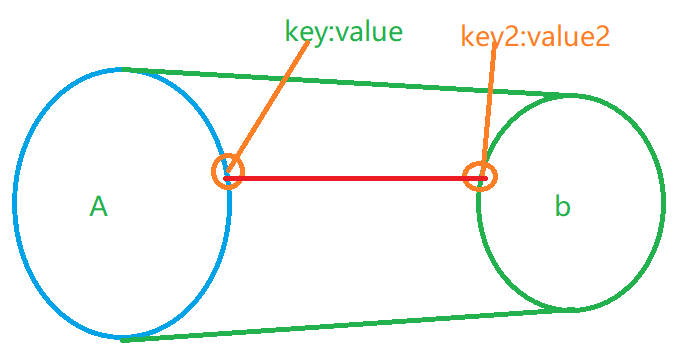
特点:一个key对应了多个位置.



### 负载

说明:负载是从另一个角度谈论分散性.

特点:一个位置,对应多个key



综上所述:

在分布式应用时,应该使用全部的node节点,避免出现这样的现象.

# Redis哨兵实现

## 分片特点

优点:实现内存动态扩容

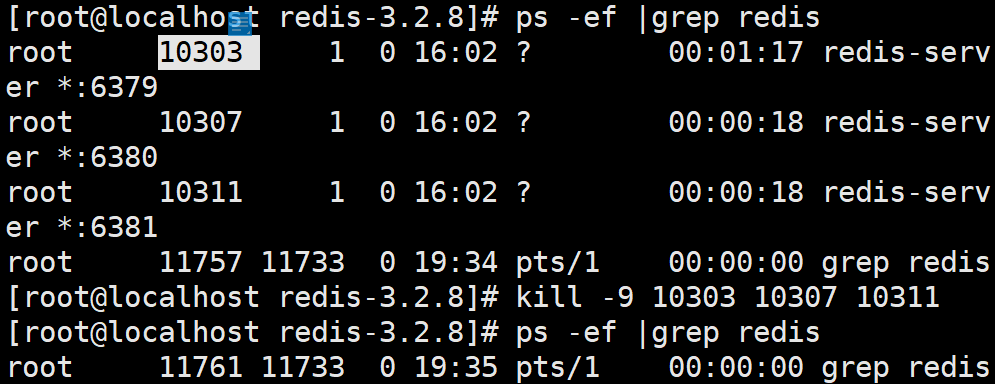
缺点:

1. 如果分片的节点宕机,则数据丢失
2. 如果分片的节点宕机,那么整个服务器都不能正常使用.

## 配置redis主从服务

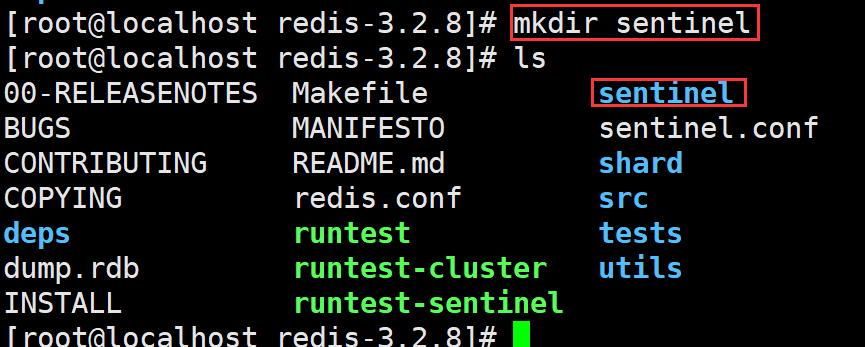
### 准备主从配置文件

1. 关闭redis分片节点

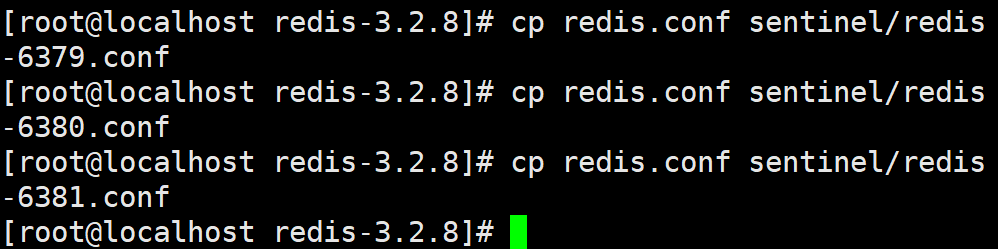


1. 新建文件夹sentinel

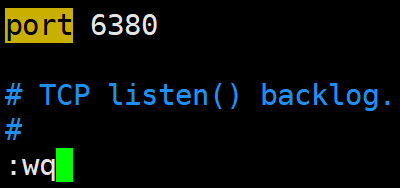
mkdir sentinel



1. 拷贝配置文件



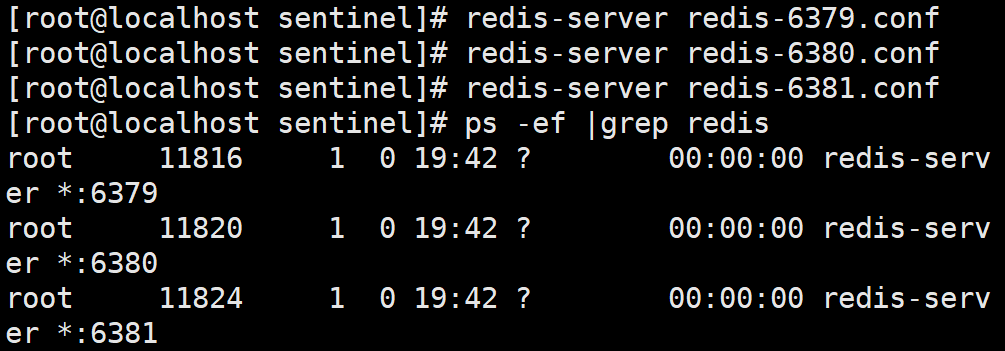
1. 修改端口号 6380/6381



vim redis-6380.conf

vim redis-6381.conf

1. 启动redis节点

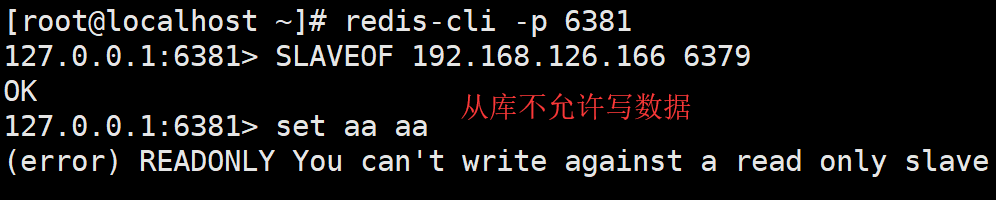


### 实现主从挂载

1.执行命令

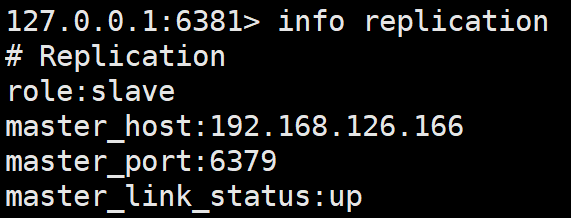
命令:在redis客户端中执行

127.0.0.1:6381> SLAVEOF 192.168.126.166 6379

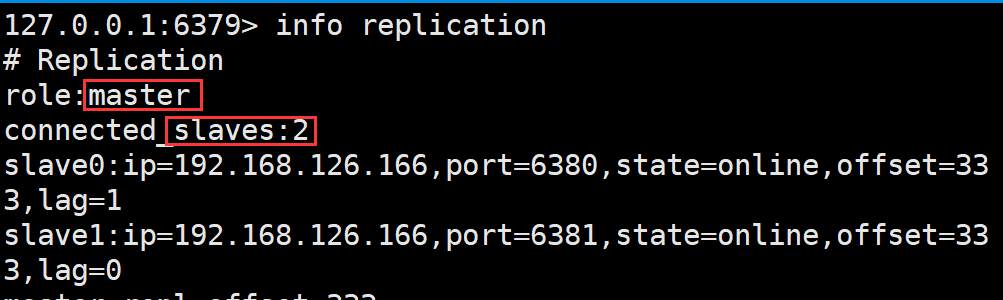


2.检查主从状态

主库状态

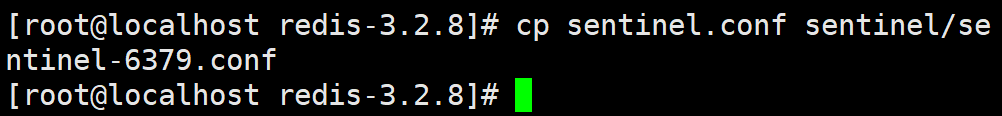


主机状态

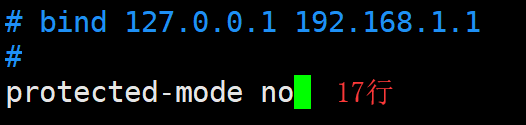


### 添加哨兵的配置

1. 复制哨兵的配置文件



1. 关闭保护模式



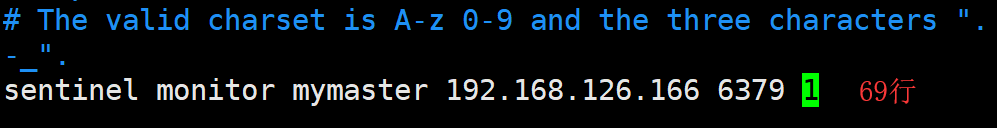
3.修改哨兵的监控

sentinel monitor mymaster 127.0.0.1 6379 1

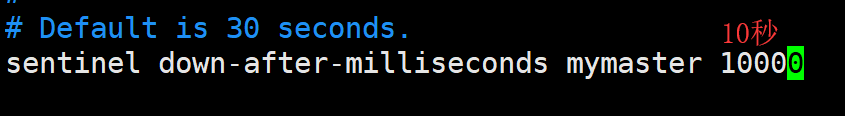
mymaster:代表当前节点中的主机的变量

ip:端口 :主机的IP和端口

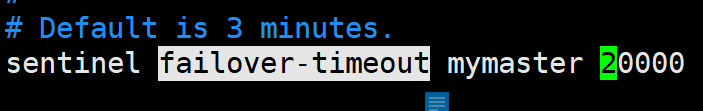
2 :推选的票数



4.修改哨兵的推选时间



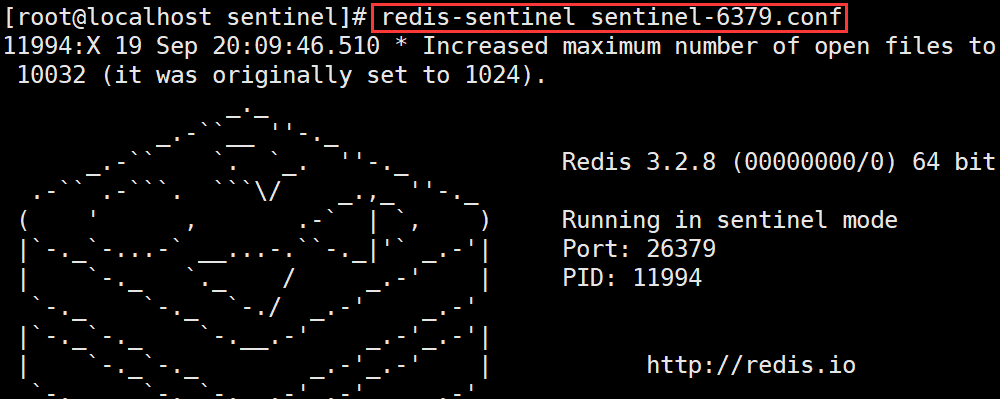
5.修改推选失败的超时时间



### 单台哨兵高可用测试

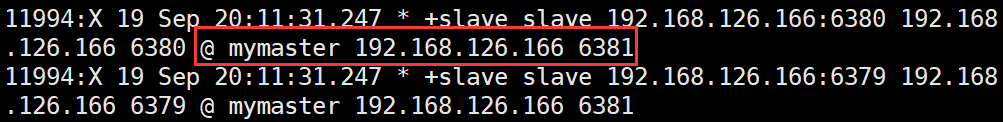
1. 启动哨兵

命令: redis-sentinel sentinel-6379.conf



1. 检查高可用实现

将redis主机6379关闭后,检查是否进行推选.



## Spring整合哨兵

### 入门案例

@Test

**public** **void** test01(){

Set<String> sentinels = **new** HashSet<>();

sentinels.add("192.168.126.166:26379");

JedisSentinelPool sentinelPool =

**new** JedisSentinelPool("mymaster", sentinels);

Jedis jedis = sentinelPool.getResource();

jedis.set("name", "tomcat猫");

System.***out***.println(jedis.get("name"));

}

### 编辑properties文件

redis.sentinel=192.168.126.166:26379

redis.sentinel.master=mymaster

redis.maxTotal=1000

redis.maxIdle=100

redis.testOnBorrow=true

### 编辑Spring配置文件

<!--1.定义poolConfigBean -->

<bean id=*"poolConfig"* class=*"redis.clients.jedis.JedisPoolConfig"*>

<property name=*"maxTotal"* value=*"${redis.maxTotal}"*/>

<property name=*"maxIdle"* value=*"${redis.maxIdle}"*/>

<property name=*"testOnBorrow"* value=*"${redis.testOnBorrow}"*/>

</bean>

<!--定义哨兵连接池对象 -->

<bean id=*"jedisSentinelPool"* class=*"redis.clients.jedis.JedisSentinelPool"*>

<constructor-arg name=*"masterName"* value=*"${redis.sentinel.master}"*/>

<constructor-arg name=*"sentinels"*>

<set>

<value>${redis.sentinel}</value>

</set>

</constructor-arg>

<constructor-arg name=*"poolConfig"* ref=*"poolConfig"*/>

</bean>

### 编辑工具类

@Service

**public** **class** RedisService {

//注入分片的池对象

@Autowired(required=**false**) //使用时才注入对象

//public ShardedJedisPool shardedJedisPool;

**private** JedisSentinelPool jedisSentinelPool; //连接哨兵

**public** **void** set(String key,String value){

Jedis jedis = jedisSentinelPool.getResource();

jedis.set(key, value);

jedis.close();

}

**public** String get(String key){

Jedis jedis = jedisSentinelPool.getResource();

String result = jedis.get(key);

jedis.close();

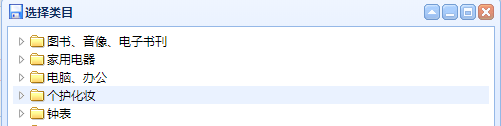
**return** result;

}

}

修改工具类后,将项目打包

### 效果展现



## Redis中持久化策略

### Redis中持久化策略说明

说明:因为redis中保存的数据都在内存中,当断电/宕机.缓存中的数据都会被清空.如果redis中没有配置持久化策略,安全性不够完善.

策略说明:

1. RDB方式

该方式是redis默认选择的持久化策略

特点:持久化的效率更高,定期持久化可能会丢失数据

1. AOF方式

该方式需要通过配置文件手动开启

特点:持久化效率低,每秒持久化/每次操作持久化,保证数据尽可能不丢失

持久化步骤:

1. 当用户set操作时,redis中的数据会新增/更新
2. 这时根据用户选择的持久化的策略.自动的进行数据持久化操作.以下以RDB模式为例.
3. **定期**会将redis中全部的数据通过xxx.RDB文件的方式保存.
4. 如果redis服务器宕机重启时,首先会加载持久化文件xxx.RDB.恢复内存中的数据.
5. 当用户使用redis时,这时redis内存中已经恢复了数据,为用户继续提供服务.

### RDB模式

1. RDB持久化策略

save 900 1

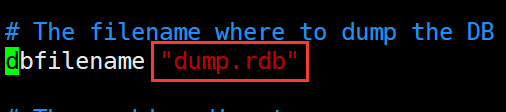
save 300 10

save 60 10000

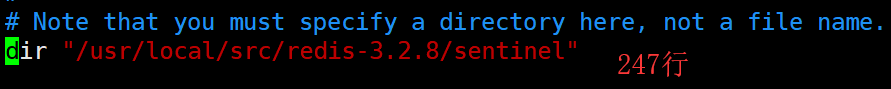
1. 在15分钟内,如果用户执行了一次set操作则持久化一次
2. 在5分钟内,如果用户执行了10次set操作则持久化一次
3. 在5分钟内,如果用户执行了10000set操作则持久化一次

1. 持久化文件名称配置

在237行可以修改持久化文件的名称



1. 修改持久化文件的位置



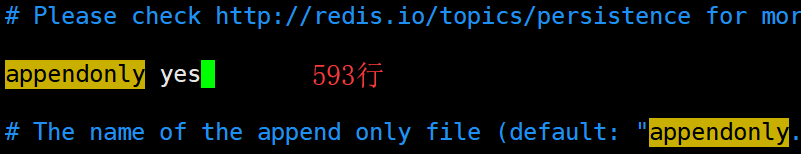
总结:

如果使用redis服务时,如果允许丢失小部分数据,则使用RDB模式,因为它的效率是最高的.

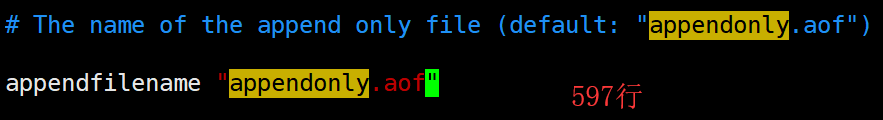
### AOF模式

1. 开启AOF模式

说明:如果在配置文件中开启AOF模式,则redis中的RDB模式将不生效.



1. 持久化文件名称



1. 持久化策略

说明:

AOF模式是记录用户的执行的过程.将用户的全部的操作步骤,以文件的形式进行记录.当redis服务器重新启动时,会根据AOF文件中的步骤,重新执行一次.最终实现数据的恢复.

appendfsync always

appendfsync everysec

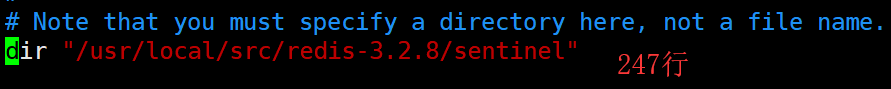
appendfsync no

always:用户每次操作都是追加到aof文件中

everysec:每秒记录用户的操作步骤

no:不记录.

1. 持久化文件位置



总结:

AOF模式相当于记录了用户的执行过程.从而实现了数据的持久化.默认的条件下AOF模式采用每秒备份.保证数据的有效性.但是效率低于RDB.

AOF文件较大,需要的执行时间较长.

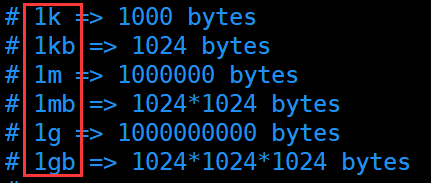
## Redis中内存策略

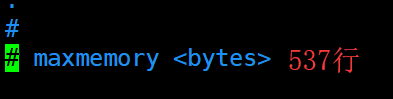
### 需求说明

因为redis中的数据全部都写入内存中.当redis中的数据不能再次写入时这时redis服务会有问题.所以应该保证redis服务永远可写.如何实现redis数据内存优化呢??

### Redis中内存策略说明

1. 定义redis中最大内存





1. LRU算法:

[内存管理](https://baike.baidu.com/item/%E5%86%85%E5%AD%98%E7%AE%A1%E7%90%86/5633616)的一种页面置换算法，对于在内存中但又不用的[数据块](https://baike.baidu.com/item/%E6%95%B0%E6%8D%AE%E5%9D%97/107672)（内存块）叫做LRU，操作系统会根据哪些数据属于LRU而将其移出内存而腾出空间来加载另外的数据。

1. 内存优化手段:

volatile-lru -> 将设定超时时间的数据并且其中使用较少的数据进行删除

allkeys-lru -> 将redis中全部key进行LRU筛选,之后进行删除

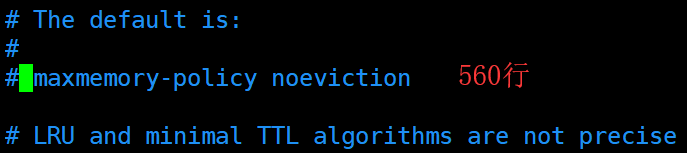
volatile-random -> 设定了超时间的数据,随机删除

allkeys-random -> 全部的key随机删除

volatile-ttl -> 将已经设定了超时时间的数据,按照存活时间排序,将马上要过期的数据进行删除.

noeviction -> 不做任何操作,将报错信息返回给用户.

1. 修改优化策略



# Redis集群搭建

## 分片和哨兵说明

### 特点介绍

优点:

1. 分片可以实现内存的动态的扩容
2. 使用分片能够将数据分开保存,如果宕机/数据损坏只丢失1/n的数据
3. 哨兵可以实现redis的高可用.

缺点:

1. 如果分片中的一个节点宕机,那么整个分片将不能正常运行.
2. 分片中的数据因为hash一致性的原因.可能会导致数据分布不是特别的平均.
3. 如果哨兵宕机,则整个服务陷入瘫痪.

## Redis集群实现

### Redis集群特点

说明:redis集群实质上将redis分片和redis哨兵的机制进行整合.redis集群中每个节点多可以与其他节点进行通讯.同时集群内部有心跳检测.如果节点发生宕机的现象.由所在集群的全部服务器负责推选.保证服务的正常运行.如果全部的从节点宕机,并且这时主节点宕机那么整个集群才会奔溃.

### 集群规模定义

原则:一般集群中的主节点的数量一定是奇数个.目的防止出现平票现象

规模:

3主 6从 redis节点7000-7008

## 搭建步骤

### 创建文件夹

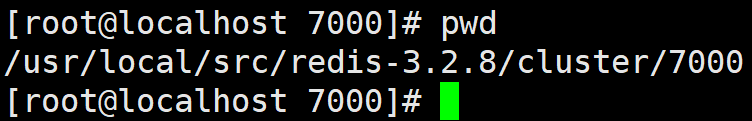
在redis根目录中创建cluster文件夹后新增7000-7008文化夹



### 导入配置文件

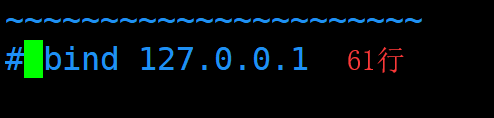
将redis.conf文件分别导入7000-7008文件夹中.

[root@localhost redis-3.2.8]# cp redis.conf cluster/7008/

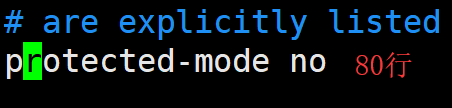


### 修改redis.conf配置文件

1. 关闭ip绑定



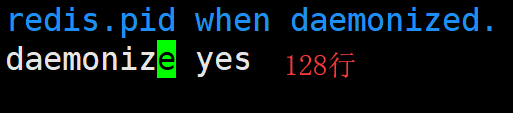
1. 关闭保护模式



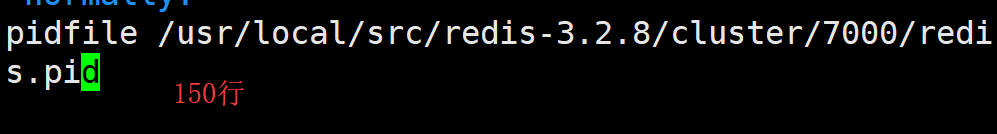
1. 修改端口号



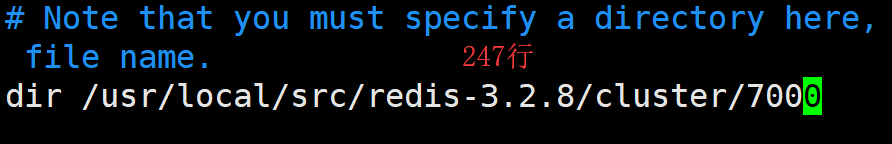
1. 开启后台启动



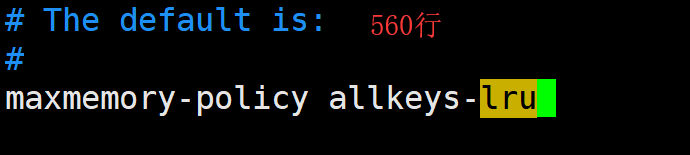
1. 修改PID位置



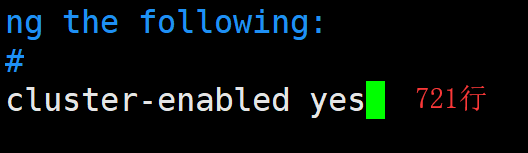
1. 修改持久化文件路径



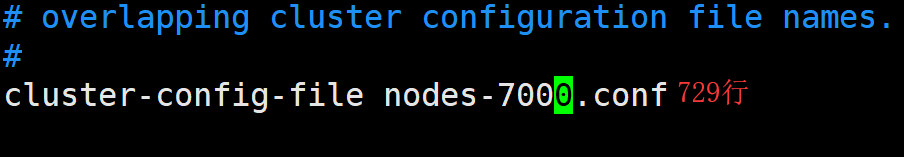
1. 修改内存策略



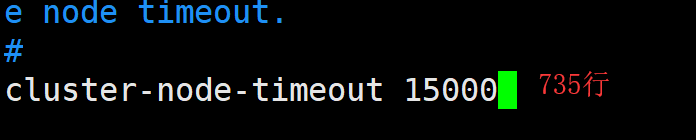
1. 开启集群



1. 开启集群的日志文件



10.设定推选时间



### 拷贝配置文件

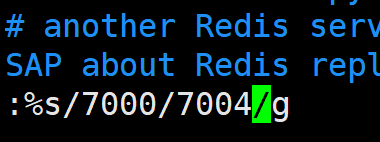
说明:将redis中的7000下的redis.conf文件导入7001-7008中

cp 7000/redis.conf 7003/………

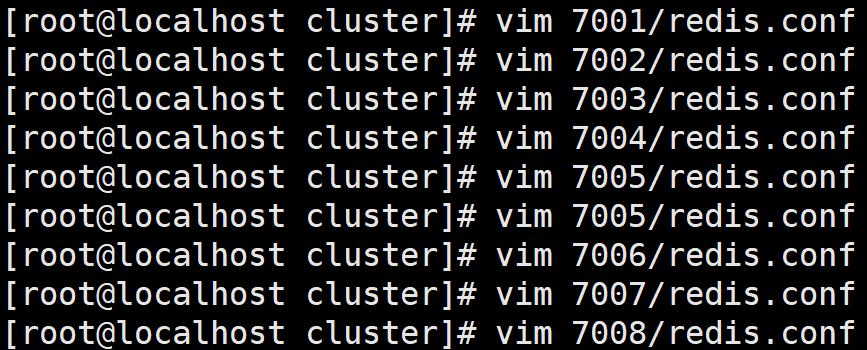
### 批量修改配置文件

在vim命令下执行

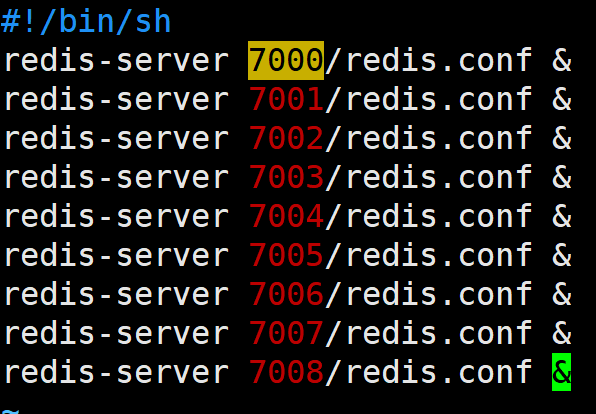
命令: **:%s/7000/7001/g**



分别修改配置文件



### 编辑redis启动脚本文件



命令:启动redis

sh start.sh

作业:自己完成redis停止的脚本

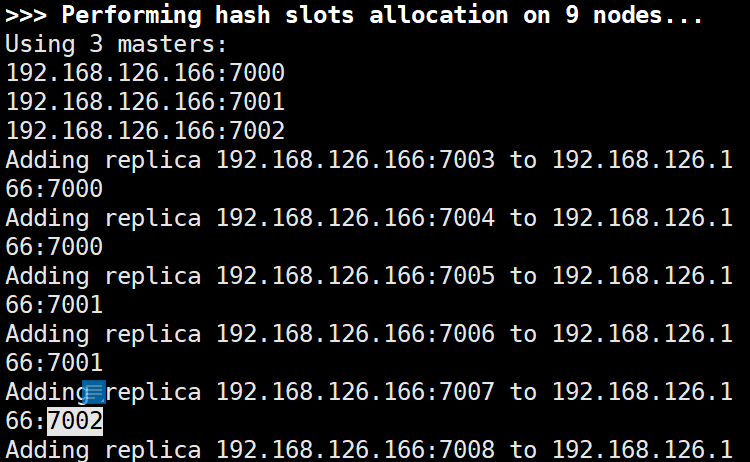
redis-cli -p 7000 shutdown

### 实现集群搭建

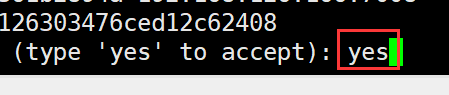
说明:使用ruby工具管理redis集群.

./src/redis-trib.rb create --replicas 2 192.168.126.166:7000 192.168.126.166:7001 192.168.126.166:7002 192.168.126.166:7003 192.168.126.166:7004 192.168.126.166:7005 192.168.126.166:7006 192.168.126.166:7007 192.168.126.166:7008

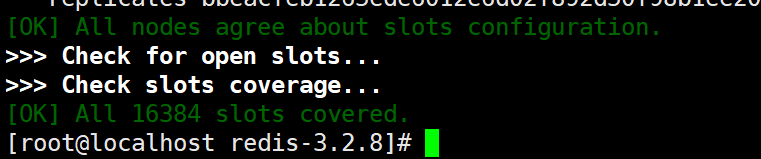
replicas 2:代表一个主节点有2台从机



输入yes



搭建成功:



### 作业:

预习:1.集群操作API

预习:2 spring整合redis集群 (根据API实现)

**预习:3 如何简化spring配置项实现集群访问**