jenkins持续集成pigx

Ver 1.0.0

PigX内部群资料

目录

写在前面

算了，每次都是写在前面，可是我已经没什么好写在前面的了额。

今天我们来讲一讲如何通过jenkins持续集成pigx。

基础环境准备

本篇文章是一篇实操性的文章，但是不会介绍持续集成的概念，想了解这些概念我推荐一下**毛子坤**的《k8s部署教程》，毛神的k8s教程大概是我研究k8s时上手最快的教程，他人很好，水平极高，这里给大家强势安利一波！

同样，本文不会再带着大家一步一步地从搭建jdk开始讲起，如果有需要，请移步我的其他文章。当你阅读这篇文章时，我默认你已经具备如下的环境，如果这些环境不具备，可以从我的另一篇《从零开始部署pigx》中得到他们的安装方法：

- jdk1.8
- maven3.5.4
- git最新版
- docker最新版
- docker-compose最新版
- npm最新版
- nginx最新版

安装jenkins

创建安装目录

```
sudo mkdir -p /opt/jenkins && cd /opt/jenkins
```

下载通用war包

```
wget https://mirrors.tuna.tsinghua.edu.cn/jenkins/war-stable/2.138.2/jenkins.war
```

后台运行jenkins

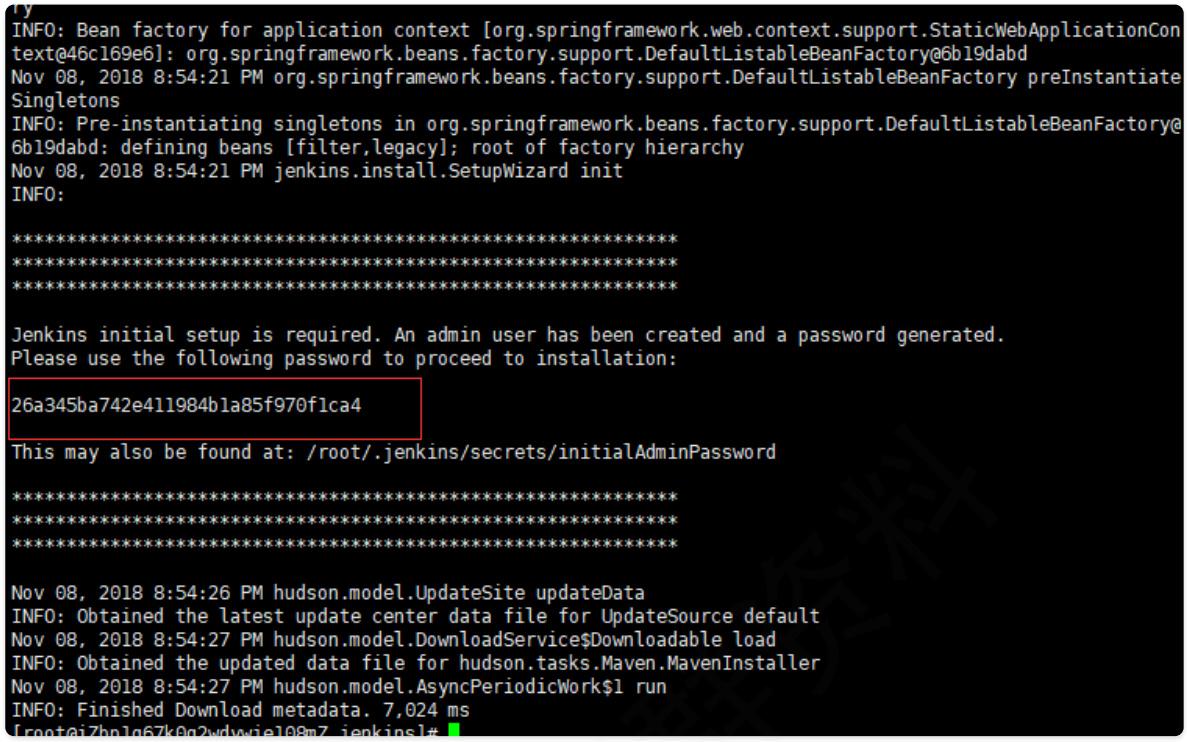
我这里的8085没啥特殊的意义，只是我喜欢，你可以换成任何一个你自己喜欢的端口号。

```
nohup java -jar jenkins.war --httpPort=8085 > temp.txt &
```

获取登录密码

```
cat  temp.txt
```

然后找到以下内容并复制：



访问jenkins

然后访问你虚拟机的8085端口，第一次运行应该看到这个界面：

将上图中复制的密码粘贴到输入框中，然后耐心等待其完成初始化工作即可。

安装插件并完成初始化

新手用户建议直接选择第一个：安装推荐的插件。

安装完成后会出现初始化用户界面，大家可以自己选择是否进行初始化。如果不想初始化新用户的话可以点击左下角的**使用admin账户继续**即可。

初始化的工作完成后就会进入jenkins的首页。

全局工具配置

本机的jdk安装路径可以通过命令:

如果在配置了jdk环境变量$JAVA_HOME的情况下，只需要一句命令:

```
echo  $JAVA_HOME
```

如果以上内容什么也没输出就证明你并没有配置环境变量**$JAVA_HOME**,那么也可以选择以下方法。

也可以通过以下命令来实现：

```
which  java
```

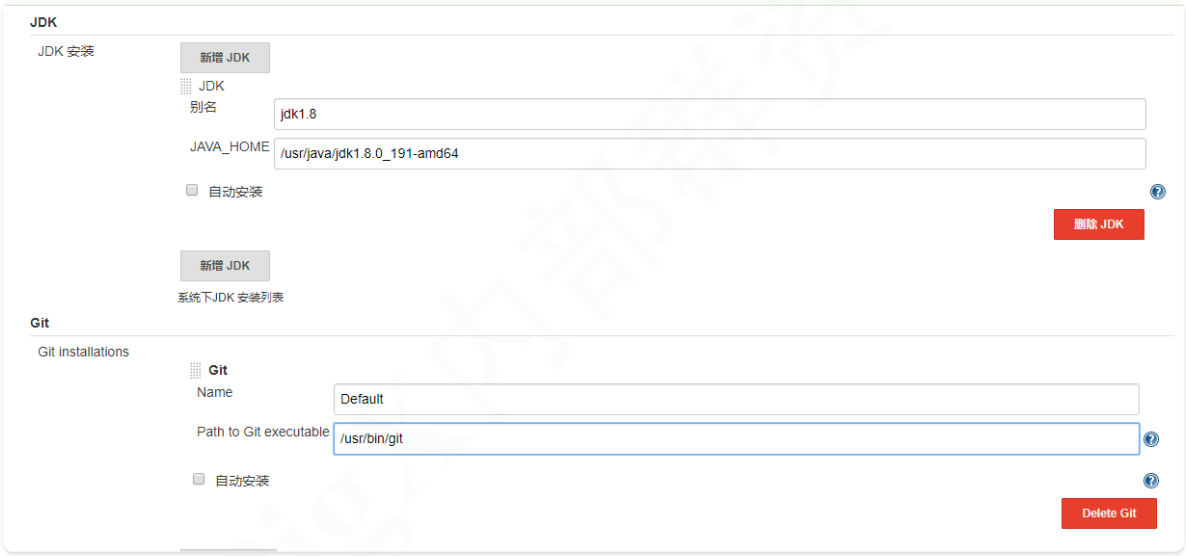
不过要注意，which java是定位不到安装路径的。which java定位到的是java程序的执行路径。
至于如何查找真实路径，可以通过**ls -lrt**来继续查找，整个过程大概如下：

```
[root@localhost]# java -version
java version "1.8.0_191"
Java(TM) SE Runtime Environment (build 1.8.0_191-b12)
Java HotSpot(TM) 64-Bit Server VM (build 25.191-b12, mixed mode)
[root@localhost]# which java
/usr/bin/java
[root@localhost]# ls -lrt /usr/bin/java
lrwxrwxrwx 1 root root 22 Oct 29 07:03 /usr/bin/java -> /etc/alternatives/jav
a
[root@localhost]# ls -lrt /etc/alternatives/java
lrwxrwxrwx 1 root root 41 Oct 29 07:03 /etc/alternatives/java -> /usr/java/jdl
.8.0_191-amd64/jre/bin/java
[root@localhost]#
```

whereis java也是如此，它本身不能定位到安装路径。可以通过上面例子去定位安装路径。

注意:
上图输出的是JRE的路径，因此它的JAVA_HOME应该是**/usr/java/jdk1.8.0_191-amd64**

git的话jenkins要求的便是git的可执行路径，因此直接通过**which git**进行获取即可。配置完Java和git环境后不要忘记点击apply然后save。



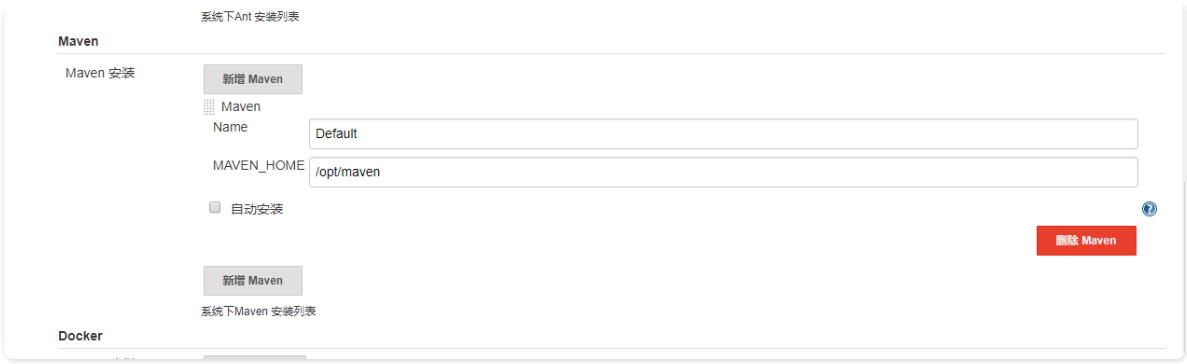
配置maven也是同理，通过：

```
which mvn
```

查找到maven可执行文件路径:

```
[root@localhost]# which mvn
/opt/maven/bin/mvn
```

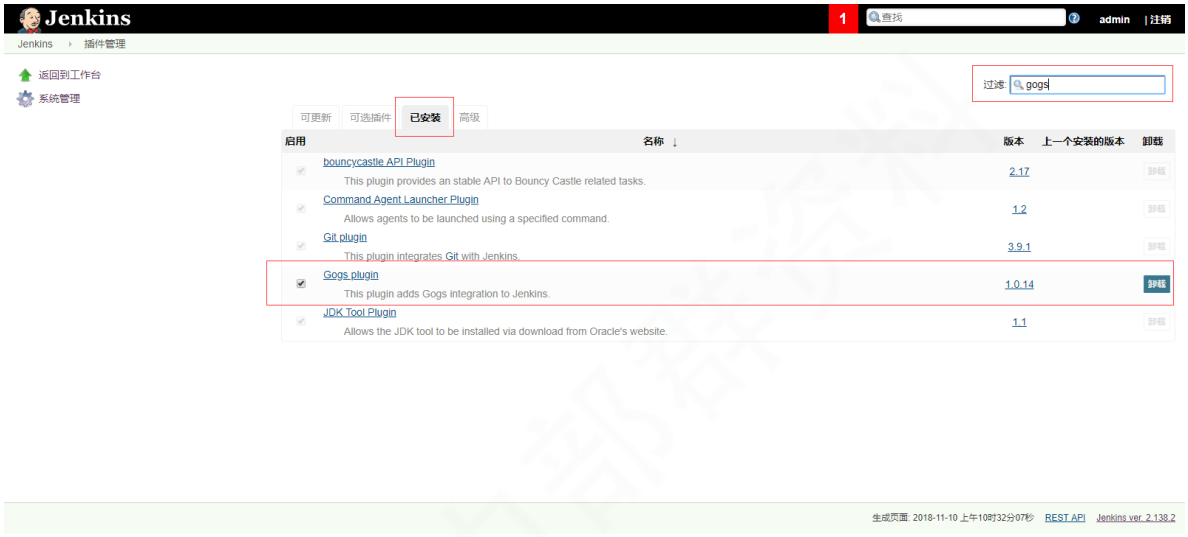
然后~~大肉凭~~经验判断它的**MAVEN_HOME**为**/opt/maven**



这样基本上全局的工具就配置好了。

配置jenkins的gogs插件

pigx项目托管在gogs上，而gogs已经有官方的gogs插件了。可以在**系统管理->插件管理->可选插件**中进行搜索，我因为已经安装过，所以可以在**已安装**这里选择出来



进行构建操作

编写构建pigx-ui的过程

任务配置

点击左侧菜单的**新建任务**按钮，然后直接选择第一个构建一个自由风格的软件项目并输入一个任务名称，然后点击确定按钮，因为我这里pigx-ui这个任务已经存在了，所以会报如下的警告。

源码管理

之后我们可以直接切换到**源码管理**选项卡或者下拉到源码管理菜单，填入以下参数：

- 源码管理类型选择**git**
- 仓库地址可以填入项目地址，没有意外的话，诸位应该没有往仓库里提交的权限，所以建议fork一份项目到自己本地，配置自己fork后的仓库内容，**Credentials**区域配置的是自己仓库的账号和密码，因为这个是一个私有仓库。如果配置过，直接以下拉的方式进行选择，如果没有配置过，点击边上的**add**按钮进行添加即可，添加的时候输入一个用户名、一个密码以及一个自己用于识别的描述就可以，id它是会自动进行识别的。
- git类型的版本控制工具支持选择分支，我这里构建master分支，所以填入*/master即可

构建触发器

触发的话支持远程触发，定时触发等等，我这里选择**Build when a change is pushed to Gogs**，就是有提交记录提交到gogs上时自动触发构建。当然，这个需要配置以下gogs的webhook，这个我们放在下面讲。

构建

pigx-ui的构建非常简单，我这里只通过一个**执行shell**的构建过程就能描述，因此只需要在**命令**区域填入以下内容即可：

```
cd ${WORKSPACE}
npm install --registry=https://registry.npm.taobao.org
npm run build
cd dist && cp -r * /opt/nginx/html/*
```

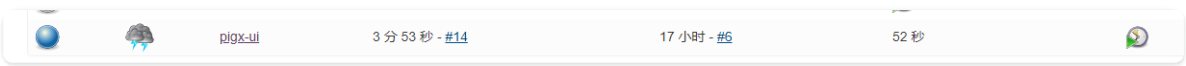
整体代码非常简单，首先进入jenkins的工作目录，**${WORKSPACE}** 是jenkins的一个环境变量，所有的环境变量可以通过 **查看 可用的环境变量列表**这个功能进行查找，
然后利用npm工具安装依赖，这个地方调用的就是本地shell里的命令，所以要求在进行整个构建配置之前必须要配置好这些必要的环境。安装完依赖之后打包，然后进入默认的打包生成的目录**dist**中将编译后的静态资源文件复制到nginx的静态资源目录中。

构建后操作

为了提高复用性，使每次的构建环境尽可能一致，我在构建后操作里选择一个**Delete workspace when build is done**,这样，在构建完成时，可以删掉工作空间，避免工作空间中已有的文件造成干扰。

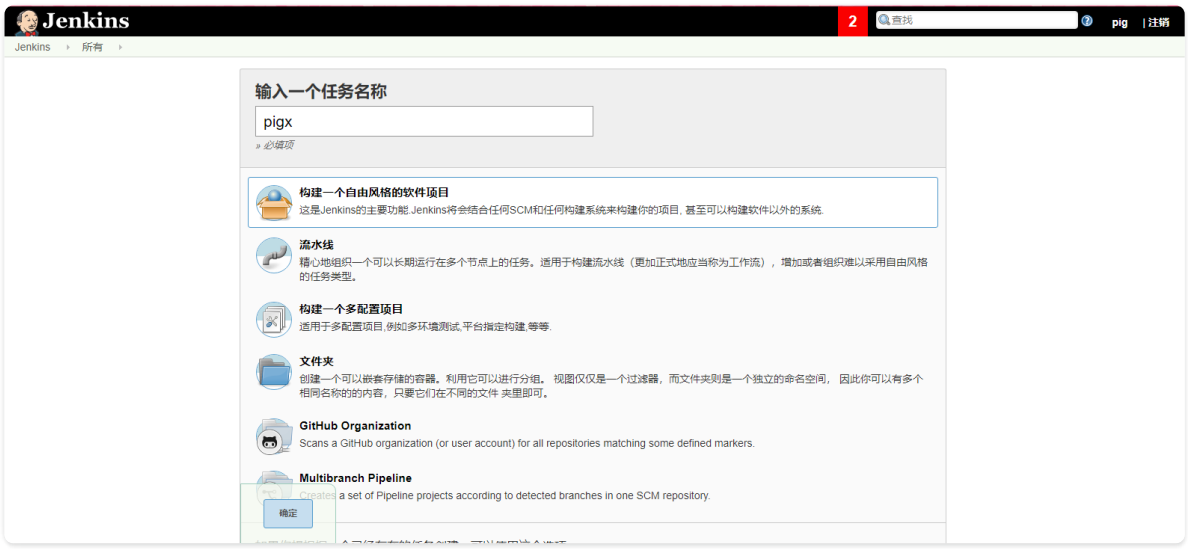
测试构建

在编写完成之后，我们可以点击构建测试一下脚本是否没有问题。
在任务面板选中这个任务之后，点击任务名后面的下拉框，选择**立即构建**，如果有问题,可以点击每个**任务id**后面的下拉框，选择控制台输出，输出的结果就是执行shell的结果，所以可以根据报错的结果调整自己的shell脚本。

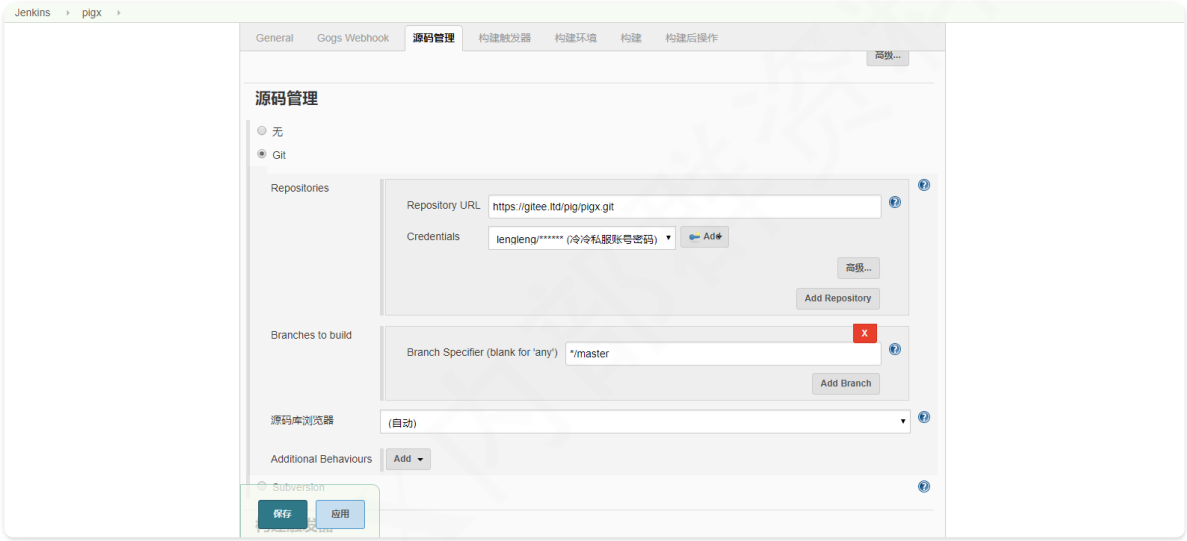


编写构建pigx的过程

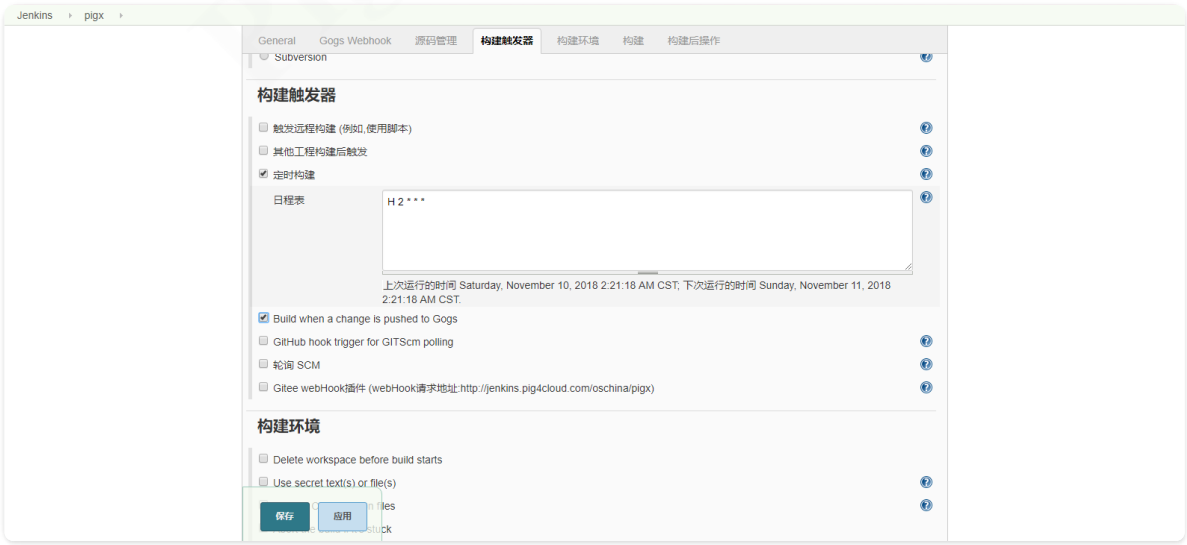
任务配置



源码管理



构建触发器



构建

后端的构建就比较复杂，一步完不成，需要增加一个调用maven构建步骤。

- 调用顶层maven目标

  增加一个调用顶层maven目标的构建步骤,maven版本选择配置好的**Default**。
  这句调用顶层maven目标的意思就是它会去帮你调用**mvn**命令，因此不需要你输入mvn。

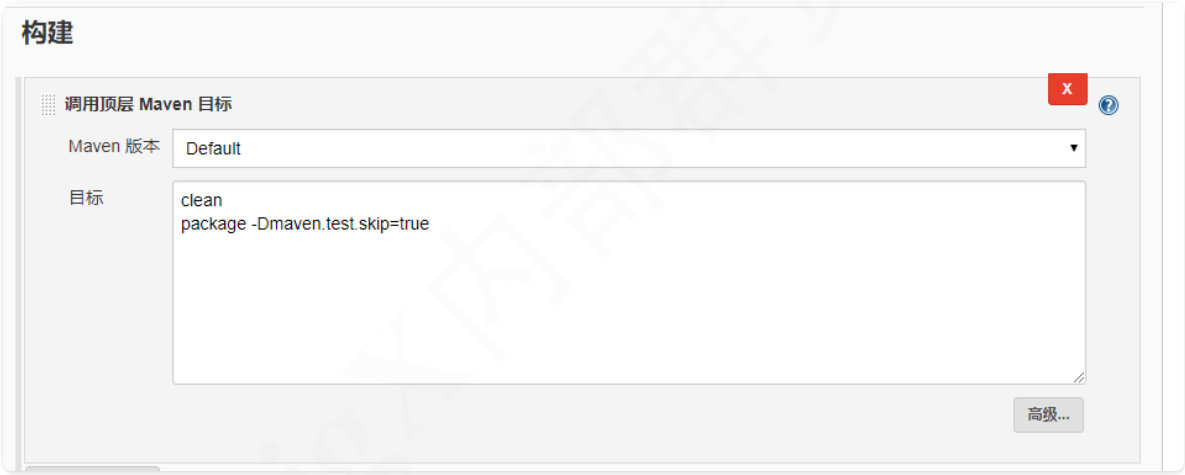
```
clean
package -Dmaven.test.skip=true
```

- 执行shell

```
cd ${WORKSPACE}
docker-compose down
docker system prune -f
docker-compose build
docker-compose up -d
```

很简单的几句话，都是docker-compose相关的，有docker-compose基础的可以很简单的就能看懂。

当然，也可以在shell里调用maven，那样仅需一步就能完成。

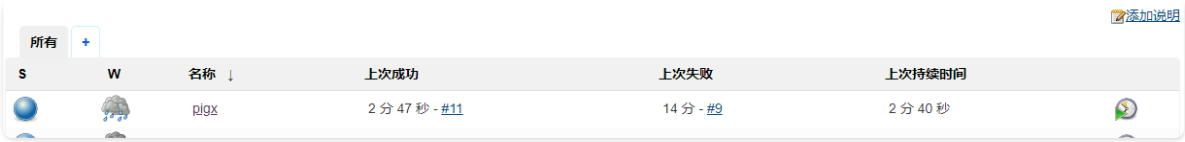


我这边测试就不执行了，虽然更好的方式的确是应该要去执行单元测试。毕竟，持续集成的意义也就是在于能够更早地发现问题。

构建后操作

同上。

测试构建



配置gogs的webhook

首先登录私服，然后选择你fork的项目，点击右上角**仓库设置**，选择**管理web钩子**，之后**添加web钩子**，选择**gogs**,需要配置的只有一个推送地址，格式如下：

```
http://jenkinsIP:jenkins端口/gogs-webhook/?job=jenkins中的任务名
```

以pigx-ui的环境为例，只需要填入

```
https://jenkins.pig4cloud.com/gogs-webhook/?job=pigx-ui
```

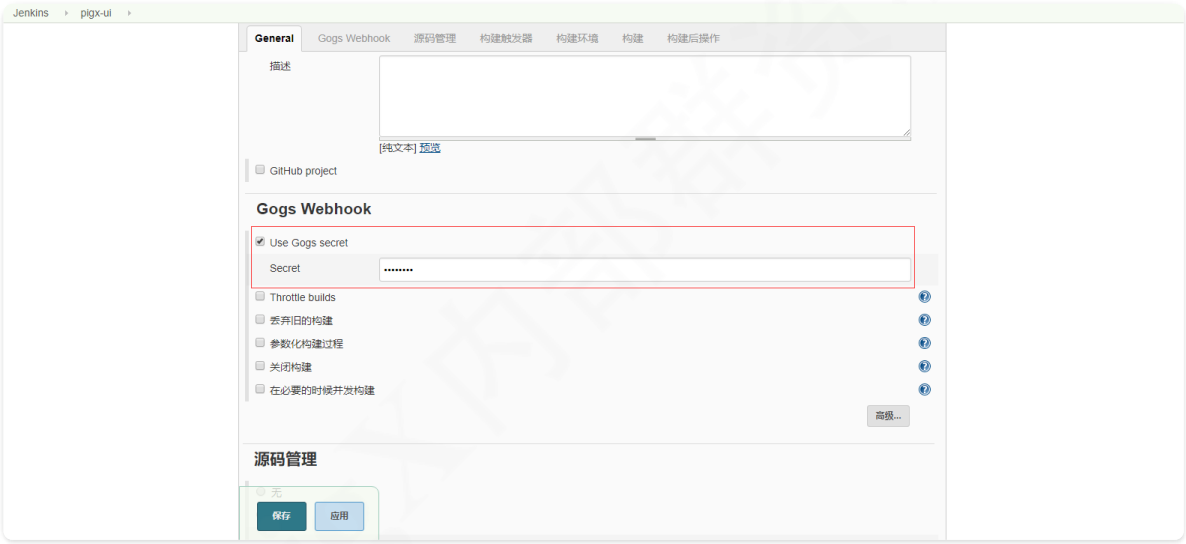
即可。

以pigx的环境为例，只需要填入

```
https://jenkins.pig4cloud.com/gogs-webhook/?job=pigx
```

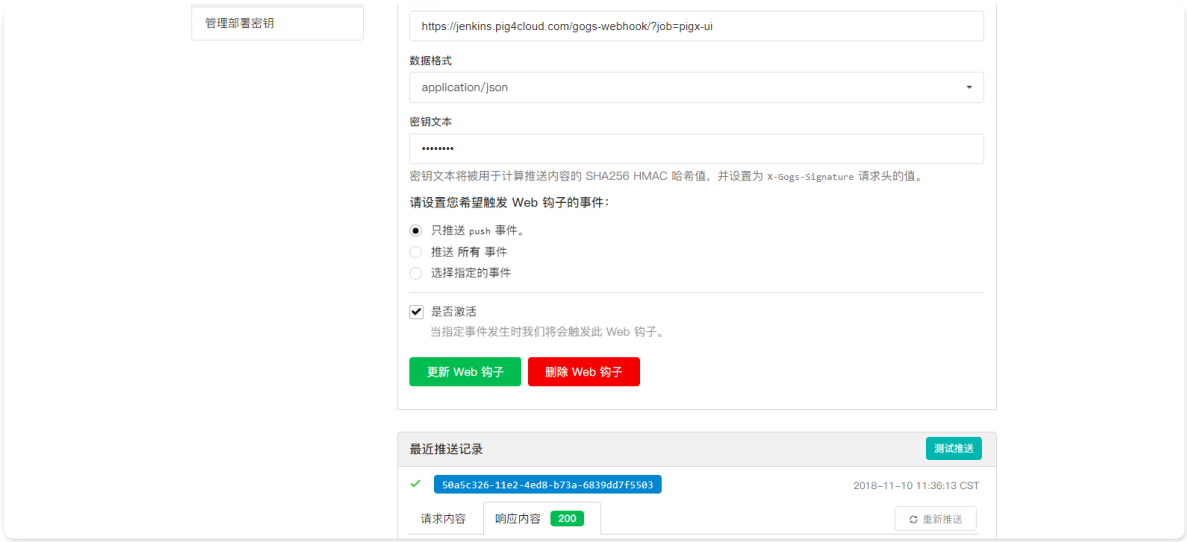
即可。

密钥文本的内容和jenkins中对应任务配置的secret一致。



之后可以点击下方的测试推送，如果没有问题，会显示：

```
{"result":"OK","message":"Job  '你配置的任务名'  is  executed"}
```



同时可以观察你的jenkins中是否已经开始构建作业。