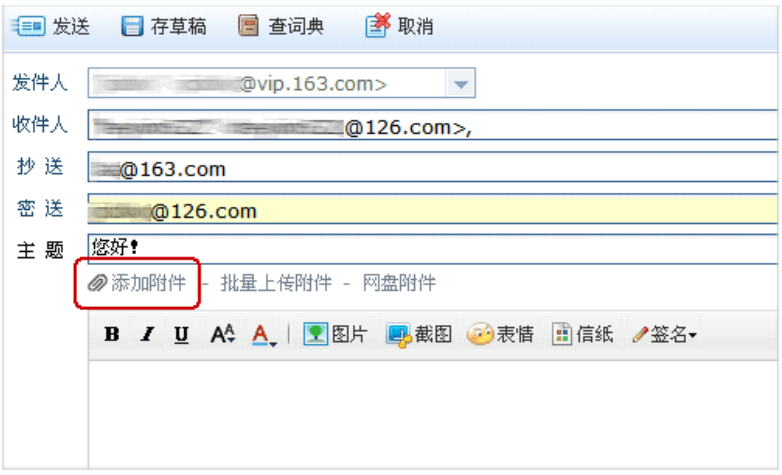
一、网络-UDP

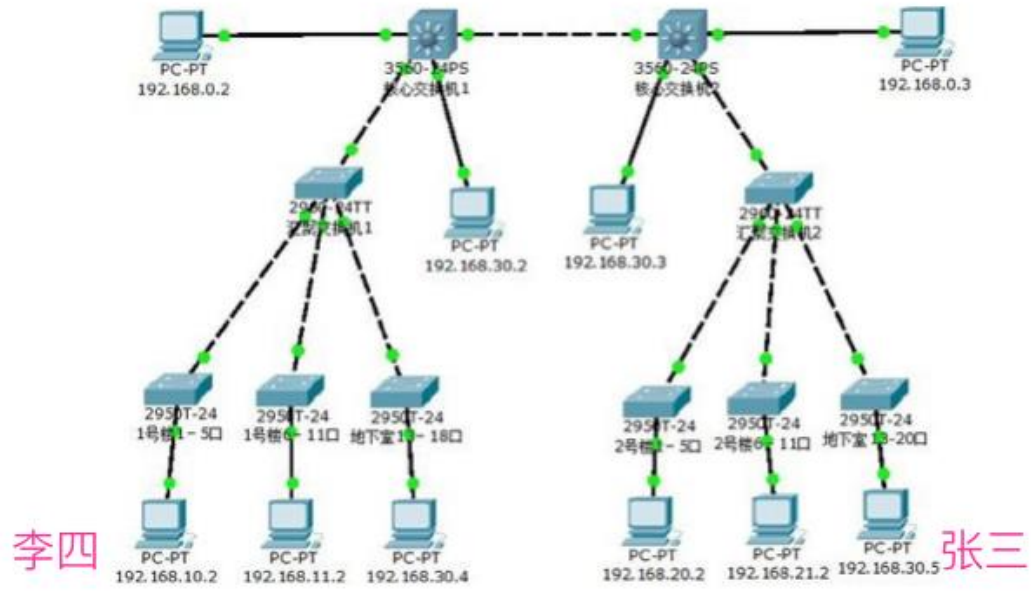
1.1 ip 地址

1.1.1. 什么是地址



地址就是用来标记地点的

1.1.2. ip 地址的作用



怎么传过去?

to:张三
content：来吃晚饭

ip 地址：用来在网络中标记一台电脑，比如 192.168.1.1；在本地局域网上是唯一的。

1.1.3. ip 地址的分类

每一个 IP 地址包括两部分：网络地址和主机地址

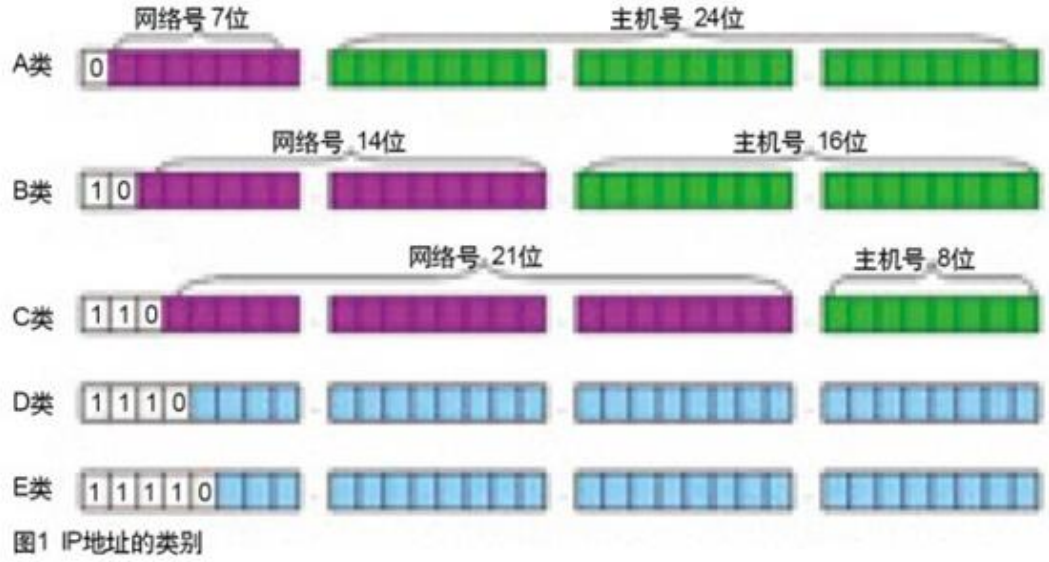


图1 IP地址的类别

3.1 A 类 IP 地址

一个 A 类 IP 地址由 1 字节的网络地址和 3 字节主机地址组成，网络地址的最高位必须是“0”，
地址范围 1.0.0.1-126.255.255.254
二进制表示为：00000001 00000000 00000000 00000001 - 01111110 11111111 11111111 11111110
可用的 A 类网络有 126 个，每个网络能容纳 1677214 个主机

3.2 B 类 IP 地址

一个 B 类 IP 地址由 2 个字节的网络地址和 2 个字节的主机地址组成，网络地址的最高位必须是“10”，
地址范围 128.1.0.1-191.255.255.254
二进制表示为：10000000 00000001 00000000 00000001 - 10111111 11111111 11111111 11111110
可用的 B 类网络有 16384 个，每个网络能容纳 65534 主机

3.3 C 类 IP 地址

一个 C 类 IP 地址由 3 字节的网络地址和 1 字节的主机地址组成，网络地址的最高位必须是“110”
范围 192.0.1.1-223.255.255.254
二进制表示为: 11000000 00000000 00000001 00000001 - 11011111 11111111 11111110 11111110
C 类网络可达 2097152 个，每个网络能容纳 254 个主机

3.4 D 类地址用于多点广播

D 类 IP 地址第一个字节以“1110”开始，它是一个专门保留的地址。
它并不指向特定的网络，目前这一类地址被用在多点广播（Multicast）中
多点广播地址用来一次寻址一组计算机 s 地址范围 224.0.0.1-239.255.255.254

3.5 E 类 IP 地址

以“1111”开始，为将来使用保留
E 类地址保留，仅作实验和开发用

3.6 私有 ip

在这么多网络 IP 中，国际规定有一部分 IP 地址是用于我们的局域网使用，也就

是属于私网 IP，不在公网中使用的，它们的范围是：

10.0.0.0 ~ 10.255.255.255

172.16.0.0 ~ 172.31.255.255

192.168.0.0 ~ 192.168.255.255

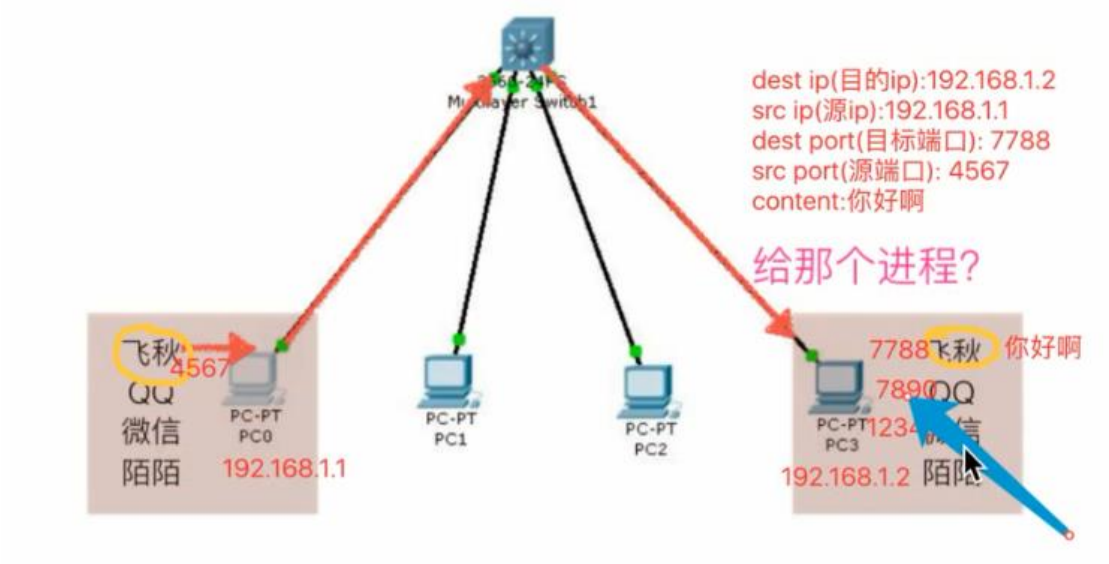
3.7 注意

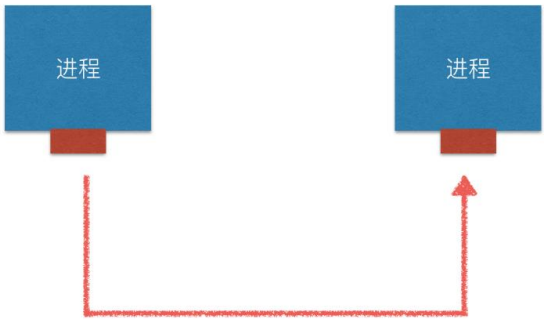
IP 地址 127．0．0．1~127．255．255．255 用于回路测试，

如：127.0.0.1 可以代表本机 IP 地址，用 http://127.0.0.1 就可以测试本机中配置的 Web 服务器。

1.2 端口

1.2.1 什么是端口





如果一个程序需要收发网络数据，那么就需要有这样的端口

在 linux 系统中，端口可以有 65536（2 的 16 次方）个之多！

既然有这么多，操作系统为了统一管理，所以进行了编号，这就是端口号

1.2.2. 端口是怎样分配的

端口号不是随意使用的，而是按照一定的规定进行分配。

端口的分类标准有好几种，我们这里不做详细讲解，只介绍一下知名端口和动态端口

1.2.2.1 知名端口（Well Known Ports）

知名端口是众所周知的端口号，范围从 0 到 1023

80 端口分配给 HTTP 服务
21 端口分配给 FTP 服务

可以理解为，一些常用的功能使用的号码是估计的，好比 电话号码 110、10086、10010 一样
一般情况下，如果一个程序需要使用知名端口的需要有 root 权限

1.2.2.2 动态端口（Dynamic Ports）

动态端口的范围是从 1024 到 65535
之所以称为动态端口，是因为它一般不固定分配某种服务，而是动态分配。
动态分配是指当一个系统程序或应用程序程序需要网络通信时，它向主机申请一个端口，主机从可用的端口号中分配一个供它使用。
当这个程序关闭时，同时也就释放了所占用的端口号

1.2.2.3 怎样查看端口 ？

- 用“netstat - an”查看端口状态
- lsof -i [tcp/udp]:2425

1.3 socket 简介

1.3.1 不同电脑上的进程之间如何通信

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 1 台电脑上可以通过进程号（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进程（进程）。
这样利用 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

注意：

- 所谓进程指的是：运行的程序以及运行时用到的资源这个整体称之为进程（在讲解多任务编程时进行详细讲解）
- 所谓进程间通信指的是：运行的程序之间的数据共享
- 后面课程中会详细说到，像网络层等知识，不要着急

1.3.2. 什么是 socket

socket(简称 套接字) 是进程间通信的一种方式，它与其他进程间通信的一个主要不同是：
它能实现不同主机间的进程间通信，我们网络上各种各样的服务大多都是基于 Socket 来完成通信的
例如我们每天浏览网页、QQ 聊天、收发 email 等等

1.3.3. 创建 socket

在 Python 中 使用 socket 模块的函数 socket 就可以完成：

```
import socket
socket.socket(AddressFamily, Type)
```

说明：

函数 socket.socket 创建一个 socket，该函数带有两个参数：

- Address Family：可以选择 AF_INET（用于 Internet 进程间通信） 或者 AF_UNIX（用于同一台机器进程间通信）,实际工作中常用 AF_INET
- Type：套接字类型，可以是 SOCK_STREAM（流式套接字，主要用于 TCP 协议）或者 SOCK_DGRAM（数据报套接字，主要用于 UDP 协议）
- 创建一个 tcp socket（tcp 套接字）

```
import socket

# 创建 tcp 的套接字
s = socket.socket(socket.AF_INET, socket.SOCK_STREAM)

# ...这里是使用套接字的功能（省略）...

# 不用的时候，关闭套接字
s.close()
```

创建一个 udp socket（udp 套接字）

```
import socket

# 创建 udp 的套接字
s = socket.socket(socket.AF_INET, socket.SOCK_DGRAM)

# ...这里是使用套接字的功能（省略）...

# 不用的时候，关闭套接字
s.close()
```

说明

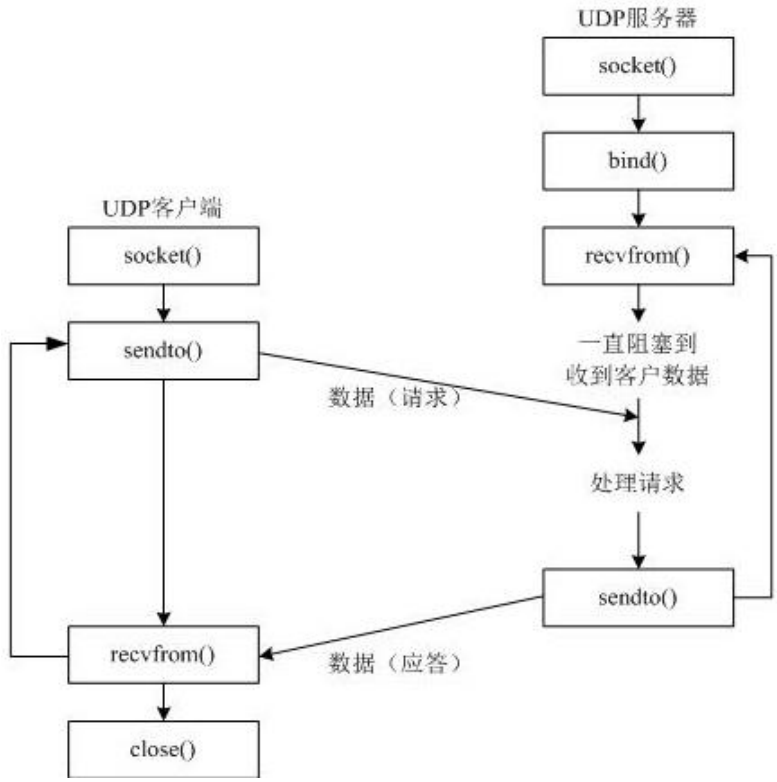
- 套接字使用流程 与 文件的使用流程很类似
  - 创建套接字
  - 使用套接字收/发数据
  - 关闭套接字

1.4 udp 网络程序-发送、接收数据

1.4.1. udp 网络收发数据流程

创建一个基于 udp 的网络程序流程很简单，具体步骤如下：

1. 创建客户端套接字
2. 发送/接收数据
3. 关闭套接字



1.4.1.1. udp 网络程序-发送数据

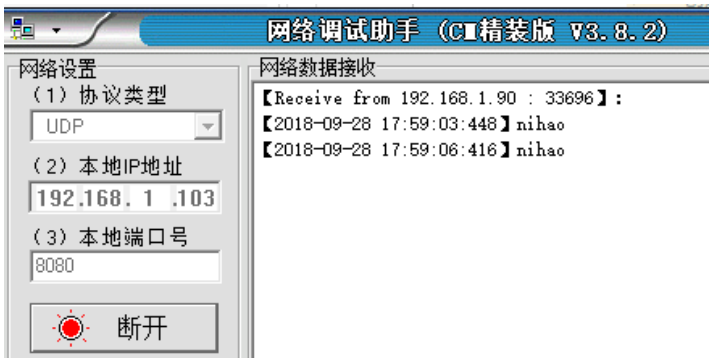
```python
import socket

def main():
    # 创建udp套接字
    udp_socket = socket.socket(socket.AF_INET, socket.SOCK_DGRAM)

    while True:

        # 从键盘获取数据
        send_data = input("输入要发送的数据：")

        # 如果输入exit，则退出
        if send_data == 'exit':
            break

        # 使用udp收发数据
        # send_data = "hello udp"

        udp_socket.sendto(send_data.encode('utf-8'), ("192.168.1.103", 8080))

    # 关闭udp套接字
    udp_socket.close()

if __name__ == '__main__':
    main()
```

运行现象：

在 Ubuntu 中运行脚本

```
ython@lhq:~/Desktop/网络编程$ python3 02-socket-udp-发送任意数据.py
输入要发送的数据：nihao
输入要发送的数据：nihao
输入要发送的数据：
```

在 windows 中运行“网络调试助手”：



1.4.1.2. udp 网络程序-接收数据

```python
# encoding: utf-8

import socket

def main():
    # 创建udp套接字
    udp_socket = socket.socket(socket.AF_INET, socket.SOCK_DGRAM)

    # 绑定端口
    local_addr = ("", 7788)
    udp_socket.bind(local_addr)  # 必须绑定自己电脑的ip

    while True:

        # 接受数据
        recv_data = udp_socket.recvfrom(1204)

        # 打印数据
        print("打印完整接收内容：%s" % str(recv_data))
        print("打印接收数据：%s" % recv_data[0].decode('gbk'))

    # 关闭udp套接字
    udp_socket.close()

if __name__ == '__main__':
    main()
```

网络调试助手截图：

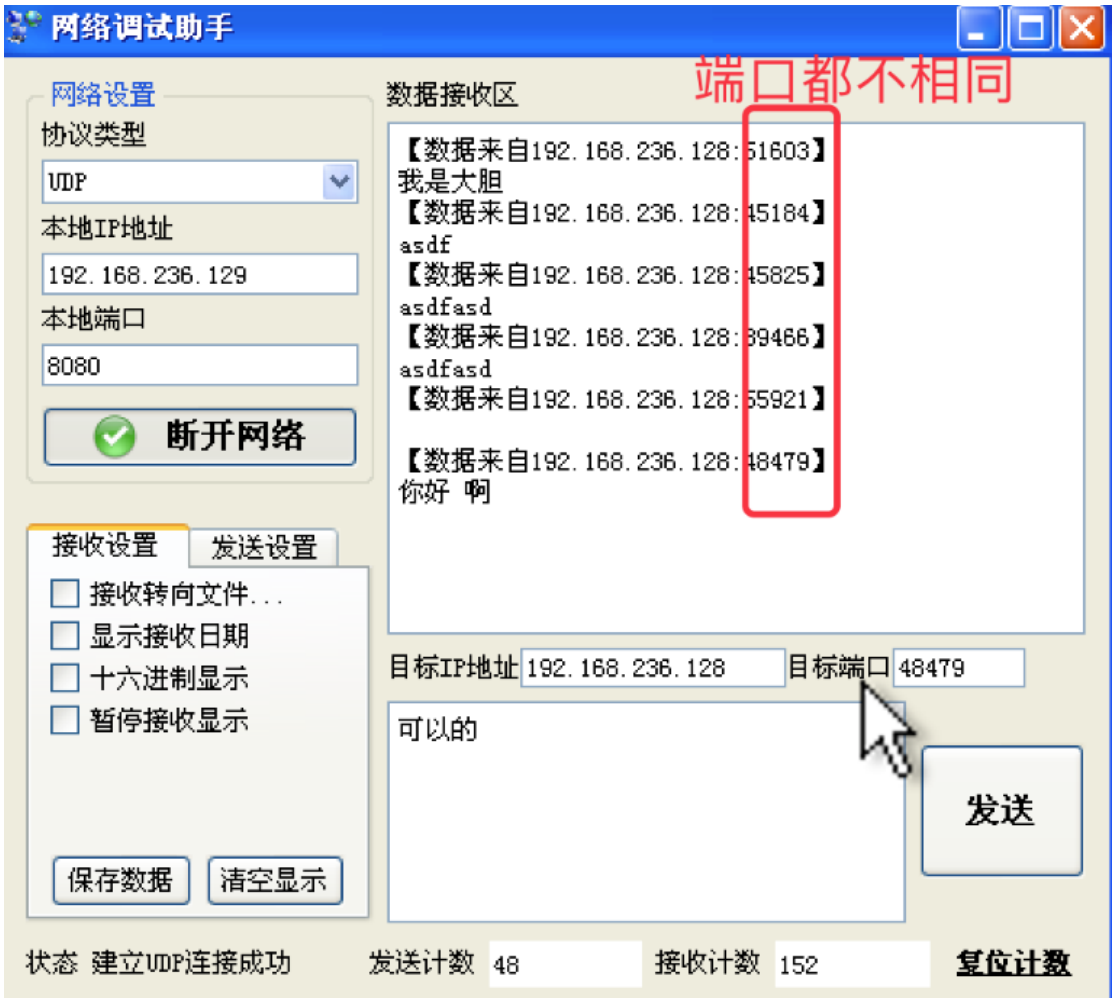
python 脚本：

```
python@lhq:~/Desktop/网络编程$ python3 03-socket-upd-接收任意数据.py
打印完整接收内容：(b'\xc4\xe3\xba\xc3', ('192.168.1.103', 8080))
打印接收数据：你好
打印完整接收内容：(b'\xc4\xe3\xba\xc3', ('192.168.1.103', 8080))
打印接收数据：你好
```

1.5 udp 绑定信息

1.5.1. udp 网络程序-端口问题

- 会变的端口号
  重新运行多次脚本，然后在“网络调试助手”中，看到的现象如下：



说明：

- 每重新运行一次网络程序，上图中红圈中的数字，不一样的原因在于，这个数字标识这个网络程序，当重新运行时，如果没有确定到底用哪个，系统默认会随机分配
- 记住一点：这个网络程序在运行的过程中，这个就唯一标识这个程序，所以如果其他电脑上的网络程序如果想要向此程序发送数据，那么就需要向这个数字（即端口）标识的程序发送即可

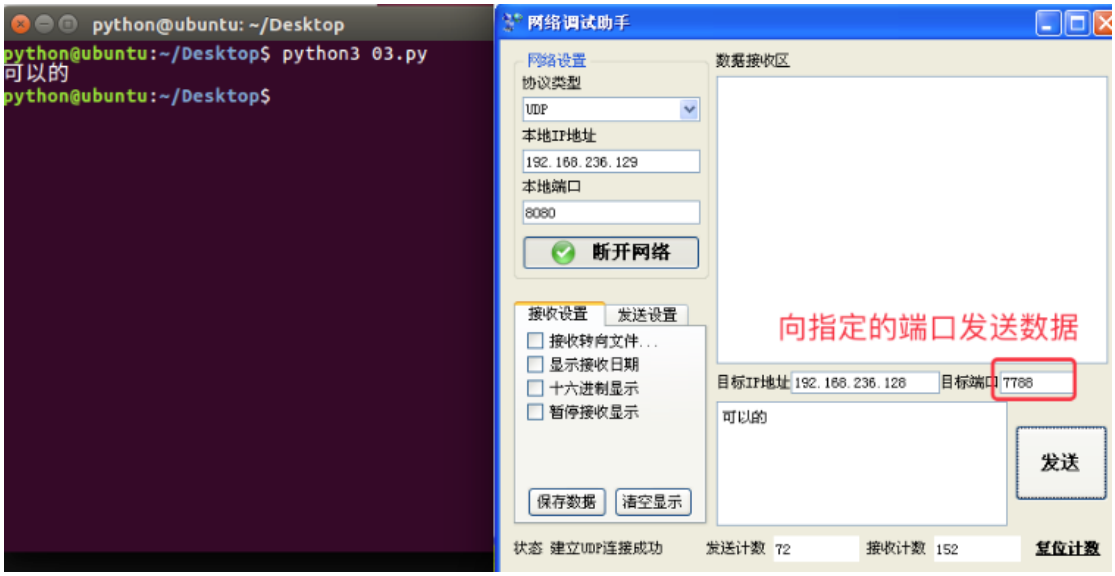
1.5.2. udp 绑定信息

<1>. 绑定信息

一般情况下，在一台电脑上运行的网络程序有很多，为了不与其他的网络程序占用同一个端口号，往往在编程中，udp 的端口号一般不绑定

但是如果需要做成一个服务器端的程序的话，是需要绑定的，想想看这又是为什么呢？

如果报警电话每天都在变，想必世界就会乱了，所以一般服务性的程序，往往需要一个固定的端口号，这就是所谓的端口绑定



<2>. 总结

- 一个 udp 网络程序，可以不绑定，此时操作系统会随机进行分配一个端口，如果重新运行此程序端口可能会发生变化
- 一个 udp 网络程序，也可以绑定信息（ip 地址，端口号），如果绑定成功，那么操作系统用这个端口号来进行区别收到的网络数据是否是此进程的

发送端也可以绑定端口：

```
import socket

def main():
    # 创建udp套接字
    udp_socket = socket.socket(socket.AF_INET, socket.SOCK_DGRAM)

    # 绑定端口
    local_addr = ("", 7788)
    udp_socket.bind(local_addr)

    while True:

        # 从键盘获取数据
        send_data = input("输入要发送的数据：")

        # 如果输入exit，则退出
        if send_data == 'exit':
            break

        # 使用udp收发数据
        # send_data = "hello udp"

        udp_socket.sendto(send_data.encode('utf-8'), ("192.168.1.103", 8080))

    # 关闭udp套接字
    udp_socket.close()

if __name__ == '__main__':
    main()
```

1.6 应用：udp 聊天器

单工：收音机

半双工：对讲机

全双工：手机

Soceket 套接字是全双工；

```python
def send_msg(udp_socket, dist_addr):
    """发送消息"""
    # 发送
    send_data = input("输入信息：")
    udp_socket.sendto(send_data.encode('utf-8'), dist_addr)


def recv_msg(udp_socket):
    """接收消息"""
    recv_data = udp_socket.recvfrom(1024)
    # windows的中文是gbk编码，linux的中文都是utf-8编码
    print("%s : %s" % (str(recv_data[1]), recv_data[0].decode('utf-8')))


def main():

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

    # 绑定端口
    local_addr = ("", 7788)
    udp_socket.bind(local_addr)  # 必须绑定自己电脑的ip

    # 目标地址
    dist_addr = ("127.0.0.1", 8877)

    while True:

        # 发送
        send_msg(udp_socket, dist_addr)

        # 接收信息
        recv_msg(udp_socket)

    # 关闭udp套接字
    udp_socket.close()


if __name__ == '__main__':
    main()
```

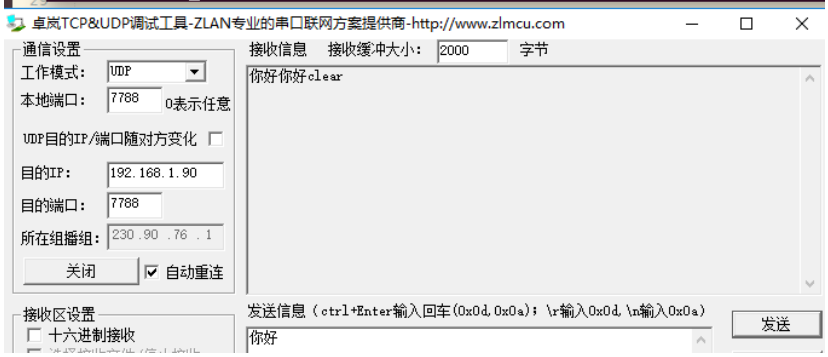
- 以上的程序如果选择了接收数据功能，并且此时没有数据，程序会堵塞在这，那么怎样才能让这个程序收发数据一起进行呢？别着急，学习完多任务知识之后就解决了 O(∩_∩)O...

  以上是半双工的实现，属于堵塞状态，需要加入线程才能实现全双工。

  多线程版本：

```python
import socket
import threading


def recv_msg(udp_socket):
    """接收并显示数据"""

    while True:
        recv_data = udp_socket.recvfrom(1024)
        print(recv_data[0].decode('gbk'))


def send_msg(udp_socket, dist_ip, dist_port):
    """发送数据"""
    while True:
        send_data = input("输入发送数据:")
        udp_socket.sendto(send_data.encode('gbk'), (dist_ip, dist_port))


def main():
    """完成UDP聊天的整体控制"""

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

    # 绑定端口
    udp_socket.bind(("", 7788))

    # 对方ip/port
    dist_ip = "192.168.1.103"
    dist_port = 7788

    # 创建线程进行收发数据
    t1 = threading.Thread(target=send_msg, args=(udp_socket, dist_ip, dist_port))
    t2 = threading.Thread(target=recv_msg, args=(udp_socket, ))

    t1.start()
    t2.start()

    t1.join()
    t2.join()

    # 关闭套接字
    udp_socket.close()


if __name__ == '__main__':
    main()
```

```
python@lhq:~/Desktop/网络编程$ python3 11-upd聊天-多线程.py
输入发送数据:你好
输入发送数据:你好
clear
输入发送数据:
```



二、网络-TCP

2.1 TCP 简介

TCP 介绍

TCP 协议，传输控制协议（英语：Transmission Control Protocol，缩写为 TCP）是一种面向连接的、可靠的、基于字节流的传输层通信协议，由 IETF 的 RFC 793 定义。

TCP 通信需要经过**创建连接、数据传送、终止连接**三个步骤。

TCP 通信模型中，在通信开始之前，一定要先建立相关的链接，才能发送数据，类似于生活中，"打电话""

TCP 特点

1. 面向连接

通信双方必须先建立连接才能进行数据的传输，双方都必须为该连接分配必要的系统内核资源，以管理连接的状态和连接上的传输。

双方间的数据传输都可以通过这一个连接进行。

完成数据交换后，双方必须断开此连接，以释放系统资源。

这种连接是一对一的，因此 TCP 不适用于广播的应用程序，基于广播的应用程序请使用 UDP 协议。

2. 可靠传输

1）TCP 采用发送应答机制

TCP 发送的每个报文段都必须得到接收方的应答才认为这个 TCP 报文段传输成功

2）超时重传

发送端发出一个报文段之后就启动定时器，如果在定时时间内没有收到应答就重新发送这个报文段。

TCP 为了保证不发生丢包，就给每个包一个序号，同时序号也保证了传送到接收端实体的包的按序接收。然后接收端实体对已成功收到的包发回一个相应的确认（ACK）；如果发送端实体在合理的往返时延（RTT）内未收到确认，那么对应的数据包就被假设为已丢失将会被进行重传。

3）错误校验

TCP 用一个校验和函数来检验数据是否有错误；在发送和接收时都要计算校验和。

4) 流量控制和阻塞管理

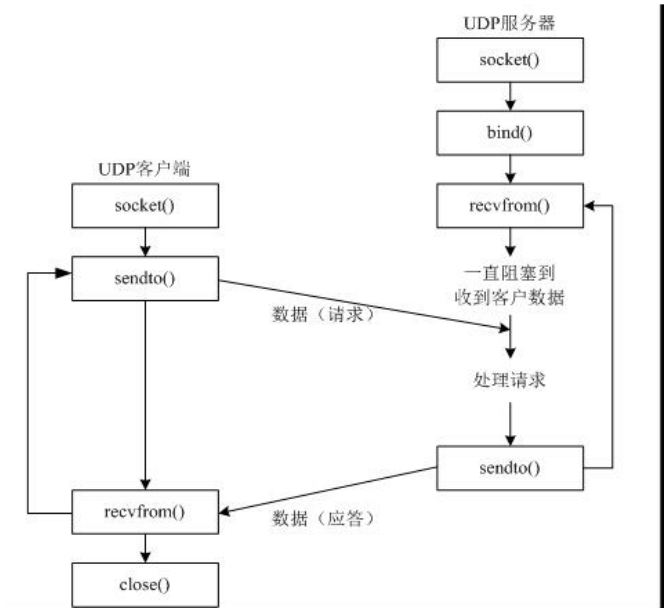
流量控制用来避免主机发送得过快而使接收方来不及完全收下。

TCP 与 UDP 的不同点

- 面向连接（确认有创建三方交握，连接已创建才作传输。）
- 有序数据传输
- 重发丢失的数据包
- 舍弃重复的数据包
- 无差错的数据传输
- 阻塞/流量控制

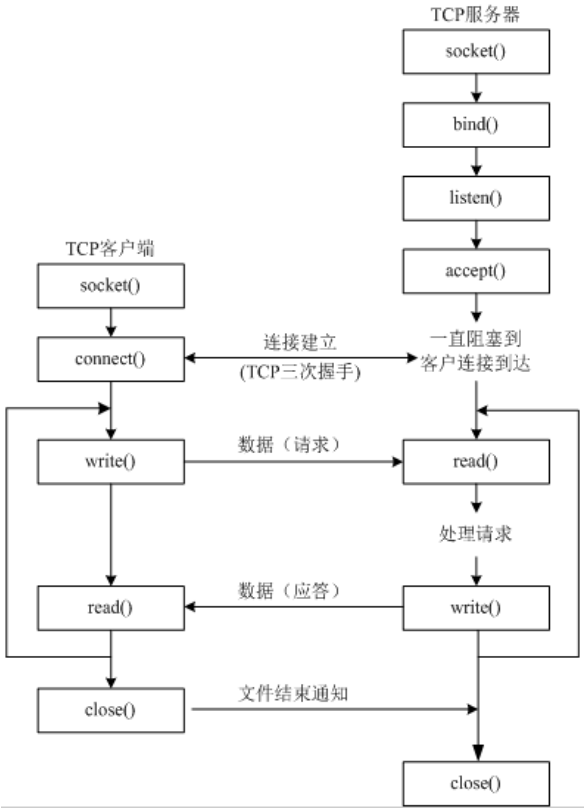
2.1.1 UDP 信模型

udp 通信模型中，在通信开始之前，不需要建立相关的链接，只需要发送数据即可，类似于生活中，"写信""



2.1.2 TCP 通信模型

udp 通信模型中，在通信开始之前，一定要先建立相关的链接，才能发送数据，类似于生活中，"打电话""



2.2 tcp 客户端

tcp 客户端，并不是像之前一个段子：一个顾客去饭馆吃饭，这个顾客要点菜，就问服务员咱们饭店有客户端么，然后这个服务员非常客气的说道：先生 我们饭店不用客户端，我们直接送到您的餐桌上

如果，不学习网络的知识是不是 说不定也会发生那样的笑话 ，哈哈

所谓的服务器端：就是提供服务的一方，而客户端，就是需要被服务的一方

tcp 客户端构建流程

tcp 的客户端要比服务器端简单很多，如果说服务器端是需要自己买手机、查手机卡、设置铃声、等待别人打电话流程的话，那么客户端就只需要找一个电话亭，拿起电话拨打即可，流程要少很多

步骤（4 布）：

- 创建套接字
- 建立链接
- 发送数据
- 关闭套接字

备注：tcp 发送数据比 udp 多一个 connect 过程，同时 send 时候不需要传入地址；

```python
import socket


def main():

    # 创建tcp套接字
    tcp_socket = socket.socket(socket.AF_INET, socket.SOCK_STREAM)

    # 链接服务器
    dist_addr = ("192.168.1.103", 8877)
    tcp_socket.connect(dist_addr)

    # 发送数据
    send_data = input("输入发送数据：")
    tcp_socket.send(send_data.encode("gbk"))

    # 关闭tcp套接字
    tcp_socket.close()


if __name__ == '__main__':
    main()
```

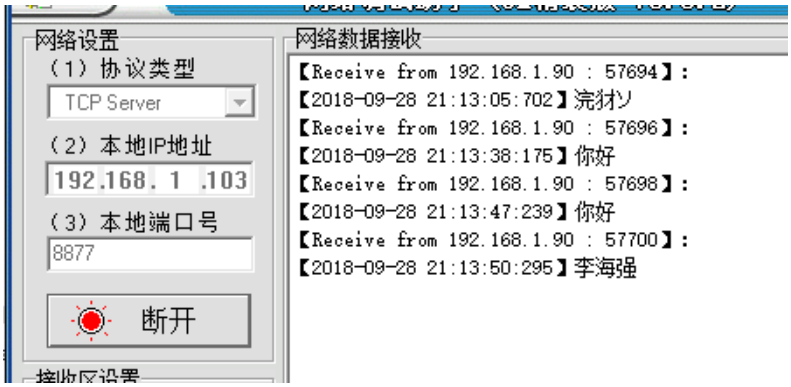
定义是TCP

给windows系统发送数据用gbk，linux用utf-8;
接收windows的数据用gbk解码，linux用utf-8

结果：

```
python@lhq:~/Desktop/网络编程$ python3 05-tcp客户端-client.py
输入发送数据：你好
python@lhq:~/Desktop/网络编程$ python3 05-tcp客户端-client.py
输入发送数据：李海强
python@lhq:~/Desktop/网络编程$
```

Windows 客户端：



2.3 tcp 服务器

生活中的电话机

如果想让别人能更够打通咱们的电话获取相应服务的话，需要做以下几件事情：

1. 买个手机
2. 插上手机卡
3. 设计手机为正常接听状态（即能够响铃）
4. 静静的等着别人拨打

tcp 服务器

如同上面的电话机过程一样，在程序中，如果想要完成一个 tcp 服务器的功能，需要的流程如下：

1. socket 创建一个套接字
2. bind 绑定 ip 和 port
3. listen 使套接字变为可以被动链接
4. accept 等待客户端的链接
5. recv/send 接收发送数据
6. 关闭套接字

服务一个客服

```python
import socket


def main():

    # 1、创建套接字
    tcp_server_socket = socket.socket(socket.AF_INET, socket.SOCK_STREAM)

    # 2、绑定本地信息
    tcp_server_socket.bind(('', 7788))

    # 3、listen-状态改为被动
    tcp_server_socket.listen(128)

    # 4、accept-等待链接
    new_client_socket, client_addr = tcp_server_socket.accept()
    print(client_addr)

    # 5、接收/发送数据
    recv_data = new_client_socket.recv(1024)
    print(recv_data.decode('gbk'))

    new_client_socket.send("收到数据".encode('gbk'))

    # 6、关闭套接字
    new_client_socket.close()
    # tcp_server_socket.close()


if __name__ == '__main__':
    main()
```

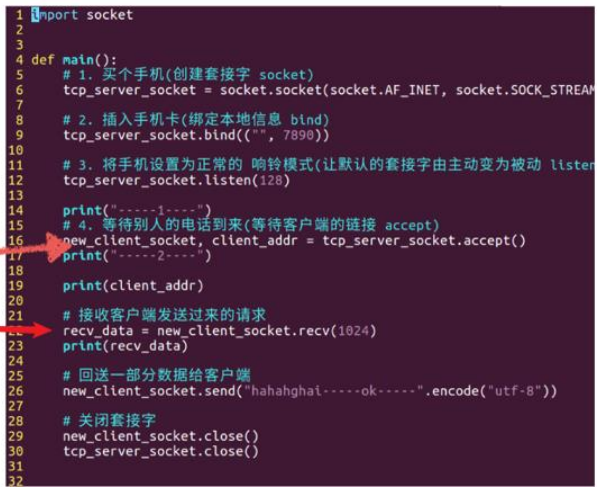
堵塞；当有客户端连接时解堵塞

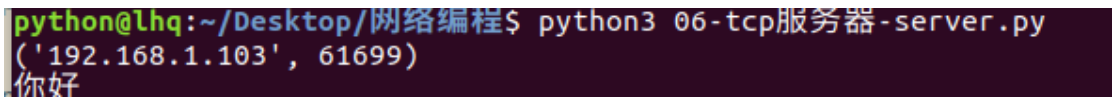
堵塞；收到数据有解堵塞

给windows调试工具发送用gbk

客户端

服务器



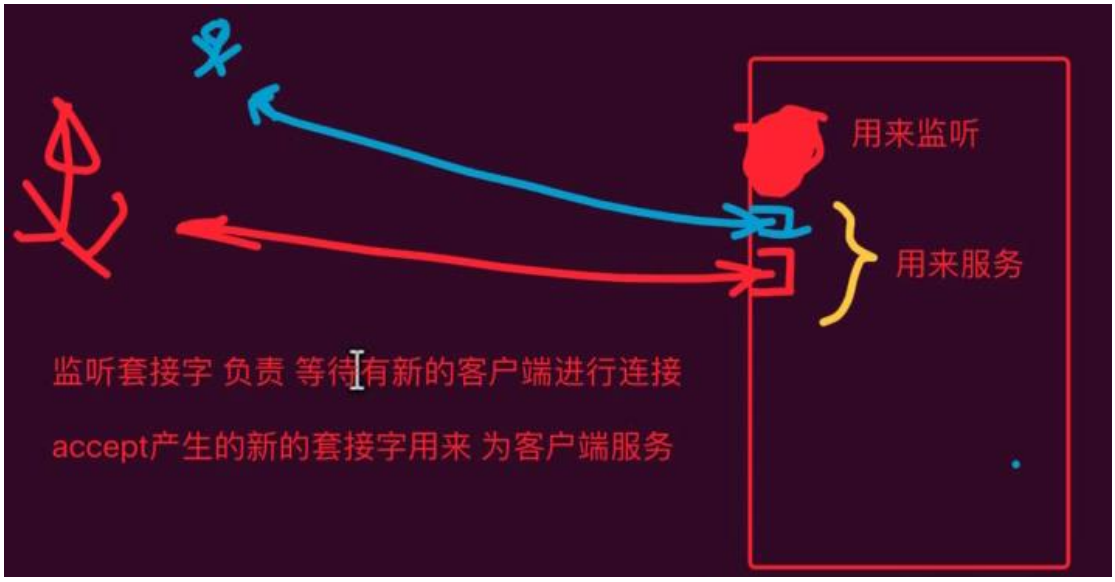


Windows 客户端：

tcp 注意点

1. tcp 服务器一般情况下都需要绑定，否则客户端找不到这个服务器
2. tcp 客户端一般不绑定，因为是主动链接服务器，所以只要确定好服务器的 ip、port 等信息就好，本地客户端可以随机
3. tcp 服务器中通过 listen 可以将 socket 创建出来的主动套接字变为被动的，这是做 tcp 服务器时必须要做的
4. 当客户端需要链接服务器时，就需要使用 connect 进行链接，udp 是不需要链接的而是直接发送，但是 tcp 必须先链接，只有链接成功才能通信
5. **当一个 tcp 客户端连接服务器时，服务器端会有 1 个新的套接字，这个套接字用来标记这个客户端，单独为这个客户端服务**
6. **listen 后的套接字是被动套接字，用来接收新的客户端的链接请求的，而 accept 返回的新套接字是标记这个新客户端的**
7. **关闭 listen 后的套接字意味着被动套接字关闭了，会导致新的客户端不能够链接服务器，但是之前已经链接成功的客户端正常通信。**
8. **关闭 accept 返回的套接字意味着这个客户端已经服务完毕**
9. **当客户端的套接字调用 close 后，服务器端会 recv 解堵塞，并且返回的长度为 0，因此服务器可以通过返回数据的长度来区别客户端是否已经下线**

备注：accept 后 tcp 服务端会创建新的 socket 给客户服务，有几个链接就创捷几个服务套接字；



服务多个客服，但只服务一次

```python
import socket


def main():

    # 1、创建套接字
    tcp_server_socket = socket.socket(socket.AF_INET, socket.SOCK_STREAM)

    # 2、绑定本地信息
    tcp_server_socket.bind(('', 7788))

    # 3、listen-状态改为被动
    tcp_server_socket.listen(128)

    while True:

        # 4、accept-等待链接
        new_client_socket, client_addr = tcp_server_socket.accept()
        print(client_addr)

        # 5、接收/发送数据
        recv_data = new_client_socket.recv(1024)
        print(recv_data.decode('gbk'))

        new_client_socket.send("收到数据".encode('gbk'))

        # 6、关闭套接字
        new_client_socket.close()

    tcp_server_socket.close()


if __name__ == '__main__':
    main()
```

关闭两种 socket 的不同：

```
    # 关闭套接字
    # 关闭accept返回的套接字 意味着 不会在为这个客户端服务
    new_client_socket.close()
    print("已经服务器完毕。。。。")

# 如果将监听套接字 关闭了，那么会导致 不能再次等待新客户端的到来，即xxxx.accept就会失败
tcp_server_socket.close()
```

```
python@lhq:~/Desktop/网络编程$ python3 07-tcp服务端-循环服务.py
('192.168.1.103', 65018)
你好
('192.168.1.103', 65070)
你是谁
('192.168.1.103', 65154)
我是李海强
('192.168.1.103', 65156)
```

服务多个客服，服务多次

```python
def main():
    # 1. 买个手机(创建套接字 socket)
    tcp_server_socket = socket.socket(socket.AF_INET, socket.SOCK_STREAM)

    # 2. 插入手机卡(绑定本地信息 bind)
    tcp_server_socket.bind(("", 7890))

    # 3. 将手机设置为正常的 响铃模式(让默认的套接字由主动变为被动 listen)
    tcp_server_socket.listen(128)

    while True:
        print("等待一个新的客户端的到来...")
        # 4. 等待别人的电话到来(等待客户端的链接 accept)
        new_client_socket, client_addr = tcp_server_socket.accept()

        print("一个新的客户端已经到来%s" % str(client_addr))

        while True:
            # 接收客户端发送过来的请求
            recv_data = new_client_socket.recv(1024)
            print("客户端福送过来的请求是:%s" % recv_data.decode("utf-8"))

            # 回送一部分数据给客户端
            new_client_socket.send("hahahghai-----ok-----".encode("utf-8"))

        # 关闭套接字
        # 关闭accept返回的套接字 意味着 不会在为这个客户端服务
        new_client_socket.close()
        print("已经服务器完毕。。。。")

    # 如果将监听套接字 关闭了，那么会导致 不能再次等待新客户端的到来，即xxxx.accept就会失败
```

这个while True循环为多个客户端服务

这个while True 循环多次为同一个客户端服务多次

循环服务多次，但是没有解堵塞；
思考：什么时候需要解除该次服务；
答：当客户端断开连接时；
问题：那什么判断断开连接？

```python
def main():

    # 1、创建套接字
    tcp_server_socket = socket.socket(socket.AF_INET, socket.SOCK_STREAM)

    # 2、绑定本地信息
    tcp_server_socket.bind(('', 7788))

    # 3、listen-状态改为被动
    tcp_server_socket.listen(128)

    while True:

        # 4、accept-等待链接
        new_client_socket, client_addr = tcp_server_socket.accept()
        print(client_addr)

        while True:

            # 5、接收/发送数据
            recv_data = new_client_socket.recv(1024)
            print(recv_data.decode('gbk'))

            # 如果recv解堵塞，那么有两种可能：
            # 1、客户端发送数据
            # 2、客户端调用close导致
            if recv_data:
                new_client_socket.send("收到数据".encode('gbk'))
            else:
                print("当前服务完成")
                break

        # 6、关闭套接字
        new_client_socket.close()

    tcp_server_socket.close()
```

rec解堵塞两种情况

```python
import socket


def main():

    # 1、创建套接字
    tcp_server_socket = socket.socket(socket.AF_INET, socket.SOCK_STREAM)

    # 2、绑定本地信息
    tcp_server_socket.bind(('', 7788))

    # 3、listen-状态改为被动
    tcp_server_socket.listen(128)

    while True:

        # 4、accept-等待链接
        new_client_socket, client_addr = tcp_server_socket.accept()
        print(client_addr)

        while True:

            # 5、接收/发送数据
            recv_data = new_client_socket.recv(1024)
            print(recv_data.decode('gbk'))

            # 如果recv解堵塞，那么有两种可能：
            # 1、客户端发送数据
            # 2、客户端调用close导致
            if recv_data:
                new_client_socket.send("收到数据".encode('gbk'))
            else:
                print("当前服务完成")
                break

        # 6、关闭套接字
        new_client_socket.close()

    tcp_server_socket.close()


if __name__ == '__main__':
    main()
```

```
python@lhq:~/Desktop/网络编程$ python3 08-tcp服务端-循环服务并且多次为一个客户服务.py
('192.168.1.90', 55796)

当前服务完成
('192.168.1.90', 55798)

当前服务完成
('192.168.1.90', 55800)

当前服务完成
```

备注：如果需要同时服务多个客户并且服务多次，这需要引入多线程，将每个客服服务创建一个线程。

2.4 案例:文件下载器

思路：

- 写客户端，用网络调试工具调试客户端
- 写服务端，用网络调试工具调试服务端
- 联合调试 bug

客户端代码：

```python
import socket


def main():

    # 创建套接字
    tcp_socket = socket.socket(socket.AF_INET, socket.SOCK_STREAM)

    # 获取服务器的i port
    dist_ip = input("输入服务器ip：")
    dist_port = input("输入服务器port：")

    # 链接服务器
    tcp_socket.connect((dist_ip, int(dist_port)))

    # 获取下载的文件名
    download_file_name = input("输入下载的文件名字：")

    # 将文件名发送服务器
    tcp_socket.send(download_file_name.encode('utf-8'))

    # 接收文件数据
    recv_data = tcp_socket.recv(1024*1024)  # 1024*1024 --> 1M

    if recv_data:
        # 保存文件数据
        with open("[接收]" + download_file_name, 'wb') as f:
            f.write(recv_data)
    else:
        print("文件不存在！")

    # 关闭套接字
    tcp_socket.close()


if __name__ == '__main__':
    main()
```

服务端：

```python
def main():

    # 1、创建套接字
    tcp_server_socket = socket.socket(socket.AF_INET, socket.SOCK_STREAM)

    # 2、绑定本地信息
    tcp_server_socket.bind(('', 7788))

    # 3、listen-状态改为被动
    tcp_server_socket.listen(128)

    while True:

        # 4、accept-等待链接
        new_client_socket, client_addr = tcp_server_socket.accept()

        # 5、接收下载文件名
        recv_data = new_client_socket.recv(1024)

        if recv_data:
            upload_file_name = recv_data.decode('utf-8')
            print("客户端（%s）需要下载的文件：%s" % (client_addr, upload_file_name))

            # 6、读取文件数据
            try:
                print("读取文件")
                f = open(upload_file_name, 'rb')
                send_data = f.read()
                f.close()

                # 7、传输文件
                new_client_socket.send(send_data)
            except:
                print("文件不存在！")

        else:
            pass

        print("当前服务完成")

        # 8、关闭套接字
        new_client_socket.close()

    tcp_server_socket.close()
```

直接传输2进制文件

在ｕｂｕｎｔｕ下联合调试，进行本机文件传输：

```
python@lhq:~/Desktop/网络编程$ python3 09-案例\:文件下载-客户端.py
输入服务器ip：192.168.1.90
输入服务器port：7788
输入下载的文件名字：1
文件不存在！
python@lhq:~/Desktop/网络编程$ python3 09-案例\:文件下载-客户端.py
输入服务器ip：192.168.1.90
输入服务器port：7788
输入下载的文件名字：01-socket基本使用.py
python@lhq:~/Desktop/网络编程$
```

```
python@lhq:~/Desktop/网络编程$ python3 10-案例\:文件下载-服务端.py
客户端（('192.168.1.90', 55832)）需要下载的文件：1
读取文件
文件不存在！
当前服务完成
客户端（('192.168.1.90', 55834)）需要下载的文件：01-socket基本使用.py
读取文件
当前服务完成
```

传输结果：

思考：

- 对于客户端，可以将 recv 开辟一个线程；
- 对于服务端，将一个 accept 开辟一个线程；

三、网络通信

3.1 tcp-ip 简介

作为新时代标杆的我们，已经离不开手机、离不开网络，对于互联网大家可能耳熟能详，但是计算机网络的出现比互联网要早很多

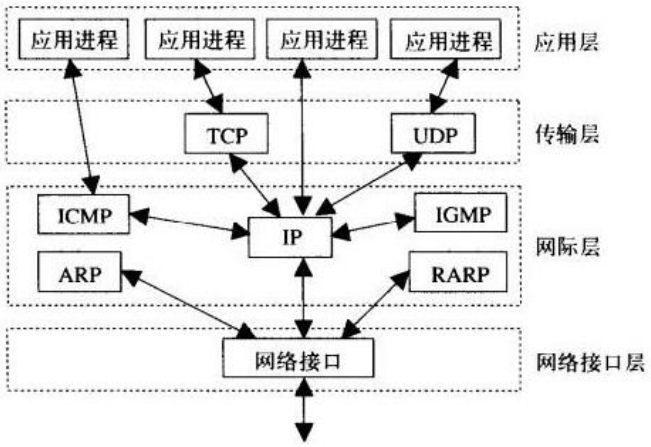
TCP/IP 协议(族)

早期的计算机网络，都是由各厂商自己规定一套协议，IBM、Apple 和 Microsoft 都有各自的网络协议，互不兼容

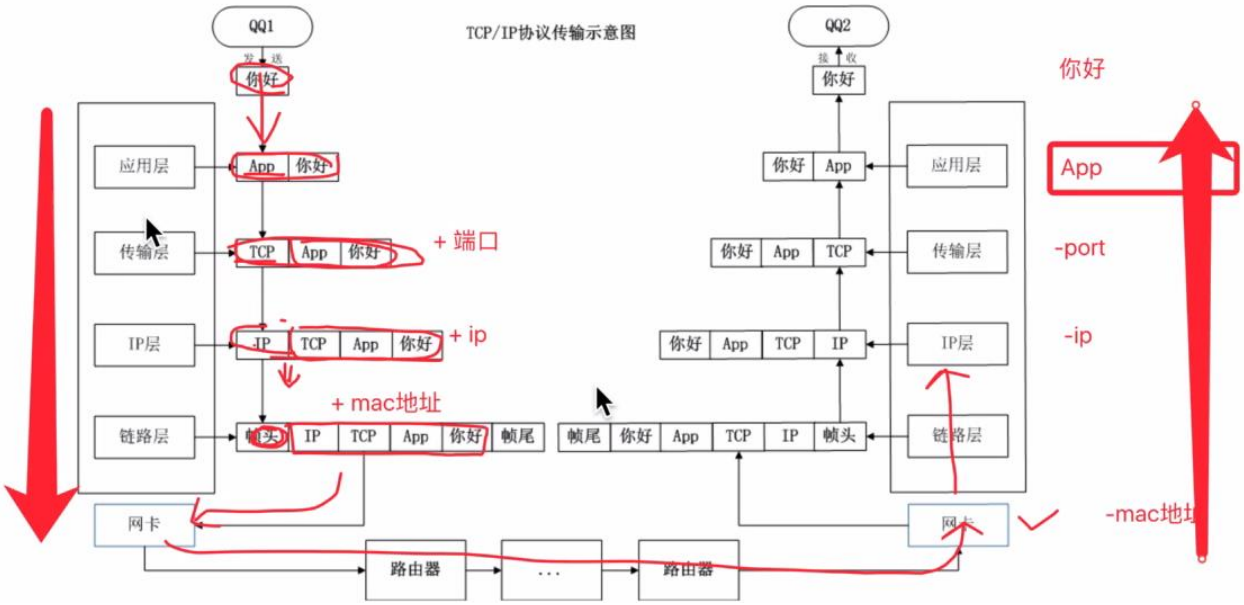
为了把全世界的所有不同类型的计算机都连接起来，就必须规定一套全球通用的协议，为了实现互联网这个目标，互联网协议族（Internet Protocol Suite）就是通用协议标准。

因为互联网协议包含了上百种协议标准，但是最重要的两个协议是 TCP 和 IP 协议，所以，大家把互联网的协议简称 TCP/IP 协议(族)

常用的网络协议如下图所示：



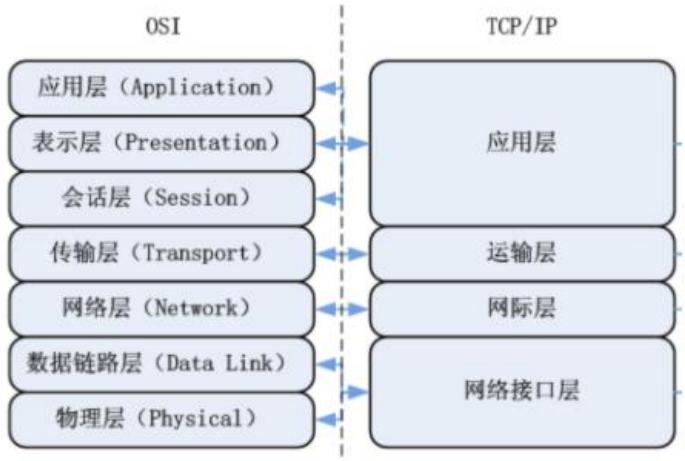
TCP/IP协议族中各协议之间的关系



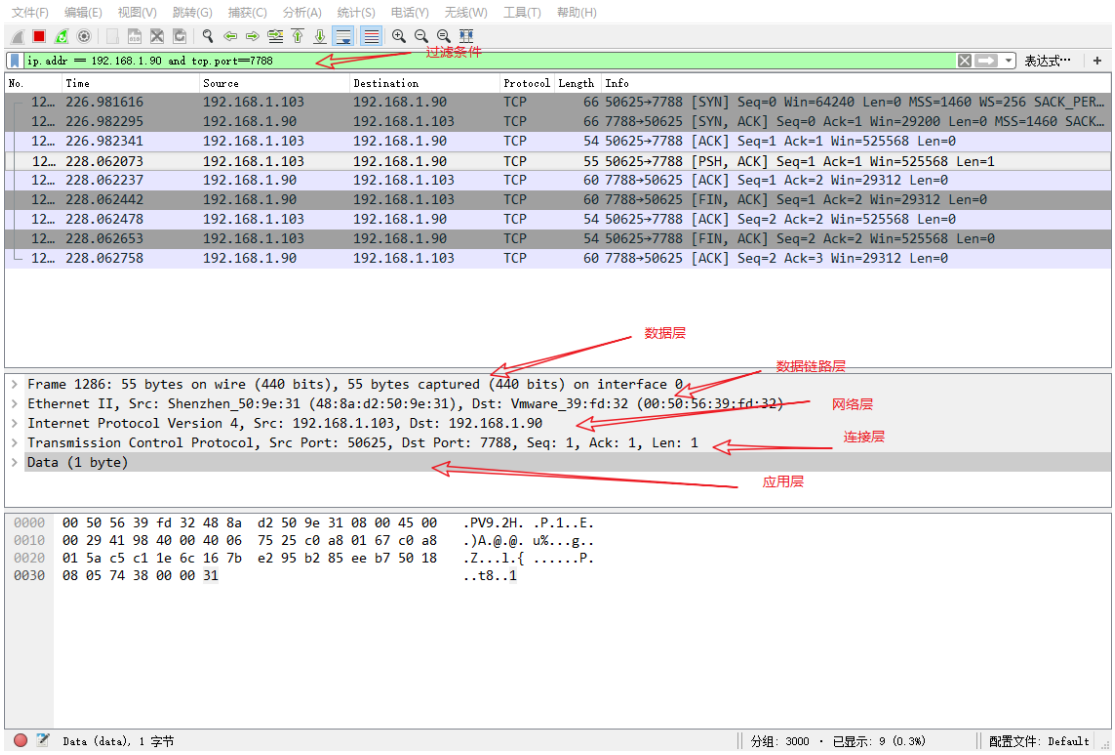
说明：

网际层也称为：网络层
网络接口层也称为：链路层

另外一套标准

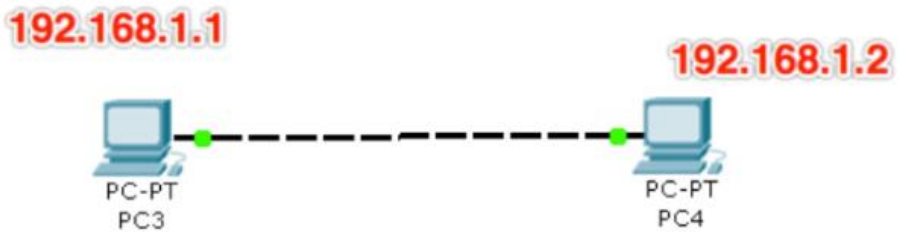


3.2 wireshark 抓包工具使用



3.3 网络通信过程

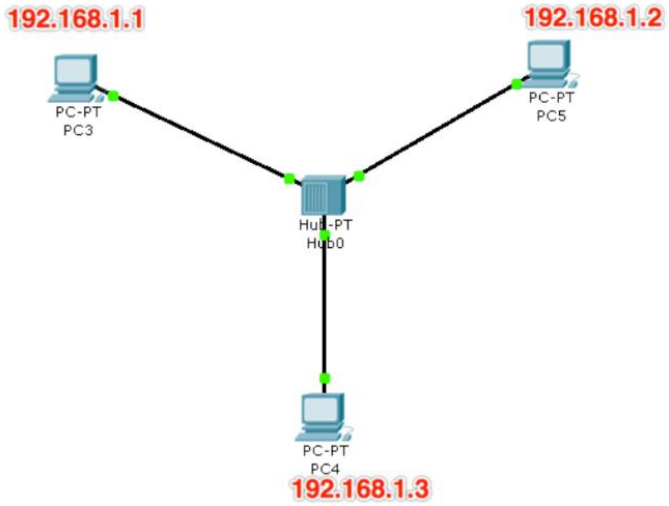
3.3.1 1 台电脑的网络



说明

- 如果两台电脑之间通过网线连接是可以直接通信的，但是需要提前设置好 ip 地址以及网络掩码
- 并且 ip 地址需要控制在同一网段内（子网掩码一致），例如 一台为 192.168.1.1 另一台为 192.168.1.2 则可以进行通信

3.3.2. 使用集线器组成一个网络

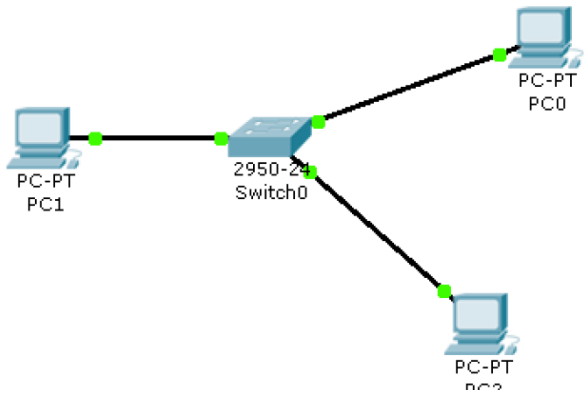


备注：为啥中间需要设备不能直接用网线连接：网络传输的是数据信号，而非电流，中间有间断，如果多台电脑用网线连接，数据信号就被打乱，不能正常通信。

说明

- 当有多态电脑需要组成一个网时，那么可以通过集线器（Hub）将其链接在一起
- 一般情况下集线器的接口较少
- 集线器有个缺点，它以广播的方式进行发送任何数据，即如果集线器接收到来自 A 电脑的数据本来是想转发给 B 电脑，如果此时它还连接着另外两台电脑 C、D，那么它会把这个数据给每个电脑都发送一份，因此会导致网络拥堵

3.3.3. 使用交换机组成一个网络

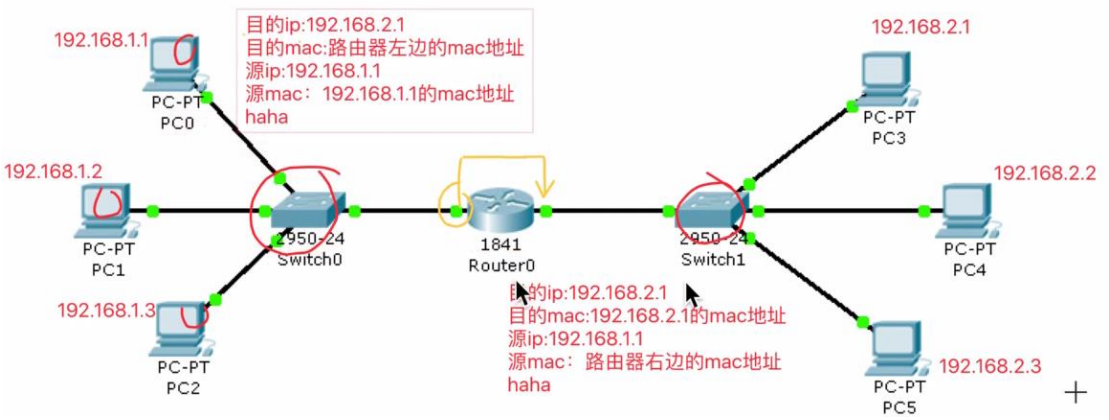


说明

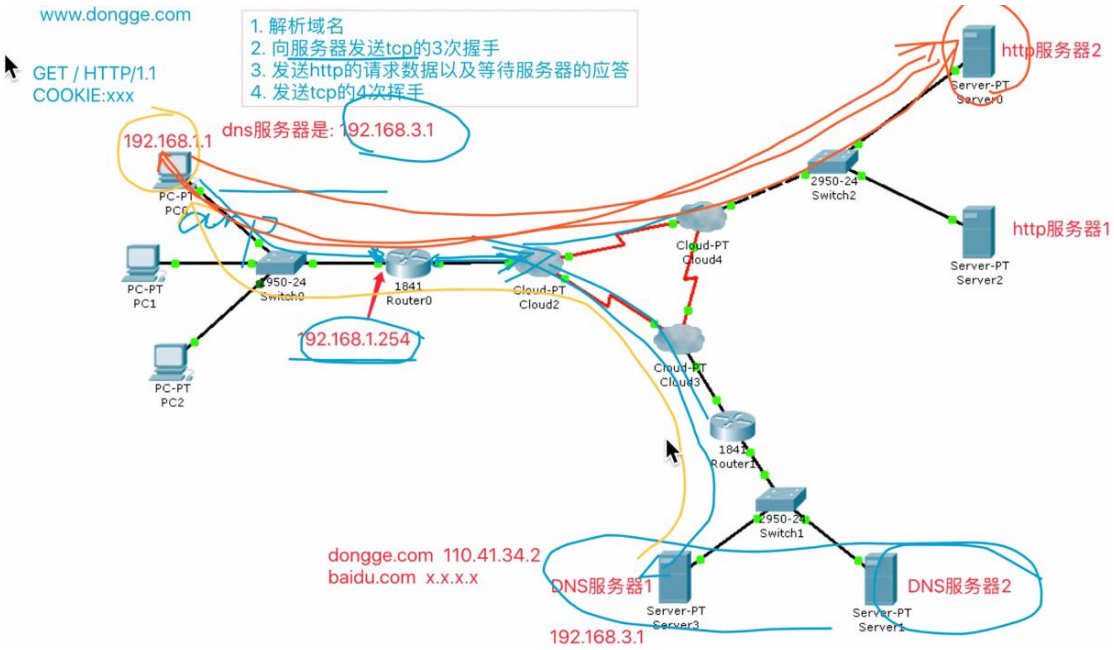
- 克服了集线器以广播发送数据的缺点，当需要广播的时候发送广播，当需要单播的时候又能够以单播的方式进行发送
- 它已经替代了之前的集线器
- 企业中就是用交换机来完成多态电脑设备的链接成网络的

3.3.4 使用路由器连接多个网络

路由器功能：
将多个子网连接成一个大网；



备注：传输时 ip 地址不变，mac 变化；
通信过程（复杂）



说明

- 在浏览器中输入一个网址时，需要将它先解析出 ip 地址来
- 当得到 ip 地址之后，浏览器以 tcp 的方式 3 次握手链接服务器
- 以 tcp 的方式发送 http 协议的请求数据 给 服务器
- 服务器 tcp 的方式回应 http 协议的应答数据 给浏览器

总结

- MAC 地址：在设备与设备之间数据通信时用来标记收发双方（网卡的序列号）
- IP 地址：在逻辑上标记一台电脑，用来指引数据包的收发方向（相当于电脑的序列号）
- 网络掩码：用来区分 ip 地址的网络号和主机号
- 默认网关：当需要发送的数据包的目的 ip 不在本网段内时，就会发送给默认的一台电脑，成为网关
- 集线器：已过时，用来连接多态电脑，缺点：每次收发数据都进行广播，网络会变的拥堵
- 交换机：集线器的升级版，有学习功能知道需要发送给哪台设备，根据需要进行单播、广播
- 路由器：连接多个不同的网段，让他们之间可以进行收发数据，每次收到数据后，ip 不变，但是 MAC 地址会变化
- DNS：用来解析出 IP（类似电话簿）
- http 服务器：提供浏览器能够访问到的数据