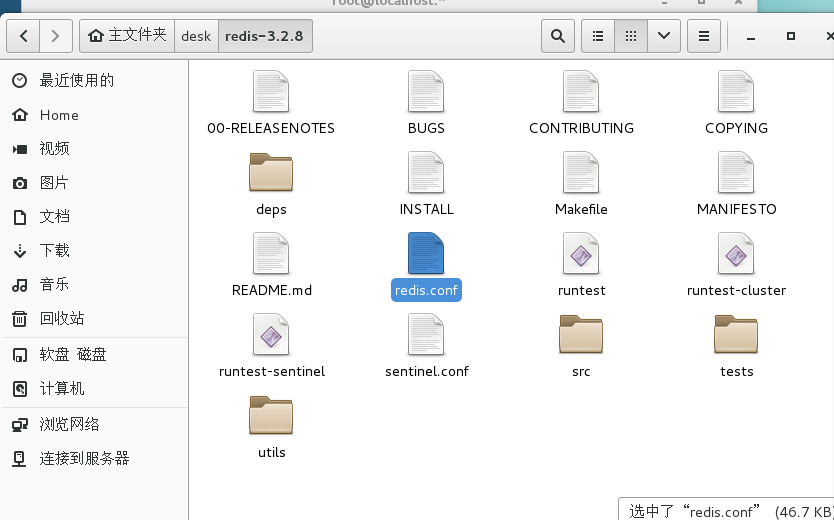
1.首先 ifconfig 得到自己电脑的ip

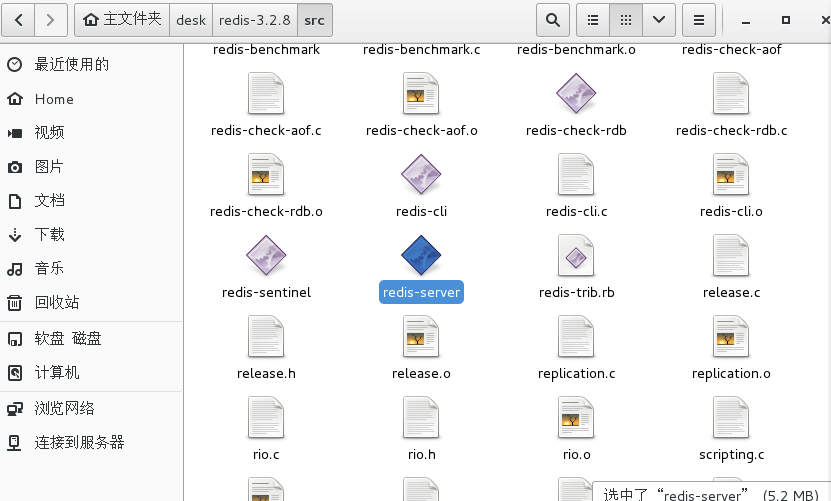


可以看到当前ip为192.168.1.86

1. 找到redis目录下的redis.conf



和redis/src目录下的 redis-server



1. 修改redis.conf

最主要是需要这几个

port 7000 --端口指定

daemonize yes --守护进程启用

cluster-enabled yes --打开redis集群

cluster-config-file nodes.conf --cluster配置文件(启动自动生成)

cluster-node-timeout 5000 --节点互连超时的阀值

appendonly yes --打开aof持久化

bind 192.168.1.11 --填写本地ip

# 守护进程模式

daemonize yes

# pid file

pidfile /var/run/redis.pid

# 监听端口

port 7004

# 绑定ip

bind 192.168.1.86

# TCP接收队列长度，受/proc/sys/net/core/somaxconn和tcp\_max\_syn\_backlog这两个内核参数的影响

tcp-backlog 511

# 一个客户端空闲多少秒后关闭连接(0代表禁用，永不关闭)

timeout 0

#打开redis集群

cluster-enabled yes

# 如果非零，则设置SO\_KEEPALIVE选项来向空闲连接的客户端发送ACK

tcp-keepalive 300

# 指定服务器调试等级

# 可能值：

# debug （大量信息，对开发/测试有用）

# verbose （很多精简的有用信息，但是不像debug等级那么多）

# notice （适量的信息，基本上是你生产环境中需要的）

# warning （只有很重要/严重的信息会记录下来）

loglevel notice

# 指明日志文件名

logfile "./redis7004.log"

# 设置数据库个数

databases 16

# 会在指定秒数和数据变化次数之后把数据库写到磁盘上

# 900秒（15分钟）之后，且至少1次变更

# 300秒（5分钟）之后，且至少10次变更

# 60秒之后，且至少10000次变更

save 900 1

save 300 10

save 60 10000

# 默认如果开启RDB快照(至少一条save指令)并且最新的后台保存失败，Redis将会停止接受写操作

# 这将使用户知道数据没有正确的持久化到硬盘，否则可能没人注意到并且造成一些灾难

stop-writes-on-bgsave-error yes

# 当导出到 .rdb 数据库时是否用LZF压缩字符串对象

rdbcompression yes

# 版本5的RDB有一个CRC64算法的校验和放在了文件的最后。这将使文件格式更加可靠。

rdbchecksum yes

# 持久化数据库的文件名

dbfilename dump.rdb

# 工作目录

dir ./

# 当master服务设置了密码保护时，slav服务连接master的密码

#masterauth 0234kz9\*l

#密码最后创建

# 当一个slave失去和master的连接，或者同步正在进行中，slave的行为可以有两种：

#

# 1) 如果 slave-serve-stale-data 设置为 "yes" (默认值)，slave会继续响应客户端请求，

# 可能是正常数据，或者是过时了的数据，也可能是还没获得值的空数据。

# 2) 如果 slave-serve-stale-data 设置为 "no"，slave会回复"正在从master同步

# （SYNC with master in progress）"来处理各种请求，除了 INFO 和 SLAVEOF 命令。

slave-serve-stale-data yes

# 你可以配置salve实例是否接受写操作。可写的slave实例可能对存储临时数据比较有用(因为写入salve

# 的数据在同master同步之后将很容易被删除

slave-read-only yes

# 是否在slave套接字发送SYNC之后禁用 TCP\_NODELAY？

# 如果你选择“yes”Redis将使用更少的TCP包和带宽来向slaves发送数据。但是这将使数据传输到slave

# 上有延迟，Linux内核的默认配置会达到40毫秒

# 如果你选择了 "no" 数据传输到salve的延迟将会减少但要使用更多的带宽

repl-disable-tcp-nodelay no

# slave的优先级是一个整数展示在Redis的Info输出中。如果master不再正常工作了，哨兵将用它来

# 选择一个slave提升=升为master。

# 优先级数字小的salve会优先考虑提升为master，所以例如有三个slave优先级分别为10，100，25，

# 哨兵将挑选优先级最小数字为10的slave。

# 0作为一个特殊的优先级，标识这个slave不能作为master，所以一个优先级为0的slave永远不会被

# 哨兵挑选提升为master

slave-priority 100

# 密码验证

# 警告：因为Redis太快了，所以外面的人可以尝试每秒150k的密码来试图破解密码。这意味着你需要

# 一个高强度的密码，否则破解太容易了

#requirepass 0234kz9\*l

#密码最后创建

# redis实例最大占用内存，不要用比设置的上限更多的内存。一旦内存使用达到上限，Redis会根据选定的回收策略（参见：

# maxmemmory-policy）删除key

maxmemory 3gb

# 最大内存策略：如果达到内存限制了，Redis如何选择删除key。你可以在下面五个行为里选：

# volatile-lru -> 根据LRU算法删除带有过期时间的key。

# allkeys-lru -> 根据LRU算法删除任何key。

# volatile-random -> 根据过期设置来随机删除key, 具备过期时间的key。

# allkeys->random -> 无差别随机删, 任何一个key。

# volatile-ttl -> 根据最近过期时间来删除（辅以TTL）, 这是对于有过期时间的key

# noeviction -> 谁也不删，直接在写操作时返回错误。

maxmemory-policy volatile-lru

# 默认情况下，Redis是异步的把数据导出到磁盘上。这种模式在很多应用里已经足够好，但Redis进程

# 出问题或断电时可能造成一段时间的写操作丢失(这取决于配置的save指令)。

#

# AOF是一种提供了更可靠的替代持久化模式，例如使用默认的数据写入文件策略（参见后面的配置）

# 在遇到像服务器断电或单写情况下Redis自身进程出问题但操作系统仍正常运行等突发事件时，Redis

# 能只丢失1秒的写操作。

#

# AOF和RDB持久化能同时启动并且不会有问题。

# 如果AOF开启，那么在启动时Redis将加载AOF文件，它更能保证数据的可靠性。

appendonly no

# aof文件名

appendfilename "appendonly.aof"

# fsync() 系统调用告诉操作系统把数据写到磁盘上，而不是等更多的数据进入输出缓冲区。

# 有些操作系统会真的把数据马上刷到磁盘上；有些则会尽快去尝试这么做。

#

# Redis支持三种不同的模式：

#

# no：不要立刻刷，只有在操作系统需要刷的时候再刷。比较快。

# always：每次写操作都立刻写入到aof文件。慢，但是最安全。

# everysec：每秒写一次。折中方案。

appendfsync everysec

# 如果AOF的同步策略设置成 "always" 或者 "everysec"，并且后台的存储进程（后台存储或写入AOF

# 日志）会产生很多磁盘I/O开销。某些Linux的配置下会使Redis因为 fsync()系统调用而阻塞很久。

# 注意，目前对这个情况还没有完美修正，甚至不同线程的 fsync() 会阻塞我们同步的write(2)调用。

#

# 为了缓解这个问题，可以用下面这个选项。它可以在 BGSAVE 或 BGREWRITEAOF 处理时阻止主进程进行fsync()。

#

# 这就意味着如果有子进程在进行保存操作，那么Redis就处于"不可同步"的状态。

# 这实际上是说，在最差的情况下可能会丢掉30秒钟的日志数据。（默认Linux设定）

#

# 如果你有延时问题把这个设置成"yes"，否则就保持"no"，这是保存持久数据的最安全的方式。

no-appendfsync-on-rewrite yes

# 自动重写AOF文件

auto-aof-rewrite-percentage 100

auto-aof-rewrite-min-size 64mb

# AOF文件可能在尾部是不完整的（这跟system关闭有问题，尤其是mount ext4文件系统时

# 没有加上data=ordered选项。只会发生在os死时，redis自己死不会不完整）。

# 那redis重启时load进内存的时候就有问题了。

# 发生的时候，可以选择redis启动报错，并且通知用户和写日志，或者load尽量多正常的数据。

# 如果aof-load-truncated是yes，会自动发布一个log给客户端然后load（默认）。

# 如果是no，用户必须手动redis-check-aof修复AOF文件才可以。

# 注意，如果在读取的过程中，发现这个aof是损坏的，服务器也是会退出的，

# 这个选项仅仅用于当服务器尝试读取更多的数据但又找不到相应的数据时。

aof-load-truncated yes

# Lua 脚本的最大执行时间，毫秒为单位

lua-time-limit 5000

# Redis慢查询日志可以记录超过指定时间的查询

slowlog-log-slower-than 10000

# 这个长度没有限制。只是要主要会消耗内存。你可以通过 SLOWLOG RESET 来回收内存。

slowlog-max-len 128

# redis延时监控系统在运行时会采样一些操作，以便收集可能导致延时的数据根源。

# 通过 LATENCY命令 可以打印一些图样和获取一些报告，方便监控

# 这个系统仅仅记录那个执行时间大于或等于预定时间（毫秒）的操作,

# 这个预定时间是通过latency-monitor-threshold配置来指定的，

# 当设置为0时，这个监控系统处于停止状态

latency-monitor-threshold 0

# Redis能通知 Pub/Sub 客户端关于键空间发生的事件，默认关闭

notify-keyspace-events ""

# 当hash只有少量的entry时，并且最大的entry所占空间没有超过指定的限制时，会用一种节省内存的

# 数据结构来编码。可以通过下面的指令来设定限制

hash-max-ziplist-entries 512

hash-max-ziplist-value 64

# set有一种特殊编码的情况：当set数据全是十进制64位有符号整型数字构成的字符串时。

# 下面这个配置项就是用来设置set使用这种编码来节省内存的最大长度。

set-max-intset-entries 512

# 与hash和list相似，有序集合也可以用一种特别的编码方式来节省大量空间。

# 这种编码只适合长度和元素都小于下面限制的有序集合

zset-max-ziplist-entries 128

zset-max-ziplist-value 64

# HyperLogLog稀疏结构表示字节的限制。该限制包括

# 16个字节的头。当HyperLogLog使用稀疏结构表示

# 这些限制，它会被转换成密度表示。

# 值大于16000是完全没用的，因为在该点

# 密集的表示是更多的内存效率。

# 建议值是3000左右，以便具有的内存好处, 减少内存的消耗

hll-sparse-max-bytes 3000

# 启用哈希刷新，每100个CPU毫秒会拿出1个毫秒来刷新Redis的主哈希表（顶级键值映射表）

activerehashing yes

# 客户端的输出缓冲区的限制，可用于强制断开那些因为某种原因从服务器读取数据的速度不够快的客户端

client-output-buffer-limit normal 0 0 0

client-output-buffer-limit slave 256mb 64mb 60

client-output-buffer-limit pubsub 32mb 8mb 60

# 默认情况下，“hz”的被设定为10。提高该值将在Redis空闲时使用更多的CPU时，但同时当有多个key

# 同时到期会使Redis的反应更灵敏，以及超时可以更精确地处理

hz 10

# 当一个子进程重写AOF文件时，如果启用下面的选项，则文件每生成32M数据会被同步

aof-rewrite-incremental-fsync yes

1. 复制redis-server 去redis.conf的目录 也可以使用命令

cp redis-server /usr/local/redis/cluster/7004/

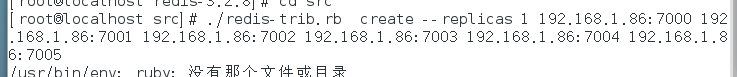


1. 在7004目录下 使用 redis-server redis.conf 开启服务



1. 同理 另外创建7000 7001 7002 7003 7005
2. 进入redis /src 目录下使用下面命令

./redis-trib.rb create --replicas 1 192.168.1.86:7000 192.168.1.86:7001 192.168.1.86:7002 192.168.1.86:7003 192.168.1.86:7004 192.168.1.86:7005

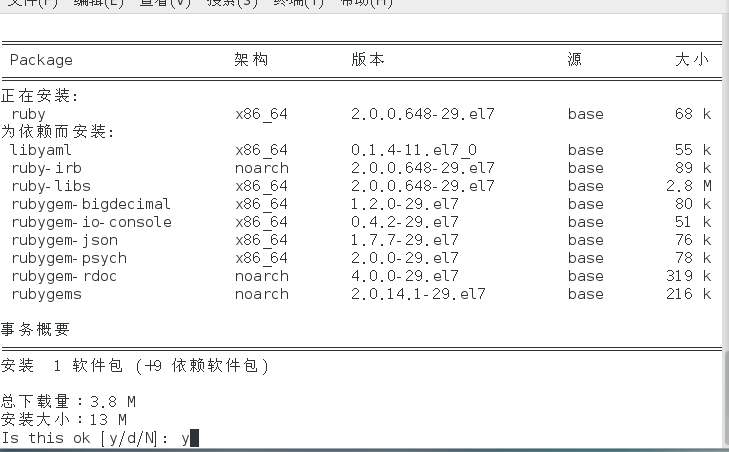


这里的错误 提示缺少rubygems组件，使用yum安装

使用命令 yum install ruby



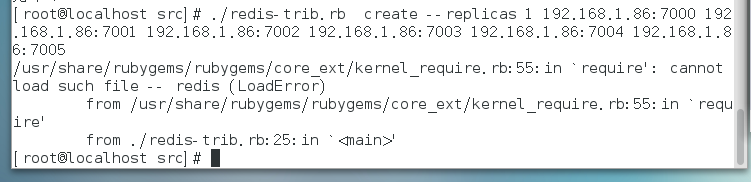
是否 打y 然后回车 如下图



安装完毕后 重新执行

./redis-trib.rb create --replicas 1 192.168.1.86:7000 192.168.1.86:7001 192.168.1.86:7002 192.168.1.86:7003 192.168.1.86:7004 192.168.1.86:7005

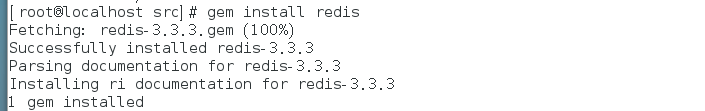
出现下面问题 提示不能加载redis，是因为缺少redis和ruby的接口，使用gem 安装



使用命令 gem install redis

如果更新失败 可以尝试

gem sources –remove [https://rubygems.org/](https://rubygems.org/" \t "http://blog.csdn.net/u013680097/article/details/_blank)   
gem sources -a [https://ruby.taobao.org/](https://ruby.taobao.org/" \t "http://blog.csdn.net/u013680097/article/details/_blank)



再次输入

./redis-trib.rb create --replicas 1 192.168.1.86:7000 192.168.1.86:7001 192.168.1.86:7002 192.168.1.86:7003 192.168.1.86:7004 192.168.1.86:7005



输入yes 如上图 然后回车 如下图



9. 查看状态

输入 redis-cli -h 192.168.1.86 -p 7002 cluster nodes



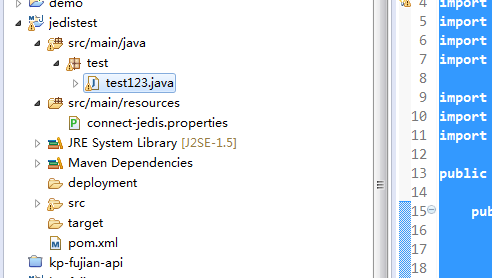
10. 设置密码(一个一个设置)

./redis-cli -h 192.168.1.86 -p 7000    
config set masterauth password123

config set requirepass password123

示例代码

结构示意图：



Maven

<!-- https://mvnrepository.com/artifact/redis.clients/jedis -->

<dependency>

<groupId>redis.clients</groupId>

<artifactId>jedis</artifactId>

<version>2.9.0</version>

</dependency>

test123.java

**package** test;

**import** java.io.InputStream;

**import** java.net.URL;

**import** java.util.HashSet;

**import** java.util.Properties;

**import** java.util.Set;

**import** redis.clients.jedis.HostAndPort;

**import** redis.clients.jedis.JedisCluster;

**import** redis.clients.jedis.JedisPoolConfig;

**public** **class** test123 {

**public** **void** s() {

System.***out***.println("start");

InputStream in = test123.**class**.getResourceAsStream("/connect-jedis.properties");

**try** {

Properties properties = **new** Properties();

properties.load(in);

// 数据库链接池配置

JedisPoolConfig config = **new** JedisPoolConfig();

config.setMaxTotal(Integer.*parseInt*(properties.getProperty("MaxTotal")));

config.setMaxIdle(Integer.*parseInt*(properties.getProperty("MaxIdle")));

config.setMinIdle(Integer.*parseInt*(properties.getProperty("MinIdle")));

config.setMaxWaitMillis(Integer.*parseInt*(properties.getProperty("MaxWaitMillis")));

config.setTestOnBorrow(Boolean.*getBoolean*(properties.getProperty("TestOnBorrow")));

String address = properties.getProperty("address");

String[] hostandportlist = address.split(",");

// Redis集群的节点集合

Set<HostAndPort> jedisClusterNodes = **new** HashSet<HostAndPort>();

**for** (String hostandport : hostandportlist ) {

String[] host\_port = hostandport.split(":");

jedisClusterNodes.add(**new** HostAndPort(host\_port[0] , Integer.*parseInt*(host\_port[1])));

}

// jedisClusterNodes.add(new HostAndPort("192.168.1.85", 7000));

// jedisClusterNodes.add(new HostAndPort("192.168.1.85", 7001));

// jedisClusterNodes.add(new HostAndPort("192.168.1.85", 7002));

// jedisClusterNodes.add(new HostAndPort("192.168.1.85", 7003));

// jedisClusterNodes.add(new HostAndPort("192.168.1.85", 7004));

// jedisClusterNodes.add(new HostAndPort("192.168.1.85", 7005));

// 根据节点集创集群链接对象

// JedisCluster jedisCluster = newJedisCluster(jedisClusterNodes);

// 节点，超时时间，最多重定向次数，链接池

**int** connectionTimeout = Integer.*parseInt*(properties.getProperty("connectionTimeout"));

**int** soTimeout = Integer.*parseInt*(properties.getProperty("soTimeout"));

**int** maxAttempts = Integer.*parseInt*(properties.getProperty("maxAttempts"));

String password = properties.getProperty("password");

JedisCluster jedisCluster = **new** JedisCluster(jedisClusterNodes , connectionTimeout , soTimeout , maxAttempts , password , config);

// JedisCluster jedisCluster = new JedisCluster(jedisClusterNodes,

// 2000, 100,config);

**int** num = 1000;

String key = "wusc";

String value = "";

**for** (**int** i = 0 ; i <= num ; i++ ) {

// 存数据

jedisCluster.set(key + i , "WuShuicheng" + i);

// 取数据

value = jedisCluster.get(key + i);

System.***out***.println(key + i + "=" + value);

}

System.***out***.println("finish");

} **catch** (Exception e) {

e.printStackTrace();

}

}

**public** **static** **void** main(String[] argsd) {

**new** test123().s();

}

}

connect-jedis.properties

address=192.168.1.85:7000,192.168.1.85:7001,192.168.1.85:7002,192.168.1.85:7003,192.168.1.85:7004,192.168.1.85:7005

MaxTotal=100

#pool最多有多少个状态为idle(空闲的)的jedis实例，默认值8

MaxIdle=50

#pool最少有多少个状态为idle(空闲的)的jedis实例

MinIdle=20

#等待可用连接的最大时间，单位毫秒，默认值为-1，表示永不超时。如果超过等待时间，则直接抛出JedisConnectionException

MaxWaitMillis =600

TestOnBorrow = true

password=password123

connectionTimeout=2000

soTimeout=2000

maxAttempts=100