nginx+uwsgi+supervisor在ubuntu部署flask项目

2020年11月18日　20:56

序

上周做了一个基础的flask web应用架子，但是只是我自己去把他写一个大概让他在本地运行并没有用处，后续的工作必然需要在服务器上部署且需要一个完整清晰的应用流程。
这块内容说起来似乎是个简单东西（也可能确实是个简单的东西，不过我太菜罢了= =）：

a. 项目内成员在本地完成自己任务内的调试，并push自己的功能代码到远程仓库（如github）。
b. 在累积了一定的更新后，后端维护人员将服务器上的git库通过git pull完成服务器库对远程库上最新项目的拉取。
c. 在测试环境下的完成项目测试。
d. 将项目切换至生产环境。

其中a步骤对于有过合作开发经验的人来说应该并不难理解，IDE集成的git功能并无需多少学习成本就能完成基本git操作。但是git操作中可能会遇到的问题也许还是需要针对git的一些深入的学习才能更好地理解解决。

对于c，d步骤，事实上我自己也没有很多经验，曾经维护管理学院的服务器的经历似乎并没有多少参考价值，因为开发环境和操作的标准程度也许并不适用于这次的项目。目前对它的认知停留在基本的文件配置与部署调试。

b步骤，看起来也就寥寥数语。实际操作起来以后就会发现这是一个和大学两年年来学习内容没什么关系的部分。至少在我解决问题的过程中还是补了挺多没学过的知识内容的。一周左右的时间做下来，查阅的资料还真挺多的。这篇笔记主要记录的也就是这一周对这块的学习。

概述

①购置腾讯云服务器 -> ②服务器基本配置 -> ③mysql安装与配置 -> ④flask虚拟环境搭建与配置 -> ⑤项目从本地到服务器的转移 -> ⑥配置uwsgi -> ⑦配置supervisor -> ⑧配置nginx

以上是完成目标计划的基本过程，下面讲讲涉及到的基本概念以及它们在项目中的作用。

服务器（server）

就是一台远程的电脑，由提供云服务的产商提供使用权。项目正式投入生产时需要部署到一台服务器上才能真正成为一个能够提供给客户服务的应用。

Mysql

目前最流行的关系式数据库管理系统。要完成某项服务，几乎必然要获取某些信息又或者是保存某些信息。数据库管理系统就是提供给我们信息管理服务的系统。而与平时课设或是实验不同的是，我们的数据库应该部署在服务器上而不是本地（废话）。

Flask

一个轻量级的Web应用程序框架。在这个框架中，对用户发送来的请求进行处理转发，并返回给用户他们所希望得到的内容。者提供服务的过程中它将与别的模块交换信息以完成任务。它如同一个空的屋子，人们来来往往拜访这个地方，而后面的主要工作也就是往这个屋子里添加东西，让这个屋子真正拥有它的功能。

uWSGI

一个Web服务器，它实现了WSGI协议、uwsgi协议、http协议等（WSGI让任意的web服务器能够与任意一个web框架搭配）。我们只需要关注它为我们提供的接受客户端请求并交给web框架再返回处理结果给服务端的功能就可

以了。（事实上我也还没有深入了解、、、）

Nginx

实现了反向代理的代理web服务器。其实它的主要是提供了高并发情况下需要多台web服务器一起工作的情况的解决方案。在目前的项目中我们并没有用到多台服务器的配置，但是它帮助我们隐藏了提供服务的源服务器地址（并没啥用。。。）。总之先做了。

Supervisor

用来管理守护进程。我也是为了解决问题而用的这个东西。目前它解决了uWSGI在系统管理用户退出后断开的问题。即用于创建uWSGI进程并将其收至后台守护其运行。

服务器的购买与配置

购买

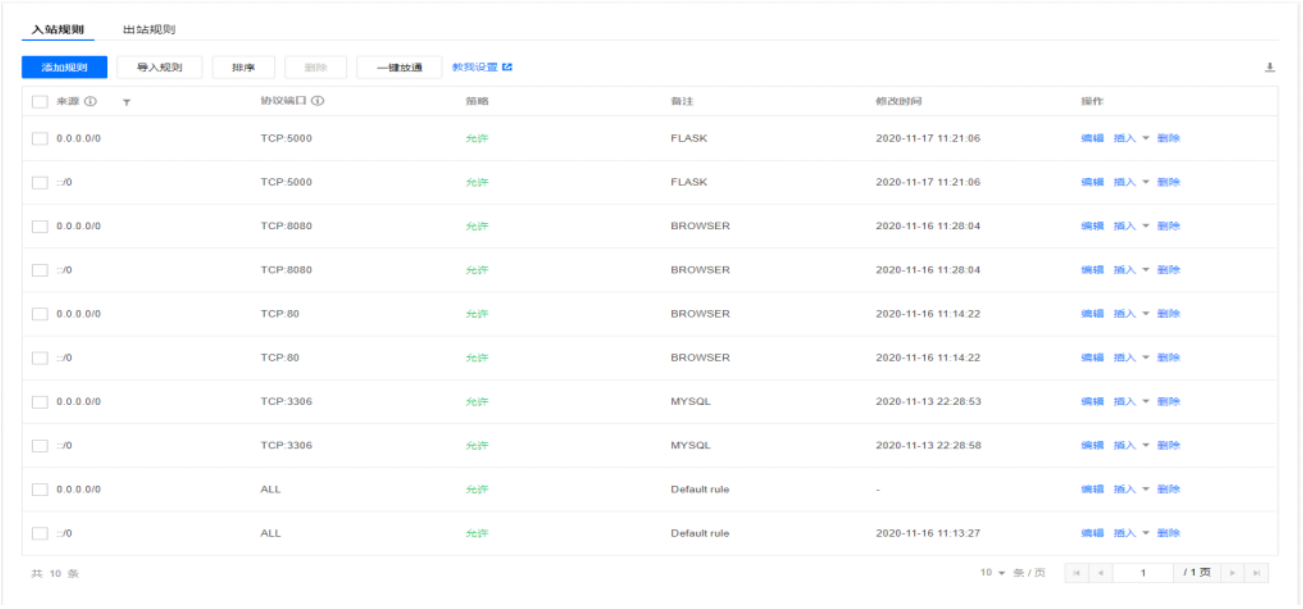
购买应该不用说吧。鹅厂腾讯云购买一条龙，在系统镜像上选择ubuntu镜像即可。

配置

在完成服务器的购买之后，因为我也是第一次用没有图形化界面的ubuntu，所以还是比较懵逼的。但是在真正进行进入其系统进行操作之前我们可以先完成服务器安全组的配置。

安全组

通俗理解，配置服务器对外界开放的端口。如果已经完成计算机网络这门课程的学习的话不难理解，一个程序在运行的时候如果需要用到网络模块是需要占用端口的。服务器上同理，如果希望从外界访问服务器提供的某个服务也就需要完成对服务器的某个端口的访问。那么要让客户能够完成这一访问操作，服务器需要对外开放程序所需要的端口来完成这一过程。



注意入站规则与出站规则都需要完成配置。在配置完成后可以简单的通过cmd控制台telnet指令来验证是否能够连接到服务器目标端口。

```
C:\Users\VincentZzZ>telnet 49.235.73.129 22
```

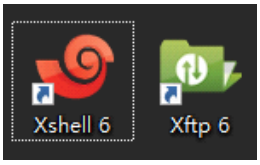
```
SSH-2.0-OpenSSH_7.6p1 Ubuntu-4ubuntu0.3
```

确认22端口已经连通了以后就可以远程连接到服务器系统进行操作了。

xshell与xftp

在此之前，我维护的服务器的系统镜像是windows的镜像，提供了可视化的界面，通过mstsc可以直接连接访问，这次情况则大不相同，需要借助其他软件来完成对服务器的连接，这里我才用xshell与xftp完成对服务器的控制

xshel提供了远程连接服务器的服务，xftpll提供了远程连接服务器的服务，xftp提供了远程文件传输到服务器的服务。



具体的下载安装连接就不展开讲了。

经由xshell完成连接了以后就进入到了服务器上ubuntu的系统。

```
Connecting to 49.235.73.129:22...
Connection established.
```

在配置完上面说的安全组后可能会出现访问某些端口如80 8080 5000等端口仍然失败的情况，这里大概率是因为服务器上的防火墙在工作的原因导致端口被屏蔽。这里就需要在服务器上额外对防火墙进行设置去开启对应端口。

链接：https://www.cnblogs.com/superming/p/11315634.html

如此一来服务器的基本配置就完成了。其他基本的服务器的必须额外包我也记不清都需要装些啥了，具体报错的时候再具体百度吧。

Mysql

安装与配置。

链接：https://yq.aliyun.com/articles/758177

在完成了这一步的操作以后服务器就成功具有了mysql提供的服务，但是仍然无法通过外界完成对服务器mysql的访问，这是因为我们并没有建立一个接受外界连接数据库的用户，在完成这个用户的建立以后我们就能够从任意终端访问并操作服务端数据库给予的在用户权限内的功能。

在上面给出的链接中的最后一步，

```
GRANT ALL PRIVILEGES ON *.* TO 'administrator'@'localhost' IDENTIFIED BY 'very_strong_passw
```

我们需要将localhost改成%，代表任意终端的用户都有登录许可。然后再进行对这个用户的密码修改操作即可完成我们的目标。

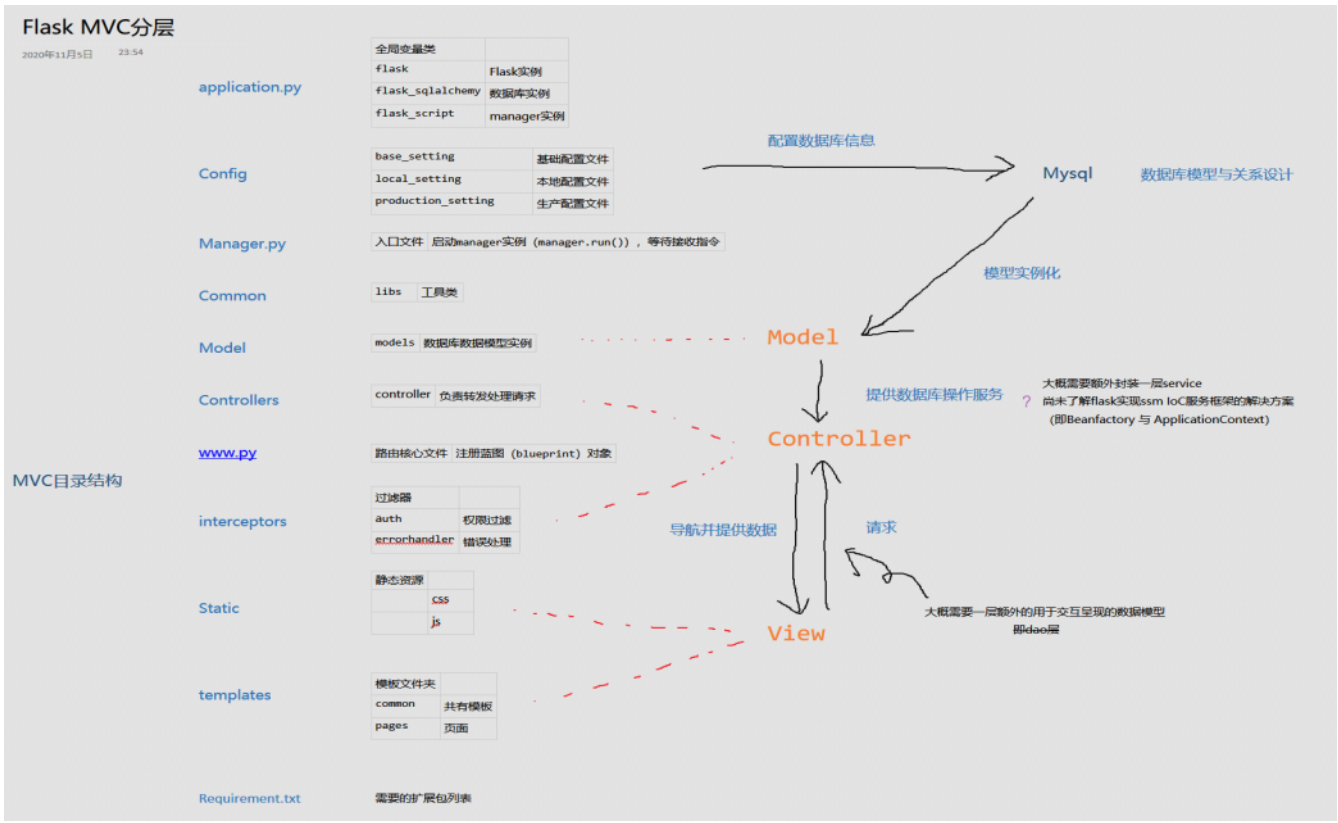
链接：https://blog.csdn.net/skh2015java/article/details/80156278

在完成所有操作后，我们可以在本地的mysql图形化应用中尝试对服务器的mysql进行连接登录。这里我用的是SQLyog。

在这过程中也许你会遇到各式各样的报错，但请不要气馁，逐个百度寻找答案，最后一般都能得到解决。登录成功后，我们就完成了mysql的基本配置。

Flask

这块也就是具体应用的编写，具体可以参考我上周的flask笔记，那篇笔记完整介绍基本项目的结构。这里给出github链接。



链接：https://github.com/vanot313/flask.demo

虚拟环境搭建

要在服务器上跑python工程，那必须要有python的虚拟机环境。这里涉及到的python环境的概念可以参考暑假里那份关于anaconda的笔记，里面有对这方面比较详细的解读。具体的配置搭建过程因为实操的时候没有截图记录。。。所以也不大好重现。。

大致步骤： apt-get安装python -> pip安装virtualenv -> 通过virtualenv创建一个虚拟环境 -> 通过指令进入虚拟环境。

将flask项目转移到服务器

在服务器系统上新建一个文件夹 -> 通过xftp将flask项目传输到目标文件夹 -> 进入先前创建虚拟环境 -> 通过执行指令将项目中requirement.txt中的需要的包安装到虚拟环境

如此一来项目便成功转移了。

当然更合理的做法是通过git的方式直接从github将项目clone到目标文件夹。

在服务器系统上新建一个文件夹 -> 初始化git文件夹 -> 通过git clone指令将项目下载到当前文件夹 -> 日后项目在github更新 -> 进行git pull更新项目

在命令行运行指令 python manager.py runserver （就像我们在pycharm中做的一样），得到成功在0.0.0.0:5000开放的反馈后，在浏览器上访问xx.xx.xx.xx:5000，将可以看到和你在本地跑出来的一致的结果。当然前提是完成了本地与服务器mysql同步的配置。

uWSGI

前面已经介绍了uWSGI的基本定义与功能，这里为了运作流程更加清晰，这里给出简单的uWSGI+nginx工作过程。

//图是网上找的，把Django换成Flask就行。。。

中间的箭头上的东西代表了它们之间采用遵守的协议。其实这三者之间只需要完成两两之间的两个配置文件的配置就可以完成它们之间的连接。

其中uWSGI与flask连接的配置文件可以直接生成在项目根目录下。

..					
common		文件夹	2020/11/18, 19:51	drwxr-xr-x	ubuntu
config		文件夹	2020/11/18, 19:51	drwxr-xr-x	ubuntu
controllers		文件夹	2020/11/18, 19:51	drwxr-xr-x	ubuntu
interceptors		文件夹	2020/11/18, 19:51	drwxr-xr-x	ubuntu
templates		文件夹	2020/11/18, 19:51	drwxr-xr-x	ubuntu
application.py	855 Bytes	Python File	2020/11/17, 19:33	-rw-r--r--	ubuntu
application.pyc	972 Bytes	Compiled...	2020/11/17, 19:34	-rw-r--r--	ubuntu
flask.conf	446 Bytes	CONF 文件	2020/11/17, 23:10	-rw-rw-r--	ubuntu
manager.py	628 Bytes	Python File	2020/11/18, 19:54	-rw-rw-r--	ubuntu
manager.pyc	1014 Byt...	Compiled...	2020/11/17, 22:29	-rw-r--r--	ubuntu
requirement.txt	66 Bytes	文本文档	2020/11/17, 19:23	-rw-r--r--	ubuntu
uwsgi_config.ini	283 Bytes	配置设置	2020/11/18, 16:00	-rw-rw-r--	ubuntu
www.py	355 Bytes	Python File	2020/11/17, 19:33	-rw-r--r--	ubuntu
www.pyc	507 Bytes	Compiled...	2020/11/17, 19:42	-rw-r--r--	root
__init__.py	0 Bytes	Python File	2020/11/18, 19:51	-rw-rw-r--	ubuntu

图中uwsgi_config.ini就是配置文件，具体内容为

```
[uwsgi]
socket=127.0.0.1:5000

home=/home/project/venv
#pythonpath = /home/project/venv/bin/python3

chdir=/home/vanot313/repositories/flask.demo
wsgi-file=manager.py
#module=manager
callable=app

py-autoreload=1
processes=4
threads=2
master=True
buffer-size=32768
```

- socket：这个参数可以是http或socket-http，改为后两者的话在uWGSI-flask二者直接向外界服务的情况下采用，因为socket是nginx与uWGSI连接时采用的协议，而在直接与外界连接的情况下则需要采用http协议。后面的参数代表着其对外开放的地址与端口。在socket的情况下如果在本地直接与nginx连接则直接设置为127.0.0.1，否则需要与外界连接的情况下采用0.0.0.0。
- home：虚拟环境目录
- chdir：项目目录
- wsgi-file：flask程序启动文件
- callable：flask特有，程序内application变量名
- 别的目前看来无关紧要，复制粘贴即可

配置完成后，可以通过 uwsgi --ini uwsgi.ini 启动服务了，在得到正确的反馈后（这期间可能会处理若干bug），同之前一样在浏览器中输入ip与port即可访问到项目。（注意：此时应直接采用http协议，因为没有经过nginx）

Supervisor

正如前面对他的介绍，其提供的功能可以简单理解为将uWSGI服务器放到后台去运作。

其默认配置文件在/etc/supervisro/supervisord.conf, 自己开发可以将配置文件写在 /etc/supervisor/conf.d/目录下，文件扩展名必须为*.conf。具体内容为：

```
[program:uwsgi]
command=/home/project/venv/bin/uwsgi /home/vanot313/repositories/flask.demo/uwsgi_config.ini
user=root
autostart=true
autorestart=true
stdout_logfile=/home/log/uwsgi_supervisor.log
stderr_logfile=/home/log/uwsgi_supervisor_err.log
```

- command 程序启动命令如: /usr/bin/python - app.py
- user 进程运行的用户身份
- autostart=true 跟随Supervisor一起启动
- autorestart=true 挂掉之后自动重启
- stderr_*logfile, stdout*_logfile 标准输出，错误日志文件

配置完成后，启动supervisor

```
supervisord -c /etc/supervisord.conf
```

若没有报错，则启动成功，进一步查看进程状态。

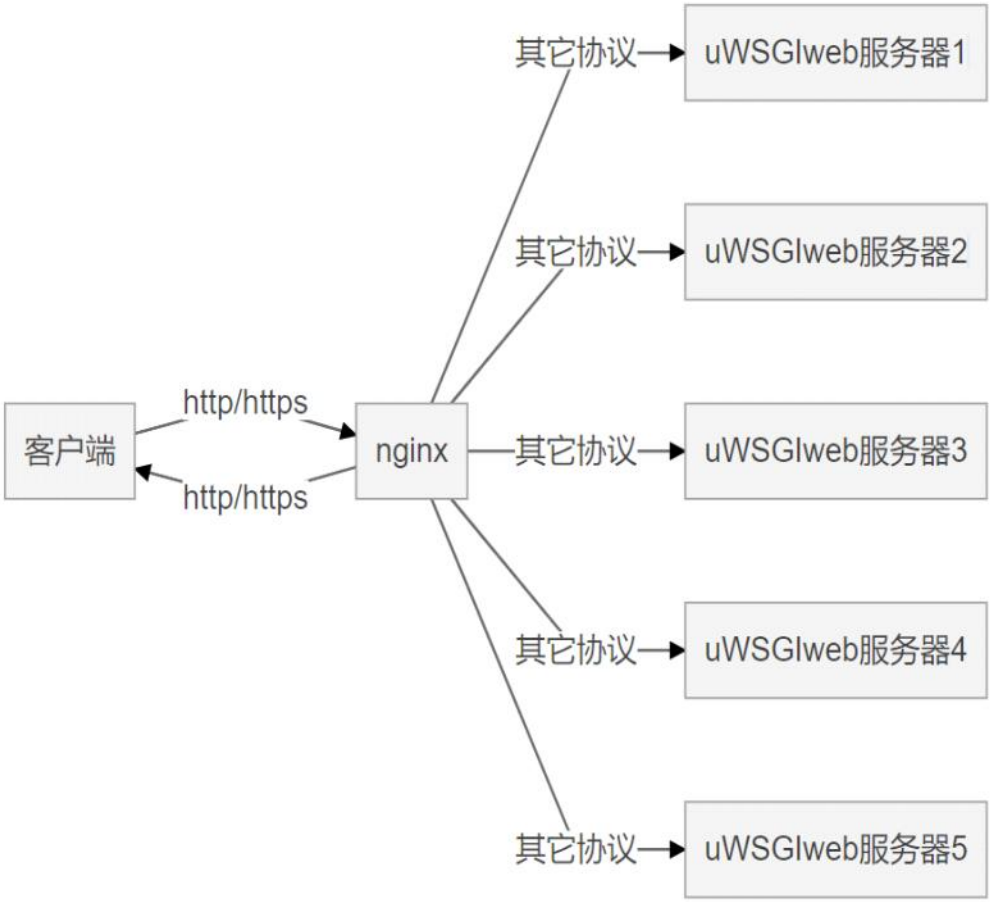
```
supervisorctl status
```

若得到，

进程正在运行的反馈，那么代表后台进程已经创建。如果你的uWSGI配置文件采用的协议仍然是http协议，那么你可以通过浏览器ip:port的形式浏览到项目。若是已经将配置变更成socket，则需要紧接着将nginx配置完成才能到达下一个里程碑。

Nginx

在真正用到它的时候，它的功能应当是类似于这样的。负责反向代理各种web server并有条不紊地向客户端回馈回答。



但是事实上在目前地项目中我们只配置了一个uWSGI web服务器。因为并没有开始处理高并发的情况。

那么剩下的问题就只是配置他与目标uWSGI（也是唯一一个，就是前不久配置的那个）的配置文件。

因为它的配置文件中自动include了sites-enabled 与 conf.d文件夹中的配置文件，因此我们只需要将配置文件塞进这两个文件夹中的任意一个就能达到我们目标的效果，但是需要注意的是若要放入conf.d文件夹需要带有.conf后缀。

```
include /etc/nginx/conf.d/*.conf;
include /etc/nginx/sites-enabled/*;
```

（主配置文件内的描述）

具体配置文件内容为：

```
    server {
            listen 80;
            server_name 127.0.0.1;
```

```
        location / {
              include      uwsgi_params;
              uwsgi_pass   127.0.0.1:5000;
              uwsgi_param  UWSGI_PYHOME /home/project/venv;
              uwsgi_param  UWSGI_CHDIR /home/vanot313/repositories/flask.demo;
              uwsgi_param  UWSGI_SCRIPT manager:app;
              uwsgi_read_timeout 300;
              uwsgi_connect_timeout 300;
              uwsgi_send_timeout 300;
        }
    }
```

- 外层的server代表了这是一个对web server 的配置文件
- Listen&server_name：确定了监听的端口
- 内层的location配置对uWSGI的映射
- uwsgi_pass：这里我们先前的uWSGI配置文件中socket配置的内容应当被放入这里，这个参数是它们二者连接的桥梁，但是它们有其他连接方式即通过sock文件连接
- UWSGI_PYHOME：python虚拟环境目录
- UWSGI_CHDIR：项目目录
- UWSGI_SCRIPI：application入口
- 别的无关紧要，照抄

//是不是有种似曾相识的感觉，没错和uWSGI的配置文件相差无几。。。

完成配置后，重启nginx服务。

```
sudo service nginx restart
```

如果没有报错，就可以进入最终的测试了。

因为nginx监听的是80端口，所以我们只需要在浏览器上直接访问ip，就可以访问到项目反馈给我们的页面。不难理解，经由nginx的反向代理，我们项目本身的端口5000被隐藏，外界访问的ip端口统一为nginx监听的ip端口。

末

事实上在这一整个过程中我踩了无数个坑。。。但是在这个过程中我建立起了更为完整的web体系知识。虽然投入的时间巨大（作业实验明天马上补。。。），但也还算有收获。这篇笔记以更接近博客教程的形式写成（以非常非常小白的视角入手，就如同刚刚开始搞的我。。。），参考了大量网络上的博客资料（之后再一起整理）。一来供自己日后回顾，二来也可以供想做这块的人参考。网络上的教程参差不齐，踩坑的感觉真是非常不好受。另外之后再把webhook自动部署给处理了，就要开始真正的功能搭建了。就先酱。