实验2：配置Web服务器，编写简单页面，分析交互过程

学号：**2112066**　　姓名：**于成俊**

一、搭建Web服务器

- 使用python的`Django`搭建Web服务器
  - 使用`django-admin startproject myweb`创建名为myweb的Django项目
  - 使用`python manage.py startapp network`创建名为network的应用
- 配置项目
  - 在`myweb/settings.py`文件中，将 'network' 字符串添加到INSTALLED_APPS 项

```
INSTALLED_APPS = [
    'django.contrib.admin',
    'django.contrib.auth',
    'django.contrib.contenttypes',
    'django.contrib.sessions',
    'django.contrib.messages',
    'django.contrib.staticfiles',
    'network',
]
```

  - 调整路由（**为了能让本机环境访问**）：在`myweb/settings.py`文件中，修改其中的ALLOWED_HOSTS 项。将其赋值为["*",]。

```
ALLOWED_HOSTS = ["*",]
```

  - 创建视图函数，在`network/views.py`文件中编写函数：

```
from django.shortcuts import render

# Create your views here.
def login(request):
    return render(request, 'login.html')
```

    并在`myweb/urls.py`文件中，引入刚刚写好的视图函数，然后在 urlpatterns 列表中增加路由映射：

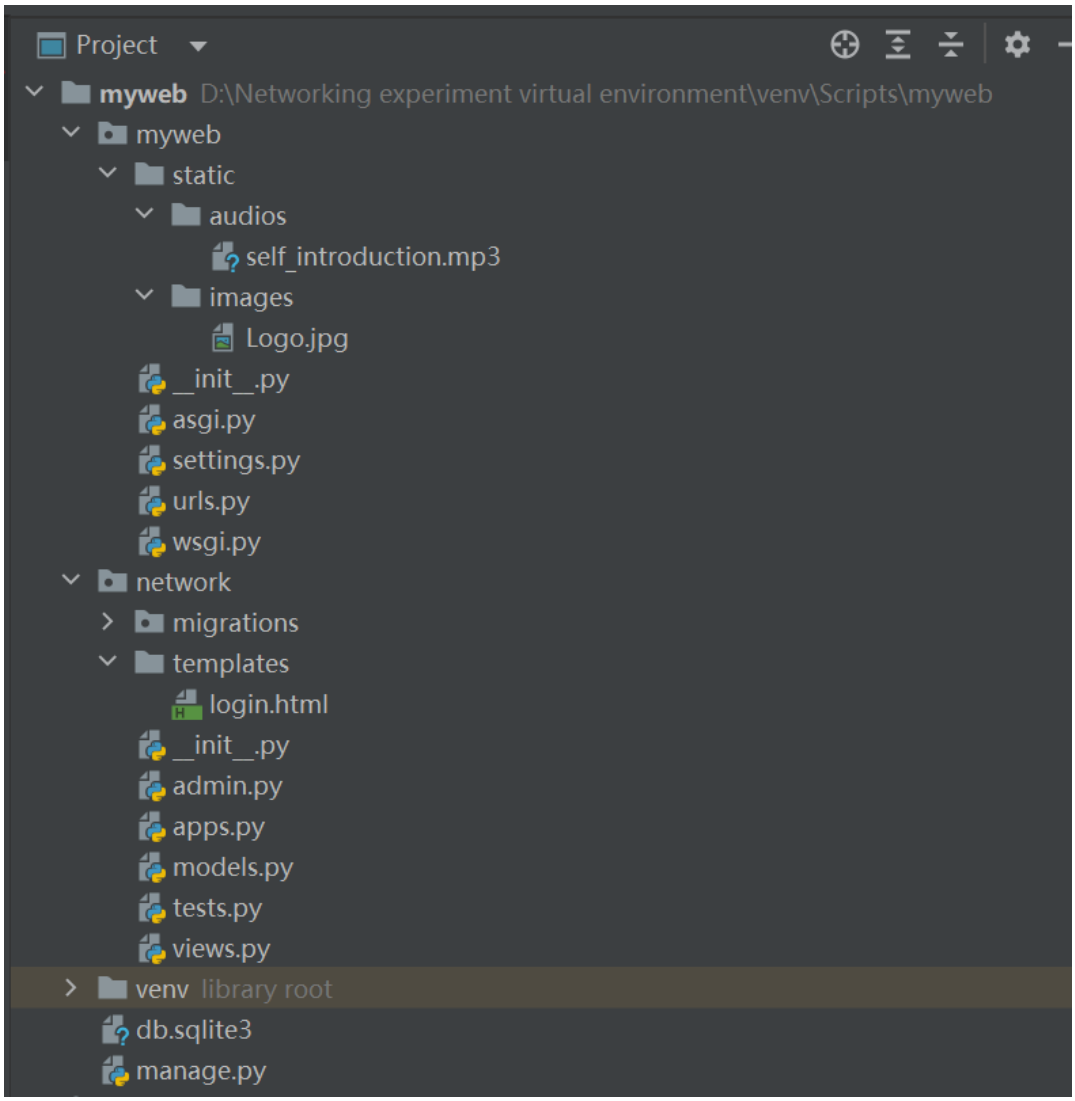
```
from network.views import login   # 引入视图函数
urlpatterns = [
    path('admin/', admin.site.urls),
    path('', login),              # 新增路由映射
]
```

  - 在`network`中创建模板文件`templates`，在`templates`目录下，创建`login.html`文件
  - 设置**静态资源**
    - 在`myweb/myweb`目录下，创建static文件夹，用于存放静态资源，如图片和音频。

- 并在 `myweb/settings.py` 文件中，进行如下更改：

```
# 按下面的设置，则所有的静态资源均会到项目（myweb）目录下的静态资源文件夹中查找
STATIC_URL = '/static/'
STATICFILES_DIRS = ['myweb/static']
```

  - 最终的**项目文件**如下：（`manage.py` 是项目的入口文件）



二、编写Web页面

注意使用 `{% load static %}` 来在页面当中引入静态资源。

其中用到了Django模板语言中用于在Django项目中引用静态文件的方式，若想在**本地环境**运行HTML文件，需要用通常的HTML方式来引用图片和音频资源。

```
<!DOCTYPE html>
<html>
<head>
    <meta charset="UTF-8">
    <title>个人信息</title>
    {% load static %}
</head>
<body>
    <h1>个人信息</h1>
```

```
    <p>专业：密码科学与技术</p>
    <p>学号：2112066</p>
    <p>姓名：于成俊</p>

    <h2>LOGO</h2>
    <img src="{% static 'images/Logo.jpg' %}" alt="皇马"width="200" height="150">

    <h2>自我介绍音频</h2>
    <audio controls>
        <source src="{% static 'audios/self_introduction.mp3' %}"
type="audio/mp3">
        Your browser does not support the audio element.
    </audio>
</body>
</html>
```

三、项目演示

- 启动项目：命令如下

```
D:\Networking experiment virtual environment\venv\Scripts>activate

(venv) D:\Networking experiment virtual environment\venv\Scripts>cd myweb

(venv) D:\Networking experiment virtual environment\venv\Scripts\myweb>python manage.py runserver
Watching for file changes with StatReloader
Performing system checks...

System check identified no issues (0 silenced).
October 21, 2023 - 20:52:05
Django version 4.2.6, using settings 'myweb.settings'
Starting development server at http://127.0.0.1:8000/
Quit the server with CTRL-BREAK.
```

- 点击http://127.0.0.1:8000/进入页面

四、用`Wireshark`捕获交互过程

- 因为我们的实验环境运行在本机的 127.0.0.1 上，所以选择在 Wireshark 里要选择“**Adapter for loopback traffic capture**”网络接口进行捕获，它是本机的环回接口。
- 在过滤器输入ip.addr == 127.0.0.1，只查看ip为127.0.0.1的数据包。
- 首先，可以看到，TCP建立需要进行三次握手
  - 第一次握手：客户端发送SYN包到服务器，并进入SYN_SEND状态，等待服务器确认；
  - 第二次握手：服务器收到SYN包，必须确认客户的SYN，同时自己也发送一个SYN包，即SYN+ACK包，此时服务器进入SYN_RECV状态；
  - 第三次握手：客户端收到服务器的SYN + ACK包，向服务器发送确认包ACK，此包发送完毕，客户端和服务器进入ESTABLISHED状态，完成三次握手。
  - 握手过程中传送的包里不包含数据，可以看到，**Len=0**。三次握手完毕后，客户端与服务器才正式开始传送数据。
  - 可以看到，在发送HTTP请求前，建立了两次TCP连接，这可能与浏览器有关，可能是为了预加载网页，以便实现更快速的浏览和搜索。

```
127.0.0.1          TCP        56 53845 → 8000 [SYN] Seq=0 Win=65535 Len=0 MSS=65495 W
127.0.0.1          TCP        56 8000 → 53845 [SYN, ACK] Seq=0 Ack=1 Win=65535 Len=0
127.0.0.1          TCP        44 53845 → 8000 [ACK] Seq=1 Ack=1 Win=327424 Len=0
127.0.0.1          TCP        56 53846 → 8000 [SYN] Seq=0 Win=65535 Len=0 MSS=65495 W
127.0.0.1          TCP        56 8000 → 53846 [SYN, ACK] Seq=0 Ack=1 Win=65535 Len=0
127.0.0.1          TCP        44 53846 → 8000 [ACK] Seq=1 Ack=1 Win=327424 Len=0
127.0.0.1          HTTP      836 GET / HTTP/1.1
```

- 接着，发送HTTP 1.1的GET请求。
- PSH代表有数据传输，最后显示OK，表示接收到html文本。

```
TCP         44 8000 → 53846 [ACK] Seq=1 Ack=793 Win=2160384 Len=0
TCP         61 8000 → 53846 [PSH, ACK] Seq=1 Ack=793 Win=2160384 Len=17 [TC
TCP         44 53846 → 8000 [ACK] Seq=793 Ack=18 Win=327424 Len=0
TCP         81 8000 → 53846 [PSH, ACK] Seq=18 Ack=793 Win=2160384 Len=37 [T
TCP         44 53846 → 8000 [ACK] Seq=793 Ack=55 Win=327168 Len=0
TCP         84 8000 → 53846 [PSH, ACK] Seq=55 Ack=793 Win=2160384 Len=40 [T
TCP         44 53846 → 8000 [ACK] Seq=793 Ack=95 Win=327168 Len=0
TCP        234 8000 → 53846 [PSH, ACK] Seq=95 Ack=793 Win=2160384 Len=190 |
TCP         44 53846 → 8000 [ACK] Seq=793 Ack=285 Win=327168 Len=0
HTTP       591 HTTP/1.1 200 OK  (text/html)
```

- 在过滤器输入http，使其仅显示HTTP协议。

No.	Time	Source	Destination	Protocol	Length	Info
71	204.368309	127.0.0.1	127.0.0.1	HTTP	836	GET / HTTP/1.1
81	204.384223	127.0.0.1	127.0.0.1	HTTP	591	HTTP/1.1 200 OK (text/html)

- 但是我的html文件中有图片和音频资源，却没有看到相关的请求。查资料了解到，这跟浏览器的缓存机制相关，可以通过ctrl+F5强制刷新，跳过缓存，重新请求数据。

```
127.0.0.1           HTTP      836 GET / HTTP/1.1
127.0.0.1           HTTP      591 HTTP/1.1 200 OK  (text/html)
127.0.0.1           HTTP      879 GET / HTTP/1.1
127.0.0.1           HTTP      591 HTTP/1.1 200 OK  (text/html)
127.0.0.1           HTTP      816 GET /static/images/Logo.jpg HTTP/1.1
127.0.0.1           HTTP      798 GET /static/audios/self_introduction.mp3 HTTP/1.1
127.0.0.1           HTTP     3524 HTTP/1.1 200 OK  (audio/mpeg)
127.0.0.1           HTTP    21033 HTTP/1.1 200 OK  (JPEG JFIF image)
127.0.0.1           HTTP      805 GET /favicon.ico HTTP/1.1
127.0.0.1           HTTP     2253 HTTP/1.1 404 Not Found  (text/html)
```

- 刷新之后，可以看到它请求了图片和音频资源，并接收到了（200 OK）。
- 其中，它还请求了favicon.ico，查资料了解到，favicon.ico是网页图标，一般由一个置于Web服务器的根目录下名为“favicon.ico”的文件来定义，由于我没有设置，所以显示Not Found。
- 然后，我再查看整个过程。可以看出，这是Http1.1区别于Http1.0的地方，GET请求不用每次都要三次握手建立连接，只需一次。

```
HTTP       591 HTTP/1.1 200 OK  (text/html)
TCP         44 54346 → 8000 [ACK] Seq=836 Ack=832 Win=326400 Len=0
HTTP       816 GET /static/images/Logo.jpg HTTP/1.1
TCP         44 8000 → 54346 [ACK] Seq=832 Ack=1608 Win=2159616 Len=0
HTTP       798 GET /static/audios/self_introduction.mp3 HTTP/1.1
```

- 一段时间后，还可以看到四次挥手。

```
TCP         44 54347 → 8000 [FIN, ACK] Seq=755 Ack=147096 Win=2161152 Len=0
TCP         44 8000 → 54347 [ACK] Seq=147096 Ack=756 Win=2160384 Len=0
TCP         44 54346 → 8000 [FIN, ACK] Seq=2369 Ack=589556 Win=303872 Len=0
TCP         44 8000 → 54346 [ACK] Seq=589556 Ack=2370 Win=2158848 Len=0
```

详细说一下TCP四次挥手的过程：

  - **第一次挥手**： 客户端发起挥手请求，向服务器端发送标志位是FIN报文段，设置序列号seq，此时，客户端进入`FIN_WAIT_1`状态，这表示客户端没有数据要发送给服务器端了。
  - **第二次挥手**：服务器端收到了客户端发送的FIN报文段，向客户端返回一个标志位是ACK的报文段，ack设为seq加1，客户端进入`FIN_WAIT_2`状态，服务器端告诉客户端，我确认并同意你的关闭请求。但是服务器端还是可以发送数据到客户端的。
  - **第三次挥手**： 服务器端向客户端发送标志位是FIN的报文段，请求关闭连接，同时客户端进入`LAST_ACK`状态。

- **第四次挥手**： 客户端收到服务器端发送的FIN报文段，向服务器端发送标志位是ACK的报文段，然后客户端进入`TIME_WAIT`状态。服务器端收到客户端的ACK报文段以后，就关闭连接。此时，客户端等待**2MSL**的时间后依然没有收到回复，则证明服务器端已正常关闭，那好，客户端也可以关闭连接了。
- 等待2MSL是为了**保证TCP协议的全双工连接能够可靠关闭**和**保证这次连接的重复数据段从网络中消失。**

- 我们再看一下Http包的详情：

```
Hypertext Transfer Protocol
> GET /static/images/Logo.jpg HTTP/1.1\r\n
  Host: 127.0.0.1:8000\r\n
  Connection: keep-alive\r\n
```

  - /static/images/Logo.jpg 是文件路径
  - HTTP/1.1是传输协议
  - Host是服务器主机地址
  - Connection表示连接状态，keep-alive表示保持连接，还有一个状态是Closed，表示连接断开。

五、知识点相关补充：

- **HTTP状态码**：
  - HTTP 状态码由三个十进制数字组成，第一个十进制数字定义了状态码的类型。响应分为五类：信息响应(100–199)，成功响应(200–299)，重定向(300–399)，客户端错误(400–499)和服务器错误 (500–599)。
  - 常见的有：
    - 200 - 请求成功
    - 301 - 资源（网页等）被永久转移到其它URL
    - 404 - 请求的资源（网页等）不存在
    - 500 - 内部服务器错误
    - 505 - 服务器不支持请求中使用的HTTP版本
- TCP的重要字段：
  - **序号(sequence number)**：seq序号，占32位，用来标识从TCP源端向目的端发送的字节流，发起方发送数据时对此进行标记。为了安全，TCP在开始传输数据前，客户端和服务器需要**随机**生成自己的初始序列号。
  - **确认号（acknowledgement number）**：ack序号，占32位，只有ACK标志位为1时，确认序号字段才有效，ack=seq+1。
  - **标志位（Flags）**：共6个，即URG、ACK、PSH、RST、SYN、FIN等。具体含义如下：
    - ACK：确认序号有效。
    - PSH：接收方应该尽快将这个报文交给应用层，即代表有数据传输
    - RST：重置连接。
    - SYN：发起一个新连接。
    - FIN：释放一个连接。
    - URG：紧急指针（urgent pointer）有效。
- **TCP为什么要三次握手而不是两次握手**：

如果只是两次握手， 至多只有连接发起方的起始序列号能被确认， 另一方选择的序列号则得不到确认，在网络阻塞的情况下，可能会导致历史连接。而三次握手，可以让连接发起方对接收方选择序列号进行确认，从而可以发现这是不是历史连接，如果是，则即使阻断，避免了资源的浪费。