

计算机网络实验报告

Lab2 配置Web服务器，编写简单页面，分析交互过程

网络空间安全学院 物联网工程

2110951 梁晓储

代码已发布到github: https://github.com/WangshuXC/Computer_network

一、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（至少包含专业、学号、姓名）、自己的LOGO、自我介绍的音频信息。页面不要太复杂，包含要求的基本信息即可。
- (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，并进行简单的分析说明。
- (3) 使用HTTP，不要使用HTTPS。
- (4) 提交实验报告。

二、功能实现与代码分析

Web服务器启动文件：app.js

```
const express = require('express');
const app = express();
const path = require('path');
const os = require('os');

app.use(express.static(path.join(__dirname, './')));

const port = 6200;

app.listen(port, '0.0.0.0', () => {
  const networkInterfaces = os.networkInterfaces();
  let ipAddress;

  // 遍历网络接口，找到无线网和有线网口的IPv4地址
  Object.keys(networkInterfaces).forEach((interfaceName) => {
    networkInterfaces[interfaceName].forEach((networkInterface) => {
      if (
        !networkInterface.internal &&
        (networkInterface.family === 'IPv4') &&
        (interfaceName.includes('WLAN') || interfaceName.includes('ETH'))
        //排除掉虚拟机的Ipv4地址
      ) {
        ipAddress = networkInterface.address;
        console.log(`Device Name: ${interfaceName}`);
        console.log(`IP Address: ${ipAddress}`);
      }
    });
  });
});
```

```

});

//方便直接打开网址
console.log(`Available on:`);
console.log(`http://localhost:${port}`);
console.log(`http://${ipAddress}:${port}`);
});

```

在文件目录下使用 `node app.js` 即可启动web服务器，同时会在控制台以 `ip地址+端口号` 的形式输出web地址，方便直接打开web

除此之外，在我在实验设备安装WSL之前只有WLAN一个网络接口，固不需要进行ip地址的筛选即是正确的WLAN校园网ipv4，但是存在虚拟机只后可能会获取到虚拟机的ipv4地址，所以添加了 `(interfaceName.includes('WLAN') || interfaceName.includes('ETH'))` 来进行网络接口的筛选，排除掉虚拟机的ipv4地址

HTML部分代码

```

<!DOCTYPE html>
<html lang="en">

<head>
  <meta charset="UTF-8">
  <meta name="viewport" content="width=device-width, initial-scale=1.0">
  <title>2110951自我介绍</title>
  <style>
    /* css代码已省略 */
  </style>
</head>

<body>
  <div id="app">
    <div class="container">
      <div class="logo">
        
      </div>
      <div class="info">
        <p class="title">自我介绍</p>
        <p>梁晓储</p>
        <p>2110951</p>
        <p>物联网工程</p>
      </div>
      <a class="github" href="https://www.github.com/wangshuxc"
target="_blank">
        访问我的Github
      </a>
      <div class="audio">
        <p>音频介绍</p>
        <audio id="audioElement" src="audio.mp3"></audio>
        <button id="playButton" onclick="togglePlay()"></button>
        <div class="progressBar">

```

```

        <div class="progressBarFill" :style="{ width: progress + '%'"
    }"></div>
    </div>
</div>
</div>
</div>
</div>

<script>
    var audioElement = document.getElementById('audioElement');
    audioElement.addEventListener('timeupdate', updateProgressBar);
    audioElement.addEventListener('ended', resetAudio);

    var isPlaying = false;
    var playButton = document.getElementById('playButton');
    playButton.textContent = isPlaying ? '⏸' : '▶';
    var progress = 0;

    function togglePlay() {
        if (isPlaying) {
            audioElement.pause();
            playButton.textContent = '▶';
        } else {
            audioElement.play();
            playButton.textContent = '⏸';
        }
        isPlaying = !isPlaying;
    }

    function updateProgressBar() {
        var progressBarFill = document.querySelector('.progressBarFill');
        progress = (audioElement.currentTime / audioElement.duration) * 100;
        progressBarFill.style.width = progress + '%';
    }

    function resetAudio() {
        audioElement.currentTime = 0;
        progress = 0;
        isPlaying = false;
        playButton.textContent = '▶';
    }

    var audioElement = new Audio('audio.mp3');
    audioElement.addEventListener('timeupdate', updateProgressBar);
    audioElement.addEventListener('ended', resetAudio);
    var isPlaying = false;
    var progress = 0;
</script>
</body>

</html>

```

在html中将logo、info、audio存放在一个名为 `container` 的div标签中，方便使用css对它进行美化

同时使用JavaScript代码添加了点击后会在  和  的按钮（当播放音频完毕后按钮会重置为初始状态），添加了展示音频播放进度的进度条

因为本人对于网页的外观有一定的需求，固使用css对上述页面进行美化

```
<style>
  body {
    display: flex;
    justify-content: center;
    align-items: center;
  }

  .container {
    display: flex;
    flex: 1;
    flex-direction: column;
    justify-content: center;
    align-items: center;
    width: 50vw;
    height: auto;
    padding: 40px;
    background-color: #edf4fb;
    border-radius: 10px;
    margin-top: 5vh;
    box-shadow: 0 2px 4px rgba(0, 0, 0, 0.1);
  }

  .logo {
    display: flex;
    flex-direction: column;
    align-items: center;
    justify-content: center;
    width: auto;
    height: 15vh;
    background-color: white;
    color: rgb(0, 0, 0);
    border-radius: 10px;
    padding: 10px;
    box-shadow: 0 2px 4px rgba(0, 0, 0, 0.1);
  }

  .logo img {
    width: 100%;
    height: 100%;
    border-radius: 10px;
  }

  .info {
    display: flex;
    flex-direction: column;
    align-items: center;
    justify-content: center;
    width: 30%;
```

```
    height: 10%;
    background-color: white;
    color: rgb(0, 0, 0);
    border-radius: 10px;
    padding: 20px;
    margin: 20px;
    box-shadow: 0 2px 4px rgba(0, 0, 0, 0.1);
    overflow: hidden;
}

.info p {
    font-size: inherit;
    margin: 5px;
    padding: 0;
    white-space: nowrap;
    overflow: hidden;
    text-overflow: ellipsis;
}

.title {
    font-size: calc(1em + 3px);
    font-weight: bold;
    margin-top: 10px;
}

.github {
    display: flex;
    flex-direction: column;
    align-items: center;
    justify-content: center;
    width: auto;
    background-color: white;
    border-radius: 10px;
    border: none;
    outline: none;
    padding: 10px;
    margin-bottom: 20px;
    box-shadow: 0 2px 4px rgba(0, 0, 0, 0.1);

    cursor: pointer;
}

.github:focus,
.github:hover {
    background-color: rgb(231, 231, 231);
}

a.github {
    text-decoration: none;
    color: black;
}

.audio {
    display: flex;
    flex-direction: column;
```

```
        align-items: center;
        justify-content: center;
        width: 40%;
        height: 5%;
        background-color: white;
        color: rgb(0, 0, 0);
        border-radius: 10px;
        padding: 20px;
        box-shadow: 0 2px 4px rgba(0, 0, 0, 0.1);
    }

    .audio p {
        font-size: larger;
        padding-bottom: 20px;
        padding: 0;
    }

    .audio button {
        display: flex;
        flex-direction: column;
        align-items: center;
        justify-content: center;
        margin-bottom: 20px;
        background-color: white;
        border-radius: 10px;
        padding: 10px;
        box-shadow: 0 2px 4px rgba(0, 0, 0, 0.1);

        font-size: larger;
        outline: none;
        border: none;
    }

    .progressBar {
        width: 80%;
        height: 10px;
        background-color: #ccc;
        box-shadow: 0 2px 4px rgba(0, 0, 0, 0.1);
        margin-top: 10px;
        border-radius: 10px;
    }

    .progressBarFill {
        height: 100%;
        width: 0%;
        background-color: #c0daf7;
        border-radius: 10px;
    }
</style>
```

三、抓包过程与分析

前期准备

为了方便进行抓包，我使用计算机A中部署web服务器并且在计算机B访问网页并且进行抓包

计算机A的ipv4地址为 10.130.101.102，开放端口 6200 用于部署web服务器

打开wireshark后在计算机B访问 http://10.130.101.102:6200 ,等待一段时间后关闭抓包并且对已经抓好的包进行分析

抓包过程

首先在wireshark中对抓好的包进行筛选，除去其他请求和接收的干扰

使用 ip.addr == 10.130.101.102 and tcp.port == 6200，只对网页进行抓包

No.	Time	Source	Destination	Protocol	Length	Info
1212	17.842101	10.136.60.226	10.130.101.102	TCP	66	10054 → 6200 [SYN] Seq=0 Win=64240 Len=0 MSS=1460 W
1213	17.842371	10.136.60.226	10.130.101.102	TCP	66	10055 → 6200 [SYN] Seq=0 Win=64240 Len=0 MSS=1460 W
1214	17.851007	10.130.101.102	10.136.60.226	TCP	66	6200 → 10054 [SYN, ACK] Seq=0 Ack=1 Win=65535 Len=0
1215	17.851007	10.130.101.102	10.136.60.226	TCP	66	6200 → 10055 [SYN, ACK] Seq=0 Ack=1 Win=65535 Len=0
1218	17.851223	10.136.60.226	10.130.101.102	TCP	54	10054 → 6200 [ACK] Seq=1 Ack=1 Win=131328 Len=0
1219	17.851288	10.136.60.226	10.130.101.102	TCP	54	10055 → 6200 [ACK] Seq=1 Ack=1 Win=131328 Len=0
1220	17.878511	10.136.60.226	10.130.101.102	HTTP	641	GET / HTTP/1.1
1222	17.894323	10.130.101.102	10.136.60.226	HTTP	320	HTTP/1.1 304 Not Modified
1227	17.940565	10.136.60.226	10.130.101.102	TCP	54	10054 → 6200 [ACK] Seq=588 Ack=267 Win=131072 Len=0
1234	18.105384	10.136.60.226	10.130.101.102	HTTP	560	GET /icon.jpg HTTP/1.1
1235	18.113060	10.130.101.102	10.136.60.226	HTTP	320	HTTP/1.1 304 Not Modified
1253	18.154626	10.136.60.226	10.130.101.102	TCP	54	10054 → 6200 [ACK] Seq=1094 Ack=533 Win=130816 Len=
1262	18.280930	10.136.60.226	10.130.101.102	HTTP	447	GET /audio.mp3 HTTP/1.1

1. 三次握手分析

Source	Destination	Protocol	Length	Info
10.136.60.226	10.130.101.102	TCP	66	10054 → 6200 [SYN] Seq=0 Win=64240 Len=0 MSS=1460 W
10.136.60.226	10.130.101.102	TCP	66	10055 → 6200 [SYN] Seq=0 Win=64240 Len=0 MSS=1460 W
10.130.101.102	10.136.60.226	TCP	66	6200 → 10054 [SYN, ACK] Seq=0 Ack=1 Win=65535 Len=0
10.130.101.102	10.136.60.226	TCP	66	6200 → 10055 [SYN, ACK] Seq=0 Ack=1 Win=65535 Len=0
10.136.60.226	10.130.101.102	TCP	54	10054 → 6200 [ACK] Seq=1 Ack=1 Win=131328 Len=0
10.136.60.226	10.130.101.102	TCP	54	10055 → 6200 [ACK] Seq=1 Ack=1 Win=131328 Len=0

对于红色的第一次连接：

Frame 1212: 66 bytes on wire (528 bits), 66 bytes captured (528 bits) on interface \Device\NPF_{04EBF258-82BF-48CE-9E93-CBAA0C2B8}

Ethernet II, Src: IntelCor_fb:80:6f (48:51:c5:fb:80:6f), Dst: IETF-VRRP-VRID_08 (00:00:5e:00:01:08)

Internet Protocol Version 4, Src: 10.136.60.226, Dst: 10.130.101.102

Transmission Control Protocol, Src Port: 10054, Dst Port: 6200, Seq: 0, Len: 0

Source Port: 10054

Destination Port: 6200

[Stream index: 19]

[TCP Segment Len: 0]

Sequence Number: 0 (relative sequence number)

Sequence Number (raw): 906184286

[Next Sequence Number: 1 (relative sequence number)]

Acknowledgment Number: 0

Acknowledgment number (raw): 0

1000 = Header Length: 32 bytes (8)

Flags: 0x002 (SYN)

000. = Reserved: Not set

...0 = Nonce: Not set

...0... = Congestion Window Reduced (CWR): Not set

....0... = ECN-Echo: Not set

......0. = Urgent: Not set

.......0 = Acknowledgment: Not set

.......0... = Push: Not set

.......0... = Reset: Not set

.......0... = Syn: Set

.......0... = Fin: Not set

[TCP Flags:S.]

Window: 64240

[Calculated window size: 64240]

No.: 1212 • Time: 17.842101 • Source: 10.136.60.226 • Destination: 10.130.101.102 • Protocol: TCP • Length: 66 • Info: 10054 → 6200 [SYN] Seq=0 Win=64240 Len=0 MSS=1460 WS=256 SACK_PERM=1

CloseHelp

第一次连接是客户端主动要连接服务端的，可以看到传输控制协议里Seq=0(Seq是序列号)，代表初次连接，Ack=0(确认码)，初次连接为0

除了此之外，还要给标志位，也就是flags=初次连接需要给SYN=1的标志位表示请求建立连接。

对于黄色的第二次连接：

```
> Frame 1214: 66 bytes on wire (528 bits), 66 bytes captured (528 bits) on interface \Device\NPF_{04EBF258-82BF-48CE-9E93-CBAA0C2B8...
> Ethernet II, Src: IETF-VRRP-VRID_08 (00:00:5e:00:01:08), Dst: IntelCor_fb:80:6f (48:51:c5:fb:80:6f)
> Internet Protocol Version 4, Src: 10.130.101.102, Dst: 10.136.60.226
< Transmission Control Protocol, Src Port: 6200, Dst Port: 10054, Seq: 0, Ack: 1, Len: 0
  Source Port: 6200
  Destination Port: 10054
  [Stream index: 19]
  [TCP Segment Len: 0]
  Sequence Number: 0 (relative sequence number)
  Sequence Number (raw): 3174381848
  [Next Sequence Number: 1 (relative sequence number)]
  Acknowledgment Number: 1 (relative ack number)
  Acknowledgment number (raw): 906184287
  1000 .... = Header Length: 32 bytes (8)
  Flags: 0x012 (SYN, ACK)
    000. .... = Reserved: Not set
    ...0 .... = Nonce: Not set
    ....0... = Congestion Window Reduced (CWR): Not set
    ....0... = ECN-Echo: Not set
    ....0... = Urgent: Not set
    ....1... = Acknowledgment: Set
    ....0... = Push: Not set
    ....0... = Reset: Not set
    ....1... = Syn: Set
    ....0... = Fin: Not set
  [TCP Flags: .....A..S.]
  Window: 65535
  [Calculated window size: 65535]
```

No.: 1214 • Time: 17.851007 • Source: 10.130.101.102 • Destination: 10.136.60.226 • Protocol: TCP • Length: 66 • Info: 6200 → 10054 [SYN, ACK] Seq=0 Ack=1 Win=65535 Len=0 MSS=1460 WS=256 SACK_PERM=1

Close Help

第二次握手是服务端的回馈6200端口给10054/10055的端口数据，初次连接所以Seq=0，Ack=上一次客户端的序列号+1；

标志位是SYN=1和ACK=1，代表这是一个确认的回馈连接

对于蓝色的第三次连接：

```
> Frame 1218: 54 bytes on wire (432 bits), 54 bytes captured (432 bits) on interface \Device\NPF_{04EBF258-82BF-48CE-9E93-CBAA0C2B8...
> Ethernet II, Src: IntelCor_fb:80:6f (48:51:c5:fb:80:6f), Dst: IETF-VRRP-VRID_08 (00:00:5e:00:01:08)
> Internet Protocol Version 4, Src: 10.136.60.226, Dst: 10.130.101.102
< Transmission Control Protocol, Src Port: 10054, Dst Port: 6200, Seq: 1, Ack: 1, Len: 0
  Source Port: 10054
  Destination Port: 6200
  [Stream index: 19]
  [TCP Segment Len: 0]
  Sequence Number: 1 (relative sequence number)
  Sequence Number (raw): 906184287
  [Next Sequence Number: 1 (relative sequence number)]
  Acknowledgment Number: 1 (relative ack number)
  Acknowledgment number (raw): 3174381849
  0101 .... = Header Length: 20 bytes (5)
  Flags: 0x010 (ACK)
    000. .... = Reserved: Not set
    ...0 .... = Nonce: Not set
    ....0... = Congestion Window Reduced (CWR): Not set
    ....0... = ECN-Echo: Not set
    ....0... = Urgent: Not set
    ....1... = Acknowledgment: Set
    ....0... = Push: Not set
    ....0... = Reset: Not set
    ....0... = Syn: Not set
    ....0... = Fin: Not set
  [TCP Flags: .....A....]
  Window: 513
  [Calculated window size: 1313281]
```

Close Help

客户端1004/10055给6200服务端的反馈，Seq=1,因为这是客户端的第二次交互了，Ack=上一次服务端连接的序列号+1

标志位为：ACK，表示确认收到了连接回复，三次握手就建立连接完毕

2. HTTP分析

1220	17.878511	10.136.60.226	10.130.101.102	HTTP	641 GET / HTTP/1.1
1222	17.894323	10.130.101.102	10.136.60.226	HTTP	320 HTTP/1.1 304 Not Modified
1227	17.940565	10.136.60.226	10.130.101.102	TCP	54 10054 → 6200 [ACK] Seq=588 Ack=267 Win=131072 Len=0
1234	18.105384	10.136.60.226	10.130.101.102	HTTP	560 GET /icon.jpg HTTP/1.1
1235	18.113060	10.130.101.102	10.136.60.226	HTTP	320 HTTP/1.1 304 Not Modified
1253	18.154626	10.136.60.226	10.130.101.102	TCP	54 10054 → 6200 [ACK] Seq=1094 Ack=533 Win=130816 Len=0
1262	18.280930	10.136.60.226	10.130.101.102	HTTP	447 GET /audio.mp3 HTTP/1.1

对于第一个HTTP的GET请求，是建立HTTP连接，传输HTML页面内容

对于两个 304 Not Modified，意味着客户端缓存的资源仍然是最新的，并且与服务器上的资源相同，服务器将返回 HTTP 304 Not Modified 状态码。这表示服务器并没有返回请求的资源，而是返回一个空包和 304 状态码告诉客户端它可以继续使用它们本地缓存的内容。这对于减少网络流量和提高性能非常有用，因为此时客户端可以避免不必要的数据传输。

对于GET /icon.jpg 和 GET /audio.mp3 是将web中展示的图片 and 音频请求下来

3. 四次挥手分析

1946	23.323235	10.130.101.102	10.136.60.226	TCP	60 6200 → 10054 [FIN, ACK] Seq=194972 Ack=1487 Win=104
1947	23.323315	10.136.60.226	10.130.101.102	TCP	54 10054 → 6200 [ACK] Seq=1487 Ack=194973 Win=131328 Len=0
1952	23.339180	10.130.101.102	10.136.60.226	TCP	60 6200 → 10055 [FIN, ACK] Seq=194440 Ack=394 Win=1049
1953	23.339275	10.136.60.226	10.130.101.102	TCP	54 10055 → 6200 [ACK] Seq=394 Ack=194441 Win=131328 Len=0

- 在第一次挥手中，客户端发送了一个带有 [FIN, ACK] 标志位的包给服务器，表示客户端已经完成数据的发送，并要求关闭连接。同时，客户端还确认了服务器发送的序列号为1487的数据。
- 服务器接收到客户端的第一次挥手后，发送一个带有 [ACK] 标志位的确认包给客户端，确认客户端的请求，并告知客户端服务器端已经准备好关闭连接。
- 客户端接收到服务器的确认后，发送一个带有 [FIN, ACK] 标志位的包给服务器，表示客户端也准备关闭连接。
- 服务器接收到客户端的第三次挥手后，发送一个带有 [ACK] 标志位的确认包给客户端，确认客户端的请求，并告知客户端服务器端也准备好关闭连接。

通过以上四次挥手，双方成功地关闭了TCP连接。

四、Web运行结果展示

