**一、简介**  
  
    　　事先说明一下，本篇文章不涉及对redis-trib.rb源代码的分析，只是从使用的角度来阐述一下，对第一次使用的人来说很重要。redis-trib.rb是redis官方推出的管理redis集群的工具，集成在redis的源码src目录下，是基于redis提供的集群命令封装成简单、便捷、实用的操作工具。这个文件可以在src这个目录下使用，也可以像redis-cli等命令一样，可以拷贝到单独目录，方便使用。redis-trib.rb是redis作者用ruby写成的，为了看懂redis-trib.rb的源码，我还要花一些时间来学学ruby。ruby这门语言我是在老赵的教程里面最早听说的，认为这个语言和我关系不大，当初也就没有在意，自然也就没学了。现在看来不行了，所以我就特意花了一个月的时间学习了一下ruby这门语言，同时也被ruby语言的简洁、明了所吸引。阅读本文需要对redis集群功能有一定的了解。关于redis集群功能的介绍，可以参考本人的另两篇文章，第一篇是《[Redis进阶实践之十一 Redis的Cluster集群搭建](http://www.cnblogs.com/PatrickLiu/p/8458788.html" \t "http://www.cnblogs.com/PatrickLiu/p/_blank)》，第二篇是《[Redis进阶实践之十二 Redis的Cluster集群动态扩容](http://www.cnblogs.com/PatrickLiu/p/8473135.html" \t "http://www.cnblogs.com/PatrickLiu/p/_blank)》。  
  
  
**二、Redis-trib.rb详细介绍**  
  
    　　　　Redis-Trib.rb这个脚本文件的功能很强大，使用起来也很方便，我们就由浅入深来探讨这个脚本文件的使用吧。我们先来看看他的帮助信息吧，这个最简单，学习一门新技术的时候，也应该先看看帮助的东西。

**1、#ruby redis-trib.rb help （Redis-Trib.rb的帮助信息）**

[IMG_256](http://www.cnblogs.com/PatrickLiu/p/javascript:void(0);)

2 3 [root@linux redis]# ruby redis-trib.rb help 4 Usage: redis-trib <command> <options> <arguments ...> 5 6 create host1:port1 ... hostN:portN 7 --replicas <arg> 8 check host:port 9 info host:port10 fix host:port11 --timeout <arg>12 reshard host:port13 --from <arg>14 --to <arg>15 --slots <arg>16 --yes17 --timeout <arg>18 --pipeline <arg>19 rebalance host:port20 --weight <arg>21 --auto-weights22 --use-empty-masters23 --timeout <arg>24 --simulate25 --pipeline <arg>26 --threshold <arg>27 add-node new\_host:new\_port existing\_host:existing\_port28 --slave29 --master-id <arg>30 del-node host:port node\_id31 set-timeout host:port milliseconds32 call host:port command arg arg .. arg33 import host:port34 --from <arg>35 --copy36 --replace37 help (show this help)38 39 For check, fix, reshard, del-node, set-timeout you can specify the host and port of any working node in the cluster.

[IMG_257](http://www.cnblogs.com/PatrickLiu/p/javascript:void(0);)

　　　　　　（对于check，fix，reshard，del-node，settimeout，你可以指定集群中任何工作节点的主机和端口。）  
  
        　　　　可以看到redis-trib.rb具有以下功能：  
  
        　　　　　　1、create：创建集群  
  
       　　　　 　　2、check：检查集群  
  
        　　　　　　3、info：查看集群信息  
  
       　　　　　　 4、fix：修复集群  
  
        　　　　　　5、reshard：在线迁移slot  
  
        　　　　　　6、rebalance：平衡集群节点slot数量  
  
        　　　　　　7、add-node：将新节点加入集群  
  
        　　　　　　8、del-node：从集群中删除节点  
  
        　　　　　　9、set-timeout：设置集群节点间心跳连接的超时时间  
  
        　　　　　　10、call：在集群全部节点上执行命令  
  
        　　　　　　11、import：将外部redis数据导入集群  
  
  
**2、Create创建集群**  
  
**命令格式：ruby redis-trib.rb  create  [--replicas <arg>]  host1:port1 ... hostN:portN【地址列表必须大于等于3】**  
  
        　　　　replicas参数是可选参数，表示每个Master主节点有几个slave从节点，如果该参数存在，必须在 **host1:port1 ... hostN:portN**之前，如果省略了**[--replicas <arg>]**这个参数，只会创建最少3个Master主节点（因为Redis的Cluster集群模式要求最少3个主节点），并且分配好数据槽slots，但是，Slave从节点是不会创建的。  
  
**Master主节点没有Slave从节点，最简单命令使用如下：**

[IMG_258](http://www.cnblogs.com/PatrickLiu/p/javascript:void(0);)

　　　　　　　　[root@linux redis]# ruby redis-trib.rb create 192.168.127.129:7000 192.168.127.129:7001 192.168.127.129:7002

　　　　　　　　>>> Creating cluster

　　　　　　　　>>> Performing hash slots allocation on 3 nodes...

　　　　　　　　Using 3 masters:

　　　　　　　　192.168.127.129:7000

　　　　　　　　192.168.127.129:7001

　　　　　　　　192.168.127.129:7002

　　　　　　　　M: 454e7e2ca5e70103ae19926e139dc212ad084637 192.168.127.129:7000

　　　　　　　　 slots:0-5460 (5461 slots) master

　　　　　　　　M: d16665971f9a1f5715fd7365314436c99ba6e7ef 192.168.127.129:7001

　　　　　　　　 slots:5461-10922 (5462 slots) master

　　　　　　　　M: 7e41f26c96c796c6ad546df836b93d2aadf9e13c 192.168.127.129:7002

　　　　　　　　 slots:10923-16383 (5461 slots) master

　　　　　　　　Can I set the above configuration? (type 'yes' to accept): yes

　　　　　　　　>>> Nodes configuration updated

　　　　　　　　>>> Assign a different config epoch to each node

　　　　　　　　>>> Sending CLUSTER MEET messages to join the cluster

　　　　　　　　Waiting for the cluster to join.

　　　　　　　　>>> Performing Cluster Check (using node 192.168.127.129:7000)

　　　　　　　　M: 454e7e2ca5e70103ae19926e139dc212ad084637 192.168.127.129:7000

　　　　　　　　 slots:0-5460 (5461 slots) master

　　　　　　　　 0 additional replica(s)

　　　　　　　　M: d16665971f9a1f5715fd7365314436c99ba6e7ef 192.168.127.129:7001

　　　　　　　　 slots:5461-10922 (5462 slots) master

　　　　　　　　 0 additional replica(s)

　　　　　　　　M: 7e41f26c96c796c6ad546df836b93d2aadf9e13c 192.168.127.129:7002

　　　　　　　　 slots:10923-16383 (5461 slots) master

　　　　　　　　 0 additional replica(s)

　　　　　　　　[OK] All nodes agree about slots configuration.

　　　　　　　　>>> Check for open slots...

　　　　　　　　>>> Check slots coverage...

　　　　　　　　[OK] All 16384 slots covered.

[IMG_259](http://www.cnblogs.com/PatrickLiu/p/javascript:void(0);)

**每个Master主节点都有一个slave从节点的创建命令如下：**  
        　　　　[root@linux redis]# ruby redis-trib.rb create --replicas 1 192.168.127.130:7000 192.168.127.130:7001  192.168.127.130:7002 192.168.127.130:7003 192.168.127.130:7004 192.168.127.130:7005   
  
        　　　　创建流程如下：  
  
             　　　　　　1、首先为每个节点创建ClusterNode对象，包括连接每个节点。检查每个节点是否为独立且db为空的节点。执行load\_info方法导入节点信息。  
  
            　　 　　　　2、检查传入的master节点数量是否大于等于3个。只有大于3个节点才能组成集群。  
  
             　　　　　　3、计算每个master需要分配的slot数量，以及给master分配slave。  
  
            　　 　　　　4、打印出分配信息，并提示用户输入“yes”确认是否按照打印出来的分配方式创建集群。  
  
             　　　　　　5、输入“yes”后，会执行flush\_nodes\_config操作，该操作执行前面的分配结果，给master分配slot，让slave复制master，对于还没有握手（cluster meet）的节点，slave复制操作无法完成，不过没关系，flush\_nodes\_config操作出现异常会很快返回，后续握手后会再次执行flush\_nodes\_config。  
  
             　　　　　　6、给每个节点分配epoch，遍历节点，每个节点分配的epoch比之前节点大1。  
  
             　　　　　　7、节点间开始相互握手，握手的方式为节点列表的其他节点跟第一个节点握手。  
  
             　　　　　　8、然后每隔1秒检查一次各个节点是否已经消息同步完成，使用ClusterNode的get\_config\_signature方法，检查的算法为获取每个节点cluster nodes信息，排序每个节点，组装成node\_id1:slots|node\_id2:slot2|...的字符串。如果每个节点获得字符串都相同，即认为握手成功。  
  
             　　　　　　9、此后会再执行一次flush\_nodes\_config，这次主要是为了完成slave复制操作。  
  
                  　　　　10、最后再执行check\_cluster，全面检查一次集群状态。包括和前面握手时检查一样的方式再检查一遍。确认没有迁移的节点。确认所有的slot都被分配出去了。  
  
            　  　　　　11、至此完成了整个创建流程，返回[OK] All 16384 slots covered.。  
  
  
**3、Check检查集群主从状态信息**  
  
**命令格式：ruby redis-trib.rb  check  host:port【此地址可以是集群中任何一个节点的地址，相当于获取集群信息的入口】**  
  
        　　　　检查集群状态的命令，没有其他参数，只需要选择一个集群中的一个节点即可。执行命令以及结果如下：

[IMG_260](http://www.cnblogs.com/PatrickLiu/p/javascript:void(0);)

　　　　[root@linux redis]# ruby redis-trib.rb check 192.168.127.130:7000

　　　　>>> Performing Cluster Check (using node 192.168.127.130:7000)

　　 　　M: a44081c39e7978c731a5c64476f4dc64e8d10c20 192.168.127.130:7000

　　　　 slots:66-5460 (5395 slots) master

　　　　 1 additional replica(s)

　　　　 S: be14cfa7cc064e75977ec8517c1eccf96e92aebf 192.168.127.130:7005

　　　　 slots: (0 slots) slave

　　　　 replicates 3b025b3ecfa65f462de639c7a412be443cf1dd1c

　　　　M: bafe517c084aadaacb0b0249dac1e706f24bc21f 192.168.127.130:7001

　　　　 slots:5528-10922 (5395 slots) master

　　　　 1 additional replica(s)

　　 　　 S: b04d39427329b256fcb9b851e02d4b814314b280 192.168.127.130:7004

　　　　 slots: (0 slots) slave

　　　　 replicates bafe517c084aadaacb0b0249dac1e706f24bc21f

　　 　　 M: 3b025b3ecfa65f462de639c7a412be443cf1dd1c 192.168.127.130:7002

　　 　　 slots:0-65,5461-5527,10923-16383 (5594 slots) master

　　 　　 1 additional replica(s)

　　　　 S: b96aac031170a2aa42c619fbc08450bb51af372c 192.168.127.130:7003

　　　　 slots: (0 slots) slave

　　　　 replicates a44081c39e7978c731a5c64476f4dc64e8d10c20

　　　　[OK] All nodes agree about slots configuration.

　　　　>>> Check for open slots...

　　　　>>> Check slots coverage...

　　 　 [OK] All 16384 slots covered.

[IMG_261](http://www.cnblogs.com/PatrickLiu/p/javascript:void(0);)

        　　　　检查前会先执行load\_cluster\_info\_from\_node方法，把所有节点数据load进来。load的方式为通过自己的cluster nodes发现其他节点，然后连接每个节点，并加入nodes数组。接着生成节点间的复制关系。load完数据后，开始检查数据，检查的方式也是调用创建时候使用的check\_cluster。  
  
  
**4、Info查看集群主节点信息**  
  
 **命令格式：ruby redis-trib.rb  info  host:port【此地址可以是集群中任何一个节点的地址，相当于获取集群信息的入口】**  
  
       　　 　　info命令用来查看集群中Master主节点的信息。info命令也是先执行load\_cluster\_info\_from\_node获取完整的集群信息。  
  
        　　　　示例如下：

[IMG_262](http://www.cnblogs.com/PatrickLiu/p/javascript:void(0);)

　　　　　　[root@linux redis]# ruby redis-trib.rb info 192.168.127.130:7000

　　192.168.127.130:7000 (a44081c3...) -> 3 keys | 5395 slots | 1 slaves.

　　192.168.127.130:7001 (bafe517c...) -> 1 keys | 5395 slots | 1 slaves.

　　192.168.127.130:7002 (3b025b3e...) -> 1 keys | 5594 slots | 1 slaves.

　　[OK] 5 keys in 3 masters.

　　0.00 keys per slot on average.

[IMG_263](http://www.cnblogs.com/PatrickLiu/p/javascript:void(0);)

**5、Fix修复集群**  
  
**命令格式：ruby redis-trib.rb  fix  --timeout <arg>  host:port【此地址可以是集群中任何一个节点的地址，相当于获取集群信息的入口】**  
  
        　　　　执行效果如下：

[IMG_264](http://www.cnblogs.com/PatrickLiu/p/javascript:void(0);)

　　　　　　[root@linux redis]# ruby redis-trib.rb fix 192.168.127.130:7002

　　>>> Performing Cluster Check (using node 192.168.127.130:7002)

　　 M: 3b025b3ecfa65f462de639c7a412be443cf1dd1c 192.168.127.130:7002

　　 slots:0-65,5461-5527,10923-16383 (5594 slots) master

　　 1 additional replica(s)

　　 M: bafe517c084aadaacb0b0249dac1e706f24bc21f 192.168.127.130:7001

　　 slots:5528-10922 (5395 slots) master

　　 1 additional replica(s)

　　　　 S: b96aac031170a2aa42c619fbc08450bb51af372c 192.168.127.130:7003

　　 slots: (0 slots) slave

　　 replicates a44081c39e7978c731a5c64476f4dc64e8d10c20

　　S: b04d39427329b256fcb9b851e02d4b814314b280 192.168.127.130:7004

　　 slots: (0 slots) slave

　　 replicates bafe517c084aadaacb0b0249dac1e706f24bc21f

　　M: a44081c39e7978c731a5c64476f4dc64e8d10c20 192.168.127.130:7000

　　 slots:66-5460 (5395 slots) master

　　 1 additional replica(s)

　　S: be14cfa7cc064e75977ec8517c1eccf96e92aebf 192.168.127.130:7005

　　slots: (0 slots) slave

　　replicates 3b025b3ecfa65f462de639c7a412be443cf1dd1c

　　[OK] All nodes agree about slots configuration.

　　>>> Check for open slots...

　　>>> Check slots coverage...

　　[OK] All 16384 slots covered.

　　[root@linux redis]# ruby redis-trib.rb fix 192.168.127.130:7005

　　>>> Performing Cluster Check (using node 192.168.127.130:7005)

　　S: be14cfa7cc064e75977ec8517c1eccf96e92aebf 192.168.127.130:7005

　　 slots: (0 slots) slave

　　 replicates 3b025b3ecfa65f462de639c7a412be443cf1dd1c

　　S: b96aac031170a2aa42c619fbc08450bb51af372c 192.168.127.130:7003

　　 slots: (0 slots) slave

　　 replicates a44081c39e7978c731a5c64476f4dc64e8d10c20

　　 M: 3b025b3ecfa65f462de639c7a412be443cf1dd1c 192.168.127.130:7002

　　 slots:0-65,5461-5527,10923-16383 (5594 slots) master

　　 1 additional replica(s)

　　 M: a44081c39e7978c731a5c64476f4dc64e8d10c20 192.168.127.130:7000

　　 slots:66-5460 (5395 slots) master

　　 1 additional replica(s)

　　S: b04d39427329b256fcb9b851e02d4b814314b280 192.168.127.130:7004

　　 slots: (0 slots) slave

　　 replicates bafe517c084aadaacb0b0249dac1e706f24bc21f

　　M: bafe517c084aadaacb0b0249dac1e706f24bc21f 192.168.127.130:7001

　　 slots:5528-10922 (5395 slots) master

　　 1 additional replica(s)

　　[OK] All nodes agree about slots configuration.

　　>>> Check for open slots...

　　>>> Check slots coverage...

　　[OK] All 16384 slots covered.

[IMG_265](http://www.cnblogs.com/PatrickLiu/p/javascript:void(0);)

        　　　　1、fix命令的流程跟check的流程很像，显示加载集群信息，然后在check\_cluster方法内传入fix为  
true的变量，会在集群检查出现异常的时候执行修复流程。目前fix命令能修复两种异常，一种是集群有处于迁移中的slot的节点，一种是slot未完全分配的异常。  
  
        　　　　**2、fix\_open\_slot方法是修复集群中在迁移slots数据槽的过程中节点的异常。**  
  
            　　　　　　2.1、先检查该slot是谁负责的，迁移的源节点如果没完成迁移，owner还是该节点。没有owner的slot无法完成修复功能。  
  
            　　　　　　2.2、遍历每个节点，获取哪些节点的slot被标记为migrating【迁移】状态，哪些节点的slot被标记为importing【输入】状态。对于owner【所有者】不是该节点，但是通过cluster countkeysinslot获取到该节点有数据的情况，也认为该节点为importing【输入】状态。  
  
            　　　　　　2.3、如果migrating【迁移】和importing【输入】状态的节点均只有1个，这可能是迁移过程中redis-trib.rb被中断所致，直接执行move\_slot继续完成迁移任务即可。传递dots和fix为true。  
  
            　　　　　　2.4、如果migrating【迁移】为空，importing【输入】状态的节点大于0，那么这种情况执行回滚流程，将importing【输入】状态的节点数据通过move\_slot方法导给slot的owner【所有者】节点，传递dots、fix和cold为true。接着对importing的节点执行cluster stable命令恢复稳定。  
  
           　　　　　　 2.5、如果importing【输入】状态的节点为空，有一个migrating【迁移】状态的节点，而且该节点在当前slot没有数据，那么可以直接把这个slot设为stable。  
  
            　　　　　　2.6、如果migrating【迁移】和importing【输入】状态不是上述情况，目前redis-trib.rb工具无法修复，上述的三种情况也已经覆盖了通过redis-trib.rb工具迁移出现异常的各个方面，人为的异常情形太多，很难考虑完全。  
  
  
        　　　　**3、fix\_slots\_coverage方法能修复slot未完全分配的异常。未分配的slot有三种状态。**  
  
           　　　　　　 3.1、所有节点的该slot都没有数据。该状态redis-trib.rb工具直接采用随机分配的方式，并没有考虑节点的均衡。本人尝试对没有分配slot的集群通过fix修复集群，结果slot还是能比较平均的分配，但是没有了连续性，打印的slot信息非常离散。  
      
            　　　　　　3.2、有一个节点的该slot有数据。该状态下，直接把slot分配给该slot有数据的节点。  
  
            　　　　　　3.3、有多个节点的该slot有数据。此种情况目前还处于TODO状态，不过redis作者列出了修复的步骤，对这些节点，除第一个节点，执行cluster migrating命令，然后把这些节点的数据迁移到第一个节点上。清除migrating状态，然后把slot分配给第一个节点。  
  
  
**6、Reshard在线迁移slot**  
  
**命令格式：ruby redis-trib.rb  reshard  --from <arg>  host:port【此地址可以是集群中任何一个主节点的地址，相当于获取集群信息的入口】**  
**--to <arg>**  
**--slots <arg>**  
**--yes**  
**--timeout <arg>**  
**--pipeline <arg>**  
  
  
        　　　　　　reshard命令可以在线把集群的一些slot从集群原来slot负责节点迁移到新的节点，利用reshard可以完成集群的在线横向扩容和缩容。  
  
**reshard的参数很多，下面来一一解释一番：**  
                             
       　　 　　　　host:port：这个是必传参数，用来从一个节点获取整个集群信息，相当于获取集群信息的入口。  
  
        　　　　　　--from <arg>：需要从哪些源节点上迁移slot，可从多个源节点完成迁移，以逗号隔开，传递的是节点的node id，还可以直接传递--from all，这样源节点就是集群的所有节点，不传递该参数的话，则会在迁移过程中提示用户输入。  
  
        　　　　　　--to <arg>：slot需要迁移的目的节点的node id，目的节点只能填写一个，不传递该参数的话，则会在迁移过程中提示用户输入。  
  
        　　　　　　--slots <arg>：需要迁移的slot数量，不传递该参数的话，则会在迁移过程中提示用户输入。  
  
        　　　　　　--yes：设置该参数，可以在打印执行reshard计划的时候，提示用户输入yes确认后再执行reshard。  
  
        　　　　　　--timeout <arg>：设置migrate命令的超时时间。  
  
       　　 　　　　--pipeline <arg>：定义cluster getkeysinslot命令一次取出的key数量，不传的话使用默认值为10。

**执行代码实例：**

[IMG_266](http://www.cnblogs.com/PatrickLiu/p/javascript:void(0);)

　　　　　　　　　　[root@linux redis]# ruby redis-trib.rb reshard --from all --to 34aff9de9f005b23dc1ae090d1791edb6277b255 --slots 200 --yes 192.168.127.129:7000

　　　　　　　　　　>>> Performing Cluster Check (using node 192.168.127.129:7000)

　　　　　　　　　　M: 454e7e2ca5e70103ae19926e139dc212ad084637 192.168.127.129:7000

　　　　　　　　　　 slots:0-5460 (5461 slots) master

　　　　　　　　　　 1 additional replica(s)

　　　　　　　　　　S: 30a872bdf2ec16fabbd8fcb7e08514dffec09364 192.168.127.129:7005

　　　　　　　　　　 slots: (0 slots) slave

　　　　　　　　　　 replicates 7e41f26c96c796c6ad546df836b93d2aadf9e13c

　　　　　　　　　　M: 7e41f26c96c796c6ad546df836b93d2aadf9e13c 192.168.127.129:7002

　　　　　　　　　　 slots:10923-16383 (5461 slots) master

　　　　　　　　　　 1 additional replica(s)

　　　　　　　　　　S: 69ada8a4fe3784597480bd8bddbeb57ab08a8f34 192.168.127.129:7004

　　　　　　　　　　 slots: (0 slots) slave

　　　　　　　　　　 replicates d16665971f9a1f5715fd7365314436c99ba6e7ef

　　　　　　　　　　M: d16665971f9a1f5715fd7365314436c99ba6e7ef 192.168.127.129:7001

　　　　　　　　　　 slots:5461-10922 (5462 slots) master

　　　　　　　　　　 1 additional replica(s)

　　　　　　　　　　M: 34aff9de9f005b23dc1ae090d1791edb6277b255 192.168.127.129:7006

　　　　　　　　　　 slots: (0 slots) master

　　　　　　　　　　 0 additional replica(s)

　　　　　　　　　　S: ee0e0688c0258c34f9673a44e5bc97374abaeb5d 192.168.127.129:7003

　　　　　　　　　　 slots: (0 slots) slave

　　　　　　　　　　 replicates 454e7e2ca5e70103ae19926e139dc212ad084637

　　　　　　　　　　[OK] All nodes agree about slots configuration.

　　　　　　　　　　>>> Check for open slots...

　　　　　　　　　　>>> Check slots coverage...

　　　　　　　　　　[OK] All 16384 slots covered.

　　　　　　　　　　Ready to move 200 slots.

　　　　　　　　　　 Source nodes:

　　　　　　　　　　 M: 454e7e2ca5e70103ae19926e139dc212ad084637 192.168.127.129:7000

　　　　　　　　　　 slots:0-5460 (5461 slots) master

　　　　　　　　　　 1 additional replica(s)

　　　　　　　　　　 M: 7e41f26c96c796c6ad546df836b93d2aadf9e13c 192.168.127.129:7002

　　　　　　　　 slots:10923-16383 (5461 slots) master

　　　　　　　　　　 1 additional replica(s)

　　　　　　　　　　 M: d16665971f9a1f5715fd7365314436c99ba6e7ef 192.168.127.129:7001

　　　　　　　　　　 slots:5461-10922 (5462 slots) master

　　　　　　　　　　 1 additional replica(s)

　　　　　　　　　　 Destination node:

　　　　　　　　　　 M: 34aff9de9f005b23dc1ae090d1791edb6277b255 192.168.127.129:7006

　　　　　　　　　　 slots: (0 slots) master

　　　　　　　　　　 0 additional replica(s)

　　　　　　　　　　 Resharding plan:

　　　　　　　　　　Moving slot 5461 from d16665971f9a1f5715fd7365314436c99ba6e7ef

　　　　　　　　　　Moving slot 5462 from d16665971f9a1f5715fd7365314436c99ba6e7ef

　　　　　　　　　　Moving slot 5463 from d16665971f9a1f5715fd7365314436c99ba6e7ef

　　　　　　　　　　Moving slot 5464 from d16665971f9a1f5715fd7365314436c99ba6e7ef

　　　　　　　　　　Moving slot 5465 from d16665971f9a1f5715fd7365314436c99ba6e7ef

　　　　　　　　　　//省略。。。。

　　　　　　　　　　Moving slot 10943 from 192.168.127.129:7002 to 192.168.127.129:7006:

　　　　　　　　　　Moving slot 10977 from 192.168.127.129:7002 to 192.168.127.129:7006:

　　　　　　　　　　Moving slot 10978 from 192.168.127.129:7002 to 192.168.127.129:7006:

　　　　　　　　　　Moving slot 10979 from 192.168.127.129:7002 to 192.168.127.129:7006:

　　　　　　　　　　Moving slot 10980 from 192.168.127.129:7002 to 192.168.127.129:7006:

　　　　　　　　　　Moving slot 10981 from 192.168.127.129:7002 to 192.168.127.129:7006:

　　　　　　　　　　Moving slot 10982 from 192.168.127.129:7002 to 192.168.127.129:7006:

　　　　　　　　　　Moving slot 10983 from 192.168.127.129:7002 to 192.168.127.129:7006:

　　　　　　　　　　Moving slot 10984 from 192.168.127.129:7002 to 192.168.127.129:7006:

　　　　　　　　　　Moving slot 10985 from 192.168.127.129:7002 to 192.168.127.129:7006:

　　　　　　　　　　Moving slot 10986 from 192.168.127.129:7002 to 192.168.127.129:7006:

　　　　　　　　　　Moving slot 10987 from 192.168.127.129:7002 to 192.168.127.129:7006:

　　　　　　　　　　Moving slot 10988 from 192.168.127.129:7002 to 192.168.127.129:7006:

[IMG_267](http://www.cnblogs.com/PatrickLiu/p/javascript:void(0);)

**迁移的流程如下：**  
  
            　　　　　　1、通过load\_cluster\_info\_from\_node方法装载集群信息。  
  
            　　　　　　2、执行check\_cluster方法检查集群是否健康。只有健康的集群才能进行迁移。  
  
            　　　　　　3、获取需要迁移的slot数量，用户没传递--slots参数，则提示用户手动输入。  
  
            　　　　　　4、获取迁移的目的节点，用户没传递--to参数，则提示用户手动输入。此处会检查目的节点必须为master节点。  
  
           　　 　　　　5、获取迁移的源节点，用户没传递--from参数，则提示用户手动输入。此处会检查源节点必须为master节点。--from all的话，源节点就是除了目的节点外的全部master节点。这里为了保证集群slot分配的平均，建议传递--from all。  
  
            　　　　　　6、执行compute\_reshard\_table方法，计算需要迁移的slot数量如何分配到源节点列表，采用的算法是按照节点负责slot数量由多到少排序，计算每个节点需要迁移的slot的方法为：迁移slot数量 \* (该源节点负责的slot数量 / 源节点列表负责的slot总数)。这样算出的数量可能不为整数，这里代码用了下面的方式处理：  
  
            　　　　　　n = (numslots/source\_tot\_slots\*s.slots.length)  
            　　　　　　if i == 0  
            　　　　　　    n = n.ceil  
            　　　　　　else  
             　　　　　　   n = n.floor  
          
           　　 　　　　这样的处理方式会带来最终分配的slot与请求迁移的slot数量不一致，这个BUG已经在github上提给作者，    https://github.com/antirez/redis/issues/2990。  
  
            　　　　　　7、打印出reshard计划，如果用户没传--yes，就提示用户确认计划。  
  
            　　　　　　8、根据reshard计划，一个个slot的迁移到新节点上，迁移使用move\_slot方法，该方法被很多命令使用，具体可以参见下面的迁移流程。move\_slot方法传递dots为true和pipeline数量。  
  
            　　　　　　9、至此，就完成了全部的迁移任务。  
  
  
            　　　　　　move\_slot方法可以在线将一个slot的全部数据从源节点迁移到目的节点，fix、reshard、rebalance都需要调用该方法迁移slot。  
  
**move\_slot接受下面几个参数，**  
  
                　　　　　　1、pipeline：设置一次从slot上获取多少个key。  
  
                　　　　　　2、quiet：迁移会打印相关信息，设置quiet参数，可以不用打印这些信息。  
  
                　　　　　　3、cold：设置cold，会忽略执行importing和migrating。  
  
                　　　　　　4、dots：设置dots，则会在迁移过程打印迁移key数量的进度。  
  
               　　 　　　　5、update：设置update，则会更新内存信息，方便以后的操作。  
  
  
**move\_slot流程如下：**  
  
                　　　　　　1、如果没有设置cold，则对源节点执行cluster importing命令，对目的节点执行migrating命令。fix的时候有可能importing和migrating已经执行过来，所以此种场景会设置cold。  
  
                　　　　　　2、通过cluster getkeysinslot命令，一次性获取远节点迁移slot的pipeline个key的数量.  
  
                　　　　　　3、对这些key执行migrate命令，将数据从源节点迁移到目的节点。  
  
                　　　　　　4、如果migrate出现异常，在fix模式下，BUSYKEY的异常，会使用migrate的replace模式再执行一次，BUSYKEY表示目的节点已经有该key了，replace模式可以强制替换目的节点的key。不是fix模式就直接返回错误了。  
  
                　　　　　　5、循环执行cluster getkeysinslot命令，直到返回的key数量为0，就退出循环。  
  
                　　　　　　6、如果没有设置cold，对每个节点执行cluster setslot命令，把slot赋给目的节点。  
  
                　　　　　　7、如果设置update，则修改源节点和目的节点的slot信息。  
  
                　　　　　　8、至此完成了迁移slot的流程。  
  
  
**7、Rebalance平衡集群节点slot数量**  
  
   **命令格式：ruby redis-trib.rb  rebalance --weight <arg>  host:port【此地址可以是集群中任何一个节点的地址，相当于获取集群信息的入口】**  
**--auto-weights**  
**--threshold <arg>**  
**--use-empty-masters**  
**--timeout <arg>**  
**--simulate**  
                                                                                     **--pipeline <arg>**  
  
        　　　　　　rebalance命令可以根据用户传入的参数平衡集群节点的slot数量，rebalance功能非常强大，可以传入的参数很多。

　　　　　　　　以下是rebalance的参数列表和命令示例。  
       　　　　 　　# ruby redis-trib.rb rebalance --threshold 1 --weight b31e3a2e=5 --weight 60b8e3a1=5 --use-empty-masters  --simulate 10.180.157.199:6379

        　　　　　　**下面也先一一解释下每个参数的用法：**  
  
            　　　　　　 host:port：这个是必传参数，用来从一个节点获取整个集群信息，相当于获取集群信息的入口。  
  
            　　　　　　--weight <arg>：节点的权重，格式为node\_id=weight，如果需要为多个节点分配权重的话，需要添加多个--weight <arg>参数，即--weight b31e3a2e=5 --weight 60b8e3a1=5，node\_id可为节点名称的前缀，只要保证前缀位数能唯一区分该节点即可。没有传递    –weight的节点的权重默认为1。  
  
            　　　　　　--auto-weights：这个参数在rebalance流程中并未用到。  
  
            　　　　　　--threshold <arg>：只有节点需要迁移的slot阈值超过threshold，才会执行rebalance操作。具体计算方法可以参考下面的rebalance命令流程的第四步。  
  
            　　　　　　--use-empty-masters：rebalance是否考虑没有节点的master，默认没有分配slot节点的master是不参与rebalance的，设置--use-empty-masters可以让没有分配slot的节点参与rebalance。  
  
           　　　　 　　--timeout <arg>：设置migrate命令的超时时间。  
  
            　　　　　　--simulate：设置该参数，可以模拟rebalance操作，提示用户会迁移哪些slots，而不会真正执行迁移操作。  
  
           　　　　 　　--pipeline <arg>：与reshar的pipeline参数一样，定义cluster getkeysinslot命令一次取出的key数量，不传的话使用默认值为10。  
  
  
       　　　**rebalance命令流程如下：**  
  
            　　　　　　　　1、load\_cluster\_info\_from\_node方法先加载集群信息。  
  
            　　　　　　　　2、计算每个master的权重，根据参数--weight <arg>，为每个设置的节点分配权重，没有设置的节点，则权重默认为1。  
  
            　　　　　　　　3、根据每个master的权重，以及总的权重，计算自己期望被分配多少个slot。计算的方式为：总slot数量 \* （自己的权重 / 总权重）。  
  
            　　　　　　　　4、计算每个master期望分配的slot是否超过设置的阈值，即--threshold <arg>设置的阈值或者默认的阈值。计算的方式为：先计算期望移动节点的阈值，算法为：(100-(100.0\*expected/n.slots.length)).abs，如果计算出的阈值没有超出设置阈值，则不需要为该节点移动slot。只要有一个master的移动节点超过阈值，就会触发rebalance操作。  
  
            　　　　　　　　5、如果触发了rebalance操作。那么就开始执行rebalance操作，先将每个节点当前分配的slots数量减去期望分配的slot数量获得balance值。将每个节点的balance从小到大进行排序获得sn数组。  
  
            　　　　　　　　6、用dst\_idx和src\_idx游标分别从sn数组的头部和尾部开始遍历。目的是为了把尾部节点的slot分配给头部节点。  
  
    　　　　　　　　　　sn数组保存的balance列表排序后，负数在前面，正数在后面。负数表示需要有slot迁入，所以使用dst\_idx游标，正数表示需要有slot迁出，所以使用src\_idx游标。理论上sn数组各节点的balance值加起来应该为0，不过由于在计算期望分配的slot的时候只是使用直接取整的方式，所以可能出现balance值之和不为0的情况，balance值之和不为0即为节点不平衡的slot数量，由于slot总数有16384个，不平衡数量相对于总数，基数很小，所以对rebalance流程影响不大。  
  
            　　　　　　　　7、获取sn[dst\_idx]和sn[src\_idx]的balance值较小的那个值，该值即为需要从sn[src\_idx]节点迁移到sn[dst\_idx]节点的slot数量。  
  
           　　　　 　　　　8、接着通过compute\_reshard\_table方法计算源节点的slot如何分配到源节点列表。这个方法在reshard流程中也有调用，具体步骤可以参考reshard流程的第六步。  
  
            　　　　　　　　9、如果是simulate模式，则只是打印出迁移列表。  
  
            　　　　　　　　10、如果没有设置simulate，则执行move\_slot操作，迁移slot，传入的参数为:quiet=>true,:dots=>false,:update=>true。  
  
            　　　　　　　　11、迁移完成后更新sn[dst\_idx]和sn[src\_idx]的balance值。如果balance值为0后，游标向前进1。  
  
            　　　　　　　　12、直到dst\_idx到达src\_idx游标，完成整个rebalance操作。  
  
  
    　　　　**8、add-node将新节点加入集群**  
  
        　　　**命令格式：ruby redis-trib.rb add-node --slave  new\_host:new\_port existing\_host:existing\_port【此地址可以是集群中任何一个节点的地址，相当于获取集群信息的入口】**  
**--master-id <arg>**  
  
       　　　　 　　add-node命令可以将新节点加入集群，节点可以为master，也可以为某个master节点的slave。  
  
        　　　　　　add-node有两个可选参数：  
  
            　　　　　　　　--slave：设置该参数，则新节点以slave的角色加入集群  
  
           　　　　 　　　　--master-id：这个参数需要设置了--slave才能生效，--master-id用来指定新节点的master节点。如果不设置该参数，则会随机为节点选择master节点。

　　　　　　　　　　　 这两个参数（--slave，--master-id）并没有先后顺序，但是这两个参数必须在**add-node** 之后，**new\_host:new\_port existing\_host:existing\_port**参数之前。  
  
**可以看下add-node命令的执行示例（我们将192.168.127.129:7003作为192.168.127.129:7000的从节点增加进集群）：**

[IMG_268](http://www.cnblogs.com/PatrickLiu/p/javascript:void(0);)

　　　　　　　　　　[root@linux redis]# ruby redis-trib.rb add-node --slave --master-id 454e7e2ca5e70103ae19926e139dc212ad084637 192.168.127.129:7003 192.168.127.129:7000

　　　　　　　　　　>>> Adding node 192.168.127.129:7003 to cluster 192.168.127.129:7000

　　　　　　　　　　>>> Performing Cluster Check (using node 192.168.127.129:7000)

　　　　　　　　　　M: 454e7e2ca5e70103ae19926e139dc212ad084637 192.168.127.129:7000

　　　　　　　　　　 slots:0-5460 (5461 slots) master

　　　　　　　　　　 0 additional replica(s)

　　　　　　　　　　M: d16665971f9a1f5715fd7365314436c99ba6e7ef 192.168.127.129:7001

　　　　　　　　　　 slots:5461-10922 (5462 slots) master

　　　　　　　　　　 0 additional replica(s)

　　　　　　　　　　M: 7e41f26c96c796c6ad546df836b93d2aadf9e13c 192.168.127.129:7002

　　　　　　　　　　 slots:10923-16383 (5461 slots) master

　　　　　　　　　　 0 additional replica(s)

　　　　　　　　　　[OK] All nodes agree about slots configuration.

　　　　　　　　　　>>> Check for open slots...

　　　　　　　　　　>>> Check slots coverage...

　　　　　　　　　　[OK] All 16384 slots covered.

　　　　　　　　　　>>> Send CLUSTER MEET to node 192.168.127.129:7003 to make it join the cluster.

　　　　　　　　　　Waiting for the cluster to join.

　　　　　　　　　　>>> Configure node as replica of 192.168.127.129:7000.

　　　　　　　　　　[OK] New node added correctly.

[IMG_269](http://www.cnblogs.com/PatrickLiu/p/javascript:void(0);)

**可以看下add-node命令的执行示例（我们将192.168.127.129:7004作为192.168.127.129:7001的从节点增加进集群）：**

[IMG_270](http://www.cnblogs.com/PatrickLiu/p/javascript:void(0);)

　　　　　　　　　　[root@linux redis]# ruby redis-trib.rb add-node --slave --master-id d16665971f9a1f5715fd7365314436c99ba6e7ef 192.168.127.129:7004 192.168.127.129:7001

　　　　　　　　　　>>> Adding node 192.168.127.129:7004 to cluster 192.168.127.129:7001

　　　　　　　　　　>>> Performing Cluster Check (using node 192.168.127.129:7001)

　　　　　　　　　　M: d16665971f9a1f5715fd7365314436c99ba6e7ef 192.168.127.129:7001

　　　　　　　　　　 slots:5461-10922 (5462 slots) master

　　　　　　　　　　 0 additional replica(s)

　　　　　　　　　　S: ee0e0688c0258c34f9673a44e5bc97374abaeb5d 192.168.127.129:7003

　　　　　　　　　　 slots: (0 slots) slave

　　　　　　　　　　 replicates 454e7e2ca5e70103ae19926e139dc212ad084637

　　　　　　　　　　M: 454e7e2ca5e70103ae19926e139dc212ad084637 192.168.127.129:7000

　　　　　　　　　　 slots:0-5460 (5461 slots) master

　　　　　　　　　　 1 additional replica(s)

　　　　　　　　　　M: 7e41f26c96c796c6ad546df836b93d2aadf9e13c 192.168.127.129:7002

　　　　　　　　　　 slots:10923-16383 (5461 slots) master

　　　　　　　　　　 0 additional replica(s)

　　　　　　　　　　[OK] All nodes agree about slots configuration.

　　　　　　　　　　>>> Check for open slots...

　　　　　　　　　　>>> Check slots coverage...

　　　　　　　　　　[OK] All 16384 slots covered.

　　　　　　　　　　>>> Send CLUSTER MEET to node 192.168.127.129:7004 to make it join the cluster.

　　　　　　　　　　Waiting for the cluster to join.

　　　　　　　　　　>>> Configure node as replica of 192.168.127.129:7001.

　　　　　　　　　　[OK] New node added correctly.

　　　　　　　　　　[root@linux redis]#

[IMG_271](http://www.cnblogs.com/PatrickLiu/p/javascript:void(0);)

**可以看下add-node命令的执行示例（我们将192.168.127.129:7005作为192.168.127.129:7000的从节点增加进集群，此时7000已有从节点，但是7005并没有作为7000的从节点，而是作为7002的从节点，因为当时7002没从节点）：**

[IMG_272](http://www.cnblogs.com/PatrickLiu/p/javascript:void(0);)

　　　　　　　　　　[root@linux redis]# ruby redis-trib.rb add-node --master-id 454e7e2ca5e70103ae19926e139dc212ad084637 --slave 192.168.127.129:7005 192.168.127.129:7000

　　　　　　　　　　>>> Adding node 192.168.127.129:7005 to cluster 192.168.127.129:7000

　　　　　　　　　　>>> Performing Cluster Check (using node 192.168.127.129:7000)

　　　　　　　　　　M: 454e7e2ca5e70103ae19926e139dc212ad084637 192.168.127.129:7000

　　　　　　　　　　 slots:0-5460 (5461 slots) master

　　　　　　　　　　 1 additional replica(s)

　　　　　　　　　　M: 7e41f26c96c796c6ad546df836b93d2aadf9e13c 192.168.127.129:7002

　　　　　　　　　　 slots:10923-16383 (5461 slots) master

　　　　　　　　 0 additional replica(s)

　　　　　　　　　　S: 69ada8a4fe3784597480bd8bddbeb57ab08a8f34 192.168.127.129:7004

　　　　　　　　　　 slots: (0 slots) slave

　　　　　　　　　　 replicates d16665971f9a1f5715fd7365314436c99ba6e7ef

　　　　　　　　　　M: d16665971f9a1f5715fd7365314436c99ba6e7ef 192.168.127.129:7001

　　　　　　　　　　 slots:5461-10922 (5462 slots) master

　　　　　　　　　　 1 additional replica(s)

　　　　　　　　　　S: ee0e0688c0258c34f9673a44e5bc97374abaeb5d 192.168.127.129:7003

　　　　　　　　　　 slots: (0 slots) slave

　　　　　　　　　　 replicates 454e7e2ca5e70103ae19926e139dc212ad084637

　　　　　　　　　　[OK] All nodes agree about slots configuration.

　　　　　　　　　　>>> Check for open slots...

　　　　　　　　　　>>> Check slots coverage...

　　　　　　　　　　[OK] All 16384 slots covered.

　　　　　　　　　　>>> Send CLUSTER MEET to node 192.168.127.129:7005 to make it join the cluster.

　　　　　　　　　　Waiting for the cluster to join.

　　　　　　　　　　>>> Configure node as replica of 192.168.127.129:7000.

　　　　　　　　　　[OK] New node added correctly.

[IMG_273](http://www.cnblogs.com/PatrickLiu/p/javascript:void(0);)

**add-node流程如下：**  
  
            　　　　　　　　1、通过load\_cluster\_info\_from\_node方法转载集群信息，check\_cluster方法检查集群是否健康。  
  
            　　　　　　　　2、如果设置了--slave，则需要为该节点寻找master节点。设置了--master-id，则以该节点作为新节点的master，如果没有设置--master-id，则调用get\_master\_with\_least\_replicas方法，寻找slave数量最少的master节点。如果slave数量一致，则选取load\_cluster\_info\_from\_node顺序发现的第一个节点。load\_cluster\_info\_from\_node顺序的第一个节点是add-node设置的existing\_host:existing\_port节点，后面的顺序根据在该节点执行cluster nodes返回的结果返回的节点顺序。  
  
            　　　　　　　　3、连接新的节点并与集群第一个节点握手。  
  
            　　　　　　　　4、如果没设置–slave就直接返回ok，设置了–slave，则需要等待确认新节点加入集群，然后执行cluster replicate命令复制master节点。  
  
            　　　　　　　　5、至此，完成了全部的增加节点的流程。  
  
  
  **9、del-node从集群中删除节点**  
  
**命令格式：ruby redis-trib.rb del-node host:port node-id【node id是要删除的节点ID标识】**  
  
        　　　　　　del-node可以把某个节点从集群中删除。del-node只能删除没有分配slot的节点。删除命令传递两个参数：  
  
       　　　　 　　host:port：从该节点获取集群信息。  
  
        　　　　　　node\_id：需要删除的节点id。  
  
**del-node执行结果示例如下：**

　　　　　　　　　　[root@linux redis]# ruby redis-trib.rb del-node 192.168.127.129:7006 34aff9de9f005b23dc1ae090d1791edb6277b255

　　　　　　　　　　>>> Removing node 34aff9de9f005b23dc1ae090d1791edb6277b255 from cluster 192.168.127.129:7006

　　　　　　　　　　>>> Sending CLUSTER FORGET messages to the cluster...

　　　　　　　　　　>>> SHUTDOWN the node.

**del-node流程如下：**  
  
           　　　　 　　1、通过load\_cluster\_info\_from\_node方法转载集群信息。  
  
            　　　　　　2、根据传入的node id获取节点，如果节点没找到，则直接提示错误并退出。  
  
            　　　　　　3、如果节点分配的slot不为空，则直接提示错误并退出。  
  
            　　　　　　4、遍历集群内的其他节点，执行cluster forget命令，从每个节点中去除该节点。如果删除的节点是master，而且它有slave的话，这些slave会去复制其他master，调用的方法是get\_master\_with\_least\_replicas，与add-node没设置--master-id寻找master的方法一样。  
  
           　　　　 　　5、然后关闭该节点  
  
  
**10、set-timeout设置集群节点间心跳连接的超时时间**  
  
**命令格式：ruby redis-trib.rb set-timeout host:port  milliseconds**  
  
        　　　　　　set-timeout用来设置集群节点间心跳连接的超时时间，单位是毫秒，不得小于100毫秒，因为100毫秒对于心跳时间来说太短了。该命令修改是节点配置参数cluster-node-timeout，默认是15000毫秒。通过该命令，可以给每个节点设置超时时间，设置的方式使用config set命令动态设置，然后执行config rewrite命令将配置持久化保存到硬盘。以下是示例：

[IMG_274](http://www.cnblogs.com/PatrickLiu/p/javascript:void(0);)

　　　　　　　　[root@linux redis]# ruby redis-trib.rb set-timeout 192.168.127.130:7000 15000

　　　　　　　　>>> Reconfiguring node timeout in every cluster node...

　　　　　　　　\*\*\* New timeout set for 192.168.127.130:7000

　　　　　　　　\*\*\* New timeout set for 192.168.127.130:7002

　　　　　　　　\*\*\* New timeout set for 192.168.127.130:7003

　　　　　　　　\*\*\* New timeout set for 192.168.127.130:7005

　　　　　　　　\*\*\* New timeout set for 192.168.127.130:7004

　　　　　　　　\*\*\* New timeout set for 192.168.127.130:7001

　　　　　　　　>>> New node timeout set. 6 OK, 0 ERR.

[IMG_275](http://www.cnblogs.com/PatrickLiu/p/javascript:void(0);)

**11、call在集群全部节点上执行命令**  
  
**命令格式：ruby redis-trib.rb call host:port [command redis的命令]**  
  
       　　　　 　　call命令可以用来在集群的全部节点执行相同的命令。call命令也是需要通过集群的一个节点地址，连上整个集群，然后在集群的每个节点执行该命令。

[IMG_276](http://www.cnblogs.com/PatrickLiu/p/javascript:void(0);)

　　　　[root@linux redis]# ruby redis-trib.rb call 192.168.127.130:7000 get key

　　　　　　　　　>>> Calling GET key  
　　　　　　　　　192.168.127.130:7000: MOVED 12539 192.168.127.130:7002  
　　　　　　　　　192.168.127.130:7002:   
　　　　　　　　　192.168.127.130:7003: MOVED 12539 192.168.127.130:7002  
　　　　　　　　　192.168.127.130:7005: MOVED 12539 192.168.127.130:7002  
　　　　　　　　　192.168.127.130:7004: MOVED 12539 192.168.127.130:7002  
　　　　　　　　　192.168.127.130:7001: MOVED 12539 192.168.127.130:7002

[IMG_277](http://www.cnblogs.com/PatrickLiu/p/javascript:void(0);)

    　　　　**12、import将外部redis数据导入集群**  
  
        　　　　　　**命令格式：ruby redis-trib.rb import --from host:port host:port**  
  
        　　　　　　执行示例如下：

　　　　 [root@linux redis]# ruby redis-trib.rb import --from 192.168.127.130:7006 192.168.127.129:7000

        　　　　　　上面的命令是把 192.168.127.130:7006（redis 4.8）上的数据导入到 192.168.127.129:7000 这个节点所在的集群  
  
  
        　　　　　　import命令可以把外部的redis节点数据导入集群。导入的流程如下：  
  
        　　　　　　　　1、通过load\_cluster\_info\_from\_node方法转载集群信息，check\_cluster方法检查集群是否健康。  
  
        　　　　　　　　2、连接外部redis节点，如果外部节点开启了cluster\_enabled，则提示错误。  
  
        　　　　　　　　3、通过scan命令遍历外部节点，一次获取1000条数据。  
  
       　　　　 　　　　4、遍历这些key，计算出key对应的slot。  
  
        　　　　　　　　5、执行migrate命令,源节点是外部节点,目的节点是集群slot对应的节点，如果设置了--copy参数，则传递copy参数，如果设置了--replace，则传递replace参数。  
  
        　　　　　　　　6、不停执行scan命令，直到遍历完全部的key。  
  
        　　　　　　　　7、至此完成整个迁移流程  
  
        　　　　　　　　这中间如果出现异常，程序就会停止。没使用--copy模式，则可以重新执行import命令，使用--copy的话，最好清空新的集群再导入一次。  
  
        　　　　　　　　import命令更适合离线的把外部redis数据导入，在线导入的话最好使用更专业的导入工具，以slave的方式连接redis节点去同步节点数据应该是更好的方式。  
  
  
**三、总结**  
  
    　　　　好了，今天就写到这里，有关redis-trib.rb脚步文件的内容还很多，以后再补充进来吧。这个脚本文件对我们很重要，集群的操作很多功能都可以通过这个脚本文件来完成。