

# 目录

<b>1</b>	<b>实验目的与要求</b>	<b>3</b>
1.1	实验目的 . . . . .	3
1.2	实验要求 . . . . .	3
<b>2</b>	<b>实验原理与实验内容</b>	<b>4</b>
2.1	实验原理 . . . . .	4
2.1.1	Socket 编程接口 . . . . .	4
2.1.2	HTTP 传输协议 . . . . .	6
2.2	实验内容 . . . . .	7
<b>3</b>	<b>实验具体设计实现及结果</b>	<b>7</b>
3.1	实验具体设计实现 . . . . .	7
3.1.1	导入 socket 模块 . . . . .	7
3.1.2	handle_request 函数 . . . . .	7
3.1.3	run_server 函数 . . . . .	9
3.2	程序流程图 . . . . .	11
3.3	实验结果 . . . . .	12
3.3.1	启动后控制台输出 . . . . .	12
3.3.2	浏览器默认访问结果 . . . . .	12
3.3.3	浏览器访问 Aboutus.html 结果 . . . . .	13
3.3.4	浏览器访问 Contactus.html 结果 . . . . .	13
3.3.5	浏览器访问 work.pdf 结果 . . . . .	14
3.3.6	浏览器访问不存在的文件结果 . . . . .	14
3.3.7	使用错误方法访问结果 . . . . .	15
<b>4</b>	<b>实验设备与实验环境</b>	<b>15</b>
<b>5</b>	<b>实验总结</b>	<b>15</b>

<b>6</b>	<b>附录</b>	<b>16</b>
6.1	server.py 源码 . . . . .	16
6.2	index.html 源码 . . . . .	18
6.3	Aboutus.html 源码 . . . . .	20
6.4	Contactus.html 源码 . . . . .	21

# 1 实验目的与要求

## 1.1 实验目的

首先学习面向 TCP 连接的套接字编程基础知识：如何创建套接字，将其绑定到特定的地址和端口，以及发送和接收数据包。其次还将学习 HTTP 协议格式的相关知识。在此基础上，本实验开发一个简单的 Web 服务器，它仅能处理一个 HTTP 连接请求。

## 1.2 实验要求

Web 服务器的基本功能是接受并解析客户端的 HTTP 请求，然后从服务器的文件系统获取所请求的文件，生成一个由头部和响应文件内容所构成的 HTTP 响应消息，并将该响应消息发送给客户端。如果请求的文件不存在于服务器中，则服务器应该向客户端发送“404 Not Found”差错报文。具体的过程和步骤分为：

1. 当一个客户（浏览器）连接时，创建一个连接套接字；
2. 从这个连接套接字接收 HTTP 请求；
3. 解释该请求以确定所请求的特定文件；
4. 从服务器的文件系统获得请求的文件；
5. 创建一个由请求的文件组成的 HTTP 响应报文，报文前面有首部行；
6. 经 TCP 连接向请求浏览器发送响应；
7. 如果浏览器请求一个在该服务器中不存在的文件，服务器应当返回一个“404 Not Found”差错报文。

## 2 实验原理与实验内容

### 2.1 实验原理

#### 2.1.1 Socket 编程接口

要实现 Web 服务器，需使用套接字 Socket 编程接口来使用操作系统提供的网络通信功能。Socket 是应用层与 TCP/IP 协议族通信的中间软件抽象层，是一组编程接口。它把复杂的 TCP/IP 协议族隐藏在 Socket 接口后面，对用户来说，一组简单的接口就是全部，让 Socket 去组织数据，以符合指定的协议。使用 Socket 后，无需深入理解 TCP/UDP 协议细节（因为 Socket 已经为我们封装好了），只需要遵循 Socket 的规定去编程，写出的程序自然就是遵循 TCP/UDP 标准的。Socket 的地位如下图所示：

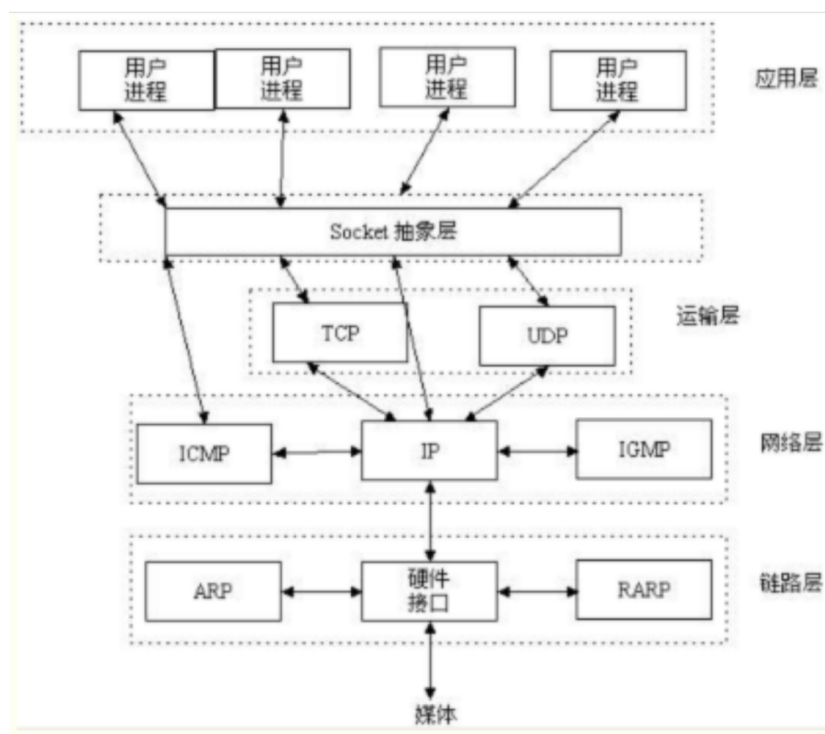


图 1: Socket 的地位

从某种意义上说，Socket 由地址 IP 和端口 Port 构成。IP 是用来标识互联网中的一台主机的位置，而 Port 是用来标识这台机器上的一个应用程序，IP 地址是配置到网卡上的，而 Port 是应用程序开启的，IP 与 Port 的绑定就标识了互联网中独一无二的一个应用程序。

套接字类型流式套接字 (SOCK\_STREAM): 用于提供面向连接、可靠的数据传输服务。数据报套接字 (SOCK\_DGRAM): 提供了一种无连接的服务。该服务并不能保证数据传输的可靠性，数据有可能在传输过程中丢失或出现数据重复，且无法保证顺序地接收到数据。原始套接字 (SOCK\_RAW): 主要用于实现自定义协议或底层网络协议。

在本 WEB 服务器程序实验中，采用流式套接字进行通信。其基本模型如下图所示：

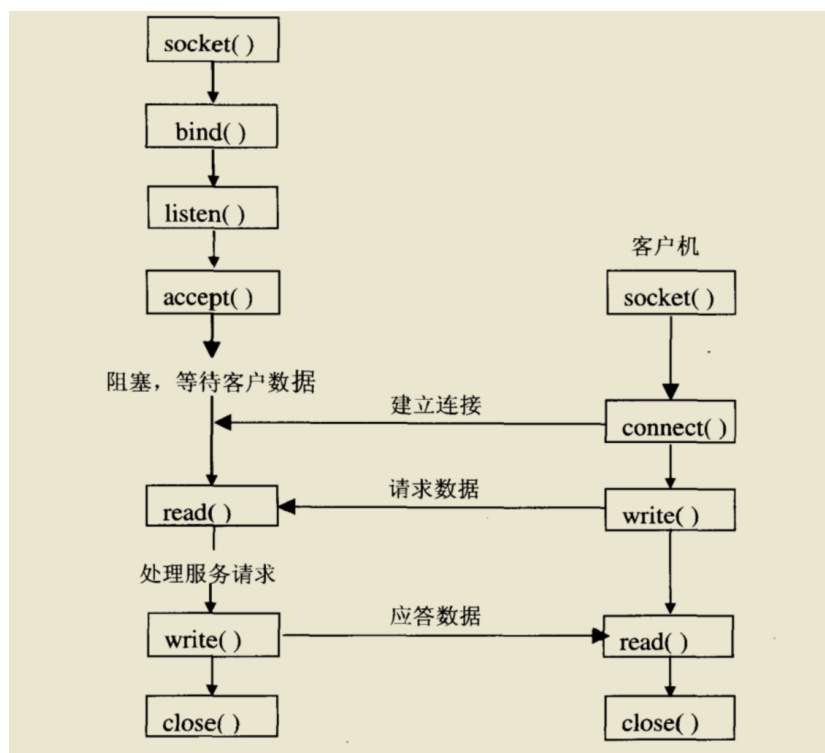


图 2: Socket 的基本模型

其工作过程如下：服务器首先启动，通过调用 `socket()` 建立一个套接字，然后调用绑定方法 `bind()` 将该套接字和本地网络地址联系在一起，再调用 `listen()` 使套接字做好侦听连接的准备，并设定的连接队列的长度。客户端在建立套接字后，就可调用连接方法 `connect()` 向服务器端提出连接请求。服务器端在监听到连接请求后，建立和该客户端的连接，并放入连接队列中，并通过调用 `accept()` 来返回该连接，以便后面通信使用。客户端和服务器连接一旦建立，就可以通过调用接收方法 `recv()` / `recvfrom()` 和发送方法 `send()` / `sendto()` 来发送和接收数据。最后，待数据传送结束后，双方调用 `close()` 关闭套接字。

### 2.1.2 HTTP 传输协议

超文本传输协议 (HTTP) 是用于 Web 上进行通信的协议：它定义 Web 浏览器如何从 Web 服务器请求资源以及服务器如何响应。为简单起见，在该实验中将处理 HTTP 协议的 1.0 版。HTTP 通信以事务形式进行，其中事务由客户端向服务器发送请求，然后读取响应组成。请求和响应消息共享一个通用的基本格式：

- 初始行（请求或响应行）
- 零个或多个头部行
- 空行（CRLF）
- 可选消息正文

对于大多数常见的 HTTP 事务，协议归结为一系列相对简单的步骤：

首先，客户端创建到服务器的连接；然后客户端通过向服务器发送一行文本来发出请求。这请求行包 HTTP 方法（比如 GET, POST、PUT 等），请求 URI（类似于 URL），以及客户机希望使用的协议版本（比如 HTTP/1.0）；接着，服务器发送响应消息，其初始行由状态线（指示请求是否成功），响应状态码（指示请求是否成功完成的数值），以及推理短语（一

种提供状态代码描述的英文消息组成)；最后一旦服务器将响应返回给客户端，它就会关闭连接。

## 2.2 实验内容

建立一个简单的 Web 服务器端，它能够接收客户端的请求，解析请求的方法和路径，然后返回相应的文件内容作为响应。

## 3 实验具体设计实现及结果

### 3.1 实验具体设计实现

#### 3.1.1 导入 socket 模块

#### 3.1.2 `handle_request` 函数

`handle_request` 函数用于处理客户端的请求，其主要功能是根据客户端的请求，返回相应的响应报文。

首先，从客户端接收请求报文，接收最多 1024 个字节的数据，解码成字符串，存储在变量 `request_data` 中；

然后，将请求报文按照回车换行符分割成请求行和请求头部，请求行中包含请求方法、请求资源路径和 HTTP 版本号，请求头部中包含请求的其他信息；

从第一行中获取请求方法、请求路径和协议版本，如果请求方法是 GET 则：

如果请求路径是 `/`，则将请求路径设置为 `/index.html`，即默认返回 `index.html` 文件；

否则构造文件路径，将请求路径前面的 `/` 去掉，尝试打开文件，如果文件不存在，则返回 404 错误；如果文件存在，则读取文件内容，构造响应报文，将响应头部和响应内容拼接起来，构成完整的响应报文；

如果请求方法不是 GET，则返回 405 错误；

发送响应报文给客户端，关闭连接。

以下是实现代码：

```
1 def handle_request(client_socket):
2     # 接收客户端请求数据从客户端套接字接收最多 1024 字节的数据，
3     # 并将其解码为字符串，存储在变量 request_data 中，
4     request_data = client_socket.recv(1024).decode()
5
6     # 解析请求数据，获取请求文件路径对这个字符串进行分割操作，将
7     # 其按照回车换行符（\r\n）进行切割，将其拆分成多行。
8     request_lines = request_data.split('\r\n')
9     if len(request_lines) > 0:
10         # 获取请求方法和文件路径
11         # method: 表示请求方法，如 GET、POST、PUT 等。它是请求行
12         # 中的第一个部分。
13         # path: 表示请求的路径，即请求访问的资源在服务器上的位
14         # 置。它是请求行中的第二个部分。
15         method, path, _ = request_lines[0].split(' ')
16         if method == 'GET':
17             if path == '/':
18                 # 默认返回 index.html 文件
19                 file_path = 'index.html'
20             else:
21                 # 构造文件路径
22                 file_path = path[1:] # 去除路径中的斜杠
23
24         try:
25             # 读取文件内容
26             with open(file_path, 'rb') as file:
27                 file_content = file.read()
28             # 构造响应报文将响应头和文件内容拼接起来，构成完
29             # 整的响应数据。
30             # response_data 变量通过将 response_headers 编码
31             # 为字节流，并与 file_content 拼接在一起，得到
32             # 最终的响应数据。
```



```

27         response_headers = 'HTTP/1.1 200 OK\r\n\r\n'
        response_data = response_headers.encode() +
            file_content
29     except FileNotFoundError:
        # 请求的文件不存在，返回 404 Not Found 错误
        response_headers = 'HTTP/1.1 404 Not Found\r\n\r\n'
31        response_data = response_headers.encode()

33    # 发送响应数据给客户端
    client_socket.sendall(response_data)
35    else:
        response_headers = 'HTTP/1.1 405 Method Not Allowed\r\n\r\n'
37        response_data = response_headers.encode()
        client_socket.sendall(response_data)

39    # 关闭客户端连接
41    client_socket.close()

```

handle\_request 函数

### 3.1.3 run\_server 函数

run\_server 函数用于启动服务器，创建套接字，绑定地址和端口，监听客户端连接，接收客户端请求，处理客户端请求等。

在 run\_server 函数中：

- 创建服务器套接字
- 调用 bind 方法绑定服务器的主机地址和端口号
- 调用 listen 方法开始监听客户端的连接请求。
- 进入一个无限循环，等待客户端连接。

- 当有客户端连接时，接受连接并获取客户端套接字对象和客户端地址。
- 打印客户端连接信息。

以下是实现代码：

```
1 def run_server():
    # 创建服务器套接字.
3     # socket.AF_INET 参数表示使用 IPv4 地址族，socket.
        SOCK_STREAM 参数表示使用 TCP 协议。
    server_socket = socket.socket(socket.AF_INET, socket.
        SOCK_STREAM)
5
    # 调用 server_socket.bind(('localhost', 80)) 绑定服务器的主
        机地址和端口号。'localhost' 表示服务器在本地主机上运行，
        80 是服务器的端口号
7     server_socket.bind(('localhost', 80))
9
    # 调用 server_socket.listen(1) 开始监听客户端的连接请求。参
        数 1 表示允许同时处理的最大连接数为 1。
    server_socket.listen(1)
11    print('Server is running on http://localhost:80/')
13
    while True:
        # 等待客户端连接
15        client_socket, addr = server_socket.accept()
        print('Client connected:', addr)
17        # 处理客户端请求
        handle_request(client_socket)
```

run\_server 函数

## 3.2 程序流程图

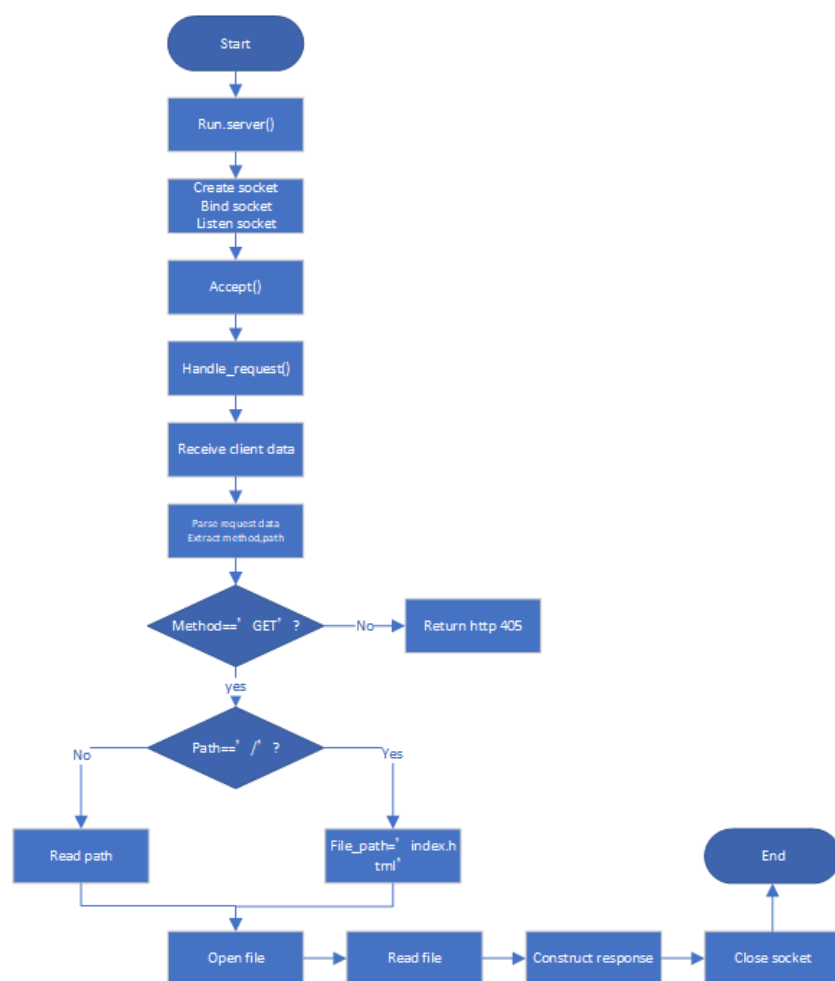
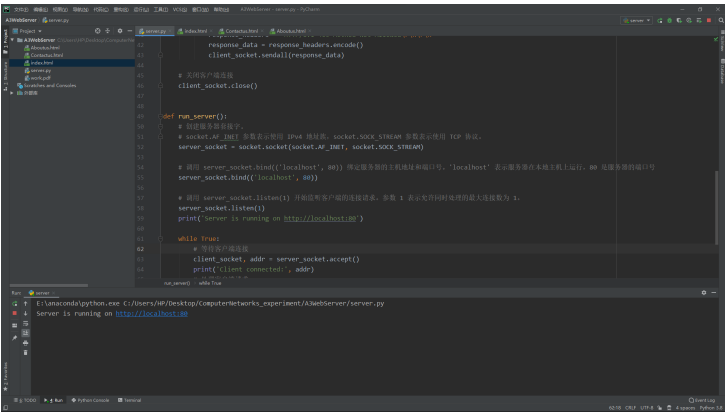


图 3: 程序流程图

### 3.3 实验结果

#### 3.3.1 启动后控制台输出



```
1 # 返回响应数据
2 response_data = response_headers.encode()
3 client_socket.sendall(response_data)
4
5 # 关闭客户端连接
6 client_socket.close()
7
8 def run_server():
9     # 创建套接字对象
10     # socket.AF_INET 参数表示使用 IPv4 协议套, socket.SOCK_STREAM 参数表示使用 TCP 协议。
11     server_socket = socket.socket(socket.AF_INET, socket.SOCK_STREAM)
12
13     # 调用 server_socket.bind() 绑定服务器的主机地址和端口号, 'localhost' 表示服务器在本机上运行, 80 是服务器的端口号
14     server_socket.bind(('localhost', 80))
15
16     # 调用 server_socket.listen() 开始监听客户端的连接请求, 参数 1 表示在同时处理的最大连接数为 1。
17     server_socket.listen(1)
18     print("Server is running on http://localhost:80")
19
20     while True:
21         # 等待客户端连接
22         client_socket, addr = server_socket.accept()
23         print("Connection from:", addr)
24
25         # 处理请求
26         # 这里只是简单的返回一个响应数据
```

图 4: 启动后控制台输出

#### 3.3.2 浏览器默认访问结果

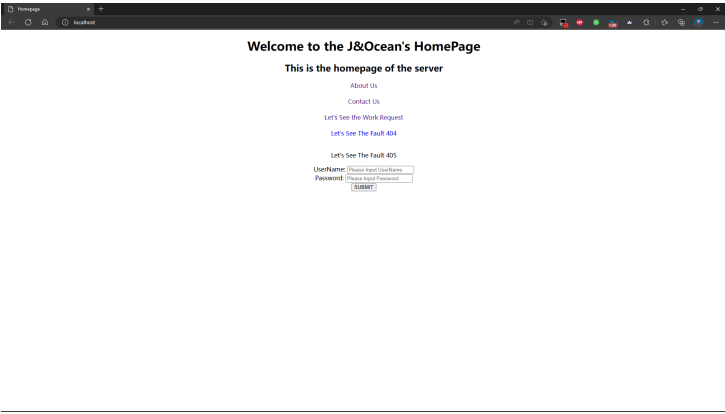


图 5: 浏览器默认访问结果

### 3.3.3 浏览器访问 Aboutus.html 结果

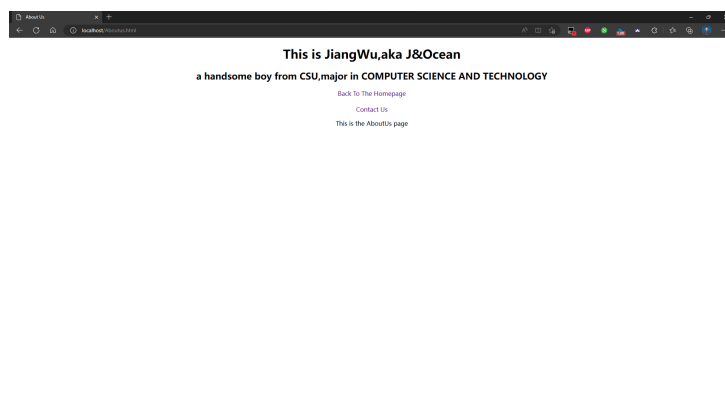


图 6: 浏览器访问 Aboutus.html 结果

### 3.3.4 浏览器访问 Contactus.html 结果

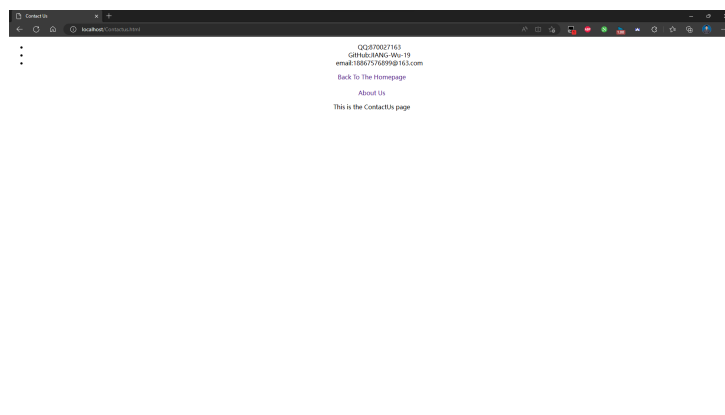


图 7: 浏览器访问 Contactus.html 结果

### 3.3.5 浏览器访问 work.pdf 结果



图 8: 浏览器访问 work.pdf 结果

### 3.3.6 浏览器访问不存在的文件结果

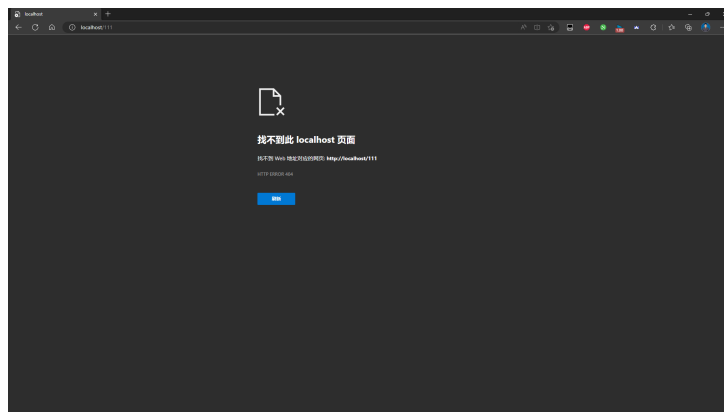


图 9: 浏览器访问不存在的文件结果

### 3.3.7 使用错误方法访问结果

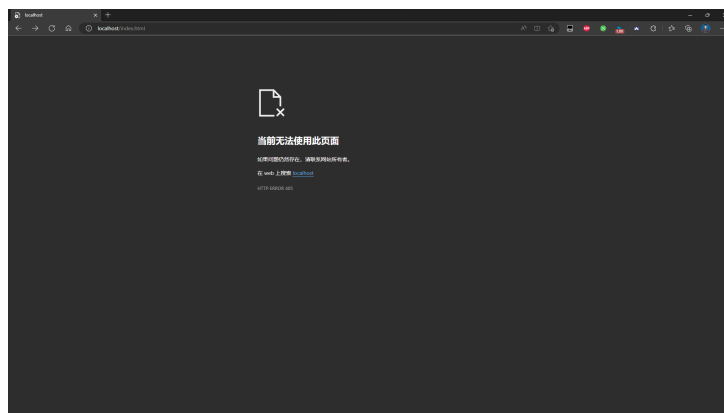


图 10: 使用错误方法访问结果

## 4 实验设备与实验环境

1. 编程语言: python
2. 编程环境: pycharm, windows11 操作系统

## 5 实验总结

在本次实验中，程序实现了一个简单的 Web 服务器端。该服务器能够接收客户端的请求，解析请求的方法和路径，然后返回相应的文件内容作为响应。实现过程中遇到的困难：在这个实验中，主要的困难是理解和处理 HTTP 协议相关的内容。需要熟悉 HTTP 请求和响应的格式以及常见的状态码。另外，还需要处理文件的读取和异常情况。

通过这个实验，我对基本的 Web 服务器端实现有了更深入的理解。我学会了使用 Python 的 socket 库创建服务器套接字、绑定地址和端口，监听客户端连接请求，并能够处理 HTTP 请求和构造响应报文。

这个简单的 Web 服务器端实现还有很大的改进空间。例如，可以添加更多的 HTTP 方法的支持，如 POST、PUT 等。还可以引入并发处理多个客户端连接的能力，提高服务器的并发性能。此外，还可以考虑对请求参数进行解析和处理，增加服务器的功能和灵活性。

总体而言，这个实验帮助我更好地理解 Web 服务器端的基本原理和实现方式。通过实际编码和调试的过程，我对 HTTP 协议和 socket 编程有了更深入的了解，为以后进一步探索网络编程和 Web 开发打下了坚实的基础。

## 6 附录

### 6.1 server.py 源码

```
2      # 实验A.3：简单Web服务器端实现
3
4      import socket
5
6      def handle_request(client_socket):
7          # 接收客户端请求数据从客户端套接字接收最多 1024 字节的数据，并将其解码为字符串，存储在变量 request_data 中，
8          request_data = client_socket.recv(1024).decode()
9
10         # 解析请求数据，获取请求文件路径对这个字符串进行分割操作，将其按照回车换行符（\r\n）进行切割，将其拆分成多行。
11         request_lines = request_data.split('\r\n')
12         if len(request_lines) > 0:
13             # 获取请求方法和文件路径
14             # method: 表示请求方法，如 GET、POST、PUT 等。它是请求行中的第一个部分。
15             # path: 表示请求的路径，即请求访问的资源在服务器上的位置。它是请求行中的第二个部分。
```



```

16 method, path, _ = request_lines[0].split(' ')
17 if method == 'GET':
18     if path == '/':
19         # 默认返回 index.html 文件
20         file_path = 'index.html'
21     else:
22         # 构造文件路径
23         file_path = path[1:] # 去除路径中的斜杠
24
25     try:
26         # 读取文件内容
27         with open(file_path, 'rb') as file:
28             file_content = file.read()
29         # 构造响应报文将响应头和文件内容拼接起来，构
30         # 成完整的响应数据。
31         # response_data 变量通过将 response_headers
32         # 编码为字节流，并与 file_content 拼接在一
33         # 起，得到最终的响应数据。
34         response_headers = 'HTTP/1.1 200 OK\r\n\r\n'
35         response_data = response_headers.encode() +
36             file_content
37     except FileNotFoundError:
38         # 请求的文件不存在，返回 404 Not Found 错误
39         response_headers = 'HTTP/1.1 404 Not Found\r\n\r\n'
40         response_data = response_headers.encode()
41
42     # 发送响应数据给客户端
43     client_socket.sendall(response_data)
44 else:
45     response_headers = 'HTTP/1.1 405 Method Not
46         Allowed\r\n\r\n'
47     response_data = response_headers.encode()
48     client_socket.sendall(response_data)

```

```

46         # 关闭客户端连接
        client_socket.close()

48
49
50     def run_server():
51         # 创建服务器套接字。
52         # socket.AF_INET 参数表示使用 IPv4 地址族，socket.
53         # SOCK_STREAM 参数表示使用 TCP 协议。
54         server_socket = socket.socket(socket.AF_INET, socket.
55         SOCK_STREAM)
56
57         # 调用 server_socket.bind(('localhost', 80)) 绑定服务器
58         # 的主机地址和端口号。'localhost' 表示服务器在本地主机
59         # 上运行，80 是服务器的端口号
60         server_socket.bind(('localhost', 80))
61
62         # 调用 server_socket.listen(1) 开始监听客户端的连接请
63         # 求。参数 1 表示允许同时处理的最大连接数为 1。
64         server_socket.listen(1)
65         print('Server is running on http://localhost:80')
66
67         while True:
68             # 等待客户端连接
69             client_socket, addr = server_socket.accept()
70             print('Client connected:', addr)
71             # 处理客户端请求
72             handle_request(client_socket)
73
74     run_server()

```

server.py

## 6.2 index.html 源码

```

1      <!DOCTYPE html>
      <html lang="en">
3      <head>
          <meta charset="UTF-8">
          <title>Homepage</title>
          <style>
7              body{
                  text-align: center;
9              }
              a{
11                 text-decoration: none;
              }
13             a:hover{
                  background: #0000ff;
15                 color: #fff;
              }
17         </style>
      </head>
19      <body>
          <h1>Welcome to the J&Ocean's HomePage</h1>
          <h2>This is the homepage of the server</h2>
21         <a href="Aboutus.html">About Us</a>
23         <br>
          <br>
25         <a href="Contactus.html">Contact Us</a>
          <br>
27         <br>
          <a href="work.pdf">Let's See the Work Request</a>
29         <br>
          <br>
31         <a href="111">Let's See The Fault 404</a>
          <br>
33         <br>
          <p>Let's See The Fault 405</p>

```

```

35     <form method="post">
        <label for="username">UserName:</label>
37     <input type="text" name="username" id="username"
        placeholder="Please Input UserName">
        <br>
39     <label for="password">Password:</label>
        <input type="password" name="password" id="password"
        placeholder="Please Input Password">
41     <br>
        <input type="submit" value="SUBMIT">
43     </form>
    </body>
45 </html>

```

index.html

### 6.3 Aboutus.html 源码

```

1  <!DOCTYPE html>
    <html lang="en">
3  <head>
        <meta charset="UTF-8">
5        <title>About Us</title>
        <style>
7            body{
                text-align: center;
9            }
            a{
11               text-decoration: none;
            }
13           a:hover{
                background: #0000ff;
15               color: #fff;
            }
17        </style>

```

```

19 </head>
20 <body>
21   <h1>This is JiangWu, aka J&Ocean</h1>
22   <h2>a handsome boy from CSU, major in COMPUTER SCIENCE
23     AND TECHNOLOGY</h2>
24   <a href="index.html">Back To The Homepage</a>
25   <br>
26   <br>
27   <a href="Contactus.html">Contact Us</a>
28   <br>
29   <p>This is the AboutUs page</p>
30 </body>
31 </html>

```

Aboutus.html

## 6.4 Contactus.html 源码

```

1  <!DOCTYPE html>
2  <html lang="en">
3  <head>
4    <meta charset="UTF-8">
5    <title>Contact Us</title>
6    <style>
7      body{
8        text-align: center;
9      }
10     a{
11       text-decoration: none;
12     }
13     a:hover{
14       background: #0000ff;
15       color: #fff;
16     }
17   </style>

```

```
19 </head>
20 <body>
21   <ul>
22     <li>QQ:870027163</li>
23     <li>GitHub:JIANG-Wu-19</li>
24     <li>email:18867576899@163.com</li>
25   </ul>
26   <a href="index.html">Back To The Homepage</a>
27   <br>
28   <br>
29   <a href="Aboutus.html">About Us</a>
30   <br>
31   <p>This is the ContactUs page</p>
32 </body>
33 </html>
```

Contactus.html