SpringBoot In Linux

1. **部署前准备**
   1. **安装Java**

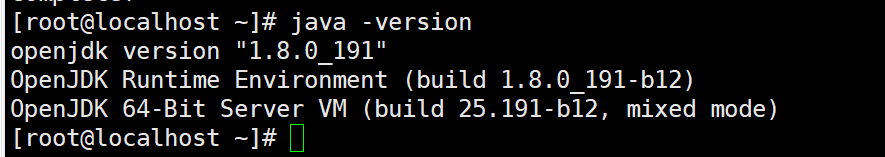
sudo yum -y install java-1.8.0-openjdk #jre

sudo yum -y install java-1.8.0-openjdk-devel #jdk

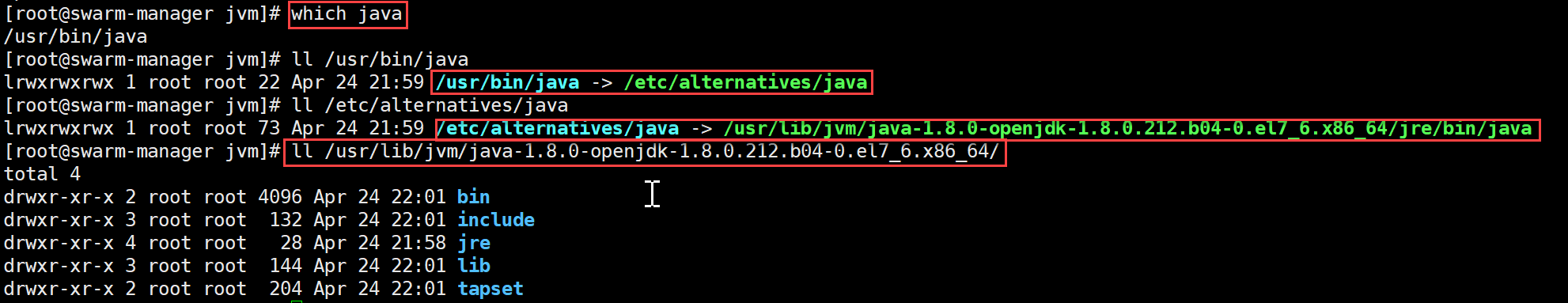
但对于centos有个问题，不可以安装成gcj(例如：gij(GNU libgcj) version 4.4.6 20110731(Red Hat 4.4.6-3)),导致[Jekins不工作](https://issues.jenkins-ci.org/browse/JENKINS-743),如果查看java版本类似上例，需要卸载，安装其他版本

java -version #查看Java版本

yum remove java #卸载异常版本



which java #查看java执行路径，备注：java实际的路径为-/usr/lib/jvm



设置java路径：vi /etc/profile 在文件最后添加下列配置

export JAVA\_HOME=/usr/lib/jvm/java-1.8.0-openjdk-1.8.0.212.b04-0.el7\_6.x86\_64

export JRE\_HOME=$JAVA\_HOME/jre

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

export PATH=$JAVA\_HOME/bin:$JRE\_HOME/bin:$PATH

使配置生效： source /etc/profile

* 1. **安装Maven**
     1. **下载Maven安装包**

wget <https://mirrors.tuna.tsinghua.edu.cn/apache/maven/maven-3/3.6.0/binaries/apache-maven-3.6.0-bin.tar.gz>

* + 1. **解压Maven安装包**

tar -zxvf apache-maven-3.6.0-bin.tar.gz

mv apache-maven-3.6.0 /usr/local/maven3

* + 1. **配置Maven路径**

vi /etc/profile #最后添加以下内容

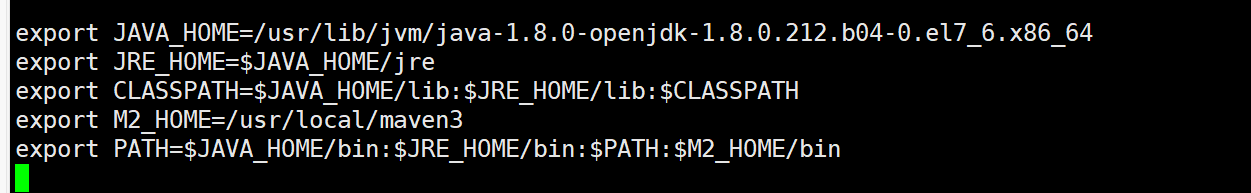
export M2\_HOME=/usr/local/maven3

export PATH=$PATH:$JAVA\_HOME/bin:$M2\_HOME/bin

#保存退出后运行下面的命令使配置生效，或者重启服务器生效

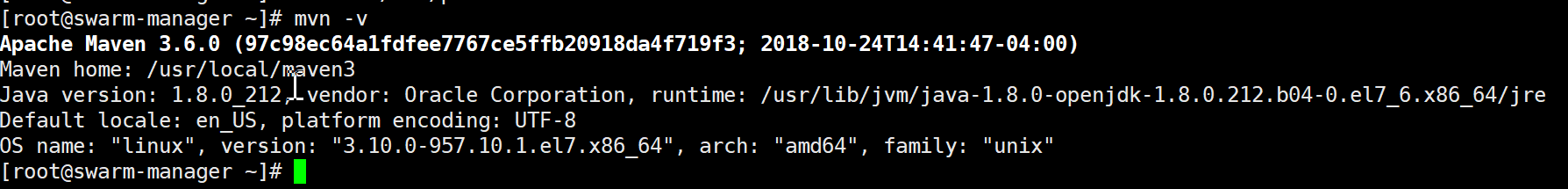
source /etc/profile

配置结果如下图：



* + 1. **验证Maven是否配置成功**

mvn -v



* + 1. **配置国内阿里云Maven镜像**

vi /usr/local/maven3/conf/settings.xml

配置mirrors的子节点，添加如下mirror

<mirror>

<id>nexus-aliyun</id>

<mirrorOf>central</mirrorOf>

<name>Nexus aliyun</name>

<url>http://maven.aliyun.com/nexus/content/groups/public</url>

</mirror>



* 1. **安装Git**

sudo yum -y install git

git --version

* 1. **安装Docker**
     1. **使用yum源安装**

sudo wget -P /etc/yum.repos.d/ <http://mirrors.aliyun.com/docker-ce/linux/centos/docker-ce.repo>

sudo yum install -y docker-ce docker-ce-cli containerd.io

docker --version

* + 1. **安装Docker Compose**

curl -L https://github.com/docker/compose/releases/download/1.23.2/docker-compose-`uname -s`-`uname -m` -o /usr/local/bin/docker-compose

chmod +x /usr/local/bin/docker-compose

docker-compose --version

* + 1. **安装Docker Machine**

sudo wget -P /etc/yum.repos.d/ http://download.virtualbox.org/virtualbox/rpm/rhel/virtualbox.repo

sudo yum install -y VirtualBox-5.1 #安装VirtualBox

curl -L https://github.com/docker/machine/releases/download/v0.16.1/docker-machine-`uname -s`-`uname -m` >/tmp/docker-machine &&

chmod +x /tmp/docker-machine &&

sudo cp /tmp/docker-machine /usr/local/bin/docker-machine

docker-machine -version

* + 1. **使用国内Docker镜像源**

vi /etc/docker/daemon.json #修改为以下内容

{

"registry-mirrors": ["https://registry.docker-cn.com"],

"live-restore": true

}

* + 1. **启动及测试**
       1. **启动Docker**

service docker start

chkconfig docker on

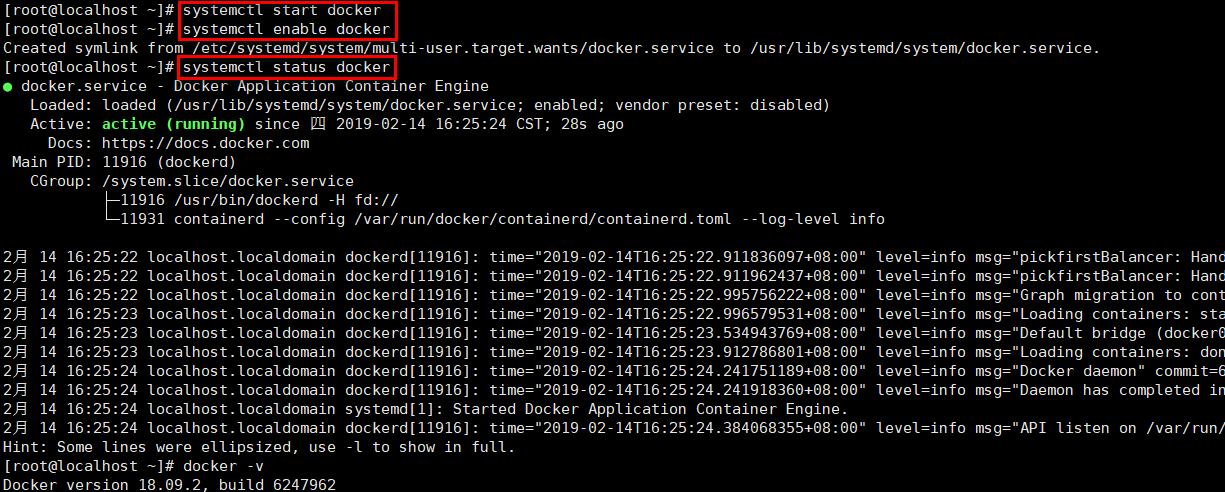
#LCTT 译注：此处采用了旧式的 sysv 语法，如采用CentOS 7中支持的新式 systemd 语法，如下：

systemctl start docker.service #开机自启 Docker CE

systemctl enable docker.service

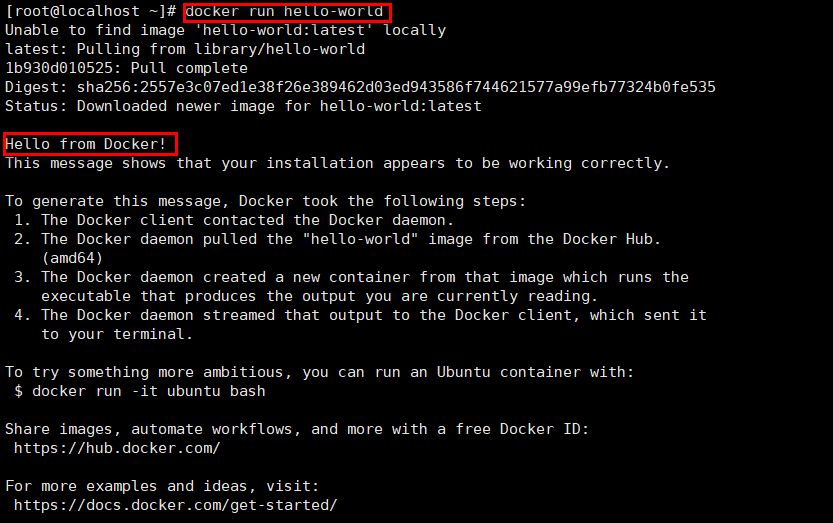
systemctl status docker #查看Docker状态

docker -v #查看docker版本



* + - 1. **测试Docker是否安装正确**

docker run hello-world



* 1. **安装Jenkins**
     1. **安装前准备**
        1. **防火墙设置**

#配置Jenkins端口8080

firewall-cmd --zone=public --add-port=8080/tcp --permanent

#开端口后必须重新加载

firewall-cmd --reload

* + - 1. **安装JQ**

yum install epel-release -y

yum install jq -y

jq --version

#或者

wget -o jq <https://github.com/stedolan/jq/releases/download/jq-1.6/jq-linux64>

chmod +x ./jq

cp jq /usr/bin

#验证

echo '{"first\_name":"John","full\_name":"John Doe","last\_name":"Doe"}' | jq .

echo '{"first\_name":"John","full\_name":"John Doe","last\_name":"Doe"}' | jq . full\_name

* + 1. **安装Jenkins**
       1. **拉取库的配置到本地**

sudo wget -O /etc/yum.repos.d/jenkins.repo http://pkg.jenkins-ci.org/redhat/jenkins.repo

* + - 1. **导入公钥**

sudo rpm --import https://jenkins-ci.org/redhat/jenkins-ci.org.key

* + - 1. **安装Jenkins**

＃`-y`参数：回答全部问题为是

sudo yum -y install jenkins

* + - 1. **更新Jenkins**

yum update jenkins

* + - 1. **卸载Jenkins**

rpm -e jenkins

删除其他相关文件

find / -iname jenkins | xargs -n 1000 rm -rf

* + 1. **Jenkins相关配置信息**
       - 1. 系统配置文件

cat /etc/sysconfig/jenkins | more

可以获得几个重要配置项目信息

JENKINS\_HOME="/var/lib/jenkins",存放jenkins 配置及工作文件

JENKINS\_PORT="8080",jenkins默认8080端口

* + - * 1. 配置文件夹

ls /var/lib/jenkins

有jobs、logs、plugins等文件夹及文件若干。  
这次主要看了看plugins文件夹，所有插件都在里面，如插件ssh-slaves,会有一个ssh-slaves文件夹及ssh-slaves.jpi。  
当某个插件未安装成功时，会有一个以.tmp结尾的文件

* + - * 1. 日志

/var/lib/jenkins/logs

/var/log/jenkins/jenkins.log,记录了插件安装等日志，失败信息原因等很清晰

* + - * 1. 其他
    1. **启动Jenkins及安装插件**
       1. **启动服务**

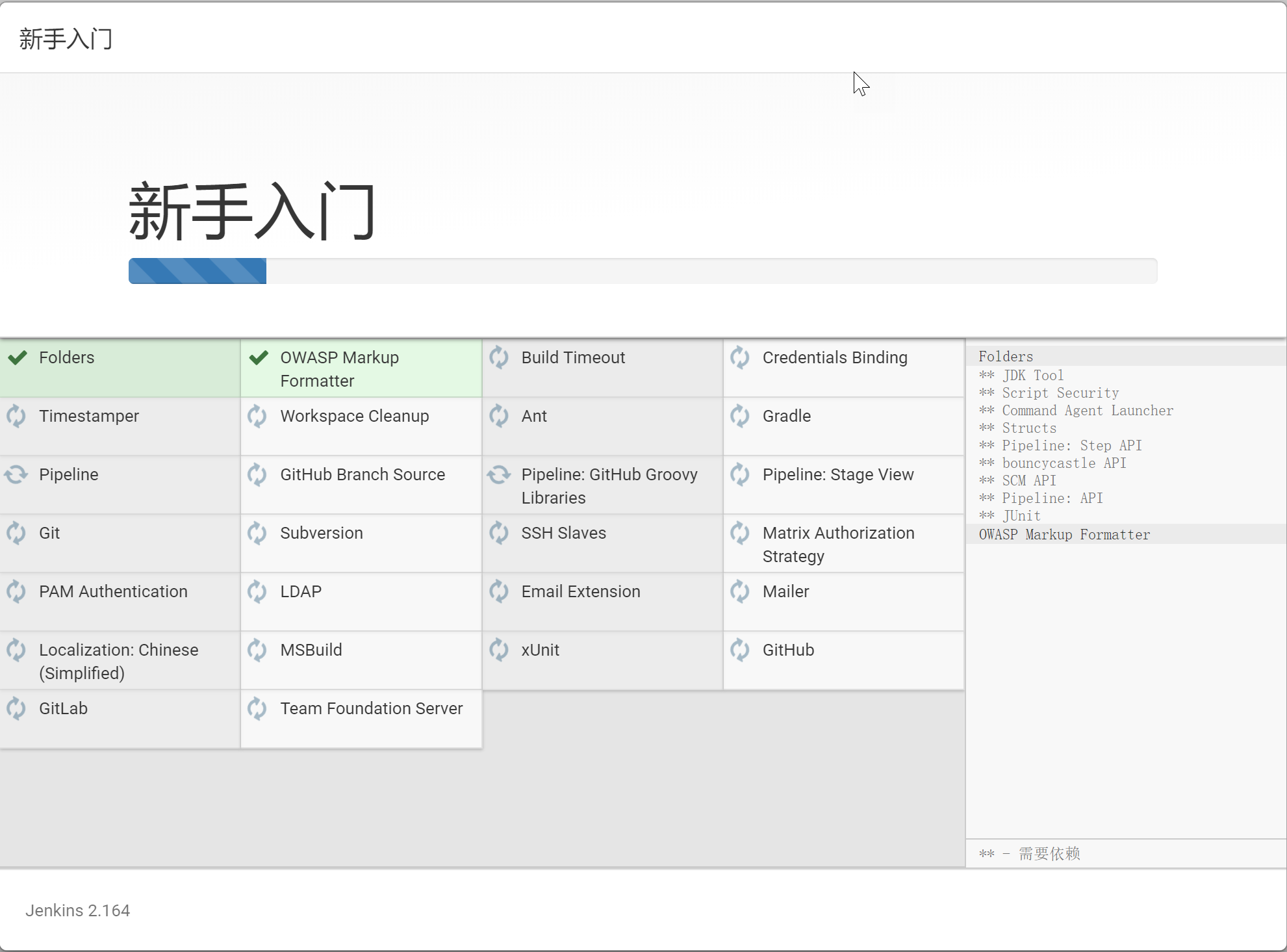
service jenkins start

* + - 1. **浏览器访问：http://192.168.56.99:8080**
      2. **获取管理员密码**

vi /var/lib/jenkins/secrets/initialAdminPassword



* + - 1. **安装插件**

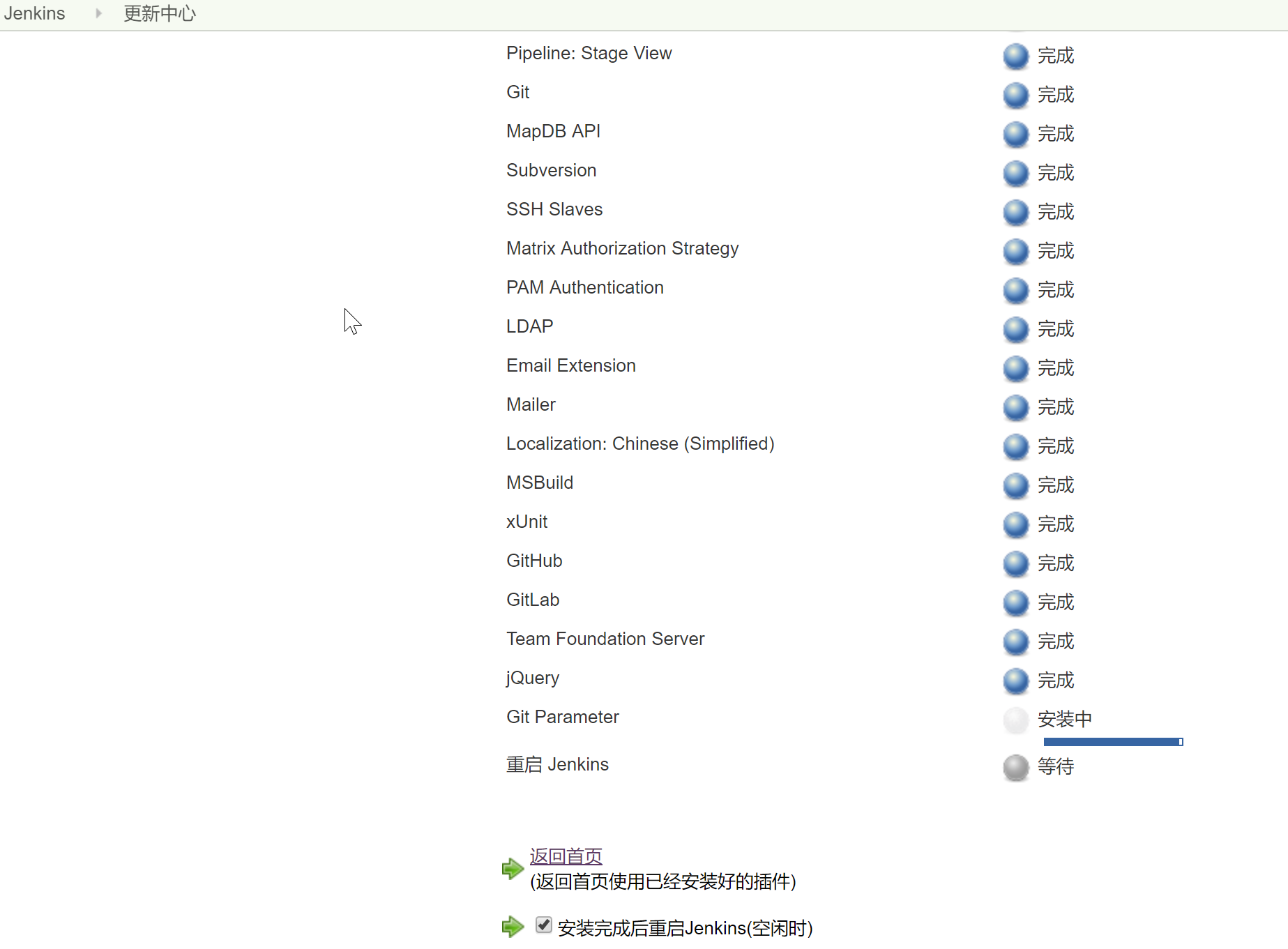


* + - 1. **安装Git Parameter插件**

1. 系统管理-->插件管理-->可选插件-->搜索git parameter；



1. 安装插件



* + 1. **配置Nginx的反向代理**
       1. **创建Nginx下Jenkins的配置文件**

vi /usr/local/nginx/conf/conf.d/jenkins.conf

* + - 1. **配置文件内容如下**

upstream jenkins {

keepalive 32; # keepalive connections

server 127.0.0.1:8080; # jenkins ip and port

}

server {

#注意这里，要把默认的那个default\_server去掉,因为我们在下面要单独配置域名访问，所以这里不要留default\_server，不然会报错。

listen 80;

#这里写你想设置的域名，可以写多个，与名之间用空格隔开

server\_name jenkins.domain.com;

# Load configuration files for the default server block.

location / {

proxy\_set\_header Host $host:$server\_port;

proxy\_set\_header X-Real-IP $remote\_addr;

proxy\_set\_header X-Forwarded-For $proxy\_add\_x\_forwarded\_for;

proxy\_set\_header X-Forwarded-Proto $scheme;

# Fix the "It appears that your reverse proxy set up is broken" error.

proxy\_pass http://127.0.0.1:8080;

proxy\_read\_timeout 90;

proxy\_redirect http://127.0.0.1:8080 https://jenkins.domain.com;

# Required for new HTTP-based CLI

proxy\_http\_version 1.1;

proxy\_request\_buffering off;

# workaround for https://issues.jenkins-ci.org/browse/JENKINS-45651

add\_header 'X-SSH-Endpoint' 'jenkins. domain.com:50022' always;

}

access\_log /var/log/jenkins/access.log;

error\_log /var/log/jenkins/error.log;

error\_page 404 /404.html;

location = /404.html {

root /usr/local/nginx/html;

}

# redirect server error pages to the static page /50x.html

error\_page 500 502 503 504 /50x.html;

location = /50x.html {

root /usr/local/nginx/html;

}

}

* + - 1. **重启Nginx**

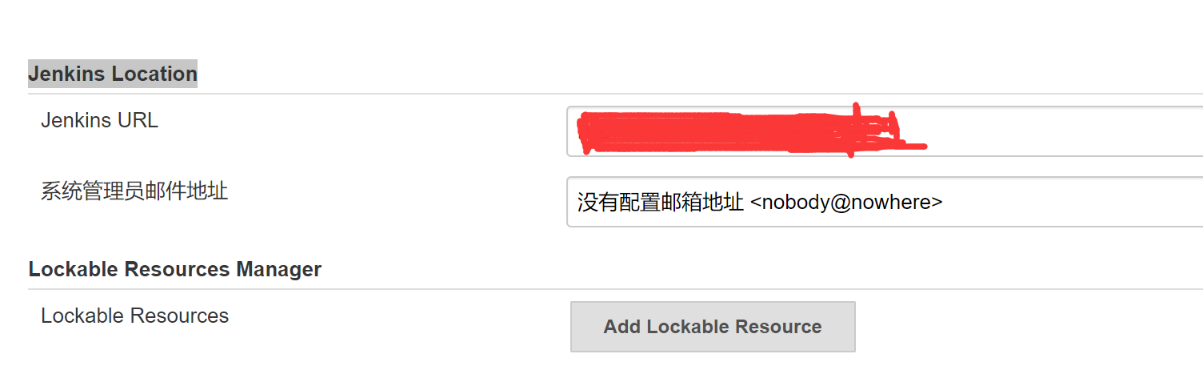
nginx -t

nginx -s reload

* + - 1. **配置Jenkins访问路径：**

访问Jenkins：<http://jenkins.domain.com>

进入系统管理-->系统设置-->**Jenkins Location，设置URL为：**<http://jenkins.domain.com>



* 1. **其他**

1. **SpringBoot相关Starter**
   1. **Web容器（默认使用Tomcat）**
      1. **Jetty容器**

* 修改pom.xml

<dependency>

<groupId>org.springframework.boot</groupId>

<artifactId>spring-boot-starter-web</artifactId>

<!-- 排除默认使用tomcat容器 -->

<exclusions>

<exclusion>

<groupId>org.springframework.boot</groupId>

<artifactId>spring-boot-starter-tomcat</artifactId>

</exclusion>

</exclusions>

</dependency>

<!-- 使用Jetty容器 -->

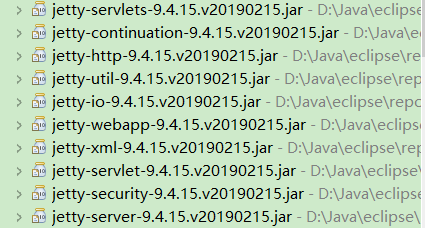
<dependency>

<groupId>org.springframework.boot</groupId>

<artifactId>spring-boot-starter-jetty</artifactId>

</dependency>

* 相关的jar包



* + 1. **undertow容器**

修改pom.xml

<dependency>

<groupId>org.springframework.boot</groupId>

<artifactId>spring-boot-starter-web</artifactId>

<!-- 排除默认使用tomcat容器 -->

<exclusions>

<exclusion>

<groupId>org.springframework.boot</groupId>

<artifactId>spring-boot-starter-tomcat</artifactId>

</exclusion>

</exclusions>

</dependency>

<!-- undertow不支持jsp -->

<!-- 使用undertow容器 -->

<dependency>

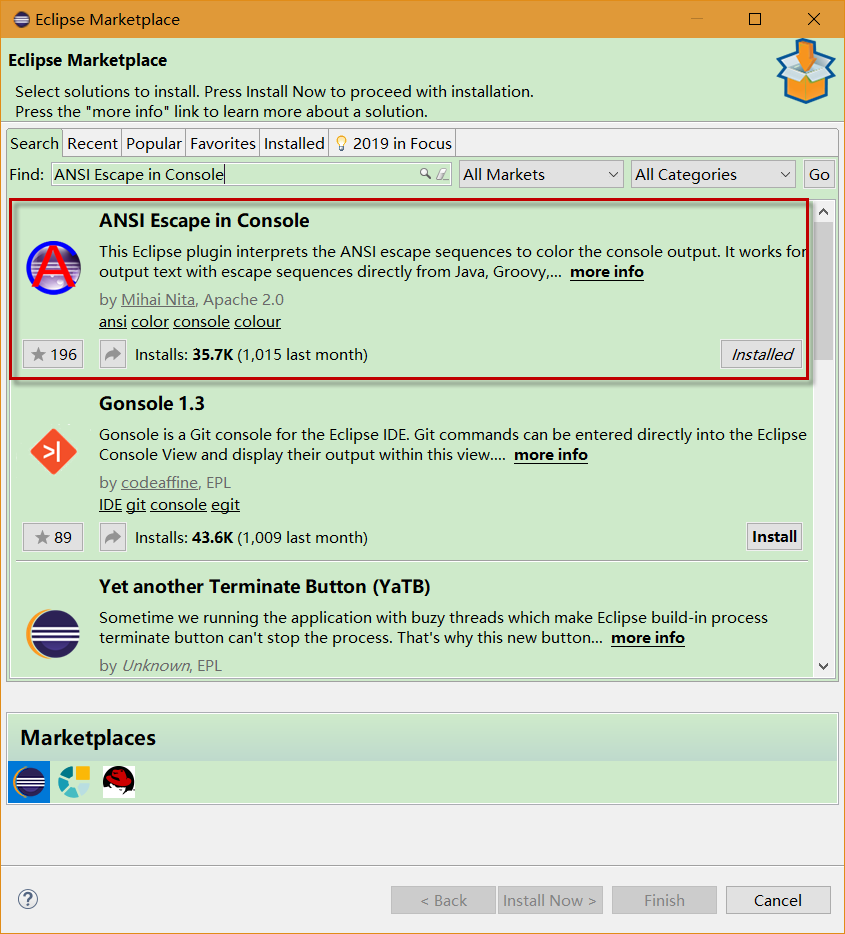
<groupId>org.springframework.boot</groupId>

<artifactId>spring-boot-starter-undertow</artifactId>

</dependency>

* + 1. **其他**
  1. **其他**

1. **SpringBoot相关插件**
   1. **Console支持彩色输出插件：ANSI Escape in Console**
      1. 进入Eclipse Marketplace，搜索ANSI Escape in Console并插件



* + 1. 设置application.properties

#Console支持彩色输出，需要安装插件：ANSI Escape in Console

spring.output.ansi.enabled=DETECT

* + 1. 配置日志（logback.xml）

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

<configuration scan="true" scanPeriod="3 seconds">

<property name="LOG\_HOME" value="D://logs" />

<!-- 彩色日志 -->

<!-- 彩色日志依赖的渲染类 -->

<conversionRule conversionWord="clr" converterClass="org.springframework.boot.logging.logback.ColorConverter" />

<conversionRule conversionWord="wex" converterClass="org.springframework.boot.logging.logback.WhitespaceThrowableProxyConverter" />

<conversionRule conversionWord="wEx" converterClass="org.springframework.boot.logging.logback.ExtendedWhitespaceThrowableProxyConverter" />

<!-- 彩色日志格式 -->

<property name="CONSOLE\_LOG\_PATTERN" value="${CONSOLE\_LOG\_PATTERN:-%clr(%d{yyyy-MM-dd HH:mm:ss.SSS}){faint} %clr(${LOG\_LEVEL\_PATTERN:-%5p}) %clr(${PID:- }){magenta} %clr(---){faint} %clr([%15.15t]){faint} %clr(%-40.40logger{39}){cyan} %clr(:){faint} %m%n${LOG\_EXCEPTION\_CONVERSION\_WORD:-%wEx}}" />

<!--设置日志输出为控制台-->

<appender name="STDOUT" class="ch.qos.logback.core.ConsoleAppender">

<encoder>

<pattern>${CONSOLE\_LOG\_PATTERN}</pattern>

<charset>utf8</charset>

</encoder>

</appender>

<!-- 不带彩色的日志在控制台输出时候的设置 -->

<!-- <appender name="STDOUT" class="ch.qos.logback.core.ConsoleAppender">

<encoder class="ch.qos.logback.classic.encoder.PatternLayoutEncoder">

格式化输出：%d表示日期，%thread表示线程名，%-5level：级别从左显示5个字符宽度%msg：日志消息，%n是换行符

<pattern>%d{yyyy-MM-dd HH:mm:ss.SSS} [%thread] %-5level %logger{50} - %msg%n</pattern>

</encoder>

</appender> -->

<!-- 按照每天生成日志文件 -->

<appender name="FILE" class="ch.qos.logback.core.rolling.RollingFileAppender">

<!-- 日志记录器的滚动策略，按日期，按大小记录 -->

<rollingPolicy class="ch.qos.logback.core.rolling.TimeBasedRollingPolicy">

<!--日志文件输出的文件名-->

<FileNamePattern>${LOG\_HOME}/MultiTenant-%d{yyyy-MM-dd}.log</FileNamePattern>

<!--日志文件保留天数-->

<MaxHistory>30</MaxHistory>

</rollingPolicy>

<!-- 追加方式记录日志 -->

<append>true</append>

<!-- 日志文件的格式 -->

<encoder class="ch.qos.logback.classic.encoder.PatternLayoutEncoder">

<!--格式化输出：%d表示日期，%thread表示线程名，%-5level：级别从左显示5个字符宽度%msg：日志消息，%n是换行符-->

<pattern>%d{yyyy-MM-dd HH:mm:ss.SSS} [%thread] %-5level %logger{50} - %msg%n</pattern>

<charset>utf-8</charset>

</encoder>

<!--日志文件最大的大小-->

<triggeringPolicy class="ch.qos.logback.core.rolling.SizeBasedTriggeringPolicy">

<MaxFileSize>10MB</MaxFileSize>

</triggeringPolicy>

</appender>

<!-- 日志输出级别 -->

<root>

<level value="INFO"/>

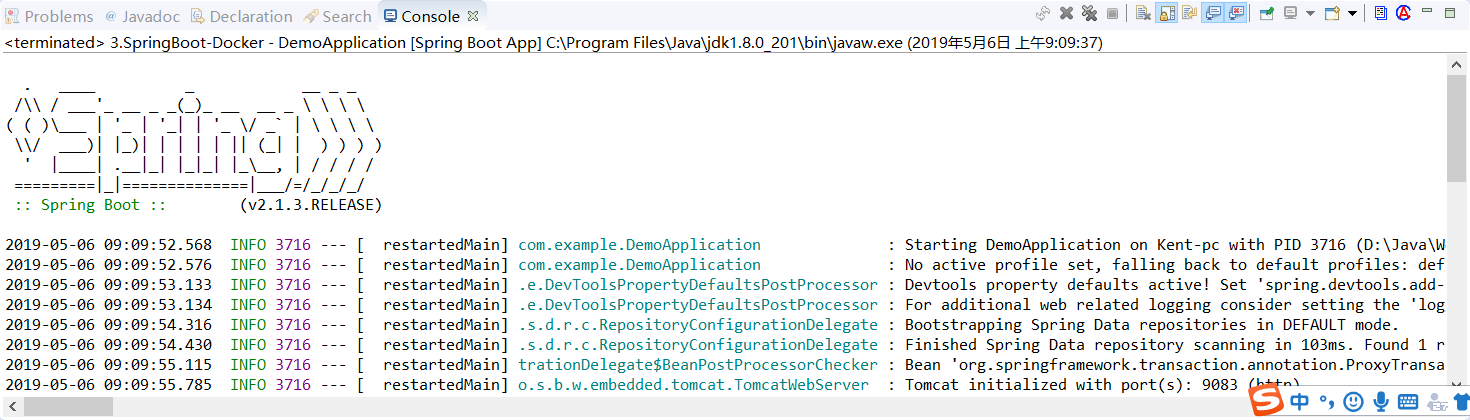
<appender-ref ref="STDOUT"/>

<appender-ref ref="FILE"/>

</root>

</configuration>

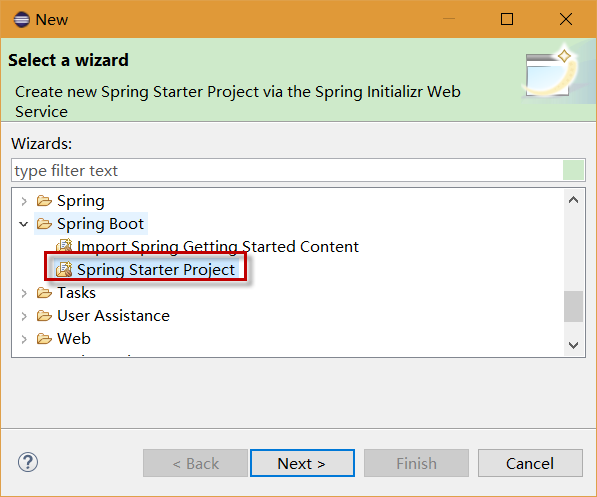
* + 1. 运行结果



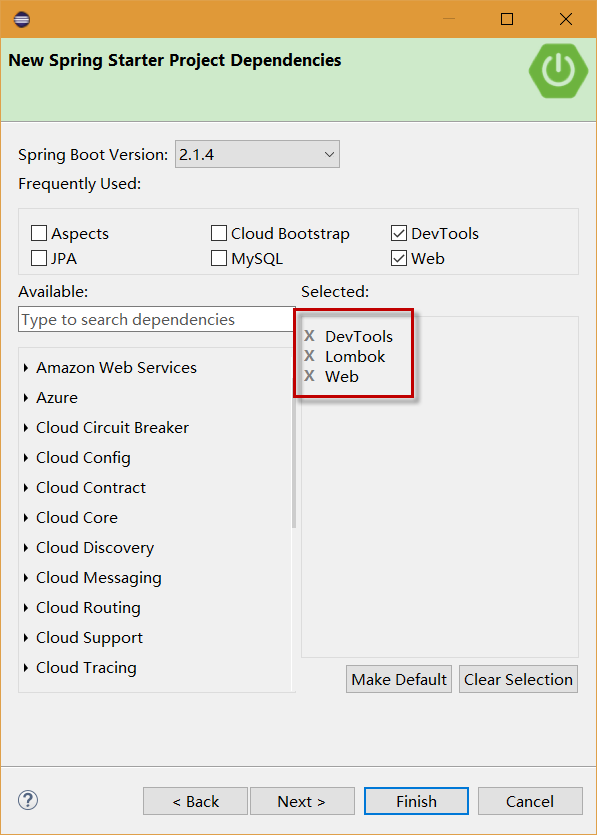
* 1. **其他**

1. **使用Docker部署SpringBoot Web项目**
   1. **创建SpringBoot的Web项目**

* 创建SpringBoot项目



* 选择项目依赖项



* 创建HelloController，添加index方法，返回字符串：Hello Docker！

@RestController

public class HelloController {

@RequestMapping("/")

public String index() {

return "Hello Docker!";

}

}

* 设置application.properties

#设置应用名

spring.application.name=Bootstrap Spring Boot

#设置启动端口

server.port=9090

#热部署生效

spring.devtools.restart.enabled: true

* 1. **Web项目添加Docker支持**
     1. **添加Docker插件**
* pom.xml文件添加相关属性

<properties>

<java.version>1.8</java.version>

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

<docker.image.prefix>springboot</docker.image.prefix>

<maven.build.timestamp.format>yyyyMMddHHmmss</maven.build.timestamp.format>

</properties>

* pom.xml中添加Docker插件

<build>

<plugins>

<plugin>

<groupId>org.springframework.boot</groupId>

<artifactId>spring-boot-maven-plugin</artifactId>

</plugin>

<!-- Docker maven plugin -->

<plugin>

<groupId>com.spotify</groupId>

<artifactId>docker-maven-plugin</artifactId>

<version>1.0.0</version>

<configuration>

<!-- 镜像名称 -->

<imageName>${docker.image.prefix}/${project.artifactId}</imageName>

<!-- 镜像标签 -->

<imageTags>

<!--可以指定多个标签 建议弄成自增序列或者时间戳类型，用于区分版本 -->

<imageTag>${maven.build.timestamp}</imageTag>

</imageTags>

<baseImage>java</baseImage>

<entryPoint>["java", "-jar", "/${project.build.finalName}.jar"]</entryPoint>

<!-- dockerfile文件路径 -->

<dockerDirectory>src/main/docker</dockerDirectory>

<resources>

<resource>

<targetPath>/</targetPath>

<directory>${project.build.directory}</directory>

<include>${project.build.finalName}.jar</include>

</resource>

</resources>

</configuration>

</plugin>

<!-- Docker maven plugin -->

</plugins>

</build>

* + 1. **添加Dockerfile文件**

在项目路径（src/main/docker）下添加dockerfile文件，文件内容如下：

FROM openjdk:8-jdk-alpine

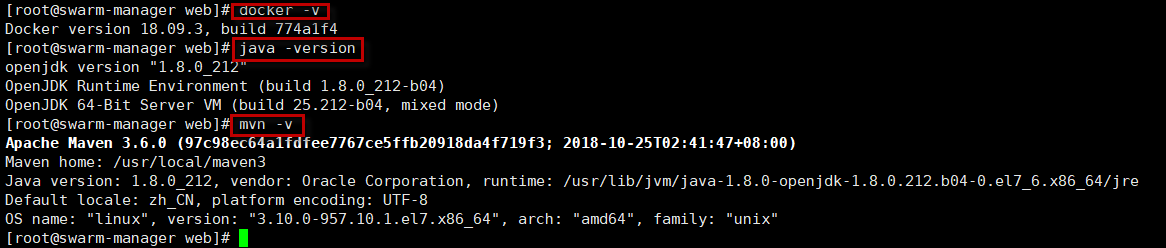
VOLUME /tmp

ADD SpringBootInDocker-1.0.0.1-SNAPSHOT.jar app.jar

ENTRYPOINT ["java","-Djava.security.egd=file:/dev/./urandom","-jar","/app.jar"]

* 1. **将项目部署至Docker**
     1. **检测环境**

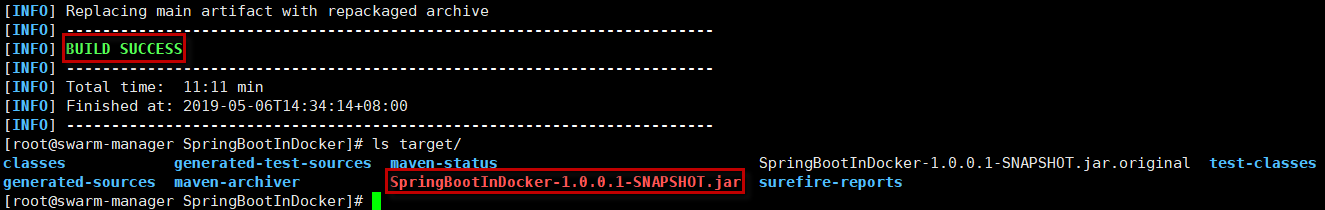
查看Linux系统是否安装Docker、Java、Maven等环境



* + 1. **测试项目**
* 打包Java项目

mvn pacakge

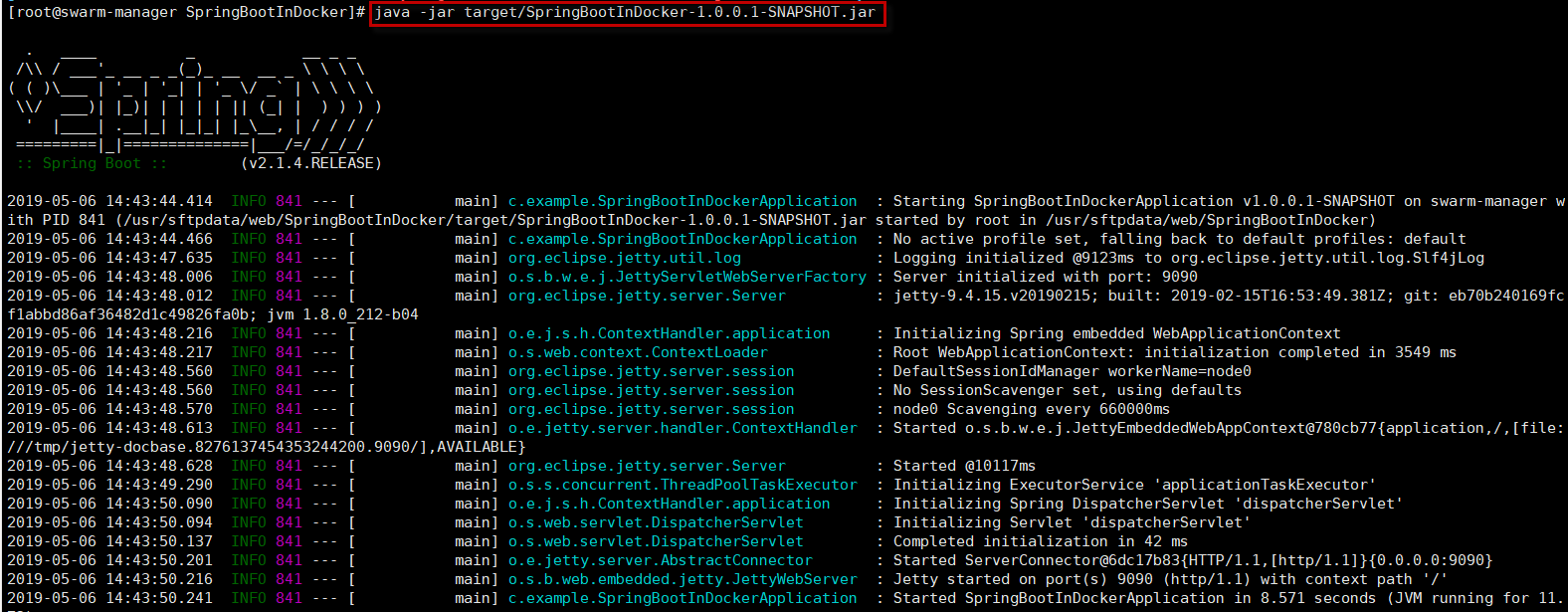
打包结果如下：



* 测试项目是否正常运行

java -jar target/SpringBootInDocker-1.0.0.1-SNAPSHOT.jar

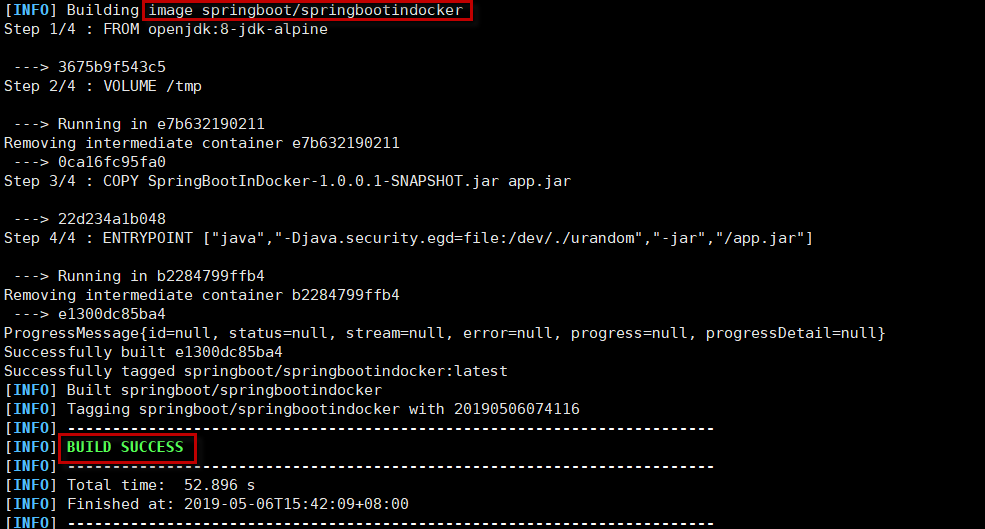
运行结果如下：



* 访问项目：curl http://192.168.56.99:9090
  + 1. **生成Docker镜像**

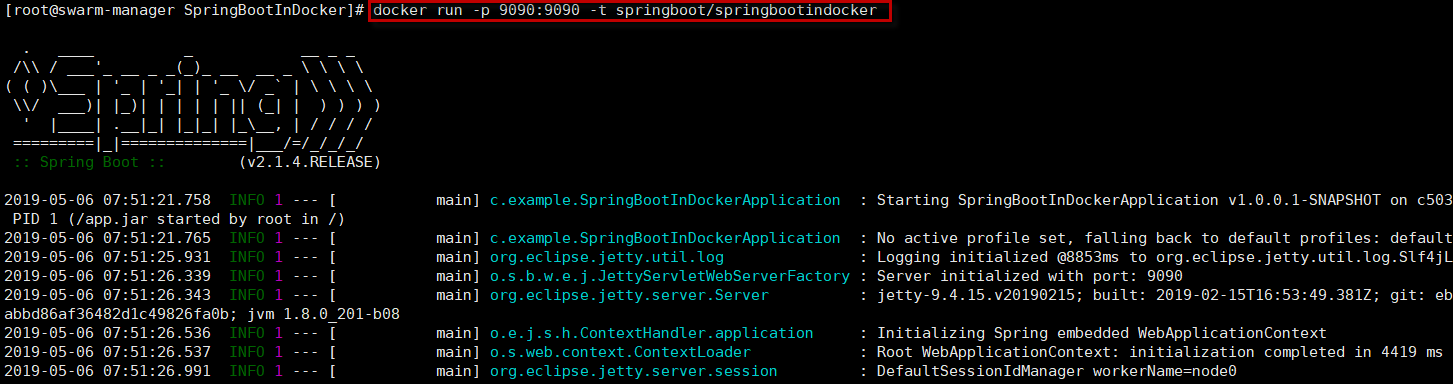
mvn package docker:build

查看结果如下：



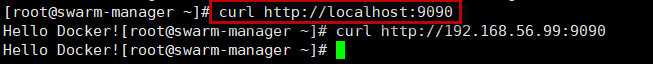
* + 1. **运行Docker镜像**

docker run -p 9090:9090 -t springboot/springbootindocker



* + 1. **测试Docker项目**

curl <http://192.168.56.99:9090>

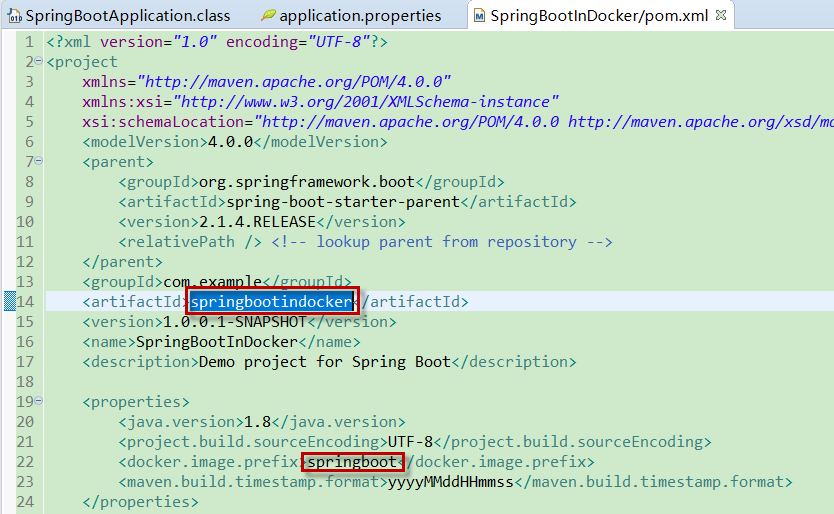


* 1. **相关问题**
     1. **mvn docker:build fails with "{}->unix://localhost:80: Connection reset by peer**

相关问题：<https://github.com/spotify/docker-maven-plugin/issues/357>

原因：docker镜像名称不支持大写字母

解决方案：pom.xml文件中的imageName修改成小写字母，即project.artifactId改为小写



* + 1. **other**
  1. **相关资料**
* 官方资料：[**https://spring.io/guides/gs/spring-boot-docker/**](https://spring.io/guides/gs/spring-boot-docker/)
  1. **其他**

1. **其他**