**Day3-06月19日-版本控制系统与构建工具实践**

|  |  |  |
| --- | --- | --- |
| **时间** | **标题** | **内容** |
| **上午** | **版本控制系统（GitLab）** | 1. GitLab安装部署配置 2. GitLab web页面的基本功能 3. 从创建项目到提交代码的开发过程 4. GitLab WebHook简介与应用 5. 配置代码提交触发CI流水线(限制分支) |
| **下午** | **构建工具与消息通知** | 1. 提交流水线的优化（避免创建分支/tag后自动触发） 2. CI流水线中的Clone Code 阶段实现 3. 配置CI流水线中的构建反馈（邮件） 4. 更新Gitlab中commit的提交状态 |
| 1. 常用的项目构建工具简介（maven/npm） 2. CI流水线集成代码Build阶段 |
|  | **资源清单**：  Postman下载： <https://www.postman.com/downloads/>  Gitlab接口文档：<https://docs.gitlab.com/13.10/ee/api/README.html> | |

Jenkins完整代码快照： day3 （完整的jenkinshome目录， 可以下载后Docker方式运行起来） [📎jenkins-day03.tar.gz](https://www.yuque.com/attachments/yuque/0/2021/gz/2584012/1624096366392-299029c6-4f11-46ef-b642-92e6f0a9b0bf.gz)

day3 sublime text 代码： [📎day3.zip](https://www.yuque.com/attachments/yuque/0/2021/zip/2584012/1624095239611-4d67c89b-93ca-4cb1-abee-0733f3cac926.zip)



jenkins共享库代码： [📎jenkinslib-day3.zip](https://www.yuque.com/attachments/yuque/0/2021/zip/2584012/1624095069447-c2a2eb3d-f48f-482b-8c80-c1f8f43b459a.zip)

postman 实验接口文件： [📎test.postman\_collection.json](https://www.yuque.com/attachments/yuque/0/2021/json/2584012/1624095100117-25e37e14-72d0-40f7-b3b3-5bb0b0c68375.json)  下载这个文件然后导入到你的本地postman



**1. 预备知识**

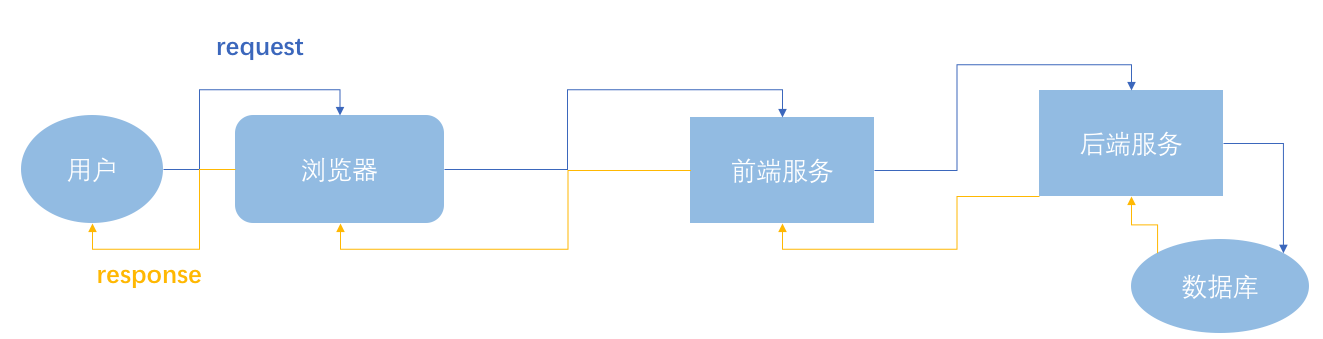
**目标：学习HTTP基础知识，掌握如何使用Postman和Curl调用接口的方法。**

**1.1 Web HTTP基础知识**

**HTTP请求是什么？**

HTTP超文本传输协议，是确保服务器（Server）和客户端（Client）之间的正确通信。

一个请求和响应的过程：



* **Request** 用户通过浏览器向我们的服务端发起请求。
* **Response**服务端将客户端请求的资源数据进行响应。

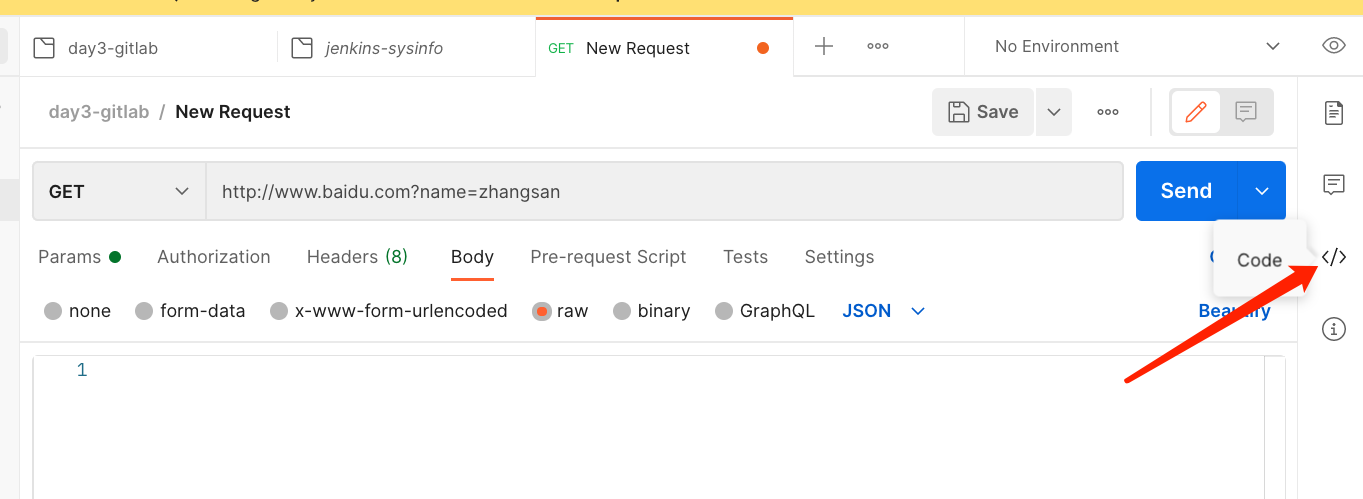
**调用接口的方法**

postman下载：

请求类型：

* GET 向指定的URL请求资源，可携带参数（明文）。
* POST 向指定的URL提交资源，表单数据提交，数据进行封装（比Get方法安全）。
* PUT 与POST类似，通常用于对资源数据的更新修改。
* DELETE 删除指定的资源。

演示：以baidu为例，用Postman调用一个接口的过程。并结合使用Curl操作。



**HTTP常见的错误码**

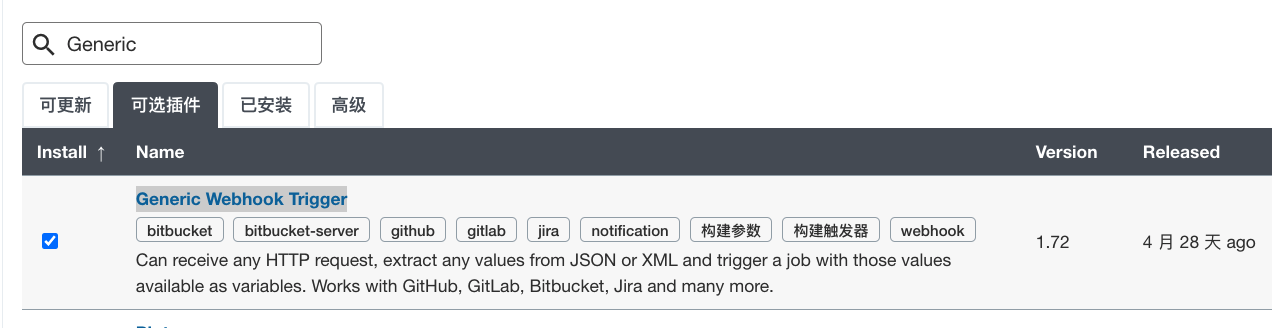
* 1xx : 服务已收到请求，请求者继续执行操作。
* **2xx：请求成功，常见（201）**
* 3xx：请求成功，页面发生重定向（301）
* **4xx：客户端发生了错误**
  + 常见的是资源找不到了（404）
  + 资源请求需要认证认证失败（401）
  + 认证成功但是权限不够（403）
* **5xx： 服务端发生了错误**
  + 网关超时（504）
  + 内部错误（500）
  + 网关错误（502）
  + 服务不可用（503）

**1.2 Jenkins Generic Webhook实践**

**目的： 为Jenkins作业添加触发器，便于其他系统调用。**

**安装配置Generic WebHook**

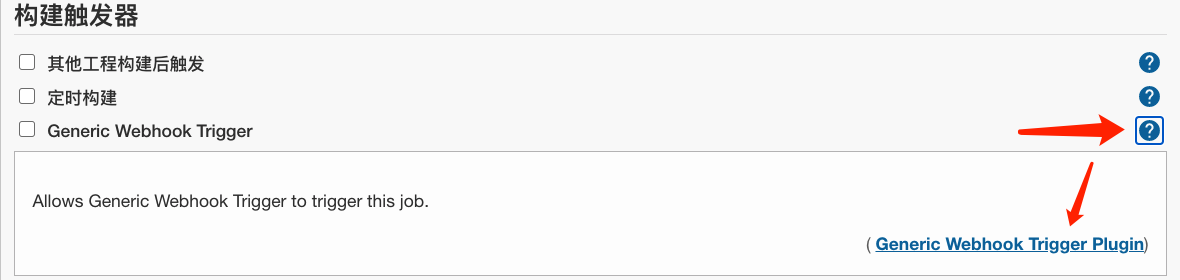
**插件名称：**[**Generic Webhook Trigger**](https://plugins.jenkins.io/generic-webhook-trigger)



**重启后**，进入一个Pipeline项目设置，已经可以选择这个触发器了....

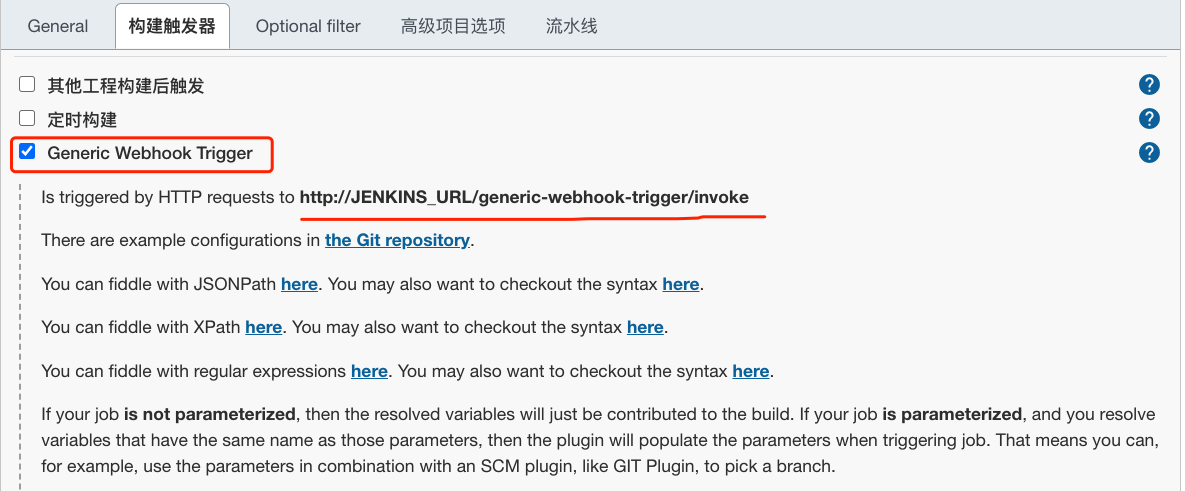


**插件文档**：<https://plugins.jenkins.io/generic-webhook-trigger/>



**Jenkins作业配置触发器**

启动Generic Webhook触发器后， 相当于给Jenkins加了一个**新的接口**（<http://JENKINS_URL/generic-webhook-trigger/invoke>）。



调用的时候：**这里要把 JENKINS\_URL 换成自己真实的Jenkins 服务器地址**，有端口就加上端口，是域名就写域名。下面是一个参考的URL：

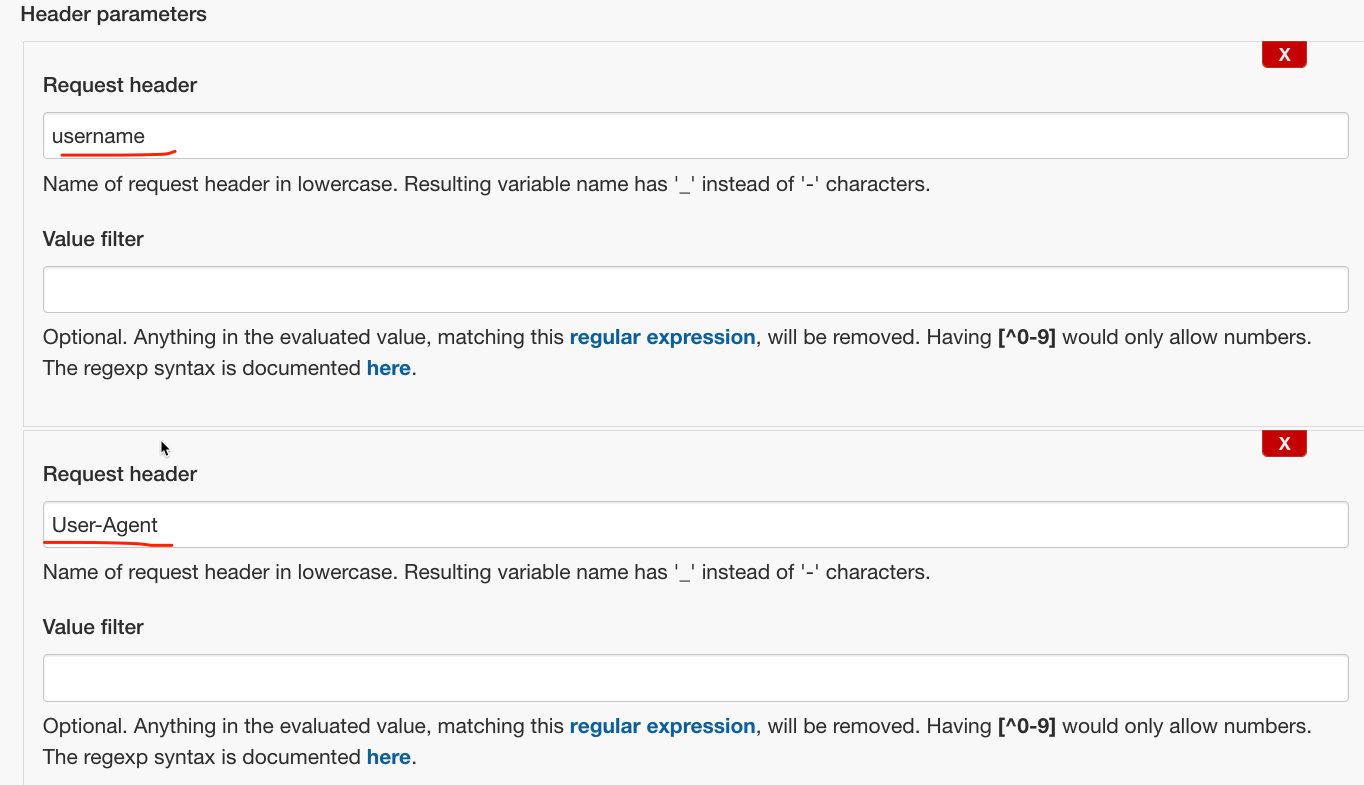
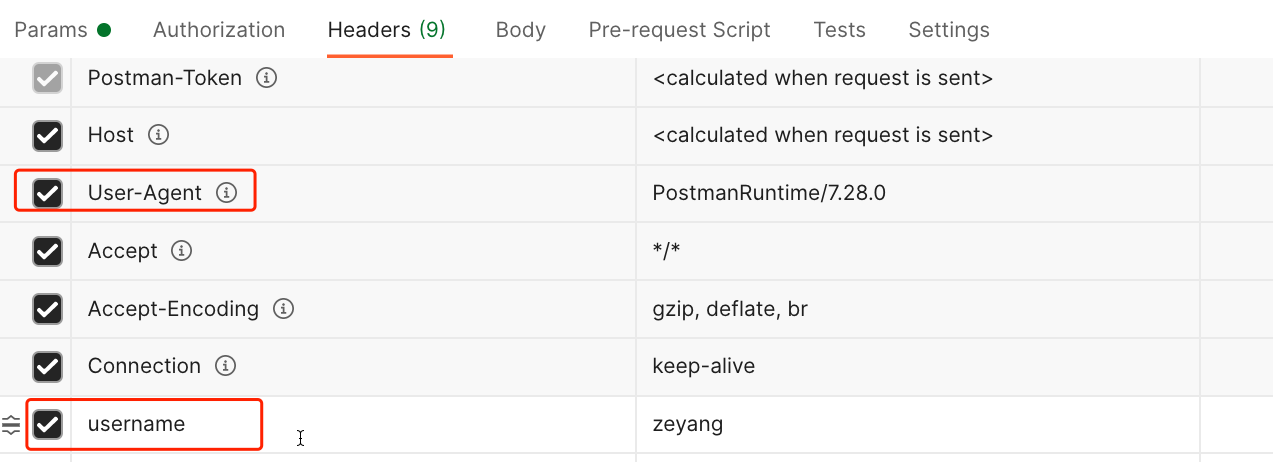
http://192.168.1.200:8080/generic-webhook-trigger/invoke

**Post content parameters： 获取调用接口传进来的数据**

JsonPath语法文档： <https://github.com/json-path/JsonPath>

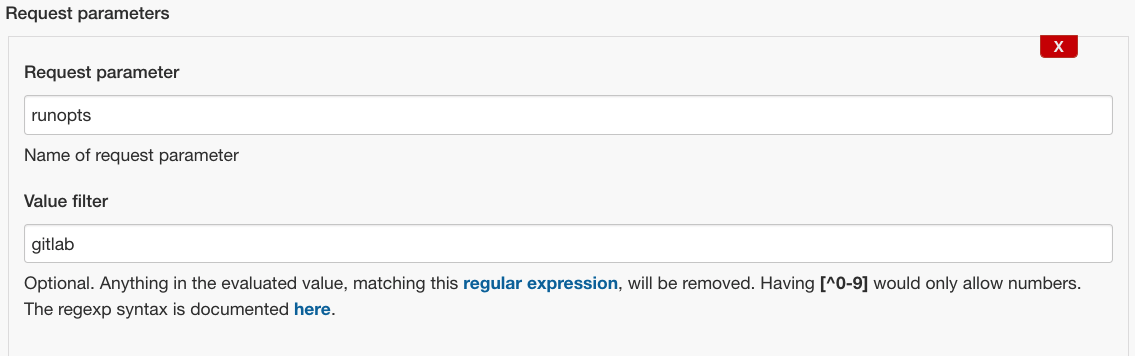


**Header parameters： 获取Header中的参数**



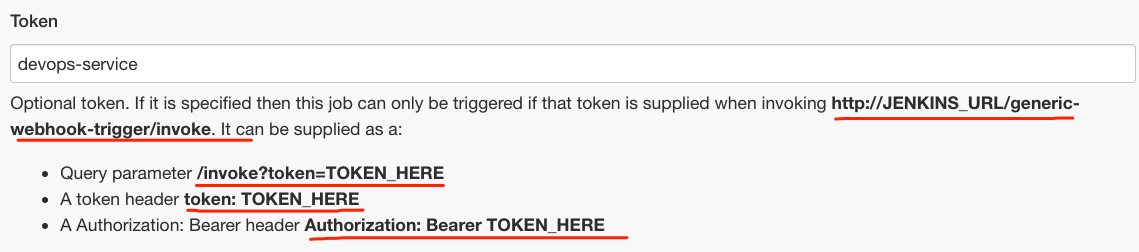
**Request parameters： 获取URL中的请求参数**

curl http://192.168.1.200:8080/generic-webhook-trigger/invoke?runopts=gitlab

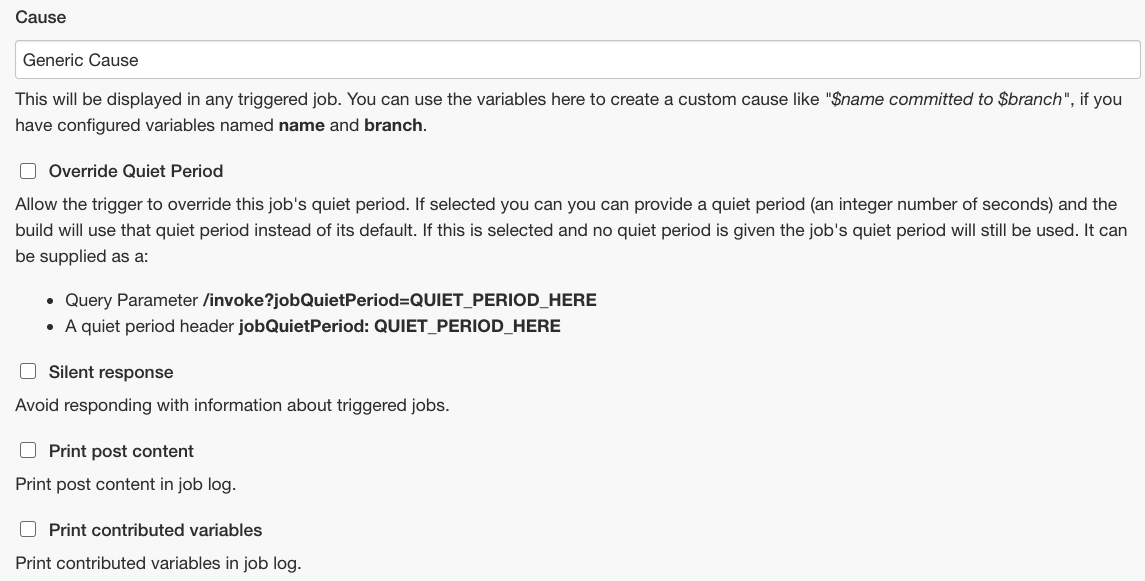


**Token: 给URL添加一个触发的认证**

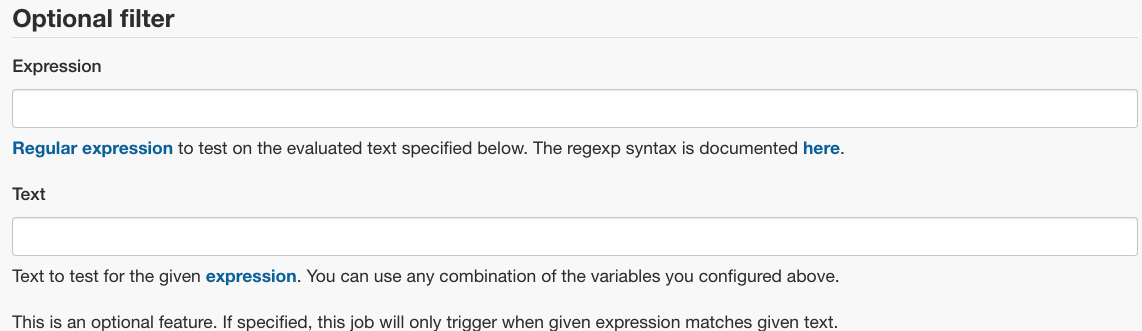
curl http://192.168.1.200:8080/generic-webhook-trigger/invoke?token=devops-service



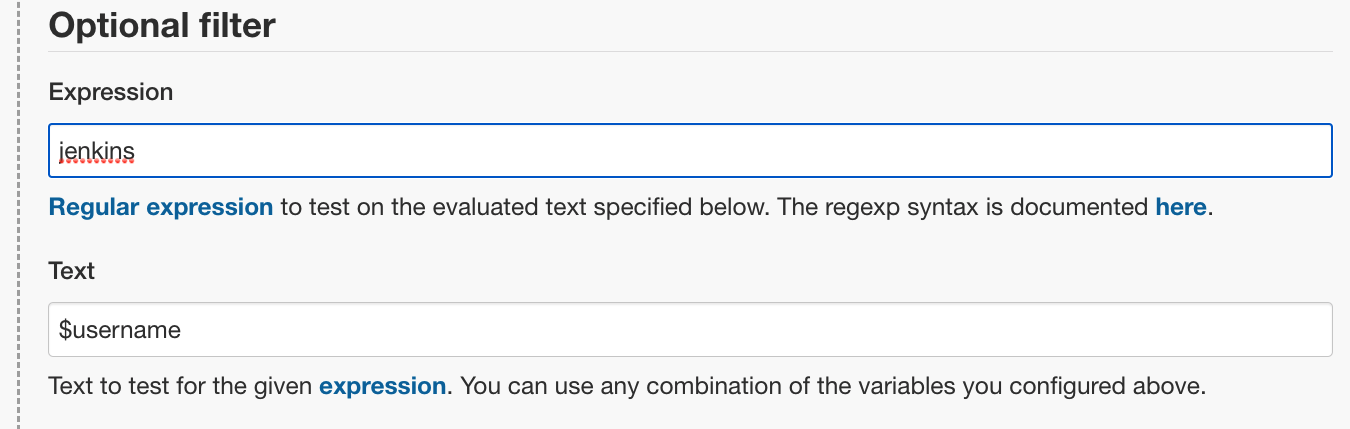
**打印调试信息到日志中**



**触发条件过滤：仅满足条件才能触发此作业**



实践：demo

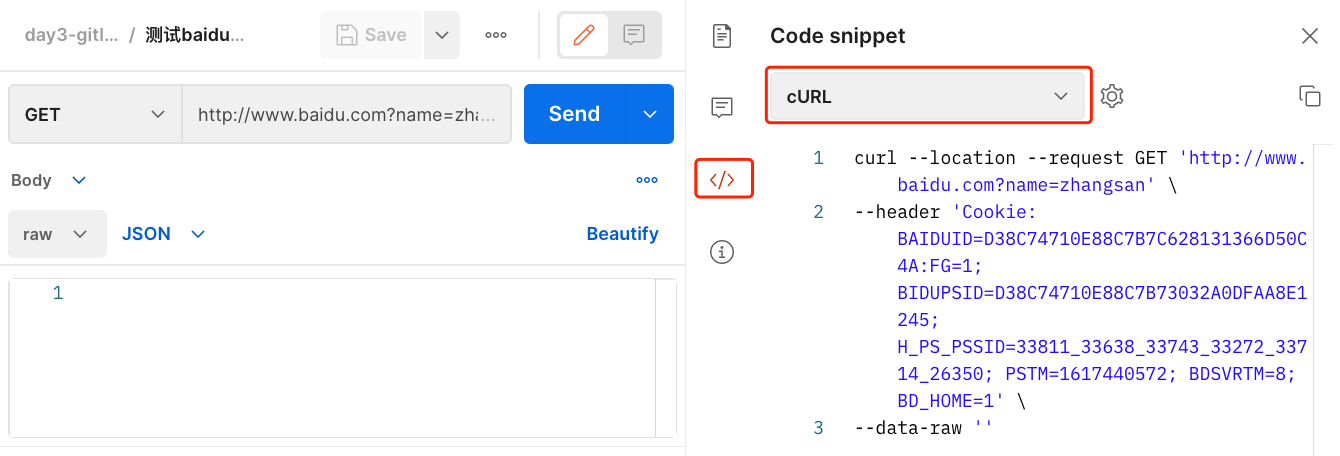


**解析GET/POST请求数据**

传参注意，**第一个参数使用?号连接， 后面的参数使用&符号连接。**

?token=demo-pipeline-service&user=jenkins&a=1&b=2

演示将postman中的请求转换curl方式。



[root@zeyang-nuc-service ~]# curl --location --request GET 'http://192.168.1.200:8080/generic-webhook-trigger/invoke?token=demo-pipeline-service'

{"jobs":{"demo-pipeline-service":{"regexpFilterExpression":"","triggered":true,"resolvedVariables":{},"regexpFilterText":"","id":209,"url":"queue/item/209/"}},"message":"Triggered jobs."}[root@zeyang-nuc-service ~]#

**解析GET数据**

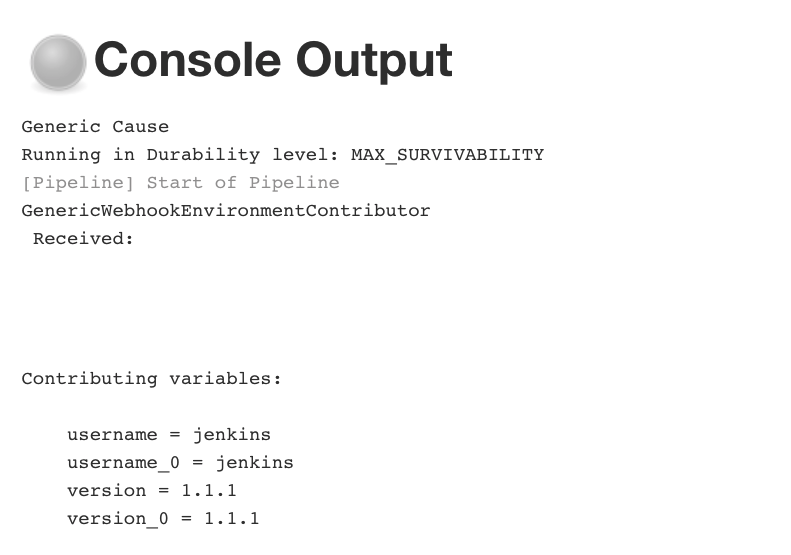
* 客户端发送Get请求， 带有两个参数 version 和 username 。

http://192.168.1.200:8080/generic-webhook-trigger/invoke?token=demo-pipeline-service&version=1.1.1&username=jenkins

* Jenkins 配置Generic hook，获取请求参数version 和 username （**参数名称要一致**）



* 验证测试（**Jenkins日志中能够打印出获取的值，则正常**）



* 通过jenkinsfile读取传递的参数

println("${username}")

println("${version}")

**解析HEADER参数数据**

* 客户端发送请求（**什么请求都可以，这里的header与请求类型无关**）， 带有两个参数 header\_name 和 header\_id 。

curl --location --request GET 'http://192.168.1.200:8080/generic-webhook-trigger/invoke?token=demo-pipeline-service' \

--header 'header\_name: jenkins' \

--header 'header\_id: 100'

* Jenkins 配置Generic hook，获取请求参数header\_name 和 header\_id 。



* 通过jenkinsfile读取传递的参数

println("${header\_id}")

println("${header\_name}")

println(header\_id)

println(header\_name)

String headerName = "${header\_id}"

println(headerName)

//pipeline {

//}

**解析POST数据**

* 客户端发送POST请求， 参数存储在body体中（**参考POSTMAN中的样例**）

curl --location --request POST 'http://192.168.1.200:8080/generic-webhook-trigger/invoke?token=demo-pipeline-service' \

--header 'Content-Type: application/json' \

--data-raw '{

"name": "zhangsan",

"id": "123",

"group1": {

"name": "jenkins",

"id" : "001",

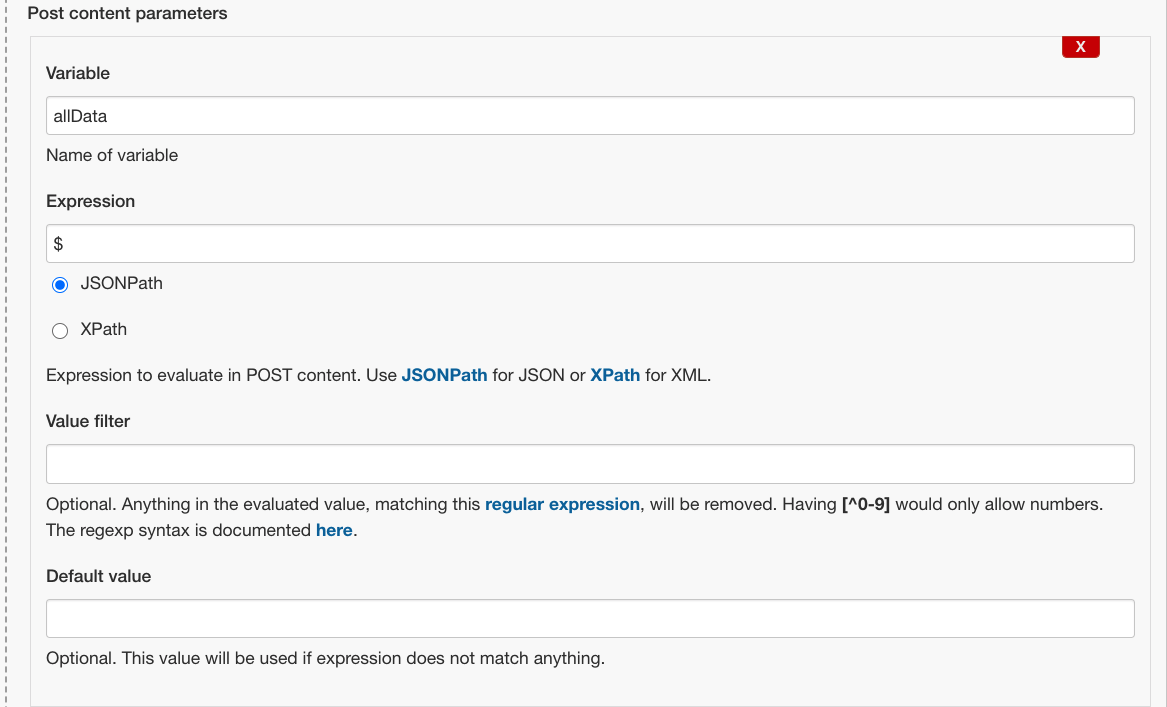
"age": "40"

}

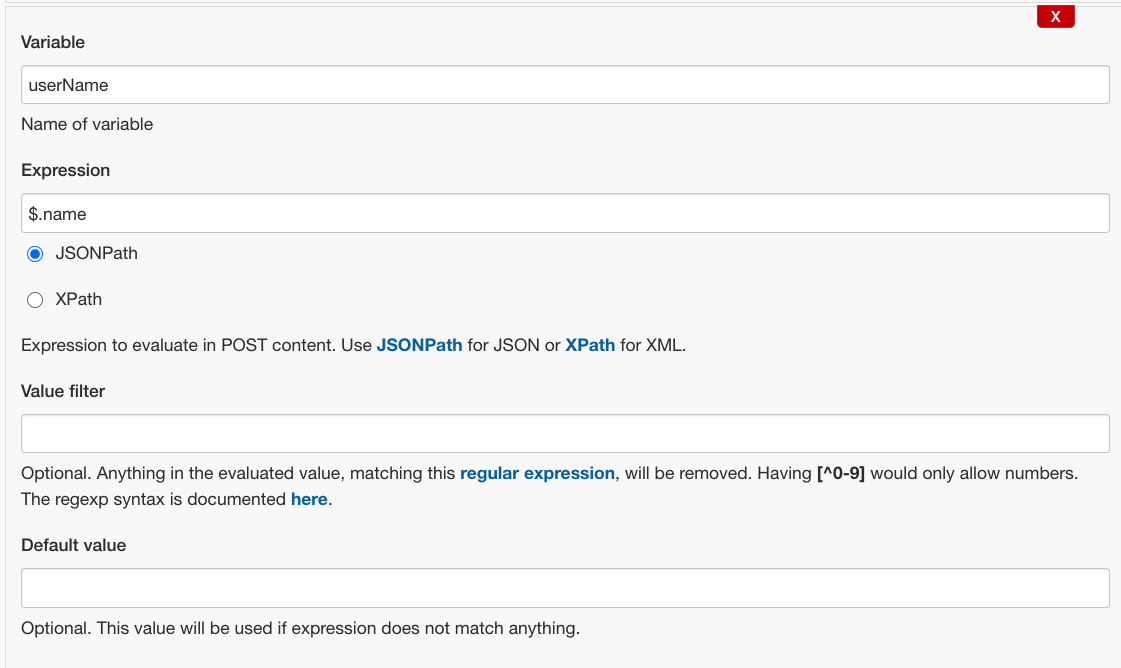
}'

* Jenkins 配置触发器来获取Post参数。

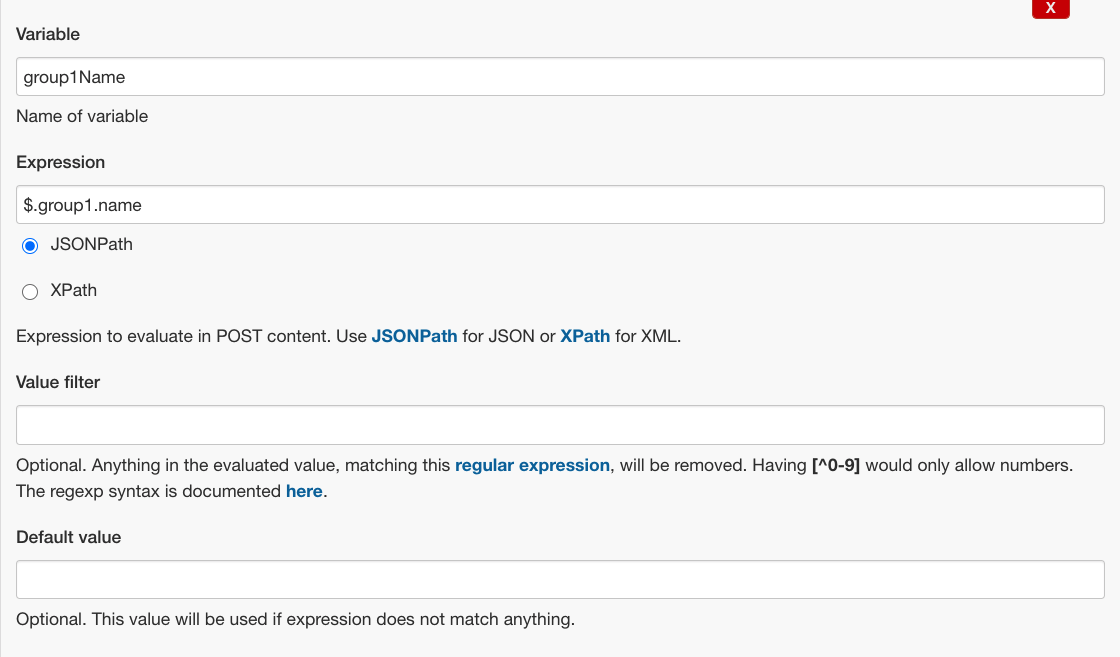
获取所有数据



获取username字段



获取group1Name字段



* Jenkinsfile中使用参数

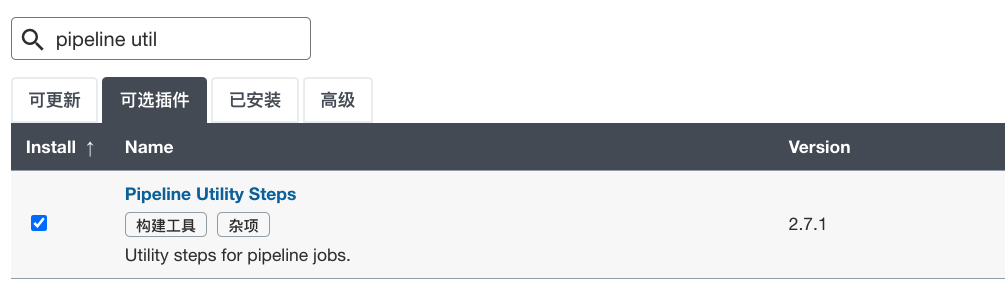
println("所有body数据 --> ${allData}")

println('最外层name ---> $.name' + "${userName}")

println('第二层name ---> $.group1.name' + "${group1Name}")

**扩展流水线解析JSON数据**

安装插件： [Pipeline Utility Steps](https://plugins.jenkins.io/pipeline-utility-steps)



readJSON： 处理json数据

println("所有body数据 --> ${allData}")

def webHookData = readJSON text: "${allData}"

String userName = webHookData["name"]

String userName2 = webHookData.name

String group1Name = webHookData["group1"]["name"]

String group1Name2 = webHookData.group1.name

println('最外层name ---> $.name' + "${userName}")

println('最外层name ---> $.name' + "${userName2}")

println('第二层name ---> $.group1.name' + "${group1Name}")

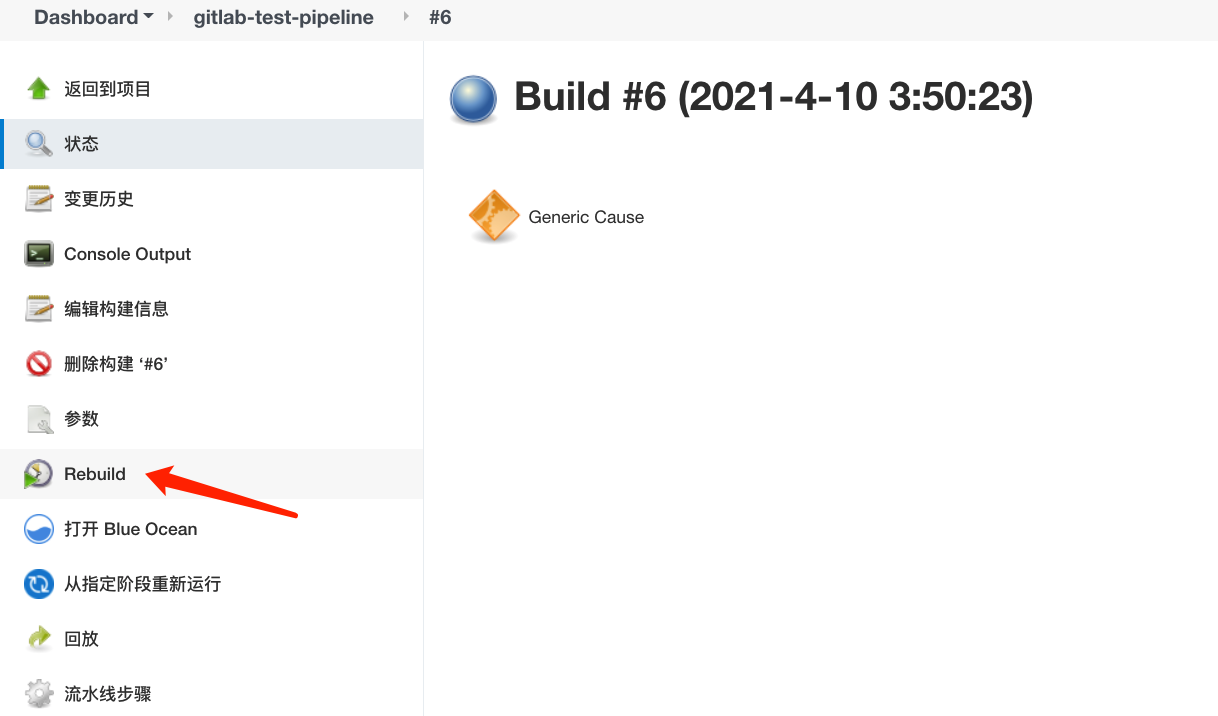
println('第二层name ---> $.group1.name' + "${group1Name2}")

**Rebuilder 插件使用**

可以直接携带原触发参数进行触发，不用重复触发（不用在重复的提交代码了）。



进入某一次构建后，可以点击rebuild。



休息20分钟， 11点继续

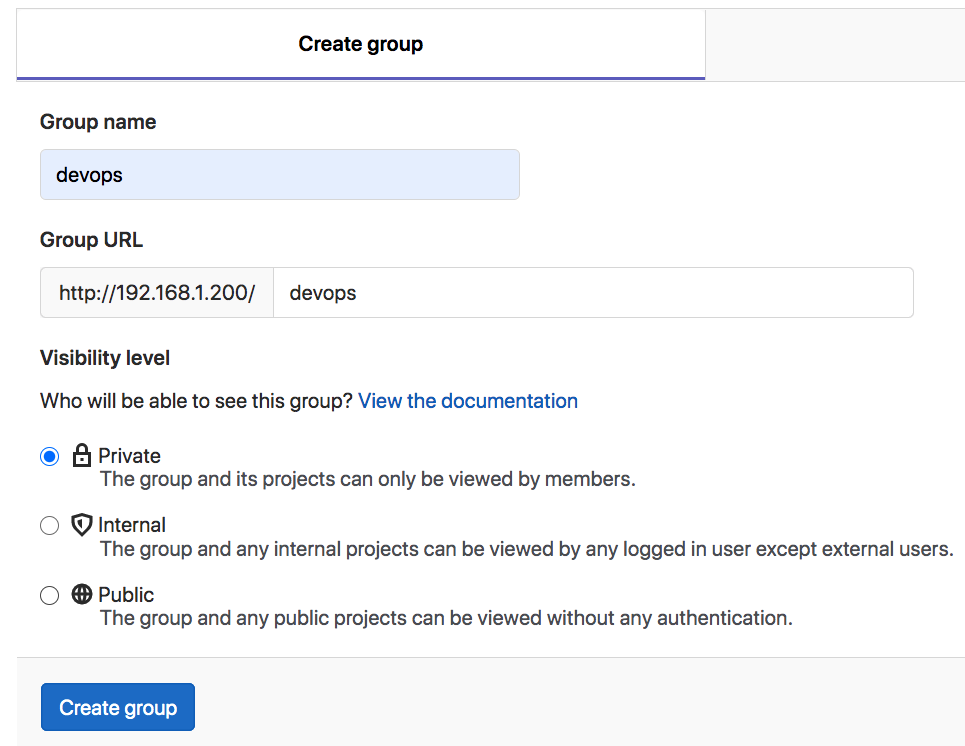
**2. 基于GitLab的开发工作流**

root /admin123

**创建一个项目组**

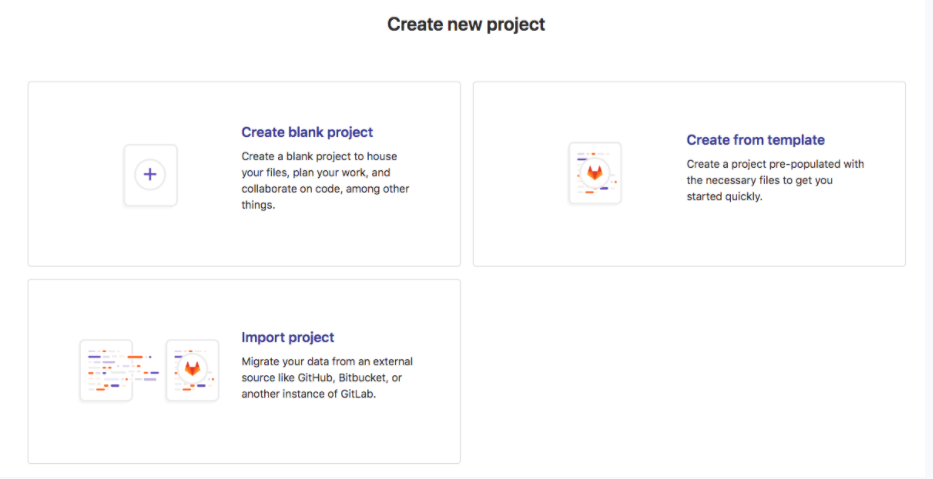
配置group名称最好与项目组有关的，例如业务的简称等等。项目组的类型分为 Private、Internal、Public三种类型。

* Private 私有类型（当group为私有类型，后面组下面的项目都是私有类型）
* Public 公开类型

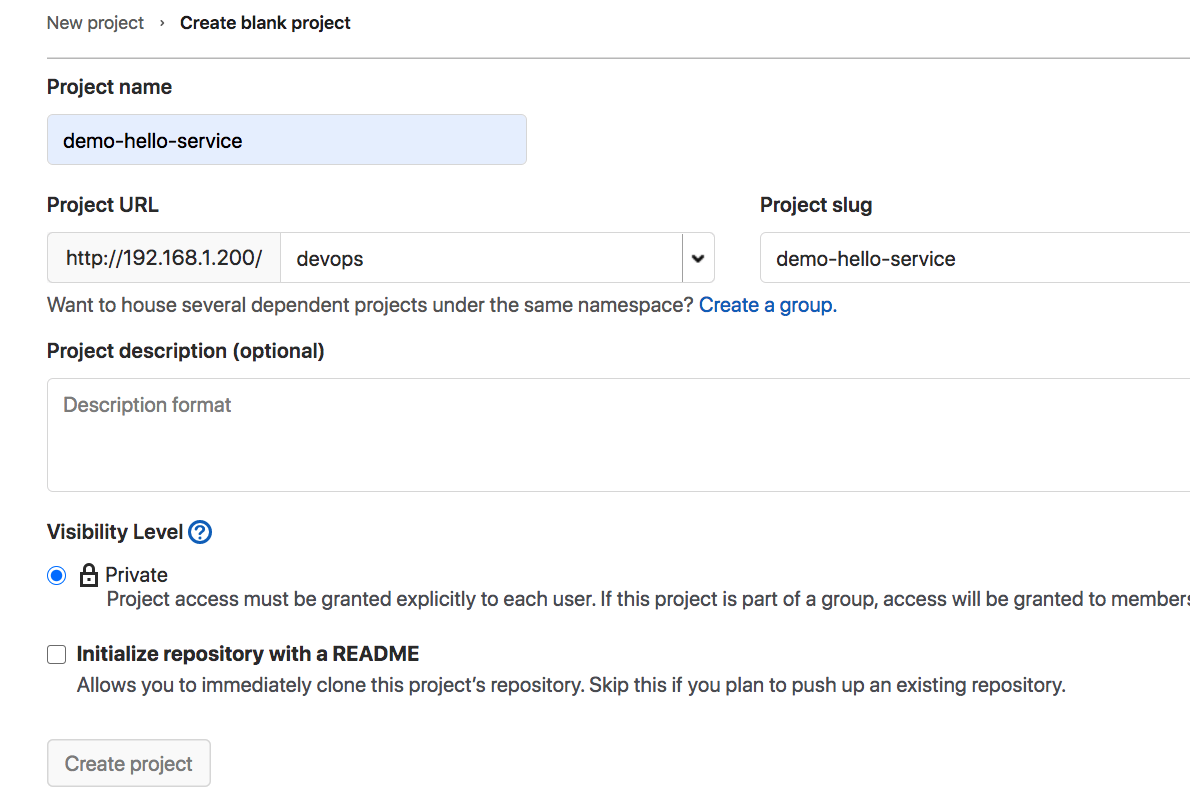


**创建一个项目**

在这个页面可以创建一个空的项目、根据一个模板创建项目、导入一个已存在的项目（Gitlab、GitHub等系统）



创建一个 demo-hello-service 项目，私有类型。



**将代码导入项目**

在执行git 客户端命令之前，可以先设置当前的用户信息，名称和邮箱。（可选、规范化）

git config --global user.name "Administrator"

git config --global user.email "admin@example.com"

下载代码库，并在代码库中创建文件提交。

### 下载项目

# git clone http://192.168.1.200/devops/demo-hello-service.git

Cloning into 'demo-hello-service'...

Username for 'http://192.168.1.200': root

Password for 'http://root@192.168.1.200':

warning: You appear to have cloned an empty repository.

### 查看项目

# ls | grep demo

demo-hello-service

demo-java-service

microservicecicd-demo-service

spinnaker-canary-demo

### 进入项目中

#### 创建文件

# cd demo-hello-service/

# echo devopsdevops >> jenkins.txt

# ls

jenkins.txt

### 提交文件到远程仓库

# git add jenkins.txt

# git commit -m "add jenkins.txt "

[master (root-commit) d74d541] add jenkins.txt

1 file changed, 1 insertion(+)

create mode 100644 jenkins.txt

[root@zeyang-nuc-service demo-hello-service]# git push origin master

Username for 'http://192.168.1.200': root

Password for 'http://root@192.168.1.200':

Enumerating objects: 3, done.

Counting objects: 100% (3/3), done.

Writing objects: 100% (3/3), 221 bytes | 221.00 KiB/s, done.

Total 3 (delta 0), reused 0 (delta 0)

To http://192.168.1.200/devops/demo-hello-service.git

\* [new branch] master -> master

扩展：将本地已存在的代码提交到远程仓库

cd existing\_folder

git init

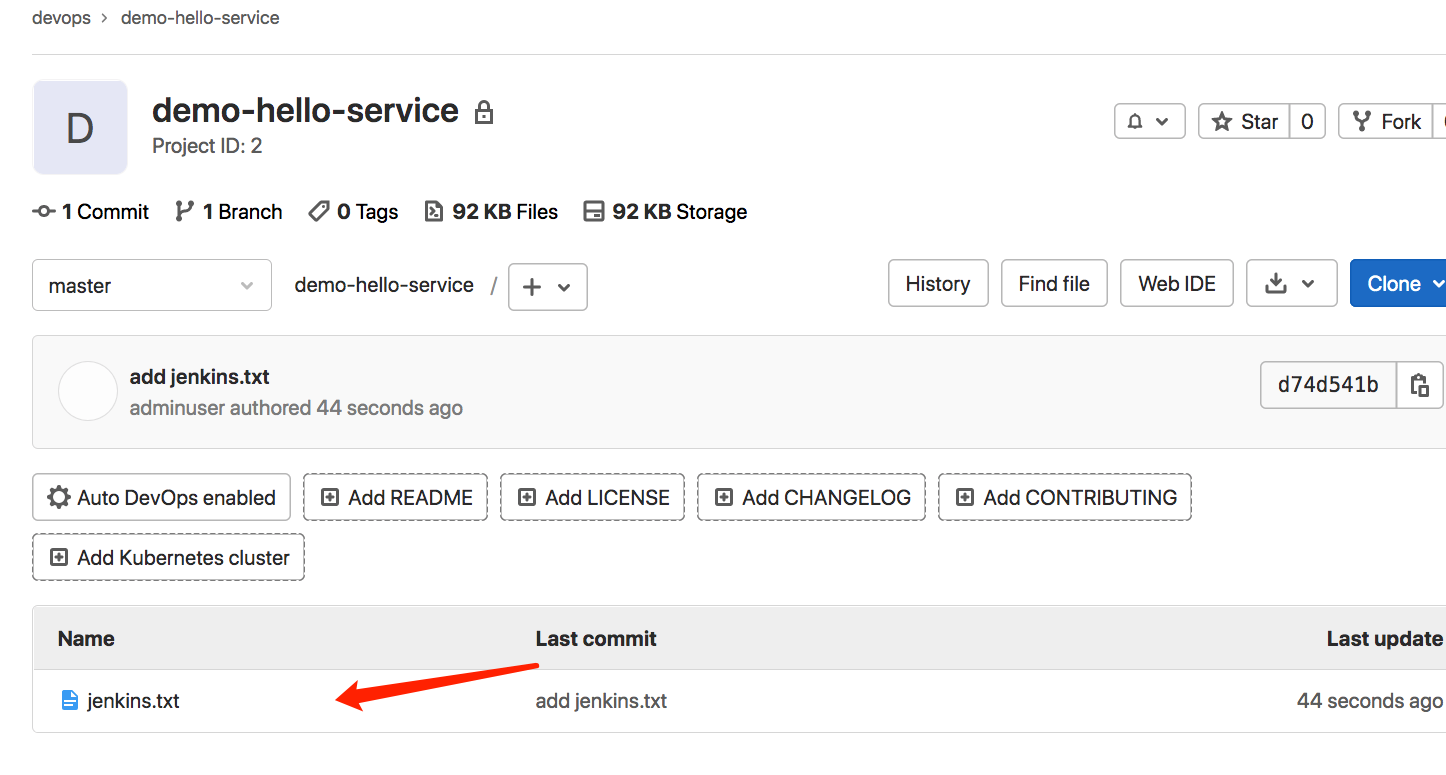
git remote add origin http://192.168.1.200/devops/demo-hello-service.git

git add .

git commit -m "Initial commit"

git push -u origin master

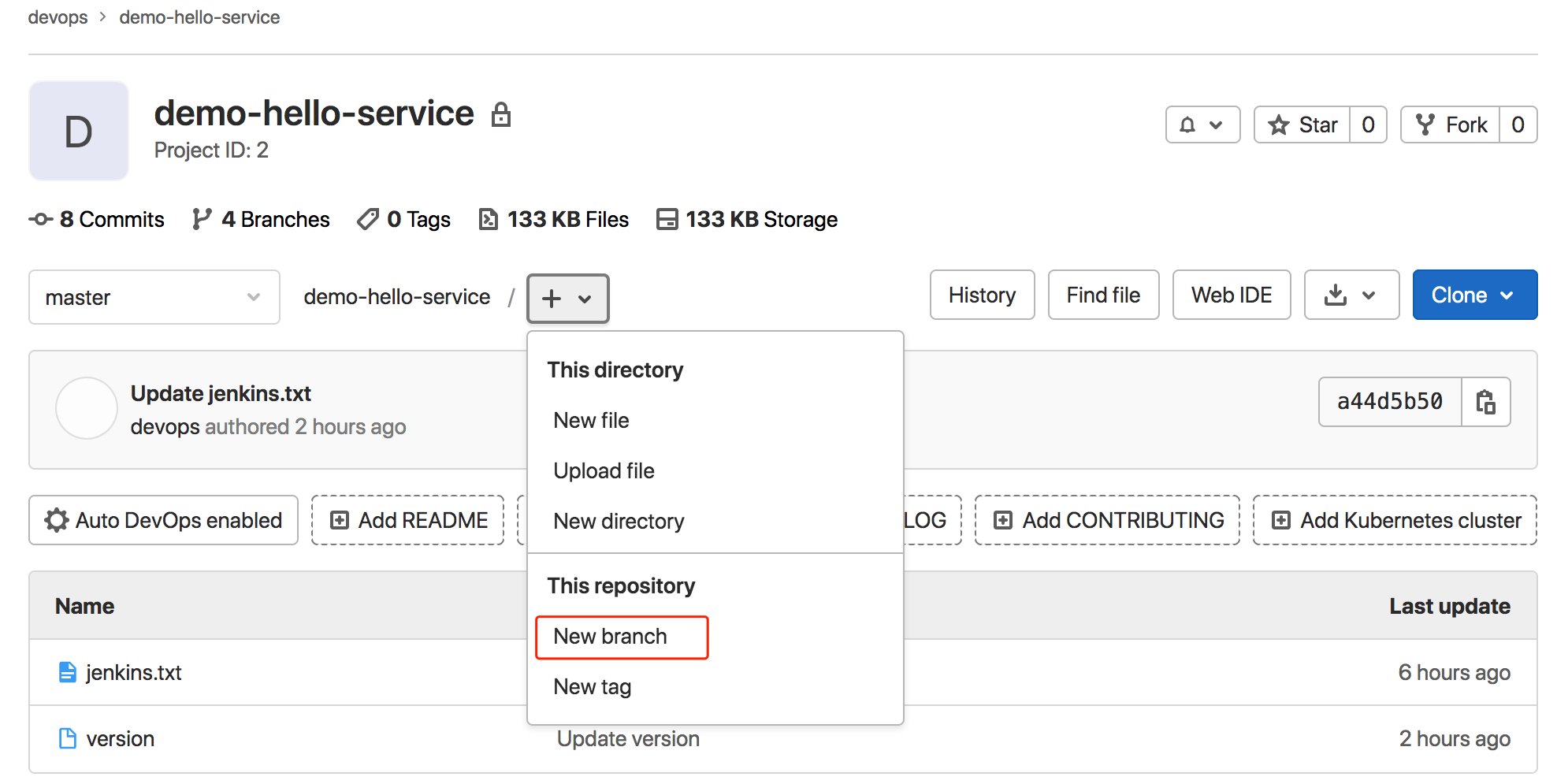
代码提交后的效果：发现 jenkins.txt 文件已经从本地同步到了远程的gitlab仓库中了。

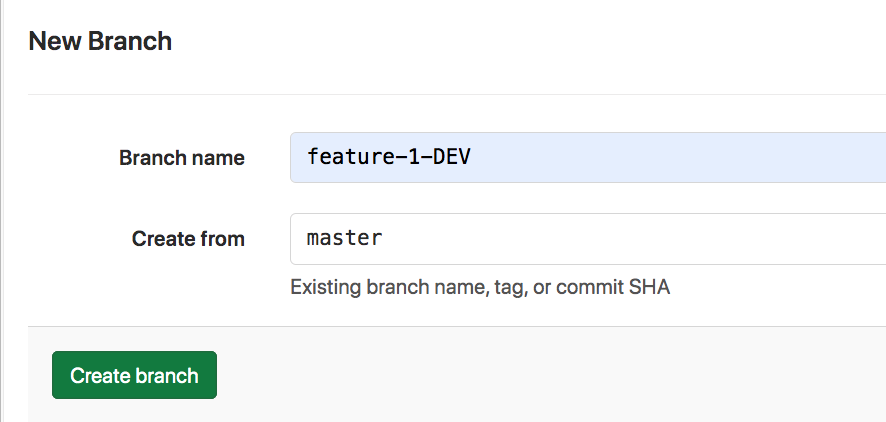


**拉取特性分支**

为什么要拉取分支？ 一个分支不够吗？ 一般我们使用 master 主干分支存放最新的能够发布生产的代码，而单独创建一些特性分支来做项目需求任务的开发分支。 这样的好处是防止主干分支污染，对分支起到了保护的作用。

下面进入 demo-hello-service 项目主页，然后基于主干分支master，创建特性分支feature-1-DEV。操作如下：





**特性分支开发与提交**

查看当前本地分支，发现没有刚刚远程创建的 feature-1-DEV 分支。

[root@zeyang-nuc-service demo-hello-service]# git branch -a

\* master

remotes/origin/master

`git pull` 同步远程仓库所做的更新到本地， 这样远程的feature-1-DEV 分支就同步到了本地。然后我们使用 git checkout feature-1-DEV 切换到特性分支。

root@zeyang-nuc-service demo-hello-service]# git pull

Username for 'http://192.168.1.200': root

Password for 'http://root@192.168.1.200':

From http://192.168.1.200/devops/demo-hello-service

\* [new branch] feature-1-DEV -> origin/feature-1-DEV

Already up to date.

[root@zeyang-nuc-service demo-hello-service]# git branch -a

\* master

remotes/origin/feature-1-DEV

remotes/origin/master

[root@zeyang-nuc-service demo-hello-service]# git checkout feature-1-DEV

Branch 'feature-1-DEV' set up to track remote branch 'feature-1-DEV' from 'origin'.

Switched to a new branch 'feature-1-DEV'

[root@zeyang-nuc-service demo-hello-service]# git branch -a

\* feature-1-DEV

master

remotes/origin/feature-1-DEV

remotes/origin/master

我们更改了Jenkins.txt文件内容，并创建一个新的version文件，然后将更改内容提交到远程仓库。

### 更改文件内容

[root@zeyang-nuc-service demo-hello-service]# ls

jenkins.txt

[root@zeyang-nuc-service demo-hello-service]# vi jenkins.txt

[root@zeyang-nuc-service demo-hello-service]# cat jenkins.txt

pipeline {

agent { label "master"}

stages{

stage("Build"){

steps {

script {

echo "hello"

}

}

}

}

}

[root@zeyang-nuc-service demo-hello-service]# echo 1.1.1 >version

[root@zeyang-nuc-service demo-hello-service]# ls

jenkins.txt version

### 提交到远程仓库分支

[root@zeyang-nuc-service demo-hello-service]# git add jenkins.txt version

[root@zeyang-nuc-service demo-hello-service]# git commit -m "add pipeline "

[feature-1-DEV 97ae23e] add pipeline

2 files changed, 14 insertions(+), 1 deletion(-)

create mode 100644 version

[root@zeyang-nuc-service demo-hello-service]# git push origin feature-1-DEV

Username for 'http://192.168.1.200': root

Password for 'http://root@192.168.1.200':

Enumerating objects: 6, done.

Counting objects: 100% (6/6), done.

Delta compression using up to 8 threads.

Compressing objects: 100% (3/3), done.

Writing objects: 100% (4/4), 380 bytes | 380.00 KiB/s, done.

Total 4 (delta 0), reused 0 (delta 0)

remote:

remote: To create a merge request for feature-1-DEV, visit:

remote: http://192.168.1.200/devops/demo-hello-service/-/merge\_requests/new?merge\_request%5Bsource\_branch%5D=feature-1-DEV

remote:

To http://192.168.1.200/devops/demo-hello-service.git

d74d541..97ae23e feature-1-DEV -> feature-1-DEV

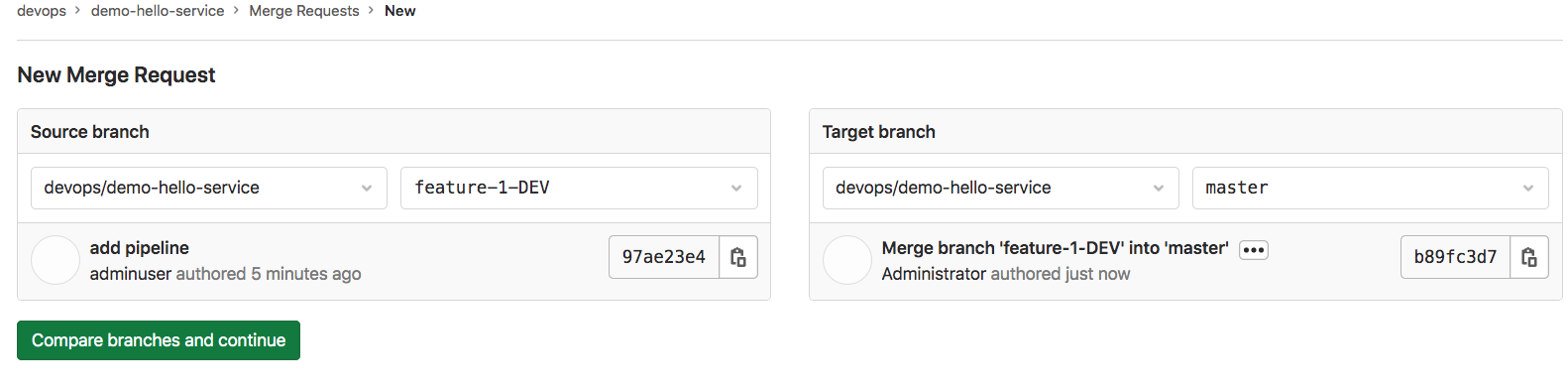
这样我们就把本地的特性分支开发的代码提交到了远程特性分支中了， 接下来对应该对该特性分支进行测试验证，没问题后合并到主干分支。

**特性分支合并操作**

将特性分支 feature-1-DEV 代码合并到主干分支master  Merge Request。

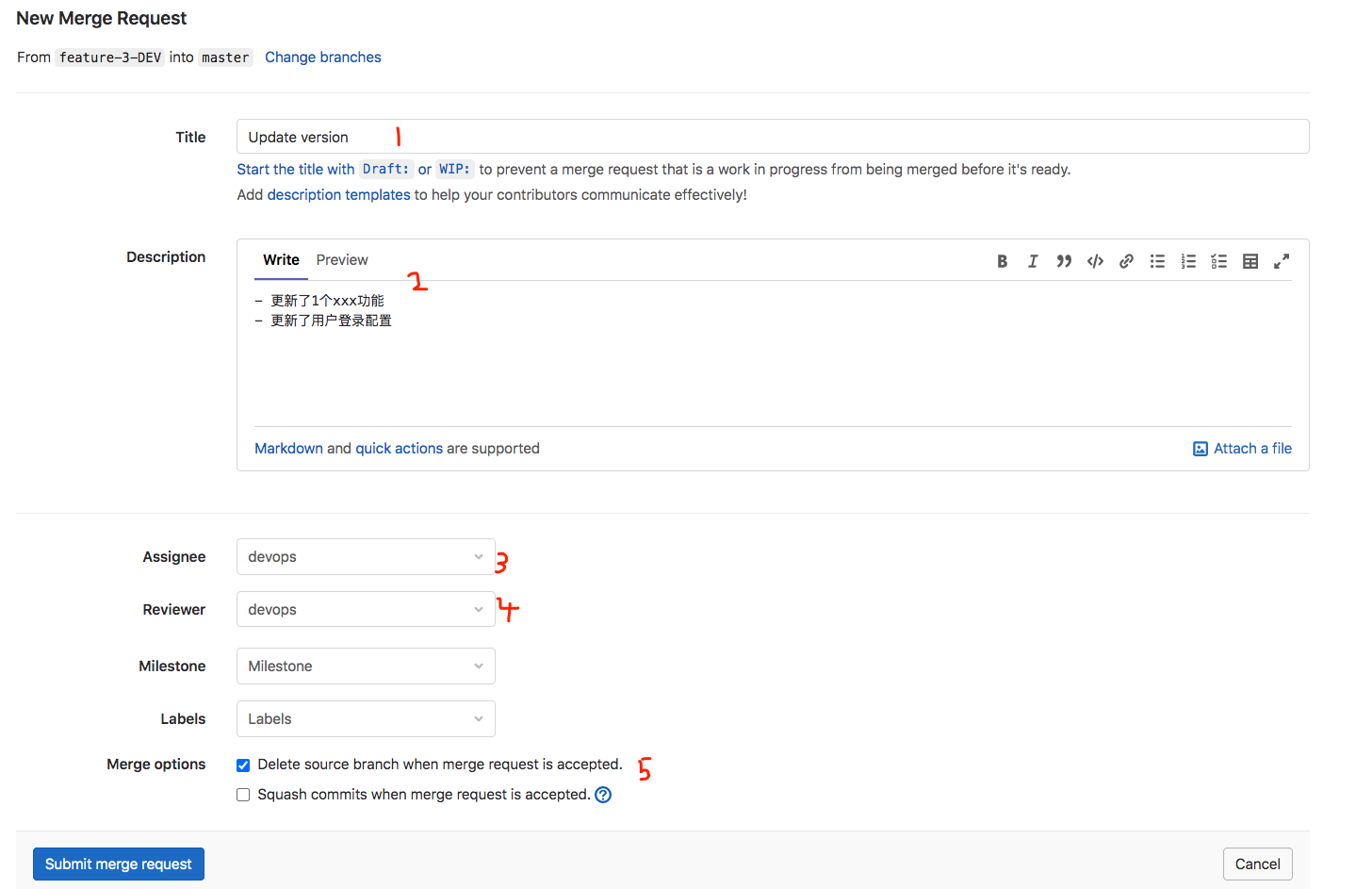


在这个页面，选择源分支和目标分支。

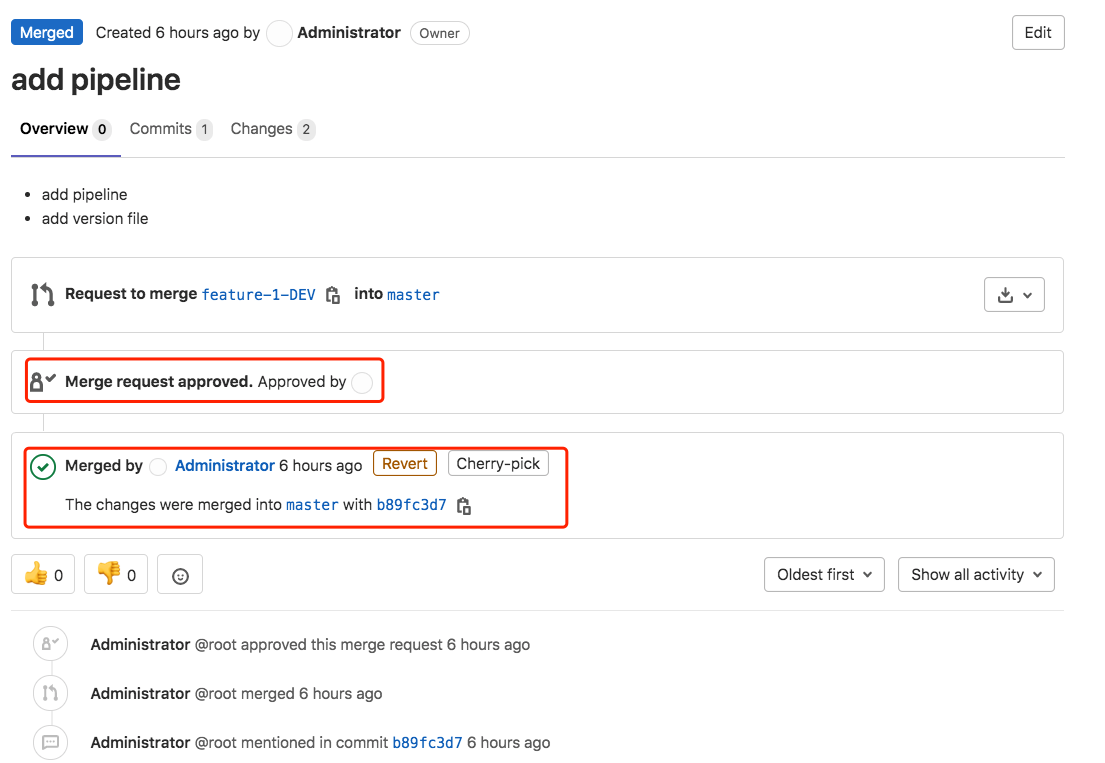


在这个页面：

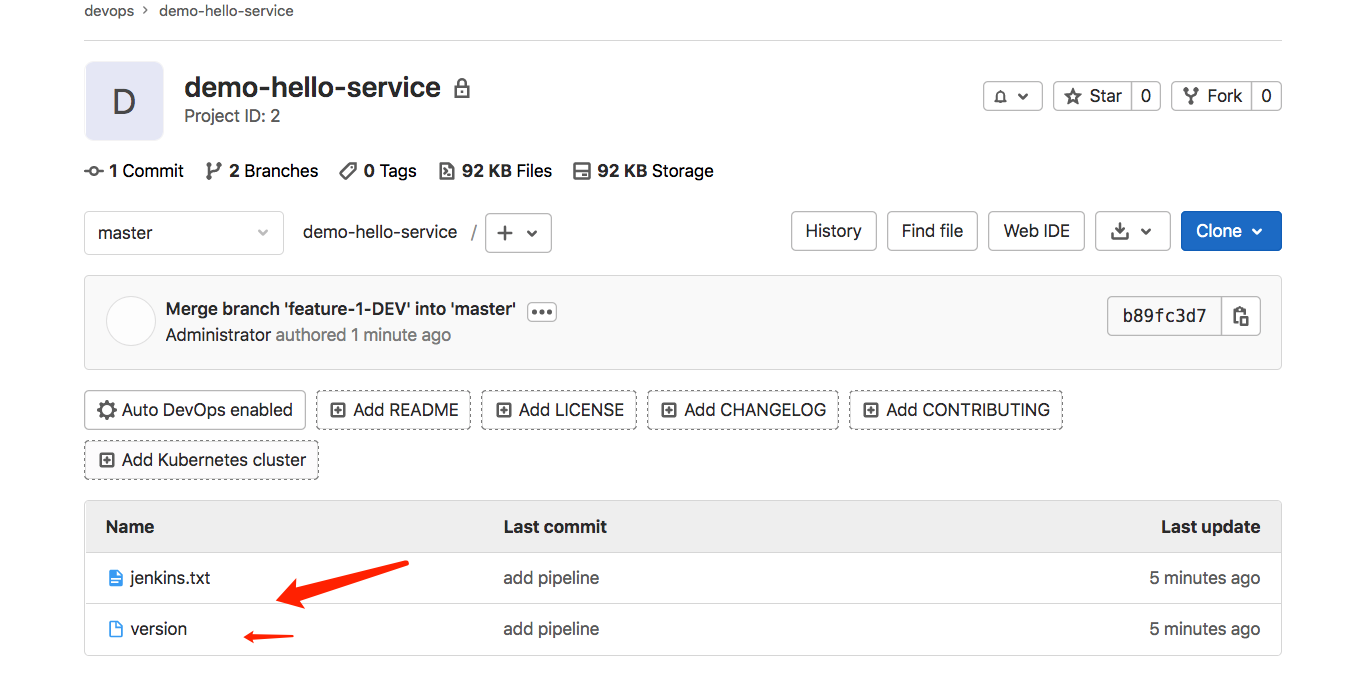
* 1 指定合并请求的标题
* 2 描述信息，一般都是变更信息
* 3 指定主管进行审核（最终该用户决定是否合并）
* 4 指定进行代码审查的同事
* 5 合并成功后删除源分支（最后很定要删除源分支，可以先保留一个版本后再删除，此处最好取消勾选）



提交合并后，由管理员审查进行合并。



合并后的效果： 特性分支的更改已经同步到了主干分支。

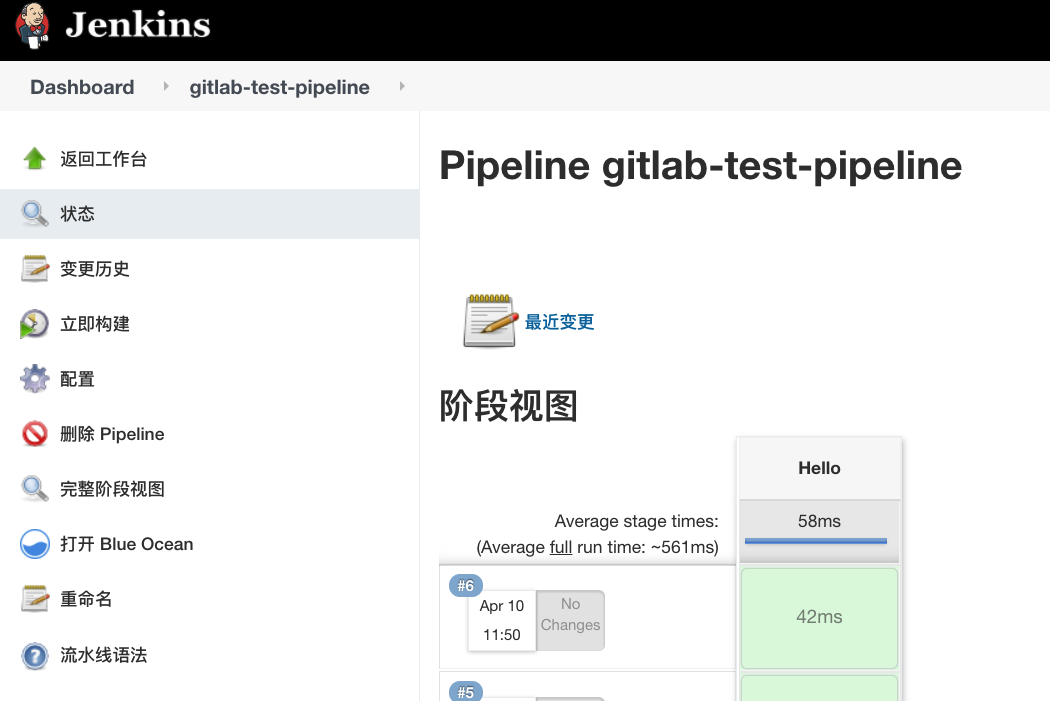


到此一个基本的项目开发提交代码过程就已经完成了。（多熟悉一下这个过程）

**3. GitLab触发器与提交流水线**

**Jenkins WebHook配置**

新建一个“gitlab-test-pipeline”项目， 开启Generic webhook。 配置触发token 为作业名称gitlab-test-pipeline。

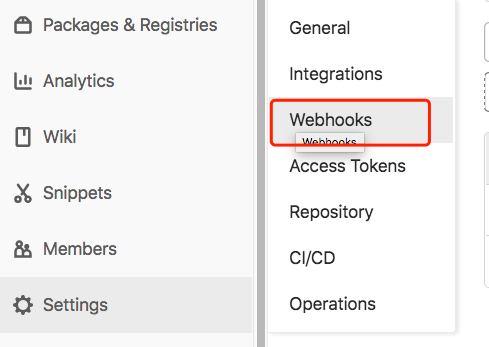


生成的触发URL

http://192.168.1.200:8080/generic-webhook-trigger/invoke?token=gitlab-test-pipeline

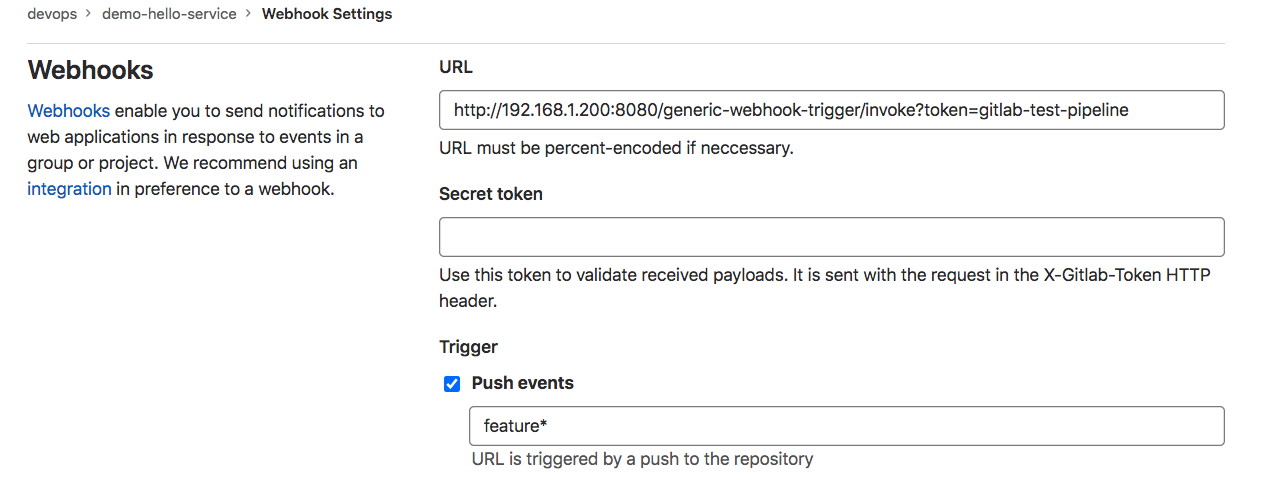
**GitLab WebHook 配置**

进入项目设置， gitlab webhook的配置页面：



**添加Jenkins 触发器地址，选择对应的事件。最后点击**https://cdn.nlark.com/yuque/0/2021/png/2584012/1618047708565-a64da695-d03f-4660-b290-afc9a4259df4.png

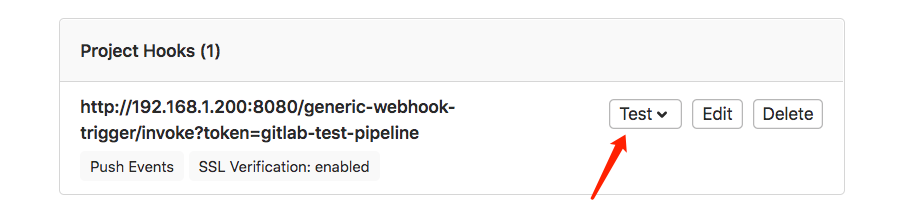
**这里选择的是代码提交事件， 过滤的分支是以feature开头的所有分支。**



事件：

* Push 提交事件
* Tag Push 创建事件
* MergeRequest 合并事件
* Issue 问题创建更新事件

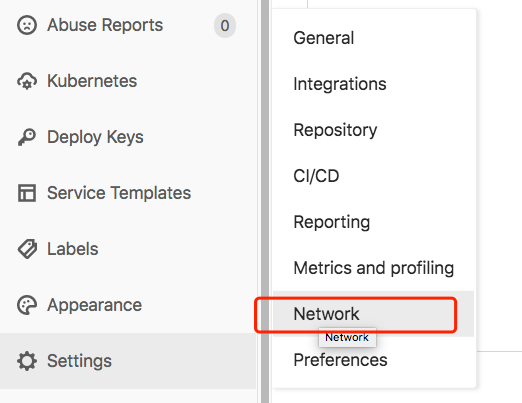
**模拟事件触发，点击test按钮选择push事件，此时去看下Jenkins是否成功被触发。**



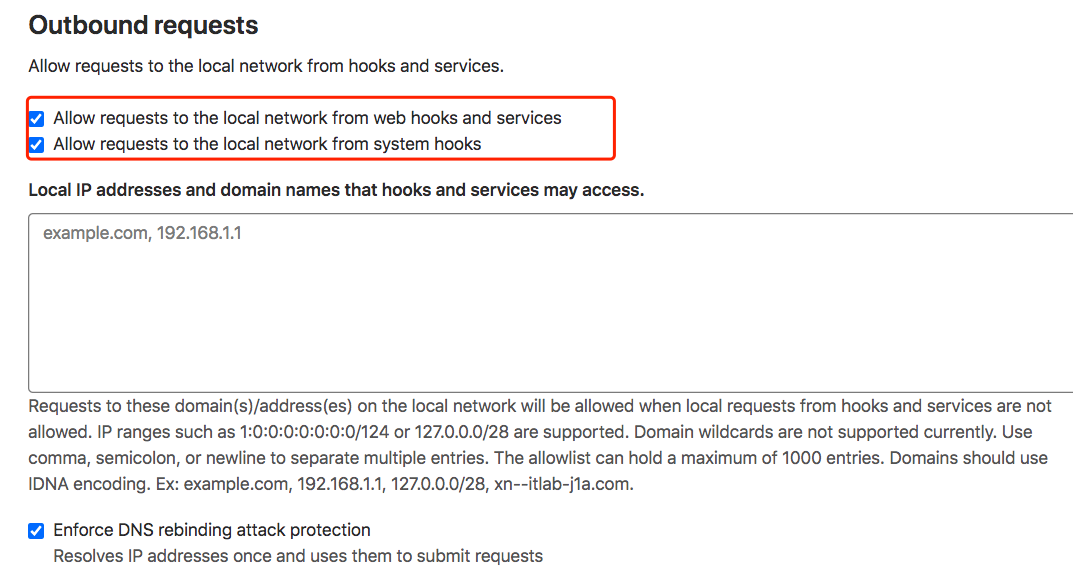
**触发成功提示**： Hook executed successfully: HTTP 200

**出现此FAQ：Url is blocked: Requests to the local network are not allowed**

解决方法：进入admin管理页面设置 > network

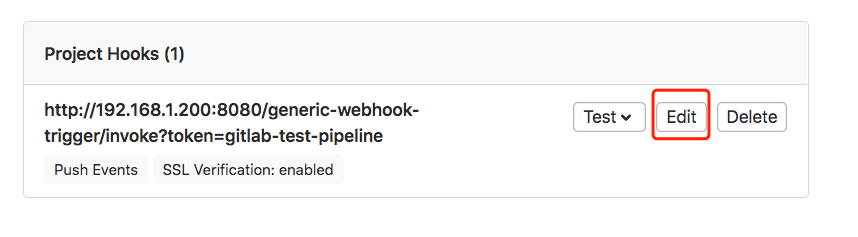


找到"Outbound requests"勾选允许请求webhooks和服务。（更改后，重启触发即可）

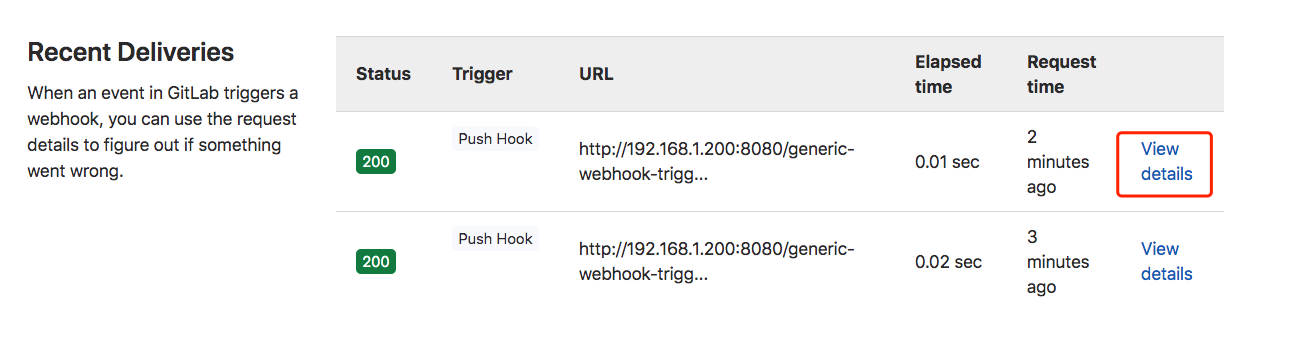


**Webhook问题排查调试**

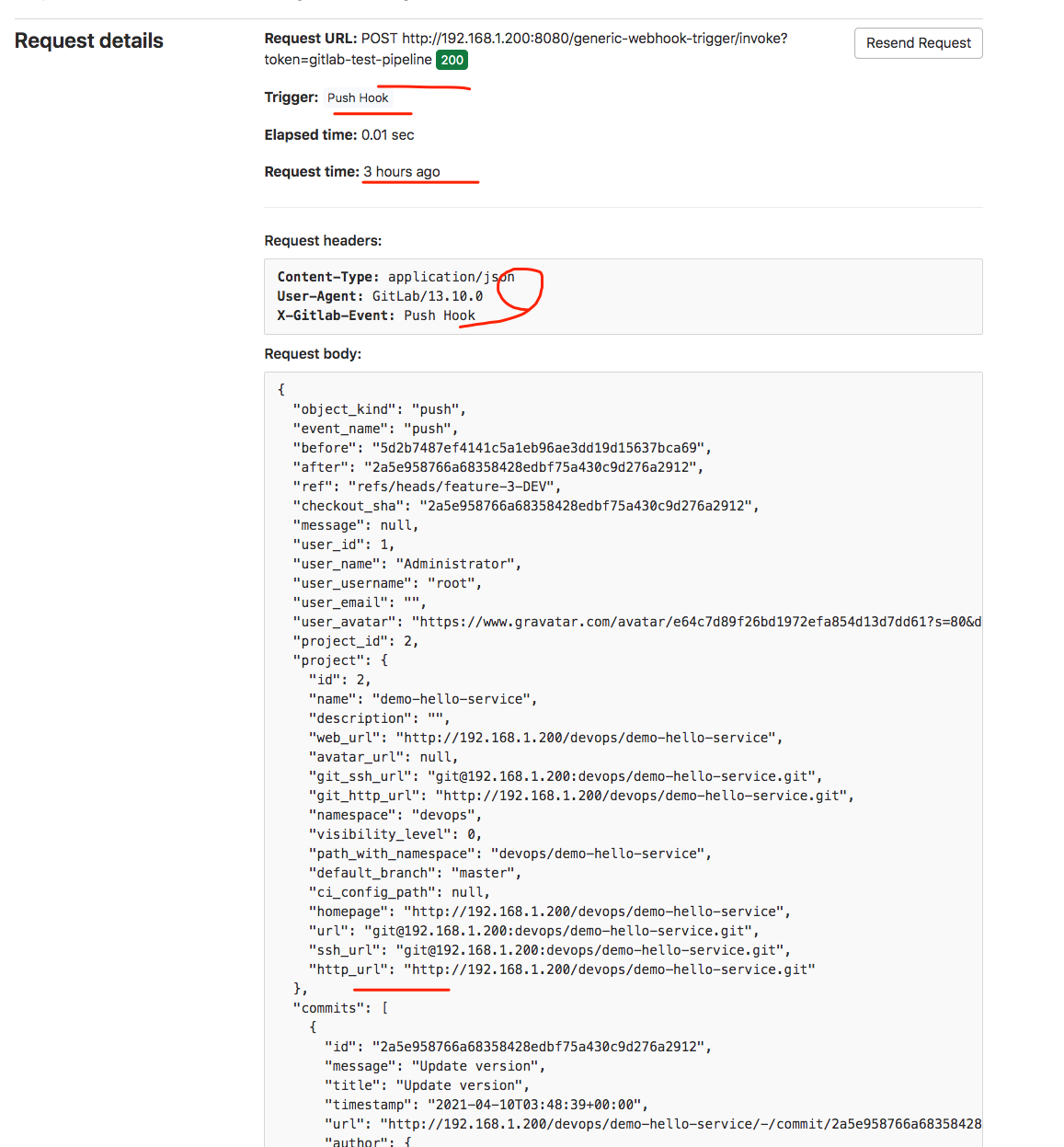
进入webhook添加页面的最下方，点击你所创建的webhook的 Edit按钮



可以查看到webhook的历史记录， 可以看到本次提交是否正常产生了webhook事件。



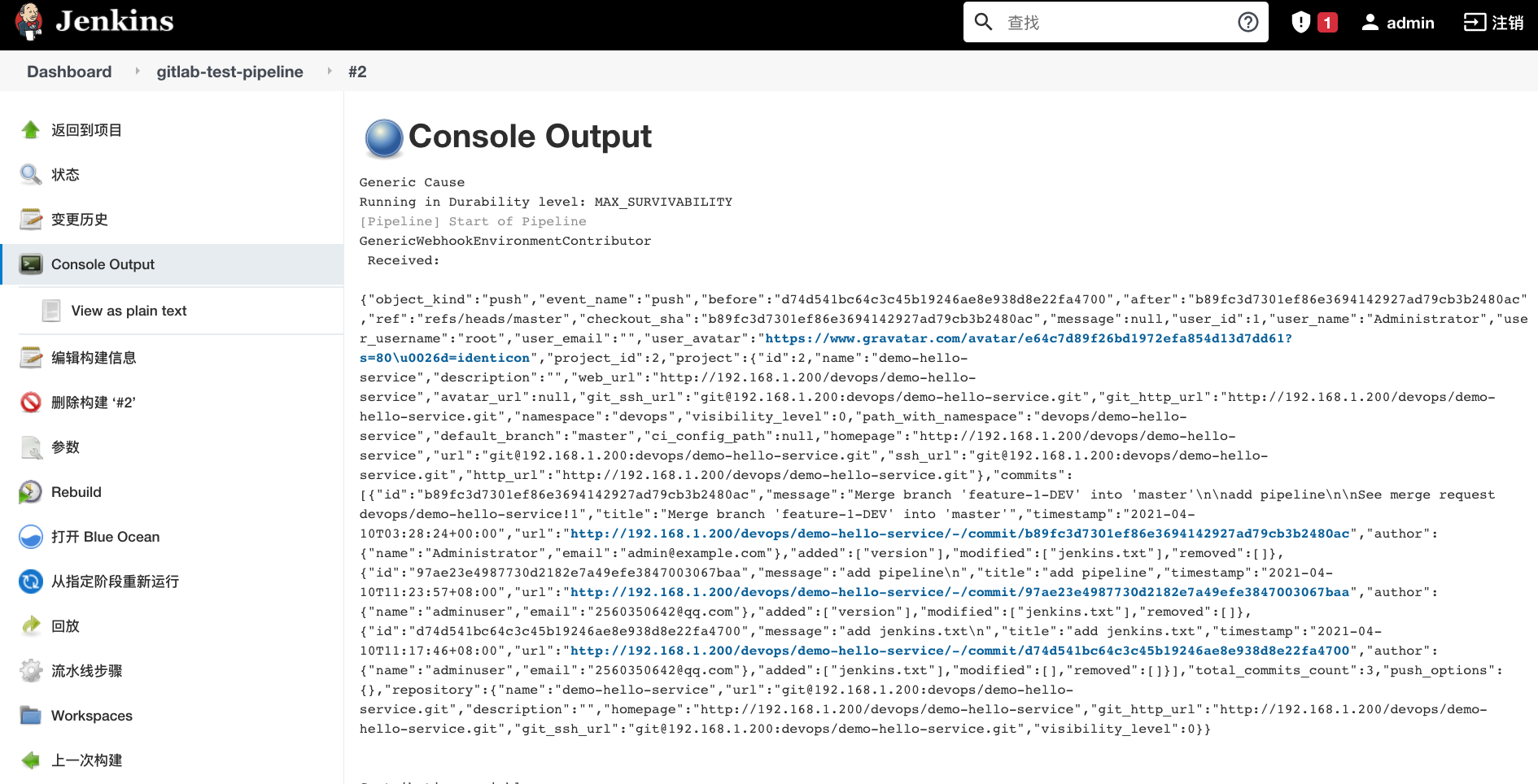
点击 View details 可以看到此webhook发送给对端Jenkins的数据信息，和请求状态。



如果此次请求由于xxx原因导致没有触发jenkins构建， 可以在这里点击 Resend Request 按钮进行重新发送请求，而不是再次提交代码。



如果Jenkins触发成功了之后，我们可以在Jenkins的构建日志中查看效果。 这些数据就是gitlab post传递过来的。



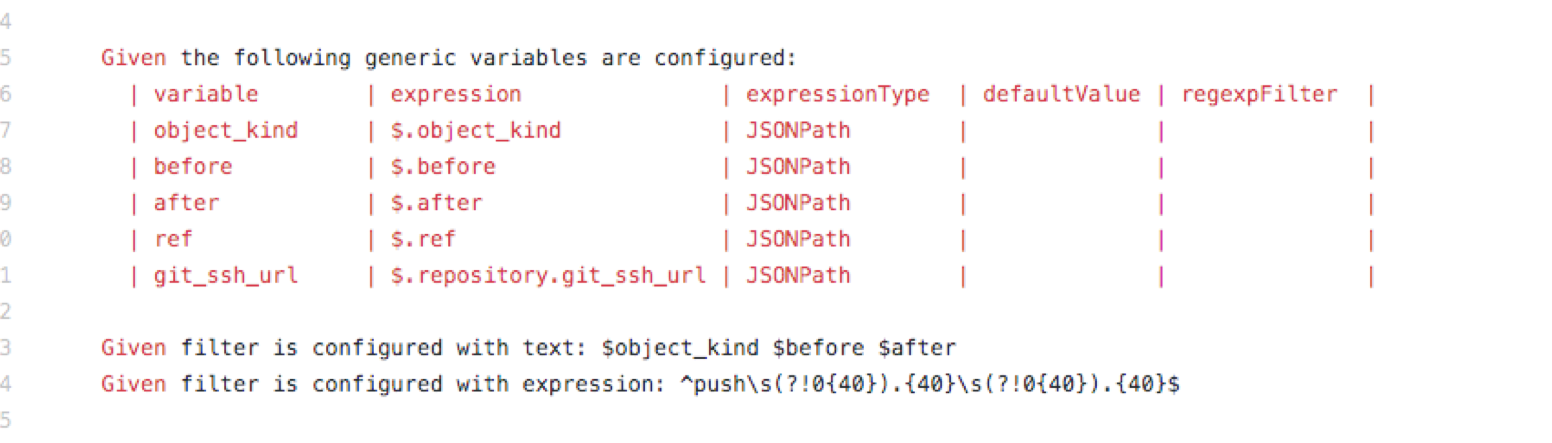
到此：你基本上已经知道了Gitlab如何触发Jenkins的了。（多看几遍，多练习几遍）

**4. GitLab提交流水线优化**

**过滤新建分支和tag的触发**

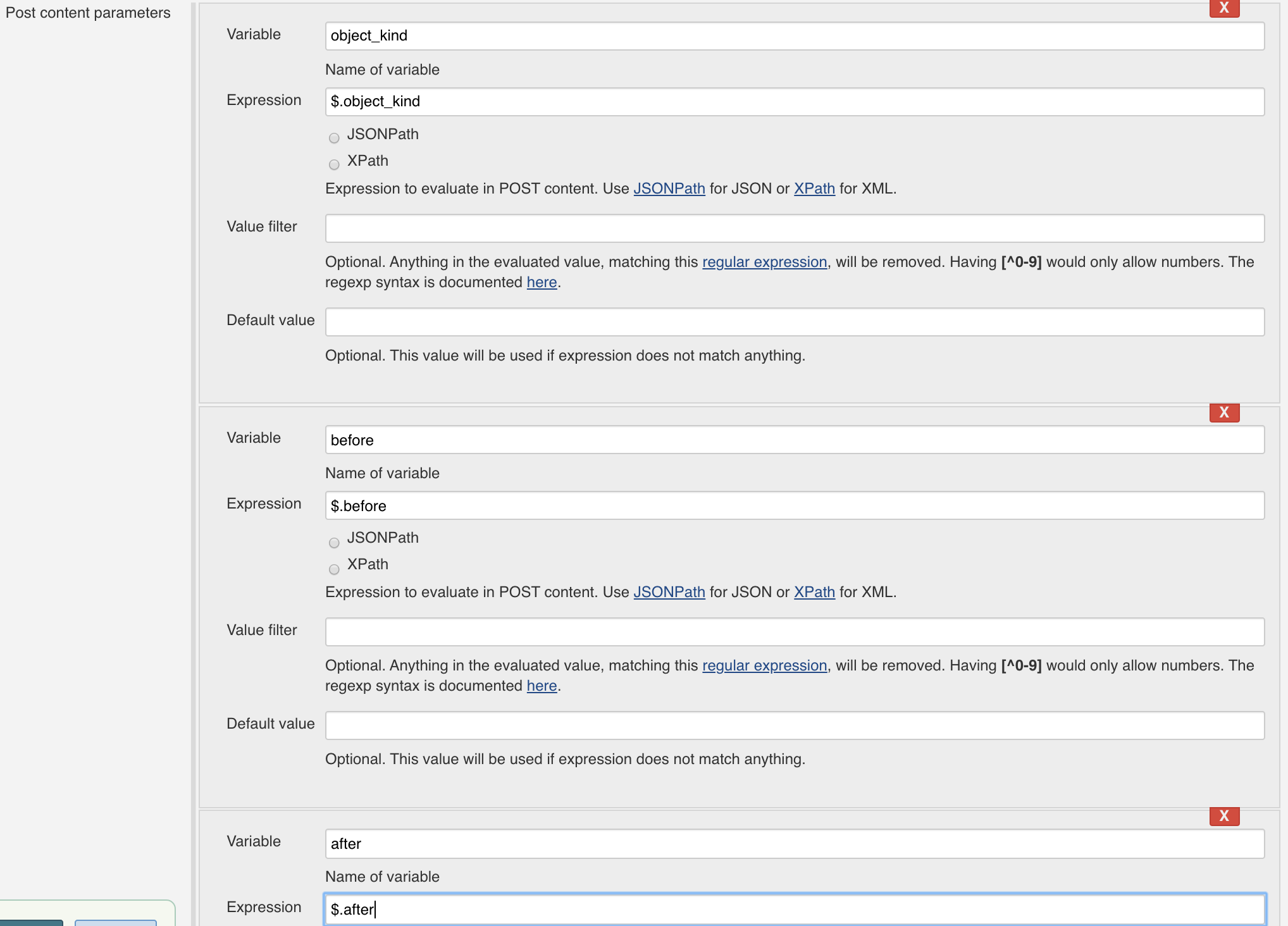
你可能发现问题了，**新建一个分支或者标签也会出现构建**，这个构建是没有意义的。我们需要排除掉。没错，jenkins 的 Generic webHook 也是支持的。

参考官方的说明:<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>

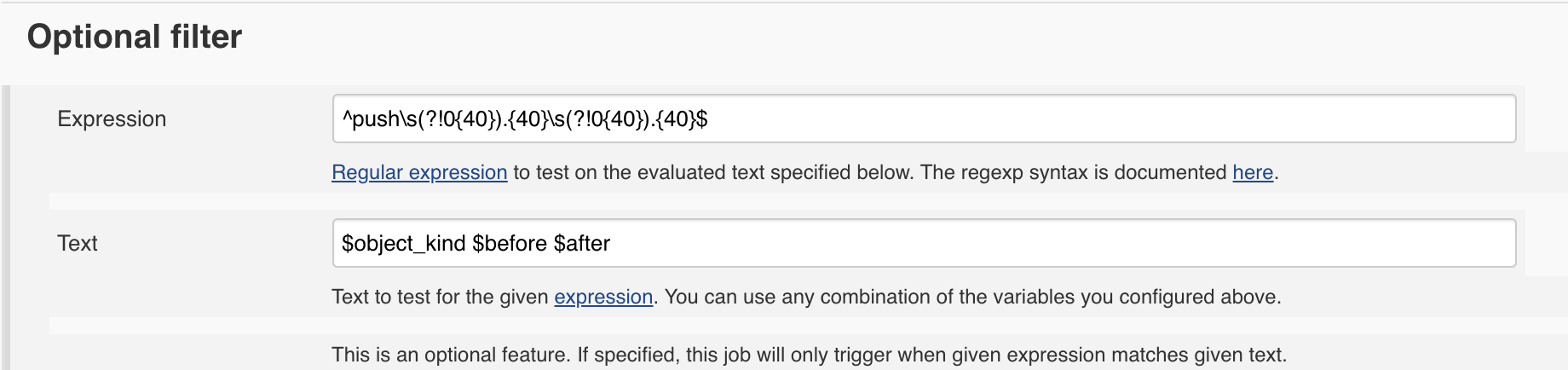


添加三个变量，获取当前的提交信息 $object\_kind $before $after

（**此步骤一定要注意下参数名和值后面的空格，要去掉**）

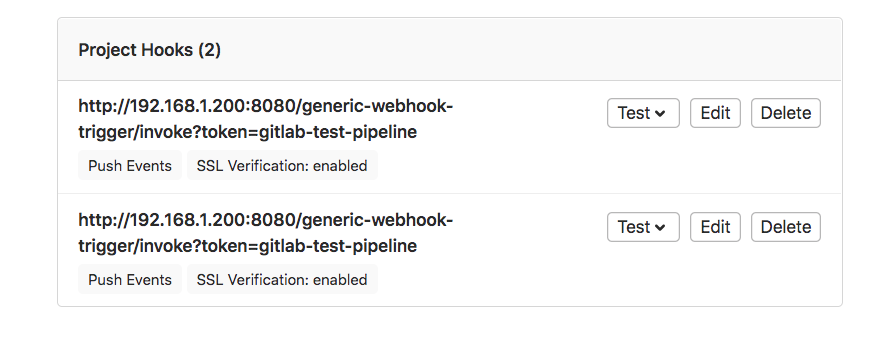


通过正则表达式配置触发条件：Expression ^push\s(?!0{40}).{40}\s(?!0{40}).{40}$ Text $object\_kind $before $after。 push请求只有after和before的值都不是40个0的时候触发构建（为40个0的情况是删除分支或者新建分支）



**如何支持多个分支触发构建？**

创建多个gitlab webhook 指向同一个jenkins 作业就可以了。

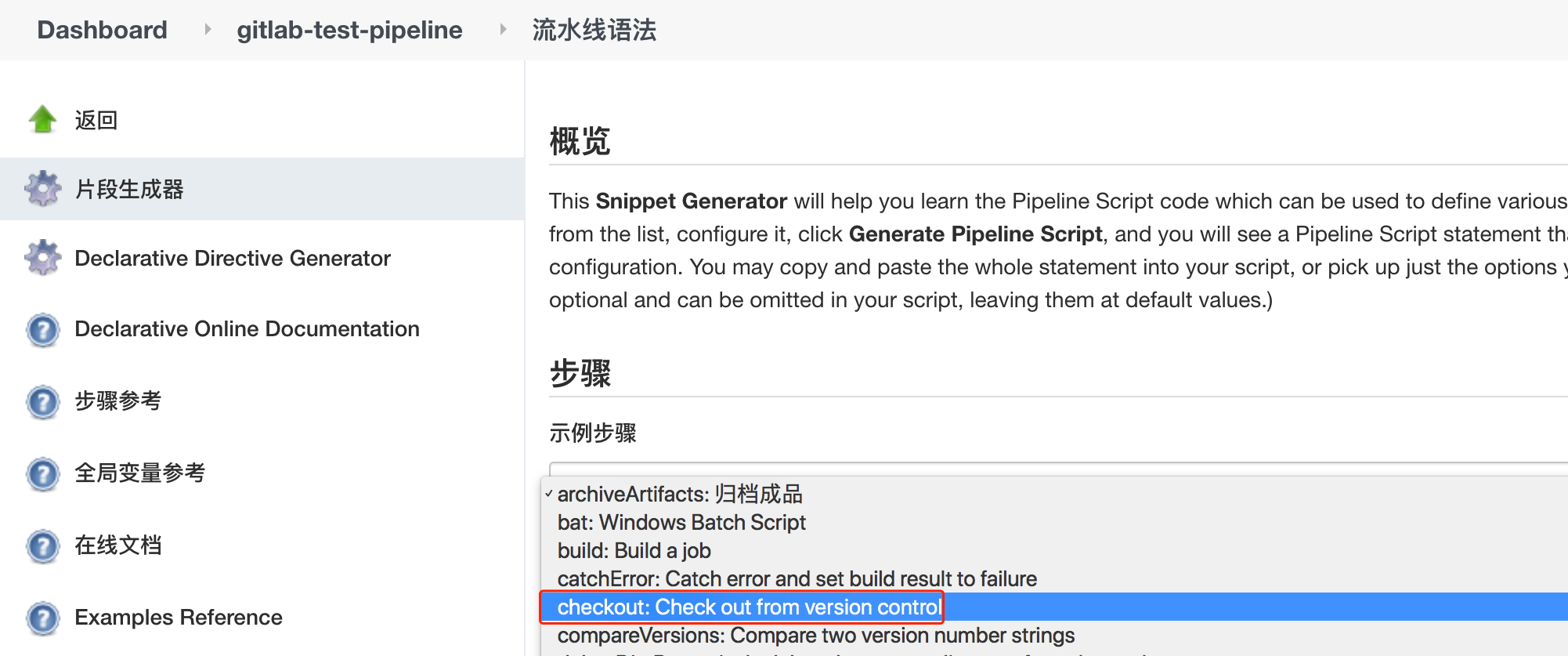


下午继续！

**5. GitLab & Jenkins集成**

**代码下载部分**

找一个pipeline 类型的项目进入流水线语法， 找到片段生成器中的 checkout 。 我们使用checkout方法来进行代码下载（svn也是支持的哦）

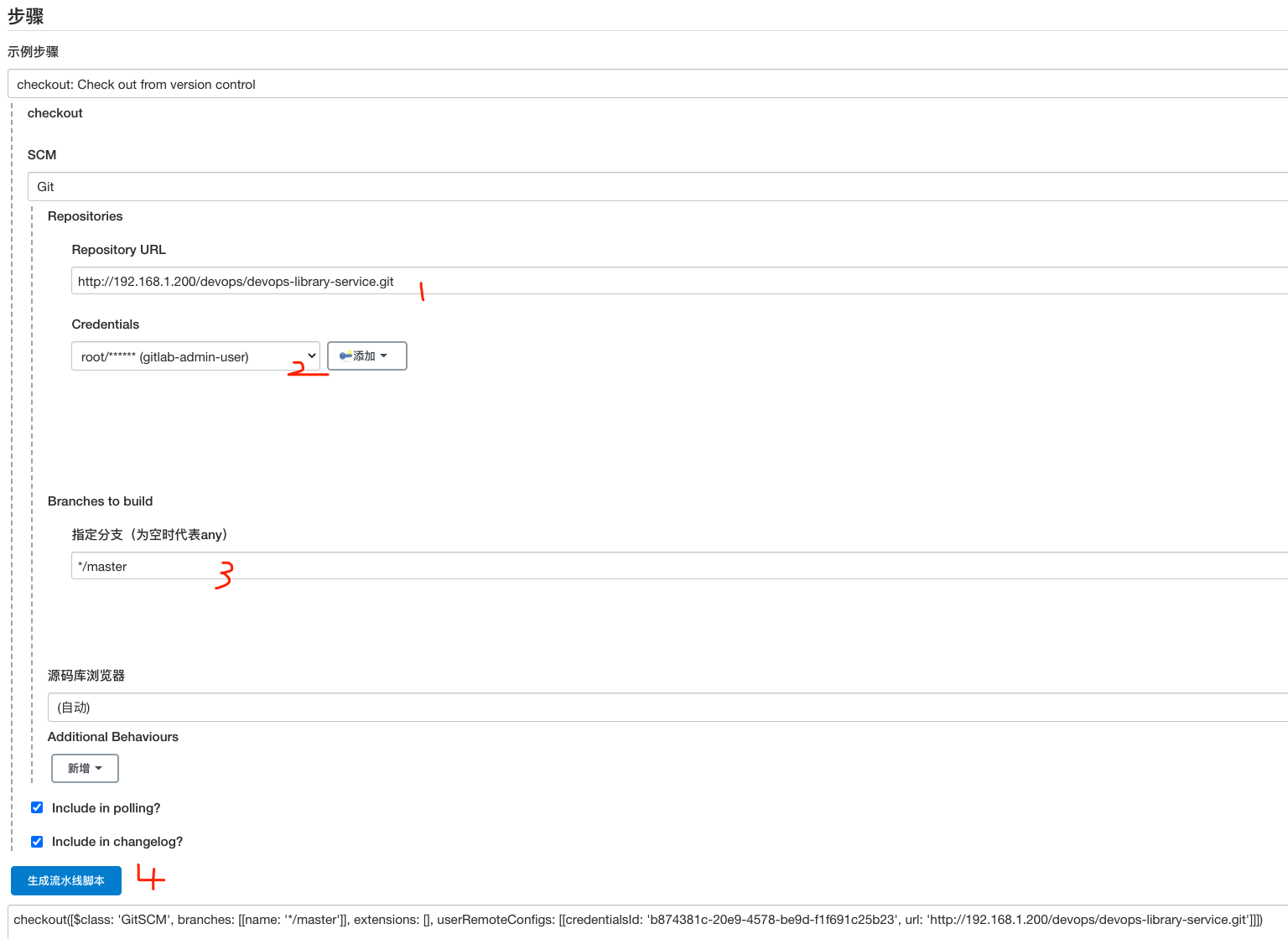


我们的代码库是私有类型的，需要一个具有clone 代码权限的用户账号来下载代码。（测试场景你可以将gitlab admin用户的账号存储在Jenkins的凭据中用于后期代码下载， 线上环境最好创建一个单独的reporter角色的用户进行代码下载，机器人账号。）



然后我们来生成代码片段吧，

* 1 填写项目仓库地址
* 2 指定访问项目仓库的账号
* 3 指定要下载的代码库分支
* 4 点击生成代码



一个基本的下载代码的demo示例：

stage("CheckOut"){

steps {

script {

checkout( [$class: 'GitSCM',

branches: [[name: "${branchName}"]],

extensions: [],

userRemoteConfigs: [[

credentialsId: 'bfe51cb6-9281-447d-86a7-adc06b964e48',

url: 'http://192.168.1.200/devops02/devops02-golang-service.git']]])

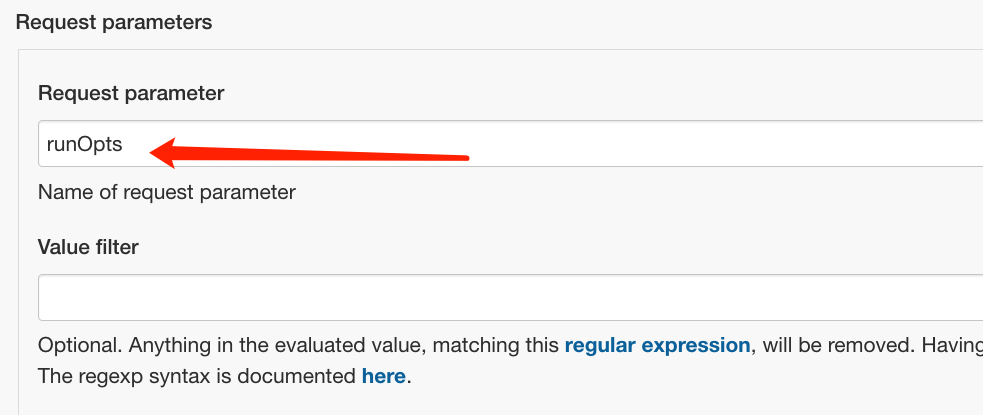
}

}

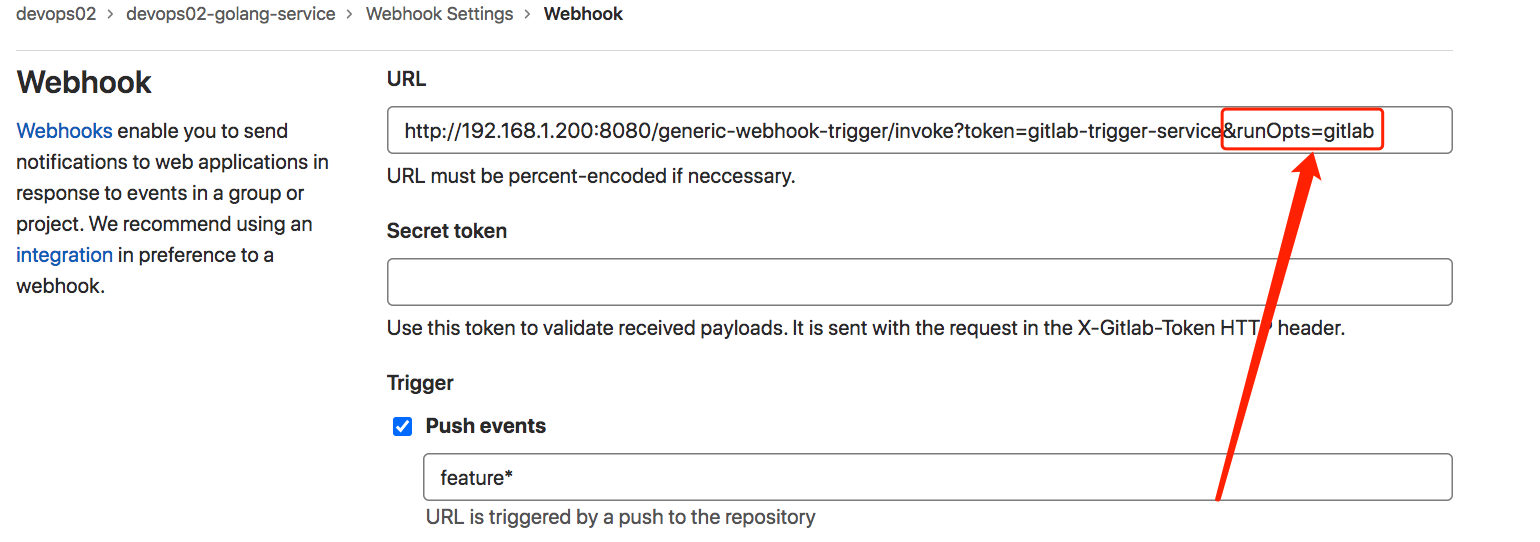
}

一个动态传递分支参数的示例： （这部分最好要看下视频）

首先，Jenkins触发器中配置一个request参数， runOpts作为构建类型判断。



第二步， Gitlab webhook中指定此参数。



第三步， 构建测试。 **完整代码参考实验代码： day3/gitlab02.jenkinsfile**

/\*

支持自动构建和手动构建：

1. 在触发添加了一个request参数 runOpts

2. Gitlab webhook中添加runOpts参数=gitlab

3. JenkinsPipeline 根据runOpts值进行运行

\*/

//gitlab触发

try {

if ( "${runOpts}" == "gitlab"){

//自动

// 操作webhook数据

println("${WebHookData}")

webHookData = readJSON text: "${WebHookData}"

env.userName = webHookData["user\_username"]

env.userEmail = webHookData["user\_email"]

env.branchName = webHookData["ref"] - "refs/heads/"

env.srcUrl = webHookData["project"]["git\_http\_url"]

env.commitID = webHookData["checkout\_sha"]

currentBuild.description = "Trigger by user ${env.userName} \n branch: ${env.branchName}"

currentBuild.displayName = env.commitID

}

} catch(e){

println(e)

env.userEmail = "2560350642@qq.com"

currentBuild.description ="Trigger by 手动触发 \n branch: ${env.branchName}"

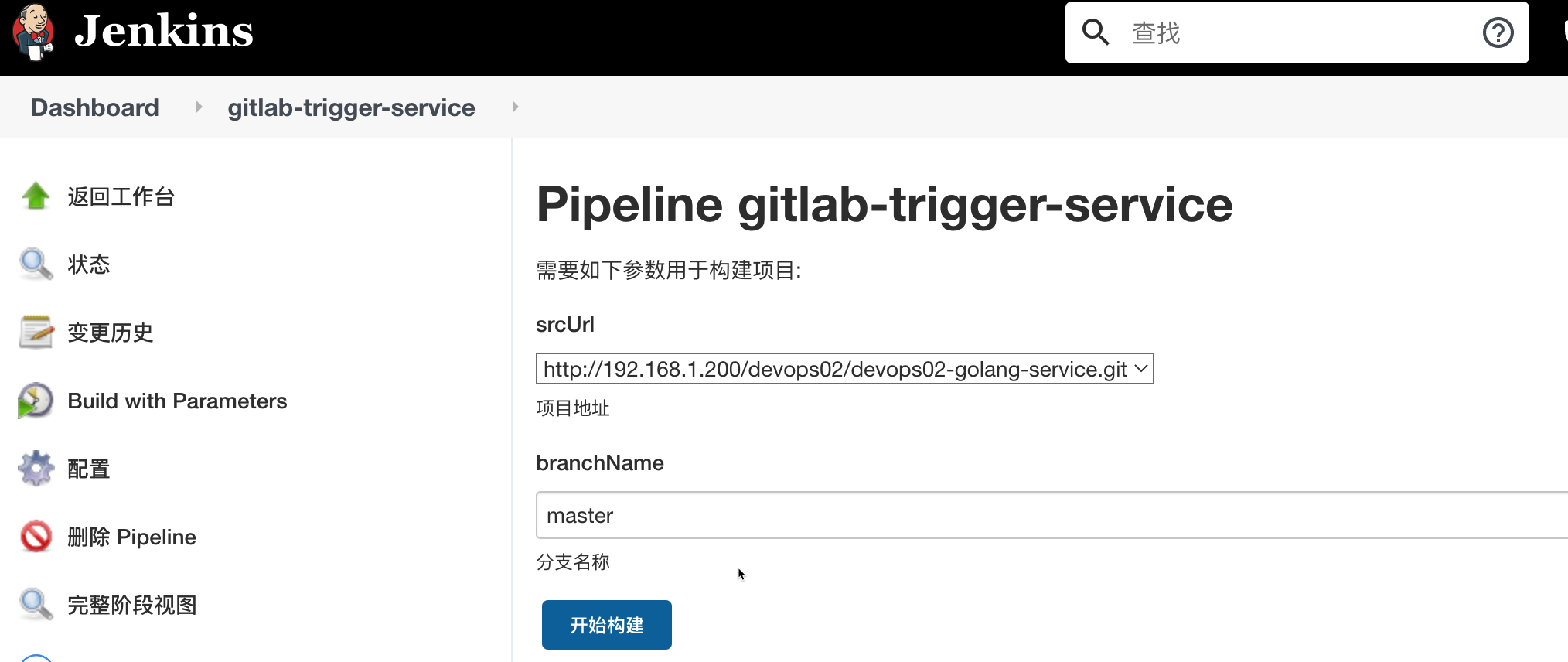
}

try 语句块中的是解析Gitlab传递过来的数据的，然后将 checkout 中的分支字符串和仓库地址使用变量替换。 branchName 和 srcUrl 都是解析的Gitlab hook数据。

如果想让这个作业同时支持gitlab自动触发和手动点击触发， ~~可以为项目添加 parameters 字段，生成UI参数。（注意这种将参数定义在Jenkinsfile的方式第一次运行需要初始化，第一次失败可以忽略，第二次以及后续可以正常使用的）~~

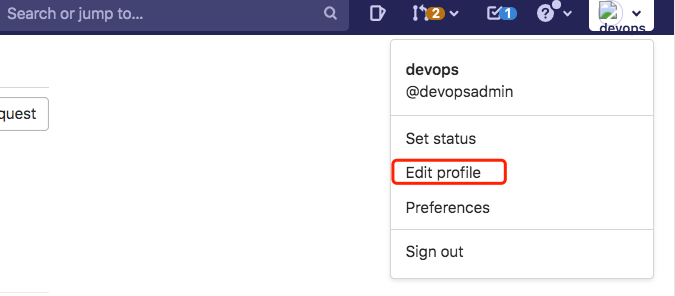
添加 try 语句块的目的也是为了忽略这些错误的， 因为手动触发是拿不到gitlab 传递的数据的，这是两种不同的触发方式所以一定要注意。 所以最后我们在UI定义两个参数作为手动触发使用的。

效果：

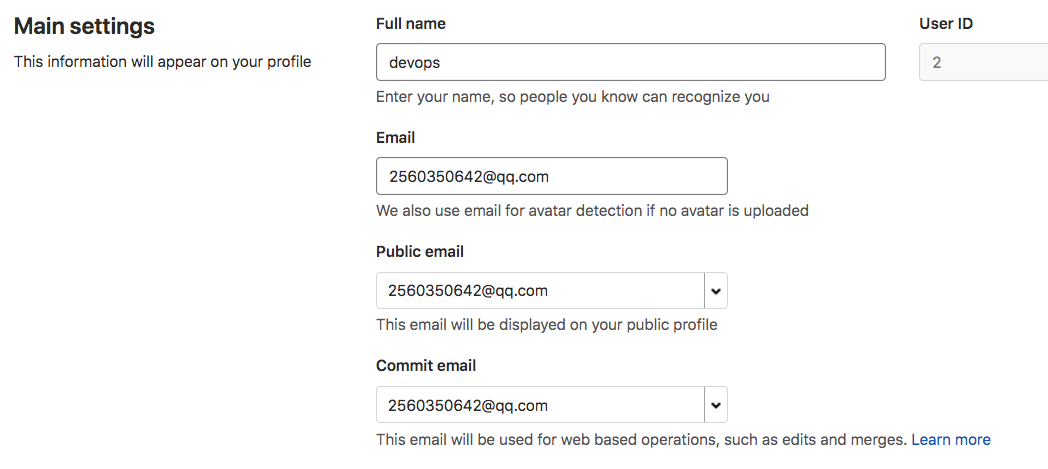


**邮件通知反馈**

**默认情况可能每个Gitlab用户没有配置邮箱的， 需要Gitlab用户要配置好邮箱。 点击头像进入 edit profile**



在这个页面配置好邮箱地址，最好这几个email都配置上吧......最后https://cdn.nlark.com/yuque/0/2021/png/2584012/1618049088742-7c281bcf-d29e-4629-b3a2-3760dd67095c.png



Jenkins解析Gitlab的POST参数来获取用户邮箱地址， 我们这里获取到然后使用 currentBuild 来增加描述信息。这样更加醒目直观的看到此次构建的主要信息。

此段代码没有加到Pipeline{}中， 直接写就可以运行的。

webHookData = readJSON text: "${WebHookData}"

userName = webHookData["user\_username"]

userEmail = webHookData["user\_email"]

branchName = webHookData["ref"] - "refs/heads/"

srcUrl = webHookData["project"]["git\_http\_url"]

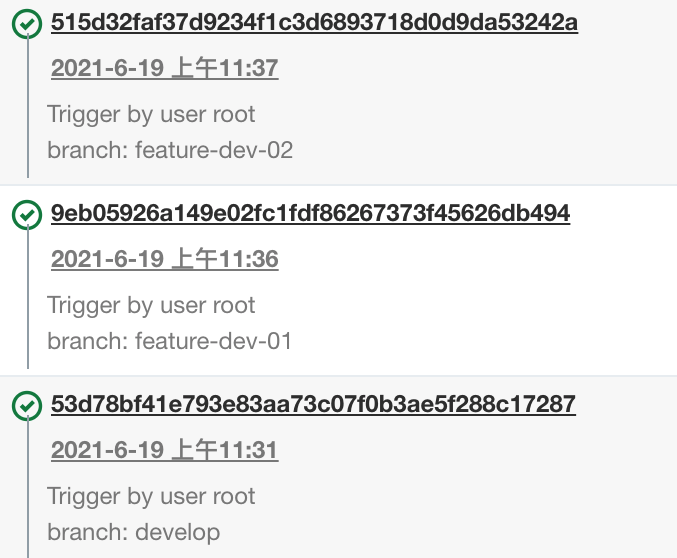
println("Trigger User: ${userName}, UserEmail: ${userEmail}")

println("Trigger Branch : ${branchName}")

currentBuild.description = "Trigger by user ${userName} \n branch: ${branchName}"

commitID = webHookData["checkout\_sha"]

currentBuild.displayName = commitID



Jenkins需要配置邮件通知，安装插件Email Extension**安装后重启Jenkins。**



然后进入系统管理-> 系统设置 ， 先配置下全局的admin的邮箱地址。（最后配置下不然可能会出错的）



->Extended E-email Notification。设置邮件系统配置信息。

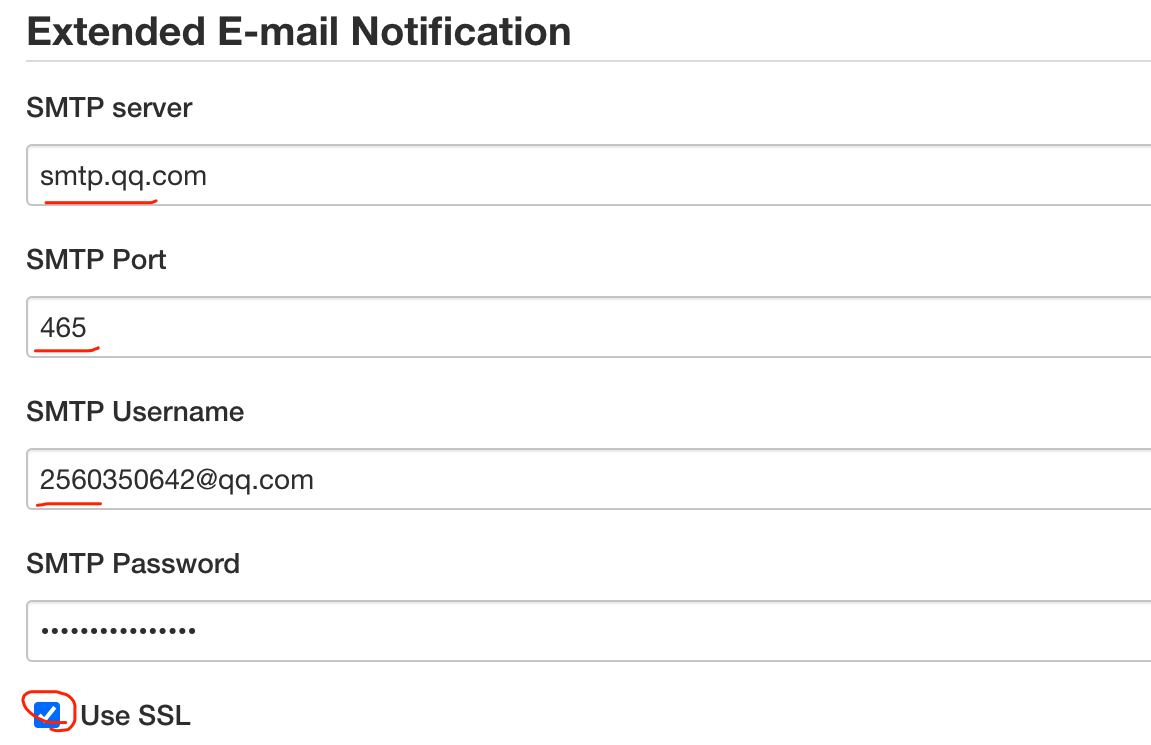
登入邮箱拿到授权码



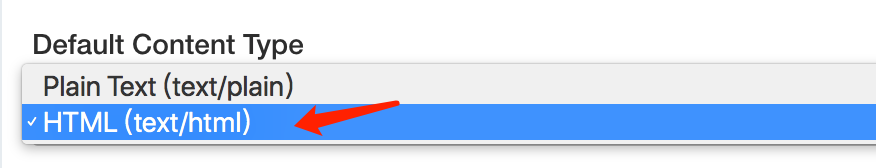
发送短信之后获取授权码



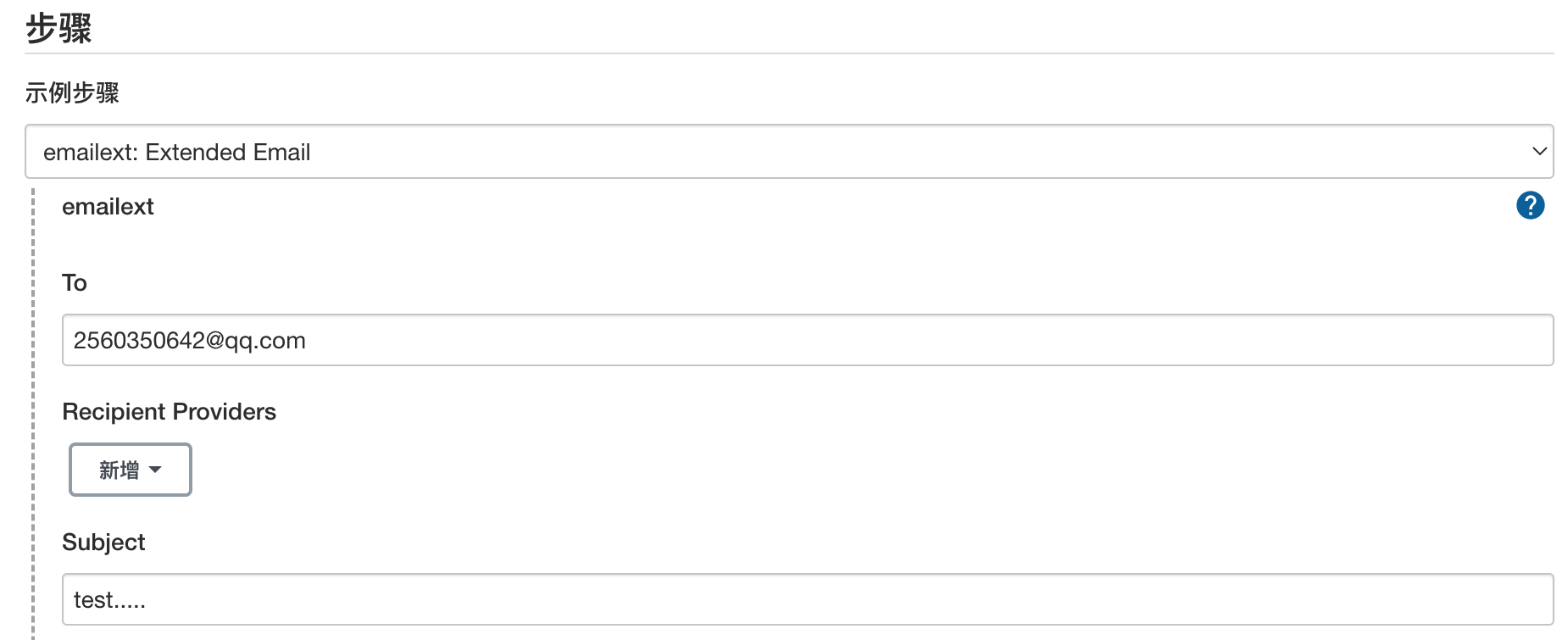
这里我使用的是QQ邮箱，填写SMTP服务器地址smtp.qq.com 和端口 465注意要开启SSL，密码为授权码。



换个选项注意下： 不选择 HTML 就是普通的文本， HTML 可以支持html网页，更加美观。这里选择 HTML



pipeline as code , 进入片段生成器，生成邮件通知代码。



emailext body: 'hello world!....jenkins', subject: 'test.....', to: '2560350642@qq.com'

然后我们将此段代码加入到Jenkins pipeline 中运行， 可以看到效果：



**jenkins as code 将email 写成一个函数。这个通知信息是一个html格式的。**

def EmailUser(userEmail,status){

emailext body: """

<!DOCTYPE html>

<html>

<head>

<meta charset="UTF-8">

</head>

<body leftmargin="8" marginwidth="0" topmargin="8" marginheight="4" offset="0">

<img src="http://192.168.1.200:8080/static/0eef74bf/images/headshot.png">

<table width="95%" cellpadding="0" cellspacing="0" style="font-size: 11pt; font-family: Tahoma, Arial, Helvetica, sans-serif">

<tr>

<td><br />

<b><font color="#0B610B">构建信息</font></b>

</td>

</tr>

<tr>

<td>

<ul>

<li>项目名称：${JOB\_NAME}</li>

<li>构建编号：${BUILD\_ID}</li>

<li>构建状态: ${status} </li>

<li>项目地址：<a href="${BUILD\_URL}">${BUILD\_URL}</a></li>

<li>构建日志：<a href="${BUILD\_URL}console">${BUILD\_URL}console</a></li>

</ul>

</td>

</tr>

<tr>

</table>

</body>

</html> """,

subject: "Jenkins-${JOB\_NAME}项目构建信息 ",

to: userEmail

}

最后收到的邮件效果：



**6. 常用的项目构建工具集成**

**此部分代码参考共享库中的src/org/devops/mytools.groovy**



**6.1 Java项目构建工具**

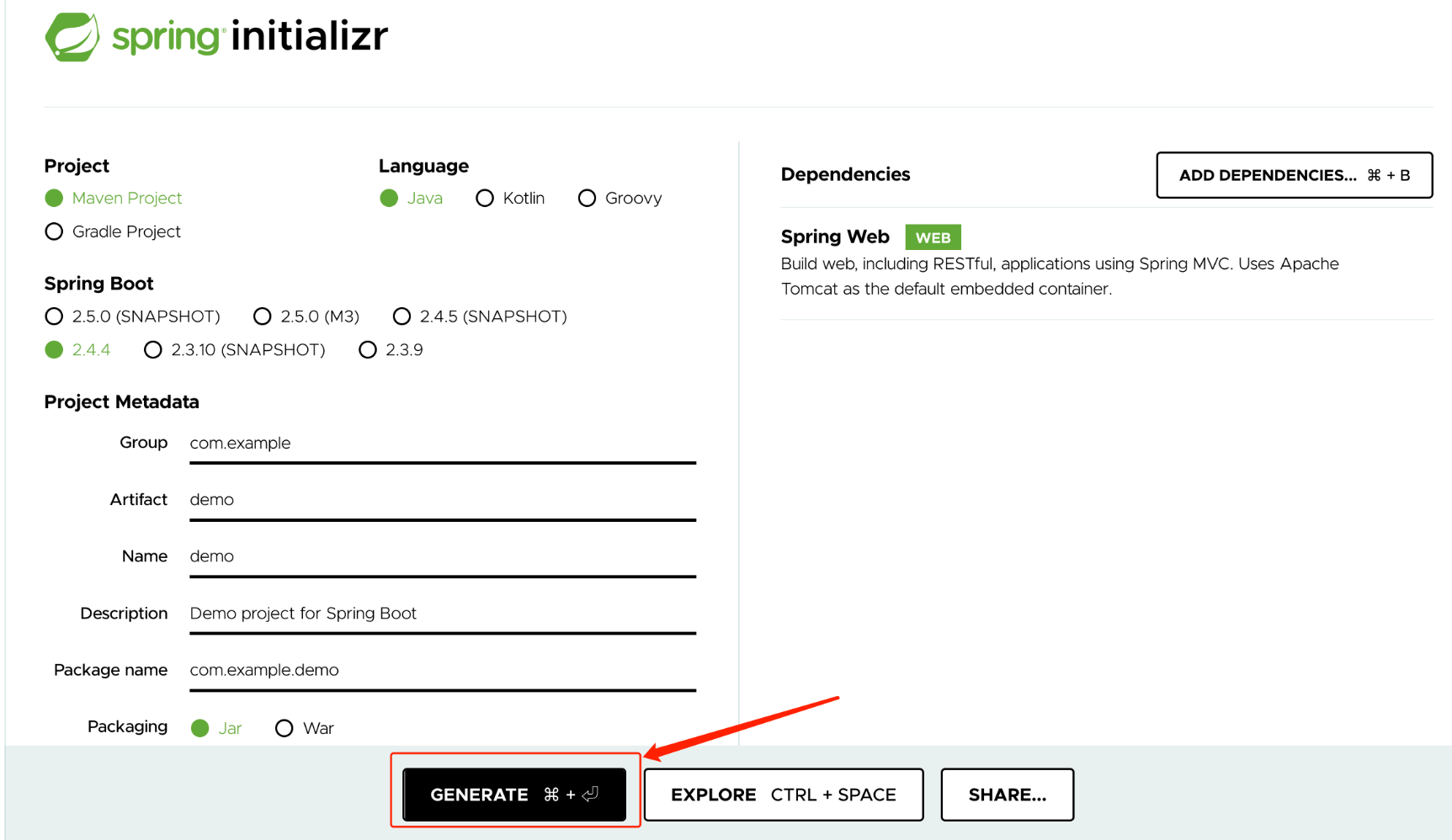
**maven项目实验代码：** [📎devops-maven-service-master.zip](https://www.yuque.com/attachments/yuque/0/2021/zip/2584012/1623935903916-ed5ec55f-09d7-4390-a10e-6143dea4297d.zip)

**gradle 项目实验代码：** [📎devops-gradle-service-master.zip](https://www.yuque.com/attachments/yuque/0/2021/zip/2584012/1623935904076-3bf00811-8c2d-459b-94fa-cb574491a367.zip)

**初始化一个springboot项目** <https://start.spring.io/>

**6.1.1 Maven**

官网：<http://maven.apache.org/download.cgi> Maven是一个项目的构建依赖管理工具。通常项目的根目录会存在一个**pom.xml**文件（该文件用于定义项目的依赖包信息和构建配置）



**安装配置**

### 下载

wget https://mirrors.bfsu.edu.cn/apache/maven/maven-3/3.8.1/binaries/apache-maven-3.8.1-bin.tar.gz

tar zxf apache-maven-3.8.1-bin.tar.gz -C /usr/local/

cd /usr/local/apache-maven-3.8.1/

pwd /usr/local/apache-maven-3.8.1

### 配置环境变量

vi /etc/profile

export M2\_HOME=/usr/local/apache-maven-3.8.1

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

source /etc/profile

### 验证

mvn -v

Apache Maven 3.8.1 (05c21c65bdfed0f71a2f2ada8b84da59348c4c5d)

Maven home: /usr/local/apache-maven-3.8.1

Java version: 1.8.0\_282, vendor: AdoptOpenJDK, runtime: /usr/local/jdk8u282-b08/jre

Default locale: en\_US, platform encoding: ANSI\_X3.4-1968

OS name: "linux", version: "4.18.0-80.el8.x86\_64", arch: "amd64", family: "unix"

将这个项目上传项目到Git仓库中。（提前创建一个 devops-maven-service 的仓库）

localhost:~ zeyang$ cd Desktop/demo

localhost:demo zeyang$ ls

HELP.md mvnw mvnw.cmd pom.xml src

localhost:demo zeyang$ git init

Initialized empty Git repository in /Users/zeyang/Desktop/demo/.git/

localhost:demo zeyang$ git remote add origin http://192.168.1.200/devops/devops-maven-service.git

localhost:demo zeyang$ git add .

localhost:demo zeyang$ git commit -m "Initial commit"

[master (root-commit) f907177] Initial commit

10 files changed, 712 insertions(+)

create mode 100644 .gitignore

create mode 100644 .mvn/wrapper/MavenWrapperDownloader.java

create mode 100644 .mvn/wrapper/maven-wrapper.jar

create mode 100644 .mvn/wrapper/maven-wrapper.properties

create mode 100755 mvnw

create mode 100644 mvnw.cmd

create mode 100644 pom.xml

create mode 100644 src/main/java/com/example/demo/DemoApplication.java

create mode 100644 src/main/resources/application.properties

create mode 100644 src/test/java/com/example/demo/DemoApplicationTests.java

localhost:demo zeyang$ git push -u origin master

Username for 'http://192.168.1.200': devopsadmin

Password for 'http://devopsadmin@192.168.1.200':

Enumerating objects: 26, done.

Counting objects: 100% (26/26), done.

Delta compression using up to 4 threads.

Compressing objects: 100% (16/16), done.

Writing objects: 100% (26/26), 52.35 KiB | 8.72 MiB/s, done.

Total 26 (delta 0), reused 0 (delta 0)

To http://192.168.1.200/devops/devops-maven-service.git

\* [new branch] master -> master

Branch 'master' set up to track remote branch 'master' from 'origin'.

**常用命令**

* + mvn clean 清理构建目录
  + mvn clean package 打包
  + mvn clean install 打包部署
  + mvn clean test 单元测试
  + mvn clean package -f ../pom.xml -f指定pom位置\
  + mvn clean package -DskipTests / -Dmaven.test.skip=true 跳过单测
  + mvn deploy 发布包到制品库

配置阿里maven源 、 本地仓库

**Jenkins集成**

stage("build"){

steps {

script{

sh "mvn clean package"

}

}

}

更加严谨的方式：使用绝对路径

def buildTools = ["maven": "/usr/local/apache-maven-3.8.1"]

pipeline {

agent { label "build" }

stages {

stage("GetCode"){

steps{

script{

println("下载代码 --> 分支： ${env.branchName}")

checkout([$class: 'GitSCM', branches: [[name: "${env.branchName}"]],

extensions: [],

userRemoteConfigs: [[credentialsId: 'b874381c-20e9-4578-be9d-f1f691c25b23',

url: "${env.gitHttpURL}"]]])

}

}

}

stage("Build"){

steps {

script {

//sh "/usr/local/apache-maven-3.8.1/bin/mvn clean package"

sh "${buildTools["maven"]}/bin/mvn clean package"

}

}

}

stage("UnitTest"){

steps{

script{

sh "${buildTools["maven"]}/bin/mvn test"

}

}

post {

success {

script{

junit 'target/surefire-reports/\*.xml'

}

}

}

}

}

post {

always {

script{

echo "always......"

}

}

success {

script {

echo "success....."

}

}

}

}

def EmailUser(userEmail,status){

emailext body: """

<!DOCTYPE html>

<html>

<head>

<meta charset="UTF-8">

</head>

<body leftmargin="8" marginwidth="0" topmargin="8" marginheight="4" offset="0">

<img src="http://192.168.1.200:8080/static/0eef74bf/images/headshot.png">

<table width="95%" cellpadding="0" cellspacing="0" style="font-size: 11pt; font-family: Tahoma, Arial, Helvetica, sans-serif">

<tr>

<td><br />

<b><font color="#0B610B">构建信息</font></b>

</td>

</tr>

<tr>

<td>

<ul>

<li>项目名称：${JOB\_NAME}</li>

<li>构建编号：${BUILD\_ID}</li>

<li>构建状态: ${status} </li>

<li>项目地址：<a href="${BUILD\_URL}">${BUILD\_URL}</a></li>

<li>构建日志：<a href="${BUILD\_URL}console">${BUILD\_URL}console</a></li>

</ul>

</td>

</tr>

<tr>

</table>

</body>

</html> """,

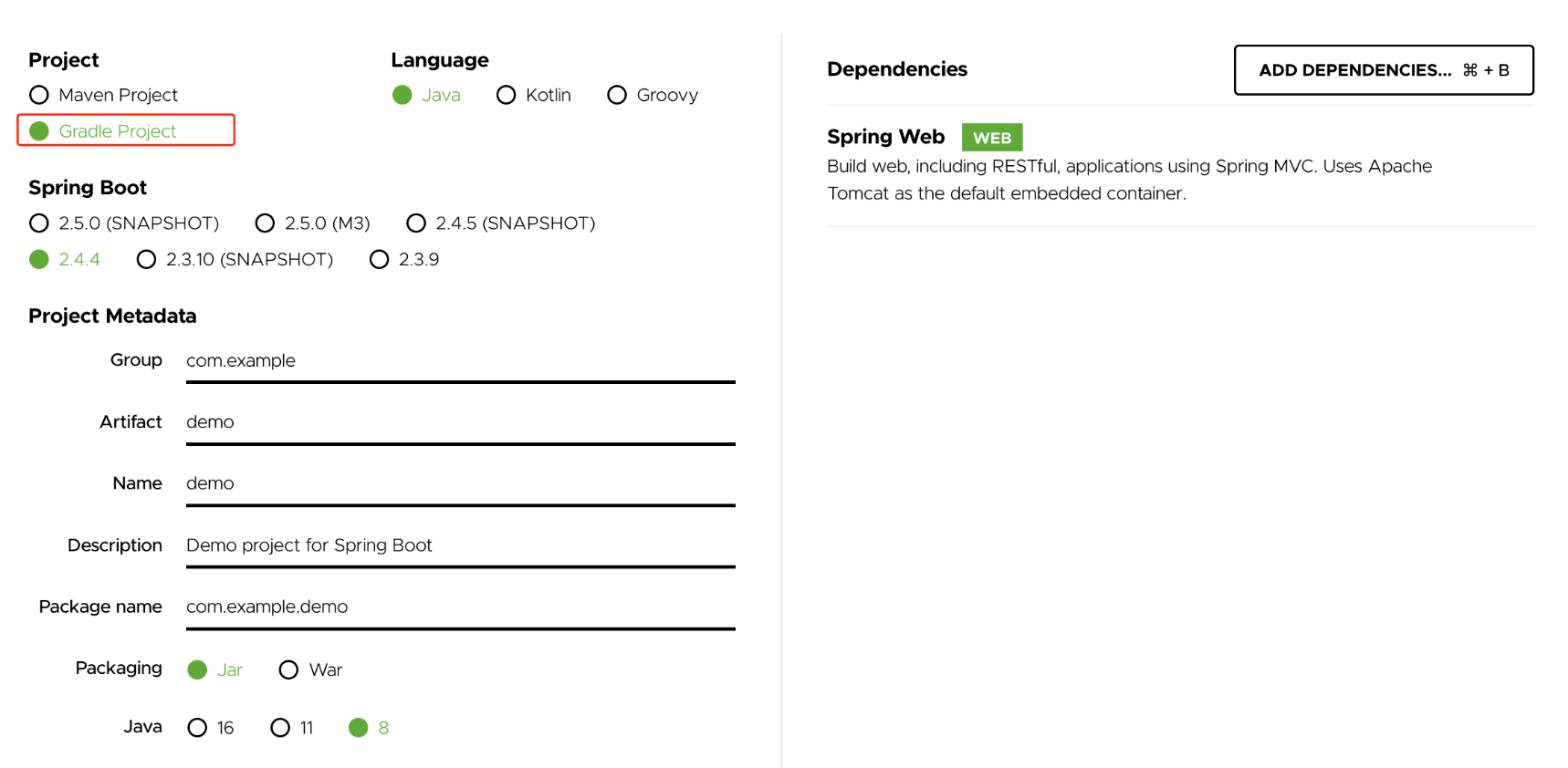
subject: "Jenkins-${JOB\_NAME}项目构建信息 ",

to: userEmail

}

**6.1.2 Gradle**

官网：<https://gradle.org/releases/> Gradle基于Groovy，具有更灵活更强大的构建系统，能帮助我们构建更复杂的项目。



上传git

localhost:~ zeyang$ cd Desktop/demo

localhost:demo zeyang$ ls

HELP.md build.gradle gradle gradlew gradlew.bat settings.gradle src

localhost:demo zeyang$ build.gradl

localhost:demo zeyang$ git init

Initialized empty Git repository in /Users/zeyang/Desktop/demo/.git/

localhost:demo zeyang$ git remote add origin http://192.168.1.200/devops/devops-gradle-service.git

localhost:demo zeyang$ git add .

localhost:demo zeyang$ git commit -m "Initial commit"

[master (root-commit) c27be12] Initial commit

10 files changed, 366 insertions(+)

create mode 100644 .gitignore

create mode 100644 build.gradle

create mode 100644 gradle/wrapper/gradle-wrapper.jar

create mode 100644 gradle/wrapper/gradle-wrapper.properties

create mode 100755 gradlew

create mode 100644 gradlew.bat

create mode 100644 settings.gradle

create mode 100644 src/main/java/com/example/demo/DemoApplication.java

create mode 100644 src/main/resources/application.properties

create mode 100644 src/test/java/com/example/demo/DemoApplicationTests.java

localhost:demo zeyang$ git push -u origin master

Username for 'http://192.168.1.200': devopsadmin

Password for 'http://devopsadmin@192.168.1.200':

Enumerating objects: 26, done.

Counting objects: 100% (26/26), done.

Delta compression using up to 4 threads.

Compressing objects: 100% (15/15), done.

Writing objects: 100% (26/26), 57.35 KiB | 8.19 MiB/s, done.

Total 26 (delta 0), reused 0 (delta 0)

To http://192.168.1.200/devops/devops-gradle-service.git

\* [new branch] master -> master

Branch 'master' set up to track remote branch 'master' from 'origin'.

localhost:demo zeyang$

**安装配置**

unzip gradle-6.8.3-bin.zip -d /usr/local/

export GRADLE\_HOME=/usr/local/gradle-6.8.3/

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

gradle -v

**常用命令**

* gradle build 构建项目
* gradle build -x test 构建项目跳过测试
* gradle clean 清空构建目录

**Jenkins集成**

stage("build"){

steps{

script {

sh "gradle clean && gradle build "

}

}

}

**6.1.3 Ant(扩展)**

官网：<https://ant.apache.org/bindownload.cgi> Ant现在用的比较少了，也是一个项目构建依赖管理工具，使用xml的文件保存配置（build.xml）

**安装配置**

wget

tar

export

export

source

ant -version

**常用命令**

* ant
* ant -f ../build.xml

**Jenkins集成**

stage("build"){

steps{

script {

sh "ant -f build.xml"

}

}

}

**6.2 Go项目构建工具**

<https://studygolang.com/dl>

**此处代码和Jenkinsfile参考这个代码库** [📎devops-golang-service-master.zip](https://www.yuque.com/attachments/yuque/0/2021/zip/2584012/1623935904225-5056e4c0-ff05-456b-b314-4dd42c8d8c84.zip)

**安装配置**

[root@zeyang-nuc-service cicd]# tar zxf go1.16.3.linux-amd64.tar.gz -C /usr/local/

[root@zeyang-nuc-service cicd]# cd /usr/local/go/

[root@zeyang-nuc-service go]# ls

AUTHORS CONTRIBUTORS PATENTS SECURITY.md api doc lib pkg src

CONTRIBUTING.md LICENSE README.md VERSION bin favicon.ico misc robots.txt test

[root@zeyang-nuc-service go]# pwd

/usr/local/go

[root@zeyang-nuc-service go]# vi /etc/profile

export GOROOT=/usr/local/go

export GOPATH=/opt/godir

export PATH=$GOROOT/bin:$PATH

[root@zeyang-nuc-service go]# source /etc/profile

[root@zeyang-nuc-service go]# go version

go version go1.16.3 linux/amd64

* GOROOT设置golang的安装位置
* GOBIN目录是执行 go install 后生成可执行文件的目录
* GOPATH 工作目录

**常用命令**

* go clean 清空构建
* go build 构建源文件
* go doc 生成godoc文档
* go install 编译并安装指定的代码包
* go fmt 代码格式化
* go get 获取一个包
* go run 运行一个go文件
* go test 运行测试

**6.3 前端项目构建工具**

**实验项目代码和Jenkinsfile**: [📎devops-web-service-master.zip](https://www.yuque.com/attachments/yuque/0/2021/zip/2584012/1623935904400-b6477709-bcce-492a-8c0c-4eda4ec9ee25.zip)

**6.3.1 npm**

官网：<https://nodejs.org/en/download/>

安装npm

wget https://nodejs.org/dist/v14.16.1/node-v14.16.1-linux-x64.tar.xz

tar xf node-v14.16.1-linux-x64.tar.xz -C /usr/local/

[root@zeyang-nuc-service node-v14.16.1-linux-x64]# vi /etc/profile

export NODE\_HOME=/usr/local/node-v14.16.1-linux-x64

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

[root@zeyang-nuc-service node-v14.16.1-linux-x64]# source /etc/profile

[root@zeyang-nuc-service node-v14.16.1-linux-x64]#

[root@zeyang-nuc-service node-v14.16.1-linux-x64]# node -v

v14.16.1

[root@zeyang-nuc-service node-v14.16.1-linux-x64]# npm -v

6.14.12

通过下面的命令，初始化一个vue项目

[root@zeyang-nuc-service node-v14.16.1-linux-x64]# npm install -g vue-cli

+ vue@2.6.12

added 1 package from 1 contributor in 3.342s

[root@zeyang-nuc-service node-v14.16.1-linux-x64]# vue-init webpack vuedemo

? Project name vuedemo

? Project description A Vue.js project

? Author adminuser <2560350642@qq.com>

? Vue build standalone

? Install vue-router? No

? Use ESLint to lint your code? No

? Set up unit tests No

? Setup e2e tests with Nightwatch? No

? Should we run `npm install` for you after the project has been created? (recommended) npm

vue-cli · Generated "vuedemo".

**常用命令**

* npm install <moduleName> -g 包安装到全局
* npm list：查看当前已安装的包。
* npm config set registry <https://registry.npm.taobao.org> 设置淘宝源
* npm config set cache "/opt/npmcache/" 设置缓存路径

**6.3.2 yarn**

yarn，facebook取代npm的包管理工具，速度快。Yarn 缓存包，无需重复下载。 并行下载，安装速度快。

**安装配置**

npm install -g yarn

yarn info

**常用命令**

* yarn / yarn install
* yarn clean
* yarn config set cache-folder "/opt/yarncache"

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

yarn config set sass\_binary\_site "https://npm.taobao.org/mirrors/node-sass/"

**7. 总结与共享库封装**

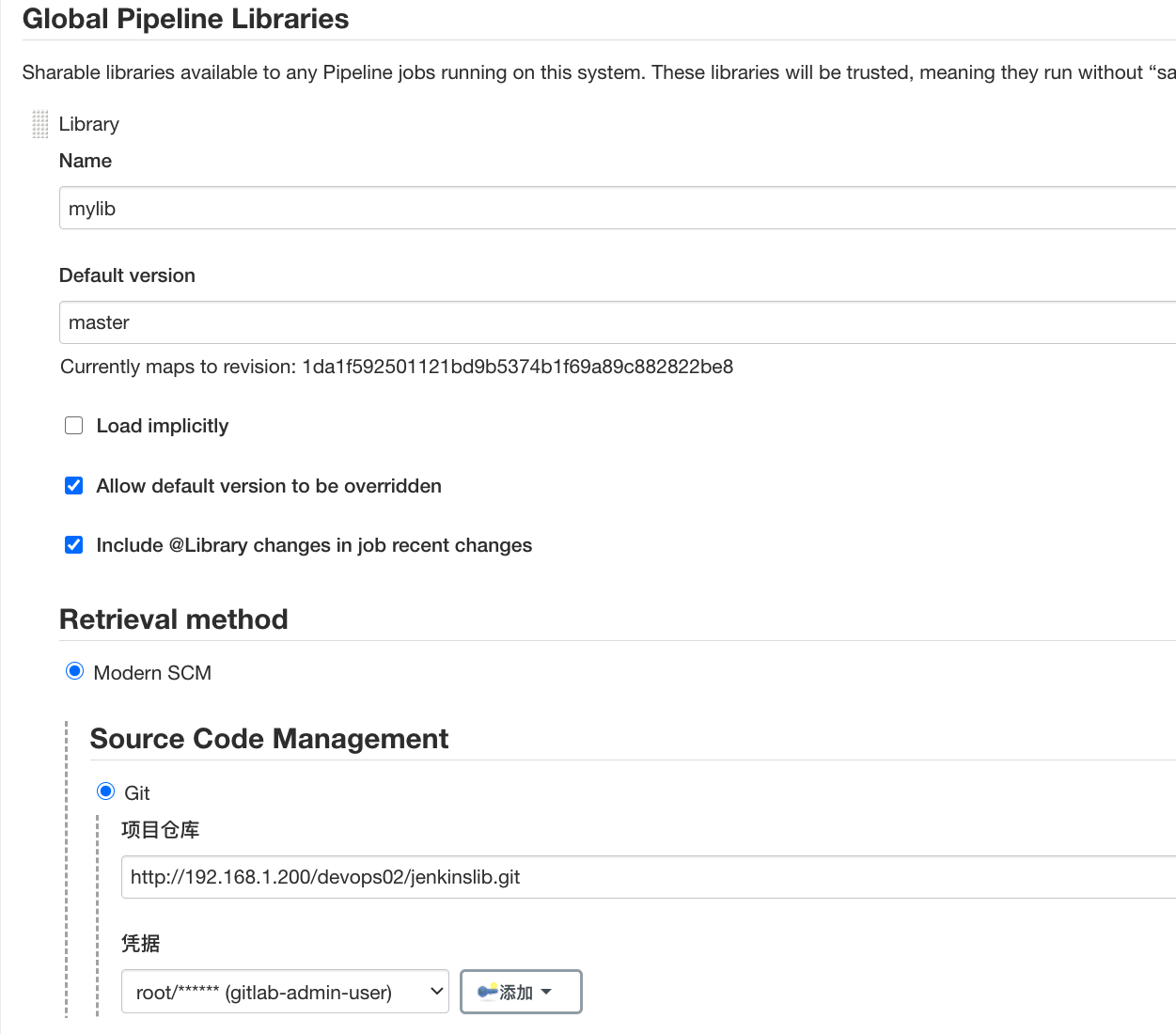
共享库实验代码：[📎devops-library-service-master.zip](https://www.yuque.com/attachments/yuque/0/2021/zip/2584012/1623935904594-a0fac03c-bca2-401c-9634-f64601bc2d79.zip)

**创建共享库**



**配置共享库**

进入Jenkins 设置， 定义共享库名称 mylib  默认的版本 master 。设置共享库的Git地址 `<http://192.168.1.200/devops02/jenkinslib.git>` , 设置共享库的凭据。（如果没有提前创建好凭据，需要先去创建好然后再重新配置这个页面。）



保存设置，共享库就配置好了。

**共享库内容分析**

mytools.groovy 存放构建相关的代码配置， 这里面定义了一个 CustomBuild 函数， 接受两个参数 buildMap 和 buildType ， 前者定义的是一个构建工具的位置map，后者是定义的构建类型例如 mvn/ant/gradle/web/golang 。

代码主体是根据不同的构建类型执行不同的构建命令，默认打印信息。

package org.devops

// 构建

def CustomBuild(buildType,buildMap ){

/\*if ("${JOB\_NAME}".contains('maven')){

sh """ ${buildMap["mvn"]}/bin/mvn clean package -DskipTests """

} else if ("${JOB\_NAME}".contains('gradle')){

sh """ ${buildMap["gradle"]}/bin/gradle clean && ${buildMap["gradle"]}/bin/gradle build """

}\*/

switch("${buildType}"){

case "mvn":

sh """ ${buildMap["${buildType}"]}/bin/mvn clean package -DskipTests """

break;

case "gradle":

sh """ ${buildMap["${buildType}"]}/bin/gradle clean && ${buildMap["${buildType}"]}/bin/gradle build -x test """

break;

case "golang":

sh """ ${buildMap["${buildType}"]}/bin/go build main.go """

break;

case "npm":

sh """ ${buildMap["${buildType}"]}/bin/npm install && ${buildMap["${buildType}"]}/bin/npm run build"""

break;

case "yarn":

sh """ ${buildMap["npm"]}/bin/yarn build """

break;

default:

println("buildTools choice error!")

break;

}

}

// 单元测试

def CustomTest(buildType,buildMap){

/\*if ("${JOB\_NAME}".contains('maven')){

sh """ ${buildMap["mvn"]}/bin/mvn test """

} else if ("${JOB\_NAME}".contains('gradle')){

sh """ ${buildMap["gradle"]}/bin/gradle test """

}\*/

switch("${buildType}"){

case "mvn":

sh """ ${buildMap["${buildType}"]}/bin/mvn test """

break;

case "gradle":

sh """ ${buildMap["${buildType}"]}/bin/gradle test """

break;

default:

println("buildTools choice error!")

break;

}

}

EmailUser 函数用于发送邮件通知， 接受两个参数：

* userEmail 邮件接收人（多个使用逗号分隔）
* status 作业状态（成功、失败）

// 邮件通知

def EmailUser(userEmail,status){

emailext body: """

<!DOCTYPE html>

<html>

<head>

<meta charset="UTF-8">

</head>

<body leftmargin="8" marginwidth="0" topmargin="8" marginheight="4" offset="0">

<img src="http://192.168.1.200:8080/static/0eef74bf/images/headshot.png">

<table width="95%" cellpadding="0" cellspacing="0" style="font-size: 11pt; font-family: Tahoma, Arial, Helvetica, sans-serif">

<tr>

<td><br />

<b><font color="#0B610B">构建信息</font></b>

</td>

</tr>

<tr>

<td>

<ul>

<li>项目名称：${JOB\_NAME}</li>

<li>构建编号：${BUILD\_ID}</li>

<li>构建状态: ${status} </li>

<li>项目地址：<a href="${BUILD\_URL}">${BUILD\_URL}</a></li>

<li>构建日志：<a href="${BUILD\_URL}console">${BUILD\_URL}console</a></li>

</ul>

</td>

</tr>

<tr>

</table>

</body>

</html> """,

subject: "Jenkins-${JOB\_NAME}项目构建信息 ",

to: userEmail

}

Jenkinsfile

// 使用共享库

@Library("mylib@master") \_

// 导入类方法

def mytools = new org.devops.mytools()

//定义构建工具

buildMap = ["mvn" : "/usr/local/apache-maven-3.8.1",

"gradle": "/usr/local/gradle-6.8.3",

"golang": "/usr/local/go",

"npm" : "/usr/local/node-v14.16.1-linux-x64"]

// 项目使用的构建工具类型

buildType = "${env.buildType}"

//gitlab触发

try {

if ( "${runOpts}" == "gitlab"){

//自动

// 操作webhook数据

println("${WebHookData}")

webHookData = readJSON text: "${WebHookData}"

env.userName = webHookData["user\_username"]

env.userEmail = webHookData["user\_email"]

env.branchName = webHookData["ref"] - "refs/heads/"

env.srcUrl = webHookData["project"]["git\_http\_url"]

env.commitID = webHookData["checkout\_sha"]

currentBuild.description = "Trigger by user ${env.userName} \n branch: ${env.branchName}"

currentBuild.displayName = env.commitID

}

} catch(e){

println(e)

env.userEmail = "2560350642@qq.com"

currentBuild.description ="Trigger by admin \n branch: ${env.branchName}"

}

pipeline {

agent {

label "build"

}

options{

skipDefaultCheckout true

}

stages{

// 下载代码

stage("CheckOut"){

steps {

script {

checkout( [$class: 'GitSCM',

branches: [[name: "${env.branchName}"]],

extensions: [],

userRemoteConfigs: [[

credentialsId: 'bfe51cb6-9281-447d-86a7-adc06b964e48',

url: "${env.srcUrl}"]]])

}

}

}

// 构建代码

stage("Build"){

steps{

script{

mytools.CustomBuild(buildType,buildMap)

}

}

}

//单元测试

stage("UnitTest"){

steps{

script {

mytools.CustomTest(buildType,buildMap)

}

}

post {

success {

script{

switch("${buildType}"){

case "mvn":

junit 'target/surefire-reports/\*.xml'

break;

case "gradle":

junit 'build/test-results/test/\*.xml'

break;

default:

println("buildTools choice error!")

break;

}

}

}

}

}

}

post {

always{

script {

mytools.EmailUser("${env.userEmail}","success")

}

}

failure{

script{

mytools.EmailUser("${env.userEmail}","failure")

}

}

}

}

**Jenkins 作业配置**

[📎newbuild-library-service.tar.gz](https://www.yuque.com/attachments/yuque/0/2021/gz/2584012/1623935904739-3333ab11-044f-4b6e-a7c2-74bb82ac542f.gz)**这是课程中作业的配置， 下载后解压到Jenkinshome的jobs目录中，然后重启Jenkins就可以加载了。**



定义一个选项参数： 存放gitlab仓库地址.选择构建工具类型

