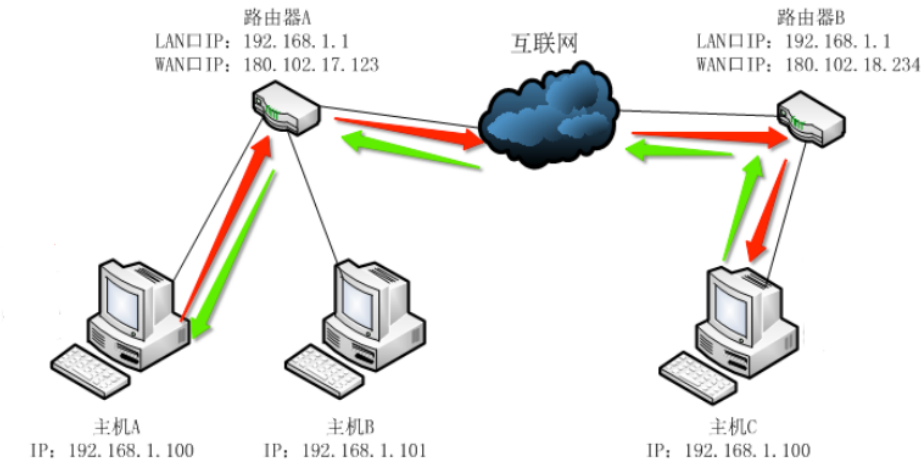
2. 网络编程

2.1 网络基础认知

2.1.1 什么是网络

- 什么是网络：计算机网络功能主要包括实现资源共享，实现数据信息的快速传递。



2.1.2 网络通信标准

- 面临问题
  - 不同的国家和公司都建立自己的通信标准不利于网络互连
  - 多种标准并行情况下不利于技术的发展融合

- OSI 7层模型

  - 好处
    - 建立了统一的通信标准
    - 降低开发难度，每层功能明确，各司其职
    - 七层模型实际规定了每一层的任务，该完成什么事情

- TCP/IP模型
  - 七层模型过于理想，结构细节太复杂
  - 在工程中应用实践难度大
  - 实际工作中以TCP/IP模型为标准

OSI	TCP/IP协议集	
应用层	应用层	Telnet，FTP，SMTP，DNS，HTTP 以及其他应用协议
表示层		
会话层		
传输层	传输层	TCP， UDP
网络层	网络层	IP，ARP，RARP，ICMP
数据链路层	网络接口	各种通信网络接口（以太网等） （物理网络）
物理层		

- 网络协议
  - 什么是网络协议：在网络数据传输中，都遵循的执行规则
  - 网络协议实际上规定了每一层在完成自己的任务时应该遵循什么规范
- 需要应用工程师做的工作： 编写应用功能，明确对方地址，选择传输服务

2.1.3 通信地址

- IP地址
  - IP地址： 即在网络中标识一台计算机的地址编号

- IP地址分类
  - IPv4： 192.168.1.5
  - IPv6： fe80::680a:76cf:ab11:2d73
- IPv4 特点
  - 分为4个部分，每部分是一个整数，取值分为0-255
- IPv6 特点（了解）
  - 分为8个部分，每部分4个16进制数，如果出现连续的数字 0 则可以用 :: 省略中间的0
- IP地址相关命令
  - ifconfig : 查看Linux系统下计算机的IP地址【windows用ipconfig】

```
   robin    :~$ ifconfig
ens33: flags=4163<UP,BROADCAST,RUNNING,MULTICAST>  mtu 1500
        inet 192.168.94.136  本机IP地址        55.0  broadcast 192.168.94.255
        inet6 fe80::80a:76cf:ab11:2d73  IPv6地址          peid 0x20<link>
        ether 00:0c:29:a0:7d:55  txqueuelen 1000  (以太网)
        RX packets 82512  bytes 10750221 (10.7 MB)
        RX errors 0  dropped 0  overruns 0  frame 0
        TX packets 14248  bytes 1308719 (1.3 MB)
        TX errors 0  dropped 0 overruns 0  carrier 0  collisions 0

lo: flags=73<UP,LOOPBACK,RUNNING>  mtu 65536
        inet 127.0.0.1  用于本机本地测试地址
        inet6 ::1  prefixlen 128  scopeid 0x10<host>
        loop  txqueuelen 1000  (本地环回)
        RX packets 801580  bytes 102760244 (102.7 MB)
        RX errors 0  dropped 0  overruns 0  frame 0
        TX packets 801580  bytes 102760244 (102.7 MB)
        TX errors 0  dropped 0 overruns 0  carrier 0  collisions 0
```

  - ping [ip]：查看计算机的连通性

```
   robin    :~$ ping www.baidu.com
PING www.a.shifen.com (182.61.200.7) 56(84) bytes of data.
64 bytes from 182.61.200.7 (182.61.200.7): icmp_seq=1 ttl=128 time=24.0 ms
64 bytes from 182.61.200.7 (182.61.200.7): icmp_seq=2 ttl=128 time=5.39 ms
64 bytes from 182.61.200.7 (182.61.200.7): icmp_seq=3 ttl=128 time=7.02 ms
^C
--- www.a.shifen.com ping statistics ---
3 packets transmitted, 3 received, 0% packet loss, time 2003ms
rtt min/avg/max/mdev = 5.398/12.174/24.099/8.458 ms
```

- 公网IP和内网IP
  - 公网IP指的是连接到互联网上的公共IP地址，大家都可以访问（将来进公司，公司会申请公网IP作为网络项目的被访问地址）
  - 内网IP指的是一个局域网络范围内由网络设备分配的保留IP地址

- 端口号
  - 端口：网络地址的一部分，在一台计算机上，每个网络程序对应一个端口

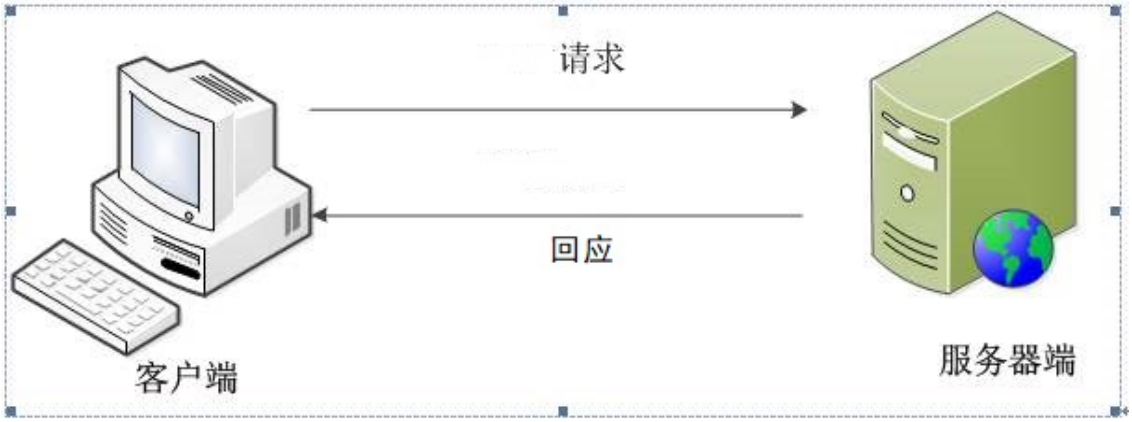
- 端口号特点
  - 取值范围： 0 —— 65535 的整数
  - 一台计算机上的网络应用所使用的端口不会重复
  - 通常 0——1023 的端口会被一些有名的程序或者系统服务占用，个人一般使用 > 1024的端口
  - netstat -nultp 显示当前系统中tcp和udp端口使用情况
  - lsof -i :[port] 显示一个端口是否被占用，需要管理员权限执行

```
sudo  lsof  -i  :3306
sudo kill -9 12888
```

```
> windows:
> 查看被占用端口对应的 PID： netstat -aon|findstr 5555
> 查看指定 PID 的进程：tasklist|findstr 16168
> 结束进程：taskkill /t /f /pid 16168
```

2.1.4 服务端与客户端

- 服务端（Server）：服务端是为客户端服务的，服务的内容诸如向客户端提供资源，保存客户端数据，处理客户端请求等。
- 客户端（Client）：也称为用户端，是指与服务端相对应，为客户提供一定应用功能的程序，我们平时使用的手机或者电脑上的程序基本都是客户端程序。



2.2 UDP 传输方法

2.2.1 套接字简介

- 套接字(Socket)： 实现网络编程进行数据传输的一种技术手段，各种各样的网络服务大部分都是基于 Socket 来完成通信的。

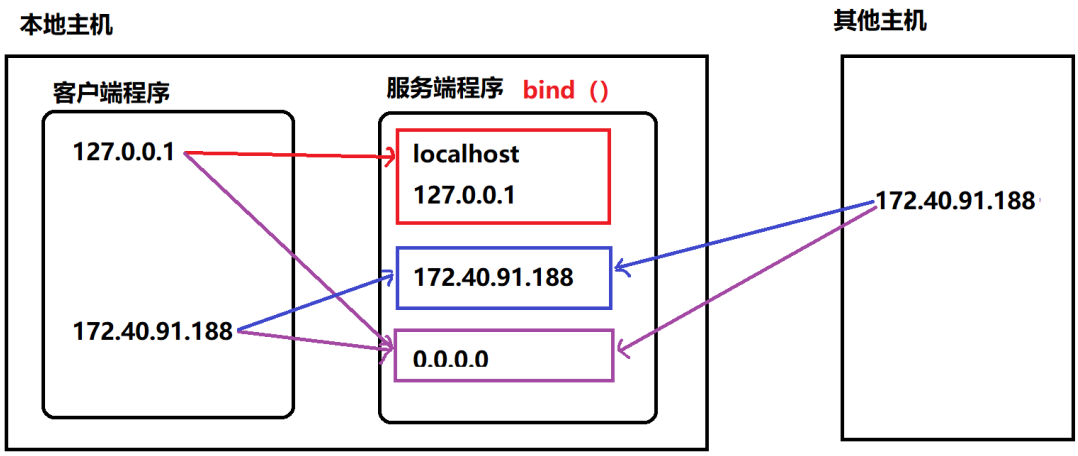
- Python套接字编程模块：import socket

2.2.2 UDP套接字编程

- 创建套接字

```
    sock=socket.socket(family,type)
    功能：创建套接字
    参数：family  网络地址类型 AF_INET表示ipv4
        type     套接字类型 SOCK_DGRAM 表示udp套接字 （也叫数据报套接字）
    返回值：      套接字对象
```

- 绑定地址
  - 本地地址： 'localhost' , '127.0.0.1'
  - 网络地址： '172.40.91.185' (通过ifconfig查看)
  - 自动获取地址： '0.0.0.0'




```
sock.bind(addr)
功能：  绑定本机网络地址
参数：  二元元组 (ip,port)  ('0.0.0.0',8888)
```

- 消息收发

```
data,addr = sock.recvfrom(buffersize)
功能：  接收UDP消息
参数：  每次最多接收多少字节
返回值：  data  接收到的内容
        addr  消息发送方地址

n = sock.sendto(data,addr)
功能：  发送UDP消息
参数：  data  发送的内容 bytes格式
      addr  目标地址
返回值：发送的字节数
```

- 关闭套接字

```
sock.close()
功能：关闭套接字
```

```
UDP服务端代码示例：
'''
udp_server.py
基于UDP协议的聊天程序服务器端
'''
from socket import *

# 1. 创建UDP socket对象
```

```python
udp_socket = socket(AF_INET, SOCK_DGRAM)
# 2. 绑定地址
udp_socket.bind(('0.0.0.0', 5555))
# 3. 循环收发消息
while True:
    # 收消息
    data, addr = udp_socket.recvfrom(256)
    # 服务器端接收到##退出
    if data == b'##':
        break
    print(f'从{addr}获取到消息：{data.decode()}')
    msg = input('>>')
    # 发消息
    udp_socket.sendto(msg.encode(), addr)

# 4. 释放资源
udp_socket.close()
```

UDP 客户端代码示例：

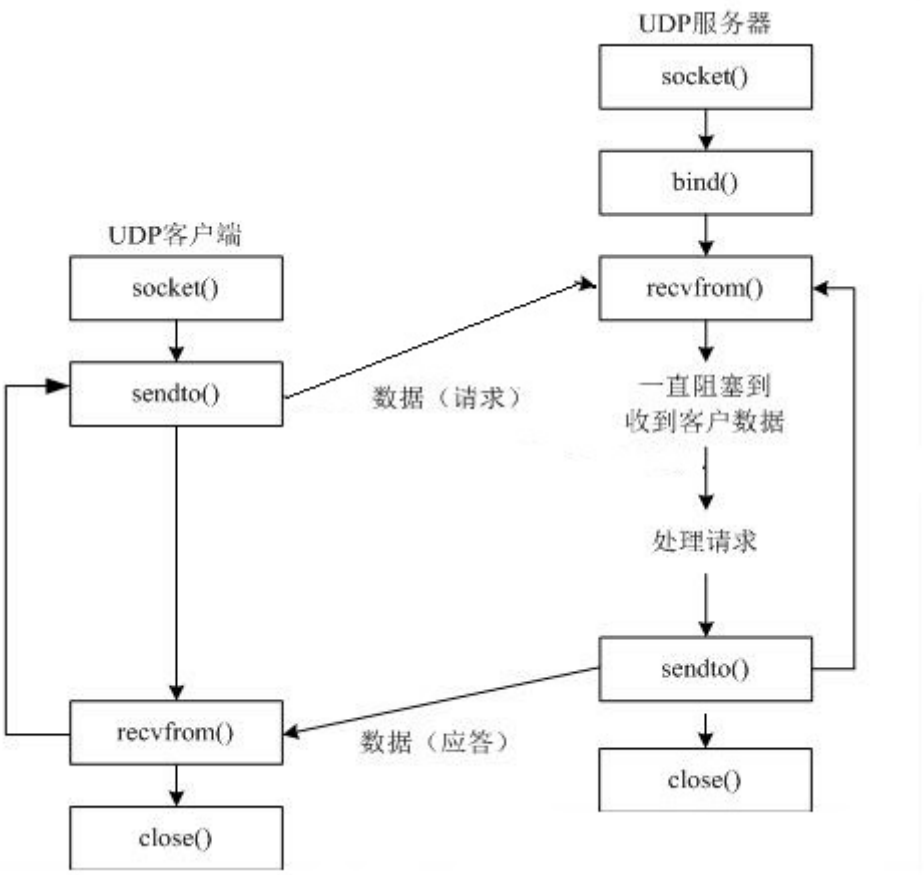
```python
'''
udp_client.py
基于UDP协议的聊天程序客户端
'''
from socket import *

ADDR = ('127.0.0.1', 5555)

# 1. 创建UDP socket对象
udp_socket = socket(AF_INET, SOCK_DGRAM)
# 2. 循环收发消息
while True:
    # 发消息
    msg = input('>>')
    udp_socket.sendto(msg.encode(), ADDR)
    if msg == '##':
        break
    # 收消息
    data, addr = udp_socket.recvfrom(256)
    print(f'从服务器端收到消息：{data.decode()}')
```

```
# 3. 释放资源
udp_socket.close()
```

- 服务端客户端流程



随堂练习：

使用udp完成网络单词查询
从客户端输入单词，发送给服务端，从服务端得到单词的解释，发送给客户端打印出来

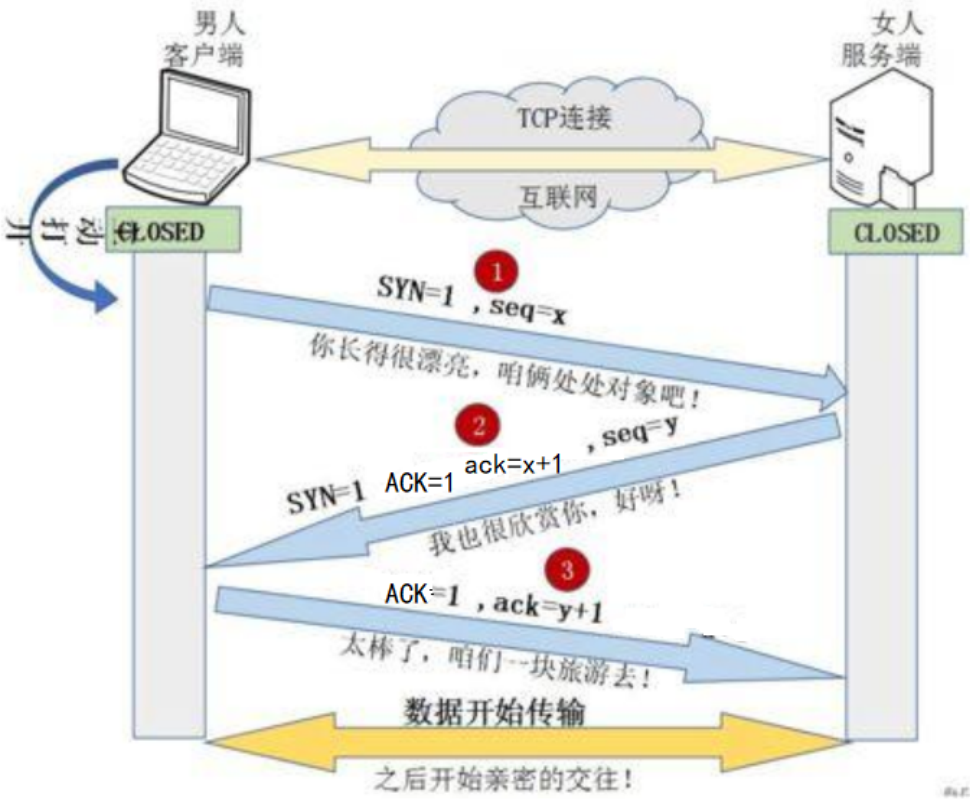
2.2.3 UDP套接字特点

- 可能会出现数据丢失的情况
- 传输过程简单，实现容易
- 数据以数据包形式表达传输
- 数据传输效率较高

2.3 TCP 传输方法

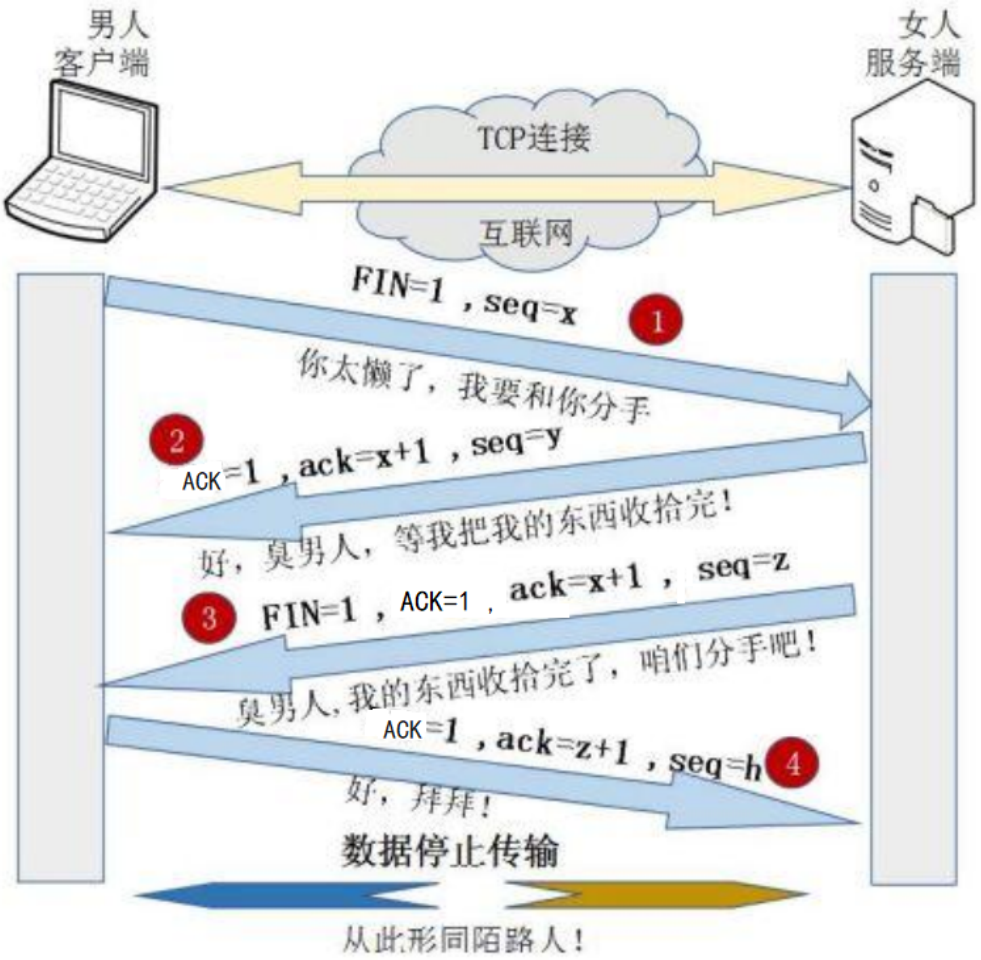
2.3.1 TCP传输特点

- 面向连接的传输服务
  - 传输特征： 提供了可靠的数据传输，可靠性指数据传输过程中无丢失、无乱序、无差错、无重复
  - 可靠性保障机制（都是操作系统网络服务自动帮应用完成的）：
    - 在通信前需要建立数据连接
    - 确认应答机制
    - 通信结束要正常断开连接
- 三次握手（建立连接）
  - 客户端向服务器发送消息报文请求连接
  - 服务器收到请求后，回复报文确定可以连接
  - 客户端收到回复，发送最终报文连接建立

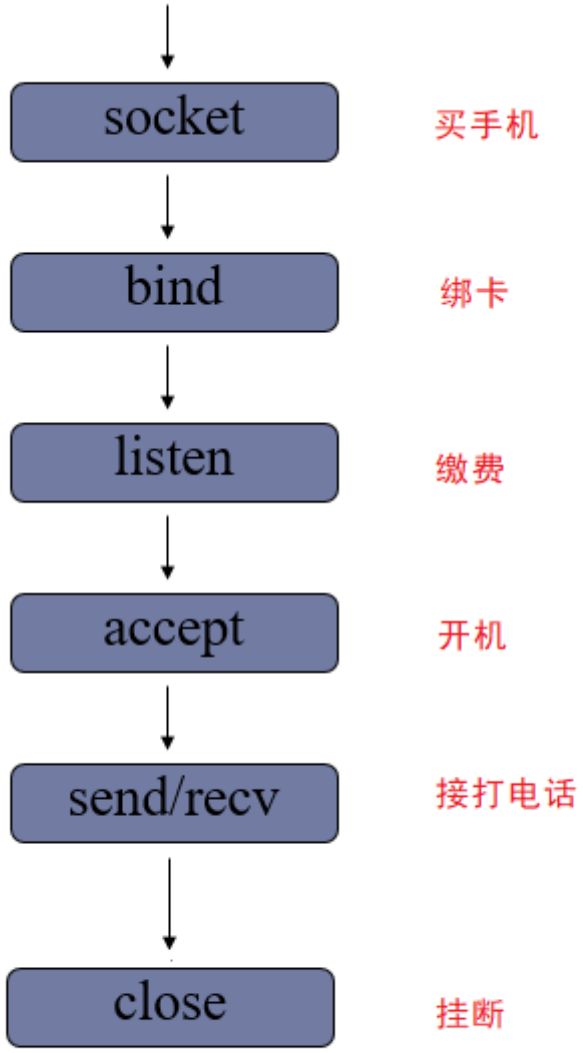


- 四次挥手（断开连接）

- 主动方发送报文请求断开连接
- 被动方收到请求后，立即回复，表示准备断开
- 被动方准备就绪，再次发送报文表示可以断开
- 主动方收到确定，发送最终报文完成断开



2.3.2 TCP服务端



- 创建套接字

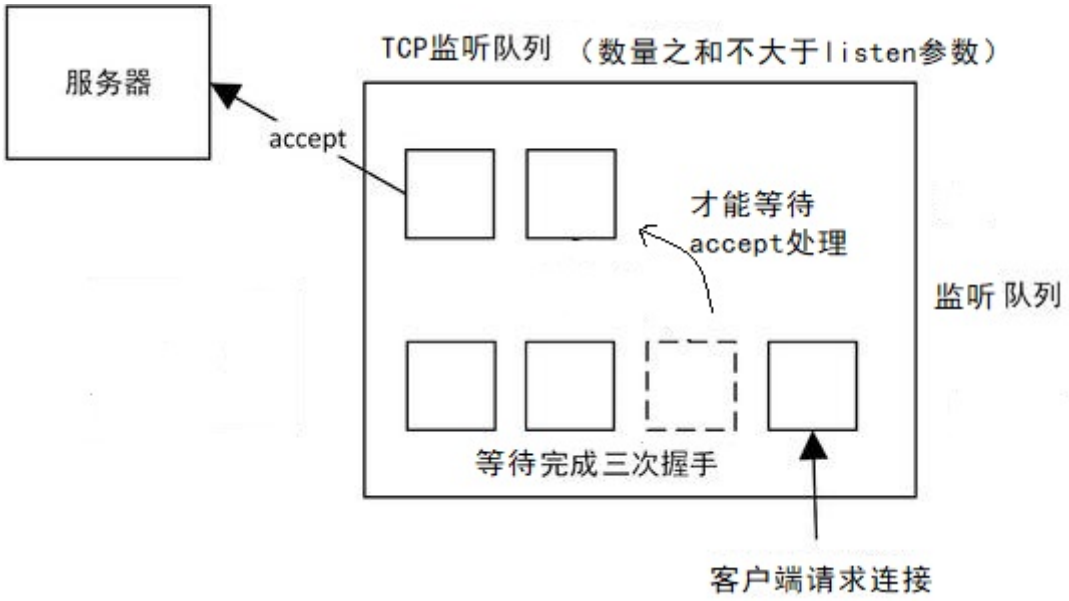
```
    sock=socket.socket(family,type)
    功能：创建套接字
    参数：family  网络地址类型 AF_INET表示ipv4
        type   套接字类型 SOCK_STREAM 表示tcp套接字 （也叫流式套接字）
    返回值：  套接字对象
```

- 绑定地址 （与udp套接字相同）

```
     sock.bind(addr)
    功能：  绑定本机网络地址
    参数：  二元元组 (ip,port)  ('0.0.0.0',8888)
```

- 设置监听

```
sock.listen(n)
功能 ： 将套接字设置为监听套接字，确定监听队列大小
参数 ： 监听队列大小
```



- 处理客户端连接请求

```
conn,addr = sock.accept()
功能： 阻塞等待处理客户端请求
返回值： conn 客户端连接套接字
       addr 连接的客户端地址
```

- 消息收发

```
data = conn.recv(buffersize)
功能 ： 接受客户端消息
参数 ：每次最多接收消息的大小
返回值： 接收到的内容

n = conn.send(data)
功能 ： 发送消息
参数 ：要发送的内容 bytes格式
返回值： 发送的字节数
```

6. 关闭套接字 (与udp套接字相同)

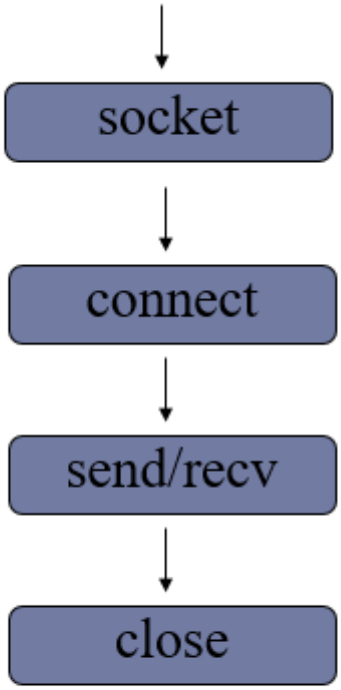
```
sock.close()
功能：关闭套接字
```

案例：

```
TCP服务端示例01：
'''
基于TCP协议的聊天程序服务器端
'''
from socket import *

# 1. 创建TCP socket对象
tcp_socket = socket(AF_INET, SOCK_STREAM)
# 2. 绑定地址
tcp_socket.bind(('0.0.0.0', 5555))
# 3. 设置监听
tcp_socket.listen(5)
# 4. 循环接收客户端连接
while True:
    print('等待客户端连接')
    conn, addr = tcp_socket.accept()
    # 5. 循环收发消息
    while True:
        # 收消息
        data = conn.recv(256)
        if data == b'##':
            break
        print(f'从客户端收到消息：{data.decode()}')
        # 发消息
        msg = input('>>')
        conn.send(msg.encode())
    conn.close()
# 6. 释放资源
tcp_socket.close()
```

2.3.3 TCP客户端



- 创建TCP套接字
- 请求连接

```
sock.connect(server_addr)
功能：连接服务器
参数：元组   服务器地址
```

- 收发消息

> 注意： 防止两端都阻塞，recv、send要配合

- 关闭套接字

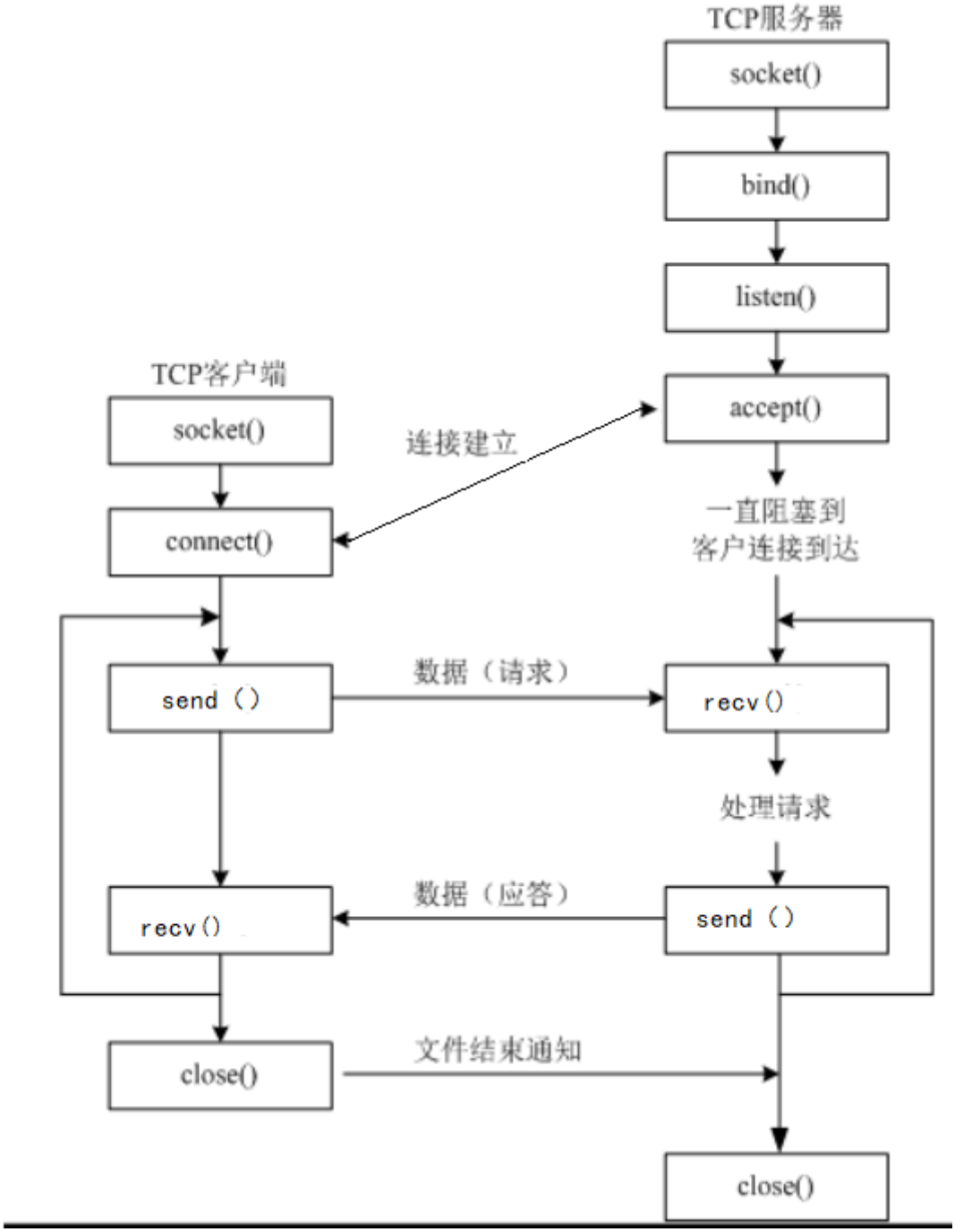
```
TCP客户端示例01：
'''
基于TCP协议聊天程序客户端
'''
from socket import *

ADDR = ('127.0.0.1', 5555)

# 1. 创建TCP socket对象
```

```python
tcp_socket = socket(AF_INET, SOCK_STREAM)
# 2. 连接服务器
tcp_socket.connect(ADDR)
# 3. 循环收发消息
while True:
    msg = input('>>')
    # 发消息
    tcp_socket.send(msg.encode())
    if msg == '##':
        break
    # 收消息
    data = tcp_socket.recv(256)
    print(f'从服务器收到消息：{data.decode()}')

# 4. 释放资源
tcp_socket.close()
```



随堂练习：

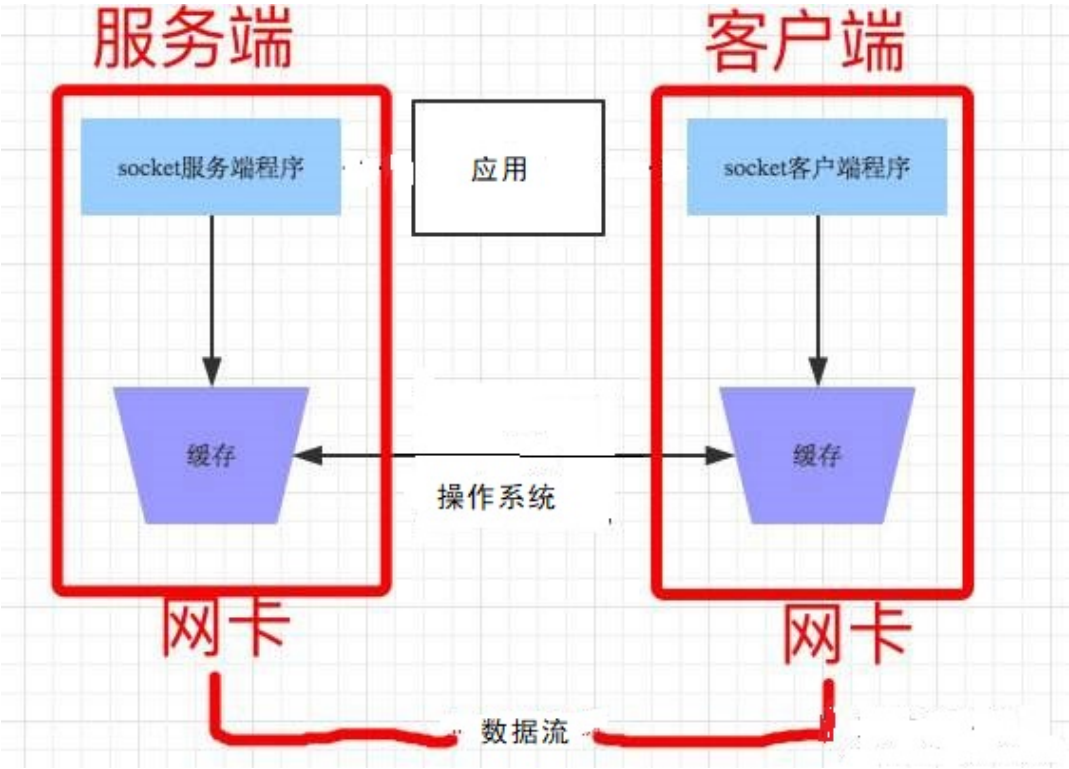
在客户端将一张图片上传到服务端，图片自选，上传到服务端后命名为 new.jpg

思路： 客户端 获取文件内容--》发送出去

服务端 接收文件内容--》写入磁盘

2.3.4 TCP套接字细节

- 一个服务端可以同时连接多个客户端，也能够重复被连接
- TCP粘包问题
  - 产生原因
    - 为了解决数据在传输过程中可能产生的速度不协调问题，操作系统设置了缓冲区
    - 实际网络工作过程比较复杂，导致消息收发速度不一致
    - TCP以字节流方式进行数据传输，在接收时不区分消息边界



  - 带来的影响
    - 如果每次发送内容是一个独立的含义，需要接收端独立解析，此时粘包会有影响
  - 处理方法
    - 消息格式化处理，如人为的添加消息边界，用作消息之间的分割
    - 控制发送的速度

随堂练习：
在客户端有一些数据

```
data = [
  "张三  18   177",
  "李四  19   180",
  "王五  120  183"
]
```

从客户端向服务端发送这些数据，在服务端将这些数据分别写入到一个文件中，每个数据占一行
客户端发送完成后向服务端发送 '#' 表示发送完毕

2.3.5 TCP与UDP对比

- 传输特征
  - TCP提供可靠的数据传输，但是UDP则不保证传输的可靠性
  - TCP传输数据处理为字节流，而UDP处理为数据包形式
  - TCP传输需要建立连接才能进行数据传，效率相对较低，UDP比较自由，无需连接，效率较高
- 套接字编程区别
  - 创建的套接字类型不同
  - TCP套接字会有粘包，UDP套接字有消息边界不会粘包
  - TCP套接字依赖listen()、accept()建立连接才能收发消息，UDP套接字则不需要
  - TCP套接字使用send()、recv()收发消息，UDP套接字使用sendto()、recvfrom()
- 使用场景
  - TCP更适合对数据准确性要求比较高的场景
    - 文件传输：如下载电影、上传照片
    - 邮件收发
    - 点对点数据传输：如点对点聊天、登录请求、远程访问、发红包
  - UDP更适合对可靠性要求没有那么高，实时性要求比较高的场景
    - 视频流的传输： 如部分直播、视频点播
    - 广播：如网络广播
    - 实时传输：如游戏画面

- 在一个大型的项目中，可能既涉及到TCP网络又有UDP网络

随堂练习：
完成一个对话小程序，客户端可以发送问题给服务端，服务端接收到问题将对应答案给客户端，客户端打印出来
要求可以同时多个客户端提问，如果问题没有指定答案，则回答 “人家还小，不知道。”

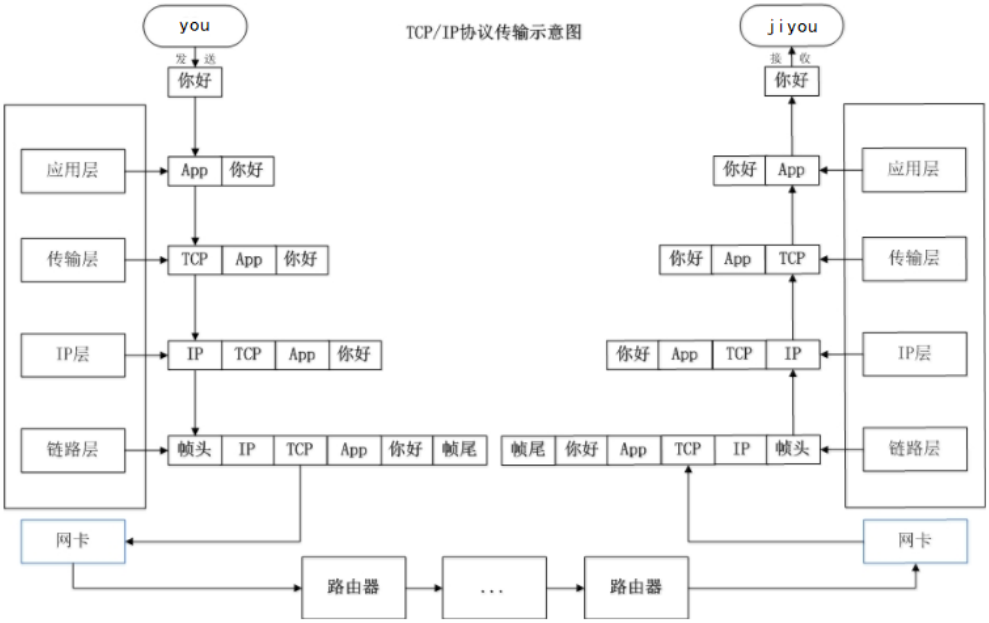
注意： 不需要使用数据库文件存储应答内容，在服务端用字典表示关键字和答案之间的对应关系即可

```
{"key":"value"}
data = {
    "你好": "你好啊",
    "叫什么": "我叫小美",
    "几岁": "我两岁啦",
    "男生女生": "我是机器人",
    "睡了": "吸引才是最重要的"
}
```

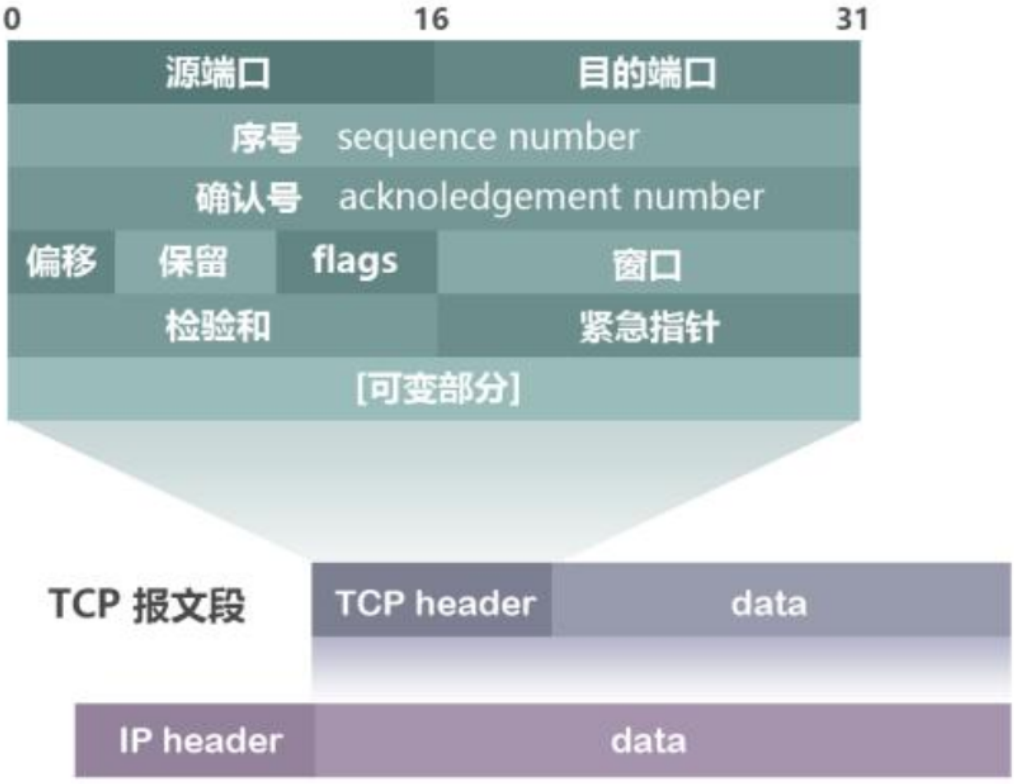
2.4 TCP协议和HTTP协议

2.4.1 传输流程

- 发送端由应用程序发送消息，逐层添加首部信息，最终在物理层发送消息包
- 发送的消息经过多个节点（交换机、路由器）传输，最终到达目标主机
- 目标主机由物理层逐层解析首部消息包，最终在应用程序呈现消息



2.4.2 TCP协议首部信息



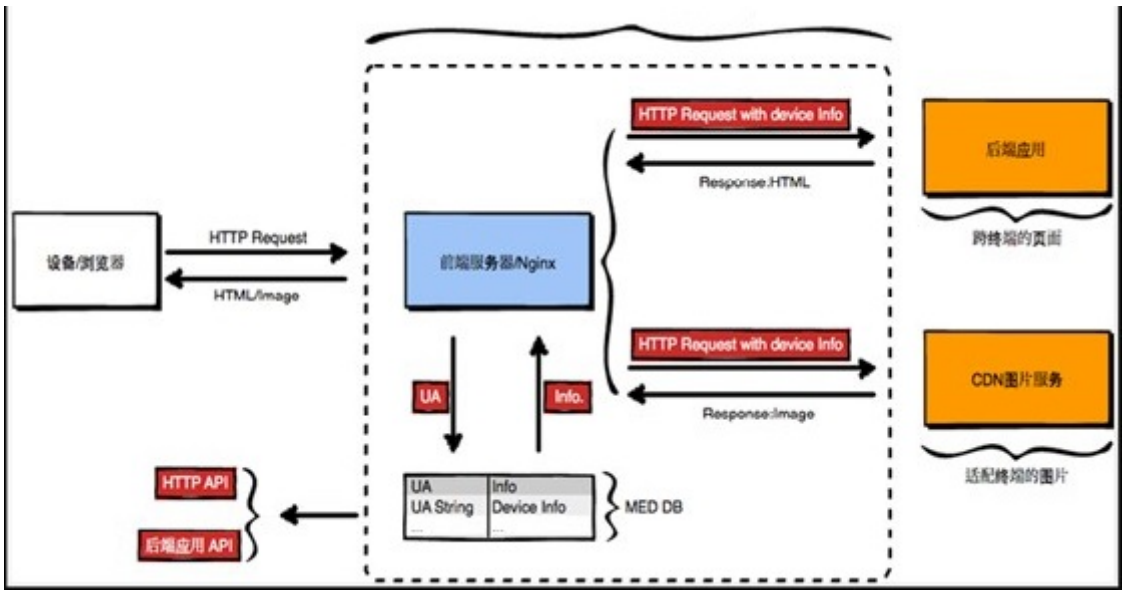
- 源端口和目的端口各占2个字节，分别写入源端口和目的端口。
- 序号占4字节，TCP是面向字节流的，在一个TCP连接中传送的字节流中的每一个字节都按顺序编号。例如，一报文段的序号是301，而接待的

数据共有100字节。这就表明本报文段数据的第一个字节的序号是301，最后一个字节的序号是400。

- 确认号占4字节，是期望收到对方下一个报文段的第一个数据字节的序号。例如，B正确收到了A发送过来的一个报文段，其序号字段值是501，而数据长度是200字节（序号501~700），这表明B正确收到了A发送的到序号700为止的数据。因此，B期望收到A的下一个数据序号是701，于是B在发送给A的确认报文段中把确认号置为701。
- 确认ACK（ACKnowledgment） 仅当ACK = 1时确认号字段才有效，当ACK = 0时确认号无效。TCP规定，在连接建立后所有的传送的报文段都必须把ACK置为1。
- 同步SYN（SYNchronization） 在连接建立时用来同步序号。当SYN=1而ACK=0时，表明这是一个连接请求报文段。对方若同意建立连接，则应在响应的报文段中使SYN=1和ACK=1，因此SYN置为1就表示这是一个连接请求或连接接受报文。
- 终止FIN（FINis，意思是“完”“终”） 用来释放一个连接。当FIN=1时，表明此报文段的发送发的数据已发送完毕，并要求释放运输连接。

2.4.3 网页访问流程

1. 客户端（浏览器）通过TCP传输，发送HTTP请求给服务端
2. 服务端接收到HTTP请求后进行解析
3. 服务端处理请求内容，组织响应内容
4. 服务端将响应内容以HTTP响应格式发送给浏览器
5. 浏览器接收到响应内容，解析展示



2.4.4 HTTP协议

- 用途： 网页获取，数据传输
- 特点
  - 应用层协议，使用TCP进行数据传输
  - 简单、灵活，很多语言都有HTTP专门接口
  - 有丰富的请求类型
  - 可以传输的数据类型众多

2.4.5 HTTP请求

- 请求行： 具体的请求类别和请求内容

```
GET             /           HTTP/1.1
请求类别      请求内容        协议版本
```

```
请求类别：每个请求类别表示要做不同的事情
GET ： 获取网络资源
POST ：提交一定的信息，得到反馈
HEAD ：只获取网络资源的响应头
PUT ：  更新服务器资源
DELETE ：删除服务器资源
```

- 请求头：对请求的进一步解释和描述

```
Accept-Encoding: gzip
```

- 空行
- 请求体：请求参数或者提交内容

2.4.6 HTTP响应

- 响应行：反馈基本的响应情况

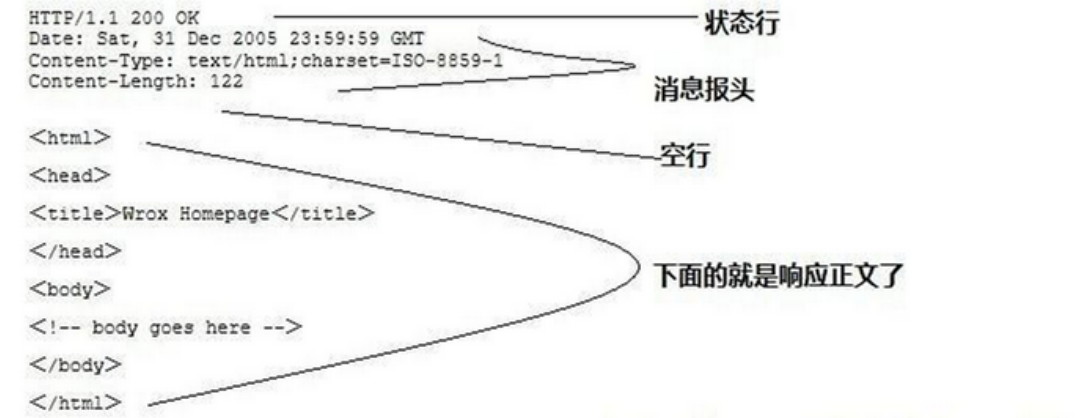
```
HTTP/1.1      200        OK
版本信息      响应码     附加信息
```

```
响应码 ：
1xx   提示信息，表示请求被接收
2xx   响应成功
3xx   响应需要进一步操作，重定向
4xx   客户端错误
5xx   服务器错误
```

- 响应头：对响应内容的描述

```
Content-Type: text/html
```

- 空行
- 响应体：响应的主体内容信息




```
"""
http请求和响应演示
"""
from socket import *

sock = socket()
sock.bind(("0.0.0.0",8000))
sock.listen(5)

# 等待浏览器连接
connfd,addr = sock.accept()
```

```
print("Connect from",addr)

# 接收HTTP请求
request = connfd.recv(1024)
print(request.decode())

# 组织响应
response = """HTTP/1.1 200 OK
Content-Type:text/html

hello world
"""
connfd.send(response.encode())

connfd.close()
sock.close()
```

随堂练习：将网页 一个图片 通过浏览器访问显示出来
提示 ： Content-Type:image/jpeg