第0章 Jenkins介绍

```
1.jenkins是一个开源的持续集成工具，由java语言开发
2.jenkins是一个调度平台，拥有众多的插件，绝大部分功能都是由插件来完成的
```

第1章 Jenkins安装

1.官方网站

```
https://www.jenkins.io/zh/
https://www.jenkins.io/zh/doc/
```

2.安装部署

清华源直接下载rpm包安装即可,下载地址如下：

```
https://mirrors.tuna.tsinghua.edu.cn/jenkins/redhat/
```

安装命令如下：

```
rpm -ivh jdk-8u181-linux-x64.rpm
rpm -ivh jenkins-2.176.1-1.1.noarch.rpm
```

3.目录文件说明

```
[root@jenkins ~]# rpm -ql jenkins
/etc/init.d/jenkins                #启动文件
/etc/logrotate.d/jenkins           #日志切割脚本
/etc/sysconfig/jenkins             #配置文件
/usr/lib/jenkins                   #安装目录
/usr/lib/jenkins/jenkins.war       #安装包
/usr/sbin/rcjenkins
/var/cache/jenkins
/var/lib/jenkins                   #数据目录
/var/log/jenkins                   #日志目录
```

4.配置使用root账户运行

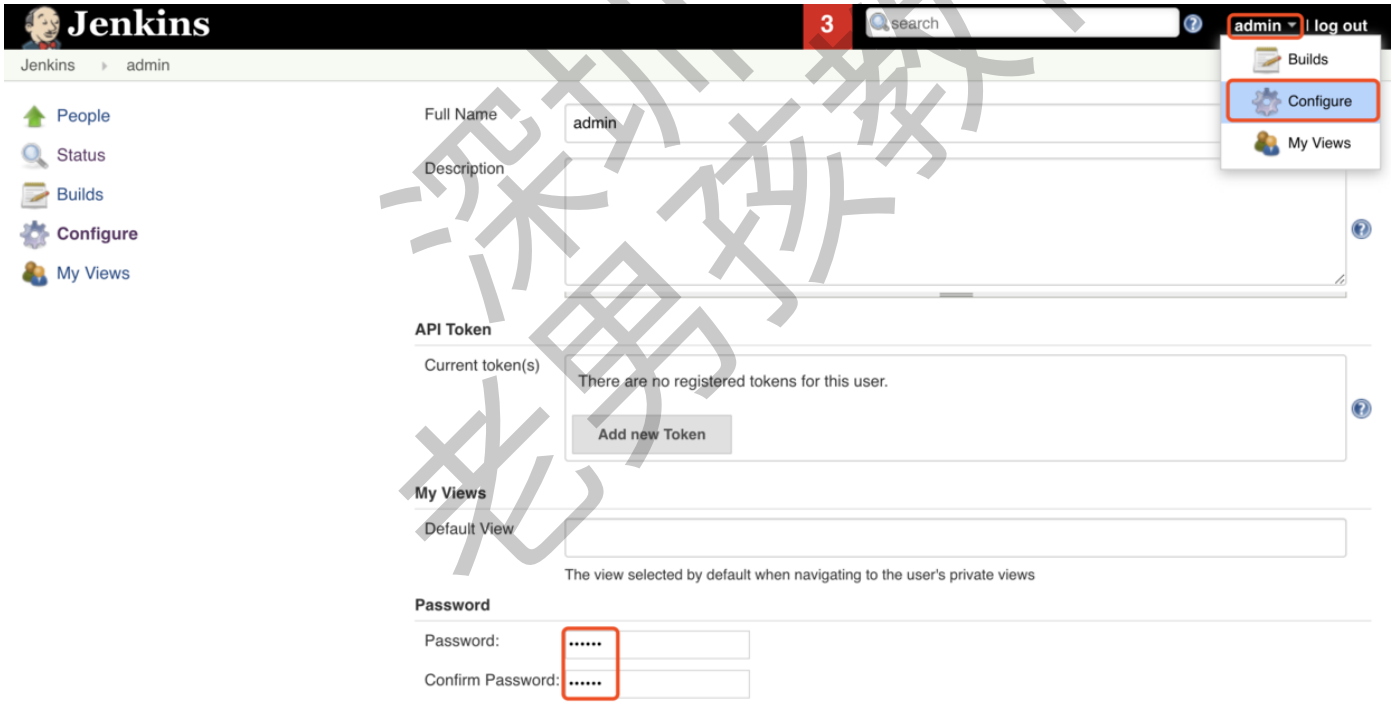
```
vim /etc/sysconfig/jenkins
JENKINS_USER="root"
```

5.启动jenkins

```
systemctl start jenkins
```

6.解锁Jenkins

7.修改admin密码

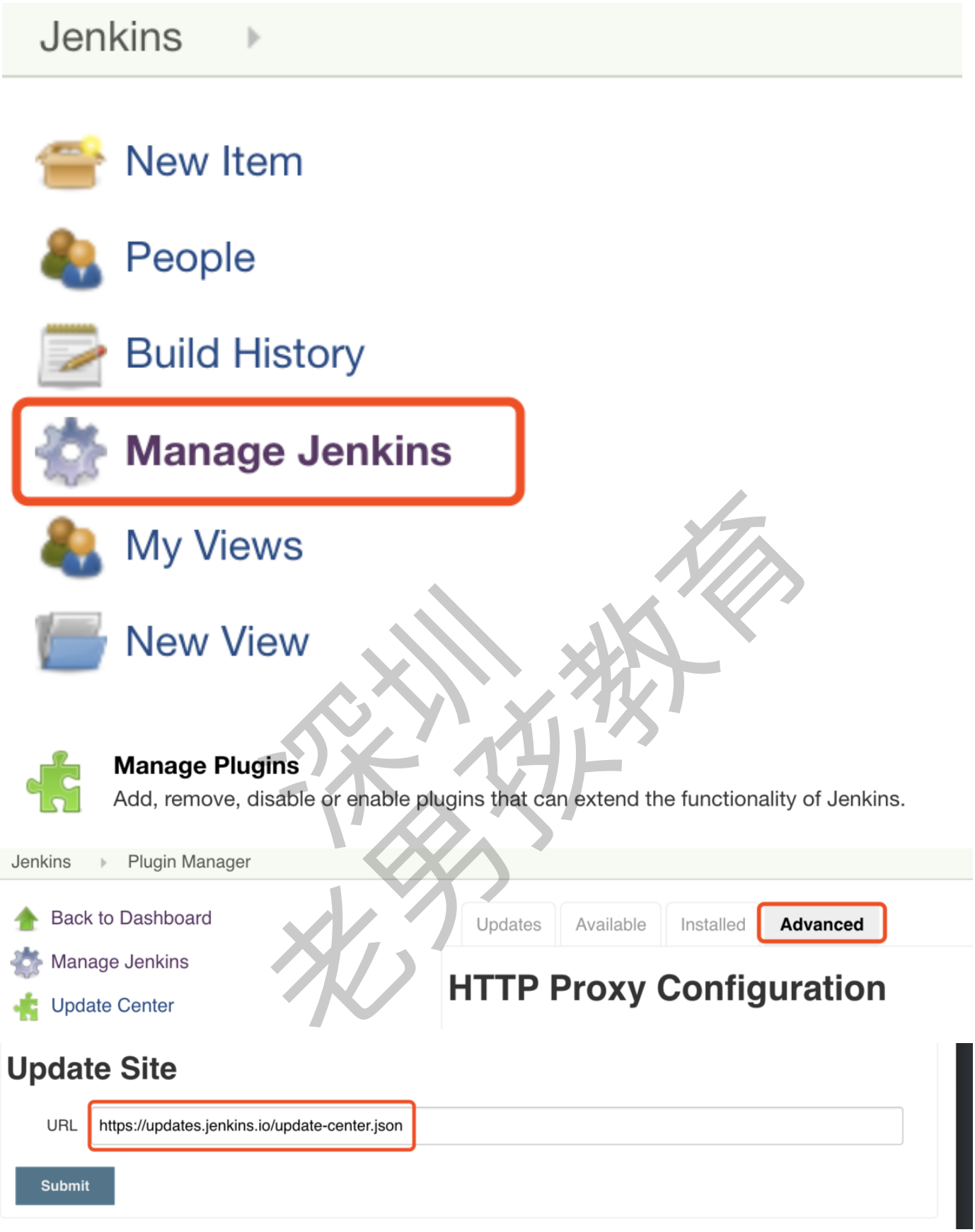


8.使用清华源作为插件地址

清华源地址：

```
https://mirrors.tuna.tsinghua.edu.cn/jenkins/updates/update-center.json
```

配置步骤：



9.使用离线安装插件

我们可以将插件提前下好，然后只需要解压到jenkins对应的目录即可

```
tar zxf jenkins_plugins.tar.gz -C /var/lib/jenkins/
ll /var/lib/jenkins/plugins/
```

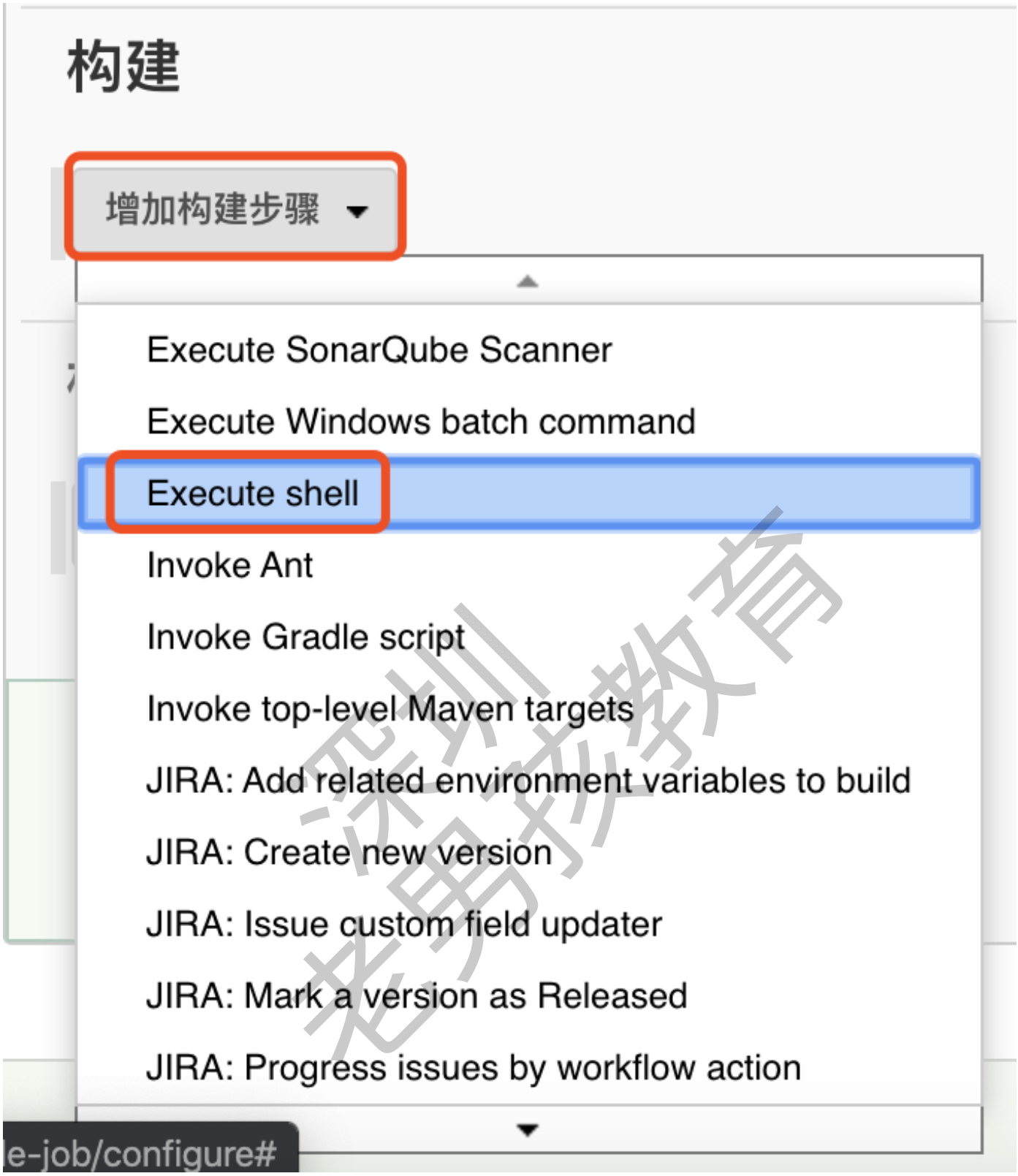
重启jenkins：

```
systemctl restart jenkins
```

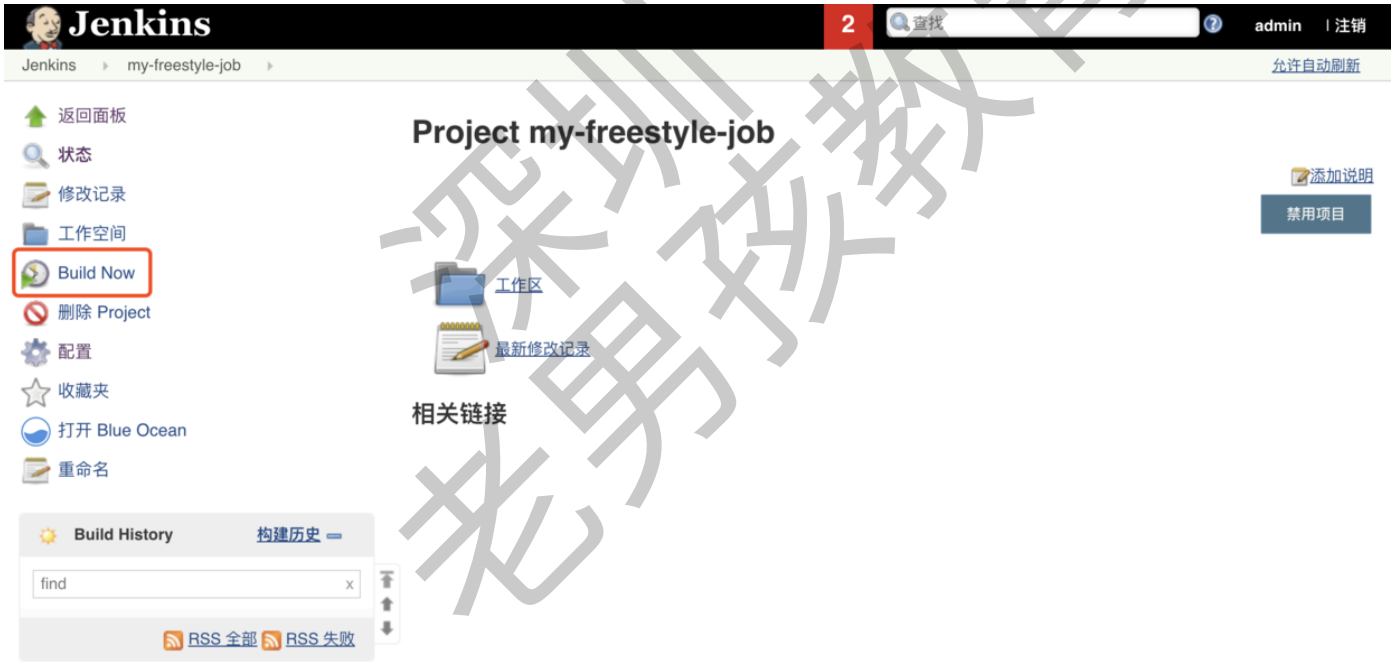
第2章 构建自由风格的项目

1.创建新任务

2.添加构建步骤


构建
增加构建步骤
Execute SonarQube Scanner
Execute Windows batch command
Execute shell
Invoke Ant
Invoke Gradle script
Invoke top-level Maven targets
JIRA: Add related environment variables to build
JIRA: Create new version
JIRA: Issue custom field updater
JIRA: Mark a version as Released
JIRA: Progress issues by workflow action
e-job/configure#

3.点击立即构建



4.查看控制台输出

控制台输出

```
Started by user admin
Running as SYSTEM
Building in workspace /var/lib/jenkins/workspace/my-freestyle-job
[my-freestyle-job] $ /bin/sh -xe /tmp/jenkins4787290991829802901.sh
+ pwd
/var/lib/jenkins/workspace/my-freestyle-job
Finished: SUCCESS
```

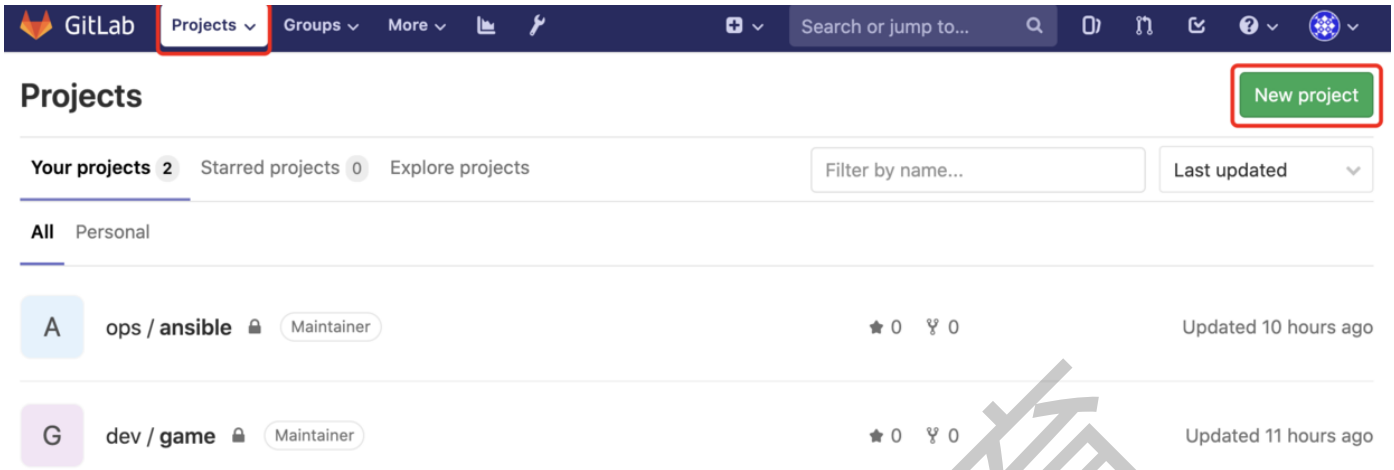
第3章 发布gitlab中的静态项目

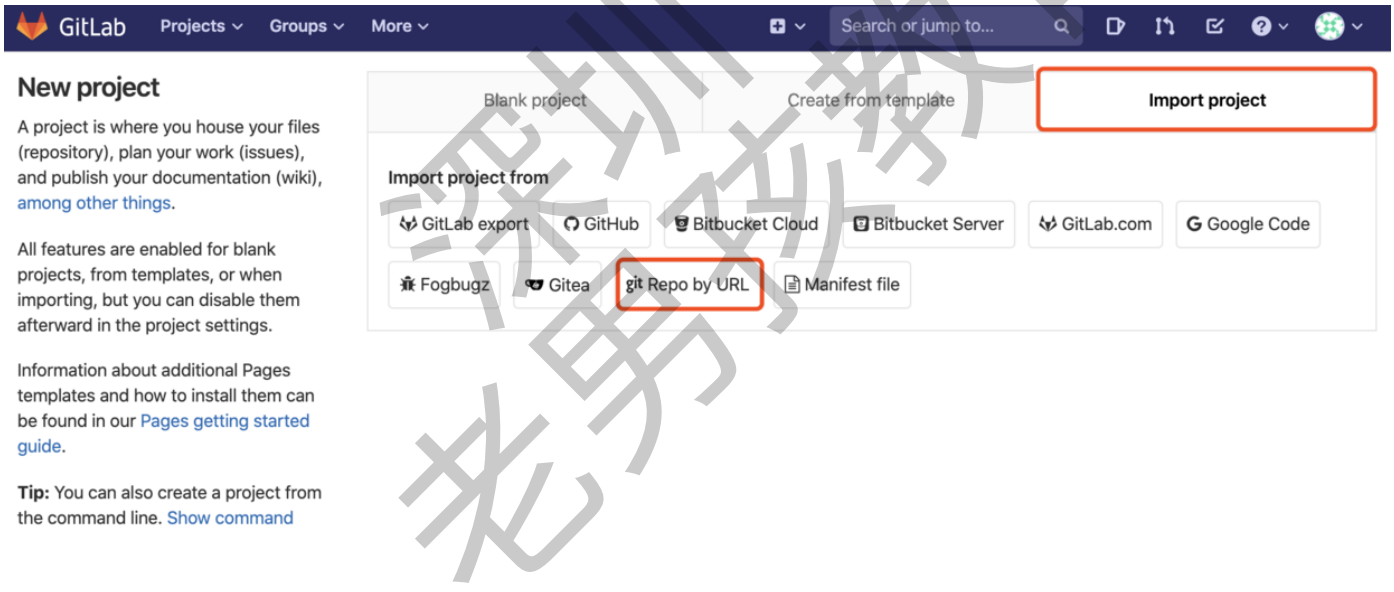
1.gitlab导入工程

这是一个h5小游戏的项目，项目地址：

```
https://gitee.com/skips/game.git
```

使用gitlab直接导入项目：





Import project from

GitLab export | GitHub | Bitbucket Cloud | Bitbucket Server | GitLab.com

Google Code | Fogbugz | Gitea | git Repo by URL | Manifest file

从git仓库的URL导入

Git repository URL

https://gitee.com/skips/game.git gitee的项目地址

Username (optional)

Password (optional)

- The repository must be accessible over `http://`, `https://` or `git://`.
- If your HTTP repository is not publicly accessible, add your credentials.
- The import will time out after 180 minutes. For repositories that take longer, use a clone/push combination.
- To import an SVN repository, check out this document.

Project name

h5game 项目名称

Project URL

http://10.0.0.200/ dev 选择组

Project slug

h5game 自动补充的项目名

Want to house several dependent projects under the same namespace? Create a group.

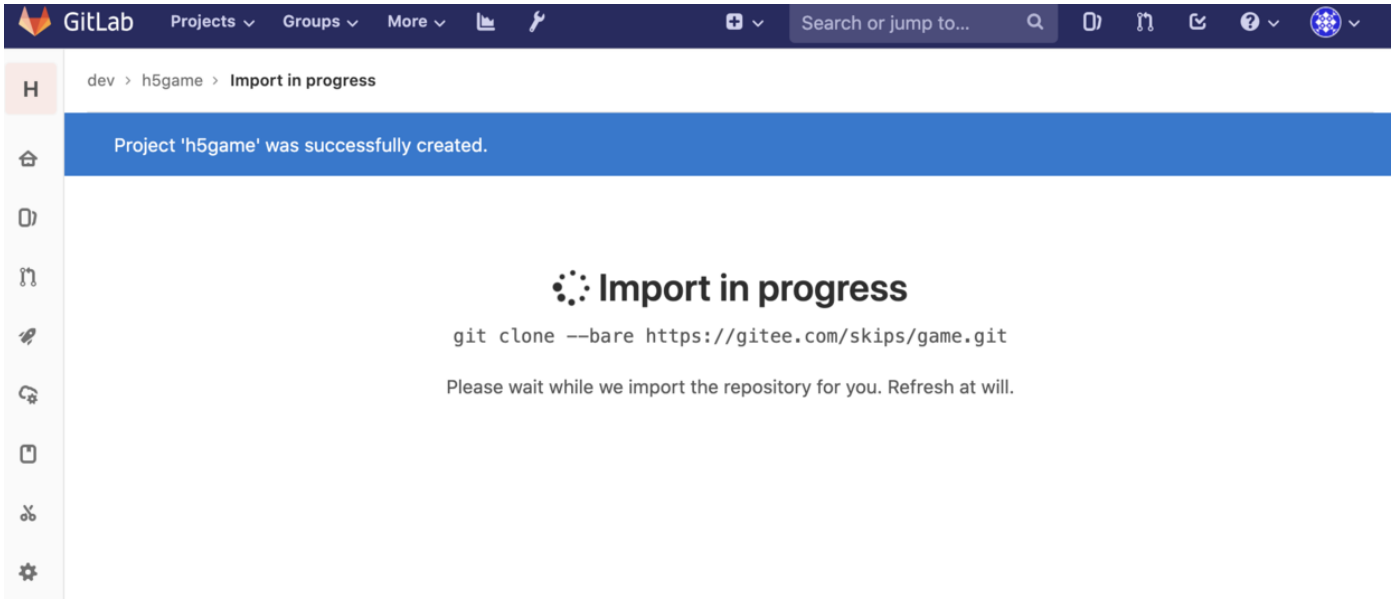
Project description (optional)

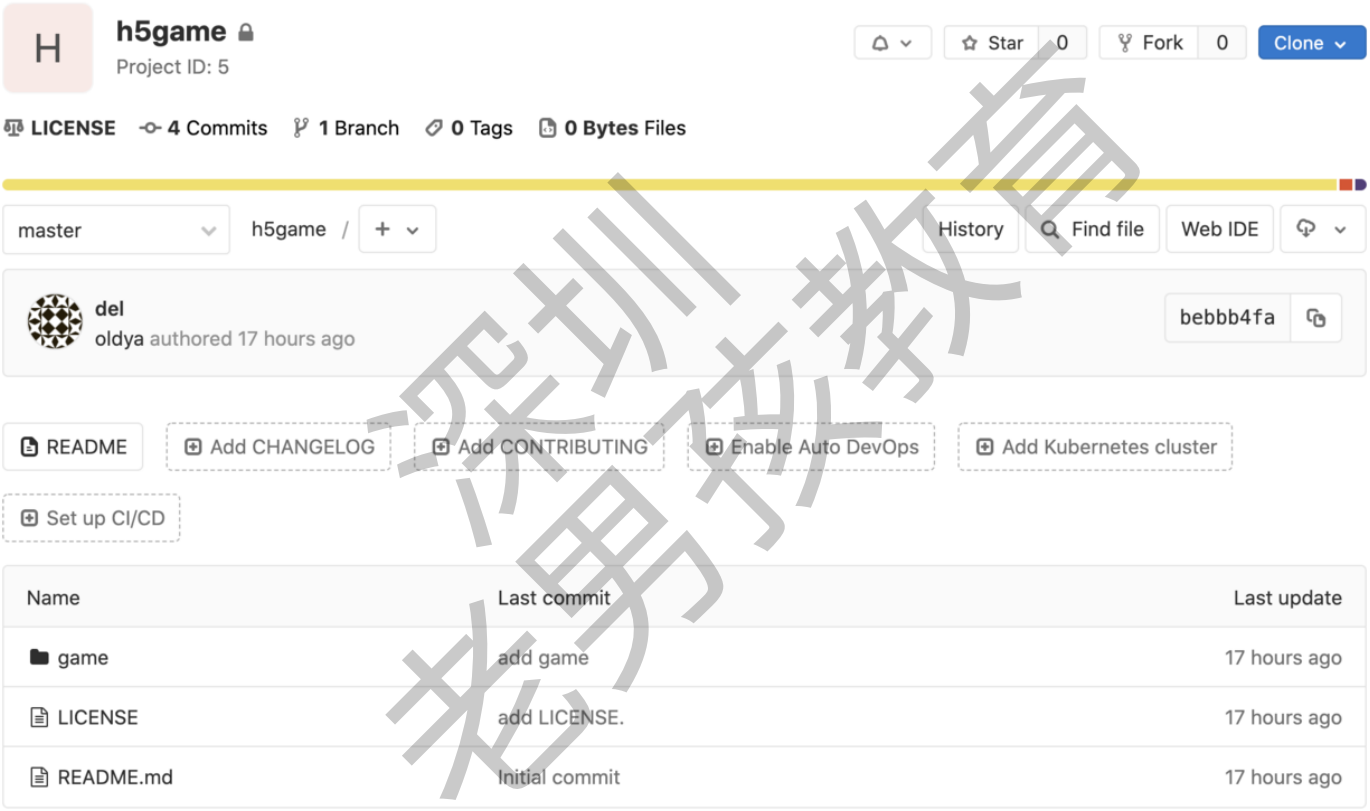
Description format

Visibility Level

Private 私有项目

Project access must be granted explicitly to each user.





2.在jenkins中关联gitlab的h5game项目

2.1 创建新项目

2.2 填写仓库地址

选择源码管理，然后填写gitlab仓库信息，但是我们发现报错了，因为jenkins没有拉取gitlab项目的权限。

3.配置jenkins访问gitlab的权限

3.1 部署公钥解释和步骤

解释

```
1.如果我们想让jenkins从gitlab上拉取代码，那么需要将jenkins的公钥信息放在gitlab上。
2.gitlab针对这种情况有一个专门的功能，叫做部署部署公钥。
3.部署公钥的作用是不需要创建虚拟用户和组，直接在需要拉取的项目里关联部署公钥即可。
```

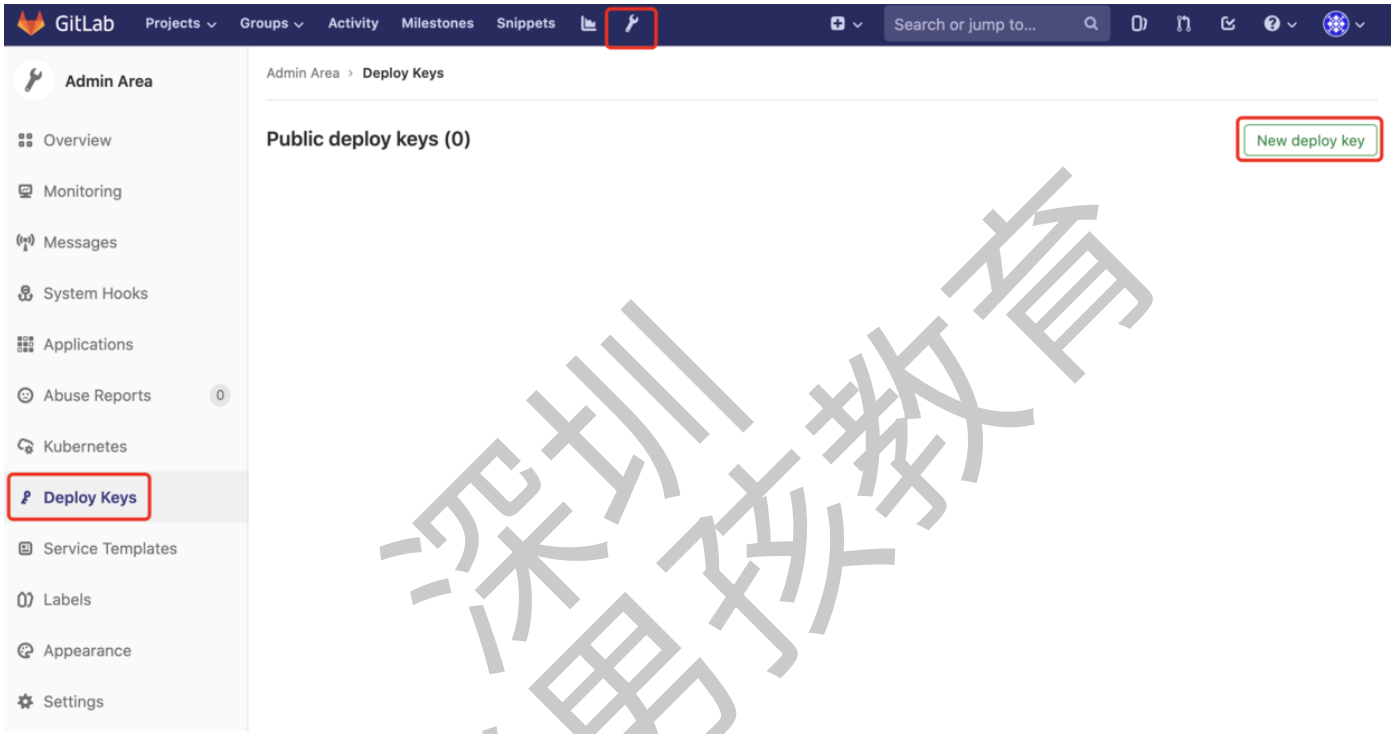
步骤

```
1.获取jenkins公钥信息
2.将jenkins公钥信息填写到gitlab的部署公钥里
3.由项目管理员操作，在需要jenkins拉取的项目里关联部署公钥
4.jenkins配置私钥凭证，部署项目时关联凭证
```

3.2 获取jenkins服务器的SSH公钥信息

```
[root@jenkins-201 ~]# cat .ssh/id_rsa.pub
ssh-rsa
AAAAB3NzaC1yc2EAAAADAQABAAABAQCg8+DQFOjR+gl1Xw83CIyGJ50vI4DBeTaMRFdu5+5pT/IMnYq1iS7/
lRS6JxXLYvVeNMDUfDxA1sOL70okyA3npjASXgJPGE1FsbpqzWjsN0TAGoZkR1VWuP9Yn0CrH7dA4lhZQfUU
VjvqzFBZK8N9iZMzIu6KOiSY/aD4Ol59vbDS4kO0rTG1DYQNnjZzMPNlIiJ+0EVkfuYRwABRFA8fmL+6btqZ
qhjGY29EHuIfzIMTDTysrtCTGxQn2ql1zwjReGiNXzmFncwvyy92DAuMbnOQiE1YNn72wThy2oWSHsCwKdIv
cNHqY2xBvFnkZ9Ltga7PgR33kbJ7Gl8tjiZF root@jenkins-201
```

3.3 gitlab添加部署公钥

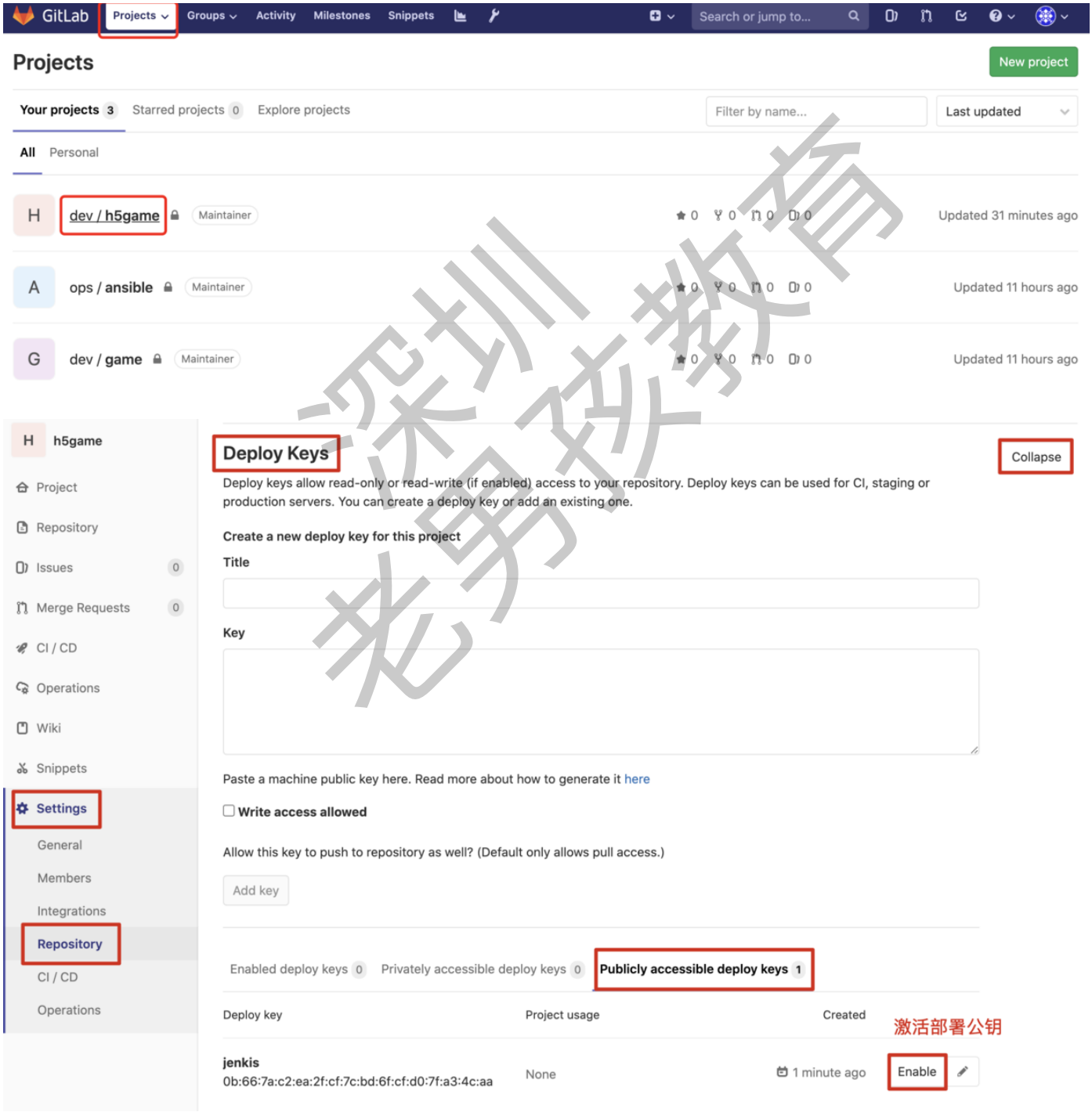


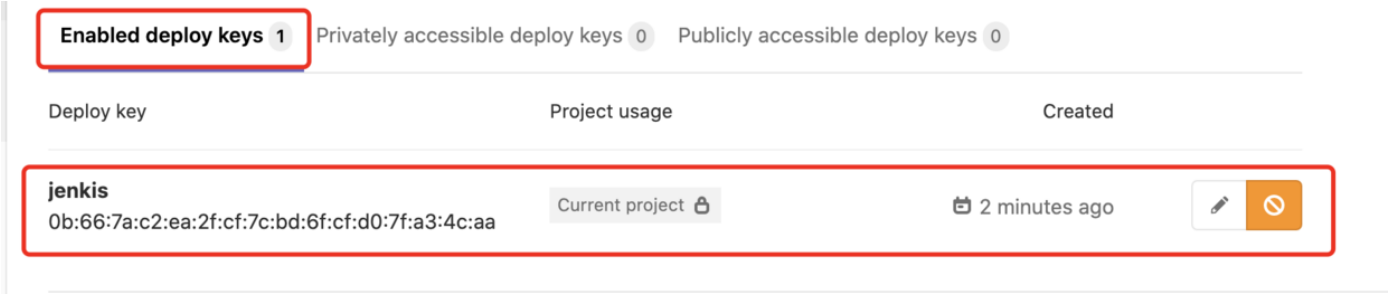
Public deploy keys (1)

New deploy key

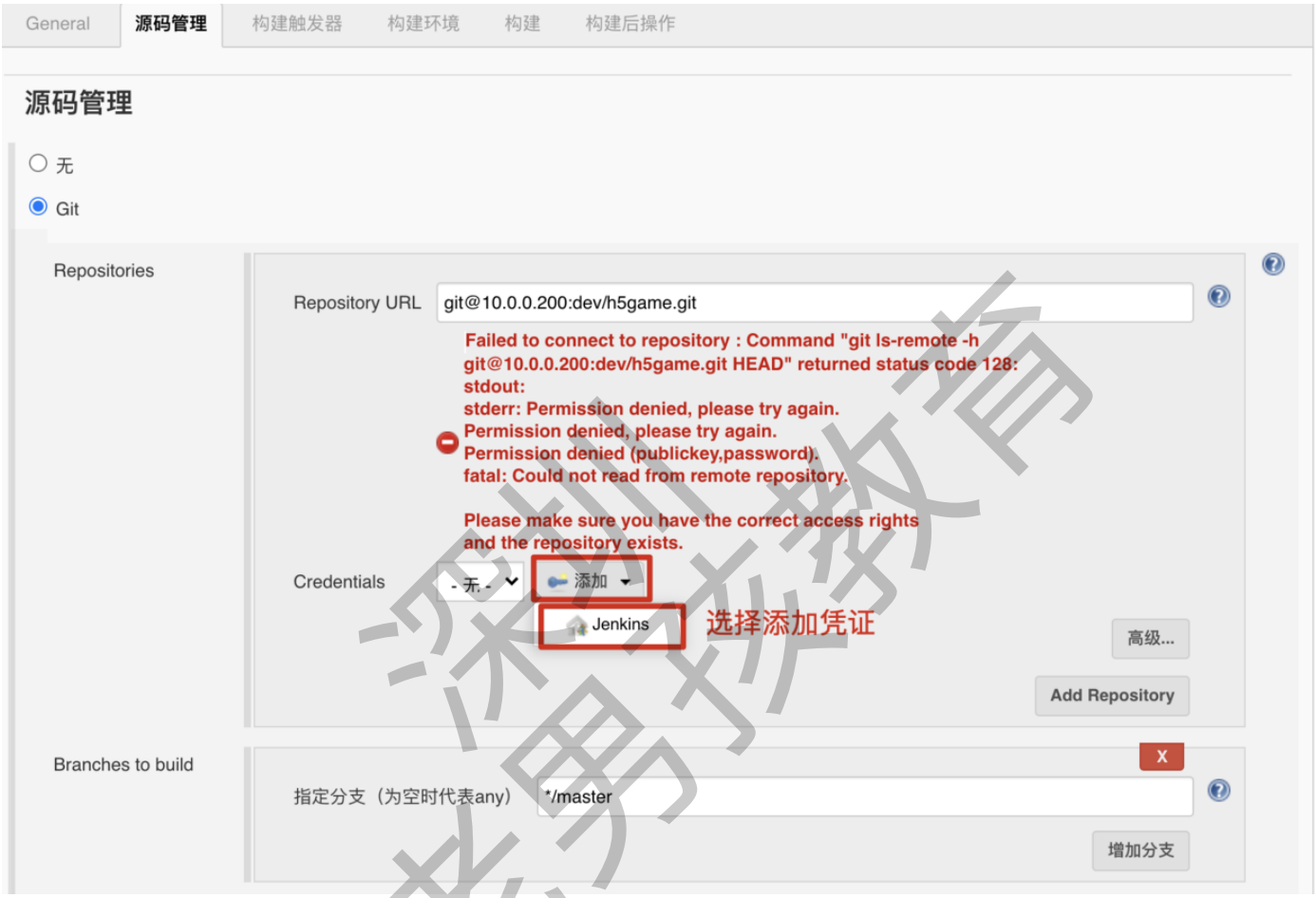
Title	Fingerprint	Projects with write access	Added at	
jenkis	0b:66:7a:c2:ea:2f:cf:7c:bd:6f:cf:d0:7f:a3:4c:aa		added just now	Edit Remove

3.4 gitlab项目关联部署公钥





3.5 jenkins配置私钥凭证




Jenkins 凭据提供者: Jenkins
添加凭据
Domain
全局凭据 (unrestricted)
类型
SSH Username with private key
选择SSH私钥
范围
全局 (Jenkins, nodes, items, all child items, etc)
ID
描述
Username
Private Key
Enter directly
Key
注意！这里填写的 是jenkins的私钥！！！
Enter New Secret Below
X4o223ehAHR2CTDoB4n0Cq0tNFXVTrngy4rByrN5MlH5PSucjBeMkXCwPBLV3w1Z
N0NWqIg/hYct6kIxx2zvIkjlWgrgD76pcpnFWLmAjEF4b9hzCKk=
-----END RSA PRIVATE KEY-----
Passphrase
添加
取消
源码管理
无
Git
Repositories
Repository URL
git@10.0.0.200:dev/h5game.git
Credentials
jenkins
添加
可以发现，警告消失了
高级...
Add Repository
Branches to build
指定分支（为空时代表any）
*/master
增加分支
源码库浏览器
(自动)
Additional Behaviours
新增

3.6 测试获取代码


Build History
构建历史
find
x
#1
2020-8-6 上午9:37
变更记录
控制台输出
编辑编译信息
删除构建 '#1'
Git Build Data
No Tags
打开 Blue Ocean
S 全部
RSS 失败

控制台输出

```
Started by user admin
Running as SYSTEM
Building in workspace /var/lib/jenkins/workspace/h5game
using credential b8c1f793-47ed-4903-995d-2273673d8f87
Cloning the remote Git repository
Cloning repository git@10.0.0.200:dev/h5game.git
 > git init /var/lib/jenkins/workspace/h5game # timeout=10
Fetching upstream changes from git@10.0.0.200:dev/h5game.git
 > git --version # timeout=10
using GIT_SSH to set credentials
 > git fetch --tags --progress git@10.0.0.200:dev/h5game.git +refs/heads/*:refs/remotes/origin/*
 > git config remote.origin.url git@10.0.0.200:dev/h5game.git # timeout=10
 > git config --add remote.origin.fetch +refs/heads/*:refs/remotes/origin/* # timeout=10
 > git config remote.origin.url git@10.0.0.200:dev/h5game.git # timeout=10
Fetching upstream changes from git@10.0.0.200:dev/h5game.git
using GIT_SSH to set credentials
 > git fetch --tags --progress git@10.0.0.200:dev/h5game.git +refs/heads/*:refs/remotes/origin/*
 > git rev-parse refs/remotes/origin/master^{commit} # timeout=10
 > git rev-parse refs/remotes/origin/origin/master^{commit} # timeout=10
Checking out Revision bebbb4fa1e2d69baf9bfe6c4a322af3abd0a5e84 (refs/remotes/origin/master)
 > git config core.sparsecheckout # timeout=10
 > git checkout -f bebbb4fa1e2d69baf9bfe6c4a322af3abd0a5e84
Commit message: "del"
First time build. Skipping changelog.
Finished: SUCCESS
```

拉取代码成功

查看拉取的代码：

```
[root@jenkins-201 ~]# ll /var/lib/jenkins/workspace/h5game
总用量 16
drwxr-xr-x 4 jenkins jenkins   47 8月   6 09:37 game
-rw-r--r-- 1 jenkins jenkins 9349 8月   6 09:37 LICENSE
-rw-r--r-- 1 jenkins jenkins  937 8月   6 09:37 README.md
```

4.编写部署脚本

```
#创建目录
mkdir -p /scripts/jenkins/

#编写脚本
cat > /scripts/jenkins/deploy.sh << 'EOF'
#!/bin/bash

PATH_CODE=/var/lib/jenkins/workspace/h5game/
PATH_WEB=/usr/share/nginx
TIME=$(date +%Y%m%d-%H%M)
IP=10.0.0.7

#打包代码
cd ${PATH_CODE}
```

```
tar zcf /opt/${TIME}-web.tar.gz ./*

#拷贝打包好的代码发送到web服务器代码目录
ssh ${IP} "mkdir ${PATH_WEB}/${TIME}-web -p"
scp /opt/${TIME}-web.tar.gz ${IP}:${PATH_WEB}/${TIME}-web

#web服务器解压代码
ssh ${IP} "cd ${PATH_WEB}/${TIME}-web && tar xf ${TIME}-web.tar.gz && rm -rf ${TIME}-web.tar.gz"
ssh ${IP} "cd ${PATH_WEB} && rm -rf html && ln -s ${TIME}-web html"
EOF

#添加可执行权限
chmod +x /scripts/jenkins/deploy.sh
```

也可以使用jenkins内置的变量来代替自定义变量，查看jenkins内置变量的地址如下：

```
http://10.0.0.201:8080/env-vars.html
```

4.jenkins调用构建脚本

在构建的位置填写执行shell脚本的命令

然后立即构建，发现报错了，提示权限不足：

控制台输出

```
Started by user admin
Running as SYSTEM
Building in workspace /var/lib/jenkins/workspace/h5game
using credential b8c1f793-47ed-4903-995d-2273673d8f87
 > git rev-parse --is-inside-work-tree # timeout=10
Fetching changes from the remote Git repository
 > git config remote.origin.url git@10.0.0.200:dev/h5game.git # timeout=10
Fetching upstream changes from git@10.0.0.200:dev/h5game.git
 > git --version # timeout=10
using GIT_SSH to set credentials
 > git fetch --tags --progress git@10.0.0.200:dev/h5game.git +refs/heads/*:refs/remotes/origin/*
 > git rev-parse refs/remotes/origin/master^{commit} # timeout=10
 > git rev-parse refs/remotes/origin/origin/master^{commit} # timeout=10
Checking out Revision bebbb4fa1e2d69baf9bfe6c4a322af3abd0a5e84 (refs/remotes/origin/master)
 > git config core.sparsecheckout # timeout=10
 > git checkout -f bebbb4fa1e2d69baf9bfe6c4a322af3abd0a5e84
Commit message: "del"
 > git rev-list --no-walk bebbb4fa1e2d69baf9bfe6c4a322af3abd0a5e84 # timeout=10
[h5game] $ /bin/sh -xe /tmp/jenkins4676819158569642509.sh
+ bash /scripts/jenkins/deploy.sh
tar (child): /opt/20200806-0957-web.tar.gz: 无法 open: 权限不够
tar (child): Error is not recoverable: exiting now
Host key verification failed.
Host key verification failed.
lost connection
Host key verification failed.
Host key verification failed.
Build step 'Execute shell' marked build as failure
Finished: FAILURE
```

报错原因是因为jenkins是以jenkins用户运行的，所以提示权限不足，我们可以修改jenkins为root用户运行。

```
[root@jenkins-201 ~]# vim /etc/sysconfig/jenkins
[root@jenkins-201 ~]# grep "USER" /etc/sysconfig/jenkins
JENKINS_USER="root"
[root@jenkins-201 ~]# systemctl restart jenkins
```

重启好之后我们重新构建一下：

控制台输出

```
Started by user admin
Running as SYSTEM
Building in workspace /var/lib/jenkins/workspace/h5game
using credential b8c1f793-47ed-4903-995d-2273673d8f87
 > git rev-parse --is-inside-work-tree # timeout=10
Fetching changes from the remote Git repository
 > git config remote.origin.url git@10.0.0.200:dev/h5game.git # timeout=10
Fetching upstream changes from git@10.0.0.200:dev/h5game.git
 > git --version # timeout=10
using GIT_SSH to set credentials
 > git fetch --tags --progress git@10.0.0.200:dev/h5game.git +refs/heads/*:refs/remotes/origin/*
 > git rev-parse refs/remotes/origin/master^{commit} # timeout=10
 > git rev-parse refs/remotes/origin/origin/master^{commit} # timeout=10
Checking out Revision bebbb4fa1e2d69baf9bfe6c4a322af3abd0a5e84 (refs/remotes/origin/master)
 > git config core.sparsecheckout # timeout=10
 > git checkout -f bebbb4fa1e2d69baf9bfe6c4a322af3abd0a5e84
Commit message: "del"
 > git rev-list --no-walk bebbb4fa1e2d69baf9bfe6c4a322af3abd0a5e84 # timeout=10
[h5game] $ /bin/sh -xe /tmp/jenkins8504823544345295828.sh
+ bash /scripts/jenkins/deploy.sh
Finished: SUCCESS
```

查看一下web服务器的代码目录

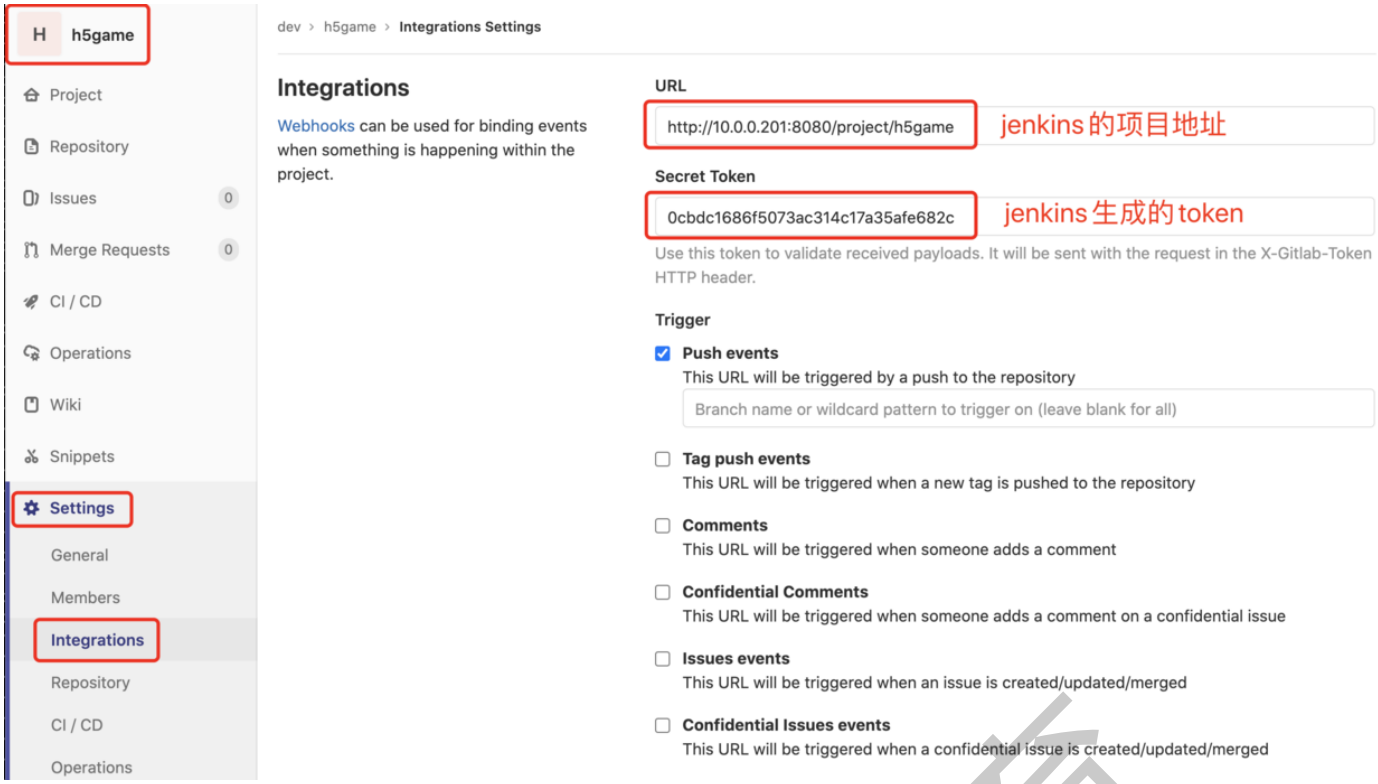
```
[root@web-7 ~]# ll /usr/share/nginx/
总用量 0
drwxr-xr-x 3 root root 50 8月   6 10:13 20200806-1013-web
lrwxrwxrwx 1 root root 17 8月   6 10:13 html -> 20200806-1013-web
```

第4章 监听gitlab自动触发构建

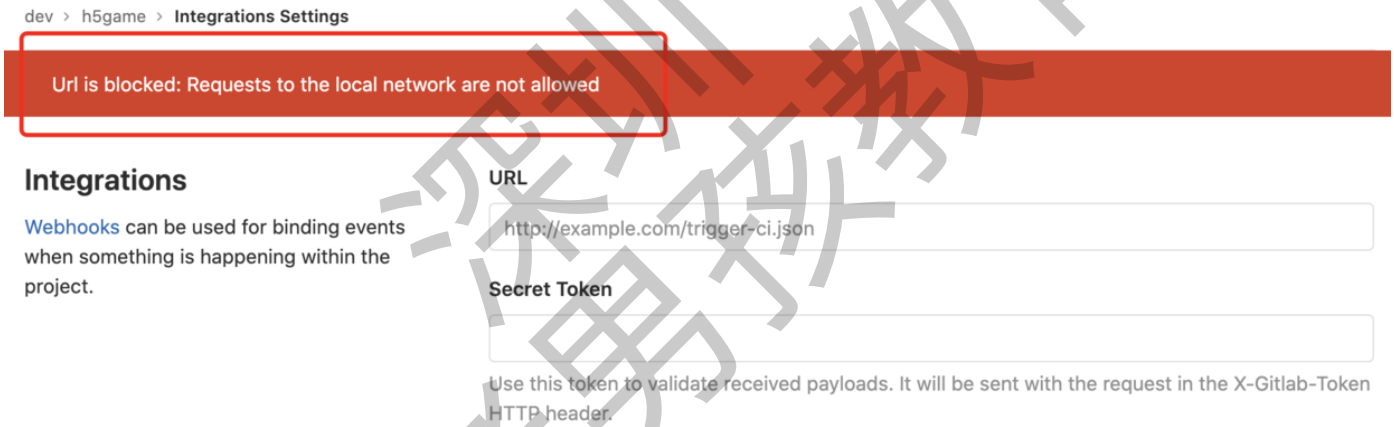
1.jenkins项目里添加构建触发器

2.gitlab添加webhook

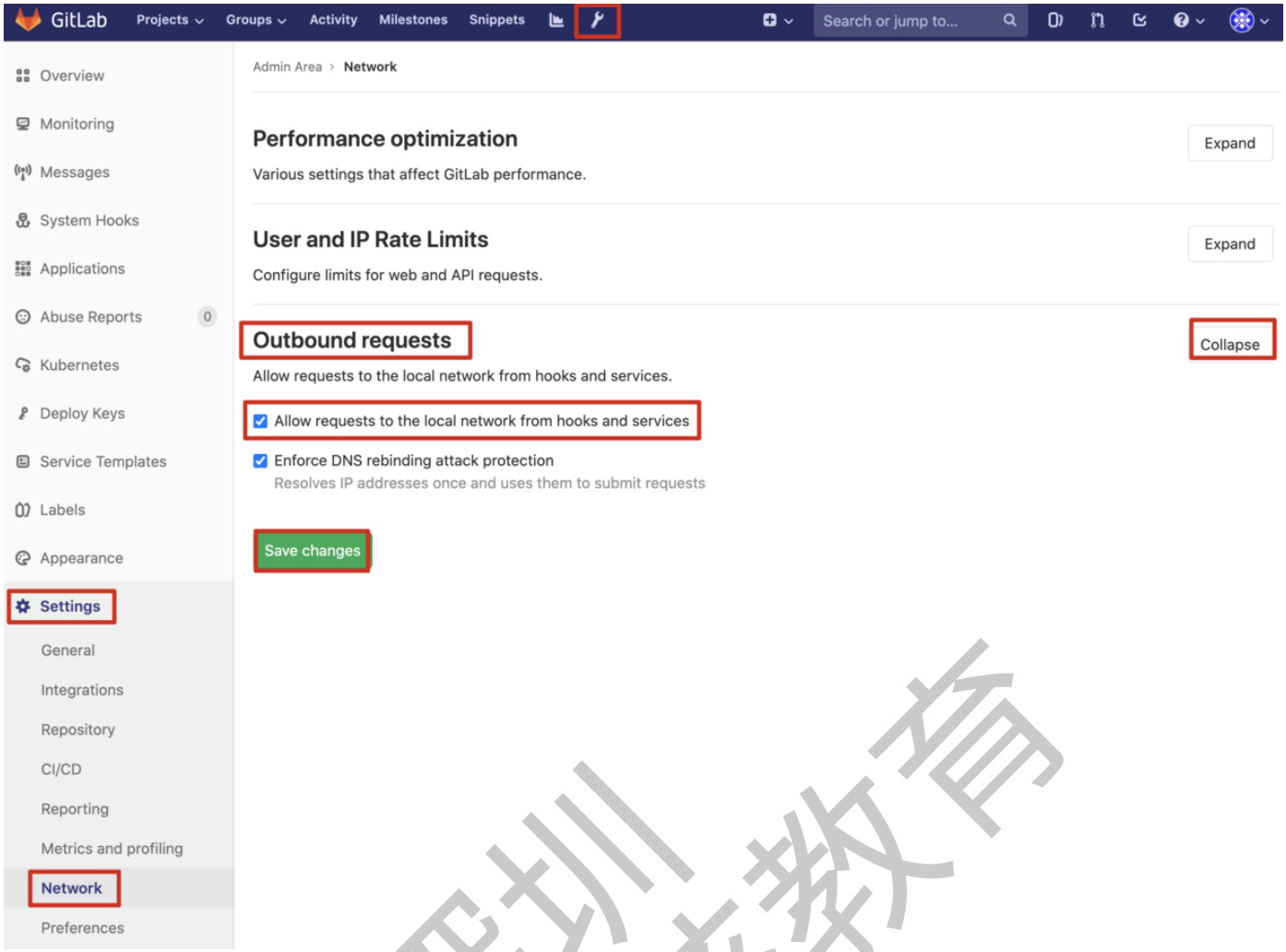
将刚才jenkins里配置的token和URL地址复制进去



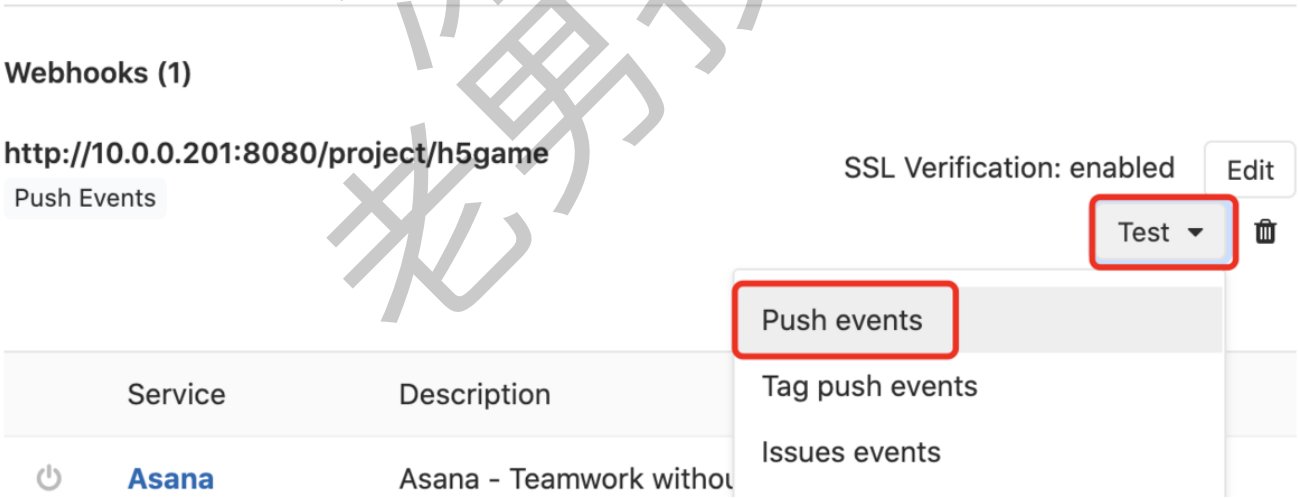
较新版本的gitlab此时点击添加会提示报错：



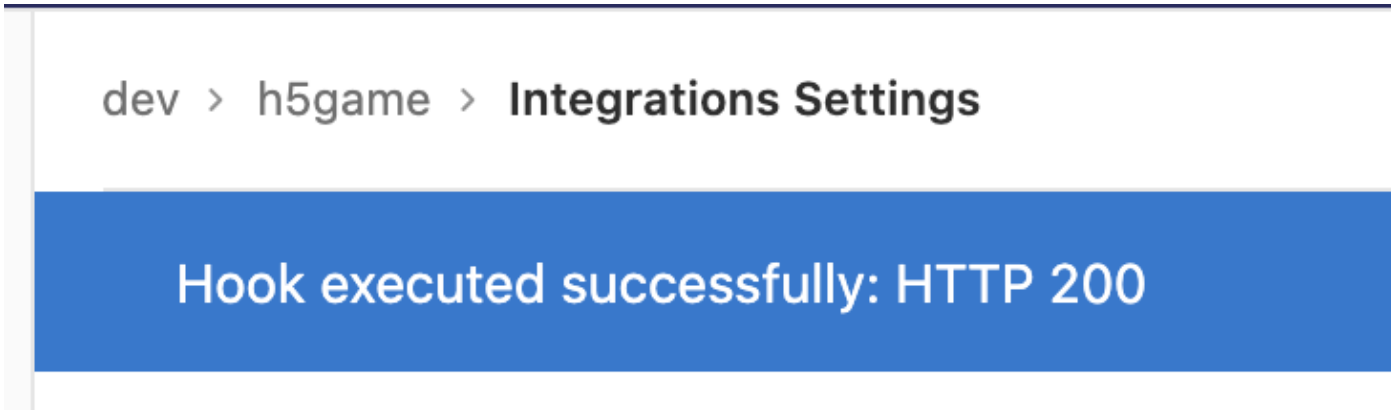
解决方法：进入admin area区域，然后点击setting-->network进行设置



正常添加成功之后，会在下方出现测试的选项



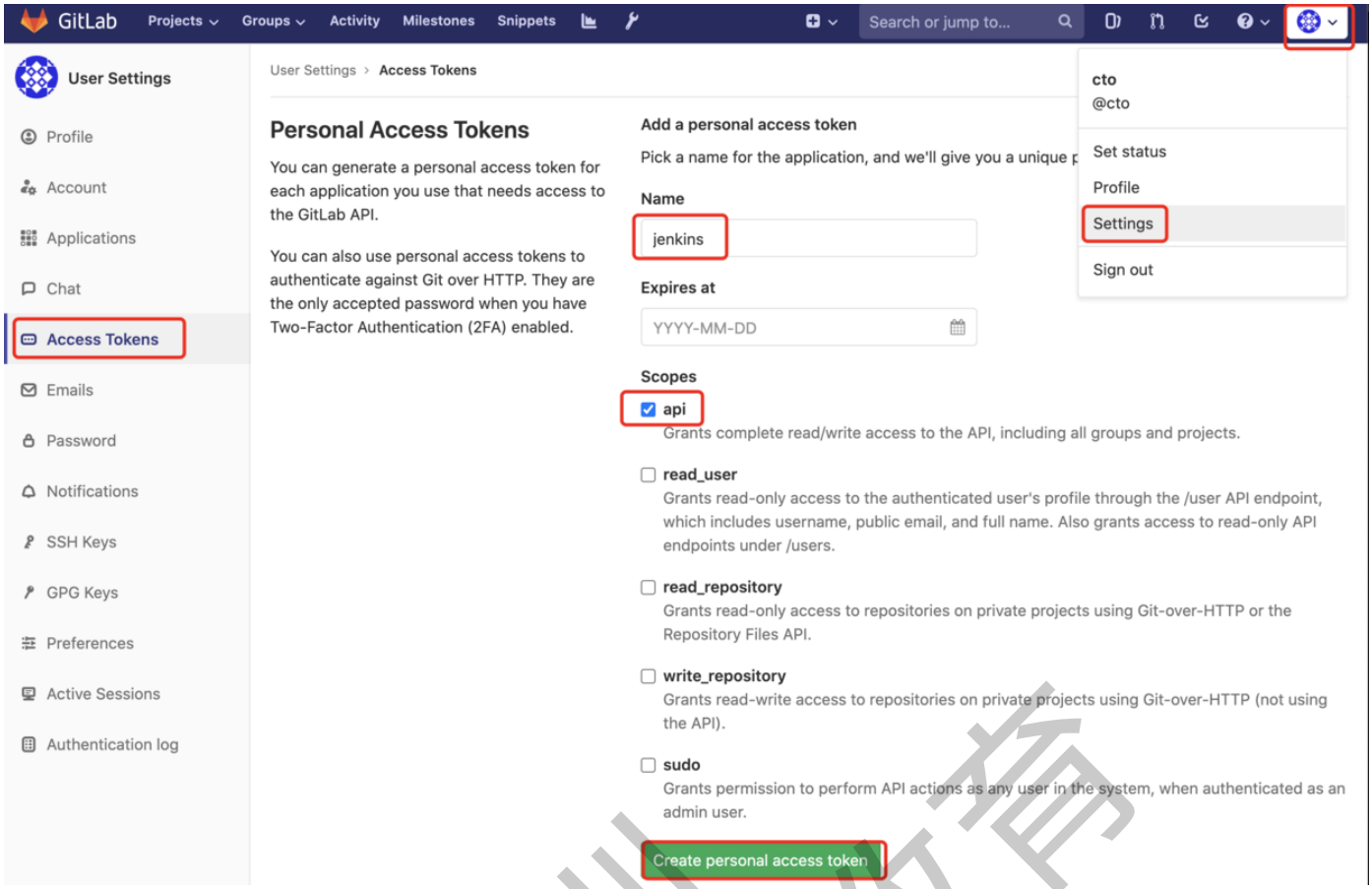
选择push事件来测试是否可以正常出发，如果可以，会在页面上方显示200状态码



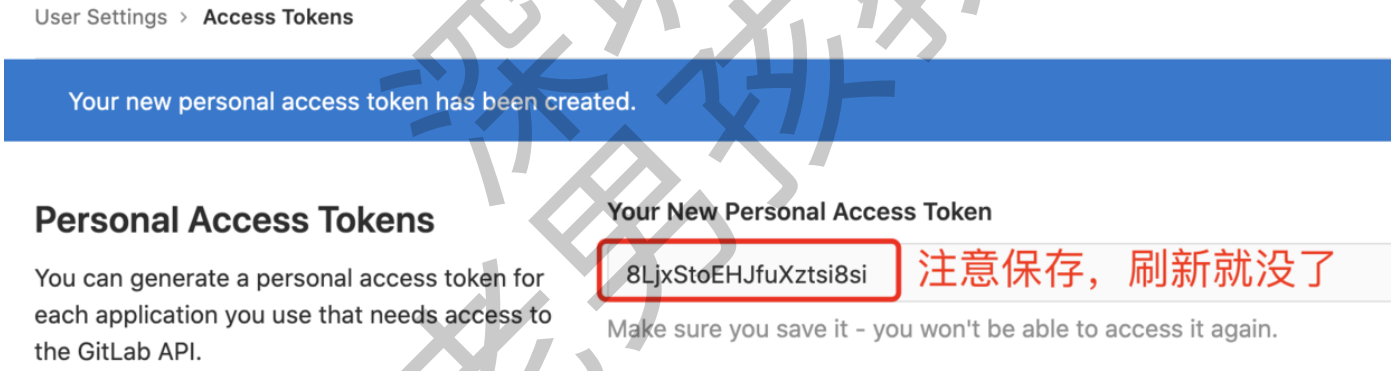
此时我们去查看jenkins项目页面，就会发现以及出发了来自gitlab的构建任务

第5章 返回构建状态给gitlab

1.gitlab生成access token

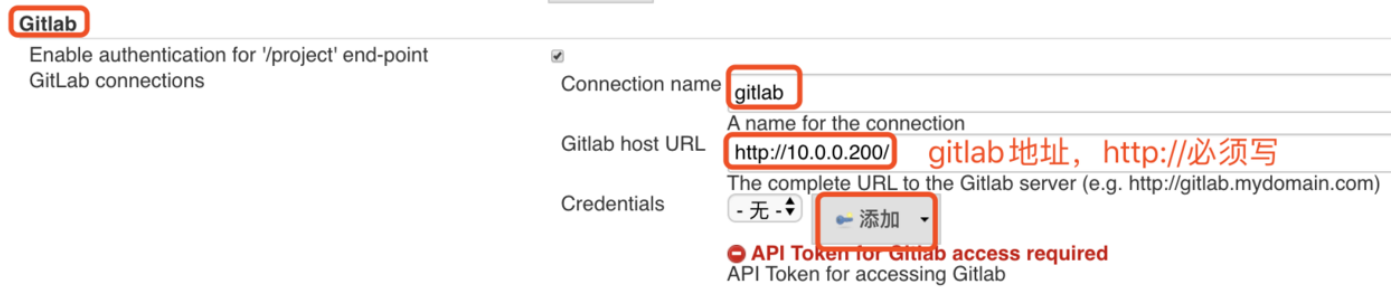


点击创建之后会生成一串token,注意及时保存，因为刷新就没有了

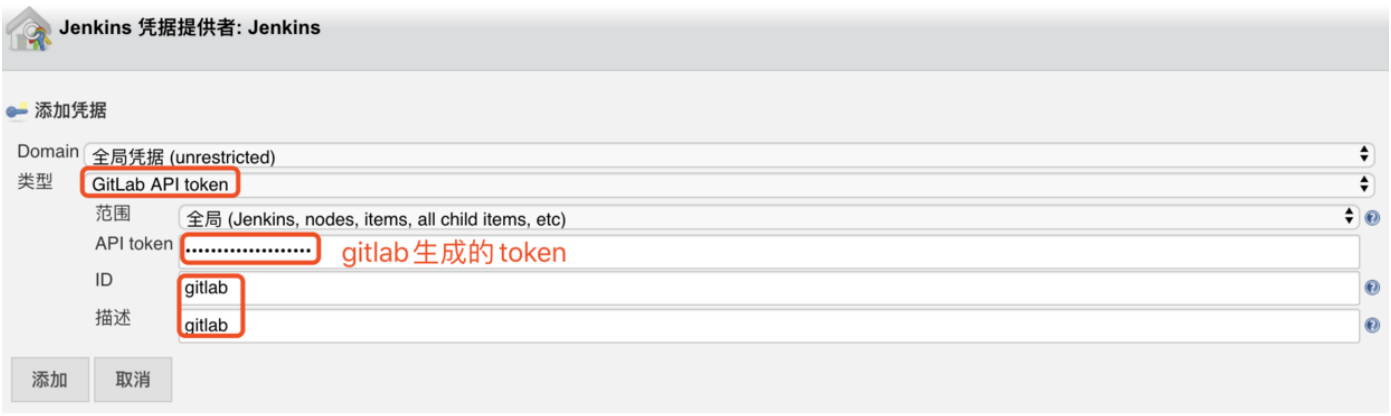


2.jenkins配置gitlab的token

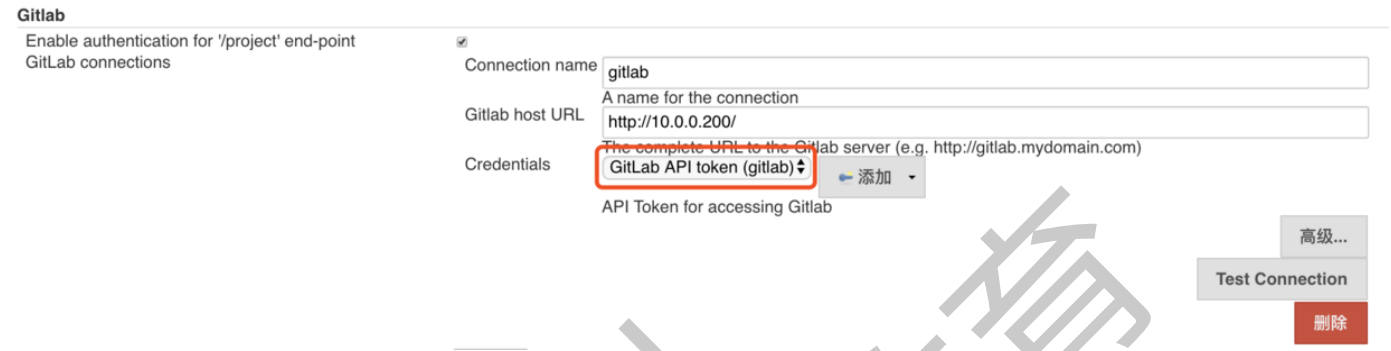
点击jenkins的系统管理-->系统设置，然后找到gitlab选项



填写gitlab的信息：



点击添加后返回上一层页面，然后选中刚才添加的gitlab凭证



3.设置项目构建后将结果通知给gitlab

4.合并分支然后检查gitlab能否收到消息

5.防止重复构建

jenkins具有很多内置变量，点击项目-->构建--> 查看 可用的环境变量列表

```
http://10.0.0.201:8080/env-vars.html/
```

这里我们使用两个变量，一个是此次提交的commit的hash，另一个是上一次提交成功的commit的hash

我们可以在部署脚本里添加一行判断，如果这两个变量一样，那么就不用再次提交了

这些变量不需要在脚本里定义，直接引用即可

第6章 tag方式发布版本

1.给代码打标签

首先我们先给代码打上标签，然后提交2个版本

v1.0版本：修改代码，然后发布v1.0版本

```
git commit -am 'v1.0'
git tag -a v1.0 -m "v1.0 稳定版"
git push -u origin v1.0
git tag
```

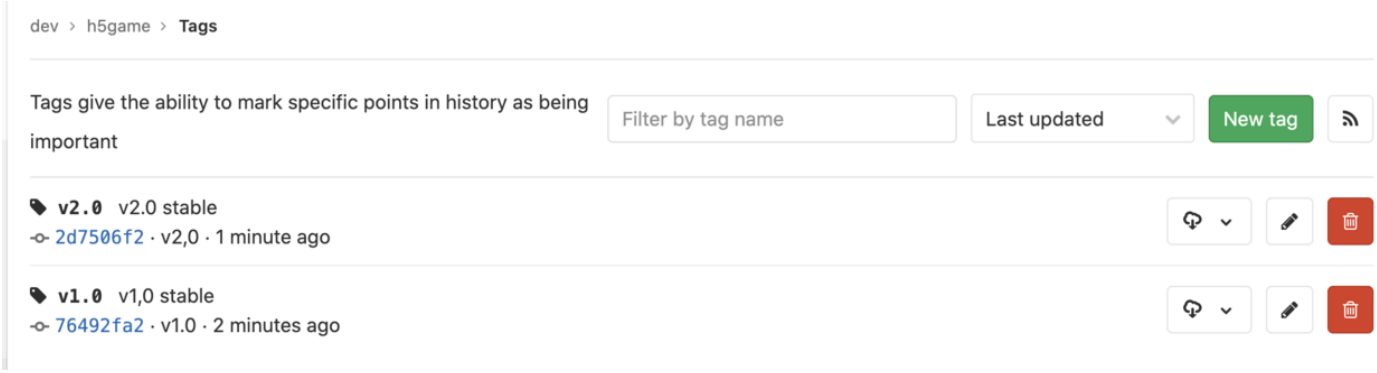
v2.0版本：修改代码，然后发布v2.0版本

```
git commit -am 'v2.0'
git tag -a v2.0 -m "v2.0 稳定版"
git push -u origin v2.0
git tag
```

2.gitlab查看标签

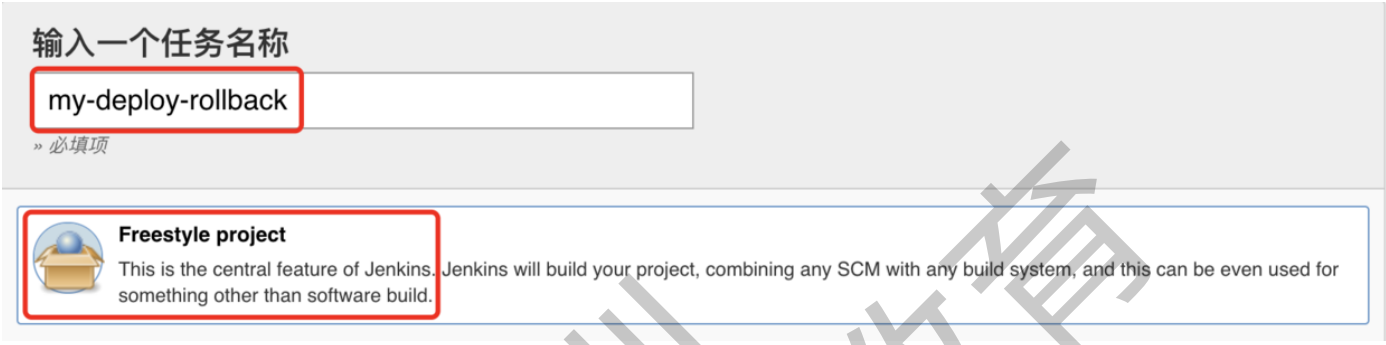
此时gitlab上可以看到2个标签

点进去之后可以看到具体标签名称



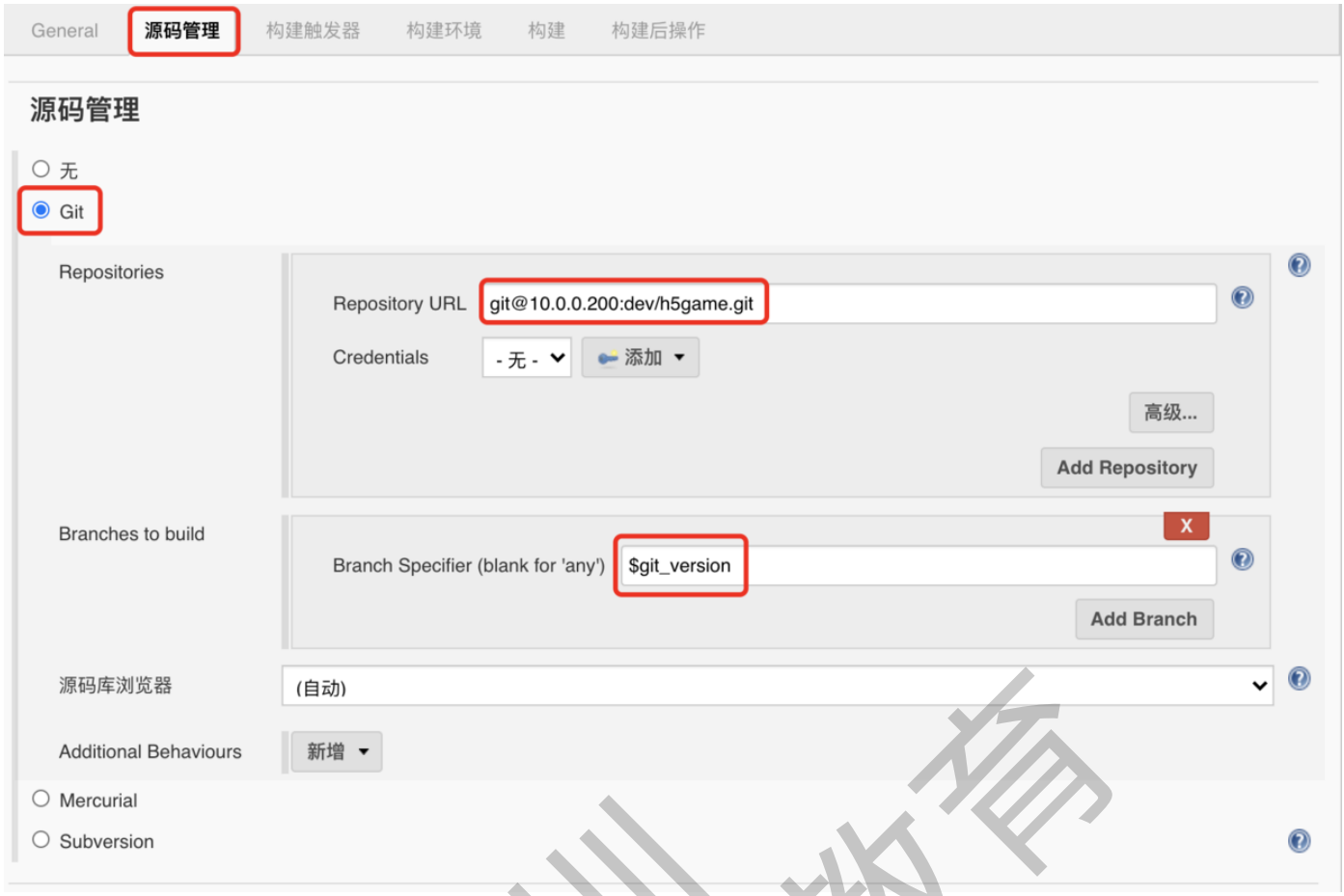
3.jenkins配置参数化构建

jenkins上我们新建一个参数化构建项目



然后配置git的标签参数:

最后还需要配置一下git仓库地址,注意需要修改拉取的版本的变量为 $git_version



此时点击项目的build with parameters就会看到对应的版本号：

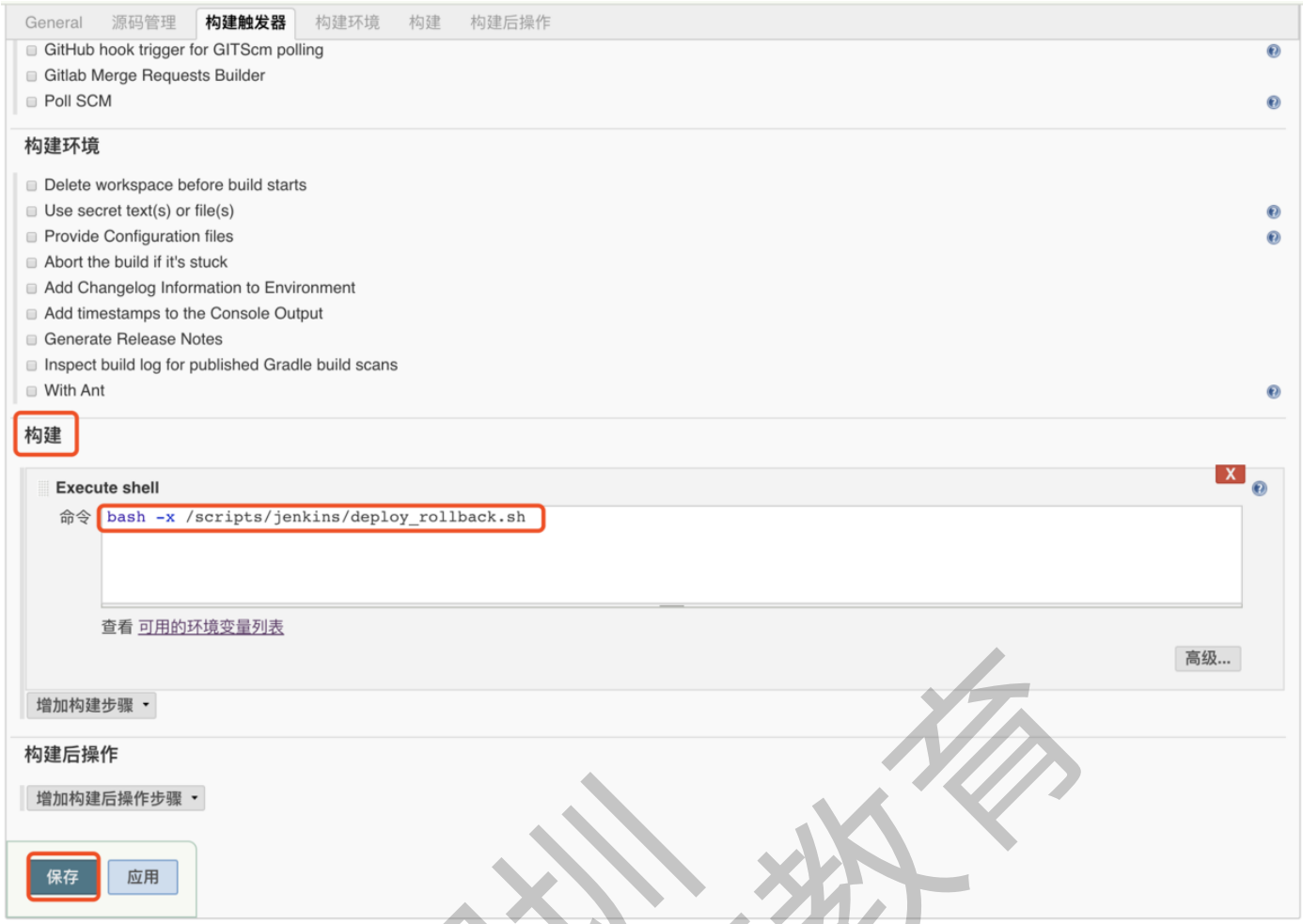
然后去jenkins工作目录下查看是否拉取了对应版本:

```
/var/lib/jenkins/workspace/my-deploy-rollback
```

4.优化部署脚本

```
cat >/scripts/jenkins/deploy_rollback.sh<<'EOF'
#!/bin/bash

PATH_CODE=/var/lib/jenkins/workspace/my-deploy-rollback/
PATH_WEB=/usr/share/nginx
IP=10.0.0.7

#打包代码
cd ${PATH_CODE}
tar zcf /opt/web-${git_version}.tar.gz ./*

#拷贝打包好的代码发送到web服务器代码目录
ssh ${IP} "mkdir ${PATH_WEB}/web-${git_version} -p"
scp /opt/web-${git_version}.tar.gz ${IP}:${PATH_WEB}/web-${git_version}

#web服务器解压代码
ssh ${IP} "cd ${PATH_WEB}/web-${git_version} && tar xf web-${git_version}.tar.gz && rm -rf web-${git_version}.tar.gz"
ssh ${IP} "cd ${PATH_WEB} && rm -rf html && ln -s web-${git_version} html"
EOF
```

5.jenkins添加执行脚本动作并测试



6.测试发布

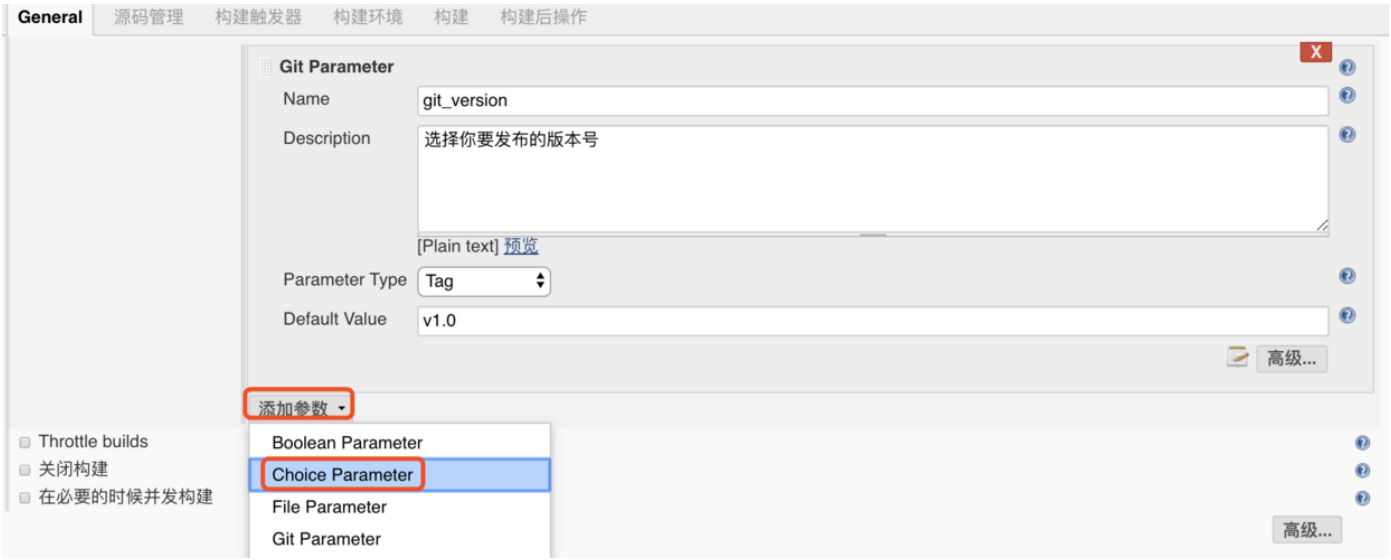
然后去web服务器上查看发现已经发布了

```
[root@web-7 ~]# ll /usr/share/nginx/
总用量 0
lrwxrwxrwx 1 root root   8 8月   6 15:59 html -> web-v2.0
drwxr-xr-x 3 root root 68 8月   6 15:59 web-v2.0
```

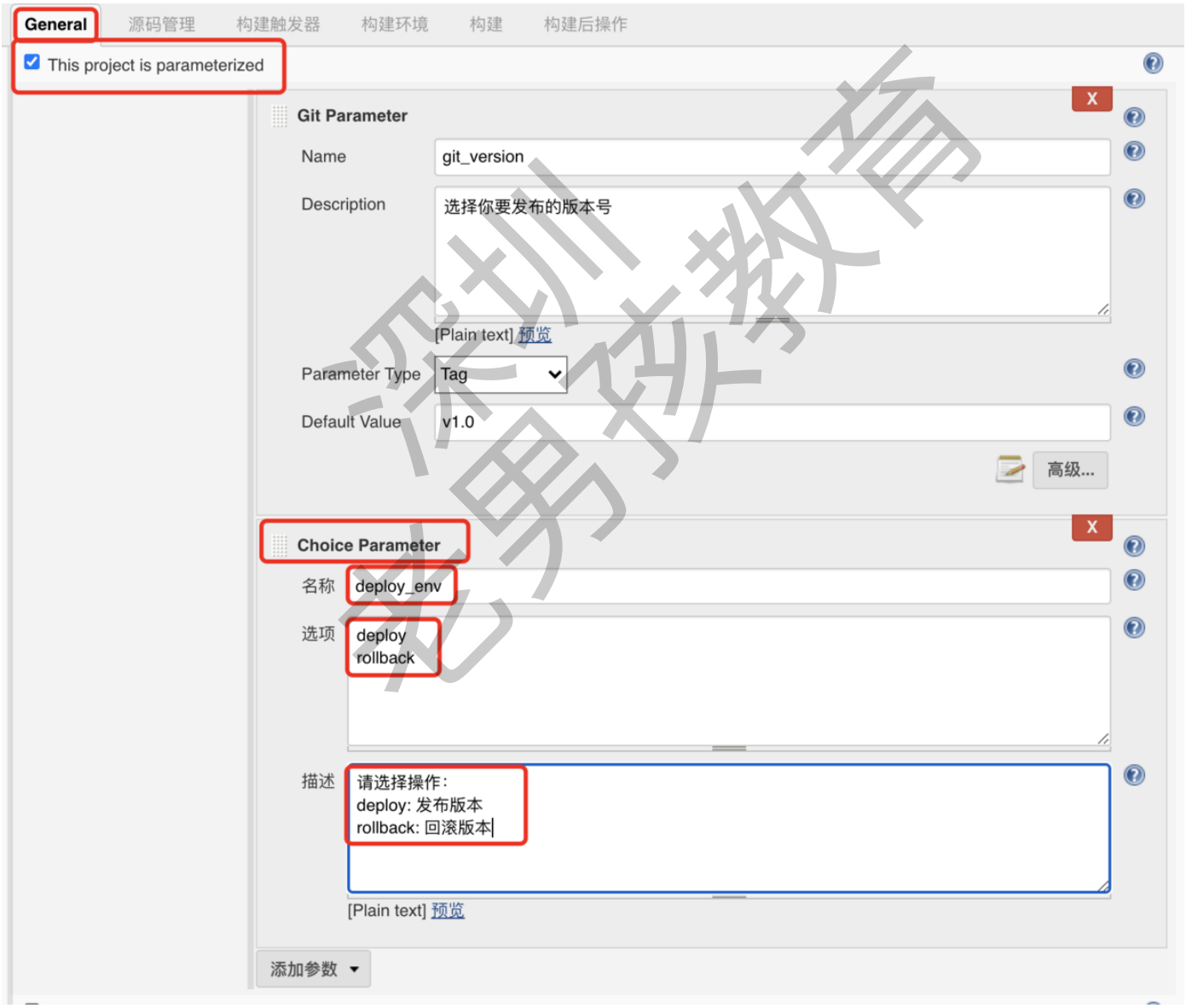
第7章 tag方式回滚版本

1.jenkins配置回滚选项参数

在工程配置里添加选项参数:



在参数化构建里添加2个选项：发布和回滚



此时查看构建页面就会发现多了选项卡:

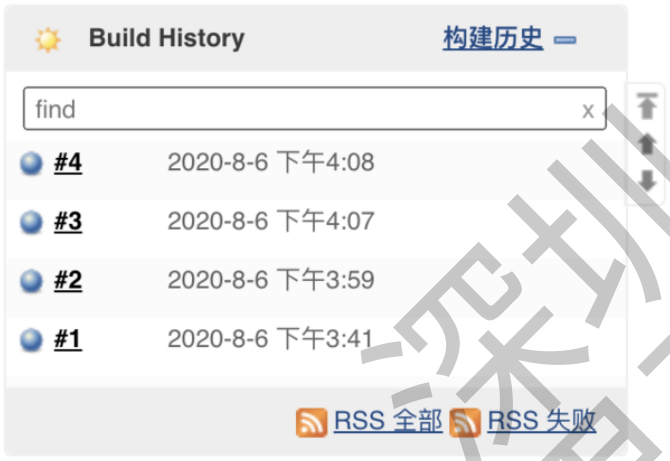
2.修改发布脚本增加条件判断

```
cat >/scripts/jenkins/deploy_rollback.sh <<'EOF'
#!/bin/bash

PATH_CODE=/var/lib/jenkins/workspace/my-deploy-rollback/
PATH_WEB=/usr/share/nginx
IP=10.0.0.7

#打包代码
code_tar(){
        cd ${PATH_CODE}
        tar zcf /opt/web-${git_version}.tar.gz ./*
}

#拷贝打包好的代码发送到web服务器代码目录
code_scp(){
        ssh ${IP} "mkdir ${PATH_WEB}/web-${git_version} -p"
        scp /opt/web-${git_version}.tar.gz ${IP}:${PATH_WEB}/web-${git_version}
}

#web服务器解压代码
code_xf(){
        ssh ${IP} "cd ${PATH_WEB}/web-${git_version} && tar xf web-
${git_version}.tar.gz && rm -rf web-${git_version}.tar.gz"
}

#创建代码软链接
code_ln(){
        ssh ${IP} "cd ${PATH_WEB} && rm -rf html && ln -s web-${git_version} html"
}

```

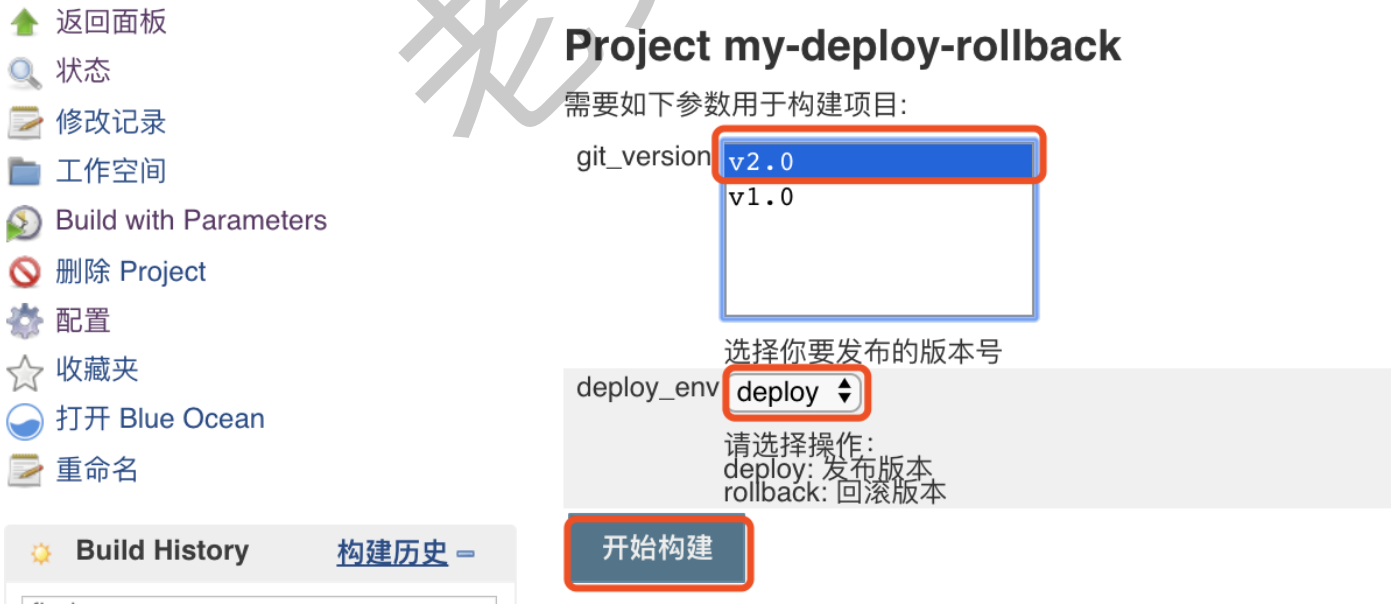
```
main(){
        code_tar
        code_scp
        code_xf
        code_ln
}

#选择发布还是回滚
if [ "${deploy_env}" == "deploy" ]
then
        ssh ${IP} "ls ${PATH_WEB}/web-${git_version}" >/dev/null 2>&1
        if [ $? == 0 -a ${GIT_COMMIT} == ${GIT_PREVIOUS_SUCCESSFUL_COMMIT} ]
        then
                echo "web-${git_version} 已部署,不允许重复构建"
                exit
        else
                main
        fi
elif [ "${deploy_env}" == "rollback" ]
then
        code_ln
fi
EOF
```

3.测试回滚功能

部署v1.0版本



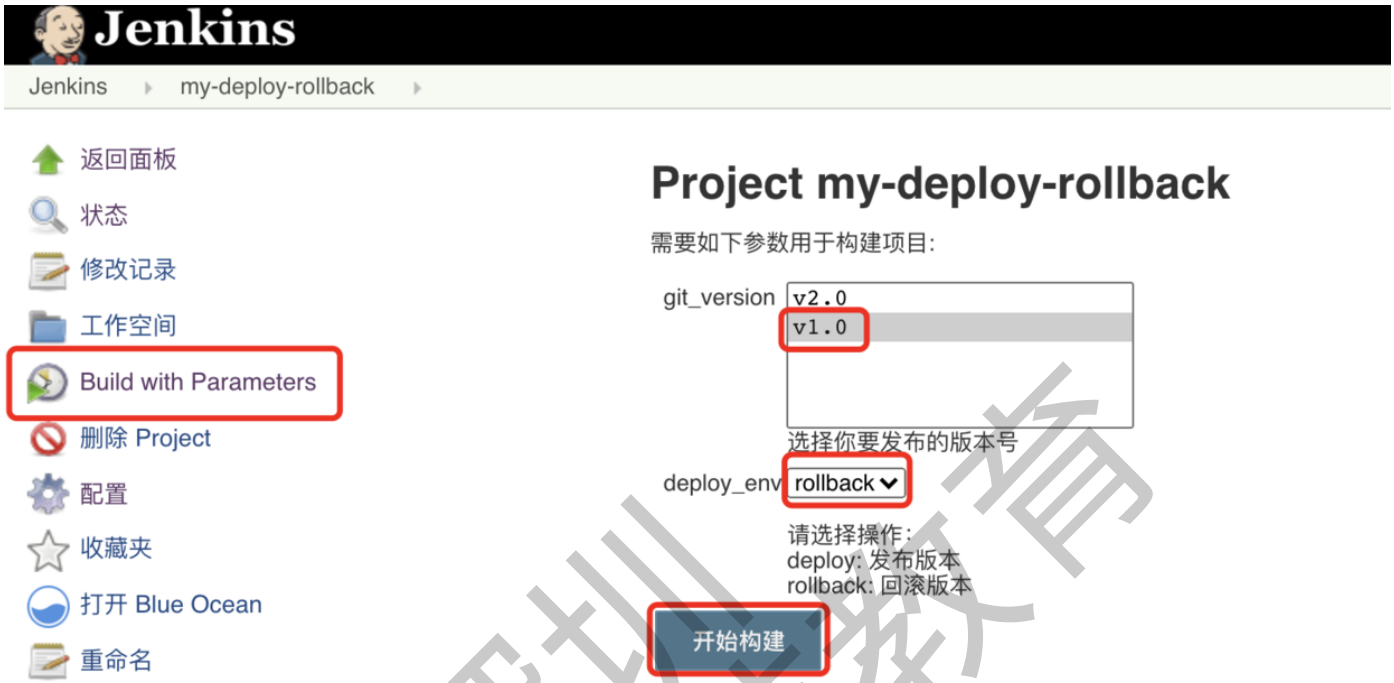
部署v2.0版本:



检查web服务器当前的版本

```
[root@web-7 ~]# ll /usr/share/nginx/
总用量 0
lrwxrwxrwx 1 root root  8 8月   6 16:52 html -> web-v2.0
drwxr-xr-x 3 root root 68 8月   6 16:51 web-v1.0
drwxr-xr-x 3 root root 68 8月   6 16:52 web-v2.0
```

然后我们选择v1.0版本并且选择回滚操作：



查看控制台显示回滚成功：

在web服务器上查看发现已经回滚成功：

```
[root@web-7 ~]# ll /usr/share/nginx/
总用量 0
lrwxrwxrwx 1 root root  8 8月   6 16:56 html -> web-v1.0
drwxr-xr-x 3 root root 68 8月   6 16:51 web-v1.0
drwxr-xr-x 3 root root 68 8月   6 16:52 web-v2.0
```

4.发布新代码并打标签测试

修改代码并发布v3.0:

```
cd h5game/
echo v3.0 >> index.html
git commit -am 'v3.0'
git tag -a v3.0 -m 'v3.0 稳定版'
git push -u origin v3.0
git tag
```

jenkins查看并发布3.0版本:

web服务器查看发布情况:

```
[root@web-7 ~]# ll /usr/share/nginx/
总用量 0
lrwxrwxrwx 1 root root  8 8月   6 16:58 html -> web-v3.0
drwxr-xr-x 3 root root 68 8月   6 16:51 web-v1.0
drwxr-xr-x 3 root root 68 8月   6 16:52 web-v2.0
drwxr-xr-x 3 root root 68 8月   6 16:58 web-v3.0
```

然后工程选择回滚到v2.0版本:

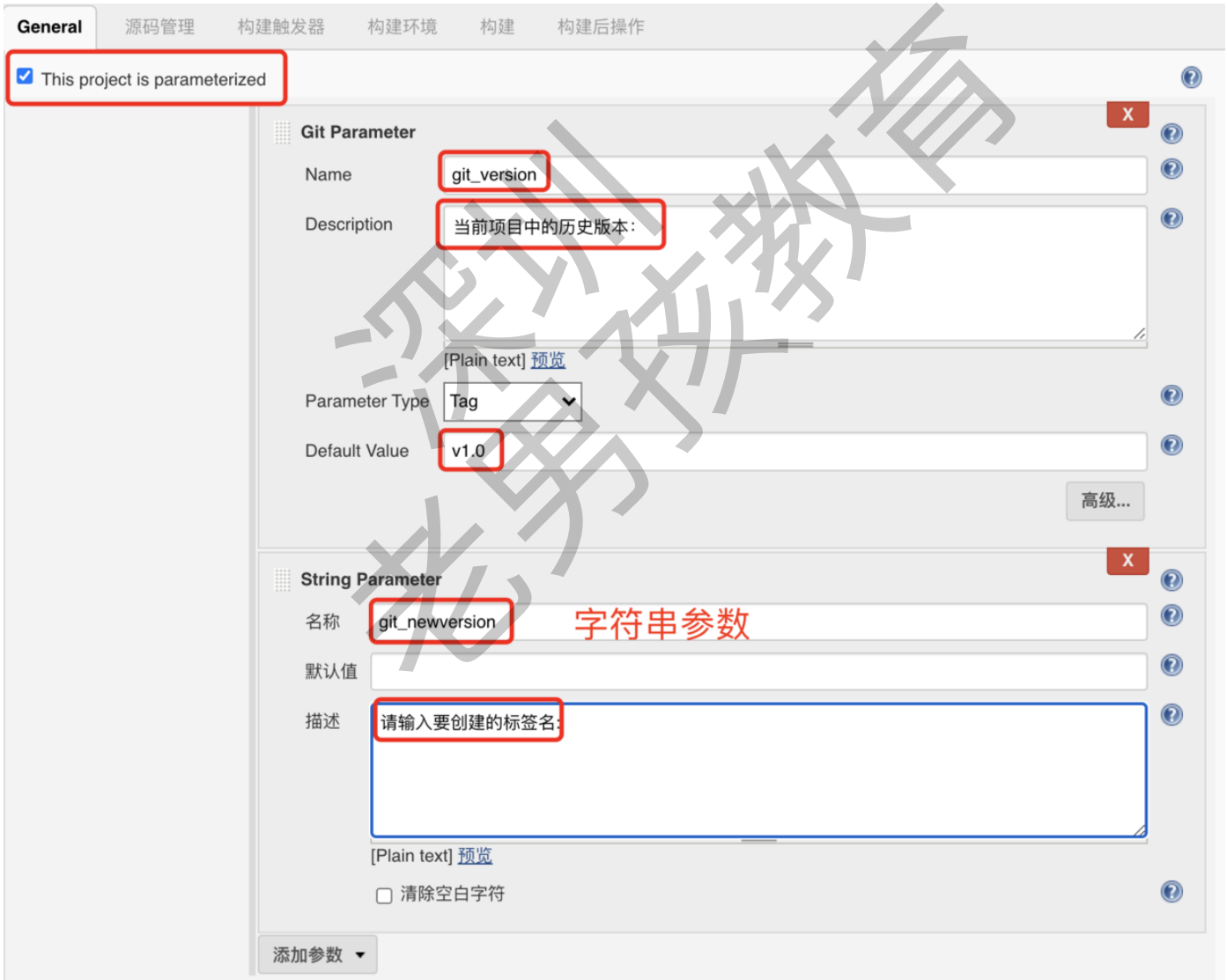
再次在web服务器上查看:

```
[root@web-7 ~]# ll /usr/share/nginx/
总用量 0
lrwxrwxrwx 1 root root  8 8月   6 16:59 html -> web-v2.0
drwxr-xr-x 3 root root 68 8月   6 16:51 web-v1.0
drwxr-xr-x 3 root root 68 8月   6 16:52 web-v2.0
drwxr-xr-x 3 root root 68 8月   6 16:58 web-v3.0
```

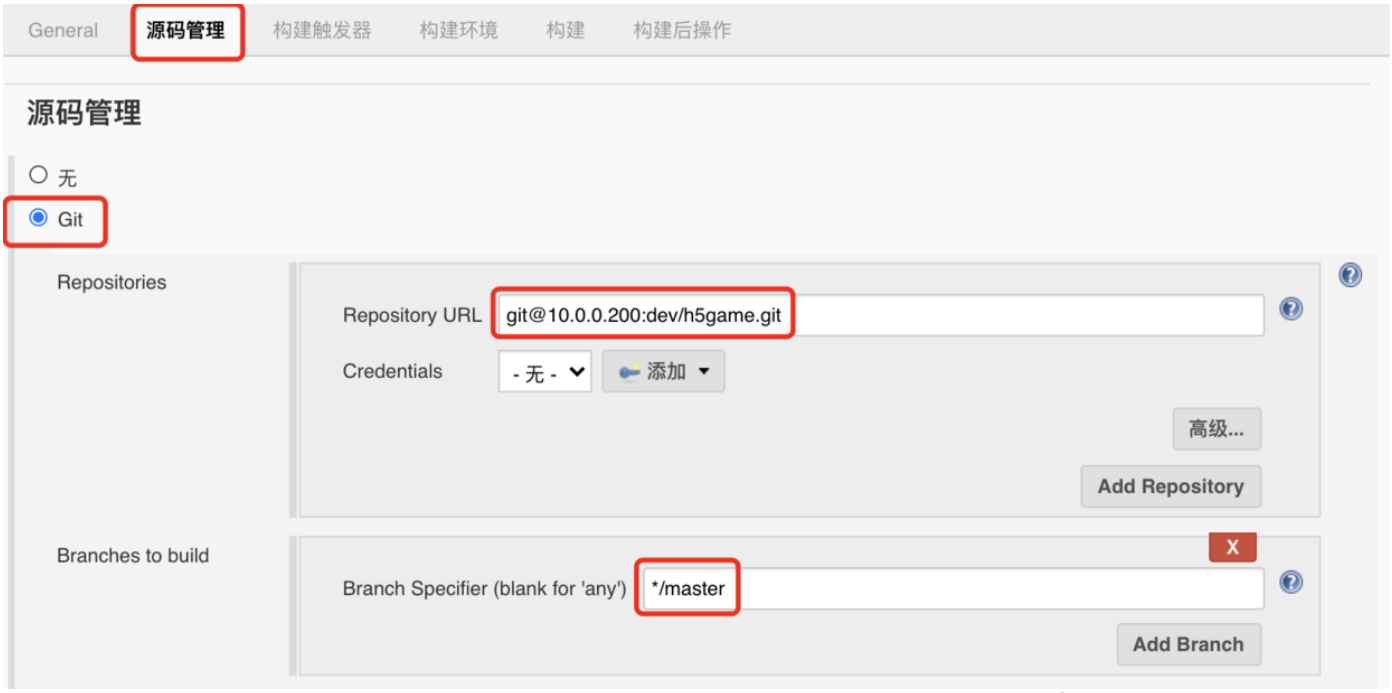
第8章 打标签工程

1.jenkins新建打标签工程

2.配置参数化构建



3.配置工程的源码管理



4.配置构建动作

配置命令：

```
git tag -a $git_newversion -m "$git_newversion 稳定版"
git push -u origin $git_newversion
```

5.发布新版本并提交到master

```
echo "v4.0" >> index.html
git add .
git commit -m "v4.0 稳定版"
git push -u origin v4.0
```

6.jenkins使用打标签工程来打标签

查看控制台输出发现提示没有权限:

我们需要在gitlab上把deploy key设置允许写入

然后重新构建

```

```

返回到deploy-rollback工程查看可以发现已经生成了新标签，然后我们开始构建：

在web服务器上查看可以发现新版本已经发布成功了：

```
[root@web ~]# ll /usr/share/nginx/
总用量 20
lrwxrwxrwx 1 root root    8 5月  13 16:09 html -> web-v4.0
drwxr-xr-x 8 root root 4096 5月  13 16:05 web-
drwxr-xr-x 8 root root 4096 5月  13 15:00 web-v1.0
drwxr-xr-x 8 root root 4096 5月  13 15:00 web-v2.0
drwxr-xr-x 8 root root 4096 5月  13 15:37 web-v3.0
drwxr-xr-x 8 root root 4096 5月  13 16:09 web-v4.0
```