# Redis高级

## Redis主从同步

### 高可用介绍(HA)

“高可用性”（High Availability）通常来描述一个系统经过专门的设计，从而减少停工时间，而保持其服务的高度可用性。

概括:利用技术手段实现了当服务器宕机,自动的实现故障的迁移.

### 业务说明

Redis的最终形态必须实现高可用,实现高可用的前提必须满足**主从同步**.

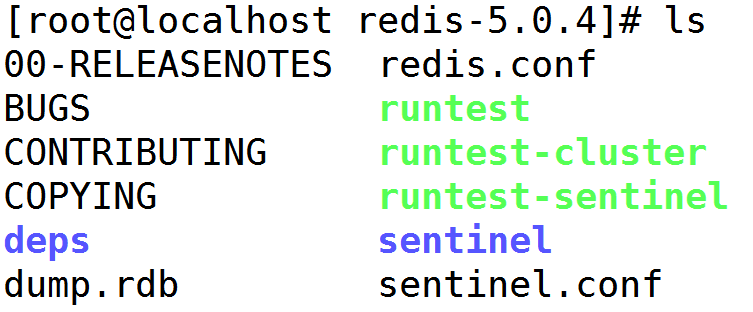
当发生故障,由于从机与主机的数据是相同的,所以可以非常灵活实现数据的故障迁移.

## 主从同步搭建

### 复制文件

说明:将分片文件夹shards复制为sentinel.

cp –r shards sentinel



### 检查主从关系

进入redis-6379客户端



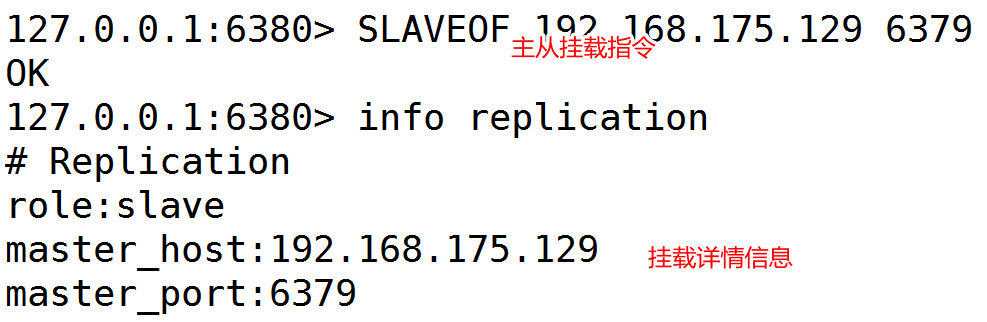
### 实现主从挂载

说明:6379当主机, 6380/6381当从机.

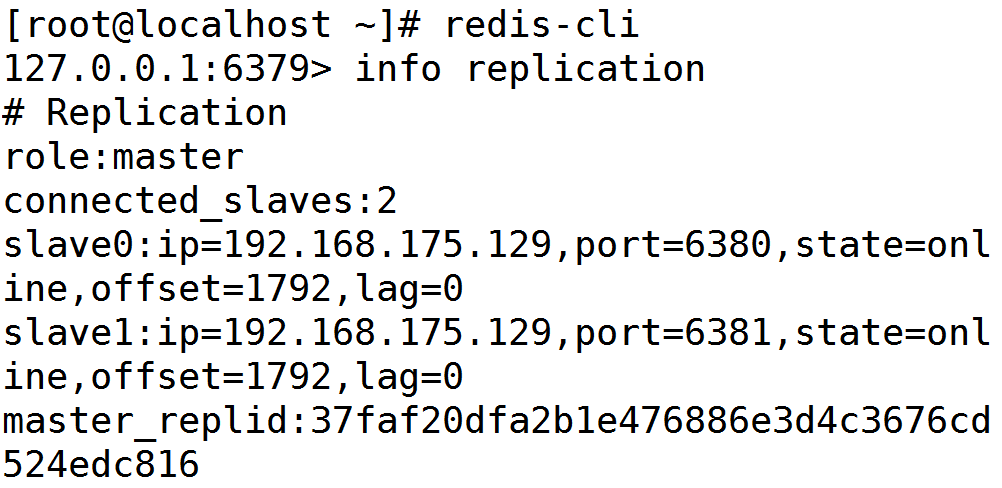
命令:在客户端中cli端 执行指令

saveof IP 端口





2.挂载情况



3.主从测试

当在6379中写入数据时,从机80/81也同步了数据.

### 主从同步持久性

问题说明:通过slaveof指令可以指定主从关系.如果从机关闭之后重启问情况如何??

回答:

当6381重启之后,发现主从结构不存在了.6381变为主机.

如何解决该问题:

需求:当6381为从机时,当重启服务器依然是从机.

需要人为的动态修改redis.conf配置文件.

## Redis哨兵实现高可用

### Redis中哨兵的作用

分片作用:redis分片实现了redis内存扩容.

Redis哨兵:主要实现了redis节点的高可用.

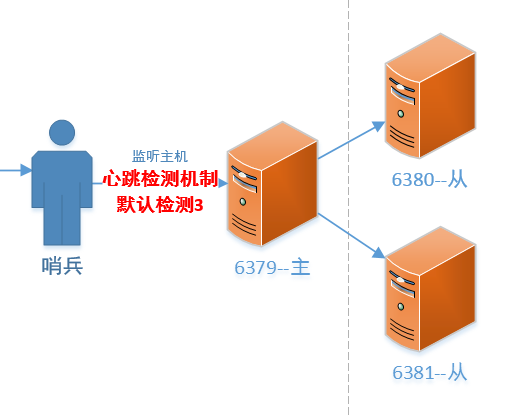
### Redis哨兵实现步骤

1. redis哨兵会监听redis主节点.

目的1:检查主节点是否存活

目的2:获取连接主节点的从机. IP:端口

1. 当利用ping-pong检测机制.检查主节点是否存活,当哨兵连续3次检测都没有数据返回.则表明主节点宕机.
2. 哨兵根据从主节点获取的从节点信息,进行推选.从中挑选一台新的从节点当做现在的主节点.将新的主从关系写入其他节点的redis.conf文件中.
3. 当服务器重启后,能够了解当前主从关系,实现了redis高可用.



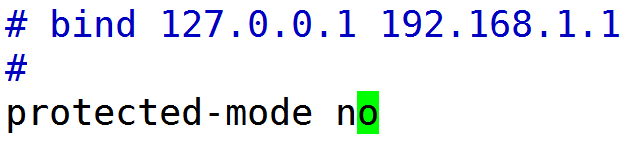
## Redis哨兵配置

### 复制哨兵配置文件

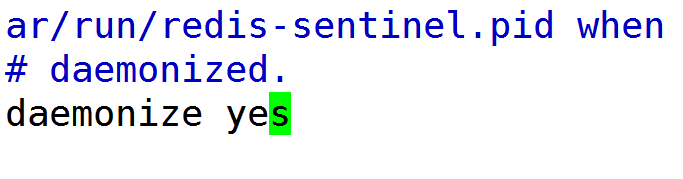
cp sentinel.conf sentinel/

### 编辑sentinel配置文件

1. 关闭保护模式 17行



2.开启哨兵后台启动 26行



3.修改哨兵监控 84行

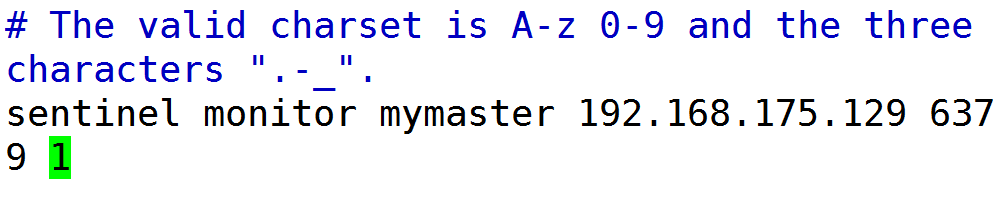
sentinel monitor mymaster 127.0.0.1 6379 1

mymaster:当前主机的变量

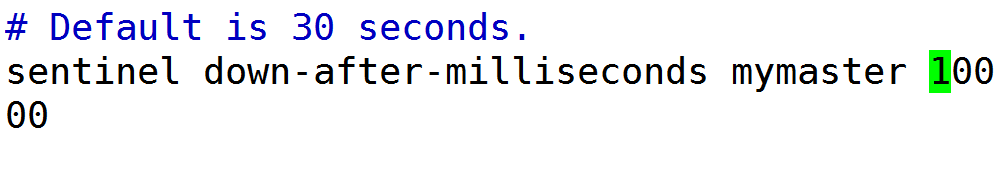
IP:端口: redis主节点的IP和端口

投票生效票数:

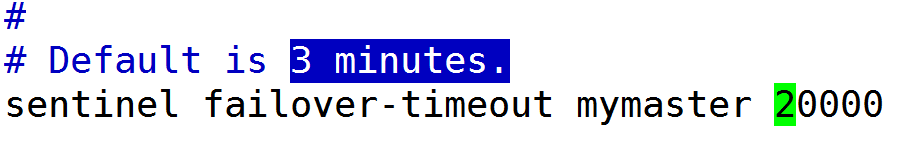
规则:超半数即可,哨兵的数量是奇数.



4.主机宕机后多久进行推选.



5.修改哨兵推选失败超时时间.

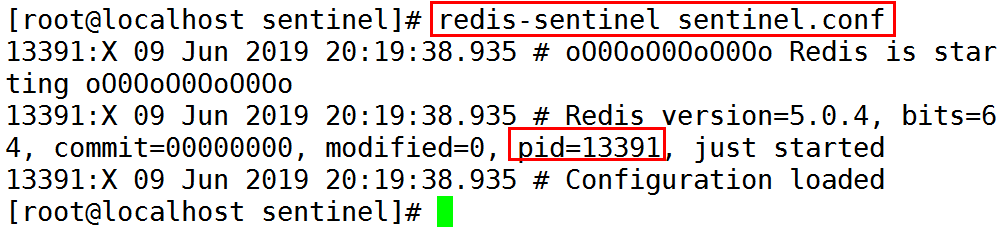


### Redis哨兵测试

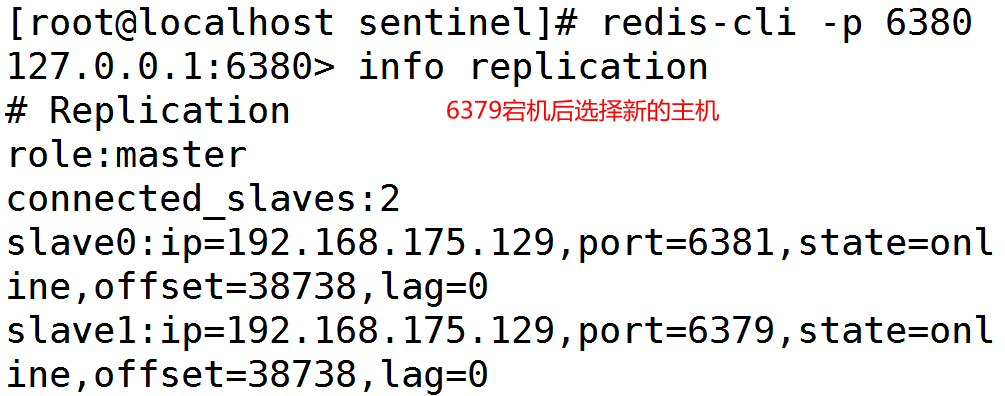
业务说明:

1. 启动redis哨兵

redis-sentinel sentinel.conf



1. 当关闭redis-6379之后等10秒之后.检查主从结构是否正确.



## Spring整合哨兵

### 入门案例

//测试哨兵get/set操作

@Test

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

//masterName 代表主机的变量名称

//sentinels Set<String> IP:端口

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

sentinels.add("192.168.175.129:26379");

JedisSentinelPool sentinelPool =

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

Jedis jedis = sentinelPool.getResource();

jedis.set("cc", "端午节过后没假了!!!!");

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

jedis.close(); //关闭链接

}

### Spring整合哨兵思路

1. 利用配置类,封装JedisSentinelPool.
2. 利用工具API封装2个方法.set/get

### 编辑pro文件

redis.sentinels=192.168.175.129:26379

redis.sentinel.masterName=mymaster

### 编辑配置类

//表示redis配置类

@Configuration //xml

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

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

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

**private** String jedisSentinelNodes;

@Value("${redis.sentinel.masterName}")

**private** String masterName;

@Bean

**public** JedisSentinelPool jedisSentinelPool() {

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

sentinels.add(jedisSentinelNodes);

**return** **new** JedisSentinelPool(masterName, sentinels);

}

}

### 编辑工具API

说明:在jt-common中编辑操作redis哨兵的工具API.其中注入哨兵池对象

@Service

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

@Autowired(required = **false**)//调用时才注入

**private** JedisSentinelPool sentinelPool;

//封装方法 get

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

Jedis jedis = sentinelPool.getResource();

String result = jedis.get(key);

jedis.close();

**return** result;

}

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

Jedis jedis = sentinelPool.getResource();

jedis.set(key, value);

jedis.close();

}

**public** **void** setex(String key,**int** seconds,String value) {

Jedis jedis = sentinelPool.getResource();

jedis.setex(key, seconds, value);

jedis.close();

}

}

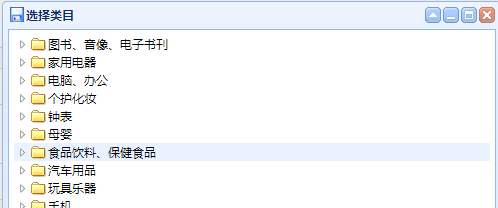
### 编辑RedisAOP配置项

在redis AOP中注入工具API redisService.



### 项目测试

说明:重启tomcat服务器.检查程序调用是否正确.当redis节点宕机时.检查是否影响用户操作redis.



## Redis集群

### 知识回顾

1. Redis分片

用户通过API利用hash一致性算法,实现了数据存储.利用分片机制实现了内存的扩容!!!

缺点:如果一个节点宕机.,则违反单调性要求,分片失效.

1. Redis哨兵

哨兵基于心跳检测机制.实现redis节点高可用.但是前提必须配置主从.

哨兵缺点:

1. 操作的redis依然是单台,内存无法扩容.
2. Redis哨兵也有可能宕机.

### 问题总结

分片:没有高可用效果.

哨兵:哨兵本身没有高可用需要借助第三方. 内存没有扩容.

### Redis集群介绍

说明:搭建redis集群可以实现redis分片和哨兵的全部功能!!!.Redis集群中所有的**主节点参与选举**.redis集群中全部的节点都能互相通信.所有的节点都有投票权(发现某个节点宕机投票数+1)

## 集群搭建

### 集群设计

最小的集群单位3个主节点,从机不做要求,搭建策略一主一从

主机3台

端口:7000/7001/7002

从机3台

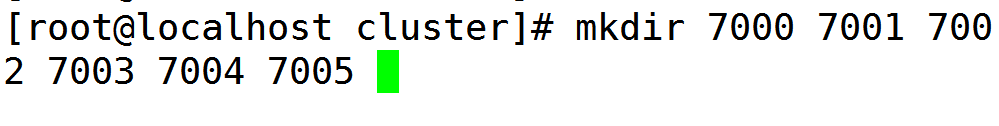
端口:7003/7004/7005

### 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

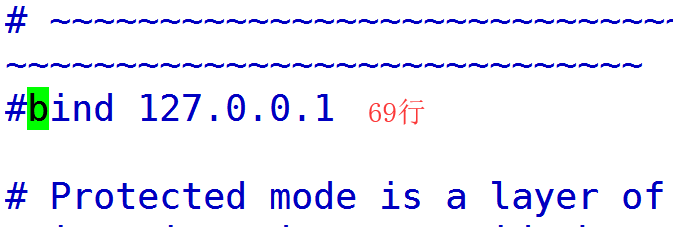
说明:

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

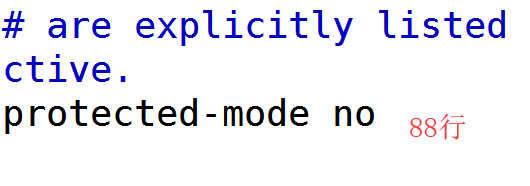
cp redis.conf cluster/7000/

### 编辑配置文件

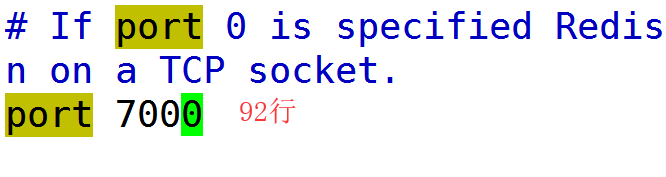
1. 注释本地绑定IP地址



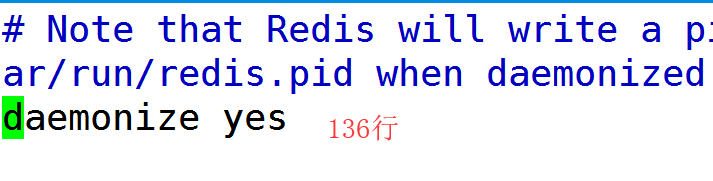
1. 关闭保护模式



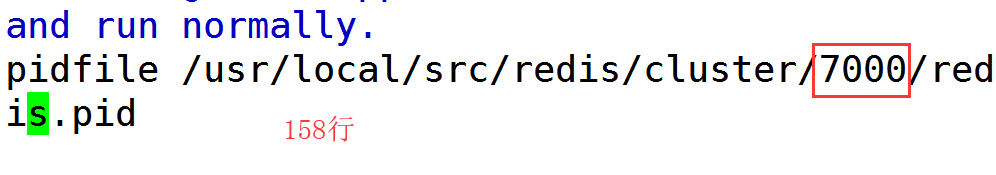
1. 修改端口号



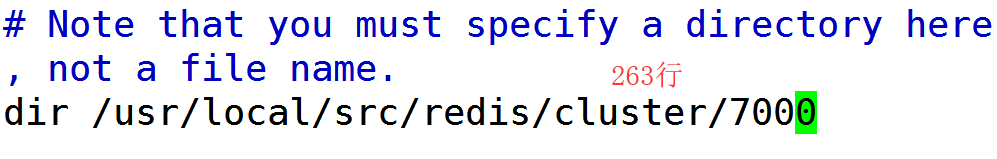
1. 启动后台启动



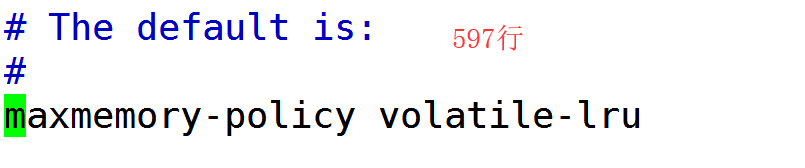
1. 修改pid文件



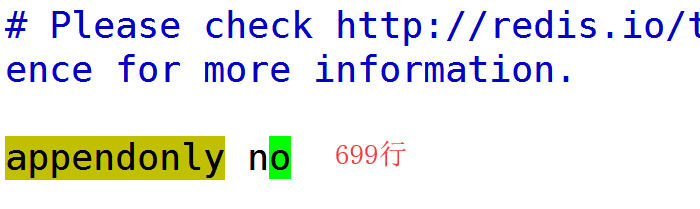
1. 修改持久化文件路径



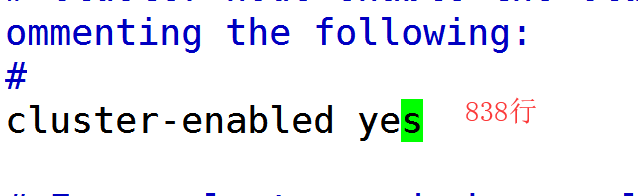
1. 设定内存优化策略



1. 关闭AOF模式

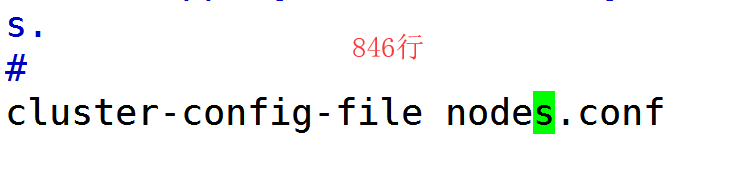


1. 开启集群配置

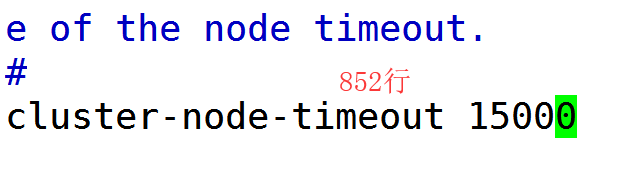


1. 开启集群配置文件

保存集群中的状态信息.谁是主机,谁是从机.



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/

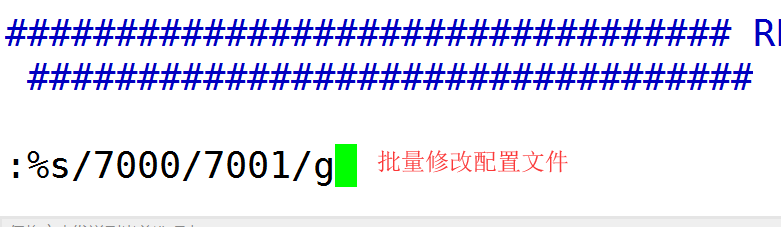
[root@localhost cluster]# cp 7000/redis.conf 7004/

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

### 批量修改

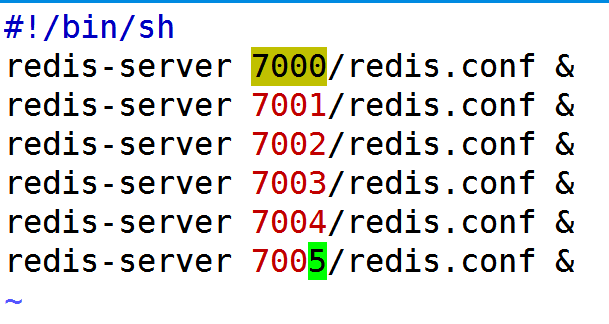
说明:分别将7001-7005文件中的7000改为对应的端口号的名称,

修改时注意方向键的使用

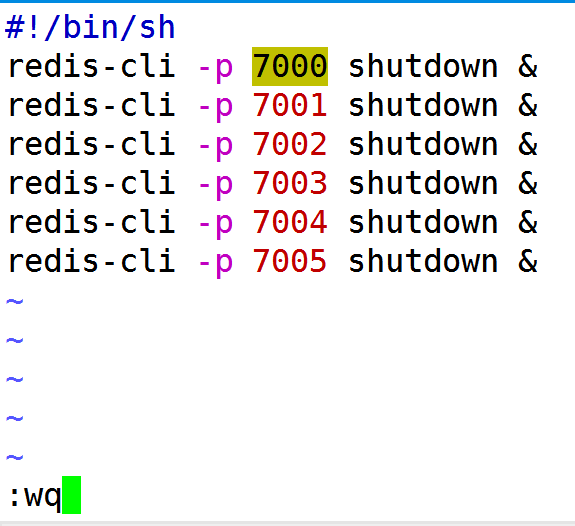


### 通过脚本编辑启动/关闭指令

1. 创建启动脚本 vim start.sh



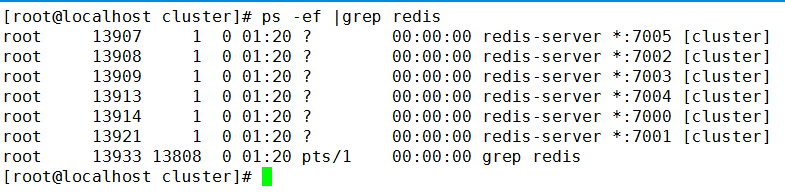
1. 编辑关闭的脚本 vim shutdown.sh



1. 启动redis节点

sh start.sh

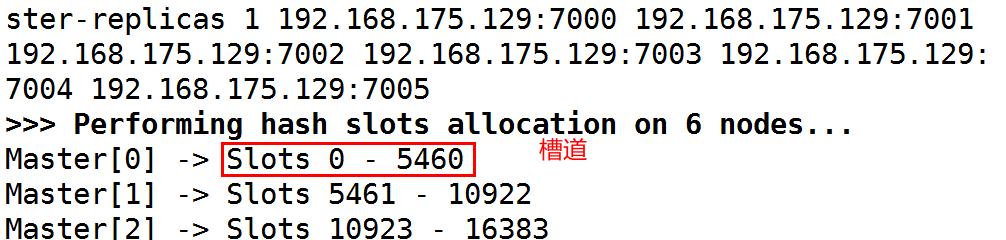
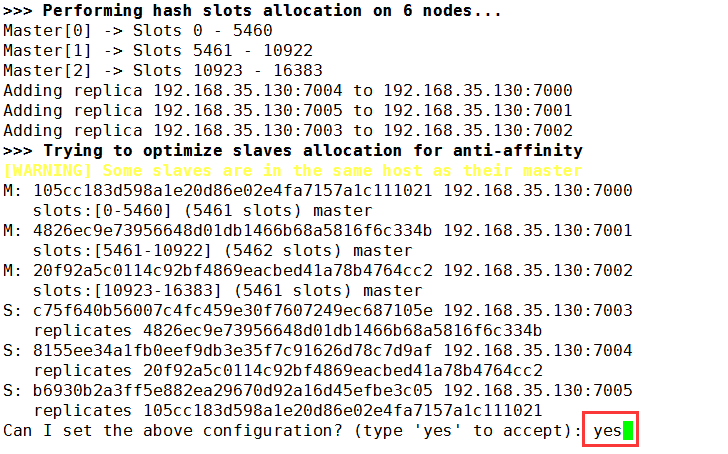
1. 检查redis节点启动是否正常



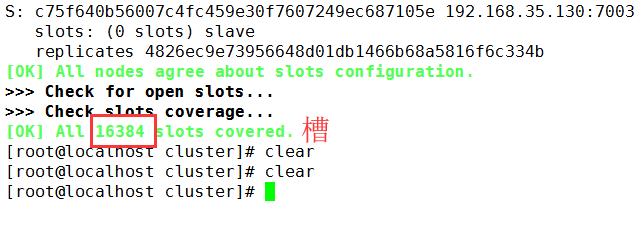
### 创建redis集群

#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

