**Redis高级**

# 可能的问题

一般来说，要将Redis运用于工程项目中，只使用一台Redis是万万不能的，原因如下：

1、从结构上，单个Redis服务器会发生单点故障，并且一台服务器需要处理所有的请求负载，压力较大； (容错性)

2、从容量上，单个Redis服务器内存容量有限，就算一台Redis服务器内容容量为256G，也不能将所有内容用作Redis存储内存，一般来说，单台Redis最大使用内存不应该超过**20G**。

# 基本概述

## 高可用

“高可用性”（High Availability）通常来描述一个系统经过专门的设计，从而减少停工时间，而保持其服务的高度可用性。(一直都能用)

## 高并发

高并发（High Concurrency）是互联网分布式系统架构设计中必须考虑的因素之一，它通常是指，通过设计保证系统能够同时并行处理很多请求。

高并发相关常用的一些指标有响应时间（Response Time），吞吐量（Throughput），每秒查询率QPS（Query Per Second），并发用户数等。

**响应时间**：系统对请求做出响应的时间。例如系统处理一个HTTP请求需要200ms，这个200ms就是系统的响应时间。

**吞吐量**：单位时间内处理的请求数量。

**QPS**：每秒响应请求数。在互联网领域，这个指标和吞吐量区分的没有这么明显。

**并发用户数**：同时承载正常使用系统功能的用户数量。例如一个即时通讯系统，同时在线量一定程度上代表了系统的并发用户数。

### 提升系统的并发能力

提高系统并发能力的方式，方法论上主要有两种：垂直扩展（Scale Up）与水平扩展（Scale Out）。

#### 垂直扩展

**垂直扩展：提升单机处理能力。垂直扩展的方式又有两种：**

（1）增强单机硬件性能，例如：增加CPU核数如32核，升级更好的网卡如万兆，升级更好的硬盘如SSD，扩充硬盘容量如2T，扩充系统内存如128G；

（2）提升单机架构性能，例如：使用Cache来减少IO次数，使用异步来增加单服务吞吐量，使用无锁数据结构来减少响应时间；

在互联网业务发展非常迅猛的早期，如果预算不是问题，强烈建议使用“增强单机硬件性能”的方式提升系统并发能力，因为这个阶段，公司的战略往往是发展业务抢时间，而“增强单机硬件性能”往往是最快的方法。

**总结：**不管是提升单机硬件性能，还是提升单机架构性能，都有一个致命的不足：单机性能总是有极限的。所以互联网分布式架构设计高并发终极解决方案还是水平扩展。

#### 水平扩展

水平扩展：只要增加服务器数量，就能线性扩充系统性能。水平扩展对系统架构设计是有要求的，难点在于：如何在架构各层进行可水平扩展的设计，

## 高性能

高性能（High Performance）就是指程序处理速度快，所占内存少，cpu低

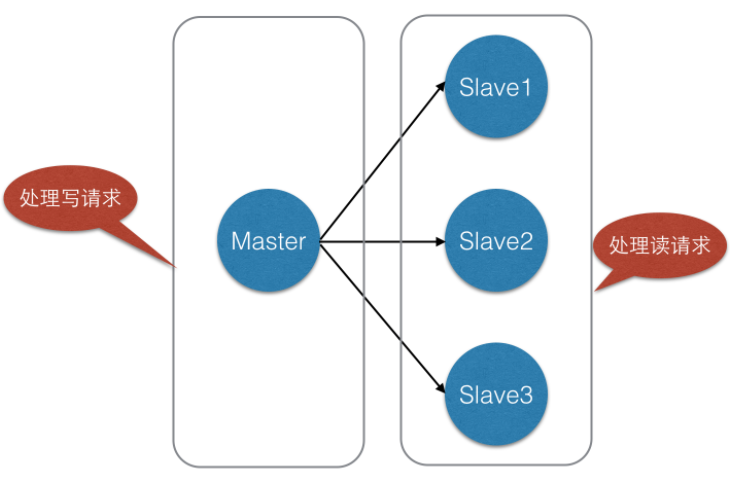
# Redis主从复制

## 简介

应用场景：电子商务网站上的商品，一般都是一次上传，无数次浏览的，说专业点也就是”多读少写”。

**主从复制：**

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，其它复制称为Slaves



如图中所示，我们将一台Redis服务器作主库(Matser)，其他三台作为从库(Slave)，主库只负责写数据，每次有数据更新都将更新的数据同步到它所有的从库，而从库只负责读数据。这样一来，就有了两个好处：

1. 读写分离，不仅可以提高服务器的负载能力，并且可以根据读请求的规模自由增加或者减少从库的数量。
2. 数据被复制成了了好几份，就算有一台机器出现故障，也可以使用其他机器的数据快速恢复。

需要注意的是：在Redis主从模式中，一台主库可以拥有多个从库，但是一个从库只能隶属于一个主库。

## Redis主从复制 配置

在Redis中，要实现主从复制架构非常简单，只需要在从数据库的配置文件中加上如下命令即可：

1、主数据库不需要任务配置，创建一个从数据库：

-- port 6380 //从服务的端口号

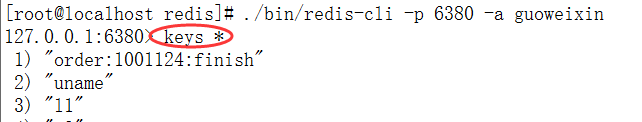
--slaveof 127.0.0.1 6379 //指定主服务器

./bin/redis-server ./redis.conf **--port 6380 --slaveof 127.0.0.1 6379**

加上slaveof参数启动另一个Redis实例作为从库，并且监听6380端口

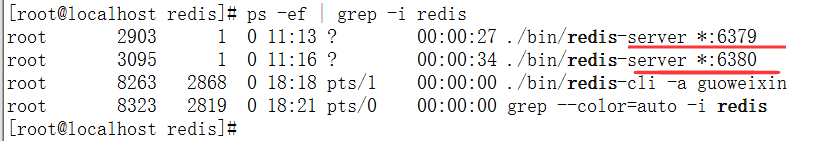
2、登录到从服务客户端：

./bin/redis-cli -p 6380 -a guoweixin



变回主： slaveof on one //不是任何从

变回从： slaveof ip地址 端口号



# Redis Cluster集群

## 简介

* 为什么使用redis-cluster？
* 为了在大流量访问下提供稳定的业务，集群化是存储的必然形态
* 未来的发展趋势肯定是云计算和大数据的紧密结合
* 只有分布式架构能满足要求

|  |  |
| --- | --- |
| Redis集群搭建方案：  （1）Twitter开发的twemproxy  （2）豌豆荚开发的codis  （3）**redis官方的redis-cluster** |  |

Redis集群搭建的方式有多种，但从redis 3.0之后版本支持redis-cluster集群，**至少需要3(Master)+3(Slave)才能建立集群**。Redis-Cluster采用无中心结构，每个节点保存数据和整个集群状态,每个节点都和其他所有 节点连接。其redis-cluster架构图如右上侧：

## Redis Cluster集群特点

1、所有的redis节点彼此互联(PING-PONG机制),内部使用二进制协议优化传输速度和带宽。  
2、节点的fail是通过集群中超过半数的节点检测失效时才生效。  
3、客户端与redis节点直连,不需要中间proxy层.客户端不需要连接集群所有节点,连接集群中任何一个可用节点即可。  
4、redis-cluster把所有的物理节点映射到[0-16383]slot上（不一定是平均分配）,cluster 负责维护

5、Redis集群预分好16384个哈希槽，当需要在 Redis 集群中放置一个 key-value 时， redis 先对key 使用 crc16 算法算出一个结果，然后把结果对 16384 求余数，这样每个 key 都会对应一个编号在 0-16383 之间的哈希槽，redis 会根据节点数量大致均等的将哈希槽映射到不同的节

## Redis Cluster容错

容错性，是指软件检测应用程序所运行的软件或硬件中发生的错误并从错误中恢复的能力，通常可以从系统的可靠性、可用性、可测性等几个方面来衡量。

**redis-cluster投票:容错**

|  |  |
| --- | --- |
| 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. 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. |  |

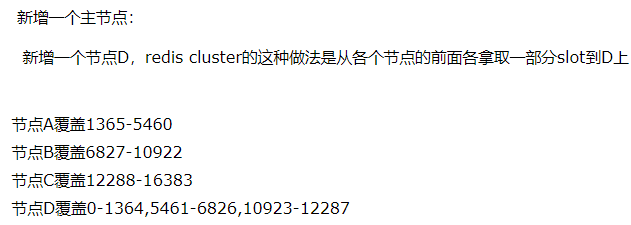
## redis-cluster节点分配

**（官方推荐）**三个主节点分别是：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



# Redis Cluster高可用

## Redis Cluster主从模式

redis cluster 为了保证数据的高可用性，加入了主从模式，一个主节点对应一个或多个从节点，主节点提供数据存取，从节点则是从主节点拉取数据备份，当这个主节点挂掉后，就会有这个从节点选取一个来充当主节点，从而保证集群不会挂掉。

集群有ABC三个主节点, 如果这3个节点都没有加入从节点，如果B挂掉了，我们就无法访问整个集群了。A和C的slot也无法访问。

所以我们在集群建立的时候，一定要为每个主节点都添加了从节点, 比如像这样, 集群包含主节点A、B、C, 以及从节点A1、B1、C1, 那么即使B挂掉系统也可以继续正确工作。

B1节点替代了B节点，所以Redis集群将会选择B1节点作为新的主节点，集群将会继续正确地提供服务。 当B重新开启后，它就会变成B1的从节点。

不过需要注意，如果节点B和B1同时挂了，Redis集群就无法继续正确地提供服务了。

**Redis Cluster总结：**

将数据自动切分split到多个节点的能力。

当集群中的一部分节点失效或者无法进行通讯时， 仍然可以继续处理命令请求的能力

# Redis Cluster集群搭建

## 简介

集群中至少应该有奇数个节点，所以搭建集群最少需要3台主机。同时每个节点至少有一个备份节点，所以下面最少需要创建使用6台机器，才能完成Redis Cluster集群（主节点、备份节点由redis-cluster集群确定）

真集群： 准备6条服务器

192.168.1.110：6739

192.168.1.111：6739

192.168.1.112：6739

假集群：一台服务器存在6个redis服务

192.168.1.111：6739 6380 6381 6382…

## 搭建流程

1、创建Redis节点安装目录

**mkdir /usr/local/redis\_cluster**  //指定目录下 创建 redis\_cluster

2、在redis\_cluster目录下，创建7001-7006个文件夹下

**mkdir 7001 7002 7003 7004 7005 7006**

3、并将redis-conf分别拷贝到7001-7006文件夹下

**cp /root/redis-4.0.1/redis.conf ./7001**

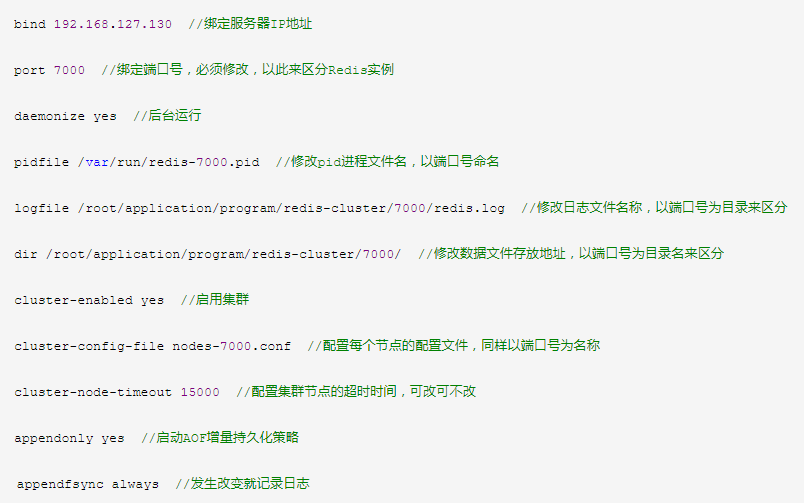
4、分别修改如下配置文件，修改如下内容

同时**protected-mode** 是为了禁止公网访问redis cache，加强redis安全的。

它启用的条件，有两个：  
**1） 没有bind IP  
2） 没有设置访问密码**

由于Linux上的redis处于安全保护模式，这就让你无法从虚拟机外部去轻松建立连接。

如果外部访问：redis.conf中设置保护模式为 **protected-mode** **no**



5、启动各个redis节点：

**将桌面redis-4.0.1/ 下src文件拷贝**到各个redis 7001-7006目录下。

**cd redis-4.0.1**  //进入桌面redis解压目录下

**cp -r ./src /usr/local/redis\_cluster/7001** //进行拷贝 依次复制7001-7006

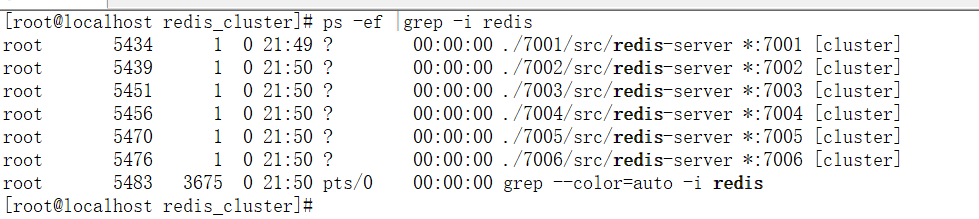
**启动各个Redis节点：**

**cd /usr/local/redis\_cluster/**  //进入redis集群配置文件目录下

**./7001/src/redis-server ./7001/redis.conf**  //依次启动 7001-7006各节点服务

6、检查Redis启动情况

**ps -ef | grep -i redis**



## 创建集群

Redis 官方提供了 redis-trib.rb 这个工具，就在解压目录的 src 目录中。（为了方便操作）将其文件复制到/usr/local/bin目录下，可直接访问此命令

**cd redis-4.0.1/**

**cd src**

**cp redis-trib.rb /usr/local/bin/**

可以直接在命令行中执行： ip:port格式

**redis-trib.rb create --replicas 1 127.0.0.1:7001 127.0.0.1:7002 127.0.0.1:7003 127.0.0.1:7004 127.0.0.1:7005 127.0.0.1:7006**

**如果出现此错误：**



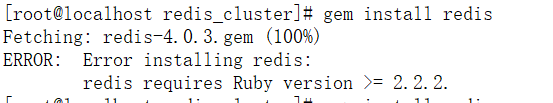
这个工具是用 ruby 实现的，所以需要安装 ruby。

## 安装ruby

**安装ruby**

**yum -y install ruby ruby-devel rubygems rpm-build**

**gem install redis**

****

**如上错误原因：Centos默认支持ruby到2.0.0，可gem 安装[redis](http://lib.csdn.net/base/redis)需要最低是2.2.2**

**解决办法是 先安装rvm，再把ruby版本提升**

### 1. 安装RVM

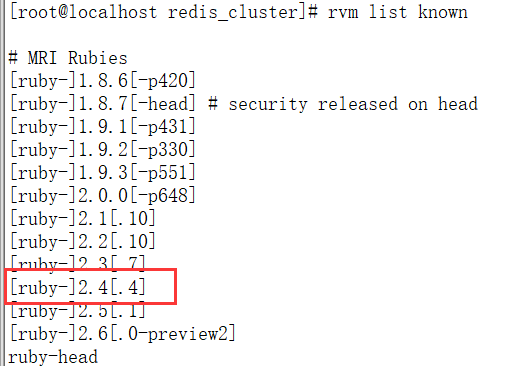
**gpg --keyserver hkp://keys.gnupg.net --recv-keys 409B6B1796C275462A1703113804BB82D39DC0E3 7D2BAF1CF37B13E2069D6956105BD0E739499BDB**

**curl -sSL https://get.rvm.io | bash -s stable**

**find / -name rvm -print**

**source /usr/local/rvm/scripts/rvm**

**rvm list known**  //查看rvm库中已知的ruby版本

****

### 安装一个ruby版本

**rvm install 2.4.4**

**使用一个ruby版本**

**rvm use 2.4.4**

**设置默认ruby版本**

**rvm use 2.4.4 --default**

**查看ruby版本**

**ruby --version**

**安装redis**

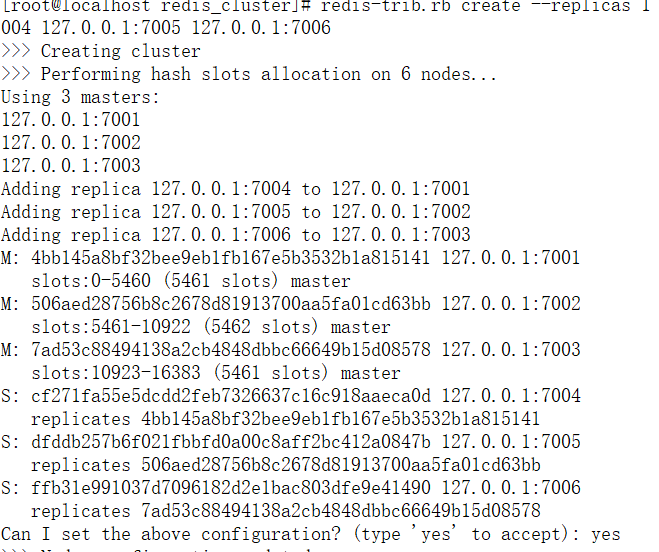
**gem install redis**

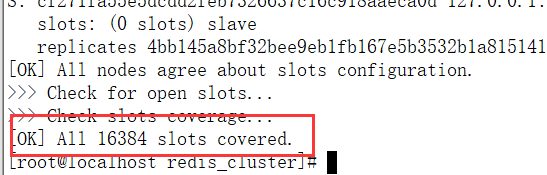
## Redis Cluster集群搭建

**ruby安装完成后，再次执行此命令：**

可以直接在命令行中执行： ip:port格式

redis-trib.rb create --replicas 1 127.0.0.1:7001 127.0.0.1:7002 127.0.0.1:7003 127.0.0.1:7004 127.0.0.1:7005 127.0.0.1:7006





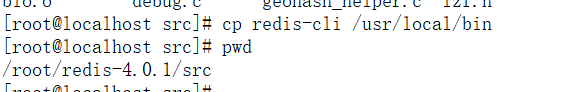
# Redis Cluster集群验证

**Redis集群测试：**

**为方便测试，可将redis-cli 客户端命令放到/usr/local/bin目录下**

**cd /root/redis-4.0.1/src //进入该目录**

**cp redis-cli /usr/local/bin**



在某台机器上（或）连接集群的7001端口的节点：

 redis-cli -h 127.0.0.1 -c -p 7001

加参数 -C 可连接到集群，因为上面 redis.conf 将 bind 改为了ip地址，所以 -h 参数不可以省略。

(在该节点下 添加对应key数据）

启动另一个集群中的客户节点：例如：7005

进行读取命令。

redis cluster在设计的时候，就考虑到了去中心化，去中间件，也就是说，集群中的每个节点都是平等的关系，都是对等的，每个节点都保存各自的数据和整个集群的状态。每个节点都和其他所有节点连接，而且这些连接保持活跃，这样就保证了我们只需要连接集群中的任意一个节点，就可以获取到其他节点的数据

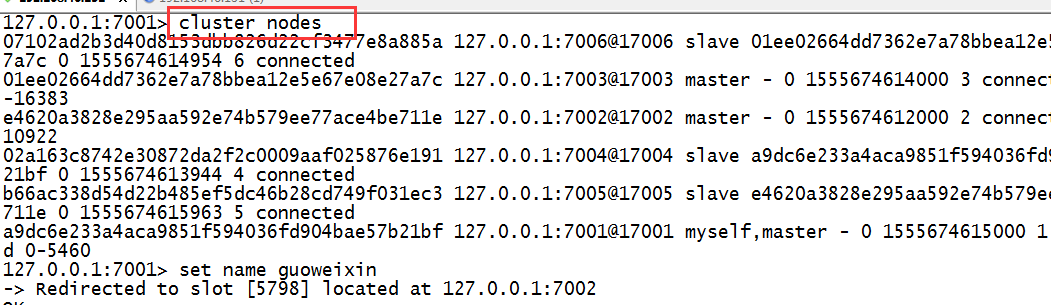
## 基本命令

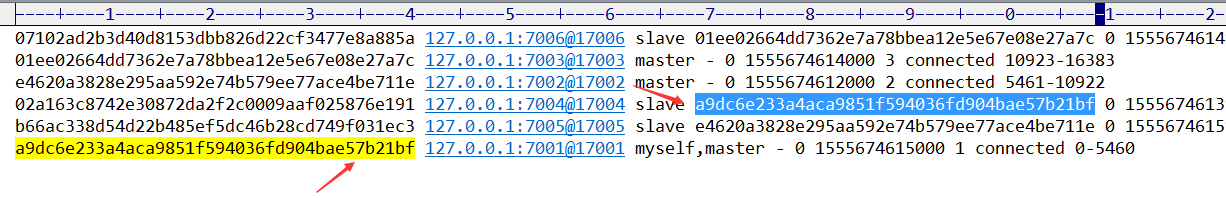
**info replication**  **通过Cluster Nodes命令和Cluster Info命令来看看集群效果**



## 测试集群数据

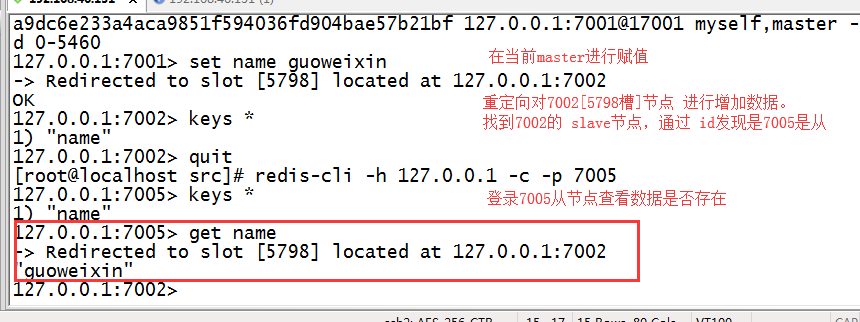
### 输入命令 cluster nodes





每个Redis的节点都有一个ID值，此ID将被此特定redis实例永久使用，以便实例在集群上下文中具有唯一的名称。 每个节点都会记住使用此ID的每个其他节点，而不是通过IP或端口。IP地址和端口可能会发生变化，但唯一的节点标识符在节点的整个生命周期内都不会改变。 我们简单地称这个标识符为节点ID。

### 测试数据



# Redis Cluster集群重新开启

**启动各个Redis节点：**

**cd /usr/local/redis\_cluster/** //进入redis集群配置文件目录下

**./7001/src/redis-server ./7001/redis.conf**  //依次启动 7001-7006各节点服务

检查**Redis启动情况**

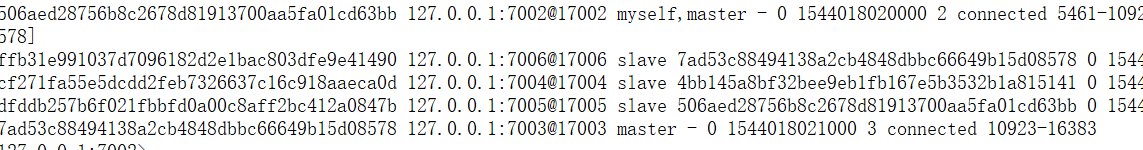
**ps -ef | grep -i redis**

在某台机器上（或）连接集群的7001端口的节点：

**redis-cli -h 127.0.0.1 -c -p 7001**

加参数 -C 可连接到集群，因为上面 redis.conf 将 bind 改为了ip地址，所以 -h 参数不可以省略。

(在该节点下 添加对应key数据）



# 代码示例

## 开启端口权限

**依次开放端口7001 7002…7006（如下命令只针对Centos7以上）**

查看已经开放的端口：firewall-cmd --list-ports

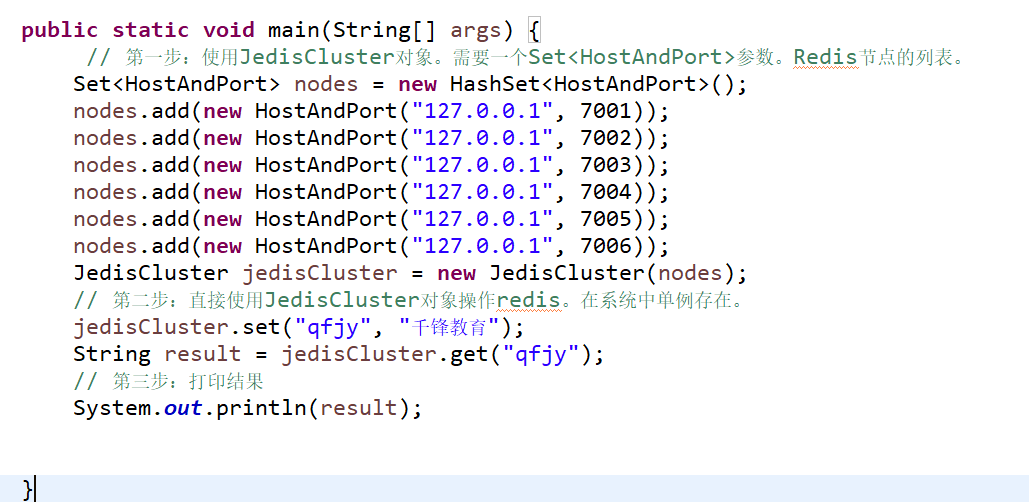
开启端口:

firewall-cmd --zone=public --add-port=7001/tcp --permanent

重启防火墙

firewall-cmd --reload #重启

* 参考文贡：https://www.sojson.com/blog/203.html



|  |
| --- |
| Set<HostAndPort> nodes = **new** LinkedHashSet<HostAndPort>();  nodes.add(**new** HostAndPort("192.168.46.131", 7001));  nodes.add(**new** HostAndPort("192.168.46.131", 7002));  nodes.add(**new** HostAndPort("192.168.46.131", 7003));  nodes.add(**new** HostAndPort("192.168.46.131", 7004));  nodes.add(**new** HostAndPort("192.168.46.131", 7005));  nodes.add(**new** HostAndPort("192.168.46.131", 7006));  JedisCluster cluster =**new** JedisCluster(nodes); |