Flask多线程实现技术方法

# Flask+uWSGI

## 框架结构

### Flask

Flask是一个使用[Python](https://zh.wikipedia.org/wiki/Python)编写的轻量级[Web应用框架](https://zh.wikipedia.org/wiki/Web%E5%BA%94%E7%94%A8%E6%A1%86%E6%9E%B6)。Flask是单线程，一般外面挂gunicorn或者uwsgi实现多线程并发。

### uWSGI

Flask是 Web 框架，并不是 Web 服务器，一般只能用于开发，不能用于生产，对于 Web 服务器，我们有更专业的选择，那就是 uWSGI, uWSGI 是一个全站式的托管服务，它实现了应用服务器（支持多种编程语言）、代理、进程管理器、监视器。取名为 uWSGI 是因为它最早实现的是 Python 语言的 WSGI。

## 环境搭建

### 安装Flask

安装Flask前，首先确认环境中已经有Python2.X版本；

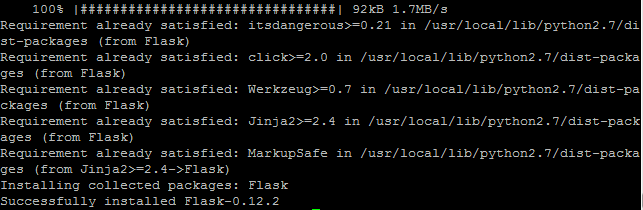
1. 安装pip工具，在docker容器中输入：apt-get install python-pip

C:\Users\ligang\AppData\Roaming\JunDaoIM\tempImages\image_0c.png

1. 安装Flask，在docker容器中输入：pip install Flask

C:\Users\ligang\AppData\Roaming\JunDaoIM\tempImages\image_PE.png

界面上显示以下提示，则安装成功。



### 安装uWSGI

安装uWSGI，在docker容器中输入：pip install uWSGI

C:\Users\ligang\AppData\Roaming\JunDaoIM\tempImages\image_o4.png

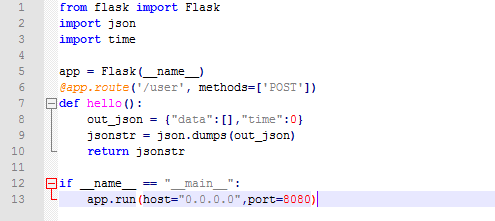
界面上显示以下提示，则安装成功。



## 服务搭建

### Flask服务

app.py文件内容



### uWSGI配置

uWSGI启动可以为参数启动，也可以将参数写才ini文件中启动。本方案只介绍由ini启动方法。uwsgi.ini文件配置如下：



http ：对外使用地址及端口号。

wsgi-file：为flask的启动程序文件

callable ：启动程序文件中application变量名。在此项目中为app = Flask(\_\_name\_\_)的app。

precesses：进程数

threads ：线程数

### 运行

在docker容器中运行：uwsgi --ini /path/to/ini/uwsgi.ini

C:\Users\ligang\AppData\Roaming\JunDaoIM\tempImages\image_mO.png

界面上显示以下提示，则运行成功。

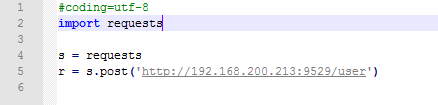
C:\Users\ligang\AppData\Roaming\JunDaoIM\tempImages\image_lC.png

启动 ：uwsgi --ini /path/to/uwsgi.ini

停止 ：pkill -f -9 uwsgi

### 测试

在服务器所在的局域网内，运行以下脚本post.py，脚本内容：



在putty中输入：Python post.py

C:\Users\ligang\AppData\Roaming\JunDaoIM\tempImages\image_a3.png

返回以下结果：其中{u’data’:[],u’time’:0}为服务返回结果。

C:\Users\ligang\AppData\Roaming\JunDaoIM\tempImages\image_4K.png

注：i、服务所在主机的地址为：192.168.200.213，所以post.py中发送地址为：192.168.200.213。

ii、uwsgi.ini中http参数所对应的端口号为8080，服务所运行的docker容器的8080端口映射到主机的9529端口，所以post.py中发送的端口号为：9529。

iii、如果服务所运行的容器没有和主机的端口映射，那么要运行post.py文件，必须进入到服务所在的容器中运行：python post.py，并将post.py文件中的192.186.200.213:9529修改为0.0.0.0:8080即可。

# 睿课项目多进程&多线程测试

本试验过程全部在深度学习框架使用GPU环境下测试。

## 多进程测试

uWSGI设置多进程情况下，服务不能正常预测。

## 多线程测试

### Theano

当设置多线程时，只有一个线程可以正常预测，其它线程不能正常预测。

### Tensorflow

单站点：正常预测；

多站点：正常预测；（注意：在服务脚本中必须设置 os.environ['CUDA\_LAUNCH\_BLOCKING'] = '1'，否则不能正常预测）

### Caffe

单站点：正常预测；

多站点：正常预测。

## 多线程时间测试

i、由于Theano框架不能实现多线程预测，所以所测试服务中仅包含Tensorflow和Caffe框架，不包括theano框架。

ii、本段所输出的时间为连续预测100次一张图片（大小：1920\*1080，人脸个数：15）所需要的时间和。

### 单站点

|  |  |
| --- | --- |
| 线程数 | 时间（s） |
| 1 | 64.5 |
| 2 | 70.1 |
| 4 | 70.2 |
| 6 | 69.7 |
| 8 | 69.9 |
| 16 | 68.5 |

### 多站点(5个站点)

|  |  |
| --- | --- |
| 线程数 | 时间（s） |
| 1 | 309 |
| 2 | 210 |
| 4 | 170 |
| 6 | 168 |
| 8 | 166 |
| 16 | 168 |

### 结论

单站点：多线程对其耗时影响较少，反而单线程所耗时间最少；

多站点：多线程对其耗时影响较大。

在线程数小于站点数时，耗时随线程数增加而减小；

在线程数大于等于站点数时，耗时无明显变化。

# 一级标题三

## 二级标题1

### 三级标题

正文（四号宋体）

### 三级标题

正文（四号宋体）

## 二级标题2

### 三级标题

正文（四号宋体）

### 三级标题

正文（四号宋体）

## 二级标题3

### 三级标题

正文（四号宋体）

### 三级标题

正文（四号宋体）