**Redis集群**

数工-研发部-龙力

数工-研发部-王季元

## 1.什么是Redis

Redis是一种基于内存的key-value型数据库,属于nosql（not only sql）的一种， Redis支持五种数据类型：string（字符串），hash（哈希），list（列表），set（集合）及zset(sorted set：有序集合)。

\*\*\*基于内存：像mysql,Oracle等关系型数据库，是将数据存放在磁盘上，而基于内存的数据库则是将数据存放在内存中，由于内存的读写特性，基于内存的数据库可以提高几个数量级的读写速度，从而提高性能

## 2.Redis集群的用处

Redis 集群是一个可以在多个 Redis 节点之间进行数据共享的设施installation。因为redis是基于内存的，如果数据量过大，一台电脑十几到几十G的内存显然无法满足，因此需要使用多台redis（集群）作为缓存数据库，以便达到数据量的要求。

## 3.Redis集群搭建说明（windows环境）

Redis 在3.0版本前只支持单实例模式，虽然支持主从模式、哨兵模式部署来解决单点故障，但是现在互联网企业动辄大几百G的数据，可完全是没法满足业务的需求，所以，Redis 在 3.0 版本以后就推出了集群模式。Redis 客户端可以在任意一个 Redis 实例发出请求，如果所需数据不在该实例中，通过重定向命令引导客户端访问所需的实例。

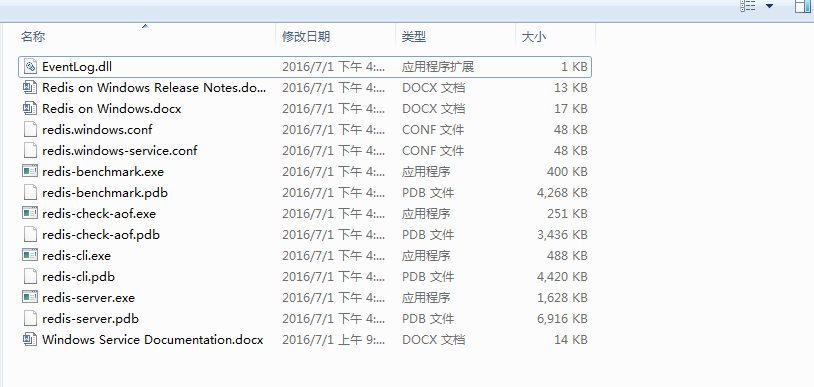
Redis官方是推荐使用linux操作系统的，本文档是在windows操作系统中安装使用redis简单集群，但是建议redis集群作为生产环境时，还是使用linux操作系统！本文后面的成熟的redis集群方案将使用linux系统进行演示。此外，redis集群需要依赖ruby环境。本文档测试环境为win 7+redis 3.2.100(稳定版本)+ruby 2.2.4(Ruby语言运行环境)+redis-3.2.2.gem(Redis的Ruby驱动)+ redis-trib.rb(创建Redis集群的工具)。

所需文件下载地址（FTP服务器）：<ftp://192.168.22.112/常用软件/Redis集群>

## 4.安装Redis

### 1）解压redis-x86-3.2.100.zip（免安装版）

建议在某盘根目录下创建一个redis文件夹，用于存放解压redis-x86-3.2.100.zip后的文件，解压后的文件夹结构如下：



### 2）指令启动redis

cmd转到解压的redis文件夹目录下(或者直接在redis文件夹目录下打开命令窗口)，输入:

redis-server redis.windows.conf



出现以上信息，则表示redis启动成功！

### 3）服务启动redis

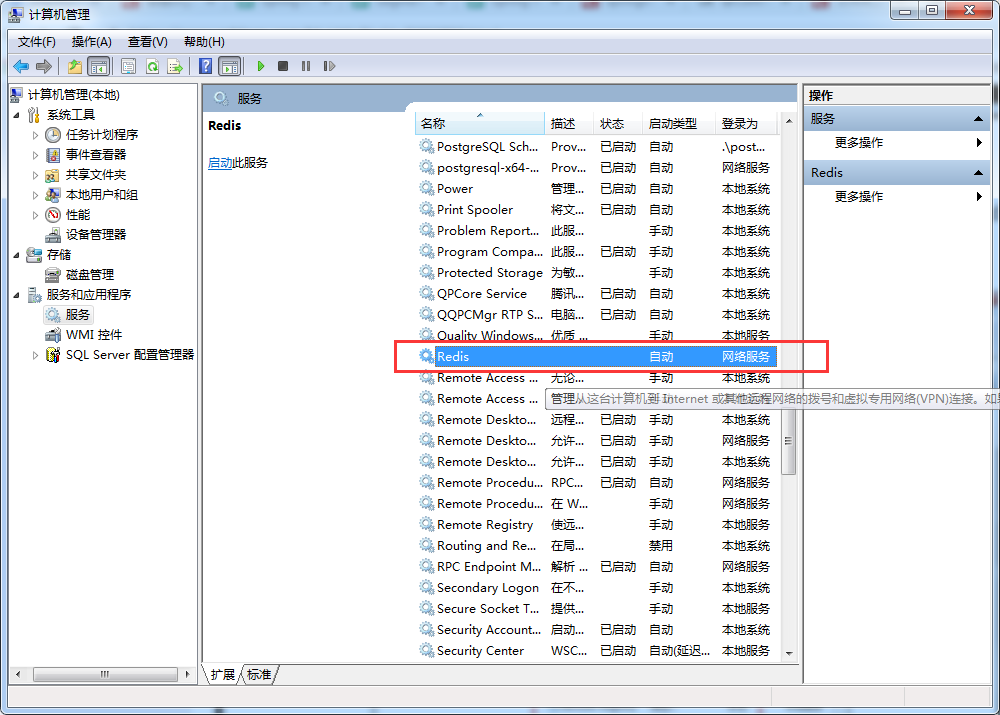
虽然上面方式可以启动redis，但是只要一关闭cmd窗口，redis服务就会关闭。所以要把redis设置成windows下的服务。

cmd转到解压的redis文件夹目录下(或者直接在redis文件夹目录下打开命令窗口)，输入:

redis-server --service-install redis.windows-service.conf --loglevel verbose



再查看windows服务中是否出现redis



如上面所示，则表示服务已添加成功！

### 4）注意事项

解压文件后，含有redis.windows.conf和redis.windows-service.conf两个文件，其中redis.windows.conf文件会影响通过指令启动的redis服务，比如修改文件中的端口号，设置密码，允许集群等等，redis.windows-service.conf文件会影响通过服务添加启动的redis服务。此文档操作在修改redis.windows.conf文件，通过cmd指令启动redis服务情况下进行。

### 5）redis使用与测试

#### I.常用的redis服务命令

皆在redis目录下通过指令窗口运行：

卸载服务：redis-server --service-uninstall

开启服务：redis-server redis.windows.conf（redis-server --service-start）

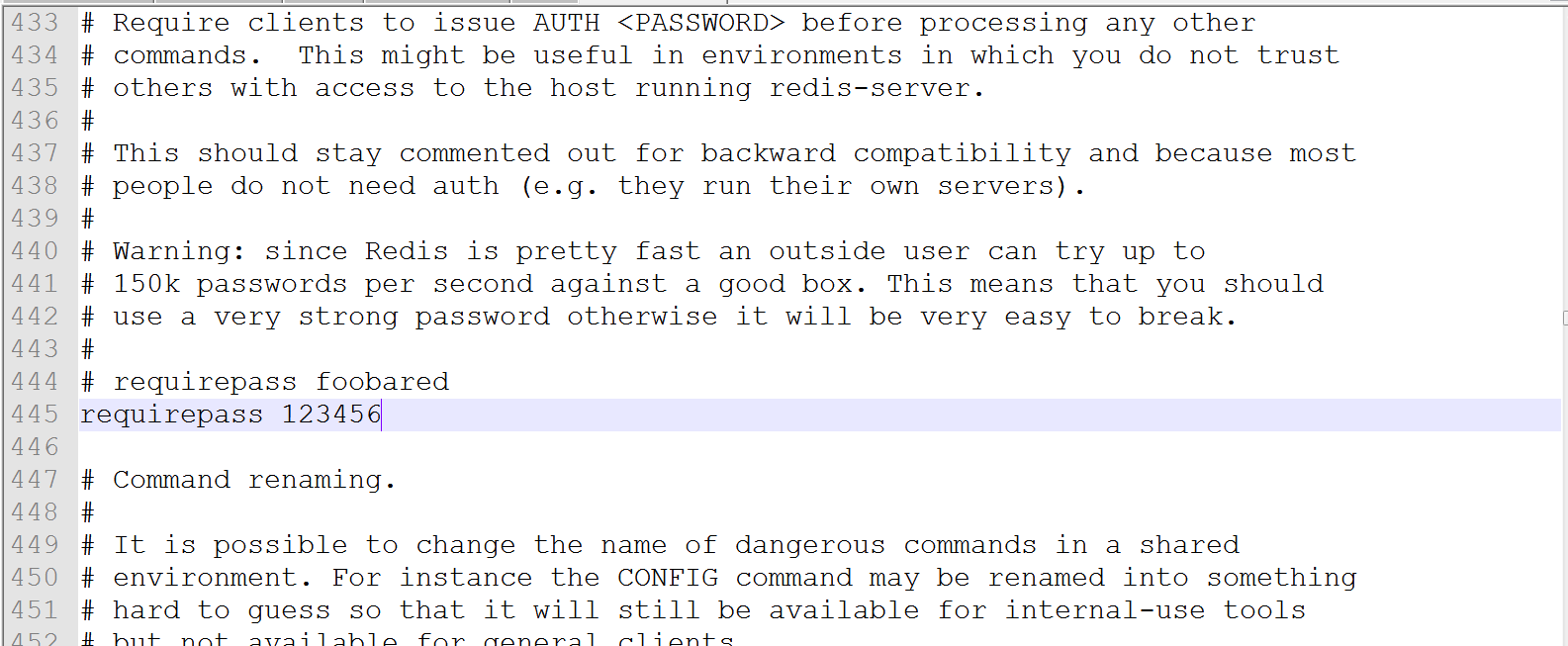
停止服务：redis-server --service-stop

#### II.设置redis密码

可不用设置，了解即可！

在redis文件夹中打开redis.windows.conf文件，找到requirepass foobared

在下面添加”requirepass 123456”，保存关闭（注意前面不能有空格）



注意每次修改conf文件后，需要重启redis才能生效

#### III.测试

测试redis：

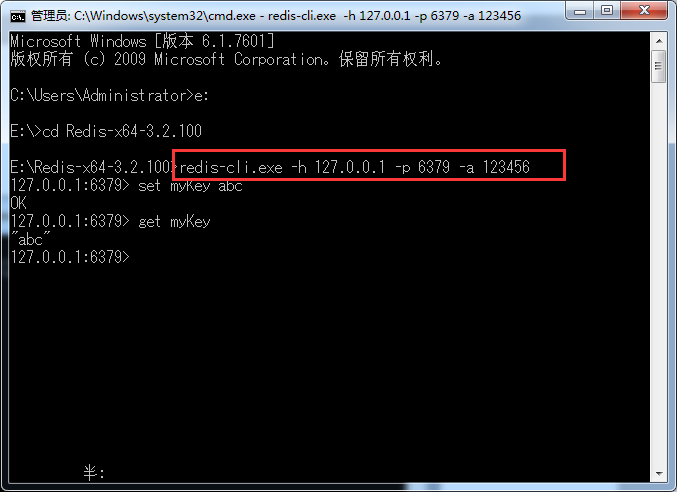
切换到redis目录下运行： redis-cli.exe -h 127.0.0.1 -p 6379 -a 123456

如果没有设置密码，则不用追加 -a 123465

127.0.0.1为conf文件默认的访问端口号

设置键值对 set myKey abc

取出键值对 get myKey

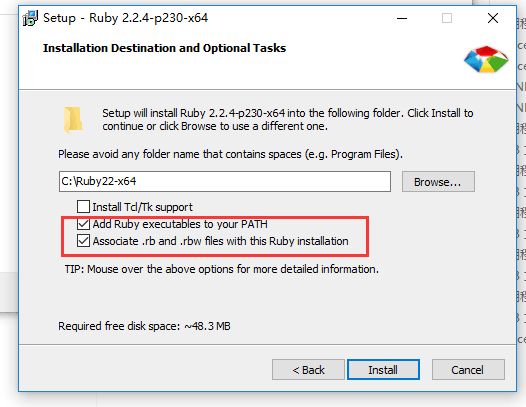


以上则为redis安装测试的过程！

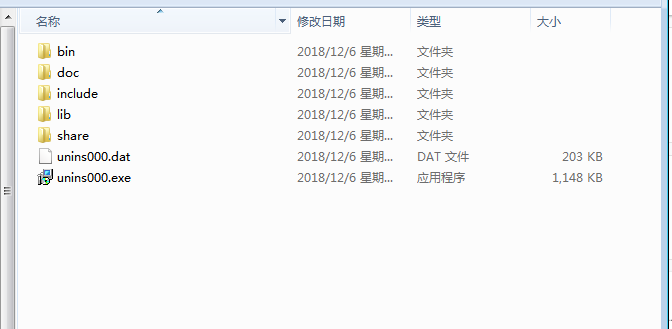
## 5.安装Ruby

### 1）安装ruby(Ruby语言运行环境)

启动[rubyinstaller-2.2.4-x64.exe](http://dl.bintray.com/oneclick/rubyinstaller/rubyinstaller-2.2.4-x64.exe)。为了操作方便，也是建议安装在盘符根目录下，如： C:\Ruby22-x64 ，安装时这里选中后两个选项，意思是将ruby添加到系统的环境变量中，在cmd命令中能直接使用ruby的命令

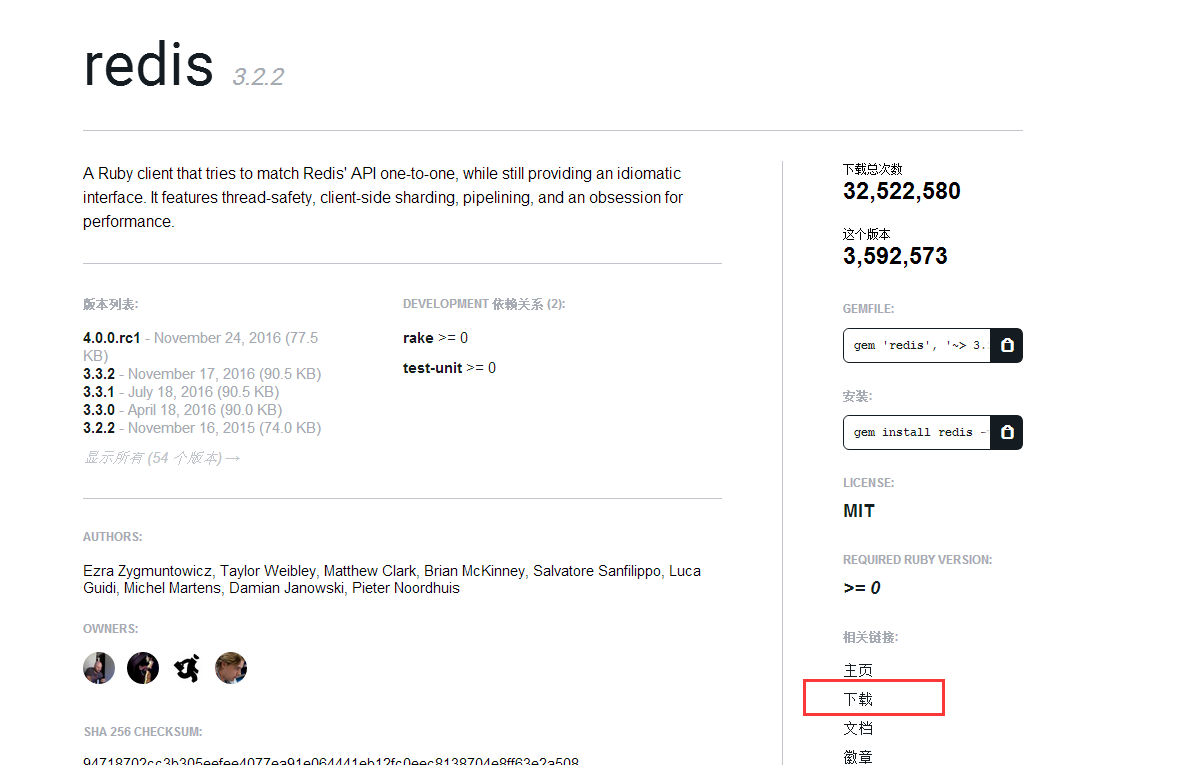


安装完成后，则可在相应安装目录下看到



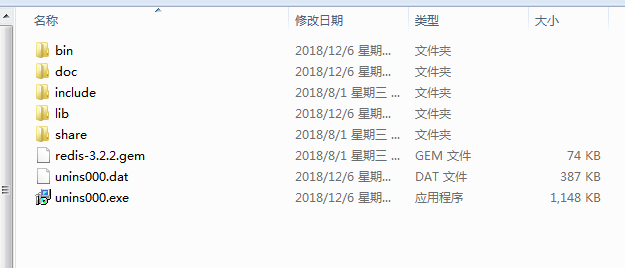
### 2）安装redis-3.2.2.gem(Redis的Ruby驱动)

　下载地址：[https://rubygems.org/gems/redis/versions/3.2.2](https://rubygems.org/gems/redis/versions/3.2.2" \t "_blank)



也可直接从上述FTP中下载下来

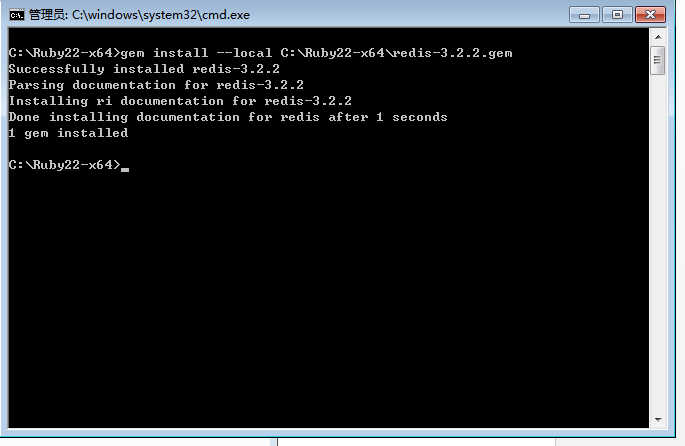
放到上面ruby的目录中，如下：



然后调用命令窗口输入：

gem install --local C:\Ruby22-x64\redis-3.2.2.gem

运行



如上所示，表示安装成功！

### 3）下载redis-trib.rb(创建Redis集群的工具)

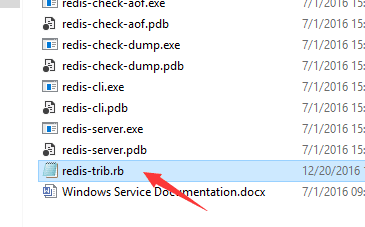
[https://raw.githubusercontent.com/MSOpenTech/redis/3.0/src/redis-trib.rb](https://raw.githubusercontent.com/MSOpenTech/redis/3.0/src/redis-trib.rb" \t "_blank)

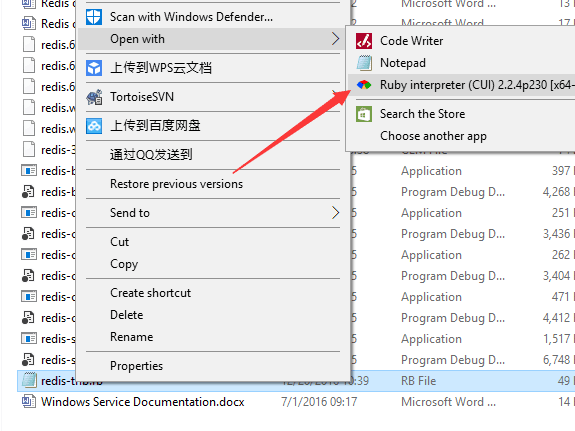
打开该链接如果没有下载，而是打开一个页面，那么将该页面保存为**redis-trib.rb**

也可直接从FTP上下载该文件

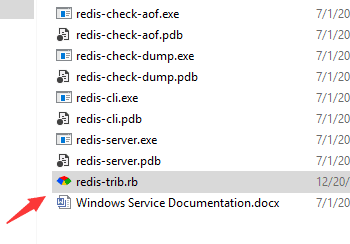
建议保存到Redis的目录下

注意：因为redis-trib.rb是ruby代码，必须用ruby来打开，若redis-trib.rb无法识别，需要手动选择该文件的打开方式：





选择ruby为的打开方式后，redis-trib.rb的logo都会发生改变，如下图：



至此，ruby环境安装成功！

## 6.Redis集群安装

### 1）在同一台机器上安装多个Redis服务集群

在生产环境中并不建议在一台物理机上/虚拟机上部署多个Redis节点，因为这样大大增加了多个Redis节点同时不可用的风险，但这是示例场景所以无所谓啦！

通过配置文件来启动4个不同的Redis实例，由于Redis默认端口为6379，所以这里使用了6380、6381、6382、6384来运行4个Redis实例。

注意：为了避免不必要的错误，配置文件尽量保存为utf8格式，并且不要包含注释；

      配置文件中以下两种保存日志的方式(保存在文件中、保存到System Log中)请根据需求选择其中一种即可：

　　 loglevel notice #日志的记录级别，notice是适合生产环境的

logfile "D:/Redis/Logs/redis6380\_log.txt" #指定log的保持路径,默认是创建

在Redis安装目录下，如果有子目录需要手动创建，如此处的Logs目录

　 syslog-enabled yes #是否使用系统日志 　　  
　　 syslog-ident redis6380 #在系统日志的标识名

这里使用了保存在文件中的方式，所以先在Redis目录下新建Logs文件夹用于存放log文件



另新建四个conf文件,分别命名为redis.6380.conf; redis.6381.conf;

redis.6382.conf; redis.6383.conf;

redis.6380.conf 内容如下：  
port 6380

loglevel notice

logfile "D:/Redis/Logs/redis6380\_log.txt"

appendonly yes

appendfilename "appendonly.6380.aof"

cluster-enabled yes

cluster-config-file nodes.6380.conf

cluster-node-timeout 15000

cluster-slave-validity-factor 10

cluster-migration-barrier 1

cluster-require-full-coverage yes

redis.6381.conf 内容如下：  
port 6381

loglevel notice

logfile "D:/Redis/Logs/redis6381\_log.txt"

appendonly yes

appendfilename "appendonly.6381.aof"

cluster-enabled yes

cluster-config-file nodes.6381.conf

cluster-node-timeout 15000

cluster-slave-validity-factor 10

cluster-migration-barrier 1

cluster-require-full-coverage yes

redis.6382.conf 内容如下：  
port 6382

loglevel notice

logfile "D:/Redis/Logs/redis6382\_log.txt"

appendonly yes

appendfilename "appendonly.6382.aof"

cluster-enabled yes

cluster-config-file nodes.6382.conf

cluster-node-timeout 15000

cluster-slave-validity-factor 10

cluster-migration-barrier 1

cluster-require-full-coverage yes

redis.6383.conf 内容如下：  
port 6383

loglevel notice

logfile "D:/Redis/Logs/redis6383\_log.txt"

appendonly yes

appendfilename "appendonly.6383.aof"

cluster-enabled yes

cluster-config-file nodes.6383.conf

cluster-node-timeout 15000

cluster-slave-validity-factor 10

cluster-migration-barrier 1

cluster-require-full-coverage yes

配置内容解释如下：

port 6380 #端口号

loglevel notice #日志的记录级别，notice是适合生产环境的

logfile "Logs/redis6380\_log.txt" #指定log的保持路径,默认是创建在Redis安装目录下，如果有子目录需要手动创建，如此处的Logs目录

syslog-enabled yes #是否使用系统日志

syslog-ident redis6380 #在系统日志的标识名

appendonly yes #数据的保存为aof格式

appendfilename "appendonly.6380.aof" #数据保存文件

cluster-enabled yes #是否开启集群

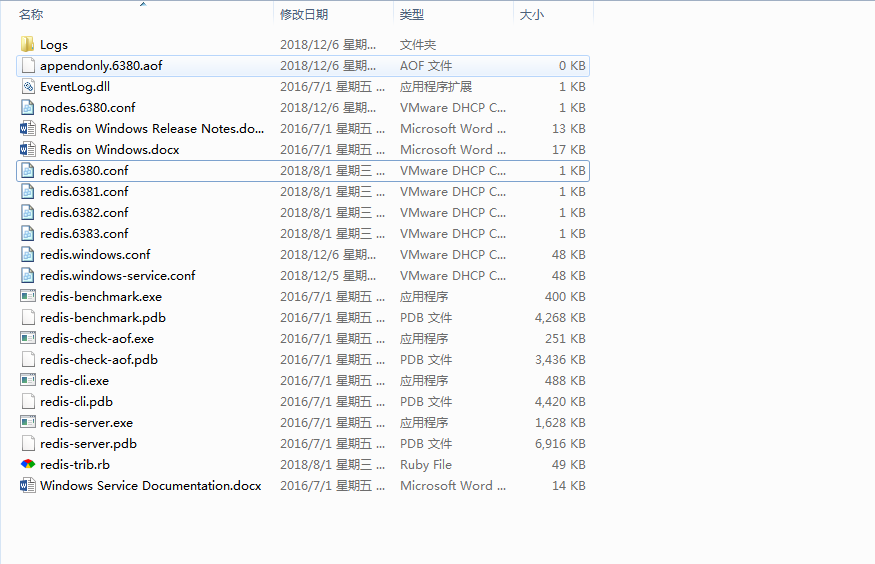
cluster-config-file nodes.6380.conf

cluster-node-timeout 15000

cluster-slave-validity-factor 10

cluster-migration-barrier 1

cluster-require-full-coverage yes



然后在命令窗口中运行以下指令，将4个redis服务添加到windows中并启动

C:/redis/redis-server.exe --service-install C:/redis/redis.6380.conf --service-name redis6380

C:/redis/redis-server.exe --service-install C:/redis/redis.6381.conf --service-name redis6381

C:/redis/redis-server.exe --service-install C:/redis/redis.6382.conf --service-name redis6382

C:/redis/redis-server.exe --service-install C:/redis/redis.6383.conf --service-name redis6383

C:/redis/redis-server.exe --service-start --service-name redis6380

C:/redis/redis-server.exe --service-start --service-name redis6381

C:/redis/redis-server.exe --service-start --service-name redis6382

C:/redis/redis-server.exe --service-start --service-name redis6383

注意，redis安装目录地址不一样，上面的访问目录就需要进行更改

启动后，则会在windows服务中看到

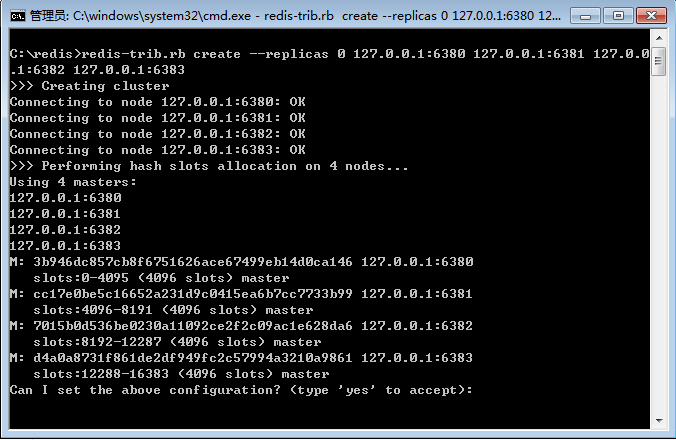


接着就可以开始创建集群了：

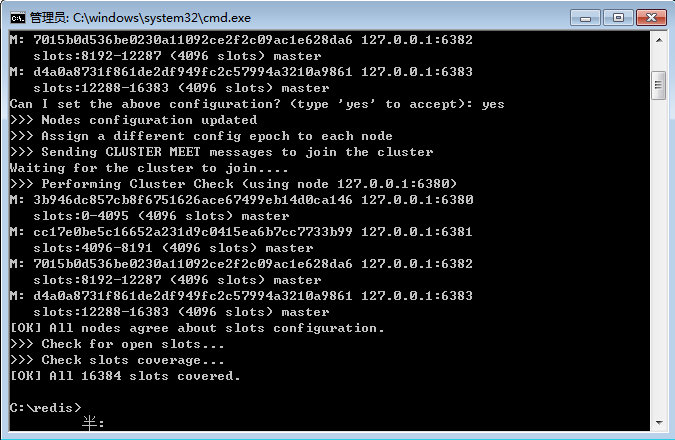
在redis目录下启动cmd命令窗口，输入（注意redis目录中redis-trib.rb的图标是）：

redis-trib.rb create --replicas 0 127.0.0.1:6380 127.0.0.1:6381 127.0.0.1:6382 127.0.0.1:6383

运行



输入yes



此时，在同一台机器上安装多个redis服务集群成功！

测试一下：

在redis目录下启动命令行窗口，输入redis-cli.exe –c –p 6380

运行

然后输入cluster info

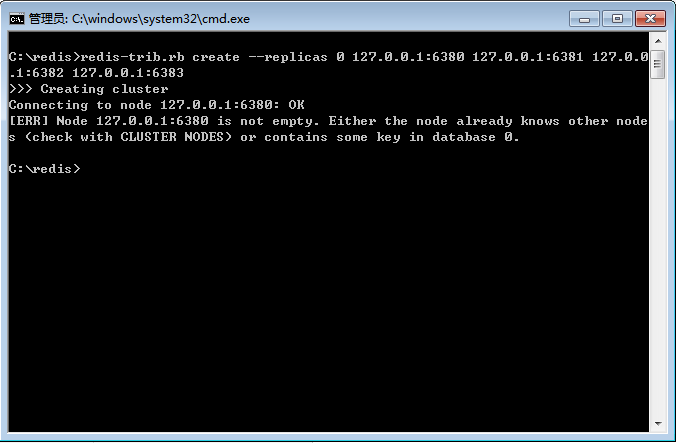


可看到相关信息，然后通过set key \*\*\*;get key;等指令可看到如图所示



Redis相互直接可以进行访问！

注意：经过如上配置后，启动服务会在redis目录中相应生成appendonly.\*\*\*.aof,nodes.\*\*\*.conf文件，这些文件在你下次重新配置集群时造成印象。如：关闭集群cmd，再打开运行集群指令时。可以看到



此时，关闭四个redis服务，将上述的appendonly.\*\*\*.aof,nodes.\*\*\*.conf文件删除，然后重启四个服务即可，注意，不要误删redis.windows.conf文件，然后再次在cmd窗口中运行集群指令即可成功！

### 2）在多台机器（每台机器都有一到多个redis服务）中集群redis（主要使用）

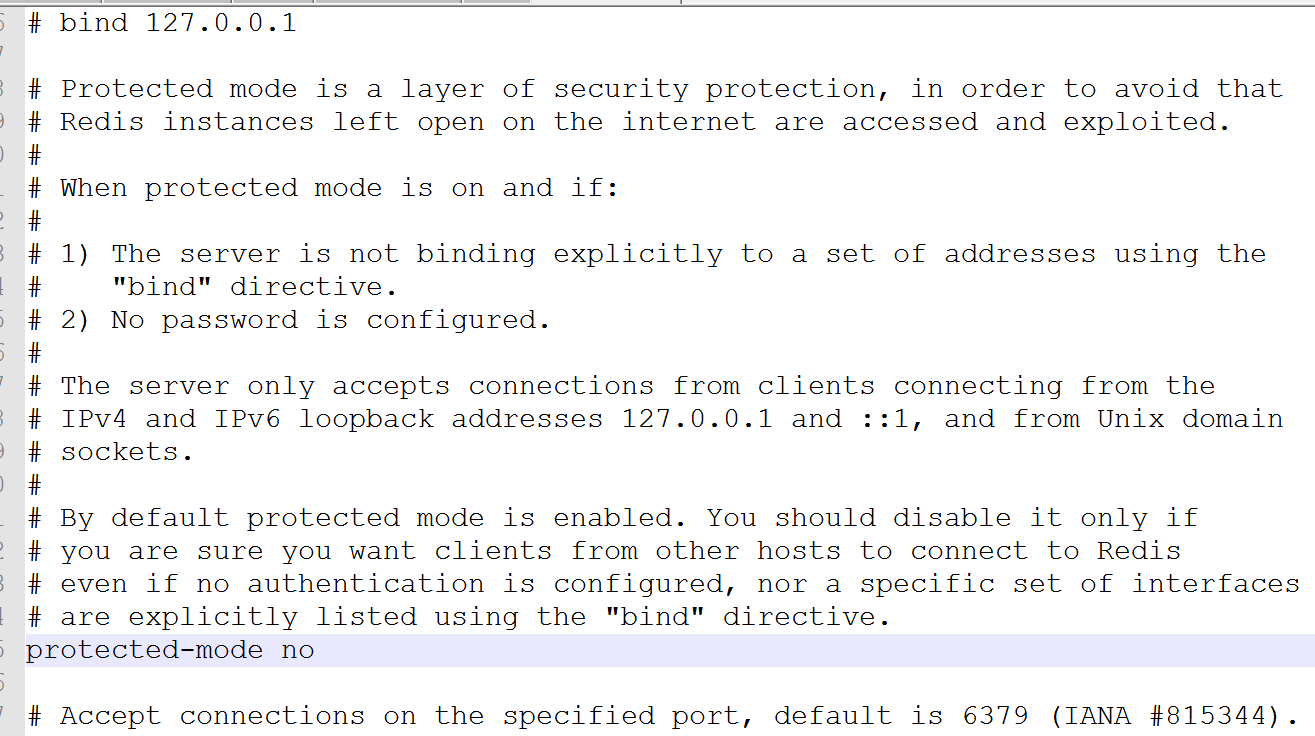
注意事项：与上述在同一台机器上安装多个redis服务集群不同的是，在多台机器中集群redis，需要你能对其他机器的redis进行访问。因此需要进行一点改动！在多台机器中集群redis，ruby环境只需安装在一台机器上即可，接下来的操作是在本机192.168.22.11安装ruby环境，集群192.168.22.24和192.168.22.108两台机器的redis服务！（redis集群至少需要三个redis服务）

首先，需要更改每台机器redis的访问ip（包括本机）。前面提到过，redis.windows.conf是默认以127.0.0.1为IP，6379为端口号。既然要集群多台机器的redis服务，则需要通过相应电脑的ip进行访问！因此，请注意redis.windows.conf文件中的以下内容：

Bind 127.0.0.1

protected-mode yes

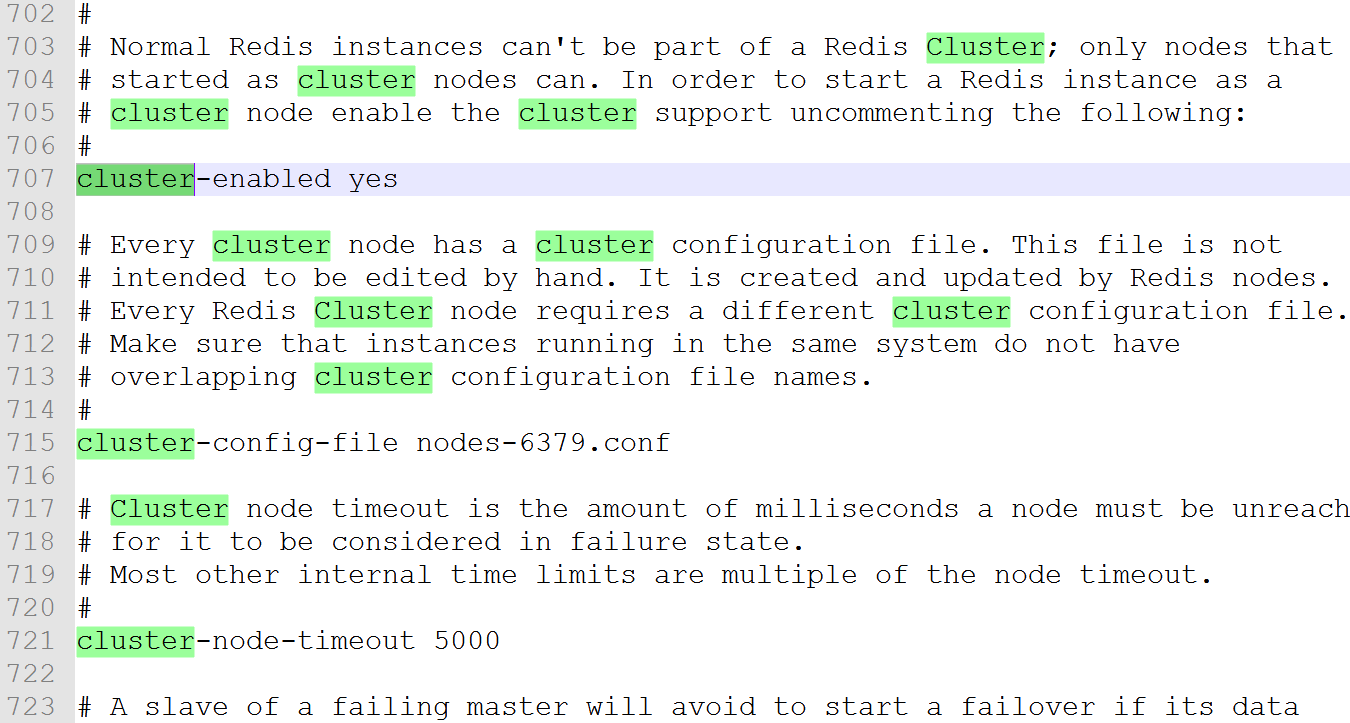
此时只需在bind 127.0.0.1前加上#，将protected-mode后的yes改为no，然后重启redis服务，即可让其他机器通过ip端口号对本机器redis进行访问。可在另外一台机器的redis目录下，启动命令行窗口，输入redis-cli.exe –h \*访问ip\* -p \*端口号\*，进行测试。注意：三台机器的redis服务均要进行此更改，然后重启。

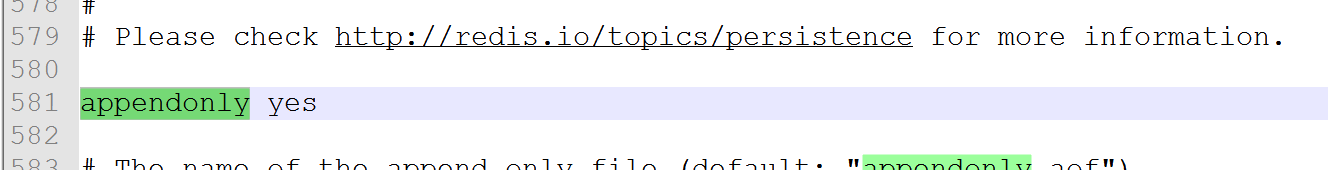


此外需要修改三个redis服务中有关集群的内容，在redis.windows.conf中查看以下参数：

cluster-enabled yes  
cluster-config-file nodes.conf  
cluster-node-timeout 5000  
appendonly yes

将参数后面的值分别改为以上内容



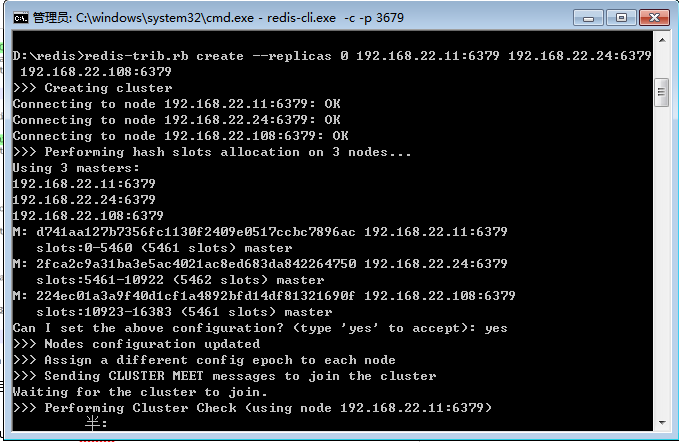


三台机器的redis服务均要进行此更改，然后重启。（类似于在同一台机器上安装多个redis服务集群中新建的redis.6380.conf中的配置）

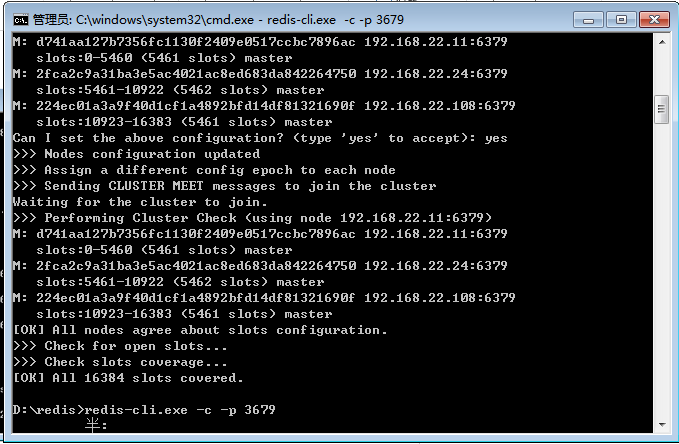
在安装ruby环境的redis目录中（此文档安装ruby环境的redis目录在本机192.168.22.11中），在redis目录下启动cmd命令窗口，输入（注意redis目录中redis-trib.rb的图标是）：

redis-trib.rb create --replicas 0 192.168.22.11:6379 192.168.22.24:6379 192.168.22.108：6379

运行

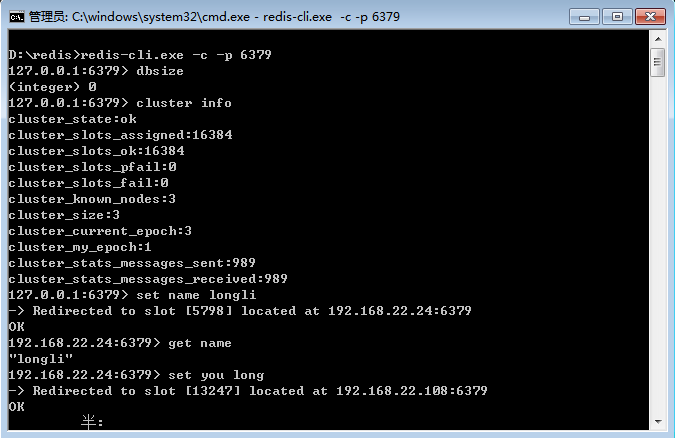


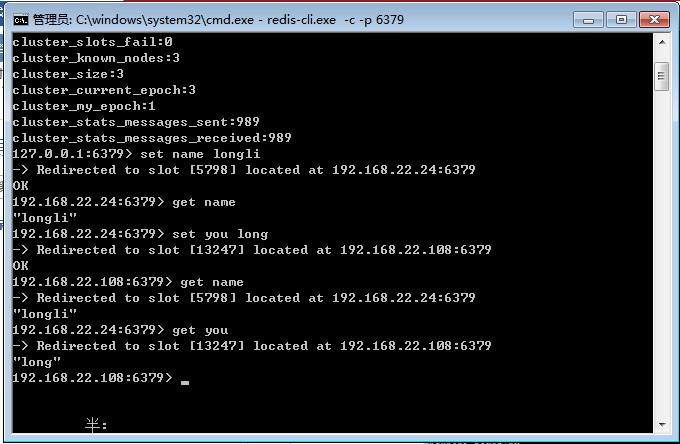
输入yes



如上所示，redis集群完成。

可在redis目录中启动命令行窗口，执行以下指令进行测试：





可以看到三台redis服务可进行数据共享！

## 7.Redis集群—主从模式

redis单例提供了一种数据缓存方式和丰富的数据操作api，但是将数据完全存储在单个redis中主要存在两个问题：数据备份和数据体量较大造成的性能降低。这里redis的主从模式为这两个问题提供了一个较好的解决方案。主从模式指的是使用一个redis实例作为主机，其余的实例作为备份机。主机和从机的数据完全一致，主机支持数据的写入和读取等各项操作，而从机则只支持与主机数据的同步和读取，也就是说，客户端可以将数据写入到主机，由主机自动将数据的写入操作同步到从机。主从模式很好的解决了数据备份问题，并且由于主从服务数据几乎是一致的，因而可以将写入数据的命令发送给主机执行，而读取数据的命令发送给不同的从机执行，从而达到读写分离的目的。

\*一个Redis服务可以有多个该服务的复制品，这个Redis服务称为Master，其他复制称为Slaves

\*只要网络连接正常，Master会一直将自己的数据更新同步给Slaves，保持主从同步

\*只有Master可以执行写命令，Slaves只能执行读命令

\*客户端可以连接Slaves执行读请求，来降低Master的读压力

**简单的主从配置**：

在redis目录下，新建三个文件，分别为：redis.6385.conf;redis.6386.conf;redis.6387.conf;(前面的测试中已将6379-6384端口占用，这里只要是没有被占用的端口号都可以用)，编辑其中内容分别为：

Redis.6385.conf:

bind 127.0.0.1

port 6385

logfile "6385.log"

dbfilename "dump-6385.rdb"

Redis.6386.conf:

bind 127.0.0.1

port 6386

logfile "6386.log"

dbfilename "dump-6386.rdb"

Redis.6387.conf:

bind 127.0.0.1

port 6387

logfile "6387.log"

dbfilename "dump-6387.rdb"

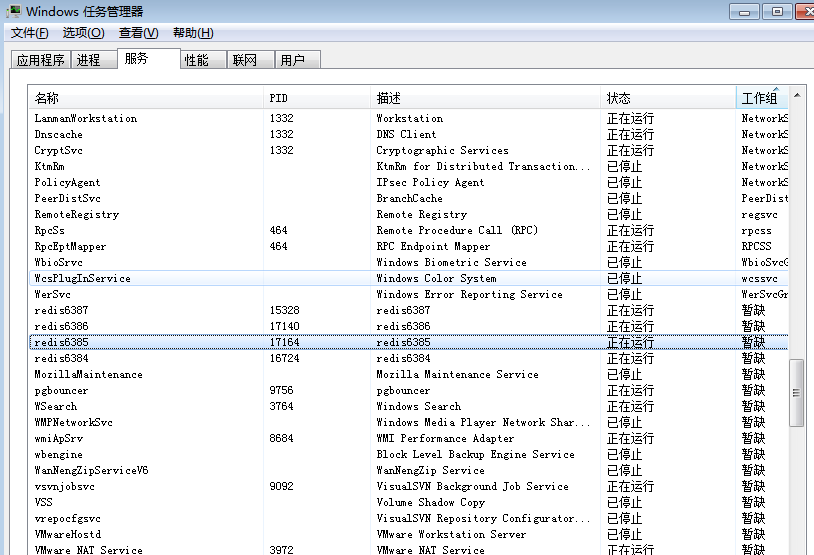
然后在redis目录下启动命令行窗口，执行以下命令将这三个redis服务添加到windows服务中：

redis-server.exe --service-install redis.6385.conf --service-name redis6385

redis-server.exe --service-install redis.6386.conf --service-name redis6386

redis-server.exe --service-install redis.6387.conf --service-name redis6387

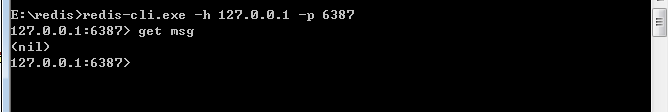
然后启动这三个服务



在redis目录下启动三个命令行窗口，分别输入redis-cli.exe –h 127.0.0.1 –p 6385；redis-cli.exe –h 127.0.0.1 –p 6386；redis-cli.exe –h 127.0.0.1 –p 6387分别进入到三个redis服务中，分别输入get msg可以看到





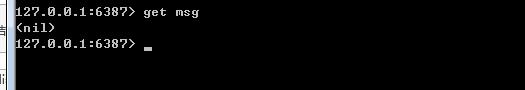


说明三个redis实例中都不存在键为msg的数据。

现在我们在6385中添加一个键为msg的数据



此时其余的6386,6387redis服务是访问不到这个数据的

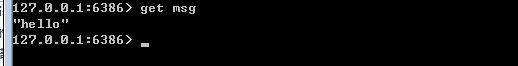


不过我们设置主从关系后，就不一样了：

在6386的命令行窗口中，执行slaveof 127.0.0.1 6385



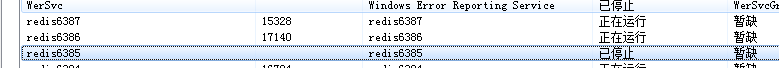
然后get msg可以看到



同样对6387的redis服务进行同样操作也能看到



此时我们将6385（主）关掉



看6386,6387是否还能取到值



取值成功！

## 8.Redis集群方案简介

Node为redis节点，也就是一个redis服务，slots为分配在redis服务中的数据，master主节点，slave从节点

安装上面的集群部署，我们不难发现，一旦将其中的一台redis关闭（生产中可能会出故障关闭），整个redis集群将会失效！如果某个node发生故障，那它负责的slots也就失效，整个Redis Cluster将不能工作。因此官方推荐的方案是将node都配置成主从结构，即一个master主节点，挂n个slave从节点。

**关于redis的集群化方案 目前有三种**

（1）Twitter开发的twemproxy

（2）豌豆荚开发的codis

（3）redis官方的redis-cluster

简介：twemproxy架构简单，就是用proxy对后端redis server进行代理。但是由于代理层的消耗性能很低，而且通常涉及多个key的操作都是不支持的，而且本身不支持动态扩容和透明的数据迁移，而且也失去维护 Twitter内部已经不使用了。

      redis-cluster是三个里性能最强大的，因为他使用去中心化的思想，使用hash slot方式 将16348个hash slot覆盖到所有节点上，对于存储的每个key值 使用CRC16(KEY)&16348=slot 得到他对应的hash slot 并在访问key时就去找他的hash slot在哪一个节点上 然后由当前访问节点从实际被分配了这个hash slot的节点去取数据 节点之间使用轻量协议通信 减少带宽占用 性能很高 自动实现负载均衡与高可用 自动实现failover  并且支持动态扩展 官方已经玩到可以1000个节点 实现的复杂度低 总之个人比较喜欢这个架构 因为他的去中心化思想免去了proxy的消耗，是全新的思路。

\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*

hash slot（哈希槽）

Redis 集群没有并使用传统的一致性哈希来分配数据，而是采用另外一种叫做哈希槽 (hash slot)的方式来分配的。redis cluster 默认分配了 16384 个slot，当我们set一个key 时，会用CRC16算法来取模得到所属的slot，然后将这个key 分到哈希槽区间的节点上，具体算法就是：CRC16(key) % 16384。

所以，我们假设现在有3个节点已经组成了集群，分别是：A, B, C 三个节点，它们可以是一台机器上的三个端口，也可以是三台不同的服务器。那么，采用哈希槽 (hash slot)的方式来分配16384个slot 的话，它们三个节点分别承担的slot 区间是：

* 节点A覆盖0－5460;
* 节点B覆盖5461－10922;
* 节点C覆盖10923－16383.

这种将哈希槽分布到不同节点的做法使得用户可以很容易地向集群中添加或者删除节点。 比如说：

* 如果用户将新节点 D 添加到集群中， 那么集群只需要将节点 A 、B 、 C 中的某些槽移动到节点 D 就可以了。

比如我想新增一个节点D，redis cluster的这种做法是从各个节点的前面各拿取一部分slot到D上。大致就会变成这样：

* + 节点A覆盖1365-5460
  + 节点B覆盖6827-10922
  + 节点C覆盖12288-16383
  + 节点D覆盖0-1364,5461-6826,10923-12287
* 与此类似， 如果用户要从集群中移除节点 A ， 那么集群只需要将节点 A 中的所有哈希槽移动到节点 B 和节点 C ， 然后再移除空白（不包含任何哈希槽）的节点 A 就可以了。

因为将一个哈希槽从一个节点移动到另一个节点不会造成节点阻塞， 所以无论是添加新节点还是移除已存在节点， 又或者改变某个节点包含的哈希槽数量， 都不会造成集群下线。

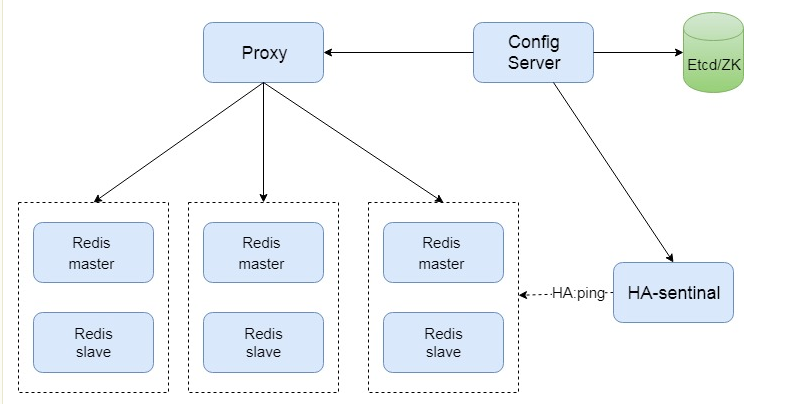
\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*

      但是它也有一些不足 例如官方没有提供图形化管理工具 运维体验差 全手工数据迁移 并且自己对自己本身的redis命令支持也不完全等 但是这些问题 我觉得不能掩盖他关键的新思想所带来的的优势 随着官方的推进 这些问题应该都能在一定时间内得到解决 那么这时候去中心化思想带来的高性能就会表现出他巨大的优势

      codis使用的也是proxy思路 但是做的比较好 是这两种之间的一个中间级 而且支持redis命令是最多的 有图形化GUI管理和监控工具 运维友好 这个过段时间会详细另外写出来原理 工作机制和搭建实现。

这里着重讲解codis方案与官方的redis-cluster方案！

### 类codis架构



这套架构的特点：

·分片算法：基于 slot hash桶；

·分片实例之间相互独立，每组一个master 实例和多个slave；

·路由信息存放到第三方存储组件，如 zookeeper 或etcd

·旁路组件探活

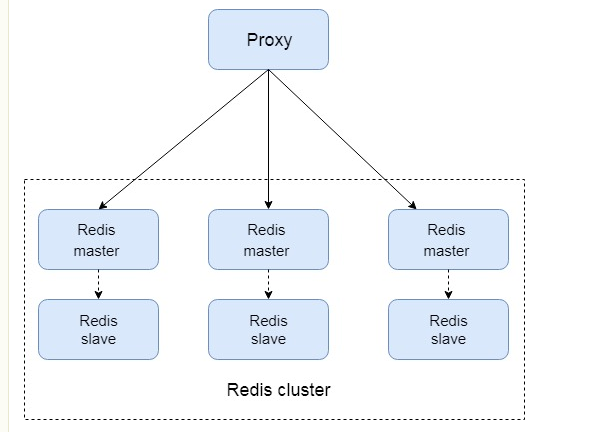
### 基于官方 redis cluster 方案

1、官方推荐，毋庸置疑。

　　2、去中心化，集群最大可增加1000个节点，性能随节点增加而线性扩展。

　　3、管理方便，后续可自行增加或摘除节点，移动分槽等等。

　　4、简单，易上手。



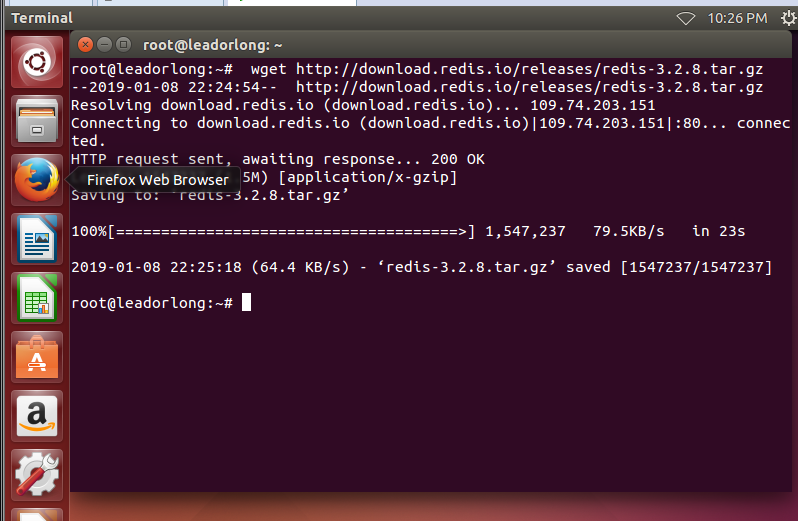
和上一套方案比，所有功能都集成在 redis cluster 中，路由分片、拓扑信息的存储、探活都在redis cluster中实现；各实例间通过 gossip 通信；这样的好处是简单，依赖的组件少，应对400个节点以内的场景没有问题（按单实例8w read qps来计算，能够支持 200 \* 8 = 1600w 的读多写少的场景）；但当需要支持更大的规模时，由于使用 gossip协议导致协议之间的通信消耗太大，redis cluster 不再合适；

## 9.Redis集群方案实现（基于linux操作系统）

### 1）redis-cluster实现

下载redis:

wget http://download.redis.io/releases/redis-3.2.8.tar.gz

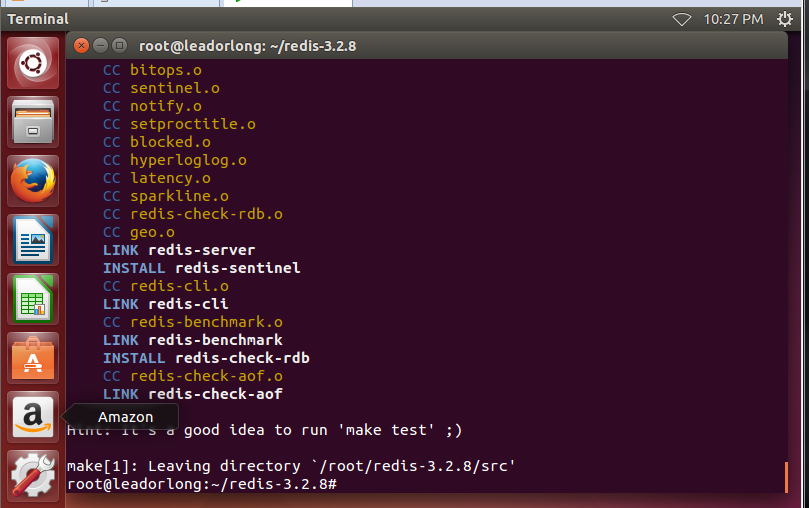


安装redis:

tar xzf redis-3.2.8.tar.gz

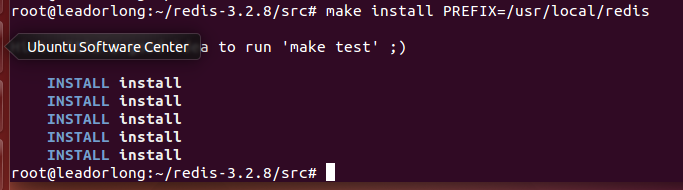
cd redis-3.2.8

make



cd src

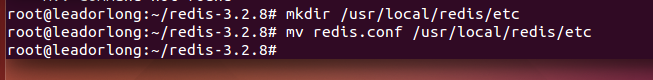
　　 make install PREFIX=/usr/local/redis



cd ../

　　mkdir /usr/local/redis/etc

　　mv redis.conf /usr/local/redis/etc



然后配置redis为后台启动

vi /usr/local/redis/etc/redis.conf

将**daemonize no** 改成daemonize yes

然后即可调用如下命令开启redis

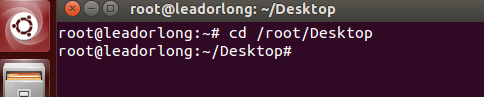
/usr/local/redis/bin/redis-server /usr/local/redis/etc/redis.conf



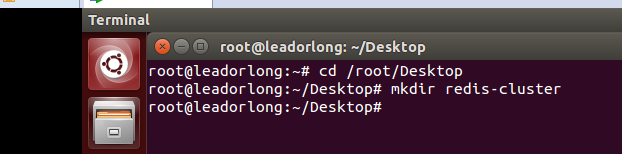
此处我们采用一台机器创建多个redis实例进行集群：

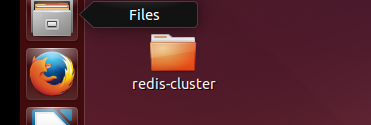
首先创建一个redis-cluster目录(演示是在桌面中创建redis-cluster)：

cd /root/Desktop //跳转到桌面目录

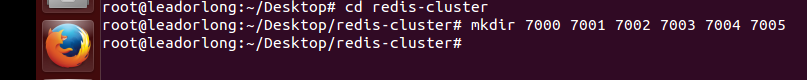


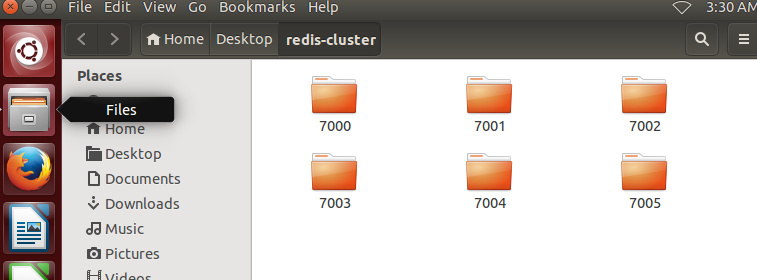
创建redis-cluster文件夹





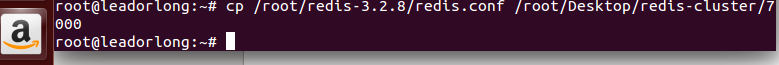
然后切换到redis-cluster目录，创建6个文件夹，分别命名为7000,7001,7002,7003,7004,7005

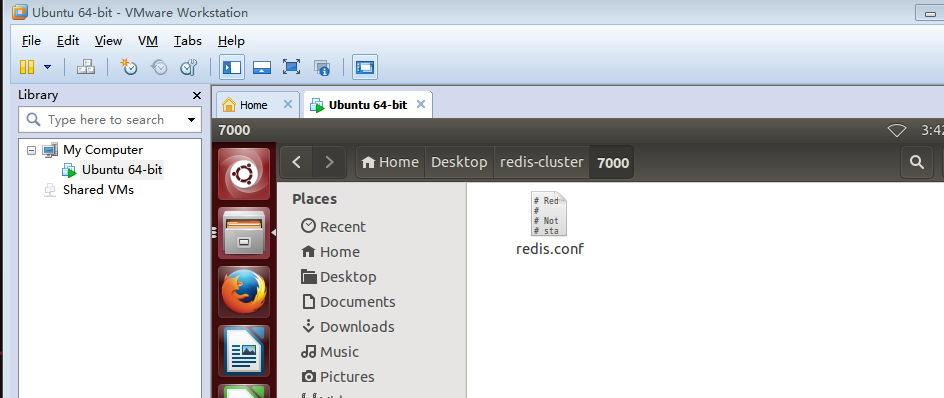




目录创建好后，我们把Redis源文件里面包含的配置文件redis.conf拷贝一份，存放在7000目录下，然后对其配置项进行修改，这个配置文件Redis.conf会作为其他Redis实例的配置文件的模板，并拷贝到其他目录。

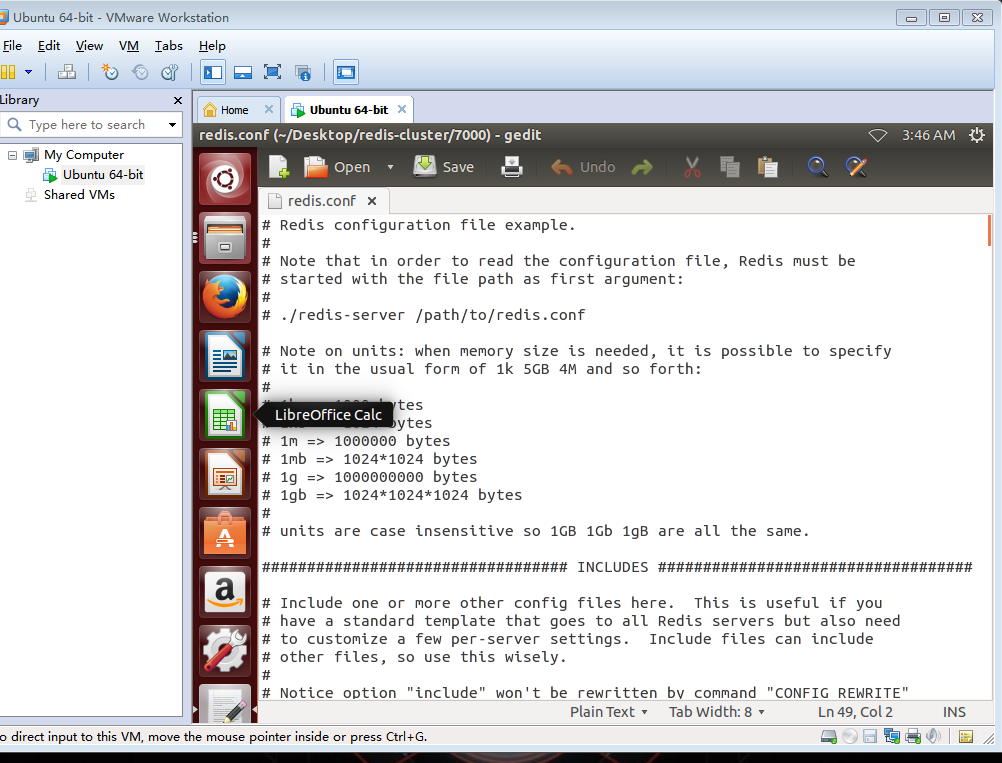
cp /root/redis-3.2.8/redis.conf /root/Desktop/redis-cluster/7000





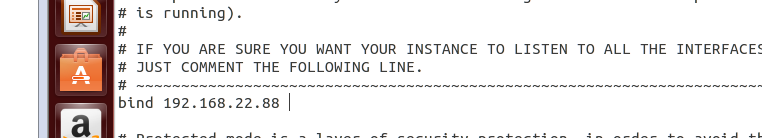
由于我们是做测试，并没有启动6个真正的物理节点，而是把6个Redis实例都部署在了同一台Linux服务器上，地址：**192.168.127.130**，为了区分Redis实例，我们是以不同的端口号来区分Redis实例的。然后我们修改Redis.conf的配置文件。（可通过指令进行修改，也可直接打开文件编辑修改）

为了方便起见，我们直接打开编辑：

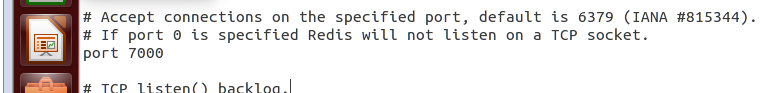


修改项如下（中文注释不要加进去）：

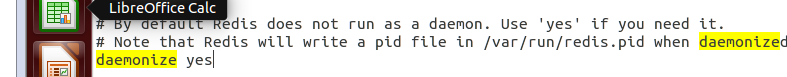
bind 192.168.22.88 //绑定服务器IP地址（22.88是我linux本地的ip）



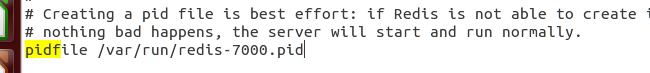
　　　 port 7000 //绑定端口号，必须修改，以此来区分Redis实例



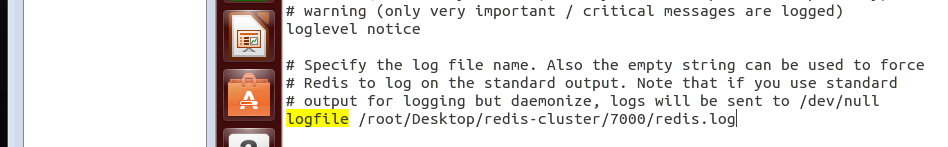
　　　 daemonize yes //后台运行



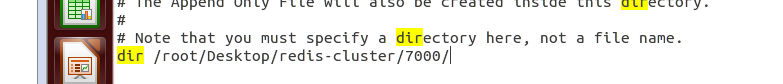
　　　 pidfile /var/run/redis-7000.pid //修改pid进程文件名，以端口号命名



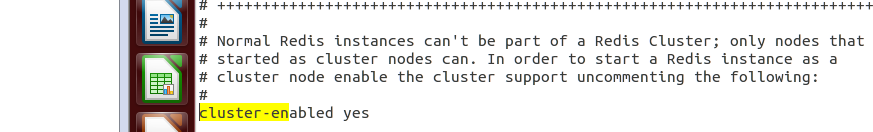
　　　logfile /root/Desktop/redis-cluster/7000/redis.log //修改日志文件名称，以端口号为目录来区分



　　　dir /root/Desktop/redis-cluster/7000/ //修改数据文件存放地址，以端口号为目录名来区分

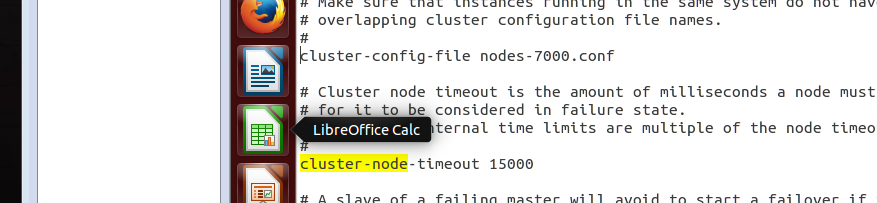


　　　cluster-enabled yes //启用集群

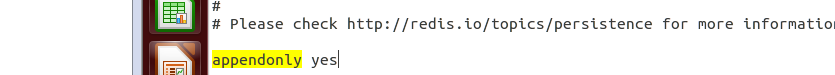


　　　cluster-config-file nodes-7000.conf //配置每个节点的配置文件，同样以端口号为名称

　　　cluster-node-timeout 15000 //配置集群节点的超时时间，可改可不改



　　　appendonly yes //启动AOF增量持久化策略



　　　appendfsync always //发生改变就记录日志

接着讲7000目录中redis.conf文件拷贝至7001,7002,7003,7004,7005目录中，并将文件中的7000改为相对应的端口号，比如拷贝至7001中后，修改7001目录中的redis.conf文件中的port 7000，为port 7001,修改cluster-config-file nodes-7000.conf 为cluster-config-file nodes-7001.conf.此处同样可以分别打开修改，也可用指令进行处理。为确保所有包含7000的字段都替换为自身端口号，建议此处用指令进行修改：

首先复制redis.conf到7001,7002,7003,7004,7005中：

指令切换到7000目录中

cp redis.conf ../7001

cp redis.conf ../7002

cp redis.conf ../7003

cp redis.conf ../7004

cp redis.conf ../7005



然后修改每个文件redis.conf中的7000为相应的端口：

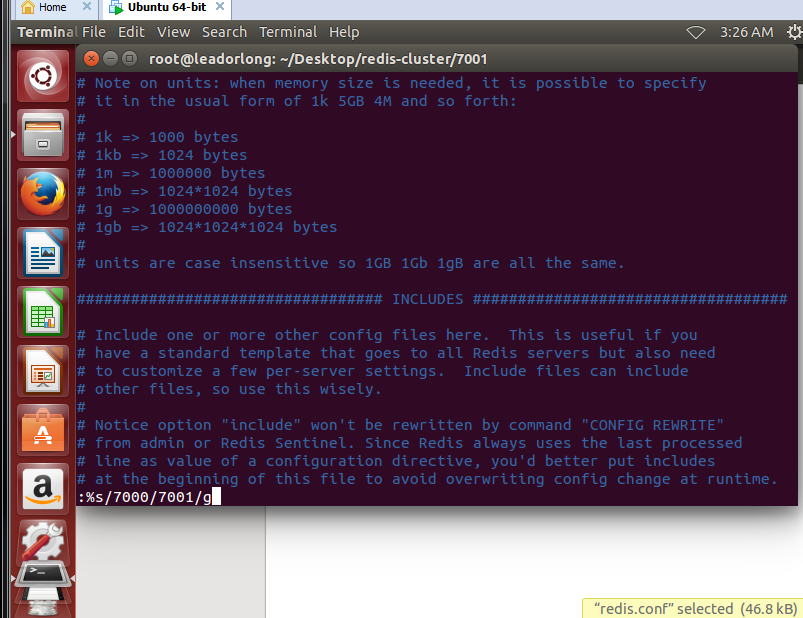
**：%s/7000/7001/g**

**：%s/7000/7002/g**

**：%s/7000/7003/g**

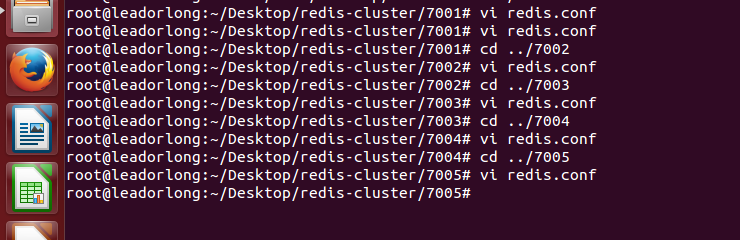
**：%s/7000/7004/g**

**：%s/7000/7005/g**



然后输入：wq保存

对其余文中的redis.conf进行同样操作！



接着，安装ruby环境

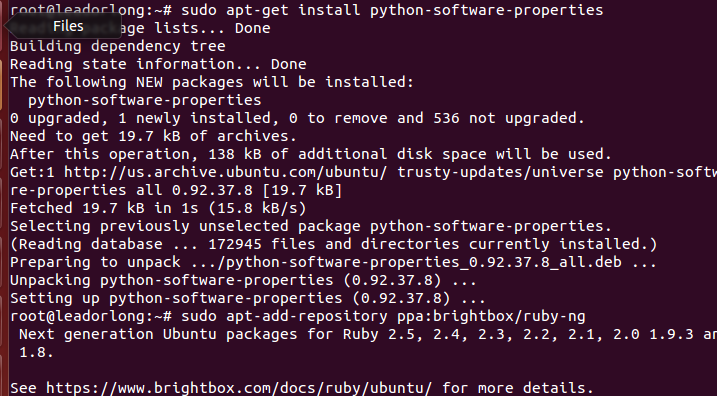
使用brightbox ppa仓库安装

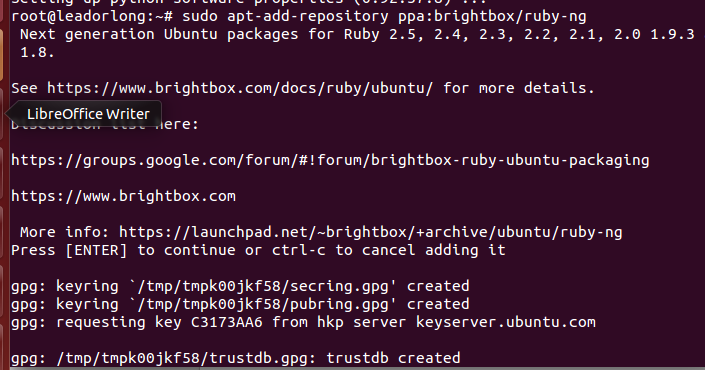
apt-get install python-software-properties

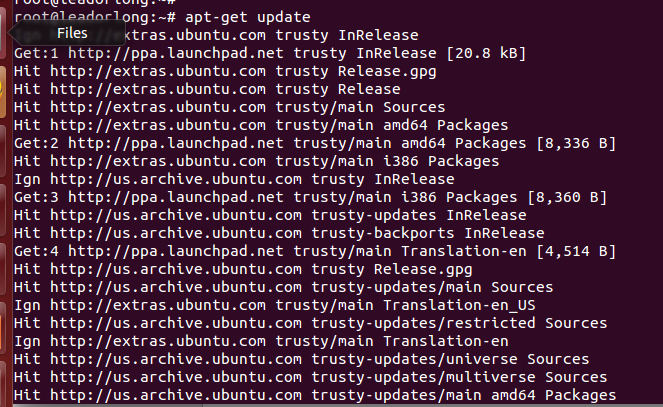
apt-add-repository ppa:brightbox/ruby-ng

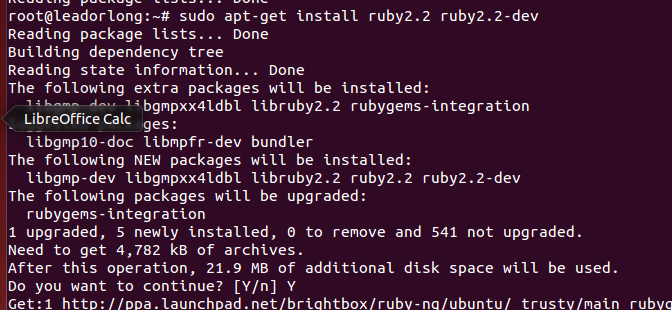
apt-get update

apt-get install ruby2.2 ruby2.2-dev

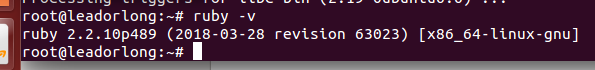






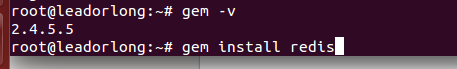


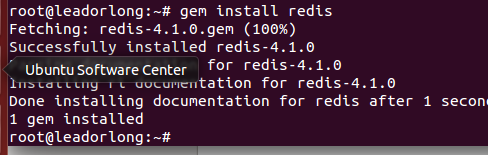
输入ruby –v查看是否安装完成并查看版本



**安装完 Ruby 和 Rubygems 后，还需要继续安装Redis的Ruby接口程序。**

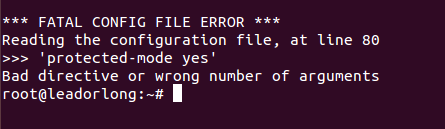
　gem install redis





**开始启动我们6个Redis实例，并且要指定配置文件，这些配置文件分别在各自的子目录下面**

**使用redis-server /root/Desktop/redis-cluster/7000/redis.conf进行启动，会报错**



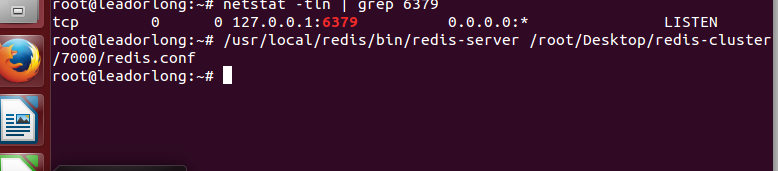
之前使用下载安装包自己编译的，后来输入redis-server无效，然后又使用apt-get安装了redis，两次版本不一致导致conf文件的某些字段老版本不支持（测试是本机apt-get安装的是2.8.4版本，安装包为3.2.8版本），所以就删除apt-get remove redis-server



**使用**

/usr/local/redis/bin/redis-server /root/Desktop/redis-cluster/7000/redis.conf

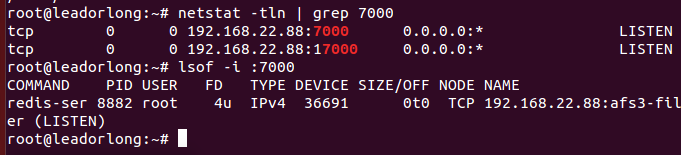
**开启端口号为7000的redis服务**



**此时我们可以通过**

netstat -tln | grep 7000 //查找被占用的端口

lsof -i :7000 //查看端口属于哪个程序



**为了方便起见，我们一次性开启其余5个服务**

/usr/local/redis/bin/redis-server /root/Desktop/redis-cluster/7001/redis.conf

/usr/local/redis/bin/redis-server /root/Desktop/redis-cluster/7002/redis.conf

/usr/local/redis/bin/redis-server /root/Desktop/redis-cluster/7003/redis.conf

/usr/local/redis/bin/redis-server /root/Desktop/redis-cluster/7004/redis.conf

/usr/local/redis/bin/redis-server /root/Desktop/redis-cluster/7005/redis.conf

然后通过ps -ef | grep redis指令可看到所有redis服务



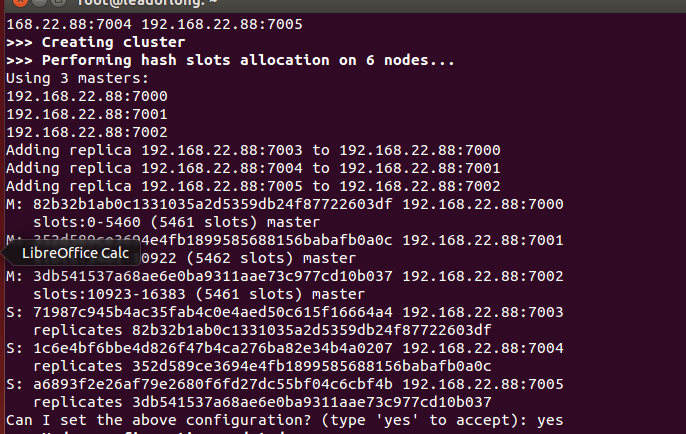
上面6379为非集群redis，可忽略

**现在六个服务均已启动，开始进行集群：**

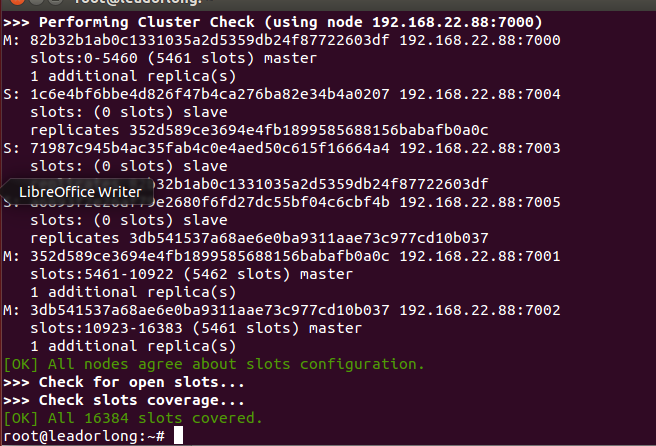
**执行：**

ruby /root/redis-3.2.8/src/redis-trib.rb create --replicas 1 192.168.22.88:7000 192.168.22.88:7001 192.168.22.88:7002 192.168.22.88:7003 192.168.22.88:7004 192.168.22.88:7005

(这里使用的命令是create，因为我们想创建一个新的集群。 选项--replicas 1 意味着我们需要为每个创建的主服务器节点创建一个从服务器节点。)



**输入yes**

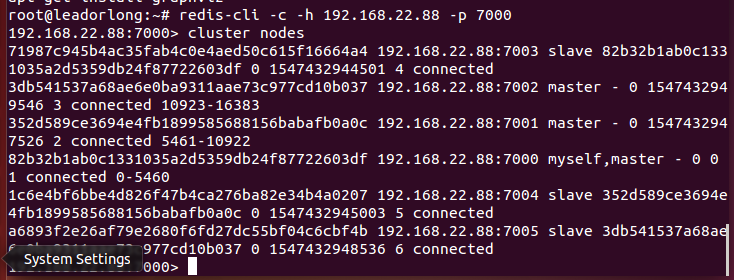


**上面我们可以看到 7000,7001,7002三个为主节点（M），其余三个为从节点（S）**

### ****2）关于集群节点的操作及说明****

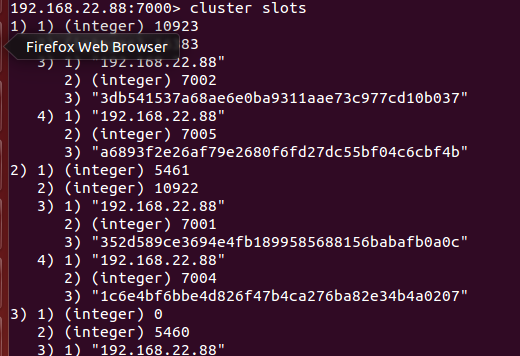
#### ****I.查看集群信息****

**Redis-cli进入某一个redis节点，然后使用cluster nodes指令即可查看**



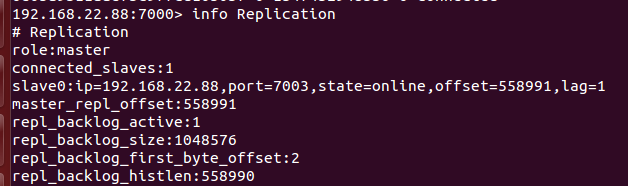
#### ****II.查看集群主从关系****

**Redis-cli进入某一个redis节点，然后使用cluster slots指令即可查看**



**从上可以看到redis集群的主从关系以及分配的slots节点**

**也可以通过info Replication指令查看到当前redis服务对应的主从角色和主从对象**



#### ****III.防止集群故障****

**先来看一下key为name的在哪一个redis服务中**

**现在我们模拟将7001节点挂掉**

**ps -ef | grep redis**

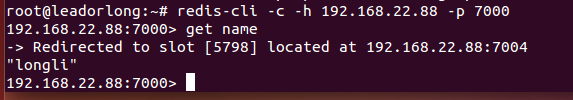
**kill 7001ID**





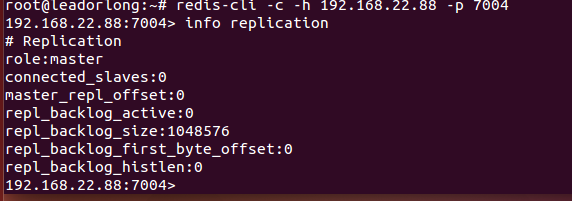
**现在可以看到7001节点已被kill**

**我们再来查看一下key为name的值**



**这里就跑到7004中去了，我们现在来看看7004的集群信息**

**Info replication**



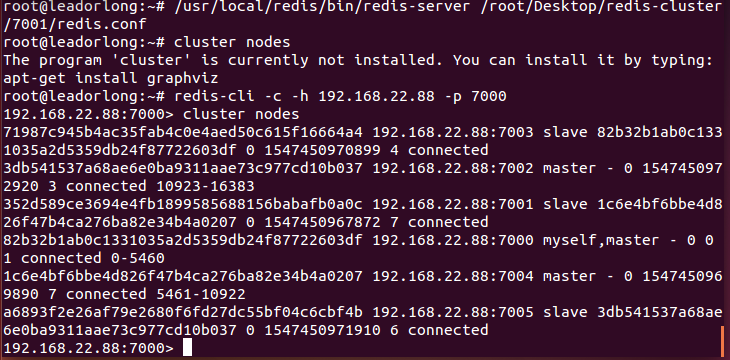
**这个时候7004节点就由从节点变成了主节点**

**我们现在再来恢复一下7001节点**

/usr/local/redis/bin/redis-server /root/Desktop/redis-cluster/7001/redis.conf

**再来看一看redis集群的主从关系**

**Cluster nodes**



**可以看到7001是从节点，7004继续保持主节点的身份**

#### ****IV.集群瘫痪发生情况---投票****

1. 投票过程是集群中所有master参与,如果半数以上master节点与master节点通信超时(cluster-node-timeout),认为当前master节点挂掉.
2. 什么时候整个集群不可用(cluster\_state:fail)?
   * 如果集群任意master挂掉,且当前master没有slave.集群进入fail状态,也可以理解成集群的slot映射[0-16383]不完整时进入fail状态.
     + redis-3.0.0.rc1加入cluster-require-full-coverage参数,默认关闭,打开集群兼容部分失败.
   * 如果集群超过半数以上master挂掉，无论是否有slave，集群进入fail状态.

#### ****V.重启集群时异常****

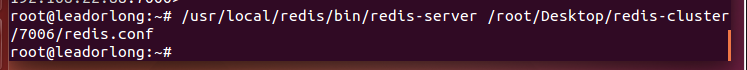
**重启集群可能会出现node: \*\*\*\*\*\*\*\*\*\*:\*\* is not empty,解决办法是删除原来生产的一些配置文件，如.aof,.rdb文件，删除后异常将解决**

#### ****VI.添加节点****

**我们新建一个端口为7006的节点，跟上面新建7000-7005节点一样（直接在原来文件夹中新建一个7006文件夹，然后将其他文件夹中的redis.conf文件粘贴复制到7006目录下，将其中的700\*换为7006）。**

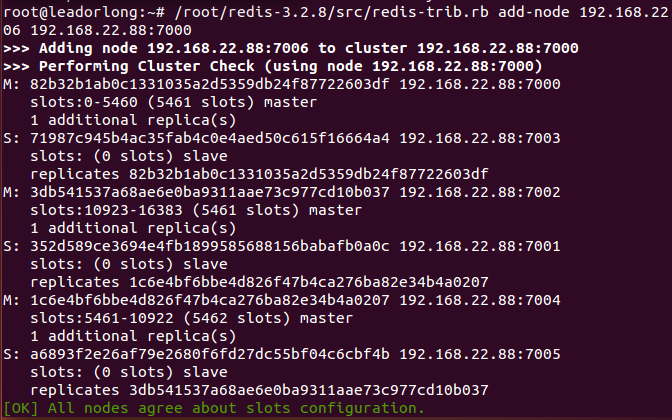
**现在我们启动7006节点的redis服务**

/usr/local/redis/bin/redis-server /root/Desktop/redis-cluster/7006/redis.conf



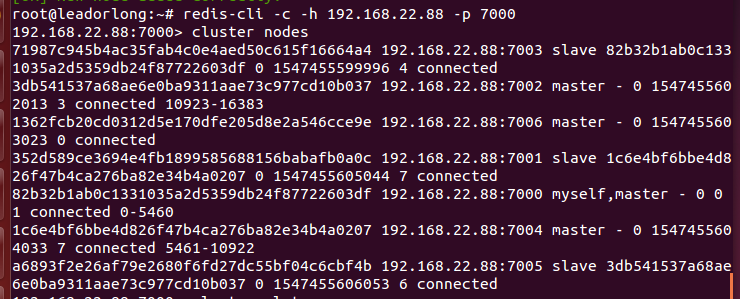
**（1）.将7006作为主节点节点添加进去**

**/root/redis-3.2.8/src/redis-trib.rb add-node 192.168.22.88:7006 192.168.22.88:7000**

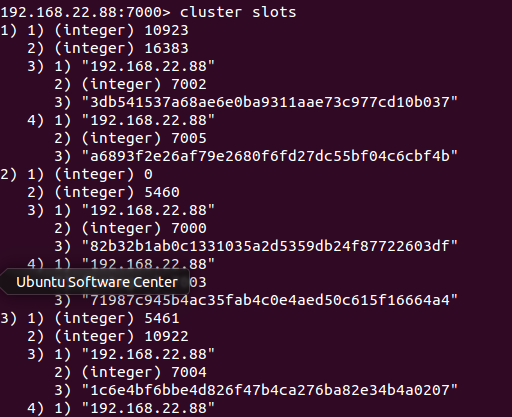




**现在我们可以通过cluster nodes查看集群信息**



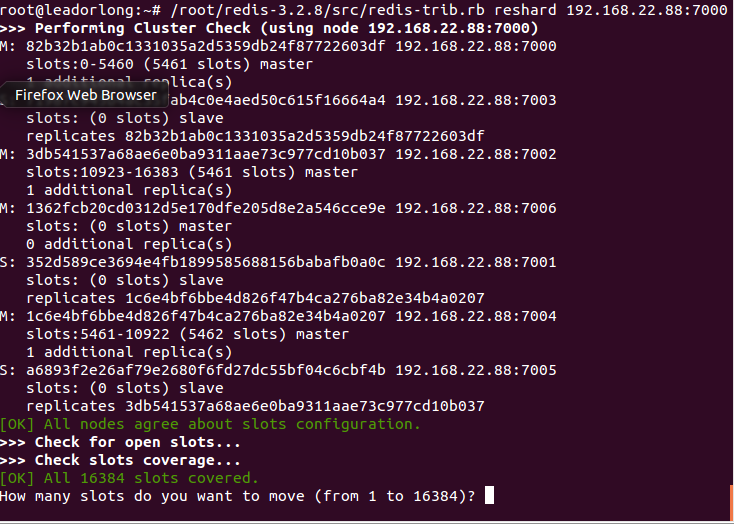
**现在我们用cluster slots查看集群信息**



**可以看到上面并没有7006节点的信息**

**这是因为我们还没有分配slots节点给7006**

**/root/redis-3.2.8/src/redis-trib.rb reshard 192.168.22.88:7000**



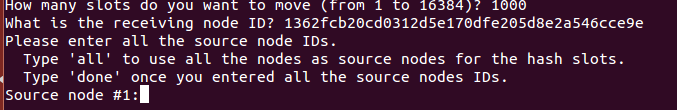
**这里输入需要转移的哈希槽的个数，比如这里我们输入1000**



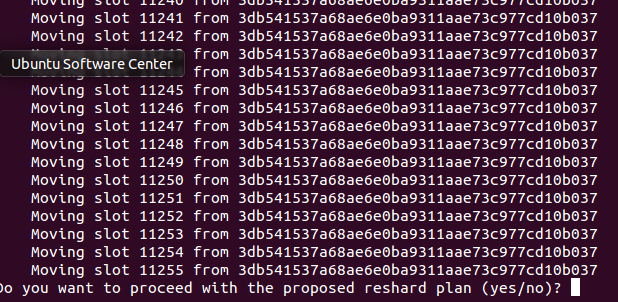
**这里需要指定这1000个哈希槽转移的地址ID，即7006的ID**

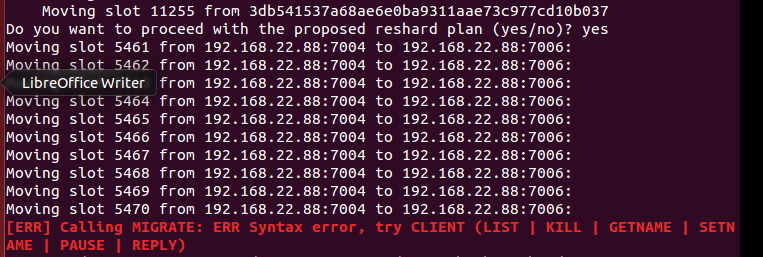


**1362fcb20cd0312d5e170dfe205d8e2a546cce9e**



**这里选择‘all’，表示从所有的主节点中随机转移，凑够1000个哈希槽，然后输入yes，redis集群就开始分配哈希槽了**



**在哈希槽迁移过程中，如果出现这个问题**

**发生这个错误是因为原来的哈希槽中有key-value对象，所以会发生这个错误。官网回复，这是因为ruby的gem不同造成的。以后5.0版本会抛弃redis-trib.rb。直接使用redis-cli客户端实现集群管理。现在这个问题的解决办法是，修改redis.3.2.8/src/redis-trib.rb文件**

**while true**

**keys = source.r.cluster("getkeysinslot",slot,o[:pipeline])**

**break if keys.length == 0**

**begin**

**source.r.client.call(["migrate",target.info[:host],target.info[:port],"",0,@timeout,"replace",:keys,\*keys])**

**STDOUT.flush**

**rescue => e**

**if o[:fix] && e.to\_s =~ /BUSYKEY/**

**xputs "\*\*\* Target key exists. Replacing it for FIX."**

**source.r.client.call(["migrate",target.info[:host],target.info[:port],"",0,@timeout,:replace,:keys,\*keys])**

**else**

**puts ""**

**xputs "[ERR] Calling MIGRATE: #{e}"**

**exit 1**

**end**

**end**

**将上面的**

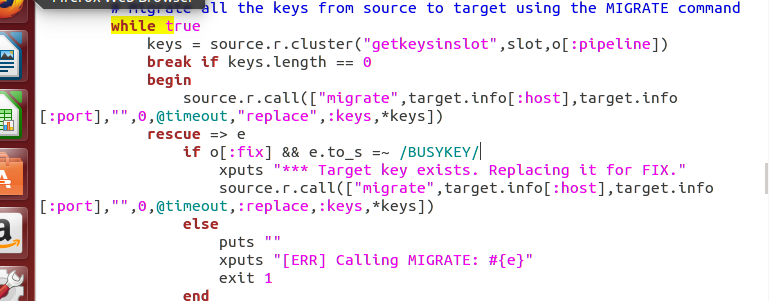
**source.r.client.call(["migrate",target.info[:host],target.info[:port],"",0,@timeout,"replace",:keys,\*keys])**

**source.r.client.call(["migrate",target.info[:host],target.info[:port],"",0,@timeout,:replace,:keys,\*keys])**

**修改为**

**source.r.call(["migrate",target.info[:host],target.info[:port],"",0,@timeout,"replace",:keys,\*keys])**

**source.r.call(["migrate",target.info[:host],target.info[:port],"",0,@timeout,:replace,:keys,\*keys])**



**然后调用/root/redis-3.2.8/src/redis-trib.rb reshard 192.168.22.88:7000指令，这时我们可以看到**

[WARNING] Node 192.168.22.88:7000 has slots in migrating state (7) [WARNING] Node 192.168.22.88:7006 has slots in migrating state (7)

**我们需要修复这些哈希槽，这时我们需要使用如下指令**

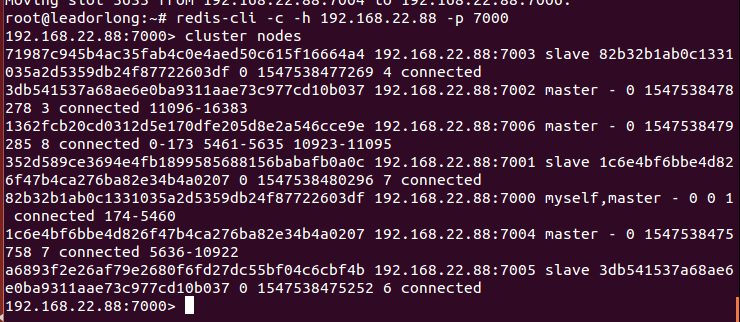
**Redis-cli –c –h 192.168.22.88 –p 7000**

**cluster setslot 7 stable**

**Redis-cli –c –h 192.168.22.88 –p 7006**

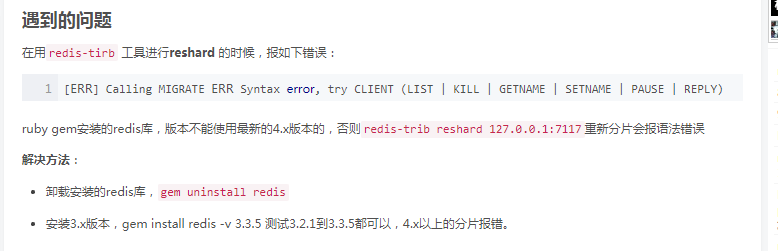
**cluster setslot 7 stable**

**修复这些哈希槽，然后再调用/root/redis-3.2.8/src/redis-trib.rb reshard 192.168.22.88:7000指令，按照上面操作继续执行即可完成哈希槽的移动**



**（这里我们第二次只移动了500个）**

**另外可以用这种方式解决**



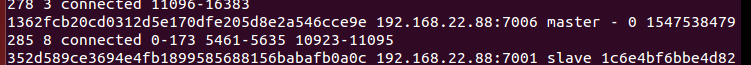
**（2）作为从节点添加进去**

**因为上面的7006以为作为主节点并且分配了哈希槽，所以不再适合。现在安装新建7006服务的方法再建一个7007端口服务。**

**启动7007服务**

**/usr/local/redis/bin/redis-server /root/Desktop/redis-cluster/7007/redis.conf**

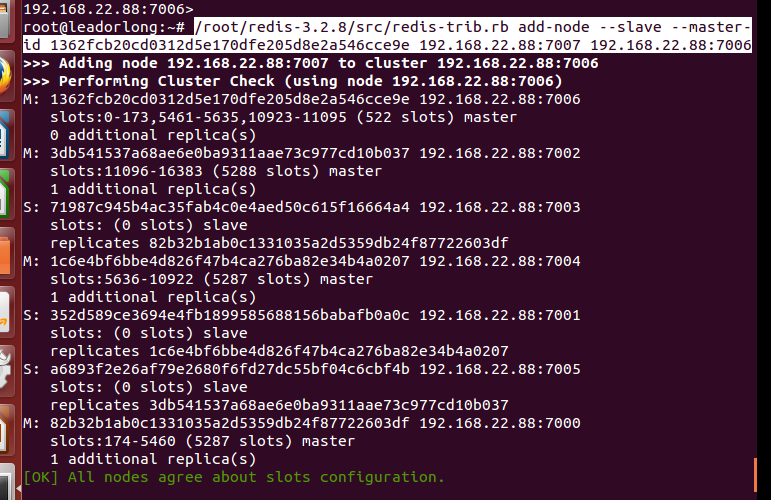
**现在我们将这个7007变成7006主节点下的从节点，首先查看7006主节点的id**



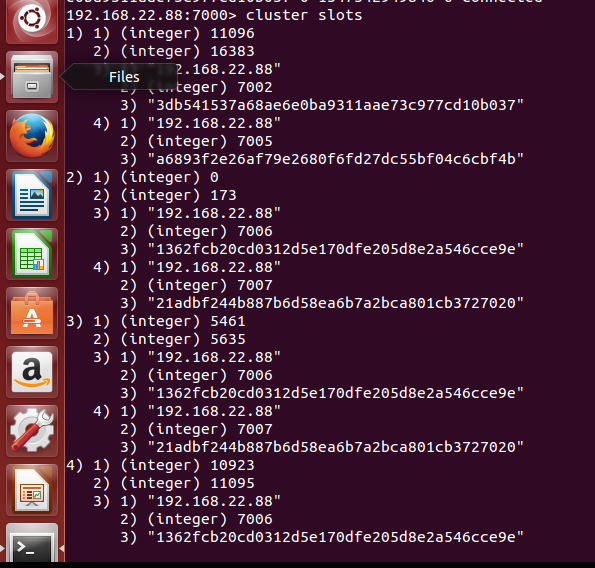
**1362fcb20cd0312d5e170dfe205d8e2a546cce9e**

**然后调用如下指令将7007服务添加到集群当中**

**/root/redis-3.2.8/src/redis-trib.rb add-node --slave --master-id 1362fcb20cd0312d5e170dfe205d8e2a546cce9e 192.168.22.88:7007 192.168.22.88:7006**



**这时我们再来查看一下redis集群的主从关系：**

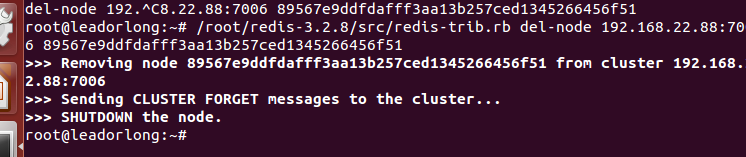


**此时我们已成功将7007从节点添加到7006主节点下**

#### ****VII.删除节点****

**（1）删除从节点(删除7007从节点)**

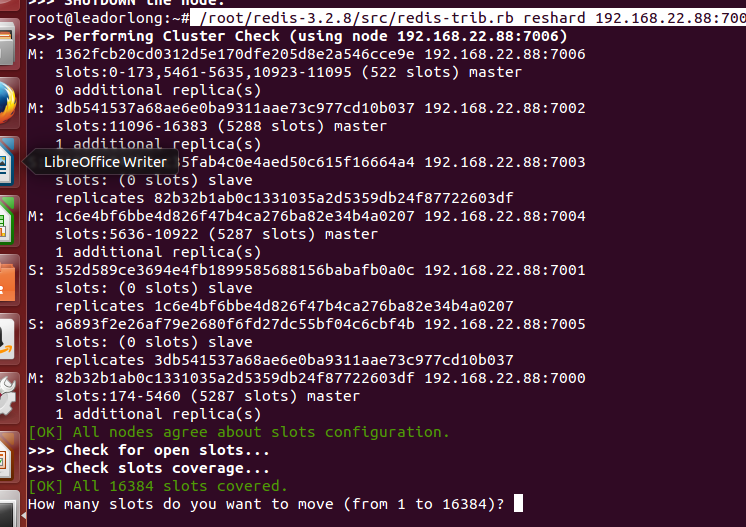
/root/redis-3.2.8/src/redis-trib.rb del-node 192.168.22.88:7006 89567e9ddfdafff3aa13b257ced1345266456f51



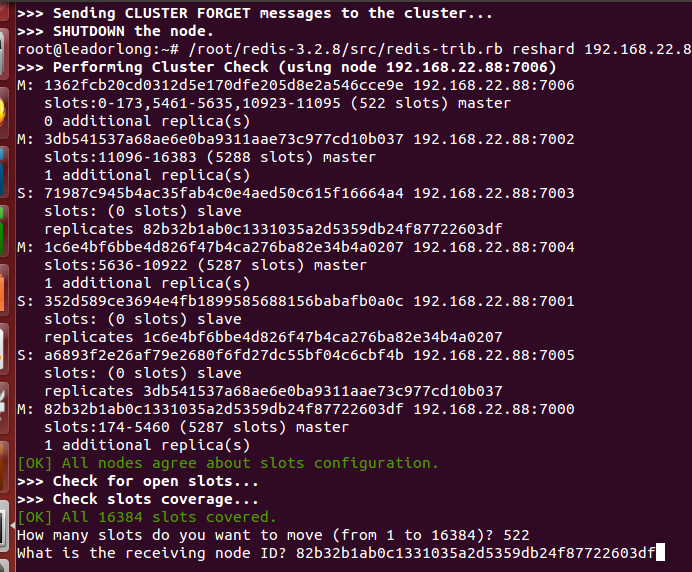
**（2）删除主节点（删除7006主节点）**

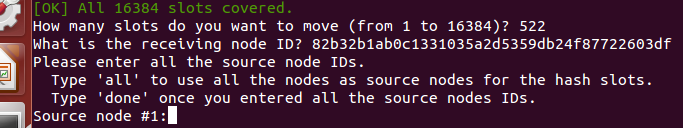
**删除主节点时应当注意主节点上的哈希槽，需要将主节点上的哈希槽移动至其他主节点中**

/root/redis-3.2.8/src/redis-trib.rb reshard 192.168.22.88:7006



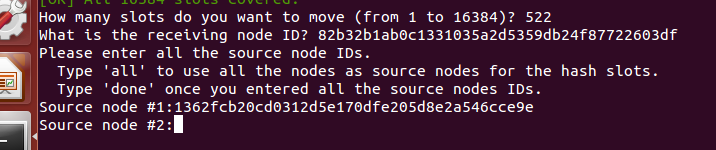
从上面我们可以看到7006中有522个slots，那么我们需要将这522个哈希槽移动到其他主节点中，输入522，然后再输入任意一个主节点ID：



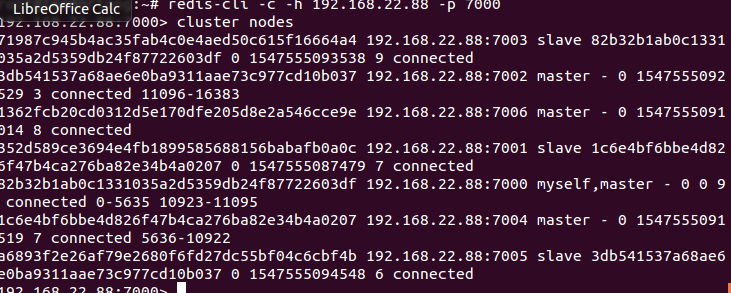


这里需要输入转移哈希槽的来源，即我们需要删除的主节点的ID



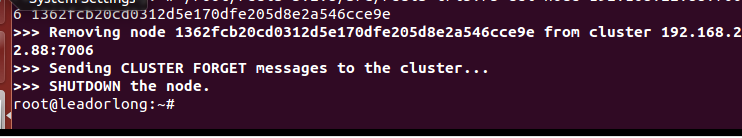


然后输入done，yes即可完成哈希槽的转移，此时我们再俩看看每个主节点分配的hash槽

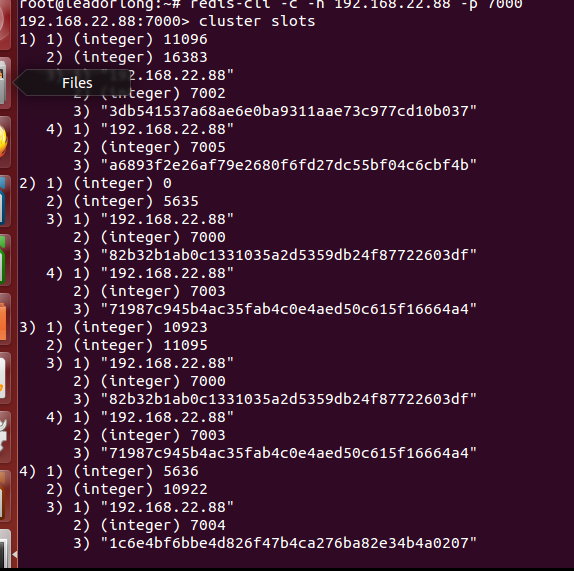


这里我们可以看到7006上的hash槽已被清除完毕，然后我们再来执行清除7006节点的命令

/root/redis-3.2.8/src/redis-trib.rb del-node 192.168.22.88:7006 1362fcb20cd0312d5e170dfe205d8e2a546cce9e



此时，7006节点已被清空，我们来看看现在的集群信息

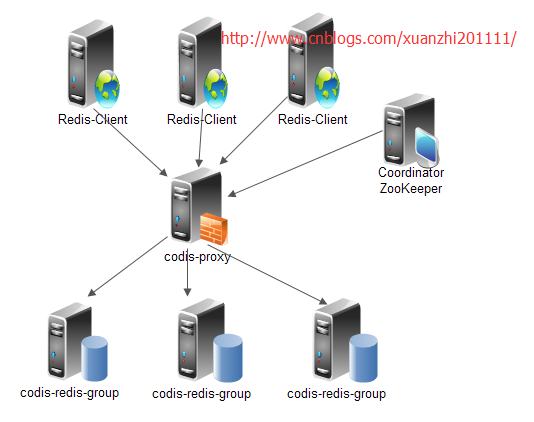


注意：如果想将上面522个哈希槽平均分配到每个主接点中，则多执行几次转移指令即可

### ****codis方案实现****

Codis是一个分布式的Redis解决方案，对于上层的应用来说，连接Codis Proxy和连接原生的Redis Server没有明显的区别（不支持的命令列表），上层应用可以像使用单机的Redis一样使用，Codis底层会处理请求的转发，不停机的数据迁移等工作，所有后边的一切事情，对于前面客户端来说是透明的，可以简单的认为后边连接是一个内存无限大的Redis服务。

Codis架构图：

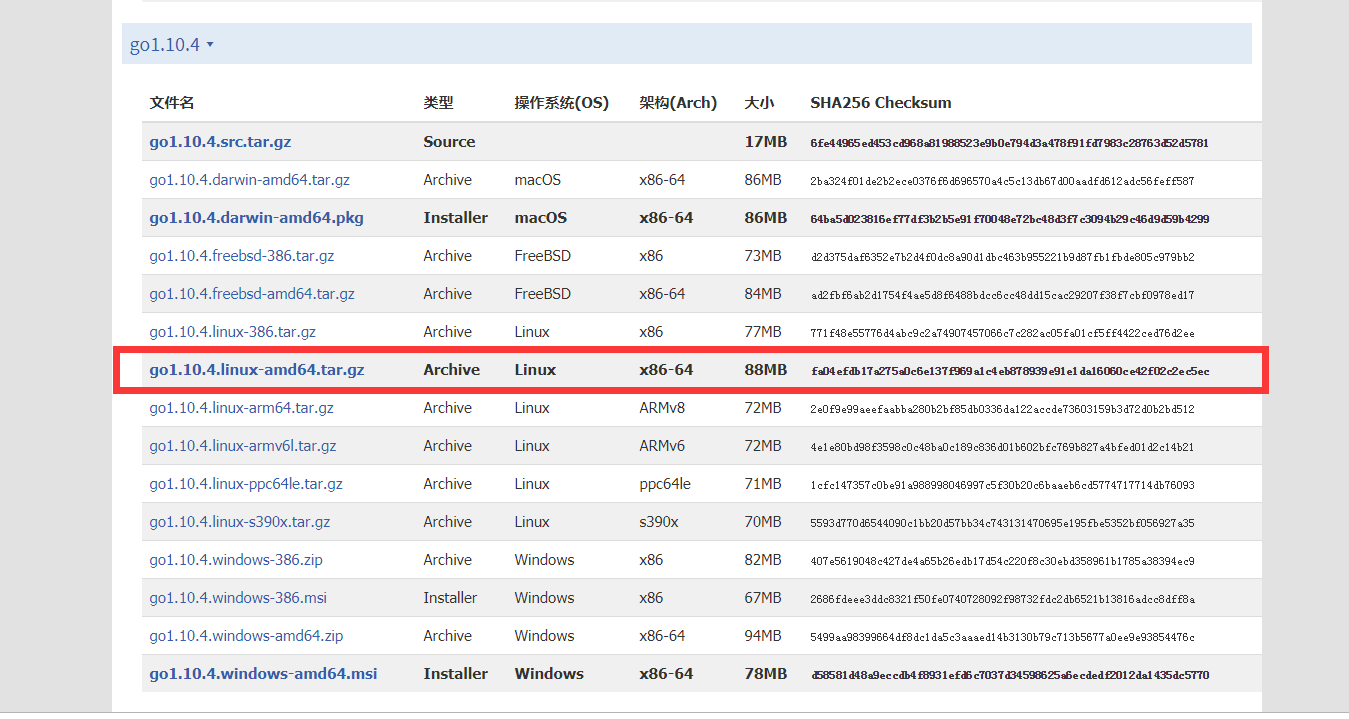


我们在这里只部署单个codis-proxy节点的环境

1. 安装go环境

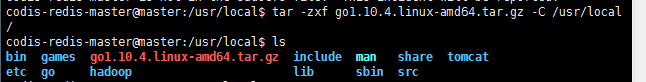
因为海外服务器不可访问，我从GO中文网下载后再传到ubuntu下，进入

<https://studygolang.com/dl>



下的是这个版本，下载完之后通过FlashFXP传输到ubuntu用户的/usr/local/下，然后在该目录下执行解压命令

tar -zxf go1.10.4.linux-amd64.tar.gz -C /usr/local/



再发现该路径下多了一个go文件夹，说明解压缩成功，即可使用

1. 安装JDK（版本要求不严格，后面的zookeeper会用到JDK）

地址: http://www.oracle.com/technetwork/articles/javase/index-jsp-138363.html

选择相应的 .gz包下载

用FlashFXP放到指定目录(以jdk-8u191-linux-x64.tar.gz为例)

创建目录:

sudo mkdir /usr/lib/jvm

解压缩到该目录:

sudo tar -zxvf /data/jdk-8u191-linux-x64.tar.gz -C /usr/lib/jvm

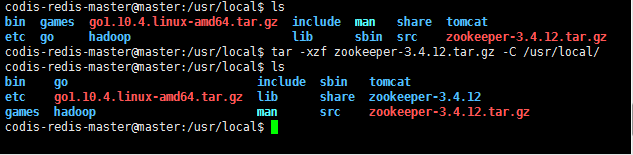


1. 安装zookeeper

官方稳定版下载地址[http://apache.fayea.com/zookeeper/](http://apache.fayea.com/zookeeper/" \t "https://blog.csdn.net/Yan_Chou/article/details/_blank)



选择一个版本的.gz包下载（这里选的zookeeper-3.4.12.tar.gz)，然后通过FlashFXP传到ubuntu的/usr/local下，解压缩到当前目录下输入命令tar -xzf zookeeper-3.4.12.tar.gz -C /usr/local/



设置java,go,zookeeper的配置路径，修改环境变量为 vim /etc/profile 在最后添加以下内容：

#set oracle jdk environment

export JAVA\_HOME=/usr/lib/jvm/jdk1.8.0\_191 ## 这里要注意目录要换成自己解压的jdk 目录

export JRE\_HOME=${JAVA\_HOME}/jre

export CLASSPATH=.:${JAVA\_HOME}/lib:${JRE\_HOME}/lib

export PATH=${JAVA\_HOME}/bin:$PATH

#set go

export GOROOT=/usr/local/go

export GOPATH=/usr/local/codis

#set zookeeper

export ZOOKEEPER\_HOME=/usr/local/zookeeper-3.4.12

export PATH=$PATH:$GOROOT/bin:$JAVA\_HOME/bin:$ZOOKEEPER\_HOME/bin

然后执行 source /etc/profile

测试一下go环境是否安装成功：创建一个hello.go文件，内容为

package main

import "fmt"

func main(){

fmt.Printf("hello,world\n")

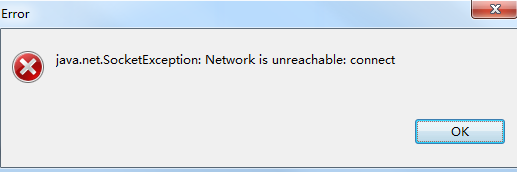
}

go run hello.go成功打印hello,world

1. 安装codis

## 10.微服务整合redis集群

首先看window上的redis client客户端工具能否访问到linux上的redis服务



可以看到出现上面的问题，此时我们应该在liunx下的防火墙开放7000-7005六个端口

执行

sudo ufw allow 7000 #开放7000端口

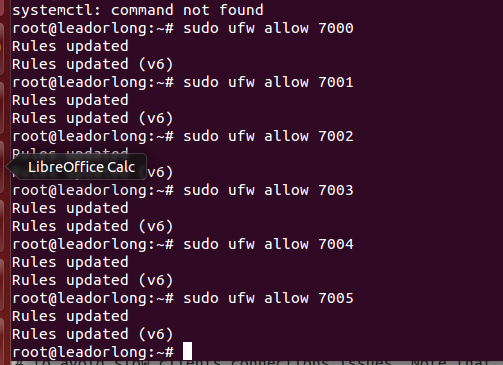
sudo ufw allow 7001 #开放7001端口

sudo ufw allow 7002 #开放7002端口

sudo ufw allow 7003 #开放7003端口

sudo ufw allow 7004 #开放7004端口

sudo ufw allow 7005 #开放7005端口



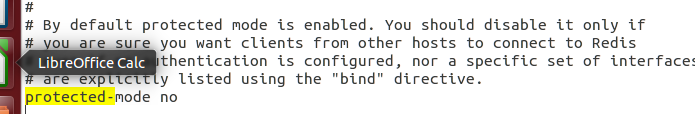
然后重启防火墙

sudo ufw disable

sudo ufw enable

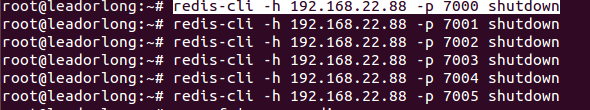
修改7000-7005文件夹中的redis-conf文件，如下

将protected-mode yes改为protected-mode no

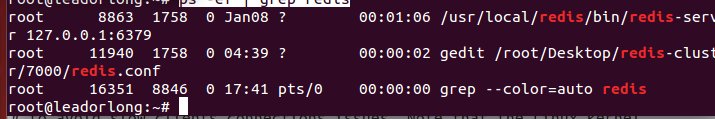


然后重启这六个服务

redis-cli -h 192.168.22.88 -p 7000 shutdown关闭redis服务



输入ps -ef | grep redis可以看到



然后再开启六个redis服务

/usr/local/redis/bin/redis-server /root/Desktop/redis-cluster/7000/redis.conf

/usr/local/redis/bin/redis-server /root/Desktop/redis-cluster/7001/redis.conf

/usr/local/redis/bin/redis-server /root/Desktop/redis-cluster/7002/redis.conf

/usr/local/redis/bin/redis-server /root/Desktop/redis-cluster/7003/redis.conf

/usr/local/redis/bin/redis-server /root/Desktop/redis-cluster/7004/redis.conf

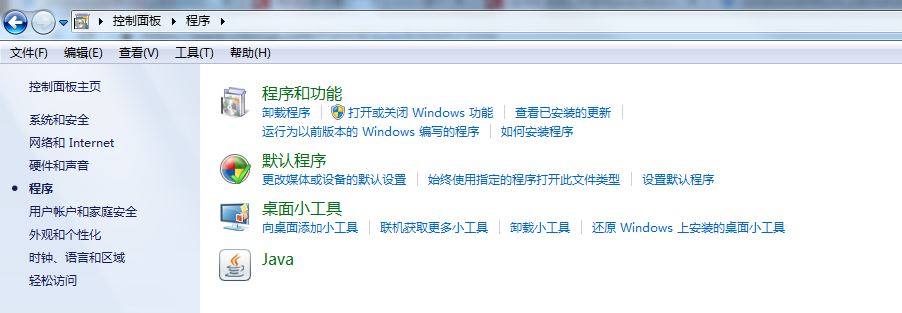
/usr/local/redis/bin/redis-server /root/Desktop/redis-cluster/7005/redis.conf

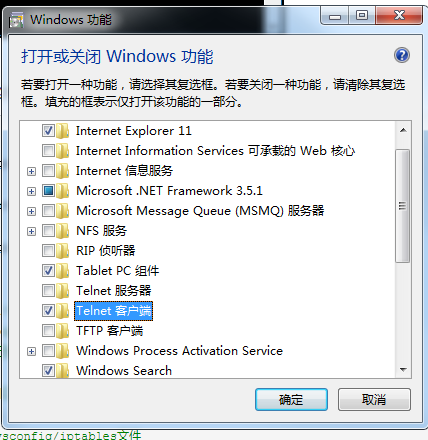
现在在windows命令窗口访问192.168.22.88的7000服务

Telnet 192.168.22.88 7000



**如果telnet命令无效，提示“telnet不是内部或外部命令”，你需要去【控制面板】，打开【程序和功能】，然后打开【启动和关闭Windows功能】窗口，在窗口滑动列表，找到【telnet客户端】选项，然后选择该项，Window系统会安装对telnet命的支持。**





随后再执行Telnet 192.168.22.88 7000

可以看到一片漆黑

此时我们可以通过redis client客户端工具访问linux上的redis服务了

### 1.导入相关依赖

<dependency>  
 <groupId>org.springframework.boot</groupId>  
 <artifactId>spring-boot-starter-data-redis</artifactId>  
</dependency>

### 2.添加redis配置

在application.yml中

**spring：**

**redis:  
 cluster:  
 nodes:** 192.168.22.88:7000,192.168.22.88:7001,192.168.22.88:7002,192.168.22.88:7003,192.168.22.88:7004,192.168.22.88:7005  
 *#最大重定向，由于集群失败机制是多数失败则失败，那么集群的最低标准是3个分片，1主1从，总共六个节点，* **max-redirects:** 5  
 *#所以如果当前节点失败，最大的重定向为6-1=5，spring默认的也为5* **timeout:** 60000  
 *#redis集群响应超时时间*

### 3.启动项目自动加载redis集群配置信息

@Configuration  
public class JedisClusterConfig {  
  
 @Value("${spring.redis.clusterNodes}")  
 private String clusterNodes;  
  
 @Value("${spring.redis.password}")  
 private String password;  
  
 @Value("${spring.redis.commandTimeout}")  
 private int commandTimeout;  
  
 */\*\*  
 \* 注意：  
 \* 这里返回的JedisCluster是单例的，并且可以直接注入到其他类中去使用  
 \*  
 \** ***@return*** *\*/* @Bean  
 public JedisCluster getJedisCluster() {  
 //获取服务器数组(这里要相信自己的输入，所以没有考虑空指针问题)  
 String[] serverArray = clusterNodes.split(",");  
 Set<HostAndPort> nodes = new HashSet<>();  
  
 for (String ipPort : serverArray) {  
 String[] ipPortPair = ipPort.split(":");  
 nodes.add(new HostAndPort(ipPortPair[0].trim(), Integer.*valueOf*(ipPortPair[1].trim())));  
 }  
  
 JedisCluster jedisCluster = null;  
 if (StringUtils.*isBlank*(password)) {  
 //不需要密码连接的创建对象方式  
 jedisCluster = new JedisCluster(nodes, commandTimeout, 1000, 1, new GenericObjectPoolConfig());  
 } else {  
 //需要密码连接的创建对象方式  
 jedisCluster = new JedisCluster(nodes, commandTimeout, 1000, 1, password, new GenericObjectPoolConfig());  
 }  
  
 return jedisCluster;  
 }  
}

### 4.使用jedisCluster创建redisutil操作工具类以供使用

