南开大学

Nankai University

南　　开　　大　　学

计算机与网络空间安全学院

计算机网路实验报告

实验二：建立简单的 Web 服务器与 Wireshark 分析

2011763 黄天昊

年级：2020 级

指导教师：张建忠，徐敬东

2022 年 10 月 28 日

摘要

在本次实验中，编写了简单的 HTML 静态网页页面文档，初步了解认识了超文本标记语言的语法与语义。使用了 Python 中的 Django 环境在本机上搭建了简单的 Web 服务器。（IP 地址为 127.0.0.1）在搭建好的 Web 服务器的上通过 Wireshark 对本机进行抓包，设置合适的过滤器可以捕获完整的 Web 服务器与浏览器的 Http 协议交互。

关键字：HTML，Django，Wireshark，http/1.1

目录

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）和自己的 LOGO。
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。

二、 Web 页面设计

在本次设计中，主要设计了一个静态的 HTML 网页来展示个人信息。将 HTML 中的 img、div 等部分的 style 格式保留在 CSS 格式文件对页面进行优化。通过 link 使得 HTML 文档应用该 CSS 文件，最终的网页展示效果包括动态效果和静态信息、Logo 展示。

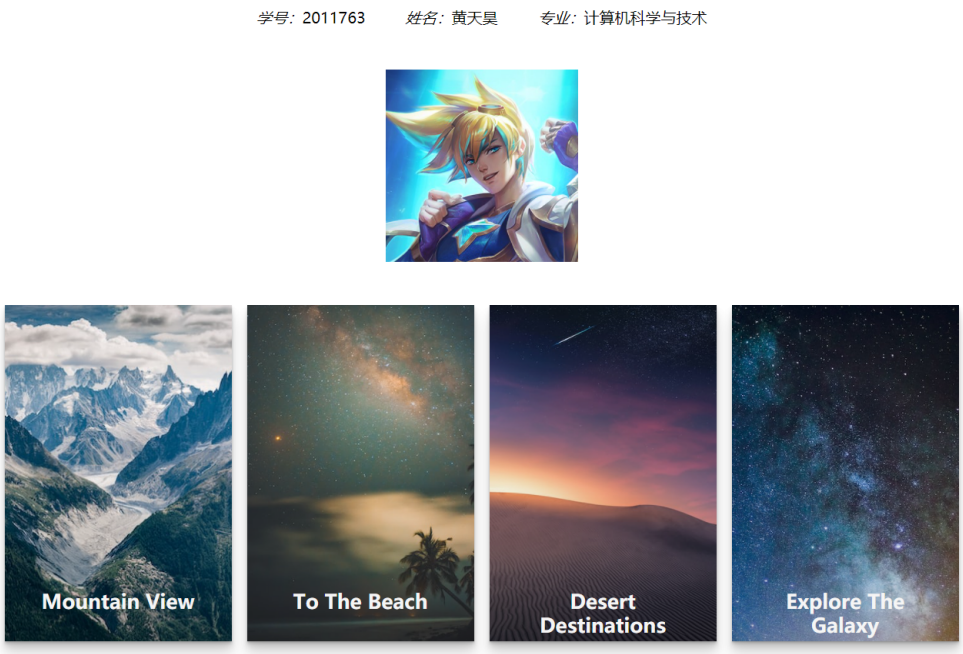

图 1: 网页设计展示

具体代码如下:

HTML 代码

```
<link rel="stylesheet" type="text/css" href="../static/css/MyCSS.css">

<p><em>学号: </em>2011763    <em>姓名: </em>黄天昊    <em>专业: </em>计算机科学与技术</p>
<img src="../static/images/StarGuardian_Ezreal_Splash.jpg" alt="LOGO" width="200" height="200">


<main class="page-content">
  <div class="card">
```

```
    <div class="content">
      <h2 class="title">Mountain View</h2>
      <p class="copy">Check out all of these gorgeous mountain trips with
          beautiful views of, you guessed it, the mountains</p>
      <button class="btn" onclick="window.open('https://github.com/Skyyyy0920
          ?tab=repositories')">View Trips</button>
    </div>
  </div>
  <div class="card">
    <div class="content">
      <h2 class="title">To The Beach</h2>
      <p class="copy">Plan your next beach trip with these fabulous
          destinations</p>
      <button class="btn">View Trips</button>
    </div>
  </div>
  <div class="card">
    <div class="content">
      <h2 class="title">Desert Destinations</h2>
      <p class="copy">It's the desert you've always dreamed of</p>
      <button class="btn">Book Now</button>
    </div>
  </div>
  <div class="card">
    <div class="content">
      <h2 class="title">Explore The Galaxy</h2>
      <p class="copy">Seriously, straight up, just blast off into outer space
          today</p>
      <button class="btn">Book Now</button>
    </div>
  </div>
</main>
```

CSS 代码

```
:root {
  --d: 700ms;
  --e: cubic-bezier(0.19, 1, 0.22, 1);
  --font-sans: "Rubik", sans-serif;
  --font-serif: "Cardo", serif;
}

* {
  box-sizing: border-box;
}

html, body {
  height: 100%;
}
```

```
15
16 body {
17   display: grid;
18   place-items: center;
19 }
20
21 .page-content {
22   display: grid;
23   grid-gap: 1rem;
24   padding: 1rem;
25   max-width: 1024px;
26   margin: 0 auto;
27   font-family: var(--font-sans);
28 }
29 @media (min-width: 600px) {
30   .page-content {
31     grid-template-columns: repeat(2, 1fr);
32   }
33 }
34 @media (min-width: 800px) {
35   .page-content {
36     grid-template-columns: repeat(4, 1fr);
37   }
38 }
39
40 .card {
41   position: relative;
42   display: flex;
43   align-items: flex-end;
44   overflow: hidden;
45   padding: 1rem;
46   width: 100%;
47   text-align: center;
48   color: whitesmoke;
49   background-color: whitesmoke;
50   box-shadow: 0 1px 1px rgba(0, 0, 0, 0.1), 0 2px 2px rgba(0, 0, 0, 0.1), 0 4
       px 4px rgba(0, 0, 0, 0.1), 0 8px 8px rgba(0, 0, 0, 0.1), 0 16px 16px
       rgba(0, 0, 0, 0.1);
51 }
52 @media (min-width: 600px) {
53   .card {
54     height: 350px;
55   }
56 }
57 .card:before {
58   content: "";
59   position: absolute;
60   top: 0;
```

```
  left: 0;
  width: 100%;
  height: 110%;
  background-size: cover;
  background-position: 0 0;
  transition: transform calc(var(--d) * 1.5) var(--e);
  pointer-events: none;
}
.card:after {
  content: "";
  display: block;
  position: absolute;
  top: 0;
  left: 0;
  width: 100%;
  height: 200%;
  pointer-events: none;
  background-image: linear-gradient(to bottom, rgba(0, 0, 0, 0) 0%, rgba(0,
      0, 0, 0.009) 11.7%, rgba(0, 0, 0, 0.034) 22.1%, rgba(0, 0, 0, 0.072)
      31.2%, rgba(0, 0, 0, 0.123) 39.4%, rgba(0, 0, 0, 0.182) 46.6%, rgba(0,
      0, 0, 0.249) 53.1%, rgba(0, 0, 0, 0.32) 58.9%, rgba(0, 0, 0, 0.394)
      64.3%, rgba(0, 0, 0, 0.468) 69.3%, rgba(0, 0, 0, 0.54) 74.1%, rgba(0,
      0, 0, 0.607) 78.8%, rgba(0, 0, 0, 0.668) 83.6%, rgba(0, 0, 0, 0.721)
      88.7%, rgba(0, 0, 0, 0.762) 94.1%, rgba(0, 0, 0, 0.79) 100%);
  transform: translateY(-50%);
  transition: transform calc(var(--d) * 2) var(--e);
}
.card:nth-child(1):before {
  background-image: url('../images/Mountain.jfif');
}
.card:nth-child(2):before {
  background-image: url(../images/Beach.jfif);
}
.card:nth-child(3):before {
  background-image: url(../images/Desert.jfif);
}
.card:nth-child(4):before {
  background-image: url(../images/Galaxy.jfif);
}

.content {
  position: relative;
  display: flex;
  flex-direction: column;
  align-items: center;
  width: 100%;
  padding: 1rem;
  transition: transform var(--d) var(--e);
```

```
  z-index: 1;
}
.content > * + * {
  margin-top: 1rem;
}

.title {
  font-size: 1.3rem;
  font-weight: bold;
  line-height: 1.2;
}

.copy {
  font-family: var(--font-serif);
  font-size: 1.125rem;
  font-style: italic;
  line-height: 1.35;
}

.btn {
  cursor: pointer;
  margin-top: 1.5rem;
  padding: 0.75rem 1.5rem;
  font-size: 0.65rem;
  font-weight: bold;
  letter-spacing: 0.025rem;
  text-transform: uppercase;
  color: white;
  background-color: black;
  border: none;
}
.btn:hover {
  background-color: #0d0d0d;
}
.btn:focus {
  outline: 1px dashed yellow;
  outline-offset: 3px;
}

@media (hover: hover) and (min-width: 600px) {
  .card:after {
    transform: translateY(0);
  }

  .content {
    transform: translateY(calc(100% - 4.5rem));
  }
  .content > *:not(.title) {
```

```
    opacity: 0;
    transform: translateY(1rem);
    transition: transform var(--d) var(--e), opacity var(--d) var(--e);
  }

  .card:hover,
.card:focus-within {
    align-items: center;
  }
  .card:hover:before,
.card:focus-within:before {
    transform: translateY(-4%);
  }
  .card:hover:after,
.card:focus-within:after {
    transform: translateY(-50%);
  }
  .card:hover .content,
.card:focus-within .content {
    transform: translateY(0);
  }
  .card:hover .content > *:not(.title),
.card:focus-within .content > *:not(.title) {
    opacity: 1;
    transform: translateY(0);
    transition-delay: calc(var(--d) / 8);
  }

  .card:focus-within:before, .card:focus-within:after,
.card:focus-within .content,
.card:focus-within .content > *:not(.title) {
    transition-duration: 0s;
  }
}
```


三、 Web 服务器搭建

这里我使用的框架是 Django

Django 是高水准的 Python 编程语言驱动的一个开源模型。视图，控制器风格的 Web 应用程序框架，它起源于开源社区。使用这种架构，程序员可以方便、快捷地创建高品质、易维护、数据库驱动的应用程序。这也正是 OpenStack 的 Horizon 组件采用这种架构进行设计的主要原因。另外，在 Django 框架中，还包含许多功能强大的第三方插件，使得 Django 具有较强的可扩展性。Django 项目源自一个在线新闻 Web 站点，于 2005 年以开源的形式被释放出来。

Django 框架的核心组件有:

1. 用于创建模型的对象关系映射
2. 为最终用户设计较好的管理界面

3. URL 设计
4. 设计者友好的模板语言
5. 缓存系统

代码如下:

urls.py

```python
from django.contrib import admin
from django.urls import path
from . import views

urlpatterns = [
    path('admin/', admin.site.urls),

    path('test/', views.test),
]
```

settings.py

```python
from django.shortcuts import HttpResponse, render


def test(request):
    return render(request, "MyHTML.html")
```

views.py

```python
from pathlib import Path
import os

# Build paths inside the project like this: BASE_DIR / 'subdir'.
BASE_DIR = Path(__file__).resolve().parent.parent

# Quick-start development settings - unsuitable for production
# See https://docs.djangoproject.com/en/3.2/howto/deployment/checklist/

# SECURITY WARNING: keep the secret key used in production secret!
SECRET_KEY = 'django-insecure-lh^*500syi)a8k-n30od!4)ya(rwj3iuiz$3-)2&2
    @1xs6hw4#'

# SECURITY WARNING: don't run with debug turned on in production!
DEBUG = True

ALLOWED_HOSTS = []

# Application definition

```

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

MIDDLEWARE = [
    'django.middleware.security.SecurityMiddleware',
    'django.contrib.sessions.middleware.SessionMiddleware',
    'django.middleware.common.CommonMiddleware',
    'django.middleware.csrf.CsrfViewMiddleware',
    'django.contrib.auth.middleware.AuthenticationMiddleware',
    'django.contrib.messages.middleware.MessageMiddleware',
    'django.middleware.clickjacking.XFrameOptionsMiddleware',
]

ROOT_URLCONF = 'djangoProject.urls'

TEMPLATES = [
    {
        'BACKEND': 'django.template.backends.django.DjangoTemplates',
        'DIRS': [BASE_DIR / 'templates']
        ,
        'APP_DIRS': True,
        'OPTIONS': {
            'context_processors': [
                'django.template.context_processors.debug',
                'django.template.context_processors.request',
                'django.contrib.auth.context_processors.auth',
                'django.contrib.messages.context_processors.messages',
            ],
        },
    },
]

WSGI_APPLICATION = 'djangoProject.wsgi.application'

# Database
# https://docs.djangoproject.com/en/3.2/ref/settings/#databases

DATABASES = {
    'default': {
        'ENGINE': 'django.db.backends.sqlite3',
        'NAME': BASE_DIR / 'db.sqlite3',
    }
```

```
68 }
69
70 # Password validation
71 # https://docs.djangoproject.com/en/3.2/ref/settings/#auth-password-
       validators
72
73 AUTH_PASSWORD_VALIDATORS = [
74     {
75         'NAME': 'django.contrib.auth.password_validation.
              UserAttributeSimilarityValidator',
76     },
77     {
78         'NAME': 'django.contrib.auth.password_validation.
              MinimumLengthValidator',
79     },
80     {
81         'NAME': 'django.contrib.auth.password_validation.
              CommonPasswordValidator',
82     },
83     {
84         'NAME': 'django.contrib.auth.password_validation.
              NumericPasswordValidator',
85     },
86 ]
87
88 # Internationalization
89 # https://docs.djangoproject.com/en/3.2/topics/i18n/
90
91 LANGUAGE_CODE = 'en-us'
92
93 TIME_ZONE = 'UTC'
94
95 USE_I18N = True
96
97 USE_L10N = True
98
99 USE_TZ = True
100
101 # Static files (CSS, JavaScript, Images)
102 # https://docs.djangoproject.com/en/3.2/howto/static-files/
103
104 STATIC_URL = '/static/'
105
106 # Default primary key field type
107 # https://docs.djangoproject.com/en/3.2/ref/settings/#default-auto-field
108
109 DEFAULT_AUTO_FIELD = 'django.db.models.BigAutoField'
110
```

```
111 STATICFILES_DIRS = [
112     os.path.join(BASE_DIR, 'static')]
```

四、 Wireshark 捕获分析

为实现本地回环地址抓包，需使用 Wireshark 的 Adapter for loopback traffic capture 功能，因此先安装 npcap。之后再打开 Wireshark 开始捕获。因为本地还有很多与其他服务器交互的服务，所以会有许多我们不希望看到的数据包，这时我们就需要通过设置合适的过滤器来捕获我们希望分析的数据包，因为在本次设计之中，设计的 TCP 端口为 8000，所以过滤器设置为 tcp.port == 8000 就可以成功只捕获自己想要的数据包了，用浏览器访问 http://127.0.0.1:8000/test 进行捕获，关闭网页之后得到的捕获结果如下：

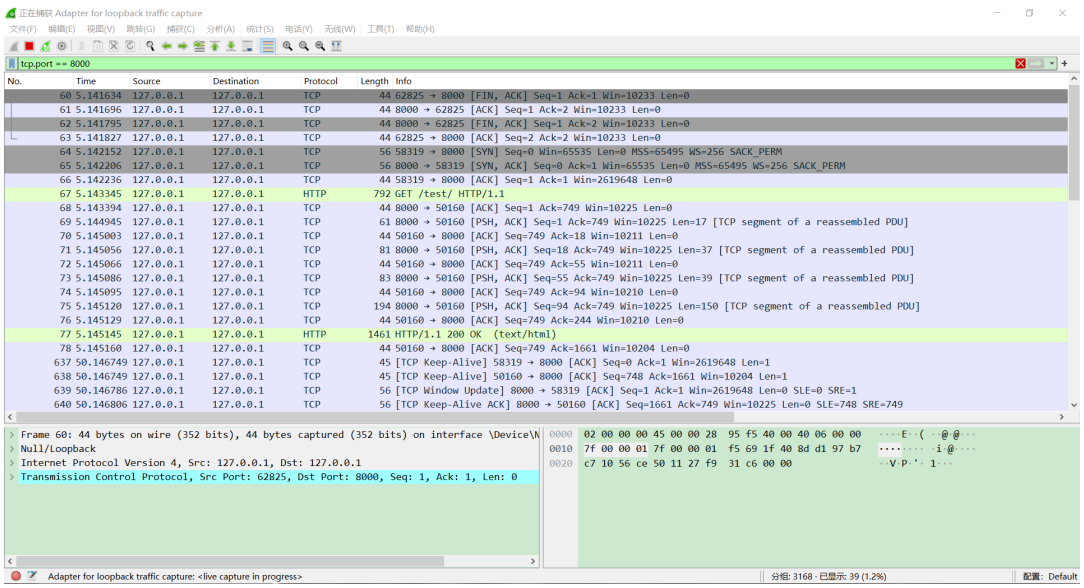

图 2: Wireshark 捕获结果

HTTP 交互整体流程：

1. 地址解析
   从中分解出协议名、主机名、端口、对象路径等部分。同时需要域名系统 DNS 解析域名，得到主机的 IP 地址。

2. 封装 HTTP 请求数据包
   把主机名、端口号等信息结合本机自己的信息，封装成一个 HTTP 请求数据包。

3. 封装成 TCP 包，TCP 三次握手建立连接
   在 HTTP 工作开始之前，客户机（Web 浏览器）首先要通过网络与服务器建立连接，该连接是通过 TCP 来完成的，该协议与 IP 协议共同构建 Internet，即著名的 TCP/IP 协议族。

4. 客户端向服务器发送请求命令
   建立 TCP 连接后，客户机发送一个请求给服务器，请求方式的格式为：统一资源标识符（URL）、协议版本号，后边是 MIME 信息包括请求修饰符、客户机信息和可内容。

5. 服务器响应
   服务器接到请求后，给予相应的响应信息，其格式为一个状态行，包括信息的协议版本号、一个成功或错误的代码，后边是 MIME 信息包括服务器信息、实体信息和可能的内容。

6. TCP 四次挥手断开连接
   在断开连接之前客户端和服务器都处于 ESTABLISHED 状态，双方都可以主动断开连接，以客户端主动断开连接为优。

(一) TCP 三次握手建立连接

使用 TCP 协议进行通信的双方必须先建立连接，然后才能开始传输数据。为了确保连接双方可靠性（要确认双方的收发能力都是正常的），在双方建立连接时，TCP 协议采用了三次握手策略：

- 第一次握手，客户端发送给服务器，服务器能收到，那么这个时候服务端能确定客户端的发送能力正常，服务端的接受能力正常。
- 第二次握手，客户端确认服务端的接受能力正常（服务端接收到了我的包），发送能力也是正常的（服务端发来了新的），也知道了自己的发送能力是正常的（服务端接收到了我的包）
- 第三次握手，因为服务端只知道客户端的发送能力和自己的接受能力正常，通过第三次握手，服务端知道客户端的接收能力正常（客户端对服务端有了第二次的回应），服务端的发送能力正常（发出去了）

下图展示了 TCP 三次握手具体实现过程（图片来源）：

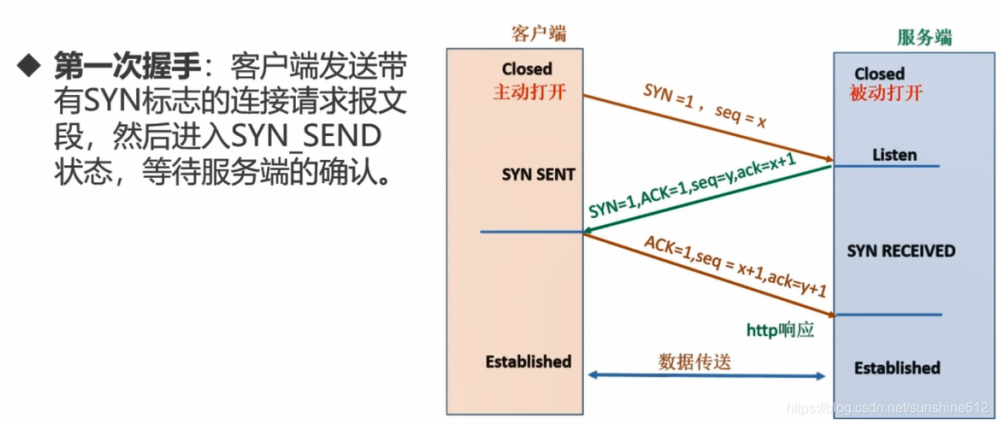

图 3: 第一次握手

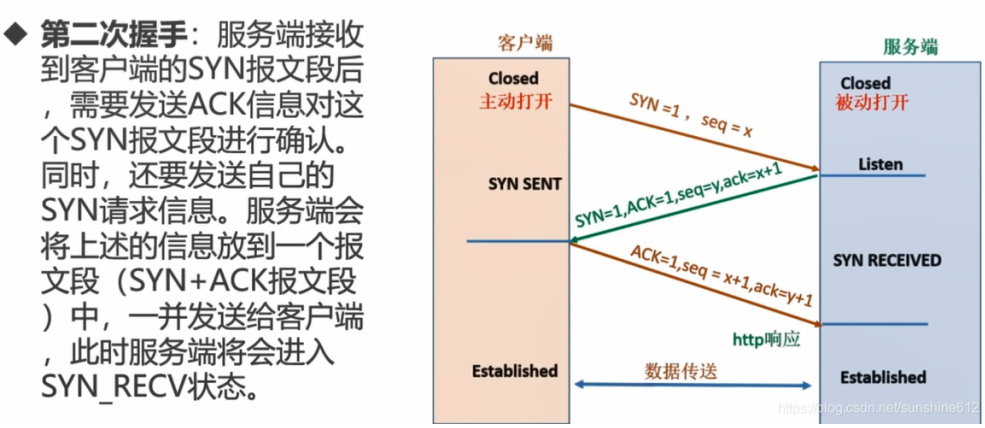

图 4: 第二次握手

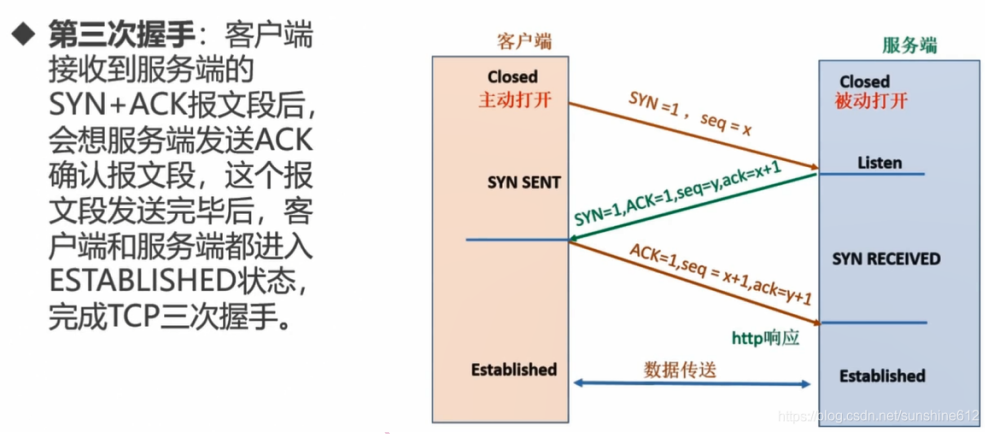

图 5: 第三次握手

关于 TCP 三次握手的细节：

1. 客户端给服务器发送一个 SYN 段 (在 TCP 标头中 SYN 位字段为 1 的 TCP/IP 数据包)，该段中也包含客户端的初始序列号 (Sequence number = J)。

2. 服务器返回客户端 SYN +ACK 段 (在 TCP 标头中 SYN 和 ACK 位字段都为 1 的 TCP/IP 数据包)，该段中包含服务器的初始序列号 (Sequence number = K)；同时使 Acknowledgment number = J + 1 来表示确认已收到客户端的 SYN 段 (Sequence number = J)。

3. 客户端给服务器响应一个 ACK 段 (在 TCP 标头中 ACK 位字段为 1 的 TCP/IP 数据包)，该段中使 Acknowledgment number = K + 1 来表示确认已收到服务器的 SYN 段 (Sequence number = K)。

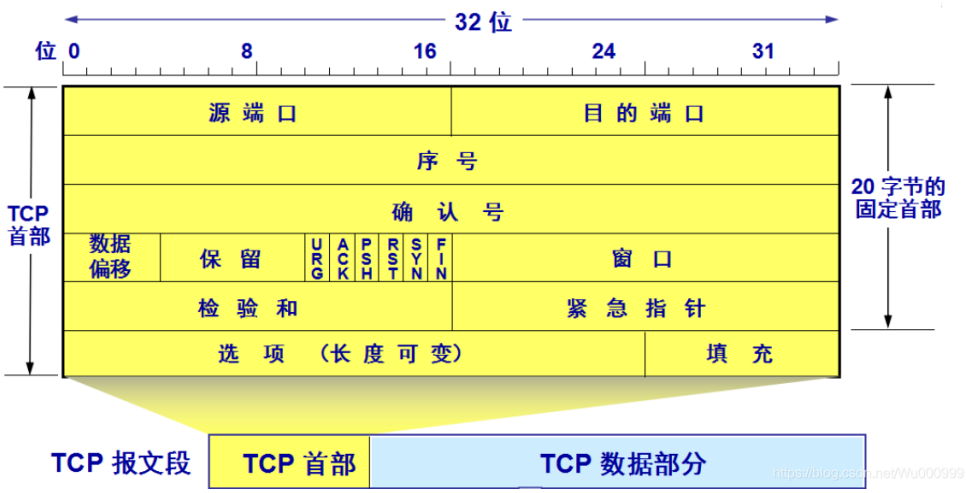

图 6: TCP 首部

如图7，展示了源端口号与目的端口号：

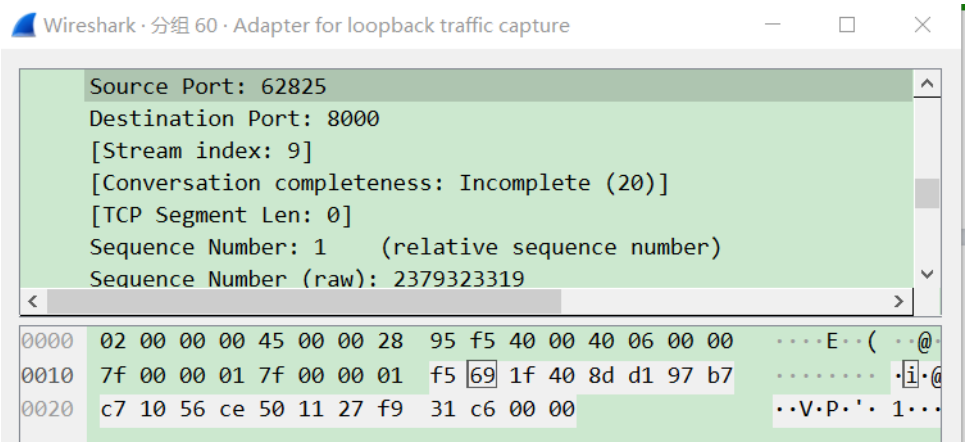

图 7: 第一次握手 TCP 报文首部

如图8，这里可以看到 Sequence Number 为 2379323319

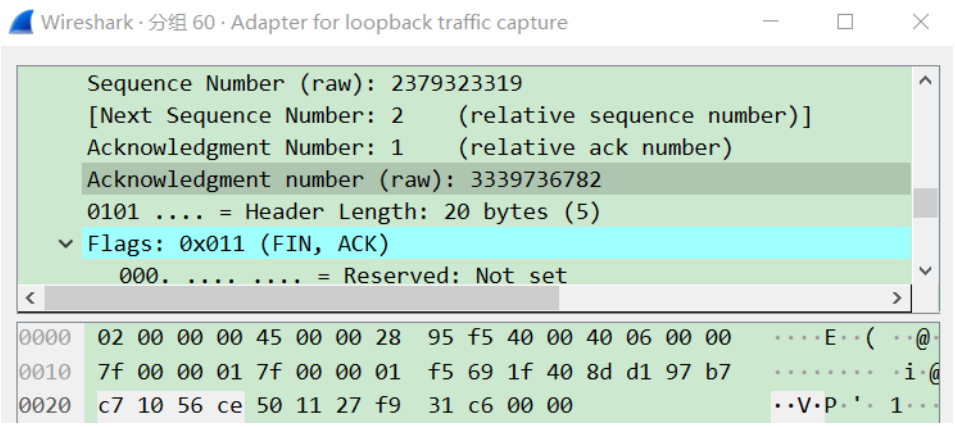

图 8: 第一次握手 TCP 报文首部

如图9，可以看到第二次握手的 TCP 报文首部的 Acknowledge Number 为 2379323320，即 Sequence Number + 1

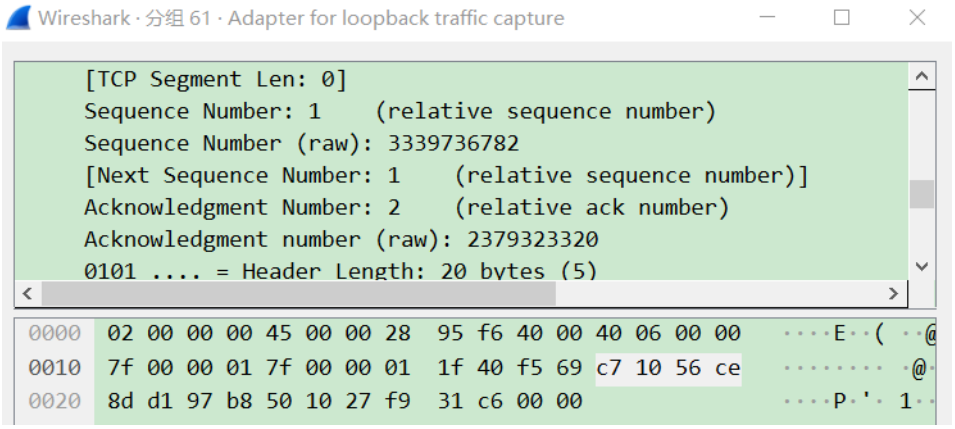

图 9: 第二次握手 TCP 报文首部

如图10，可以看到第一次握手的 TCP 报文首部的 ACK 标志位为 0，SYN 位为 1

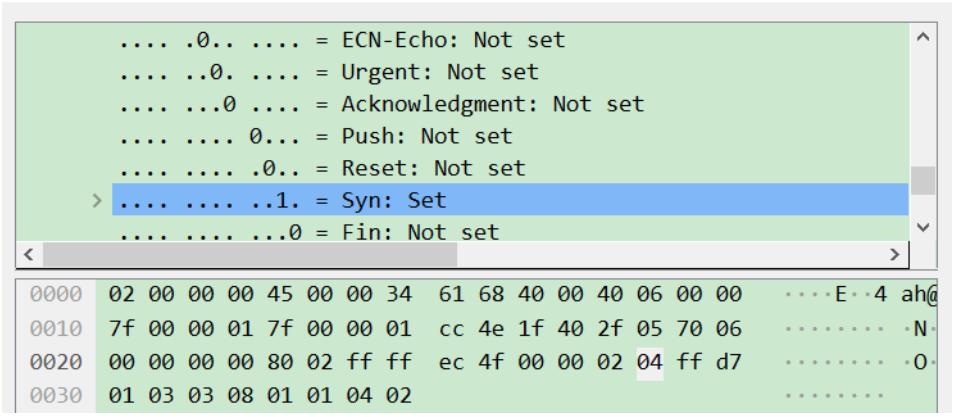

图 10: 第一次握手 TCP 报文首部

如图11，可以看到第二次握手的 TCP 报文首部的 ACK 位为 1，SYN 位也为 1

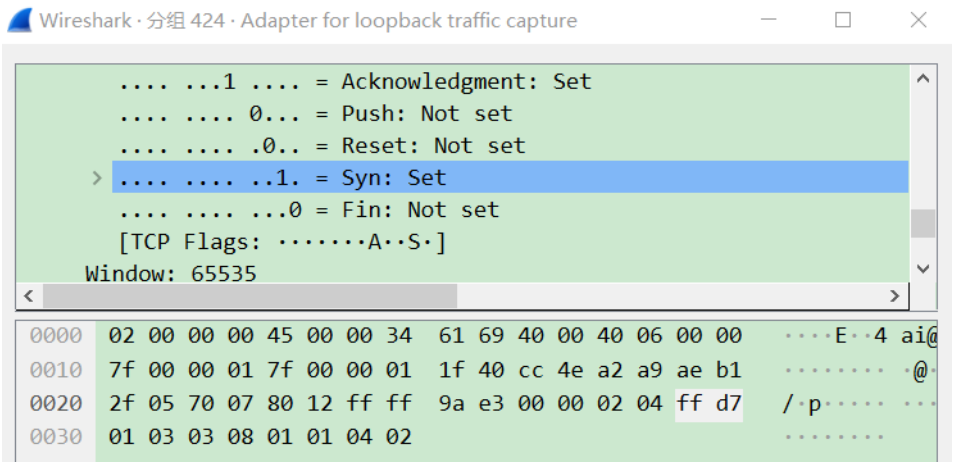

图 11: 第二次握手 TCP 报文首部

(二) HTTP 请求交互

HTTP 请求报文格式如下:

图 12: HTTP 请求报文格式

HTTP 请求是指从客户端到服务器端的请求消息。包括：消息首行中，对资源的请求方法、资源的标识符及使用的协议。

有 GET，POST，HEAD 等类型：GET 方法是获取 URL 指定资源。使用 GET 方法时，可以将请求参数和对应的值附加在 URL 后面 POST 请求一般是客户端提交给服务器的表单数据。

当然，如果是上传文件，也应当使用 POST 请求。POST 请求还可以较 GET 请求更好的安全性。HEAD 方法与 GET 用法相同，但没有响应实体。需要注意的是，我们在 Wireshark 捕获到的带有 Protocol 信息为 HTTP 的数据报文，通常都包含了其从 HTTP 报文转换为 TCP 数据包，再经过 IP 网络层封装的过程，因为在本次设计之中，使用的是本机的 IP 地址即 127.0.0.1，所以 Wireshark 中捕获到的数据包的封装结构只会延续到 IP 层，我们不再需要 IP 层去寻找一个前往目标 IP 的路径，因为是本机访问本机 IP。

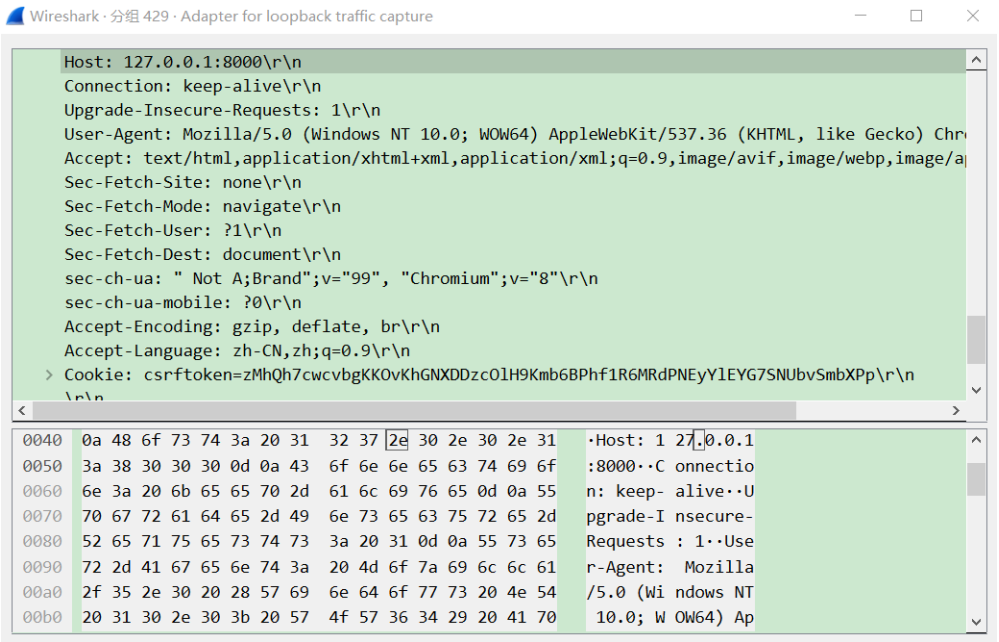

图 13: HTTP 报文

有趣的是，这里出现了很多 keep alive，如图14

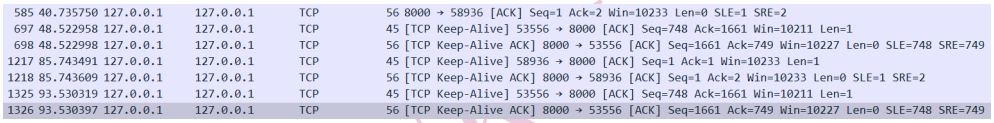

图 14: TCP KeepAlive

这里介绍一下 TCP KeepAlive：

TCP 长连接下，客户端和服务器若长时间无数据交互情况下，若一方出现异常情况关闭连接，抑或是连接中间路由出于某种机制断开连接，而此时另一方不知道对方状态而一直维护连接，浪费系统资源的同时，也会引起下次数据交互时出错。

为了解决此问题，引入了 TCP KeepAlive 机制（并非标准规范，但操作系统一旦实现，默认情况下须为关闭，可以被上层应用开启和关闭）。其基本原理是在此机制开启时，当长连接无数据交互一定时间间隔时，连接的一方会向对方发送保活探测包，如连接仍正常，对方将对此确认回应。

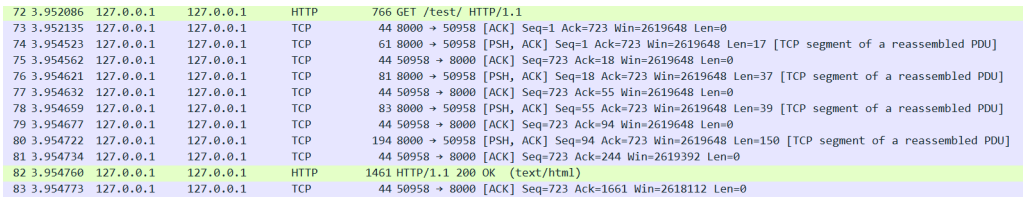

图 15: HTTP 报文

这里可以看到，当客户端请求 html 文件之后，服务器端将其切分成多个 TCP 包传输，当所有包都接收到以后，客户端发送 ACK 确认收到。

这里一开始我没抓到请求图片和 css 那些静态文件的包，百思不得其解，第二天又抓了一次，结果抓到了，如图16

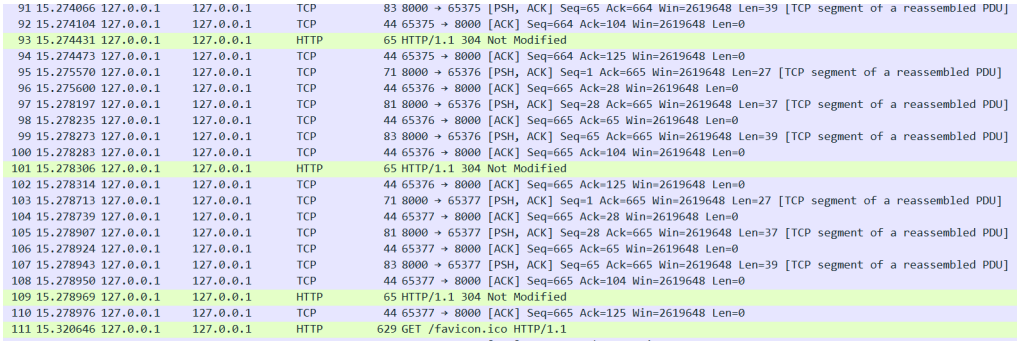

图 16: HTTP 报文

这里通过查阅资料推测有以下两种可能的原因:

1. Django MemCache 缓存。缓存将一个某个 views 视图函数的返回值保存至内存或者 memcached 中，若在时间约定范围内该用户又对此视图发起了请求，则不再去执行 views 中的操作，而是直接从内存或者 Redis 中获取之前已经缓存的数据，并将其返回给浏览器，这也是动态网站使用缓存的常用流程。(详见：Django 中的缓存机制及其实现方法)

2. 与 Django 管理静态文件的机制有关，Django 默认会从 settings.py 中设置的路径中查找静态文件，此时如果浏览器缓存了该文件，Django 将不会去请求该文件。(这部分是我自己推测的)(详见：管理静态文件（比如图片、JavaScript、CSS）| Django 官方文档)

这里，请求 css 以及图片文件返回的是 304 状态码，意思是浏览器已预先缓存。
实验中的三种不同 HTTP 响应状态码:

1. 状态码 200
   OK：请求成功，一般用于 GET 与 POST 请求。

2. 状态码 304
   Not Modified：未修改。所请求的资源未修改，服务器返回此状态码时，不会返回任何资源。客户端通常会缓存访问过的资源，通过提供一个头信息指出客户端希望只返回在指定日期之后修改的资源。

3. 状态码 404
   Not Found：服务器无法根据客户端的请求找到资源（网页）。通过此代码，网站设计人员可设置”您所请求的资源无法找到”的个性页面。

(三) TCP 四次挥手断开连接

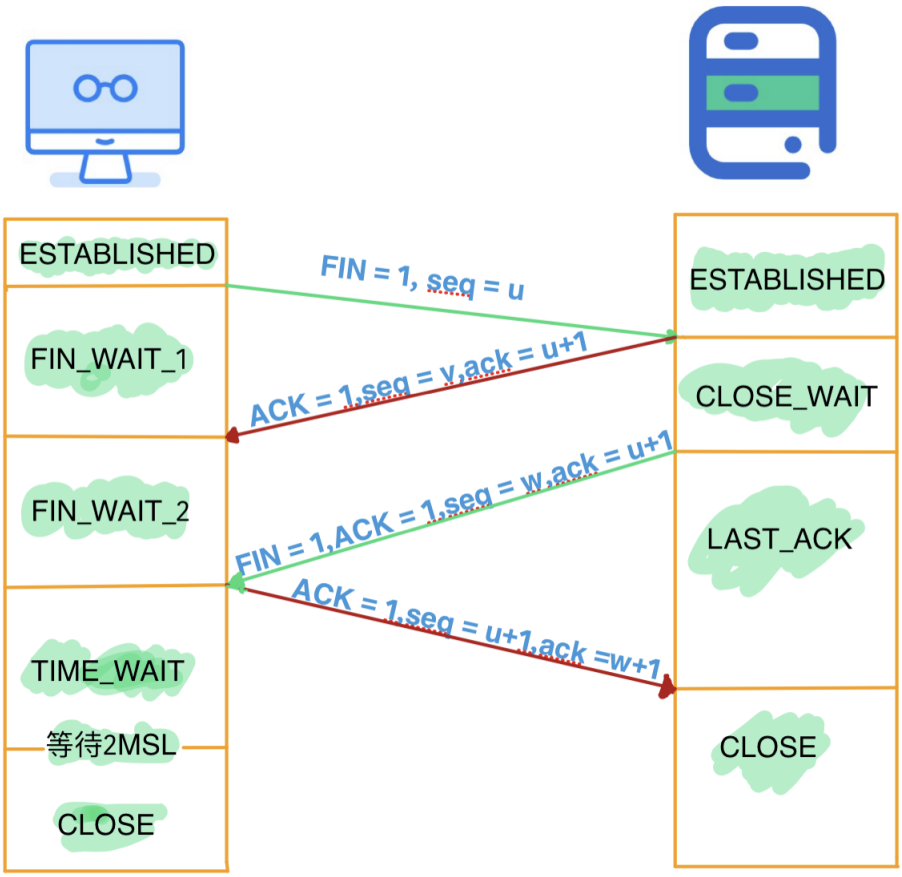

图 17: TCP 四次挥手过程

在断开连接之前客户端和服务器都处于 ESTABLISHED 状态，双方都可以主动断开连接，以客户端主动断开连接为优。

1. 第一次挥手
   客户端打算断开连接，向服务器发送 FIN 报文 (FIN 标记位被设置为 1，1 表示为 FIN，0 表示不是)，FIN 报文中会指定一个序列号，之后客户端进入 FIN_WAIT_1 状态。也就是客户端发出连接释放报文段 (FIN 报文)，指定序列号 seq = u，主动关闭 TCP 连接，等待服务器的确认。

2. 第二次挥手
   服务器收到连接释放报文段 (FIN 报文) 后，就向客户端发送 ACK 应答报文，以客户端的 FIN 报文的序列号 seq+1 作为 ACK 应答报文段的确认序列号 ack = seq+1 = u + 1。接着服务器进入 CLOSE_WAIT(等待关闭) 状态，此时的 TCP 处于半关闭状态，客户端到服务器的连接释放。客户端收到来自服务器的 ACK 应答报文段后，进入 FIN_WAIT_2 状态。

3. 第三次挥手
   服务器也打算断开连接,向客户端发送连接释放 (FIN) 报文段,之后服务器进入 LASK_ACK(最后确认) 状态，等待客户端的确认。服务器的连接释放 (FIN) 报文段的 FIN=1，ACK=1，序列号 seq=m，确认序列号 ack=u+1。

4. 第四次挥手
   客户端收到来自服务器的连接释放 (FIN) 报文段后，会向服务器发送一个 ACK 应答报文段，以连接释放 (FIN) 报文段的确认序号 ack 作为 ACK 应答报文段的序列号 seq，以连接释放 (FIN) 报文段的序列号 seq+1 作为确认序号 ack。之后客户端进入 TIME_WAIT(时间等待) 状态，服务器收到 ACK 应答报文段后，服务器就进入 CLOSE(关闭) 状态，到此服务器的连接已经完成关闭。
   客户端处于 TIME_WAIT 状态时，此时的 TCP 还未释放掉，需要等待 2MSL 后，客户端才进入 CLOSE 状态。

由客户端到服务器需要一个 FIN 和 ACK，再由服务器到客户端需要一个 FIN 和 ACK，因此通常被称为四次握手。

客户端和服务器都可以主动关闭连接，只有率先请求关闭的一方才会进入 TIME_WAIT(时间等待状态)。

为什么挥手需要四次?

这是由于 TCP 的半关闭 (half-close) 造成的。半关闭是指：TCP 提供了连接的一方在结束它的发送后还能接受来自另一端数据的能力。通俗来说，就是不能发送数据，但是还可以接受数据。TCP 不允许连接处于半打开状态时，就单向传输数据，因此完成三次握手后才可以传输数据 (第三次握手可以携带数据)。当连接处于半关闭状态时，TCP 是允许单向传输数据的，也就是说服务器此时仍然可以向客户端发送数据，等服务器不再发送数据时，才会发送 FIN 报文段，同意现在关闭连接。这一特性是由于 TCP 双向通道互相独立所导致的，也使得关闭连接必须经过四次握手。

为什么 TIME_WAIT 等待的时间是 2MSL?

MSL(Maximum Segment LifeTime) 是报文最大生成时间，它是任何报文在网络上存在的最长时间，超过这个时间的报文将被丢弃。因为 TCP 协议是基于 IP 协议 (位于 IP 协议的上一层)，IP 数据报中有限制其生存时间的 TTL 字段，是 IP 数据报可以经过的最大路由器的个数，每经过处理它的路由，TTL 就会减一。TTL 为 0 时还没有到达目的地的数据报将会被丢弃，同时发送 ICMP 报文通知源主机。MSL 的单位为时间，TTL 的单位为跳转数。所以 MSL 应该大于等于 TTL 变为 0 的时间，以确保报文已被丢弃。TIME_WAIT 等待的 2MSL 时间，可以理解为数据报一来一回所需要的最大时间。2MSL 时间是从客户端接收到 FIN 后发送 ACK 开始计时的。如果在这个时间段内，服务器没有收到 ACK 应答报文段，会重发 FIN 报文段，如果客户端收到了 FIN 报文段，那么 2MSL 的时间将会被重置。如果在 2MSL 时间段内，没有收到任何数据报，客户端则会进入 CLOSE 状态。

等待 2MSL 的意义

1. 保证客户端最后发送的 ACK 能够到达服务器，帮助其正常关闭。
   由于这个 ACK 报文段可能会丢失，使得处于 LAST_ACK 状态的服务器得不到对已发送 FIN 报文段的确认，从而会触发超时重传。服务器会重发 FIN 报文段，客户端能保证在 2MSL 时间内收到来自服务器的重传 FIN 报文段，从而客户端重新发送 ACK 应答报文段，并重置 2MSL 计数。
   假如客户端不等待 2MSL 就之间进入 CLOSE 状态，那么服务器会一直处于 LAST_ACK 状态。
   当客户端发起建立 SYN 报文段请求建立新的连接时，服务端会发送 RST 报文段给客户端，连接建立的过程就会被终止。

2. 防止已失效的连接请求报文段出现在本连接中。
   TIME_WAIT 等待的 2MSL 时间，确保本连接内所产生的所有报文段都从网络中消失，使下一个新的连接中不会出现这种旧的连接请求报文段。

五、 总结

通过本次实验，我对服务器和 Wireshark 捕获的相关知识有了进一步的了解，且对于 TCP 和 HTTP 的相关知识有了更深刻的理解。对于 TCP 和 HTTP 报文中的具体字段有了初步的了解，此外，我还学习了 Web 开发框架 Django 的基础运用，对整体的协议的封装与转换有了进一步的认识。

参考文献

[1] HTTP 协议简介/数据包封装/三次握手/DNS 解析

[2] TCP 四次挥手详解

[3] 管理静态文件（比如图片、JavaScript、CSS）| Django 官方文档

[4] Django 中的缓存机制及其实现方法