11. 网络编程

11.1. 通信协议

计算机为了联网，就必须规定通信协议，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，这就好比一群人有的说英语，有的说中文，有的说德语，说同一种语言的人可以交流，不同的语言之间就不行了。

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，**互联网协议簇（Internet Protocol Suite）就是通用协议标准**。Internet 是由 inter 和 net 两个单词组合起来的，原意就是**连接“网络”的网络**，有了Internet，任何私有网络，只要支持这个协议，就可以联入互联网。

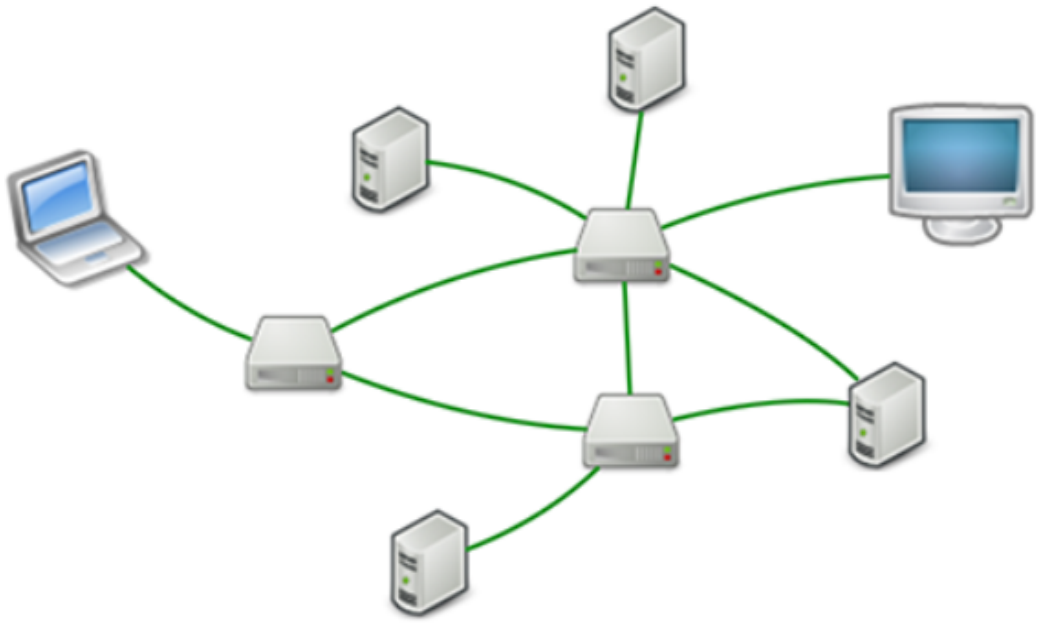
11.1.1. TCP/IP 协议

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。

1. IP 协议

通信的时候，双方必须知道对方的标识，好比发邮件必须知道对方的邮件地址。**互联网上每个计算机的唯一标识就是 IP 地址**，类似`123.123.123.123`。如果一台计算机同时接入到两个或更多的网络，比如路由器，它就会有两个或多个 IP 地址，所以，IP 地址对应的实际上是计算机的网络接口，通常是网卡。

IP 协议负责把数据从一台计算机通过网络发送到另一台计算机。数据被分割成一小块一小块，然后通过IP包发送出去。由于互联网链路复杂，两台计算机之间经常有多条线路，因此，路由器就负责决定如何把一个 IP 包转发出去。**IP 包的特点是按块发送，途径多个路由，但不保证能到达，也不保证顺序到达。**



IP 地址实际上是一个 32 位整数（称为`IPv4`），以字符串表示的 IP 地址如`192.168.0.1`实际上是把 32 位整数按 8 位分组后的数字表示，目的是便于阅读。其二进制格式为：

`11000000.10101000.00000000.00000001`

`IPv6`地址实际上是一个`128`位整数，它是目前使用的`IPv4`的升级版，以字符串表示类似于`2001:0db8:85a3:0042:1000:8a2e:0370:7334`。

2. TCP 协议

TCP 协议则是建立在 IP 协议之上的。TCP 协议负责在两台计算机之间建立**可靠连接**，保证数据包按顺序到达。**TCP 协议会通过握手建立连接，然后，对每个 IP 包编号，确保对方按顺序收到，如果包丢掉了，就自动重发。**

TCP 协议会通过 3 次握手建立可靠连接。其过程如下：

1. 客户端发送 SYN（SEQ=x）报文给服务器端，进入`SYN_SEND`状态。
2. 服务器端收到 SYN 报文，回应一个 SYN（SEQ=y）ACK（ACK=x+1）报文，进入`SYN_RECV`状态。

3. 客户端收到服务器端的 SYN 报文，回应一个 ACK（ACK=y+1）报文，进入`Established`状态。

三次握手完成，TCP客户端和服务器端成功地建立连接，可以开始传输数据了。

许多常用的更高级的协议都是建立在 TCP 协议基础上的，比如用于浏览器的 HTTP 协议、发送邮件的 SMTP 协议等。

一个 TCP 报文除了包含要传输的数据外，还包含源 IP 地址和目标 IP 地址，源端口和目标端口。

端口有什么作用？在两台计算机通信时，只发送 IP 地址是不够的，因为同一台计算机上跑着多个网络程序。一个 TCP 报文来了之后，到底是交给浏览器还是QQ，就需要端口号来区分。每个网络程序都向操作系统申请唯一的端口号，这样，两个进程在两台计算机之间建立网络连接就需要各自的 IP 地址和各自的端口号。

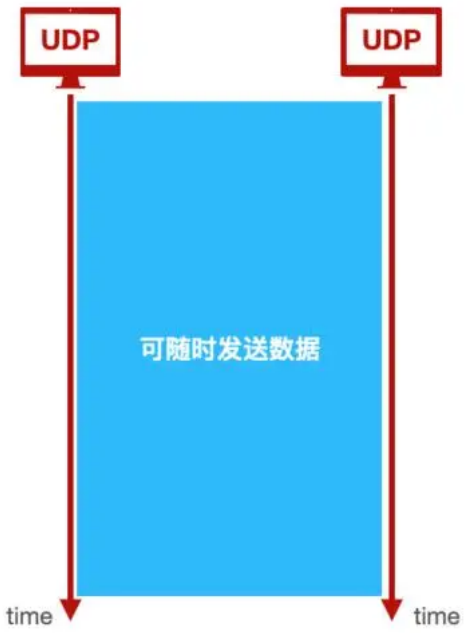
一个进程也可能同时与多个计算机建立链接，因此它会申请很多端口。端口不是随意使用的，而是按照一定的规定进行分配的。例如：80 端口分配给 HTTP 服务，21 端口分配给 FTP 服务。

11.1.2. UDP 协议

TCP 是建立可靠连接，并且通信双方都可以**以流的形式**发送数据。相对 TCP，UDP 则是面向无连接的协议。

使用 UDP 协议时，不需要建立连接，只需要知道对方的 IP 地址和端口号，就可以直接发数据包。但是，能不能到达就不知道了。

虽然用 UDP 传输数据不可靠，但它的优点是和 TCP 比，**速度快**，对于不要求可靠到达的数据，就可以使用 UDP 协议。



11.1.3. Socket 套接字

Socket 是网络编程的一个抽象概念。通常我们用一个 Socket 表示“打开了一个网络链接”，而打开一个 Socket 需要知道目标计算机的 IP 地址和端口号，再指定协议类型即可。

为了让两个程序通过网络进行通信，两者均必须使用 Socket（套接字）。Socket 的英文原意是“**孔**”或“**插座**”，通常也称为“**套接字**”。用于描述 IP 地址和端目，它是不通信链的句柄，可以用来实现不同虚拟机或不同计算机之间的通信。

Socket 正如其英文原意那样，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。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。

在 Python 中使用 socket 模块的`socket()`函数可以实现网络通信，语法格式如下：

```
s = socket.socket(AddressFamily, Type)
```

参数说明如下：

- Address Family：可以是 AF_INET （用于 Internet 进程之间的通信）或 AF_UNIX（于同一台主机进程之间的通信），在实际工作中常用 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 SOCK_DGRAM（数据报套接字，主要用于 UDP 协议）。

例如，为了创建 `TCP/IP` 套接字，可以用下面的方式调用`socket.socket()`：

```
tcpSock = socket.socket(socket.AF_INET,
socket.SOCK_STREAM)
```

同样地，为了创建 `UDP/IP` 套接字，需要执行以下语句：

```
udpSock = socket.socket(socket.AF_INET,
socket.SOCK_DGRAM)
```

创建完成后，生成一个 socket 对象，socket 对象的主要函数及其说明：

函数	说明
`bind()`	绑定地址到套接字。在 AF_INFT 下，以**元组**的形式表示地址
`listen()`	开始 TCP监听，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`，大部分应用程序设置为 `5` 就可以了
`accept()`	被动接受 TCP 客户端连接（**阻塞式**），等待连接的到来
`connect()`	主动初始化 TCP 服务器连接，一般 address 的格式为**元组** `(hostname, port)`。如果连接出错，则返回 `socket.error` 错误
`recv()`	接收 TCP 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
`send()`	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

函数	说明
`sendall()`	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。如果成功，则返回 None，失败则抛出异常
`recvfrom()`	接收 UDP 数据，与 `rcev()` 函数类似，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
`sendto()`	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
`close()`	关闭套接字

11.2. TCP 编程

11.2.1. 客户端

大多数连接都是可靠的 TCP 连接。创建 TCP 连接时，主动发起连接的叫客户端，被动响应连接的叫服务器。

创建一个基于 TCP 连接的 Socket，可以这样做：

```
# 导入socket库：
import socket

# 创建一个socket：
s = socket.socket(socket.AF_INET,
socket.SOCK_STREAM)
# 建立连接：
s.connect(('www.baidu.com', 80))
```

> 提示
>
> 创建`Socket`时，`AF_INET`指定使用IPv4协议，如果要用更先进的IPv6，就指定为`AF_INET6`。`SOCK_STREAM`指定使用面向流的TCP协议，这样，一个`Socket`对象就创建成功，但是还没有建立连接。

客户端要主动发起 TCP 连接，必须知道服务器的 IP 地址和端口号。百度网站的 IP 地址可以用域名`www.baidu.com`自动转换到 IP 地址，但是怎么知道百度服务器的端口号呢？

答案是作为服务器，提供什么样的服务，**端口号就必须固定下来**。由于我们想要访问网页，因此百度提供网页服务的服务器必须把端口号固定在`80`端口，**因为`80`端口是Web服务的标准端口**。其他服务都有对应的标准端口号，例如 SMTP 服务是`25`端口，FTP 服务是`21`端口，等等。端口号小于 1024 的是 Internet 标准服务的端口，端口号大于 1024 的，可以任意使用。

建立 TCP 连接后，我们就可以向百度服务器发送请求，要求返回首页的内容：

```
# 发送数据：
s.send(b'GET / HTTP/1.1\r\nHost:
www.baidu.com\r\nConnection: close\r\n\r\n')
```

TCP连接创建的是**双向通道**，双方都可以同时给对方发数据。但是谁先发谁后发，怎么协调，要根据具体的协议来决定。例如，HTTP 协议规定客户端必须先发请求给服务器，服务器收到后才发数据给客户端。

发送的文本格式必须符合 HTTP 标准，如果格式没问题，接下来就可以接收百度服务器返回的数据了：

```
# 接收数据:
buffer = []
while True:
    # 每次最多接收1k字节:
    d = s.recv(1024)
    if d:
        buffer.append(d)
    else:
        break
data = b''.join(buffer)
```

接收数据时，调用`recv(max)`方法，一次最多接收指定的字节数，因此，在一个`while`循环中反复接收，直到`recv()`返回空数据，表示接收完毕，退出循环。

当我们接收完数据后，调用`close()`方法关闭 Socket，这样，一次完整的网络通信就结束了：

```
# 关闭连接
s.close()
```

接收到的数据包括 HTTP 头和网页本身，我们只需要把 HTTP 头和网页分离一下，把 HTTP 头打印出来，网页内容保存到文件：

```
header, html = data.split(b'\r\n\r\n', 1)
print(header.decode('utf-8'))
# 把接收的数据写入文件:
with open('baidu.html', 'wb') as f:
    f.write(html)
```

现在，只需要在浏览器中打开这个`baidu.html`文件，就可以看到百度的首页了。

完整代码：

```
import socket

s = socket.socket(socket.AF_INET,
socket.SOCK_STREAM)
s.connect(('www.baidu.com', 80))
# 发送数据：
s.send(b'GET / HTTP/1.1\r\nHost:
www.baidu.com\r\nConnection: close\r\n\r\n')
# 接收数据：
buffer = []
while True:
    # 每次最多接收1k字节：
    d = s.recv(1024)
    if d:
        buffer.append(d)
    else:
        break
data = b''.join(buffer)
# 关闭连接
s.close()

header, html = data.split(b'\r\n\r\n', 1)
print(header.decode('utf-8'))
# 把接收的数据写入文件：
with open('baidu.html', 'wb') as f:
    f.write(html)
```

```
    f.close()
```

11.2.2. 服务端

简介

和客户端编程相比，服务器编程就要复杂一些。

服务器进程首先要绑定一个端口并监听来自其他客户端的连接。如果某个客户端连接过来了，服务器就与该客户端建立 Socket 连接，随后的通信就靠这个 Socket 连接了。

所以，服务器会打开固定端口（比如`80`）监听，每来一个客户端连接，就创建该 Socket 连接。由于服务器会有大量来自客户端的连接，所以，服务器要能够区分一个 Socket 连接是和哪个客户端绑定的。一个 Socket 依赖 4 项：**服务器地址**、**服务器端口**、**客户端地址**、**客户端端口**来唯一确定一个 Socket。

但是服务器还需要同时响应多个客户端的请求，所以，每个连接都需要一个新的进程或者新的线程来处理，否则，服务器一次就只能服务一个客户端了。

简单的服务器程序

首先，创建一个基于 IPv4 和 TCP 协议的 Socket：

```
s = socket.socket(socket.AF_INET,
socket.SOCK_STREAM)
```

然后，我们要绑定监听的地址和端口。服务器可能有多块网卡，可以绑定到某一块网卡的 IP 地址上，也可以用`0.0.0.0`绑定到所有的网络地址，还可以用`127.0.0.1`绑定到本机地址。`127.0.0.1`是一个特殊的 IP 地址，表示本机地址，如果绑定到这个地址，客户端必须同时在本机运行才能连接，也就是说，外部的计算机无法连接进来。

端口号需要预先指定。因为我们写的这个服务不是标准服务，所以用`9999`这个端口号。**请注意，小于`1024`的端口号必须要有管理员权限才能绑定**：

```
# 绑定端口:
s.bind(('127.0.0.1', 9999))
```

紧接着，调用`listen()`方法开始监听端口，传入的参数指定等待连接的最大数量：

```
s.listen(5)
print('Waiting for connection...')
```

接下来，服务器程序通过一个**永久循环**来接受来自客户端的连接，`accept()`会等待并返回一个客户端的连接：

```
while True:
    # 接受一个新连接:
    sock, addr = s.accept()
    # 创建新线程来处理TCP连接:
    t = threading.Thread(target=tcp_link,
args=(sock, addr))
    t.start()
```

每个连接都必须创建新线程（或进程）来处理，否则，单线程在处理连接的过程中，无法接受其他客户端的连接：

```
def tcp_link(sock, addr):
    print('接收一个新的连接:{}...'.format(addr))
    sock.send(b'Welcome!')
    while True:
        data = sock.recv(1024)
        time.sleep(1)
        if not data or data.decode('utf-8') ==
'exit':
            break
        sock.send(('Hello, %s!' %
data.decode('utf-8')).encode('utf-8'))
    sock.close()
    print('来自%s的连接:%s 连接关闭。' % addr)
```

连接建立后，服务器首先发一条欢迎消息，然后等待客户端数据，并加上`Hello`再发送给客户端。如果客户端发送了`exit`字符串，就直接关闭连接。

完整代码：

```
import socket
import threading
import time

s = socket.socket(socket.AF_INET,
socket.SOCK_STREAM)


def tcp_link(sock, addr):
    print('接收一个新的连接:{}...'.format(addr))
    sock.send(b'Welcome!')
    while True:
        data = sock.recv(1024)
        time.sleep(1)
        if not data or data.decode('utf-8') ==
'exit':
            break
        sock.send(('Hello, %s!' %
data.decode('utf-8')).encode('utf-8'))
    sock.close()
    print('来自%s的连接:%s 连接关闭。' % addr)


# 绑定IP和端口
s.bind(('127.0.0.1', 9999))
# 最大监听数（客户端同时连接数）
s.listen(5)
print('等待连接中。。。')
```

```
while True:
    # 接受一个新连接:
    sock, addr = s.accept()
    # 创建新线程来处理TCP连接:
    t = threading.Thread(target=tcp_link,
args=(sock, addr))
    t.start()
```

测试这个服务器程序，还需要编写一个客户端程序:

```
import socket


s = socket.socket(socket.AF_INET,
socket.SOCK_STREAM)
# 建立连接:
s.connect(('127.0.0.1', 9999))
# 接收欢迎消息:
print(s.recv(1024).decode('utf-8'))
for data in ['张三', '李四', '王五']:
    # 发送数据:
    s.send(data.encode())
    print(s.recv(1024).decode('utf-8'))
s.send(b'exit')
s.close()
```

11.2.3. 小结

服务端：

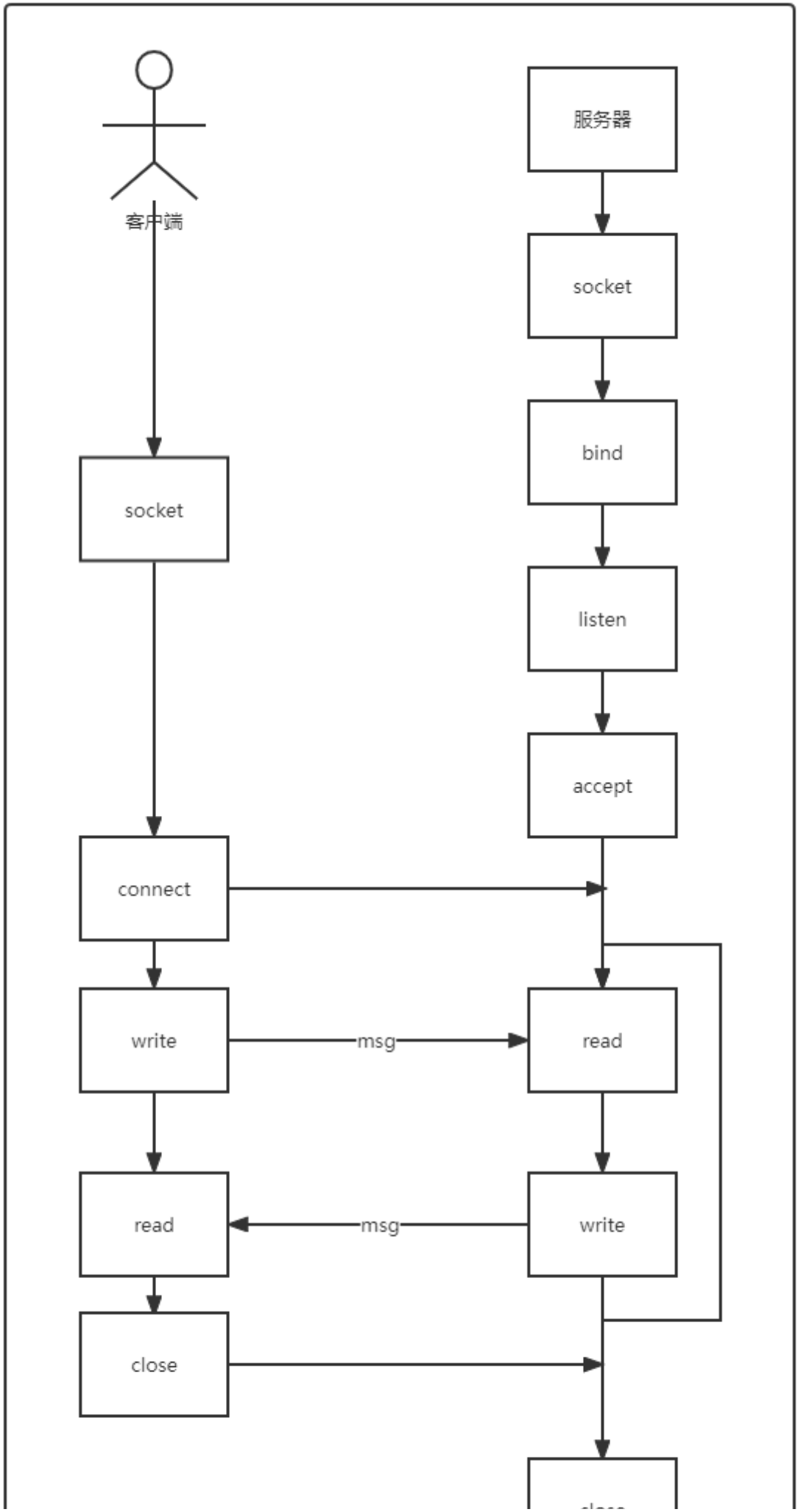
1. 使用`socket()`函数创建一个套接字
2. 使用`bind()`函数绑定 IP 和端口号（例如：`bing(('127.0.0.1', 9999))`）
3. 使用`listen()`函数使套接字变为可被动连接
4. 使用`accept()`函数等待客户端的连接
5. 使用`recv()`和`send()`函数接收发送数据
6. 使用`close()`关闭套接字

客户端：

1. 使用`socket()`函数创建一个套接字
2. 使用`connect()`函数连接目标服务器（例如：`connect(('127.0.0.1', 9999))`）
3. 使用`recv()`和`send()`函数接收发送数据
4. 使用`close()`关闭套接字

11.2.4. c/s 结合

TCP 客户端和服务器的通信模型


客户端
服务器
socket
bind
listen
accept
socket
connect
write
msg
read
read
msg
write
close

实现简易聊天窗口：

当接收到`bye`信息聊天结束，窗口关闭。

服务器：

```
import socket

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

# 2 绑定IP端口号
s.bind(('127.0.0.1', 9999))

# 3 监听连接
s.listen(1)
print('等待连接中...')

# 4 阻塞等待连接中
sock, addr = s.accept()
print(f'同{addr}已建立连接')

# decode()解码
```

```
info = sock.recv(1024).decode()

while info != 'bye':
    if info:
        print('接收到的内容：' + info)

    send_info = input('输入发送的内容：')
    sock.send(send_info.encode())
    if send_info == 'bye':
        print('通话结束。')
        break
    info = sock.recv(1024).decode()
sock.close()
s.close()
```

客户端：

```
import socket

s = socket.socket(socket.AF_INET,
socket.SOCK_STREAM)

s.connect(('127.0.0.1', 9999))
print('已同(\'127.0.0.1\', 9999)建立连接')
info = ''
while info != 'bye':
    if info:
        print('接收到消息：' + info)
```

```
    send_info = input('输入发送内容：')
    s.send(send_info.encode())
    if send_info == 'bye':
        print('通话结束。')
        break
    info = s.recv(1024).decode()
s.close()
```

11.3. UDP 编程

TCP 是建立可靠连接，并且通信双方都可以以流的形式发送数据。相对 TCP，**UDP 则是面向无连接的协议。**

UDP 是面向消息的协议，如果通信时不需要建立连接，那么数据传输自然是不可靠的，UDP 协议一般用于多点通信和实时的数据业务，如下：

- 语音广播
- 视频
- 聊天软件
- 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）

和TCP类似，使用UDP的通信双方也分为客户端和服务器。

11.3.1. UDP 服务器

服务器首先需要绑定端口：

```
import socket
s = socket.socket(socket.AF_INET,
socket.SOCK_DGRAM)
# 绑定端口:
s.bind(('127.0.0.1', 9999))
```

创建 Socket 时，`SOCK_DGRAM`指定了这个 Socket 的类型是 UDP。绑定端口和 TCP 一样，但是不需要调用`listen()`方法，而是直接接收来自任何客户端的数据：

```
print('Bind UDP on 9999...')
while True:
    # 接收数据:
    data, addr = s.recvfrom(1024)
    print('Received from %s:%s.' % addr)
    s.sendto(b'Hello, %s!' % data, addr)
```

`recvfrom()`方法返回数据和客户端的地址与端口，这样，服务器收到数据后，直接调用`sendto()`就可以把数据用 UDP 发给客户端。

11.3.2. UDP 客户端

客户端使用 UDP 时，首先仍然创建基于 UDP 的 Socket，然后，不需要调用`connect()`，直接通过`sendto()`给服务器发数据：

```
import socket
s = socket.socket(socket.AF_INET,
socket.SOCK_DGRAM)
for data in [b'Michael', b'Tracy', b'Sarah']:
    # 发送数据：
    s.sendto(data, ('127.0.0.1', 9999))
    # 接收数据：
    print(s.recv(1024).decode('utf-8'))
s.close()
```

从服务器接收数据仍然调用`recv()`方法。

11.3.3. 小结

服务器

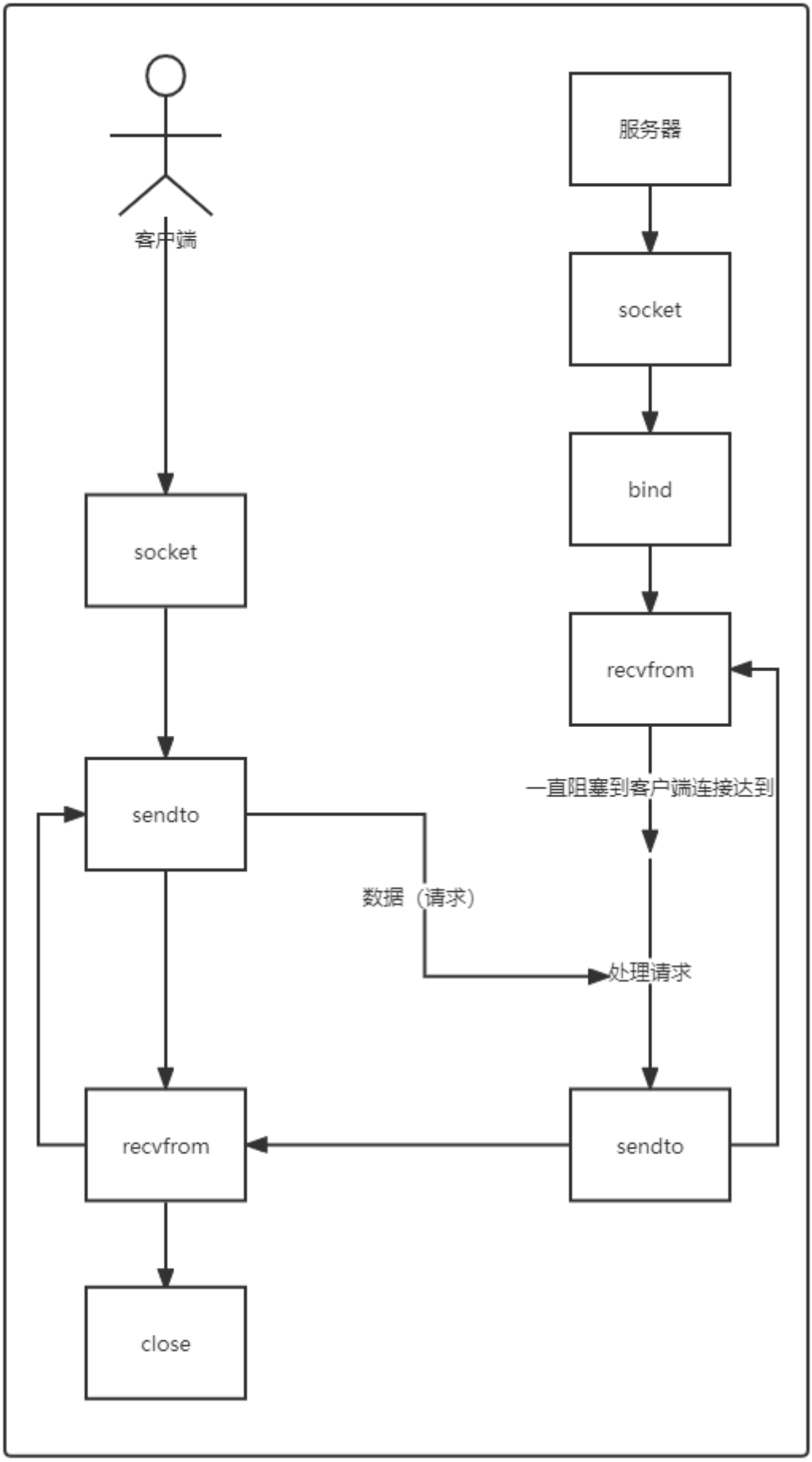
1. 使用`socket()`函数创建一个套接字
2. 使用`bind()`函数绑定 IP 和端口号
3. 使用`recvfrom()`和`sendto()`函数接收和发送数据
4. 使用`close()`关闭套接字

客户端

1. 使用`socket()`函数创建一个套接字
2. 使用`recvfrom()`和`sendto()`函数接收和发送数据
3. 使用`close()`关闭套接字

11.2.4. c/s 结合

UDP 客户端和服务器的通信模型


客户端
服务器
socket
socket
bind
recvfrom
一直阻塞到客户端连接达到
sendto
数据（请求）
处理请求
recvfrom
sendto
close

在客户端输入要转换的摄氏温度，然后发送到服务器，服务器根据转换公式，将摄氏温度转换为华氏温度，发送到客户端显示。

华氏温度 = 摄氏温度*1.8+32

服务器：

```
import socket

s = socket.socket(socket.AF_INET,
socket.SOCK_DGRAM)
s.bind(('127.0.0.1', 9999))
data, addr = s.recvfrom(1024)
print("从{}收到数据".format(addr))
# 转换公式
data = round(float(data) * 1.8 + 32, 2)
send_data = '转换为华氏温度为: ' + str(data)
s.sendto(send_data.encode(), addr)
s.close()
```

客户端：

```
import socket

s = socket.socket(socket.AF_INET,
socket.SOCK_DGRAM)
data = input('请输入要转换的温度（单位：摄氏温
度）：')
s.sendto(data.encode(), ('127.0.0.1', 9999))
print(s.recv(1024).decode())
s.close()
```