Day07-HTTP协议和Web服务器

今日课程学习目标

```
知道 HTTP协议的作用及其URL地址、请求报文、响应报文的格式
理解 Web服务器的作用及其简单实现原理
常握 FastAPI框架的基本使用
```

今日课程内容大纲

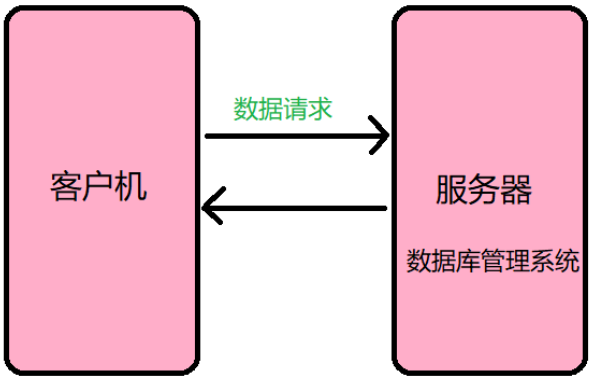
```
# 1. HTTP协议
    C/S架构和B/S架构
    HTTP协议
        功能简介
        URL地址
        请求报文
        响应报文
    谷歌浏览器开发者工具
# 2. Web服务器
    Web服务器简介
    简单Web服务器的实现
# 3. FastAPI框架
    框架简介
    基本使用
```

知识点1：C/S架构和B/S架构【了解】

典型的软件设计架构分成如下两种：

1）**C/S软件架构**：Client/Server

C/S架构软件（即客户机/服务器模式）分为**客户机和服务器两层**。第一层是在客户机系统上结合了表示与业务逻辑，第二层是通过网络结合了数据库服务器。



典型软件：音乐播放器、杀毒软件、QQ、微信等

优点：

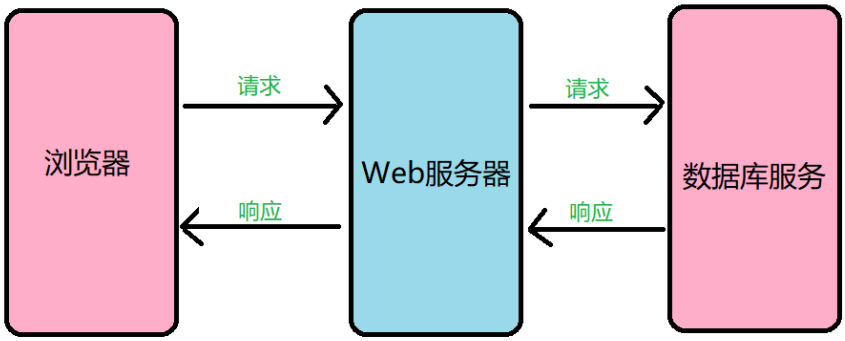
- 客户端和服务器直接相连，响应速度更快，安全性更好
- 客户端可以处理一些逻辑事务，分担服务器的压力
- 客户端操作界面可以随意排列，满足个性化需求

缺点：

- **必须下载安装客户端软件才能进行使用**，部署难度大，不易扩展
- 用户数增多会出现通信拥堵，服务器响应速度慢等情况

2）**B/S软件架构**：Browser/Server

B/S架构软件(即浏览器/服务器模式)，**是C/S架构的一种改进，可以说属于三层C/S架构。第一层是浏览器**，即客户端，只有简单的输入输出功能，处理极少部分的事务逻辑。**第二层是WEB服务器**，扮演着信息传送的角色。**第三层是数据库服务器**，他扮演着重要的角色，因为它存放着大量的数据。



典型软件：各种网站 → 百度网、京东网、淘宝网。。。

优点：

- **不需要下载客户端软件，有浏览器，能联网就能进行访问**，容易扩展
- 界面人性化，通用化，不需要多少培训就能掌握

缺点：

- 页面通用化，不突出个性，页面需要不断地动态刷新，尤其是用户增多，网速慢得情况，很费时
- 服务器承担着重要的责任，数据负荷较重，服务器一旦崩溃，后果不堪设想

知识点2：HTTP协议简介【熟悉】

思考问题：

```
1. 什么是HTTP协议？
2. HTTP协议和TCP协议有什么区别和联系？
3. B/S软件架构中，为什么需要HTTP协议？
```

什么是HTTP协议?

HTTP 协议的全称是(HyperText Transfer Protocol)，翻译过来就是**超文本传输协议**，它规定了浏览器和 Web 服务器通信数据的格式，也就是说浏览器和 Web 服务器通信需要使用HTTP协议。

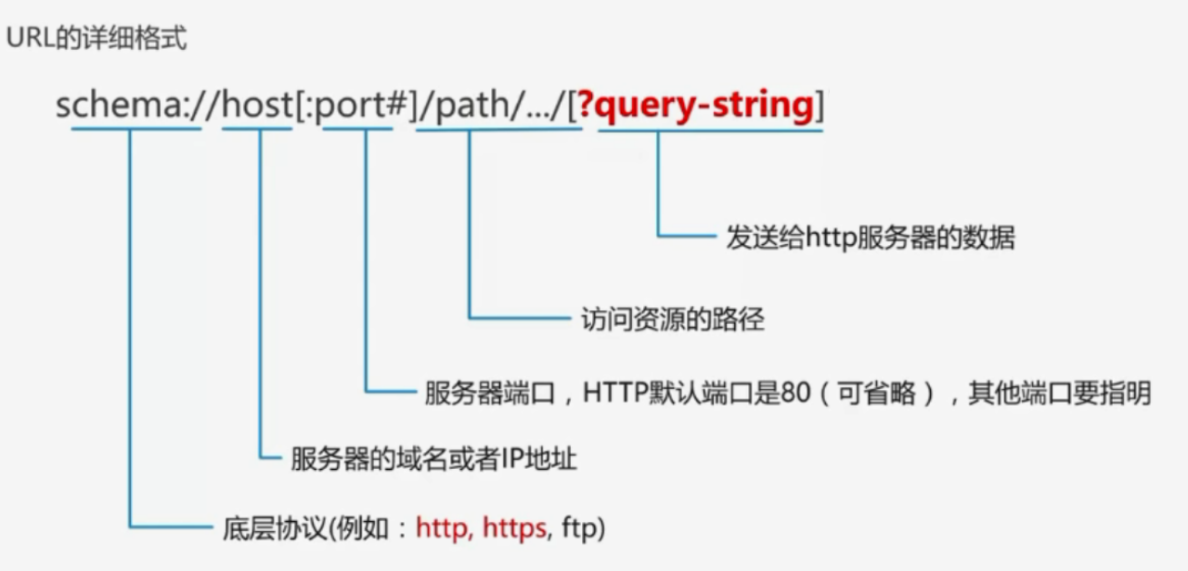
HTTP协议和TCP协议有什么区别和联系?

HTTP协议是应用层协议，它规定了数据传输的格式；TCP协议是传输层协议，负责数据的传输，它不关心数据传输的格式；HTTP依赖于TCP协议进行数据的传输。

知识点3：HTTP协议之URL地址【熟悉】

B/S程序中，浏览器访问服务器时，需要通过 URL 地址。

URL的英文全拼是(Uniform Resoure Locator)，表达的意思是统一资源定位符，通俗理解就是网络资源地址，也就是我们常说的网址。

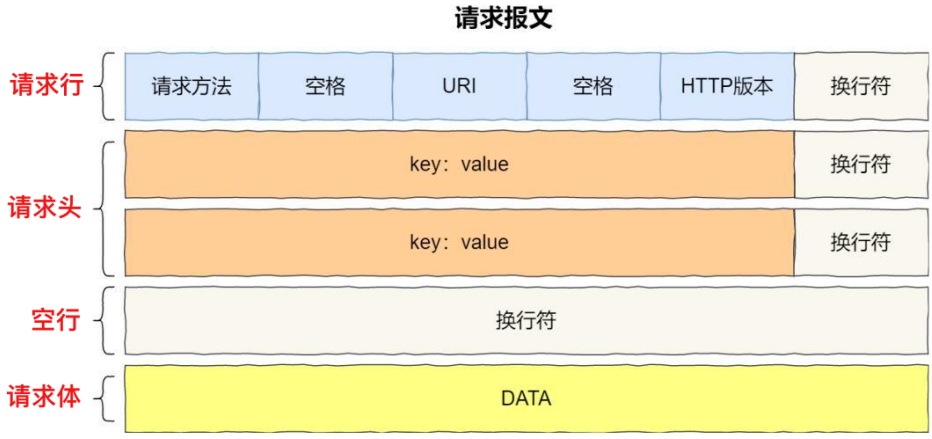


知识点4：HTTP协议之请求报文【熟悉】

浏览器向 Web 服务器发送请求时，发送的是请求报文。

请求报文格式如下：

> 四部分组成：请求行、请求头、空行、请求体




```
----------------------------- 请求行 -----------------------------
GET /annual/2019?source=navigation HTTP/1.1\r\n
----------------------------- 请求头 -----------------------------
Host: movie.douban.com
Connection: keep-alive\r\n
Pragma: no-cache\r\n
Cache-Control: no-cache\r\n
Upgrade-Insecure-Requests: 1\r\n
User-Agent: Mozilla/5.0 (Macintosh; Intel Mac OS X 10_14_6)\r\n
Accept: text/html,application/xhtml+xml,application/xml\r\n
-----------------------------  空行 -----------------------------
\r\n
```

注意：

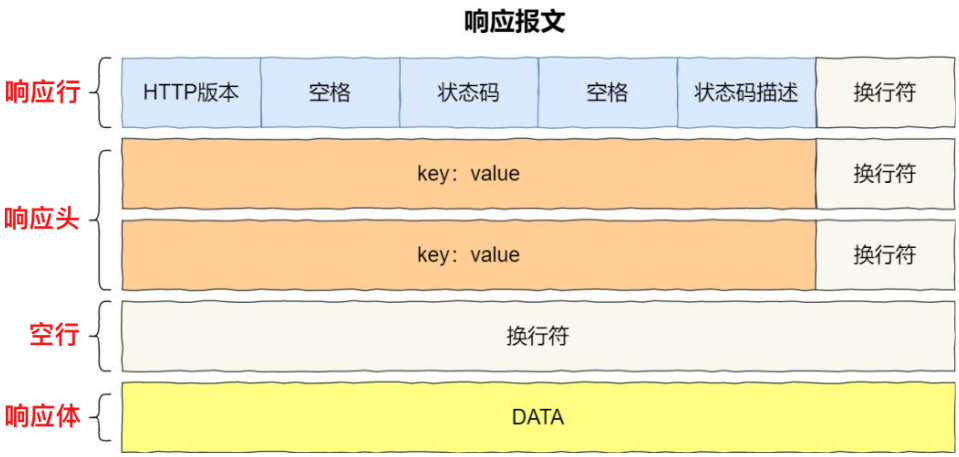
- 请求方法常见的有GET和POST，GET用于获取web服务器数据，POST向web服务器提交数据
- GET请求时，不能携带请求体数据

知识点5：HTTP协议之响应报文【熟悉】

Web 服务器给浏览器返回响应时，返回的是响应报文。

响应报文格式如下：

> 四部分组成：响应行、响应头、空行、响应体




```
----------------------------- 响应行 -----------------------------
HTTP/1.1 200 OK\r\n
```

```
------------------------------ 响应头 ------------------------------
Date: Mon, 14 Sep 2020 07:24:14 GMT\r\n
Content-Type: text/html; charset=utf-8\r\n
Transfer-Encoding: chunked\r\n
Connection: keep-alive\r\n
Keep-Alive: timeout=30\r\n
Vary: Accept-Encoding\r\n
Content-Encoding: gzip\r\n
------------------------------  空行 ------------------------------
\r\n
------------------------------ 响应体 ------------------------------
<html>
    <head>
        ...
    </head>
    <body>
        ...
    </body>
</html>
```

响应状态码：

HTTP 状态码是**用于表示web服务器响应状态的3位数字代码**，常见状态码如下：

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

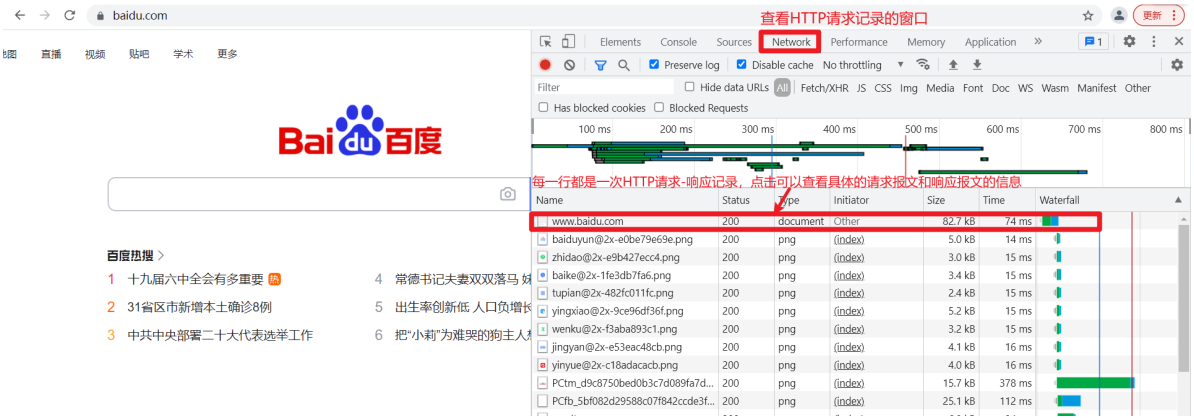
> 注意：很多时候，也会使用 302 表示重定向。

知识点6：谷歌浏览器开发者工具【熟悉】

在使用浏览器访问 Web 网站时，可以通过浏览器开发者工具查询每次的 HTTP 请求记录。

打开谷歌浏览器开发者工具：

- 浏览器右键，点击检查
- 快捷键：F12 或 Ctrl+Shift+i



知识点7：Web服务器和非Web服务器【熟悉】

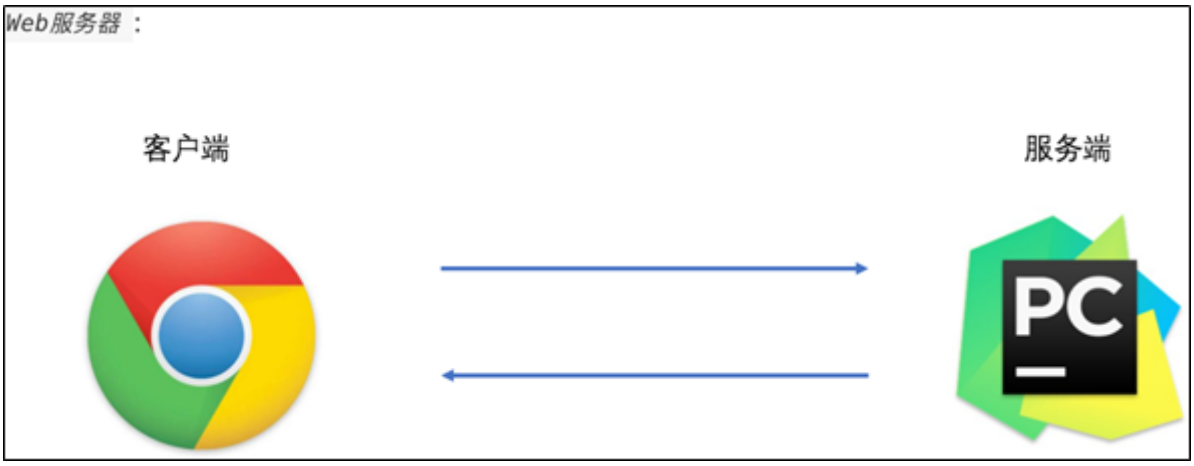
思考问题：

```
1. 什么是Web服务器？Web服务器的作用是什么？
2. Web服务器和之前写的TCP服务器有什么区别？
```

什么是Web服务器？Web服务器的作用是什么？

Web服务器是可以和浏览器进行HTTP通信的服务器。不能浏览器进行HTTP通信的服务器统称为非Web服务器。

Web服务器的作用是接收HTTP请求(解析请求报文)，返回HTTP响应(返回响应报文)。



Web服务器和之前写的TCP服务器有什么区别？

之前写的TCP服务器是不能和浏览器进行HTTP通信的，因为之前写的TCP服务器不能解析浏览器请求时发送的HTTP请求报文，并且不能给浏览器返回HTTP响应报文。

Web服务器本质也是一个TCP服务器，但是它能解析HTTP请求报文，并且能组织返回响应报文。

> 即 Web 服务器能够知道浏览器请求的资源是什么，并能将对应的资源组织成响应报文的格式进行返回

知识点8：简单 Web 服务器-返回HelloWorld内容【掌握】

需求：将之前的 TCP 服务端程序代码改造成简单的 Web 服务器程序，实现能够和浏览器进行HTTP通信，并返回Hello World内容给浏览器

```python
"""
简单Web服务器-返回HelloWorld
学习目标：能够实现浏览器和Web服务器简单通信
"""

import socket

# 创建一个服务端监听套接字socket对象
# socket.AF_INET：表示使用 IPV4 地址
# socket.SOCK_STREAM：表示使用 TCP 协议
server = socket.socket(socket.AF_INET, socket.SOCK_STREAM) # 门迎
# 设置端口重用，服务器程序关闭之后，端口马上能够重复使用
server.setsockopt(socket.SOL_SOCKET, socket.SO_REUSEADDR, True)

# 绑定服务端程序监听的 IP 和 PORT
```

```python
server.bind(('127.0.0.1', 8080))

# 设置服务器监听，127表示同一时间最多处理127个客户端连接请求
server.listen(127)
print('服务器开始进行监听...')

while True:
    # 等待接受客户端的连接：如果没有客户端连接时，这句代码会阻塞等待，直到有客户端连接
    # server_client：和客户端进行通信的 socket 对象，服务员
    # ip_port：客户端的 IP 和 Port
    server_client, ip_port = server.accept()
    print(f'接受到来自客户端{ip_port}的连接请求...')

    # 接受客户端发送的消息，1024表示最多接受1024个字节
    # 如果客户端没有发消息，recv方法会阻塞等待
    recv_msg = server_client.recv(1024) # 返回值是 bytes 类型
    print('客户端发送的消息为：\n', recv_msg.decode())

    # 组织 HTTP 响应报文并返回
    # 响应行
    response_line = 'HTTP/1.1 200 OK\r\n'
    # 响应头
    response_header = 'Server: Python\r\n'
    # 响应体
    response_body = 'Hello World'
    # 响应内容
    send_msg = response_line + response_header + '\r\n' + response_body

    server_client.send(send_msg.encode())

    # 关闭和客户端通信的套接字、监听套接字
    server_client.close()

server.close()
```

知识点9：简单 Web 服务器-返回固定html内容【掌握】

需求：改造上面的简单 Web 服务器代码，当浏览器请求时，给浏览器返回 gdp.html 网页内容

```python
"""
简单Web服务器-返回固定html内容
学习目标：能够实现Web服务器给浏览器返回固定html内容
"""

import socket

# 创建一个服务端监听套接字socket对象
# socket.AF_INET：表示使用 IPV4 地址
# socket.SOCK_STREAM：表示使用 TCP 协议
server = socket.socket(socket.AF_INET, socket.SOCK_STREAM) # 门迎
# 设置端口重用，服务器程序关闭之后，端口马上能够重复使用
server.setsockopt(socket.SOL_SOCKET, socket.SO_REUSEADDR, True)

# 绑定服务端程序监听的 IP 和 PORT
server.bind(('127.0.0.1', 8080))

# 设置服务器监听，127表示同一时间最多处理127个客户端连接请求
server.listen(127)
```

```python
print('服务器开始进行监听...')

while True:
    # 等待接受客户端的连接：如果没有客户端连接时，这句代码会阻塞等待，直到有客户端连接
    # server_client：和客户端进行通信的 socket 对象，服务员
    # ip_port：客户端的 IP 和 Port
    server_client, ip_port = server.accept()
    print(f'接受到来自客户端{ip_port}的连接请求...')

    # 接受客户端发送的消息，1024表示最多接受1024个字节
    # 如果客户端没有发消息，recv方法会阻塞等待
    recv_msg = server_client.recv(1024) # 返回值是 bytes 类型
    print('客户端发送的消息为：\n', recv_msg.decode())

    # 组织 HTTP 响应报文并返回
    # 响应行
    response_line = 'HTTP/1.1 200 OK\r\n'
    # 响应头
    response_header = 'Server: Python\r\n'
    # 响应体：读取 gdp.html 内容
    file = open('./html/gdp.html', 'r', encoding='utf8')
    response_body = file.read()
    file.close()

    # 响应内容
    send_msg = response_line + response_header + '\r\n' + response_body

    server_client.send(send_msg.encode())

    # 关闭和客户端通信的套接字、监听套接字
    server_client.close()

server.close()
```

知识点10：简单 Web 服务器-返回任意html内容【掌握】

需求：

```python
"""
简单Web服务器-返回任意html内容
学习目标：能够实现Web服务器给浏览器返回任意html内容
"""

import socket

# 创建一个服务端监听套接字socket对象
# socket.AF_INET：表示使用 IPV4 地址
# socket.SOCK_STREAM：表示使用 TCP 协议
server = socket.socket(socket.AF_INET, socket.SOCK_STREAM) # 门迎
# 设置端口重用，服务器程序关闭之后，端口马上能够重复使用
server.setsockopt(socket.SOL_SOCKET, socket.SO_REUSEADDR, True)

# 绑定服务端程序监听的 IP 和 PORT
server.bind(('127.0.0.1', 8080))

# 设置服务器监听，127表示同一时间最多处理127个客户端连接请求
server.listen(127)
print('服务器开始进行监听...')
```

```python

while True:
    # 等待接受客户端的连接：如果没有客户端连接时，这句代码会阻塞等待，直到有客户端连接
    # server_client：和客户端进行通信的 socket 对象，服务员
    # ip_port：客户端的 IP 和 Port
    server_client, ip_port = server.accept()
    print(f'接受到来自客户端{ip_port}的连接请求...')

    # 接受客户端发送的消息，1024表示最多接受1024个字节
    # 如果客户端没有发消息，recv方法会阻塞等待
    recv_msg = server_client.recv(1024) # 返回值是 bytes 类型
    print('客户端发送的消息为：\n', recv_msg.decode())

    # 解析请求报文内容，获取客户端请求的 URL 地址
    request_msg = recv_msg.decode()
    split_result = request_msg.split('\r\n')
    # 获取请求行
    request_line = split_result[0]
    request_line = request_line.split(' ')
    # 获取请求的URL提取
    requset_url = request_line[1]

    # 组织 HTTP 响应报文并返回
    # 响应行
    response_line = 'HTTP/1.1 200 OK\r\n'
    # 响应头
    response_header = 'Server: Python\r\n'
    # 响应体：读取指定的 html 页面内容
    # 判断文件是否存在
    import os

    if requset_url == '/':
        # 当客户端请求的是 / 地址时，默认返回 gdp.html 内容
        file_path = './html/gdp.html'
    else:
        file_path = './html' + requset_url

    if not os.path.isfile(file_path):
        # 如果html不存在，响应内容返回 Not Found!
        response_body = 'Not Found!'
    else:
        # 如果html存在，读取html文件内容
        file = open(file_path, 'r', encoding='utf8')
        response_body = file.read()
        file.close()

    # 响应内容
    send_msg = response_line + response_header + '\r\n' + response_body

    server_client.send(send_msg.encode())

    # 关闭和客户端通信的套接字、监听套接字
    server_client.close()

server.close()
```

知识点11：文件操作-with 关键字使用【常握】

之前操作文件的步骤：

```
# 1. 以写的方式打开文件
f = open('1.txt', 'w', encoding='utf8')
# 2. 写入文件内容
f.write('hello world')
# 3. 关闭文件
f.close()
```

> 因为文件对象会占用操作系统的资源，并且操作系统同一时间能打开的文件数量也是有限的，所以文件使用完后必须关闭释放占用的资源

针对于文件操作之后，必须关闭文件，python中提供了一个 with 关键字，能在文件操作之后，自动进行文件的关闭：

```
with open('1.txt', 'w', encoding='utf8') as f:
    f.write('hello world')
```

知识点12：FastAPI 框架简介【了解】

思考问题：

```
1. 我们前面实现的 Web 服务器代码完善吗？
2. Web服务器的代码中，哪些是固定套路的？哪些是变化的？
```

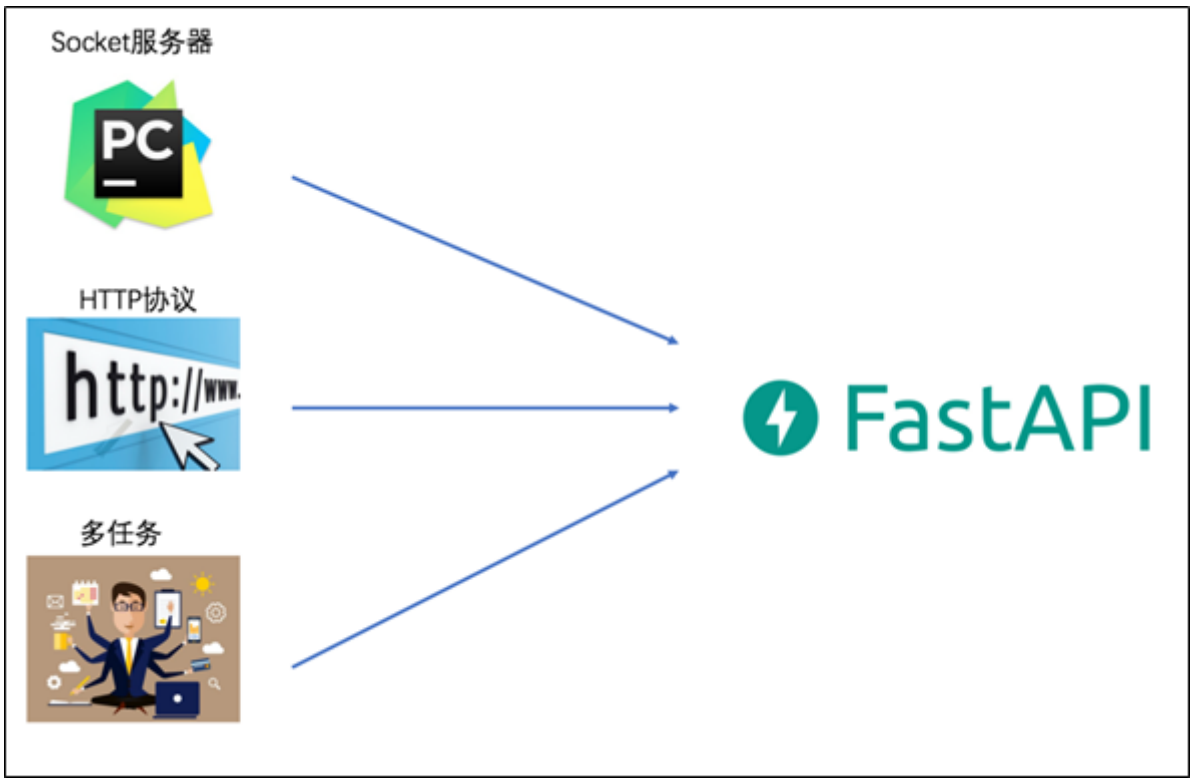
> 我们自己写 Web 服务器，代码是不够完善的.....
>
> Web服务器的代码中，socket设置监听、接收客户端请求、解析HTTP请求报文、组织HTTP响应报文这些其实都是固定的套路，**动态变化的是客户端每一次请求该如何进行业务处理**

FastAPI是什么？什么是 Web 开发框架？

而 FastAPI 就是一个现代的，快速（高性能）python web框架. 基于标准的python类型提示，使用python3.6+构建API的Web框架。

Web开发框架就是将 Web 服务器程序开发过程中，一些固定的套路进行封装，开发者在使用 Web 开发框架开发 Web 应用程序时，不必关注于一些重复性的工作，只关注于具体的业务逻辑即可。

> Python 中 Web 开发框架，常用的还有 Django、Flask.



知识点13：FastAPI 基本使用-HelloWorld程序【熟悉】

安装：

```
# 安装 fastapi 框架
pip install fastapi
# 安装 uvicorn 服务器：运行 fastapi 程序
pip install uvicorn
```

小提示：

```
1. 如何学习一个框架？？？
    学框架的就是学套路，每个框架都有自己的规定，使用框架开发程序，必须遵守框架规定
```

Hello World程序：

```
# 导入 FastAPI 类
from fastapi import FastAPI
# 导入 uvicorn
import uvicorn

# 创建 FastAPI 对象
app = FastAPI()


# 定义业务处理函数并设置对应的 URL 地址
# get：表示请求方式
# /index：表示请求的 URL 地址
@app.get('/index')
def index():
    # 'Hello World'是响应体的内容
    return 'Hello World'


if __name__ == '__main__':
    # 启动 Web 服务器
```

```
    # host：指定 Web 服务器监听的IP
    # port：指定 Web 服务器监听的端口
    uvicorn.run(app, host='127.0.0.1', port=8080)
```

知识点14：FastAPI 基本使用-返回html内容【熟悉】

需求：

1）浏览器访问 / 或 /gdp.html 地址时，返回 gdp.html 内容

2）浏览器访问 /render.html 地址时，返回 render.html 内容

实现代码：

```
# 导入 FastAPI 类
from fastapi import FastAPI
# 导入 uvicorn
import uvicorn
# 导入 Response 响应类
from fastapi import Response

# 创建 FastAPI 对象
app = FastAPI()


# 定义业务处理函数并设置对应的 URL 地址
# get：表示请求方式
# /index：表示请求的 URL 地址
@app.get('/index')
def index():
    # 'Hello World'是响应体的内容
    return 'Hello World'


@app.get('/')
@app.get('/gdp.html')
def gdp():
    with open('./html/gdp.html', 'r', encoding='utf8') as f:
        content = f.read()

    # 返回响应对象，并指定响应内容为 html 类型
    return Response(content, media_type='html')


@app.get('/render.html')
def render():
    with open('./html/render.html', 'r', encoding='utf8') as f:
        content = f.read()

    # 返回响应对象，并指定响应内容为 html 类型
    return Response(content, media_type='html')


if __name__ == '__main__':
    # 启动 Web 服务器
    uvicorn.run(app, host='127.0.0.1', port=8080)
```

> 注意：修改代码之后，必须停止 FastAPI 程序，重启才会生效。

知识点15：FastAPI 基本使用-设置服务器自动重启【了解】

```python
if __name__ == '__main__':
    # 启动 Web 服务器
    # reload=True：检测到代码修改之后，服务器会自动进行重启
    # 注意：设置reload=True时，第一个参数的格式："文件名:app"
    uvicorn.run('05-FastAPI 基本使用-返回html内容:app',
                host='127.0.0.1', port=8080, reload=True)
```

知识点16：扩展内容-本机回环地址、局域网地址、外网地址【了解】

本机回环地址：127.0.0.1或localhost域名，如果运行网络程序时，使用的是本地回环地址，那么该网络程序只有在本机才能进行访问；

局域网地址：局域网中的每天机器，都会有一个局域网的地址，比如：一个班级、一个公司，如果运行网络程序时，使用的是局域网地址，那么该网络程序可以被同一个局域网的其他机器进行访问；

外网地址：也叫公网地址，如果运行网络程序时，使用的是外网地址，那么只要有网，就能访问该网络程序