第 3 章

套接字编程

本章学习目标

（1）熟悉计算机网络的基本概念。
（2）理解常见网络协议的工作原理和用途。
（3）了解 IP、ICMP、TCP、UDP 的数据包格式与各字段含义。
（4）熟练掌握标准库 socket、socketserver 中常用对象的用法。
（5）熟练掌握套接字对象方法的语法和功能。
（6）熟练掌握 TCP 和 UDP 编程的基本思路。
（7）熟练掌握多线程 / 多进程编程技术在套接字编程中的应用。
（8）理解 TCP 断包与粘包的原理。
（9）理解 TCP 连接保活机制的原理和应用。
（10）理解 TCP 连接限速的原理与实现。
（11）理解代理服务器和端口映射技术的原理和实现。
（12）了解使用套接字编程读取网页源代码的原理和实现。
（13）理解 UDP 广播的原理和应用。
（14）理解 UDP 数据包乱序和丢失的原因以及在应用层实现可靠传输的原理。
（15）理解 UDP 套接字连接操作的作用。
（16）理解屏幕广播软件的原理与实现。
（17）了解多路复用技术的原理与标准库 selectors 的使用。
（18）了解网络嗅探器的原理和实现。
（19）了解网络抓包软件 wireshark 和 Python 扩展库 scapy 的使用。
（20）了解传输层安全协议 TLS 的原理和使用。
（21）了解端口扫描器的原理和实现。
（22）了解扩展库 psutil 在网络管理方面的应用。

3.1 计算机网络基础知识

了解计算机网络的原理和相关协议是编写网络应用程序的重要基础。本节简单介绍计算机网络编程用到的一些基础知识和基本概念，后面几节中再根据案例需要进行适当展开，更详细的计算机网络原理和协议细节请参考相关图书。

（1）网络体系结构。目前主流的网络体系结构是 ISO/OSI 参考模型和 TCP/IP 族。这两种体系结构都采用了分层设计和实现的方式，ISO/OSI 七层参考模型从上而下划分为应用层、表示层、会话层、传输层、网络层、数据链路层和物理层，TCP/IP 四层协议族将网络划分为应用层、传输层、网络层和网络接口层。网络体系结构分层设计的好处是，各层可以独立设计和实现，只要保证相邻层之间的接口和调用规范不变，就可以方便、灵活地改变各层的内部实现以进行优化或完成其他需求，不影响其他层的实现。

（2）网络协议。网络协议是计算机网络中为了进行数据交换而制定的规则、标准或约定的集合。语法、语义和时序是网络协议的三要素。可以这么理解，语义用来说明要做什么，语法用来保证准确无歧义地传输指令和数据，时序规定了各种事件出现的顺序。

① 语法：语法规定了用户数据与控制信息的结构与格式。

② 语义：语义用来解释控制信息每个部分的含义，规定了需要发出何种控制信息，以及需要完成的动作和做出什么样的响应。

③ 时序：时序是对事件发生顺序的详细说明，也可称为“同步”。

（3）应用层协议。应用层协议直接与最终用户进行交互，用来确定运行在不同终端系统上的应用进程之间如何交换数据以及数据的具体含义。下面列出了几种常见的应用层协议。

① DNS：域名系统（Domain Name System），用来实现域名与 IP 地址的转换，运行于 UDP 之上，默认使用 53 端口。

② FTP：文件传输协议（File Transfer Protocol），可以通过网络在不同平台之间实现文件传输，是一种基于 TCP 的明文传输协议，默认使用 20 和 21 端口。

③ HTTP：超文本传输协议（HyperText Transfer Protocol），是万维网能够工作的重要协议，运行于 TCP 之上，默认使用 80 端口。

④ HTTPS：超文本传输安全协议（Hyper Text Transfer Protocol Secure），在传输层之上增加了一个安全加密的夹层 SSL（Secure Socket Layer），默认使用 443 端口，既可以使用 TCP 也可以使用 UDP 来实现，开销比 HTTP 大不少。

⑤ SMTP：简单邮件传输协议（Simple Mail Transfer Protocol），建立在 TCP 的基础上，使用明文传递邮件和发送命令，默认使用 25 端口。SMTP-over-SSL（基于 SSL 的 SMTP）默认使用 465 端口。

Pwjzw.indd 104 2021.7.20 1:37:06 PM

⑥ TELNET：远程登录协议，运行于 TCP 之上，默认使用 23 端口。

⑦ DHCP：动态主机配置协议（Dynamic Host Configuration Protocol），用于自动分配和获取 IP 地址，服务端默认使用 67 端口，客户端默认使用 68 端口。

⑧ POP3：邮局协议（Post Office Protocol Version 3），主要用于支持使用客户端远程管理在服务器上的电子邮件，允许电子邮件客户端下载服务器上的邮件，但是在客户端的操作（例如移动邮件、标记已读）不会反馈到服务器上。POP3 默认使用 110 端口，POP3-over-SSL（基于 SSL 的 POP3）默认使用 995 端口。

⑨ IMAP：互联网邮件访问协议（Internet Mail Access Protocol），从邮件服务器上获取邮件信息和下载邮件，客户端的操作直接反馈到服务器上，功能也比 POP3 要强大一些。IMAP 默认使用 143 端口，IMAP-over-SSL（基于 SSL 的 IMAP）默认使用 993 端口。

⑩ SNMP：简单网络管理协议（Simple Network Management Protocol），是一系列网络管理规范的集合，默认使用 161 和 163 端口。

（4）传输层协议。在传输层主要运行传输控制协议（Transmission Control Protocol，TCP）和用户数据报协议（User Datagram Protocol，UDP）两个协议，其中 TCP 是面向连接的、具有质量保证的可靠传输协议，但开销较大；UDP 是尽最大能力传输的无连接协议，开销小，常用于视频在线点播（Video On Demand，VOD）之类的应用。TCP 和 UDP 本身并没有优劣之分，仅仅是适用场合有所不同。在传输层，使用端口号来唯一标识和区分同一台计算机上运行的多个应用层进程，每当创建一个网络应用进程时系统就会为其分配一个端口号，是实现网络上端到端通信的重要基础。例如，SQL Server 默认使用 1433 端口，远程桌面连接默认使用 3389 端口，MySQL 默认使用 3306 端口，MongoDB 默认使用 27017 端口，大多数情况下 IRC 服务器默认使用 6667 端口，Oracle 使用 1521、1158、8080、210 等几个端口。

（5）IP 地址。IP 运行于网络层，是网络互连的重要基础。IP 地址（32 位或 128 位二进制数）用来标识网络上的主机，在公开网络上或同一个局域网内部，每台主机都必须使用不同的 IP 地址。由于网络地址转换（Network Address Translation，NAT）和代理服务器等技术的广泛应用，不同内网中的主机可以使用相同的 IP 地址并且互不影响。

（6）IP 地址用来标识网络上一台主机，端口号（port number）用来标识主机上联网的进程，IP 地址和端口号的组合称为套接字（Socket）。套接字是网络程序设计的重要基础，也是网络爬虫程序、网站服务器与客户端、电子邮件服务器与客户端、网络管理或其他网络应用系统依赖的底层重要技术之一。

（7）MAC 地址：介质访问控制地址（Media Access Control Address），也称网卡物理地址，是一个 48 位二进制数，用来标识不同的网卡。本机的 IP 地址和 MAC 地址可以在命令提示符窗口中使用 ipconfig/all 命令查看，如图 3-1 所示。

Pwjzw.indd 105　　2021.7.20 1:37:10 PM

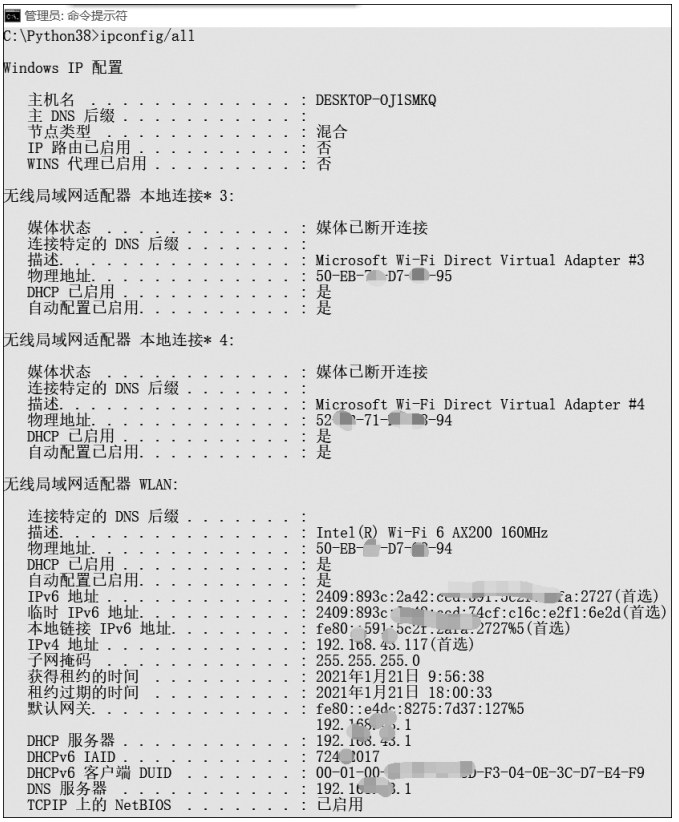

图 3-1　使用 ipconfig/all 命令查看本机 IP 地址和网卡物理地址

3.2　socket模块简介

3.2.1　socket模块常用函数

Python 标准库 socket 提供了套接字编程所需要的所有对象，表 3-1 列出了常用的一部分。需要注意的是，socket 模块最终会把相关的调用转换为底层操作系统的 API，所以 socket 提供的功能是依赖于操作系统的，并不是每个功能都通用于所有操作系统，本书重点介绍适用于 Windows 平台的用法，其中大部分功能也适用于其他操作系统。

表 3-1　socket 模块常用对象

对　　象	说　　明
create_connection(address, timeout=<object object at 0x000001A69B1E2E80>, source_address=None)	创建 TCP 连接，返回套接字对象

Pwjzw.indd 106　　2021.7.20 1:37:14 PM

续表

对　　象	说　　明
create_server(address, *, family=<AddressFamily.AF_INET: 2>, backlog=None, reuse_port=False, dualstack_ipv6=False)	创建 TCP 服务端套接字
getdefaulttimeout()	返回创建新套接字时使用的默认超时时间（单位为秒），值为 None 时表示没有超时时间限制
gethostname()	返回计算机名
gethostbyname(host)	根据计算机名获取并返回对应的 IP 地址
gethostbyaddr(host)	根据 IP 地址或计算机名，返回包含计算机名、别名和 IP 地址列表的元组
gethostbyname_ex(host)	根据计算机名获取并返回包含计算机名、别名和所有 IP 地址的元组 (name, aliaslist, addresslist)
getservbyname(servicename [, protocolname])	返回指定服务和协议对应的端口号，如果没有对应的端口号会抛出异常
getservbyport(port[, protocolname])	返回指定端口号和协议对应的服务名称，如果没有对应的服务会抛出异常
if_nameindex()	返回包含网络接口信息的列表，Python 3.8 新增
inet_aton(string)	把圆点分隔数字形式的 IPv4 地址字符串转换为二进制形式，例如，inet_aton('127.0.0.1') 得到 b'\x7f\x00\x00\x01'，等价于 bytes(map(int, '127.0.0.1'.split('.')))
inet_ntoa(packed_ip)	把 32 位二进制格式的 IPv4 地址转换为圆点分隔数字的字符串格式，例如，inet_ntoa(b'\x7f\x00\x00\x01') 得到 '127.0.0.1',等价于 '.'.join(map(str, tuple(b'\x7f\x00\x00\x01')))
setdefaulttimeout(timeout)	设置创建新套接字时使用的默认超时时间（单位为秒），参数 timeout 的值为 None 时表示没有超时时间限制
socket(family=-1, type=-1, proto=-1, fileno=None)	创建一个套接字对象，参数 family=socket.AF_INET 表示使用 IPv4 地址，family=socket.AF_INET6 时表示使用 IPv6 地址；参数 type=socket.SOCK_STREAM 表示使用 TCP，type=socket.SOCK_DGRAM 表示使用 UDP。套接字对象支持 with 关键字
socketpair([family[, type [, proto]]])	返回一对已连接的套接字

Pwjzw.indd 107　　2021.7.20 1:37:18 PM

下面的代码演示了 socket 模块中部分函数的用法。

```
>>> import socket
>>> socket.gethostname()                          # 查看本机主机名
'DESKTOP-OJ1SMKQ'
>>> socket.gethostbyname(socket.gethostname())  # 查看本机IP地址
'169.254.162.3'
>>> socket.gethostbyname_ex(socket.gethostname())
('DESKTOP-OJ1SMKQ', [], ['169.254.162.3', '192.168.8.141'])
>>> import urllib.request
>>> urllib.request.thishost()                     # 查看本机所有IP地址的另一种方式
('169.254.162.3', '192.168.43.117')
>>> socket.getservbyname('http', 'tcp')          # 查看服务占用的端口号
80
>>> socket.getservbyname('http', 'udp')          # HTTP不能使用UDP
                                                  # 略去了详细错误
OSError: service/proto not found
>>> socket.getservbyname('pop3', 'tcp')
110
>>> socket.getservbyname('pop3', 'udp')          # POP3不能使用UDP
                                                  # 略去了详细错误
OSError: service/proto not found
>>> socket.getservbyname('https', 'udp')         # HTTPS可以使用UDP或TCP
443
>>> socket.getservbyname('https', 'tcp')
443
>>> socket.getservbyport(80, 'tcp')              # 返回端口号对应的应用层协议
'http'
>>> socket.getservbyport(25, 'tcp')
'smtp'
>>> socket.inet_aton('192.168.8.141')            # 把IP地址打包为字节串
b'\xc0\xa8\x08\x8d'
>>> socket.inet_ntoa(b'\xc0\xa8\x08\x8d')        # 把字节串解包为IP地址
'192.168.8.141'
```

例 3-1　编写程序，获取并输出所有熟知端口、协议和服务名称之间的对应关系。

```
import socket

# 用来存放结果，形式为{'pop3s': [(995, 'tcp'), (995, 'udp')]}的字典
# 同一个服务可能既可以使用TCP，也可以使用UDP
services = {}
# 遍历[1, 1024]区间的所有端口号
```


Pwjzw.indd 108　　2021.7.20 1:37:22 PM

```
for port in range(1, 1025):
    for protocol in ('tcp', 'udp'):
        try:
            service = socket.getservbyport(port, protocol)
            t = services.get(service, [])
            t.append((port, protocol))
            services[service] = t
        except:
            pass

print(services)
```

作为一种优化写法，下面的代码使用了标准库 itertools 中的笛卡儿积函数 product()，减少了一层循环，代码更简洁一些。

```
import socket
from itertools import product

services = {}
# 遍历[1, 1024]区间的所有端口号和TCP、UDP的组合
for port, protocol in product(range(1,1025), ('tcp','udp')):
    try:
        service = socket.getservbyport(port, protocol)
        t = services.get(service, [])
        t.append((port, protocol))
        services[service] = t
    except:
        pass

print(services)
```

例 3-2　编写程序，获取本机所在局域网（假设为 C 类网络）内所有机器的 IP 地址与网卡的 MAC 地址。使用操作系统命令 arp 获取本地缓存的 ARP 表并生成文本文件，然后再从文件中读取和解析局域网内 IP 地址与 MAC 地址的对应关系。

例 3-2 讲解

```
import os
from socket import gethostbyname_ex, gethostname

# 获取本机所有IP地址列表
hosts = gethostbyname_ex(gethostname())[-1]
# 获取ARP表，写入文本文件
```


Pwjzw.indd 109　　2021.7.20 1:37:26 PM

```
os.system('arp -a > temp.txt')
# 读取和解析ARP表中的信息
with open('temp.txt') as fp:
    for line in fp:
        line = line.split()[:2]
        if not line:
            continue
        for host in hosts:
            if line[0].startswith(host.rsplit('.',1)[0]) and\
               (not line[0].endswith('255')):
                print(':'.join(line))
```

代码生成的本地 ARP 表的临时文件 temp.txt 内容格式如图 3-2 所示，如果需要可以在程序最后增加一行代码调用 os.remove() 函数删除临时文件 temp.txt，请自行修改和测试。

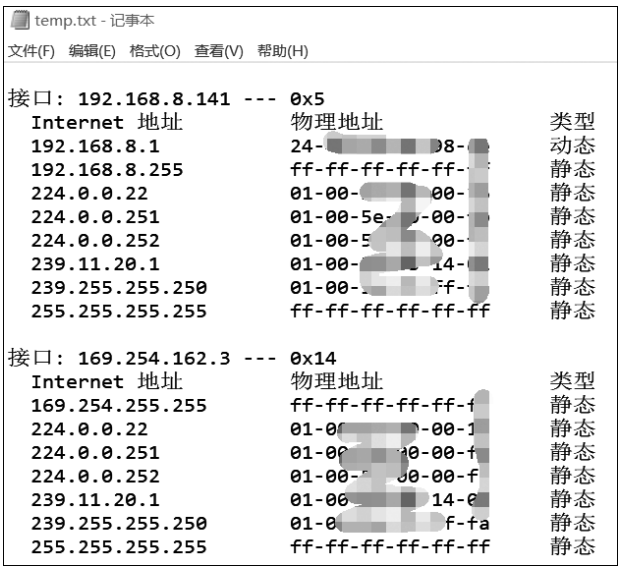

图 3-2 temp.txt 内容格式

如果需要获取本地计算机所有网卡的 MAC 地址，可以参考 3.8 节关于扩展库 psutil 的内容。

例 3-3 为了实现负载均衡或者增加黑客攻击难度，很多域名对应的 IP 地址是会经常变化的。编写程序，监视某个域名对应 IP 地址的变化情况。

```
from time import sleep
from datetime import datetime
from socket import gethostbyname
```


Pwjzw.indd 110 2021.7.20 1:37:29 PM

```
def check_ipAddress(url):
    ipAddresses = ['']
    while True:
        sleep(0.5)                              # 每隔0.5秒查询一次
        ip = gethostbyname(url)                 # 获取IP地址
        if ip != ipAddresses[-1]:               # 和上次获取的IP地址不一样
            ipAddresses.append(ip)
            print(str(datetime.now())[:19]+'===>'+ip)

check_ipAddress(r'www.microsoft.com')
```

部分运行结果如图 3-3 所示。

```
2020-10-12 21:47:48===>222.138.3.87
2020-10-12 22:03:51===>223.119.205.204
2020-10-12 22:04:10===>222.138.3.87
```

图 3-3　某域名对应的 IP 地址变化情况

例 3-4　编写程序，获取本机所在局域网内所有计算机名和 IP 地址。代码假设本机处于 C 类 IP 地址局域网内，遍历局域网内每个可能的 IP 地址并尝试获取对应的计算机名。

```
from socket import gethostname, gethostbyname, gethostbyaddr

# 本机IP地址和网络地址
this_ip = gethostbyname(gethostname())
net_addr = this_ip.rsplit('.', 1)[0]

for i in range(1, 255):
    # 局域网内每个可能的IP地址
    ip = f'{net_addr}.{i}'
    try:
        computer_name = gethostbyaddr(ip)[0]
        print(computer_name, ip, sep=':')
    except:
        print(ip, 'not exists.')
```

3.2.2　套接字对象常用方法

socket 模块中的函数 socket() 函数用来创建套接字对象支持网络通信，表 3-2 中列出了套接字对象的常用方法，本章后面几节陆续介绍相关的用法和案例。

Pwjzw.indd 111　　2021.7.20 1:37:33 PM

表 3-2　套接字对象的常用方法

方　法	说　明
accept()	接受一个客户端的连接，返回元组 (conn, address)，其中 conn 是可以实际用于收发数据的新套接字，address 是对方套接字地址，形式为 (host, port)
bind(address)	把套接字绑定到本地地址，参数 address 的形式为元组 (host, port)
close()	通知操作系统，把输出缓冲区中剩余的数据传输完成之后关闭套接字
connect(address)	连接 address 指定的套接字，address 形式为 (host, port)
getblocking()	查看套接字是否处于阻塞模式，对于非阻塞模式的套接字，调用 accept() 和 recv() 方法时如果没有数据会抛出异常。新创建的套接字默认为阻塞模式
getpeername()	返回套接字对象连接的另一端地址，可以用于获取远程套接字的 IP 地址和端口号
getsockname()	返回套接字对象的本地地址，可以用于获取系统随机分配给本地套接字的端口号
getsockopt(level, option [, buffersize])	查看套接字选项
gettimeout()	返回套接字的超时时间
ioctl(cmd, option)	调用 Windows 的 WSAIoctl 系统调用，目前只支持 SIO_RCVALL、SIO_KEEPALIVE_VALS 和 SIO_LOOPBACK_FAST_PATH，其中 SIO_KEEPALIVE_VALS 值的格式为 (onoff, timeout, interval)
listen([backlog])	声明自己为服务器端的监听套接字（listening socket）或被动套接字（passive socket），开始监听并准备好接受客户端的连接请求，参数 backlog 表示能够同时连接的客户端数量。监听套接字不能用于发送或接收任何数据，也不表示任何实际的网络会话。套接字对 listen() 方法的调用是不可逆的，无法从监听套接字变为用于数据收发的普通套接字
recv(buffersize[, flags])	从套接字中读取并返回最多 buffersize 字节，即使接收缓冲区内有足够多的数据也不能保证成功接收到 buffersize 字节，如果对方已关闭并且已读取完缓冲区内所有数据，recv() 方法返回空字节串
recvfrom(buffersize[, flags])	从套接字中读取最多 buffersize 字节，返回包含字节串数据和对方地址的元组 (data, address info)
recvfrom_into(buffer [, nbytes[, flags]])	从套接字读取最多 nbytes 字节直接写入缓冲区 buffer，返回实际接收并写入缓冲区的字节串长度以及发送方地址

Pwjzw.indd 112　　2021.7.20 1:37:37 PM

续表

方　法	说　明
recv_into(buffer, [nbytes[, flags]])	从套接字读取最多 nbtyes 字节直接写入缓冲区，返回实际接收并写入缓冲区的字节串长度
send(data[, flags])	向已连接的远程套接字发送字节串 data，返回实际发送的字节串长度（小于或等于 len(data)）
sendall(data[, flags])	向已连接的远程套接字发送字节串 data，自动重复调用 send() 方法，确保 data 全部发送完成
sendto(data[, flags], address)	向参数 address 指定的套接字发送字节串 data
setblocking(flag)	设置套接字为阻塞模式（flag=True）或非阻塞模式（flag=False）
settimeout(timeout)	设置套接字连接、读写等操作的超时时间为 timeout（要求为非负实数）秒，参数 timeout=0 时表示设置套接字为非阻塞模式，参数 timeout=None 时表示设置套接字为阻塞模式
setsockopt(level, option, value: int) setsockopt(level, option, value: buffer) setsockopt(level, option, None, optlen: int)	设置套接字选项

3.3　TCP编程案例实战

TCP 和 UDP 是网络体系结构的传输层运行的两大重要协议，二者各有特点，没有绝对的优劣之分，只是使用场合不同。其中，TCP 适用于对效率要求相对低但对准确性和可靠性要求高的场合，例如文件传输、电子邮件等；UDP 适用于对效率要求相对高但对准确性要求低的场合，例如视频在线点播、网络语音通话等，尤其适合多播和广播的应用。

使用标准库 socket 中的函数 socket() 创建套接字时指定参数 type=socket.SOCK_STREAM 表示使用 TCP，编写基于 TCP 的网络应用程序时常用的套接字对象方法主要有 connect()、send()、sendall()、recv()、recv_into()、bind()、listen()、accept()、close()，具体功能和参数含义参考表 3-2，服务端和客户端调用各方法的时间线如图 3-4 所示。

本节接下来通过几个案例演示和讲解套接字编程标准库 socket，以及多线程编程标准库 threading 和多进程编程标准库 multiprocessing 在 TCP 编程方面的应用。3.4 节介绍和演示 UDP 编程的应用，关于 ZeroMQ 编程的资料和案例，可以关注公众号“Python 小屋”，并发送消息 ZeroMQ 进行学习。

Pwjzw.indd 113　2021.7.20 1:37:41 PM

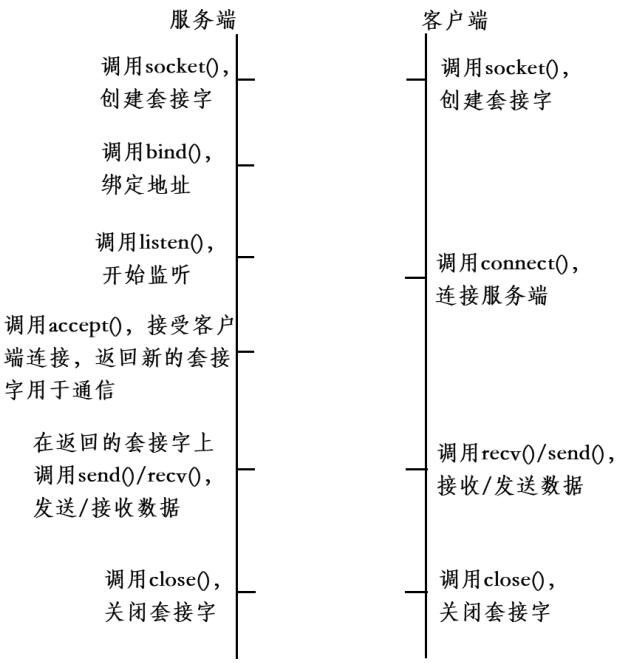

图 3-4　TCP 服务端和客户端通信示意图

例 3-5　编写 TCP 通信程序模拟机器人自动聊天软件，服务端提前建立好字典，然后根据接收到的内容查询字典并自动回复。

例 3-5 讲解

1. 服务端代码（单客户端版本）

```
import socket
from sys import exit
from struct import pack, unpack
from os.path import commonprefix

# 缓冲区大小，或者说一次能够接收的最大字节串长度
BUFFER_SIZE = 9012

words = {'how are you?': 'Fine,thank you.',
         'how old are you?': '38',
         'what is your name?': 'Dong FuGuo',
         "what's your name?": 'Dong FuGuo',
         'where do you work?': 'University',
         'bye': 'Bye'}

# 空字符串表示本地所有IP地址
# 如果需要绑定到本地特定的IP地址，可以明确指定，例如'192.168.9.1'
HOST = ''
PORT = 50007
```


Pwjzw.indd 114　2021.7.20 1:37:45 PM

```
# 创建TCP套接字，绑定socket地址
sock_server = socket.socket(socket.AF_INET, socket.SOCK_STREAM)
sock_server.bind((HOST, PORT))
# 声明自己为服务端套接字，开始监听，准备接受一个客户端连接
sock_server.listen(1)
print('Listening on port:', PORT)

# 阻塞，成功接受一个客户端连接请求之后返回新的套接字和对方地址
try:
    conn, addr = sock_server.accept()
except:
    # 接受客户端连接失败，服务器故障，直接退出
    exit()

print('Connected by', addr)
# 开始聊天，使用新套接字收发信息
while True:
    # 接收一个整数打包后的字节串，表示对方本次发送的实际字节串长度
    int_bytes = b''
    # 在struct序列化规则中，整数被打包为长度为4的字节串
    rest = 4
    # 在高并发网络服务器中，无法保证能够一次接收完4字节，使用循环更可靠
    # 使用TCP通信时，必须保证接收方恰好收完发送方的数据，不能多，也不能少
    while rest > 0:
        # 接收数据时自动分配缓冲区
        temp = conn.recv(rest)
        # 收到空字节串，表示对方套接字已关闭
        if not temp:
            break
        int_bytes = int_bytes + temp
        rest = rest - len(temp)
    # 前面的while循环没有接收到数据或者没有收够4字节
    # 表示对方已结束通信或者网络故障
    if rest > 0:
        break

    # rest表示接下来需要接收的字节串长度，unpack()的结果是一个元组
    rest = unpack('i', int_bytes)[0]
    data = b''
    while rest > 0:
        # 要接收的字节串长度可能非常大，限制一次最多接收BUFFER_SIZE字节
        temp = conn.recv(min(rest, BUFFER_SIZE))
        data = data + temp
        rest = rest - len(temp)
```


Pwjzw.indd 115　　2021.7.20 1:37:49 PM

```
        # 接收数据不完整，套接字可能损坏
        if rest > 0:
            break

        # 删除字符串中可能存在的连续多个空格
        data = ' '.join(data.decode().split())
        print('Received message:', data)
        # 尽量猜测对方要表达的真正意思
        m = 0
        key = ''
        for k in words.keys():
            # 与某个“键”有超过70%的共同前缀，认为对方就是想问这个问题
            if len(commonprefix([k, data])) > len(k)*0.7:
                key = k
                break
            # 使用选择法，选择一个重合度较高（也就是共同单词最多）的“键”
            length = len(set(data.split())&set(k.split()))
            if length > m:
                m = length
                key = k
        # 选择合适的信息进行回复
        reply = words.get(key, 'Sorry.').encode()
        # 发送数据时自动确定缓冲区长度
        conn.sendall(pack('i', len(reply)) + reply)

    conn.close()
    sock_server.close()
```

2. 客户端代码

```
import socket
from sys import exit
from struct import pack, unpack

# 服务端主机IP地址和端口号
# 如果服务端和客户端不在同一台计算机，需要自己修改变量HOST的值
HOST = '127.0.0.1'
PORT = 50007

sock = socket.socket(socket.AF_INET, socket.SOCK_STREAM)
# 设置超时时间，避免服务端不存在时客户端长时间等待或GUI界面无响应
sock.settimeout(0.3)
try:
```


Pwjzw.indd 116 2021.7.20 1:37:53 PM

```
    # 连接服务器，成功后设置当前套接字为阻塞模式
    sock.connect((HOST, PORT))
    sock.settimeout(None)
except Exception as e:
    print('Server not found or not open')
    exit()

while True:
    msg = input('Input the content you want to send:').encode()
    # 发送数据
    sock.sendall(pack('i', len(msg))+msg)
    # 从服务端接收数据
    # 客户端的实现可以比服务端简单一些，不需要循环接收来保证数据完整
    length = unpack('i', sock.recv(4))[0]
    data = sock.recv(length).decode()
    print('Received:', data)
    if msg.lower() == b'bye':
        break
# 关闭连接
sock.close()
```

打开一个命令提示符窗口运行服务端程序，再打开一个命令提示符窗口运行客户端程序，运行效果如图 3-5 所示。

图 3-5　机器人聊天程序运行效果

Pwjzw.indd 117　　2021.7.20 1:37:57 PM

上面的程序中，服务端在同一时刻只能接收和回复一个客户端的聊天信息，如果想让服务端能够同时和多个客户端聊天，需要结合第 2 章的多线程编程技术，为每个客户端创建一个单独的线程为其服务。服务端代码改写如下，客户端代码不需要修改，请自行运行程序观察运行结果。

3. 服务端代码（多客户端版本）

```
import socket
from threading import Thread
from struct import pack, unpack
from os.path import commonprefix

BUFFER_SIZE = 9012

def every_client(conn, addr):
    # 开始聊天
    while True:
        # 接收一个整数的字节串，表示对方本次发送的实际字节串长度
        int_bytes = b''
        rest = 4
        # 在高并发网络中，无法保证能够一次接收完4字节，使用循环更可靠
        while rest > 0:
            temp = conn.recv(rest)
            # 收到空字节串，表示对方套接字已关闭
            if not temp:
                break
            int_bytes = int_bytes + temp
            rest = rest - len(temp)
        # 前面的while循环没有接收到数据或者没有收够4字节
        # 表示对方已结束通信或者网络故障
        if rest > 0:
            break

        # rest表示接下来需要接收的字节串长度，unpack()的结果是一个元组
        rest = unpack('i', int_bytes)[0]
        data = b''
        while rest > 0:
            # 要接收的字节串可能非常长，限制一次最多接收BUFFER_SIZE字节
            temp = conn.recv(min(rest, BUFFER_SIZE))
            data = data + temp
            rest = rest - len(temp)
        # 接收数据不完整，套接字可能损坏
        if rest > 0:
            break
```


Pwjzw.indd 118 2021.7.20 1:38:01 PM

```
            # 删除字符串中可能存在的连续多个空格
            data = ' '.join(data.decode().split())
            print('Received message:', data)
            # 尽量猜测对方要表达的真正意思
            m = 0
            key = ''
            for k in words.keys():
                # 与某个“键”有超过70%的共同前缀，认为对方就是想问这个问题
                if len(commonprefix([k, data])) > len(k)*0.7:
                    key = k
                    break
                # 使用选择法，选择一个重合度较高的“键”
                length = len(set(data.split())&set(k.split()))
                if length > m:
                    m = length
                    key = k
            # 选择合适的信息进行回复
            reply = words.get(key, 'Sorry.').encode()
            conn.sendall(pack('i', len(reply)) + reply)

        conn.close()
        print(f'Client {addr} has left.')

words = {'how are you?': 'Fine,thank you.',
         'how old are you?': '38',
         'what is your name?': 'Dong FuGuo',
         "what's your name?": 'Dong FuGuo',
         'where do you work?': 'University',
         'bye': 'Bye'}

HOST = ''
PORT = 50007
sock_server = socket.socket(socket.AF_INET, socket.SOCK_STREAM)
# 绑定socket
sock_server.bind((HOST, PORT))
# 开始监听套接字，准备接受客户端连接，最多可以同时和50个客户端通信
sock_server.listen(50)
print('Listening on port:', PORT)
while True:
    try:
        conn, addr = sock_server.accept()
    except:
        break
```


Pwjzw.indd 119　　2021.7.20 1:38:05 PM

```
        print('Connected by', addr)
        # 为每个客户端连接创建单独的线程为其服务
        Thread(target=every_client, args=(conn,addr)).start()

    sock_server.close()
```

例 3-6　一般来说，客户端连接服务端之后每次通信只发送少量数据，完成会话之后立刻断开连接释放资源，需要时再次发起连接请求，这样可以减轻服务端压力。但建立连接和释放连接本身也是需要时间的，如果频繁创建连接和释放连接反而会浪费服务器资源，所以有些场合中需要长时间保持连接。默认情况下，如果对方长时间没有收发数据，TCP 连接会自动断开，以免服务端长期存在半开放连接浪费资源。如果需要长时间保持，需要显式设置套接字为长连接模式。编写程序，使用 TCP 进行通信，并且使得 TCP 连接能够长时间保持存活。

例 3-6 讲解

1. 服务端代码

```
import socket
from struct import unpack

sockServer = socket.socket(socket.AF_INET, socket.SOCK_STREAM)
sockServer.bind(('', 6666))
sockServer.listen(1)

conn, addr = sockServer.accept()
# 为客户端套接字开启长连接
# 保活设置只需要在一端启用就可以，不需要在服务端和客户端都设置
# 可以注释掉下面两行代码再运行，对比运行结果，理解保活机制的作用
conn.setsockopt(socket.SOL_SOCKET, socket.SO_KEEPALIVE, True)
conn.ioctl(socket.SIO_KEEPALIVE_VALS,
           (1,          # 开启保持存活机制
            60*1000,    # 60秒后如果对方还没有反应，开始探测连接是否存在
            30*1000)    # 30秒探测一次，默认探测10次，失败则断开
           )
while True:
    # 这里没有考虑高并发网络服务器，假设一次可以接收完数据
    # 接收4字节，解包为整数，表示对方要发送的字节串长度
    data_length = conn.recv(4)
    if not data_length:
        conn.close()
        break
    data_length = unpack('i', data_length)[0]
    data = conn.recv(data_length).decode()
    print(data)
```


Pwjzw.indd 120　　2021.7.20 1:38:09 PM

2. 客户端代码

```
import socket
from time import sleep
from struct import pack
from datetime import datetime

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

try:
    # 实际运行时建议使用两台计算机进行测试
    # 一台计算机运行服务端，另一台计算机运行客户端
    # 并把下面代码中的'127.0.0.1'修改为服务端计算机的IP地址
    sockClient.connect(('127.0.0.1', 6666))
except:
    print('服务器不存在。')
    exit()

for i in range(5):
    msg = str(datetime.now())[:19]
    print(msg)
    msg = msg.encode()
    sockClient.sendall(pack('i', len(msg)))
    sockClient.sendall(msg)
    # 每隔6分钟发送一次数据
    sleep(360)

sockClient.close()
```

使用联网的两台计算机测试程序，一台计算机运行服务端程序，另一台计算机运行客户端程序，并把客户端程序中的本地回环地址 '127.0.0.1' 修改为服务端程序所在计算机的真实 IP 地址，运行结果如图 3-6 所示。删除或注释掉服务端开启保活机制的两行代

图 3-6　开启保活机制时的运行过程

Pwjzw.indd 121　　2021.7.20 1:38:13 PM

码之后再运行，结果如图 3-7 所示。可以看出，如果没有开启保活机制，无法保持 TCP 长连接。如果客户端长时间不操作，再次发送数据时会因为连接已断开而失败。

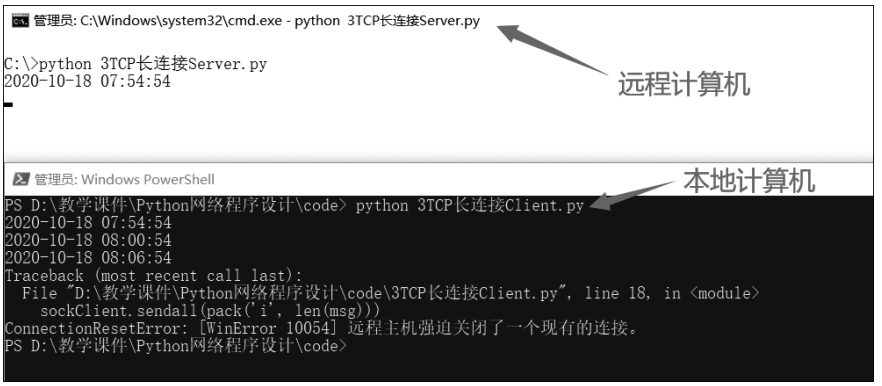

图 3-7　没有开启保活机制时的运行过程

例 3-7　一般来说，存放重要数据的服务器必须要进行强有力的保护，并且只能允许特定的机器进行访问，不能让所有客户端自由访问。可以考虑编写一个代理程序，设置服务器只允许代理程序所在计算机的 IP 地址访问，所有客户端对服务器的访问都由代理程序负责转发。编写程序模拟这个过程，在中间代理程序中使用端口映射技术实现数据转发。

例 3-7 讲解

1. 服务端程序

```
import sys
import socket
import msvcrt
from threading import Thread
from struct import pack, unpack

def replyMessage(conn):
    # 回复消息，原样返回
    while True:
        # 在服务端确保恰好接收到表示整数的4字节
        rest = 4
        data = b''
        while rest > 0:
            received = conn.recv(rest)
            if not received:
                break
            data = data + received
            rest = rest - len(received)
        if rest == 4:
            print('一个会话结束。')
```


Pwjzw.indd 122　2021.7.20 1:38:17 PM