**redis集群**

**简介**

redis集群是一个无中心的分布式Redis存储**[架构](http://lib.csdn.net/base/architecture" \t "_blank" \o "大型网站架构知识库)**，可以在多个节点之间进行数据共享，解决了Redis高可用、可扩展等问题。redis集群提供了以下两个好处

1、将数据自动切分(split)到多个节点

2、当集群中的某一个节点故障时，redis还可以继续处理客户端的请求。

一个 Redis 集群包含 16384 个哈希槽（hash slot），数据库中的每个数据都属于这16384个哈希槽中的一个。集群使用公式 CRC16(key) % 16384 来计算键 key 属于哪个槽。集群中的每一个节点负责处理一部分哈希槽。

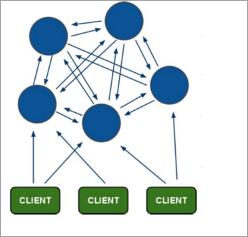
集群中的主从复制

集群中的每个节点都有1个至N个复制品，其中一个为主节点，其余的为从节点，如果主节点下线了，集群就会把这个主节点的一个从节点设置为新的主节点，继续工作。这样集群就不会因为一个主节点的下线而无法正常工作。

注意：

1、如果某一个主节点和他所有的从节点都下线的话，redis集群就会停止工作了。redis集群不保证数据的强一致性，在特定的情况下，redis集群会丢失已经被执行过的写命令

2、使用异步复制（asynchronous replication）是 Redis 集群可能会丢失写命令的其中一个原因，有时候由于网络原因，如果网络断开时间太长，redis集群就会启用新的主节点，之前发给主节点的数据就会丢失。



**安装配置**

修改配置文件redis.conf

**[python]** [view plain](http://blog.csdn.net/u011204847/article/details/51307044) [copy](http://blog.csdn.net/u011204847/article/details/51307044)

1. daemonize yes
2. port 6379
3. cluster-enabled yes
4. cluster-config-file nodes.conf
5. cluster-node-timeout 5000

要让集群正常运作至少需要三个主节点

我们这里就简单在一台主机上创建6个redis节点来演示集群配置，实际生产环境中需要每个节点一台主机。

我们要创建的6个redis节点，其中三个为主节点，三个为从节点，对应的redis节点的ip和端口对应关系如下：

**[python]** [view plain](http://blog.csdn.net/u011204847/article/details/51307044) [copy](http://blog.csdn.net/u011204847/article/details/51307044)

1. 192.168.33.130:7000
2. 192.168.33.130:7001
3. 192.168.33.130:7002
4. 192.168.33.130:7003
5. 192.168.33.130:7004
6. 192.168.33.130:7005

1、首先我们创建6个以端口为名称的文件夹（由于每个redis节点启动的时候，都会在当前文件夹下创建快照文件，所以我们需要创建每个节点的启动目录）

**[python]** [view plain](http://blog.csdn.net/u011204847/article/details/51307044) [copy](http://blog.csdn.net/u011204847/article/details/51307044)

1. mkdir 7000
2. mkdir 7001
3. mkdir 7002
4. mkdir 7003
5. mkdir 7004
6. mkdir 7005

2、接下来把每个节点启动所需要的配置文件拷贝到相应的启动目录：

**[python]** [view plain](http://blog.csdn.net/u011204847/article/details/51307044) [copy](http://blog.csdn.net/u011204847/article/details/51307044)

1. cp redis.conf  7000
2. cp redis.conf  7001
3. cp redis.conf  7002
4. cp redis.conf  7003
5. cp redis.conf  7004
6. cp redis.conf  7005

3、然后我们进入每个启动目录，修改之前拷贝的redis.conf文件中的端口port 为上面列出的对应端口。

最终每个节点的配置类似于：

**[python]** [view plain](http://blog.csdn.net/u011204847/article/details/51307044) [copy](http://blog.csdn.net/u011204847/article/details/51307044)

1. daemonize yes
2. port 6379     #只有端口不同，其他相同
3. cluster-enabled yes
4. cluster-config-file nodes.conf
5. cluster-node-timeout 5000

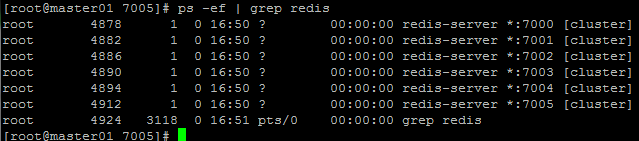
4、进入每个启动目录，以每个目录下的redis.conf文件启动



使用命令查看redis节点是否启动

**[python]** [view plain](http://blog.csdn.net/u011204847/article/details/51307044) [copy](http://blog.csdn.net/u011204847/article/details/51307044)

1. ps -ef | grep redis



5、创建集群命令

**[python]** [view plain](http://blog.csdn.net/u011204847/article/details/51307044) [copy](http://blog.csdn.net/u011204847/article/details/51307044)

1. redis-trib.rb  create --replicas 1 192.168.33.130:7000 192.168.33.130:7001 192.168.33.130:7002 192.168.33.130:7003 192.168.33.130:7004 192.168.33.130:7005

# 2 ZooKeeper的工作原理

Zookeeper的核心是原子广播，这个机制保证了各个Server之间的同步。实现这个机制的协议叫做Zab协议。Zab协议有两种模式，它们分别是恢复模式（选主）和广播模式（同步）。当服务启动或者在领导者崩溃后，Zab就进入了恢复模式，当领导者被选举出来，且大多数Server完成了和leader的状态同步以后，恢复模式就结束了。状态同步保证了leader和Server具有相同的系统状态。

为了保证事务的顺序一致性，zookeeper采用了递增的事务id号（zxid）来标识事务。所有的提议（proposal）都在被提出的时候加上了zxid。实现中zxid是一个64位的数字，它高32位是epoch用来标识leader关系是否改变，每次一个leader被选出来，它都会有一个新的epoch，标识当前属于那个leader的统治时期。低32位用于递增计数。

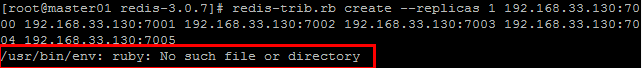
每个Server在工作过程中有三种状态：

* LOOKING：当前Server不知道leader是谁，正在搜寻
* LEADING：当前Server即为选举出来的leader
* FOLLOWING：leader已经选举出来，当前Server与之同步

注意：

5.1、执行上面的命令的时候可能会报错，因为是执行的ruby的脚本，需要ruby的环境

错误内容：

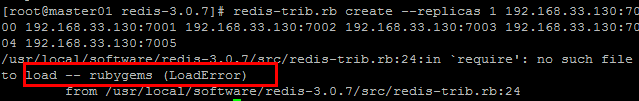


所以我们需要安装ruby的环境，这里推荐使用yum安装：

**[python]** [view plain](http://blog.csdn.net/u011204847/article/details/51307044) [copy](http://blog.csdn.net/u011204847/article/details/51307044)

1. yum install ruby

5.2、安装ruby后，执行命令可能还会报错，提示缺少rubygems组件，使用yum安装

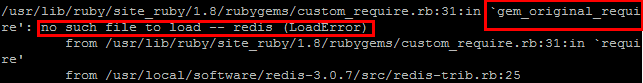


解决方法：

**[python]** [view plain](http://blog.csdn.net/u011204847/article/details/51307044) [copy](http://blog.csdn.net/u011204847/article/details/51307044)

1. yum install rubygems

5.3、上面两个步骤后，执行创建集群目录可能还会报错，提示不能加载redis，是因为缺少redis和ruby的接口，使用gem 安装。

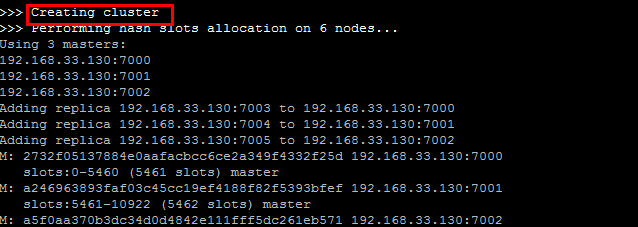


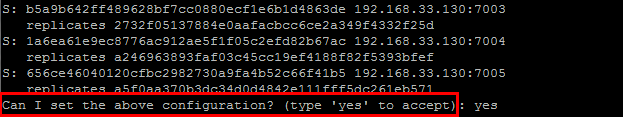
解决方法：

**[python]** [view plain](http://blog.csdn.net/u011204847/article/details/51307044) [copy](http://blog.csdn.net/u011204847/article/details/51307044)

1. gem install redis

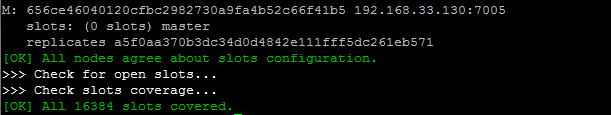
上面三个问题解决后，启动创建集群应该可以正常启动了：





这里输入yes

最后结果：



到此，我们的集群搭建成功了。

6、接下来我们使用命令进入集群环境

**[python]** [view plain](http://blog.csdn.net/u011204847/article/details/51307044) [copy](http://blog.csdn.net/u011204847/article/details/51307044)

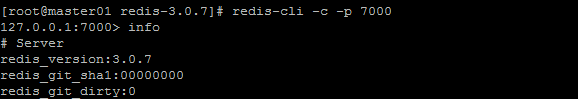
1. redis-cli -c -p 7000

**redis集群操作**

使用redis-cli客户端来操作redis集群，使用命令 ：

**[python]** [view plain](http://blog.csdn.net/u011204847/article/details/51307044) [copy](http://blog.csdn.net/u011204847/article/details/51307044)

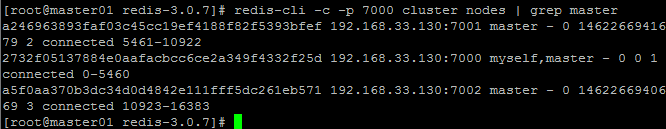
1. redis-cli -c  -p [port]



查看集群中的所有主节点信息

**[python]** [view plain](http://blog.csdn.net/u011204847/article/details/51307044) [copy](http://blog.csdn.net/u011204847/article/details/51307044)

1. redis-cli -c -p 7000 cluster nodes [| grep master]



**redis集群添加节点**

根据添加节点类型的不同，有两种方法来添加新节点

1、主节点：如果添加的是主节点，那么我们需要创建一个空节点，然后将某些哈希槽移动到这个空节点里面

2、从节点：如果添加的是从节点，我们也需要创建一个空节点，然后把这个新节点设置成集群中某个主节点的复制品。

**添加节点：**

1、首先把需要添加的节点启动

创建7006目录，拷贝7000中的redis.conf到7006中，然后修改端口port为7006，修改好后进入7006目录启动这个节点：

**[python]** [view plain](http://blog.csdn.net/u011204847/article/details/51307044) [copy](http://blog.csdn.net/u011204847/article/details/51307044)

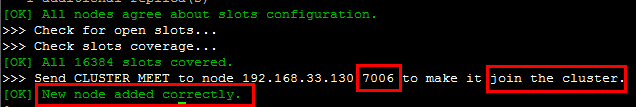
1. redis-server redis.conf

2、执行以下命令，将这个新节点添加到集群中：

**[python]** [view plain](http://blog.csdn.net/u011204847/article/details/51307044) [copy](http://blog.csdn.net/u011204847/article/details/51307044)

1. redis-trib.rb add-node 192.168.33.130:7006 192.168.33.130:7000

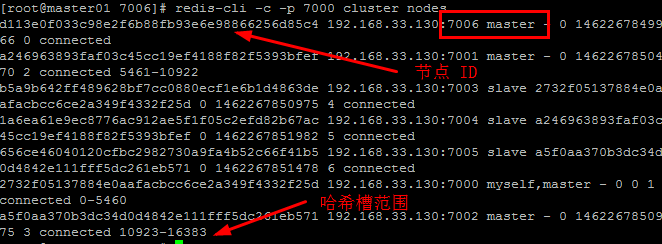
结果图示：



3、执行命令查看刚才新增的节点：

**[python]** [view plain](http://blog.csdn.net/u011204847/article/details/51307044) [copy](http://blog.csdn.net/u011204847/article/details/51307044)

1. redis-cli -c -p 7000 cluster nodes



4、增加了新的节点之后，这个新的节点可以成为主节点或者是从节点

4.1将这个新增节点变成从节点

前面我们已经把这个新节点添加到集群中了，现在我们要让新节点成为192.168.33.130:7001的从节点，只需要执行下面的命令就可以了，命令后面的节点ID就是192.168.33.130:7001的节点ID。（注意，这个从节点哈希槽必须为空，如果不为空，则需要转移掉哈希槽使之为空）

**[python]** [view plain](http://blog.csdn.net/u011204847/article/details/51307044) [copy](http://blog.csdn.net/u011204847/article/details/51307044)

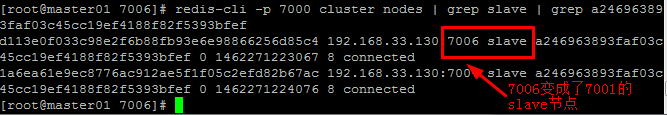
1. redis-cli -c -p 7006 cluster replicate a246963893faf03c45cc19ef4188f82f5393bfef

http://img.blog.csdn.net/20160503194045078

使用下面命令来确认一下192.168.33.130:7006是否已经成为192.168.33.130:7001的从节点。

**[python]** [view plain](http://blog.csdn.net/u011204847/article/details/51307044) [copy](http://blog.csdn.net/u011204847/article/details/51307044)

1. redis-cli -p 7000 cluster nodes | grep slave | grep a246963893faf03c45cc19ef4188f82f5393bfef



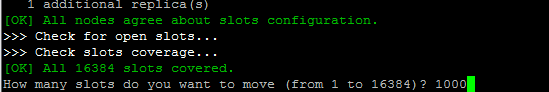
4.2、将这个新增节点变成主节点：

使用redis-trib程序，将集群中的某些哈希槽移动到新节点里面，这个新节点就成为真正的主节点了。执行下面的命令对集群中的哈希槽进行移动：

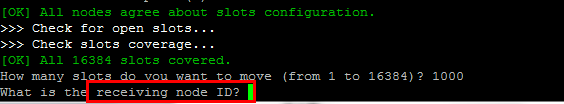
**[python]** [view plain](http://blog.csdn.net/u011204847/article/details/51307044) [copy](http://blog.csdn.net/u011204847/article/details/51307044)

1. redis-trib.rb reshard 192.168.33.130:7000

命令执行后，系统会提示我们要移动多少哈希槽，这里移动1000个



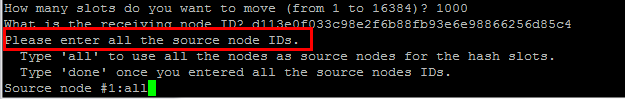
然后还需要指定把这些哈希槽转移到哪个节点上



输入我们刚才新增的节点的ID

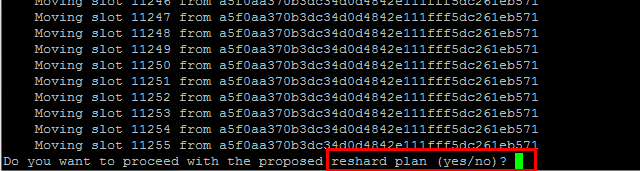
d113e0f033c98e2f6b88fb93e6e98866256d85c4

然后需要我们指定转移哪几个几点的哈希槽



输入all 表示从所有的主节点中随机转移，凑够1000个哈希槽

然后再输入yes，redis集群就开始分配哈希槽了。

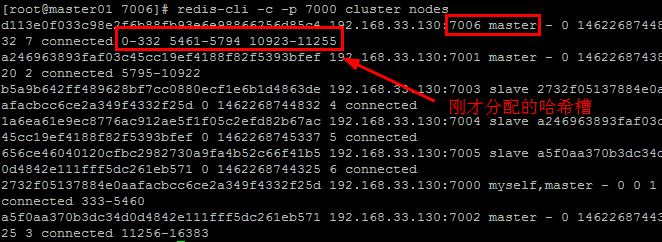


至此，一个新的主节点就添加完成了，执行命令查看现在的集群中节点的状态

**[python]** [view plain](http://blog.csdn.net/u011204847/article/details/51307044) [copy](http://blog.csdn.net/u011204847/article/details/51307044)

1. redis-cli -c -p 7000 cluster nodes

结果图示：



**Redis集群删除节点**

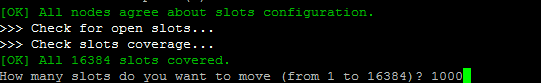
1、如果删除的节点是主节点，这里我们删除192.168.33.130:7006节点，这个节点有1000个哈希槽

首先要把节点中的哈希槽转移到其他节点中，执行下面的命令：

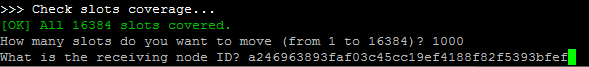
**[python]** [view plain](http://blog.csdn.net/u011204847/article/details/51307044) [copy](http://blog.csdn.net/u011204847/article/details/51307044)

1. redis-trib.rb reshard 192.168.33.130:7000

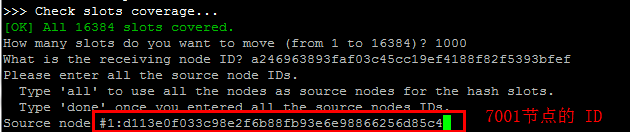
系统会提示我们要移动多少哈希槽，这里移动1000个，因为192.168.33.130:7006节点有1000个哈希槽。



然后系统提示我们输入要接收这些哈希槽的节点的ID，这里使用192.168.33.130:7001的节点ID



然后要我们选择从那些节点中转出哈希槽，这里一定要输入192.168.33.130:7006这个节点的ID

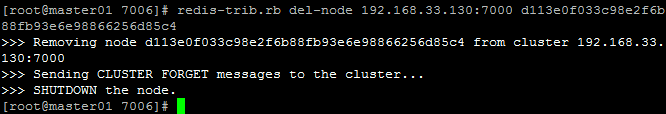


最后输入done表示输入完毕。

最后一步，使用下面的命令把这个节点删除

**[python]** [view plain](http://blog.csdn.net/u011204847/article/details/51307044) [copy](http://blog.csdn.net/u011204847/article/details/51307044)

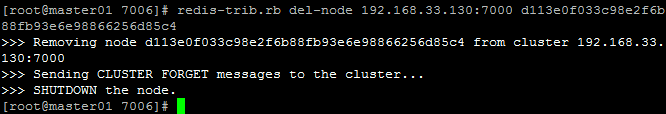
1. redis-trib.rb **del**-node 192.168.33.130:7000 d113e0f033c98e2f6b88fb93e6e98866256d85c4    //最后一个参数为需要删除的节点ID



2、如果是从节点，直接删除即可。

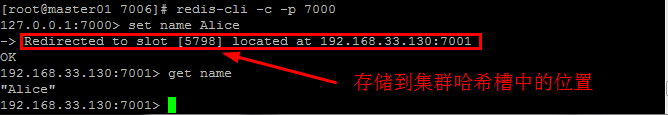
**[python]** [view plain](http://blog.csdn.net/u011204847/article/details/51307044) [copy](http://blog.csdn.net/u011204847/article/details/51307044)

1. redis-trib.rb **del**-node 192.168.33.130:7000 d113e0f033c98e2f6b88fb93e6e98866256d85c4   //最后一个参数为需要删除节点的ID



**java操作redis集群**

向Redis集群中存入键值：



代码示例：

**[java]** [view plain](http://blog.csdn.net/u011204847/article/details/51307044) [copy](http://blog.csdn.net/u011204847/article/details/51307044)

1. **import** java.util.HashSet;
2. //需要再pom.xml中引入jedis依赖
3. **import** redis.clients.jedis.HostAndPort;
4. **import** redis.clients.jedis.JedisCluster;
5. **import** redis.clients.jedis.JedisPool;
6. **import** redis.clients.jedis.JedisPoolConfig;
8. **public** **class** RedisCluster {
9. **public** **static** **void** main(String[] args) {
10. //初始化集合，用于装下面的多个主机和端口
11. HashSet<HostAndPort> nodes = **new** HashSet<HostAndPort>();
13. //创建多个主机和端口实例
14. HostAndPort hostAndPort = **new** HostAndPort("192.168.33.130", 7000);
15. HostAndPort hostAndPort1 = **new** HostAndPort("192.168.33.130", 7001);
16. HostAndPort hostAndPort2 = **new** HostAndPort("192.168.33.130", 7002);
17. HostAndPort hostAndPort3 = **new** HostAndPort("192.168.33.130", 7003);
18. HostAndPort hostAndPort4 = **new** HostAndPort("192.168.33.130", 7004);
19. HostAndPort hostAndPort5 = **new** HostAndPort("192.168.33.130", 7005);
21. //添加多个主机和端口到集合中
22. nodes.add(hostAndPort);
23. nodes.add(hostAndPort1);
24. nodes.add(hostAndPort2);
25. nodes.add(hostAndPort3);
26. nodes.add(hostAndPort4);
27. nodes.add(hostAndPort5);
29. //创建config
30. JedisPoolConfig poolConfig = **new** JedisPoolConfig();
31. //通过config创建集群实例
32. JedisCluster jedisCluster = **new** JedisCluster(nodes,poolConfig);
33. //获取集群中的key为name键的值
34. String str = jedisCluster.get("name");
35. System.out.println(str);
36. }
37. }

打印结果：

