4.1 今日介绍

4.2 [难点]HTTP协议的介绍

- 全称：超文本传输协议
- 作用：传输数据, 规定了浏览器和Web服务器数据传输的格式

4.3 [重点]URL

- 概念：统一资源定位符, 俗称网址
- 组成：
    - Https://www.baidu.com/xxx/xxx/xxx.html?page=1
    - 协议部分: `https://`
    - 域名部分: `www.baidu.com`
    - 资源路径部分: `/xxx/xxx/xxx.html`
    - 查询参数部分: `?page=1`

4.4 [重点]查看HTTP协议的通信过程

- 谷歌浏览器开发者工具的使用
    - 右键-->检查
    - F12
    - 设置-->更多工具-->开发者工具
- HTTP协议的通信过程
    - 切换到Network标签
    - 刷新网页
    - 点击查看
    - 每一项记录都是一次完整的请求与响应

4.5 [难点]HTTP请求报文

- 一个HTTP请求报文可以由**请求行、请求头、空行和请求体**4个部分组成。
- 请求行是由三部分组成:
    1. **请求方式**
    2. **请求资源路径**
    3. **HTTP协议版本**
- **GET方式的请求报文没有请求体，只有请求行、请求头、空行组成。**
- **POST方式的请求报文可以有请求行、请求头、空行、请求体四部分组成，注意:POST方式可以允许没有请求体，但是这种格式很少见。**
- 每一行结尾都有`\r\n`

4.6 [难点]HTTP响应报文

- HTTP 响应报文：

    - 响应行/状态行、响应头、空行、响应体四部分组成
        - 状态行: 协议版本 状态码 状态描述
    - 每一行结尾都有`\r\n`
- HTTP 状态码:

  HTTP 状态码是**用于表示web服务器响应状态的3位数字代码。**

状态码	说明
200	请求成功
307	重定向
400	错误的请求，请求地址或者参数有误
404	请求资源在服务器不存在
500	服务器内部源代码出现错误

4.7 [了解]搭建Python自带静态Web服务器

- 静态Web服务器:
    - 给发出请求的浏览器提供静态文档的程序
- Python自带的静态Web服务器:
    - 先进入到静态资源文件夹
    - Ubuntu: `python3 -m http.server 8000`
    - Windows: `python -m http.server 8000`

4.8 [重点]静态Web服务器-返回固定页面数据

- 静态Web服务器实现:
- 步骤:

```
# 0. 导入模块
# 1. 创建socket对象
# 2. 绑定端口
# 3. 设置监听
# 4. 等待连接-->循环等待
# 5. 接收数据-->解码
# 6. 发送数据-->响应报文
# 7. 关闭连接

# 0. 导入模块
import socket

if __name__ == '__main__':
    # 1. 创建socket对象
    server_socket = socket.socket()
    # 设置端口复用
    server_socket.setsockopt(socket.SOL_SOCKET, socket.SO_REUSEADDR, True)
```

```
    # 2. 绑定端口
    server_socket.bind(("", 8080))

    # 3. 设置监听
    server_socket.listen(128)

    while True:
        #- 4. 等待连接->循环等待
        new_client_socket, ip_port = server_socket.accept()

        # 5. 接收数据-->解码
        recv_data = new_client_socket.recv(4096)
        print("原始数据:", recv_data)
        recv_str = recv_data.decode("utf-8")
        print("转换数据:", recv_str)

        # 6. 发送数据-->响应报文

        # 响应行
        response_line = "HTTP/1.1 200 OK\r\n"

        # 响应头
        response_head = "Server: NBS/1.1\r\n"

        # 响应体
        with open("static/index.html", "rb") as file:
            response_body = file.read()

        # 注意: 发送前, 数据需要将字符串转换为二进制 str-->byte
        response_data = (response_line + response_head + "\r\n").encode("utf-8") +
response_body
        new_client_socket.send(response_data)

        # 7. 关闭连接
        new_client_socket.close()
```

4.9 [重点]静态Web服务器-返回指定页面数据

- 返回指定页面数据的实现步骤:

```
# 1. 处理客户端下线的情况
# 2. 通过空格切分请求报文, 获得资源路径
# 3. 设置根路径的访问, 增加默认首页
# 4. 进行异常捕获, 根据能否打开文件, 进行不同的响应报文发送

# 0. 导入模块
import socket

if __name__ == '__main__':
    # 1. 创建socket对象
```

```
    server_socket = socket.socket()
    # 设置端口复用
    server_socket.setsockopt(socket.SOL_SOCKET, socket.SO_REUSEADDR, True)

    # 2. 绑定端口
    server_socket.bind(("", 8080))

    # 3. 设置监听
    server_socket.listen(128)

    while True:
        #- 4. 等待连接->循环等待
        new_client_socket, ip_port = server_socket.accept()

        # 5. 接收数据-->解码
        recv_data = new_client_socket.recv(4096)
        recv_str = recv_data.decode("utf-8")

        # 处理客户端下线的情况
        if len(recv_data) == 0:
            print("客户端下线了", ip_port)
            continue

        # 通过空格切分数据，获得资源路径
        # GET / HTTP/1.1\r\nHost: 127.0.0.1:8080
        recv_list = recv_str.split(" ", maxsplit=2)
        path = recv_list[1]

        # 设置根路径的访问，增加默认首页
        if path == "/":
            path = "/index.html"

        # 6. 发送数据-->响应报文
        try:
            # 响应体
            with open("static" + path, "rb") as file:
                response_body = file.read()
        except:
            # 返回错误页面
            with open("static/error.html", "rb") as file:
                response_body = file.read()
            # 响应行
            response_line = "HTTP/1.1 404 Not Found\r\n"
            # 响应头
            response_head = "Server: NBS/1.1\r\n"
            # 注意: 发送前，数据需要将字符串转换为二进制 str-->byte
            response_data = (response_line + response_head + "\r\n").encode("utf-8") +
response_body
            new_client_socket.send(response_data)

        else:
            # 响应行
            response_line = "HTTP/1.1 200 OK\r\n"
```

```
            # 响应头
            response_head = "Server: NBS/1.1\r\n"
            # 注意：发送前，数据需要将字符串转换为二进制 str-->byte
            response_data = (response_line + response_head + "\r\n").encode("utf-8") +
response_body
            new_client_socket.send(response_data)

        finally:
            # 7. 关闭连接
            new_client_socket.close()
```

4.10 [难点]静态Web服务器-多任务版

- 多任务版web服务器程序的实现步骤:

```
    while True:
        # 等待连接->循环等待
        new_client_socket, ip_port = server_socket.accept()
        # 使用子线程处理每一个客户端的请求
        sub_threading = threading.Thread(target=handler_socket, args=
(new_client_socket, ip_port))
        # 设置线程守护
        sub_threading.setDaemon(True)
        # 启动子线程
        sub_threading.start()
```

4.11. [难点]静态Web服务器-面向对象开发

- 面向对象开发的静态Web服务器步骤:

```
"""
类: HttpWebServer()

属性: self.server_socket

方法:
    1. 创建并配置服务端Socket
    2. 启动程序，并开启子线程处理任务
    3. 处理消息收发的函数
"""
import os
import socket
import threading


class HttpWebServer(object):

    def __init__(self):
        """
        初始化方法: 创建了服务端socket，并设置了属性
        """
        server_socket = socket.socket()
```

```
        server_socket.bind(("", 9000))
        server_socket.listen(128)
        self.server_socket = server_socket

    def start(self):
        """
        启动程序，并开启子线程处理任务
        """
        while True:
            new_client_socket, ip_port = self.server_socket.accept()
            sub_threading = threading.Thread(target=self.handler_socket, args=
(new_client_socket, ip_port), daemon=True)
            sub_threading.start()

    @staticmethod
    def handler_socket(new_client_socket, ip_port):
        """
        处理消息收发的函数
        :param new_client_socket: 专门处理客户端消息的新的socket
        :param ip_port: 客户端的ip和端口号
        """
        # 3.1 接收数据
        recv_data = new_client_socket.recv(4096)

        # 3.2 处理客户端socket下线
        if len(recv_data) == 0:
            print("客户端下线了", ip_port)
            return

        # 3.3 获取资源地址
        recv_data_str = recv_data.decode("utf-8")
        path = recv_data_str.split(" ", 2)[1]
        print(path)

        # 3.4 绑定默认首页
        if path == "/":
            path = "/index.html"

        # 3.5 拼接数据，返回响应报文
        # 如果路径存在，返回正常页面
        # 如果路径不存在，就返回错误页面
        if os.path.exists("static" + path):
            with open("static" + path, "rb") as file:
                body = file.read()

            line = "HTTP/1.1 200 OK\r\n"
            head = "Server: HMS/1.0\r\n"
            blank = "\r\n"

            response_data = (line + head +blank).encode("utf-8") + body
            new_client_socket.send(response_data)
        else:
            with open("static/error.html", "rb") as file:
```

```
                body = file.read()

            line = "HTTP/1.1 404 Not Found\r\n"
            head = "Server: HMS/1.0\r\n"
            blank = "\r\n"

            response_data = (line + head +blank).encode("utf-8") + body
            new_client_socket.send(response_data)

        # 3.6 关闭客户端通信
        new_client_socket.close()


if __name__ == '__main__':
    server = HttpWebServer()
    server.start()
```

4.12 [重点]静态Web服务器-命令行启动动态绑定端口号

- 命令行启动动态绑定端口号的静态Web服务器：

```
1.获取执行python程序的终端命令行参数
2.判断参数的类型，设置端口号必须是整型
3.给web服务器类的初始化方法添加一个端口号参数，用于绑定端口号

if __name__ == '__main__':
    in_port = "8080"

    # 1. 先判断argv的长度大于等于2
    if len(sys.argv) >= 2:
        # 2. 再判断第二个参数是否是数字
        if sys.argv[1].isdigit():
            # 3. 获取端口
            in_port = sys.argv[1]

    in_port = int(in_port)

    server = HttpWebServer(in_port)
    server.start()
```

- 如果希望右键运行, 也能绑定动态端口, 需要在代码编辑区右上角, 点击配置, 在参数中进行配置

4.13 知识点总结