1. UDP

UDP 是 User Datagram Protocol 的简称， 中文名是用户数据报协议，是一个简单的面向数据报的运输层协议，在网络中用于处理数据包，是一种无连接的协议。

UDP 不提供可靠性的传输，它只是把应用程序传给 IP 层的数据报发送出去，但是并不能保证它们能到达目的地。由于 UDP 在传输数据报前不用在客户和服务器之间建立一个连接，且没有超时重发等机制，故而传输速度很快。

1.1 UDP特点:

UDP是面向无连接的通讯协议，UDP数据包括目的端口号和源端口号信息，由于通讯不需要连接，所以可以实现广播发送。 UDP传输数据时有大小限制，每个被传输的数据报必须限定在64KB之内。 UDP是一个不可靠的协议，发送方所发送的数据报并不一定以相同的次序到达接收方。

【适用情况】

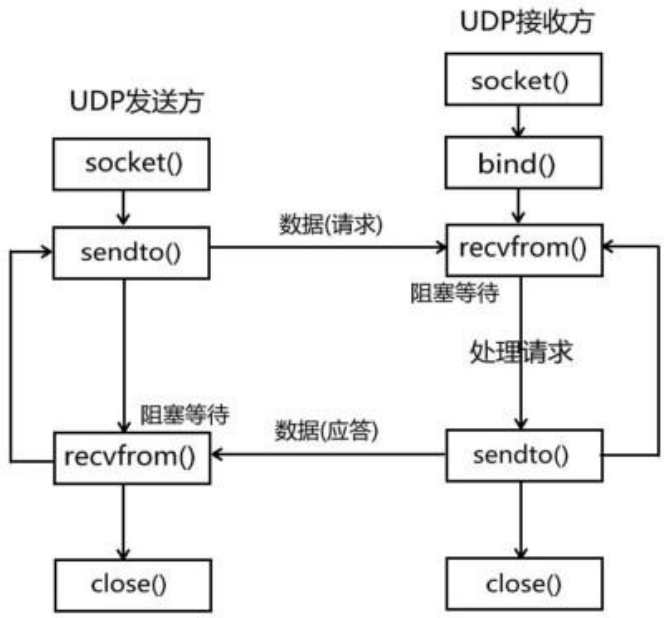
UDP是面向消息的协议，通信时不需要建立连接，数据的传输自然是不可靠的，UDP一般用于多点通信和实时的数据业务，比如

- 语音广播
- 视频
- QQ
- TFTP(简单文件传送)　SNMP（简单网络管理协议）
- RIP（路由信息协议，如报告股票市场，航空信息）
- DNS(域名解释)

注重速度流畅

UDP操作简单，而且仅需要较少的监护，因此通常用于局域网高可靠性的分散系统中client/server应用程序。例如视频会议系统，并不要求音频视频数据绝对的正确，只要保证连贯性就可以了，这种情况下显然使用UDP会更合理一些。

2. UDP通信模型



3. UDP编程-发送数据

3.1 发送流程

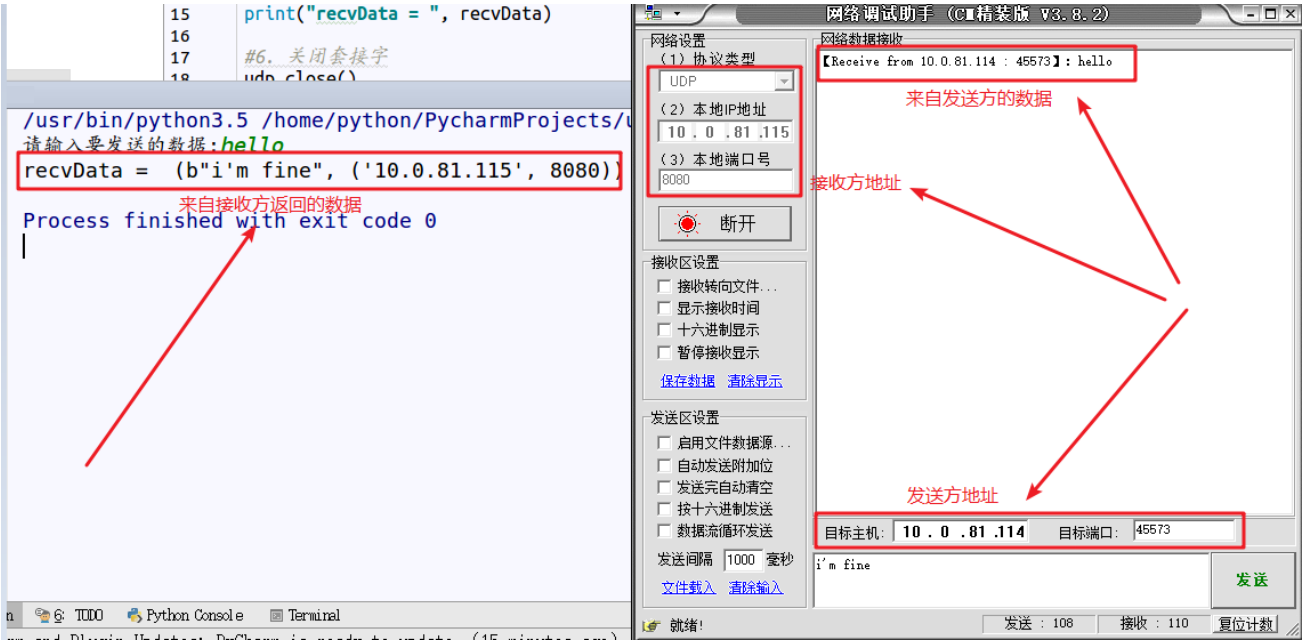
对于 UDP 发送方流程， 有点类似于写信过程：

1. 找个邮政工作人员（创建套接字socket()）
2. 写上地址装上信件并且投递（发送数据sendto() ）
3. 继续写信或者接收对方的回信（接受数据recvfrom()）
4. 结束回家(关闭套接字close())

3.2 示例

```
import socket

#1. 创建套接字
udp = socket.socket(socket.AF_INET, socket.SOCK_DGRAM)

#2. 准备接收方的地址和要发送的数据
sendAddr = ('10.0.81.115',8080)
sendData =input("请输入要发送的数据:")

#4. 发送数据
udp.sendto(sendData.encode("utf-8"),  sendAddr)

#5. 接收数据
```

```
recvData = udp.recvfrom(1024)
print("recvData = ", recvData)

#6. 关闭套接字
udp.close()
```



3.3 UDP发送方注意点

1. 本地 IP、本地端口（我是谁）
2. 目的 IP、目的端口（发给谁）
3. 在发送方的代码中，我们只设置了目的 IP、目的端口
4. 发送方的本地 ip、本地 port 是我们调用 sendto 的时候系统底层自动给客户端分配的。分配端口的方式为随机分配，即每次运行系统给的 port 不一样。

3.4 python3 编码转换

字符串通过编码成为字节码，字节码通过解码成为字符串：

str -> bytes: encode 编码

bytes -> str: decode 解码

示例如下：

```
text = '文本'
print(text)

bytesText = text.encode()
print(bytesText)

print(type(text))
print(type(bytesText))

textDecode = bytesText.decode()
print(textDecode)
```

运行结果：

```
文本
b'\xe6\x88\x91\xe6\x98\xaf\xe6\x96\x87\xe6\x9c\xac'
<class 'str'>
<class 'bytes'>
文本
```

其中 decode()与 encode()方法可以接受参数，其声明分别为：

```
bytes.decode(encoding="utf-8", errors="strict")
str.encode(encoding="utf-8",  errors="strict")
```

其中的 encoding 是指在解码编码过程中使用的编码(此处指“编码方案”是名词)，errors 是指错误的处理方案

4. UDP编程-接受数据

4.1 接收端接收数据的条件

UDP 网络程序想要收取数据需什么条件?

1. 确定的 ip 地址
2. 确定的端口（port）

这正如，我要收到别人寄过来的信，我必须告诉别人我的地址（ip），同时告诉别人我我的公寓信箱号（端口）。

接收端使用 bind() 函数，来完成地址结构与 socket 套接字的绑定，这样 ip、port 就固定了，发送端在 sendto 函数中指定接收端的 ip、port，就可以发送数据了。

4.2 接收流程

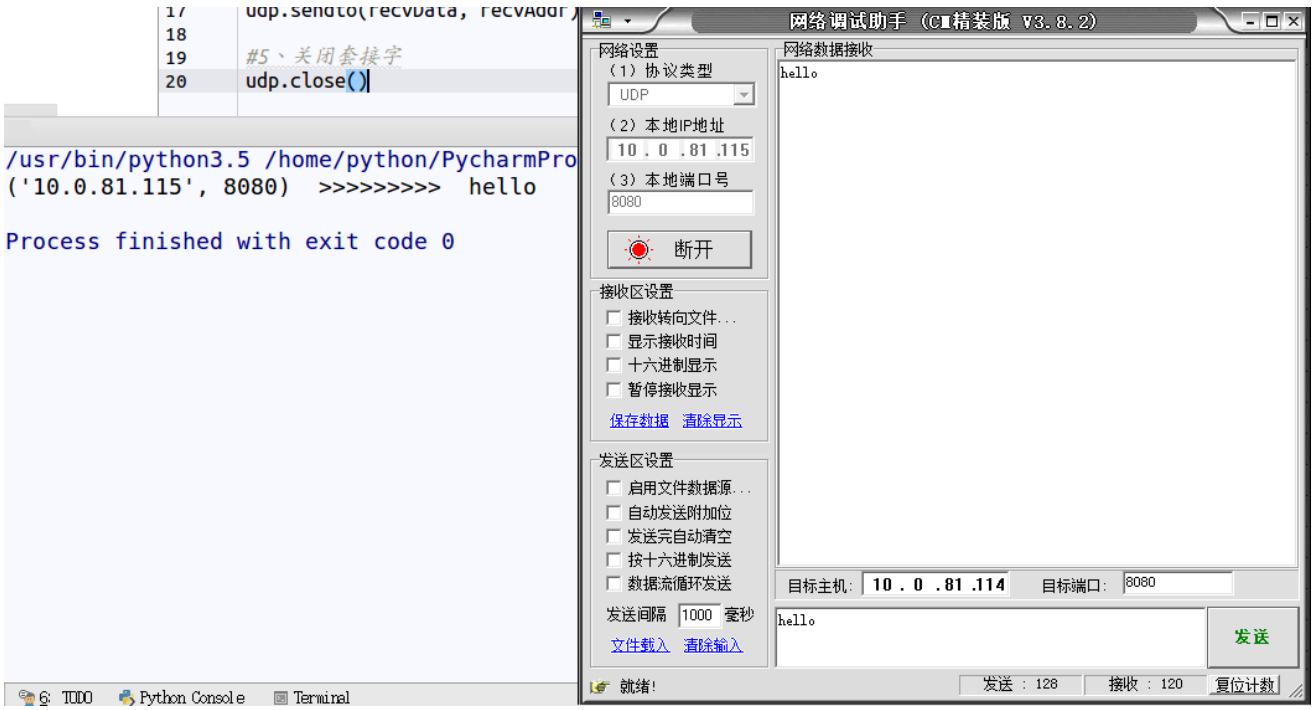
对于 UDP 服务器编程流程， 有点类似于收信过程：

1. 找个邮政工作人员（ 创建套接字socket()）
2. 确定信箱的位置：地址+信箱号（绑定bind() ）
3. 等待对方的来信（ 接受数据recvfrom()）

4. 还可以回信(发送数据sendto())，或者，继续等待对方的来信……
5. 收完回家(关闭套接字close())

4.3 示例

```
import socket

#1、创建套接字
udp = socket.socket(socket.AF_INET, socket.SOCK_DGRAM)

#2、绑定本地网络地址
# ip 一般不写，表示本机任何一个 ip
localAddr = ("", 8080)
udp.bind(localAddr)

#3、接收数据
recvData, recvAddr = udp.recvfrom(1024)#1024 表示本次接收的最大字节数
print(recvAddr, " >>>>>>>>> ", recvData.decode("utf-8"))

#4、原封不动回复数据
# recvData 为网络接收过来的数据，本来就是字节码，发送时无需编码
udp.sendto(recvData,recvAddr)

#5、关闭套接字
udp.close()
```



5. UDP应用：聊天室

5.1 案例需求：

- 在一个电脑中编写 1 个程序，有 2 个功能：
    1. 获取键盘数据，并将其发送给对方
    2. 接收数据并显示
- 并且功能数据进行选择以上的 2 个功能调用

5.2 源码：

```
import socket

def send(udp):
    # 1. 输入对方的 ip/port
    IP = input("请输入对方的 ip:")
    Port = int(input("请输入对方的 port:"))

    # 2. 输入要发送的数据
    sendData = input("请输入要发送的数据：")

    # 3. 发送数据
    udp.sendto(sendData.encode("utf-8"), (IP, Port))

def recv(udp):
    # 1. 接收数据
    recvData, recvAddr = udp.recvfrom(1024) # 默认阻塞等待用户

    # 2. 显示数据
    print(recvAddr, ">>>>", recvData.decode("utf-8"))

def main():
"""用来控制整体"""
    # 1. 创建套接字
    udp = socket.socket(socket.AF_INET, socket.SOCK_DGRAM)
    port = 8888

    # 2. 绑定端口
    udp.bind(("", port))
    print("正在使用的端口是：",port)

    # 3.根据用户的选择来调用发送/接收
    while True:
        print("1. 发送数据")
        print("2. 接收数据")
        num = input("请输入选择的功能：")
        if num == "1":
            sendMsg(udp)
        elif num == "2":
            recvMsg(udp)

if name == " main ":
    main()
```

5.3 课后思考

以上的程序如果选择了接收数据功能，并且此时没有数据，程序会堵塞在这，那么怎样才能让这个程序收发数据一起进行呢？

结合多任务编程仔细想想该如何解决