**使用Jenkins配置Git+Maven的自动化构建**

[**http://blog.csdn.net/pucao\_cug/article/details/52373655**](http://blog.csdn.net/pucao_cug/article/details/52373655)

**Java配置maven+jenkins+git（svn）+tomcat自动编译和部署（持续集成）**

<http://blog.csdn.net/u013322876/article/details/72637854>

**jenkins+svn+maven+tomcat一键构建部署**

<http://blog.csdn.net/wangqi880/article/details/70169916>

**Jenkins+Maven+Git搭建持续集成和自动化部署的配置手记**

<http://blog.csdn.net/linlzk/article/details/48808017>

**使用jekins自动构建部署java maven项目（jdk1.7+tomcat7.0+jenkins2.19.3）**

<http://blog.csdn.net/u011277123/article/details/76615236>

**一部分**

linux下的jenkins的安装：

jenkins下载网址：<http://jenkins-ci.org/>

1.准备：JDK环境，tomcat环境，maven环境，jenkins.war包

2.在linux上安装JDK，tomcat和maven，如下：

① 安装JDK

官网下载软件包，上传到服务器，解压，配置环境变量

export JAVA\_HOME=/jenkins/tomcat/jdk1.8.0\_152

export CLASSPATH=.:$JAVA\_HOME/lib/dt.jar:$JAVA\_HOME/lib/tools.jar

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

export JAVA\_HOME CLASSPATH PATH

#source /etc/profile

#java -version

② 安装maven

官网下载软件包，上传到服务器，解压，移动解压包到/usr/local ,配置环境变量

mv apache-maven-3.5.2 /usr/local/

配置环境变量

export MAVEN\_HOME=/usr/local/apache-maven-3.5.2

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

#source /etc/profile

#mvn --version

③ 安装tomcat

官网下载软件包，上传到服务器，解压

tar -xvf apache-tomcat-7.0.54.tar.gz

将jenkins.war包移动到apache-tomcat-7.0.54/webapps下

启动tomcat时，jenkins也随之启动

注意：这里如果忘记了jenkins密码

找到根目录下的jenkins

cd /root/.jenkins/users/

找到用户名，进入目录，找到config.xml文件

修改password，#jbcrypt:$2a$10$DdaWzN64JgUtLdvxWIflcuQu2fgrrMSAMabF5TSrGK5nXitqK9ZMS

保存，重启jenkins程序

然后输入用户名，密码：111111

④安装git

tar xvf git-2.10.0.tar.gz

cd git-2.10.0

./configure --prefix=/app/gitlab/

make && make install

遇到的问题

（1）[ERROR] Please refer to E:\maven\web\_nanchang\target\surefire-reports for the individual test results.

maven编译时出现There are test failures

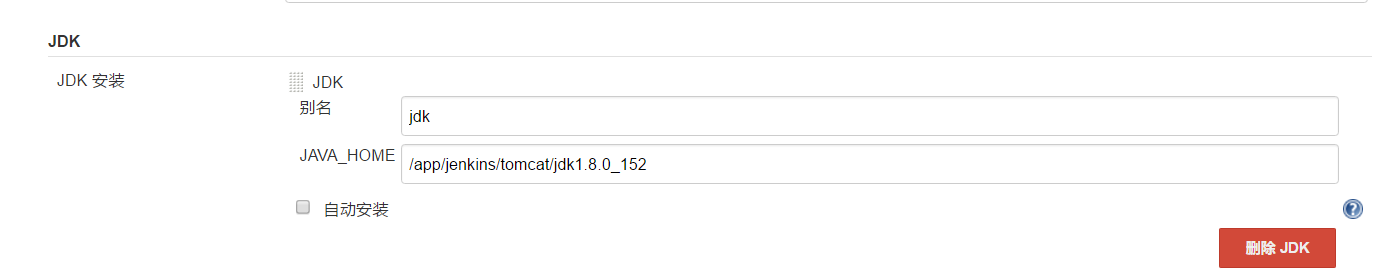
**解决方法：**

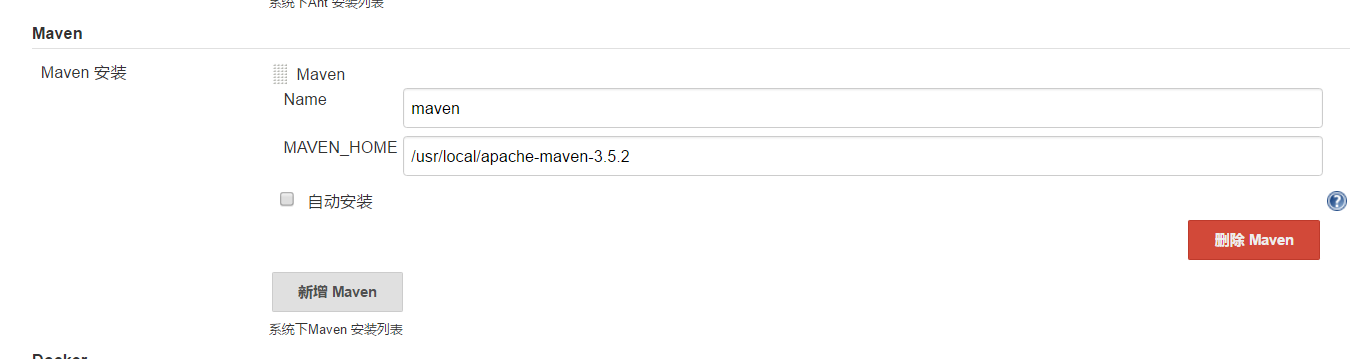
这是因为测试代码时遇到错误，它会停止编译。只需要在[pom.xml](http://www.javaw.net/)的<project>里添加以下配置，使得测试出错不影响项目的编译。

<build>
<plugins>
<plugin>
<groupId>org.apache.maven.plugins</groupId>
<artifactId>maven-surefire-plugin</artifactId>
<configuration>
<testFailureIgnore>true</testFailureIgnore> </configuration>
</plugin>
</plugins>
</build>

二部分

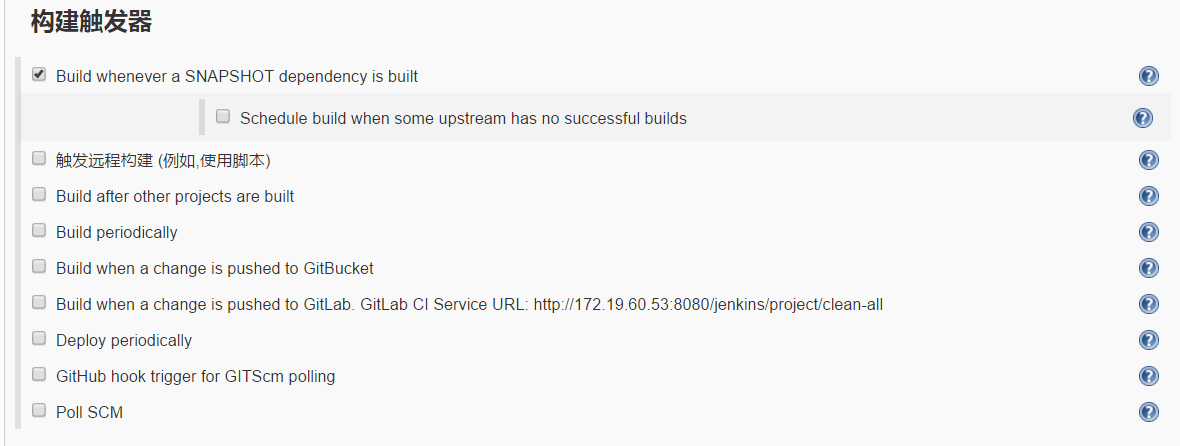
jenkins搭建完成之后，系统管理——Global Tool Configuration——选择已经安装好的，比如：JDK、ｍaven等，填写相关信息（前提是安装相关插件）



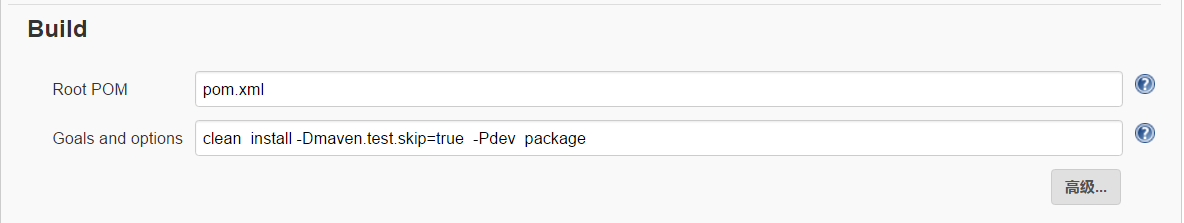


打开新建的项目，填写相关信息

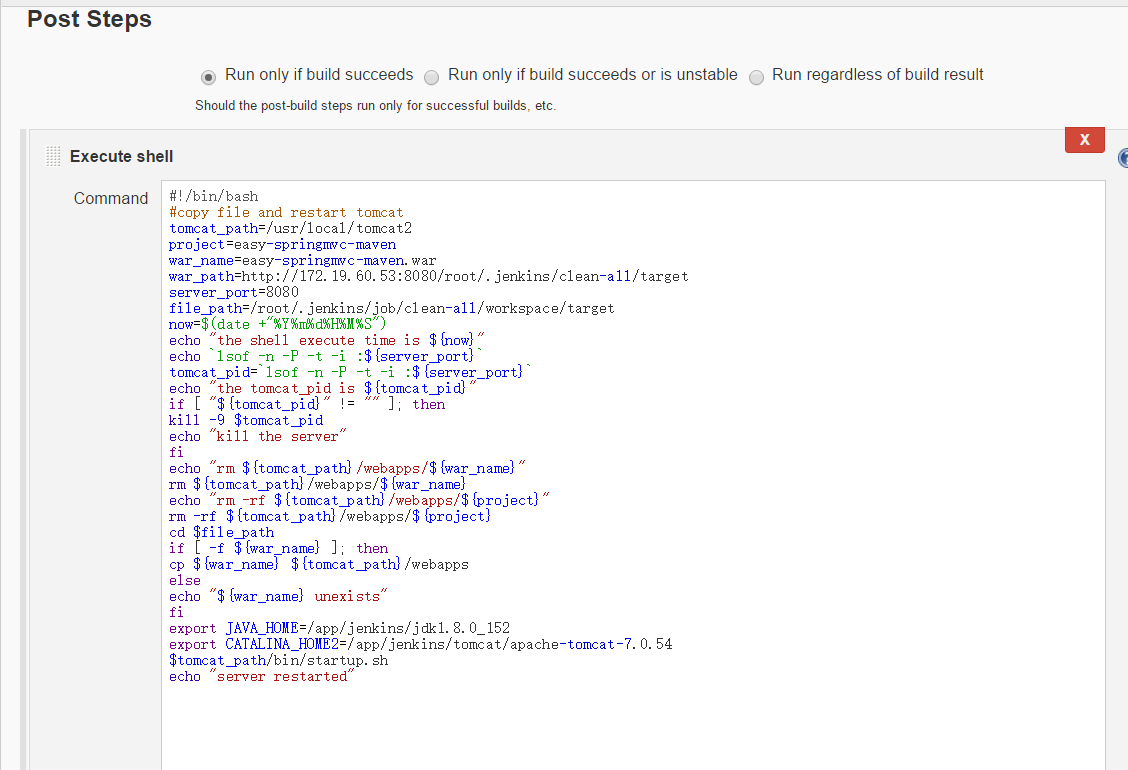
1.构建触发器



2.build



3.Post Steps



填写以下脚本

#!/bin/bash

#copy file and restart tomcat

tomcat\_path=/usr/local/tomcat2

project=easy-springmvc-maven

war\_name=easy-springmvc-maven.war

war\_path=http://172.19.60.53:8080/root/.jenkins/clean-all/target

server\_port=8080

file\_path=/root/.jenkins/job/clean-all/workspace/target

now=$(date +"%Y%m%d%H%M%S")

echo "the shell execute time is ${now}"

echo `lsof -n -P -t -i :${server\_port}`

tomcat\_pid=`lsof -n -P -t -i :${server\_port}`

echo "the tomcat\_pid is ${tomcat\_pid}"

if [ "${tomcat\_pid}" != "" ]; then

kill -9 $tomcat\_pid

echo "kill the server"

fi

echo "rm ${tomcat\_path}/webapps/${war\_name}"

rm ${tomcat\_path}/webapps/${war\_name}

echo "rm -rf ${tomcat\_path}/webapps/${project}"

rm -rf ${tomcat\_path}/webapps/${project}

cd $file\_path

if [ -f ${war\_name} ]; then

cp ${war\_name} ${tomcat\_path}/webapps

else

echo "${war\_name} unexists"

fi

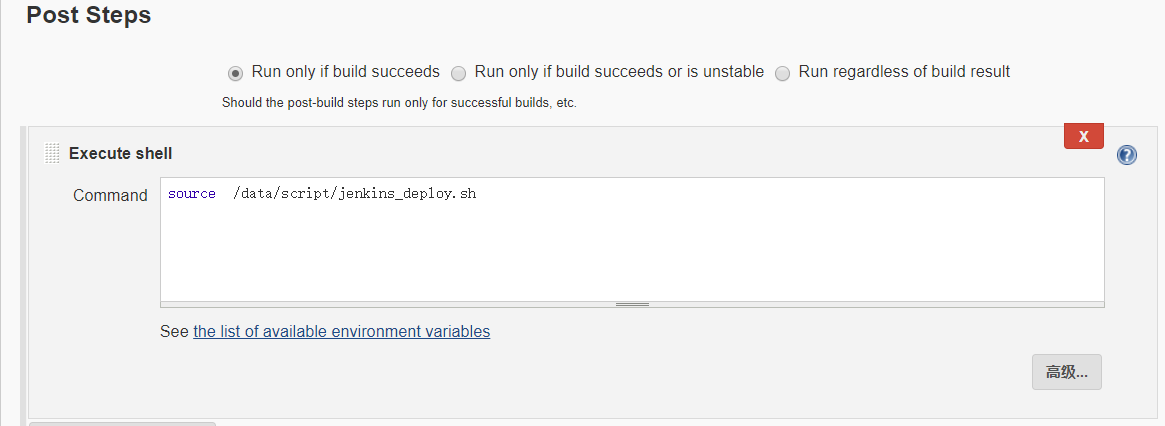
export JAVA\_HOME=/app/jenkins/jdk1.8.0\_152

export CATALINA\_HOME2=/app/jenkins/tomcat/apache-tomcat-7.0.54

$tomcat\_path/bin/startup.sh

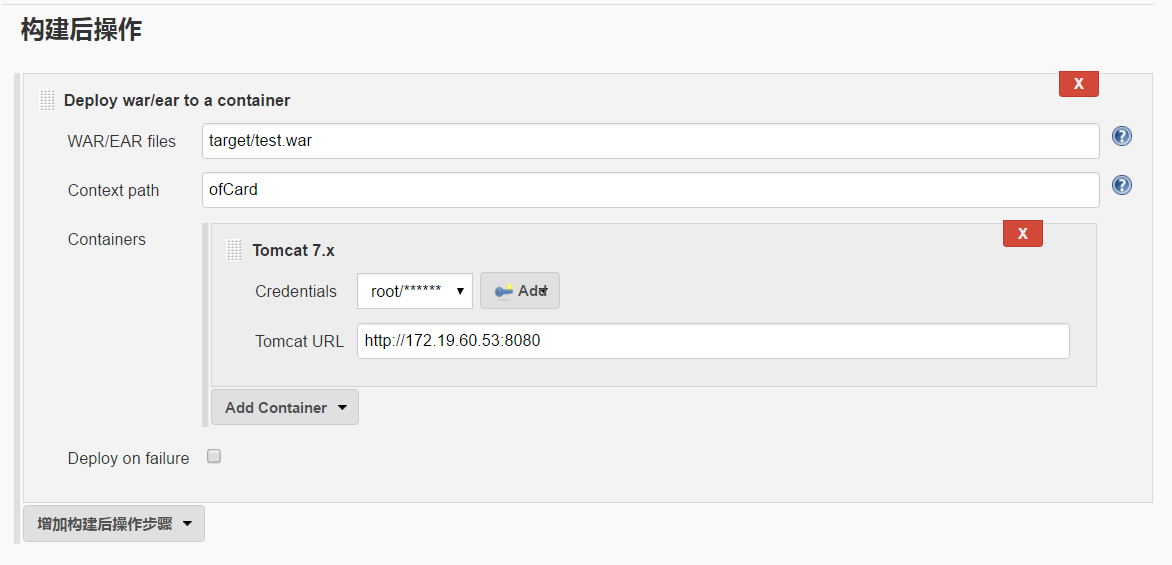
echo "server restarted"

**或者**



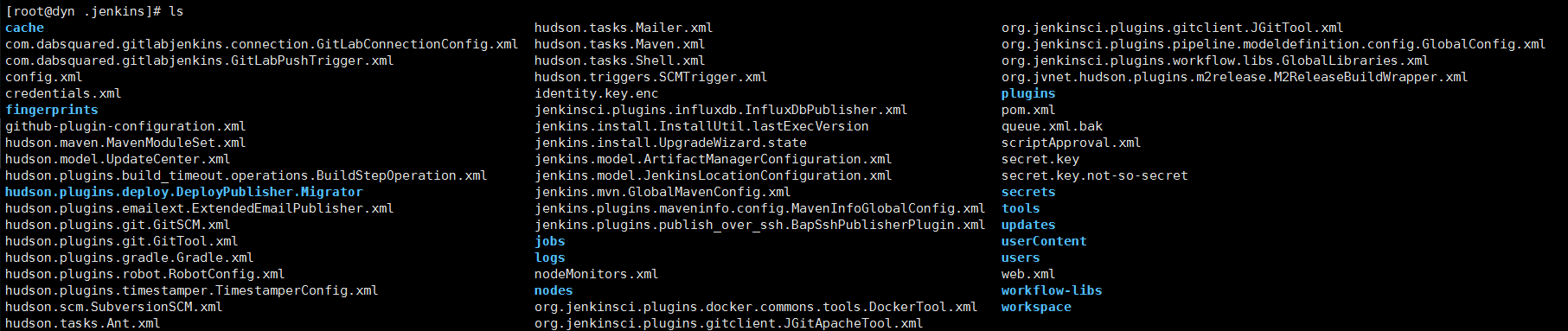
**我将jenkins部署的脚本放在了服务器/date/script/下**

4.构建后操作

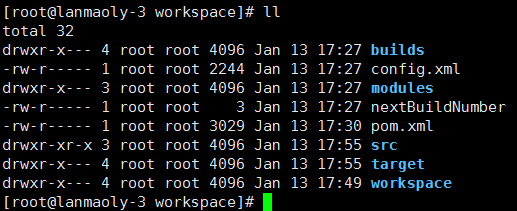


服务器配置源码相关信息

项目在服务器/root/.jenkins/下



cd /jobs/clean-all/workspace/



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

**进入到clean-all目录下，需要创建src/main/webapp/WEB-INF目录**

**touch web.xml文件，文件内容如下**

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

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

**xmlns="http://java.sun.com/xml/ns/javaee"**

**xmlns:web="http://java.sun.com/xml/ns/javaee/web-app\_2\_5.xsd"**

**xsi:schemaLocation="http://java.sun.com/xml/ns/javaee http://java.sun.com/xml/ns/javaee/web-app\_3\_0.xsd"**

**version="3.0">**

**<display-name>easy-springmvc-maven</display-name>**

**<!-- 上下文监听 -->**

**<listener>**

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

**</listener>**

**<!-- spring MVC的核心就是DispatcherServlet，使用springMVC的第一步就是将下面的servlet放入web.xml**

**servlet-name属性非常重要，默认情况下，DispatchServlet会加载这个名字-servlet.xml的文件，如下，就会加载**

**dispather-servlet.xml，也是在WEN-INF目录下。**

**-->**

**<servlet>**

**<servlet-name>dispatcher</servlet-name>**

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

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

**</servlet>**

**<!-- 设置dispatchservlet的匹配模式，通过把dispatchservlet映射到/，默认servlet会处理所有的请求，包括静态资源 -->**

**<servlet-mapping>**

**<servlet-name>dispatcher</servlet-name>**

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

**</servlet-mapping>**

**<!-- 字符集过滤器 -->**

**<filter>**

**<filter-name>encodingFilter</filter-name>**

**<filter-class>org.springframework.web.filter.CharacterEncodingFilter</filter-class>**

**<init-param>**

**<param-name>encoding</param-name>**

**<param-value>UTF-8</param-value>**

**</init-param>**

**<init-param>**

**<param-name>forceEncoding</param-name>**

**<param-value>true</param-value>**

**</init-param>**

**</filter>**

**<filter-mapping>**

**<filter-name>encodingFilter</filter-name>**

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

**</filter-mapping>**

**</web-app>**

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

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

**cd /root/.jenkins/jobs/clean-all/workspace/build**

**touch checkstyle.xml文件**

**内容如下:**

<?xml version="1.0"?>

<!DOCTYPE module PUBLIC

"-//Puppy Crawl//DTD Check Configuration 1.2//EN"

"http://www.puppycrawl.com/dtds/configuration\_1\_2.dtd">

<!--

Checkstyle configuration that checks the sun coding conventions from:

- the Java Language Specification at

http://java.sun.com/docs/books/jls/second\_edition/html/index.html

- the Sun Code Conventions at http://java.sun.com/docs/codeconv/

- the Javadoc guidelines at

http://java.sun.com/j2se/javadoc/writingdoccomments/index.html

- the JDK Api documentation http://java.sun.com/j2se/docs/api/index.html

- some best practices

Checkstyle is very configurable. Be sure to read the documentation at

http://checkstyle.sf.net (or in your downloaded distribution).

Most Checks are configurable, be sure to consult the documentation.

To completely disable a check, just comment it out or delete it from the file.

Finally, it is worth reading the documentation.

-->

<module name="Checker">

<!--

If you set the basedir property below, then all reported file

names will be relative to the specified directory. See

http://checkstyle.sourceforge.net/5.x/config.html#Checker

<property name="basedir" value="${basedir}"/>

-->

<!-- Checks whether files end with a new line. -->

<!-- See http://checkstyle.sf.net/config\_misc.html#NewlineAtEndOfFile -->

<!-- <module name="NewlineAtEndOfFile"/> -->

<!-- Checks that property files contain the same keys. -->

<!-- See http://checkstyle.sf.net/config\_misc.html#Translation -->

<module name="Translation"/>

<module name="FileLength"/>

<!-- Following interprets the header file as regular expressions. -->

<!-- <module name="RegexpHeader"/> -->

<!-- <module name="FileTabCharacter">

<property name="eachLine" value="true"/>

</module> -->

<!-- <module name="RegexpSingleline">

\s matches whitespace character, $ matches end of line.

<property name="format" value="\s+$"/>

<property name="message" value="Line has trailing spaces."/>

</module> -->

<module name="TreeWalker">

<property name="cacheFile" value="${checkstyle.cache.file}"/>

<!-- Checks for Javadoc comments. -->

<!-- See http://checkstyle.sf.net/config\_javadoc.html -->

<!-- <module name="JavadocMethod"/> -->

<!-- <module name="JavadocType"/> -->

<!-- <module name="JavadocVariable"/> -->

<!-- <module name="JavadocStyle"/> -->

<!-- Checks for Naming Conventions. -->

<!-- See http://checkstyle.sf.net/config\_naming.html -->

<module name="ConstantName"/>

<module name="LocalFinalVariableName"/>

<module name="LocalVariableName"/>

<module name="MemberName"/>

<module name="MethodName"/>

<module name="PackageName"/>

<module name="ParameterName"/>

<module name="StaticVariableName"/>

<module name="TypeName"/>

<!-- Checks for Headers -->

<!-- See http://checkstyle.sf.net/config\_header.html -->

<!-- <module name="Header"> -->

<!-- The follow property value demonstrates the ability -->

<!-- to have access to ANT properties. In this case it uses -->

<!-- the ${basedir} property to allow Checkstyle to be run -->

<!-- from any directory within a project. See property -->

<!-- expansion, -->

<!-- http://checkstyle.sf.net/config.html#properties -->

<!-- <property -->

<!-- name="headerFile" -->

<!-- value="${basedir}/java.header"/> -->

<!-- </module> -->

<!-- Checks for imports -->

<!-- See http://checkstyle.sf.net/config\_import.html -->

<module name="AvoidStarImport"/>

<module name="IllegalImport"/> <!-- defaults to sun.\* packages -->

<module name="RedundantImport"/>

<module name="UnusedImports"/>

<!-- Checks for Size Violations. -->

<!-- See http://checkstyle.sf.net/config\_sizes.html -->

<!-- <module name="LineLength"/> -->

<module name="MethodLength"/>

<module name="ParameterNumber"/>

<!-- Checks for whitespace -->

<!-- See http://checkstyle.sf.net/config\_whitespace.html -->

<module name="EmptyForIteratorPad"/>

<module name="MethodParamPad"/>

<module name="NoWhitespaceAfter"/>

<module name="NoWhitespaceBefore"/>

<module name="OperatorWrap"/>

<module name="ParenPad"/>

<module name="TypecastParenPad"/>

<module name="WhitespaceAfter"/>

<module name="WhitespaceAround"/>

<!-- Modifier Checks -->

<!-- See http://checkstyle.sf.net/config\_modifiers.html -->

<module name="ModifierOrder"/>

<module name="RedundantModifier"/>

<!-- Checks for blocks. You know, those {}'s -->

<!-- See http://checkstyle.sf.net/config\_blocks.html -->

<module name="AvoidNestedBlocks"/>

<!-- <module name="EmptyBlock"/> -->

<module name="LeftCurly"/>

<module name="NeedBraces"/>

<module name="RightCurly"/>

<!-- Checks for common coding problems -->

<!-- See http://checkstyle.sf.net/config\_coding.html -->

<!-- <module name="AvoidInlineConditionals"/> -->

<module name="DoubleCheckedLocking"/> <!-- MY FAVOURITE -->

<module name="EmptyStatement"/>

<module name="EqualsHashCode"/>

<!-- <module name="HiddenField"/> -->

<module name="IllegalInstantiation"/>

<module name="InnerAssignment"/>

<!-- <module name="MagicNumber"/> -->

<module name="MissingSwitchDefault"/>

<module name="RedundantThrows"/>

<module name="SimplifyBooleanExpression"/>

<module name="SimplifyBooleanReturn"/>

<!-- Checks for class design -->

<!-- See http://checkstyle.sf.net/config\_design.html -->

<!-- <module name="DesignForExtension"/> -->

<module name="FinalClass"/>

<module name="HideUtilityClassConstructor"/>

<module name="InterfaceIsType"/>

<!-- <module name="VisibilityModifier"/> -->

<!-- Miscellaneous other checks. -->

<!-- See http://checkstyle.sf.net/config\_misc.html -->

<module name="ArrayTypeStyle"/>

<!-- <module name="FinalParameters"/> -->

<module name="TodoComment"/>

<module name="UpperEll"/>

</module>

</module>

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

配置pom.xml，在/root/.jenkins/jobs/clean-all/workspace/下

[root@dyn clean-all]# cat pom.xml

<project xmlns="http://maven.apache.org/POM/4.0.0" xmlns:xsi="http://www.w3.org/2001/XMLSchema-instance" xsi:schemaLocation="http://maven.apache.org/POM/4.0.0 http://maven.apache.org/xsd/maven-4.0.0.xsd">

<modelVersion>4.0.0</modelVersion>

<groupId>springmvc-maven</groupId>

<artifactId>easy-springmvc-maven</artifactId>

<version>0.0.1-SNAPSHOT</version>

<packaging>war</packaging>

<name>springmvc-maven</name>

<description>simple demo about how to use maven combine spring mvc</description>

<build>

<!-- 使用默认的文件路径 -->

<!-- 生成的war文件名 避免添加版本号 -->

<finalName>easy-springmvc-maven</finalName>

<plugins>

<plugin>

<artifactId>maven-compiler-plugin</artifactId>

<version>3.1</version>

<configuration>

<source>1.6</source>

<target>1.6</target>

</configuration>

</plugin>

<plugin>

<artifactId>maven-war-plugin</artifactId>

<configuration>

<version>3.0 </version>

<!-- maven的web项目默认的webroot是在src\main\webapp。如果在此目录下找不到web.xml就抛出webxml attribute is required的异常 -->

<!-- 需要在pom.xml中增加<webXml>配置 -->

<!-- <webXml>WebContent\WEB-INF\web.xml</webXml> -->

</configuration>

</plugin>

</plugins>

</build>

<dependencies>

<dependency>

<groupId>junit</groupId>

<artifactId>junit</artifactId>

<version>4.12</version>

<scope>test</scope>

</dependency>

<!-- spring mvc dependencies start -->

<dependency>

<groupId>org.springframework</groupId>

<artifactId>spring-webmvc</artifactId>

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

</dependency>

<dependency>

<groupId>org.springframework</groupId>

<artifactId>spring-jdbc</artifactId>

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

</dependency>

<dependency>

<groupId>org.springframework</groupId>

<artifactId>spring-context</artifactId>

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

</dependency>

<dependency>

<groupId>org.springframework</groupId>

<artifactId>spring-aop</artifactId>

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

</dependency>

<dependency>

<groupId>org.springframework</groupId>

<artifactId>spring-core</artifactId>

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

</dependency>

<dependency>

<groupId>org.springframework</groupId>

<artifactId>spring-test</artifactId>

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

</dependency>

<!-- spring mvc dependencies end -->

<!-- 解决页面访问时发生错误: java.lang.NoClassDefFoundError: javax/servlet/jsp/jstl/core/Config - start -->

<dependency>

<groupId>jstl</groupId>

<artifactId>jstl</artifactId>

<version>1.2</version>

</dependency>

<dependency>

<groupId>taglibs</groupId>

<artifactId>standard</artifactId>

<version>1.1.2</version>

</dependency>

<!-- end -->

</dependencies>

<properties>

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

<spring.version>3.1.2.RELEASE</spring.version>

</properties>

</properties>

切换到/root/.jenkins/workspace/claen-all/src/main/webapp/WEB-INF

配置web.xml

[root@dyn WEB-INF]# cat web.xml

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

<web-app xmlns:xsi="http://www.w3.org/2001/XMLSchema-instance"

xmlns="http://java.sun.com/xml/ns/javaee"

xmlns:web="http://java.sun.com/xml/ns/javaee/web-app\_2\_5.xsd"

xsi:schemaLocation="http://java.sun.com/xml/ns/javaee http://java.sun.com/xml/ns/javaee/web-app\_3\_0.xsd"

version="3.0">

<display-name>easy-springmvc-maven</display-name>

<!-- 上下文监听 -->

<listener>

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

</listener>

<!-- spring MVC的核心就是DispatcherServlet，使用springMVC的第一步就是将下面的servlet放入web.xml

servlet-name属性非常重要，默认情况下，DispatchServlet会加载这个名字-servlet.xml的文件，如下，就会加载

dispather-servlet.xml，也是在WEN-INF目录下。

-->

<servlet>

<servlet-name>dispatcher</servlet-name>

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

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

</servlet>

<!-- 设置dispatchservlet的匹配模式，通过把dispatchservlet映射到/，默认servlet会处理所有的请求，包括静态资源 -->

<servlet-mapping>

<servlet-name>dispatcher</servlet-name>

<url-pattern>/</url-pattern>

</servlet-mapping>

<!-- 字符集过滤器 -->

<filter>

<filter-name>encodingFilter</filter-name>

<filter-class>org.springframework.web.filter.CharacterEncodingFilter</filter-class>

<init-param>

<param-name>encoding</param-name>

<param-value>UTF-8</param-value>

</init-param>

<init-param>

<param-name>forceEncoding</param-name>

<param-value>true</param-value>

</init-param>

</filter>

<filter-mapping>

<filter-name>encodingFilter</filter-name>

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

</filter-mapping>

</web-app>

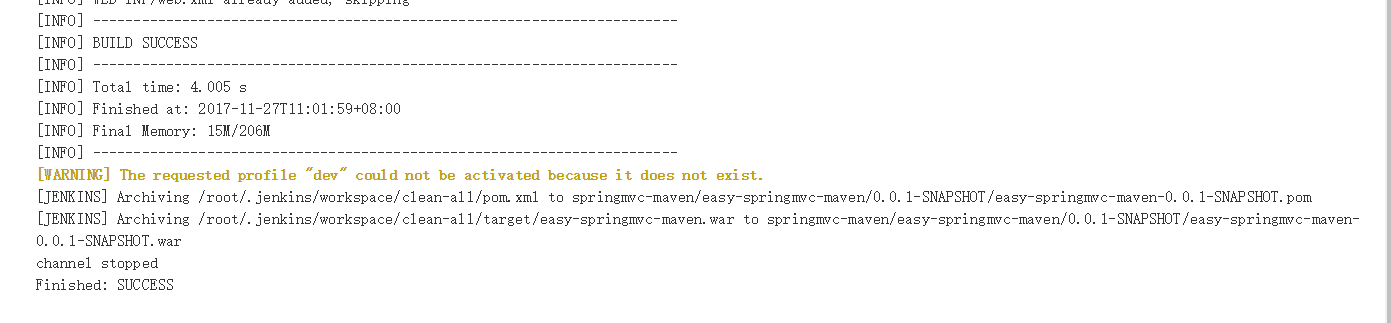
以上步骤做完之后，进入/root/.jenkins/workspace/clean-all下

#mvn clean

#mvn install

\*\*\*\*\*\*\*不报错就好\*\*\*\*\*\*显示successd\*\*\*\*\*

至此，服务器配置完成一半，进入到网页jenkins，构建刚创建的项目



上图是构建最终结果

详细关于pom文件与web文件的信息，可以去看这个<https://github.com/bingyue/easy-springmvc-maven>

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

关于脚本部署，在/data/script/

vim jenkins\_deploy.sh

#!/bin/bash

#\*\*\*\*\*\*\*\*\*\*\*\*jenkins部署项目脚本\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*

#jenkins部署war包到tomcat容器下，重新启动tomcat，包括war包重命名、删除、备份等

TOMCAT\_HOME="/app/tomcat/tomcat-clean-all"

TOMCAT\_PORT="8080"

PROJECT="clean-all"

jenkins\_workspace="/root/.jenkins/jobs/clean-all/workspace/"

WAR\_HOME="/root/.jenkins/jobs/clean-all/workspace/target"

cd $WAR\_HOME

#mv \*.war $PROJECT.war

#停止tomcat

#cd /app/tomcat/tomcat-clean-all/bin

#./shutdown.sh &

#检查tomcat进程

sleep 5

tomcat\_pid=`ps -ef | grep tomccat\_$PROJECT | grep start | grep -v 'grep'|awk '{print $2}'`

while [ -n "$tomcat\_pid" ]

do

kill -9 $tomcat\_pid

sleep 3

tomcat\_pid=`ps -ef |grep tomccat\_$PROJECT | grep start | grep -v 'grep'|awk '{print $2}'`

echo "scan tomcat pid :"$tomcat\_pid

done

#bak project

echo "scan no tomcat pid,$PROJECT publishing"

#tar -czf $WAR\_HOME/bak/$PROJECT-bak`date +%H%M%S`.tar.gz $TOMCAT\_HOME/webapps/$PROJECT

sleep 10

rm -rf $TOMCAT\_HOME/webapps/$PROJECT $TOMCAT\_HOME/webapps/$PROJECT.war

cp $WAR\_HOME/$PROJECT.war $TOMCAT\_HOME/webapps/$PROJECT.war

#remove tmp

rm -rf $WAR\_HOME/$PROJECT.war

#启动tomcat

cd /app/tomcat/tomcat-clean-all/bin/

./startup &

echo "tomcat is starting,please try to access $PROJECT conslone url"

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

**三部分**

**安装git、gitlab插件**

**用浏览器打开Jenkins，点击“系统设置”**

612215036308.png

这里写图片描述

**点击“插件管理”**



这里写图片描述

**切换到“可选插件”，分别搜索“GitLab Plugin”和“Git Plugin”,然后点击“直接安装”。如果在“可选插件”里没有搜到，可能默认你已经安装了，可以在“已安装”里查看**



这里写图片描述

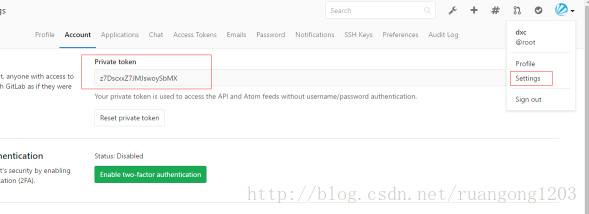
**安装完会显示：**



这里写图片描述

**配置GitLab插件**

**打开GitLab，点击“setting”——“Account”，复制“Private token”备用，如下所示：**



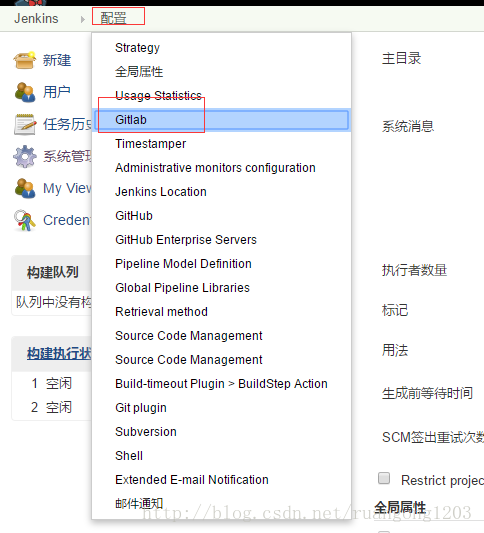
这里写图片描述

**打开Jenkins，点击“系统管理”——“系统设置”，如下所示：**



这里写图片描述

**点击“配置”下拉框，点击“Gitlab”选项，如下所示：**



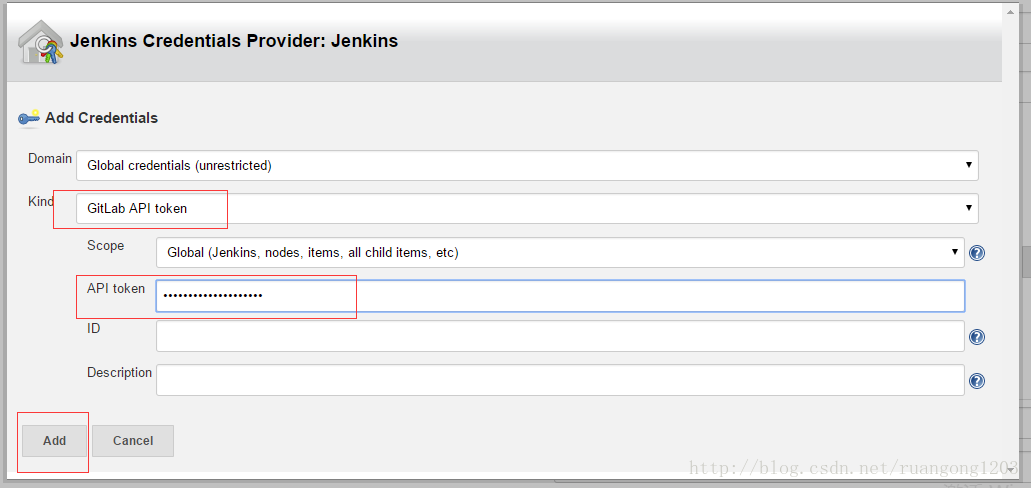
这里写图片描述

**配置GitLab，”Connection Name”随便填，“Git Host URL”填GitLab的访问地址，然后点“Add”——“jenkins”，如下所示：**



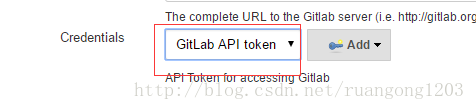
这里写图片描述

**在弹出框里，“kind”选择“GitLab API Token”，将先前复制的“Private token”粘贴到“API token”输入框中，然后点击“Add”，如下所示：**



这里写图片描述

**选择刚刚新建的“Credentials”，如下所示：**



这里写图片描述

**设置完了，要测试一下能否连接成功，点击“test connection”,要看到返回“Success”才行，如下所示：**



这里写图片描述

**然后点击页面底下的“apply”,再点击“save”，如下所示：**

612222829577.png

这里写图片描述

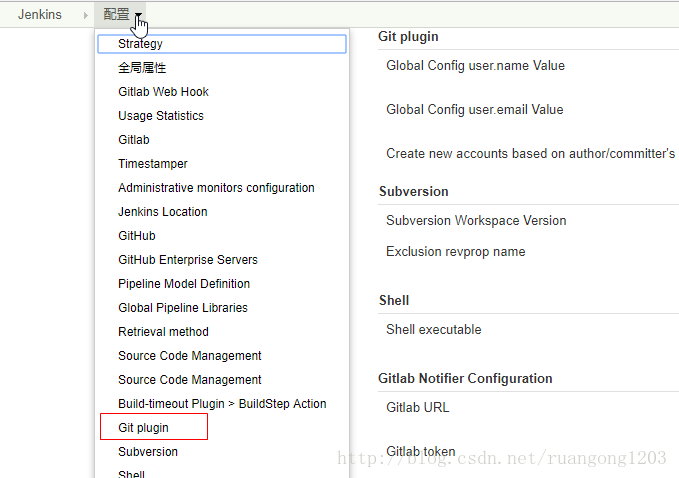
**配置Git插件**

**打开Jenkins，点击“系统管理”——“系统设置”**



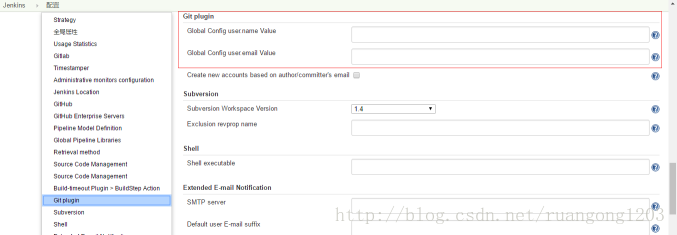
这里写图片描述

**点击“配置”下拉框，选择“Git plugin”选项**



这里写图片描述

**设置Git插件的全局配置，然后点击“apply”——“save”**

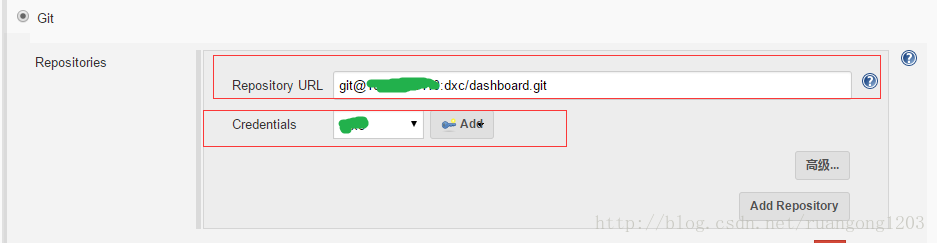


这里写图片描述

**全局配置就是上文“Gitlab创建项目”部分讲到的global setting ，在项目第一次commit前，这些信息都可以在GitLab的项目的首页里找到**

**配置Job的源码管理**

**选择“源码管理”，选择“Git”,然后去GitLab中复制项目地址，粘贴到“Repository URL”,然后点击“credentials”后面的“Add”按钮**



这里写图片描述

**在弹出页面里面：**

**Kind 选择 SSH Username with private key**

**Username 填 root**

**PrivateKey 选择 From a file on jenkins master ，然后将服务器的 私钥的存放路径（私钥、私钥、私钥，再说三遍） 粘贴进去**

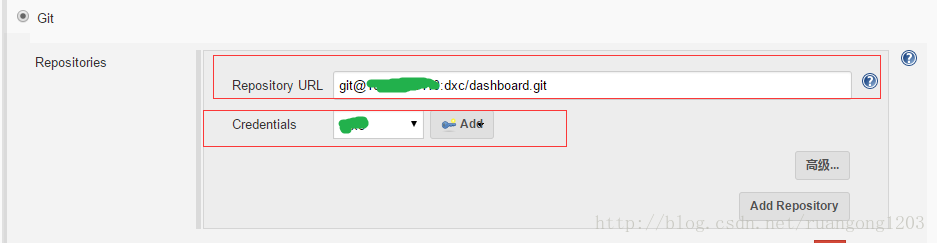
**passphrase 填创建SSH秘钥时的设置的密码，未设置可不填**

**然后点击“Add”**



这里写图片描述

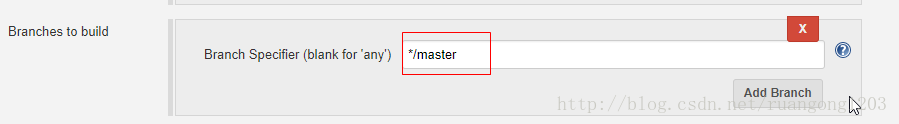
**在“credentials”里选择我们刚刚创建的认证方式：**



这里写图片描述

**如果没报错，说明成功了，点击页面底部的“apply”。如果出错了，会在“Repository URL”和“Credentials”之间显示红色的错误信息。**

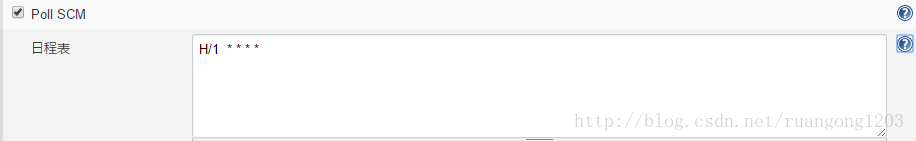
**jenkins job默认对master分支进行构建，你也可以自定义分支。这要求你的Gitlab代码仓库中要存在这个分支，一般来说，就是要向代码仓库提交一次更改，请 自行完成（Gitlab项目刚创建时是空的，一个分支也没有，这样的话，自动构建时会出错）**



这里写图片描述

**配置Job的构建触发器**

**选择“构建触发器”，勾选“Pull SCM”，这个选项会每隔一段时间检查一下GitLab仓库中代码是否有更新，有的话就执行构建操作。日程表如何设置，在这个输入框下面有说明。**



这里写图片描述

**扩展阅读：**

**常见构建触发器、：**

**Build after other projects are built 当另一个构建任务完成之后触发**

**Build periodically 周期性的触发**

**Build when a change is pushed to GitLab. GitLab CI Service URL:**[**http://191.8.2.112:12000/project/test-go-dev**](http://191.8.2.112:12000/project/test-go-dev)**当代码有更新的时候触发，通过GitLab CI**

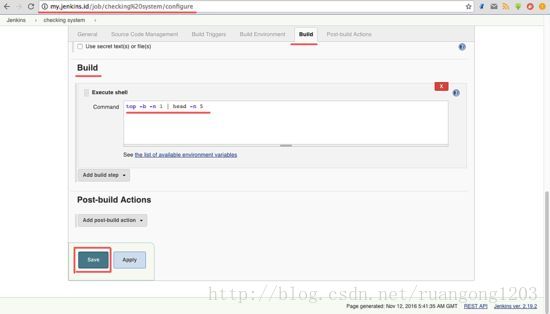
**GitHub hook trigger for GITScm polling 通过Github钩子触发**

**Poll SCM 定期检查代码有无更新，有更新时触发**

**这只是个人理解，具体怎么样大家可以试试，Poll SCM方式我是试过的。**

**配置Job的构建脚本**

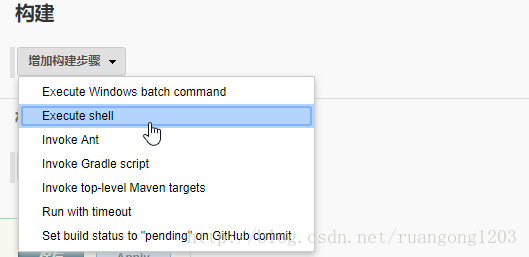
**在build栏目里，选择“jenkins execute shell”,然后输入你项目的构建命令（这依赖于你的项目，如Maven的maven build，gulp的gulp xxx 等等）**



这里写图片描述

**扩展阅读：**

**jenkins支持多种构建脚本，可以自己试一下：**



这里写图片描述

**后记**

**至此，所有工作已经完成，现在你提交代码到GitLab，jenkins会每隔一段时间帮你运行一次构建命令，这样大家的代码自动集成到一起，出了错的话很快就知道了。**

**其它问题**

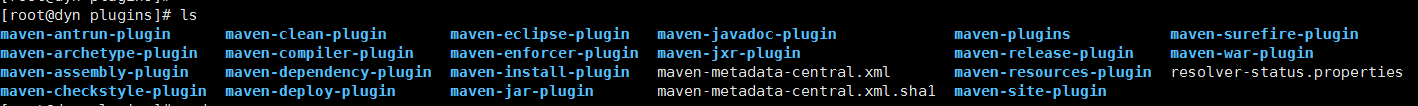
**每次jenkins运行任务的时候，都会到gitlab上拉取到最新的代码，放在workspace下。需要注意的是，默认拉取的.sh脚本文件是没有执行权限的，需要在构建命令里面添加：**

**chmod +x \*.sh**

**接下来jenkins整合maven来进行清理、编译、测试、打包、集成测试、验证、部署等步骤；**

maven的生命周期是抽象的，具体的任务通过插件来运行

定位到/root/.m2/repository/org/apache/maven/plugins，可以看到一些下载好的插件



**maven入门指南**

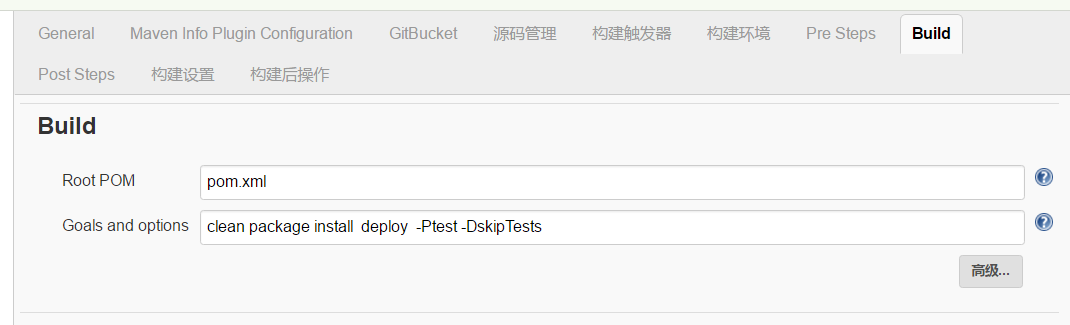
<http://www.cnblogs.com/luotaoyeah/p/3819001.html>

**jenkins项目——配置**

**build：**

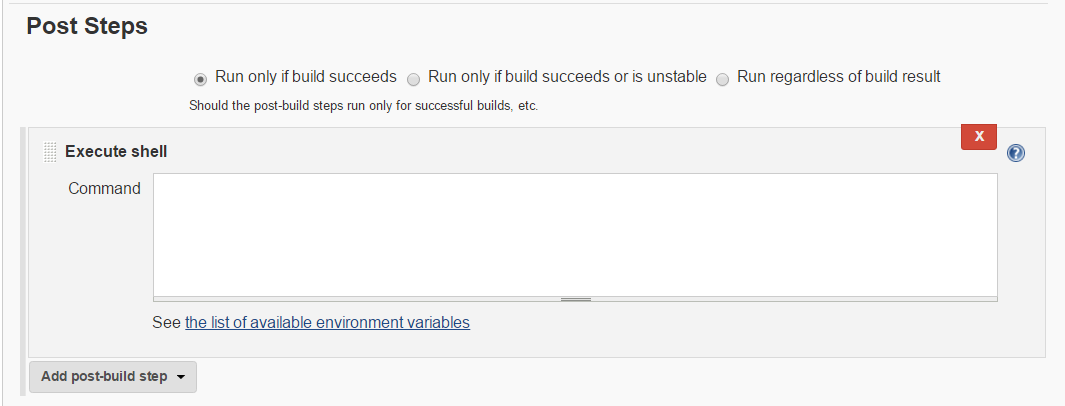
**定义maven下pom文件**

**定义maven插件参数**

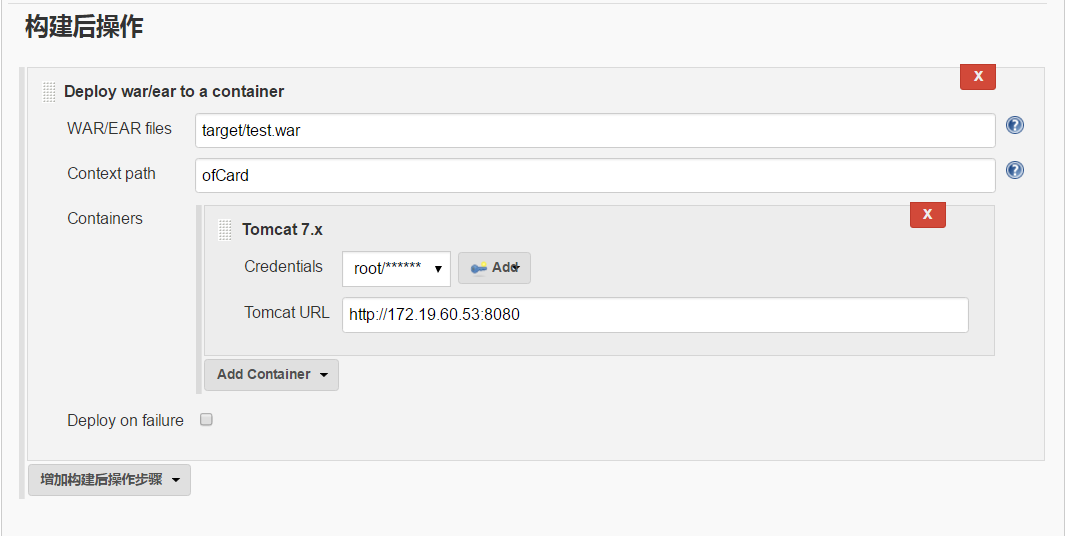


Post Steps

选择Exexute shell（可以自己写些脚本来执行）



构建后操作

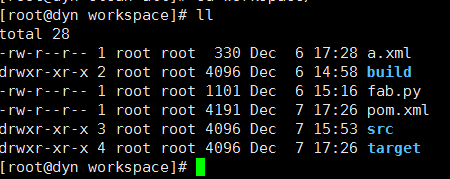


**详情请查看**

**jenkins的由来**

[**https://www.cnblogs.com/guixia621/p/7071009.html**](https://www.cnblogs.com/guixia621/p/7071009.html)

默认是在/root/.jenkins/job/clean-all/workspace/



在项目目录下，执行以下参数语句

mvn archetype:create -DgroupId=helloworld -DartifactId=helloworld

**这里利用的nexus，详情去网上查看mvn deploy远程部署“就可以了**

接下来去看下服务器上的pom文件

**pom.xml**

**<project xmlns="http://maven.apache.org/POM/4.0.0" xmlns:xsi="http://www.w3.org/2001/XMLSchema-instance" xsi:schemaLocation="http://maven.apache.org/POM/4.0.0 http://maven.apache.org/xsd/maven-4.0.0.xsd">**

**<modelVersion>4.0.0</modelVersion>**

**<groupId>springmvc-maven</groupId>**

**<artifactId>easy-maven</artifactId>**

**<version>0.0.1-SNAPSHOT</version>**

**<packaging>war</packaging>**

**<name>maven</name>**

**<description>simple demo about how to use maven combine spring mvc</description>**

**<build>**

**<!-- 使用默认的文件路径 -->**

**<!-- 生成的war文件名 避免添加版本号 -->**

**<finalName>easy-maven</finalName>**

**<plugins>**

**<plugin>**

**<artifactId>maven-compiler-plugin</artifactId>**

**<version>3.1</version>**

**<configuration>**

**<source>1.6</source>**

**<target>1.6</target>**

**</configuration>**

**</plugin>**

**<!--指定web.xml的位置，在pom.xml中加入 -->**

**<plugin>**

**<artifactId>maven-war-plugin</artifactId>**

**<configuration>**

**<version>3.0</version>**

**<webXml>src/mywebapp/src/main/webapp/WEB-INF/web.xml</webXml><!-- 这里指定位置 -->**

**</configuration>**

**</plugin>**

**<plugin>**

**<artifactId>maven-war-plugin</artifactId>**

**<configuration>**

**<version>3.0 </version>**

**<!-- maven的web项目默认的webroot是在src\main\webapp。如果在此目录下找不到web.xml就抛出webxml attribute is required的异常 -->**

**<!-- 需要在pom.xml中增加<webXml>配置 -->**

**<!-- <webXml>WebContent\WEB-INF\web.xml</webXml> -->**

**</configuration>**

**</plugin>**

**</plugins>**

**</build>**

**<dependencies>**

**<dependency>**

**<groupId>junit</groupId>**

**<artifactId>junit</artifactId>**

**<version>4.12</version>**

**<scope>test</scope>**

**</dependency>**

**<!-- spring mvc dependencies start -->**

**<dependency>**

**<groupId>org.springframework</groupId>**

**<artifactId>spring-webmvc</artifactId>**

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

**</dependency>**

**<dependency>**

**<groupId>org.springframework</groupId>**

**<artifactId>spring-jdbc</artifactId>**

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

**</dependency>**

**<dependency>**

**<groupId>org.springframework</groupId>**

**<artifactId>spring-context</artifactId>**

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

**</dependency>**

**<dependency>**

**<groupId>org.springframework</groupId>**

**<artifactId>spring-aop</artifactId>**

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

**</dependency>**

**<dependency>**

**<groupId>org.springframework</groupId>**

**<artifactId>spring-core</artifactId>**

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

**</dependency>**

**<dependency>**

**<groupId>org.springframework</groupId>**

**<artifactId>spring-test</artifactId>**

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

**</dependency>**

**<!-- spring mvc dependencies end -->**

**<!-- 解决页面访问时发生错误: java.lang.NoClassDefFoundError: javax/servlet/jsp/jstl/core/Config - start -->**

**<dependency>**

**<groupId>jstl</groupId>**

**<artifactId>jstl</artifactId>**

**<version>1.2</version>**

**</dependency>**

**<dependency>**

**<groupId>taglibs</groupId>**

**<artifactId>standard</artifactId>**

**<version>1.1.2</version>**

**</dependency>**

**<!-- end -->**

**</dependencies>**

**<properties>**

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

**<spring.version>3.1.2.RELEASE</spring.version>**

**<maven.checkstyle.version>2.17</maven.checkstyle.version>**

**</properties>**

**<reporting>**

**<plugins>**

**<plugin>**

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

**<artifactId>maven-checkstyle-plugin</artifactId>**

**<version>${maven.checkstyle.version}</version>**

**</plugin>**

**<plugin>**

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

**<artifactId>maven-deploy-plugin</artifactId>**

**<version>2.7</version>**

**</plugin>**

**</plugins>**

**</reporting>**

**<!--利用deploy将软件包部署到远程仓库 -->**

**<distributionManagement>**

**<repository>**

**<id>release</id>**

**<url>http://172.19.60.53:8081/nexus/content/repositories/release</url>**

**</repository>**

**<snapshotRepository>**

**<id>snapshots1</id>**

**<url>http://172.19.60.53:8081/nexus/content/repositories/snapshots1</url>**

**</snapshotRepository>**

**</distributionManagement>**

**</project>**

**然后配置maven的settings.xml**

**[root@dyn conf]# cat settings.xml**

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

**<settings xmlns="http://maven.apache.org/SETTINGS/1.0.0"**

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

**xsi:schemaLocation="http://maven.apache.org/SETTINGS/1.0.0 http://maven.apache.org/xsd/settings-1.0.0.xsd">**

**<localRepository>/usr/local/apache-maven-3.5.2</localRepository>**

**<!-- 在settings.xml中分别为上面配置的部署仓库配置server，其中id需要分别对应上面的部署仓库id -->**

**<servers>**

**<server>**

**<id>release</id>**

**<username>deployment</username>**

**<password>deployment</password>**

**<filePermissions>664</filePermissions>**

**<directoryPermissions>775</directoryPermissions>**

**<configuration></configuration>**

**</server>**

**<server>**

**<id>snapshots1</id>**

**<username>deployment</username>**

**<password>deployment</password>**

**<filePermissions>664</filePermissions>**

**<directoryPermissions>775</directoryPermissions>**

**<configuration></configuration>**

**</server>**

**</servers>**

**</settings>**

**接下来，在服务器上测试下**

**mvn clean install package deploy**

**运行完之后，显示success则ok**