南開大學
Nankai University

计算机学院

计算机网络实验报告

实验 3：配置 Web 服务器，捕获 HTTP 报文并分析

姓名：王旭

学号：2312166

专业：计算机科学与技术

2025 年 1 月 2 日

目录

1 实验要求

1. 搭建 Web 服务器（自由选择系统），并制作简单的 Web 页面，包含简单文本信息（包含专业、学号、姓名）和 6 幅图像。
2. 通过浏览器获取自己编写的 Web 页面，使用 Wireshark 捕获浏览器与 Web 服务器的交互过程。
3. 分析两个报文的封装层次，包括数据链路层、互连层、传输层、应用层。
4. 对整个 HTTP 交互过程进行详细说明，使用 Wireshark 过滤器使其仅显示 HTTP 协议。
5. 提交 HTML 文档、Wireshark 捕获文件和实验报告。

注：页面不要太复杂，包含所要求的基本信息即可。使用 HTTP，不要使用 HTTPS。

2 实验原理

2.1 HTTP 协议 (HyperText Transfer Protocol)

HTTP（超文本传输协议）是万维网 (WWW) 的数据通信基础，定义了客户端（浏览器）与服务器之间交换数据的格式和规则。

- **协议架构**：HTTP 是基于 **TCP/IP** 协议栈的应用层协议，默认使用 TCP 80 端口（本实验使用 8000 端口）。
- **请求-响应模型**：通信总是由客户端发起请求，服务器回送响应。服务器无法主动向客户端推送数据（除非使用 WebSocket 等新技术）。
- **无状态性 (Stateless)**：HTTP 协议本身是无状态的，即服务器不保留与客户交易过的任何状态。每次请求都是独立的。
- **关键方法 (Methods)**：
  - **GET**：请求获取 Request-URI 所标识的资源（本实验主要使用此方法）。
  - **POST**：向指定资源提交数据进行处理请求。
- **状态码 (Status Codes)**：
  - **200 OK**：请求成功。
  - **304 Not Modified**：资源未修改，使用缓存（我们在实验中遇到的情况，因为如我们使用的浏览器 firefox 等会对请求的资源进行一定的自动缓存，因此如果不通过 Ctrl+F5 进行强制刷新的话，会有很多资源都显示 304 Not Modified 的情况）。
  - **404 Not Found**：服务器无法找到请求的资源。
- **持久连接 (Keep-Alive)**：HTTP/1.1 默认支持持久连接，允许在同一个 TCP 连接上发送多个 HTTP 请求和响应，减少了建立和关闭连接的消耗。

2.2 TCP/IP 分层模型与数据封装

计算机网络通信遵循分层模型，数据在发送端进行**封装 (Encapsulation)**，在接收端进行**解封装 (Decapsulation)**。本实验涉及 TCP/IP 四层模型：

1. **应用层 (Application Layer)**
   - **功能**：为应用程序提供网络服务。
   - **数据单元**：报文 (Message)。
   - **本实验体现**：HTTP 协议生成的 GET 请求和 HTML 响应内容。
2. **传输层 (Transport Layer)**
   - **功能**：提供端到端的、可靠的进程间通信。
   - **协议**：TCP (Transmission Control Protocol)。
   - **关键机制**：
     - **端口号**：区分不同的应用程序（如浏览器临时端口 vs 服务器 8000 端口）。
     - **三次握手**：建立可靠连接（SYN → SYN+ACK → ACK）。
     - **可靠传输**：通过序列号 (Seq) 和确认号 (Ack) 保证数据无差错、不丢失、不重复、按序到达。
   - **数据单元**：报文段 (Segment)。
3. **互连层 (Internet Layer)**
   - **功能**：负责主机间的逻辑寻址和路由选择。
   - **协议**：IP (IPv4 / IPv6)。本实验使用 IPv6。
   - **关键机制**：IP 地址（如 `::1`）标识网络中的主机。
   - **数据单元**：数据报 (Packet)。
4. **数据链路层 (Data Link Layer)**
   - **功能**：负责相邻节点间的帧传输。
   - **数据单元**：帧 (Frame)。
   - **本实验体现**：Loopback 伪首部，模拟链路层传输。

2.3 Web 服务器工作原理 (B/S 架构)

Web 服务器（如我们本次实验使用的 Python `http.server`）基于 **B/S (Browser/Server)** 架构工作，其核心流程如下：

1. **创建套接字 (Socket)**：服务器启动，创建 TCP 套接字。
2. **绑定与监听 (Bind & Listen)**：绑定 IP 地址和端口（8000），进入监听状态，等待客户端连接。
3. **接受连接 (Accept)**：当浏览器发起请求时，TCP 三次握手建立连接。
4. **接收请求 (Receive)**：读取浏览器发送的 HTTP 请求报文。

5. **解析与处理 (Parse & Process)**：解析请求行（方法、URL），查找本地文件系统是否存在对应资源（如 `index.html`）。
6. **发送响应 (Send)**：
   - 若资源存在，读取文件内容，构造包含 `200 OK` 的响应报文发送给浏览器。
   - 若资源不存在，构造 `404 Not Found` 响应。
7. **释放/保持连接**：根据 `Connection` 头部决定关闭 TCP 连接还是保持连接以处理后续请求。

2.4 回环接口 (Loopback Interface)

- **定义**：回环接口是一个虚拟网络接口，通常对应 IP 地址 `127.0.0.1` (IPv4) 或 `::1` (IPv6)。
- **工作机制**：发往回环地址的数据包不会经过物理网卡（NIC），也不会出现在物理线路上。操作系统网络协议栈识别出目的地址是本机后，直接将数据包放入接收缓冲区，模拟“发送-接收”过程。
- **实验意义**：允许在单台计算机上同时运行客户端和服务器进行网络实验，排除物理网络环境干扰，专注于协议逻辑分析。

3 实验环境

- **操作系统**：Windows 11(本机)
- **Web 服务器**：使用 Python 内置 `http.server` 模块
- **浏览器**：Firefox 火狐浏览器
- **抓包工具**：Wireshark (v4.6.2) + Npcap (Loopback Adapter)
- **开发工具**：VS Code

4 实验步骤

4.1 搭建 Web 服务器与页面制作

4.1.1 页面代码实现 (index.html)

对于本次页面的设计，我们编写了如下 HTML 代码。包含了个人信息展示区和图片展示区。

index.html

```
<!DOCTYPE html>
<html lang="zh-CN">
<head>
    <meta charset="UTF-8">
    <title>计 算 机 网 络 实 验 三</title>
    <style>
        /* CSS 样 式 定 义 */
        body {
```

```
            font-family: "Microsoft YaHei", sans-serif;
            text-align: center; /* 页面内容居中 */
        }
        .info {
            margin: 20px;
            padding: 20px;
            border: 1px solid #ccc;
            display: inline-block; /* 信息框自适应宽度 */
        }
        .gallery {
            display: flex;
            flex-wrap: wrap; /* 允许图片自动换行 */
            justify-content: center;
            gap: 10px; /* 图片间距 */
            margin-top: 20px;
        }
        .gallery img {
            width: 200px;
            height: 150px;
            object-fit: cover; /* 保持图片比例填充 */
            border: 1px solid #ddd;
        }
    </style>
</head>
<body>
    <!-- 页面主标题 -->
    <h1>计算机网络实验三: Web服务器测试</h1>

    <!-- 个人信息区域 -->
    <div class="info">
        <h2>学生信息</h2>
        <p><strong>专业: </strong> 计算机科学与技术</p>
        <p><strong>学号: </strong> 2312166</p>
        <p><strong>姓名: </strong> 王旭</p>
    </div>

    <!-- 图片展示区域 -->
    <div class="gallery">
        <img src="img1.jpg" alt="图片1">
        <img src="img2.jpg" alt="图片2">
        <img src="img3.jpg" alt="图片3">
        <img src="img4.jpg" alt="图片4">
        <img src="img5.jpg" alt="图片5">
        <img src="img6.jpg" alt="图片6">
    </div>
</body>
</html>
```

4.1.2 代码详细分析

- **文档类型声明 (<!DOCTYPE html>)**：声明该文档采用 HTML5 标准，确保浏览器以标准模式渲染页面。
- **头部信息 (<head>)**：
  - `<meta charset="UTF-8">`：设置字符编码为 UTF-8，防止中文乱码。
  - `<style>`：使用内部 CSS 美化页面。采用了 Flexbox 布局 (`display: flex`) 来排列图片，使其在不同屏幕宽度下都能整齐排列；使用了 `object-fit: cover` 属性确保图片在固定大小的框内不变形。
- **主体内容 (<body>)**：
  - **标题区**：使用 `<h1>` 标签显示实验题目。
  - **信息区 (.info)**：使用 `<div>` 容器包裹我们的个人信息，通过 CSS 设置边框和内边距，使其视觉上独立。
  - **图片区 (.gallery)**：包含 6 个 `<img>` 标签，分别引用相对路径下的 `img1.jpg` 到 `img6.jpg`。这是产生后续 HTTP 请求的关键部分：浏览器解析到这些标签时，会自动向服务器发起额外的 GET 请求来下载图片。

4.1.3 服务器启动与资源准备

- **资源准备**：在 `index.html` 同级目录下放置了 6 张测试图片（命名为 `img1.jpg` 至 `img6.jpg`）以及 `favicon.ico` 图标文件，以避免 404 错误。
- **启动服务器**：使用 Python 内置的 `http.server` 模块，这是一个轻量级的 HTTP 服务器，对于我们本次的实验可以起到很方便的搭建服务器的作用。在终端执行命令：

启动服务器命令

```
python -m http.server 8000
```

运行结果：

终端显示 `Serving HTTP on :: port 8000 (http://[::]:8000/)`，表明服务器已成功在 8000 端口监听请求。如下图所示：

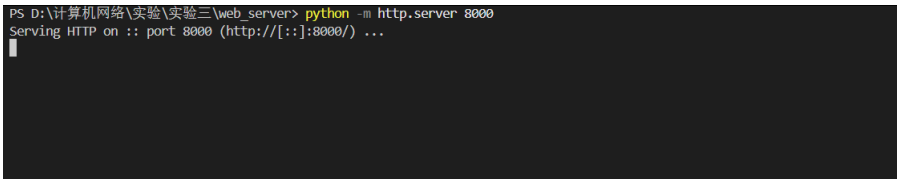

图 4.1: 服务器已成功在 8000 端口监听请求

4.1.4 访问 Web 页面与服务器终端结果

此时我们在浏览器中访问 `http://localhost:8000`，就可以看到我们通过我们的 Web 服务器搭建出来的 Web 页面了。

图 4.2: Web 页面

可以看到 Web 页面可以正确加载所有资源，我们的文字信息和六张图片以及左上角的 favicon.ico 图标都可以正常显示。

并且 Web 服务器终端也可以正常显示与我们 Firefox 浏览器的交互结果：

```
PS D:\计算机网络\实验\实验三\web_server> python -m http.server 8000
Serving HTTP on :: port 8000 (http://[::]:8000/) ...
::1 - - [02/Jan/2026 17:52:29] "GET / HTTP/1.1" 200 -
::1 - - [02/Jan/2026 17:52:29] "GET /img1.jpg HTTP/1.1" 200 -
::1 - - [02/Jan/2026 17:52:29] "GET /img2.jpg HTTP/1.1" 200 -
::1 - - [02/Jan/2026 17:52:29] "GET /img3.jpg HTTP/1.1" 200 -
::1 - - [02/Jan/2026 17:52:29] "GET /img6.jpg HTTP/1.1" 200 -
::1 - - [02/Jan/2026 17:52:29] "GET /img4.jpg HTTP/1.1" 200 -
::1 - - [02/Jan/2026 17:52:29] "GET /img5.jpg HTTP/1.1" 200 -
::1 - - [02/Jan/2026 17:52:29] "GET /favicon.ico HTTP/1.1" 200 -
```

图 4.3: Web 服务器终端显示结果

4.2 Wireshark 抓包设置

- **接口选择**：由于是本机访问本机（Localhost），选择 **"Adapter for loopback traffic capture"** 接口进行抓包。

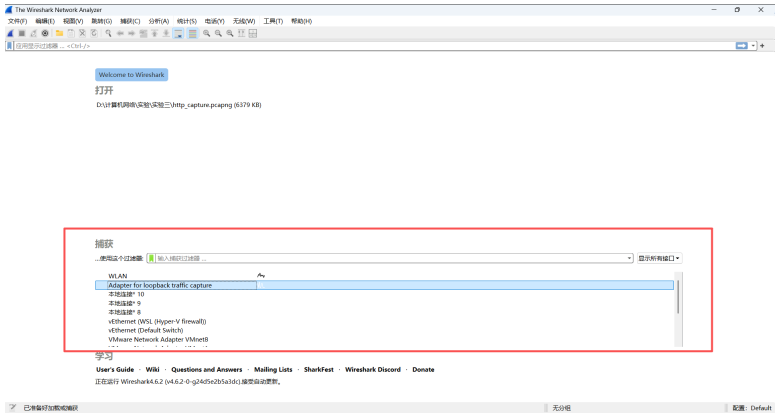

图 4.4: 选择 **"Adapter for loopback traffic capture"** 接口进行抓包

- **过滤器设置**：为了仅显示 HTTP 协议相关内容，**在过滤器栏输入 `http` 并应用**。

4.3 捕获过程

- 启动 Wireshark 开始抓包。
- 在浏览器中访问 `http://localhost:8000`。
- 使用 **Ctrl + F5** 强制刷新页面，确保浏览器不使用缓存，从而产生完整的 HTTP 请求流量。
- 停止抓包并保存为 `http_capture.pcapng`。

5 报文封装层次分析

我们对于 Wireshark 捕获的 http 过滤包内容选取了一对 HTTP 请求与响应报文，下面结合 TCP/IP 协议栈的四层模型（应用层、传输层、互连层、数据链路层）进行详细的封装分析。

5.1 HTTP 请求报文分析

选取报文：No. 86 (GET / HTTP/1.1), 该报文是浏览器向服务器发起的第一个 HTTP 请求，用于获取首页内容。

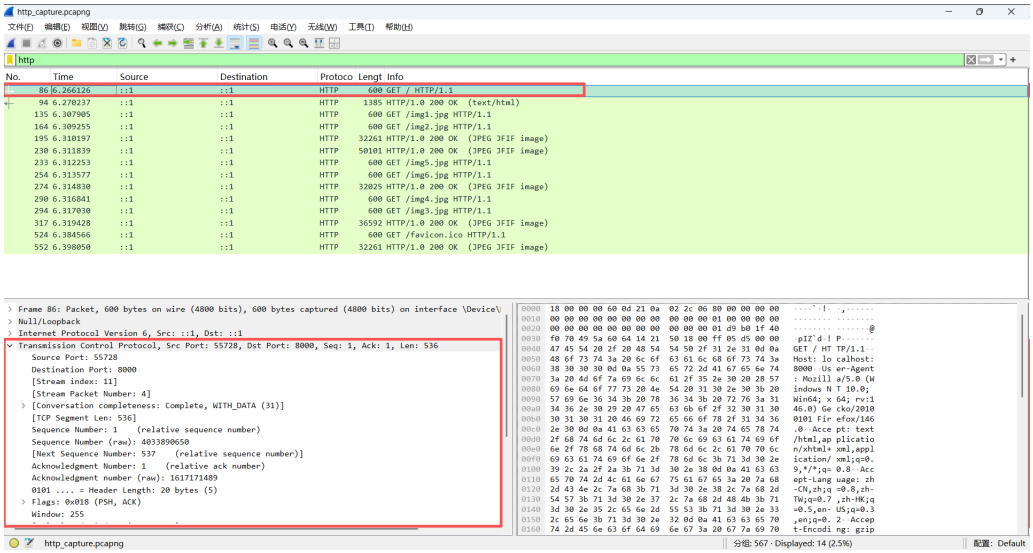

图 5.5: No. 86 (GET / HTTP/1.1) 报文

5.1.1 数据链路层 (Data Link Layer)

- **理论原理**：负责在相邻节点之间传输数据帧，封装成帧并进行差错检测。
- **实际分析**：
  - **接口**：`Null/Loopback`。
  - **协议族**：`Family: IPv6 (24)`。

- **分析**：由于实验是在本机（Localhost）进行的，数据并未经过物理网卡（如以太网卡），而是通过操作系统的回环接口（Loopback Adapter）直接转发。Wireshark 显示为”Null/Loopback”类型的帧，帧长度为 600 字节。

5.1.2 互连层 (Internet Layer)

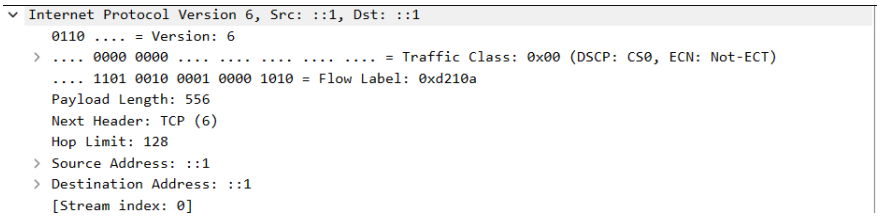

图 5.6: 互联层

- **理论原理**：负责主机到主机的逻辑寻址和路由选择，将数据报从源主机发送到目的主机。
- **实际分析**：
  - **协议**：`Internet Protocol Version 6 (IPv6)`。
  - **源地址 (Source)**：`::1` (IPv6 的 Localhost 地址)。
  - **目的地址 (Destination)**：`::1`。
  - **关键字段**：
    - `Payload Length: 556`：表示 IP 数据报的载荷长度为 556 字节（包含 20 字节 TCP 头部 + 536 字节 HTTP 数据）。
    - `Next Header: TCP (6)`：指示上层协议为 TCP。
  - **分析**：IP 层头部指明了**数据的发送方和接收方都是本机**。

5.1.3 传输层 (Transport Layer)

- **理论原理**：负责提供进程到进程（Process-to-Process）的通信，TCP 提供可靠的、面向连接的字节流服务，通过端口号区分不同的应用程序。
- **实际分析**：
  - **协议**：`Transmission Control Protocol (TCP)`。
  - **端口信息**：
    - **源端口 (Source Port)**：`55728`。这是操作系统为浏览器分配的临时端口（Ephemeral Port）。
    - **目的端口 (Destination Port)**：`8000`。这是 Python Web 服务器监听的知名端口（Well-known Port）。
  - **控制信息**：
    - **序列号 (Seq)**：`1` (Raw: 4033890650)。Wireshark 显示的是相对序列号，实际的原始序列号是一个**随机生成的 32 位整数**，用于保证安全性。

    - **确认号 (Ack)**：1 (Raw: 1617171489)。表示期望收到对方下一个报文段的第一个字节的序号。
    - **标志位 (Flags)**：0x018 (PSH, ACK)。
        - **ACK**：表示确认号字段有效，这是连接建立后的正常状态。
        - **PSH (Push)**：表示接收方（服务器）应尽快将数据交付给应用层，而不是在缓冲区排队。这符合 HTTP 请求的特性，服务器需要立即处理请求。
  - **窗口 (Window)**：255 (Calculated window size: 65280)。表示接收方的缓冲区大小，用于流量控制。
  - **载荷长度 (Len)**：536。表示 TCP 数据部分（即 HTTP 请求报文）的长度为 536 字节。

5.1.4 应用层 (Application Layer)

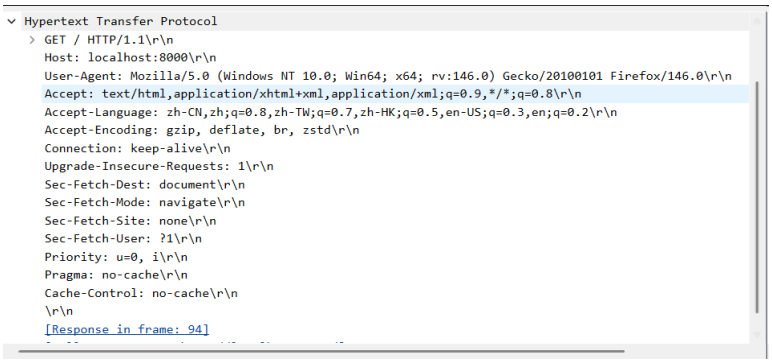

图 5.7: 应用层

- **理论原理**：定义了运行在不同端系统上的应用程序进程如何相互传递报文。
- **实际分析**：
  - **协议**：Hypertext Transfer Protocol (HTTP)。
  - **请求行**：GET / HTTP/1.1。
    - **方法**：GET，表示请求获取资源。
    - **URL**：/，表示请求根目录（即 index.html）。
    - **版本**：HTTP/1.1。
  - **关键头部字段**：
    - Host: localhost:8000：指定请求的目标主机和端口。
    - User-Agent: Mozilla/5.0 ... Firefox/146.0：表明客户端身份为 Firefox 浏览器。
    - Connection: keep-alive：请求服务器保持 TCP 连接打开，以便后续请求（如图片）复用同一连接，提高效率。
    - Accept-Language: zh-CN...：告知服务器客户端偏好的语言为简体中文。
    - Cache-Control: no-cache 和 Pragma: no-cache：**特别说明**，这两个字段的出现是因为我们在实验中使用了 `Ctrl + F5` 强制刷新。它们明确指示浏览器和服务器不要使用缓存的副本，必须重新传输最新的数据，这确保了我们能抓取到完整的 HTTP 交互流量。

5.2 HTTP 响应报文分析

选取报文：No. 94 (HTTP/1.0 200 OK)，该报文是服务器对 No. 86 请求的回复，包含了网页的 HTML 代码。

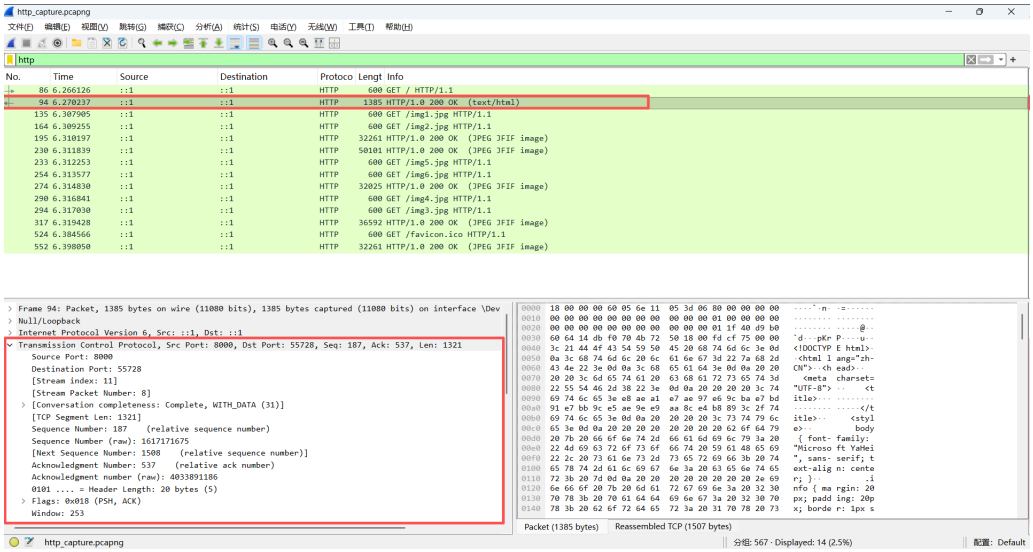

图 5.8: No. 94 (HTTP/1.0 200 OK) 报文

5.2.1 数据链路层与互连层

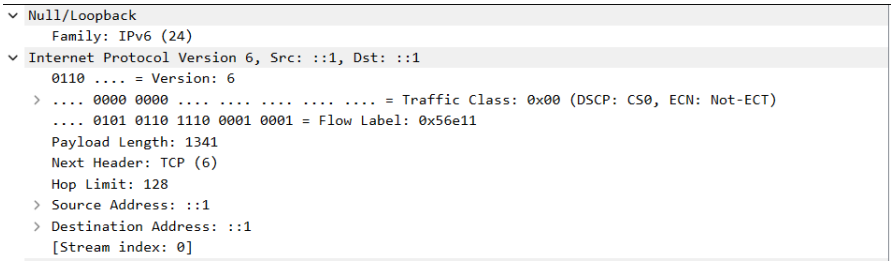

图 5.9: 数据链路层与互连层

- **分析**：与请求报文类似，依然是 Loopback 接口和 IPv6 协议。唯一的区别是逻辑方向相反，但由于源和目的 IP 都是 `::1`，看起来一样。
  - **Payload Length**: 1341。表示 IP 载荷长度为 1341 字节（20 字节 TCP 头部 + 1321 字节 HTTP 数据）。

5.2.2 传输层 (Transport Layer)

- **实际分析**：
  - **端口交换**：
    - **源端口**：8000（服务器）。
    - **目的端口**：55728（浏览器）。
    - **分析**：TCP 连接是双向的，响应报文必须准确发回给发起请求的那个浏览器进程（端口 55728）。

- **控制信息**：
  - **序列号 (Seq)**：187 (Raw: 1617171675)。
  - **确认号 (Ack)**：537 (Raw: 4033891186)。服务器确认收到了浏览器发来的 536 字节数据（1 + 536 = 537），期待浏览器的下一次发送从 537 开始。
  - **标志位 (Flags)**：0x018 (PSH, ACK)。同样带有 PSH 标志，表示服务器希望浏览器尽快将这些 HTML 数据交付给渲染引擎，以便用户能立刻看到网页。
- **载荷长度 (Len)**：1321。表示 TCP 数据部分（即 HTTP 响应体）的长度为 1321 字节，这正好对应了我们 index.html 文件的大小。

5.2.3 应用层 (Application Layer)

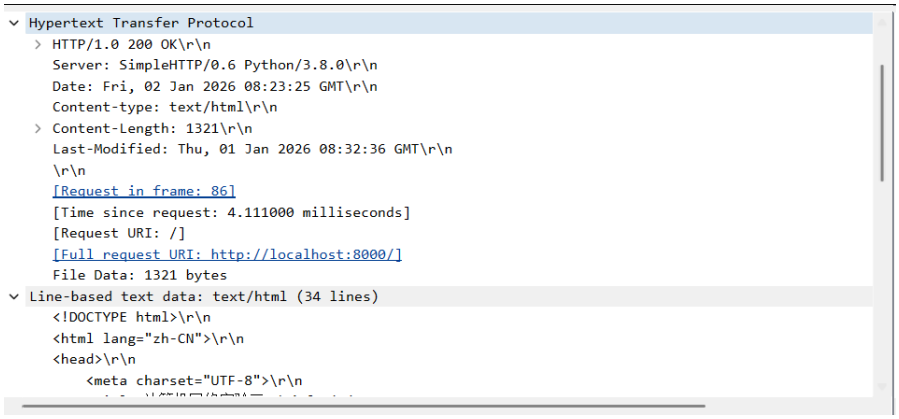

图 5.10: 应用层

- **实际分析**：
  - **状态行**：HTTP/1.0 200 OK。
    - **版本**：HTTP/1.0。Python 的 SimpleHTTPRequestHandler 默认使用 HTTP/1.0 协议。
    - **状态码**：200。表示请求成功处理。
    - **短语**：OK。
  - **关键头部字段**：
    - Server: SimpleHTTP/0.6 Python/3.8.0：明确标识了服务器软件及其版本，验证了我们使用的是 Python 内置服务器。
    - Date: Fri, 02 Jan 2026 08:23:25 GMT：响应生成的服务器时间。
    - Content-Type: text/html：告知浏览器返回的数据类型是 HTML 文档，浏览器据此决定调用 HTML 解析器。
    - Content-Length: 1321：响应体的精确字节数。
  - **响应体 (Line-based text data)**：
    - Wireshark 自动解析并展示了 HTML 源码，如 <title> 计算机网络实验三 </title>、<h1>...</h1> 等。
    - 这部分内容与我们在 VS Code 中编写的 index.html 完全一致，证明文件传输无误。

6 HTTP 交互过程详细说明

通过 Wireshark 捕获的 http 包的流量（如下图所示），我们可以清晰地还原浏览器与 Web 服务器之间的完整交互过程。本次实验共捕获了 14 个 HTTP 数据包，对应了浏览器对于 HTML 页面和图片的资源请求流程。

图 6.11: Wireshark 捕获的 http 数据包列表

6.1 初始阶段：获取 HTML 主页

- **No. 86 (Time: 6.266):** `GET / HTTP/1.1`
  - **动作**：浏览器向服务器（端口 8000）发起第一个 HTTP 请求。
  - **分析**：用户在地址栏输入 URL 并回车（或强制刷新）后，浏览器首先请求根路径 `/`。这是整个交互的起点。
- **No. 94 (Time: 6.270):** `HTTP/1.0 200 OK (text/html)`
  - **动作**：服务器响应请求，返回 `index.html` 文件内容。
  - **分析**：服务器成功找到默认首页文件，状态码 200 表示成功。`Content-Type: text/html` 告诉浏览器这是一个网页，浏览器接收到数据后开始解析 HTML 代码。

6.2 解析与并发请求阶段：获取图片资源

浏览器在解析 `index.html` 的过程中，发现代码中包含 6 个 `<img>` 标签（引用了 `img1.jpg` 到 `img6.jpg`），于是立即自动发起后续请求。

- **No. 135 (Time: 6.307):** `GET /img1.jpg HTTP/1.1`
  - **动作**：浏览器请求第一张图片。
- **No. 164 (Time: 6.309):** `GET /img2.jpg HTTP/1.1`
  - **动作**：浏览器请求第二张图片。
- **No. 195 (Time: 6.310):** `HTTP/1.0 200 OK (JPEG JFIF image)`
  - **动作**：服务器返回 `img1.jpg` 的图片数据。
  - **分析**：注意时间戳，服务器在收到请求后仅用了约 3ms 就返回了图片。`Content-Type` 为 `image/jpeg`，浏览器收到后将其渲染在页面对应位置。

- **No. 230 (Time: 6.311):** `HTTP/1.0 200 OK (JPEG JFIF image)` ——服务器返回 `img2.jpg`。
- **No. 233 (Time: 6.312):** `GET /img5.jpg HTTP/1.1` ——浏览器请求 `img5.jpg`。
- **No. 254 (Time: 6.313):** `GET /img6.jpg HTTP/1.1` ——浏览器请求 `img6.jpg`。
- **No. 274 (Time: 6.314):** `HTTP/1.0 200 OK (JPEG JFIF image)` ——服务器返回 `img5.jpg`。
- **No. 290 (Time: 6.316):** `GET /img4.jpg HTTP/1.1` ——浏览器请求 `img4.jpg`。
- **No. 294 (Time: 6.317):** `GET /img3.jpg HTTP/1.1` ——浏览器请求 `img3.jpg`。
- **No. 317 (Time: 6.319):** `HTTP/1.0 200 OK (JPEG JFIF image)` ——服务器返回 `img3.jpg`。

注：由于浏览器和服务器的并发处理机制，请求和响应的顺序可能会交错，例如浏览器可能连续发出多个图片请求，然后再陆续收到响应。

6.3 收尾阶段：获取网站图标

- **No. 524 (Time: 6.384):** `GET /favicon.ico HTTP/1.1`
  - **动作**：浏览器请求网站图标。
  - **分析**：这是现代浏览器的标准行为。在页面主要内容加载完毕后，浏览器会自动尝试获取 `favicon.ico` 以显示在标签页标题旁。
- **No. 552 (Time: 6.398):** `HTTP/1.0 200 OK (JPEG JFIF image)`
  - **动作**：服务器返回图标文件。
  - **分析**：由于我们在实验准备阶段准备了一个图片作为 `favicon.ico`，所以服务器成功找到了文件并返回 `200 OK`，避免了常见的 404 错误。

6.4 交互总结

整个过程体现了 HTTP 协议 **“请求-响应”** 的工作模式：

1. **无状态性**：每一个 `GET` 请求都是独立的，服务器处理完一个请求就发送一个响应。
2. **持久连接 (Keep-Alive)**：虽然我们在 Wireshark 过滤器中只显示了 HTTP 包，但实际上底层的 TCP 连接在请求之间是被复用的（可以看到请求头中有 `Connection: keep-alive`），这大大提高了加载多张图片的效率。
3. **并行加载**：浏览器并不是等一张图片完全下载完才请求下一张，而是几乎同时发起了多个图片请求（如 No. 233, 254, 290, 294 几乎在同一毫秒内发出），充分利用了网络带宽。

7 实验总结

本次实验我们成功搭建了基于 Python 的轻量级 Web 服务器，并制作了包含个人信息和六张图片的 Web 页面。并且我们使用 Wireshark 的 Loopback 接口，成功捕获了本机环境下的 HTTP 交互报文。通过对 Wireshark 捕获的 http 报文的详细分析，我深入理解了 HTTP 协议在应用层的工作方式，

以及其底层的 TCP 传输层、IP 网络层的封装结构，验证了网络协议栈的分层模型，了解了如何使用 Wireshark 这一工具，来帮助我们学习计算机网络中的数据包知识与传输知识。通过我一步一步地设计服务器、使用 Wireshark 抓包与分析，收获了很多知识和实践能力。

8 github 仓库源码

代码详见:https://github.com/Wangxuuuuu/Computer-Network