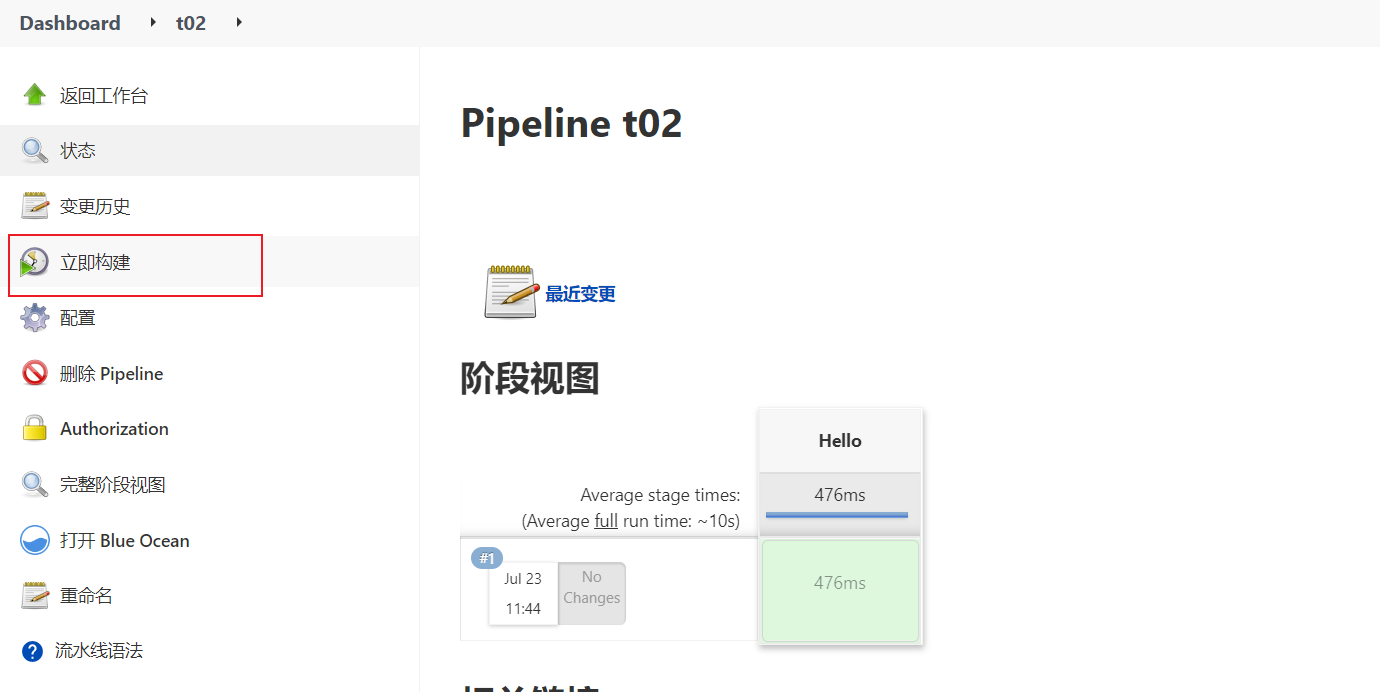
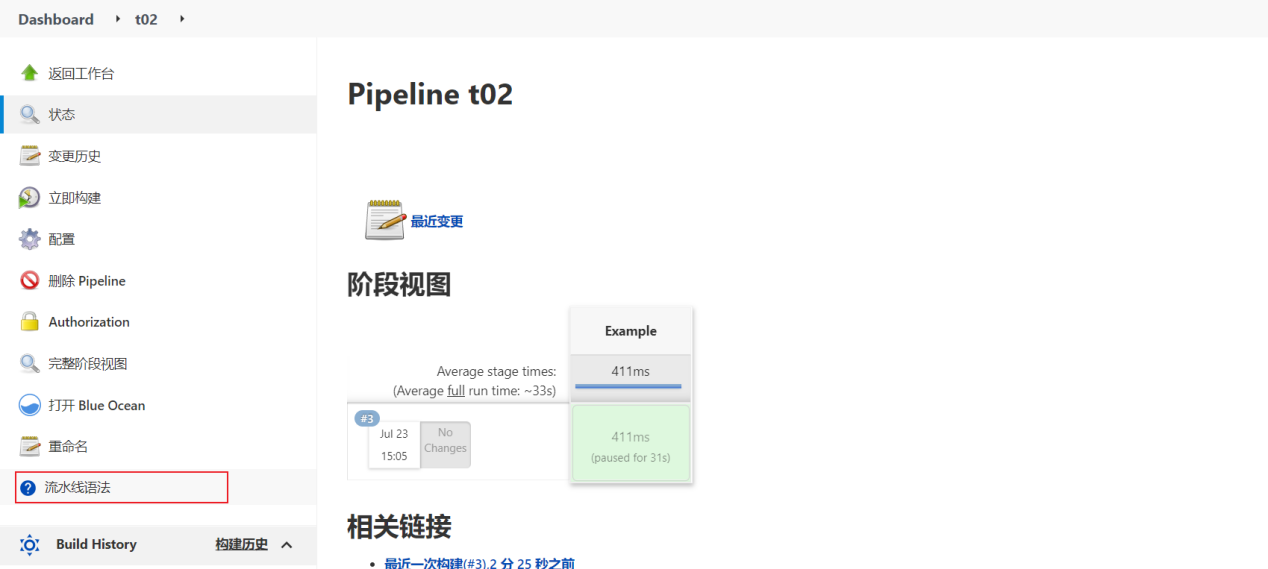
# **1、创建流水线任务**

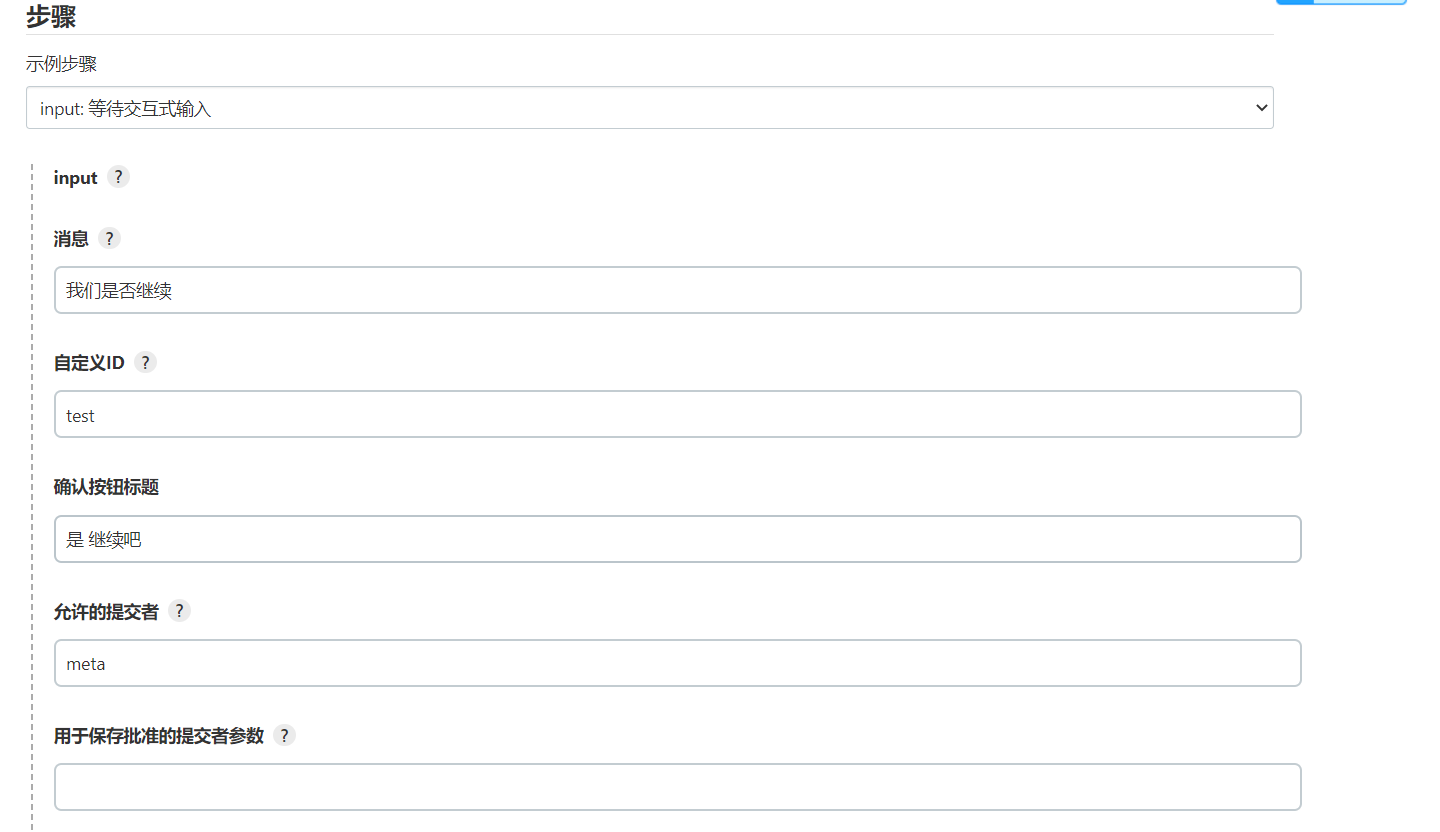


# **2、Jenkins 声明式与脚本式语法**

<https://zeyangli.github.io/chapter5/2/>

**流水线语法使用**







# **3、使用Jenkins共享库**

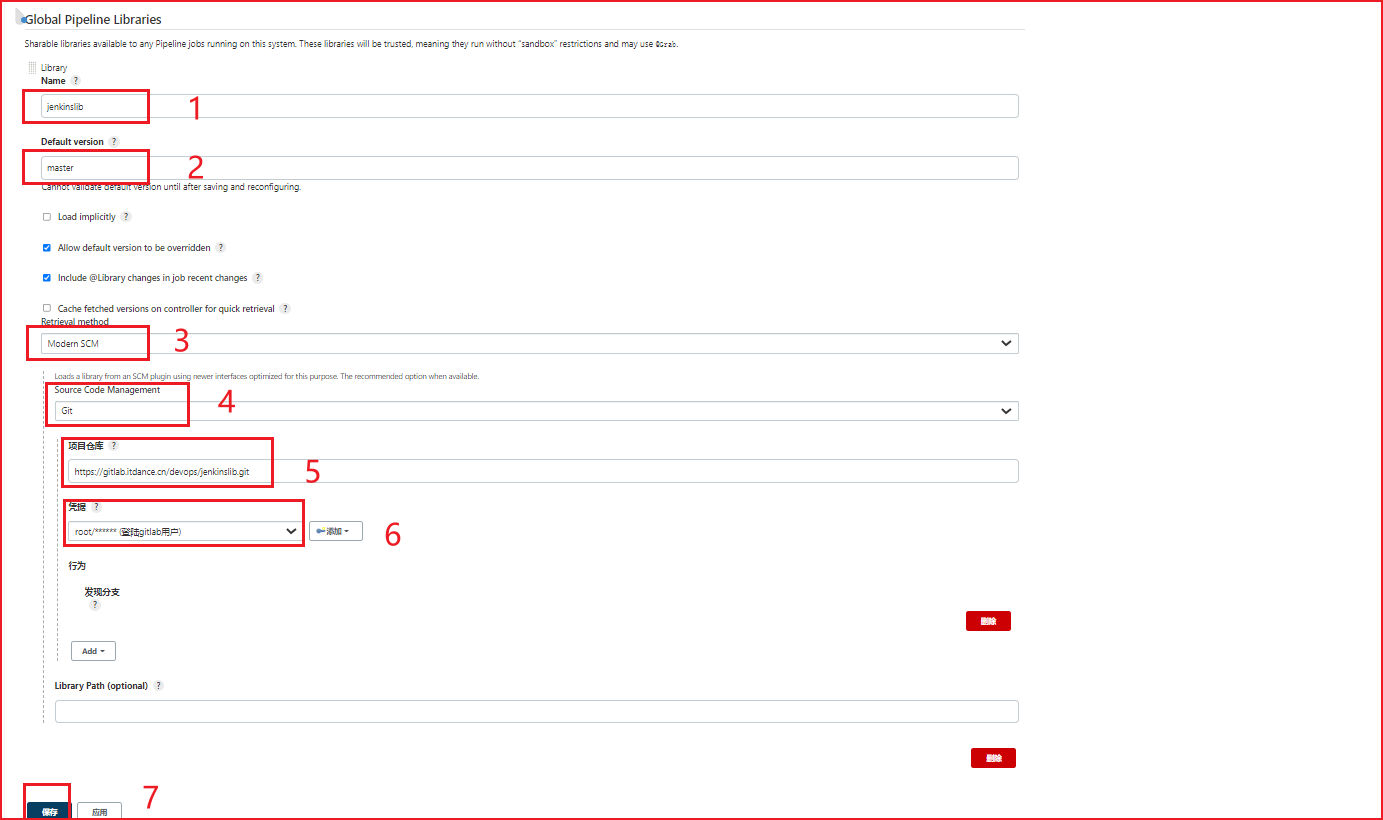


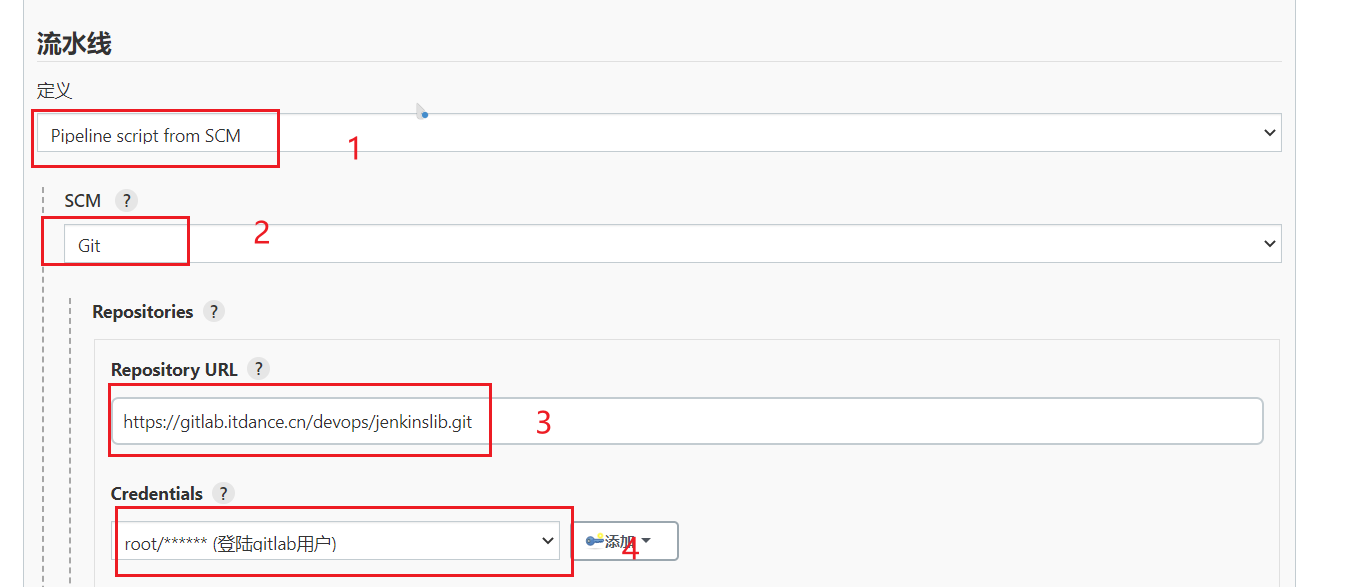
新建文件夹src/org/devops

并在src/org/devops 下新建文件tools.groovy

Jenkins 配置共享库

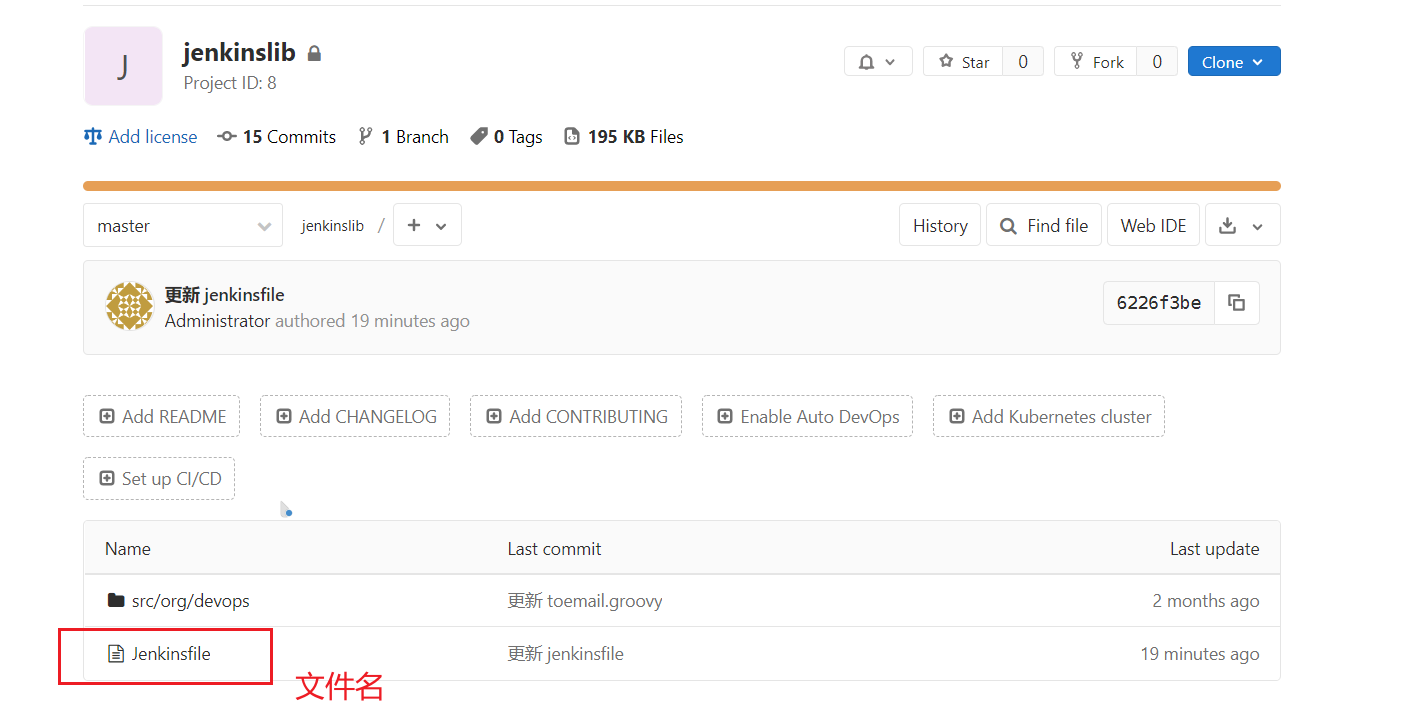
系统管理→系统配置→**Global Pipeline Libraries**







如下



还有就是路径：比如在aaa/jenkinsfile 那么Jenkins里也要配置成aaa/jenkinfile

jenkinsfile文件要提前准备号。

# **Groovy基础语法**

docker run -it --rm groovy 安装运行

# **常用的Pipeline DSL方法**

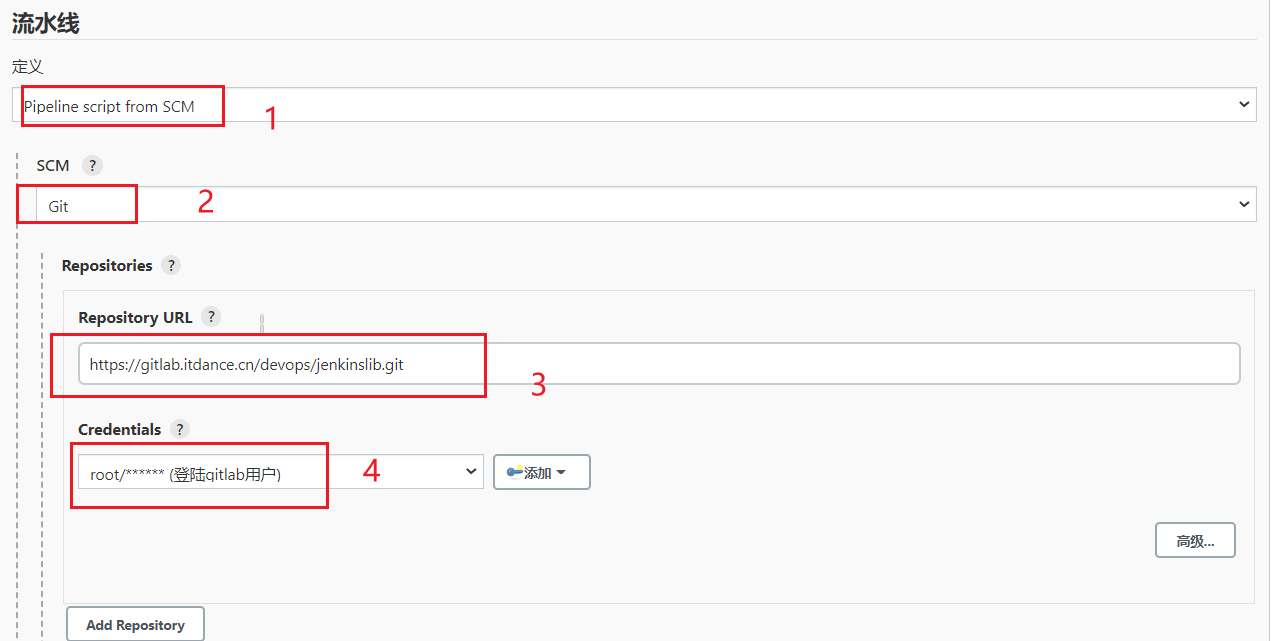
# **集成Maven工具**

**6.0安装**

|  |
| --- |
| wget <https://mirrors.sonic.net/apache/maven/maven-3/3.6.3/binaries/apache-maven-3.6.3-bin.zip>  unzip apache-maven-3.6.3-bin.zip  mv apache-maven-3.6.3 /usr/local/  --------  vim /etc/profile  export MAVEN\_HOME=/usr/local/apache-maven-3.6.3  export PARH=$PATH:$MAVEN\_HOME/bin:$PATH  --------  mvn -v |

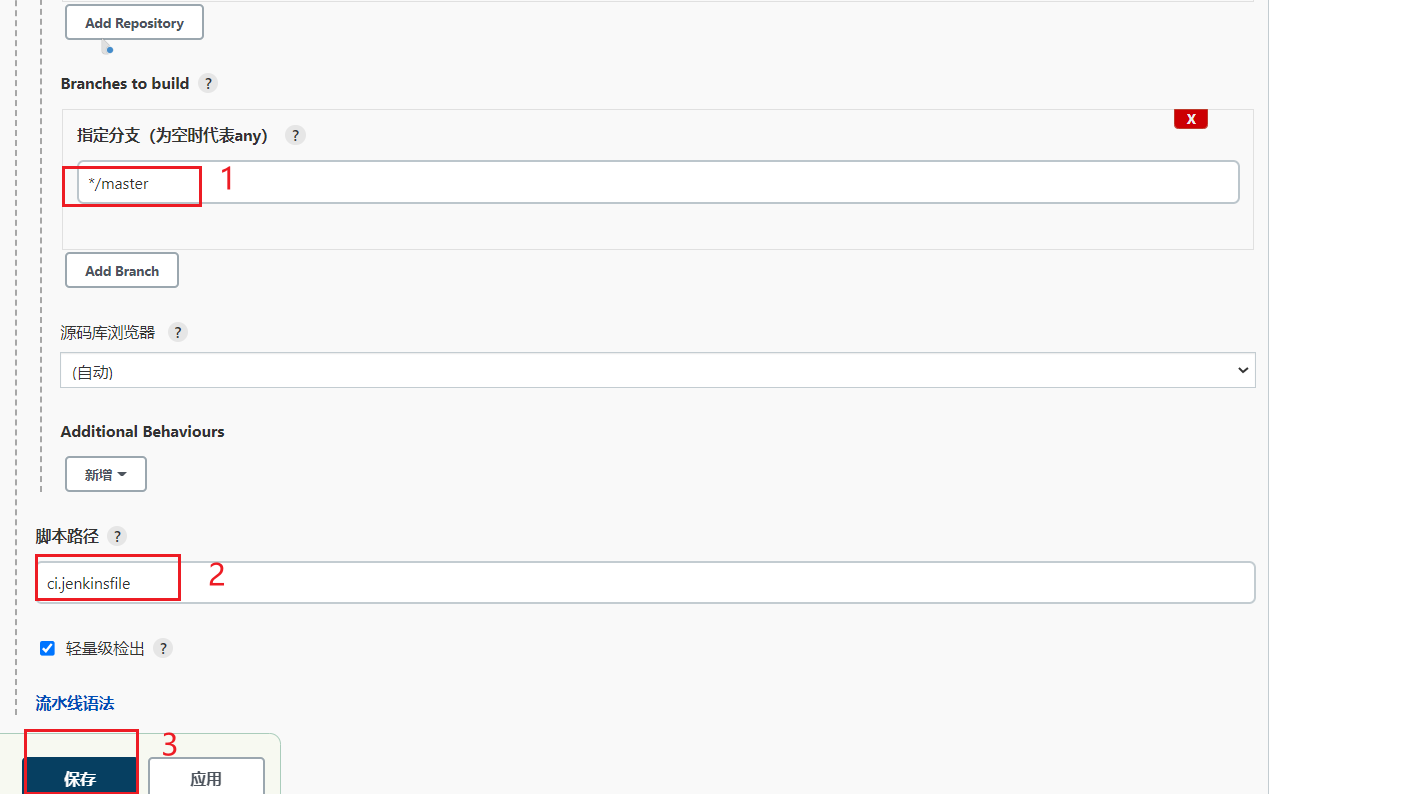
**6.1 Jenkins里设置**

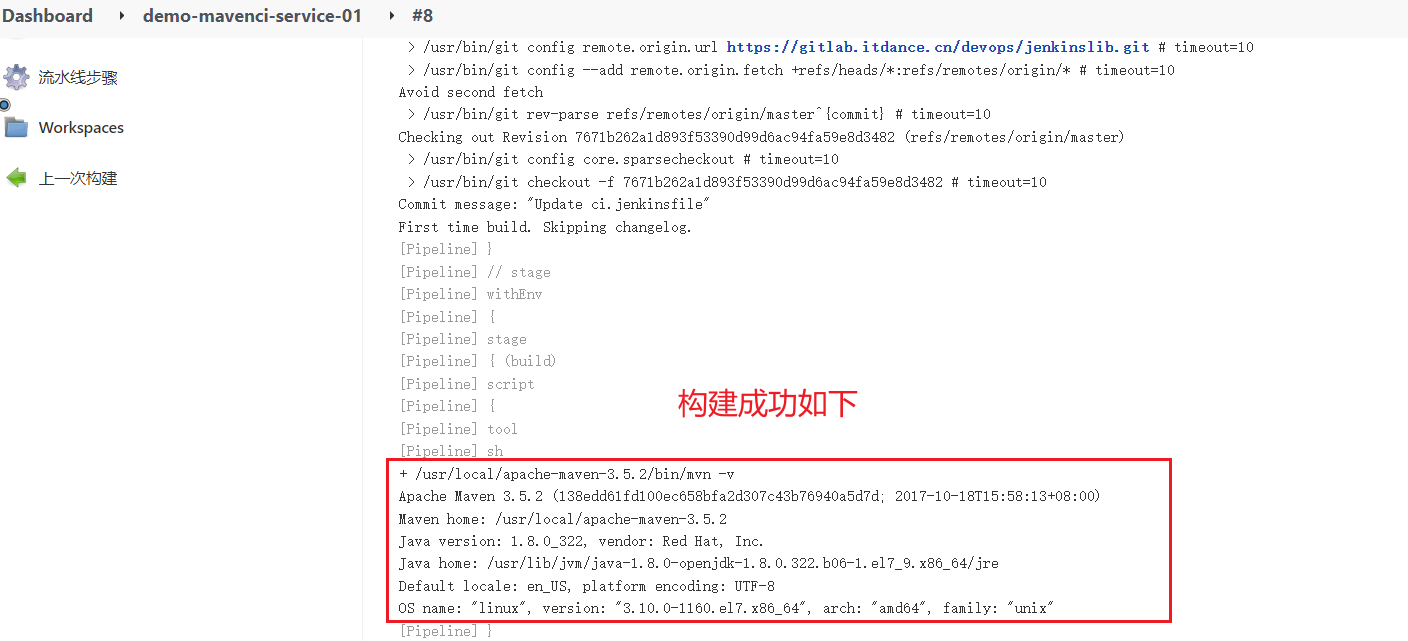




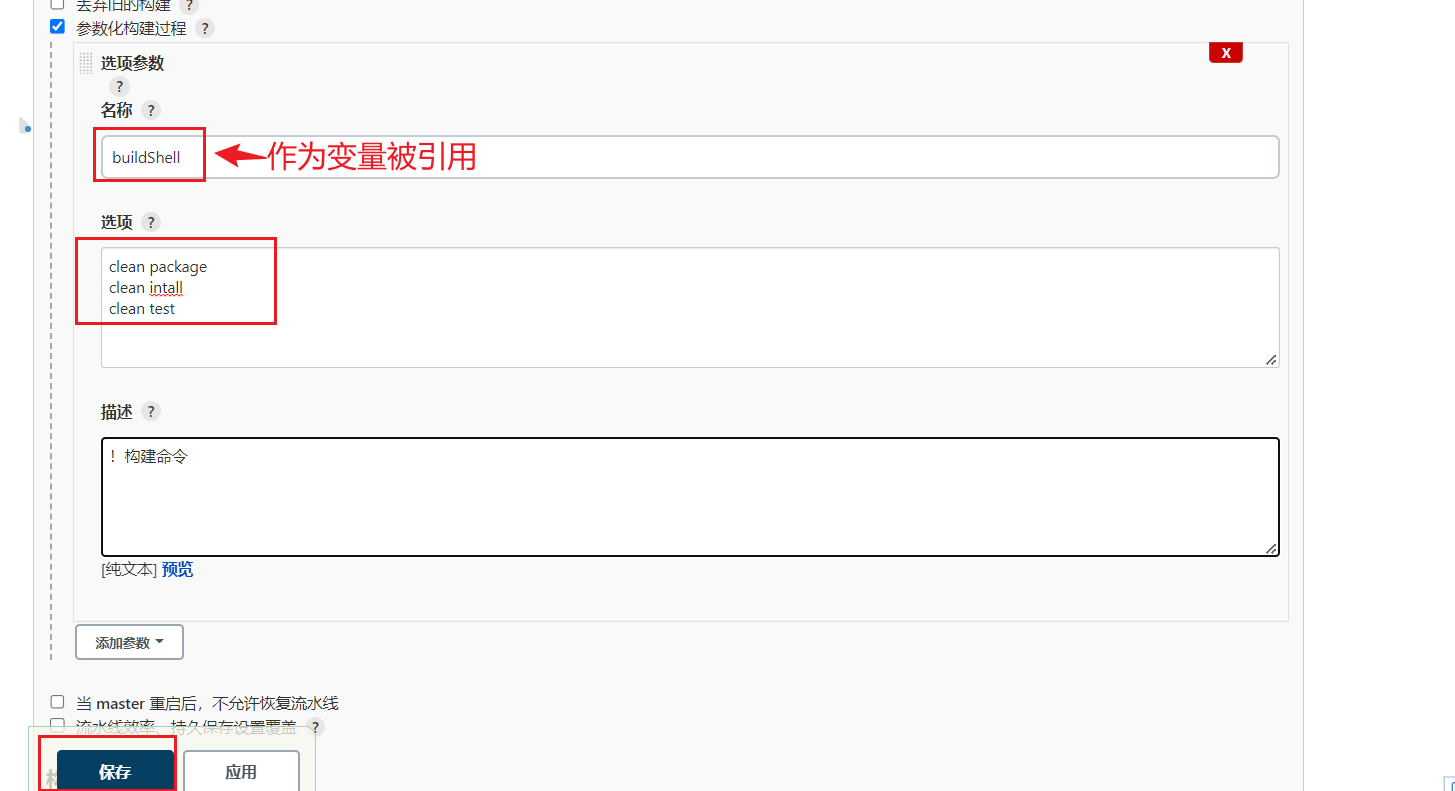
Gitlab 创建新文件ci.jenkinsfile

|  |
| --- |
| #!groovy  pipeline{  agent{ node { label "master"} }      stages{  stage("build"){  steps{  script{  mvnHome = tool "M3"  sh "${mvnHome}/bin/mvn -v"  }  }  }  }  } |





**6.2参数化构建**



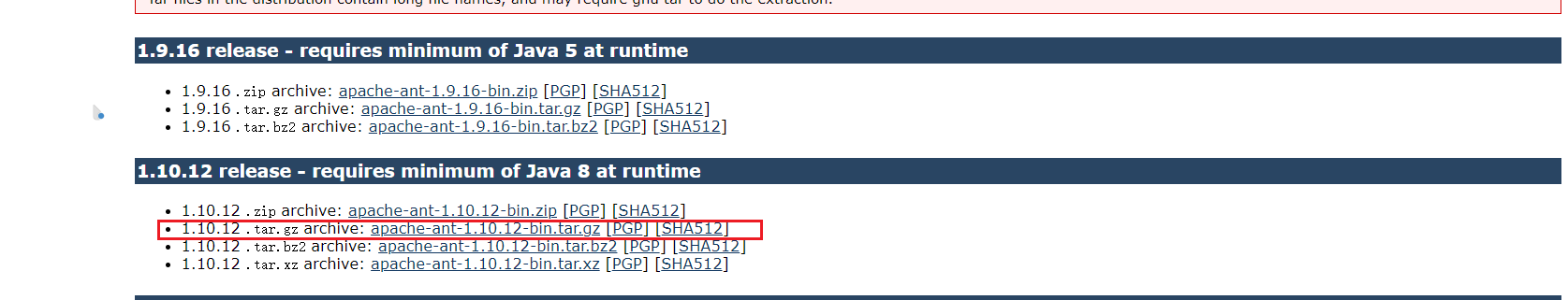
ci.jenkinsfile改为如下形式

|  |
| --- |
| #!groovy  String buildShell = "${env.buildShell}"  pipeline{  agent{ node { label "master"} }      stages{  stage("build"){  steps{  script{  mvnHome = tool "M3"  sh "${mvnHome}/bin/mvn ${buildShell}"  }  }  }  }  } |

# **集成Ant构建工具**

**7.1下载**

**<https://ant.apache.org/bindownload.cgi>**



**7.2安装ant**

tar zxf apache-ant-1.10.12-bin.tar.gz -C /usr/local/

cd /usr/local/ && ln -sfv apache-ant-1.10.12 ant

#添加全局变量（/etc/profile）

export ANT\_HOME=/usr/local/ant

export PATH=$PATH:$MAVEN\_HOME/bin:$ANT\_HOME/bin

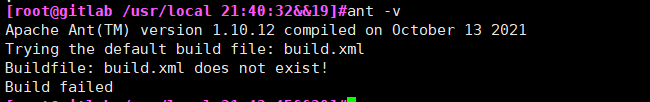
source /etc/profile

**测试**

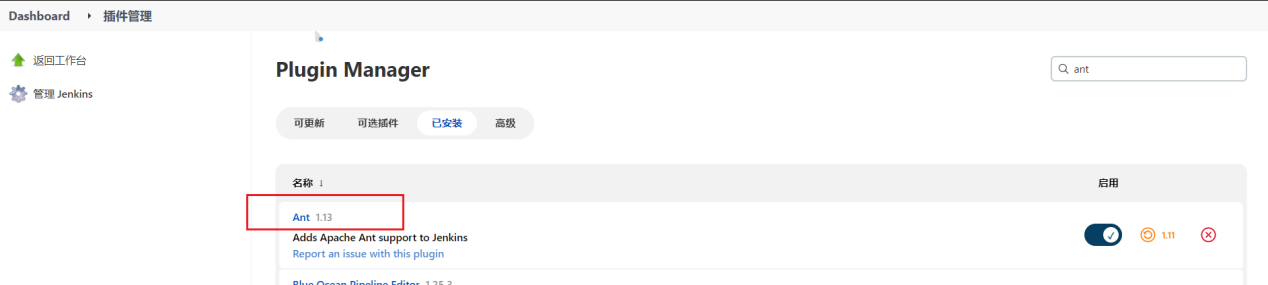
#ant -version



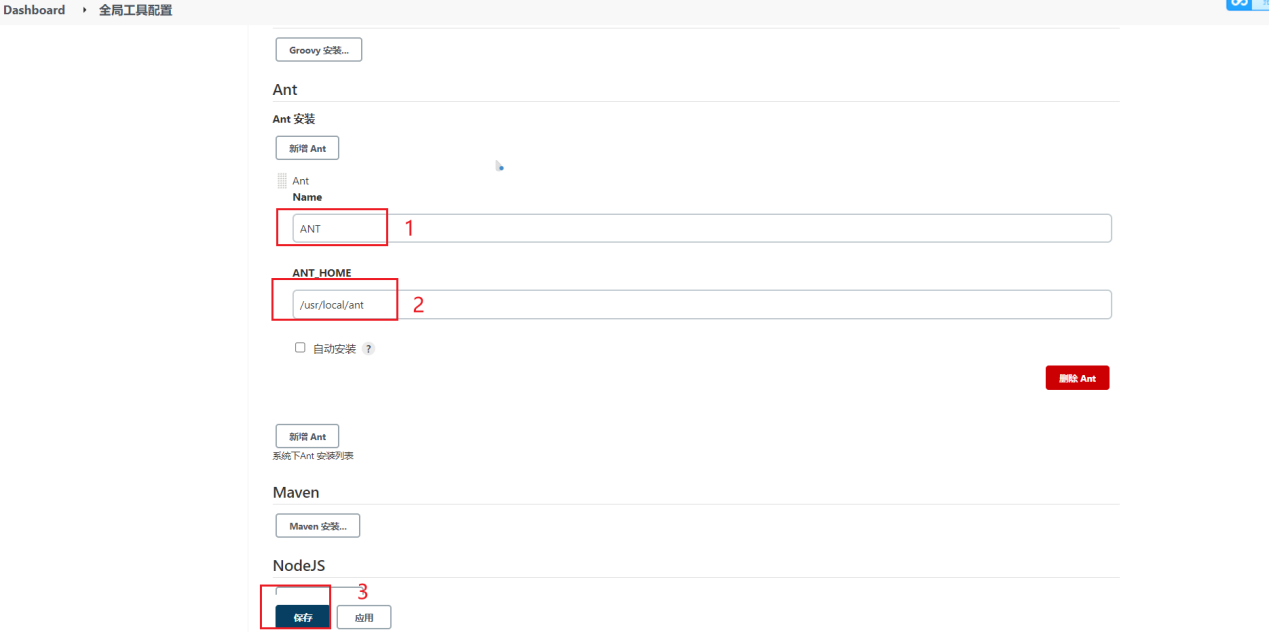
#ant -v



**7.3 jenkins确保已安装ant插件**



**7.4全局工具配置**



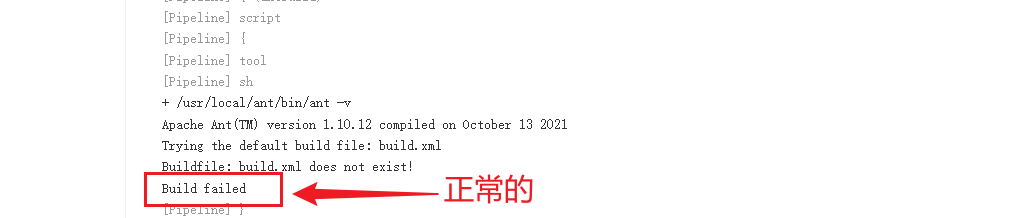
**7.5编辑ci.jenkinsfile**

|  |
| --- |
| #!groovy  String buildShell = "${env.buildShell}"  pipeline{  agent{ node { label "master"} }      stages{  stage("mavenbuild"){  steps{  script{  mvnHome = tool "M3"  sh "${mvnHome}/bin/mvn ${buildShell}"  }  }  }  stage("antbuild"){  steps{  script{  antHome = tool "ANT"  sh "${antHome}/bin/ant ${buildShell}"  }  }  }  }  } |

**7.6参数化构建**



**7.7构建验证**



**7.8 Ant常用命令**

ant -buildfile -f build.xml

# **集成Gradle构建工具**

**8.0 下载**

**<https://downloads.gradle.org/distributions/gradle-7.5-bin.zip>**

**8.1安装Gradle**

unzip gradle-5.3-bin.zip -d /usr/local/

cd /usr/local && ln -sfv gradle-7.5 gradle

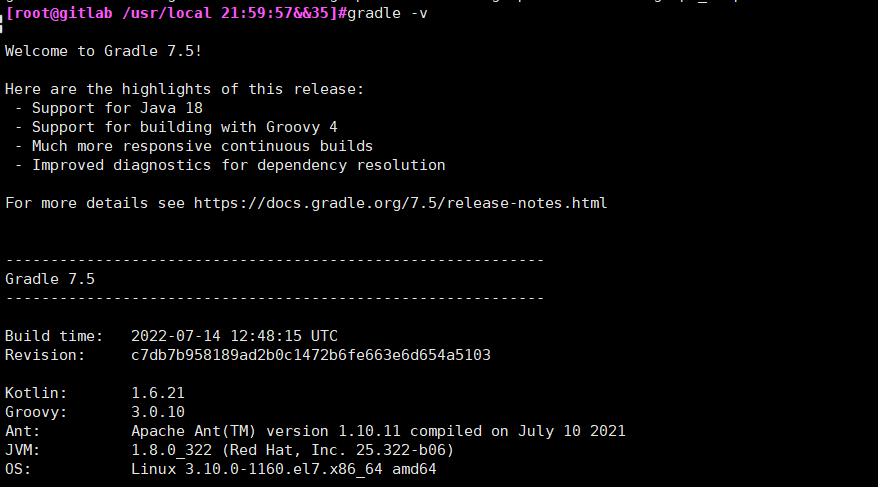
#添加全局变量（/etc/profile）

export GRADLE\_HOME=/usr/local/gradle

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

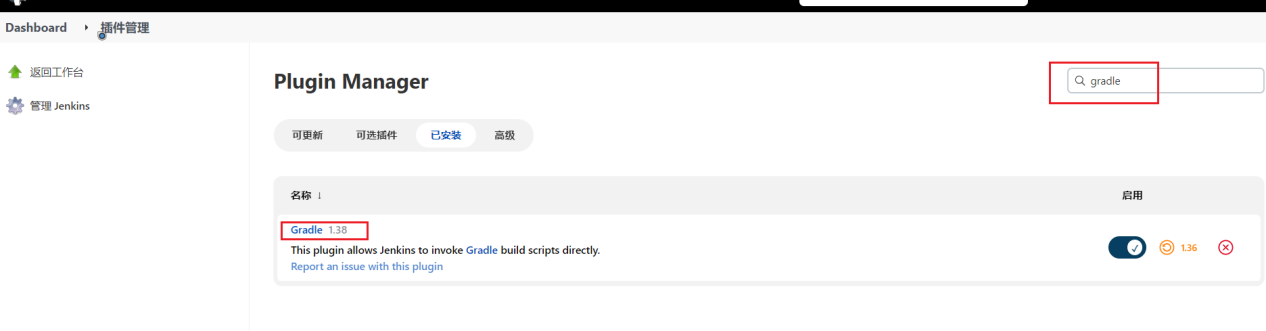
source /etc/profile

#gradle -v

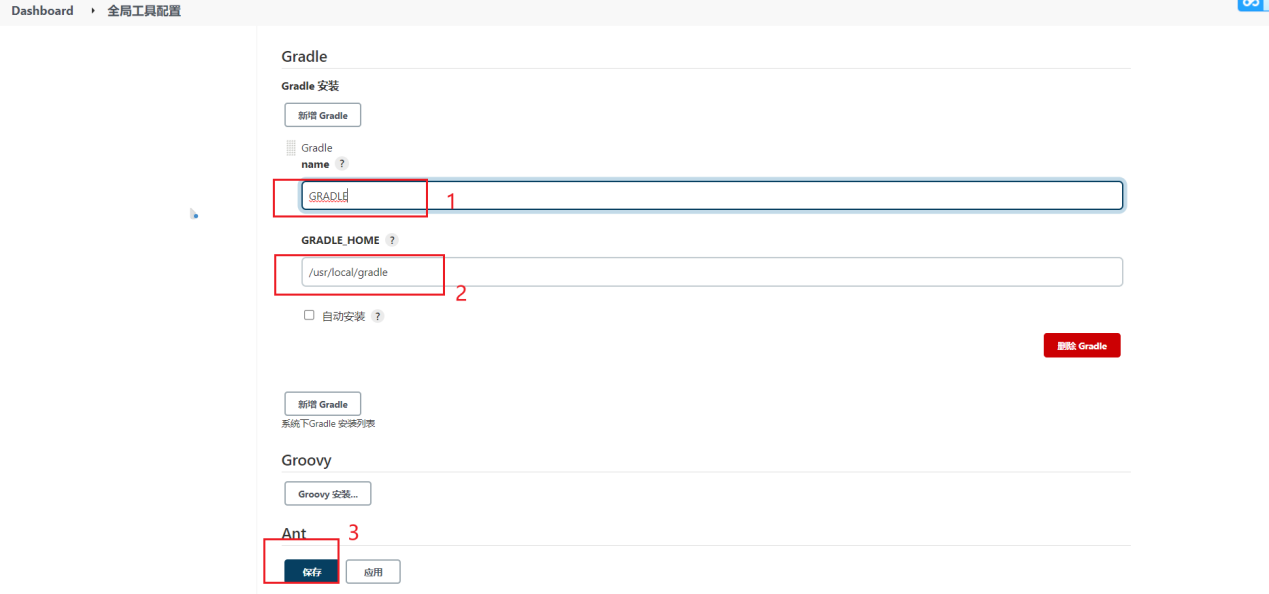


**8.3 Jenkins配置gradle**

**8.3.1确保已安装gradle插件**



**8.3.2全局工具配置**



**8.3.3编辑ci.jenkinsfile**

|  |
| --- |
| #!groovy  String buildShell = "${env.buildShell}"  pipeline{  agent{ node { label "master"} }      stages{  stage("mavenbuild"){  steps{  script{  mvnHome = tool "M3"  sh "${mvnHome}/bin/mvn ${buildShell}"  }  }  }  stage("antbuild"){  steps{  script{  antHome = tool "ANT"  sh "${antHome}/bin/ant ${buildShell}"  }  }  }  stage("gradlebuild"){  steps{  script{  gradleHome = tool "GRADLE"  sh "${gradleHome}/bin/gradle ${buildShell}"  }  }  }  }  } |



**8.3.4添加个捕获异常**

|  |
| --- |
| #!groovy  String buildShell = "${env.buildShell}"  pipeline{  agent{ node { label "master"} }      stages{  stage("mavenbuild"){  steps{  script{  mvnHome = tool "M3"  sh "${mvnHome}/bin/mvn ${buildShell}"  }  }  }  stage("antbuild"){  steps{  script{  try{  antHome = tool "ANT"  sh "${antHome}/bin/ant ${buildShell}"  } catch(e){  println(e)  }  }  }  }  stage("gradlebuild"){  steps{  script{  gradleHome = tool "GRADLE"  sh "${gradleHome}/bin/gradle ${buildShell}"  }  }  }  }  } |



**8.4 Gradle常用命令**

- ./gradlew -v 版本号，首次运行，没有gradle的要下载的哦。

- ./gradlew clean 删除HelloWord/app目录下的build文件夹

- ./gradlew build 检查依赖并编译打包

- ./gradlew assembleDebug 编译并打Debug包

- ./gradlew assembleRelease 编译并打Release的包

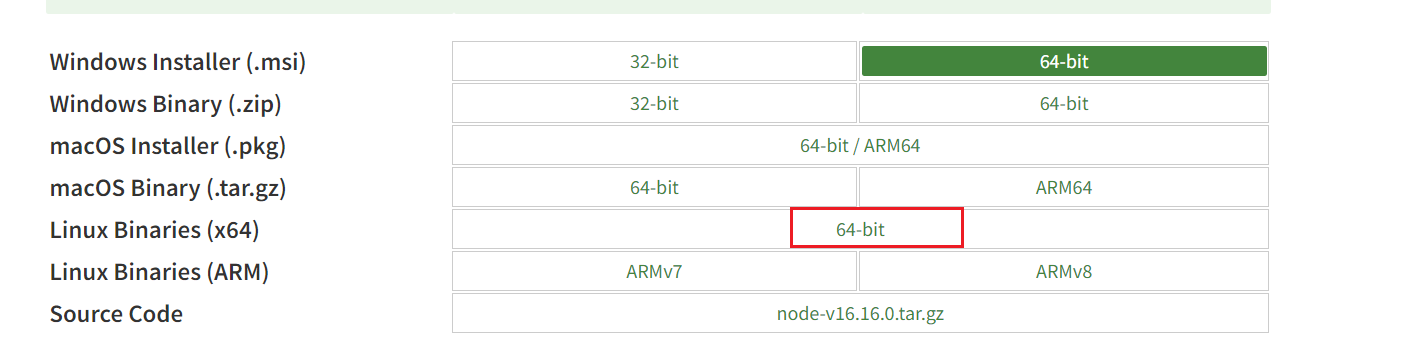
- ./gradlew installRelease Release模式打包并安装

- ./gradlew uninstallRelease 卸载Release模式包

# **集成Npm构建工具**

**9.0下载**

**<https://nodejs.org/en/download/>**



**9.1安装Node**

tar xf node-v16.16.0-linux-x64.tar.xz -C /usr/local/

cd /usr/local && ln -sfv node-v16.16.0-linux-x64 node

#添加全局变量（/etc/profile）

export NODE\_HOME=/usr/local/node

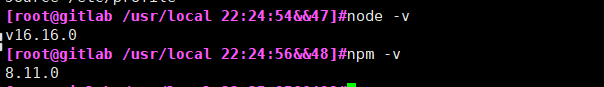
export PATH=$PATH:$NODE\_HOME/bin

source /etc/profile

验证

node -v

npm -v



**9.2 Jenkins配置Npm**

在Jenkins全局工具配置中并没有node，可以直接通过Jenkinsfile定义使用



**9.3 编辑ci.jenkinsfile**

|  |
| --- |
| #!groovy  String buildShell = "${env.buildShell}"  pipeline{  agent{ node { label "master"} }      stages{  stage("mavenbuild"){  steps{  script{  mvnHome = tool "M3"  sh "${mvnHome}/bin/mvn ${buildShell}"  }  }  }  stage("antbuild"){  steps{  script{  try{  antHome = tool "ANT"  sh "${antHome}/bin/ant ${buildShell}"  } catch(e){  println(e)  }  }  }  }  stage("gradlebuild"){  steps{  script{  gradleHome = tool "GRADLE"  sh "${gradleHome}/bin/gradle ${buildShell}"  }  }  }  stage("npmbuild"){  steps{  script{  npmHome = tool "NPM"  sh "${npmHome}/bin/npm ${buildShell}"  }  }  }  }  } |

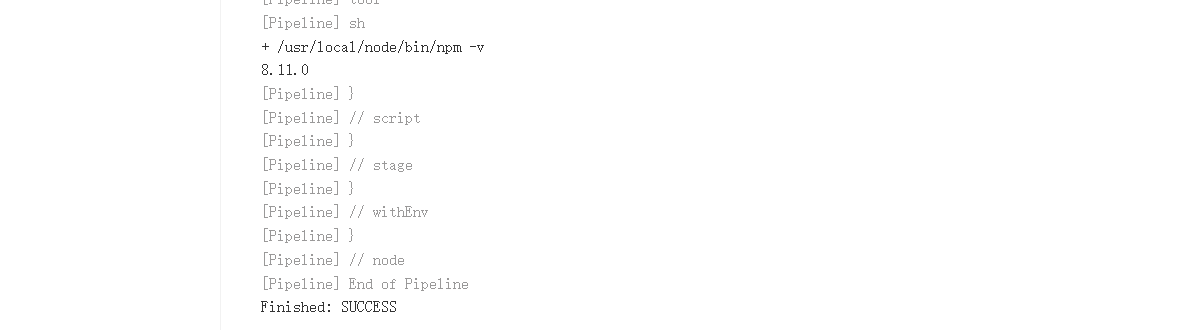
当命令不生效时。可以使用export声明。



**9.4构建前需要执行**

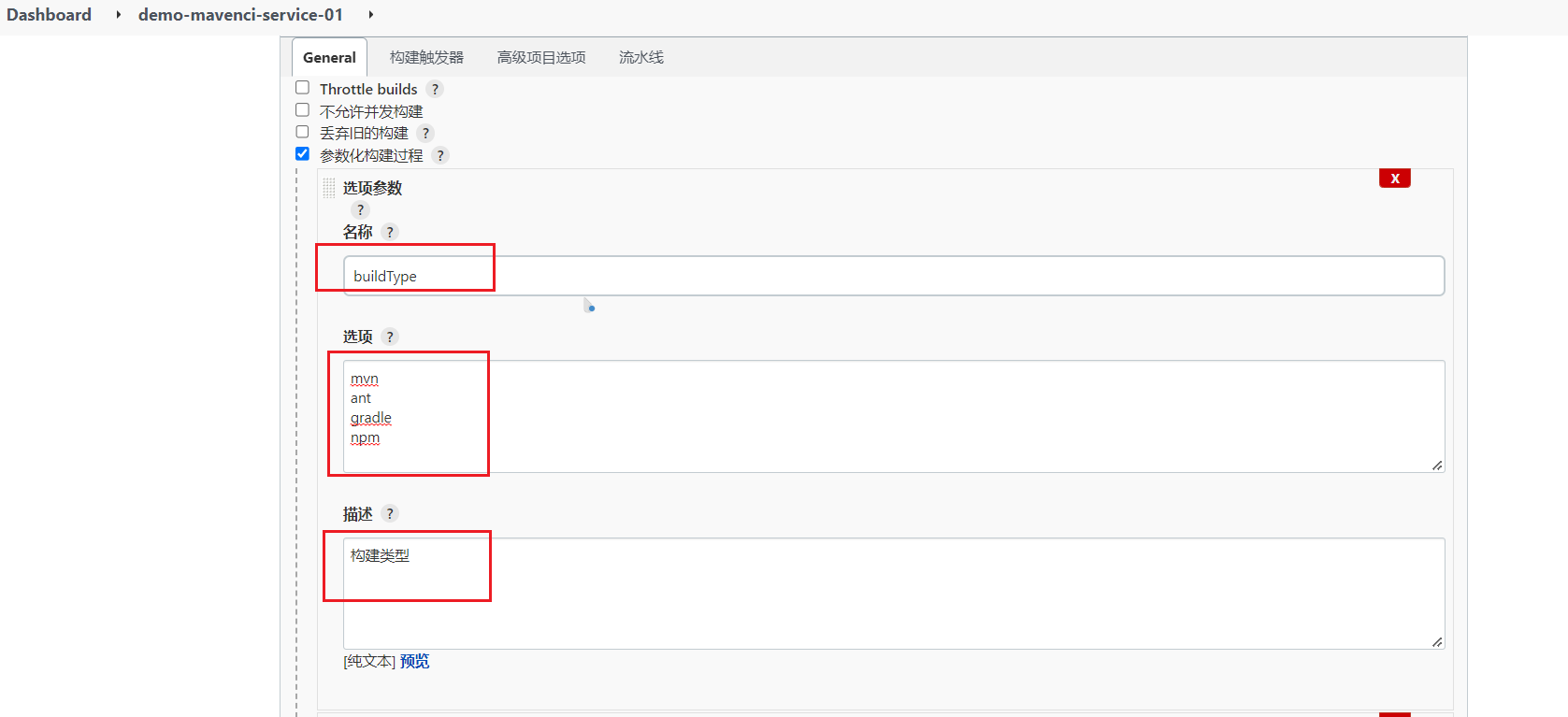
npm install -g npm

**9.5构建验证**



# **共享库方式整合构建工具**

**10.1 Jenkins配置构建类型参数**



**10.2 gitlab新建文件build.groovy**



|  |
| --- |
| package org.devops  //构建类型  def Build(buildType,buildShell){  def buildTools = ["mvn":"M3","ant":"ANT","gradle":"GRADLE","npm":"NPM"]      println("当前选择的构建类型为 ${buildType}")  buildHome= tool buildTools[buildType]    sh "${buildHome}/bin/${buildType} ${buildShell}"  } |

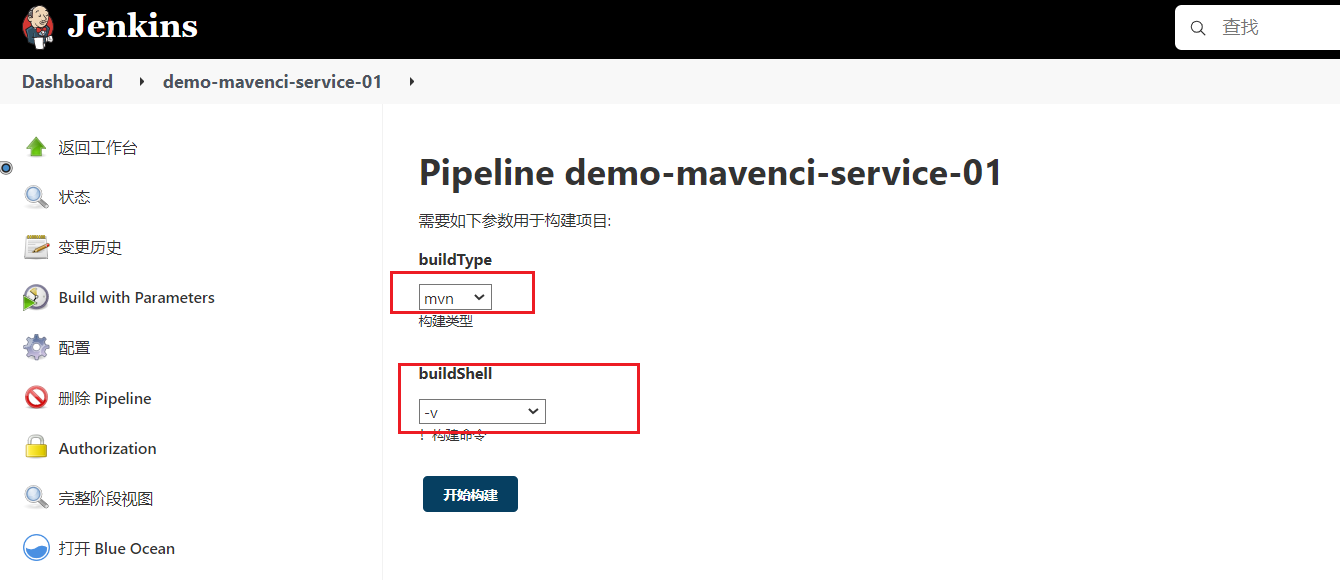
**当NPM构建失败可以使用下面方式 加了个if判断**

|  |
| --- |
| package org.devops  //构建类型  def Build(buildType,buildShell){  def buildTools = ["mvn":"M3","ant":"ANT","gradle":"GRADLE","npm":"NPM"]      println("当前选择的构建类型为 ${buildType}")  buildHome= tool buildTools[buildType]    if ("${buildType}" == "npm"){    sh """  export NODE\_HOME=${buildHome}  export PATH=\$NODE\_HOME/bin:\$PATH  ${buildHome}/bin/${buildType} ${buildShell}"""  } else {  sh "${buildHome}/bin/${buildType} ${buildShell}"  }  } |

**10.3 ci.jenkinsfile引入共享库**

|  |
| --- |
| #!groovy  @Library('jenkinslib@master') \_ #引用共享库jenkinslib  def build = new org.devops.build() #定义build函数 引用build.groovy  String buildType = "${env.buildType}" #引用Jenkins里定义的构建参数buildType  String buildShell = "${env.buildShell}"  pipeline{  agent{ node { label "master"} }      stages{  stage("build"){  steps{  script{  build.Build(buildType,buildShell) #引用build.groovy里面的Build  }  }  }  /\*stage("antbuild"){ #注释开头/\*  steps{  script{  try{  antHome = tool "ANT"  sh "${antHome}/bin/ant ${buildShell}"  } catch(e){  println(e)  }  }  }  }  stage("gradlebuild"){  steps{  script{  gradleHome = tool "GRADLE"  sh "${gradleHome}/bin/gradle ${buildShell}"  }  }  }  stage("npmbuild"){  steps{  script{  npmHome = tool "NPM"  sh "${npmHome}/bin/npm ${buildShell}"  }  }  }\*/ #注释结尾\*/  }  } |

**10.4构建验证**

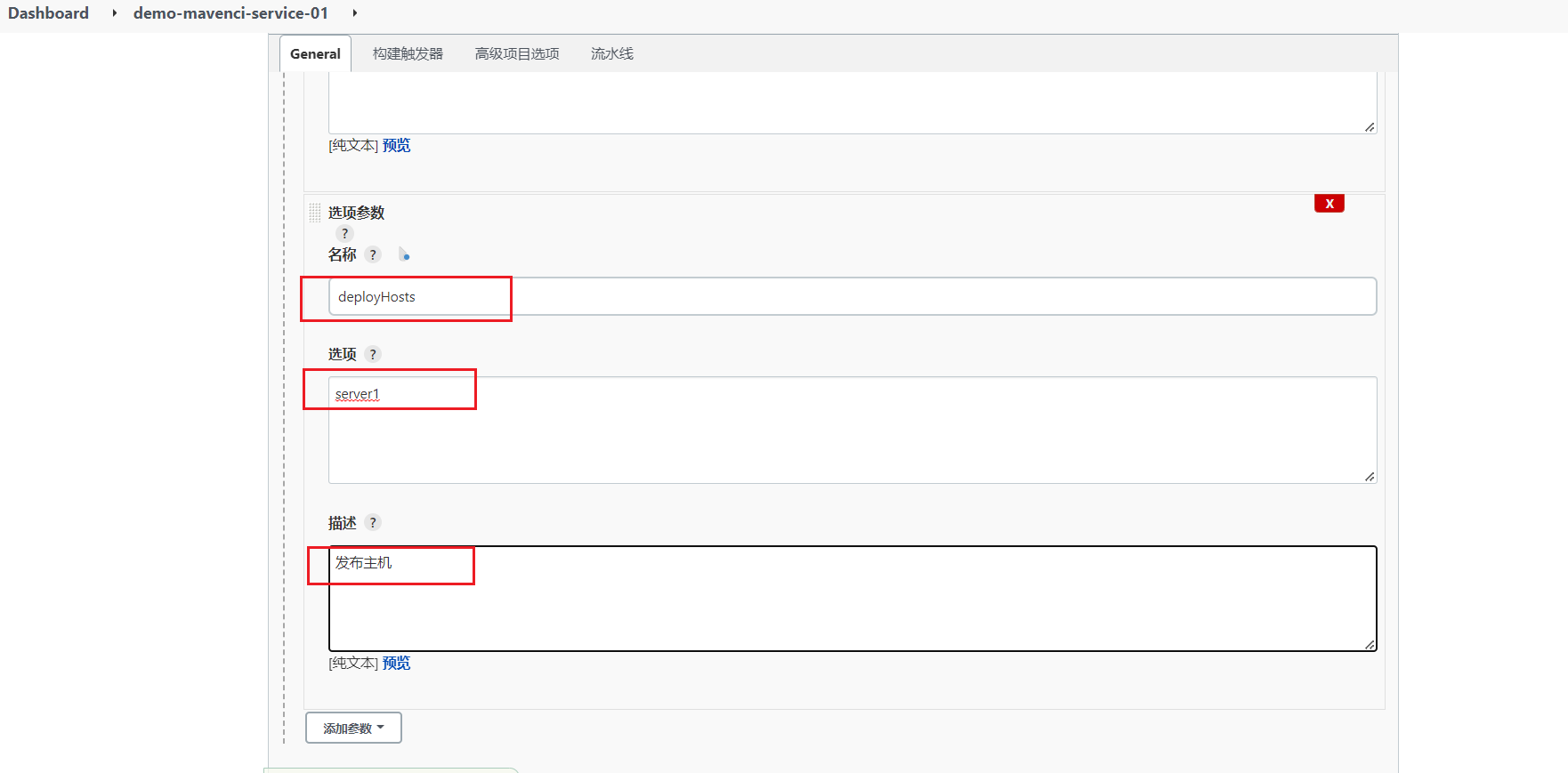




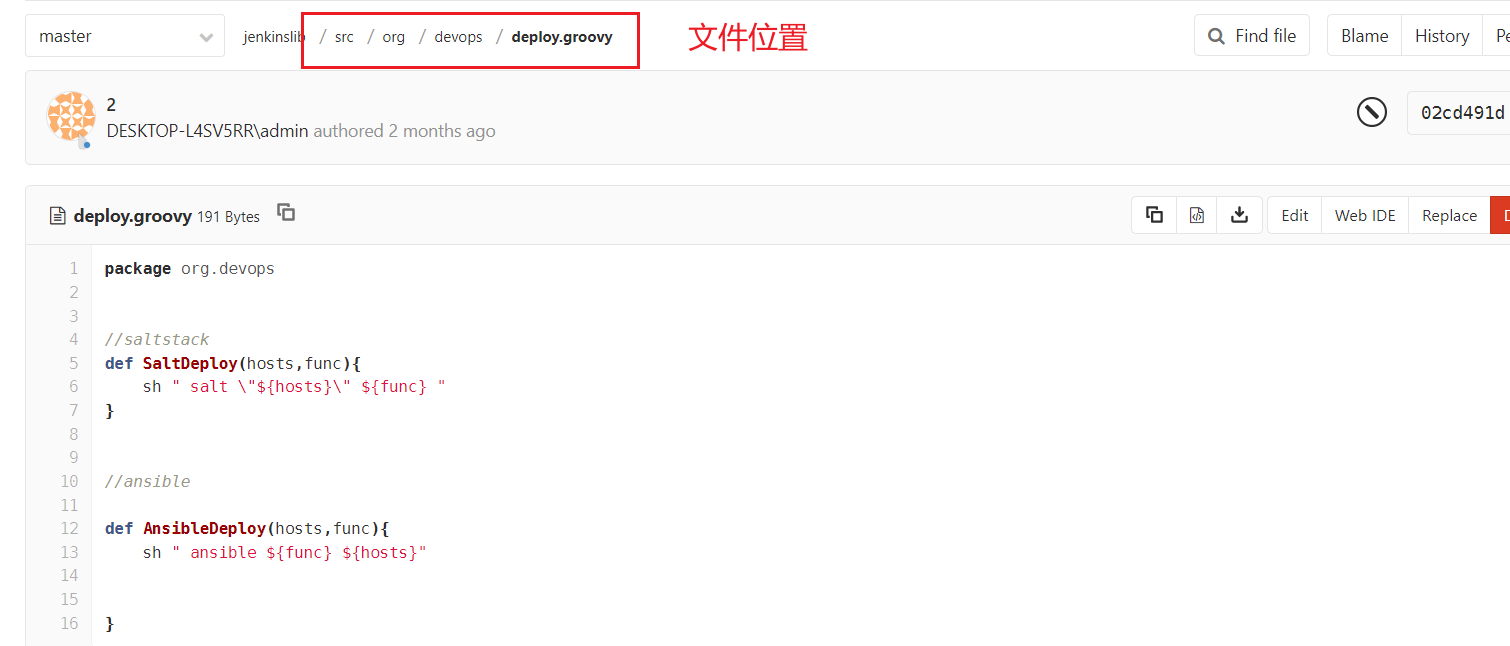
# **集成Ansible部署工具**

**11.1 添加构建参数**

**deployHosts**



**11.2 编辑 [deploy.groovy](https://gitlab.itdance.cn/devops/jenkinslib/blob/master/src/org/devops/deploy.groovy)**



|  |
| --- |
| package org.devops  //saltstack  def SaltDeploy(hosts,func){  sh " salt \"${hosts}\" ${func} "  }  //ansible  def AnsibleDeploy(hosts,func){  sh " ansible ${func} ${hosts}"      } |

**11.3 编辑ci.jenkinsfile**

蓝色部分为新增

|  |
| --- |
| #!groovy  @Library('jenkinslib@master') \_  def build = new org.devops.build()  def deploy = new org.devops.deploy()  String buildType = "${env.buildType}"  String buildShell = "${env.buildShell}"  String deployHosts= "${env.deployHosts}"  pipeline{  agent{ node { label "master"} }      stages{  stage("build"){  steps{  script{  build.Build(buildType,buildShell)  deploy.AnsibleDeploy("${deployHosts}","-m ping")  }  }  }  /\*stage("antbuild"){  steps{  script{  try{  antHome = tool "ANT"  sh "${antHome}/bin/ant ${buildShell}"  } catch(e){  println(e)  }  }  }  }  stage("gradlebuild"){  steps{  script{  gradleHome = tool "GRADLE"  sh "${gradleHome}/bin/gradle ${buildShell}"  }  }  }  stage("npmbuild"){  steps{  script{  npmHome = tool "NPM"  sh "${npmHome}/bin/npm ${buildShell}"  }  }  }\*/  }  } |

**11.4构建验证**

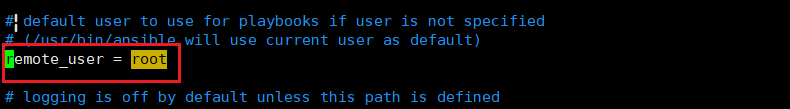
**报错**

"msg": "Failed to connect to the host via ssh: Permission denied (publickey,password).",

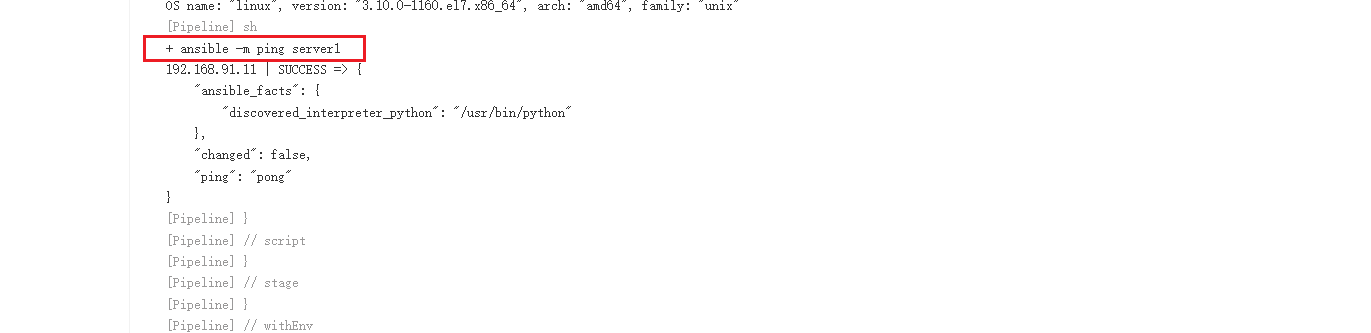
在集成jenkins和ansible实现自动化部署时，root用户下执行ansible命令时可以正常运行。由于是通过jenkins用户去执行ansible命令，而jenkins用户却报如下异常

**解决办法**

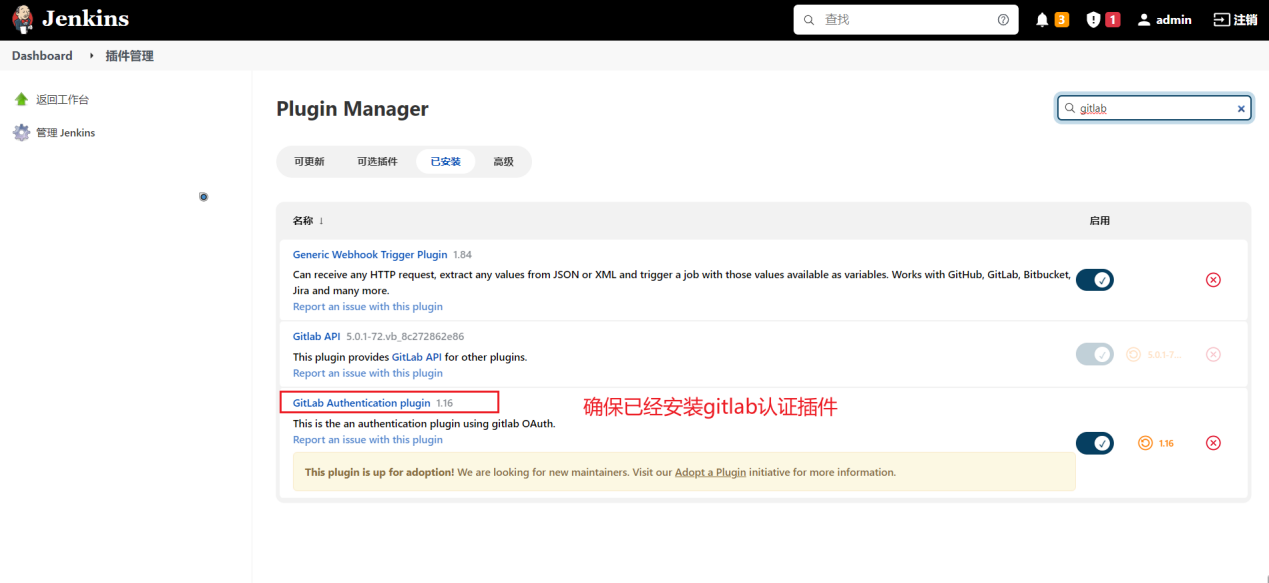
vim /etc/ansible/ansible.cfg

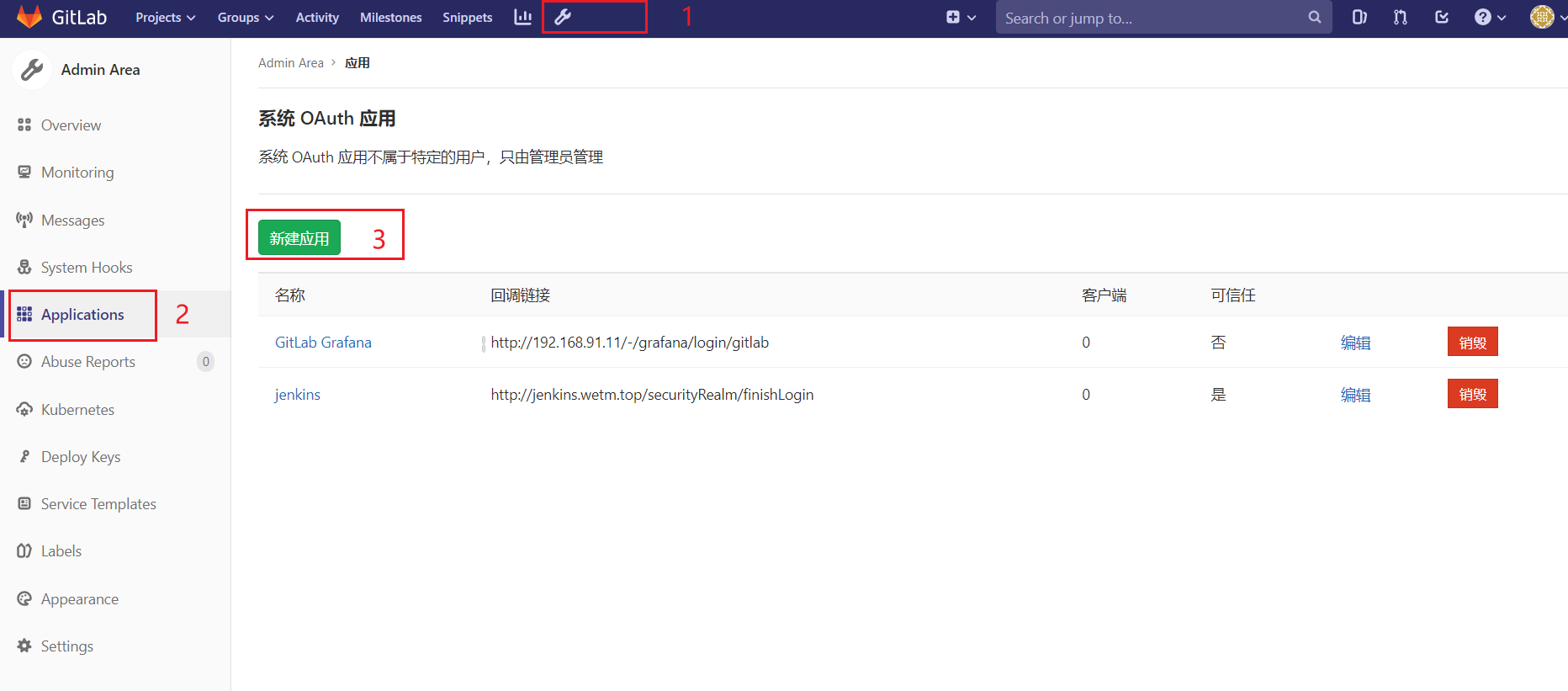


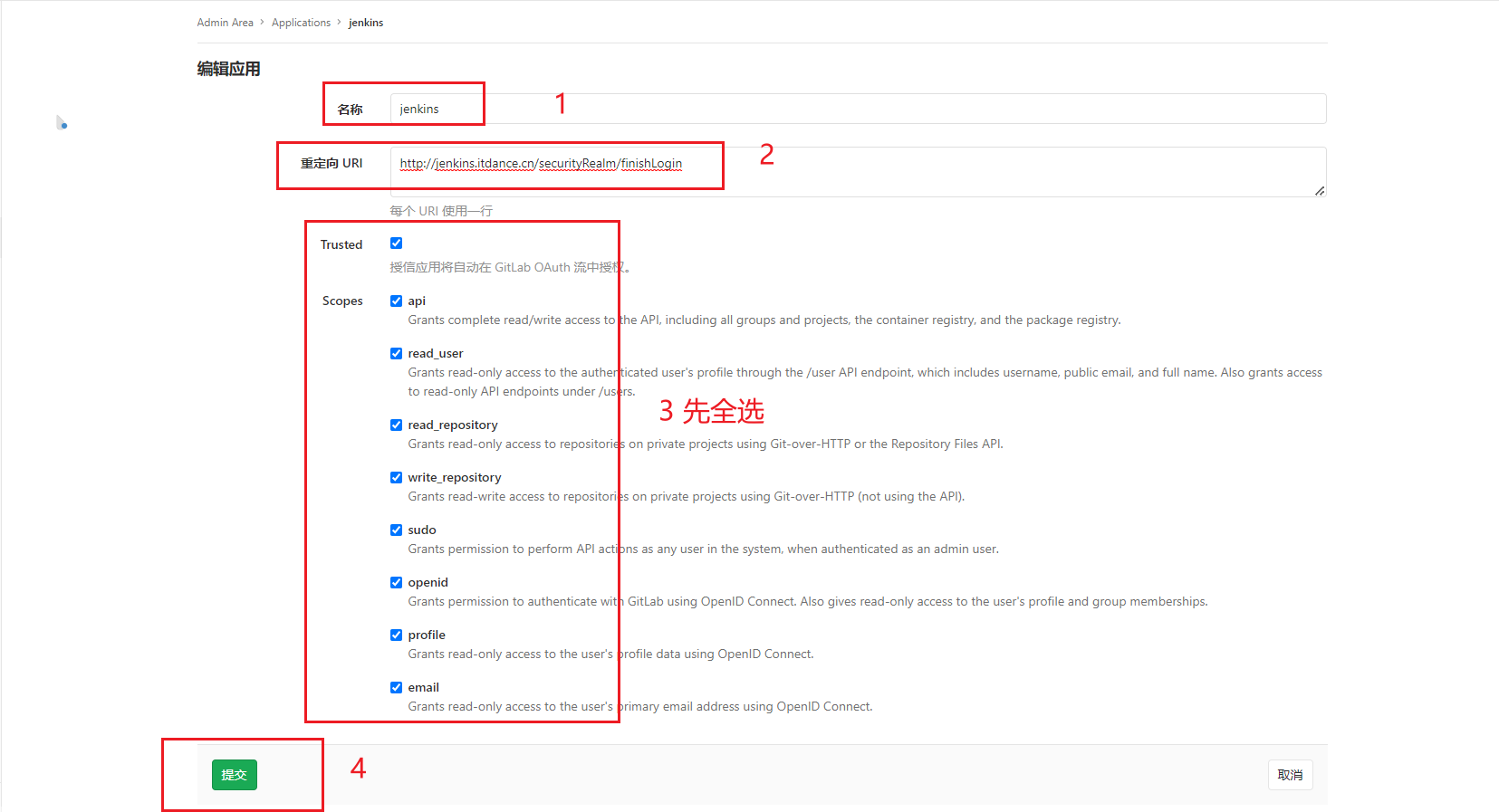
remote\_user = root 前的注释去掉

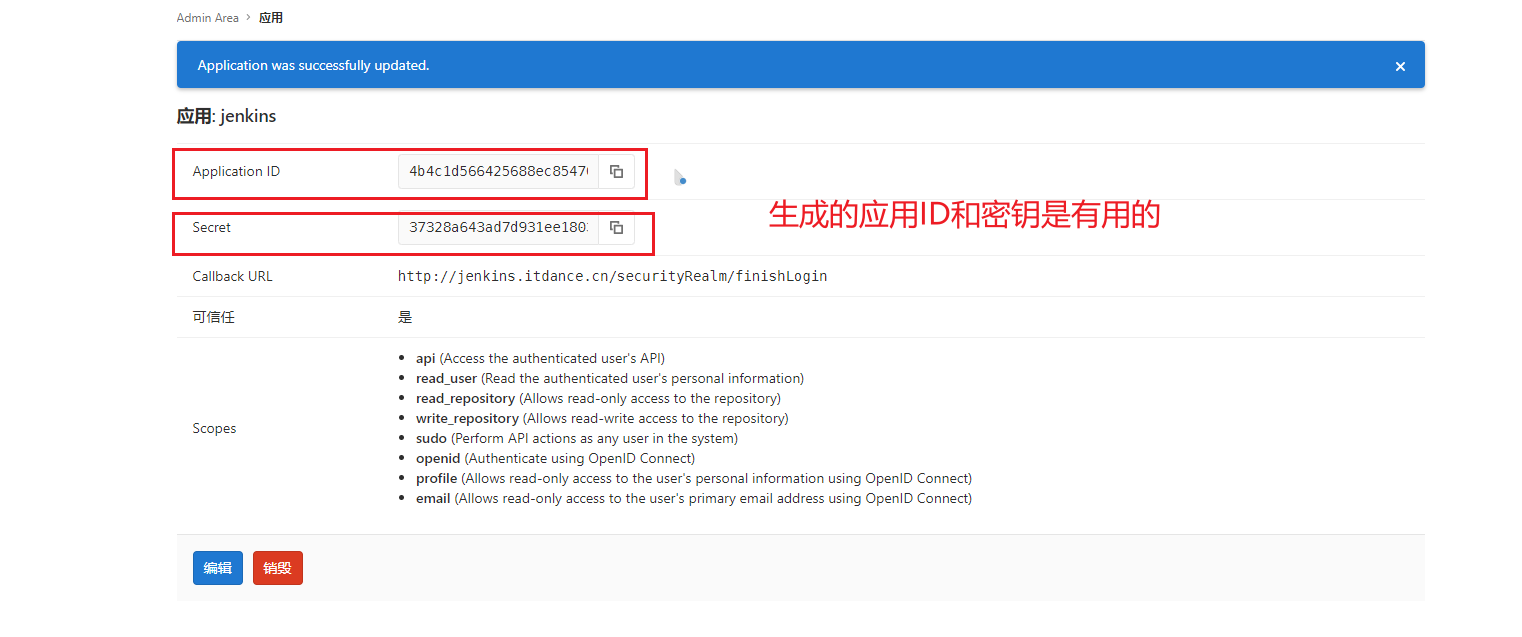


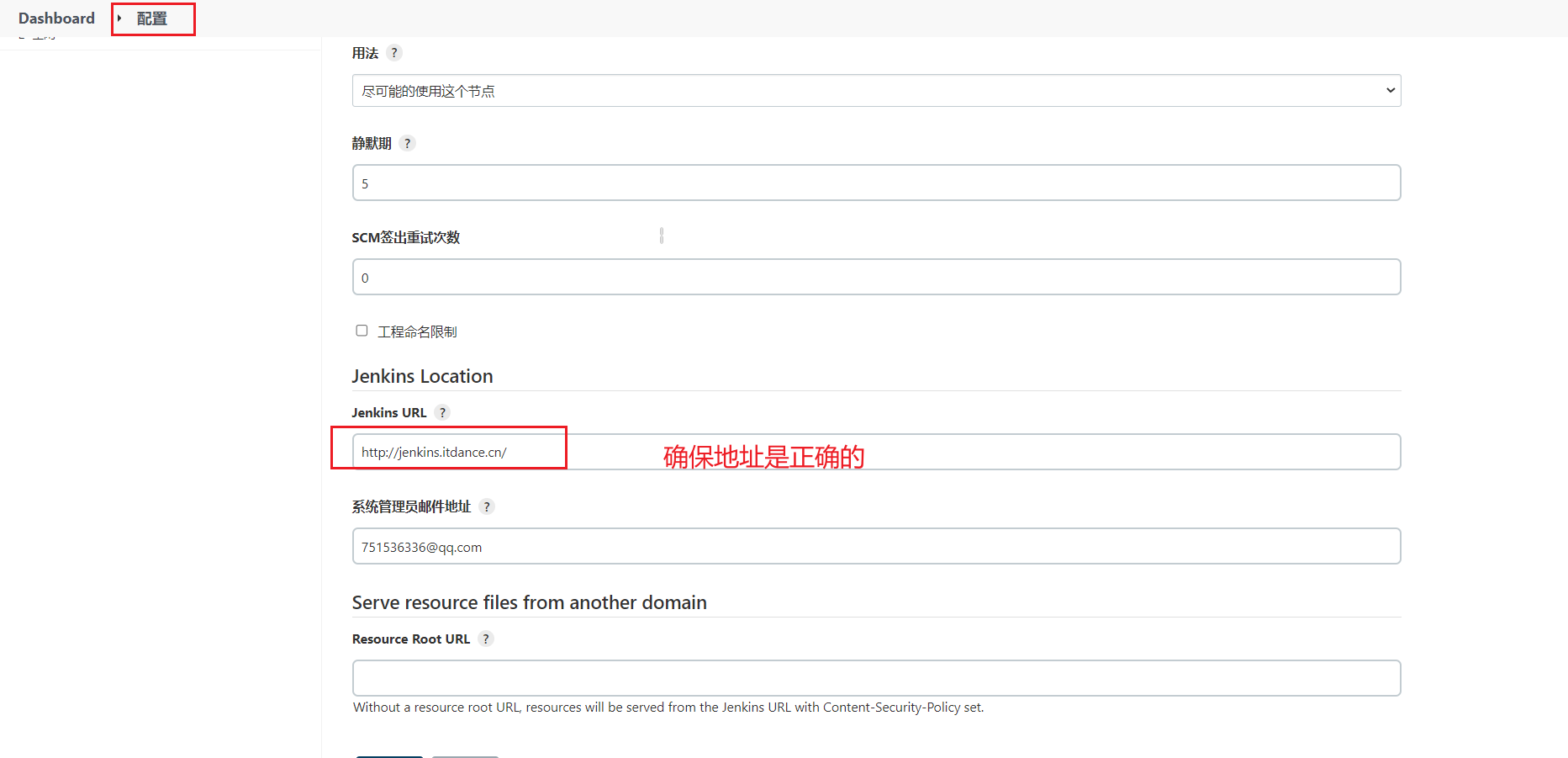
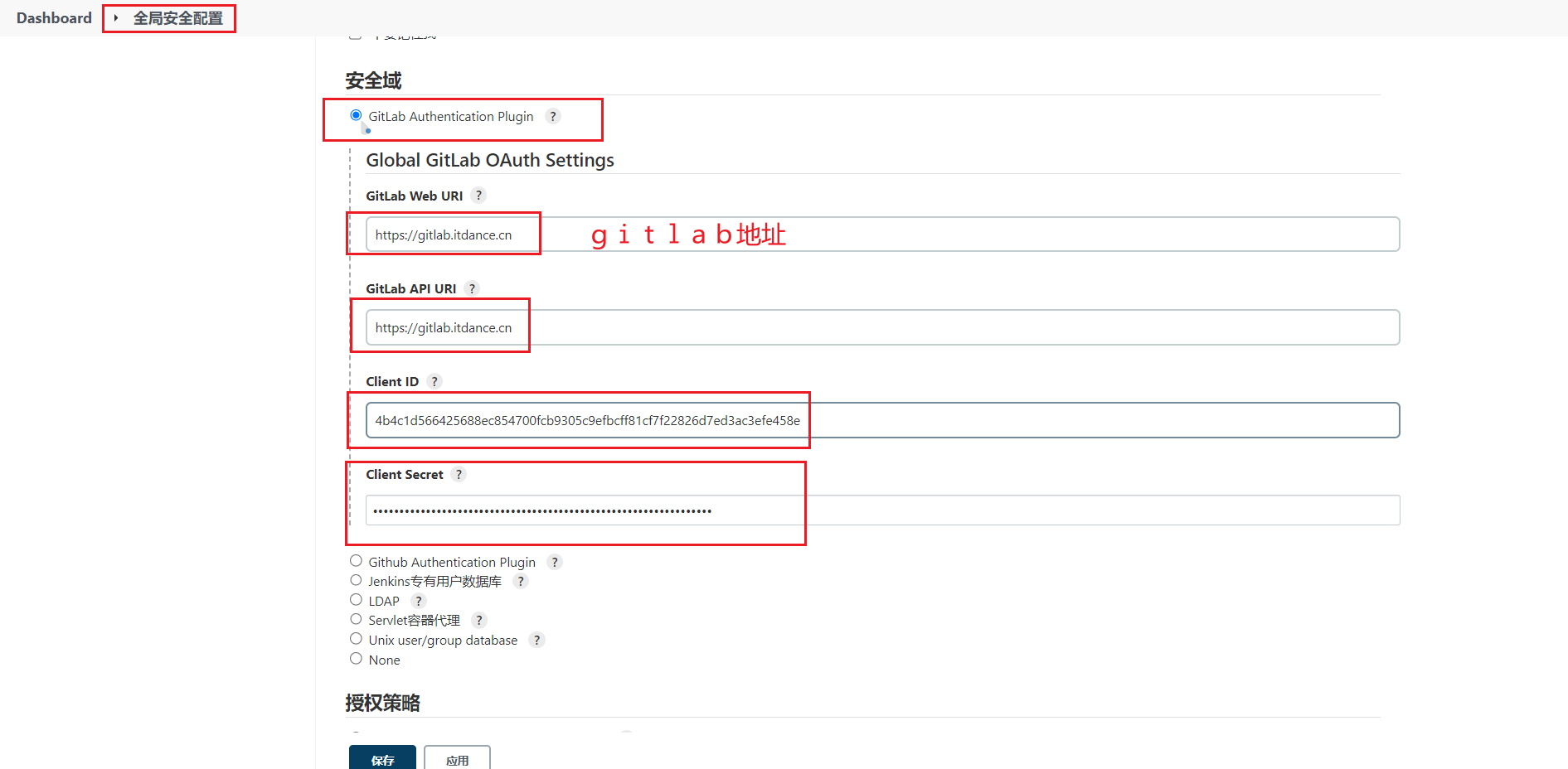
# **GitlabSSO用户认证集成**











验证

注销Jenkins 用gitlab用户登陆

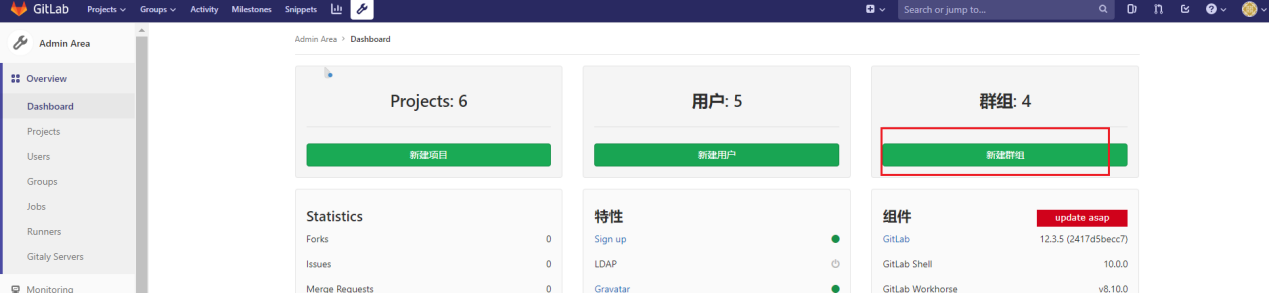
# **项目准备与流水线调试**

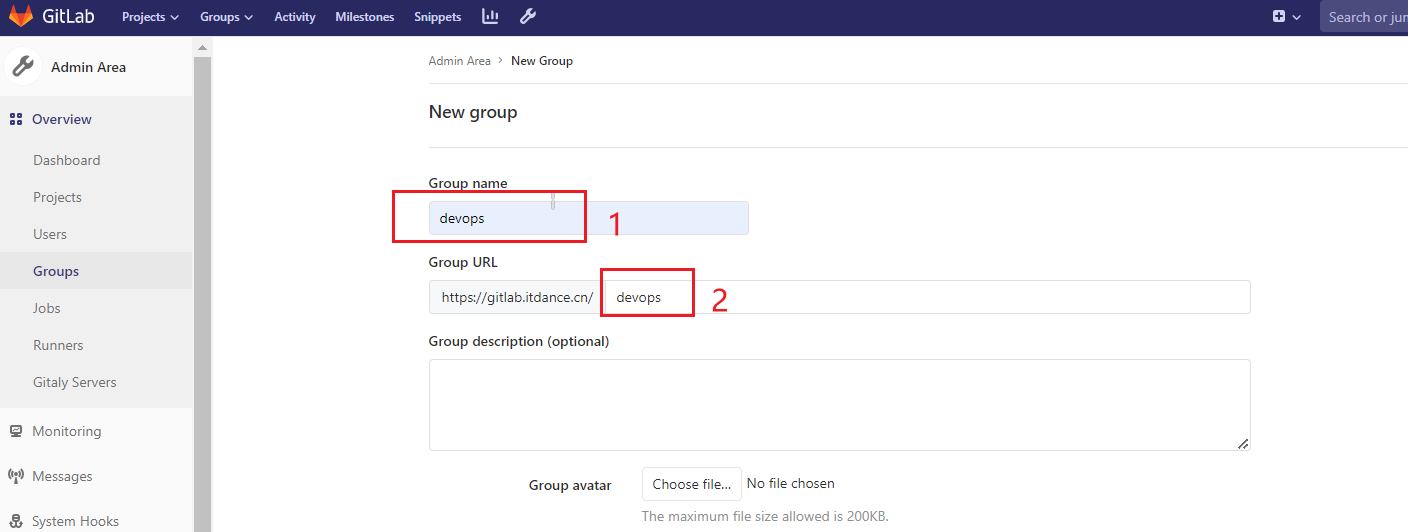
**13.1 准备demo**

**<https://github.com/zeyangli/gitlabci-demo-maven-service>**

**提交到gitlab**

gitlab创建devops 群组

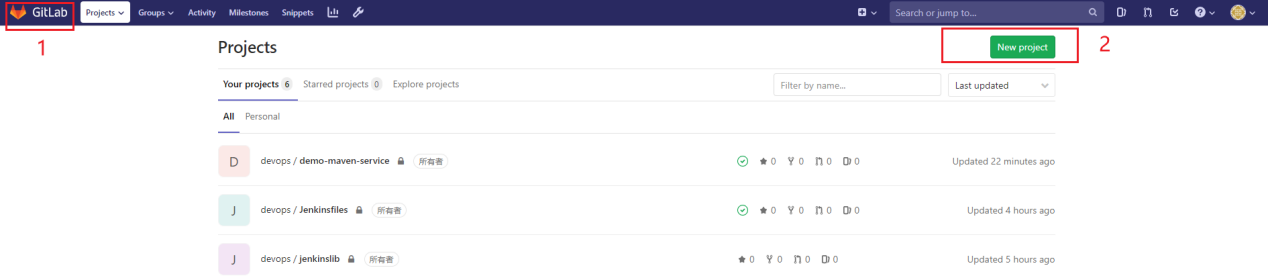


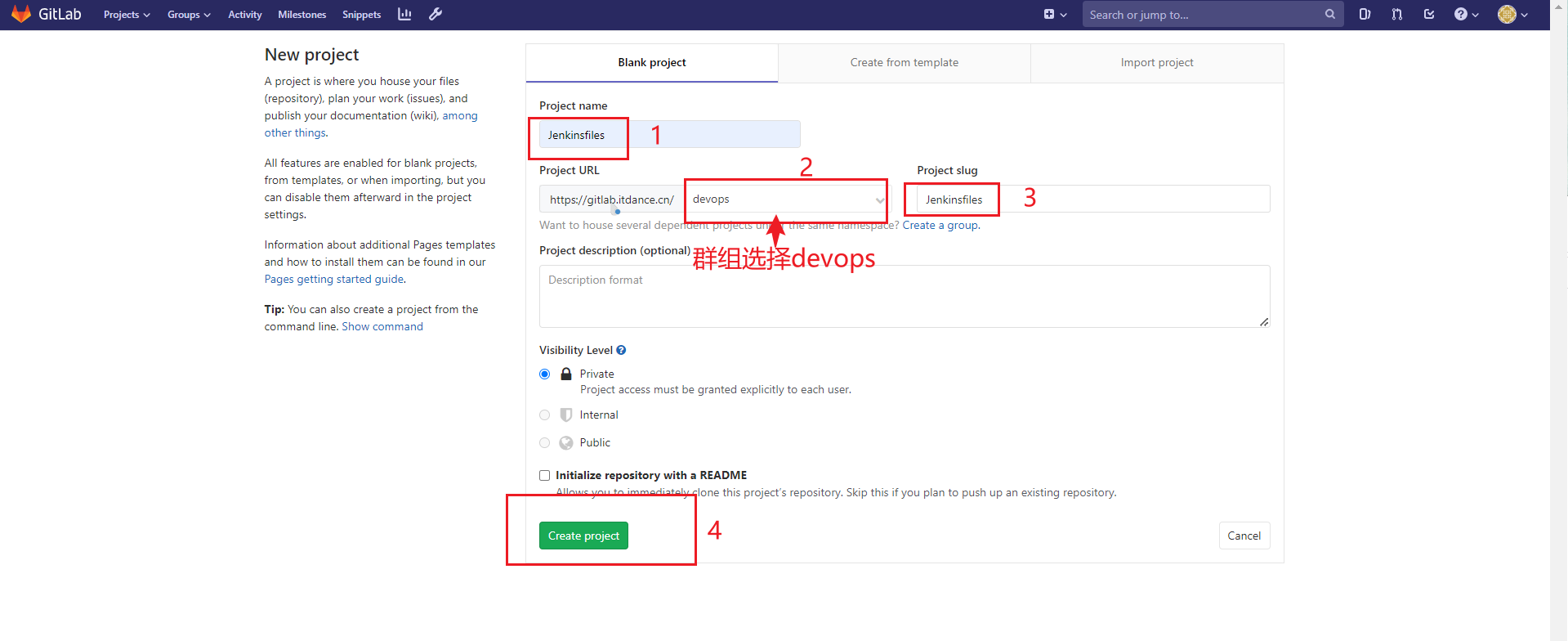


创建相关项目

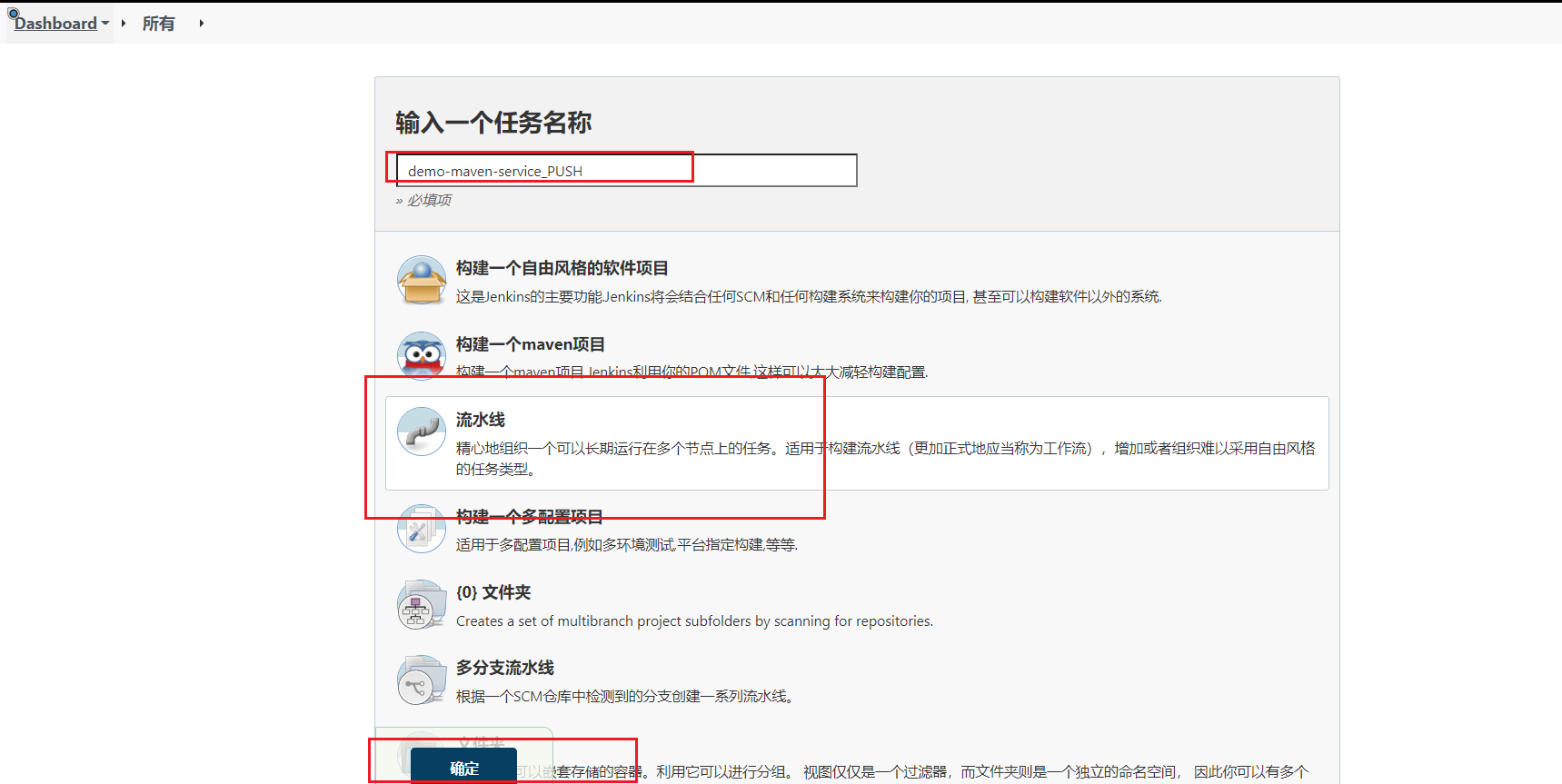


比如创建Jenkinsfile项目



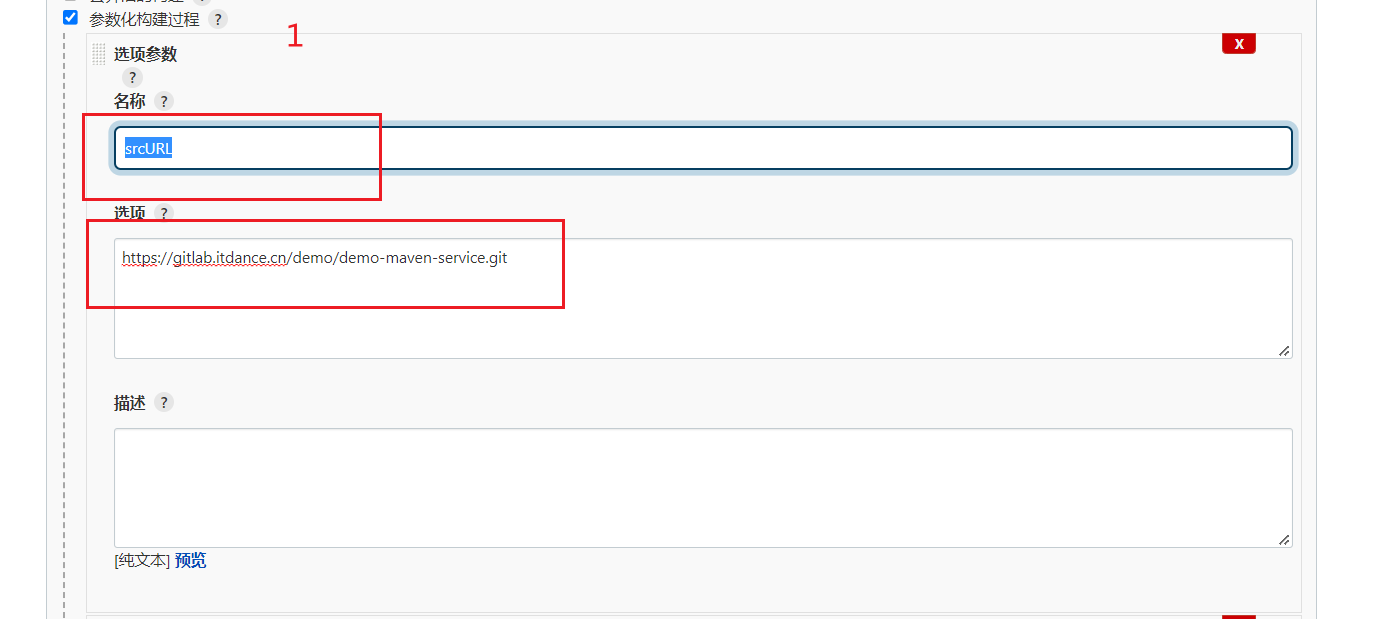


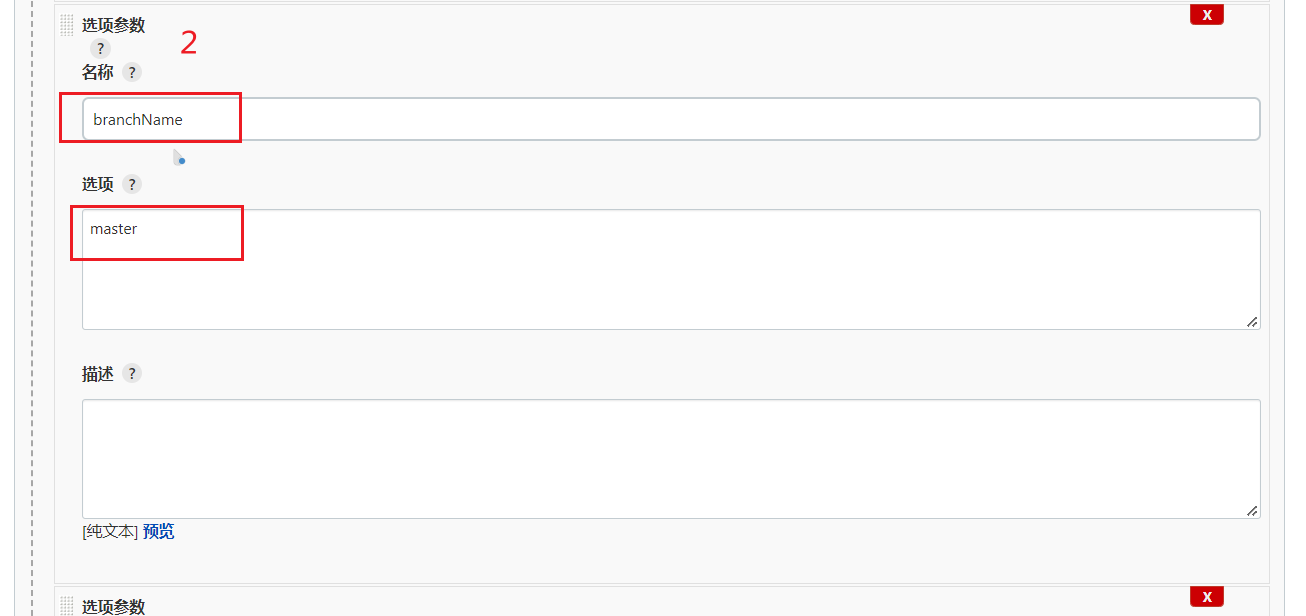
**13.2 Jenkins新建流水线任务**



**13.2.1 新建流水线任务配置**

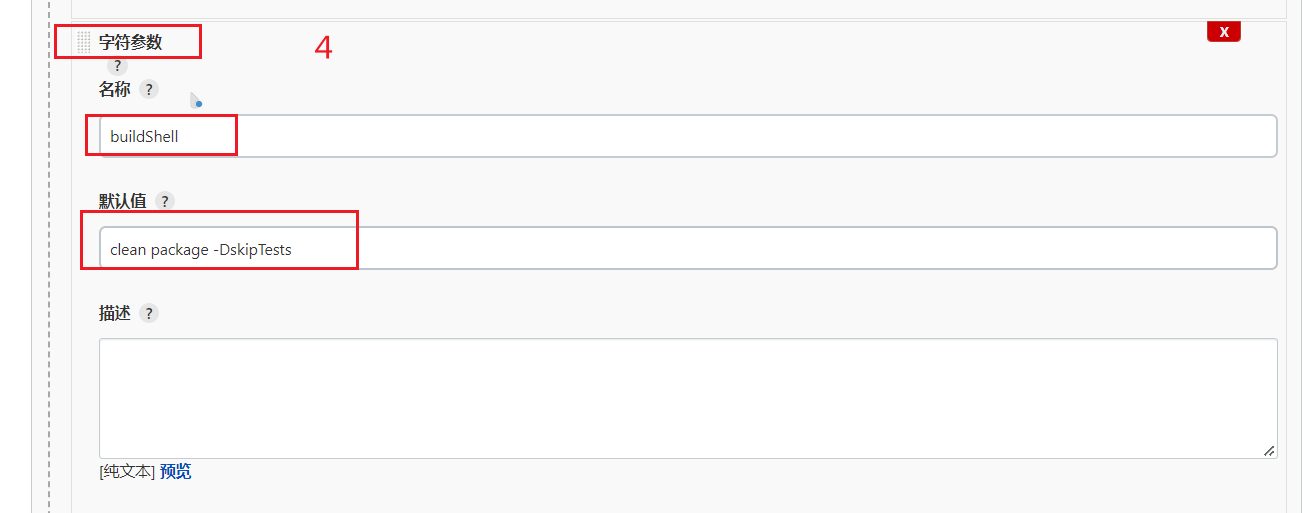
**添加构建参数**

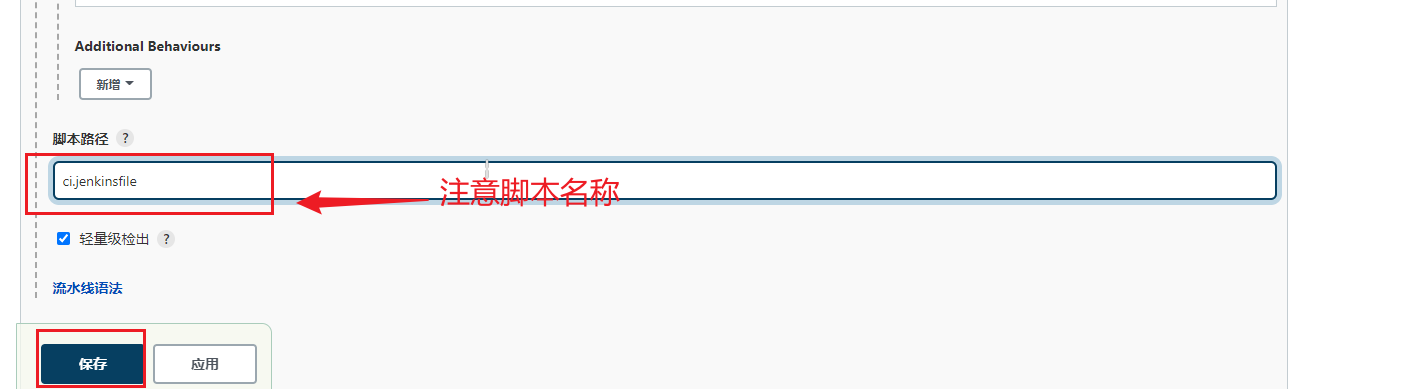
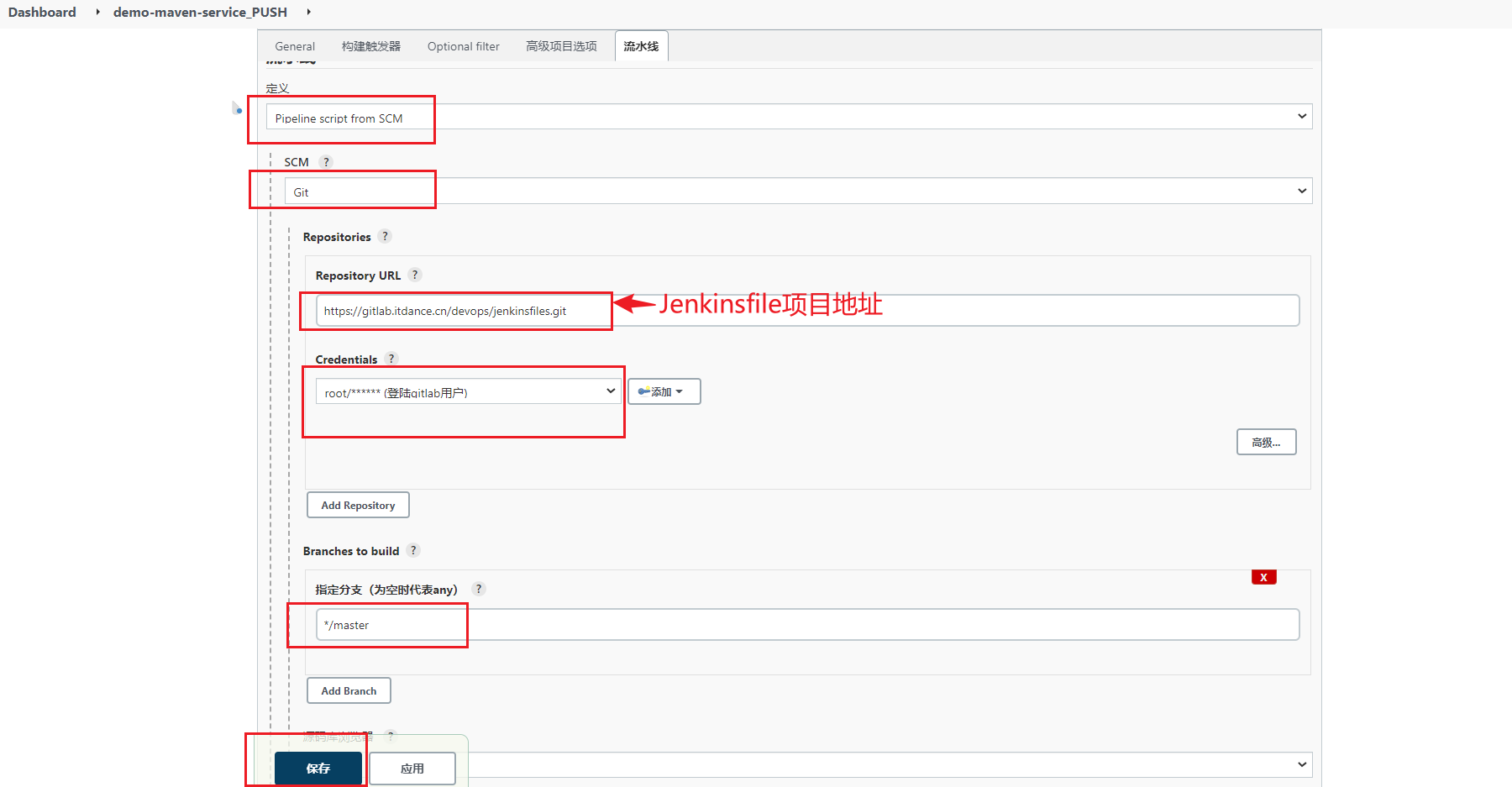




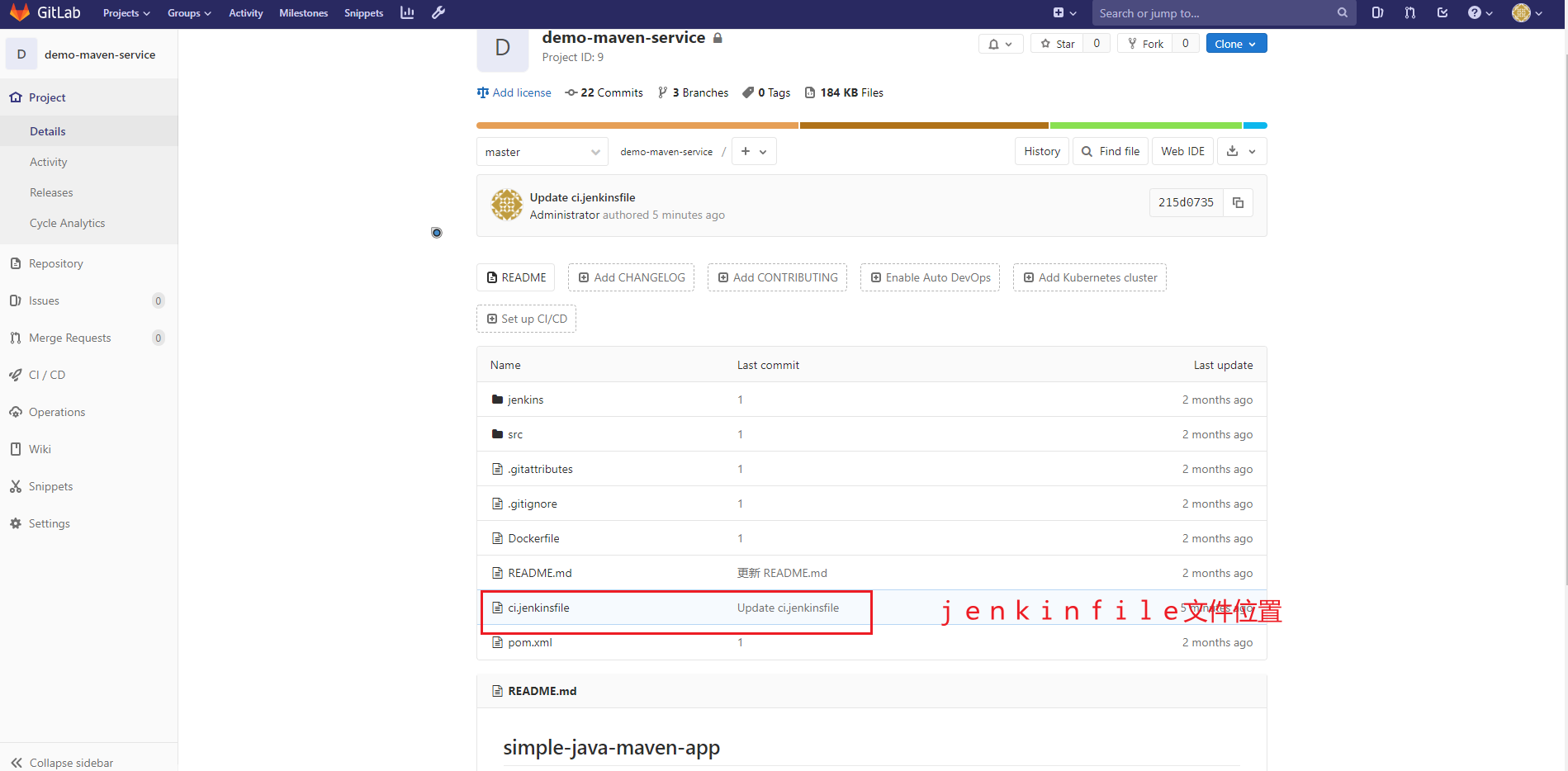


第4个是字符参数

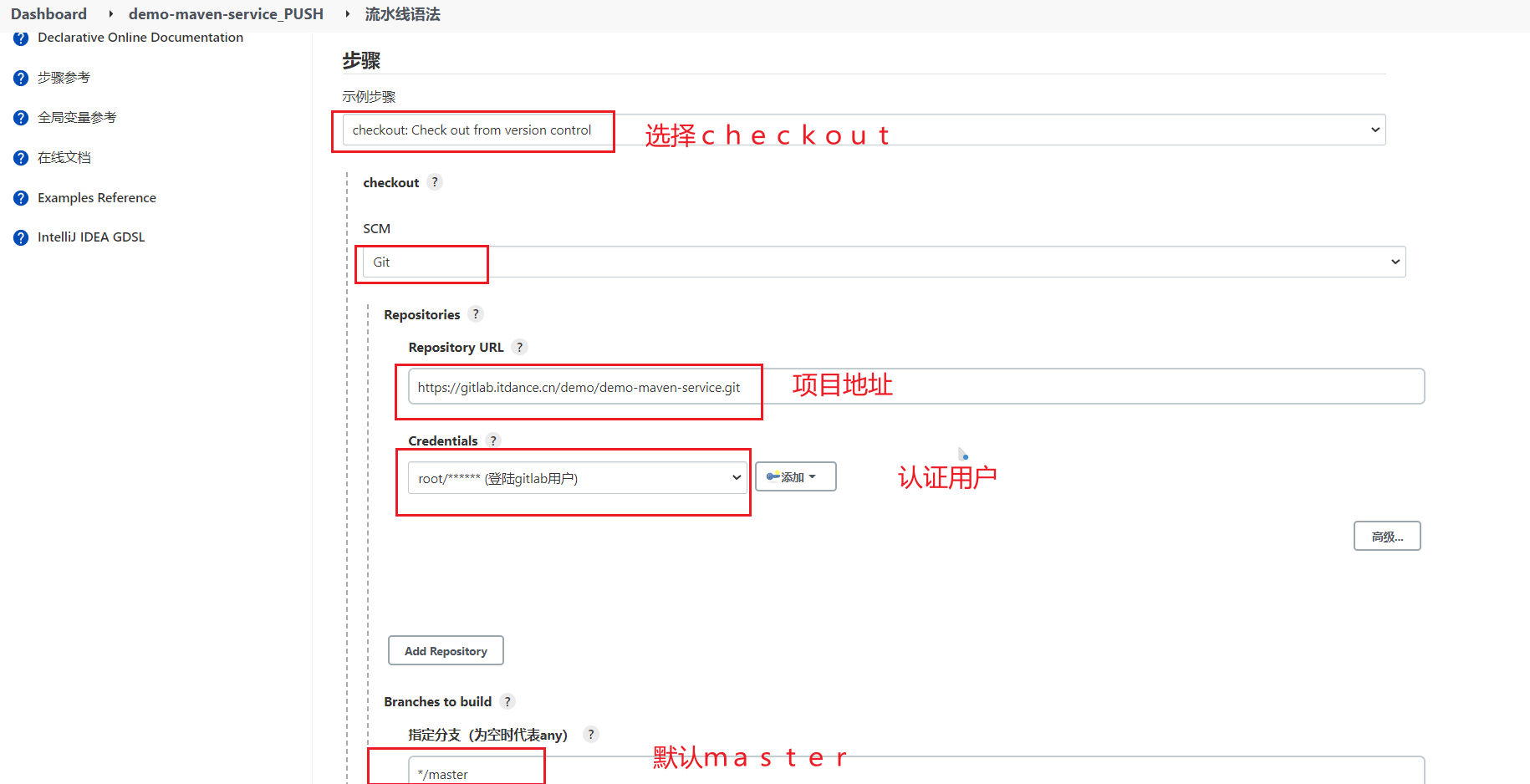




**13.3编辑ci.jenkinsfile**



流水线语法工具生成脚本





|  |
| --- |
| #!groovy  @Library('jenkinslib@master') \_  //func from sharelibrary  def build = new org.devops.build()  def deploy = new org.devops.deploy()  def tools = new org.devops.tools()  //env  String buildType = "${env.buildType}"  String buildShell = "${env.buildShell}"  String deployHosts= "${env.deployHosts}"  String srcUrl="${env.srcUrl}"  String branchName="{env.branchName}"  pipeline{  agent{ node { label "master"} }      stages{  stage("CheckOut"){  steps{  script{  tools.PrintMes("获取代码","green") //颜色输出  checkout([$class: 'GitSCM', branches: [[name: '${branchName}']], extensions: [], userRemoteConfigs: [[credentialsId: '2', url: '${srcUrl}']]])  }  }  }  stage("Build"){  steps{  script{  tools.PrintMes("执行打包","green")  build.Build(buildType,buildShell)    //deploy.AnsibleDeploy("${deployHost}","-m ping")  }  }  }  }  } |

流水线语法工具生成的标蓝部分是在原有基础上需要修改的

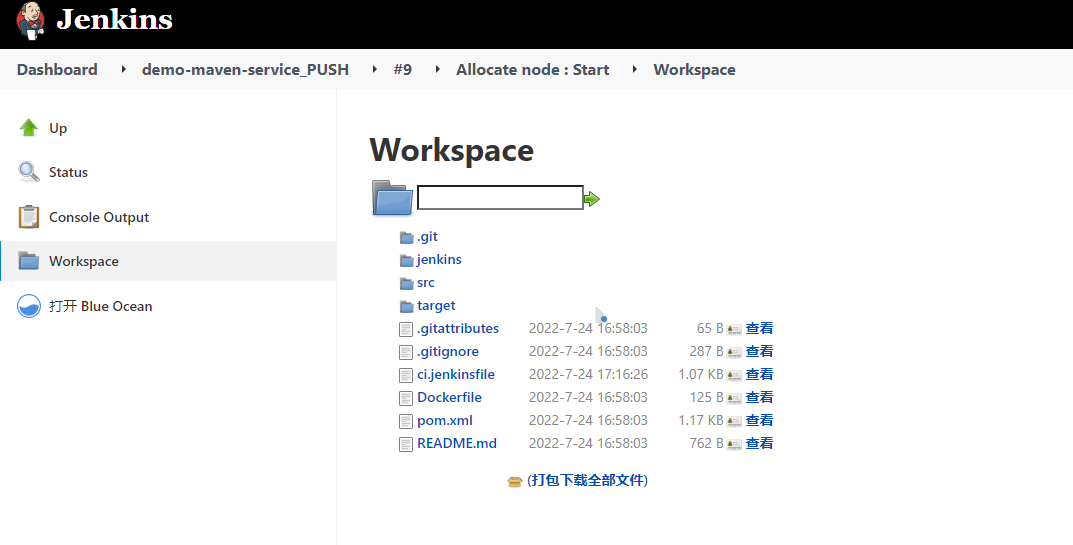
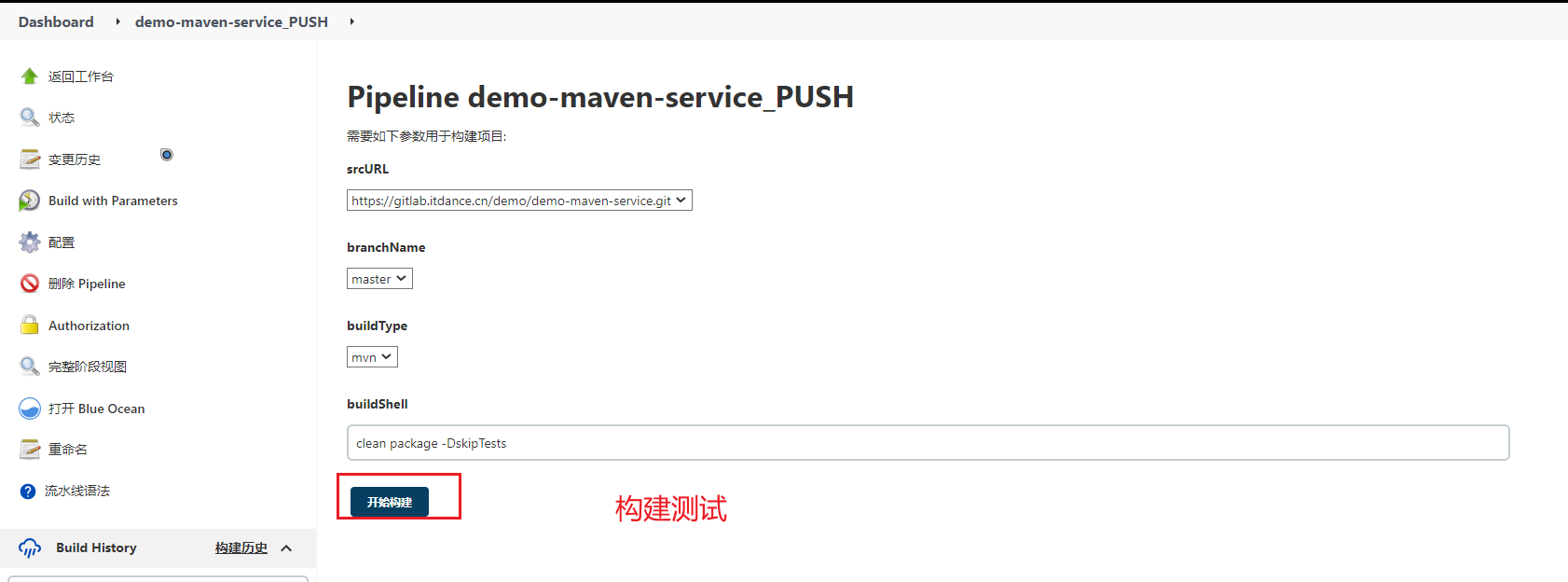
checkout([$class: 'GitSCM', branches: [[name: '\*/master']], extensions: [], userRemoteConfigs: [[credentialsId: '2', url: 'https://gitlab.itdance.cn/demo/demo-maven-service.git']]])

如

\*/master 修改成 ${branchName}

<https://gitlab.itdance.cn/demo/demo-maven-service.git> 修改成 ${srcUrl}

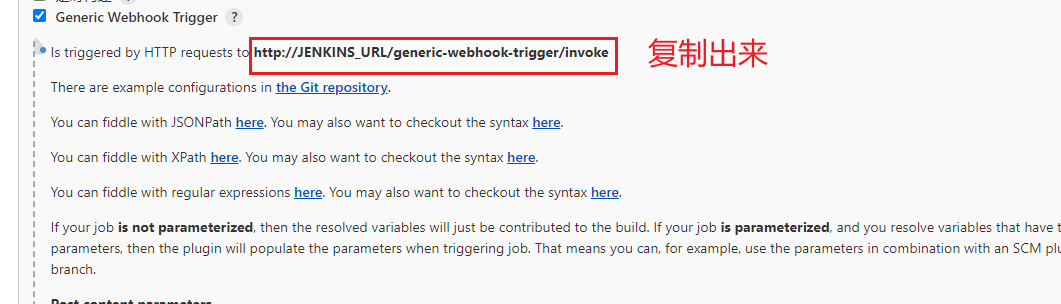
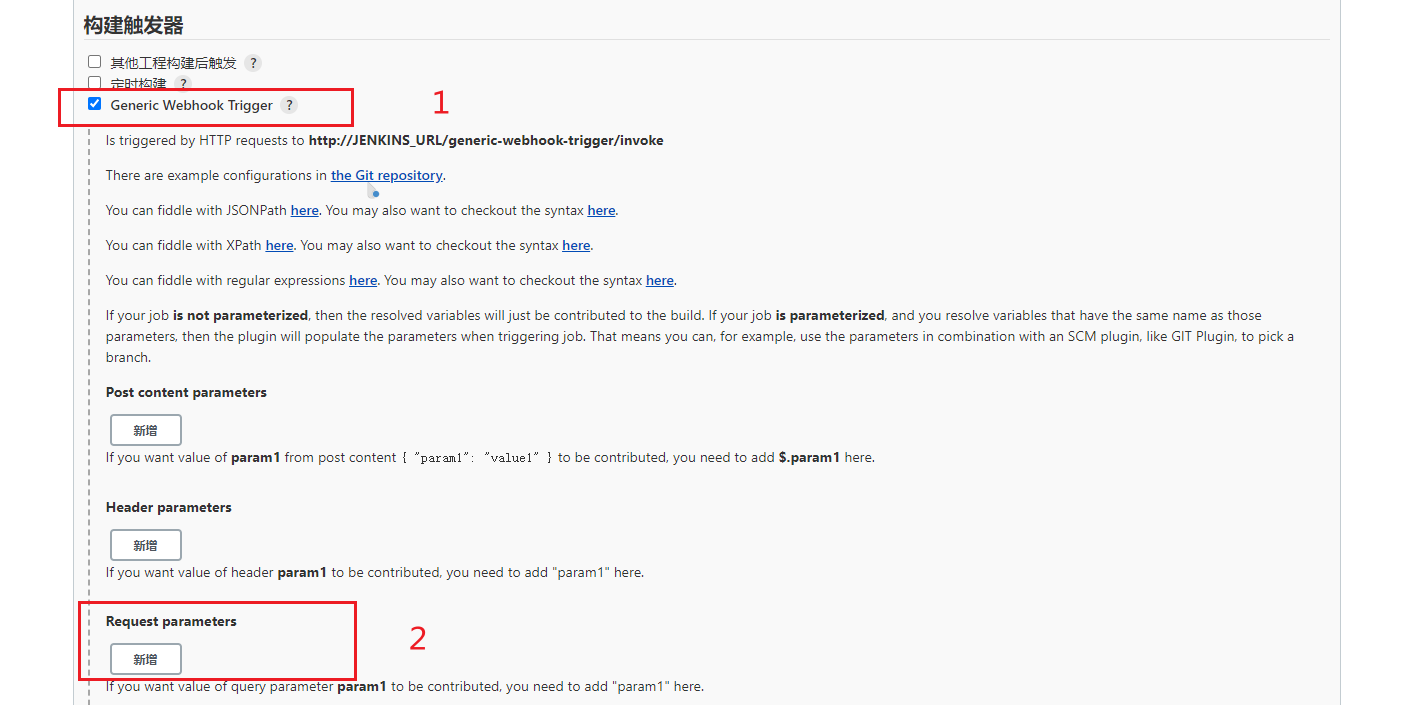
**13.4构建测试**



**13.5 配置gitlab提交流水线**

**13.5.1 构建触发器**

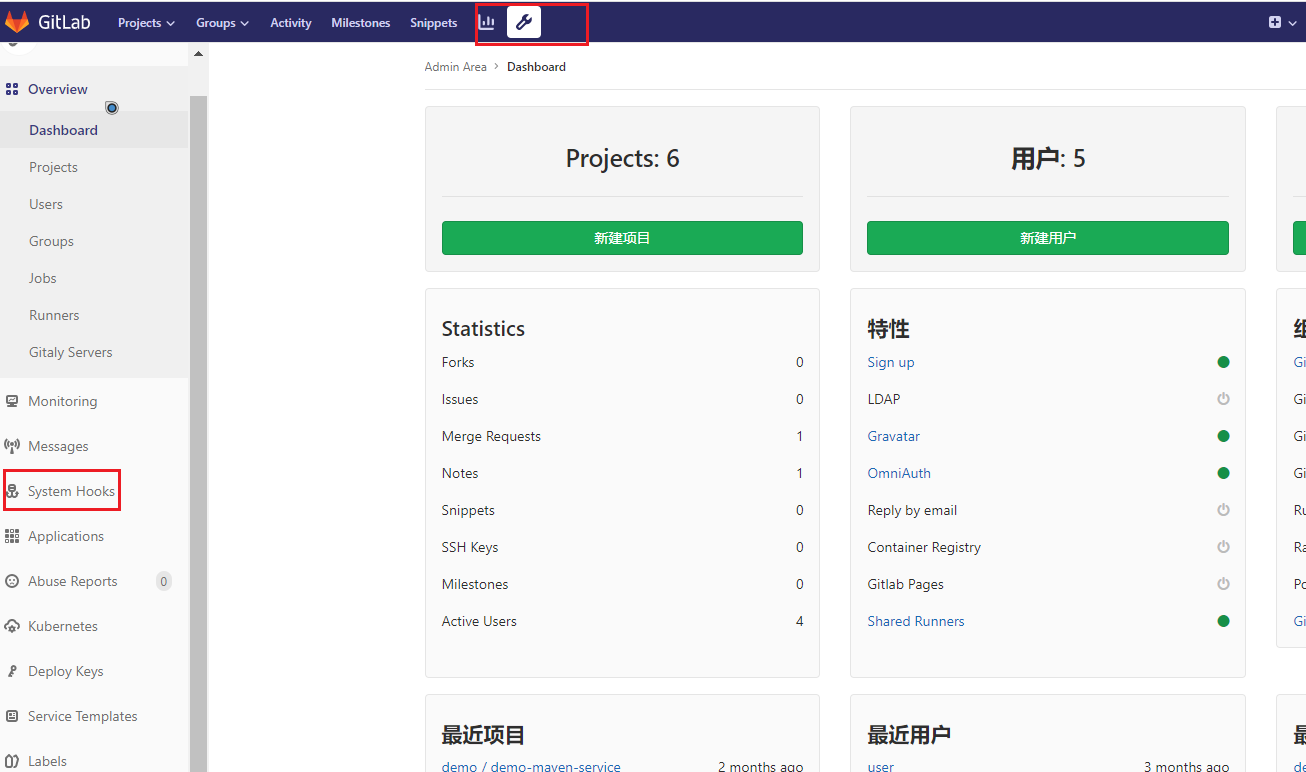
Generic Webhook Trigger -> **Request parameters**



★ runOpts 定义流水线的运行模式

★ Tocken 触发tocken

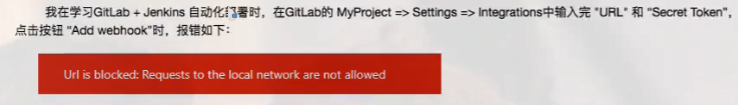
**13.5.2 Gitlab配置system hooks**

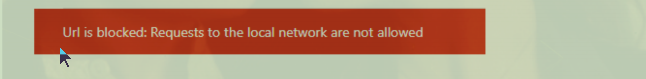




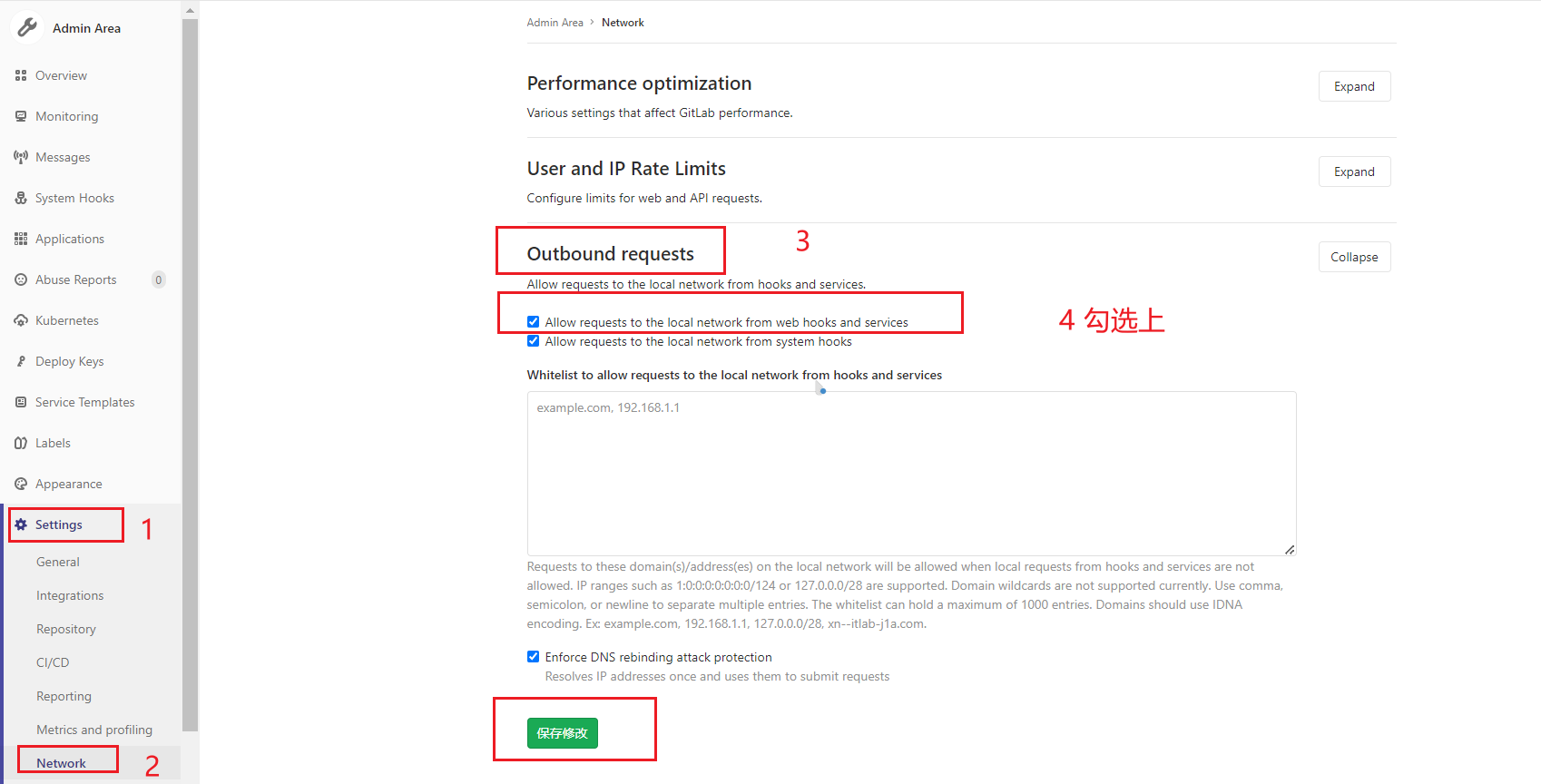
http://jenkins.itdance.cn/generic-webhook-trigger/invoke?token=demo-maven-service\_PUSH&runOpts=GitlabPush

如果遇到报错

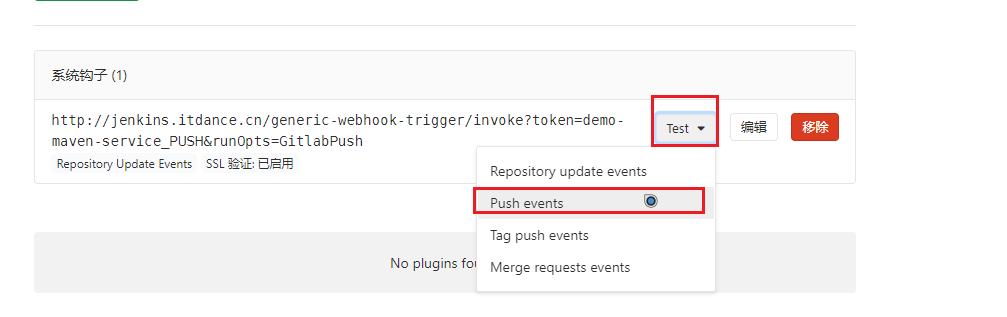




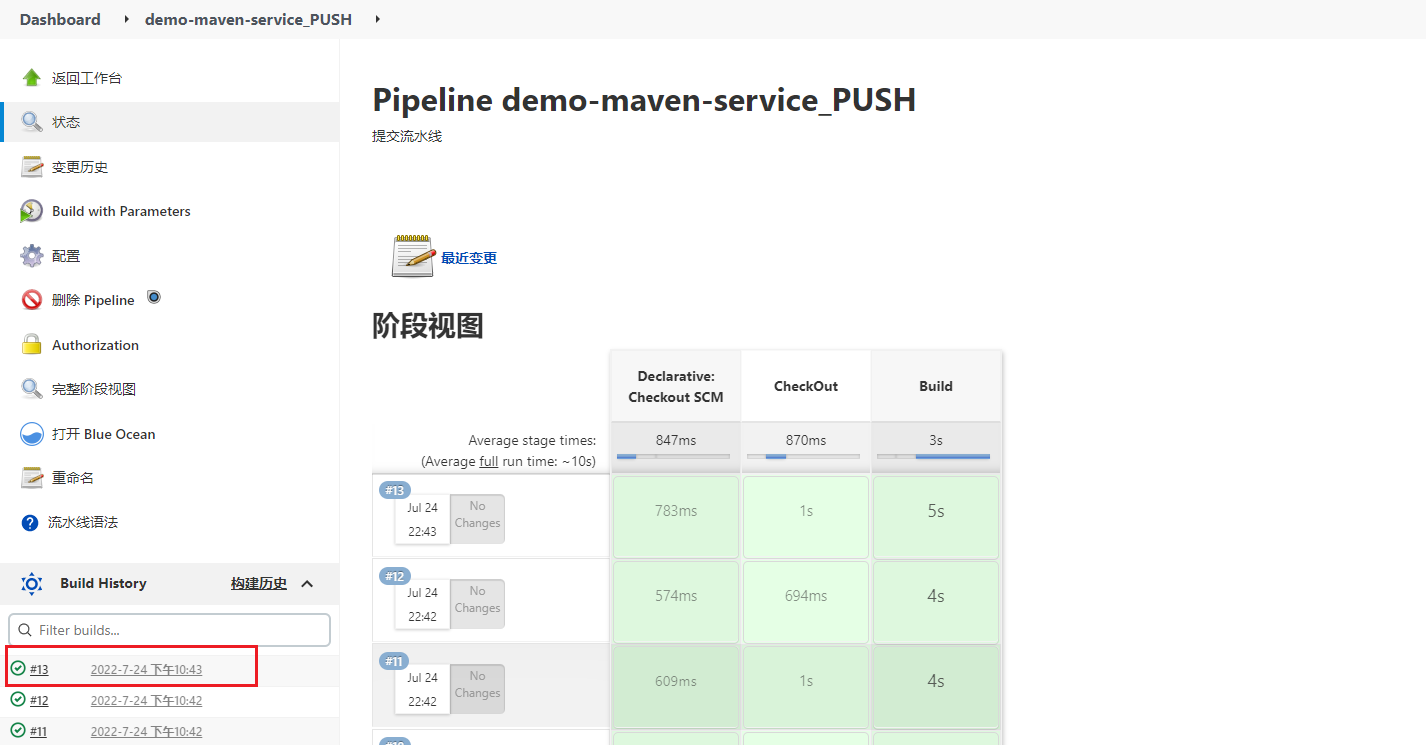
解决办法如下



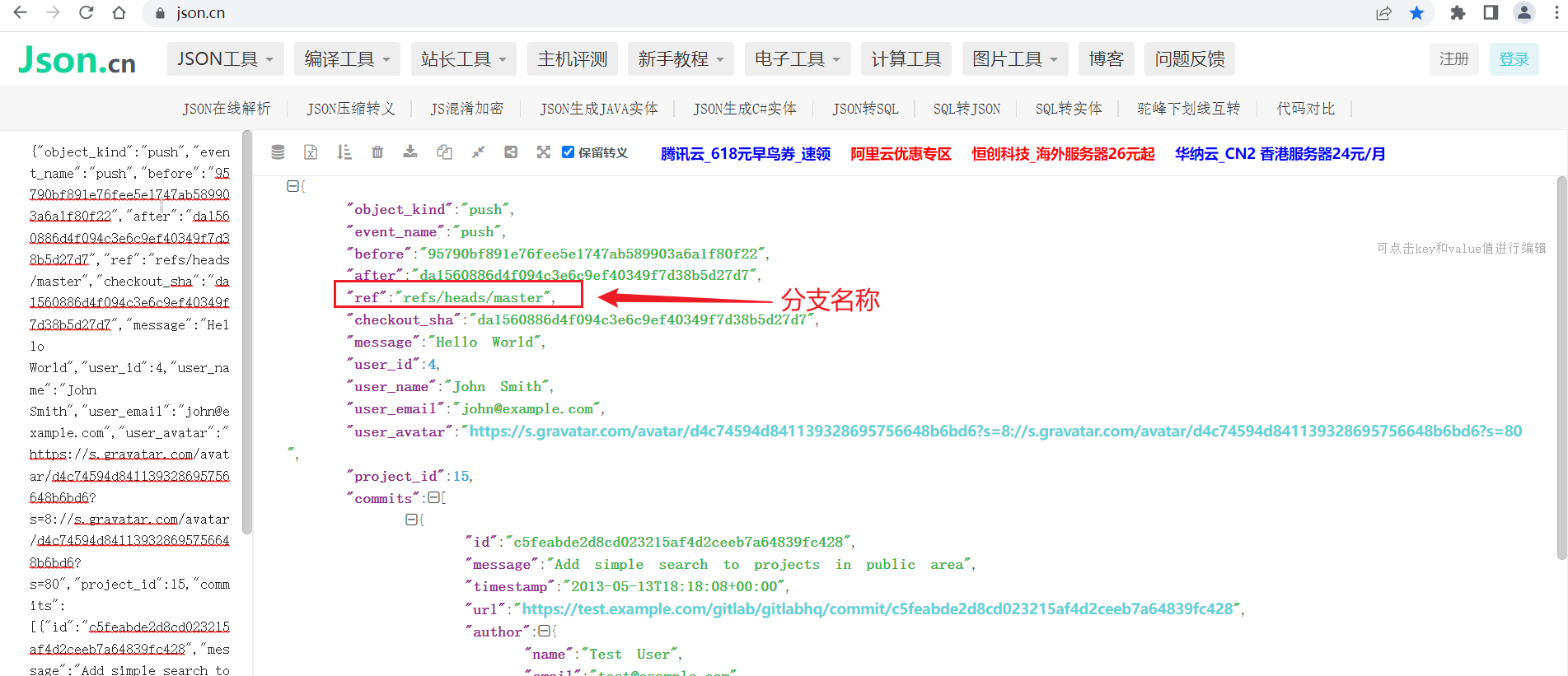
**13.5.3测试**



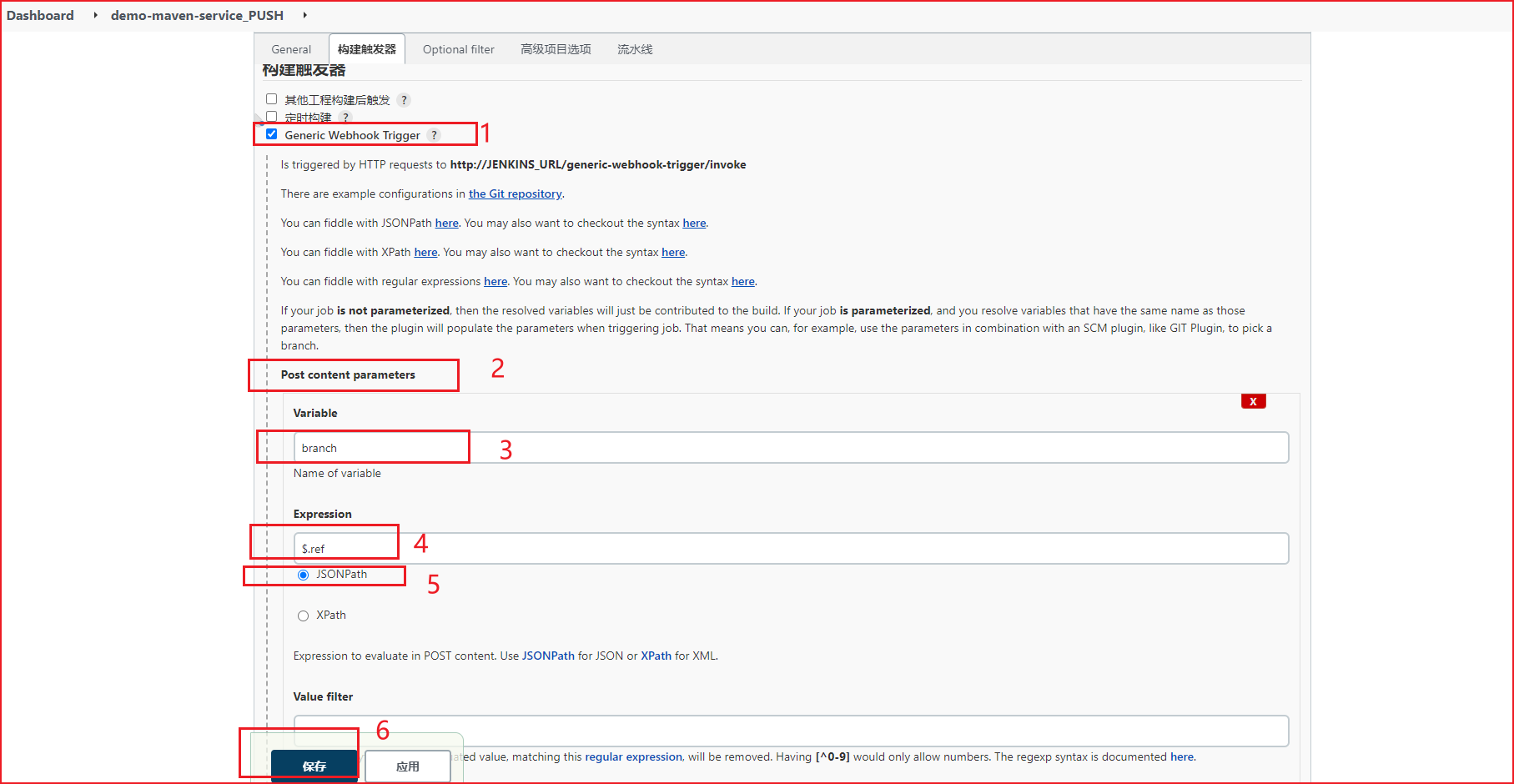
再到Jenkins查看，push events已经出发Jenkins构建。如下图

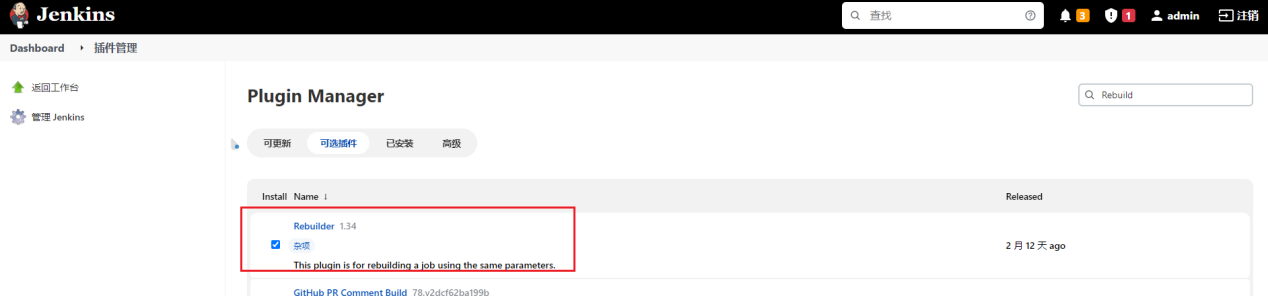


**13.5.4 优化(分支自动匹配)**



**再到Jenkins里配置**

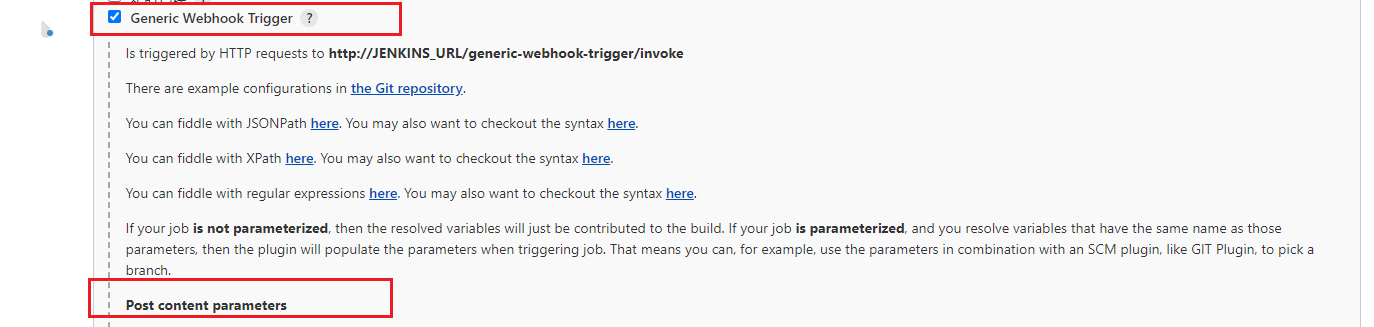


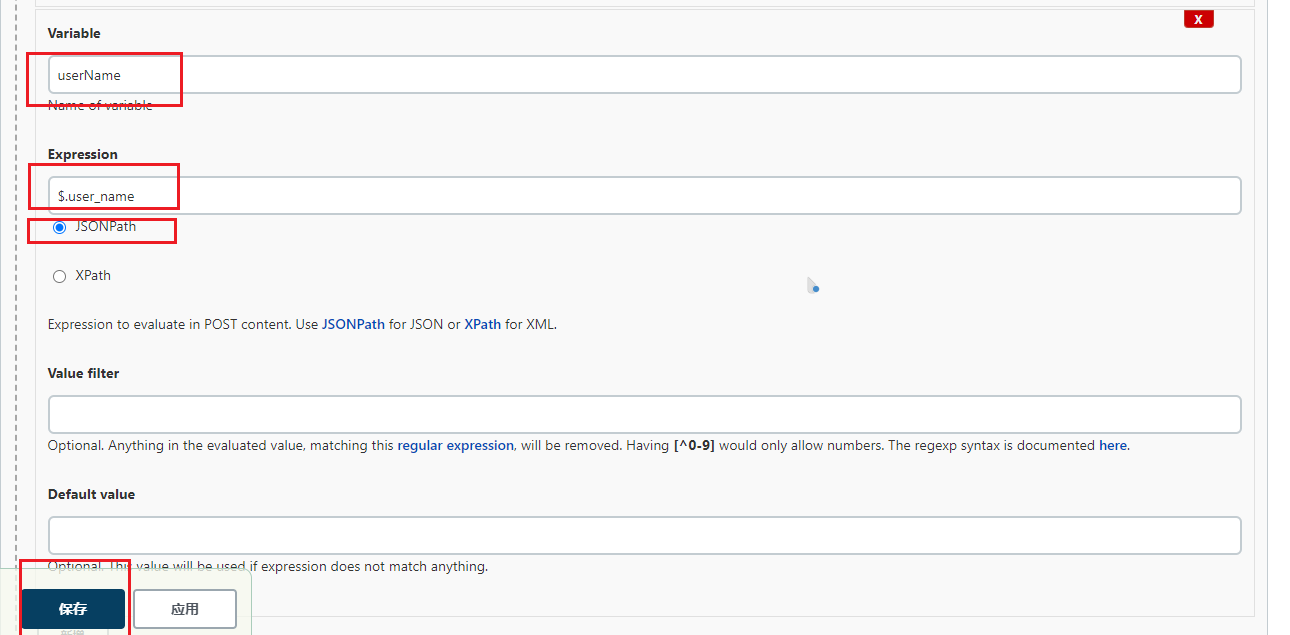


编辑ci.jenkinsfile 添加分支判断

|  |
| --- |
| #!groovy  @Library('jenkinslib@master') \_  //func from sharelibrary  def build = new org.devops.build()  def deploy = new org.devops.deploy()  def tools = new org.devops.tools()  //env  String buildType = "${env.buildType}"  String buildShell = "${env.buildShell}"  String deployHosts= "${env.deployHosts}"  String srcUrl="${env.srcUrl}"  String branchName="{env.branchName}"  pipeline{  agent{ node { label "master"} }      stages{  stage("CheckOut"){  steps{  script{  if ("${runOpts}" == "GitlabPush"){  branchName = branch - "refs/heads/"    }  println("${branchName}")  tools.PrintMes("构建分支为：${branchName}","green")  tools.PrintMes("获取代码","green")  checkout([$class: 'GitSCM', branches: [[name: '${branchName}']], extensions: [], userRemoteConfigs: [[credentialsId: '2', url: '${srcUrl}']]])  }  }  }  stage("Build"){  steps{  script{  tools.PrintMes("执行打包","green")  build.Build(buildType,buildShell)    //deploy.AnsibleDeploy("${deployHost}","-m ping")  }  }  }  }  } |

**13.5.5 优化提交流水线-增加构建描述信息。**



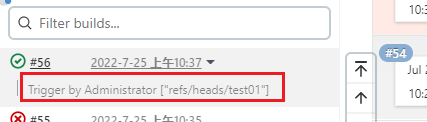


编辑ci.jenkinsfile

|  |
| --- |
| #!groovy  @Library('jenkinslib@master') \_  //func from sharelibrary  def build = new org.devops.build()  def deploy = new org.devops.deploy()  def tools = new org.devops.tools()  //env  String buildType = "${env.buildType}"  String buildShell = "${env.buildShell}"  String deployHosts= "${env.deployHosts}"  String srcUrl="${env.srcUrl}"  String branchName="{env.branchName}"  if ("${runOpts}" == "GitlabPush"){  branchName = branch - "refs/heads/"  currentBuild.description = "Trigger by ${userName} ${branch}"  }  pipeline{  agent{ node { label "master"} }      stages{  stage("CheckOut"){  steps{  script{  println("${branchName}")  tools.PrintMes("构建分支为：${branchName}","green")  tools.PrintMes("获取代码","green")  checkout([$class: 'GitSCM', branches: [[name: '${branchName}']], extensions: [], userRemoteConfigs: [[credentialsId: '2', url: '${srcUrl}']]])  }  }  }  stage("Build"){  steps{  script{  tools.PrintMes("执行打包","green")  build.Build(buildType,buildShell)    //deploy.AnsibleDeploy("${deployHost}","-m ping")  }  }  }  }  } |



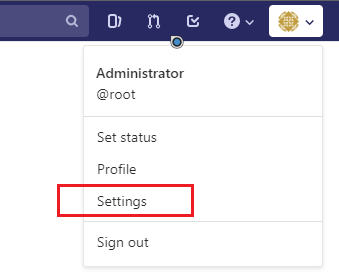
验证

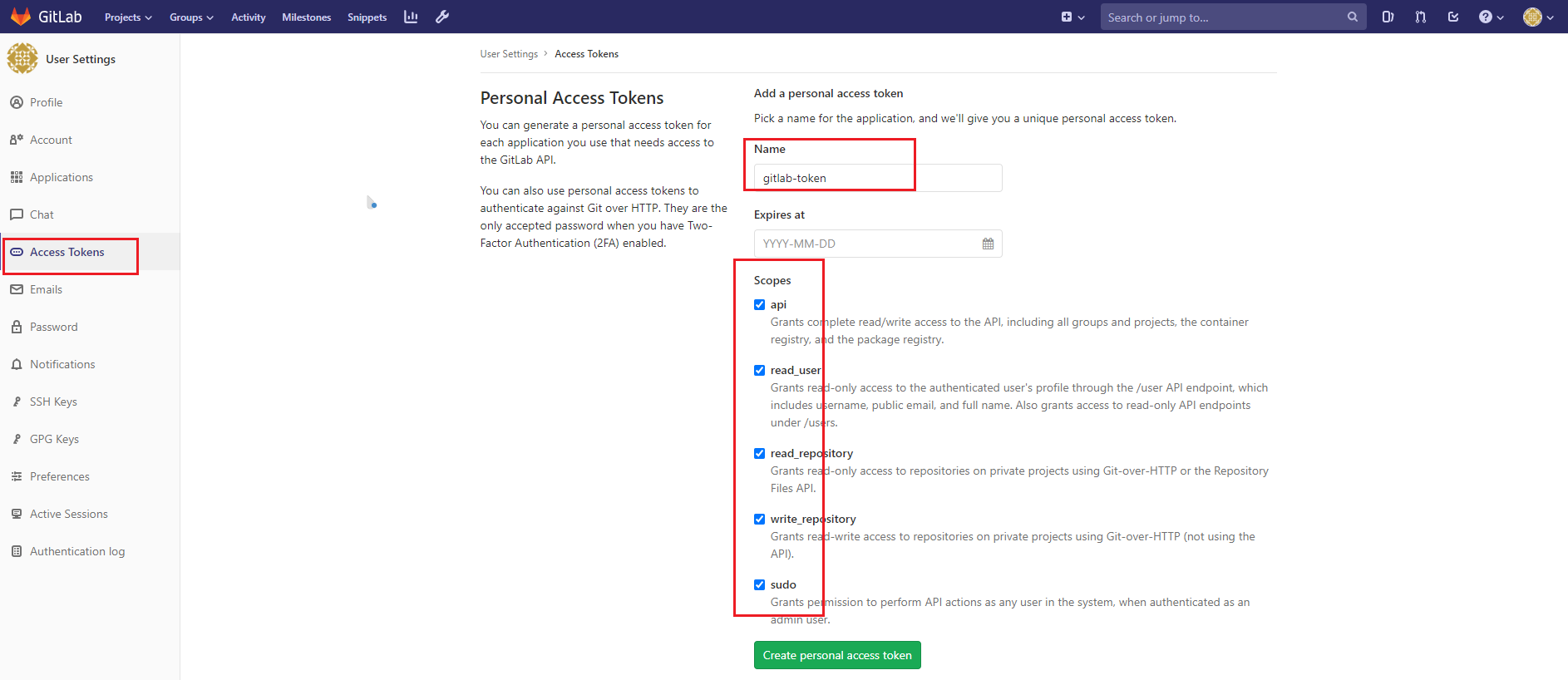


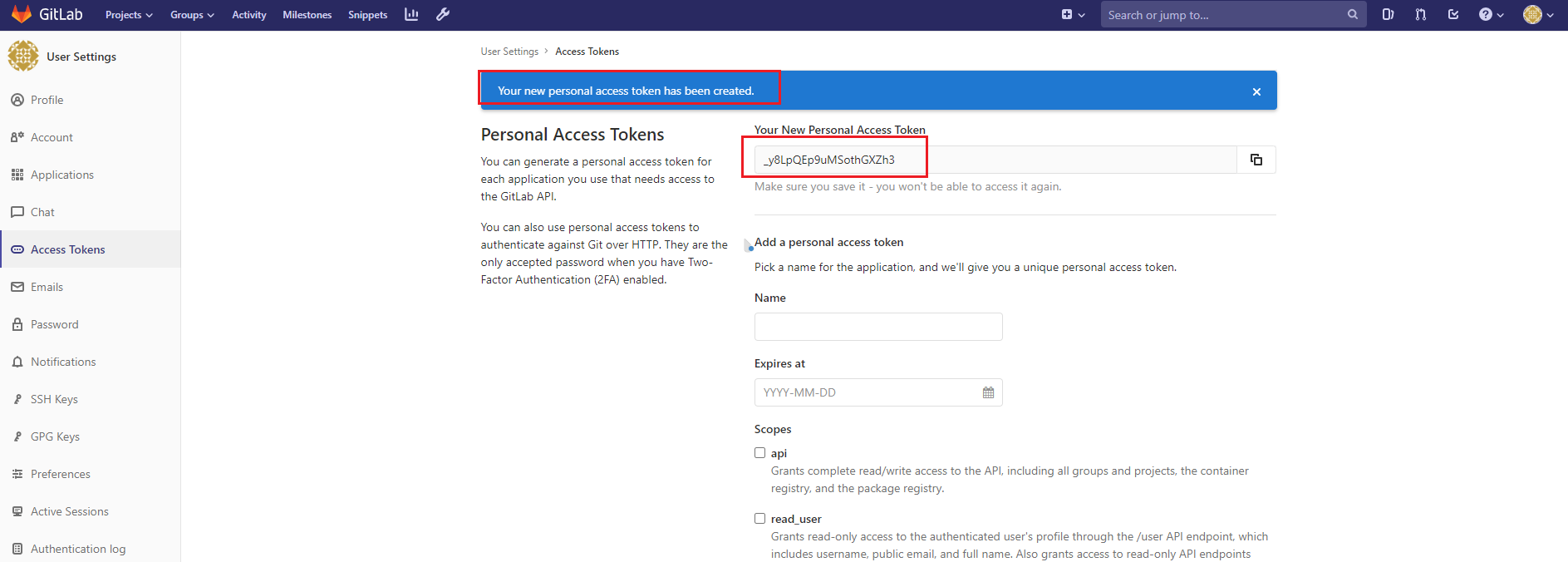
构建完成后会有信息

**13.5.6 优化提交流水线-变更commit状态。**

**13.5.6.1 Gitlab 创建api token**







创建成功后，显示token信息，复制到本地（注意：关闭此页面后Token码将再也看不见，务必复制）

\_y8LpQEp9uMSothGXZh3

**13.5.6.2 Jenkins配置凭据**

https://blog.csdn.net/weixin\_40046357/article/details/107679359







注意：凭据类型选择Secret text,将'gitlab生成的凭据(token)写到secret中'，同时'id'一定要为gitlab-token

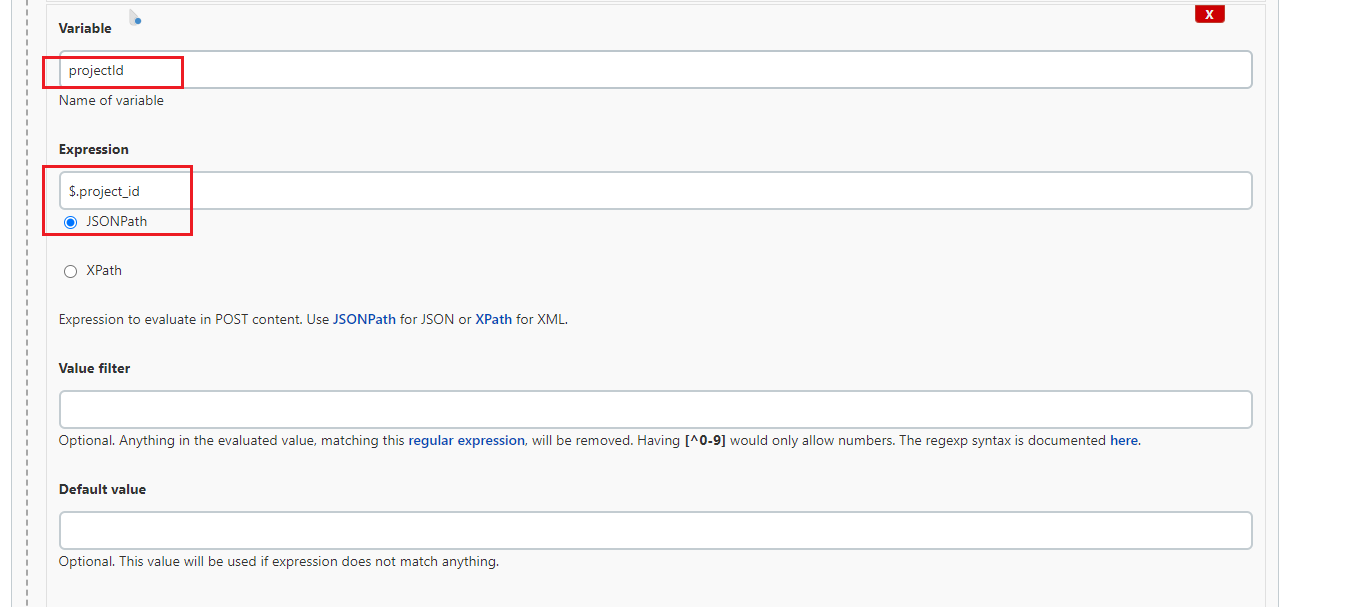
否则会报错



ERROR: Credentials 'gitlab-token' is of type 'Username with password' where 'org.jenkinsci.plugins.plaincredentials.StringCredentials' was expected

**13.5.6.3 构建触发器-》Generic Webhook Trigger-》Post content parameters**

添加两个构建参数









**13.5.6.4 修改gitlab.groovy**

**<https://blog.csdn.net/wzj_110/article/details/104495874>**



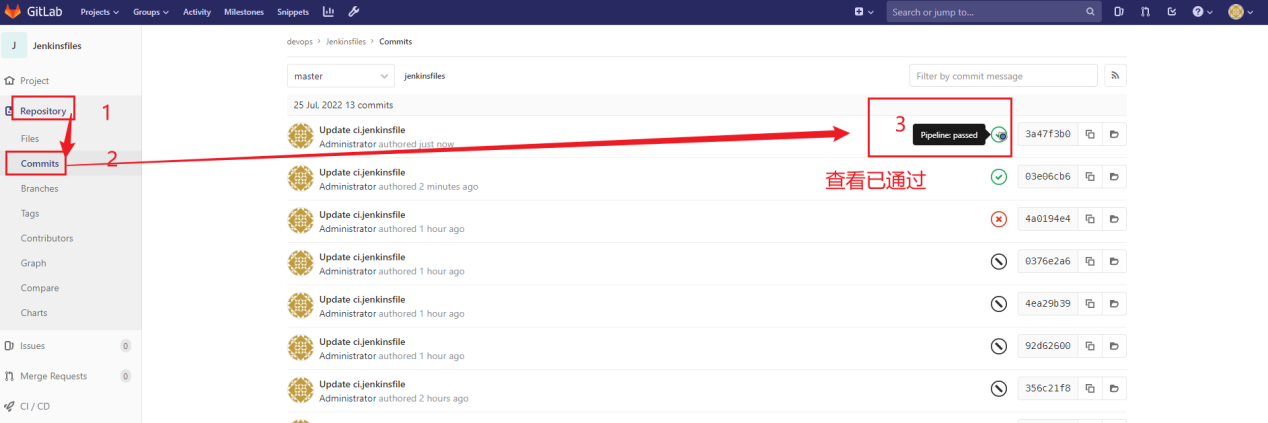
**13.5.6.5 编辑ci.jenkinsfile**

|  |
| --- |
| #!groovy  @Library('jenkinslib@master') \_  //func from sharelibrary  def build = new org.devops.build()  def deploy = new org.devops.deploy()  def tools = new org.devops.tools()  def gitlab = new org.devops.gitlab() //引用共享库gitlab.groovy  //env  String buildType = "${env.buildType}"  String buildShell = "${env.buildShell}"  String deployHosts= "${env.deployHosts}"  String srcUrl="${env.srcUrl}"  String branchName="{env.branchName}"  if ("${runOpts}" == "GitlabPush"){  branchName = branch - "refs/heads/"  currentBuild.description = "Trigger by ${userName} ${branch}"  gitlab.ChangeCommitStatus(projectId,commitSha,"running")  }  pipeline{  agent{ node { label "master"} }      stages{  stage("CheckOut"){  steps{  script{  println("${branchName}")  tools.PrintMes("构建分支为：${branchName}","green")  tools.PrintMes("获取代码","green")  checkout([$class: 'GitSCM', branches: [[name: '${branchName}']], extensions: [], userRemoteConfigs: [[credentialsId: '2', url: '${srcUrl}']]])  }  }  }  stage("Build"){  steps{  script{  tools.PrintMes("执行打包","green")  build.Build(buildType,buildShell)    //deploy.AnsibleDeploy("${deployHost}","-m ping")  }  }  }  }  post {  always{  script{  println("always")  }  }    success{  script{  println("success")  if ("${runOpts}" == "GitlabPush"){  gitlab.ChangeCommitStatus(projectId,commitSha,"success")  }  }    }  failure{  script{  println("failure")  if ("${runOpts}" == "GitlabPush"){  gitlab.ChangeCommitStatus(projectId,commitSha,"failed")  }  }  }    aborted{  script{  println("aborted")  if ("${runOpts}" == "GitlabPush"){  gitlab.ChangeCommitStatus(projectId,commitSha,"canceled")  }  }    }    }  } |



**13.5.6.6验证**

如下图可以看到有状态信息显示

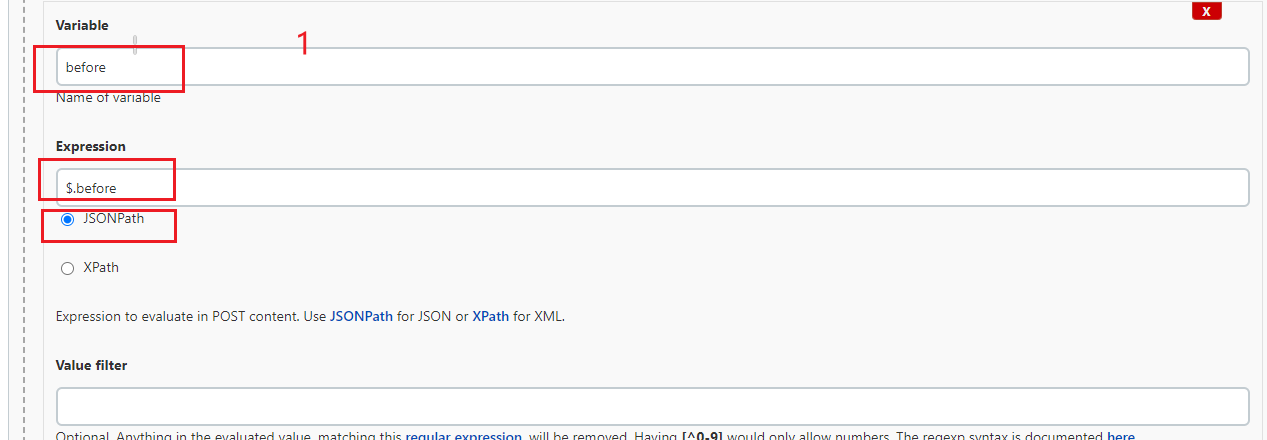


**13.5.7 优化提交流水线-过滤push请求。**

有时候添加分支或者标记tag也会触发构建

**13.5.7.1添加post参数**

**构建触发器-》Generic Webhook Trigger-》Post content parameters**







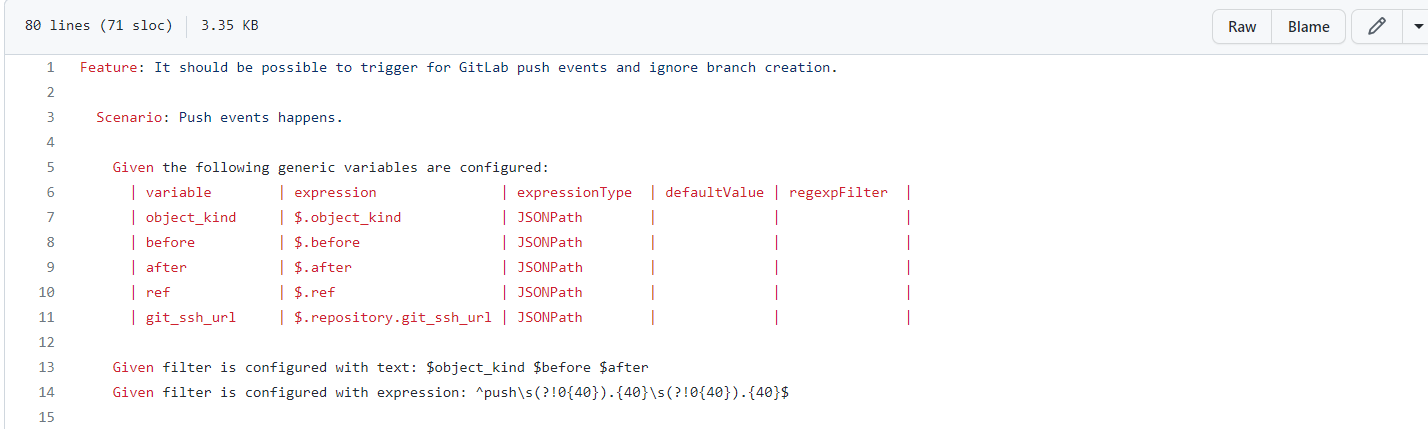
**13.5.7.2 添加过滤条件**

**Optional filter**

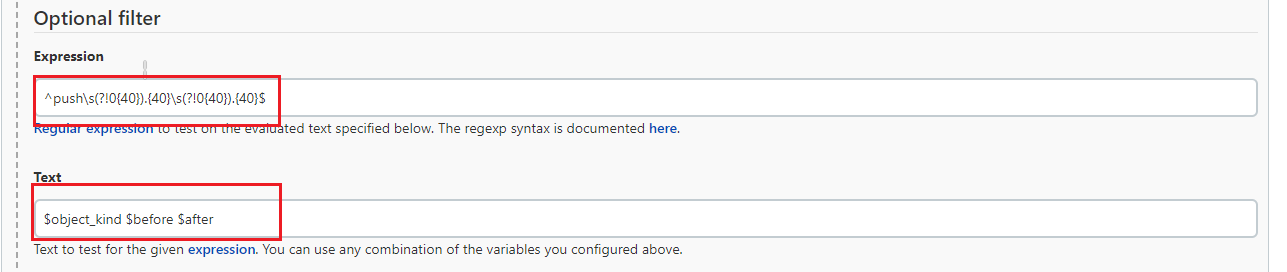
**<https://github.com/jenkinsci/generic-webhook-trigger-plugin/blob/master/src/test/resources/org/jenkinsci/plugins/gwt/bdd/gitlab/gitlab-push-ignore-create-remove-branch.feature>**

**\s 空格**

**正则表达式以^开头 $结尾 \s隔开**

Given filter is configured with text: $object\_kind $before $after

Given filter is configured with expression: ^push\s(?!0{40}).{40}\s(?!0{40}).{40}$



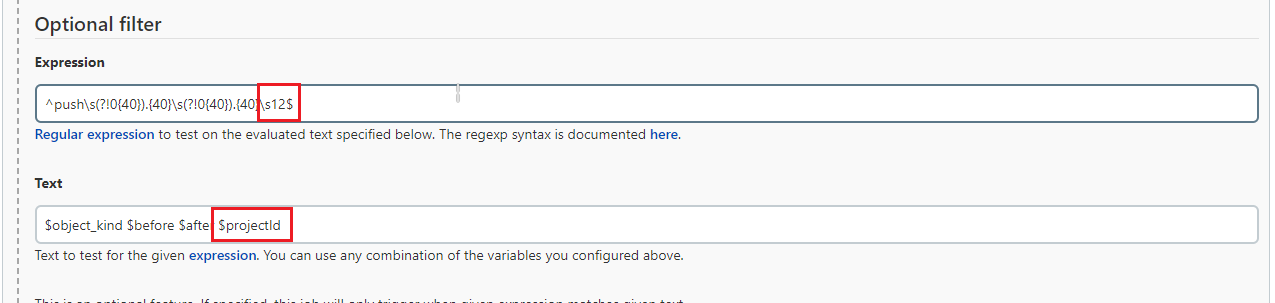
这要比如在新建分支的时候就不会触发了

**13.5.8 优化提交流水线-只限制项目提交的更改才会触发。**

比如 [demo-maven-service](https://gitlab.itdance.cn/devops/demo-maven-service)项目提交的才会触发demo-maven-service\_PUSH构建

**项目ID是区别项目的唯一方式**

构建触发器-》Generic Webhook Trigger -》 Optional filter



解释

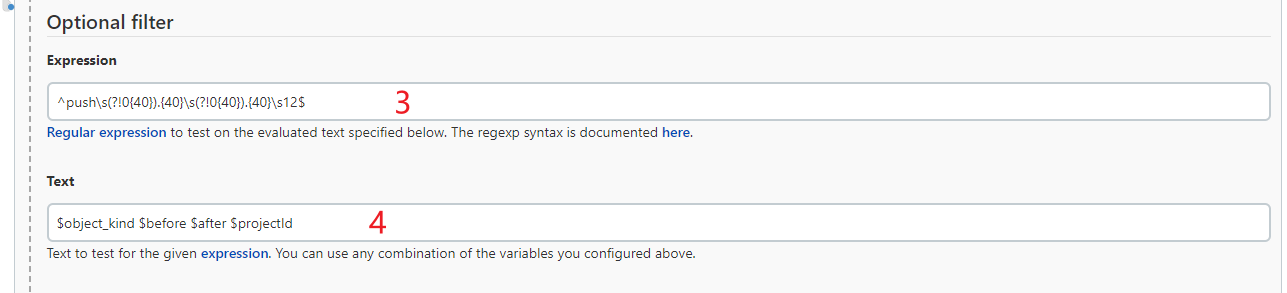


|  |
| --- |
| ^push\s(?!0{40}).{40}\s(?!0{40}).{40}\s12$  $object\_kind $before $after $projectId |

projectId是在 构建触发器-》Generic Webhook Trigger -》Post content parameters中定义的

总结：如果没有设置过滤条件，则任何项目的任何分支提交代码都会触发Jenkins构建





Optional filter的Expression表示，在Post content parameters的Expression中匹配到的数据的格式是否匹配，既是与‘$.project\_id’匹配的数据。这里应该是先确定触发器的名称，就是Text中指定的变量；

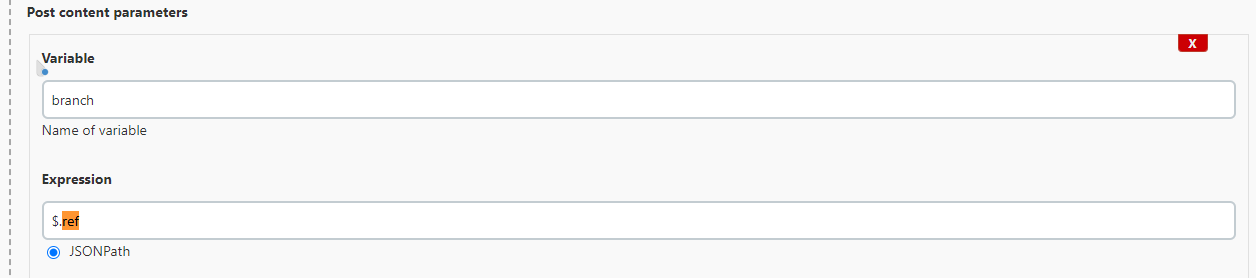
指定触发器的名字；

总的来说，通过（1）（2）（3）（4）的设置，将请求中的‘projectId’属性的值和Optional filter的表达式Expression进行比较，如果匹配则触发Jenkins构建，如果不匹配则不触发。如果只指定（1）（2）而没有（3）（4）的话，他只会在Jenkins构建界面显示匹配到的值和属性，不会按照指定项目进行触发的。

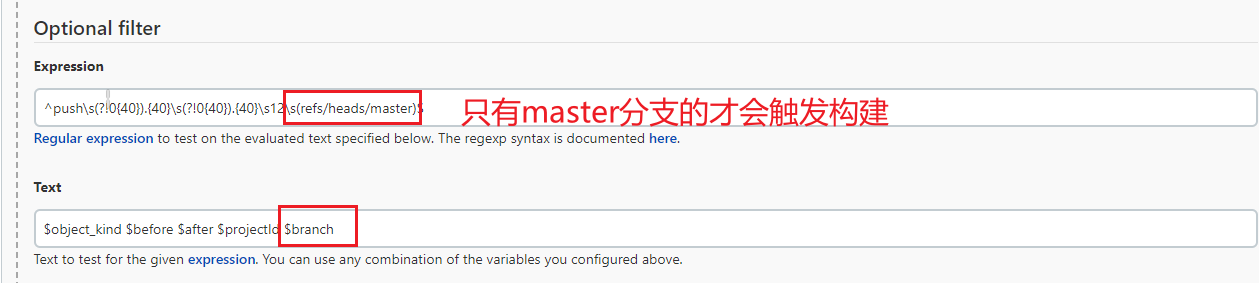
**13.5.9 优化提交流水线-只限制分支提交的更改才会触发。**

**<https://blog.51cto.com/u_15075514/4251831>**

Branch 可以很好的区分不同的分支

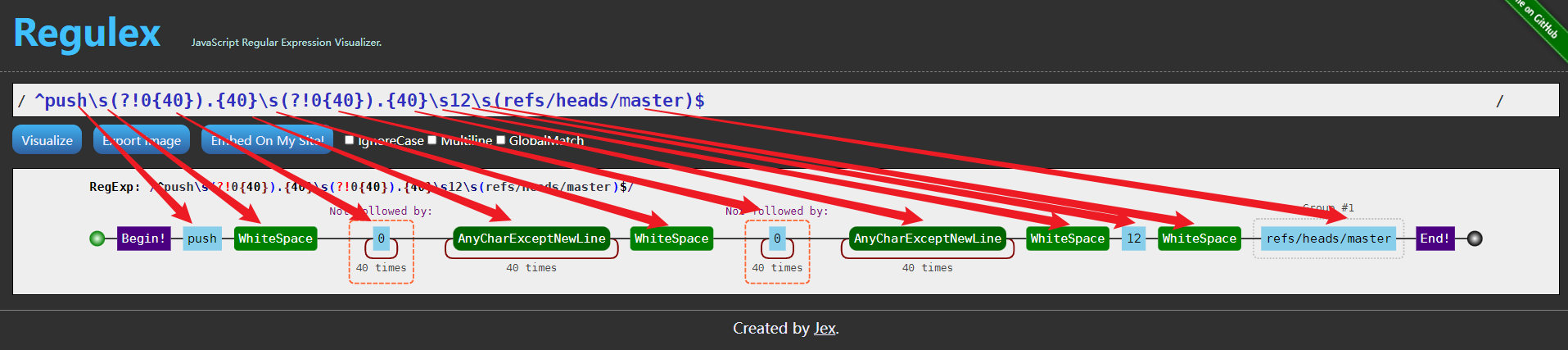


添加过滤条件

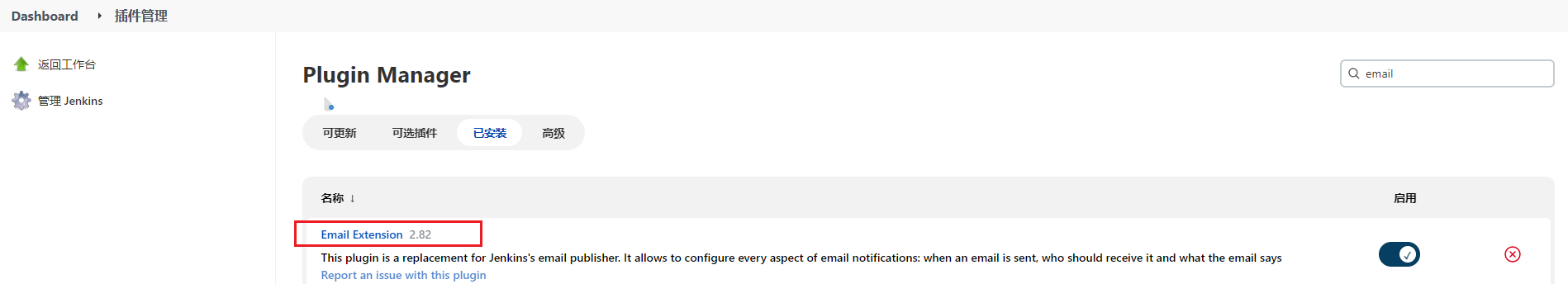


|  |
| --- |
| ^push\s(?!0{40}).{40}\s(?!0{40}).{40}\s12\s(refs/heads/master)$  $object\_kind $before $after $projectId $branch |

**其实这个规则既限制了具体项目也同时限制了项目下的具体分支**

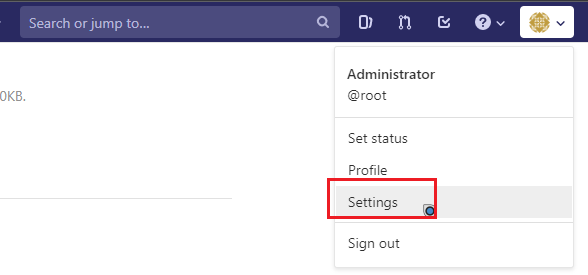


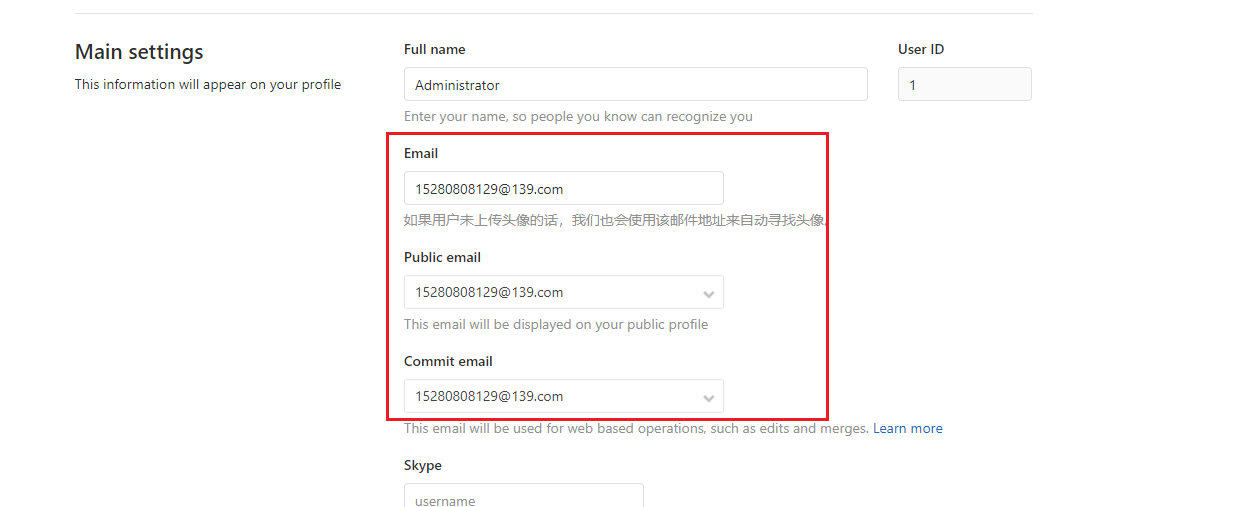
**13.5.9 构建失败邮件通知。**



确保插件已经安装

**13.5.9.1邮箱设置**

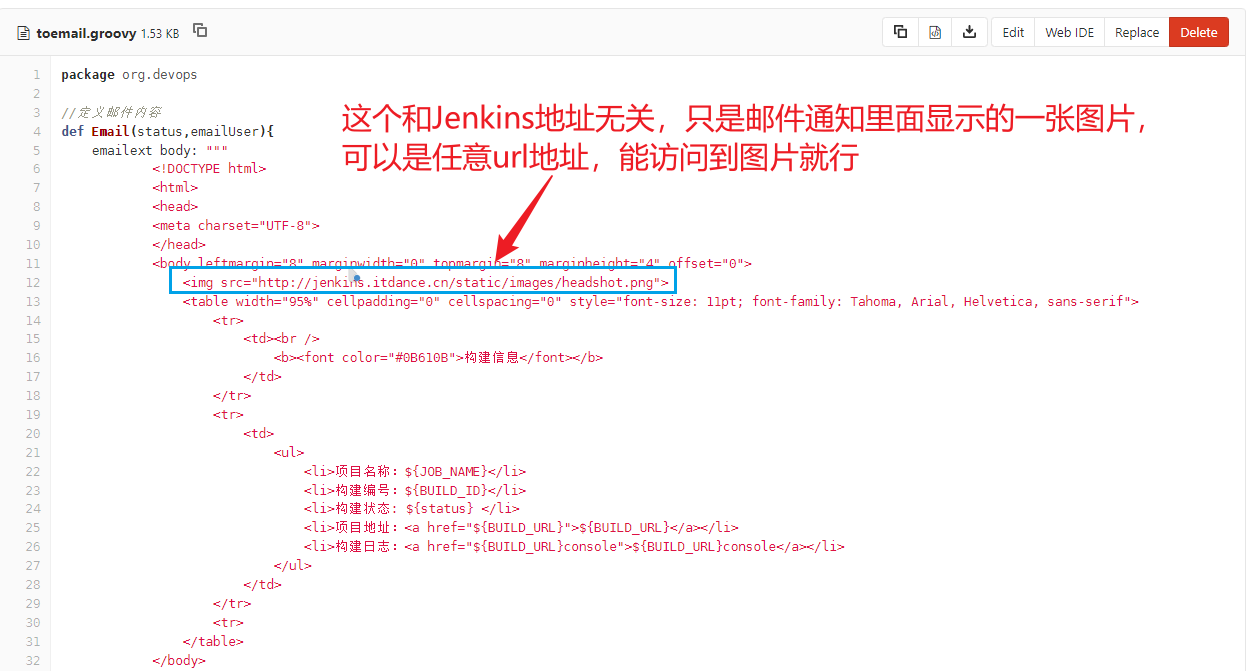




**13.5.9.2编辑toemail.groovy**

**共享库邮件发送模版**

修改Jenkins地址



**效果如下**



**13.5.9.3编辑ci.jenkinsfile**

引入 [toemail.groovy](https://gitlab.itdance.cn/devops/jenkinslib/blob/master/src/org/devops/toemail.groovy)

|  |
| --- |
| #!groovy  @Library('jenkinslib@master') \_  //func from sharelibrary  def build = new org.devops.build()  def deploy = new org.devops.deploy()  def tools = new org.devops.tools()  def gitlab = new org.devops.gitlab()  def toemail = new org.devops.toemail()  //env  String buildType = "${env.buildType}"  String buildShell = "${env.buildShell}"  String deployHosts= "${env.deployHosts}"  String srcUrl="${env.srcUrl}"  String branchName="{env.branchName}"  if ("${runOpts}" == "GitlabPush"){  branchName = branch - "refs/heads/"    currentBuild.description = "Trigger by ${userName} ${branch}"  gitlab.ChangeCommitStatus(projectId,commitSha,"running")  env.runOpts = "GitlabPush"    } else {  userEmail = "15280808129@139.com"  }  pipeline{  agent{ node { label "master"} }      stages{  stage("CheckOut"){  steps{  script{  println("${branchName}")  tools.PrintMes("构建分支为：${branchName}","green")  tools.PrintMes("获取代码","green")  checkout([$class: 'GitSCM', branches: [[name: '${branchName}']], extensions: [], userRemoteConfigs: [[credentialsId: '2', url: '${srcUrl}']]])  }  }  }  stage("Build"){  steps{  script{  tools.PrintMes("执行打包","green")  build.Build(buildType,buildShell)    //deploy.AnsibleDeploy("${deployHost}","-m ping")  }  }  }  }  post {  always{  script{  println("always")  }  }    success{  script{  println("success")  if ("${runOpts}" == "GitlabPush"){  gitlab.ChangeCommitStatus(projectId,commitSha,"success")  }  toemail.Email("流水线成功",userEmail)    }    }  failure{  script{  println("failure")  if ("${runOpts}" == "GitlabPush"){  gitlab.ChangeCommitStatus(projectId,commitSha,"failed")  }  toemail.Email("流水线失败了！",userEmail)  }  }    aborted{  script{  println("aborted")  if ("${runOpts}" == "GitlabPush"){  gitlab.ChangeCommitStatus(projectId,commitSha,"canceled")  }  toemail.Email("流水线被取消了！",userEmail)  }    }    }  } |



**13.5.9.4 Jenkins配置添加参数**

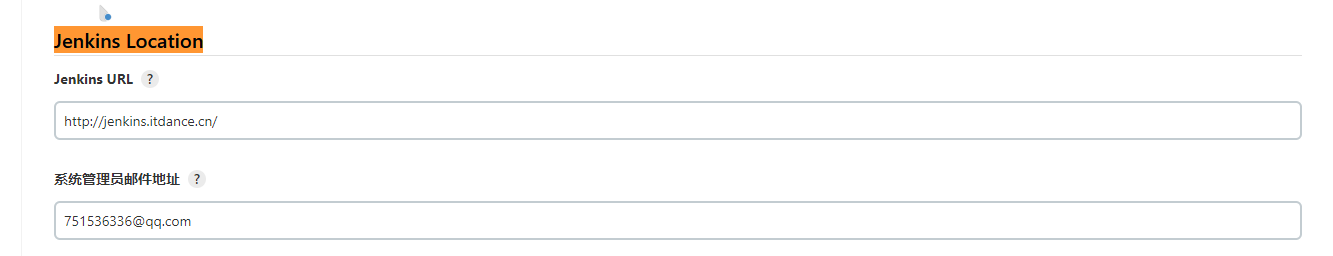
构建触发器-》Generic Webhook Trigger-》Post content parameters



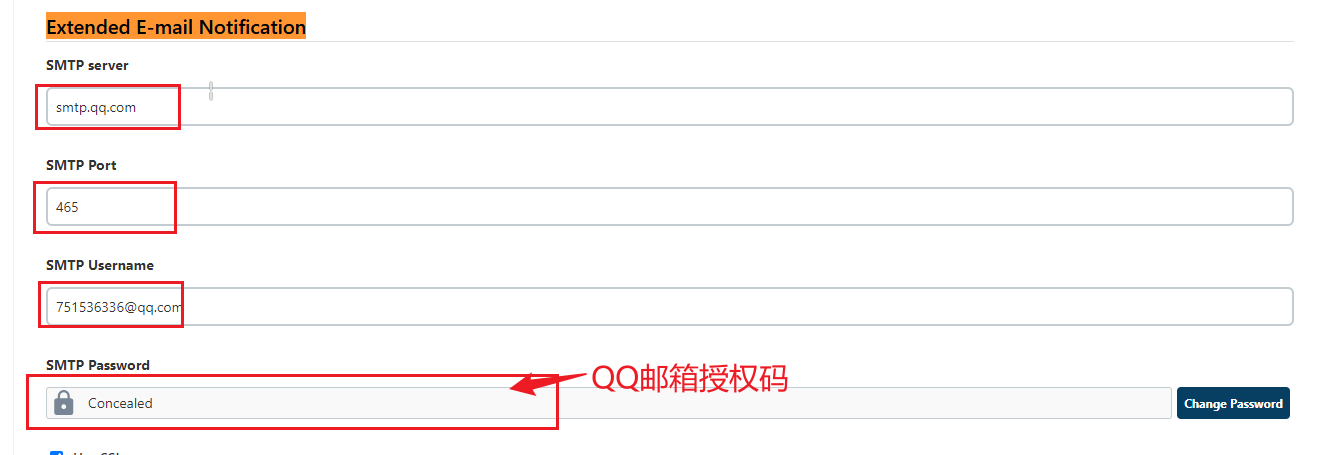
**13.5.9.5 Jenkins邮件设置**

系统管理-》系统设置

**Jenkins Location**



**Extended E-mail Notification**



**13.5.9.6验证**

[demo-maven-service](https://gitlab.itdance.cn/devops/demo-maven-service) 项目随便更改内容并提交





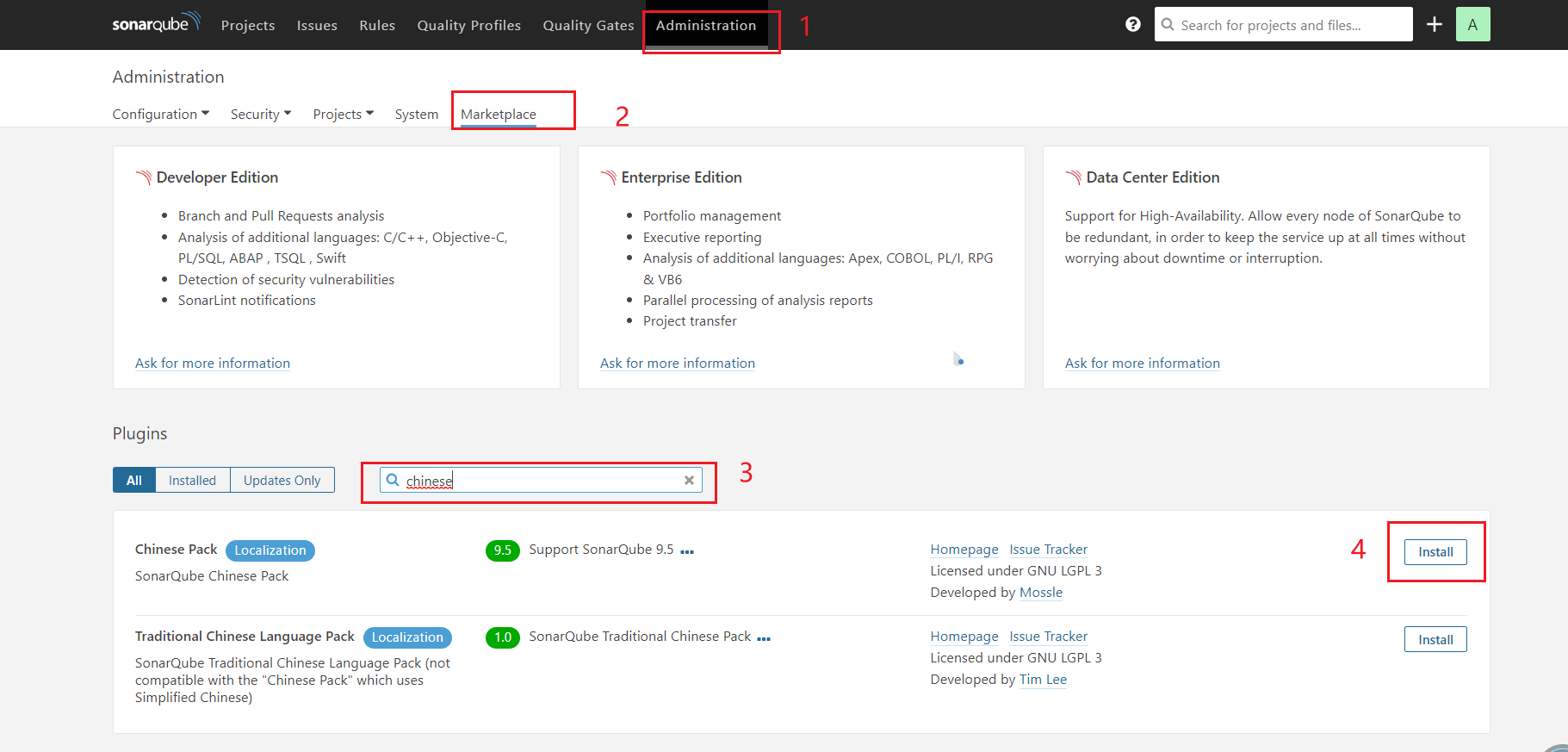
# **代码扫描集成**

sonarqube安装

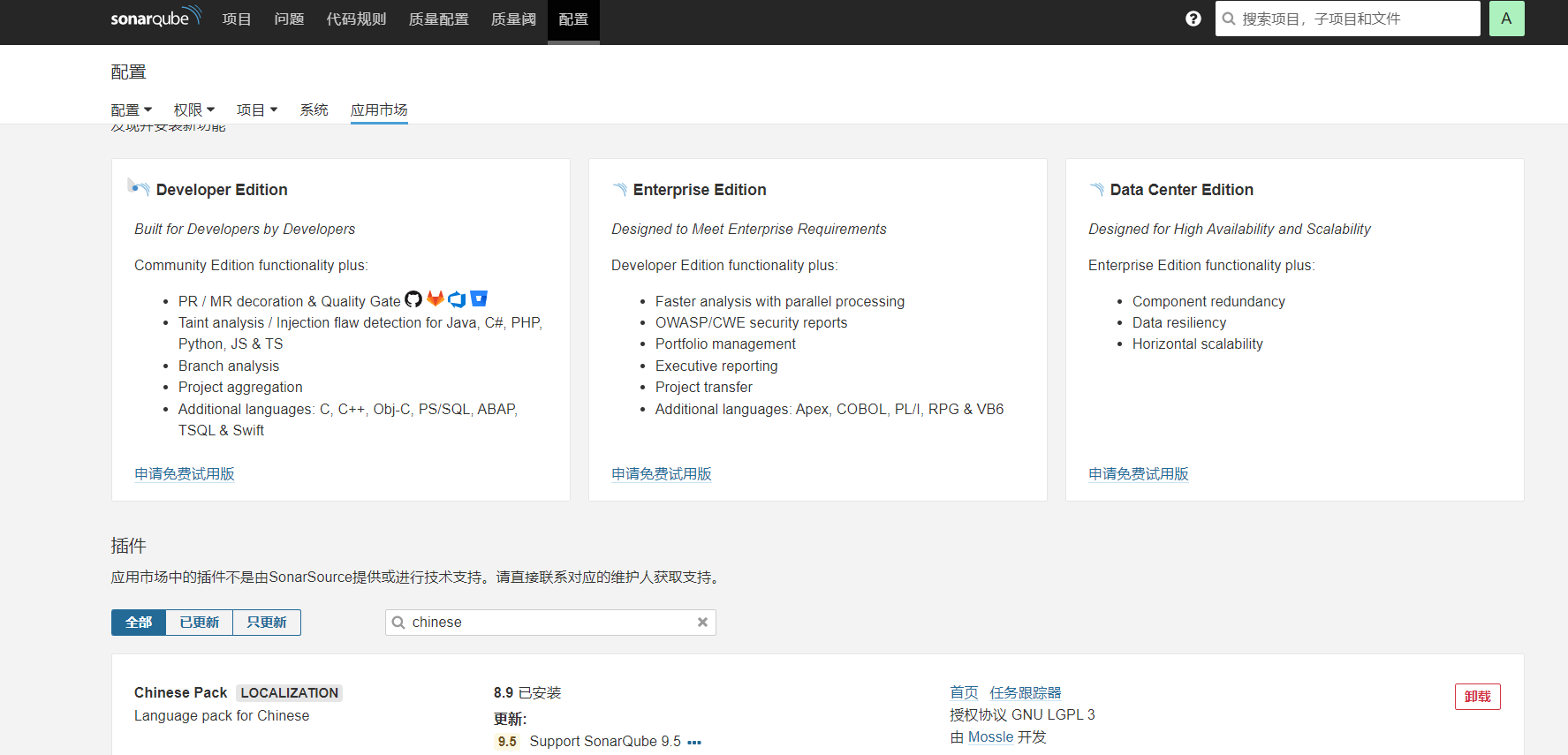
|  |
| --- |
| docker run -d --name sonarqube \  -p 9000:9000 \  -v sonarqube\_conf:/opt/sonarqube/conf \  -v sonarqube\_extensions:/opt/sonarqube/extensions \  -v sonarqube\_logs:/opt/sonarqube/logs \  -v sonarqube\_data:/opt/sonarqube/data \  sonarqube:lts-community |

<http://192.168.91.11:9000/>

admin/admin



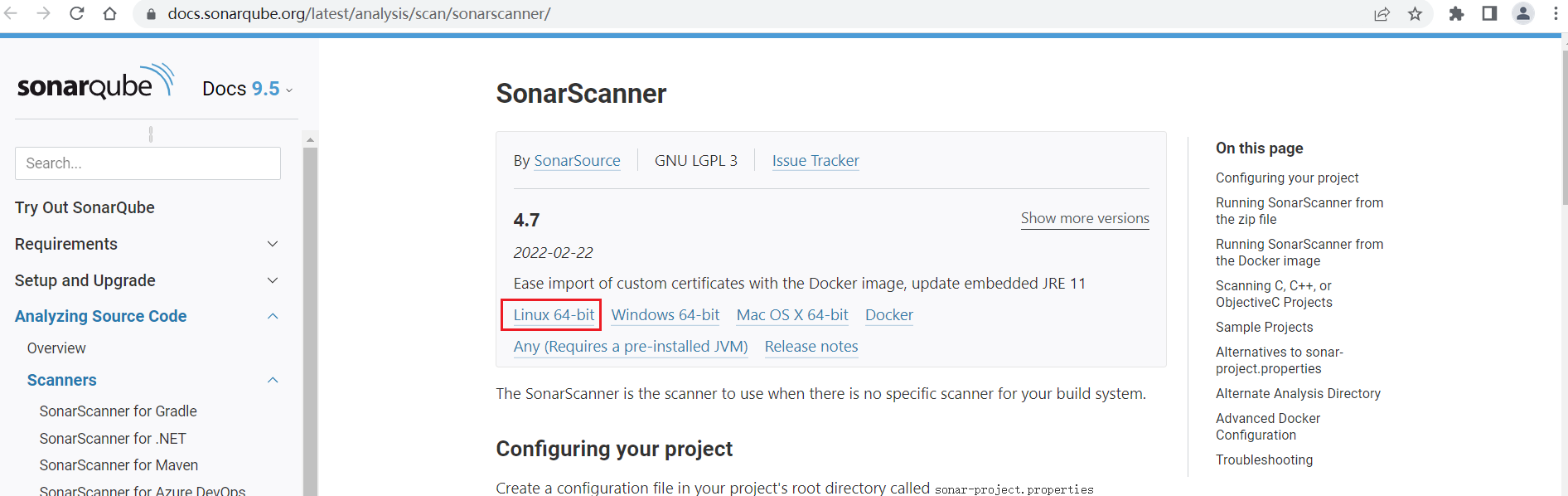
安装中文插件后会提示重启，重启后如下



**Ldap集成**

注意：sonar 8.0及之后的版本，LDAP插件就不支持了

**SonarScanner**

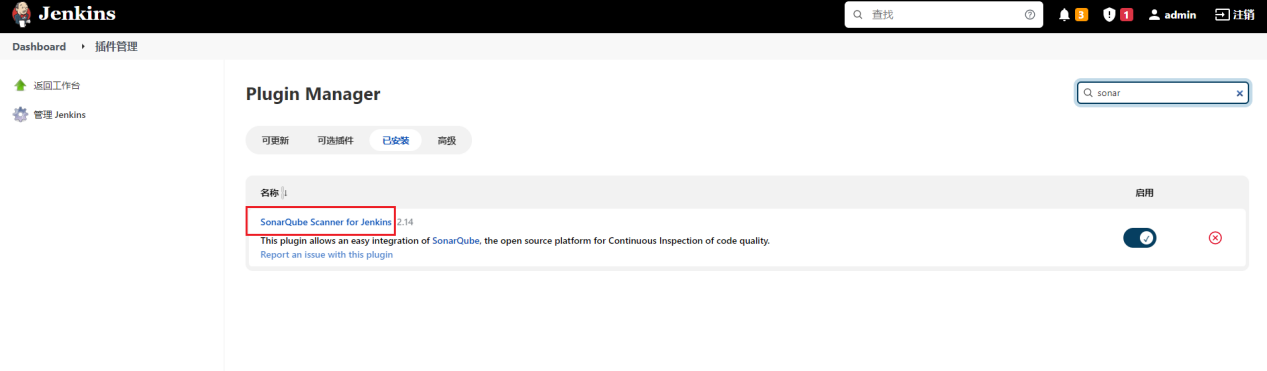
[https://docs.sonarqube.org/latest/analysis/scan/sonarscanner/](https://docs.sonarqube.org/latest/analysis/scan/sonarscanner/)

unzip sonar-scanner-cli-4.7.0.2747-linux.zip -d /usr/local/

cd /usr/local/ && ln -sfv sonar-scanner-4.7.0.2747-linux sonar-scanner

Jenkins 安装插件

[SonarQube Scanner](https://plugins.jenkins.io/sonar" \t "http://jenkins.itdance.cn/pluginManager/_blank)



编辑sonarqube.groovy



编辑ci.jenkinsfile

|  |
| --- |
| #!groovy  @Library('jenkinslib@master') \_  //func from sharelibrary  def build = new org.devops.build()  def deploy = new org.devops.deploy()  def tools = new org.devops.tools()  def gitlab = new org.devops.gitlab()  def toemail = new org.devops.toemail()  def sonar = new org.devops.sonarqube()  //env  String buildType = "${env.buildType}"  String buildShell = "${env.buildShell}"  String deployHosts= "${env.deployHosts}"  String srcUrl="${env.srcUrl}"  String branchName="{env.branchName}"  if ("${runOpts}" == "GitlabPush"){  branchName = branch - "refs/heads/"  currentBuild.description = "Trigger by ${userName} ${branch}"  gitlab.ChangeCommitStatus(projectId,commitSha,"running")  env.runOpts = "GitlabPush"  }  pipeline{  agent{ node { label "master"} }      stages{  stage("CheckOut"){  steps{  script{  println("${branchName}")  tools.PrintMes("构建分支为：${branchName}","green")  tools.PrintMes("获取代码","green")  checkout([$class: 'GitSCM', branches: [[name: '${branchName}']], extensions: [], userRemoteConfigs: [[credentialsId: '2', url: '${srcUrl}']]])  }  }  }  stage("Build"){  steps{  script{  tools.PrintMes("执行打包","green")  build.Build(buildType,buildShell)    //deploy.AnsibleDeploy("${deployHost}","-m ping")  }  }  }  stage("QA"){  steps {  script{  tools.PrintMes("代码扫描","green")  sonar.SonarScan("test","${JOB\_NAME}","${JOB\_NAME}","src")  }  }  }  }  post {  always{  script{  println("always")  }  }    success{  script{  println("success")  gitlab.ChangeCommitStatus(projectId,commitSha,"success")  toemail.Email("流水线成功",userEmail)  }    }  failure{  script{  println("failure")  gitlab.ChangeCommitStatus(projectId,commitSha,"failed")  toemail.Email("流水线失败了！",userEmail)  }  }    aborted{  script{  println("aborted")  gitlab.ChangeCommitStatus(projectId,commitSha,"canceled")  toemail.Email("流水线被取消了！",userEmail)  }    }    }  } |