Socket基本原理

内容介绍：

一、Socket 介绍

二、Socket 类型

一、Socket 介绍

（1）Socket 套接字

Python 中提供 socket.py 标准库，非常底层的接口库。

Socket是一种通用的网络编程接口，和网络层次没有——对应的关系。

协议族

AF 表示 AddressFamily，用于 socket（）第一个参数

Socket （计算机专业术语）

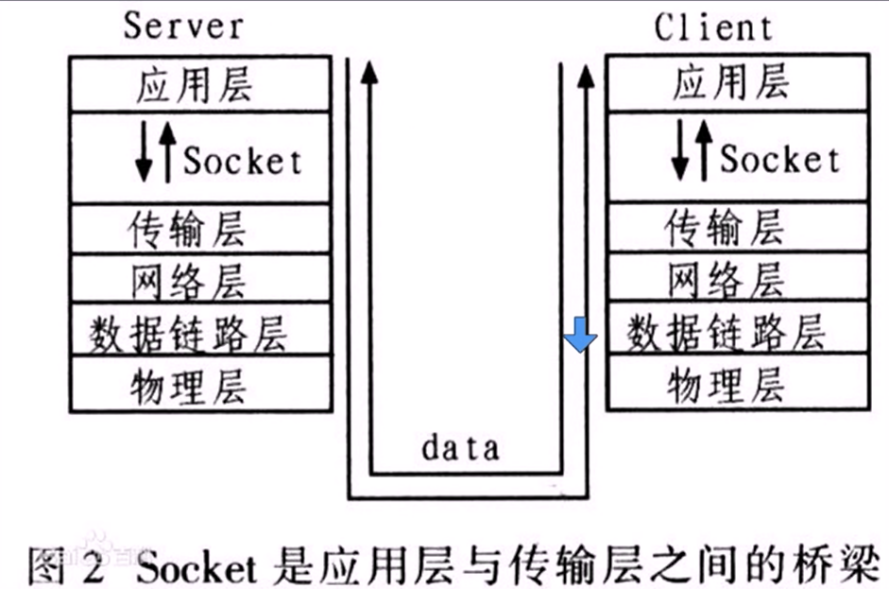
网络上的两个程序通过一个双向的通信连接实现数据的交换，这个连接的一端称为一个 socket。

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 BSDUNIX 的进程通信机制，取后一种意思。通常也称作"套接字"，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。

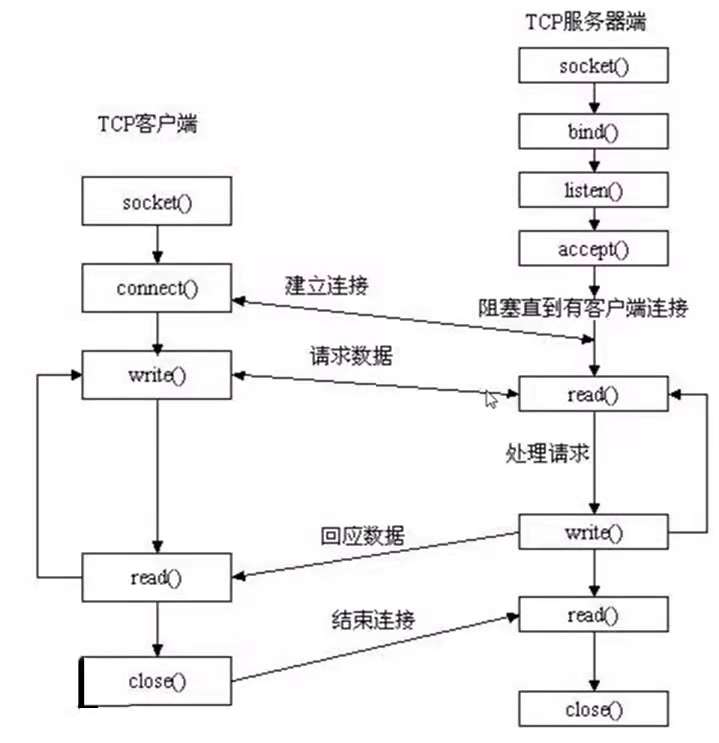
Socket 正如其英文原义那样，像一个多孔插座。一台主机犹如布满各种插座的房间，每个插座有一个编号，有的插座提供220伏交流电，有的提供110伏交流电，有的则提供有线电视节目。客户软件将插头插到不同编号的插座，就可以得到不同的服务。

|  |  |
| --- | --- |
| 名称 | 含义 |
| AF\_INET | IPV4 |
| AF\_INET6 | IPV6 |
| AF\_UNIX | Unix Domain Socket,Windows  没有 |



1. Socket 类型

|  |  |
| --- | --- |
| 名称 | 含义 |
| SOCK\_STREAM | 面向连接的流套接字、默认值，TCP  协议 |
| SOCK\_DGRAM | 无连接的数据报文套接字。UDP  协议 |



TCPServer 简单实现流程

内容介绍：

一、复习与回顾

二、实际操作

一、复习与回顾

（1）Socket套接字

Python 中提供 socket.py 标准库，非常底层的接口库。

Socket 是一种通用的网络编程接口，和网络层次没有——对应的关系。

协议族

AF表示Address Family，用于socket（）第一个参数

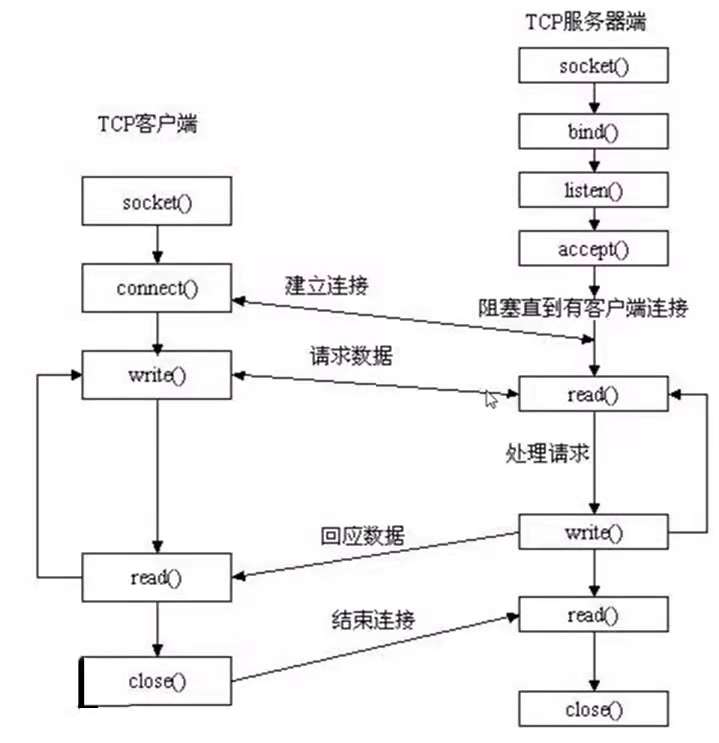
Socket （计算机专业术语）

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 , Socket 的英文原义是"孔"或"插座"。作为 BSD UNIX 的进程通信机制，取后一种意思。通常也称作"套接字"，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。

|  |  |
| --- | --- |
| 名称 | 含义 |
| AF\_INET | IPV4 |
| AF\_INET6 | IPV6 |
| AF\_UNIX | Unix Domain Socket,Windows没有 |

1. Socket 类型

|  |  |
| --- | --- |
| 名称 | 含义 |
| SOCK\_STREAM | 面向连接的流套接字、默认值，TCP协议 |
| SOCK\_DGRAM | 无连接的数据报文套接字。UDP协议 |



二、实际操作

Import socket

server = socket.socket ()

server

<socket.socket fd=12,family=AddressFamily.AF\_INET,type=SocketKind.SOCK\_ STREAM,proto=0,laddr=('0.0.0.0',0)>

server.bind?

Docstring:

bind (address)

Bind the socket to a local address. For IP sockets,the address is a pair (host,port); the host must refer to the local host. For raw packet sockets the address is a tuple (ifname,proto [,pkttype [, hatype]]) Type: builtin\_function\_or\_method

ipaddr =('127.0.0.1',9999)

server.bind(ipaddr)

!ss -tanl

State Recv-Q Send-Q Local Address:Port Peer Address:Port

LISTEN 0 128 :::22 :::\*

LISTEN 0 128 \*:22 \*:\*

LISTEN 0 128 ::1:25 :::\*

LISTEN 0 128 127.0.0.1:25 \*:\*

server.listen()

!ss -tanl

State Recv-Q Send-Q Local Address:Port Peer Address:Port

LISTEN 0 128 127.0.0.1:9999 \*:\*

LISTEN 0 128 :::22 :::\*

LISTEN 0 128 \*:22 \*:\*

LISTEN 0 128 ::1:25 :::\*

LISTEN 0 128 127.0.0.1:25 \*:\*

server.close()

!ss -tanl

State Recv-Q Send-Q Local Address:Port Peer Address:Port

LISTEN 0 128 :::22 :::\*

LISTEN 0 128 \*:22 \*:\*

LISTEN 0 100 ::1:25 :::\*

LISTEN 0 100 127.0.0.1:25 \*:\*

server = socket.socket()

server.bind(ipaddr)

server.bind(('0.0.0.0',9999))

OSError Traceback (most recent call last)

<ipython-input-14-f5b3714d4bfe>in <module>

server bind

[Errno 22] Invalid argument

TCP 群聊服务端实现

内容介绍

一、TCP 编程

二、TCP 服务端

三、练习一

四、其他方法

五、练习二

六、ChatServer 实验用完整代码

一、TCP 编程

Socket 编程，需要两端，一般来说需要一个服务端、一个客户端，服务端称为Server ，客户端称为 Client

二、TCP 服务端

（1）服务器端编程步骤

●创建 Socket 对象

● 绑定IP地址Address和端口Port.bind（）方法IPv4地址为一个二元组（（'IP地 址字符串'， Port）

● 开始监听，将在指定的IP的端口上监听。 listen（） 方法

●获取用于传送数据的 Socket 对象

socket.accept()->(socket object, address info)

accept方法阻塞等待客户端建立连接，返回一个新的 Socket 对象和客户端地址的 二元组地址是远程客户端的地址，IPv4 中它是一个二元组（clientaddr， port）

>接收数据

recv（bufsize【， flags使用缓冲区接收数据

> 发送数据

send（（bytes）发送数据

（2）问题：两次绑定同一个监听端口会怎么样?

import socket

s = socket.socket（）#创建socket对象

s.bind（（'127.0.0.1'，9999））#一个二元组s.listen（）# 开始监听

# 开启一个连接

s1，info= s.accept（）# 阻塞直到和客户端成功建立连接，返回一个socket对象 客 户端地址

# 使用缓冲区获取数据

data = s1.recv(1024)

print(data,info)

s1.send(b'magedu.com ack')

# 开启另外一个连接

s2,\_= s.accept()

data = s2.recv(1024)s2.send(b'hello python')

s.close()

上例 accept 和 recv 是阻塞的，主线程经常被阻塞住而不能工作。怎么办?

三、练习一

写一个群聊程序

需求分析

聊天工具是 CS 程序，C 是每一个客户端，S 是服务器端。

服务器应该具有的功能∶

1.启动服务，包括绑定地址和端口，监听

2.建立连接，能和多个客户端建立连接

3.接收不同用户的信息

4.分发，将接收的某个用户的信息转发到已连接的所有客户端

5.停止服务

6.记录连接的客户端

部分代码如下：

import socket

import threading

import logging

import datetime

FORMAT= %(asctime)s %(threadName)s %(thread)d %(message)s"

logging.basicConfig(format=FORMAT,level=logging.INFO)

# TCP Server

class ChatServer:

def \_init\_(self,ip='127.0.0.1',port=9999):

self.addr =(ip,port)

self.sock = socket.socket ()

self.clients ={}

def start(self):

self.sock.bind(self.addr)

self.sock.listen()# 服务启动了

threading.Thread(target=self.accept,name='accept').start()

def accept(self):

while True∶# 一个线程

s，raddr = self.sock.accept() # 阻塞

logging.info (raddr)logging.info(s)

self.clients[raddr] = s

threading.Thread(target=self.recv,name='recv',args=(s,)).start()

def start(self):

self.sock.bind(self.addr)

self.sock.listen（）# 服务启动了

threading.Thread(target=self.accept,name='accept').start()

def accept (self):

while True:

s，raddr= self.ock.accept（）# 阻塞

threading.Thread(target=self.recv,name='recv',args=(s,)).start()

def recv(self,sock):

while True:

data = sock.recv（1024）# 阻塞，bytes

logging.info (data)

sock.send('ack {}'.format (data.decode()).encode())

# TCP Server

class ChatServer:

def \_init\_(self,ip='127.0.0.1',port=9999):

self.addr =(ip,port)

self.sock = socket.socket ()

def start(self):

self.sock.bind (self.addr)

self.sock.listen() # 服务启动了

threading.Thread(target=self.accept,name='accept').start()

def accept(self):

while True:

s，raddr = self.sock.accept（） # 阻塞

logging.info (raddr)

logging.info(s)

threading.Thread(target=self.recv,name='recv',args=(s,)).star()

def recv (self,sock):

while True:

data = sock.recv（1024）# 阻塞，bytes

logging.info (data)

sock.send('ack{}'.format(data.decode()).encode())

def stop (self):

self.sock.close ()

def stop(self):

for s in self.clients.values():

s.close ()

self.sock.close ()

cs = ChatServer ()

cs.start ()

while True:

cmd = input(">>>")

if cmd.strip()== 'quit':

cs.stop ()

threading.Event.wait(3)

break

logging.info(threading.enumerate())

def accept（self）∶# 多人连接

while not self.event.is\_set():

sock，client = self.sock.accept（）#阻塞

self.clients【client】= sock # 添加到客户端字典

# 准备接收数据，recy是阻塞的，开启新的线程

threading.Thread(target=self.recv,args=(sock, client)).start()

def recv（self，sock∶socket.socket，client）∶#接收客户端数据

while not self.event.is\_set():

data = sock.recv（1024）# 阻塞到数据到来

msg= "{:%Y/%m/%d %H:%M:%S}{}:{}\n{}\n".format(datetime.datetime.now(),\*client,data.decode())

def recv（self，sock∶socket.socket，client）∶#接收客户端数据

while not self.event.is\_set():

data = sock,recv（1024）# 阻塞到数据到来

msg = data.decode().strip()

#客户端退出命令

if msg == 'quit':

Self.clients.pop(client)

sock.close()

logging.info('{} quits'.format(client))

break

msg="{:%Y/%m/%d %H:%M:%S}{}:{}\n{}\n".format(datetime.datetime.now(),\*client,data.decode())

logging.info(msg)

msg = msg.encode()

for s in self.clients.values():

cs = ChatServer()

cs.start()

while True:

cmd =input('>>').strip()

if cmd == 'quit':

cs.stop()

threading.Event().wait(3)

break

这一版基本测试通过。但是还有要完善的地方。

例如各种异常的判断，客户端断开连接后字典中的移除客户端数据等。

客户端主动断开带来的问题程序还有瑕疵，但是业务功能基本完成了

四、其他方法

|  |  |
| --- | --- |
| 名称 | 含义 |
| socket.recv(bufsizel, flags]) | 获取数据。默认是阻塞的方式 |
| socket.recvfrom(bufsize） | 获取数据，返回一个二元组（bytes address）flags) |
| socket.recv\_into(buffer[,nbytes, flags]l) | 获取到  nbytes  的数据后，存储到 buffer  中。如果  nbytes  没有指定或0，将  buffer  大小的数据存入  buffer  中。返回接收的字节数。 |
| socket.recvfrom\_into(buffer[nbytes, flgs) | 获取数据，返回一个二元组（（bytes address）到  buffer中 |
| socket.send(bytes[,flags) | TCP发送数据 |
| socket.sendallbytes[, flagsl) | TCP  发送全部数据，成功返回None |
| s.sendto(stringl,flag],address) | UDP  发送数据 |

socket.makefile(mode='r',buffering=None,\*,encoding=None,errors=None,newline=None)

创建一个与该套接字相关连的文件对象，将 recv 方法看做读方法，将 send 方法看做写方法。

重点代码如下：

# 使用 makefile

import socket

sockserver= socket.socket()

ip = '127.0.0.1'

port = 999

addr =(ip,port)

sockserver.bind(addr)

sockserver.listen()

print('-'\*30)

s,\_\_= sockserver,accept()

print('-'\*30)

f = s.makefile(mode='rw')

line = f.read（10）# 阻塞等

print('-'\*30)

print(1Line)

f.write('Return your msg:{}'.format(line))

def accept(sock:socket.socket,e:threading.Event):

s,\_ = sock.accept()

f = s.makefile(mode='rw')

while True:

line = f.readline()

print(line)

if line.strip（）= "quit"∶#注意要发quit\n

break

f.write('Return your msg: {'.format(line))

f.flush()

f.close()

sock.close()

e.wait(3)

t =threading.Thread(target=accept,args=(sockserver,event))

t.start()

t.join()

print(sockserver)

|  |  |
| --- | --- |
| 名称 | 含义 |
| socket.getpeername() | 返回连接套接字的远程地址。返回值通常是元组(ipadd,port) |
| socket.getsockname() | 返回套接字自己的地址。通常是一个元组(ipaddr,port) |
| socket.setblocking(flag) | 如果flag为0，则将套接字设为非阻塞模式，否则将套接字设为阻塞模式（默认值）非阻塞模式下，如果调用recv（）没有发现任何数据，或send（）调用无法立即发送数据，那么将引起 socket.error 异常 |
| socket.settimeout(value) | 设置套接字操作的超时期， timeout是一个浮点数，单位是秒。值为  None  表示没有超时期。一般，超时期应该在刚创建套接字时设置，因为它们可能用于连接的操作（如connect0） |
| socket.setsockopt(level,optname,value) | 设置套接字选项的值。比如缓冲区大小。太多了，去看文档。不同系统，不同版本都不尽相同 |

五、练习二

使用makefile改写群聊类

import logging

self.addr =(ip,port)self.sock = socket.socket ()self.clients ={}

self.event = threading.Event ()

def start(self):

self.sock.bind(self.addr)

self.sock.listen（）#服务启动了

threading.Thread(target=self.accept,name='accept').start()

def accept (self):

while not self.event.is set （）∶#—个线程

s，raddr = self.sock.accept （） #阻塞logging.info (raddr)logging.info(s)

self.clients[raddr] = s

threading.Thread(target=self.recv,name='recv

args=(3,).start()

def recv（self，sock∶socket.socket）∶#很多线程

while not self.event.is\_set():

tzy:

data = sock.recV（1024）# 阻塞，bytes

logging.info (data)except Exception as e:

logging.error(e)data = b'quit'if data == b'quit':

self.clients.pop(sock.getpeername ())sock.close ()break

msg = "ack{).{}{}".format(sock.getpeername (),

datetime.datetime.now().strftime("sY/sm/sd-8H:8M:S"), data.decode ()).encode ()

For s in self.clients.values():

s.send (msg)

while True:

cmd = input(">>').strip()

if cmd == 'quit":

cs.stop()

threading.Event().wait(3)

break

logging.info（threading.enumerate（））

# 用来观察断开后线程的变化

上例完成了基本功能，但是，如果客户端主动断开，或者 readline 出现异常，就不会从 clients 中移除作废的 socket。

可以使用异常处理解决这个问题。

六、ChatServer 实验用完整代码

注意，这个代码为实验用，代码中瑕疵还有很多。 Socket 太底层了，实际开发中很少使用这么底层的接口。增加一些异常处理

部分代码如下：

def main ():

cs = ChatServer()

cs.start ()

while True:

cmd = input (">>>")

if cmd.strip ()=='quit':

cs.stop ()

threading.Event.wait(3)break

logging.info(threading.enumerate())

if\_name\_=='main 一'