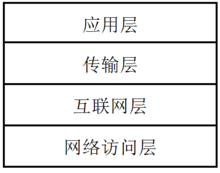
1. **Python的网络编程**

**第一部分**

**一、TCP/IP协议**

Transmission Control Protocol/Internet Protocol的简写，中译名为传输控制协议/[因特网](https://baike.baidu.com/item/%E5%9B%A0%E7%89%B9%E7%BD%91" \t "https://baike.baidu.com/item/TCP%2FIP%E5%8D%8F%E8%AE%AE/_blank)互联协议，又名网络[通讯协议](https://baike.baidu.com/item/%E9%80%9A%E8%AE%AF%E5%8D%8F%E8%AE%AE" \t "https://baike.baidu.com/item/TCP%2FIP%E5%8D%8F%E8%AE%AE/_blank)，是Internet最基本的协议、Internet国际[互联网](https://baike.baidu.com/item/%E4%BA%92%E8%81%94%E7%BD%91" \t "https://baike.baidu.com/item/TCP%2FIP%E5%8D%8F%E8%AE%AE/_blank)络的基础，由[网络层](https://baike.baidu.com/item/%E7%BD%91%E7%BB%9C%E5%B1%82" \t "https://baike.baidu.com/item/TCP%2FIP%E5%8D%8F%E8%AE%AE/_blank)的IP协议和[传输层](https://baike.baidu.com/item/%E4%BC%A0%E8%BE%93%E5%B1%82" \t "https://baike.baidu.com/item/TCP%2FIP%E5%8D%8F%E8%AE%AE/_blank)的TCP协议组成。TCP/IP 定义了电子设备如何连入[因特网](https://baike.baidu.com/item/%E5%9B%A0%E7%89%B9%E7%BD%91" \t "https://baike.baidu.com/item/TCP%2FIP%E5%8D%8F%E8%AE%AE/_blank)，以及数据如何在它们之间传输的标准。协议采用了4层的层级结构，每一层都呼叫它的下一层所提供的协议来完成自己的需求。[通俗](https://baike.baidu.com/item/%E9%80%9A%E4%BF%97" \t "https://baike.baidu.com/item/TCP%2FIP%E5%8D%8F%E8%AE%AE/_blank)而言：TCP负责发现[传输](https://baike.baidu.com/item/%E4%BC%A0%E8%BE%93" \t "https://baike.baidu.com/item/TCP%2FIP%E5%8D%8F%E8%AE%AE/_blank)的问题，一有问题就发出信号，要求重新传输，直到所有[数据安全](https://baike.baidu.com/item/%E6%95%B0%E6%8D%AE%E5%AE%89%E5%85%A8" \t "https://baike.baidu.com/item/TCP%2FIP%E5%8D%8F%E8%AE%AE/_blank)正确地传输到目的地。而IP是给[因特网](https://baike.baidu.com/item/%E5%9B%A0%E7%89%B9%E7%BD%91" \t "https://baike.baidu.com/item/TCP%2FIP%E5%8D%8F%E8%AE%AE/_blank)的每一台联网设备规定一个地址。



网络访问层(Network Access Layer)在TCP/IP参考模型中并没有详细描述，只是指出主机必须使用某种协议与网络相连。

互联网层(Internet Layer)是整个体系结构的关键部分，其功能是使主机可以把分组发往任何网络，并使分组独立地传向目标。这些分组可能经由不同的网络，到达的顺序和发送的顺序也可能不同。高层如果需要顺序收发，那么就必须自行处理对分组的排序。互联网层使用因特网协议(IP，Internet Protocol)。

传输层(Tramsport Layer)使源端和目的端机器上的对等实体可以进行会话。在这一层定义了两个端到端的协议：传输控制协议(TCP，Transmission Control Protocol)和用户数据报协议(UDP，User Datagram Protocol)。TCP是面向连接的协议，它提供可靠的报文传输和对上层应用的连接服务。为此，除了基本的数据传输外，它还有可靠性保证、流量控制、多路复用、优先权和安全性控制等功能。UDP是面向无连接的不可靠传输的协议，主要用于不需要TCP的排序和流量控制等功能的应用程序。

应用层(Application Layer)包含所有的高层协议，包括：虚拟终端协议(TELNET，TELecommunications NETwork)、文件传输协议(FTP，File Transfer Protocol)、电子邮件传输协议(SMTP，Simple Mail Transfer Protocol)、域名服务(DNS，Domain Name Service)、网上新闻传输协议(NNTP，Net News Transfer Protocol)和超文本传送协议(HTTP，HyperText Transfer Protocol)等。TELNET允许一台机器上的用户登录到远程机器上，并进行工作；FTP提供有效地将文件从一台机器上移到另一台机器上的方法；SMTP用于电子邮件的收发；DNS用于把主机名映射到网络地址；NNTP用于新闻的发布、检索和获取；HTTP用于在WWW上获取主页。

**1.1 UDP协议**

1、UDP协议适用端口分别运行在同一台设备上的多个应用程序，UDP有不提供数据报分组、组装和不能对数据包进行排序的缺点。也就是说，当报文发送之后，是无法得知其是否安全完整到达的。  
 2、在网络质量令人不十分满意的环境下，UDP协议数据包丢失会比较严重。  
 3、但是由于UDP的特性：它不属于连接型协议，因而具有资源消耗小，处理速度快的优点，所以通常音频、视频和普通数据在传送时使用UDP较多，因为它们即使偶尔丢失一两个数据包，也不会对接收结果产生太大影响。  
 **1.2 TCP协议**  
 1、TCP提供端到端、全双工通信；采用字节流方式，如果字节流太长，将其分段；提供紧急数据传送功能。  
 2、TCP特性：  
 （1）面向连接的传输；   
 （2）端到端的通信；   
 （3）高可靠性，确保传输数据的正确性，不出现丢失或乱序；   
 （4）全双工方式传输；

全双工（Full Duplex）是通讯传输的一个术语。通信允许数据在两个方向上同时传输，它在能力上相当于两个单工通信方式的结合。全双工指可以同时（瞬时）进行信号的双向传输（A→B且B→A）。指A→B的同时B→A，是瞬时同步的。

[单工](https://baike.baidu.com/item/%E5%8D%95%E5%B7%A5" \t "https://baike.baidu.com/item/%E5%85%A8%E5%8F%8C%E5%B7%A5/_blank)就是在只允许甲方向乙方传送信息，而乙方不能向甲方传送 。（比喻汽车的单行道。）

（5）采用字节流方式，即以字节为单位传输字节序列；   
 （6）紧急数据传送功能。  
 如果需要传输的数据是准确的，建议采用TCP，也就是SOCK\_STREAM,如果传输的是视频、音频等数据，丢几个包也无所谓的，可以采用UDP，也就是SOCK\_DGRAM。

SOCK\_STREAM是有保障的（即能保证数据正确传送到对方）面向连接的SOCKET，多用于资料（如文件）传送。SOCK\_DGRAM 是无保障的面向消息的socket ， 主要用于在网络上发广播信息。SOCK\_STREAM是基于TCP的，数据传输比较有保障。SOCK\_DGRAM是基于UDP的，专门用于局域网，基于广播SOCK\_STREAM 是数据流,一般是tcp/ip协议的编程,SOCK\_DGRAM分是数据包,是udp协议网络编程

**二、IPv4和IPv6**

IPv4，是[互联网协议](https://baike.baidu.com/item/%E4%BA%92%E8%81%94%E7%BD%91%E5%8D%8F%E8%AE%AE)（Internet Protocol，IP）的第四版，也是第一个被广泛使用，构成现今[互联网技术](https://baike.baidu.com/item/%E4%BA%92%E8%81%94%E7%BD%91%E6%8A%80%E6%9C%AF)的基础的协议，的下一个版本就是[IPv6](https://baike.baidu.com/item/IPv6)。IPv6正处在不断发展和完善的过程中，它在不久的将来将取代目前被广泛使用的IPv4。

IPv4有4[位字段](https://baike.baidu.com/item/%E4%BD%8D%E5%AD%97%E6%AE%B5)，共32位比特，IPv6有16位字段，共128位比特。

[AF\_INET](https://www.baidu.com/s?wd=AF_INET&tn=44039180_cpr&fenlei=mv6quAkxTZn0IZRqIHckPjm4nH00T1dBrjDzm1cdrAF-mWF-rjf10ZwV5Hcvrjm3rH6sPfKWUMw85HfYnjn4nH6sgvPsT6KdThsqpZwYTjCEQLGCpyw9Uz4Bmy-bIi4WUvYETgN-TLwGUv3EnHcLPHf3rHm4)（又称PF\_INET）是IPv4网络协议的套接字类型，[AF\_INET](https://www.baidu.com/s?wd=AF_INET&tn=44039180_cpr&fenlei=mv6quAkxTZn0IZRqIHckPjm4nH00T1dBrjDzm1cdrAF-mWF-rjf10ZwV5Hcvrjm3rH6sPfKWUMw85HfYnjn4nH6sgvPsT6KdThsqpZwYTjCEQLGCpyw9Uz4Bmy-bIi4WUvYETgN-TLwGUv3EnHcLPHf3rHm4)6则是 [IPv6](https://www.baidu.com/s?wd=IPv6&tn=44039180_cpr&fenlei=mv6quAkxTZn0IZRqIHckPjm4nH00T1dBrjDzm1cdrAF-mWF-rjf10ZwV5Hcvrjm3rH6sPfKWUMw85HfYnjn4nH6sgvPsT6KdThsqpZwYTjCEQLGCpyw9Uz4Bmy-bIi4WUvYETgN-TLwGUv3EnHcLPHf3rHm4) 的；而F\_UNIX 则是 Unix 系统本地通信。选择 [AF\_INET](https://www.baidu.com/s?wd=AF_INET&tn=44039180_cpr&fenlei=mv6quAkxTZn0IZRqIHckPjm4nH00T1dBrjDzm1cdrAF-mWF-rjf10ZwV5Hcvrjm3rH6sPfKWUMw85HfYnjn4nH6sgvPsT6KdThsqpZwYTjCEQLGCpyw9Uz4Bmy-bIi4WUvYETgN-TLwGUv3EnHcLPHf3rHm4) 的目的就是使用 IPv4 进行通信。因为 IPv4 使用 32 位地址，相比 [IPv6](https://www.baidu.com/s?wd=IPv6&tn=44039180_cpr&fenlei=mv6quAkxTZn0IZRqIHckPjm4nH00T1dBrjDzm1cdrAF-mWF-rjf10ZwV5Hcvrjm3rH6sPfKWUMw85HfYnjn4nH6sgvPsT6KdThsqpZwYTjCEQLGCpyw9Uz4Bmy-bIi4WUvYETgN-TLwGUv3EnHcLPHf3rHm4) 的 128 位来说，计算更快，便于用于局域网通信。而且 AF\_INET 相比 AF\_UNIX 更具通用性，因为 Windows 上有 AF\_INET 而没有 AF\_UNIX。

**第二部分**

**三、socket**

我们深谙信息交流的价值，那网络中进程之间如何通信，如我们每天打开浏览器浏览网页时，浏览器的进程怎么与web服务器通信的？当你用QQ聊天时，QQ进程怎么与服务器或你好友所在的QQ进程通信？这些都得靠socket？那什么是socket？socket的类型有哪些？还有socket的基本函数，这些都是本文想介绍的。本文的主要内容如下：

 1、网络中进程之间如何通信？

 2、Socket是什么？

 3、socket的基本操作

3.1、socket()函数

3.2、bind()函数

3.3、listen()、connect()函数

3.4、accept()函数

3.5、read()、write()函数等

3.6、close()函数



**1、网络中进程之间如何通信？**

首要解决的问题是如何唯一标识一个进程，否则通信无从谈起！在本地可以通过进程PID来唯一标识一个进程，但是在网络中这是行不通的。其实TCP/IP协议族已经帮我们解决了这个问题，网络层的“ip地址”可以唯一标识网络中的主机，而传输层的“协议+端口”可以唯一标识主机中的应用程序（进程）。这样利用三元组（ip地址，协议，端口）就可以标识网络的进程了，网络中的进程通信就可以利用这个标志与其它进程进行交互。

使用TCP/IP协议的应用程序通常采用应用编程接口：UNIX BSD的套接字（socket）和UNIX System V的TLI（已经被淘汰），来实现网络进程之间的通信。就目前而言，几乎所有的应用程序都是采用socket，而现在又是网络时代，网络中进程通信是无处不在，这就是我为什么说“一切皆socket”。

**2、什么是Socket？**

上面我们已经知道网络中的进程是通过socket来通信的，那什么是socket呢？socket起源于Unix，而Unix/Linux基本哲学之一就是“一切皆文件”，都可以用“打开open –> 读写write/read –> 关闭close”模式来操作。我的理解就是Socket就是该模式的一个实现，socket即是一种特殊的文件，一些socket函数就是对其进行的操作（读/写IO、打开、关闭），这些函数我们在后面进行介绍。

**3、socket的基本操作**

既然socket是“open—write/read—close”模式的一种实现，那么socket就提供了这些操作对应的函数接口。下面以TCP为例，介绍几个基本的socket接口函数。

|  |  |  |
| --- | --- | --- |
| 服务器端套接字 | |  |
| s.bind() | 绑定地址（host,port）到套接字， 在AF\_INET下,以元组（host,port）的形式表示地址。 |  |
| s.listen() | 开始TCP监听。backlog指定在拒绝连接之前，操作系统可以挂起的最大连接数量。该值至少为1，大部分应用程序设为5就可以了。 |  |
| s.accept() | 被动接受TCP客户端连接,(阻塞式)等待连接的到来 |  |
| 客户端套接字 | |  |
| s.connect() | 主动初始化TCP服务器连接，。一般address的格式为元组（hostname,port），如果连接出错，返回socket.error错误。 |  |
| s.connect\_ex() | connect()函数的扩展版本,出错时返回出错码,而不是抛出异常 |  |
| 公共用途的套接字函数 | |  |
| s.recv() | 接收TCP数据，数据以字符串形式返回，bufsize指定要接收的最大数据量。flag提供有关消息的其他信息，通常可以忽略。 |  |
| s.send() | 发送TCP数据，将string中的数据发送到连接的套接字。返回值是要发送的字节数量，该数量可能小于string的字节大小。 |  |
| s.sendall() | 完整发送TCP数据，完整发送TCP数据。将string中的数据发送到连接的套接字，但在返回之前会尝试发送所有数据。成功返回None，失败则抛出异常。 |  |
| s.recvfrom() | 接收UDP数据，与recv()类似，但返回值是（data,address）。其中data是包含接收数据的字符串，address是发送数据的套接字地址。 |  |
| s.sendto() | 发送UDP数据，将数据发送到套接字，address是形式为（ipaddr，port）的元组，指定远程地  址。返回值是发送的字节数。 |  |
| s.close() | 关闭套接字 |  |

**3.1、socket()函数**

socket(int domain, int type, int protocol);

socket函数对应于普通文件的打开操作。普通文件的打开操作返回一个文件描述字，而socket()用于创建一个socket描述符（socket descriptor），它唯一标识一个socket。这个socket描述字跟文件描述字一样，后续的操作都有用到它，把它作为参数，通过它来进行一些读写操作。

正如可以给fopen的传入不同参数值，以打开不同的文件。创建socket的时候，也可以指定不同的参数创建不同的socket描述符，socket函数的三个参数分别为：

 1、domain：即协议域，又称为协议族（family）。常用的协议族有，AF\_INET、AF\_INET6、AF\_LOCAL（或称AF\_UNIX，Unix域socket）、AF\_ROUTE等等。协议族决定了socket的地址类型，在通信中必须采用对应的地址，如AF\_INET决定了要用ipv4地址（32位的）与端口号（16位的）的组合、AF\_UNIX决定了要用一个绝对路径名作为地址。

 2、type：指定socket类型。常用的socket类型有，SOCK\_STREAM、SOCK\_DGRAM、SOCK\_RAW、SOCK\_PACKET、SOCK\_SEQPACKET等等（socket的类型有哪些？）。

3、protocol：故名思意，就是指定协议。常用的协议有，IPPROTO\_TCP、IPPTOTO\_UDP等，它们分别对应TCP传输协议、UDP传输协议。

注意：并不是上面的type和protocol可以随意组合的，如SOCK\_STREAM不可以跟IPPROTO\_UDP组合。当protocol为0时，会自动选择type类型对应的默认协议。

当我们调用socket创建一个socket时，返回的socket描述字它存在于协议族（address family，AF\_XXX）空间中，但没有一个具体的地址。如果想要给它赋值一个地址，就必须调用bind()函数，否则就当调用connect()、listen()时系统会自动随机分配一个端口。

例如： socket\_client=socket.socket(socket.AF\_INET,socket.SOCK\_STREAM)

**3.2、bind()函数**

正如上面所说bind()函数把一个地址族中的特定地址赋给socket。例如对应AF\_INET、AF\_INET6就是把一个ipv4或ipv6地址和端口号组合赋给socket。

通常服务器在启动的时候都会绑定一个众所周知的地址（如ip地址+端口号），用于提供服务，客户就可以通过它来接连服务器；而客户端就不用指定，有系统自动分配一个端口号和自身的ip地址组合。这就是为什么通常服务器端在listen之前会调用bind()，而客户端就不会调用，而是在connect()时由系统随机生成一个。

例如：socket\_server.bind((host,9999))

**3.3、listen()、connect()函数**

如果作为一个服务器，在调用socket()、bind()之后就会调用listen()来监听这个socket，如果客户端这时调用connect()发出连接请求，服务器端就会接收到这个请求。

int listen(int sockfd, int backlog);

int connect(int sockfd, conststruct sockaddr \*addr, socklen\_t addrlen);

listen函数的第一个参数即为要监听的socket描述字，第二个参数为相应socket可以排队的最大连接个数。socket()函数创建的socket默认是一个主动类型的，listen函数将socket变为被动类型的，等待客户的连接请求。

connect函数的第一个参数即为客户端的socket描述字，第二参数为服务器的socket地址，第三个参数为socket地址的长度。客户端通过调用connect函数来建立与TCP服务器的连接。

例如：socket\_server.listen(5)

socket\_client.connect((serverHost,9999))

**3.4、accept()函数**

TCP服务器端依次调用socket()、bind()、listen()之后，就会监听指定的socket地址了。TCP客户端依次调用socket()、connect()之后就想TCP服务器发送了一个连接请求。TCP服务器监听到这个请求之后，就会调用accept()函数取接收请求，这样连接就建立好了。之后就可以开始网络I/O操作了，即类同于普通文件的读写I/O操作。

注意：一个服务器通常通常仅仅只创建一个监听socket描述字，它在该服务器的生命周期内一直存在。内核为每个由服务器进程接受的客户连接创建了一个已连接socket描述字，当服务器完成了对某个客户的服务，相应的已连接socket描述字就被关闭。

例如：serverSocket, addressinfo=socket\_server.accept()

**3.5、read()、write()等函数**

万事具备只欠东风，至此服务器与客户已经建立好连接了。可以调用网络I/O进行读写操作了，即实现了网咯中不同进程之间的通信！网络I/O操作有下面几组：

 read()/write()

 recv()/send()

 readv()/writev()

 recvmsg()/sendmsg()

 recvfrom()/sendto()

推荐使用recvmsg()/sendmsg()函数，这两个函数是最通用的I/O函数，实际上可以把上面的其它函数都替换成这两个函数,read函数是负责从fd中读取内容.当读成功时，read返回实际所读的字节数，如果返回的值是0表示已经读到文件的结束了，小于0表示出现了错误。如果错误为EINTR说明读是由中断引起的，如果是ECONNREST表示网络连接出了问题;write函数将buf中的nbytes字节内容写入文件描述符fd.成功时返回写的字节数。失败时返回-1，并设置errno变量。 在网络程序中，当我们向套接字文件描述符写时有俩种可能。

1)write的返回值大于0，表示写了部分或者是全部的数据。

2)返回的值小于0，此时出现了错误。我们要根据错误类型来处理。如果错误为EINTR表示在写的时候出现了中断错误。如果为EPIPE表示网络连接出现了问题(对方已经关闭了连接)。

**3.6、close()函数**

在服务器与客户端建立连接之后，会进行一些读写操作，完成了读写操作就要关闭相应的socket描述字，好比操作完打开的文件要调用fclose关闭打开的文件。

close一个TCP socket的缺省行为时把该socket标记为以关闭，然后立即返回到调用进程。该描述字不能再由调用进程使用，也就是说不能再作为read或write的第一个参数。

注意：close操作只是使相应socket描述字的引用计数-1，只有当引用计数为0的时候，才会触发TCP客户端向服务器发送终止连接请求。

**第三部分**

1. **使用socket编程的步骤**

**4.1 TCP协议的简单例子**

**4.1.1、客户端的步骤**

1、导入模块

2、创建一个socket对象

3、得到客户端主机名字

4、连接服务端

5、收发数据

6、关闭

**4.1.2 客户端例子：**

*#encoding=utf-8***import** socket  
socket\_client=socket.socket(socket.AF\_INET,socket.SOCK\_STREAM)  
clientHost=socket.gethostname()  
print(**"客户端准备连接服务端"**)  
socket\_client.connect((clientHost,9999))  
print(**"客户端连上服务端"**)  
sendData=**"Hello"**socket\_client.send(sendData.encode(**"utf-8"**))  
print(**"客户端发送数据：{0}"**.format(sendData))  
socket\_client.close()  
print(**"客户端关闭"**)

**4.1.3 服务端的步骤**

1、导入模块

2、创建一个socket对象

3、获取本机的机器名  
4、服务端绑定

5、侦听

6、服务端接/发数据

7、关闭

**4.1.4 服务端简单案例**

*#encoding=utf-8  
#1、导入包***import** socket  
*#2、创建一个socket对象*socket\_server=socket.socket(socket.AF\_INET,socket.SOCK\_STREAM)  
*#print(socket1)***'''  
<socket.socket fd=280, family=AddressFamily.AF\_INET, type=SocketKind.SOCK\_STREAM, proto=0>  
当protocol为0时，会自动选择type类型对应的默认协议  
fd 是(file descriptor)，这种一般是BSD Socket的用法，用在Unix/Linux系统上。  
在Unix/Linux系统下，一个socket句柄，可以看做是一个文件，在socket上收发数据，  
相当于对一个文件进行读写，  
所以一个socket句柄，通常也用表示文件句柄的fd来表示。  
'''***#3、获取本机的机器名*host=socket.gethostname()  
*#4、服务端绑定*socket\_server.bind((host,9999))  
*#5、侦听*socket\_server.listen(5)  
print(**"服务器处于侦听状态......"**)  
*#6、服务端接收客户端的请求*serverSocket, addressinfo=socket\_server.accept()  
print(**"服务端接收来自{0}的{1}端口的连接请求"**.format(addressinfo[0],addressinfo[1]))  
print(**"服务端准备接收数据:"**)  
revicedData= serverSocket.recv(256)  
print(**"服务端接收到的数据:"**)  
print(revicedData.decode(**"utf-8"**))  
serverSocket.close()  
socket\_server.close()  
print(**"服务端关闭"**)

**第四部分**

由于UDP是无连接的，与TCP的区别在服务器端表现为，无需监听函数listen()对客户端的连接进行监听； 在客户端的表现为客户端的socket无需与服务器建立连接就能够进行数据的发送与接收，即无需connect()函数。

**4.2 UDP协议的简单例子**

**4.2.1 服务端步骤**

1、导入模块

2、创建一个socket

3、绑定服务器套接字

4、接收/发送数据

5、关闭

**4.2.2 服务端的例子**

*#encoding=utf-8  
#1、导入模块***import** socket  
*#2、创建一个socket*udpSocket=socket.socket(socket.AF\_INET,socket.SOCK\_DGRAM)  
*#print(udpSocket)  
#3、绑定服务器套接字*udpSocket.bind((**"127.0.0.1"**,9999))  
*#4、接收/发送数据*data, address\_info=udpSocket.recvfrom(512)  
print(data.decode(**"utf-8"**))  
*#5、关闭*udpSocket.close()

**4.2.3 客户端步骤**

1、导入模块

2、创建一个socket对象

3、向服务器端发送数据

4、关闭

**4.2.4 客户端的例子**

*#encoding=utf-8  
#1、导入模块***import** socket  
*#2、创建一个socket对象*udpclient=socket.socket(socket.AF\_INET,socket.SOCK\_DGRAM)  
*#3、向服务器端发送数据*serverHost=(**"127.0.0.1"**,9999)  
udpclient.sendto(**"hello"**.encode(**"utf-8"**),serverHost)  
*#4、关闭*udpclient.close()

**第五部分**

1. **简单案例**

**4.1 基于TCP协议的QQ聊天程序**

**1、Server.py**

*#encoding=utf-8  
#1、导入包***import** socket  
*#2、创建一个socket对象*socket\_server=socket.socket(socket.AF\_INET,socket.SOCK\_STREAM)  
*#3、获取本机的机器名*host=socket.gethostname()  
*#4、服务端绑定*socket\_server.bind((host,9999))  
*#5、侦听*socket\_server.listen(5)  
print(**"QQ服务端启动......"**)  
*#6、服务端接收客户端的请求*serverSocket, addressinfo=socket\_server.accept()  
print(**"服务端接收来自{0}的{1}端口的连接请求"**.format(addressinfo[0],addressinfo[1]))  
**while True**:  
 print(**"准备接收来自客户端的信息："**)  
 revicedData= serverSocket.recv(256)  
 **if** revicedData.decode(**"utf-8"**)!=**'quit'**:  
 print(revicedData.decode(**"utf-8"**))  
 **else**:  
 print(**"对方已经退出，QQ退出!"**)  
 **break** sendData=input(**"请输入发送给客户端的信息:"**)  
 serverSocket.sendall(sendData.encode(**"utf-8"**))  
 **if** sendData==**'quit'**:  
 print(**"QQ退出"**)  
 **break**serverSocket.close()  
socket\_server.close()

2、client.py

*#encoding=utf-8***import** socket  
socket\_client=socket.socket(socket.AF\_INET,socket.SOCK\_STREAM)  
clientHost=socket.gethostname()  
socket\_client.connect((clientHost,9999))  
print(**"QQ客户端上线:"**)  
**while True**:  
 sendData=input(**"请输入发送到服务端的数据:"**)  
 socket\_client.send(sendData.encode(**"utf-8"**))  
 **if** sendData==**'quit'**:  
 print(**"QQ退出!"**)  
 **break** print(**"准备接收来自服务端的信息："**)  
 revicedData=socket\_client.recv(512)  
 **if** revicedData.decode(**"utf-8"**)!=**'quit'**:  
 print(revicedData.decode(**"utf-8"**))  
 **else**:  
 print(**"对方已经退出，QQ退出!"**)  
 **break**socket\_client.close()

**第六部分**

4.2 基于UDP协议的QQ聊天程序

QQ的一端：

*#encoding=utf-8  
#1、导入模块***import** socket  
*#2、创建一个socket*udpSocket=socket.socket(socket.AF\_INET,socket.SOCK\_DGRAM)  
*#print(udpSocket)  
#3、绑定服务器套接字*udpSocket.bind((**"127.0.0.1"**,9999))  
*#4、接收/发送数据*print(**"QQ聊天程序启动："**)  
clientHost=(**"127.0.0.1"**,19999)  
**while True**:  
 data, address\_info=udpSocket.recvfrom(512)  
 **if** data.decode(**"utf-8"**)!=**'quit'**:  
 print(**"服务端接收到的数据："**)  
 print(data.decode(**"utf-8"**))  
 sendData=input(**"请输入服务端发送给客户端的数据"**)  
 udpSocket.sendto(bytes(sendData.encode(**"utf-8"**)),clientHost)  
 **if** sendData==**'quit'**:  
 print(**"关闭QQ聊天程序!"**)  
 **break  
 else**:  
 print(**"关闭QQ聊天程序!"**)  
 **break***#5、关闭*udpSocket.close()

**QQ的另一端：**

*#encoding=utf-8  
#1、导入模块***import** socket  
*#2、创建一个socket对象*udpclient=socket.socket(socket.AF\_INET,socket.SOCK\_DGRAM)  
*#3、向服务器端发送数据*serverHost=(**"127.0.0.1"**,9999)  
*#绑定客户端，用来接收服务端的数据*clientHost=(**"127.0.0.1"**,19999)  
udpclient.bind(clientHost)  
print(**"QQ聊天程序启动："**)  
**while True**:  
 send\_data=input(**"请输入聊天内容:"**)  
 udpclient.sendto(send\_data.encode(**"utf-8"**),serverHost)  
 **if** send\_data==**'quit'**:  
 print(**"退出QQ"**)  
 **break** print(**"客户端接收到的数据："**)  
 reciveData,address\_infro=udpclient.recvfrom(512)  
 **if** reciveData.decode(**"utf-8"**)!=**'quit'**:  
 print(reciveData.decode(**"utf-8"**))  
 **else**:  
 print(**"退出QQ"**)  
 **break***#4、关闭*udpclient.close()

**第七部分**

**五、实际开发应用**

5.1：通过socket发送文件

Client:

*#1、导入包***import** sys  
**import** socket  
client=**None  
def** getClient():  
 *# 2、创建客户端的socket对象* **global** client  
 client = socket.socket(socket.AF\_INET, socket.SOCK\_STREAM)  
 *# 获取本地主机名* host = socket.gethostname()  
 *# 3、绑定* client.connect((host,9999))  
 print(**"客户端连接上服务端:"**)  
 **return** client  
  
**def** getFile(filename):  
 getClient()  
 n=0  
 f=open(filename,mode=**"rb"**)  
 fc=f.read(1024\*1000)  
 f.flush()  
 print(**"客户端发送文件:"**)  
 **global** client  
 client.sendall(fc)  
 **while** fc:  
 fc=f.read(1024\*1000)  
 f.flush()  
 n=n+1  
 client.sendall(fc)  
 print(n)  
  
 client.close()  
  
  
getFile(**"1.mp4"**)

Server:

*#encoding=utf-8***import** sys  
**import** socket  
*#2、创建服务端的socket对象*service=socket.socket(socket.AF\_INET,socket.SOCK\_STREAM)  
*#获取本地主机名*host=socket.gethostname()  
*#3、绑定*service.bind((host,9999))  
*#4、设置侦听，其中5表示最大连接数*service.listen(5)  
T=1  
*#5、接收连接*print(**"服务端启动："**)  
serviceSocket,addr=service.accept()  
f = open(**"czf.mp4"**, mode=**"ab+"**)  
**while True**:  
 print(**"客户端地址:%s"** %str(addr))  
 print(**"服务端接收文件："**)  
 fileContent=serviceSocket.recv(1024\*1000)  
 print(len(fileContent))  
 f.write(fileContent)  
 f.flush()  
 **if** len(fileContent)==0:  
 f.close()  
 **break**

**第八部分**5.2 生成器+文件读写+网络编程 **只要修改上面的客户端即可：**

*#1、导入包***import** sys  
**import** socket  
client=**None**

**def** getClient():  
 *# 2、创建客户端的socket对象* **global** client  
 client = socket.socket(socket.AF\_INET, socket.SOCK\_STREAM)  
 *# 获取本地主机名* host = socket.gethostname()  
 *# 3、绑定* client.connect((host,9999))  
 print(**"客户端连接上服务端:"**)  
 **return** client

**def** sendFile():  
 n=1  
 **global** client  
 getClient()  
 **for** w **in** getFile(**"1.mp4"**):  
 client.sendall(w)  
 print(n)  
 n=n+1  
 client.close()

*#读取文件的生成器***def** getFile(filename):  
 f=open(filename,mode=**"rb"**)  
 fc=f.read(1024\*1000)  
 f.flush()  
 **yield** fc  
 **while** fc:  
 fc=f.read(1024\*1000)  
 f.flush()  
 **yield** fc  
 f.close()  
sendFile()