南開大學
Nankai University

南　　开　　大　　学

计　算　机　学　院

实验报告

简单浏览器测试程序报告

姓名：林雪

学号：2120220589

专业：计算机科学与技术

2023 年 1 月 1 日

目录

一、 网络连通测试程序报告

(一) 编程环境

windows x86_64、Python 3.11.0、PyQt5-5.15.7、PyQt5-Qt-5.15.2、PyQt5-sip 12.11.0
PyQtWebEngine-5.15.6、PyQtWebEngine-Qt5-5.15.2

(二) 程序流程

程序的运行流程如图1所示，首先通过用户选择的 url 框判断用户想要发送的是 GET 请求还是 HEAD 请求（后面会说明由两个输入的地方，让用户分别发送 GET 请求和 HEAD 请求，同时会进行标识）。根据用户输入的 url 以及输入的位置构造 http 数据报，然后和 url 对应的服务器建立连接，建立连接后发送数据报并接收服务器的相应数据报。根据数据报的内容显示交互过程、文本段落、支持跳转的链接以及图片。

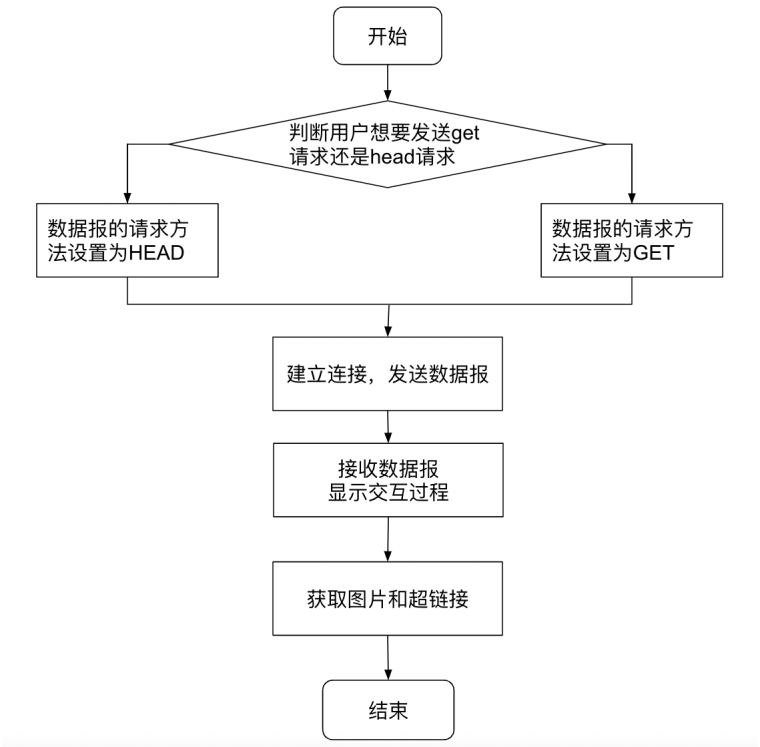

图 1: 流程图

(三) 关键问题

1. 获取用户输入

获取用户输入的代码如图2所示这里面通过 PyQt5 中的 QLineEdit 获取用户的输入，并设置相应的 returnPressed 的槽函数支持回车键跳转，然后将相应的槽函数利用 connect 方式设置为自己写的函数，即以 GET/HEAD 方式获取该 url 页面内容的函数。其中因为需要实现 HEAD 和 GET 两个请求，因此代码中写了两个 LineEdit，并为了让用户显著的确定两个 LineEdit 的作用分别是什么，利用了控件的 setText 函数将对应的作用写在控件上。

```
# # 添加URL地址栏
self.urlbar = QLineEdit()
self.urlbar2 = QLineEdit()
# 让地址栏支持输入地址回车访问
self.urlbar.returnPressed.connect(self.get_url)
self.urlbar.setText("get")
self.urlbar2.returnPressed.connect(self.head_url)
self.urlbar2.setText("head")
```

图 2: 获取用户输入

然后通过 QLineEdit 控件的 text 属性便可获得用户在该控件中输入的内容，也就是想要访问的 url，然后进行后续的发送消息的请求。

2. 发送和接收 http 数据报

想要获取相应 url 的页面内容，需要构造 socket 套接字，通过套接字和相应 url 的服务器建立连接，确保能够建立连接并能够顺畅的发送消息。

但是这里有一个比较 trick 的地方，就是 connect 的时候不能使用包含 https 的域名，否则会无法建立连接，比如如果想要访问百度，则 connect 的时候使用的域名应该是 www.baidu.com，而不是 https://www.baidu.com。这是因为前面的 https 并不是百度服务器的域名，而是超文本传输协议，是用于从万维网服务器传输超文本到本地浏览器的传送协议。所以在 connect 的时候需要将这部分去掉。

建立连接首先通过 getaddrinfo 函数获取对应 url 和端口的 IP 地址，然后利用获取到的地址进行 connect。

因此如图3所示，实现的时候首先判断了一下用户输入的 url 开头是否包含 https，如果包含了，则 connect 的时候使用 https 后面的内容。同时输入的 url 应该是完整的，需要包含 www，比如必须使用 www.baidu.com，不可以只有 baidu.com，因此代码中判断了用户输入是否包含 www，如果没有则会自动补全。

```
    url2 = self.urlbar2.text()
    if url2[0:5] == "https":
        url2 = url2[8:]

    if url2[0:3] == "www":
        HOST = url2
    else:
        HOST = "www." + url2

PORT = 80
for res in getaddrinfo(HOST, PORT, AF_UNSPEC,SOCK_STREAM, 0, AI_PASSIVE):
    af, socktype, proto, canonname, sockaddr = res

    try:
        self.server = socket(af, socktype)
    except OSError as msg:
        self.server = None
        continue
    try:
        self.server.connect(sockaddr)
    except OSError as msg:
        self.server.close()
        self.server = None
        continue
    break
```

图 3: 建立连接

建立连接后按照 http 格式构造发送数据报，然后将收到的消息按照 http 数据报的格式进行解析，显示交互过程。http 格式如下图6所示，首先是请求方法，比如 GET、POST、HEAD 请求，然后便是想要访问的 url，紧接着是 http 协议的版本，有 1.1、2.0 等等，HEAD 和 GET 请求的区别在于二者收到的相应包含的数据不一样，GET 请要求服务器传回相应的资源，HEAD 请求与 GET 请求类似，也是请求从服务器获取资源，服务器的处理机制也是一样的，但服务器不会返回请求的实体数据，只会传回响应头，也就是资源的“元信息”。

图 4: 发送 http 数据报格式

图 5: 接收 http 数据报格式

图 6: http 数据报格式

HEAD 请求可以看做是 GET 请求的一个“简化版”或者“轻量版”。因为它的响应头与 GET 完全相同，所以可以用在很多并不真正需要资源的场合，避免传输 body 数据的浪费。比如，想要检查一个文件是否存在，只要发个 HEAD 请求就可以了，没有必要用 GET 请求把整个文件都取下来。再比如，要检查文件是否有最新版本，同样也应该用 HEAD 请求，服务器会在响应头里把文件的修改时间传回来。

下图7是构造发送数据报的代码，首先构造请求行，请求行根据 head 和 get 请求分别填充不同的请求方法，url 根据用户的输入填充，紧接着是请求头和请求体，请求体按照结束即可。

```
if opt == 1:
    self.request_line = "GET / HTTP/1.1\r\n"
    self.request_header = "Host:" + url +"\r\n"
else:
    self.request_line = "HEAD / HTTP/1.1\r\n"
    self.request_header = "Host:" + url2 +"\r\n"
self.request_blank = "Connection: close\r\n\r\n"
self.request_data = self.request_line + self.request_header + self.request_blank

buf = []

self.server.send(self.request_data.encode())

while True:
    recv = self.server.recv(1024)
    # print(recv)
    if recv:
        buf.append(recv.decode('utf-8','ignore'))
    else:
        break
```

图 7: http 发送数据报代码

接收到相应数据报后，直接将数据报输出即可，因为存在回车符和换行符，所以输出的内容会自动换行。

3. 解析 html，获取文本

如图8所示，将收到的数据报看作一个超长的字符串，遍历字符串，因为汉字的编码和英文字母的 ascii 的编码不一样，因此很容易的区分出是中文还是英文字母，然后将相邻的中文放在同一个结果字符串中，通过是否包含英文字母可以判断中文是否相邻。

```
# 找到html文件中的文字
chinese = []
place = -1
for i in range(len(self.recv_text) - 1):
    i = place + 1
    if i < len(self.recv_text) and u'\u4e00' <= self.recv_text[i] <= u'\u9fff':
        ch = ''
        for j in range(i, len(self.recv_text)):
            if u'\u4e00' <= self.recv_text[j] <= u'\u9fff':
                ch += self.recv_text[j]
                place = j
            else:
                chinese.append(ch)
                place = j
                break
    else:
        place += 1
print(chinese)

for i in range(len(chinese)):
    self.output.append(chinese[i])
```

图 8: 解析 html，获取文本

4. 解析 html，获取链接

如图9所示，将数据报中的链接找到，通过对数据报的检索，发现所有与链接有关的都会包含一个 href 的标签，后面便是该链接的 url。于是遍历收到的数据报，当发现 href 字符串后，取后面的字符串，直至发现” 字符，判定链接结束，然后将其放入 QTableWidget 表格中，并设置表格的点击事件的槽函数。

```
# 找到得到的html文件中的url，放在文本框中，并支持跳转
urls = []
for i in range(len(self.recv_text)-5):
    if self.recv_text[i:i+5] == 'href=': # 判断是链接
        # print(self.recv_text[i:i+5])
        url = 'https:'
        for j in range(i + 6, len(self.recv_text)-5):
            if self.recv_text[j] == '\"':
                urls.append(url)
                # print(url)
                break
            url += self.recv_text[j]

for i in range(len(urls)):
    item = QTableWidgetItem(urls[i])
    self.tableWidget.setItem(i,0,item)
    rowPosition = self.tableWidget.rowCount()
    self.tableWidget.insertRow(rowPosition)
```

图 9: 解析 html，获取链接

为了支持跳转，设置了 QtableWidget 的点击事件的槽函数，为了获取相应点击的链接，利用点击事件获取点击所在的行，然后可以根据表格中该行的内容获取点击的链接。为了获取链接

对应的内容，选择再次发送 get 请求，获取页面的 html 进行展示，同时展示的页面选择新建一个窗口，其中该窗口的空间只有 TextEdit 用于显示 get 请求的结果，自窗口发送 get 请求同样是构造 http 数据报，这和一开始获取 html 的方式是一致的。

```
# 支持链接的点击事件，设置对应的跳转槽函数
def click_href(self,Item=None):
    if Item is None:
        return
    else:
        text = Item.text() # 获取内容
        print(text)
        self.child = ChildWindow()
        # self.child.url = text
        self.child.show()
        row = Item.row()
        deny = [0,1,6,9,10,11,14,15,17,18,19]
        if row in deny:
            self.child.get_url("no")
        else:
            self.child.get_url(text)
        # self.child.exec_()
```

图 10: 设置 QTableWidget 点击事件

```
# 用于显示html中的超链接，用表格的形式体现
self.tableWidget = QTableWidget()
self.tableWidget.setGeometry(QRect(0, 0, 700, 200))
self.tableWidget.setColumnWidth(0, 200)
self.tableWidget.setObjectName("tableWidget")
self.tableWidget.horizontalHeader().setStretchLastSection(True)
self.tableWidget.verticalHeader().setVisible(False)
self.tableWidget.horizontalHeader().setVisible(False)
self.tableWidget.setColumnCount(1)
self.tableWidget.setRowCount(1) # 8行4列
self.tableWidget.itemClicked.connect(self.click_href) # 设置点击函数
```

图 11: 获取点击链接，唤醒子窗口

5. 解析 html，获取图片

获取图片的方法和获取链接的方法类似，都是发现图片会知名 img_src，于是遍历收到的数据报，判断是否包含 img_src，然后可以获得相应图片的 url，但是怎么将图片显示在程序中呢?

如图12所示，我这里采用了将图片先下载到本地然后利用控件 load pixmap 的方式显示图片。首先利用 wget 包的 download 函数将图片加载到本地，然后获取该图片的名字，采用的方式是倒序遍历图片的 url，找到的第一个字符'/' 后面便是图片的名字，然后获取当前的工作路径，再加上图片的名字，便是图片的路径，然后利用 Qpixmap 控件的 load 函数从本地加载图片显示到程序中。

```
# 解析html中的图片src标签
img_src = 'https:'
for i in range(len(self.recv_text)-4):
    if self.recv_text[i:i+4] == 'src=':
        print(self.recv_text[i:i+4])
        img_src = 'https:'
        for j in range(i + 5, len(self.recv_text) - 5):
            if self.recv_text[j] == '\"':
                print(img_src)
                break
            img_src += self.recv_text[j]

# 有的时候因为cache的问题可能没有检索到正确的页面，就会没有图片，再运行一下就好了
if img_src != 'https:':
    from wget import download
    download(img_src)
    for i in range(len(img_src)-1,0,-1):
        if img_src[i] == '/':
            self.local_dir = img_src[i+1:len(img_src)]
            break
    self.local_dir = os.getcwd() + "/" + self.local_dir
    print(self.local_dir)
    self.pixmap.load(self.local_dir)
    new_img = self.pixmap.scaled(700, 300)##调整图片尺寸
    self.label1.setPixmap(new_img)
```

图 12: 解析 html，获取图片

6. 图形界面

本实验采用 python 中的 PyQt5 作为图形界面，界面主体是 MainWindow，其中采用了 QToolbar、QLineEdit、QTabWidget、QHBoxLayout、QLabel、QPixmap、QTextBrowser 等控件用于显示图形界面。

程序刚开始运行的时候，为了让程序更贴近真实的浏览器，一开始的时候通过控件的 load 函数直接加载百度搜索引擎的界面作为初始界面，此时用户不必输入任何东西，然后会发现界面的最上面有两个 url 框架，分别表示 head 和 get，表示通过相应的控件会发送对应的 GET 和 HEAD 请求

最终决定界面分布构造代码如图13所示，分为四个部分。最上面的部分用于显示交互过程，输出发送和接收数据报的头部。下面的左半部分是一个表格用于显示超链接，同时显示文本。下面的右半部分用于显示图片。同时为了让界面更加美观，调整控件之间并不产生空隙。

```
self.v_layout.addWidget(self.textEdit)
self.h_layout.addLayout(self.v_layout0)
self.v_layout0.addWidget(self.tableWidget)
self.v_layout0.addWidget(self.output)

if img_src != 'https':
    self.h_layout.addWidget(self.label1)


self.v_layout.addLayout(self.h_layout)
self.tabs.setLayout(self.v_layout)
```

图 13: 图形界面 1

```
# 设计整体页面的布局
self.h_layout = QHBoxLayout()
self.h_layout.setContentsMargins(0,0,0,0)
self.v_layout = QVBoxLayout()
self.v_layout.setContentsMargins(0,0,0,0)
self.v_layout0 = QVBoxLayout()
self.v_layout0.setContentsMargins(0,0,0,0)
```

图 14: 图形界面 2

(四) 测试截图

测试的最终结果如下图所示，图15显示了刚运行代码，用户还没有输入的时候的初始界面，也就是一个百度的引擎。最上方包含两个 url 输入空控件，左边写着 get，表示发送 get 请求，右边写着 head，表示发送 head 请求。其他地方仅做展示。

图 15: 结果：初始界面

当用户在 head 请求的地方输入 baidu.com 时的结果如下图16所示，可以看到交互过程，发送的数据报的请求头包含请求方法为 head，http 版本为 1.1，相应的 url 是 www.baidu.com。

此时获得到的数据报只包含头部，因为没有数据体，所以没有链接和图片，数据报包含 http 协议版本，状态码为 200 表示成功建立连接。

图 16: 结果：head 结果

当用户在 get 请求的地方输入 baidu.com 时的结果如下图17所示，可以看到最上方是交互过程，显示了发送和接收的数据报，由于返回的数据报包含相应 url 页面内容，因此展示了文本、链接和图片。

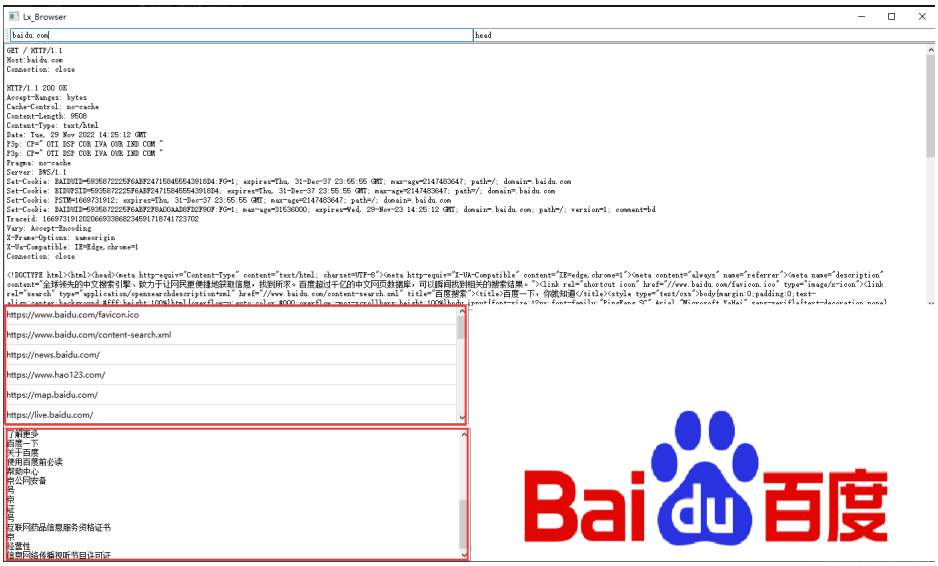

图 17: 结果：get 请求

点击其中一个 url，可以看到程序开了一个新的窗口，重新发送了 get 请求，同样展示了收到的数据报的头部。

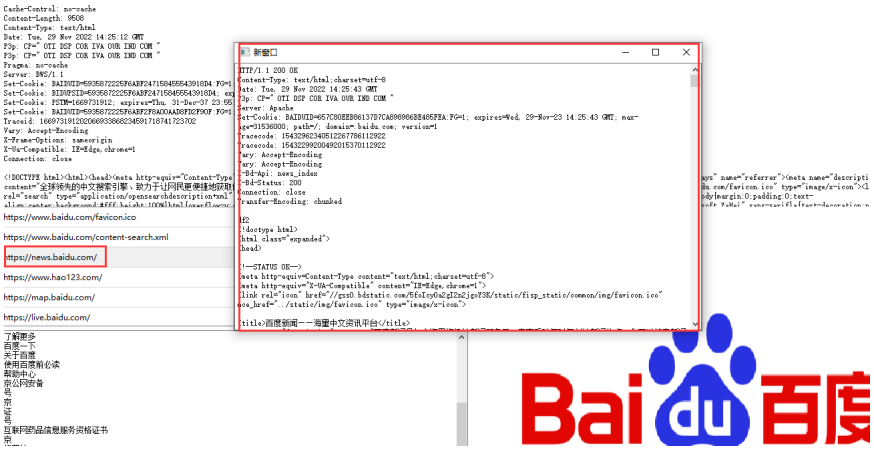

图 18: 结果：点击链接

（五） 收获

1. connect url vs http

经过了此次实验，我觉得我学到了很多，首先就是 connect 的时候像我在介绍原理的时候，connect 函数里面只能写 www.baidu.com，不可以写 https://www.baidu.com，以前因为学过计算机网络，但是没有写过 http 相关的作业，一直以为 https://www.baidu.com 就是域名，所以一开始的时候 connect 一直失败，后来发现在 ping 的时候如果写了所有的也不可以，然后百度发现 http 指明了协议类型，和域名没有关系，并且域名不足也是不可以的，比如在 connect 的时候需要使用 www.baidu.com，不可以只有 baidu.com。改完代码就感觉还是要编程才能发现自己理论知识的不足。

图 19: ping

2. http 数据报

在获取数据报的时候很多时候收获的链接是不完整的，也收不到图片，发现是 connect 的方式有问题，修改代码后可以收到了全部的连接和图片。

3. 图形界面

在进行支持链接跳转的时候尝试了很多种控件，比如 TextBrowser，虽然该控件支持跳转，但是并不支持获取点击位置，即无法很好的找到点击的链接，然后尝试获取鼠标点击事件的绝对地址，然后发现虽然将控件的属性进行了标志，但是点击 TextBrowser 的时候并不会触发事件，但是点击 QLabel 就可以，最终尝试了很多控件，最终发现以表格 QTableWidget 的形式展示链接是可行的。