# 第十五章 Socket

本节所讲内容：

15.1 socket介绍

15.2 socket编程

15.3 socket TCP

15.4 socket UDP

15.5 SocketServer

## 15.1 socket介绍

### 15.1.1 socket介绍

网络上的两个程序通过一个双向的通信连接实现数据的交换，这个连接的一端称为一个socket。

建立网络通信连接至少要一对端口号(socket)。socket本质是编程接口(API)，对TCP/IP的封装，TCP/IP也要提供可供程序员做网络开发所用的接口，这就是Socket编程接口；HTTP是轿车，提供了封装或者显示数据的具体形式；Socket是发动机，提供了网络通信的能力。

Socket的英文原义是“孔”或“插座”。作为BSD UNIX的[进程通信](https://baike.baidu.com/item/%E8%BF%9B%E7%A8%8B%E9%80%9A%E4%BF%A1" \t "_blank)机制，取后一种意思。通常也称作"[套接字](https://baike.baidu.com/item/%E5%A5%97%E6%8E%A5%E5%AD%97)"，用于描述IP地址和端口，是一个通信链的句柄，可以用来实现不同虚拟机或不同计算机之间的通信。在Internet上的[主机](https://baike.baidu.com/item/%E4%B8%BB%E6%9C%BA" \t "_blank)一般运行了多个服务软件，同时提供几种服务。每种服务都打开一个Socket，并绑定到一个端口上，不同的端口对应于不同的服务。Socket正如其英文原义那样，像一个多孔插座。一台主机犹如布满各种插座的房间，每个插座有一个编号，有的插座提供220伏交流电，有的提供110伏交流电，有的则提供有线电视节目。 客户软件将插头插到不同编号的插座，就可以得到不同的服务。



Socket非常类似于电话插座。以一个国家级电话网为例，电话的通话双方相当于相互通信的2个进程，区号是它的[网络地址](https://baike.baidu.com/item/%E7%BD%91%E7%BB%9C%E5%9C%B0%E5%9D%80" \t "_blank)；区内一个单位的交换机相当于一台[主机](https://baike.baidu.com/item/%E4%B8%BB%E6%9C%BA)，主机分配给每个用户的局内号码相当于Socket号。任何用户在通话之前，首先要占有一部电话机，相当于申请一个Socket；同时要知道对方的号码，相当于对方有一个固定的Socket。然后向对方拨号呼叫，相当于发出连接请求（假如对方不在同一区内，还要拨对方区号，相当于给出网络地址）。假如对方在场并空闲（相当于通信的另一主机开机且可以接受连接请求），拿起电话话筒，双方就可以正式通话，相当于连接成功。双方通话的过程，是一方向电话机发出信号和对方从电话机接收信号的过程，相当于向Socket发送数据和从socket接收数据。通话结束后，一方挂起电话机相当于关闭Socket，撤消连接。

## 15.2 socket编程

### 15.2.1 socket协议

TCP: 面向连接的通信协议

建立连接：三次握手

1.第一次握手:建立连接。

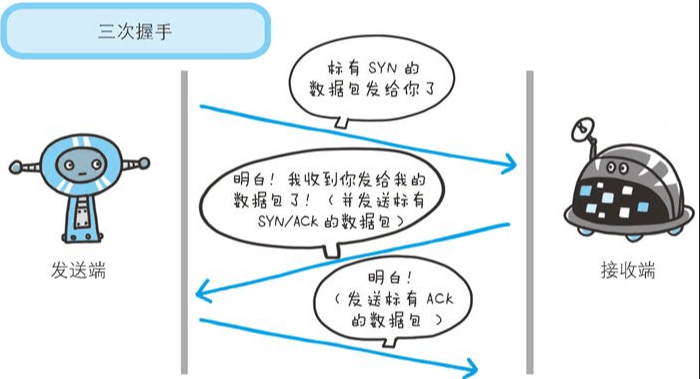
     客户端发送连接请求报文，将SYN=1，随机产生seq=x,客户端进入SYN\_SEND状态，等待服务器的确认。

2.第二次握手:服务器收到SYN报文段。

     服务器收到客户端的SYN报文段，需要对这个SYN报文段进行确认，设置ACK=x+1，同时自己还发送SYN请求SYN=1，序列号seq=y；服务器端将上述所有信息放到一个报文段(即SYN+ACK)中，一并发给客户端，此时服务器进入SYN\_RECV状态。

3.第三次握手:客户端收到服务器的SYN+ACK报文段。

     将序列号seq=y+1，向服务器发送ACK报文段，这个报文段发送完毕后，客户端与服务器端都进入ESTABLISHED状态，完成TCP三次握手。



断开连接：四次挥手

1.第一次挥手

  client设置seq和ACK，向server发送FIN报文段，此时client进入FIN\_WAIT\_1状态，表示主机1没有数据要发送给server了。

2.第二次挥手

server收到了client发送的FIN报文段，向client回一个ACK报文段，ACK=seq+1；client进入FIN\_WAIT\_2状态；server告诉client，我“同意”你的关闭请求。

3.第三次挥手

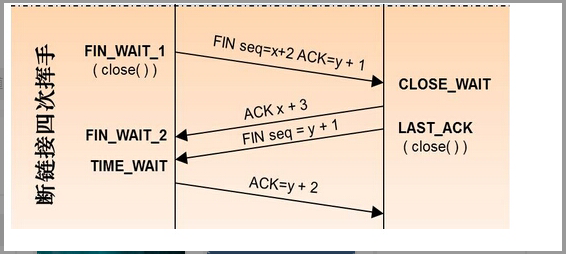
   server向client发送FIN报文段，请求关闭连接，同时server进入CLOSE\_WAIT状态；

4.第四次挥手

 client收到server发送的FIN报文段，向server发送ACK报文段，然后client进入TIME\_WAIT状态；server收到client的ACK报文段以后，就关闭连接；此时，client等待2MSL后依然没有收到回复，则证明Server端已正常关闭，那好，client也可以关闭连接了。

注：FIN是用来扫描保留的端口，发送一个FIN包（或者是任何没有ACK或SYN标记的包）到目标的一个开放的端口，然后等待回应。许多系统会返回一个复位标记。

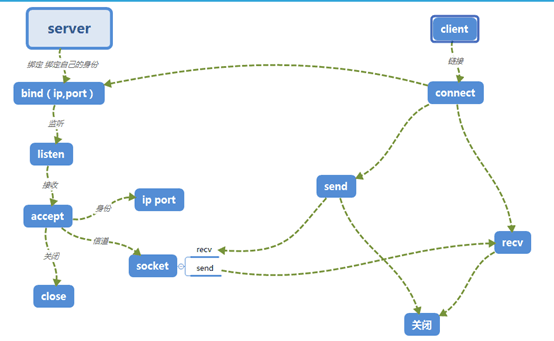
MSL是Maximum Segment Lifetime英文的缩写，中文可以译为“报文最大生存时间”，他是任何报文在网络上存在的最长时间，超过这个时间报文将被丢弃。



UDP:无连接的协议

UDP:协议传输容易丢包，成本低，但是速度快.

1. TCP通信步骤



服务器端先初始化Socket，然后与端口绑定(bind)，对端口进行监听(listen)，调用accept阻塞，等待客户端连接。在这时如果有个客户端初始化一个Socket，然后连接服务器(connect)，如果连接成功，这时客户端与服务器端的连接就建立了。客户端发送数据请求，服务器端接收请求并处理请求，然后把回应数据发送给客户端，客户端读取数据，最后关闭连接，一次交互结束。

被动：当服务器开启，不会主动访问客户端，只是被动等待请求。

阻塞：当有一个用户与服务端发起通信的时候，这个时候信道阻塞，其他用户无法访问。

注：TCP发送数据时，已经建立好TCP连接，所以不需要指定地址，UDP是面向无连接的，所以每次发送要指定是发送给谁。

服务端与客户端之间发送的数据必须是字符串。

2. 通信方式(了解下)

单 工 通信双方只有一条信道，且通信身份不可逆。 BB机

半双工 通信双方只有一条信道，且通信身份可逆。 对讲机

全双工 通信双方只有多条信道，且通信身份可逆。 手机

## 15.3 socket TCP

### 15.3.1 socket tcp协议编程思路

TCP服务端

1、创建套接字

sock = socket.socket(socket.AF\_INET,sokcet.SOCK\_STREAM)

2、绑定套接字到本地IP与端口

sock.bind(())

3、开始监听

sock.listen()

4、接受客户端的连接请求

sock.accept()

5、接收传来的数据，并发送给客户端数据

sock.recv() sock.send()

6、传输完毕，关闭套接字

sock.close()

案例：

#coding=utf-8

#导入套接字  
import socket  
*#创建一个套接字，并且以tcp连接*sock = socket.socket(socket.AF\_INET,socket.SOCK\_STREAM)  
*#绑定套接字，参数为双元素元组，如果写空代表本地所有ip  
#第二个参数是端口 0-65535，就是2^16-1*sock.bind(('',9000))  
*#监听，最大端口为5*sock.listen(5)  
*#接收连接请求  
#conntent 用来接收请求用户的消息和发送对该用户的的消息功能  
#address 是请求用户的身份（ip,port)*content,address = sock.accept()  
print('%s:%s id connected……'%address)  
*#发送数据,encode字符串编码*content.send('hello'.encode())  
*#接收数据参数是字节的形式,decode()解码*print(content.recv(521).decode())  
*#关闭套接字*sock.close()

注：[AF\_INET](https://www.baidu.com/s?wd=AF_INET&tn=SE_PcZhidaonwhc_ngpagmjz&rsv_dl=gh_pc_zhidao" \t "_blank)（又称 PF\_INET）是 IPv4 网络协议的套接字类型

TCP 客户端:

1. 创建套接字

sock = socket.socket(socket.AF\_INET,sokcet.SOCK\_STREAM)

1. 连接服务端的地址和端口

sock.connect(())

1. 连接后发送数据和接受数据

sock.recv() sock.send()

1. 传输完毕，关闭套接字

sock.close()

案例：

import socket  
*#创建一个套接字*sock = socket.socket(socket.AF\_INET,socket.SOCK\_STREAM)  
*#套接字连接*sock.connect(('127.0.0.1',9000))  
*#接收数据*print(sock.recv(521).decode())  
*#发送数据*sock.send('Hi'.encode())  
*#关闭套接字*sock.close()

## 15.4 socket UDP

### 15.4.1 socket udp协议编程思路

为方便理解，假设为UDP服务端

1．创建套接字（SOCK\_DGRAM数据报）

sock = socket.socket(socket.AF\_INET,sokcet.SOCK\_DGRAM)

2．绑定套接字到本地IP与端口

sock.bind(())

3．发送接收数据（需要指定发送给和接收具体的地址端口）

sock.sendto()

sock.recvfrom()

4．传输完毕，关闭套接字

sock.close()

创建server端

案例：

import socket  
*#创建一个个套接字 UDP*sock = socket.socket(socket.AF\_INET,socket.SOCK\_DGRAM)  
*#绑定端口*sock.bind(('',9099))  
*#接收信息，data是客户端发送的信息，address是ip和端口*data,address = sock.recvfrom(521)  
*#发送数据，到指定地址*sock.sendto('server'.encode(),('127.0.0.1',8001))  
*#打印信息和地址*print(data.decode(),address)  
*#关闭套接字*sock.close()

为方便理解，假设为UDP客户端

1、创建套接字

sock = socket.socket(socket.AF\_INET,sokcet.SOCK\_DGRAM)

2、绑定套接字到本地IP与端口

sock.bind(())

3、发送接收数据（需要指定发送给谁）

sock.sendto()

sock.recvfrom()

4、传输完毕，关闭套接字

sock.close()

案例：

import socket  
*#创建一个套接字*sock = socket.socket(socket.AF\_INET,socket.SOCK\_DGRAM)  
*#绑定端口*sock.bind(('127.0.0.1',8001))  
*#发送信息到一个地址*sock.sendto('client'.encode(),('127.0.0.1',9099))  
*#得到服务端发送的信息*data,address = sock.recvfrom(521)  
*#打印响应的信息和地址*print(data.decode(),address)  
sock.close()

### 15.4.2 实战: TCP/UDP 通信

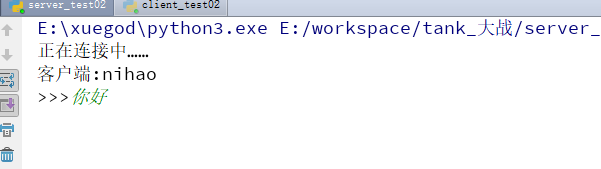
1. TCP 服务端

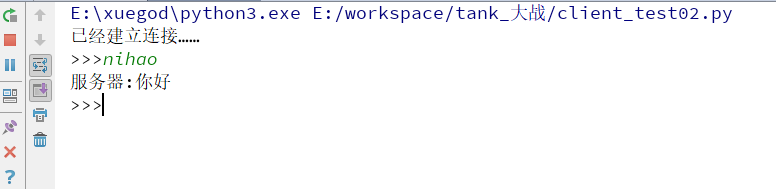
import socket  
print('正在连接中……')  
def dealclient(sock,addr):  
 *#接受客户端发送的信息* info = sock.recv(1024).decode()  
 *#判断进入死循环* while info != 'exit':  
 *#打印客户端的发的信息* print('客户端:'+info)  
 *#输入服务端发送的str* send\_mes = input('>>>')  
 *#服务端发送信息* sock.send(send\_mes.encode())  
 *#如果发送信息不是exit* if send\_mes =='exit':  
 break  
 *#接受信息* info = sock.recv(1024).decode()  
 sock.close()  
if \_\_name\_\_ == '\_\_main\_\_':  
 *#创建一个socket对象* s=socket.socket(socket.AF\_INET, socket.SOCK\_STREAM)  
 *#绑定端口* s.bind(('127.0.0.1', 9001))  
 *#监听端口* s.listen(1)  
 *#阻塞等待客户端连接* sock, addr=s.accept()  
 *#传入函数中，处理逻辑* dealclient(sock,addr)

2. TCP 客户端

import socket  
*#创建一个socket*s= socket.socket(socket.AF\_INET,socket.SOCK\_STREAM)  
*#连接端口*s.connect(('127.0.0.1',9001))  
print('已经建立连接……')  
info = ''  
while info != 'exit':  
 *#获得输入的信息* send\_mes=input('>>>')  
 s.send(send\_mes.encode())  
 if send\_mes =='exit':  
 break  
 *#得到服务器输入的信息* info = s.recv(1024).decode()  
 print('服务器:' + info)  
s.close()

运行结果如下：





3. UDP server

import socket  
s = socket.socket(socket.AF\_INET,socket.SOCK\_DGRAM)  
s.bind(('127.0.0.1',9002))  
print('wait ....')  
while True:  
 data,addr = s.recvfrom(1024)  
 print('客户端：'+data.decode())  
 info = input('>>>')  
 if data.decode() == 'exit':  
 break  
 s.sendto(info.encode(),addr)  
s.close()

4、UDP client

import socket  
s= socket.socket(socket.AF\_INET,socket.SOCK\_DGRAM)  
s.connect(('127.0.0.1',9002))  
print('connection……')  
info = ''  
while info != 'exit':  
 send\_mes=input('>>>')  
 s.sendall(send\_mes.encode())  
 if send\_mes =='exit':  
 break  
 info = s.recv(1024).decode()  
 print('服务端:' + str(info))  
s.close()

TCP和UDP对比：

1.基于连接与无连接；  
2.对系统资源的要求（TCP较多，UDP少）；  
3.UDP程序结构较简单；  
4.流模式与数据报模式 ；

5.TCP保证数据正确性，UDP可能丢包，TCP保证数据顺序，UDP不保证。

## 15.5 SocketServer

### 15.5.1 SocketServer是socket的再封装

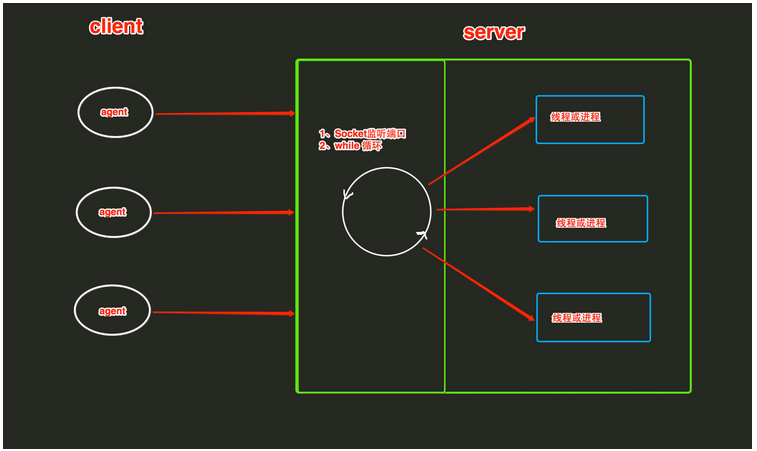
每个客户端连接服务器时，SocketServer会在服务器上创建一线程或者进程，专门负责处理客户端

Socketserver 是在socket的基础上python编写的用于编写web服务的基础框架

SocketServer内部使用 IO多路复用，以及 “多线程” 和 “多进程” ，从而实现并发处理多个客户端请求的Socket服务端。即：每个客户端请求连接到服务器时，Socket服务端都会在服务器是创建一个“线程”或者“进程” 专门负责处理当前客户端的所有请求。

在py2.x版本当中为 SocketServer

在py3.x版本当中为 socketserver



### 15.5.2 SocketServer中类分为三种类型

一是Server类：BaseServer/TCPServer/UDPServer用来接收客户的请求。TCPServer处理TCP请求，UDPServer处理UDP请求。BaserServer是基类，不能直接使用。TCPServer继承自BaseServer，UDPServer继承自TCPServer。暂时不明白为什么UDPServer要继承自TCPServer，后面再说。

二是Handler类：BaseRequestHandler/DatagramRequestHandler/StreamRequestHandler用来处理每一个客户请求。一般用使用BaseRequestHandler就行。

但StreamRequestHandler/DatagramRequestHandler提供了一些特别的功能，前者用来处理流式（TCP）请求，后者处理数据报（UDP）请求。Server每收到一个客户请求就会创建一个Handler

实例来处理该请求。默认情况下，TCPServer/UDPServer是单进程单线程的模型，依次处理每个客户请求，一个请求处理完毕才能接着处理下一个请求。

三是MixIn类：ForkingMixIn/ThreadingMixIn用来为Server提供多进程/多线程并发处理能力的。ForkingMixIn是多进程模型，ThreadingMixin是多线程模型。这里特别巧妙的是，你只要创建一个类，同时继承Server类和MixIn类就能自动获得并发处理请求的能力。该模块本身就直接提供了这种类。

服务端代码：

import socketserver  
*#创建一个类，继承BaseRwequsetHanler*class MYHandle(socketserver.BaseRequestHandler):  
 *#类似于构造函数\_\_init\_\_* def setup(self):  
 print('myhandle is start ')  
 *#用来处理逻辑* def handle(self):  
 *#self.server 当前服务  
 #self.client\_address 客户端的身份（ip.port  
 #self.request 用来接收和发送数据* print(self.server)  
 print('%s:%s is connect '%self.client\_address)  
 recv = self.request.recv(521)  
 print(recv.decode())  
 self.request.send('i am server '.encode())  
 *#类似于析构函数\_\_del\_\_* def finish(self):  
 print('myhandle is stop ')  
if \_\_name\_\_ == '\_\_main\_\_':  
 *#第一个参数来绑定ip  
 #第二个位开启对象* server = socketserver.ThreadingTCPServer(('127.0.0.1',8000),MYHandle)  
 *#开启服务，永远开启* server.serve\_forever()

客户端代码（客户端还是使用socket）

import socket  
*#创建一个sock对象*sock = socket.socket(socket.AF\_INET,socket.SOCK\_STREAM)  
*#连接IP地址*sock.connect(('127.0.0.1',8000))  
*#发送信息*sock.send('client2'.encode())  
*#接收信息*print(sock.recv(521).decode())  
*#关闭套接字*sock.close()

总结：

15.1 socket介绍

15.2 socket编程

15.3 socket tcp

15.4 socket udp

15.5 socketServer