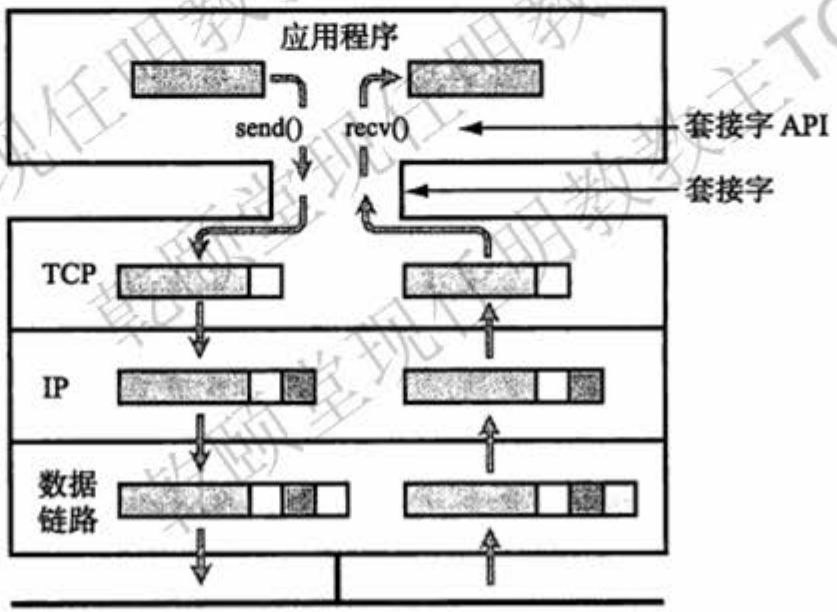
第五部分.4 Socket实现TCP Echo服务器与客户端

第五部分.4:Socket实现TCP Echo服务器与客户端

Socket（套接字）

应用在使用TCP或UDP时，会用到操作系统提供的类库。这种类库一般被称为API（ApplicationProgrammingInteffface，应用编程接口）。

使用或UDP通信时，又会广泛使用到套接字(socket)的API。套接字原本是由BSD UNIX开发的（不仅仅用于网络通信，也可以用于进程间通信），但是后被移植到了Windows的Winsock以及嵌入式操作系统中。应用程序利用套接字，可以设置对端的IP地址、端口号，并实现数据的发送与接收。



第五部分.4:Socket实现TCP Echo服务器与客户端

Socket类型

socket(family,type[,protocal]) 使用给定的地址族、套接字类型、协议编号（默认为0）来创建套接字。

socket类型	描述
socket.AF_UNIX	只能够用于单一的Unix系统进程间通信 进程间通信
socket.AF_INET	服务器之间网络通信 IPv4通信
socket.AF_INET6	IPv6 IPv6通信
socket.SOCK_STREAM	流式socket , for TCP TCP通信
socket.SOCK_DGRAM	数据报式socket , for UDP UDP通信
socket.SOCK_RAW	原始套接字，普通的套接字无法处理ICMP、IGMP等网络报文，而SOCK_RAW可以；其次，SOCK_RAW也可以处理特殊的IPv4报文；此外，利用原始套接字，可以通过IP_HDRINCL套接字选项由用户构造IP头。 原始套接字，三层协议通信
socket.SOCK_SEQPACKET	可靠的连续数据包服务
创建TCP Socket：	s=socket.socket(socket.AF_INET,socket.SOCK_STREAM) 创建TCP套接字
创建UDP Socket：	s=socket.socket(socket.AF_INET,socket.SOCK_DGRAM) 创建UDP套接字

第五部分.4:Socket实现TCP Echo服务器与客户端

Socket函数

注意点:

1）TCP发送数据时，已建立好TCP连接，所以不需要指定地址。UDP是面向无连接的，每次发送要指定是发给谁。

2）服务端与客户端不能直接发送列表，元组，字典。需要字符串化repr(data)。

socket函数	描述
服务端socket函数	
s.bind(address) 绑定地址和端口	将套接字绑定到地址，在AF_INET下,以元组（host,port）的形式表示地址.
s.listen(backlog) 侦听套接字	开始监听TCP传入连接。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() 接受连接，并且返回连接的套接字对象	接受TCP连接并返回（conn,address），其中conn是新的套接字对象，可以用来接收和发送数据。address是连接客户端的地址。
客户端socket函数	
s.connect(address) 客户端发起连接（地址和端口）	连接到address处的套接字。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(adddress)	功能与connect(address)相同，但是成功返回0，失败返回errno的值。

现任明教教主
乾颐堂TCP/IP 2017

第五部分.4:Socket实现TCP Echo服务器与客户端

公共socket函数

socket.send is a low-level method and basically just the C/syscall method send(3) / send(2). It can send less bytes than you requested, but returns the number of bytes sent.

socket.sendall is a high-level Python-only method that sends the entire buffer you pass or throws an exception. It does that by calling socket.send until everything has been sent or an error occurs.

If you're using TCP with blocking sockets and don't want to be bothered by internals (this is the case for most simple network applications), use sendall.

公共socket函数	
s.recv(bufsize[,flag]) 接受TCP数据	接受TCP套接字的数据。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(string[,flag])	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(string[,flag])	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(bufsize[.flag]) 接受UDP数据	接受UDP套接字的数据。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(string[,flag],address) 发送UDP数据	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	关闭套接字。
s.getpeername()	返回连接套接字的远程地址。返回值通常是元组（ipaddr,port）。
s.getsockname()	返回套接字自己的地址。通常是一个元组(ipaddr,port)
s.setsockopt(level,optname,value)	设置给定套接字选项的值。
s.getsockopt(level,optname[.buflen])	返回套接字选项的值。
s.settimeout(timeout)	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connect()）
s.gettimeout()	返回当前超时期的值，单位是秒，如果没有设置超时期，则返回None。
s.fileno()	返回套接字的文件描述符。
s.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。
s.makefile()	创建一个与该套接字相关连的文件

第五部分.4:Socket实现TCP Echo服务器与客户端

最简单Socket TCP Server

CentOS_1

[root@localhost 5.TCP]# ./tcp_socket_server.py

```
from socket import *
#配置本地服务器IP地址
myHost = '172.16.12.101'
#配置本地服务器端口号
myPort = 6666
import sys
import random
import time

#创建TCP Socket，AF_INET为IPv4，SOCK_STREAM为TCP
sockobj = socket(AF_INET, SOCK_STREAM)
#绑定套接字到地址，地址为（host，port）的元组
sockobj.bind((myHost, myPort))
#在拒绝连接前，操作系统可以挂起的最大连接数量，一般配置为5
sockobj.listen(5)

try:
    while True:#一直接受请求，直到ctl+c终止程序
        print('开始接受新的TCP连接!!!')
        #接受TCP连接，并且返回（conn,address）的元组，conn为新的套接字对象，可以用来接收和发送数据，address是连接客户端的地址
        connection, address = sockobj.accept()
        #打印连接客户端的IP地址
        print('Server Connected by', address)

        while True:
            data = connection.recv(1024)#接收数据，1024为bufsize，表示一次接收的最大数据量
            if not data:#如果没有数据就结束TCP会话
                print('TCP连接结束!!!')
                break
            print('收到数据'+str(len(data))+'个字节！')#打印数据长度
            print(data)#打印数据内容
            time.sleep(random.random())#等待一个随机时间
except KeyboardInterrupt:#如果出现Crtl+C,就停止服务器
    print ("Crtl+C Pressed. Shutting down.")
    connection.close()#关闭连接
    sys.exit()
finally:
    connection.close()#关闭连接
    sys.exit()
```

1 创建Socket，并绑定地址

2 接受TCP连接请求

3 一次能从Socket读取的最大数据量（剩余部分下一次继续读取）

4 随机等待一个时间来读取数据

第五部分.4:Socket实现TCP Echo服务器与客户端

最简单Socket TCP Client

CentOS_2
[root@localhostTCP]# ./tcp_socket_client.py

```
import time
import random
from socket import *
#连接的服务器地址

serverHost = '172.16.12.101'
#连接的服务器端口号
serverPort = 6666
#发送的回显信息

sockobj = socket(AF_INET, SOCK_STREAM)#创建TCP Socket, AF_INET为IPv4, SOCK_STREAM为TCP
sockobj.connect((serverHost, serverPort))#连接到套接字地址, 地址为（host, port）的元组

send_data=[
b'welcome to qytang',
b'1',
b'yyyyyyyyyy'*30,
b'2',
b'ccccccccc'*20,
b'3',
b'v'*2000
]

for data in send_data:
    time.sleep(random.random())
    sockobj.send(data)#发送数据

time.sleep(0.1)
sockobj.close()#关闭连接
```

1 创建Socket，并建立连接

2 发送的数据

3 等待一个随机时间来发送数据

4 向Socket发送数据

数据结果一

六次读取数据分别为

1:17 2:302 3:200 4:1 5:1024 6:976

```
收到数据17个字节！①
b'welcome to qytang'
收到数据302个字节！②
b'1yyyyyyyyyyyyyyyyyyyyyyyyyyyyyyyyyyyyyyyyyyyyyyyyyyyyyyyyyyyyyyyyyyyyyyyyyyyyyyyyyyyyyyyyyyyyyyyyyyyyyyyyyy
yyyyyyyyyyyyyyyyyyyyyyyyyyyyyyyyyyyyyyyyyyyyyyyyyyyyyyyyyyyyyyyyyyyyyyyyyyyyyyyyyyyyyyyyyyyyyyyyyyyyyyyyyyyyy
yyyyyyyyyyyyyyyyyyyyyyyyyyyyyyyyyyyyyyyyyyyyyyyyyyyyyyyyyyyyyyyyyyyyyyyyyyyyyyyyyyy2'
收到数据200个字节！③
b'cccccccccccccccccccccccccccccccccccccccccccccccccccccccccccccccccccccccccccccccccccccccccccccccccccccccccccc
cccccccccccccccccccccccccccccccccccccccccccccccccccccccccccccccccccccccccccccccccccccccccccc'
收到数据1个字节！④
b'3'
收到数据1024个字节！⑤
b'vvvvvvvvvvvvvvvvvvvvvvvvvvvvvvvvvvvvvvvvvvvvvvvvvvvvvvvvvvvvvvvvvvvvvvvvvvvvvvvvvvvvvvvvvvvvvvvvvvvvvvvvvvvv
vvvvvvvvvvvvvvvvvvvvvvvvvvvvvvvvvvvvvvvvvvvvvvvvvvvvvvvvvvvvvvvvvvvvvvvvvvvvvvvvvvvvvvvvvvvvvvvvvvvvvvvvvvvvvv
vvvvvvvvvvvvvvvvvvvvvvvvvvvvvvvvvvvvvvvvvvvvvvvvvvvvvvvvvvvvvvvvvvvvvvvvvvvvvvvvvvvvvvvvvvvvvvvvvvvvvvvvvvvvvv
vvvvvvvvvvvvvvvvvvvvvvvvvvvvvvvvvvvvvvvvvvvvvvvvvvvvvvvvvvvvvvvvvvvvvvvvvvvvvvvvvvvvvvvvvvvvvvvvvvvvvvvvvvvvvv
vvvvvvvvvvvvvvvvvvvvvvvvvvvvvvvvvvvvvvvvvvvvvvvvvvvvvvvvvvvvvvvvvvvvvvvvvvvvvvvvvvvvvvvvvvvvvvvvvvvvvvvvvvvvvv
vvvvvvvvvvvvvvvvvvvvvvvvvvvvvvvvvvvvvvvvvvvvvvvvvvvvvvvvvvvvvvvvvvvvvvvvvvvvvvvvvvvvvvvvvvvvvvvvvvvvvvvvvvvvvv
vvvvvvvvvvvvvvvvvvvvvvvvvvvvvvvvvvvvvvvvvvvvvvvvvvvvvvvvvvvvvvvvvvvvvvvvvvvvvvvvvvvvvvvvvvvvvvvvvvvvvvvvvvvvvv
vvvvvvvvvvvvvvvvvvvvvvvvvvvvvvvvvvvvvvvvvvvvvvvvvvvvvvvvvvvvvvvvvvvvvvvvvvvvvvvvvvvvvvvvvvvvvvvvvvvvvvvvvvvvvv
vvvvvvvvvvvvvvvvvvvvvvvvvvvvvvvvvvvvvvvvvvvvvvvvvvvvvvvvvvvvvvvvvvvvvvvvvvvvvvvvvvvvvvvvvvvvvvvvvvvvvvvvvvvvvv
vvvvvvvvvvvvvvvvvvvvvvvvvvvvvvvvvvvv'
收到数据976个字节！⑥
b'vvvvvvvvvvvvvvvvvvvvvvvvvvvvvvvvvvvvvvvvvvvvvvvvvvvvvvvvvvvvvvvvvvvvvvvvvvvvvvvvvvvvvvvvvvvvvvvvvvvvvvvvvvvv
vvvvvvvvvvvvvvvvvvvvvvvvvvvvvvvvvvvvvvvvvvvvvvvvvvvvvvvvvvvvvvvvvvvvvvvvvvvvvvvvvvvvvvvvvvvvvvvvvvvvvvvvvvvvvv
vvvvvvvvvvvvvvvvvvvvvvvvvvvvvvvvvvvvvvvvvvvvvvvvvvvvvvvvvvvvvvvvvvvvvvvvvvvvvvvvvvvvvvvvvvvvvvvvvvvvvvvvvvvvvv
vvvvvvvvvvvvvvvvvvvvvvvvvvvvvvvvvvvvvvvvvvvvvvvvvvvvvvvvvvvvvvvvvvvvvvvvvvvvvvvvvvvvvvvvvvvvvvvvvvvvvvvvvvvvvv
vvvvvvvvvvvvvvvvvvvvvvvvvvvvvvvvvvvvvvvvvvvvvvvvvvvvvvvvvvvvvvvvvvvvvvvvvvvvvvvvvvvvvvvvvvvvvvvvvvvvvvvvvvvvvv
vvvvvvvvvvvvvvvvvvvvvvvvvvvvvvvvvvvvvvvvvvvvvvvvvvvvvvvvvvvvvvvvvvvvvvvvvvvvvvvvvvvvvvvvvvvvvvvvvvvvvvvvvvvvvv
vvvvvvvvvvvvvvvvvvvvvvvvvvvvvvvvvvvvvvvvvvvvvvvvvvvvvvvvvvvvvvvvvvvvvvvvvvvvvvvvvvvvvvvvvvvvvvvvvvvvvvvvvvvvvv
vvvvvvvvvvvvvvvvvvvvvvvvvvvvvvvvvvvvvvvvvvvvvvvvvvvvvvvvvvvvvvvvvvvvvvvvvvvvvvvvvvvvvvvvvvvvvvvvvv'
```

第五部分.4:Socket实现TCP Echo服务器与客户端

数据结果二

七次读取数据分别为

1:17 2:1 3:300 4:201 5:1 6:1024 7:976

```
收到数据17个字节!
b'welcome to qytang'
收到数据1个字节!
b'1'
收到数据300个字节!
b'yyyyyyyyyyyyyyyyyyyyyyyyyyyyyyyyyyyyyyyyyyyyyyyyyyyyyyyyyyyyyyyyyyyyyyyyyyyyyyyyyyyyyyyyyyyyyyyyyyyyyyyyyyyy
yyyyyyyyyyyyyyyyyyyyyyyyyyyyyyyyyyyyyyyyyyyyyyyyyyyyyyyyyyyyyyyyyyyyyyyyyyyyyyyyyyyyyyyyyyyyyyyyyyyyyyyyyyyyyy
yyyyyyyyyyyyyyyyyyyyyyyyyyyyyyyyyyyyyyyyyyyyyyyyyyyyyyyyyyyyyyyyyyyyyyyyyyyyyyyyyy'
收到数据201个字节!
b'2ccccccccccccccccccccccccccccccccccccccccccccccccccccccccccccccccccccccccccccccccccccccccccccccccccccccccccc
ccccccccccccccccccccccccccccccccccccccccccccccccccccccccccccccccccccccccccccccccccccccccccccc'
收到数据1个字节!
b'3'
收到数据1024个字节!
b'vvvvvvvvvvvvvvvvvvvvvvvvvvvvvvvvvvvvvvvvvvvvvvvvvvvvvvvvvvvvvvvvvvvvvvvvvvvvvvvvvvvvvvvvvvvvvvvvv
vvvvvvvvvvvvvvvvvvvvvvvvvvvvvvvvvvvvvvvvvvvvvvvvvvvvvvvvvvvvvvvvvvvvvvvvvvvvvvvvvvvvvvvvvvvvvvvvvvv
vvvvvvvvvvvvvvvvvvvvvvvvvvvvvvvvvvvvvvvvvvvvvvvvvvvvvvvvvvvvvvvvvvvvvvvvvvvvvvvvvvvvvvvvvvvvvvvvvvv
vvvvvvvvvvvvvvvvvvvvvvvvvvvvvvvvvvvvvvvvvvvvvvvvvvvvvvvvvvvvvvvvvvvvvvvvvvvvvvvvvvvvvvvvvvvvvvvvvvv
vvvvvvvvvvvvvvvvvvvvvvvvvvvvvvvvvvvvvvvvvvvvvvvvvvvvvvvvvvvvvvvvvvvvvvvvvvvvvvvvvvvvvvvvvvvvvvvvvvv
vvvvvvvvvvvvvvvvvvvvvvvvvvvvvvvvvvvvvvvvvvvvvvvvvvvvvvvvvvvvvvvvvvvvvvvvvvvvvvvvvvvvvvvvvvvvvvvvvvv
vvvvvvvvvvvvvvvvvvvvvvvvvvvvvvvvvvvvvvvvvvvvvvvvvvvvvvvvvvvvvvvvvvvvvvvvvvvvvvvvvvvvvvvvvvvvvvvvvvv
vvvvvvvvvvvvvvvvvvvvvvvvvvvvvvvvvvvvvvvvvvvvvvvvvvvvvvvvvvvvvvvvvvvvvvvvvvvvvvvvvvvvvvvvvvvvvvvvvvv
vvvvvvvvvvvvvvvvvvvvvvvvvvvvvvvvvvvvvvvvvvvvvvvvvvvvvvvvvvvvvvvvvvvvvvvvvvvvvvvvvvvvvvvvvvvvvvvvvvv
vvvvvvvvvvvvvvvvvvvvvvvvvvvvvvvvvvvvvvvvvvvvvvvvvvvvvvvvvvvvvvvvvvvvvvvvvvvvvvvvvvvvvvvvvvvvvvvvvvv
vvvvvvvvvvvvvvvvvvvvvvvvvvvvvvvvvvvv'
收到数据976个字节!
b'vvvvvvvvvvvvvvvvvvvvvvvvvvvvvvvvvvvvvvvvvvvvvvvvvvvvvvvvvvvvvvvvvvvvvvvvvvvvvvvvvvvvvvvvvvvvvvvvv
vvvvvvvvvvvvvvvvvvvvvvvvvvvvvvvvvvvvvvvvvvvvvvvvvvvvvvvvvvvvvvvvvvvvvvvvvvvvvvvvvvvvvvvvvvvvvvvvvvv
vvvvvvvvvvvvvvvvvvvvvvvvvvvvvvvvvvvvvvvvvvvvvvvvvvvvvvvvvvvvvvvvvvvvvvvvvvvvvvvvvvvvvvvvvvvvvvvvvvv
vvvvvvvvvvvvvvvvvvvvvvvvvvvvvvvvvvvvvvvvvvvvvvvvvvvvvvvvvvvvvvvvvvvvvvvvvvvvvvvvvvvvvvvvvvvvvvvvvvv
vvvvvvvvvvvvvvvvvvvvvvvvvvvvvvvvvvvvvvvvvvvvvvvvvvvvvvvvvvvvvvvvvvvvvvvvvvvvvvvvvvvvvvvvvvvvvvvvvvv
vvvvvvvvvvvvvvvvvvvvvvvvvvvvvvvvvvvvvvvvvvvvvvvvvvvvvvvvvvvvvvvvvvvvvvvvvvvvvvvvvvvvvvvvvvvvvvvvvvv
```

第五部分.4:Socket实现TCP Echo服务器与客户端

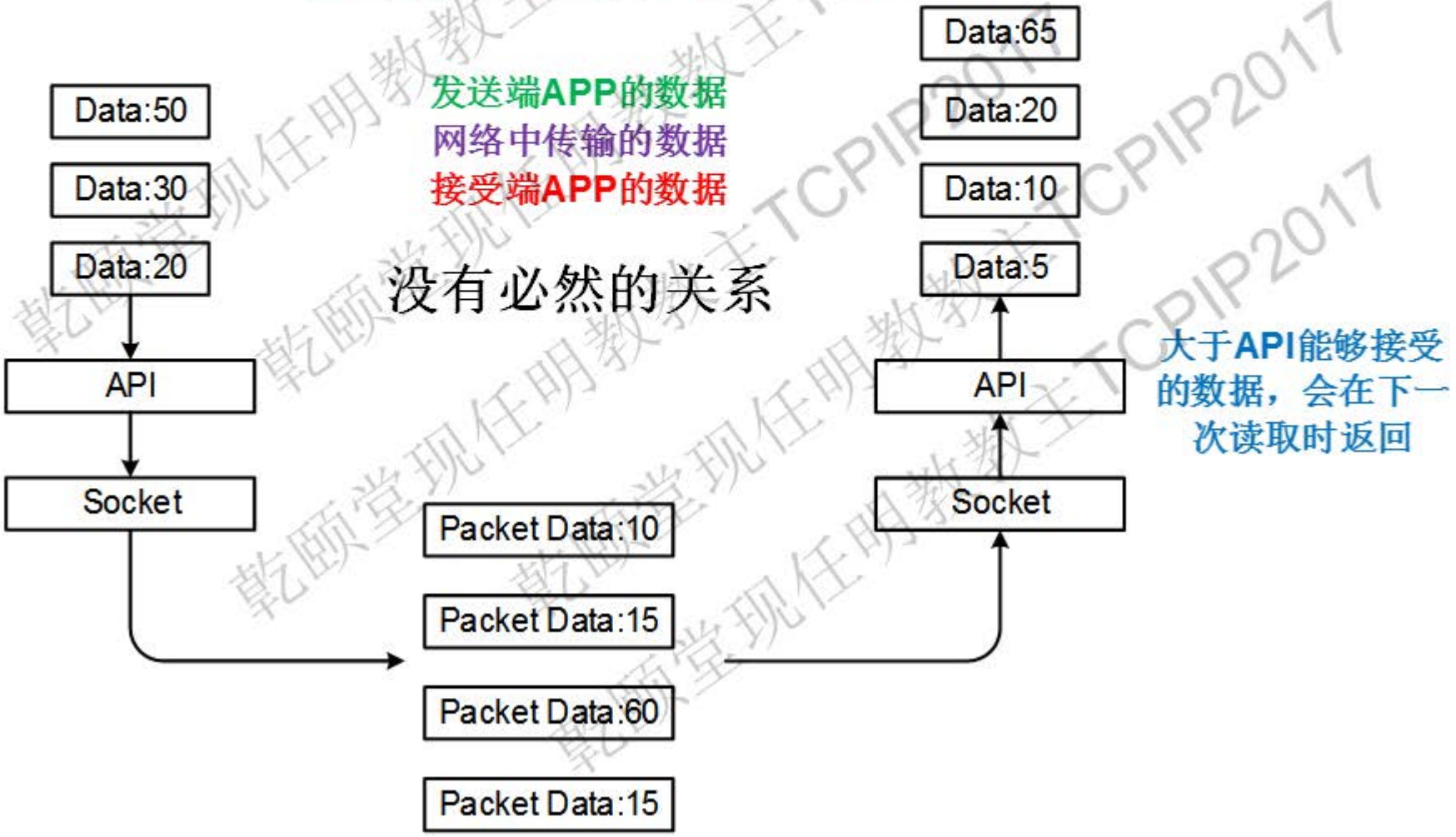
网络上的包基本是一样的

1	172.16.12.102	172.16.12.101	78	0.000000	TCP	Src=43344,Dst= 6666,....S.,S=3235...
2	172.16.12.101	172.16.12.102	78	0.000207	TCP	Src= 6666,Dst=43344,.A..S.,S=1895...
3	172.16.12.102	172.16.12.101	70	0.000224	TCP	Src=43344,Dst= 6666,.A....,S=3235...
4	172.16.12.101	172.16.12.102	70	0.000498	TCP	Src= 6666,Dst=43342,.A...F,S=1727...
5	172.16.12.102	172.16.12.101	70	0.000550	TCP	Src=43342,Dst= 6666,.A....,S=2537...
6	172.16.12.102	172.16.12.101	87	0.522342	TCP	Src=43344,Dst= 6666,.AP...,S=3235...
7	172.16.12.101	172.16.12.102	70	0.524429	TCP	Src= 6666,Dst=43344,.A....,S=1895...
8	172.16.12.102	172.16.12.101	71	0.585849	TCP	Src=43344,Dst= 6666,.AP...,S=3235...
9	172.16.12.101	172.16.12.102	70	0.585944	TCP	Src= 6666,Dst=43344,.A....,S=1895...
10	172.16.12.102	172.16.12.101	370	1.086606	TCP	Src=43344,Dst= 6666,.AP...,S=3235...
11	172.16.12.101	172.16.12.102	70	1.086690	TCP	Src= 6666,Dst=43344,.A....,S=1895...
12	172.16.12.102	172.16.12.101	71	1.309789	TCP	Src=43344,Dst= 6666,.AP...,S=3235...
13	172.16.12.101	172.16.12.102	70	1.309916	TCP	Src= 6666,Dst=43344,.A....,S=1895...
14	172.16.12.102	172.16.12.101	270	1.811190	TCP	Src=43344,Dst= 6666,.AP...,S=3235...
15	172.16.12.101	172.16.12.102	70	1.811300	TCP	Src= 6666,Dst=43344,.A....,S=1895...
16	172.16.12.102	172.16.12.101	71	1.941559	TCP	Src=43344,Dst= 6666,.AP...,S=3235...
17	172.16.12.101	172.16.12.102	70	1.941677	TCP	Src= 6666,Dst=43344,.A....,S=1895...
18	172.16.12.102	172.16.12.101	1518	2.545498	TCP	Src=43344,Dst= 6666,.A....,S=3235...
19	172.16.12.102	172.16.12.101	622	2.545510	TCP	Src=43344,Dst= 6666,.AP...,S=3235...
20	172.16.12.101	172.16.12.102	70	2.545643	TCP	Src= 6666,Dst=43344,.A....,S=1895...
21	172.16.12.102	172.16.12.101	70	2.645845	TCP	Src=43344,Dst= 6666,.A...F,S=3235...
22	172.16.12.101	172.16.12.102	70	2.685243	TCP	Src= 6666,Dst=43344,.A....,S=1895...

动态结果:

其实网络上传输的数据包在系统或者网络繁忙时也会出现不一样的结果

数据流协议（TCP）



UDP Echo Server

```
import socket
import sys
import random
import time

address = ('172.16.12.101', 6666)
s = socket.socket(socket.AF_INET, socket.SOCK_DGRAM)
s.bind(address)

try:
    while True:
        data, addr = s.recvfrom(1024)
        print('收到数据'+str(len(data))+'个字节！')
        print(data)
        time.sleep(random.random())#等待一个随机时间
except KeyboardInterrupt:
    s.close()
    sys.exit()
```

CentOS_1
[root@localhost 5.TCP]# ./udp_socket_server.py

1 创建Socket，并绑定地址

2 一次能从Socket读取的最大数据量（剩余部分截断丢弃）

3 随机等待一个时间来读取数据

UDP Echo Client

```
import socket
import time
import random

address = ('172.16.12.101', 6666)
s = socket.socket(socket.AF_INET, socket.SOCK_DGRAM)

send_data=[
b'welcome to qytang',
b'1',
b'yyyyyyyyyy'*30,
b'2',
b'ccccccccc'*20,
b'3',
b'v'*2000
]

for data in send_data:
    time.sleep(random.random())
    s.sendto(data,address)#发送数据

time.sleep(0.1)
s.close()
```

CentOS_2
[root@localhostTCP]# ./udp_socket_client.py

1 创建Socket

2 发送的数据

3 等待一个随机时间来发送数据

4 向Socket发送数据

数据读取结果

```
收到数据17个字节!
b'welcome to qytang'
收到数据1个字节!
b'1'
收到数据300个字节!
b'yyyyyyyyyyyyyyyyyyyyyyyyyyyyyyyyyyyyyyyyyyyyyyyyyyyyyyyyyyyyyyyyyyyyyyyyyyyyyyyyyyyyyyyyyyyyyyyyyyyyyyyy
yyyyyyyyyyyyyyyyyyyyyyyyyyyyyyyyyyyyyyyyyyyyyyyyyyyyyyyyyyyyyyyyyyyyyyyyyyyyyyyyyyyyyyyyyyyyyyyyyyyyyyyyyy
yyyyyyyyyyyyyyyyyyyyyyyyyyyyyyyyyyyyyyyyyyyyyyyyyyyyyyyyyyyyyyyyyyyyyyyyyyyyyyyyyy'
收到数据1个字节!
b'2'
收到数据200个字节!
b'cccccccccccccccccccccccccccccccccccccccccccccccccccccccccccccccccccccccccccccccccccccccccccccccccccccccccccc
cccccccccccccccccccccccccccccccccccccccccccccccccccccccccccccccccccccccccccccccccccccccccccc'
收到数据1个字节!
b'3'
收到数据1024个字节!
b'vvvvvvvvvvvvvvvvvvvvvvvvvvvvvvvvvvvvvvvvvvvvvvvvvvvvvvvvvvvvvvvvvvvvvvvvvvvvvvvvvvvvvvvvvvvvvvvvvvvvvvvv
vvvvvvvvvvvvvvvvvvvvvvvvvvvvvvvvvvvvvvvvvvvvvvvvvvvvvvvvvvvvvvvvvvvvvvvvvvvvvvvvvvvvvvvvvvvvvvvvvvvvvvvvvv
vvvvvvvvvvvvvvvvvvvvvvvvvvvvvvvvvvvvvvvvvvvvvvvvvvvvvvvvvvvvvvvvvvvvvvvvvvvvvvvvvvvvvvvvvvvvvvvvvvvvvvvvvv
vvvvvvvvvvvvvvvvvvvvvvvvvvvvvvvvvvvvvvvvvvvvvvvvvvvvvvvvvvvvvvvvvvvvvvvvvvvvvvvvvvvvvvvvvvvvvvvvvvvvvvvvvv
vvvvvvvvvvvvvvvvvvvvvvvvvvvvvvvvvvvvvvvvvvvvvvvvvvvvvvvvvvvvvvvvvvvvvvvvvvvvvvvvvvvvvvvvvvvvvvvvvvvvvvvvvv
vvvvvvvvvvvvvvvvvvvvvvvvvvvvvvvvvvvvvvvvvvvvvvvvvvvvvvvvvvvvvvvvvvvvvvvvvvvvvvvvvvvvvvvvvvvvvvvvvvvvvvvvvv
vvvvvvvvvvvvvvvvvvvvvvvvvvvvvvvvvvvvvvvvvvvvvvvvvvvvvvvvvvvvvvvvvvvvvvvvvvvvvvvvvvvvvvvvvvvvvvvvvvvvvvvvvv
vvvvvvvvvvvvvvvvvvvvvvvvvvvvvvvvvvvvvvvvvvvvvvvvvvvvvvvvvvvvvvvvvvvvvvvvvvvvvvvvvvvvvvvvvvvvvvvvvvvvvvvvvv
vvvvvvvvvvvvvvvvvvvvvvvvvvvvvvvvvvvvvvvvvvvvvvvvvvvvvvvvvvvvvvvvvvvvvvvvvvvvvvvvvvvvvvvvvvvvvvvvvvvvvvvvvv
vvvvvvvvvvvvvvvvvvvvvvvvvvvvvvvvvvvv'
```

数据读取的结果，与对方发送的数据是完全匹配的

虽然数据抵达了目的，但是API使用s.recvfrom(1024)对数据进行了截断，所以只是显示1024字节的数据

第五部分.4:Socket实现TCP Echo服务器与客户端

网络上的包是一样的

7	172.16.12.102	172.16.12.101	64	0.000000	UDP	Src=48218,Dst= 6666 ,L= 17
10	172.16.12.102	172.16.12.101	64	0.737424	UDP	Src=48218,Dst= 6666 ,L= 1
13	172.16.12.102	172.16.12.101	346	0.908978	UDP	Src=48218,Dst= 6666 ,L= 300
14	172.16.12.102	172.16.12.101	64	1.022627	UDP	Src=48218,Dst= 6666 ,L= 1
17	172.16.12.102	172.16.12.101	246	1.895277	UDP	Src=48218,Dst= 6666 ,L= 200
20	172.16.12.102	172.16.12.101	64	2.140295	UDP	Src=48218,Dst= 6666 ,L= 1
23	172.16.12.102	172.16.12.101	1518	3.018693	UDP	Src=48218,Dst= 6666 ,L= 2000
24	172.16.12.102	172.16.12.101	566	3.018700	IP Fragment	

静态结果:

1.其实网络上传输的数据包与发送的数据完全匹配

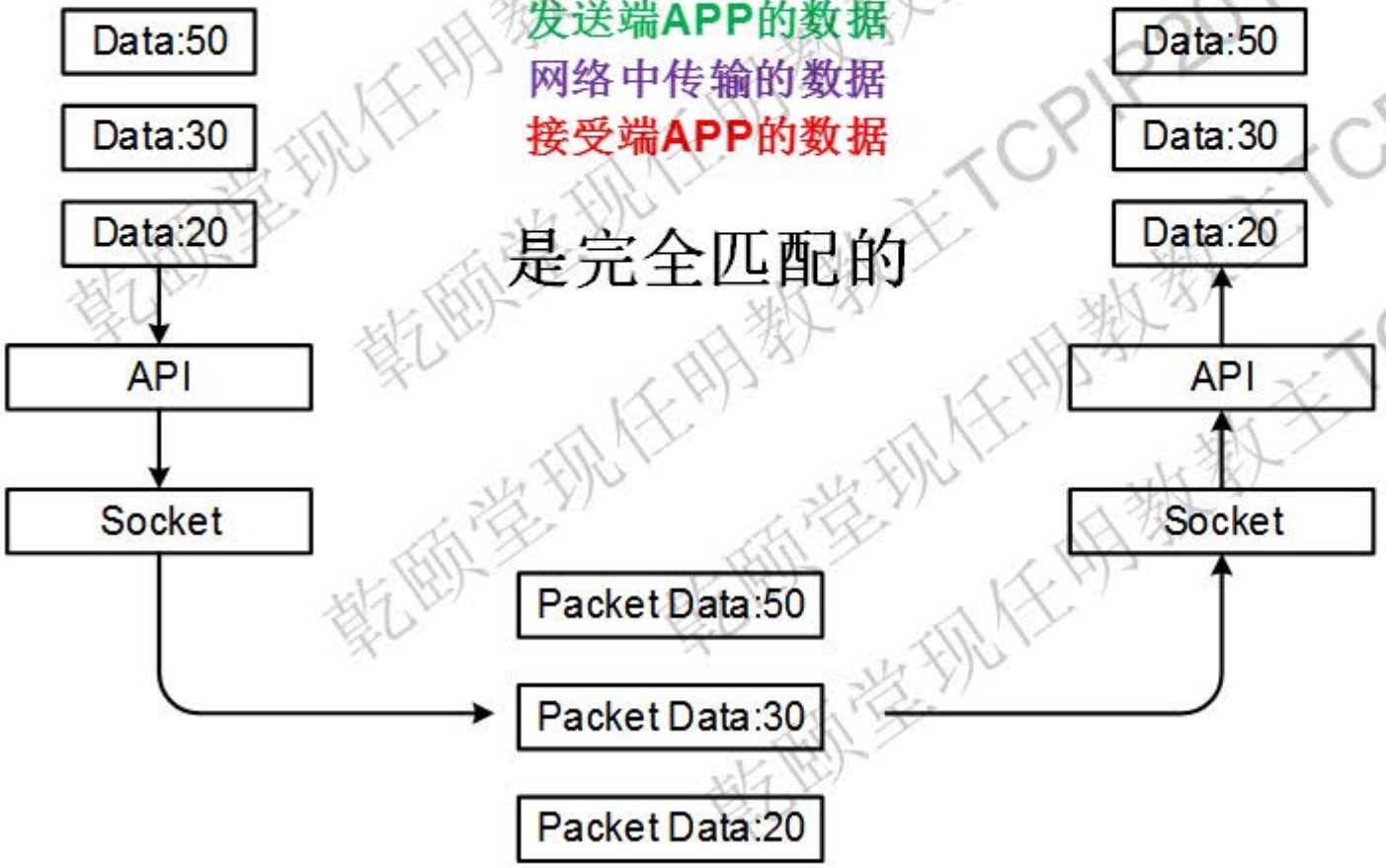
2.大于MTU会出现分片

第五部分.4:Socket实现TCP Echo服务器与客户端

数据报协议（UDP）

发送端APP的数据
网络中传输的数据
接受端APP的数据

是完全匹配的



大于API能够接受的数据，会被截断并丢弃