**分布式集群电商平台搭建方案**

|  |  |  |
| --- | --- | --- |
| 状态：  [√] 草稿  [ ] 修改中  [ ] 定稿 | 文件标签： | 新建 |
| 当前版本： | 1.0 |
| 作者： | Ya`° |
| 日期： | 2017-02-15 |
| 注： | 本文档搭建为单机伪分布式集群 |

**[1.](#_Toc18685)** [环境约定 6](#_Toc18685)

**[2.](#_Toc18950)** [目录约定 6](#_Toc18950)

[3. Host约定 7](#_Toc26188)

[4. 平台结构图 7](#_Toc3141)

[5. Java环境搭建 8](#_Toc24360)

[6. Zookeeper集群搭建 8](#_Toc14471)

[6.1. 集群参数约定（节点个数5） 8](#_Toc5380)

**[6.2.](#_Toc27060)** [搭建 9](#_Toc27060)

**[6.3.](#_Toc349)** [编写zookeeper集群启动脚本 12](#_Toc349)

[6.4. 模拟节点宕机 16](#_Toc15525)

**[7.](#_Toc31159)** [Redis集群搭建 17](#_Toc31159)

[7.1. 集群参数约定（节点个数6，三主每主一从） 17](#_Toc21681)

**[7.2.](#_Toc12390)** [搭建 17](#_Toc12390)

**[7.2.1.](#_Toc11724)** [获取redis包，安装redis 17](#_Toc11724)

**[7.2.2.](#_Toc16564)** [编写redis集群启动脚本 22](#_Toc16564)

**[7.2.3.](#_Toc28833)** [安装ruby环境（redis-trib.rb工具的相关依赖） 23](#_Toc28833)

**[7.2.4.](#_Toc18722)** [使用redis-trib.rb工具关联集群 23](#_Toc18722)

[7.3. 模拟节点宕机 26](#_Toc24632)

[7.4. 配置外部访问 26](#_Toc27867)

[7.4.1. 修改redis.conf文件 26](#_Toc30260)

[7.4.2. 修改redis-trib.rb工具源码，增加redis登录验证 27](#_Toc7568)

[7.4.3. 重新关联集群 28](#_Toc10676)

[7.5. 在连接单个节点正常的情况下，连接集群时却出现拒绝连接的坑 29](#_Toc22865)

**[8.](#_Toc32328)** [Rabbitmq集群搭建 33](#_Toc32328)

[8.1. 集群参数约定 33](#_Toc9257)

[8.2. 搭建 33](#_Toc9128)

[8.2.1. 安装erlang依赖 33](#_Toc889)

[8.2.2. 安装socat依赖 33](#_Toc29707)

[8.2.3. 获取rabbitmq包，安装 34](#_Toc12659)

[8.2.3.1. 通过yum安装 34](#_Toc20050)

[8.2.3.2. 通过源码包安装 35](#_Toc12576)

[9. Tomcat参数约定 37](#_Toc29280)

[9.1. 获取tomcat包，配置tomcat 38](#_Toc28627)

[9.2. 编写tomcat启动脚本 39](#_Toc31060)

[10. Nginx安装 40](#_Toc18168)

[10.1. 获取nginx包，安装nginx 40](#_Toc4229)

[10.2. 配置base-data项目的负载均衡 41](#_Toc2466)

[10.3. 编写nginx启动脚本 42](#_Toc5487)

**[11.](#_Toc25416)** [Dubbo控制台 44](#_Toc25416)

[11.1. 获取dubbo-admin包 44](#_Toc25074)

[11.2. 安装 44](#_Toc14724)

[11.3. 查看dubbo服务 45](#_Toc21390)

**[12.](#_Toc4321)** [Mysql集群搭建 47](#_Toc4321)

[12.1. 集群参数约定 47](#_Toc11652)

[12.2. 查看系统默认安装版本，卸载 47](#_Toc13557)

[12.3. 搭建 47](#_Toc13574)

[12.3.1. 安装mysql 47](#_Toc9812)

[12.3.2. 配置sql节点 49](#_Toc8154)

[12.3.3. 安装管理节点 51](#_Toc9509)

[12.3.4. 编写启动脚本 53](#_Toc28956)

[12.3.5. 登录mysql终端 54](#_Toc9570)

[12.3.6. 数据同步测试 55](#_Toc32000)

[12.3.7. 模拟节点宕机 59](#_Toc8710)

[12.3.8. 宕机后数据同步测试 60](#_Toc21448)

[12.4. 配置mysql远程访问 62](#_Toc11956)

[13. Cas服务器配置（基础配置） 63](#_Toc5450)

[13.1. 获取cas包（cas-server-4.0.0） 63](#_Toc25507)

[13.2. 从Cas官网下载cas包流程 64](#_Toc31160)

[13.3. 配置Tomcat 服务器https连接方式 67](#_Toc24013)

[13.4. 使用jdk的keytool工具生成https证书 68](#_Toc387)

[13.5. 访问cas-server（注意配置host） 70](#_Toc7603)

[13.6. 配置cas-server为数据库验证方式 72](#_Toc3480)

[13.7. 配置cas-server为数据库验证方式（自定义登录方式） 77](#_Toc8508)

[14. 项目模型搭建 77](#_Toc27207)

[14.1. 配置cas-server服务器 77](#_Toc14702)

[14.2. 项目结构图 79](#_Toc27281)

[14.3. 创建parent-pom父类项目 79](#_Toc25366)

[14.3.1. 作用描述 83](#_Toc24238)

[14.4. 创建framework项目 83](#_Toc24991)

[14.4.1. 作用描述 86](#_Toc18447)

[14.5. 创建wms父类项目 87](#_Toc5539)

[14.6. 创建wms-core项目 89](#_Toc32299)

[14.6.1. 作用描述 93](#_Toc5371)

[14.7. 创建wms-api项目 94](#_Toc21192)

[14.7.1. 作用描述 94](#_Toc30808)

[14.8. 创建wms-web项目 94](#_Toc31155)

[14.8.1. 整合spring mvc 96](#_Toc12977)

[14.8.1.1. 配置spring mvc 96](#_Toc22266)

[14.8.1.2. 添加springmvc-servlet.xml 97](#_Toc10501)

[14.8.2. 配置cas-client 97](#_Toc12321)

[14.8.3. 创建CasTestController测试类 101](#_Toc13634)

[14.8.3.1. Debug 102](#_Toc24563)

[14.8.4. 编写自己的cas单点登录页面 103](#_Toc21840)

[14.8.5. 异常 103](#_Toc27097)

[14.8.5.1. java.security.cert.CertificateException:异常 103](#_Toc3987)

[14.8.5.2. sun.security.validator.ValidatorException:异常 104](#_Toc28887)

[14.9. 整合mybatis 105](#_Toc8290)

[14.9.1. 添加依赖 105](#_Toc30635)

[14.9.2. 添加spring-mybatis.xml配置文件 107](#_Toc24641)

[14.9.3. 配置web.xml 110](#_Toc15006)

[14.9.4. 添加数据库配置 111](#_Toc4320)

[14.10. 配置redis集群连接 112](#_Toc14587)

[14.11. 创建parent-web项目 117](#_Toc28468)

[14.11.1. 作用描述 117](#_Toc29532)

[14.12. 同样的方式组织创建oms，base-data两个项目，并添加相应的依赖关系 119](#_Toc6941)

[14.13. maven-antrun-plugin插件 119](#_Toc19743)

[14.13.1. ant介绍 119](#_Toc10615)

[14.13.2. maven-antrun-plugin控制平台公共的布局页面 119](#_Toc12997)

[14.13.3. maven-antrun-plugin配置 121](#_Toc7054)

[14.13.4. 配置dubbo 121](#_Toc29705)

[14.13.4.1. 修改application-core.properties，添加配置zookeeper配置 121](#_Toc3910)

[14.13.4.2. 配置注册中心，创建spring-dubbo-core.xml 122](#_Toc30704)

[14.13.4.3. 配置消费者，创建spring-dubbo-consumer.xml 124](#_Toc21886)

[14.13.4.4. 配置提供者，创建spring-dubbo-consumer.xml 125](#_Toc25126)

[14.13.4.5. 修改web.xml 125](#_Toc18537)

[14.14. 配置rabbitmq连接 126](#_Toc15451)

[14.15. 功能实现 131](#_Toc18946)

[14.15.1. base-data-web 131](#_Toc2949)

[14.15.2. oms-web 132](#_Toc21185)

[14.15.3. wms-web 133](#_Toc14384)

[14.16. Java Service Wrapper 133](#_Toc31536)

[14.16.1. 介绍 133](#_Toc10812)

[14.16.2. 获取安装包 133](#_Toc32545)

[14.16.3. 使用 Java Service Wrapper将tomcat注册为window服务配置 134](#_Toc15302)

[14.16.3.1. 拷贝如下文件到指定目录 134](#_Toc28556)

[14.16.3.2. 修改${CATALINA\_HOME}/conf目录下的wrapper.conf配置 136](#_Toc1032)

[14.16.3.3. 设置读取的classpath 136](#_Toc12383)

[14.16.3.4. 配置tomcat启动类 136](#_Toc17003)

[14.16.3.5. 配置应用名称 137](#_Toc2499)

[14.16.3.6. 查看 137](#_Toc13107)

[14.16.4. 创建jsw项目，独立dubbo provider 137](#_Toc4266)

[14.16.4.1. 创建项目 137](#_Toc8801)

[14.16.4.2. 创建如下目录结构，并拷贝相关文件到指定目录 140](#_Toc2204)

[14.16.4.3. 添加maven-assembly-plugin插件配置 141](#_Toc30395)

[14.16.4.4. 创建maven-assembly-plugin的assembly文件 143](#_Toc12573)

[14.16.4.5. 创建oms-jsw-context.xml文件 145](#_Toc32294)

[14.16.4.6. wrapper.conf配置 146](#_Toc17195)

[14.16.4.7. intall 147](#_Toc19888)

[14.16.4.8. 部署 147](#_Toc18325)

[14.16.4.9. 查看服务 148](#_Toc23288)

[14.16.4.10. 剔除oms-web项目中的dubbo provider 148](#_Toc30024)

**[15.](#_Toc25054)** [附录 149](#_Toc25054)

**[15.1.](#_Toc32448)** [vi编辑器没有颜色 149](#_Toc32448)

**[15.2.](#_Toc153)** [Rz命令安装 149](#_Toc153)

**[15.3.](#_Toc12351)** [tree显示目录树 149](#_Toc12351)

[15.4. 关闭防火墙 150](#_Toc1452)

[15.4.1. 临时关闭防火墙命令 150](#_Toc3191)

**[15.4.2.](#_Toc1616)** [永久关闭防火墙命令 150](#_Toc1616)

[15.5. 查看系统位数命令 150](#_Toc17353)

[15.6. 查看隐藏文件命令 150](#_Toc18756)

# 环境约定

|  |  |  |
| --- | --- | --- |
| ISO | CentOS6.5X64 |  |
| JDK | jdk1.7.0\_80 |  |
| dubbo | dubbo-2.5 | 提供RPC远程调用 |
| rabbitMQ | rabbitmq-server-3.6.6 | 消息中间件 |
| zookeeper | zookeeper-3.4.9 | dubbo注册中心 |
| redis | redis-3.2.8 |  |
| mysql | mysql-cluster-7.4.13 |  |
| nginx | nginx-1.10.3 | http层面的负载均衡 |
| tomcat | apache-tomcat-8.5.0 |  |

# 目录约定

根目录下创建app目录

[root@localhost /]# mkdir /app



最终目录结构效果

app

├── jsw --jsw目录

├── nginx --nginx目录

├── software

│ ├── mysql-group --mysql集群安装目录

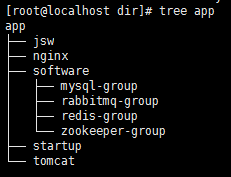
│ ├── rabbitmq-group --mq集群安装目录

│ ├── redis-group --redis集群安装目录

│ └── zookeeper-group --zk集群安装目录

├── startup --启动脚本文件存放目录

└── tomcat --tomcat目录



# Host约定

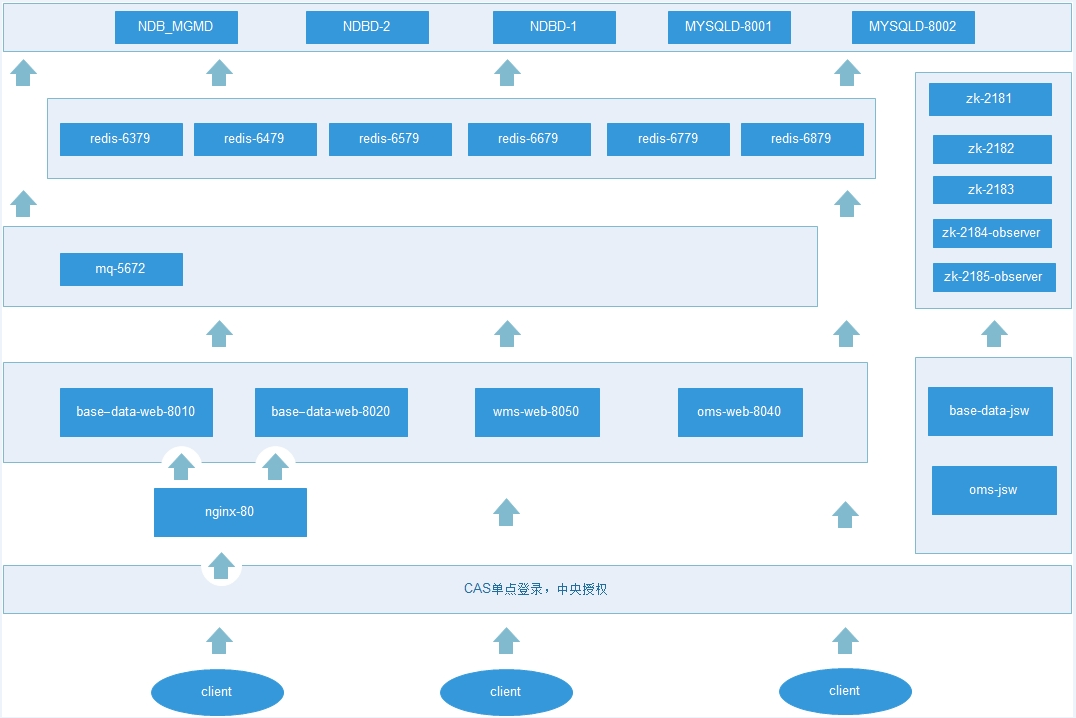
127.0.0.1 base-data.cluster.com

127.0.0.1 oms.cluster.com

127.0.0.1 wms.cluster.com

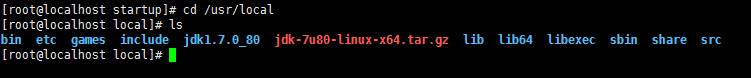
127.0.0.1 cas.cluster.com

# 平台结构图



# Java环境搭建

[root@localhost local]# cd /usr/local;tar -zxvf jdk-7u80-linux-x64.tar.gz



[root@localhost local]# cd /etc/profile.d

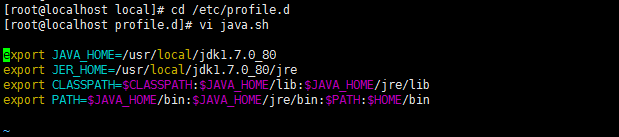
[root@localhost profile.d]# vi java.sh

export JAVA\_HOME=/usr/local/jdk1.7.0\_80

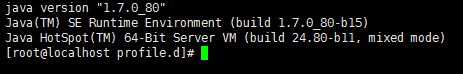
export JER\_HOME=/usr/local/jdk1.7.0\_80/jre

export CLASSPATH=$CLASSPATH:$JAVA\_HOME/lib:$JAVA\_HOME/jre/lib

export PATH=$JAVA\_HOME/bin:$JAVA\_HOME/jre/bin:$PATH:$HOME/bin



[root@localhost profile.d]# java -version



# Zookeeper集群搭建

server.A=B:C:D

A：myid（当前节点ID）

B：主机IP或域名

C：follower与leader通信端口

D：选举leader端口

## 集群参数约定（节点个数5）

|  |  |  |  |  |
| --- | --- | --- | --- | --- |
| **A** | **clientPort** | **C** | **D** | **role** |
| 1 | 2181 | 2881 | 3881 |  |
| 2 | 2182 | 2882 | 3882 |  |
| 3 | 2183 | 2883 | 3883 |  |
| 4 | 2184 | 2884 | 3884 | observer |
| 5 | 2185 | 2885 | 3885 | observer |

**节点1-3：**

随机产生一台leader，剩余的两台为follower，leader负责广播follower写请求并发起投票，follower进行投票，票数过半写请求成功

**节点4-5：**

观察者角色，处理读请求，不参与leader投票

## 搭建

[root@localhost zookeeper-group]# wget <http://apache.fayea.com/zookeeper/zookeeper-3.4.9/zookeeper-3.4.9.tar.gz>



[root@localhost zookeeper-group]# tar -zxvf zookeeper-3.4.9.tar.gz



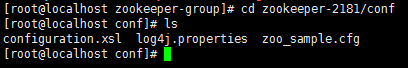
[root@localhost zookeeper-group]# mv zookeeper-3.4.9 zookeeper-2181

[root@localhost zookeeper-group]# ls



[root@localhost zookeeper-group]# cd zookeeper-2181/conf

[root@localhost conf]# ls



[root@localhost conf]# cp zoo\_sample.cfg zoo.cfg

[root@localhost conf]# ls



[root@localhost conf]# vi zoo.cfg



加入如下配置

注意dataDir 默认配置已经存在

dataDir=/app/software/zookeeper-group/zookeeper-2181/data

dataLogDir=/app/software/zookeeper-group/zookeeper-2181/logs

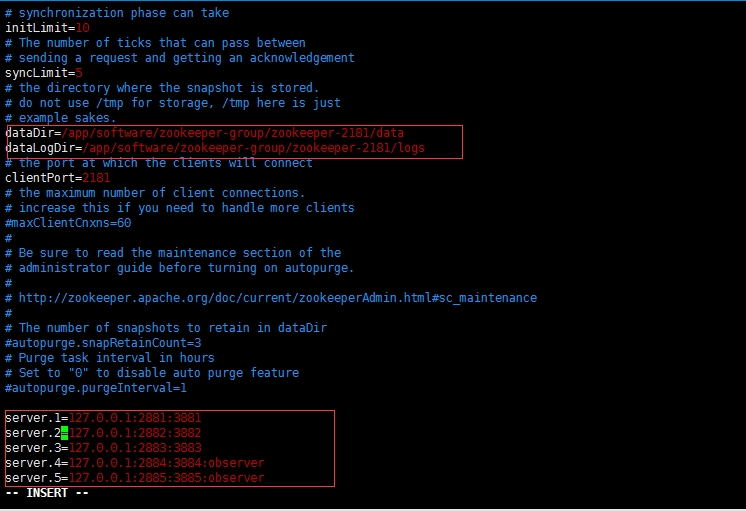
server.1=127.0.0.1:2881:3881

server.2=127.0.0.1:2882:3882

server.3=127.0.0.1:2883:3883

server.4=127.0.0.1:2884:3884:observer

server.5=127.0.0.1:2885:3885:observer



[root@localhost conf]# cd ../;mkdir data logs



[root@localhost zookeeper-2181]# ls -l



[root@localhost zookeeper-2181]# cd ../



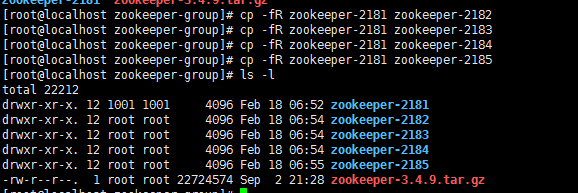
[root@localhost zookeeper-group]# cp -fR zookeeper-2181 zookeeper-2182

[root@localhost zookeeper-group]# cp -fR zookeeper-2181 zookeeper-2183

[root@localhost zookeeper-group]# cp -fR zookeeper-2181 zookeeper-2184

[root@localhost zookeeper-group]# cp -fR zookeeper-2181 zookeeper-2185

[root@localhost zookeeper-group]# ls -l



依次修改各个目录下的zoo.cfg文件

[root@localhost zookeeper-group]# cd zookeeper-2182/conf



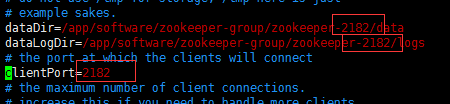
[root@localhost zookeeper-group]# vi zoo.cfg



:%s/2181/2182/g

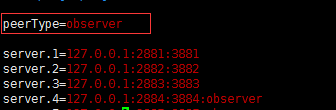






2184和2185为观察者角色

需要添加peerType=observer



[root@localhost zookeeper-group]# cd zookeeper-2181/data/

[root@localhost data]# vi myid





其他4个目录依次创建，分别为2,3,4,5

## 编写zookeeper集群启动脚本

[root@localhost zookeeper-group]# cd /app/startup/



[root@localhost startup]# vi zookeeper.sh



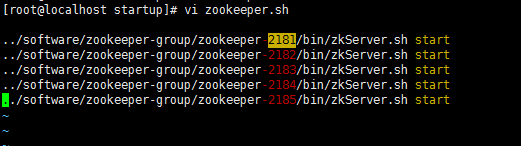
../software/zookeeper-group/zookeeper-2181/bin/zkServer.sh start

../software/zookeeper-group/zookeeper-2182/bin/zkServer.sh start

../software/zookeeper-group/zookeeper-2183/bin/zkServer.sh start

../software/zookeeper-group/zookeeper-2184/bin/zkServer.sh start

../software/zookeeper-group/zookeeper-2185/bin/zkServer.sh start



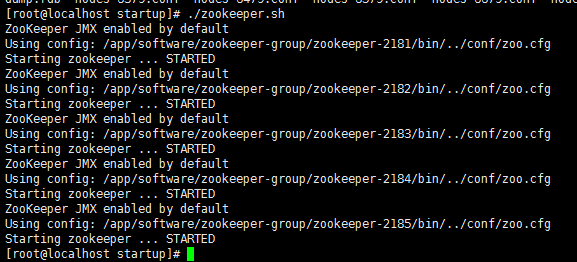
赋予脚本启动权限

[root@localhost startup]# chmod 777 zookeeper.sh

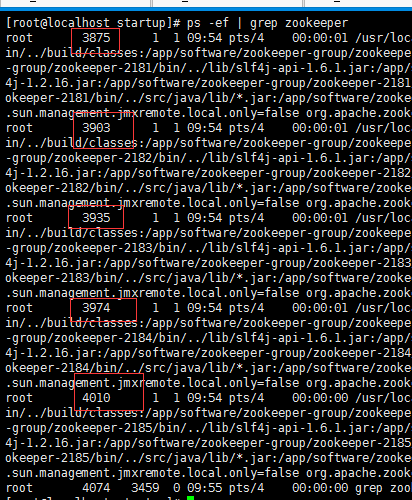




[root@localhost startup]# ./zookeeper.sh



[root@localhost startup]# ps -ef | grep zookeeper



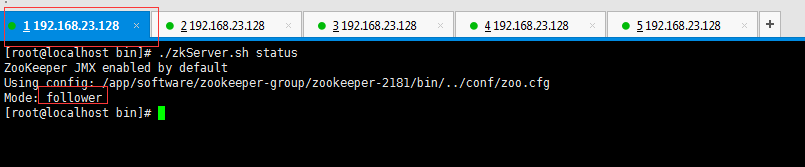
5个zookeeper进程，集群启动成功

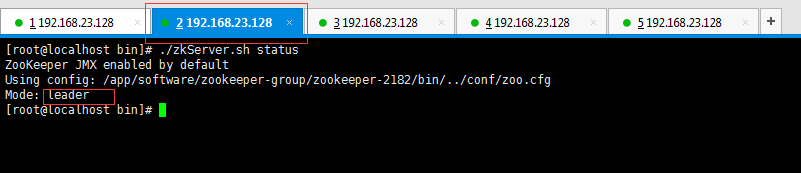
[root@localhost startup]# cd /app/software/zookeeper-group/zookeeper-2181/bin



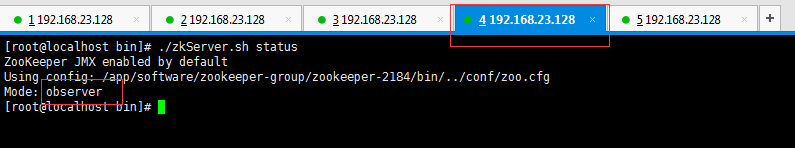


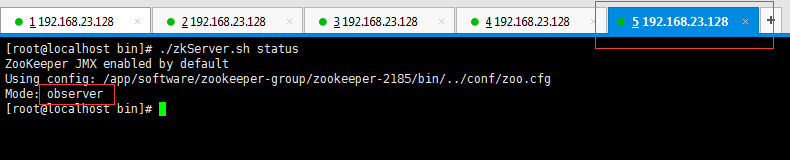
查看集群角色状态，跟设置一样，节点1-3一台leader两台follow，节点4-5observer







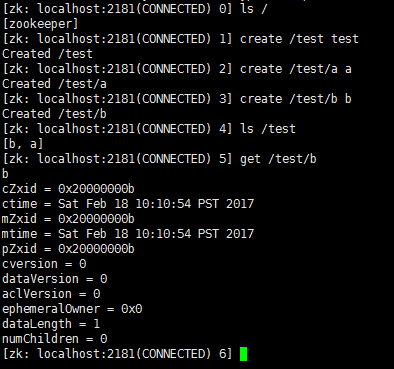




客户端登陆zookeeper



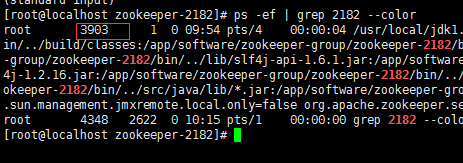
测试



## 模拟节点宕机

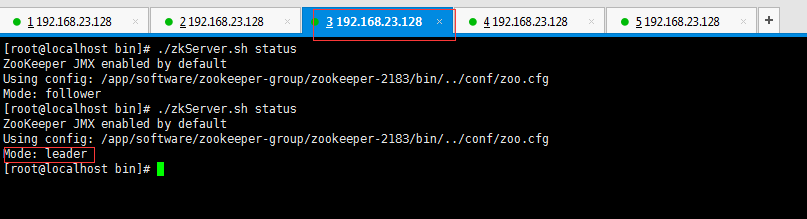
模拟节点宕机，杀死leader

[root@localhost zookeeper-2182]# ps -ef | grep 2182 --color



[root@localhost zookeeper-2182]# kill -9 3903

杀死leader后，节点3选举为leader



# Redis集群搭建

## 集群参数约定（节点个数6，三主每主一从）

|  |  |
| --- | --- |
| **port** | **role** |
| **6379** |  |
| **6479** |  |
| **6579** |  |
| **6679** |  |
| **6779** |  |
| **6879** |  |

## 搭建

### 获取redis包，安装redis

[root@localhost redis-group]# wget <http://download.redis.io/releases/redis-3.2.8.tar.gz>

[root@localhost redis-group]# tar xzf redis-3.2.8.tar.gz



[root@localhost redis-group]# mv redis-3.2.8 redis-6379

[root@localhost redis-group]# ls

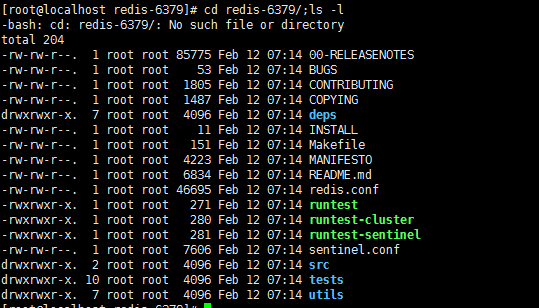


[root@localhost redis-group]# mkdir redis-6479 redis-6579 redis-6679 redis-6779 redis-6879

[root@localhost redis-group]# ls



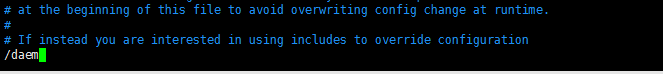
[root@localhost redis-6379]# cd redis-6379/;ls -l



[root@localhost redis-6379]# vi redis.conf

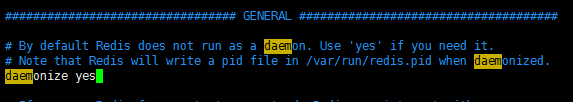


vi 中搜索daem关键字



vi 键盘a

修改daemonize no--> daemonize yes（开启后台启动）

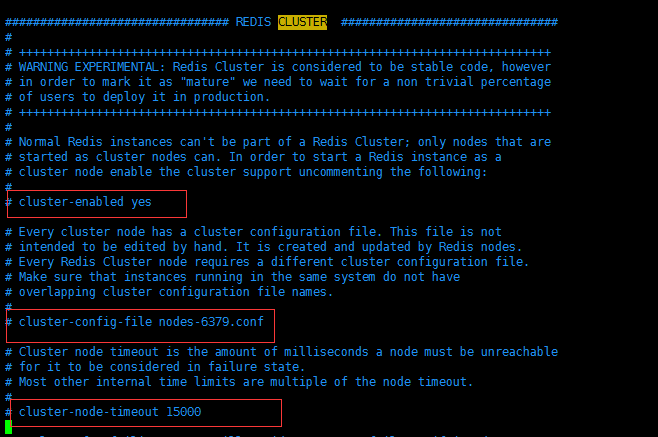


vi 键盘ESC

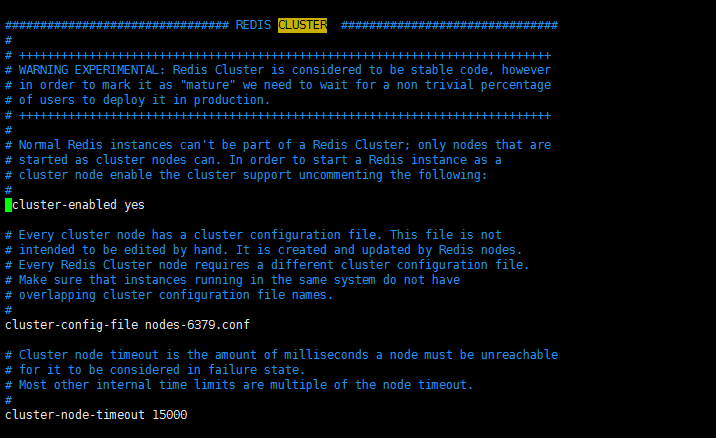
vi 中搜索CLUSTER



vi 键盘a



开启集群模式



vi 键盘ESC

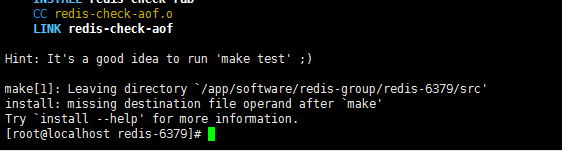
vi 键盘 shift+zz(大写ZZ)保存并退出



[root@localhost redis-6379]# make;install make



安装完成



拷贝配置到另外5个节点

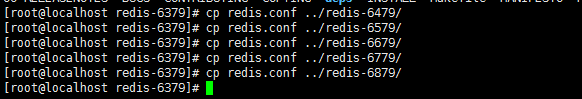
[root@localhost redis-6379]# cp redis.conf ../redis-6479/

[root@localhost redis-6379]# cp redis.conf ../redis-6579/

[root@localhost redis-6379]# cp redis.conf ../redis-6679/

[root@localhost redis-6379]# cp redis.conf ../redis-6779/

[root@localhost redis-6379]# cp redis.conf ../redis-6879/



[root@localhost redis-6379]# tree -L 2 /app/software/redis-group/





[root@localhost redis-6379]# cd ../redis-6479/

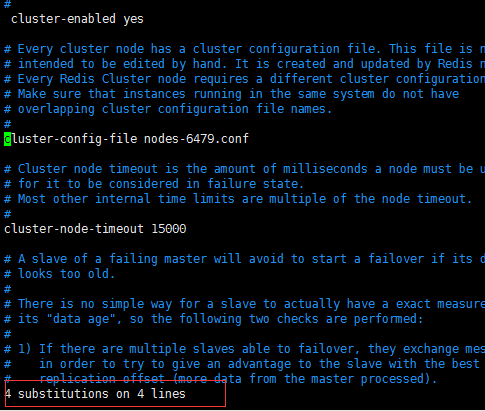
[root@localhost redis-6479]# vi redis.conf



将redis.conf中6379文本替换成6479

:%s/6379/6479/g



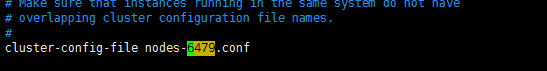


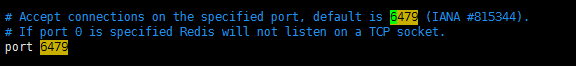
检查是否替换成功

vi 搜索6479



vi 键盘n搜索下一行







同样的方式分别修改以下目录的redis.conf配置，端口为目录端口



### 编写redis集群启动脚本

[root@localhost redis-6879]# cd /app/startup/



[root@localhost startup]# vi redis.sh



../software/redis-group/redis-6379/src/redis-server ../software/redis-group/redis-6379/redis.conf

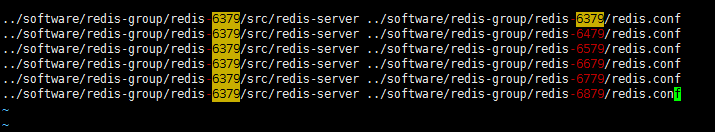
../software/redis-group/redis-6379/src/redis-server ../software/redis-group/redis-6479/redis.conf

../software/redis-group/redis-6379/src/redis-server ../software/redis-group/redis-6579/redis.conf

../software/redis-group/redis-6379/src/redis-server ../software/redis-group/redis-6679/redis.conf

../software/redis-group/redis-6379/src/redis-server ../software/redis-group/redis-6779/redis.conf

../software/redis-group/redis-6379/src/redis-server ../software/redis-group/redis-6879/redis.conf



赋予脚本启动权限

[root@localhost startup]# chmod 777 redis.sh

[root@localhost startup]# ls

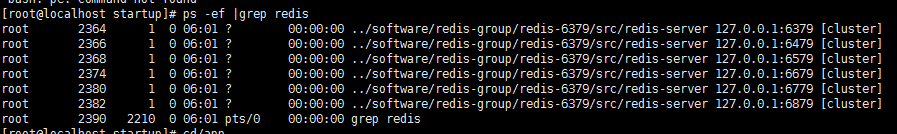


[root@localhost startup]# ./redis.sh

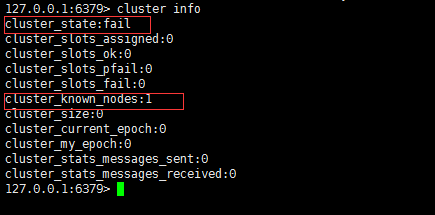


查看redis进程，看集群是否启动成功

[root@localhost startup]# ps -ef| grep redis



此时的redis集群还不可用



### 安装ruby环境（redis-trib.rb工具的相关依赖）

[root@localhost redis-group]# yum -y install ruby

[root@localhost redis-group]# yum -y install rubygems

[root@localhost redis-group]# gem install redis

### 使用redis-trib.rb工具关联集群

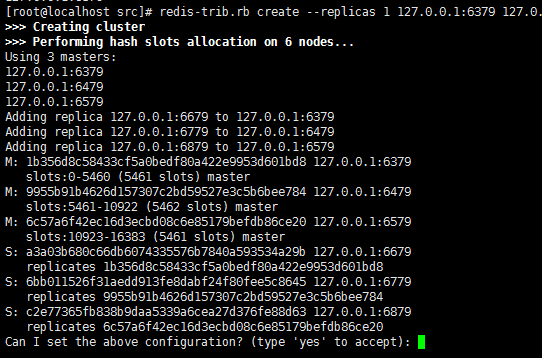
[root@localhost startup]# cd /app/software/redis-group/redis-6379/src



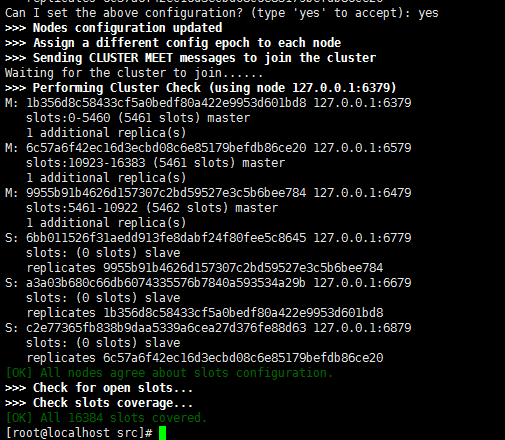
[root@localhost src]# redis-trib.rb create --replicas 1 127.0.0.1:6379 127.0.0.1:6479 127.0.0.1:6579 127.0.0.1:6679 127.0.0.1:6779 127.0.0.1:6879

参数解释：replicas 为每个master创建一个slave





yes



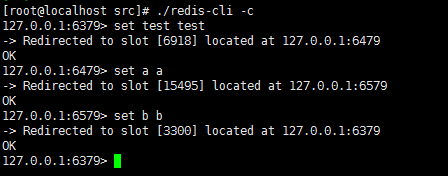
成功！

以集群的方式登陆redis

[root@localhost src]# ./redis-cli -c



测试redis集群



查看集群状态

127.0.0.1:6379> cluster info

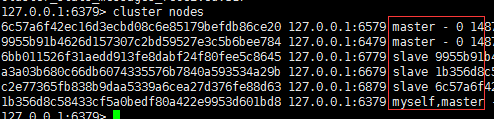
集群状态ok，节点数量6



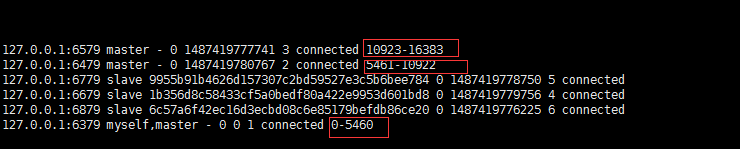
查看集群节点

127.0.0.1:6379> cluster nodes

三主，三从



Slot的分配

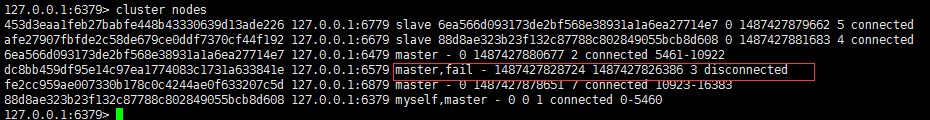


## 模拟节点宕机

模拟节点宕机，杀死master

例如本文档杀死端口为6579的master进程

kill -9 进程id



变化：原master的slave变master

保证3个master不宕机，集群正常使用，也就是说本集群允许最多挂掉3个节点

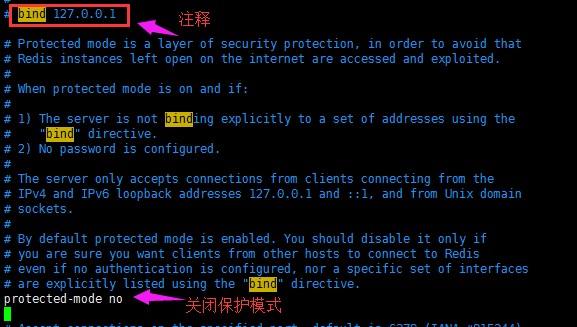
## 配置外部访问

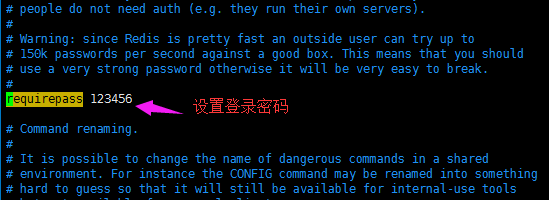
### 修改redis.conf文件

[root@localhost ~]# cd /app/software/redis-group/redis-6379/

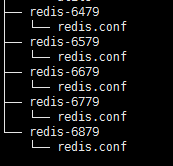
[root@localhost redis-6379]# vi redis.conf







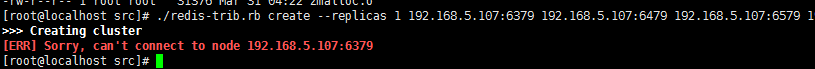
同样的方式修改其余的5个配置文件



### 修改redis-trib.rb工具源码，增加redis登录验证

因为之前使用redis-trib.rb关联集群时使用的是127.0.0.1的ip地址，这样会导致项目无法连接上集群的情况，所以需要使用192的ip重新关联集群

[root@localhost src]# ./redis-trib.rb create --replicas 1 192.168.5.107:6379 192.168.5.107:6479 192.168.5.107:6579 192.168.5.107:6679 192.168.5.107:6779 192.168.5.107:6879

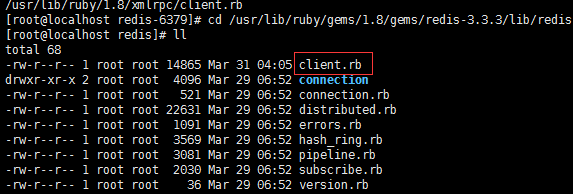


redis-trib.rb工具不支持密码参数,在配置了redis密码的情况下，工具无法使用，需要修改工具源码，加入redis登录验证密码

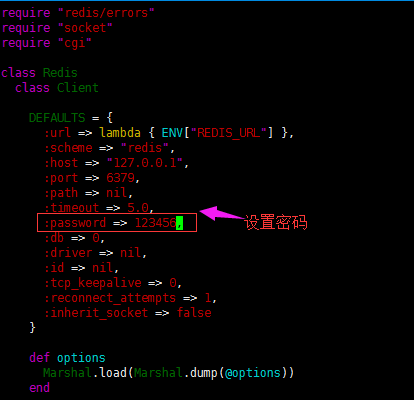
[root@localhost redis-6379]# find / -name client.rb



[root@localhost redis-6379]# cd /usr/lib/ruby/gems/1.8/gems/redis-3.3.3/lib/redis

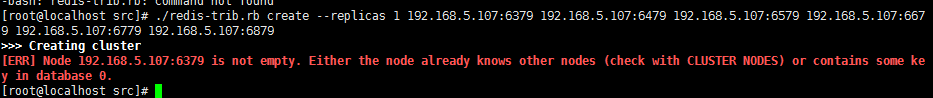


[root@localhost redis]# vi client.rb



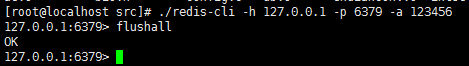
### 重新关联集群

再次执行



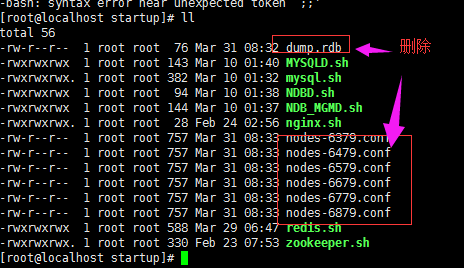
发现节点不为空，需要先清空redis节点数据

[root@localhost src]# ./redis-cli -h 127.0.0.1 -p 6379 -a 123456



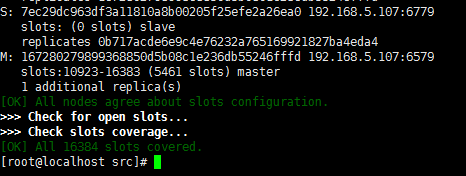
同样的方式清空剩余2个master的数据

删除redis cluster节点文件，或者用redis-trib.rb工具删除集群关联



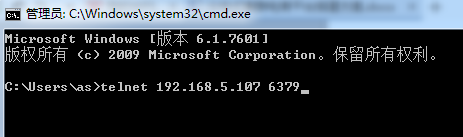
重新关联

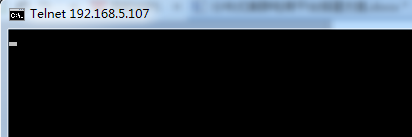
[root@localhost src]# ./redis-trib.rb create --replicas 1 192.168.5.107:6379 192.168.5.107:6479 192.168.5.107:6579 192.168.5.107:6679 192.168.5.107:6779 192.168.5.107:6879



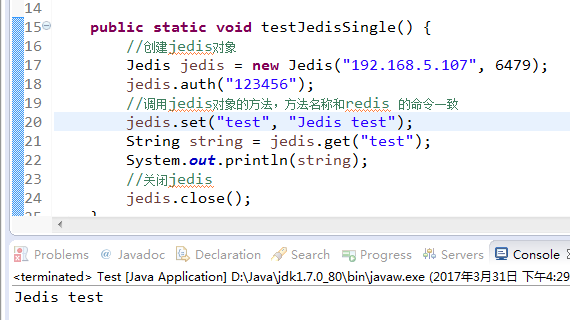
## 在连接单个节点正常的情况下，连接集群时却出现拒绝连接的坑

Telnet正常





Jedis连接集群单个节点正常



但JedisCluster连接集群时却拒绝连接

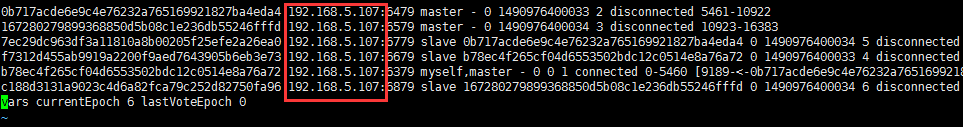


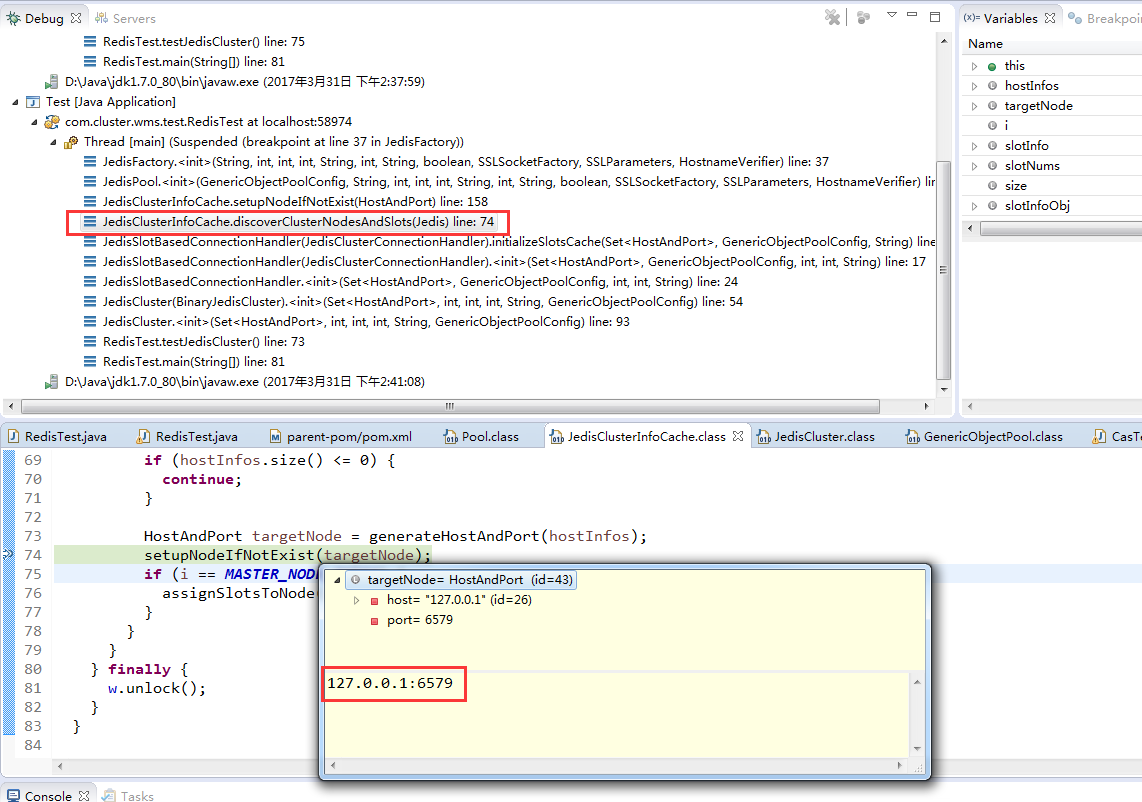
调试源码发现jedis工厂创建jedis实例时，传入的host参数居然变成127.0.0.1了，

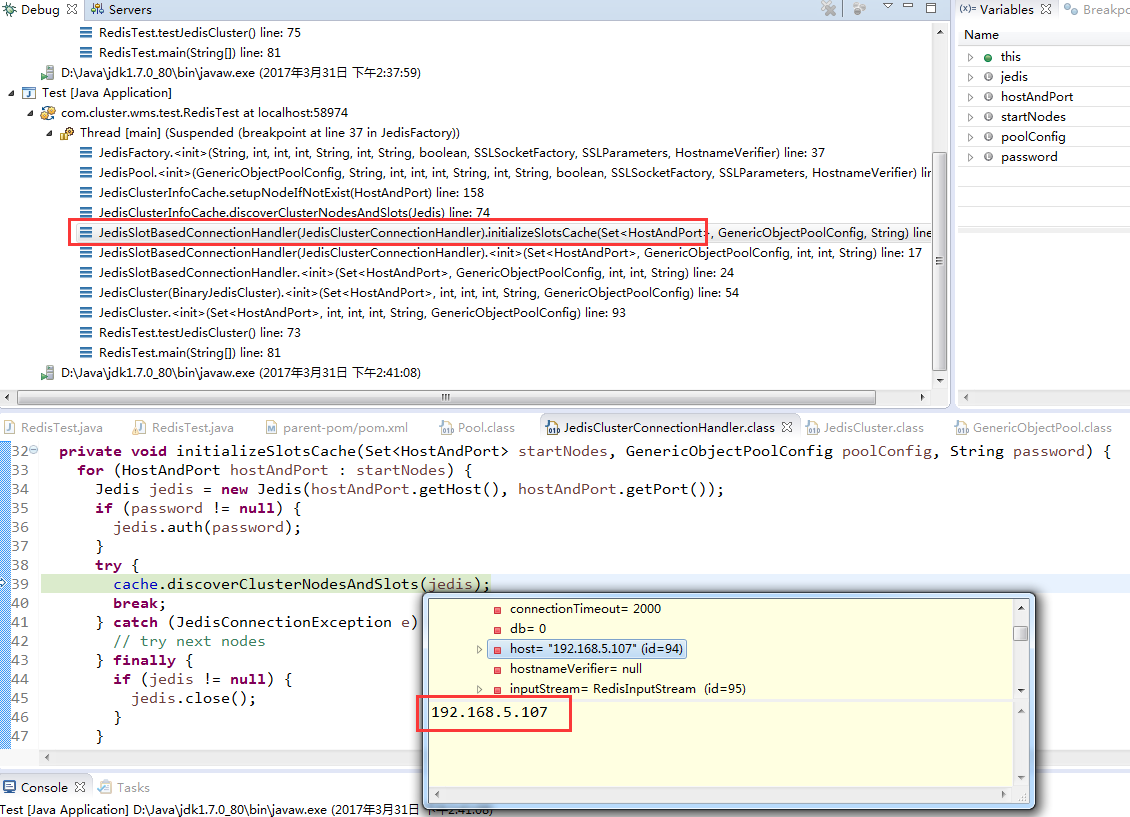


追溯后发现jedis工厂创建jedis实例时传入的host并不是我们传入的host，而是

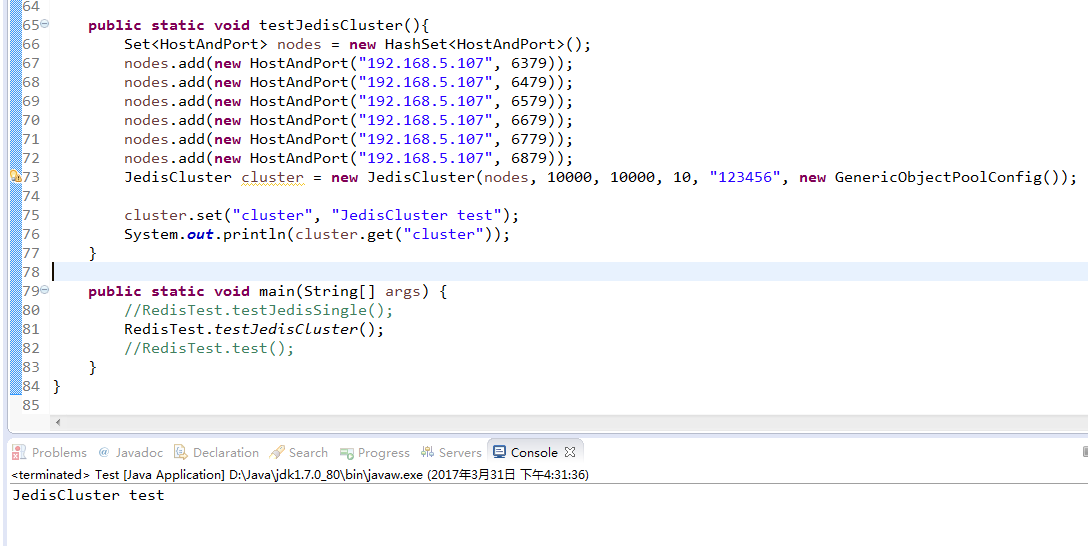
JedisClusterInfoCache类获取集群节点中的host传入的，也就是说在关联集群时需要使用192的ip，不能使用127







最后测试正常



# Rabbitmq集群搭建

## 集群参数约定

集群环境待搭建...

## 搭建

### 安装erlang依赖

[root@bogon rabbitmq-group]# wget <http://www.rabbitmq.com/releases/erlang/erlang-18.1-1.el6.x86_64.rpm>



[root@bogon rabbitmq-group]# rpm -ivh erlang-18.1-1.el6.x86\_64.rpm



### 安装socat依赖

wget ?no-cache http://www.convirture.com/repos/definitionsrhel/6.x/convirt.repo -O /etc/yum.repos.d/convirt.repo



[root@bogon rabbitmq-group]# yum install socat



### 获取rabbitmq包，安装

#### 通过yum安装

[root@bogon rabbitmq-group]#

wget http://www.rabbitmq.com/releases/rabbitmq-server/v3.6.6/rabbitmq-server-3.6.6-1.el6.noarch.rpm



[root@bogon rabbitmq-group]# rpm -ivh rabbitmq-server-3.6.6-1.el6.noarch.rpm



默认安装路径

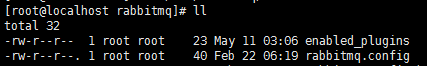


配置远程访问和web插件

[root@localhost ~]# cd /etc/rabbitmq/



创建rabbitmq.config文件和enabled\_plugins文件（注意文件内容格式，以英文句号结尾）



[{rabbit, [{loopback\_users, [test]}]}].



[rabbitmq\_management].



#### 通过源码包安装

基于源码安装的为绿色版，直接解压就能使用

[root@localhost rabbitmq-group]# wget http://www.rabbitmq.com/releases/rabbitmq-server/v3.6.6/rabbitmq-server-generic-unix-3.6.6.tar.xz



[root@localhost rabbitmq-group]# xz -d rabbitmq-server-generic-unix-3.6.6.tar.xz



[root@localhost rabbitmq-group]# tar -xvf rabbitmq-server-generic-unix-3.6.6.tar

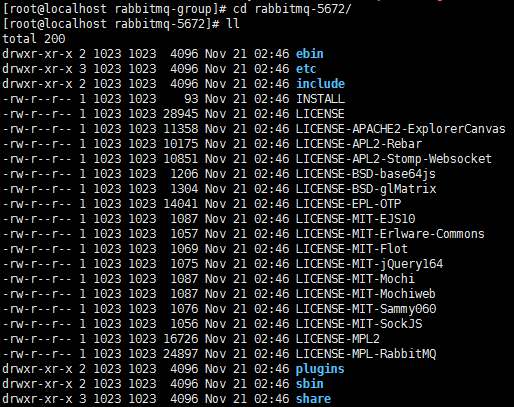


[root@localhost rabbitmq-group]# mv rabbitmq\_server-3.6.6/ rabbitmq-5672





[root@localhost rabbitmq-group]# cd rabbitmq-5672/



配置远程访问：

方式一：直接修改配置文件rabbit.app，

方式二：在${RABBITMQ\_HOME}/etc/rabbitmq目录下创建rabbitmq.config文件

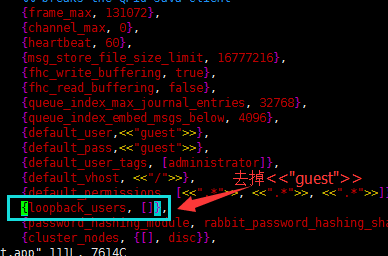
方式一

[root@localhost rabbitmq-5672]# cd ebin/



[root@localhost ebin]# vi rabbit.app





方式二

[{rabbit, [{loopback\_users, [test]}]}].



配置web插件

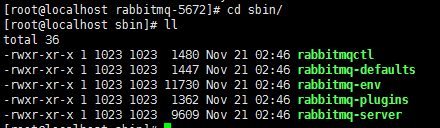
在${RABBITMQ\_HOME}/etc/rabbitmq目录下创建enabled\_plugins文件

[rabbitmq\_management].

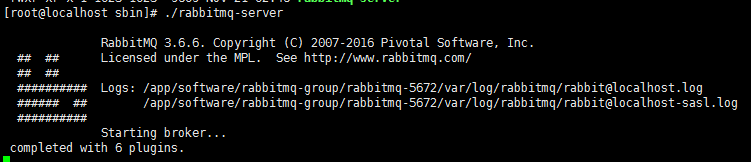


启动

[root@localhost rabbitmq-5672]# cd sbin/

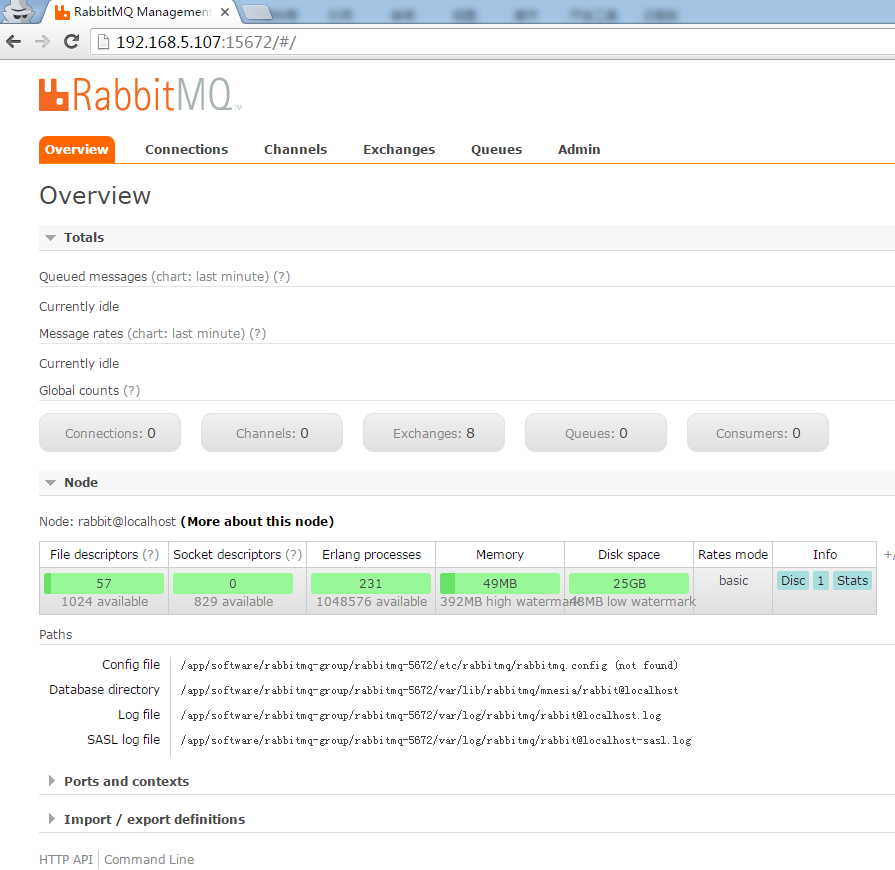


[root@localhost sbin]# ./rabbitmq-server



用户名：guest

密码：guest



# Tomcat参数约定

|  |  |  |  |
| --- | --- | --- | --- |
| **Server name** | **Tomcat port** | **HTTP/1.1 port** | **AJP/1.3 port** |
| base-data-web-8010 | 8001 | 8010 | 8100 |
| base-data-web-8020 | 8002 | 8020 | 8200 |
| oms-web-8040 | 8004 | 8040 | 8400 |
| wms-web-8050 | 8005 | 8050 | 8500 |
| cas-server-8060 | 8006 | 8060 | 8600 |
| dubbo-admin-8070 | 8007 | 8070 | 8700 |

## 获取tomcat包，配置tomcat

[root@localhost /]# cd /app/tomcat/



[root@localhost tomcat]# wget http://archive.apache.org/dist/tomcat/tomcat-8/v8.5.0/bin/apache-tomcat-8.5.0.tar.gz





[root@localhost tomcat]# tar -zxvf apache-tomcat-8.5.0.tar.gz



[root@localhost tomcat]# mv apache-tomcat-8.5.0 base-data-web-8010

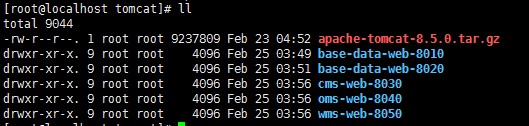


[root@localhost tomcat]# cp -fR base-data-web-8010/ base-data-web-8020/

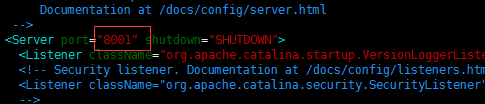
[root@localhost tomcat]# cp -fR base-data-web-8010/ cms-web-8030/

[root@localhost tomcat]# cp -fR base-data-web-8010/ oms-web-8040/

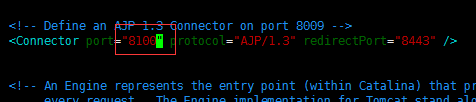
[root@localhost tomcat]# cp -fR base-data-web-8010/ wms-web-8050/



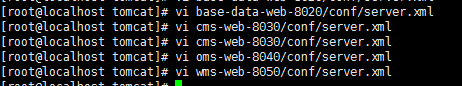
[root@localhost tomcat]# vi base-data-web-8010/conf/server.xml







分别编辑剩余五个，端口参数见上表



## 编写tomcat启动脚本

[root@localhost tomcat]# cd /app/startup/



[root@localhost startup]# vi tomcat.sh



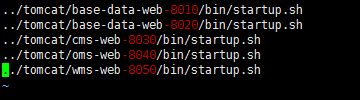
../tomcat/base-data-web-8010/bin/startup.sh

../tomcat/base-data-web-8020/bin/startup.sh

../tomcat/cms-web-8030/bin/startup.sh

../tomcat/oms-web-8040/bin/startup.sh

../tomcat/wms-web-8050/bin/startup.sh



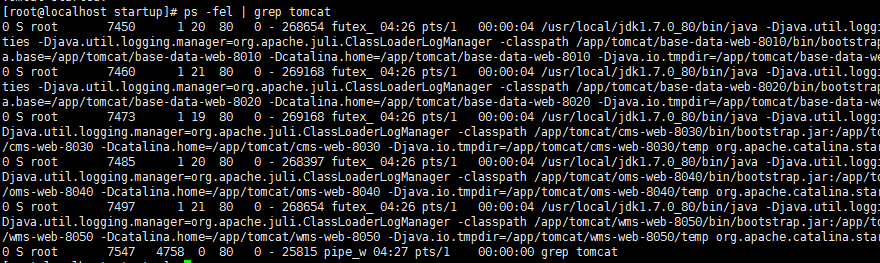
[root@localhost startup]# chmod 777 tomcat.sh







[root@localhost startup]# ps -fel | grep tomcat



启动成功！

# Nginx安装

## 获取nginx包，安装nginx

[root@bogon nginx]# wget <http://nginx.org/download/nginx-1.10.3.tar.gz>





[root@bogon nginx]# tar -zxvf nginx-1.10.3.tar.gz



[root@localhost nginx]# mkdir base-data-80





[root@localhost nginx]# cd nginx-1.10.3



[root@localhost nginx-1.10.3]# yum -y install pcre-devel



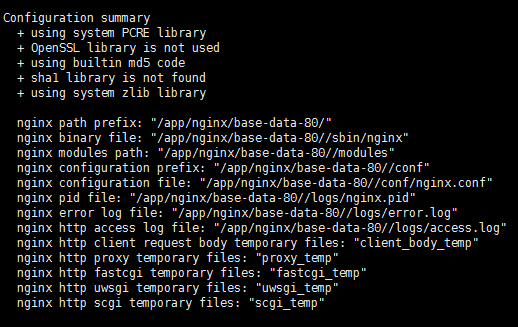
[root@localhost nginx-1.10.3]# yum -y install zlib-devel



[root@localhost nginx-1.10.3]# ./configure --prefix=/app/nginx/base-data-80/

（可用--prefix=path指定安装目录，默认  /usr/local/nginx）





[root@localhost nginx-1.10.3]# make ;make install

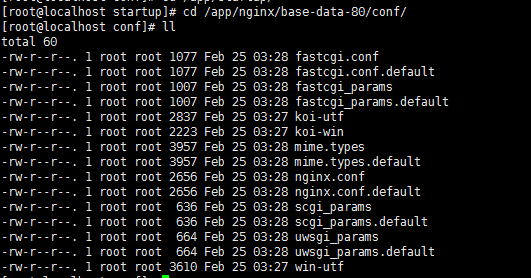




安装成功！

## 配置base-data项目的负载均衡

[root@localhost startup]# cd /app/nginx/base-data-80/conf/



[root@localhost conf]# vi nginx.conf



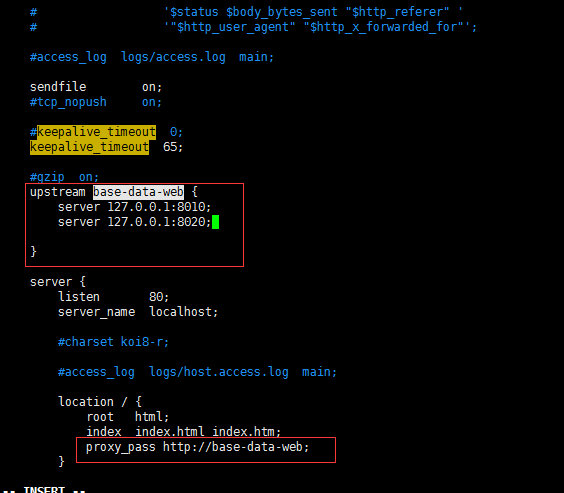
upstream base-data-web {

server 127.0.0.1:8010;

server 127.0.0.1:8020;

}

proxy\_pass http://base-data-web;



## 编写nginx启动脚本

[root@localhost nginx-1.10.3]# cd /app/startup/



[root@bogon startup]# vi nginx.sh



/app/nginx/base-data-80/sbin/nginx



[root@bogon startup]# chmod 777 nginx

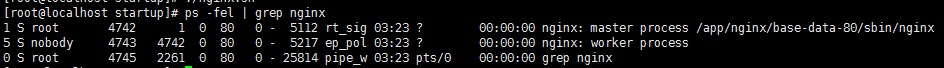




[root@bogon startup]# ./nginx.sh



[root@localhost startup]# ps -fel | grep nginx



测试，为了看清nginx具体将请求转发到哪个服务器，在base-data-web-8010和base-data-web-8020的index.jsp加入标识

[root@localhost ROOT]# vi /app/tomcat/base-data-web-8010/webapps/ROOT/index.jsp

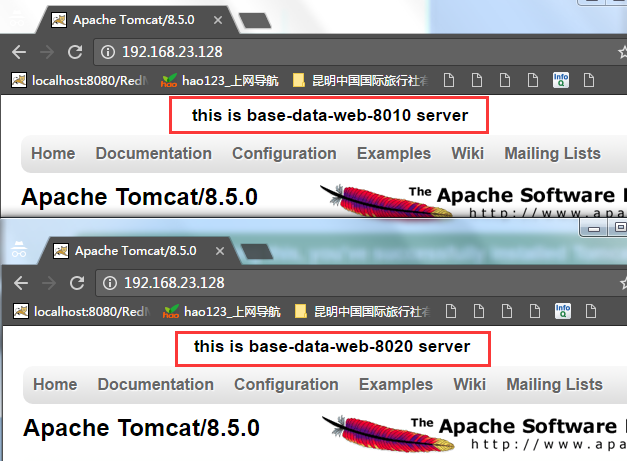
[root@localhost ROOT]# vi /app/tomcat/base-data-web-8020/webapps/ROOT/index.jsp





两次访问，发现，每次访问的tomcat服务器都不一致

（如果出现127.0.0.1可以访问，外部无法访问的情况，请关闭虚拟机防火墙，关闭方法参照附录）



# Dubbo控制台

## 获取dubbo-admin包

Dubbo官网：<http://dubbo.io/>

源码地址：<https://github.com/alibaba/dubbo>



## 安装

（见8.Tomcat参数约定）

[root@localhost tomcat]# tar -zxvf apache-tomcat-8.5.0.tar.gz



[root@localhost tomcat]# mv apache-tomcat-8.5.0 dubbo-admin-8070



[root@localhost tomcat]# ll



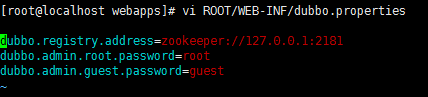
[root@localhost tomcat]# cd dubbo-admin-8070/webapps/

上传dubbo-admin-2.5.3.war到dubbo-admin-8070/webapps/目录并改名为ROOT.war

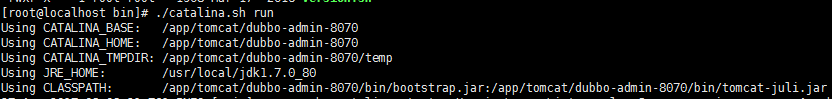
删除原来的ROOT目录



控制台的配置信息位于



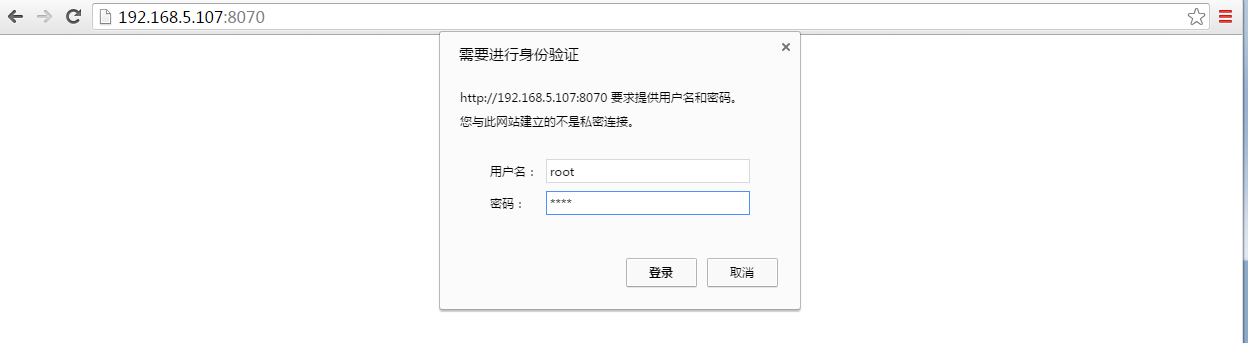
启动dubbo-admin-8070

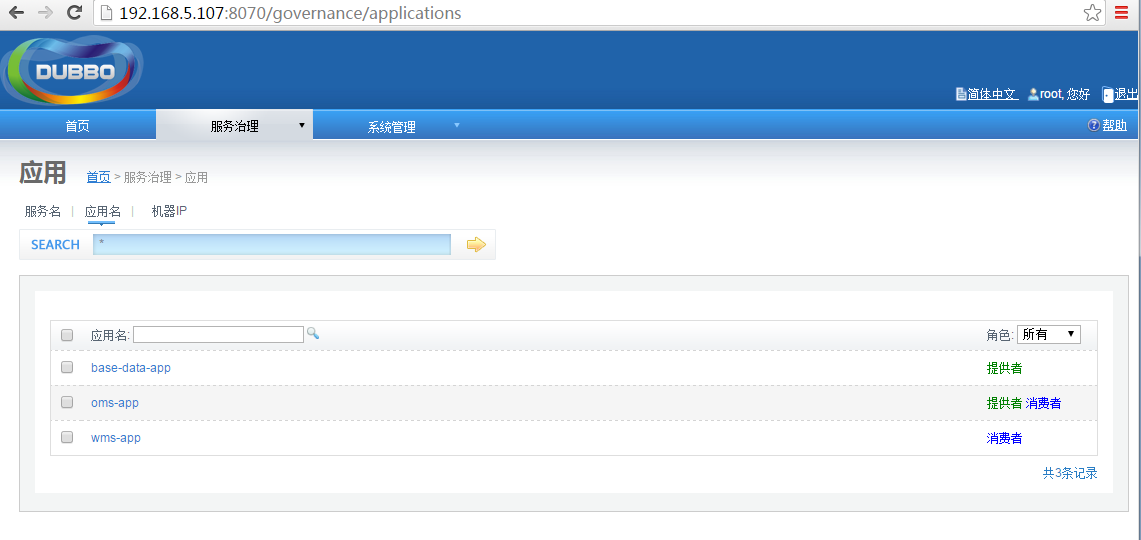


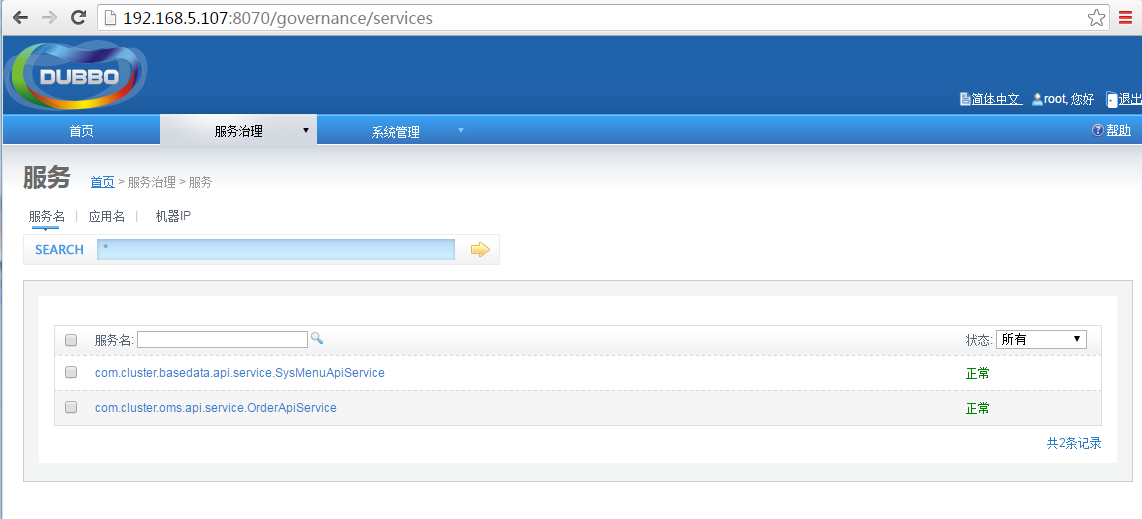
## 查看dubbo服务

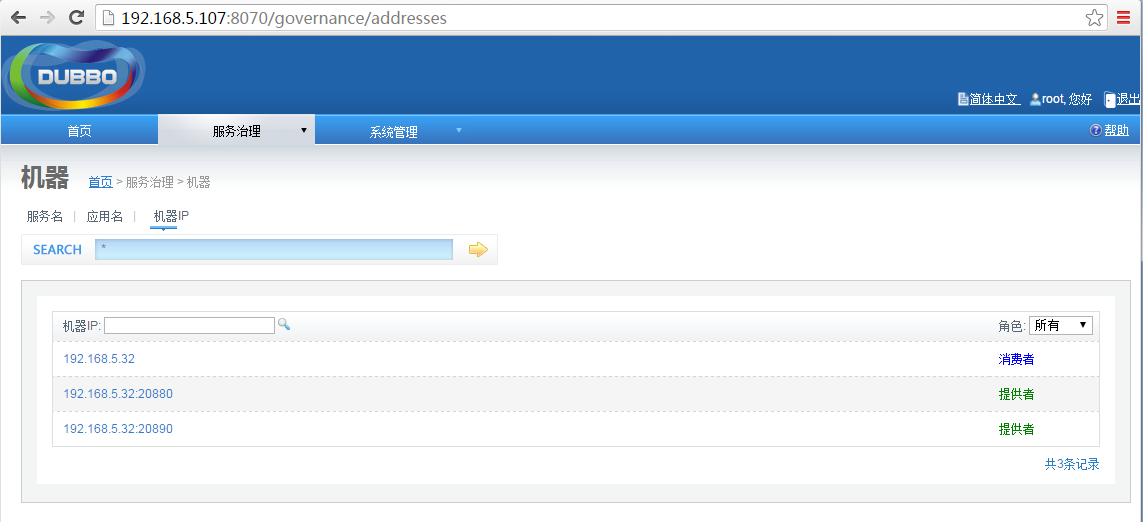
user:root

pass:root









# Mysql集群搭建

## 集群参数约定

|  |  |  |
| --- | --- | --- |
|  | **port** | **role** |
| NDB\_MGMD |  | 管理节点 |
| NDBD-1 |  | 数据节点-1 |
| NDBD-2 |  | 数据节点-2 |
| MYSQLD-8001 | 8001 | Sql节点-2 |
| MYSQLD-8002 | 8002 | Sql节点-2 |

## 查看系统默认安装版本，卸载

CentOS6.5X64默认安装了mysql服务，查看系统自带的mysql服务版本，并卸载

查看服务

[root@localhost /]# rpm -qa|grep mysql



卸载服务

[root@localhost /]# rpm -e mysql-libs --nodeps



## 搭建

### 安装mysql

[root@localhost mysql-group]# wget https://cdn.mysql.com/archives/mysql-cluster-gpl-7.4/mysql-cluster-gpl-7.4.13-linux-glibc2.5-x86\_64.tar.gz





[root@bogon mysql-group]# tar -zxvf mysql-cluster-gpl-7.4.13-linux-glibc2.5-x86\_64.tar.gz





[root@bogon mysql-group]# mv mysql-cluster-gpl-7.4.13-linux-glibc2.5-x86\_64 MYSQLD-8001



[root@bogon mysql-group]# cd MYSQLD-8001



[root@bogon MYSQLD-8001]# groupadd mysql

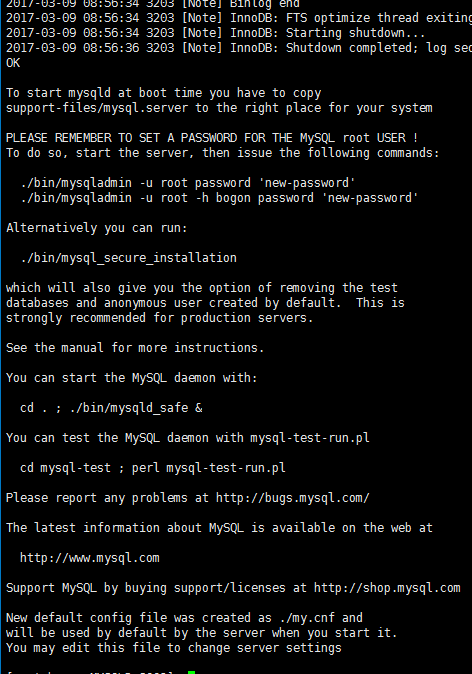


[root@bogon MYSQLD-8001]# useradd -g mysql mysql



[root@bogon MYSQLD-8001]# scripts/mysql\_install\_db --user=mysql





安装成功！



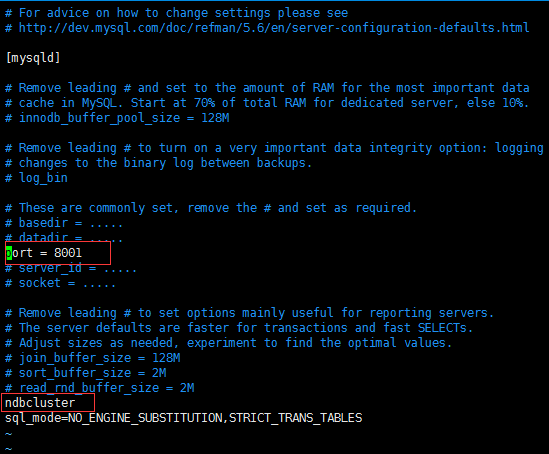
### 配置sql节点

[root@bogon MYSQLD-8001]# vi my.cnf



port = 8001 修改端口

ndbcluster 开启集群模式



[root@bogon MYSQLD-8001]# cd support-files/



[root@bogon support-files]# vi mysql.server



basedir=/app/software/mysql-group/MYSQLD-8001/

datadir=/app/software/mysql-group/MYSQLD-8001/data/



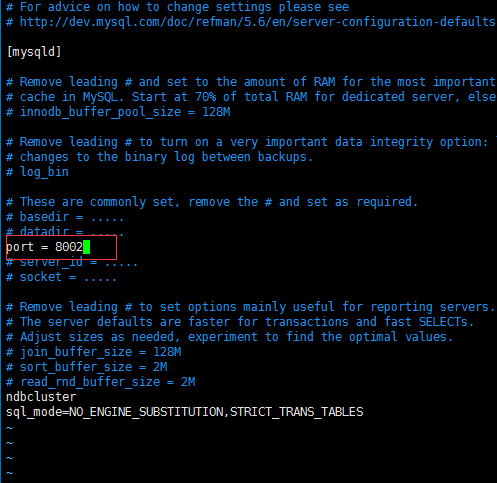
重复上面步骤 安装MYSQLD-8002

修改配置

port = 8001

basedir=/app/software/mysql-group/MYSQLD-8002/

datadir=/app/software/mysql-group/MYSQLD-8002/data/





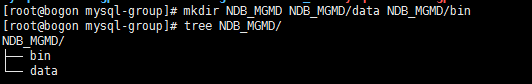
[root@bogon mysql-group]# ls MYSQLD-800\* -d



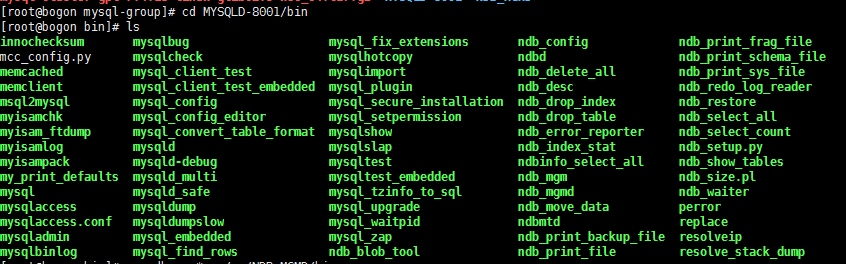
### 安装管理节点

mysql-group目录创建如下目录结构

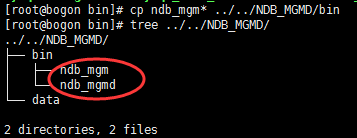
[root@bogon mysql-group]# mkdir NDB\_MGMD NDB\_MGMD/data NDB\_MGMD/bin



[root@bogon mysql-group]# cd MYSQLD-8001/bin



[root@bogon bin]# cp ndb\_mgm\* ../../NDB\_MGMD/bin



[root@bogon bin]# cd ../../NDB\_MGMD/



编写config.ini 配置

[NDBD DEFAULT]

NoOfReplicas=1

[NDB\_MGMD]

#设置管理节点服务器

HostName=127.0.0.1

DataDir=/app/software/mysql-group/NDB\_MGMD/data

[NDBD]

#设置存储节点服务器 节点1

HostName=127.0.0.1

DataDir=/app/software/mysql-group/NDBD-1

[NDBD]

#设置存储节点服务器 节点2

HostName=127.0.0.1

DataDir=/app/software/mysql-group/NDBD-2

[MYSQLD]

#设置SQL节点服务器 节点1

HostName=127.0.0.1

[MYSQLD]

#设置SQL节点服务器 节点2

HostName=127.0.0.1





### 编写启动脚本

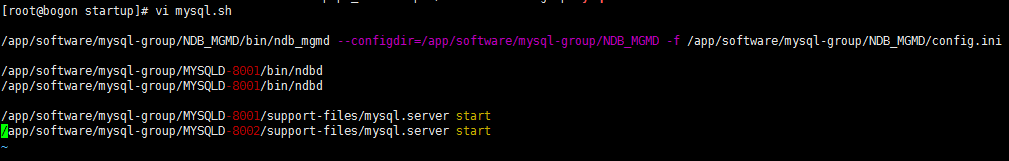
/app/software/mysql-group/NDB\_MGMD/bin/ndb\_mgmd --configdir=/app/software/mysql-group/NDB\_MGMD -f /app/software/mysql-group/NDB\_MGMD/config.ini

/app/software/mysql-group/MYSQLD-8001/bin/ndbd

/app/software/mysql-group/MYSQLD-8001/bin/ndbd

/app/software/mysql-group/MYSQLD-8001/support-files/mysql.server start

/app/software/mysql-group/MYSQLD-8002/support-files/mysql.server start



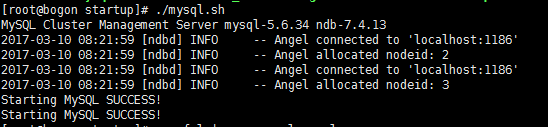


[root@bogon startup]# ./mysql.sh



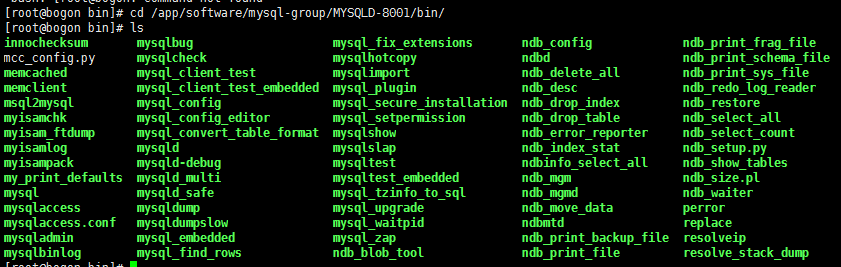
查询管理节点进程

[root@bogon startup]# ps -fel | grep mysql --color



### 登录mysql终端

[root@bogon bin]# cd /app/software/mysql-group/MYSQLD-8001/bin/

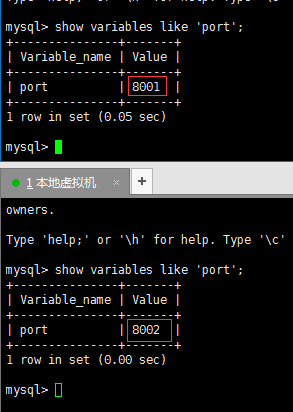


[root@bogon bin]# ./mysql -h 127.0.0.1 -P 8001 -u root

[root@bogon bin]# ./mysql -h 127.0.0.1 -P 8002 -u root

查看当前mysql端口号，检查是否登录不同的终端

mysql> show variables like 'port';

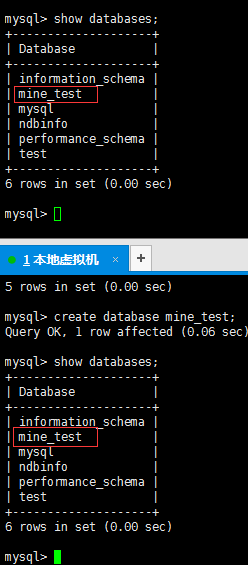


### 数据同步测试

以下命令只在一个终端执行

创建数据库 mine\_test

create database mine\_test;



创建表

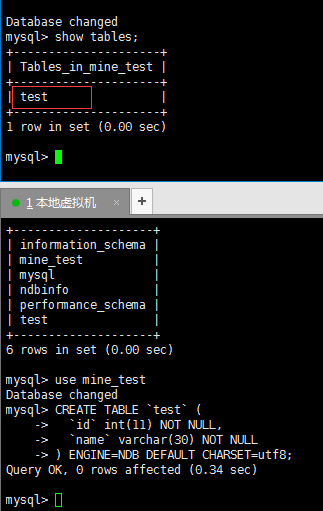
必须指定表引擎为NDB

CREATE TABLE `test` (

`id` int(11) NOT NULL,

`name` varchar(30) NOT NULL

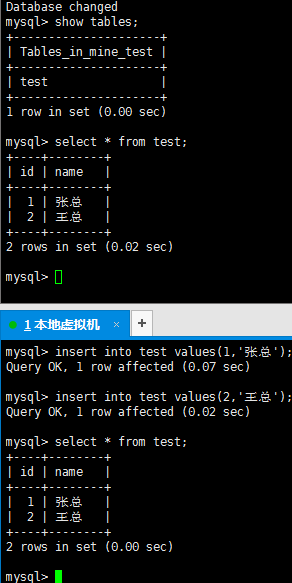
) ENGINE=NDB DEFAULT CHARSET=utf8;



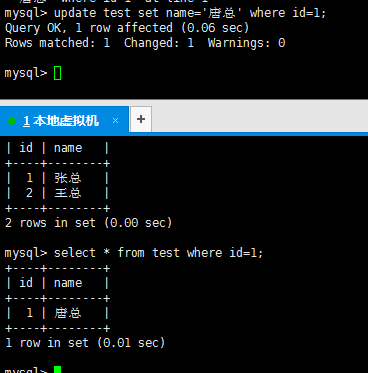
插入测试数据

insert into test values(1,'张总');

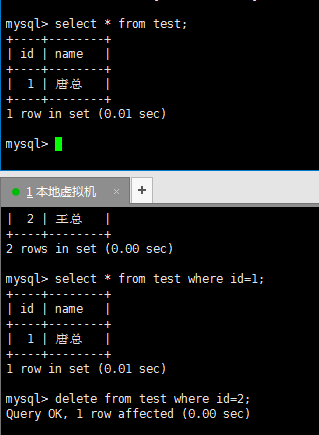
insert into test values(2,'王总');



修改测试数据

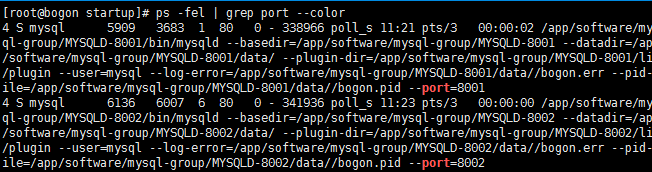


删除测试数据



### 模拟节点宕机

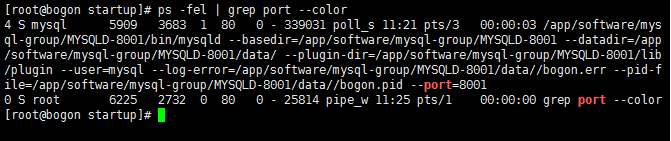
[root@bogon startup]# ps -fel | grep port --color

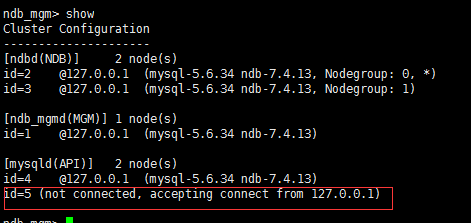


停止8002端口数据节点

[root@bogon support-files]# ./mysql.server stop







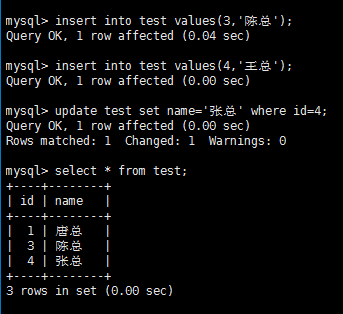
8002数据节点已关闭

### 宕机后数据同步测试

insert into test values(3,'陈总');

insert into test values(4,'王总');

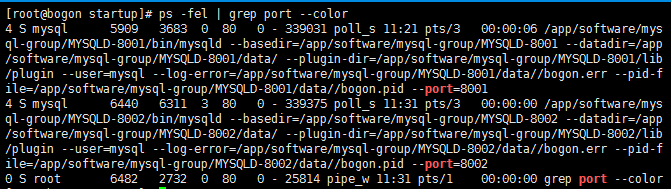
update test set name='张总' where id=4;

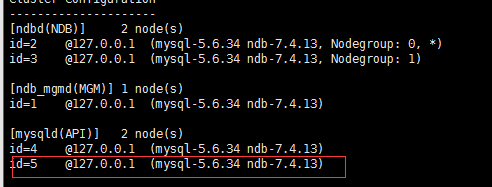


开启8002端口数据节点

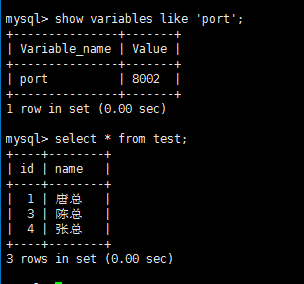
[root@bogon support-files]# ./mysql.server start







登录终端，查看数据，已经同步过来



## 配置mysql远程访问

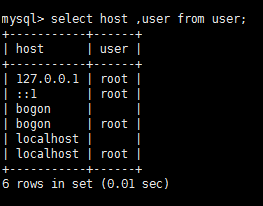


直接修改root用户用于远程访问

切换mysql数据库

mysql> use mysql;

mysql> select host ,user from user;



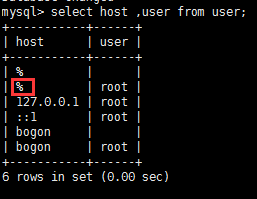
%代表任何ip都可以访问

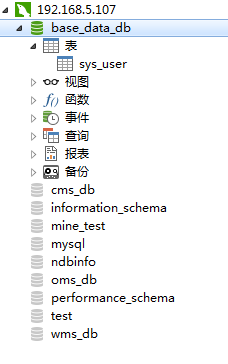
mysql> update user set host='%' where host='localhost';

刷新权限

mysql> FLUSH PRIVILEGES ;

mysql> select host ,user from user;





创建超级管理员webapp用户，用于应用连接访问

GRANT ALL PRIVILEGES ON \*.\* TO 'webapp'@'%' identified by '123456' WITH GRANT OPTION;

刷新权限生效

FLUSH PRIVILEGES;

# Cas服务器配置（基础配置）

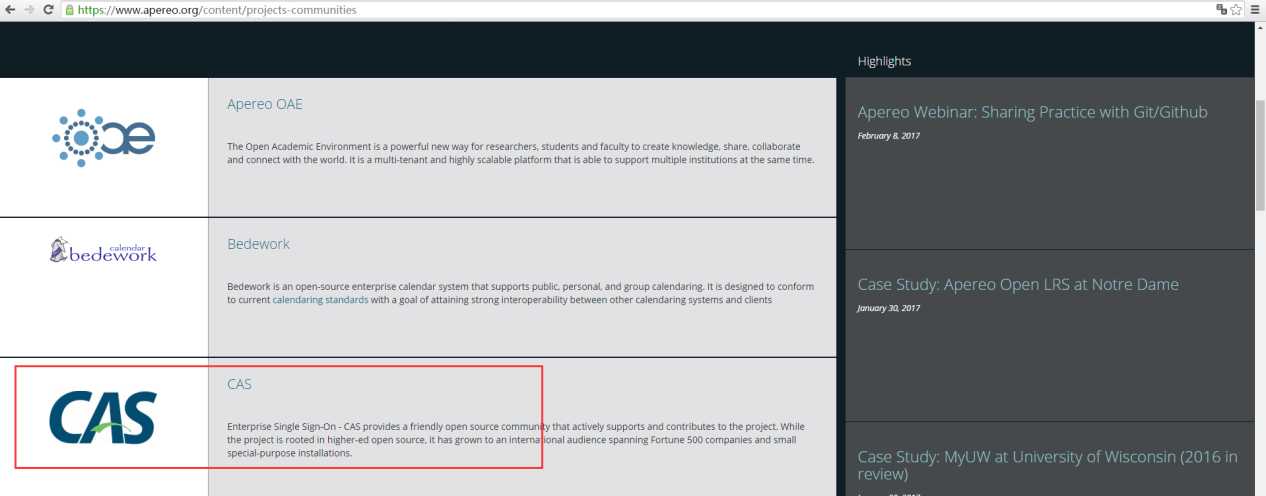
## 获取cas包（cas-server-4.0.0）

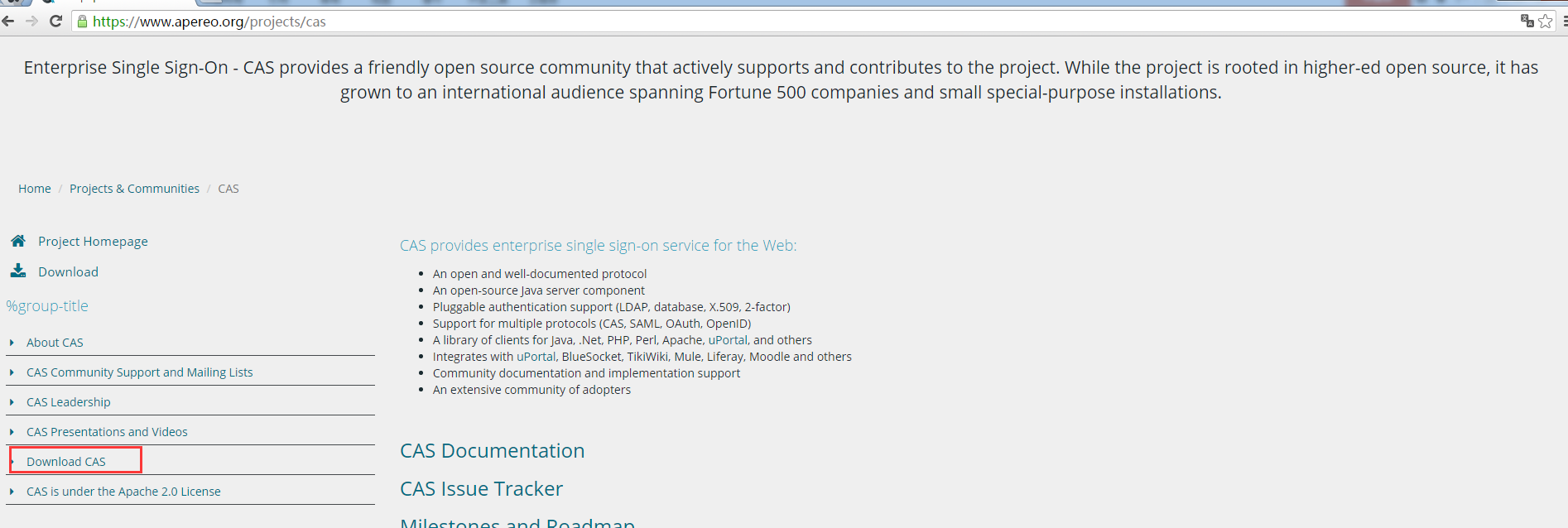
https://github-cloud.s3.amazonaws.com/releases/2352744/3dbe1b3a-8863-11e4-9274-f93d7d695c3f.zip?X-Amz-Algorithm=AWS4-HMAC-SHA256&X-Amz-Credential=AKIAISTNZFOVBIJMK3TQ%2F20170320%2Fus-east-1%2Fs3%2Faws4\_request&X-Amz-Date=20170320T022906Z&X-Amz-Expires=300&X-Amz-Signature=2e6ad530b2cf1f2378beaf0a4d692b184746ad85e81a638d89dd20ca6331d708&X-Amz-SignedHeaders=host&actor\_id=0&response-content-disposition=attachment%3B%20filename%3Dcas-server-4.0.0-release.zip&response-content-type=application%2Foctet-stream

## 从Cas官网下载cas包流程

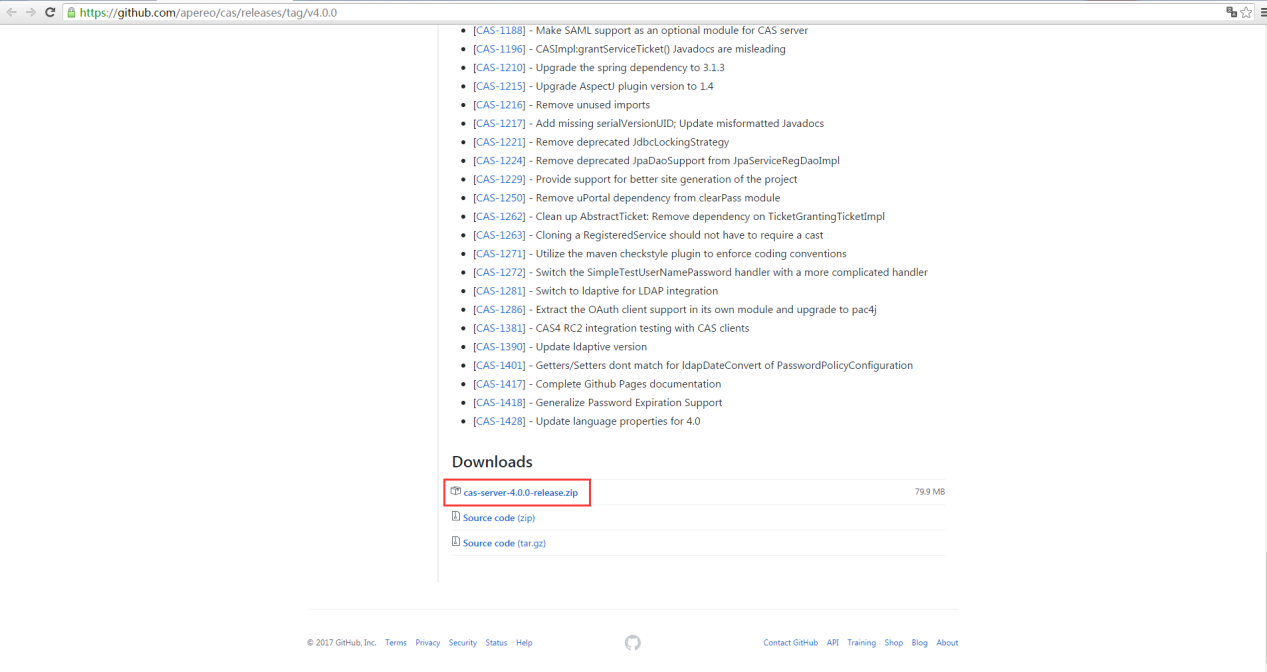
<https://www.apereo.org/>







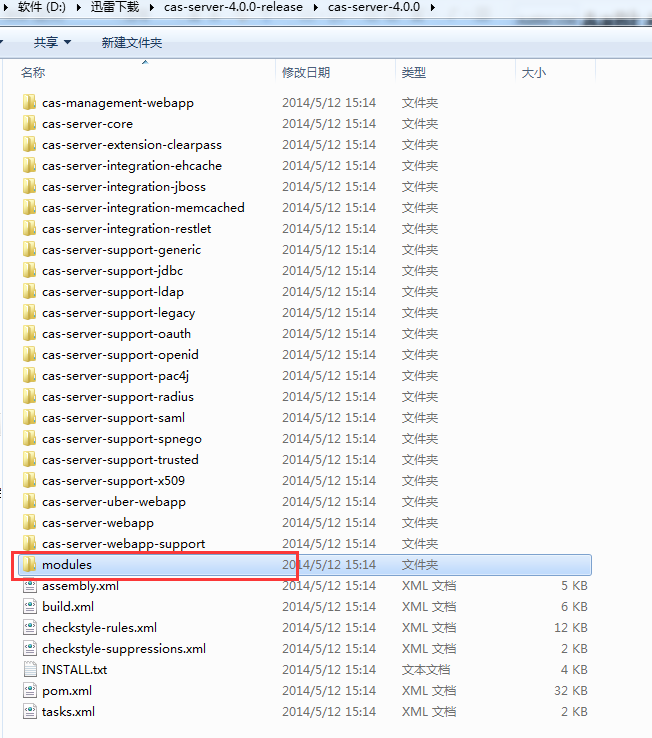


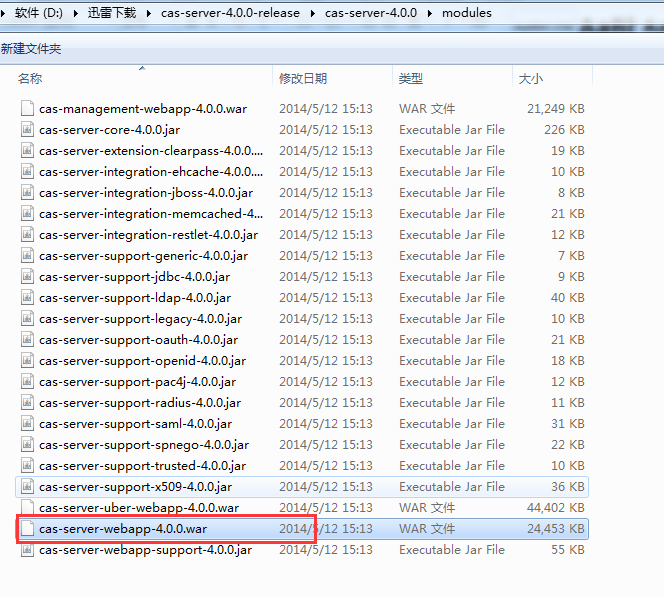


上传cas-server-webapp-4.0.0.war到cas-server-8060服务器



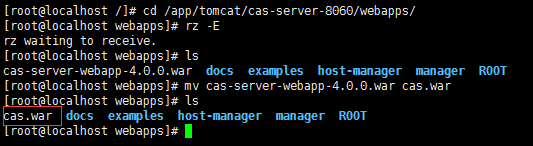
解压





[root@localhost /]# cd /app/tomcat/cas-server-8060/webapps/

[root@localhost webapps]# mv cas-server-webapp-4.0.0.war cas.war



## 配置Tomcat 服务器https连接方式

因为caas采用的是https协议，需要将tomcat默认的http连接方式改成https

[root@localhost conf]# vi /app/tomcat/cas-server-8060/conf/server.xml



<Connector

protocol="org.apache.coyote.http11.Http11NioProtocol"

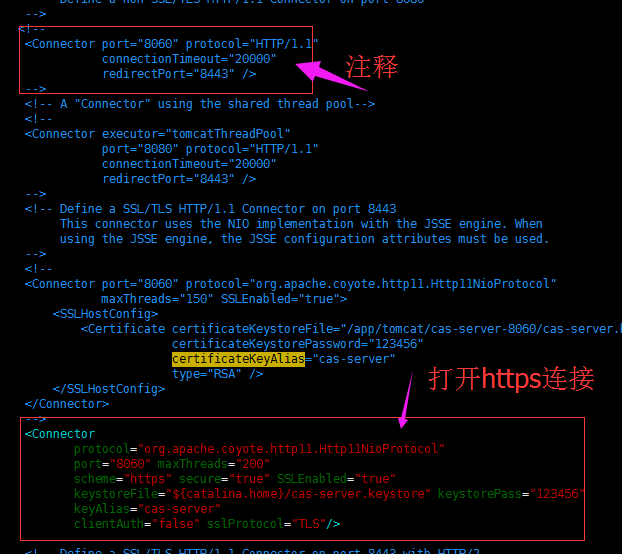
port="8060" maxThreads="200"

scheme="https" secure="true" SSLEnabled="true"

keystoreFile="${catalina.home}/cas-server.keystore" keystorePass="123456"

keyAlias="cas-server"

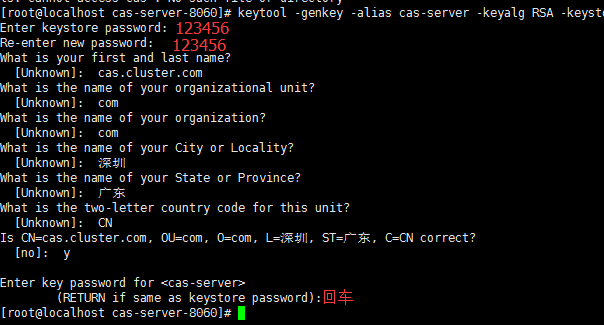
clientAuth="false" sslProtocol="TLS"/>



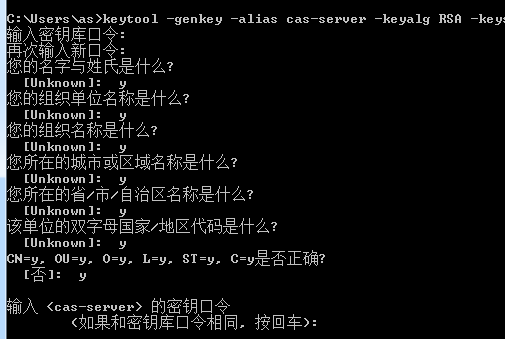
## 使用jdk的[keytool工具生成https证书](http://blog.csdn.net/huangxinyu_it/article/details/41693633)

[root@localhost cas-server-8060]# keytool -genkey -alias cas-server -keyalg RSA -keystore /app/tomcat/cas-server-8060/cas-server.keystore

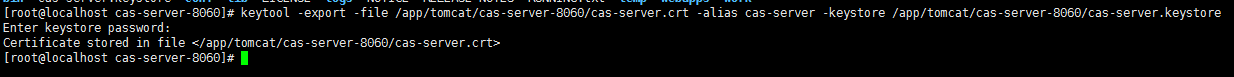




Window下，中文版



[root@localhost cas-server-8060]# keytool -export -file /app/tomcat/cas-server-8060/cas-server.crt -alias cas-server -keystore /app/tomcat/cas-server-8060/cas-server.keystore





导入证书到jvm

**（注：密码为changeit**

**这步的作用还不明确，不论证书导没导入jvm都会出现连接不是私密连接）**



[root@localhost security]# keytool -import -keystore /usr/local/jdk1.7.0\_80/jre/lib/security/cacerts -file /app/tomcat/cas-server-8060/cas-server.crt -alias cas-server



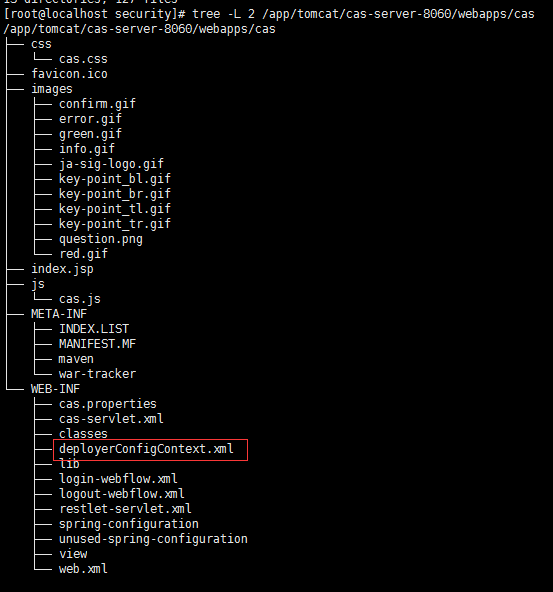
删除证书

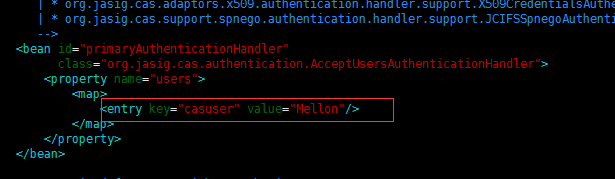
[root@localhost security]# keytool -delete -alias cas-server -keystore /usr/local/jdk1.7.0\_80/jre/lib/security/cacerts -storepass changeit

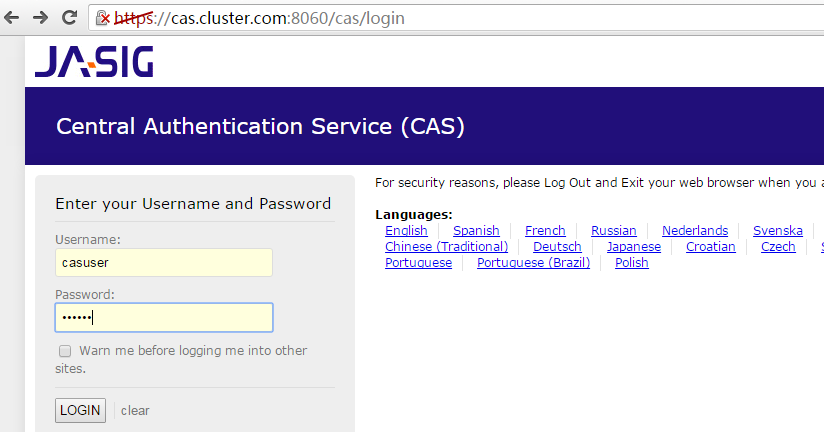


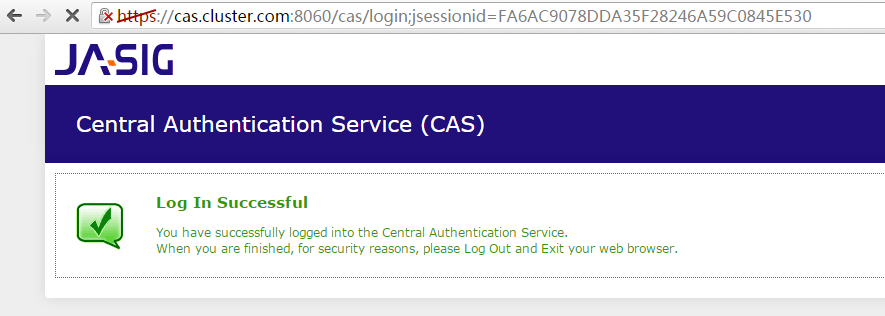
## 访问cas-server（注意配置host）

查看Cas默认的用户名和密码

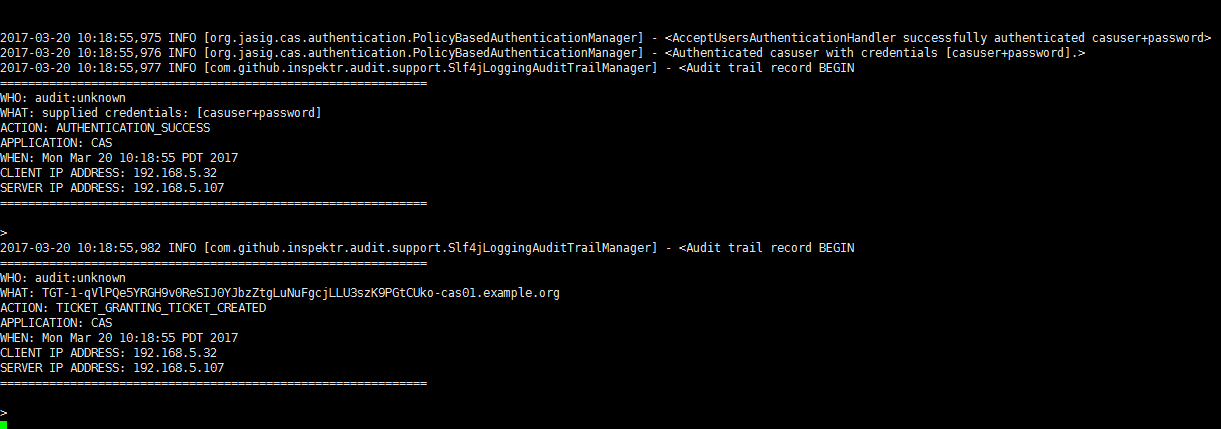








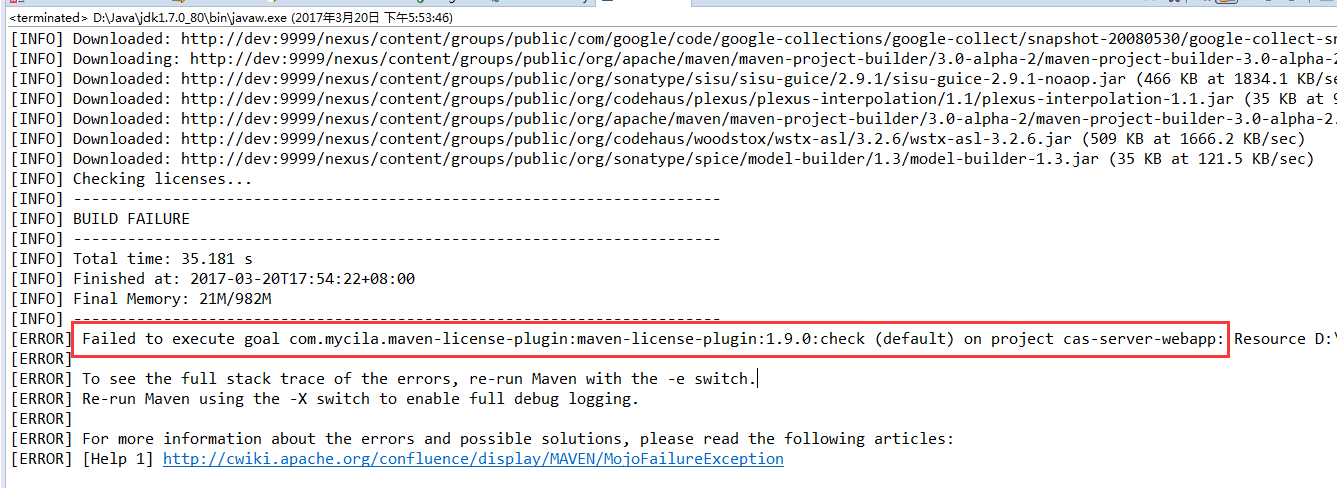
后台日志输出



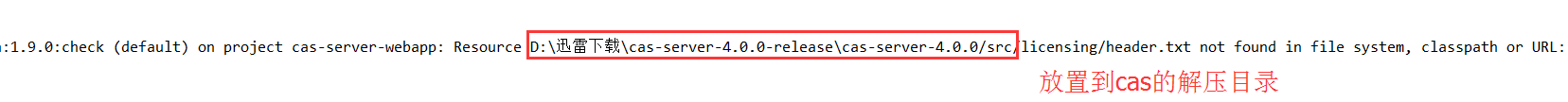
## 配置cas-server为数据库验证方式

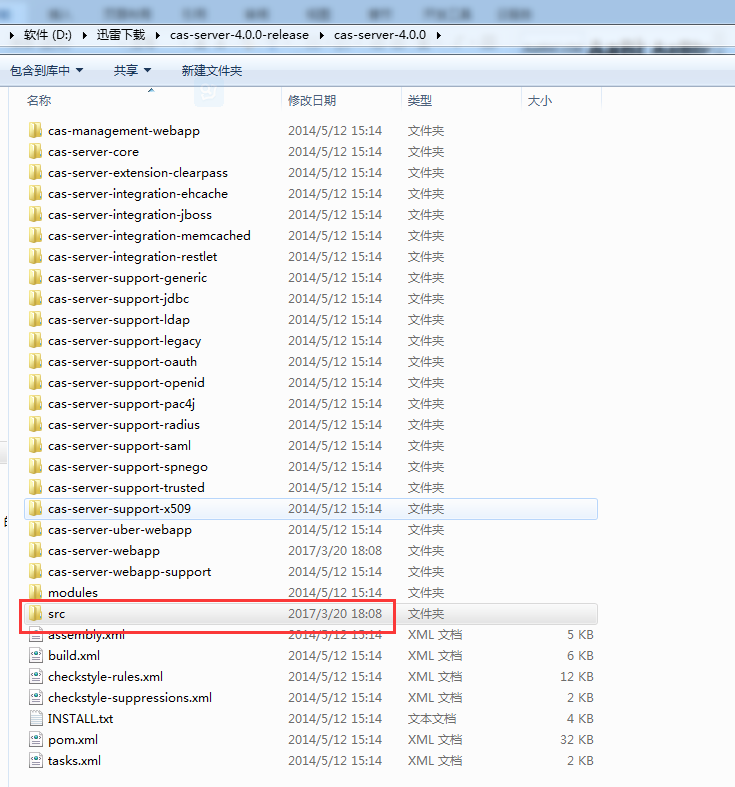
导入到eclipse

cas-server-4.0.0编译报错,缺少文件



添加到cas解压目录





添加依赖cas-support-jdbc和mysql驱动依赖

<dependency>

<groupId>org.jasig.cas</groupId>

<artifactId>cas-server-support-jdbc</artifactId>

<version>4.0.0</version>

</dependency>

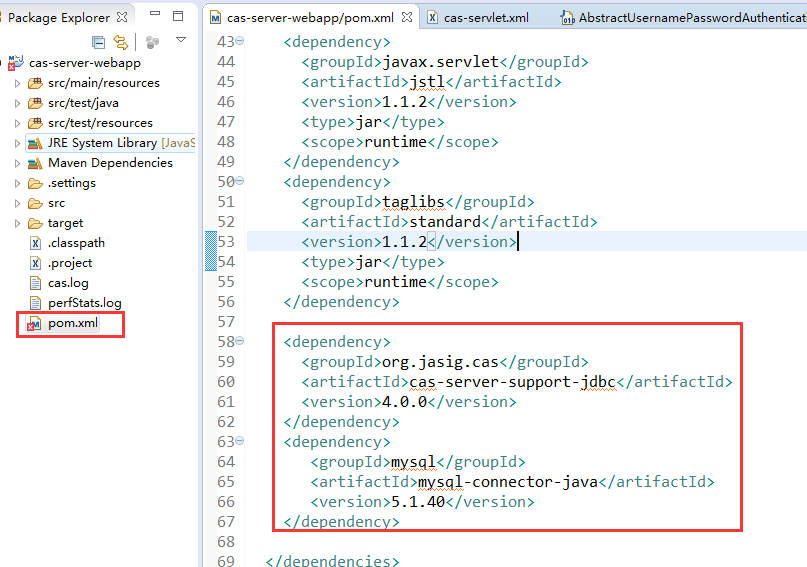
<dependency>

<groupId>mysql</groupId>

<artifactId>mysql-connector-java</artifactId>

<version>5.1.40</version>

</dependency>



修改deployerConfigContext.xml文件

添加dataSource，这里采用的是spring-jdbc模块（默认项目已经存在spring-jdbc包，使用其他连接方式需要添加依赖包），也可以配置其他连接方式，如druid连接池，c3p0等

<bean id=*"dataSource"* class=*"org.springframework.jdbc.datasource.DriverManagerDataSource"*>

<property name=*"driverClassName"* value=*"com.mysql.jdbc.Driver"*></property>

<property name=*"url"* value=*"jdbc:mysql://192.168.5.107:8002,192.168.5.107:8001/base\_data\_db"*></property>

<property name=*"username"* value=*"root"*></property>

<property name=*"password"* value=*""*></property>

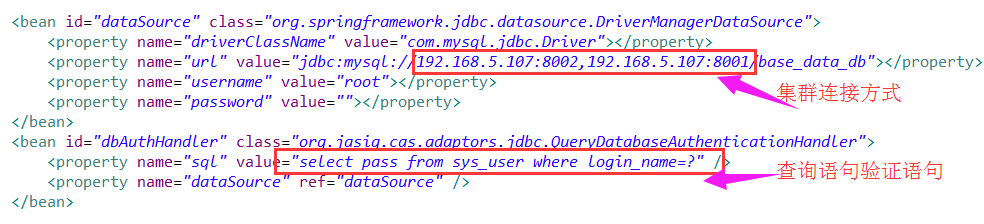
</bean>

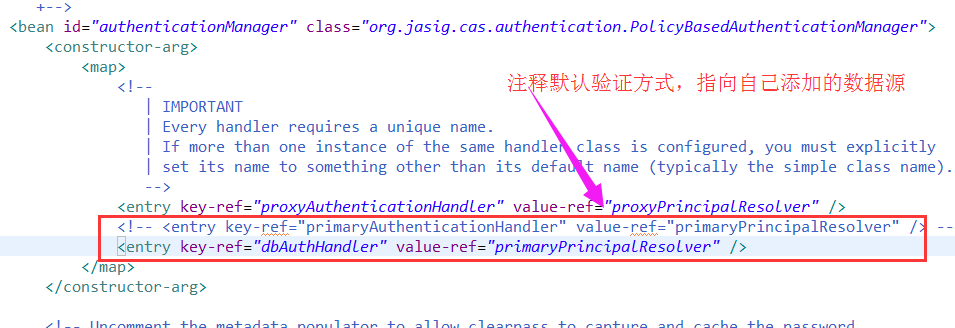
<bean id=*"dbAuthHandler"* class=*"org.jasig.cas.adaptors.jdbc.QueryDatabaseAuthenticationHandler"*>

<property name=*"sql"* value=*"select pass from sys\_user where login\_name=?"* />

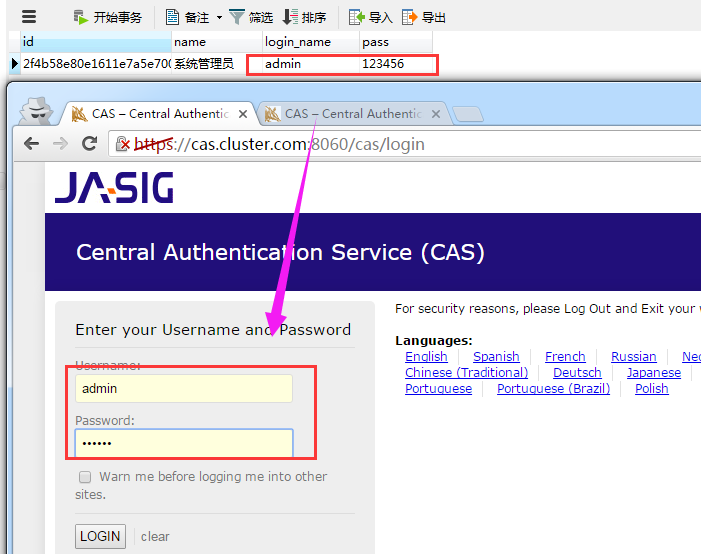
<property name=*"dataSource"* ref=*"dataSource"* />

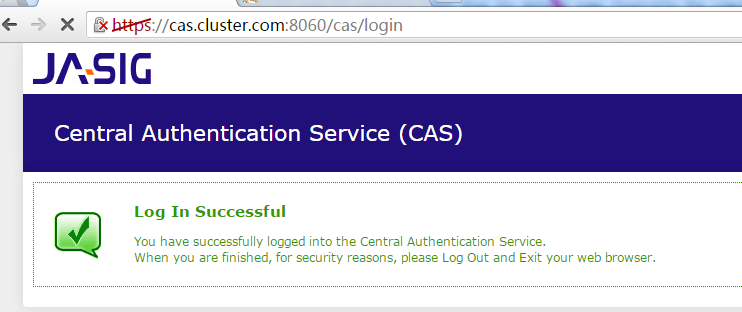
</bean>





编译，发包，验证





## 配置cas-server为数据库验证方式（自定义登录方式）

# 项目模型搭建

## 配置cas-server服务器

修改cas-server-webapp项目

修改deployerConfigContext.xml文件，增加配置

<bean id=*"attributeRepository"* class=*"org.jasig.services.persondir.support.jdbc.SingleRowJdbcPersonAttributeDao"*>

<constructor-arg index=*"0"* ref=*"dataSource"*/>

<constructor-arg index=*"1"* value=*"select id,name from sys\_user where {0}"*/>

<property name=*"queryAttributeMapping"*>

<map>

<!--这里的key需写username和登录页面一致，value对应数据库用户名字段-->

<entry key=*"username"* value=*"login\_name"*/>

</map>

</property>

<property name=*"resultAttributeMapping"*>

<map>

<!--key为对应的数据库字段名称，value为提供给客户端获取的属性名字，系统会自动填充值-->

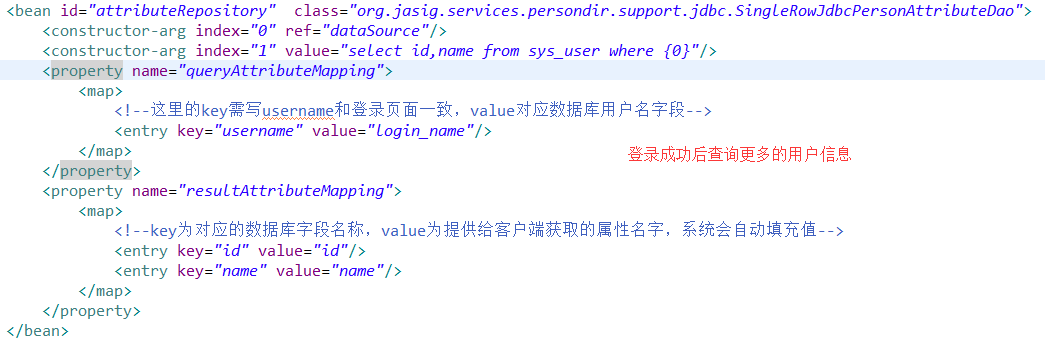
<entry key=*"id"* value=*"id"*/>

<entry key=*"name"* value=*"name"*/>

</map>

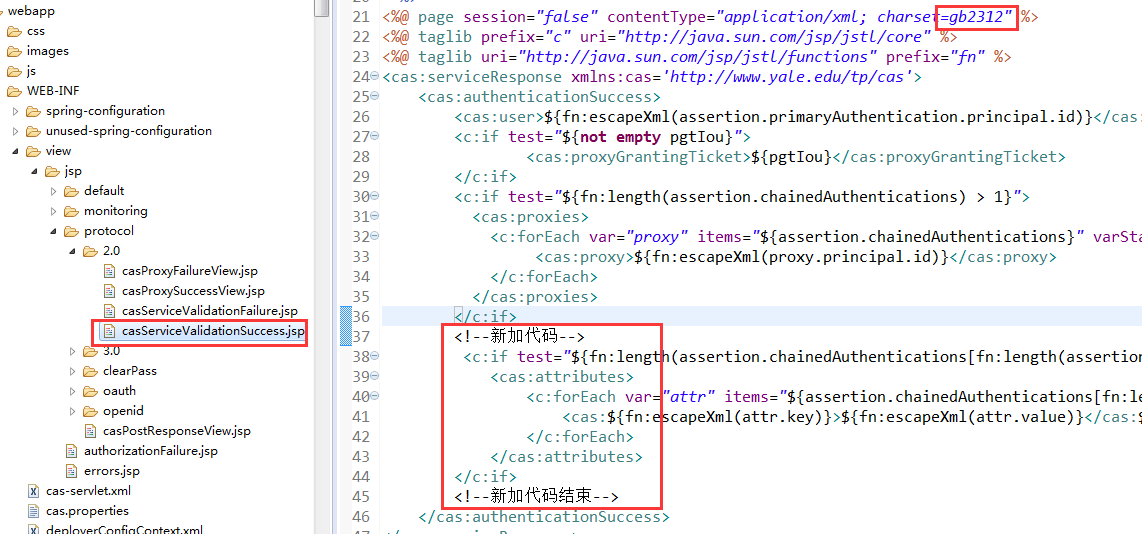
</property>

</bean>





修改casServiceValidationSuccess.jsp页面，添加代码，更多的用户信息，渲染给应用的用户数据

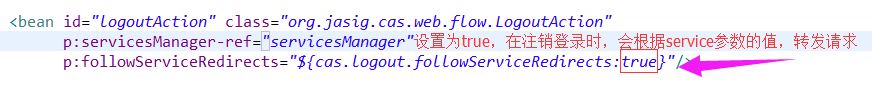


修改cas-servlet.xml

<bean id=*"logoutAction"* class=*"org.jasig.cas.web.flow.LogoutAction"*

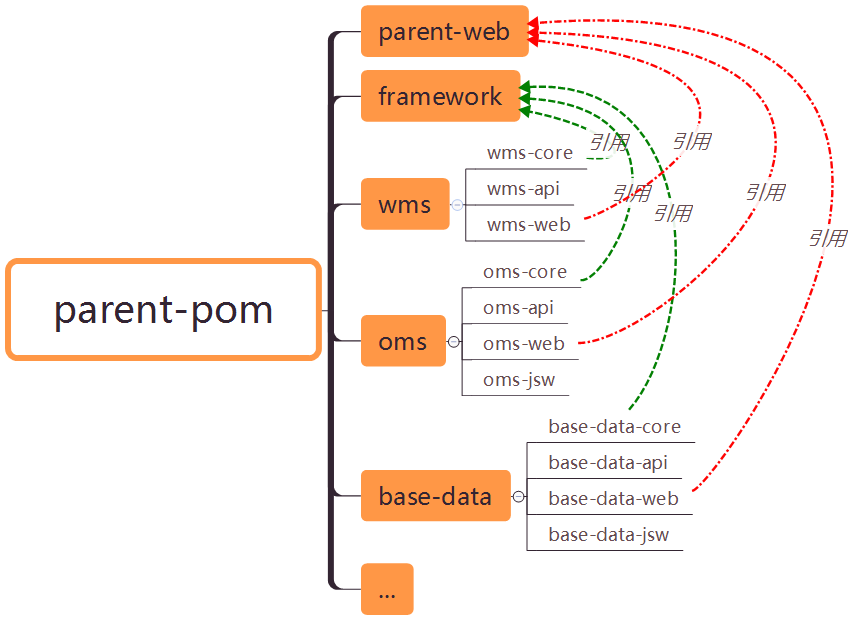
p:servicesManager-ref=*"servicesManager"*

p:followServiceRedirects=*"${cas.logout.followServiceRedirects:true}"*/>



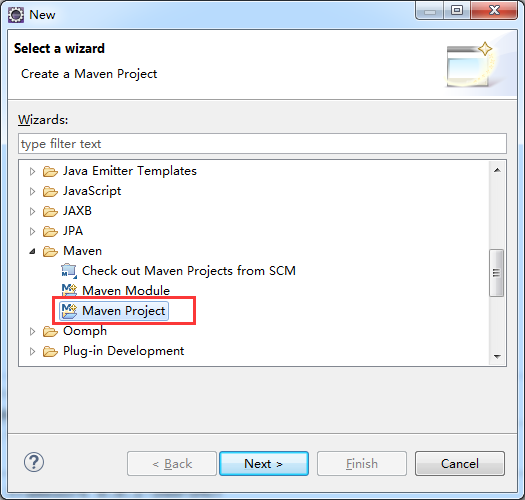
编译，部署

## 项目结构图

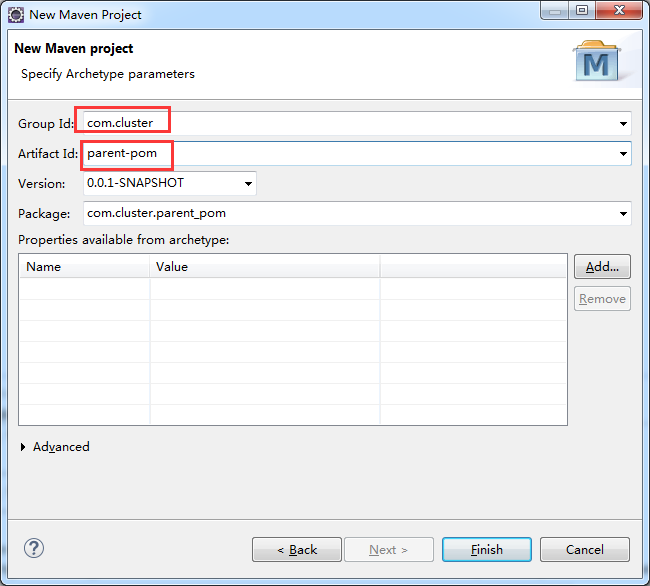


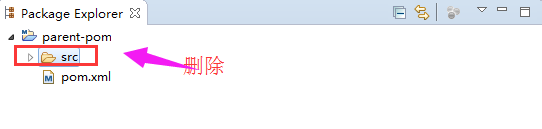
## 创建parent-pom父类项目

选择创建Maven Project项目









添加包版本

<properties>

<project.build.sourceEncoding>UTF-8</project.build.sourceEncoding>

<spring.version>4.1.4.RELEASE</spring.version>

<cas.client.core.version>3.2.1-RELEASE</cas.client.core.version>

</properties>

添加公共管理依赖包

<dependencyManagement>

<dependencies>

<!-- spring mvc -->

<dependency>

<groupId>org.springframework</groupId>

<artifactId>spring-webmvc</artifactId>

<version>${spring.version}</version>

</dependency>

<dependency>

<groupId>org.springframework</groupId>

<artifactId>spring-web</artifactId>

<version>${spring.version}</version>

</dependency>

<!-- CAS Client -->

<dependency>

<groupId>org.jasig.cas.client</groupId>

<artifactId>cas-client-core</artifactId>

<version>${cas.client.core.version}</version>

</dependency>

</dependencies>

</dependencyManagement>

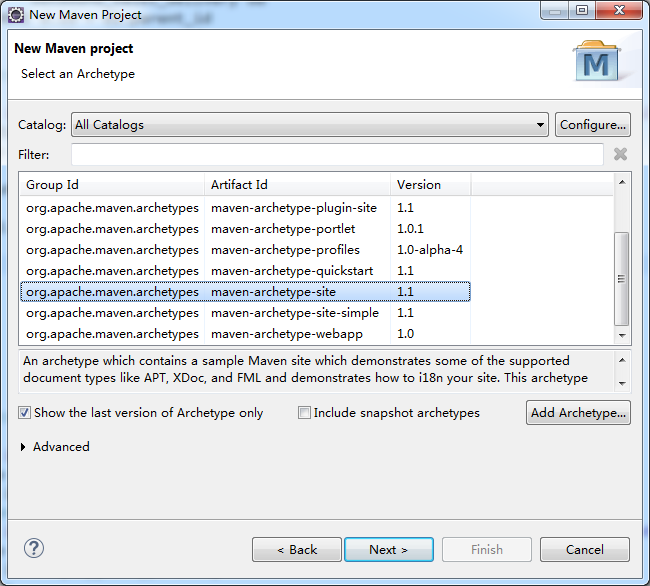
### 作用描述

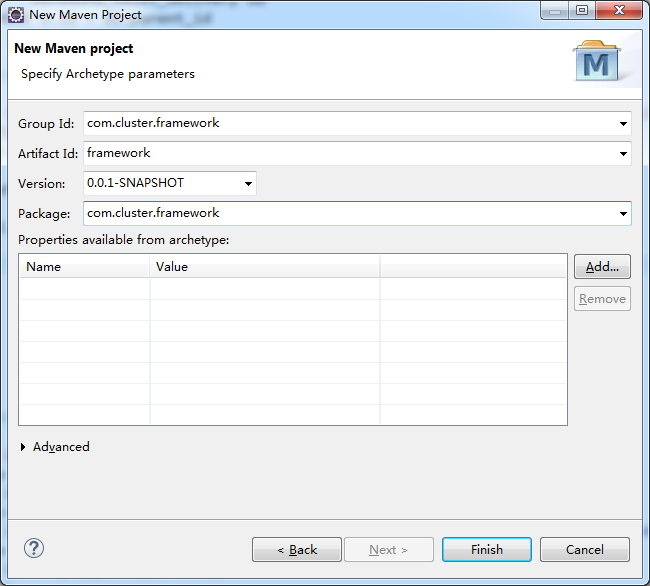
所有项目的最终父类

该子项目提供平台基础jar包的版本管理

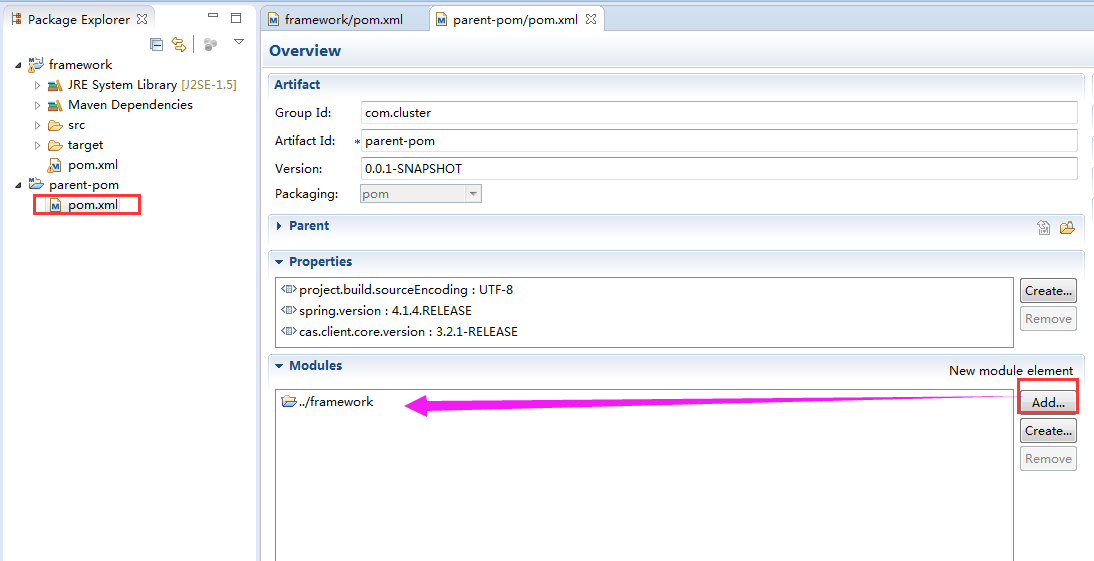
## 创建framework项目

选择创建Maven Project项目

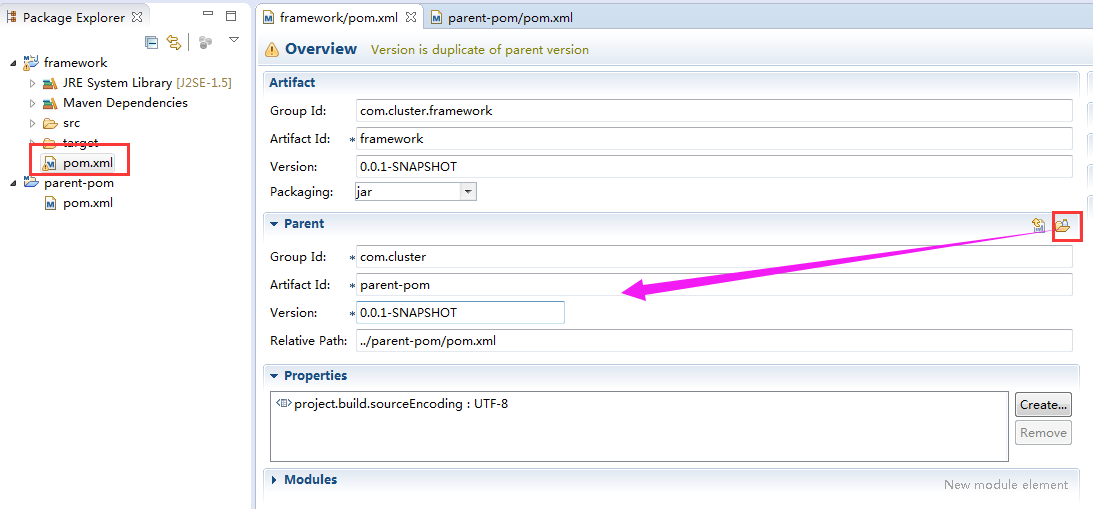




修改parent-pom项目的pom.xml文件，添加framework模块



修改framework项目的pom.xml文件，添加parent-pom依赖



修改framework项目的pom.xml文件

添加基础的包依赖

<dependencies>

<!-- spring mvc -->

<dependency>

<groupId>org.springframework</groupId>

<artifactId>spring-webmvc</artifactId>

</dependency>

<dependency>

<groupId>org.springframework</groupId>

<artifactId>spring-web</artifactId>

</dependency>

<!-- CAS Client -->

<dependency>

<groupId>org.jasig.cas.client</groupId>

<artifactId>cas-client-core</artifactId>

</dependency>

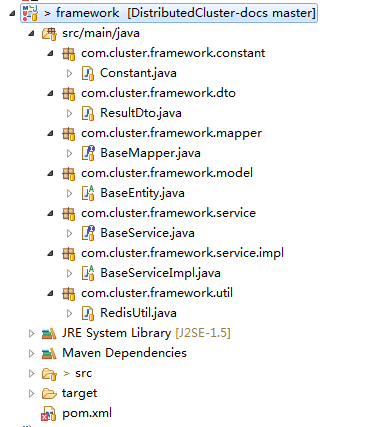
</dependencies>

### 作用描述

控制平台的总体service层资源，

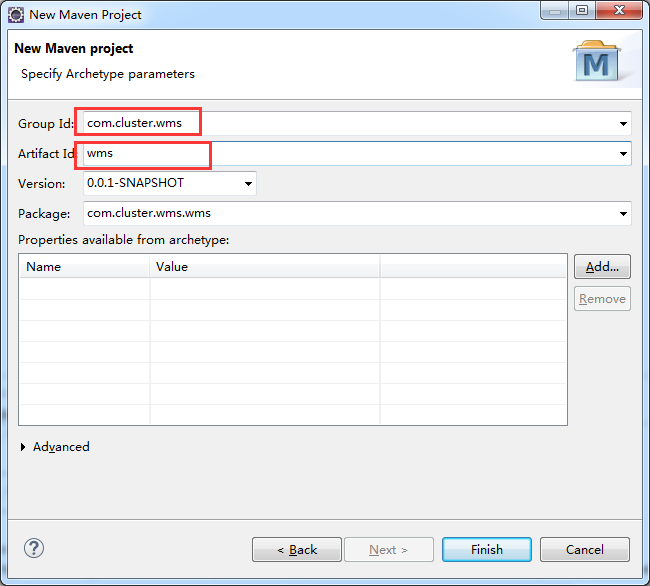
所有的XXX-core子项目将会引用framework项目

该子项目定义了基础的service实现，dao实现，entity实现，工具类，常量信息

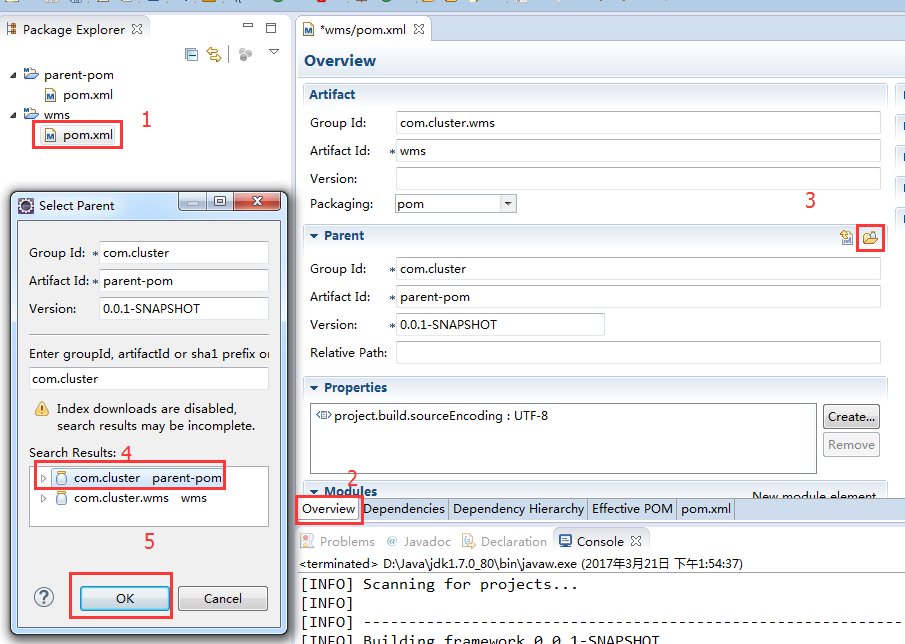


## 创建wms父类项目

选择创建Maven Project项目

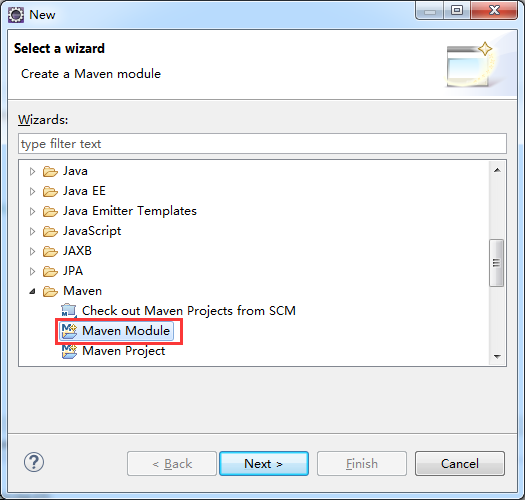


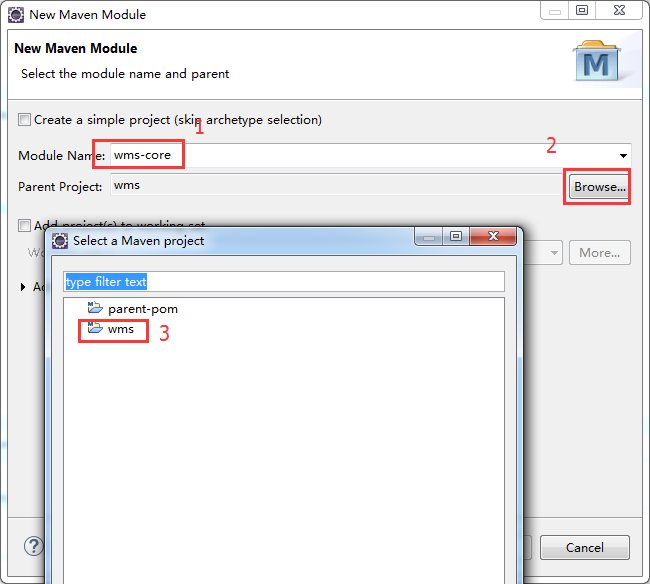
修改wms项目的pom.xml文件，添加parent-pom依赖

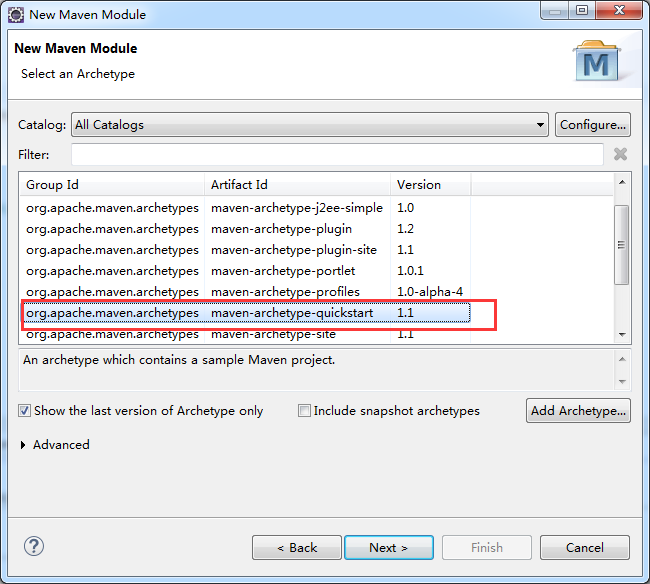


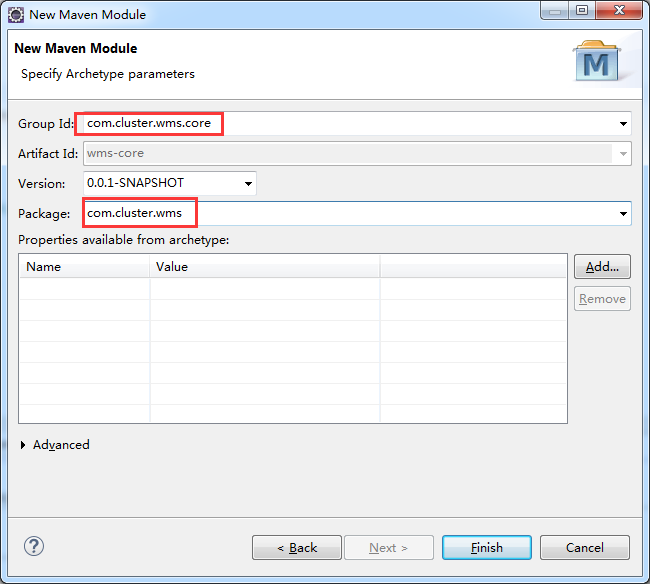
## 创建wms-core项目

选择创建Maven Module项目









添加framework依赖

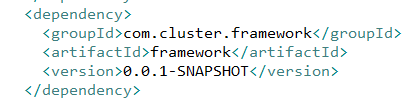
<dependency>

<groupId>com.cluster.framework</groupId>

<artifactId>framework</artifactId>

<version>0.0.1-SNAPSHOT</version>

</dependency>



### 作用描述

service层的实现

## 创建wms-api项目

选择创建Maven Module项目

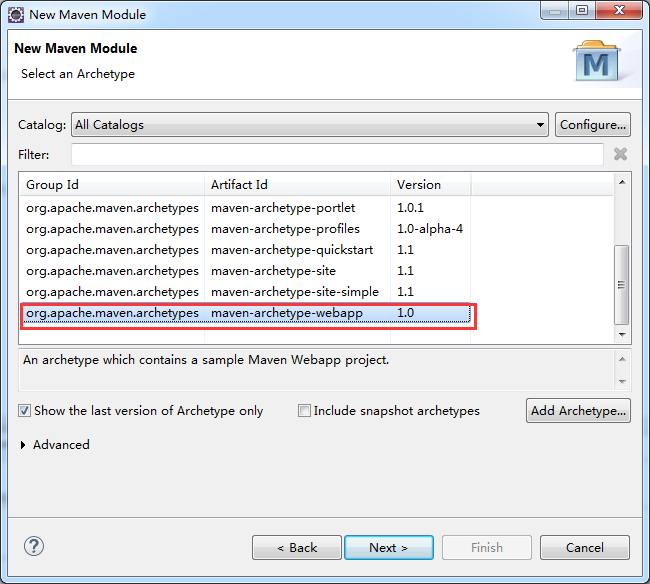
以同样的方式创建wms-api项目

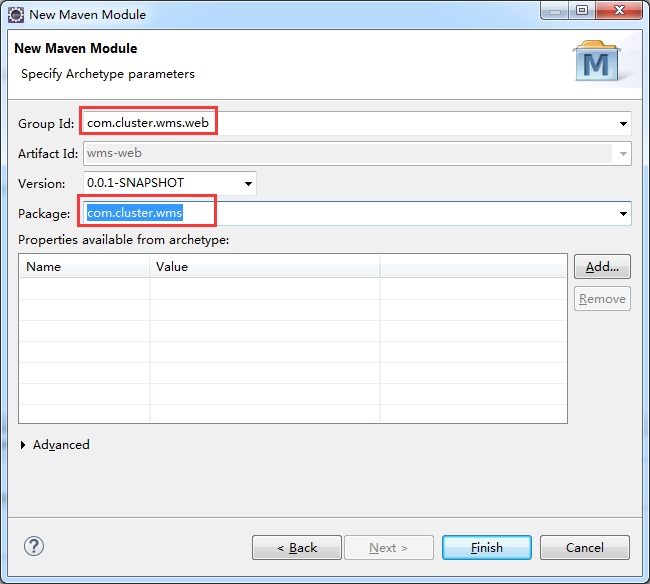
### 作用描述

对外发布

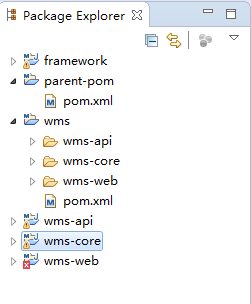
## 创建wms-web项目

选择创建Maven Module项目





最终结构

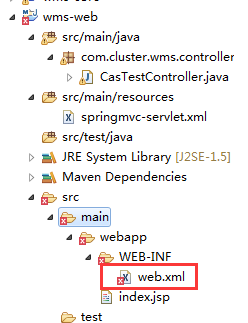


添加servlet-api依赖

### 整合spring mvc

* + - 1. 配置spring mvc

修改web.xml



<!-- Spring MVC -->

<servlet>

<servlet-name>springmvc</servlet-name>

<servlet-class>org.springframework.web.servlet.DispatcherServlet</servlet-class>

<init-param>

<param-name>contextConfigLocation</param-name>

<param-value>classpath:springmvc-servlet.xml</param-value>

</init-param>

<load-on-startup>1</load-on-startup>

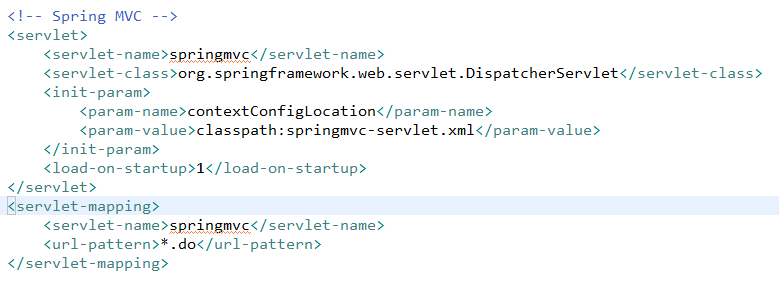
</servlet>

<servlet-mapping>

<servlet-name>springmvc</servlet-name>

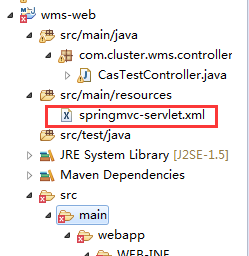
<url-pattern>\*.do</url-pattern>

</servlet-mapping>



#### 添加springmvc-servlet.xml

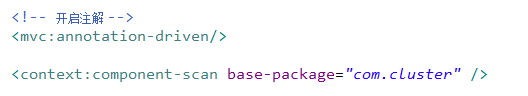
添加配置



<!-- 开启注解 -->

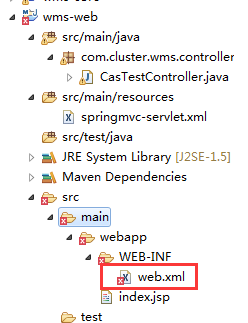
<mvc:annotation-driven/>

<context:component-scan base-package=*"com.cluster"* />



### 配置cas-client

修改web.xml



<!-- ======================== 单点登录结束 ======================== -->

<!-- 用于单点退出，该过滤器用于实现单点登出功能，可选配置 -->

<listener>

<listener-class>org.jasig.cas.client.session.SingleSignOutHttpSessionListener</listener-class>

</listener>

<!-- 该过滤器用于实现单点登出功能，可选配置。 -->

<filter>

<filter-name>CAS Single Sign Out Filter</filter-name>

<filter-class>org.jasig.cas.client.session.SingleSignOutFilter</filter-class>

</filter>

<filter-mapping>

<filter-name>CAS Single Sign Out Filter</filter-name>

<url-pattern>/\*</url-pattern>

</filter-mapping>

<!-- 该过滤器负责用户的认证工作，必须启用它 -->

<filter>

<filter-name>CASFilter</filter-name>

<filter-class>org.jasig.cas.client.authentication.AuthenticationFilter</filter-class>

<init-param>

<param-name>casServerLoginUrl</param-name>

<param-value>https://cas.cluster.com:8060/cas</param-value><!-- cas 服务器登录 地址 http://IP:PORT/CasWebProName/login -->

</init-param>

<init-param>

<!-- 这里的server是服务端的IP -->

<param-name>serverName</param-name>

<param-value>http://127.0.0.1:8080/wms-web/test.do</param-value><!-- 客户端服务器地址 http://IP:PORT -->

</init-param>

</filter>

<filter-mapping>

<filter-name>CASFilter</filter-name>

<url-pattern>/\*</url-pattern>

</filter-mapping>

<!-- 该过滤器负责对Ticket的校验工作，必须启用它 -->

<filter>

<filter-name>CAS Validation Filter</filter-name>

<filter-class>org.jasig.cas.client.validation.Cas20ProxyReceivingTicketValidationFilter</filter-class>

<init-param>

<param-name>casServerUrlPrefix</param-name>

<param-value>https://cas.cluster.com:8060/cas</param-value><!-- cas 服务器地址 http://IP:PORT/CasWebProName -->

</init-param>

<init-param>

<param-name>serverName</param-name>

<param-value>http://127.0.0.1:8080/wms-web/test.do</param-value><!-- 客户端服务器地址 http://IP:PORT -->

</init-param>

</filter>

<filter-mapping>

<filter-name>CAS Validation Filter</filter-name>

<url-pattern>/\*</url-pattern>

</filter-mapping>

<!-- 该过滤器负责实现HttpServletRequest请求的包裹， 比如允许开发者通过HttpServletRequest的getRemoteUser()方法获得SSO登录用户的登录名，可选配置。 -->

<filter>

<filter-name>CAS HttpServletRequest Wrapper Filter</filter-name>

<filter-class>org.jasig.cas.client.util.HttpServletRequestWrapperFilter</filter-class>

</filter>

<filter-mapping>

<filter-name>CAS HttpServletRequest Wrapper Filter</filter-name>

<url-pattern>/\*</url-pattern>

</filter-mapping>

<!-- 该过滤器使得开发者可以通过org.jasig.cas.client.util.AssertionHolder来获取用户的登录名。 比如AssertionHolder.getAssertion().getPrincipal().getName()。 -->

<filter>

<filter-name>CAS Assertion Thread Local Filter</filter-name>

<filter-class>org.jasig.cas.client.util.AssertionThreadLocalFilter</filter-class>

</filter>

<filter-mapping>

<filter-name>CAS Assertion Thread Local Filter</filter-name>

<url-pattern>/\*</url-pattern>

</filter-mapping>

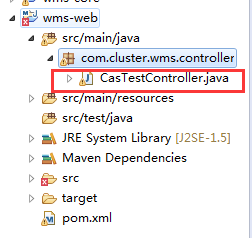
<!-- ======================== 单点登录结束 ======================== -->







### 创建CasTestController测试类



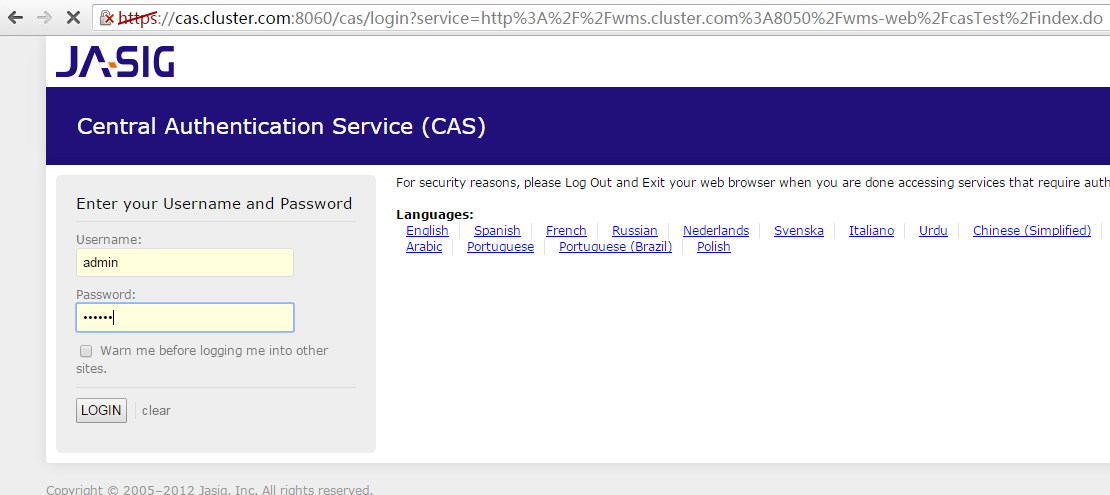


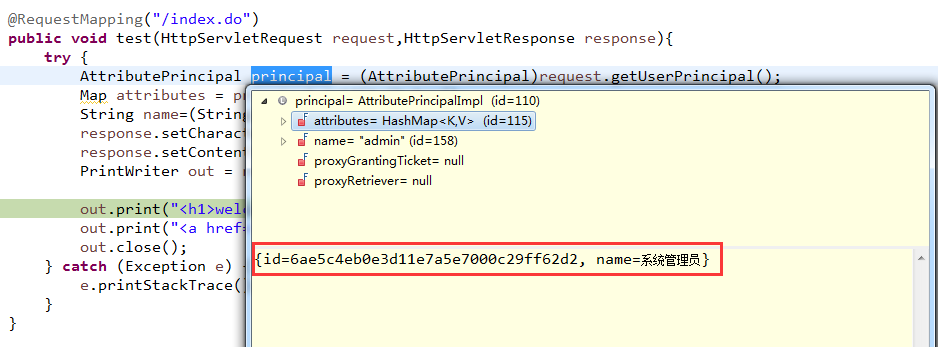


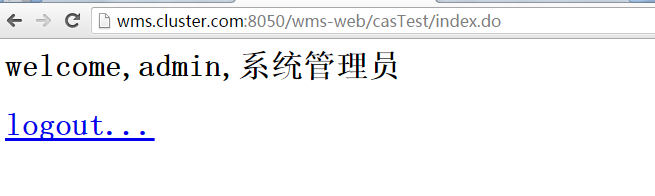
* + - 1. Debug

访问

http://wms.cluster.com:8050/wms-web/casTest/index.do







### 编写自己的cas单点登录页面

### 异常

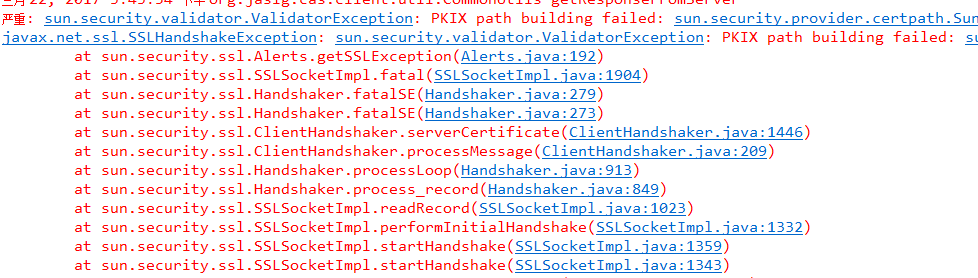
* + - 1. java.security.cert.CertificateException:异常



如何解决：



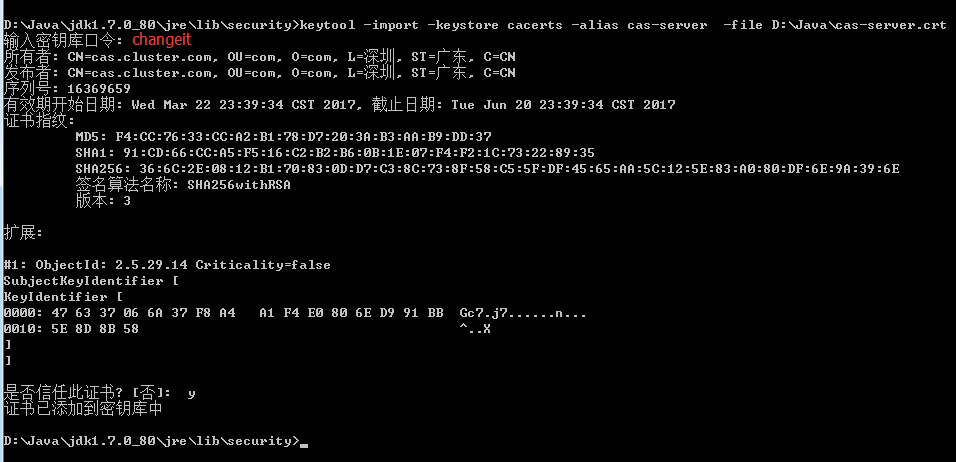
* + - 1. sun.security.validator.ValidatorException:异常



如何解决：

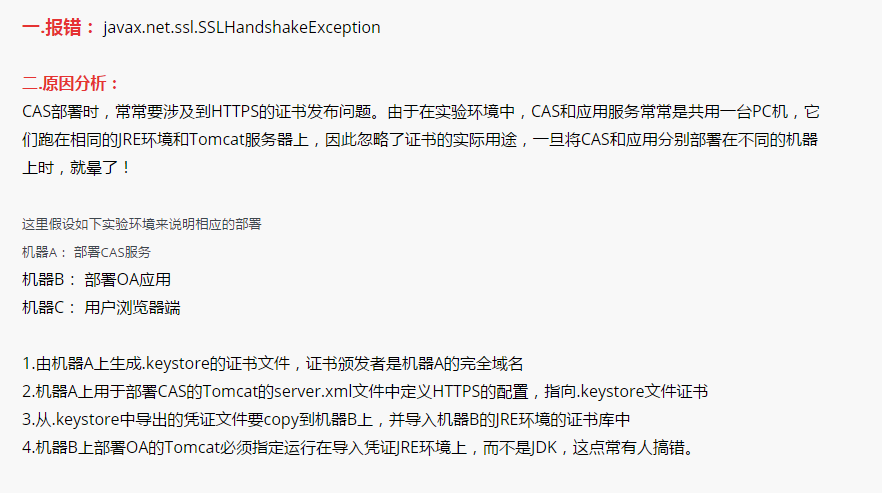
需要将cas-server服务器中生成的证书导入到应用部署的服务器

（注：进去jdk目录导入，否则可能出现IO错误）



详见：

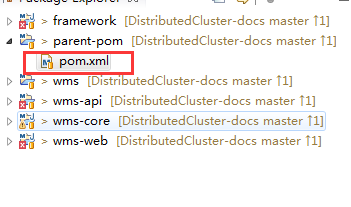
<https://my.oschina.net/laiwanshan/blog/159057>



## 整合mybatis

### 添加依赖

修改parent-pom项目的pom.xml文件



添加依赖mybatis相关依赖

<mybatis.version>3.2.8</mybatis.version>

<mybatis.spring.version>1.2.2</mybatis.spring.version>

<!-- mybatis -->

<dependency>

<groupId>org.mybatis</groupId>

<artifactId>mybatis</artifactId>

<version>${mybatis.version}</version>

</dependency>

<dependency>

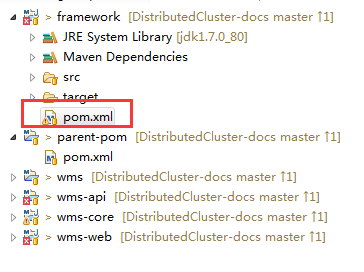
<groupId>org.mybatis</groupId>

<artifactId>mybatis-spring</artifactId>

<version>${mybatis.spring.version}</version>

</dependency>

修改framework项目的pom.xml文件



引用依赖

<!-- mybatis -->

<dependency>

<groupId>org.mybatis</groupId>

<artifactId>mybatis</artifactId>

</dependency>

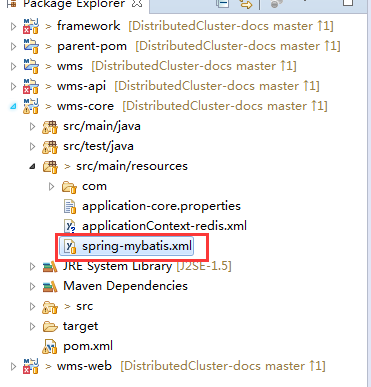
<dependency>

<groupId>org.mybatis</groupId>

<artifactId>mybatis-spring</artifactId>

</dependency>

### 添加spring-mybatis.xml配置文件



<?xml version=*"1.0"* encoding=*"UTF-8"*?>

<beans xmlns=*"http://www.springframework.org/schema/beans"*

xmlns:xsi=*"http://www.w3.org/2001/XMLSchema-instance"* xmlns:p=*"http://www.springframework.org/schema/p"*

xmlns:context=*"http://www.springframework.org/schema/context"*

xmlns:mvc=*"http://www.springframework.org/schema/mvc"*

xsi:schemaLocation=*"http://www.springframework.org/schema/beans*

*http://www.springframework.org/schema/beans/spring-beans-3.1.xsd*

*http://www.springframework.org/schema/context*

*http://www.springframework.org/schema/context/spring-context-3.1.xsd*

*http://www.springframework.org/schema/mvc*

*http://www.springframework.org/schema/mvc/spring-mvc-4.0.xsd"*>

<!-- 自动扫描 -->

<context:component-scan base-package=*"com.cluster"* />

<!-- 引入配置文件 -->

<bean id=*"propertyConfigurer"*

class=*"org.springframework.beans.factory.config.PropertyPlaceholderConfigurer"*>

<property name=*"location"* value=*"classpath:application-core.properties"* />

</bean>

<!-- 阿里 druid数据库连接池 -->

<bean id=*"dataSource"* class=*"com.alibaba.druid.pool.DruidDataSource"* destroy-method=*"close"*>

<!-- 数据库基本信息配置 -->

<property name=*"url"* value=*"${url}"* />

<property name=*"username"* value=*"${username}"* />

<property name=*"password"* value=*"${password}"* />

<property name=*"driverClassName"* value=*"${driverClassName}"* />

<property name=*"filters"* value=*"${filters}"* />

<!-- 最大并发连接数 -->

<property name=*"maxActive"* value=*"${maxActive}"* />

<!-- 初始化连接数量 -->

<property name=*"initialSize"* value=*"${initialSize}"* />

<!-- 配置获取连接等待超时的时间 -->

<property name=*"maxWait"* value=*"${maxWait}"* />

<!-- 最小空闲连接数 -->

<property name=*"minIdle"* value=*"${minIdle}"* />

<!-- 配置间隔多久才进行一次检测，检测需要关闭的空闲连接，单位是毫秒 -->

<property name=*"timeBetweenEvictionRunsMillis"* value=*"${timeBetweenEvictionRunsMillis}"* />

<!-- 配置一个连接在池中最小生存的时间，单位是毫秒 -->

<property name=*"minEvictableIdleTimeMillis"* value=*"${minEvictableIdleTimeMillis}"* />

<property name=*"validationQuery"* value=*"${validationQuery}"* />

<property name=*"testWhileIdle"* value=*"${testWhileIdle}"* />

<property name=*"testOnBorrow"* value=*"${testOnBorrow}"* />

<property name=*"testOnReturn"* value=*"${testOnReturn}"* />

<property name=*"maxOpenPreparedStatements"* value=*"${maxOpenPreparedStatements}"* />

<!-- 打开removeAbandoned功能 -->

<property name=*"removeAbandoned"* value=*"${removeAbandoned}"* />

<!-- 1800秒，也就是30分钟 -->

<property name=*"removeAbandonedTimeout"* value=*"${removeAbandonedTimeout}"* />

<!-- 关闭abanded连接时输出错误日志 -->

<property name=*"logAbandoned"* value=*"${logAbandoned}"* />

</bean>

<!-- spring和MyBatis完美整合，不需要mybatis的配置映射文件 -->

<bean id=*"sqlSessionFactory"* class=*"org.mybatis.spring.SqlSessionFactoryBean"*>

<property name=*"dataSource"* ref=*"dataSource"* />

<!-- 自动扫描mapping.xml文件 -->

<property name=*"mapperLocations"* value=*"classpath:com/cluster/wms/mapper/\*.xml"*></property>

</bean>

<!-- DAO接口所在包名，Spring会自动查找其下的类 -->

<bean class=*"org.mybatis.spring.mapper.MapperScannerConfigurer"*>

<property name=*"basePackage"* value=*"com.cluster.wms.mapper"* />

<property name=*"sqlSessionFactoryBeanName"* value=*"sqlSessionFactory"*></property>

</bean>

<!-- (事务管理)transaction manager, use JtaTransactionManager for global tx -->

<bean id=*"transactionManager"*

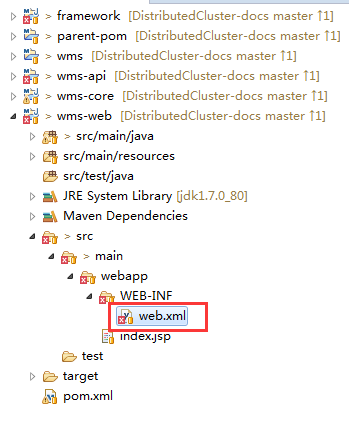
class=*"org.springframework.jdbc.datasource.DataSourceTransactionManager"*>

<property name=*"dataSource"* ref=*"dataSource"* />

</bean>

</beans>

### 配置web.xml



添加全局contextConfigLocation参数

<context-param>

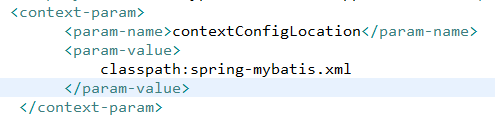
<param-name>contextConfigLocation</param-name>

<param-value>

classpath:spring-mybatis.xml

</param-value>

</context-param>



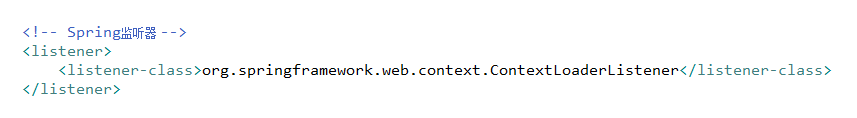
添加spring上下文监听器

<!-- Spring监听器 -->

<listener>

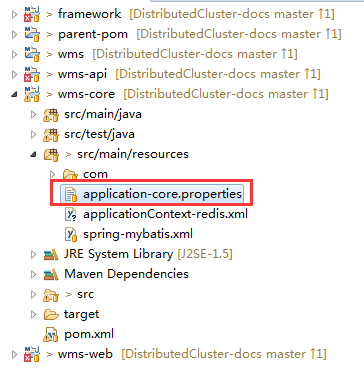
<listener-class>org.springframework.web.context.ContextLoaderListener</listener-class>

</listener>



### 添加数据库配置

添加application-core.properties文件



添加数据库配置

# db

url:jdbc:mysql://192.168.5.107:8002,192.168.5.107:8001/wms\_db

driverClassName:com.mysql.jdbc.Driver

username:webapp

password:123456

filters:stat

maxActive:20

initialSize:1

maxWait:60000

minIdle:10

maxIdle:15

timeBetweenEvictionRunsMillis:60000

minEvictableIdleTimeMillis:300000

validationQuery:SELECT 'x'

testWhileIdle:true

testOnBorrow:false

testOnReturn:false

maxOpenPreparedStatements:20

removeAbandoned:true

removeAbandonedTimeout:1800

logAbandoned:true

## 配置redis集群连接

添加applicationContext-redis.xml配置文件

<?xml version=*"1.0"* encoding=*"UTF-8"*?>

<beans xmlns=*"http://www.springframework.org/schema/beans"*

xmlns:xsi=*"http://www.w3.org/2001/XMLSchema-instance"* xmlns:p=*"http://www.springframework.org/schema/p"*

xmlns:c=*"http://www.springframework.org/schema/c"* xmlns:context=*"http://www.springframework.org/schema/context"*

xmlns:aop=*"http://www.springframework.org/schema/aop"* xmlns:util=*"http://www.springframework.org/schema/util"*

xmlns:redis=*"http://www.springframework.org/schema/redis"*

xsi:schemaLocation=*"*

*http://www.springframework.org/schema/beans http://www.springframework.org/schema/beans/spring-beans-4.1.xsd*

*http://www.springframework.org/schema/context http://www.springframework.org/schema/context/spring-context-4.1.xsd*

*http://www.springframework.org/schema/aop http://www.springframework.org/schema/aop/spring-aop-4.1.xsd*

*http://www.springframework.org/schema/util http://www.springframework.org/schema/util/spring-util-4.1.xsd*

*http://www.springframework.org/schema/redis http://www.springframework.org/schema/redis/spring-redis-1.0.xsd"*>

<!-- #################################################################### -->

<!-- # Jedis 连接池配置 -->

<!-- #################################################################### -->

<bean id=*"jedisPoolConfig"* class=*"redis.clients.jedis.JedisPoolConfig"*>

<property name=*"minIdle"*>

<value>${redis.pool.minIdle}</value>

</property>

<property name=*"maxIdle"*>

<value>${redis.pool.maxIdle}</value>

</property>

<property name=*"maxWaitMillis"*>

<value>${redis.pool.maxWaitMillis}</value>

</property>

<property name=*"testOnBorrow"*>

<value>${redis.pool.testOnBorrow}</value>

</property>

<property name=*"testOnReturn"*>

<value>${redis.pool.testOnReturn}</value>

</property>

<property name=*"testWhileIdle"*>

<value>${redis.pool.testWhileIdle}</value>

</property>

</bean>

<!-- #################################################################### -->

<!-- # Redis集群节点配置 -->

<!-- #################################################################### -->

<bean id=*"redisClusterConfig"* class=*"org.springframework.data.redis.connection.RedisClusterConfiguration"*>

<property name=*"maxRedirects"* value=*"3"*></property>

<property name=*"clusterNodes"*>

<set>

<bean class=*"org.springframework.data.redis.connection.RedisNode"*>

<constructor-arg name=*"host"* value=*"${redis.node1.host}"*></constructor-arg>

<constructor-arg name=*"port"* value=*"${redis.node1.port}"*></constructor-arg>

</bean>

<bean class=*"org.springframework.data.redis.connection.RedisNode"*>

<constructor-arg name=*"host"* value=*"${redis.node2.host}"*></constructor-arg>

<constructor-arg name=*"port"* value=*"${redis.node2.port}"*></constructor-arg>

</bean>

<bean class=*"org.springframework.data.redis.connection.RedisNode"*>

<constructor-arg name=*"host"* value=*"${redis.node3.host}"*></constructor-arg>

<constructor-arg name=*"port"* value=*"${redis.node3.port}"*></constructor-arg>

</bean>

<bean class=*"org.springframework.data.redis.connection.RedisNode"*>

<constructor-arg name=*"host"* value=*"${redis.node4.host}"*></constructor-arg>

<constructor-arg name=*"port"* value=*"${redis.node4.port}"*></constructor-arg>

</bean>

<bean class=*"org.springframework.data.redis.connection.RedisNode"*>

<constructor-arg name=*"host"* value=*"${redis.node5.host}"*></constructor-arg>

<constructor-arg name=*"port"* value=*"${redis.node5.port}"*></constructor-arg>

</bean>

<bean class=*"org.springframework.data.redis.connection.RedisNode"*>

<constructor-arg name=*"host"* value=*"${redis.node6.host}"*></constructor-arg>

<constructor-arg name=*"port"* value=*"${redis.node6.port}"*></constructor-arg>

</bean>

</set>

</property>

</bean>

<!-- #################################################################### -->

<!-- # Jedis 连接工厂 -->

<!-- #################################################################### -->

<bean id=*"jedisConnectionFactory"*

class=*"org.springframework.data.redis.connection.jedis.JedisConnectionFactory"*>

<constructor-arg name=*"clusterConfig"* ref=*"redisClusterConfig"* />

<property name=*"timeout"* value=*"${redis.timeout}"* />

<property name=*"poolConfig"* ref=*"jedisPoolConfig"* />

<property name=*"password"* value=*"${redis.password}"* />

</bean>

<!-- #################################################################### -->

<!-- # Jedis 模板 -->

<!-- #################################################################### -->

<bean id=*"stringRedisSerializer"*

class=*"org.springframework.data.redis.serializer.StringRedisSerializer"* />

<bean id=*"redisTemplate"* class=*"org.springframework.data.redis.core.RedisTemplate"*>

<property name=*"connectionFactory"*>

<ref bean=*"jedisConnectionFactory"* />

</property>

<property name=*"keySerializer"*>

<ref bean=*"stringRedisSerializer"* />

</property>

<property name=*"hashKeySerializer"*>

<ref bean=*"stringRedisSerializer"* />

</property>

<property name=*"valueSerializer"*>

<ref bean=*"stringRedisSerializer"* />

</property>

<property name=*"hashValueSerializer"*>

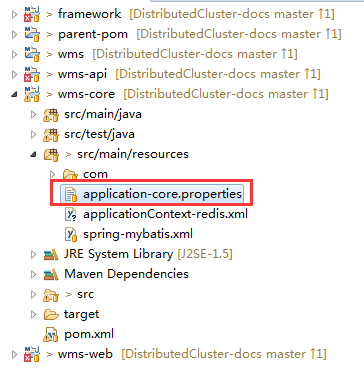
<ref bean=*"stringRedisSerializer"* />

</property>

</bean>

</beans>

修改application-core.properties，添加资源属性



#################################################

# Redis cluster environment variables

#################################################

redis.node1.host=192.168.5.107

redis.node2.host=192.168.5.107

redis.node3.host=192.168.5.107

redis.node4.host=192.168.5.107

redis.node5.host=192.168.5.107

redis.node6.host=192.168.5.107

redis.node1.port=6379

redis.node2.port=6479

redis.node3.port=6579

redis.node4.port=6679

redis.node5.port=6779

redis.node6.port=6879

redis.password=123456

redis.timeout=15000

redis.pool.minIdle=16

redis.pool.maxIdle=64

redis.pool.maxWaitMillis=30000

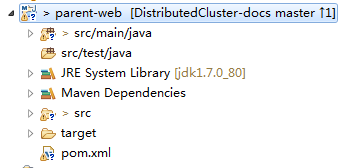
redis.pool.testOnCreate=true

redis.pool.testOnBorrow=true

redis.pool.testOnReturn=true

redis.pool.testWhileIdle=true

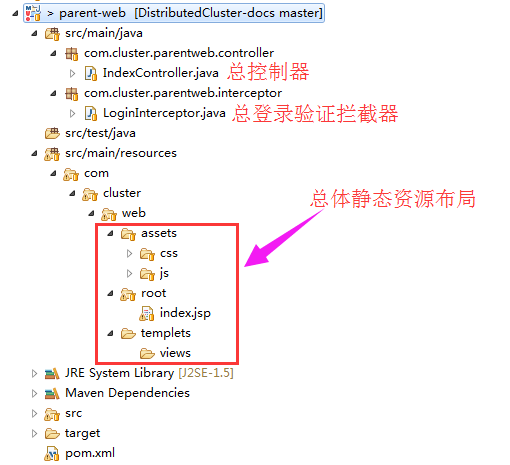
## 创建parent-web项目



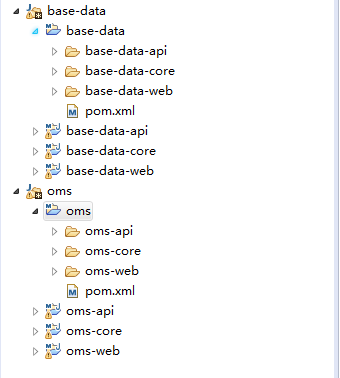
### 作用描述

控制平台的总体web层资源，

所有的XXX-web子项目将会引用parent-web项目



## 同样的方式组织创建oms，base-data两个项目，并添加相应的依赖关系

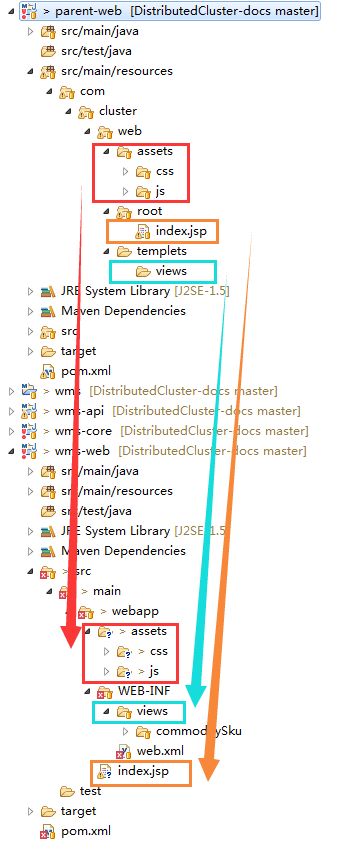


## maven-antrun-plugin插件

### ant介绍

### maven-antrun-plugin控制平台公共的布局页面

当XXX-web子项目install时，maven-antrun-plugin会将parent-web项目的平台总布局资源拷贝到各个XXX-web子项目的相应目录下



对于每个XXX-web子项目所使用的页面布局都是同一套页面，即这套页面



### maven-antrun-plugin配置

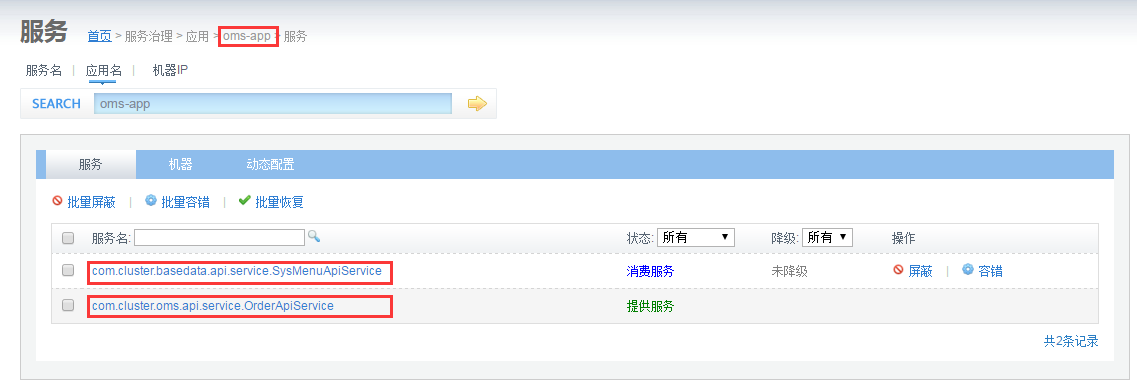
### 配置dubbo

以oms系统为例

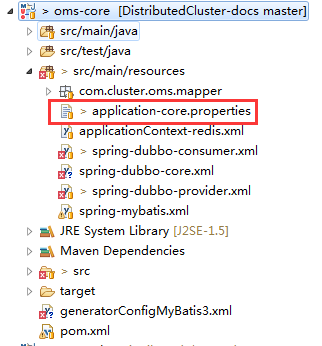
oms即消费方，又为提供方

消费[base-data-app](http://192.168.5.107:8070/governance/applications/base-data-app/services)的菜单查询服务

提供订单插入的对外服务



* + - 1. 修改application-core.properties，添加配置zookeeper配置



#################################################

# ZooKeeper cluster environment variables

#################################################

dubbo.registry.port=20890

dubbo.registry.address.node1=192.168.5.107:2181

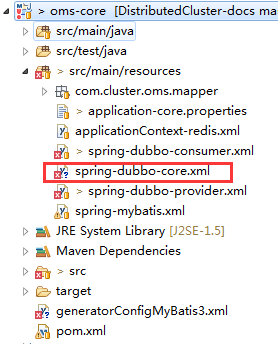
dubbo.registry.address.node2=192.168.5.107:2182

dubbo.registry.address.node3=192.168.5.107:2183

dubbo.registry.address.node4=192.168.5.107:2184

dubbo.registry.address.node5=192.168.5.107:2185

* + - 1. 配置注册中心，创建spring-dubbo-core.xml



<?xml version=*"1.0"* encoding=*"UTF-8"*?>

<beans xmlns=*"http://www.springframework.org/schema/beans"*

xmlns:xsi=*"http://www.w3.org/2001/XMLSchema-instance"*

xmlns:dubbo=*"http://code.alibabatech.com/schema/dubbo"*

xsi:schemaLocation=*"http://www.springframework.org/schema/beans http://www.springframework.org/schema/beans/spring-beans.xsd http://code.alibabatech.com/schema/dubbo http://code.alibabatech.com/schema/dubbo/dubbo.xsd"*>

<!-- 提供方应用信息，用于计算依赖关系 -->

<dubbo:application name=*"oms-app"* />

<!-- 使用multicast广播注册中心暴露服务地址 -->

<!-- <dubbo:registry address="zookeeper://192.168.1.167:2181" /> -->

<dubbo:registry protocol=*"zookeeper"* address=*"${dubbo.registry.address.node1}*

*,${dubbo.registry.address.node2}*

*,${dubbo.registry.address.node3}*

*,${dubbo.registry.address.node4}*

*,${dubbo.registry.address.node5}"* />

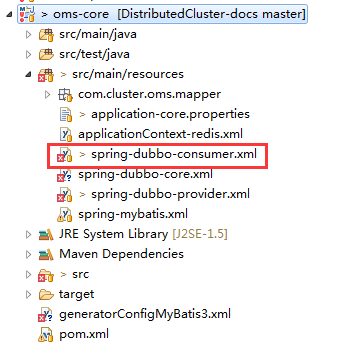
<!-- <dubbo:registry address="redis://192.168.1.167:6379" /> -->

<!-- 用dubbo协议在20890端口暴露服务 -->

<dubbo:protocol name=*"dubbo"* port=*"${dubbo.registry.port}"* />

</beans>

* + - 1. 配置消费者，创建spring-dubbo-consumer.xml



<?xml version=*"1.0"* encoding=*"UTF-8"*?>

<beans xmlns=*"http://www.springframework.org/schema/beans"*

xmlns:xsi=*"http://www.w3.org/2001/XMLSchema-instance"*

xmlns:dubbo=*"http://code.alibabatech.com/schema/dubbo"*

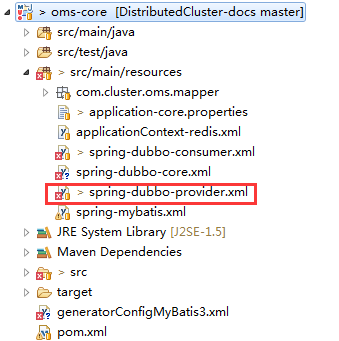
xsi:schemaLocation=*"http://www.springframework.org/schema/beans http://www.springframework.org/schema/beans/spring-beans.xsd http://code.alibabatech.com/schema/dubbo http://code.alibabatech.com/schema/dubbo/dubbo.xsd"*>

<!-- 生成远程服务代理，可以和本地bean一样使用sysMenuApiService -->

<dubbo:reference id=*"sysMenuApiService"* interface=*"com.cluster.basedata.api.service.SysMenuApiService"* />

</beans>

* + - 1. 配置提供者，创建spring-dubbo-consumer.xml



<?xml version=*"1.0"* encoding=*"UTF-8"*?>

<beans xmlns=*"http://www.springframework.org/schema/beans"*

xmlns:xsi=*"http://www.w3.org/2001/XMLSchema-instance"*

xmlns:dubbo=*"http://code.alibabatech.com/schema/dubbo"*

xsi:schemaLocation=*"http://www.springframework.org/schema/beans http://www.springframework.org/schema/beans/spring-beans.xsd http://code.alibabatech.com/schema/dubbo http://code.alibabatech.com/schema/dubbo/dubbo.xsd"*>

<!-- 订单组件实现类 -->

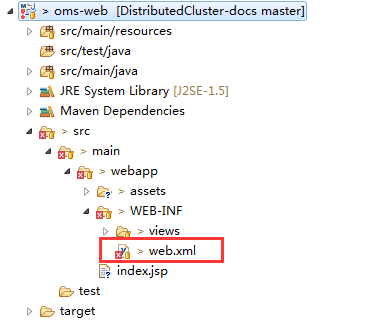
<bean id=*"orderApiService"* class=*"com.cluster.oms.api.service.impl.OrderApiServiceImpl"* />

<!-- 声明系统菜单组件暴露的服务接口 -->

<dubbo:service interface=*"com.cluster.oms.api.service.OrderApiService"* ref=*"orderApiService"* />

</beans>

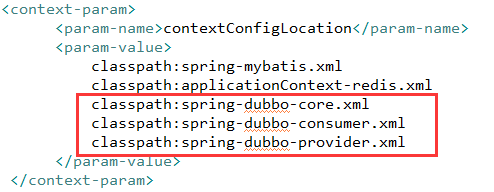
* + - 1. 修改web.xml



classpath:spring-dubbo-core.xml

classpath:spring-dubbo-consumer.xml

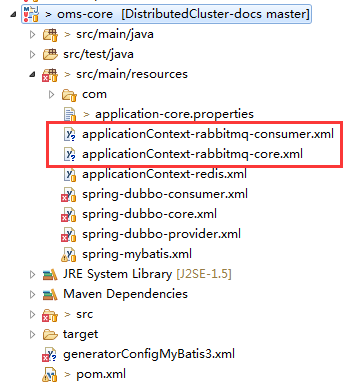
classpath:spring-dubbo-provider.xml



## 配置rabbitmq连接

创建applicationContext-rabbitmq-consumer.xml消费者配置文件

创建applicationContext-rabbitmq-core.xml核心配置文件



consumer消费者配置

<?xml version=*"1.0"* encoding=*"UTF-8"*?>

<beans xmlns=*"http://www.springframework.org/schema/beans"* xmlns:xsi=*"http://www.w3.org/2001/XMLSchema-instance"* xmlns:p=*"http://www.springframework.org/schema/p"* xmlns:c=*"http://www.springframework.org/schema/c"*

xmlns:context=*"http://www.springframework.org/schema/context"* xmlns:aop=*"http://www.springframework.org/schema/aop"* xmlns:util=*"http://www.springframework.org/schema/util"* xmlns:rabbit=*"http://www.springframework.org/schema/rabbit"*

xsi:schemaLocation=*"*

*http://www.springframework.org/schema/beans http://www.springframework.org/schema/beans/spring-beans-4.1.xsd*

*http://www.springframework.org/schema/context http://www.springframework.org/schema/context/spring-context-4.1.xsd*

*http://www.springframework.org/schema/aop http://www.springframework.org/schema/aop/spring-aop-4.1.xsd*

*http://www.springframework.org/schema/util http://www.springframework.org/schema/util/spring-util-4.1.xsd*

*http://www.springframework.org/schema/rabbit http://www.springframework.org/schema/rabbit/spring-rabbit-1.5.xsd"*>

<!-- #################################################################### -->

<!-- # RabbitMQ 消息监听容器（消费者） -->

<!-- #requeue-rejected 消息重试策略 可以参考死信队列重试 -->

<!-- #################################################################### -->

<rabbit:listener-container connection-factory=*"rabbitConnectionFactory"* message-converter=*"rabbitMessageConverter"* channel-transacted=*"false"* error-handler=*"logAndPropagateErrorHandler"* requeue-rejected=*"false"*>

<!-- 消费订单 -->

<rabbit:listener queues=*"QUEUE\_ORDER"* ref=*"orderConsumer"* method=*"onMessage"* />

</rabbit:listener-container>

</beans>

core核心配置

<?xml version=*"1.0"* encoding=*"UTF-8"*?>

<beans xmlns=*"http://www.springframework.org/schema/beans"* xmlns:xsi=*"http://www.w3.org/2001/XMLSchema-instance"* xmlns:p=*"http://www.springframework.org/schema/p"* xmlns:c=*"http://www.springframework.org/schema/c"*

xmlns:context=*"http://www.springframework.org/schema/context"* xmlns:aop=*"http://www.springframework.org/schema/aop"* xmlns:util=*"http://www.springframework.org/schema/util"* xmlns:rabbit=*"http://www.springframework.org/schema/rabbit"*

xsi:schemaLocation=*"*

*http://www.springframework.org/schema/beans http://www.springframework.org/schema/beans/spring-beans-4.1.xsd*

*http://www.springframework.org/schema/context http://www.springframework.org/schema/context/spring-context-4.1.xsd*

*http://www.springframework.org/schema/aop http://www.springframework.org/schema/aop/spring-aop-4.1.xsd*

*http://www.springframework.org/schema/util http://www.springframework.org/schema/util/spring-util-4.1.xsd*

*http://www.springframework.org/schema/rabbit http://www.springframework.org/schema/rabbit/spring-rabbit-1.5.xsd"*>

<!-- #################################################################### -->

<!-- # RabbitMQ 消息转换器 -->

<!-- #################################################################### -->

<bean id=*"rabbitMessageConverter"* class=*"org.springframework.amqp.support.converter.Jackson2JsonMessageConverter"*>

<property name=*"classMapper"*>

<bean class=*"org.springframework.amqp.support.converter.DefaultJackson2JavaTypeMapper"* />

</property>

</bean>

<!-- #################################################################### -->

<!-- # RabbitMQ 异常处理程序 -->

<!-- #################################################################### -->

<bean id=*"logAndPropagateErrorHandler"* class=*"org.springframework.beans.factory.config.FieldRetrievingFactoryBean"*>

<property name=*"staticField"*>

<value>org.springframework.scheduling.support.TaskUtils.LOG\_AND\_PROPAGATE\_ERROR\_HANDLER</value>

</property>

</bean>

<!-- #################################################################### -->

<!-- # RabbitMQ 线程池（供创建连接使用） -->

<!-- #################################################################### -->

<bean id=*"rabbitExecutor"* class=*"org.springframework.beans.factory.config.MethodInvokingFactoryBean"*>

<property name=*"staticMethod"*>

<value>java.util.concurrent.Executors.newFixedThreadPool</value>

</property>

<property name=*"arguments"*>

<array>

<value>10</value>

</array>

</property>

</bean>

<!-- #################################################################### -->

<!-- # rabbit 连接工厂 -->

<!-- #################################################################### -->

<rabbit:connection-factory id=*"rabbitConnectionFactory"* addresses=*"${rabbit.host}"* username=*"${rabbit.username}"* password=*"${rabbit.password}"*

executor=*"rabbitExecutor"* />

<!-- #################################################################### -->

<!-- # RabbitMQ 管理员 -->

<!-- #################################################################### -->

<rabbit:admin id=*"rabbitAdmin"* connection-factory=*"rabbitConnectionFactory"* />

<!-- #################################################################### -->

<!-- # RabbitMQ 队列 -->

<!-- #################################################################### -->

<!-- 订单 -->

<rabbit:queue id=*"QUEUE\_ORDER"* name=*"QUEUE\_ORDER"*>

<rabbit:queue-arguments>

<!-- 开启集群环境镜像队列 -->

<entry key=*"x-ha-policy"* value=*"all"* />

</rabbit:queue-arguments>

</rabbit:queue>

<!-- #################################################################### -->

<!-- # RabbitMQ 模板（生产者） -->

<!-- #################################################################### -->

<rabbit:template id=*"rabbitTemplate"* connection-factory=*"rabbitConnectionFactory"* channel-transacted=*"false"* message-converter=*"rabbitMessageConverter"* />

</beans>

## 功能实现

### base-data-web

dubbo：系统菜单查询功能

用户查询





### oms-web

dubbo：插入订单信息

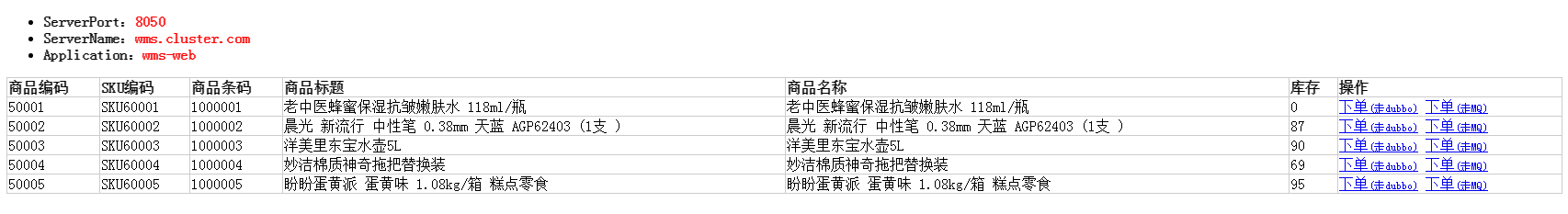
MQ：插入订单信息

订单查询



### wms-web

商品查询

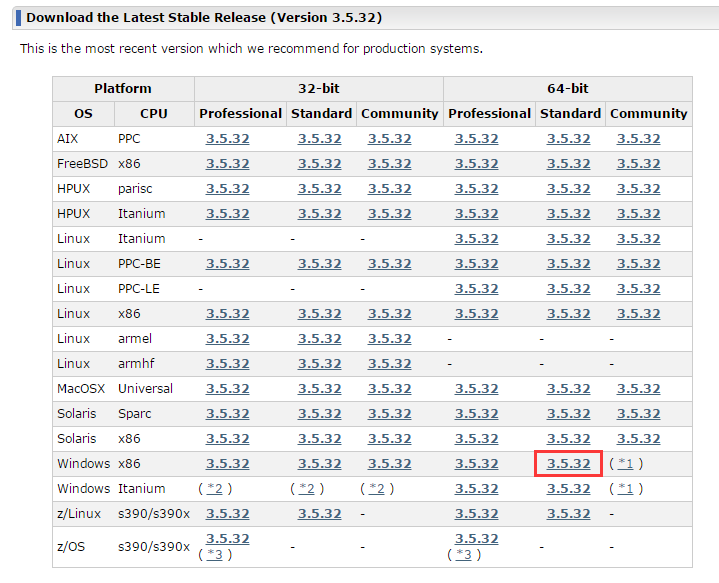


## Java Service Wrapper

### 介绍

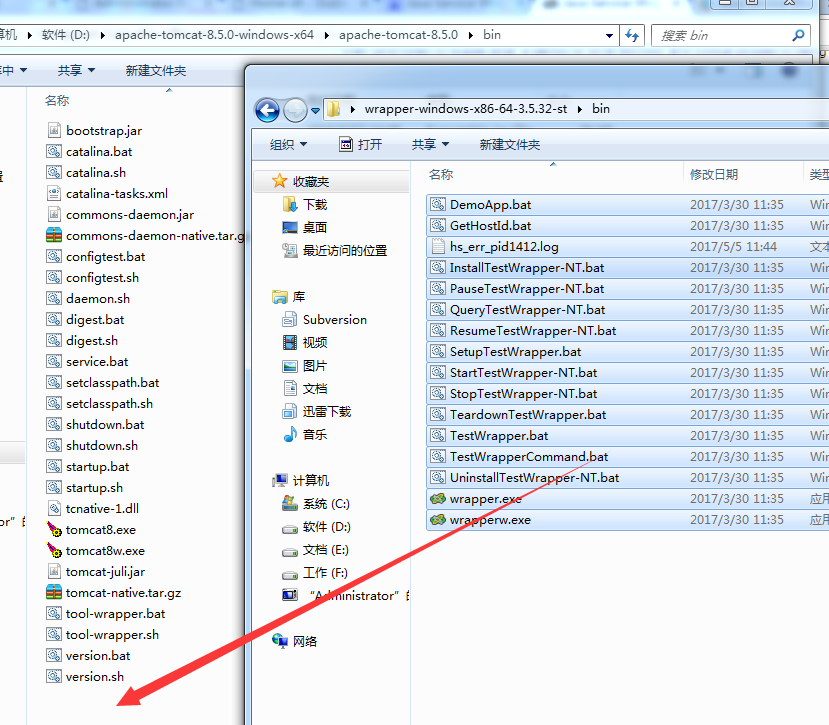
### 获取安装包

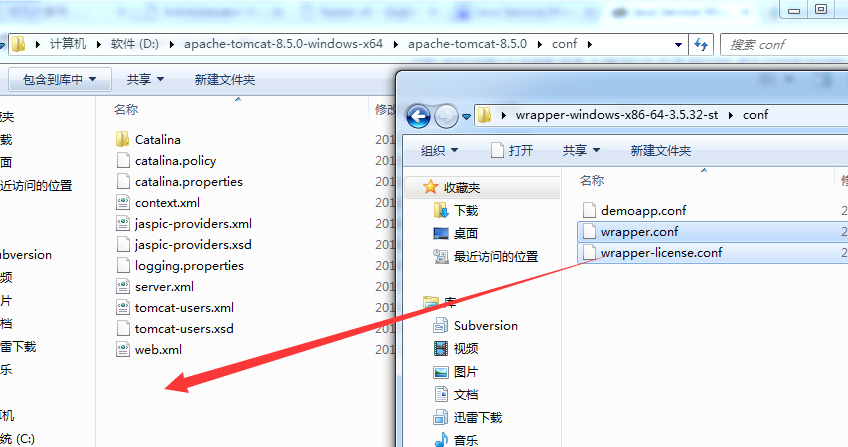
<https://wrapper.tanukisoftware.com/doc/english/download.jsp>

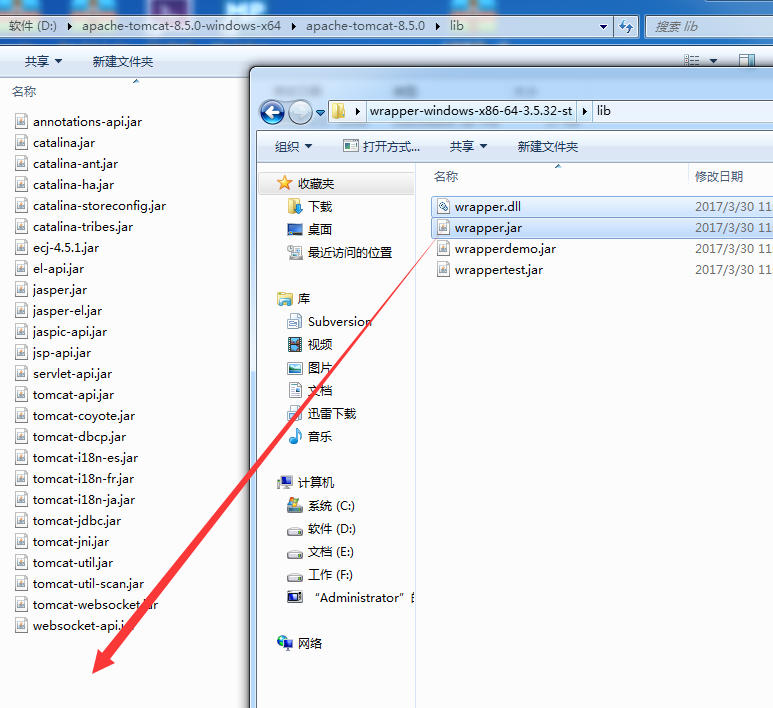


### 使用 Java Service Wrapper将tomcat注册为window服务配置

* + - 1. 拷贝如下文件到指定目录







* + - 1. 修改${CATALINA\_HOME}/conf目录下的wrapper.conf配置

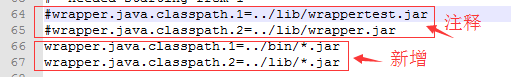
wrapper.java.mainclass=org.tanukisoftware.wrapper.WrapperStartStopApp



* + - 1. 设置读取的classpath

wrapper.java.classpath.1=../bin/\*.jar

wrapper.java.classpath.2=../lib/\*.jar



* + - 1. 配置tomcat启动类

wrapper.app.parameter.1=org.apache.catalina.startup.Bootstrap

wrapper.app.parameter.2=1

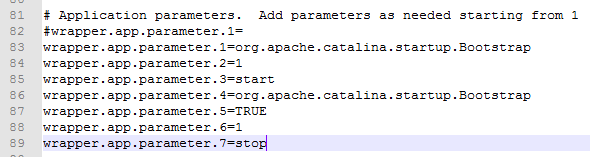
wrapper.app.parameter.3=start

wrapper.app.parameter.4=org.apache.catalina.startup.Bootstrap

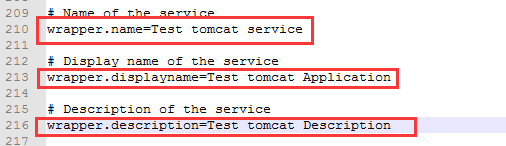
wrapper.app.parameter.5=TRUE

wrapper.app.parameter.6=1

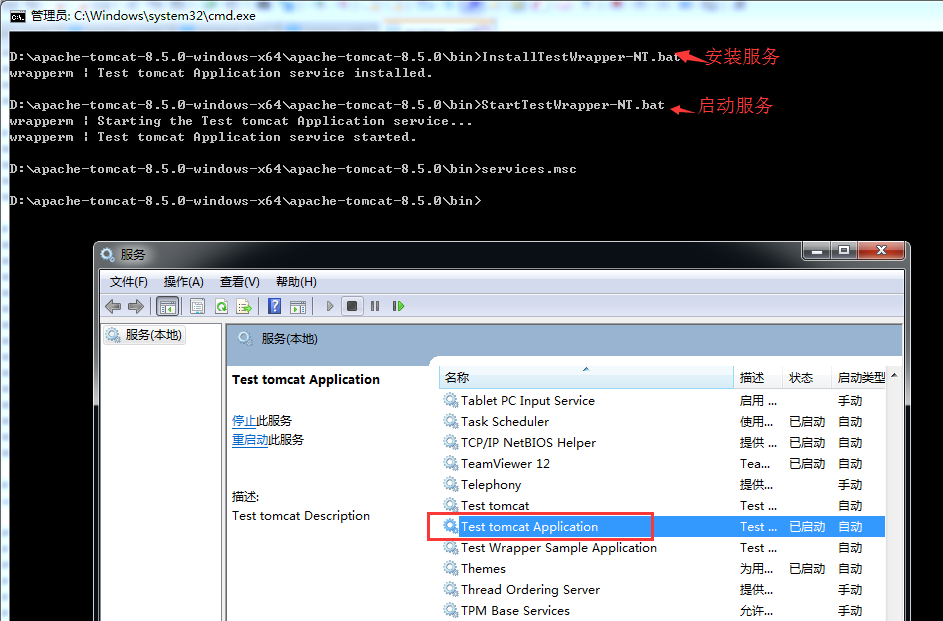
wrapper.app.parameter.7=stop



* + - 1. 配置应用名称

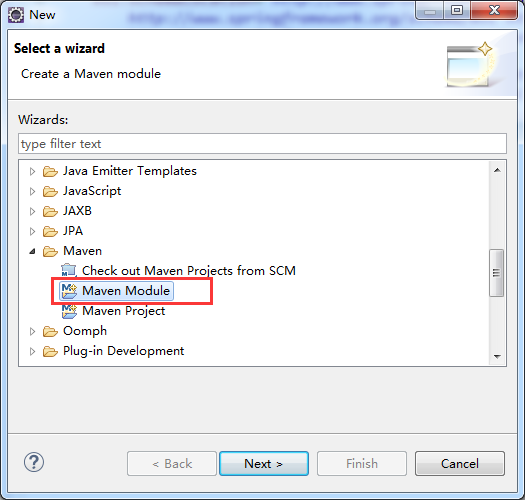


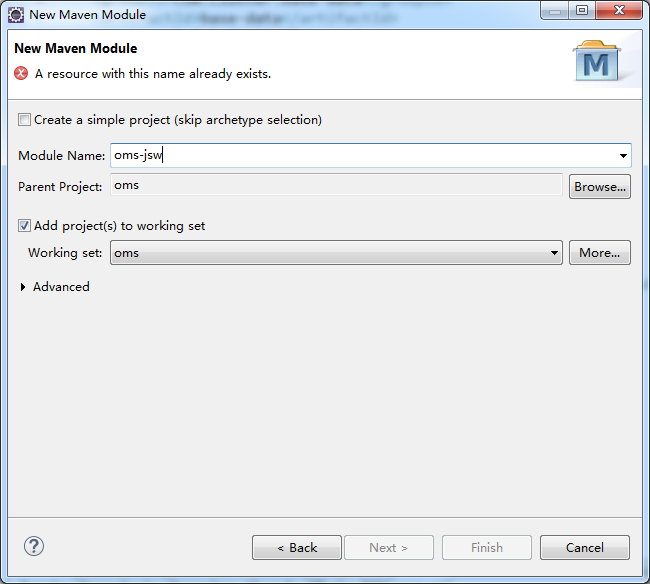
* + - 1. 查看

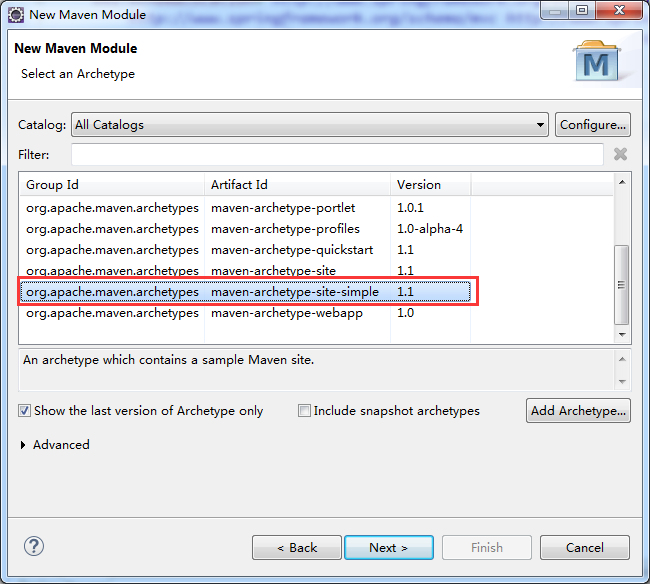


### 创建jsw项目，独立dubbo provider

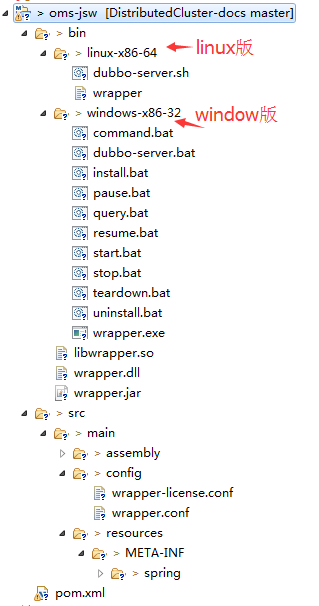
* + - 1. 创建项目







* + - 1. 创建如下目录结构，并拷贝相关文件到指定目录



* + - 1. 添加maven-assembly-plugin插件配置

修改parent-pom项目的pom.xml文件，加入配置

<!-- jsw打包配置 -->

<plugin>

<groupId>org.apache.maven.plugins</groupId>

<artifactId>maven-assembly-plugin</artifactId>

<version>${maven.assembly.plugin.version}</version>

<configuration>

<appendAssemblyId>false</appendAssemblyId>

<descriptors>

<descriptor>src/main/assembly/default.xml</descriptor>

</descriptors>

</configuration>

<executions>

<execution>

<id>make-assembly</id>

<phase>package</phase>

<goals>

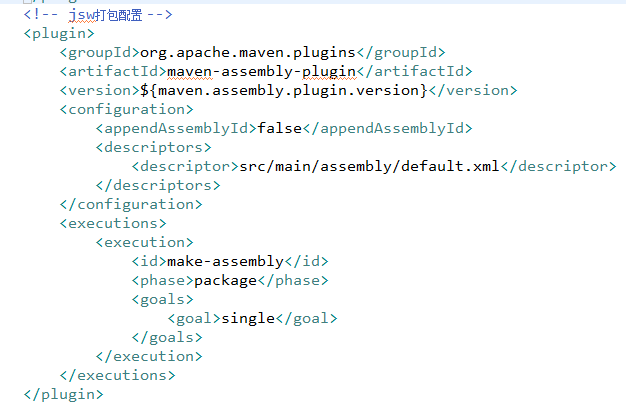
<goal>single</goal>

</goals>

</execution>

</executions>

</plugin>



<maven.assembly.plugin.version>2.5.4</maven.assembly.plugin.version>



修改oms-jsw项目的pom.xml文件，加入配置

引用oms-core

<dependencies>

<dependency>

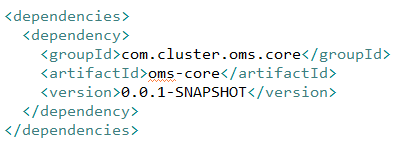
<groupId>com.cluster.oms.core</groupId>

<artifactId>oms-core</artifactId>

<version>0.0.1-SNAPSHOT</version>

</dependency>

</dependencies>



引用maven-assembly-plugin

<build>

<finalName>oms-jsw</finalName>

<plugins>

<plugin>

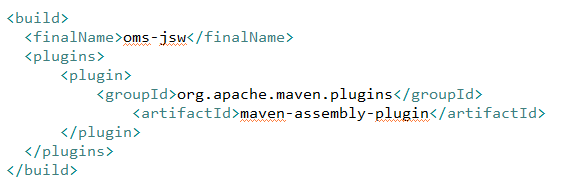
<groupId>org.apache.maven.plugins</groupId>

<artifactId>maven-assembly-plugin</artifactId>

</plugin>

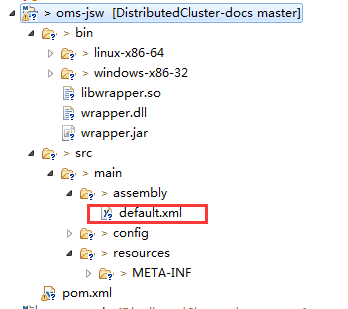
</plugins>

</build>



* + - 1. 创建maven-assembly-plugin的assembly文件

assembly目录下，创建default.xml文件



加入打包文件的配置，用来生成java service wrapper的目录结构

<assembly>

<id>dist</id>

<formats>

<format>zip</format>

</formats>

<includeBaseDirectory>true</includeBaseDirectory>

<dependencySets>

<dependencySet>

<useProjectArtifact>false</useProjectArtifact>

<outputDirectory>lib</outputDirectory>

<fileMode>0644</fileMode>

<directoryMode>0644</directoryMode>

</dependencySet>

</dependencySets>

<fileSets>

<fileSet>

<directory>bin/linux-x86-64</directory>

<outputDirectory>bin/linux-x86-64</outputDirectory>

<filtered>true</filtered>

<fileMode>0755</fileMode>

<includes>

<include>\*\*/dubbo-server.sh</include>

</includes>

</fileSet>

<fileSet>

<directory>bin/windows-x86-32</directory>

<outputDirectory>bin/windows-x86-32</outputDirectory>

<filtered>true</filtered>

<fileMode>0755</fileMode>

<includes>

<include>\*\*/dubbo-server.bat</include>

</includes>

</fileSet>

<fileSet>

<directory>bin</directory>

<outputDirectory>bin</outputDirectory>

<includes>

<include>\*\*</include>

</includes>

<excludes>

<exclude>\*\*/dubbo-server.bat</exclude>

<exclude>\*\*/dubbo-server.sh</exclude>

</excludes>

<fileMode>0744</fileMode>

<directoryMode>0744</directoryMode>

</fileSet>

<fileSet>

<directory>src/main/resources</directory>

<outputDirectory>conf</outputDirectory>

<filtered>true</filtered>

<includes>

<include>\*\*/\*.xml</include>

<include>\*\*/\*.properties</include>

<include>\*\*/com.alibaba.\*</include>

</includes>

<fileMode>0644</fileMode>

<directoryMode>0644</directoryMode>

</fileSet>

<fileSet>

<directory>src/main/config</directory>

<outputDirectory>conf</outputDirectory>

<filtered>true</filtered>

<includes>

<include>\*\*/\*.\*</include>

</includes>

<fileMode>0644</fileMode>

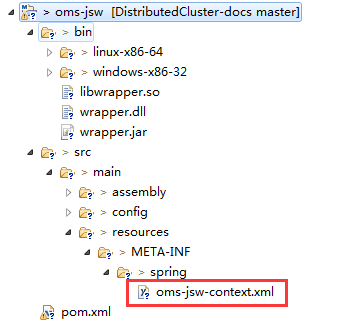
<directoryMode>0644</directoryMode>

</fileSet>

</fileSets>

</assembly>

* + - 1. 创建oms-jsw-context.xml文件



加入配置

<?xml version=*"1.0"* encoding=*"UTF-8"*?>

<beans xmlns=*"http://www.springframework.org/schema/beans"*

xmlns:xsi=*"http://www.w3.org/2001/XMLSchema-instance"*

xmlns:mvc=*"http://www.springframework.org/schema/mvc"*

xmlns:context=*"http://www.springframework.org/schema/context"*

xsi:schemaLocation=*"http://www.springframework.org/schema/beans http://www.springframework.org/schema/beans/spring-beans.xsd*

*http://www.springframework.org/schema/mvc http://www.springframework.org/schema/mvc/spring-mvc.xsd*

*http://www.springframework.org/schema/context http://www.springframework.org/schema/context/spring-context.xsd"*>

<import resource=*"classpath\*:/applicationContext-redis.xml"*/>

<import resource=*"classpath\*:/spring-mybatis.xml"*/>

<import resource=*"classpath\*:/spring-dubbo-core.xml"*/>

<import resource=*"classpath\*:/spring-dubbo-provider.xml"*/>

</beans>

* + - 1. wrapper.conf配置

开启debug模式

#include.debug

wrapper.license.debug=TRUE

设置wrapper-license.conf文件位置

include ../../conf/wrapper-license.conf

#include ../../conf/wrapper-license.conf

设置JAVA\_HOME

set.JAVA\_HOME=/usr/local/jdk1.7.0\_80

wrapper.java.command=%JAVA\_HOME%/bin/java

设置启动类

wrapper.java.mainclass=org.tanukisoftware.wrapper.WrapperSimpleApp

设置classpath

set.default.JSWORDER\_HOME=../..

wrapper.java.classpath.1=%JSWORDER\_HOME%/bin/wrapper.jar

wrapper.java.classpath.2=%JSWORDER\_HOME%/conf/

wrapper.java.classpath.3=%JSWORDER\_HOME%/lib/\*.jar

设置wrapper.jar，Wrapper.DLL or libwrapper.so路径

wrapper.java.library.path.1=../../bin

设置应用启动类和参数

wrapper.app.parameter.1=com.alibaba.dubbo.container.Main

wrapper.app.parameter.2=spring

设置应用启服务名称相关

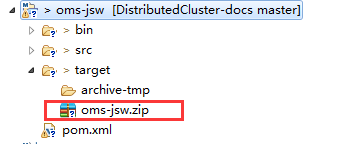
wrapper.console.title=oms-jsw Dubbo Server Console

wrapper.name=oms-jsw

wrapper.displayname=oms-jsw Dubbo Server

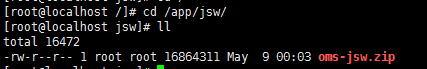
wrapper.description=oms-jsw Dubbo Server

* + - 1. intall



* + - 1. 部署

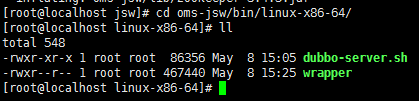
[root@localhost /]# cd /app/jsw/



[root@localhost jsw]# unzip oms-jsw.zip



[root@localhost jsw]# cd oms-jsw/bin/linux-x86-64/



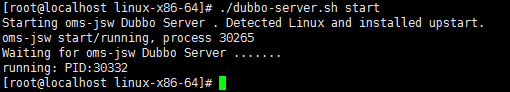
第一次先执行install

[root@localhost linux-x86-64]# ./dubbo-server.sh install

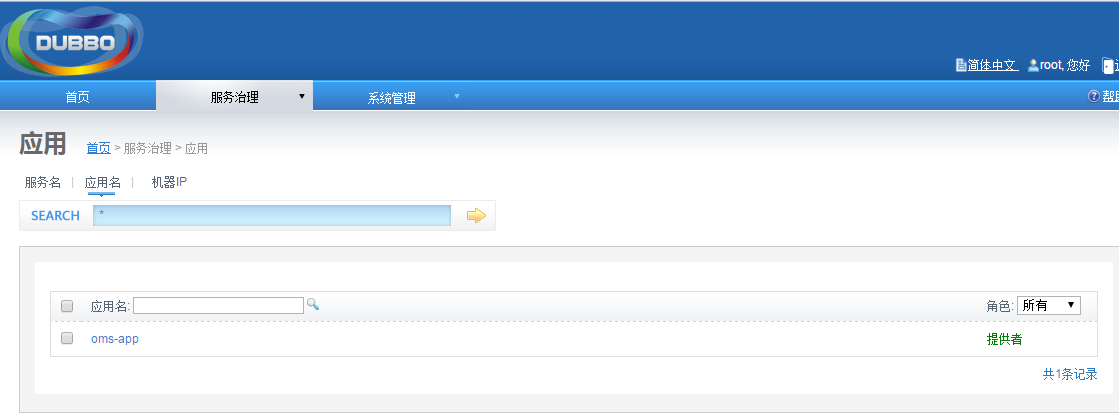


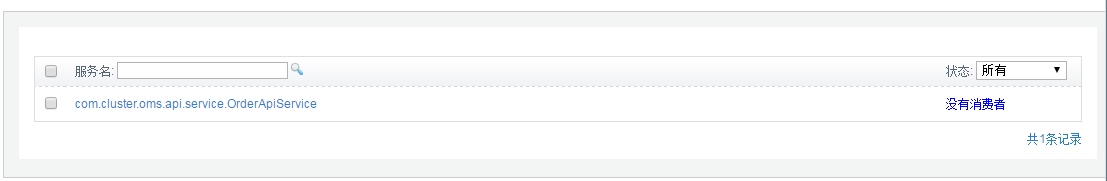
启动

[root@localhost linux-x86-64]# ./dubbo-server.sh start

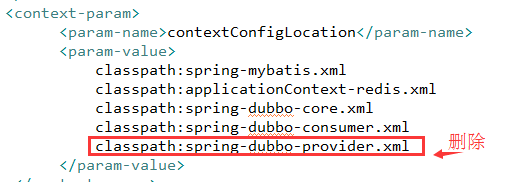


* + - 1. 查看服务





* + - 1. 剔除oms-web项目中的dubbo provider



# 附录

## vi编辑器没有颜色

[root@mail test]#rpm -qa|grep vim  
vim-minimal-6.1-29  
vim-common-6.1-29  
vim-enhanced-6.1-29  
  
直接用yum升级安装就ok了，  
[root@mail test]#yum -y install vim-enhanced  
  
安装完毕后，vi个文件试试，还是不行，原来是链接文件也要修改，ok，因为如下：  
[root@mail test]# ll /bin/vi  
lrwxrwxrwx  1 root root 12 Dec  9 10:52 /bin/vi -> /usr/bin/vim  
  
那就修改下吧：  
[root@mail test]#mv /bin/vi /bin/vi.bak  
[root@mail test]#ln -s /usr/bin/vim /bin/vi  
  
顺便修改下vi的设置，对编写perl代码更方便：  
[root@mail test]# vi /etc/vimrc  
  
  syntax on  
  set hlsearch  
  set nu  
  set tabstop=4  
  set shiftwidth=4  
  set expandtab  
  set shiftround  
  
OK，这样就可以很爽的用perl了

## Rz命令安装

## tree显示目录树

安装

[root@localhost software]# yum -y install tree

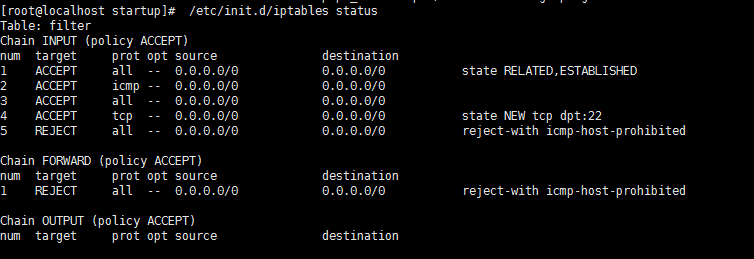


## 关闭防火墙

### 临时关闭防火墙命令

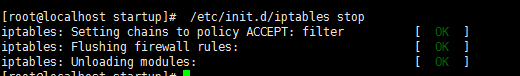
查看状态

[root@localhost startup]# /etc/init.d/iptables status



临时关闭

[root@localhost startup]# /etc/init.d/iptables stop



### 永久关闭防火墙命令

[root@localhost startup]# chkconfig iptables off

（重启电脑生效）



## 查看系统位数命令

[root@localhost ~]# getconf LONG\_BIT



## 查看隐藏文件命令

[root@bogon support-files]# ls -a

