# Jenkins安装部署与自动化部署配置手册

## 一、安装部署

### Jenkins版本和jdk对应关系

本文安装使用2.346.1版本，支持jdk8

***2.361.1 (September 2022) and newer***

Java 11 or Java 17

***2.346.1 (June 2022) and newer***

Java 8, Java 11, or Java 17

***2.164.1 (March 2019) and newer***

Java 8 or Java 11

***2.60.1 (June 2017) and newer***

Java 8

***1.625.1 (October 2015) and newer***

Java 7

### 在线安装（联网）

wget -q -O - https://pkg.jenkins.io/debian/jenkins.io.key | sudo apt-key add -

sudo sh -c 'echo deb http://pkg.jenkins.io/debian-stable binary/ > /etc/apt/sources.list.d/jenkins.list'

sudo apt-get update

sudo apt-get install jenkins

#注意： 这种安装方式默认安装最新版本，需要支持JDK11版本以上

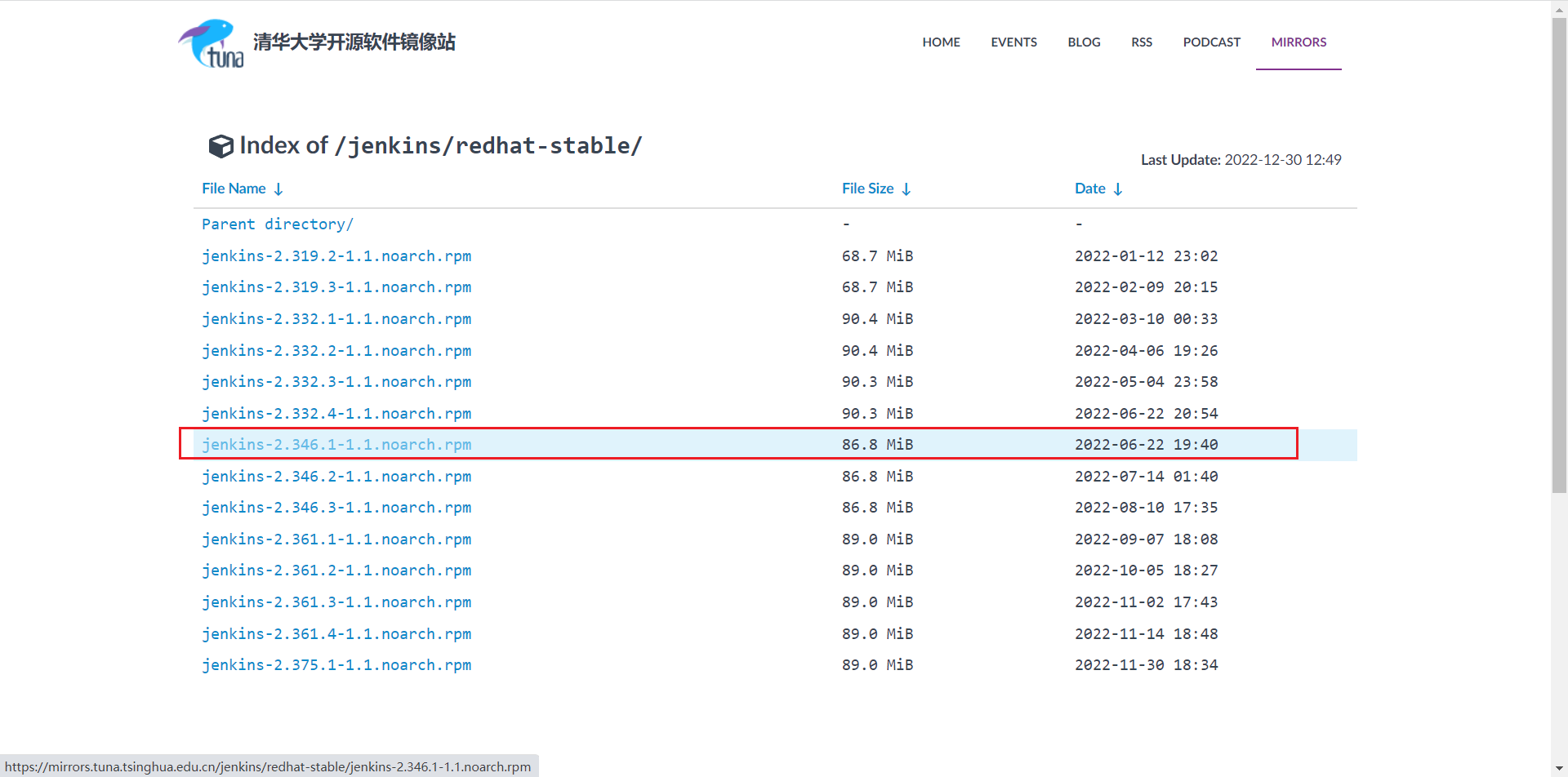
#本文使用离线安装方式，JDK版本8

### 离线安装包下载地址

下载地址：[https://jenkins.io/zh/download/](https://links.jianshu.com/go?to=https://jenkins.io/zh/download/" \t "https://www.jianshu.com/p/_blank)或者

<https://mirrors.tuna.tsinghua.edu.cn/jenkins/redhat-stable/>

选择支持JDK8版本的2.346.1-1.1



### 离线安装

1）Ubuntu使用dpkg安装

[Ubuntu](https://so.csdn.net/so/search?q=Ubuntu&spm=1001.2101.3001.7020" \t "https://blog.csdn.net/suleil1/article/details/_blank)的软件包格式是deb，如果要安装rpm的包，则要先用alien把rpm转换成deb

sudo apt-get install alien

#alien默认没有安装，所以首先要安装它

sudo alien jenkins-2.346.1-1.1.noarch.rpm

#将rpm转换位deb，完成后会生成一个同名的xxxx.deb

Sudo dpkg -i jenkins\_2.346.1-2.1\_all.deb #安装

### 配置

1. 检查/usr/bin/java下是否链接正确

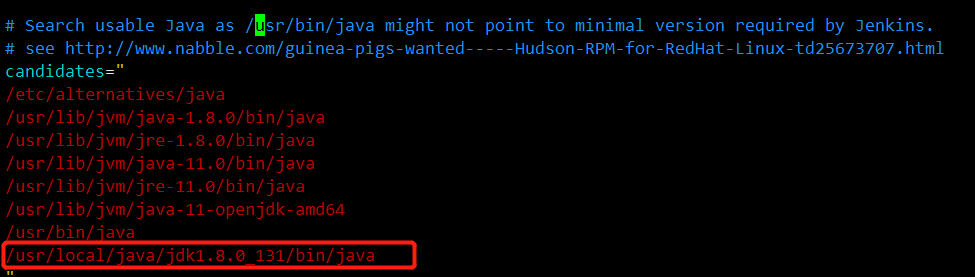


#软连接的地址必须与配置Jenkins的Java地址一致，否则会导致启动失败

ln -s /usr/local/java/jdk1.8.0\_131/bin/java /usr/bin/java

2）修改配置文件/etc/init.d/jenkins

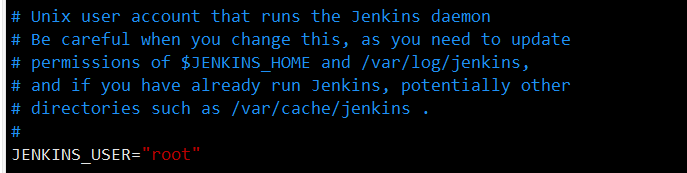
vim /etc/init.d/jenkins添加java路径



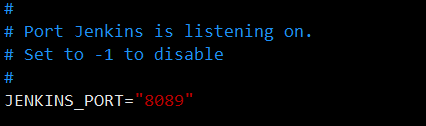
3）修改文件/etc/sysconfig/jenkins

vim /etc/sysconfig/jenkins

JENKINS\_USER="root"，如果不是root，而是jenkins或者其它，要改成root



默认8080端口被占用，则需要修改端口号



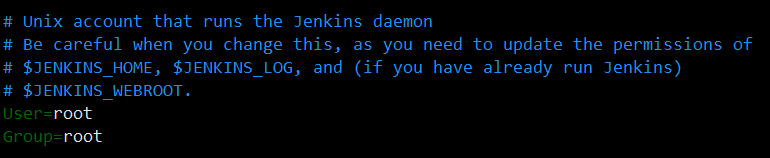
4）修改文件/usr/lib/systemd/system/jenkins.service

vim /usr/lib/systemd/system/jenkins.service

#如果不是root，而是jenkins或者其它，要改成root

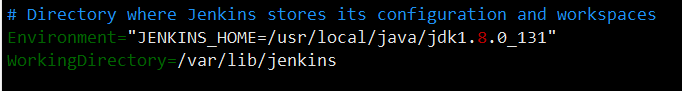
User=root

Group=root



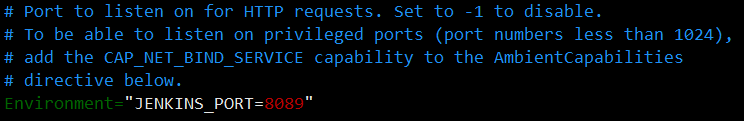
#JENKINS\_HOME配置JAVA路径

Environment="JENKINS\_HOME=/usr/local/java/jdk1.8.0\_131"



#JENKINS\_PORT配置端口号与/etc/sysconfig/jenkins一致

Environment="JENKINS\_PORT=8089"



### **启动**

systemctl daemon-reload

systemctl start jenkins

systemctl stop jenkins.service

systemctl status jenkins.service

systemctl enable jenkins.service

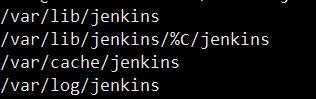
### **卸载jenkins**

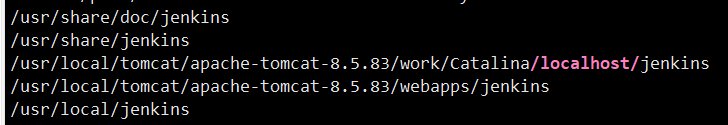
dpkg -l |grep jenkins

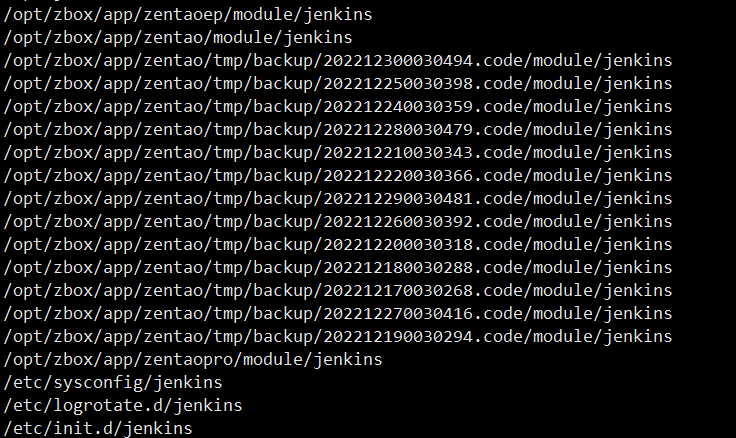
dpkg -P jenkins

dpkg -l |grep jenkins

### **Jenkins 相关文件目录**







### **Jenkins 访问地址**

IP+端口：<http://10.29.4.223:8089/>



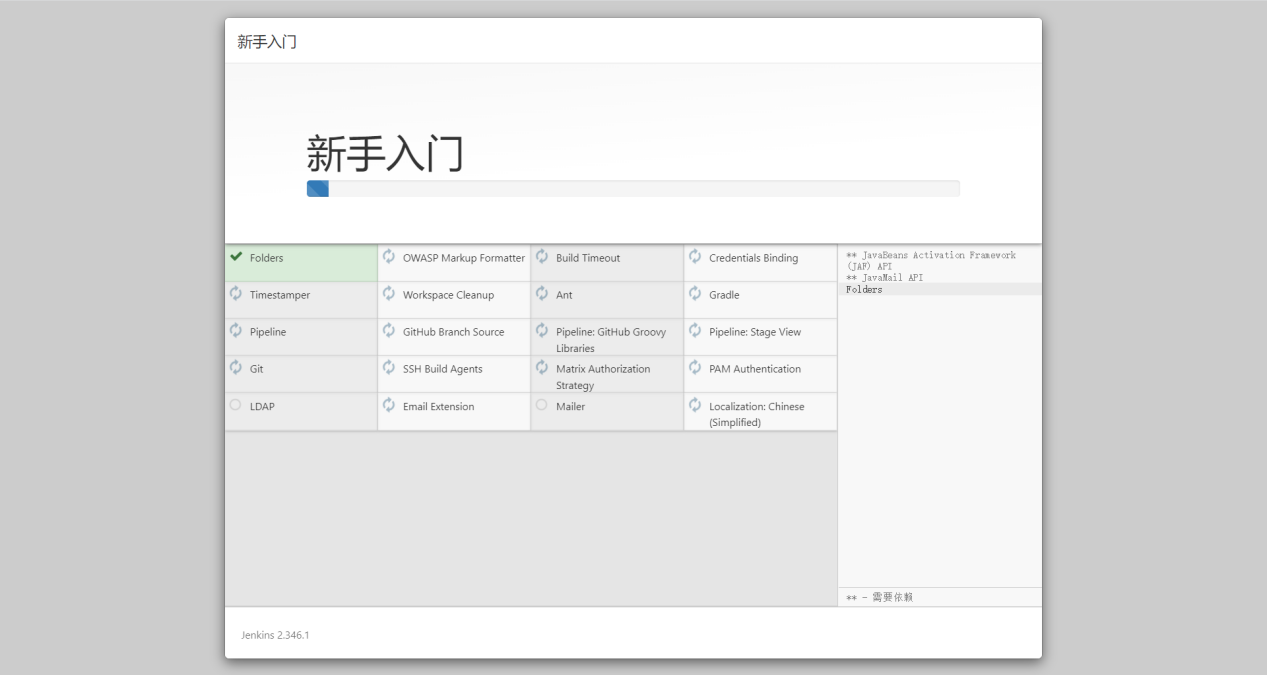
### **初始化**

获取密码



安装推荐插件





创建管理员





保存并完成

Jenkins配置用户

## 使用配置

### 用户管理

打开Manage Jenkins

- 进入Manage users

- 新建用户

### 全局工具配置

配置maven

- 打开Manage Jenkins

- 进入Global Tool Configuration

- Maven 配置

- JDK

- Git

### 插件安装

## 工具安装与自动化部署集成

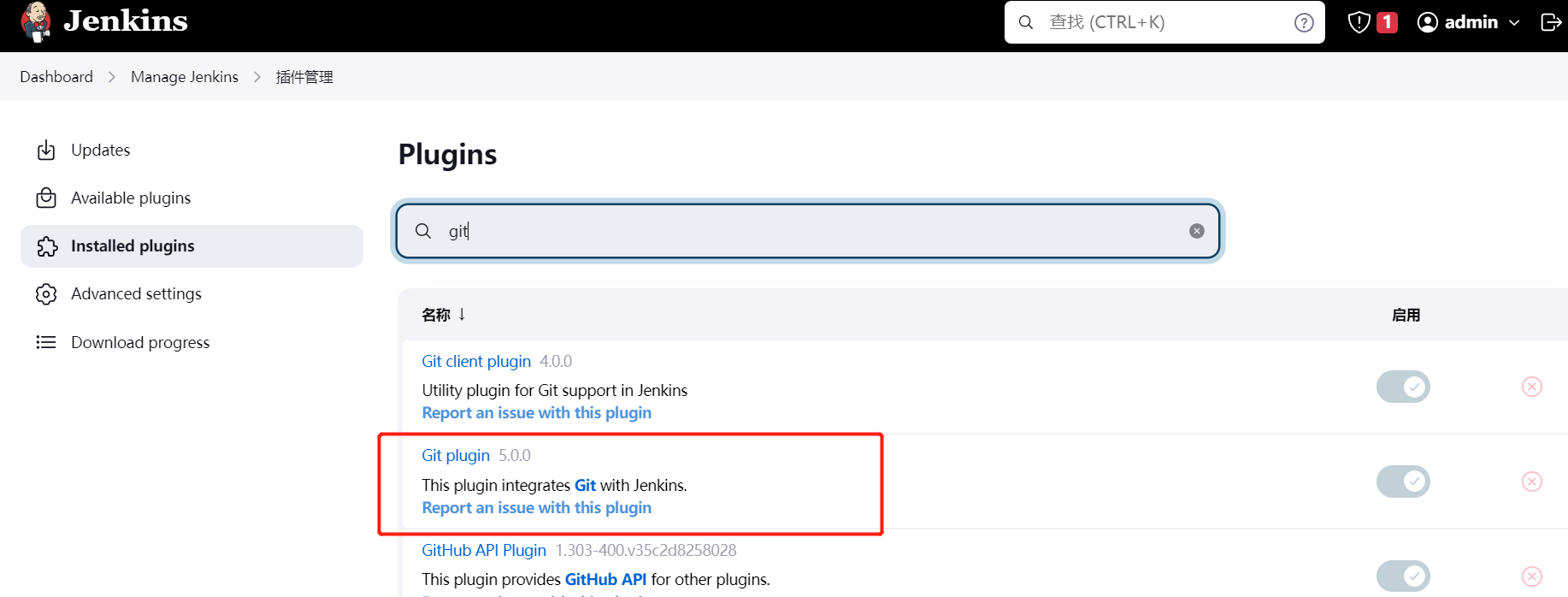
### 3.1安装git

**[配置Git](https://so.csdn.net/so/search?q=%E9%85%8D%E7%BD%AEGit&spm=1001.2101.3001.7020" \t "https://blog.csdn.net/xiaomao1993/article/details/_blank)客户端**

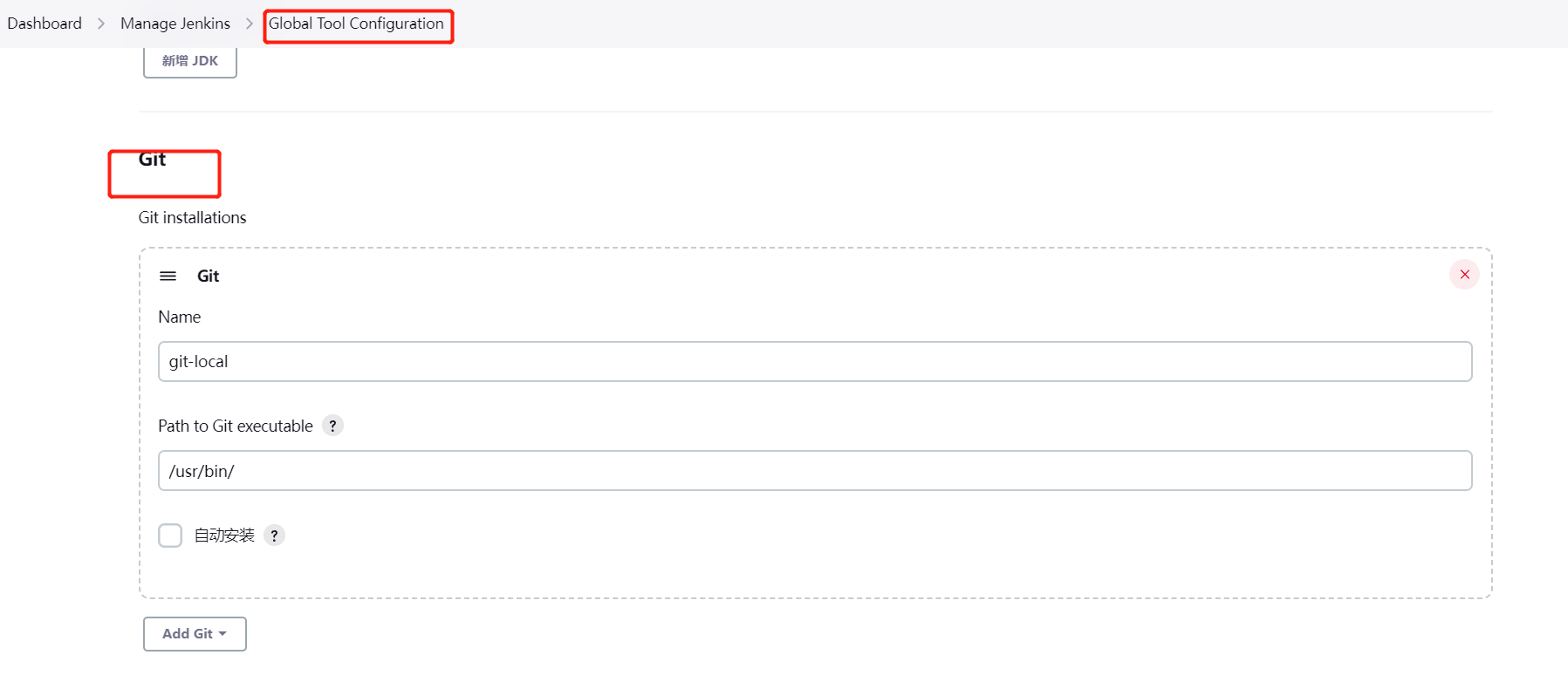
Git客户端[下载路径](https://git-scm.com/downloads)https://git-scm.com/downloads，具体安装下一步即可。可自行百度。

**Jenkins配置插件Git**

在仪表盘选择Manage Jenkins>>Plugin Manager>>进入如下页面，可以选择可选的插件，安装完成后的插件在installed里面可以看到，我这里已经安装了。



**Jenkins选择git路径（可选）**



### 3.2安装nodejs

**下载解压**

在nodejs官方网站download页面中选择需要使用的版本,可以选择LTS或current,选择对应系统右键复制链接地址

下载地址：<https://nodejs.org/en/download/>

**查看node版本**

/node-v18XXX/bin/node -V

**将node和npm设置为全局**

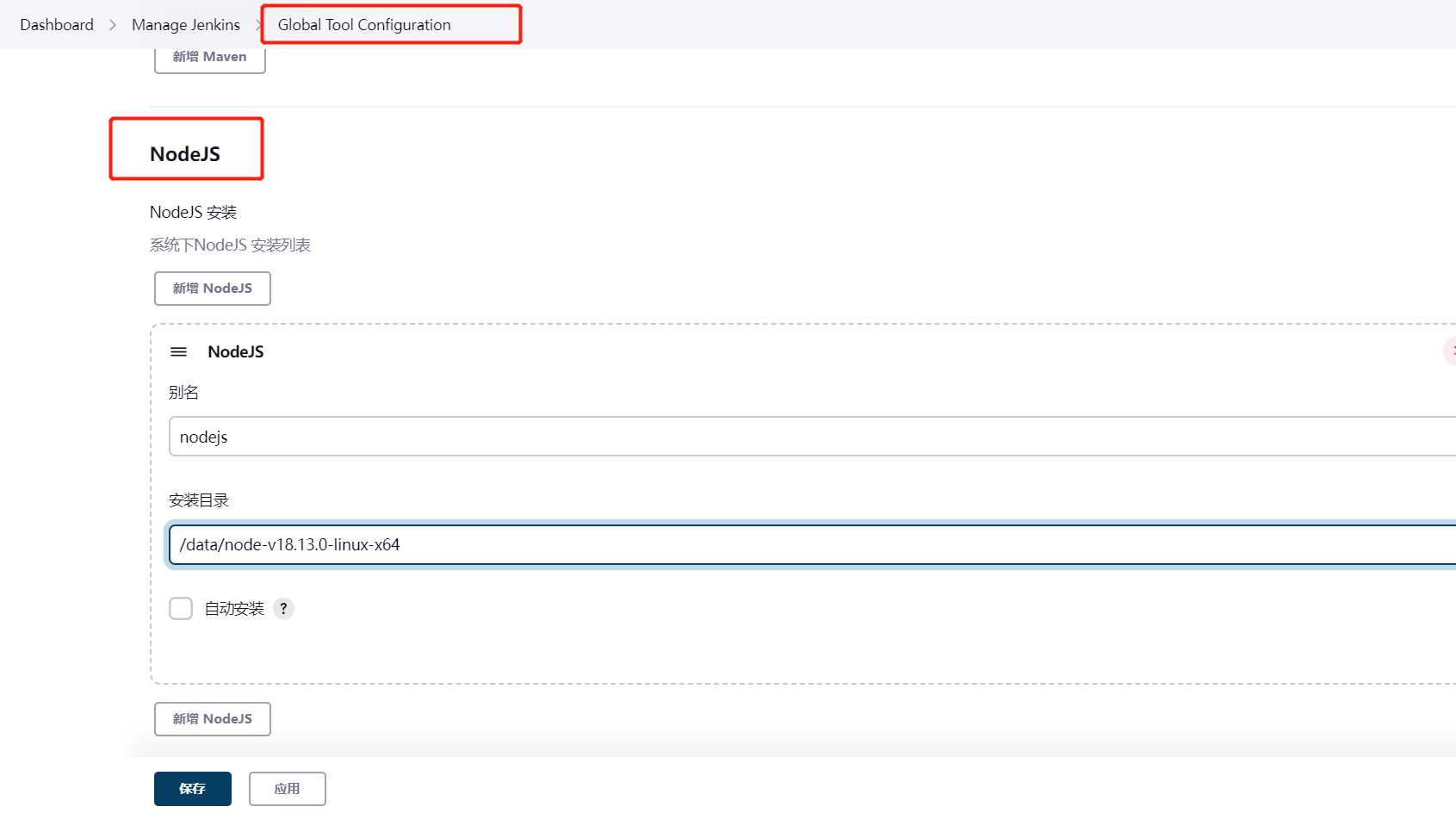
ln node -s /usr/local/bin/node

ln rpm -s /usr/local/bin/rpm

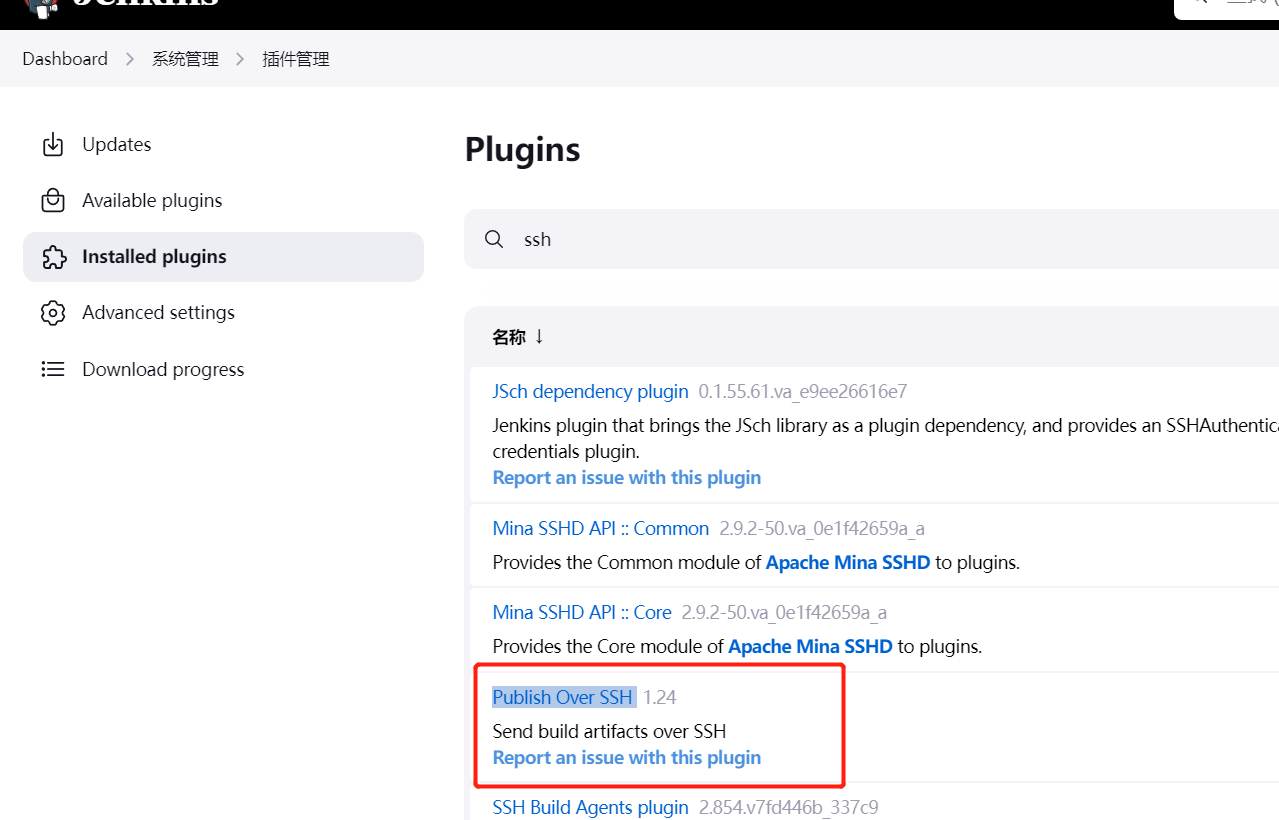
**切换为淘宝镜像命令（安装一些package容易报错）**  
npm config set registry <https://registry.npm.taobao.org>

**Jenkins选择node路径**

如果没用，需要安装插件nodejs



### 3.3安装[Publish Over SSH](https://plugins.jenkins.io/publish-over-ssh" \t "http://192.168.186.141:8080/manage/pluginManager/_blank)



配置服务器

Name：名称

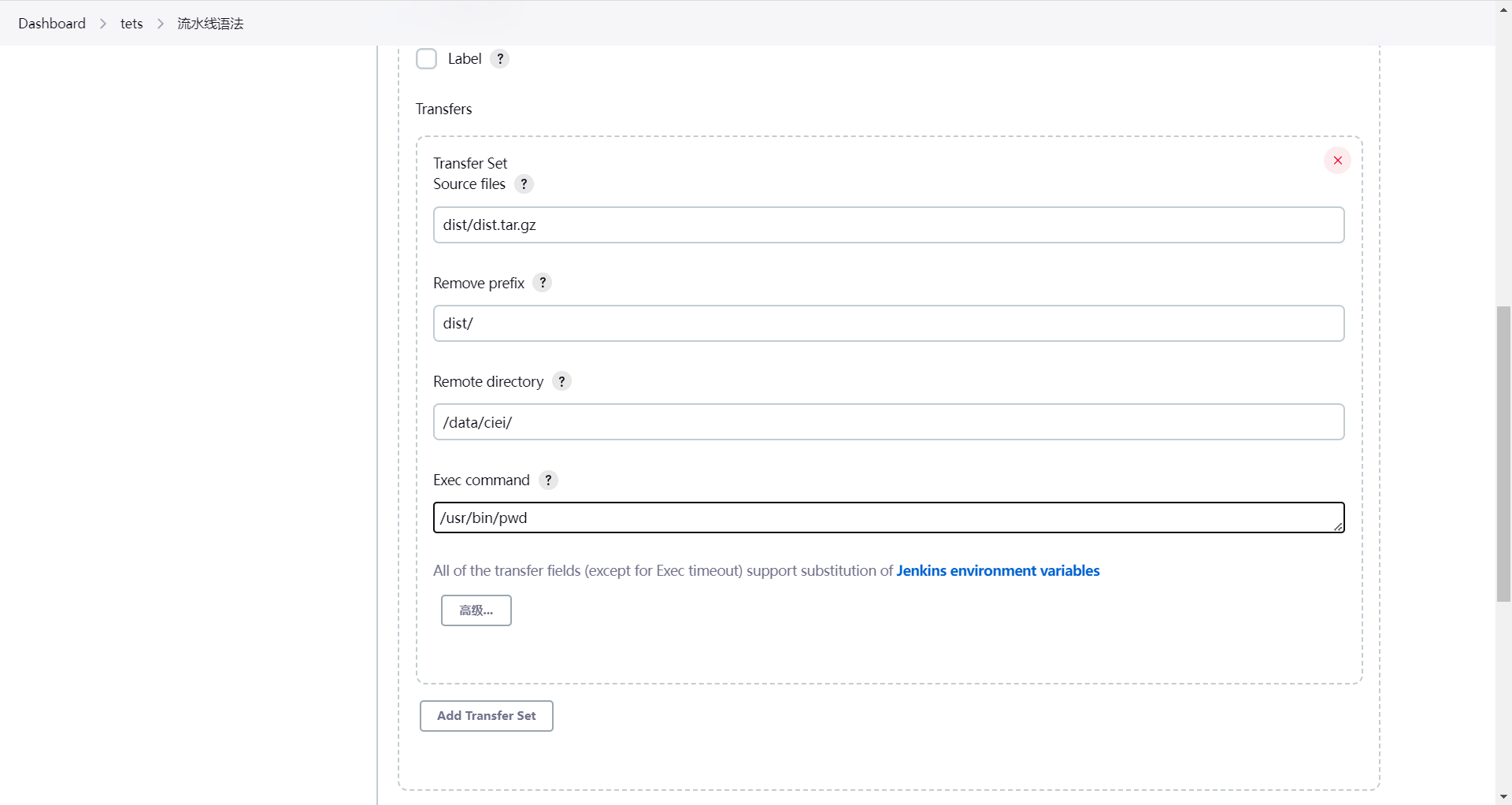
Hostname：地址

Username：账号

Remote Directory：配置目录



生成流水线



Name:

SSH Sverver的名字列表

Source files:

jenkins本机上的文件，相对workspace的路径，如上图中表示的是/home/cmq/.jenkins/workspace/项目1/test1下的所有文件和文件名

Remove prefix:

文件复制时要过滤的目录，要不然会创建多层目录

Remote directory:

远程机上的目录，此目录是相对于【SSH Server】中的【Remote directory】的，如果不存在将会自动创建，设置为/根目录的话，这里可以用绝对路径

Exec command:

远程机器上执行的命令或脚本

生成的流水线语法:

sshPublisher(publishers: [sshPublisherDesc(configName: '200', transfers: [sshTransfer(cleanRemote: false, excludes: '', execCommand: '/usr/bin/pwd', execTimeout: 120000, flatten: false, makeEmptyDirs: false, noDefaultExcludes: false, patternSeparator: '[, ]+', remoteDirectory: '/data/ciei', remoteDirectorySDF: false, removePrefix: 'dist/', sourceFiles: 'dist/dist.tar.gz')], usePromotionTimestamp: false, useWorkspaceInPromotion: false, verbose: false)])

### 3.4自动化部署前端pipeline配置

注意：此pipeline仅用于手动点击触发执行更新

pipeline需要根据实际情况修改修改的地方：

1. parameters 可以添加分支名称
2. stage('拉取代码') 中 代码仓库的地址
3. stage('SSH Publisher') 中修改传输目录和远程执行脚本名称

pipeline {

agent any

parameters {

choice choices: ['dev', 'test', 'prod', 'main', 'master'], description: '分支', name: 'BRANCH\_NAME'

credentials description: '密钥', name: 'KEY'

}

environment

{

VAR1 = ""

VAR2 = ""

VAR3 = ""

VAR4 = ""

VAR5 = ""

VAR6 = ""

}

stages {

stage('delete代码') {

steps {

deleteDir()

echo 'delete代码成功'

}

}

stage('拉取代码') {

steps {

git branch: "${BRANCH\_NAME}", credentialsId: "${KEY}", url: "https://gitee.com/smalls\_trees/disk.git"

echo '拉取代码成功'

}

}

stage('Get Last Commit Details') {

steps {

script {

sh 'git config --global --add safe.directory $WORKSPACE'

VAR1 = sh(script: "git log origin/master -n1 --format=format:'%H'", returnStdout: true).trim()

VAR2 = sh(script: "git log origin/master -n1 --format=format:'%cn'", returnStdout: true).trim()

VAR3 = sh(script: "git log origin/master -n1 --format=format:'%ce'", returnStdout: true).trim()

VAR4 = sh(script: "git log origin/master -n1 --format=format:'%cd'", returnStdout: true).trim()

VAR5 = sh(script: "git log origin/master -n1 --format=format:'%cr'", returnStdout: true).trim()

VAR6 = sh(script: "git log origin/master -n1 --format=format:'%s'", returnStdout: true).trim()

echo "VAR1=${VAR1}"

echo "VAR2=${VAR2}"

echo "VAR3=${VAR3}"

echo "VAR4=${VAR4}"

echo "VAR5=${VAR5}"

echo "VAR6=${VAR6}"

// 修改当前构建名称和描述

currentBuild.displayName = "#${env.BUILD\_NUMBER}-${VAR2}-main分支"

currentBuild.description = "提交人：${VAR2} \n 邮箱：${VAR3}\n日期：${VAR4}\n时间：${VAR5}\n提交说明：${VAR6}"

// 打印修改后的构建名称和描述

println currentBuild.displayName

println currentBuild.description

}

}

}

stage('执行构建') {

steps {

nodejs('nodejs') {

sh '''# 查看版本信息

npm -v

# 将镜像源替换为淘宝的加速访问

npm config set registry https://registry.npm.taobao.org

# 安装项目依赖

npm install

# 项目打包

npm run build

# 压缩dist文件

tar -zcvf dist.tar.gz dist

# 移动到dist下

mv dist.tar.gz dist/'''

}

echo '执行构建成功'

}

}

stage('SSH Publisher') {

steps {

script {

if (env.BRANCH\_NAME == 'master') {

echo '更新master'

sshPublisher(publishers: [sshPublisherDesc(configName: '200', transfers: [sshTransfer(cleanRemote: false, excludes: '', execCommand: '/opt/ciei/update\_fontend\_mss\_vue.sh', execTimeout: 120000, flatten: false, makeEmptyDirs: false, noDefaultExcludes: false, patternSeparator: '[, ]+', remoteDirectory: 'ciei/dist', remoteDirectorySDF: false, removePrefix: 'dist/', sourceFiles: 'dist/dist.tar.gz')], usePromotionTimestamp: false, useWorkspaceInPromotion: false, verbose: false)])

} else if (env.BRANCH\_NAME == 'main'){

echo '更新main'

}

echo '更新成功'

}

}

}

}

}

更新服务器端的前端更新脚本如下（根据实际情况修改对应的路径）：

#!/bin/sh

# 前端更新脚本

#备份旧前端包

date=$(date +%y-%m-%d-%H%M%m)

zip -r /data/ciei/backup/dist\_$date.zip /data/ciei/new/dist

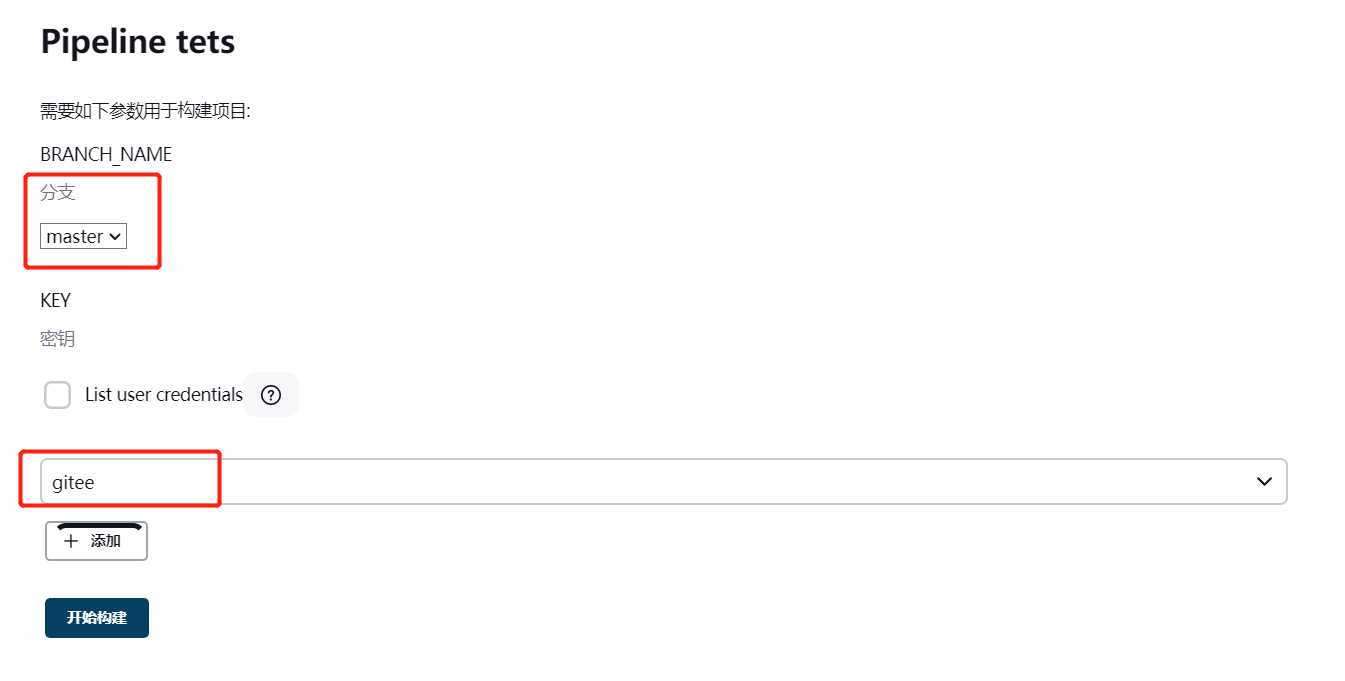
#删除旧前端包

rm -rf /data/ciei/new/dist

#替换新前端包

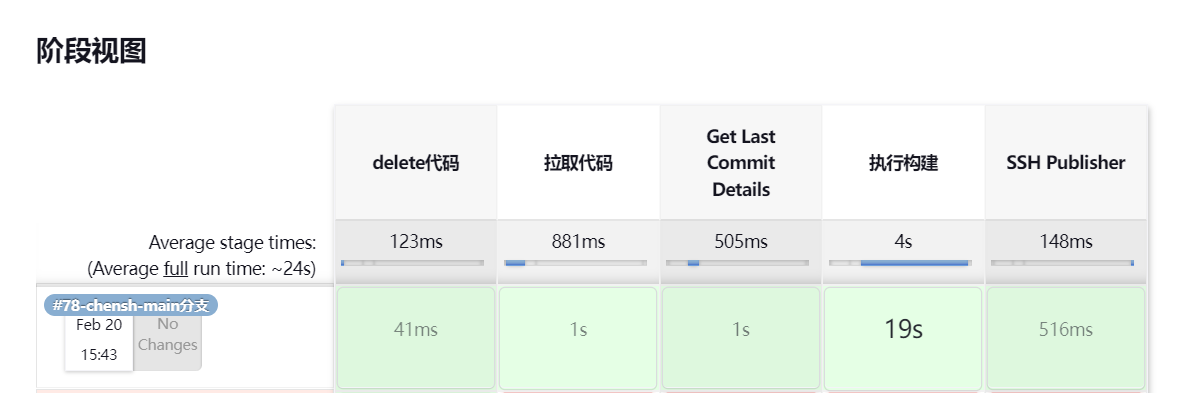
tar -zxf /data/ciei/dist/dist.tar.gz -C /data/ciei/new/

执行时，选择对应分支和密钥

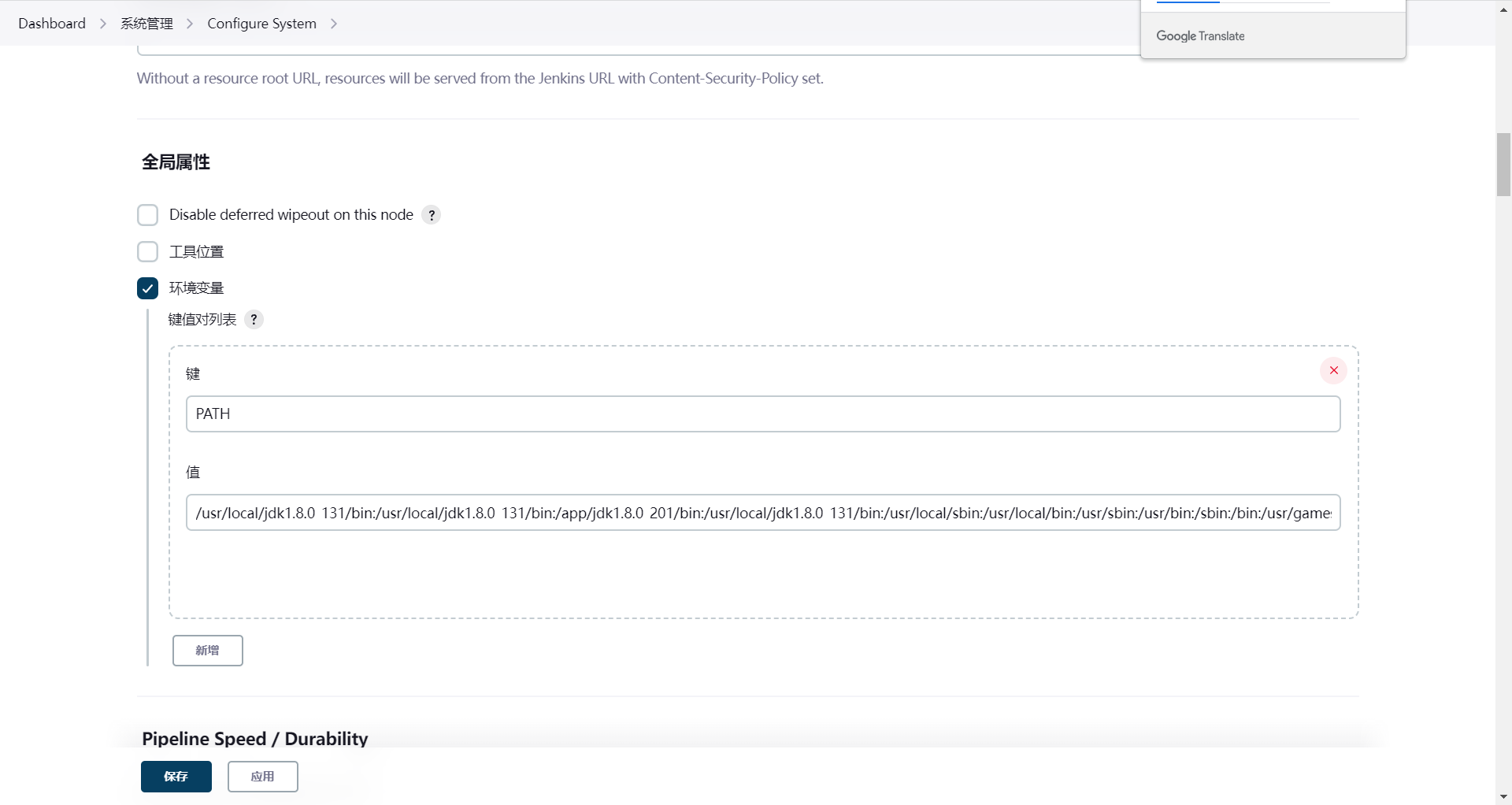


执行成功结果





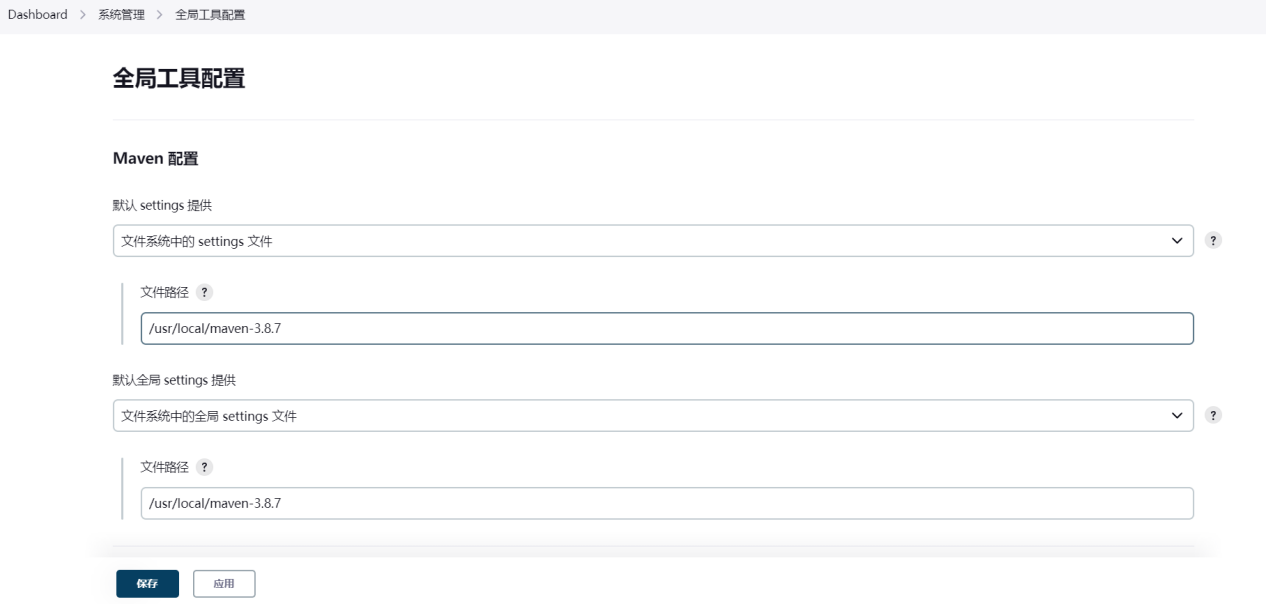
### 3.5 Jenkins配置全局的PATH环境变量



### 3.6安装配置maven

linux安装maven

配置maven



### 3.6自动化部署后端pipeline配置

注意：此pipeline仅用于手动点击触发执行更新

pipeline需要根据实际情况修改修改的地方：

1. parameters 可以添加分支名称
2. stage('拉取代码') 中 代码仓库的地址
3. stage('SSH Publisher') 中修改传输目录和远程执行脚本名称

pipeline {

agent any

parameters {

choice choices: ['dev', 'test', 'prod', 'main', 'master'], description: '分支', name: 'BRANCH\_NAME'

credentials description: '密钥', name: 'KEY'

}

environment

{

VAR1 = ""

VAR2 = ""

VAR3 = ""

VAR4 = ""

VAR5 = ""

VAR6 = ""

}

stages {

stage('delete代码') {

steps {

deleteDir()

echo 'delete代码成功'

}

}

stage('拉取代码') {

steps {

git branch: "${BRANCH\_NAME}", credentialsId: "${KEY}", url: "https://gitee.com/smalls\_trees/springboot.git"

echo '拉取代码成功'

}

}

stage('Get Last Commit Details') {

steps {

script {

sh 'git config --global --add safe.directory $WORKSPACE'

VAR1 = sh(script: "git log origin/master -n1 --format=format:'%H'", returnStdout: true).trim()

VAR2 = sh(script: "git log origin/master -n1 --format=format:'%cn'", returnStdout: true).trim()

VAR3 = sh(script: "git log origin/master -n1 --format=format:'%ce'", returnStdout: true).trim()

VAR4 = sh(script: "git log origin/master -n1 --format=format:'%cd'", returnStdout: true).trim()

VAR5 = sh(script: "git log origin/master -n1 --format=format:'%cr'", returnStdout: true).trim()

VAR6 = sh(script: "git log origin/master -n1 --format=format:'%s'", returnStdout: true).trim()

echo "VAR1=${VAR1}"

echo "VAR2=${VAR2}"

echo "VAR3=${VAR3}"

echo "VAR4=${VAR4}"

echo "VAR5=${VAR5}"

echo "VAR6=${VAR6}"

// 修改当前构建名称和描述

currentBuild.displayName = "#${env.BUILD\_NUMBER}-${VAR2}-main分支"

currentBuild.description = "提交人：${VAR2} \n 邮箱：${VAR3}\n日期：${VAR4}\n时间：${VAR5}\n提交说明：${VAR6}"

// 打印修改后的构建名称和描述

println currentBuild.displayName

println currentBuild.description

}

}

}

stage('执行maven构建') {

steps {

sh 'mvn clean'

sh 'mvn package'

echo '执行maven构建成功'

}

}

stage('SSH Publisher') {

steps {

script {

if (env.BRANCH\_NAME == 'master') {

echo '更新master'

sshPublisher(publishers: [sshPublisherDesc(configName: '200', transfers: [sshTransfer(cleanRemote: false, excludes: '', execCommand: '/opt/ciei/update\_backend.sh', execTimeout: 120000, flatten: false, makeEmptyDirs: false, noDefaultExcludes: false, patternSeparator: '[, ]+', remoteDirectory: 'ciei/server', remoteDirectorySDF: false, removePrefix: 'target/', sourceFiles: ' target/demo-0.0.1-SNAPSHOT.jar')], usePromotionTimestamp: false, useWorkspaceInPromotion: false, verbose: false)])

} else if (env.BRANCH\_NAME == 'main'){

echo '更新main'

}

echo '更新成功'

}

}

}

}

}

更新服务器端的后端更新脚本如下（根据实际情况修改对应的路径）：

#!/bin/sh

# 后端更新脚本

#备份旧后端包

date=$(date +%y-%m-%d-%H%M%m)

mv /data/ciei/new/demo-server.jar /data/ciei/backup/demo-server.jar\_$date

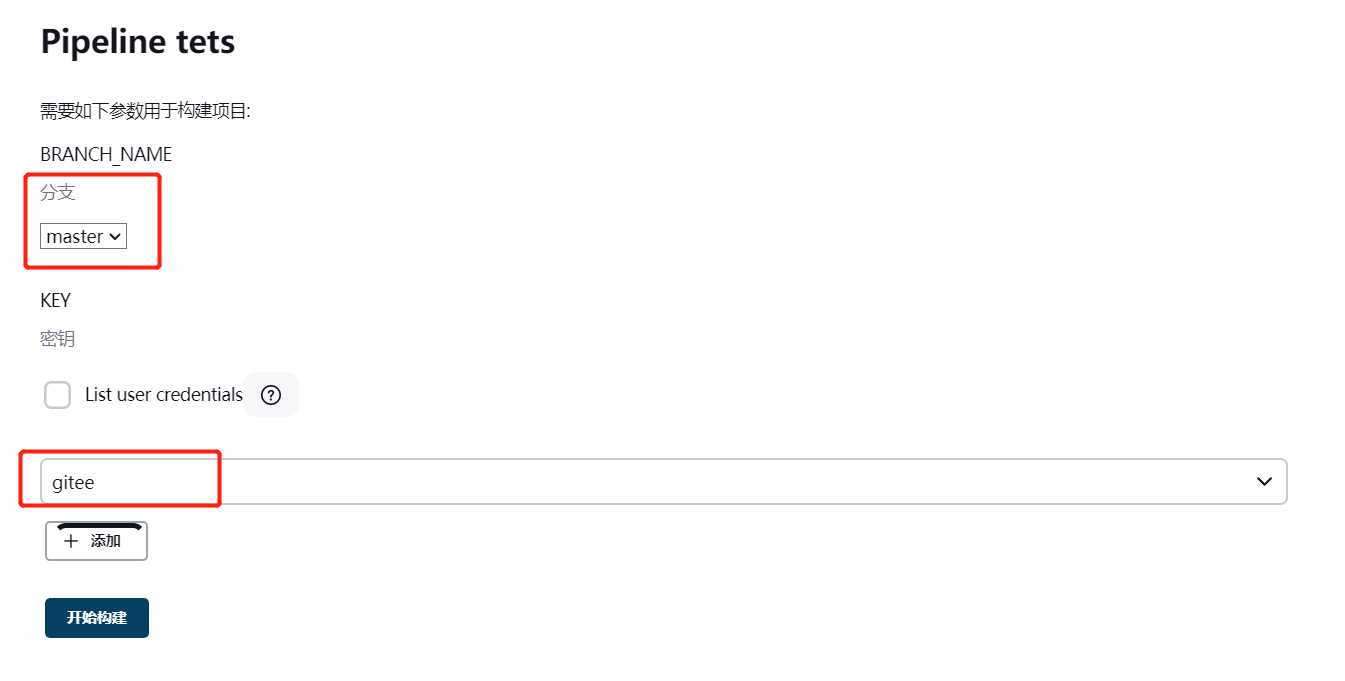
#新后端包

mv /data/ciei/server/demo-0.0.1-SNAPSHOT.jar /data/ciei/new/demo-server.jar

#重启服务运行

systemctl restart demo-server

执行时，选择对应分支和密钥



执行成功结果

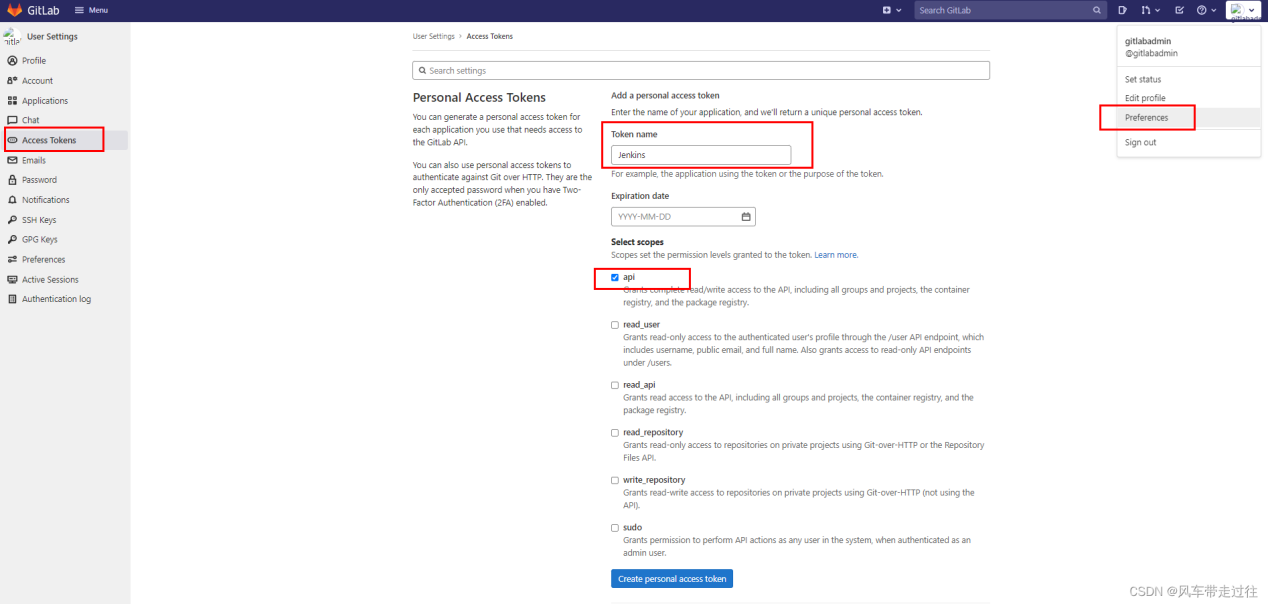


### 3.6配置webhooks

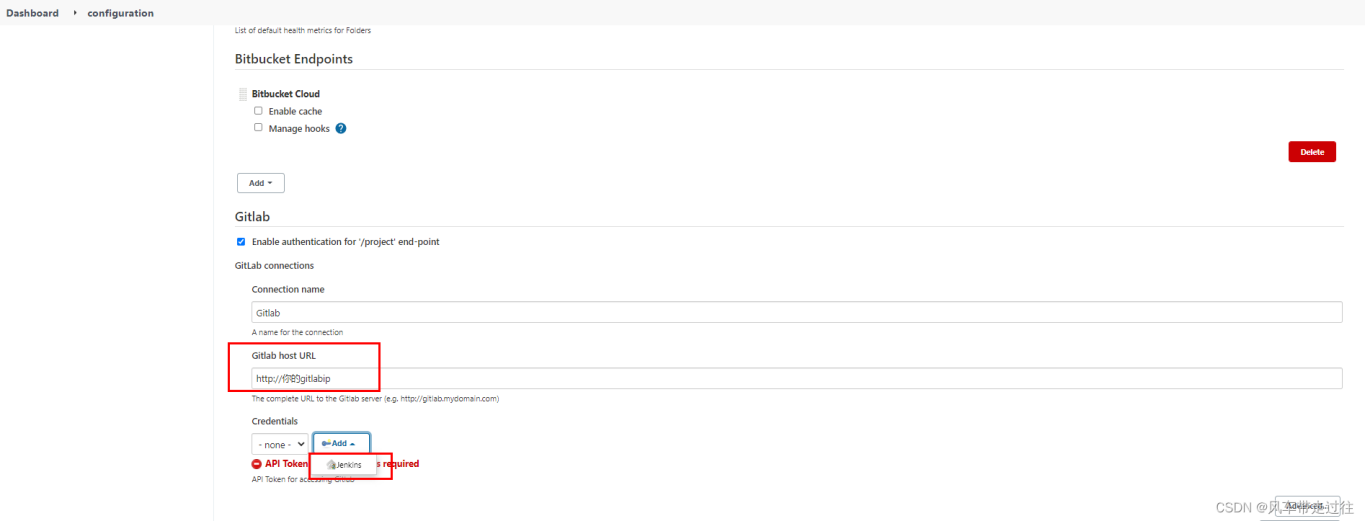
Jenkins安装插件

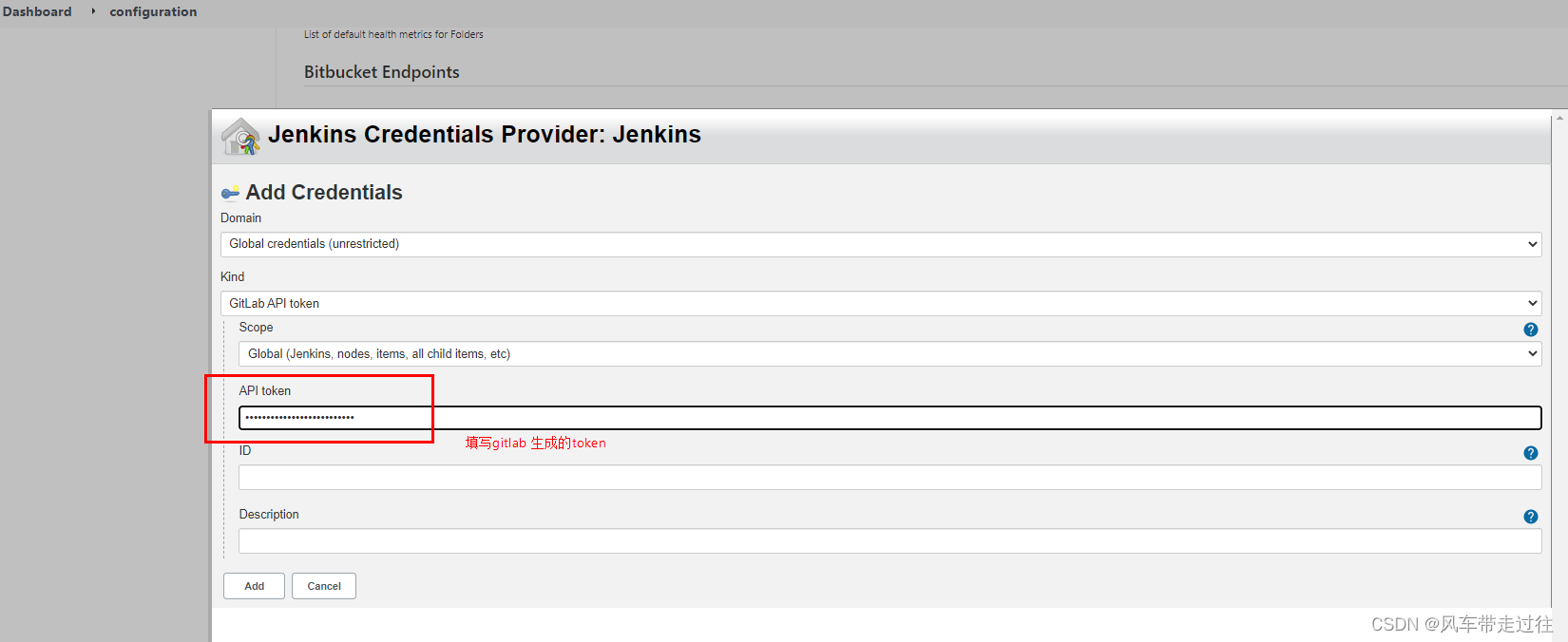
gitlab,[Generic Webhook Trigger](https://plugins.jenkins.io/generic-webhook-trigger" \t "http://192.168.186.141:8080/manage/pluginManager/_blank), gitlab authentication

gitlab生成令牌（token）



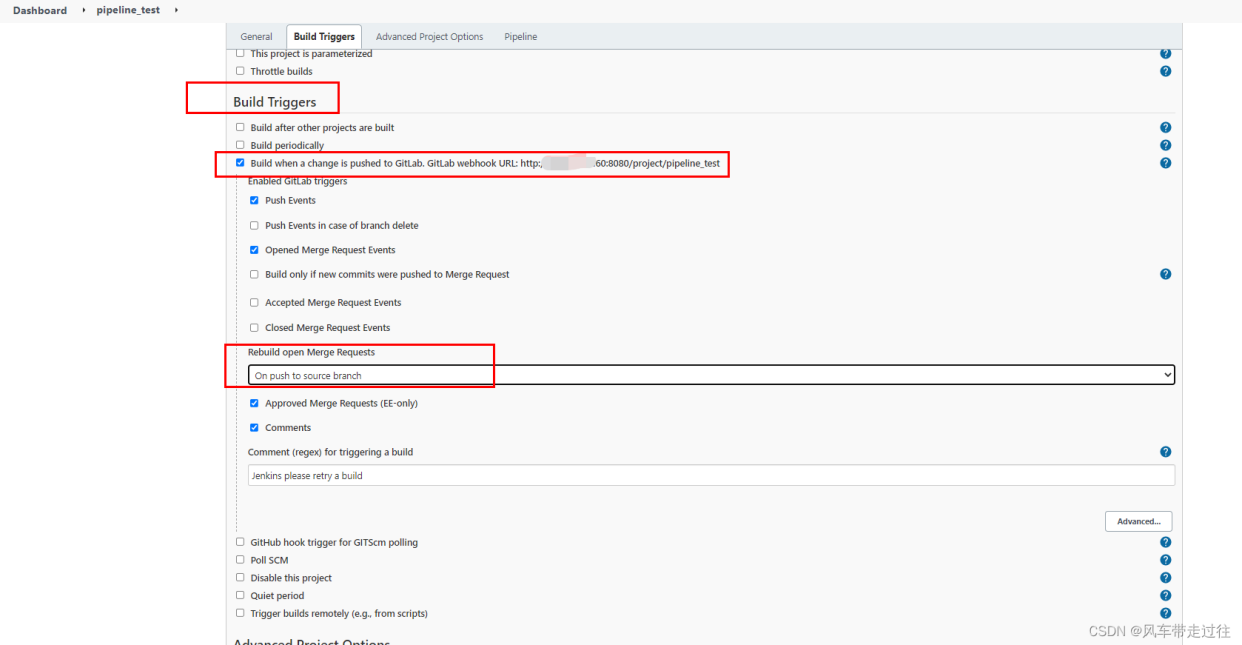
Jenkins系统配置gitlab 身份token

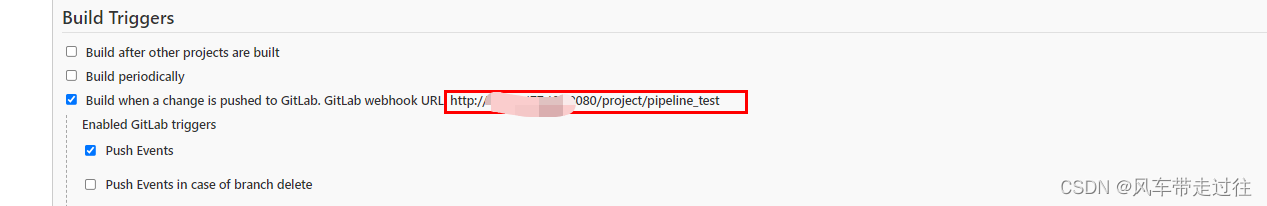
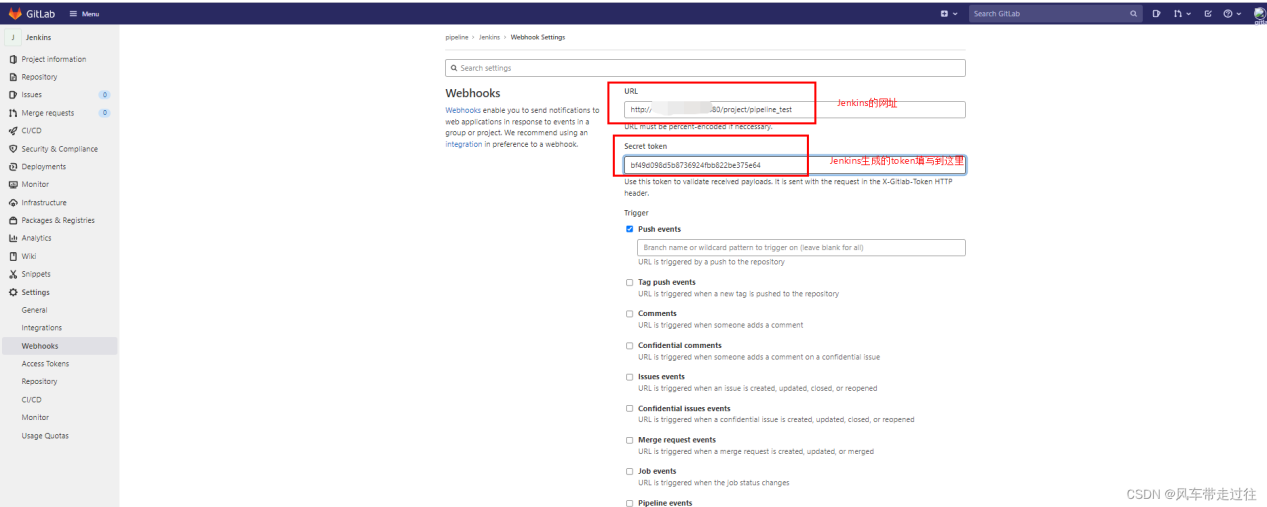




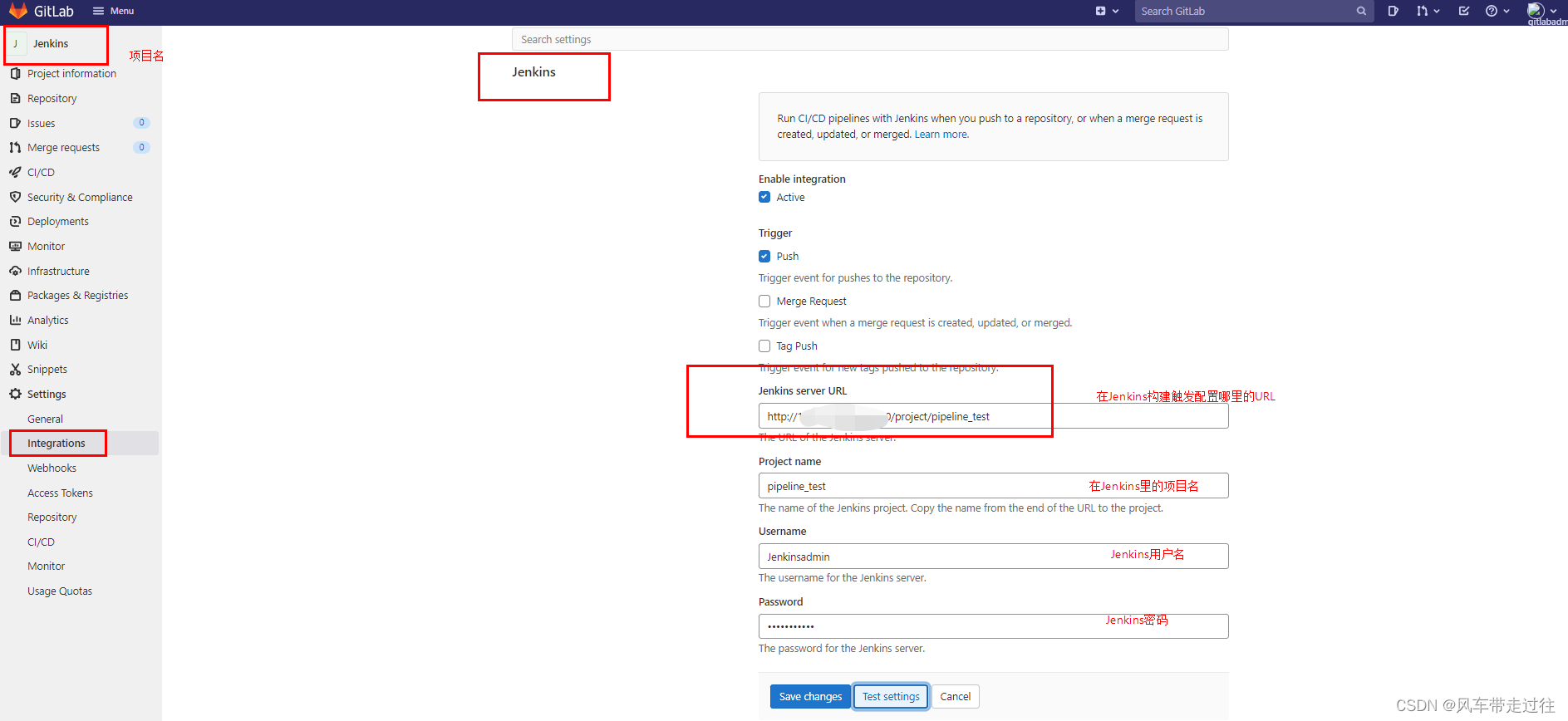
Jenkins构建项目添加构建触发条件两种方式

使用token方式



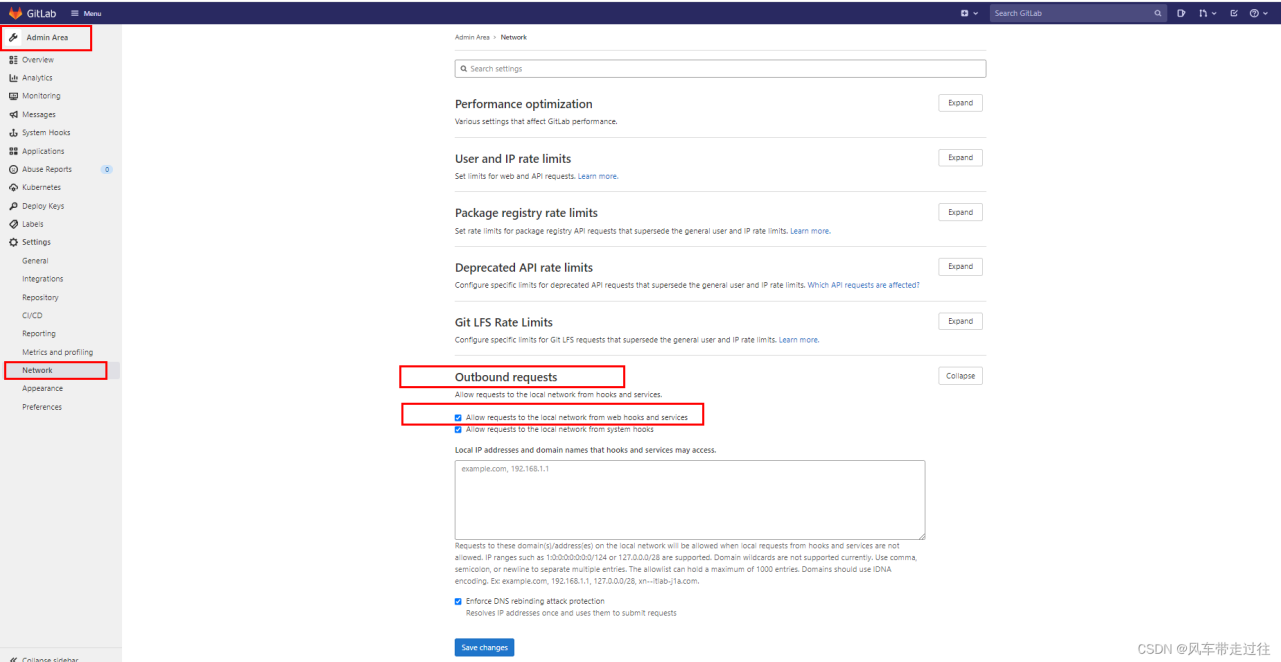
复制下图的URL添加到gitlab里  
  
  
取消下面 SSL verification认证

使用密码登录方式



当Jenkins和gitlab在同一台服务器时，gitlab提示连接Jenkins失败解决方案

在gitlab admin配置里，找到Outbound requests，允许Jenkins和gitlab在同一台服务器时启用webhooks



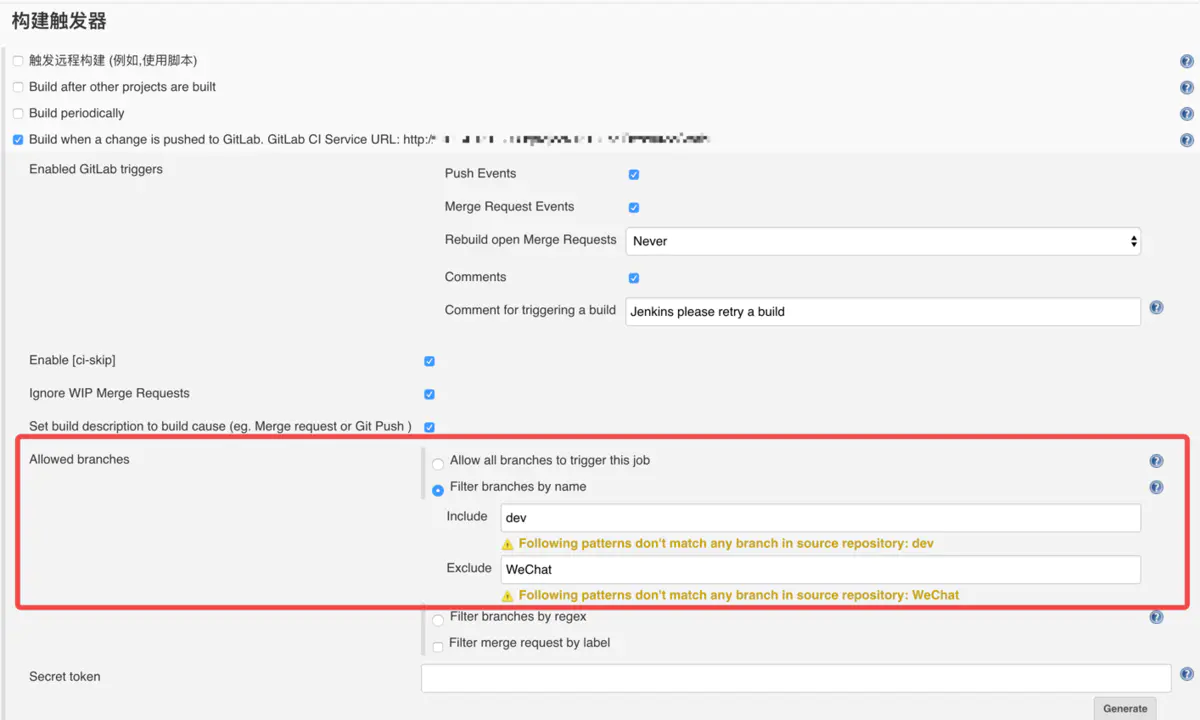
### 3.7 jenkins区分下分支构建

· 打开项目配置。

· 设置Allowed branches ，默认是 Allow all branches to trigger this job，也就是 同意所有分支触发

· 需要修改成 按分支名字触发，include里面写指定的分支名；exclude里面写排除的分支名；多个用逗号区分。

· apply&save即可。



### 3.8 自动化部署后端pipeline配置

根据不同项目修改<3.4自动化部署前端pipeline配置>章节pipeline参数为固定值即可

### 3.9 自动化部署前端pipeline配置

根据不同项目修改<3.6自动化部署后端pipeline配置>章节pipeline参数为固定值即可

### 4.0 数据库脚本自动执行

实现方法

1. 推荐：集成flyway（数据库版本控制） 开发做
2. 远程shell执行SQL脚本：脚本同代码提交到代码仓库→ssh传输脚本到更新服务器→远程执行脚本（脚本报错情况无法确定）