> 同济大学数据科学导论实验报告03

《Flask框架的网站实现》

> - 学院：电子信息工程学院

> - 专业：数据科学与大数据专业
> - 学号：1851804
> - 姓名：苗成林
> - 指导教师：郭玉臣
> - 时间：2021.09.27

实验目的及要求：

1. 了解万维网结构
2. 搭建实验的 Python 软件环境
3. 建设后继实验网站

实验原理：

Web

world wide web：全球广域网络，也称为万维网，是一种基于超文本和HTTP的全球性的、动态交互的、跨平台的分布式图形信息系统，是建立在internet上的一种网络服务。

web软件，是一种基于web为其数据交互基础的计算机软件

该类型软件，通过web万维网的数据交互协议，通过网络进行数据传输

主要目的是高效率的跨平台跨地区数据共享

Flask

Flask是一个基于Python并且依赖于Jinja2模板引擎和Werkzeug WSGI 服务的一个微型框架
WSGI： Web Server Gateway Interface(WEB服务网关接口)，定义了使用python编写的web app与web server之间接口格式

2) Flask 的框架模式 - MTV
经典三层结构： MVC模式
M： Models，模型层，负责数据库建模
V： Views，视图层，用于处理用户显示的内容，如 :html
C： Controller，控制器，处理与用户交互的部分内容。处理用户的请求并给出响应
python常用：MTV模式
M： Models，模型层，负责数据库建模
T： Templates，模板层，用于处理用户显示的内容，如：html
V： Views，视图层，处理与用户交互的部分内容。处理用户的请求并给出响应

创建实验环境：

打开CMD，输入命令：`pip3 install Flask`

```
(base) C:\Users\Administrator\Desktop\数据采集与集成\Data-Acquisition\HW1>pip install flask
WARNING: Ignoring invalid distribution -ip (f:\anacondasetup\lib\site-packages)
WARNING: Ignoring invalid distribution -ip (f:\anacondasetup\lib\site-packages)
Looking in indexes: https://pypi.tuna.tsinghua.edu.cn/simple
Requirement already satisfied: flask in f:\anacondasetup\lib\site-packages (1.1.2)
Requirement already satisfied: Jinja2>=2.10.1 in f:\anacondasetup\lib\site-packages (from flask) (2.11.2)
Requirement already satisfied: itsdangerous>=0.24 in f:\anacondasetup\lib\site-packages (from flask) (1.1.0)
Requirement already satisfied: Werkzeug>=0.15 in f:\anacondasetup\lib\site-packages (from flask) (1.0.1)
Requirement already satisfied: click>=5.1 in f:\anacondasetup\lib\site-packages (from flask) (7.1.2)
Requirement already satisfied: MarkupSafe>=0.23 in f:\anacondasetup\lib\site-packages (from Jinja2>=2.10.1->flask) (1.1.1)
WARNING: Ignoring invalid distribution -ip (f:\anacondasetup\lib\site-packages)
WARNING: Ignoring invalid distribution -ip (f:\anacondasetup\lib\site-packages)
WARNING: Ignoring invalid distribution -ip (f:\anacondasetup\lib\site-packages)
WARNING: Ignoring invalid distribution -ip (f:\anacondasetup\lib\site-packages)

(base) C:\Users\Administrator\Desktop\数据采集与集成\Data-Acquisition\HW1>
```

Git上传实验结果到远程仓库

```
(base) C:\Users\Administrator\Desktop\数据采集与集成\Data-Acquisition\HW1>git push
Enumerating objects: 24, done.
Counting objects: 100% (24/24), done.
Delta compression using up to 12 threads
Compressing objects: 100% (16/16), done.
Writing objects: 100% (20/20), 121.55 KiB | 5.06 MiB/s, done.
Total 20 (delta 1), reused 0 (delta 0), pack-reused 0
remote: Resolving deltas: 100% (1/1), completed with 1 local object.
To https://github.com/MiaoChenglin125/Data-Acquisition.git
   2246e1b..886cf95  master -> master
```

实验步骤：

搭建路由和路径间的映射关系

```
''''app.url_map，返回的是app装饰的所有路由和路径之间的映射关系

注意点：只有被app.url_map包含进来的路由（地址）才能被访问

'/'为地址（路由）

GET为请求方式，访问视图函数

index视图函数'''''

@app.route('/')
def index():
    return "this is index"

@app.route('/index2')
def index2():
    return "this is index2"

Map([<Rule '/index2' (HEAD, GET, OPTIONS) -> index2>,
 <Rule '/' (HEAD, GET, OPTIONS) -> index>,
 <Rule '/static/<filename>' (HEAD, GET, OPTIONS) -> static>])
```

app应用运行参数解析

```
"""参数1：host，默认值是127.0.0.1
参数2：port，默认值是5000
参数3：debug，默认值是False"""""

if __name__=='__main__':
    print(app.url_map)
    app.run()
```

路由参数解析

```
"""在访问路由时指定参数
格式：@app.route("/<变量名：类型名>")
常见参数类型
整数int
小数float
字符串path（default）"""""

@app.route('/change<int:age>')
def index2(age):
    return "this is %s"%age
```

Pandas读取txt

```
import pandas as pd     #引入pandas包
citys=pd.read_table('book.txt',sep='\t',encoding='utf-8')     #读入txt文件，分隔符为\t
citys=citys[:-1]

citys.columns=['序号']
citys['书名']=None
citys['分类']=None
for i in range(len(citys)):          #遍历每一行
    coordinate = citys['序号'][i].split() #分开第i行，x列的数据。split()默认是以空格等符号来分割，返回一个列表
    citys['序号'][i]=coordinate[0]         #分割形成的列表第一个数据给x列
    citys['书名'][i]=coordinate[1]         #分割形成的列表第二个数据给y列
    citys['分类'][i] = coordinate[2]  # 分割形成的列表第二个数据给y列

target=citys.to_html(index=None)
#print(target)
```

HTML表格

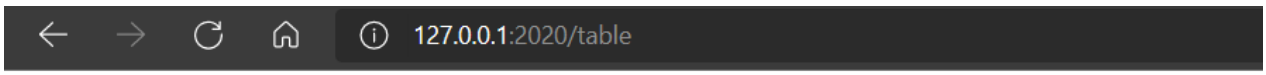
```
<table border="1" class="dataframe">
  <thead>
    <tr style="text-align: right;">
      <th>序号</th>
      <th>书名</th>
      <th>分类</th>
    </tr>
  </thead>
  <tbody>
    <tr>
      <td>1</td>
      <td>HTML5+CSS3+JavaScript从入门到精通（标准版）</td>
      <td>计算机</td>
    </tr>
    <tr>
      <td>2</td>
      <td>JavaWeb项目开发实战入门（全彩版）</td>
      <td>计算机</td>
    </tr>
```

```
    <tr>
      <td>3</td>
      <td>案例学WEB前端开发</td>
      <td>计算机</td>
    </tr>
    <tr>
      <td>4</td>
      <td>一看就停不下来的中国史</td>
      <td>历史</td>
    </tr>
    <tr>
      <td>5</td>
      <td>显微镜下的大明</td>
      <td>历史</td>
    </tr>
  <p><a href="/ ">回到首页</a></p>
  </tbody>
</table>
```

界面效果:

主页面

表格界面



回到首页

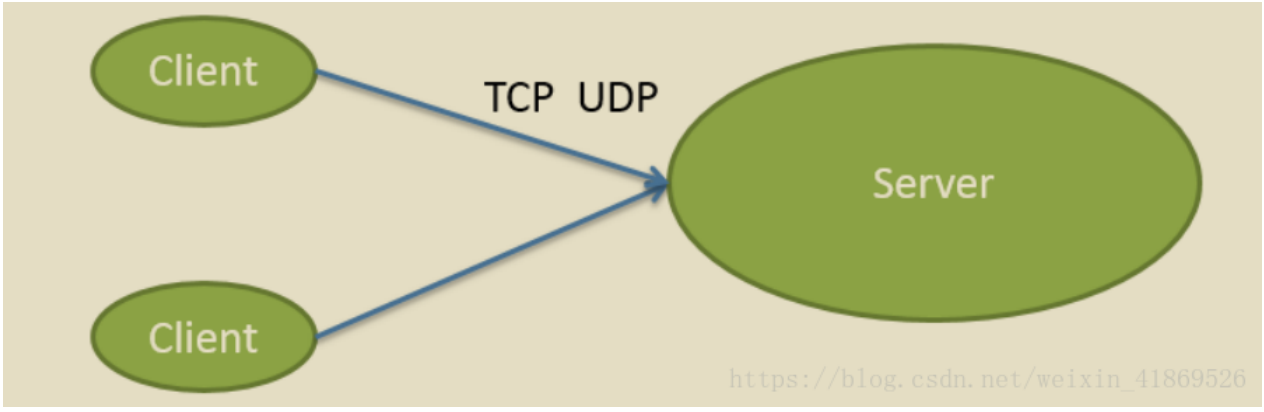
序号	书名	分类
1	HTML5+CSS3+JavaScript从入门到精通（标准版）	计算机
2	JavaWeb项目开发实战入门（全彩版）	计算机
3	案例学WEB前端开发	计算机
4	一看就停不下来的中国史	历史
5	显微镜下的大明	历史

实验总结：

本次Flask框架实现网站的实验涉及很多基础知识，但是在实现难度上来说没有涉及过多算法，更多的是掌握HTML语言，CSS网络通信和Flask界面开发模块的使用方法等。再正式完成本次实验前，我先学习了Flask和HTML，Web网络通信的相关知识。

本次实验涉及的所有源代码和数据都已上传到github数据采集课程的远程仓库，链接https://github.com/MiaoChenglin125/Data-Acquisition/tree/master/HW1，抛砖引玉，供其他同学参考。

附录：



CS即客户端、服务器编程

客户端、服务端之间需要使用Socket，约定协议、版本（往往使用的协议是TCP或者UDP），制定地址和端口，就可以通信了。

客户端、服务端传输数据，数据可以有一定的格式，双方必须线约定好。

BS编程，即Browser、Server开发

Browser浏览器，一种特殊的客户端，支持HTTP（s）协议，能够通过URL向服务端发起请求，等待服务端返回HTML等数据，并在浏览器内可视化展示的程序。

Server，支持HTTP(s)协议，能够接受众多客户端发起的HTTP协议请求，经过处理，将HTML等数据返回给浏览器。

本质上来说，BS是一种特殊的CS，即客户端必须是一种支持HTTP协议且能解析并渲染HTML的软件，服务端必须是能够接收客户端HTTP访问的服务软件。

HTTP协议底层基于TCP协议实现。

BS开发分为两端开发:

(1) 客户端开发，或称前端开发。HTML，CSS,JavaScript等。

(2)服务端开发，Python有WSGI、Flask、Tornado等。

HTTP协议

协议

HTTP协议是无状态协议。

同一个客户端的两次请求之间没有任何关系，从服务器端角度来说，它不知道这两个请求来自同一个客户端。

cookie

键值对信息。

浏览器发起每一请求时，都会把cookie信息发给服务器端。

是一种客户端、服务端传递数据的技术。

服务端可以通过判断这些信息，来确定这次请求是否和之前的请求有关联。

一般来说cookie信息实在服务器端生成，返回给客户端的。

客户端可以自己设置cookie信息。

URL组成

URL可以说就是地址，uniform resource locator 统一资源定位符，每一个链接指向一个资源供客户端访问。

schema://host[:port#]/path/.../[;url-params][?query-string][#anchor]

例如，通过下面URL访问网页：

http://www.magedu.com/pathon/index.html?id=5&name=python

访问静态资源时，通过上面的这个URL访问的是网站的某路径下的Index.html文件，而这个文件对应磁盘上的真实的文件。就会从磁盘上读取这个文件，并把文件的内容发挥浏览器端。

scheme模式、协议：

http、ftp、https、file、mailto等等。mysql等都是类似这样写。

host:port ：

www.magedu.com:80,80端口是默认端口可以不写。域名会使用DNS解析，域名会解析成IP才能使用。实际上会对解析后返回的IP的TCP的80端口发起访问。

/path/to/resource：

path，指向资源的路径。

？key1=value1&key2=value2:

query string，查询字符串，问号分割，后面key=value形式，且使用&符号分割。

HTTP消息

消息分为Request、Response。

Request：浏览器向服务器发起的请求。

Response：服务器对客户端请求的响应。

请求和响应消息都是由请求行、Header消息报头、Body消息正文组成。

请求

请求消息行：请求方法Method 请求路径 协议版本 CRLF

请求头信息　　格式化头信息

```
GET / HTTP/1.1
Host: www.magedu.com
User-Agent: Mozilla/5.0 (Windows NT 6.1; Win64; x64; rv:56.0) Gecko/20100101 Firefox/56.0
Accept: text/html,application/xhtml+xml,application/xml;q=0.9,*/*;q=0.8
Accept-Language: zh-CN,zh;q=0.8,en-US;q=0.5,en;q=0.3
Accept-Encoding: gzip, deflate
Cookie: 53gid2=10019286380004; 53revisit=1512357945900; 53uvid=1; onliner_zdfq72145423=0; ir
Connection: keep-alive
Upgrade-Insecure-Requests: 1
```


请求方法Method:

GET　请求获取URL对应的资源

POST　提交数据至服务器端

HEAD　和GET类似，不过不返回消息正文

常见传递信息的方式：

(1) GET方法使用Query String

http://www.magedu.com/pathon/index.html?id=5&name=python

通过查询字符串在URL中传递参数

(2) POST方法提交数据

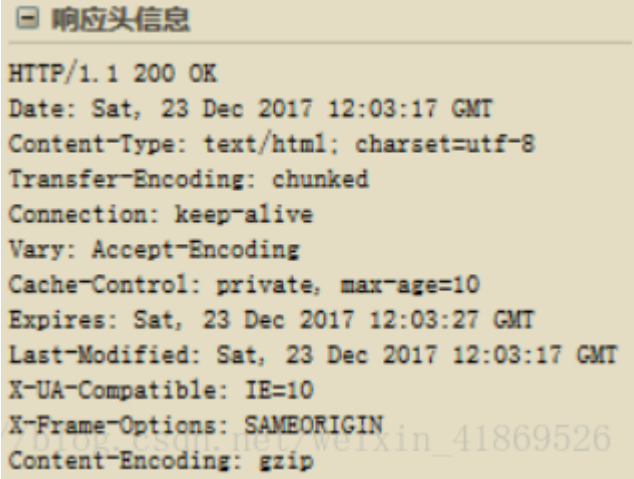
```
http://127.0.0.1:9999/xxx/yyy?id=5&name=magedu
使用表单提交数据，文本框input的name属性分别为age、weight、height

请求消息如下
POST /xxx/yyy?id=5&name=magedu HTTP/1.1
HOST: 127.0.0.1:9999
content-length: 26
content-type: application/x-www-form-urlencoded

age=5&weight=80&height=170
```


(3) URL中本身就包含着信息

http://www.magedu.com/python/student/001

响应

响应消息行：协议版本 状态码 消息描述 CRLF



status code状态码：

状态码在响应头第一行：

```
1xx ：提示信息，表示请求已被成功接收，继续处理

2xx：  表示正常响应

          200 正常返回了网页内容

3xx：     重定向

            301 页面永久性移走，永久重定向。返回新的URL，浏览器会根据返回的URL发起新的Request请求。

            302 临时重定向

            304 资源未修改，浏览器使用本地缓存

4xx：     客户端请求错误

            404  Not Found ，网页找不到，客户端请求的资源有错

            400  请求语法错误

            401  请求要求身份验证

5xx：     服务器端错误

            500  服务器内部错误

            502  上游服务器错误，如nginx反向代理的时候
```

无状态，有连接和短连接

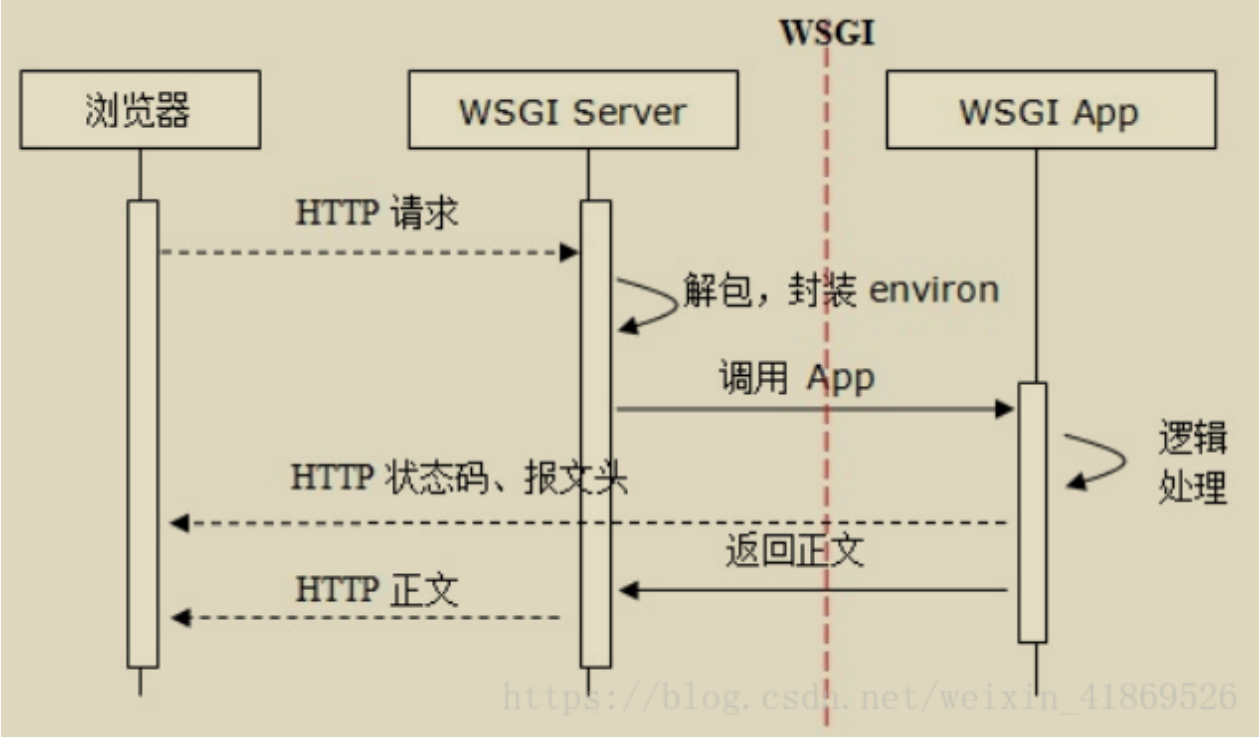
无状态：服务器无法知道两次请求之间的联系，即使是前后两次同一个浏览器也没法判断出是出于一个浏览器的请求。后面通过cookie和session来判断。

有连接：是因为它基于TCP协议，是面向连接的，需要3次握手、4次断开。

短连接：自HTTP 1.1之前，都是一个请求一个连接，而TCOP的连接创建销毁成本高，对服务器有很大的影响。所以，自HTTP 1.1之后，支持keep-alive，默认也是开启的，一个连接打开后，会保持一段时间（可设置），浏览器再访问该服务器就使用这个TCP连接，减轻了服务器压力，提高了效率。

WSGI

WSGI主要规定了服务器端和应用程序之间的接口。



WSGI服务器--wsgiref（实验用）

wsgiref是一个WSGI参考实现库。

wsgiref.simple_server 模块实现一个简单的WSGI HTTP服务器。

wsgiref.simple_server.make_server(host,port,app,server_class=WSGIServer,handler_class=WSGIRequestHandler) 启动一个WSGI服务器。

wsgiref.simple_server.demo_app(environ,start_response) 一个函数，小巧完整的WSGI的应用程序的实现。

```
# 返回文本例子
from wsgiref.simple_server import make_server, demo_app

ip = '127.0.0.1'
port = 9999
server = make_server(ip, port, demo_app) # demo_app应用程序，可调用
server.serve_forever() # server.handle_request() 执行一次
```

WSGI 服务器作用

（1） 监听HTTP服务端口（TCPServer,默认端口80）

（2）接收浏览器端的HTTP请求并解析封装成environ环境数据

（3）负责调用应用程序，将environ和start_response方法传入

（4）将应用程序响应的正文封装成HTTP响应报文返回给浏览器端

WSGI APP应用程序端

1、应用程序应该是一个可调用对象，Python中应该是函数、类、实现了**call**方法的类的实例。

2、这个可调用对象应该接收两个参数

```
# 1 函数实现
def application(environ, start_response):
    pass

# 2 类实现
class Application:
    def __init__(self, environ, start_response):
        pass

# 3 类实现
class Application:
    def __call__(self, environ, start_response):
        pass
```


3、以上可调用对象的实现，都必须返回一个可迭代对象

```
# 函数实现
def application(environ, start_response):
    return [res_str]

# 类实现
class Application:
    def __init__(self, environ, start_response):
        pass
    def __iter__(self): # 实现此方法，对象即可迭代
        yield res_str

# 类实现
class Application:
    def __call__(self, environ, start_response):
        return [res_str]
```


environ和start_response这两个参数名可以是任何合法名，但是一般默认都是这2个名字。

environ

environ是包含HTTP请求信息的dict对象

名称	含义
REQUEST_METHOD	请求方法，GET、POST等
PATH_INFO	URL中的路径部分
QUERY_STRING	查询字符串
SERVER_NAME, SERVER_PORT	服务器名、端口
HTTP_HOST	地址和端口
SERVER_PROTOCOL	协议
HTTP_USER_AGENT	UserAgent信息

start_response

它是一个可调用对象。有三个参数，定义如下：

start_response(status,response_headers,exc_info=None)

status 是状态码，如 200 OK

response_headers 是一个元素为二元组的列表，例如[('Content-Type','text/plain;charset=utf-8')]

exc_info 在错误处理的时候使用

start_response 应该在返回可迭代对象之前调用，因为它返回的是Rsponse Header。返回的可迭代对象是Response Body.

服务器端

服务器程序需要调用符合上述定义的可调用对象APP，传入environ 、start_response, APP处理后，返回响应头和可迭代对象的正文，由服务器封装返回浏览器端。

```python
# 返回网页的例子
from wsgiref.simple_server import make_server

def application(environ, start_response):
    status = '200 OK'
    headers = [('Content-Type', 'text/html;charset=utf-8')]
    start_response(status, headers)
    # 返回可迭代对象
    html = '<h1>马哥教育欢迎你</h1>'.encode("utf-8")
    return [html]

ip = '127.0.0.1'
port = 9999
server = make_server(ip, port, application)
server.serve_forever() # server.handle_request() 一次
```

simple_server 只是参考用，不能用于生产

测试用命令

```
$ curl -I http://192.168.142.1:9999/xxx?id=5
$ curl -X POST http://192.168.142.1:9999/yyy -d '{"x":2}'
```

-I 使用HEAD方法

-X 指定方法，-d 传输数据

到这里，就完成了一个简单的WEB程序开发。

WEB服务器

1、本质上就是一个TCP服务器，监听在特定端口上

2、支持HTTP协议，能够将HTTP请求报文进行解析，能够把响应数据进行HTTP协议的报文封装并返回浏览器端。

3、实现了WSGI协议，该协议约定了和应用程序之间的接口。

APP应用程序

1、遵从WSGI协议

2、本身是一个可调用对象

3、调用start_response，返回响应头部

4、返回包含正文的可迭代对象