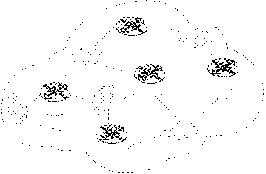
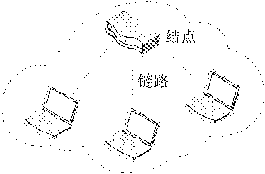
第18章网络编程

当今社会是信息化社会，信息的传播离不开网络，随着计算机与因特网的普及和发展，网 络已渗入社会生活的各行各业，大到操作系统，小到手机应用，都与网络息息相关。因此，网 络编程是Python学习中的重要环节,本章将从计算机网络的基本概念入手,对Python网络编 程相关知识进行讲解。

18ol讦算55述

计算机网络是继电信网络、有线电视网络之后出现的世界级大型网络，在计算机领域中， 网络(network)由若干结点和连接这些结点的链路组成，网络中的结点可以是计算机、交换机、 路由等，如图184 (a)所示，是一个最简单的计算机网络模型。



(a)网络

(b)互联网

图捋』网络与互联网

此外，计算机网络之间可以相互连接，组成更大的网络，如图184 (b )所示,，这种网络 被称为互联网(internet)。互联网中最为知名的是起源于美国的因特网(Intend),因特网是 当今世界上最大的国际性网络。

网络编程的本质是处于网络中的两个设备间进程的数据交换，网络只是网络编程的基础， 在实现网络编程之前，需先掌握一些与网络相关的知识。

18o2 5与傩5梅

在人类社会中，不同地域的人们可能使用不同的语言，若这些人要进行沟通、联络，就需 要有一门共通的语言。这种语言要具备一定的规则，人们根据相同的规则组织语言，方能进行 交流。

同样地，网络中存在多台主机，主机由不同的用户使用，为了保证通信能够顺利进行，且 进行通信的双方可以获取到准确、有效的数据，就应该预先制定一组用于数据传输的规则，这 组规则就是协议（pEocol） o

协议需要预先制定，同时濬为确保网络通信过程中出现的各种事项能“有法可依”，协议 应面面俱到。但网络间的通信需要经历复杂的过程，一段复杂过程中的各项操作会岀现各种各 样的结果，为复杂过程的多种结果制定的协议显然也会是复杂的。为了简化协议，人们考虑按 照通信过程中各项工作的性质，将需要实现的工作进行分层，并为每一层制定各自的协议。为 网络间通信过程所划分的层次通常称为计算机网络的体系结构。

18n2J儘蔡籍料

常见的体系结构有OSI （ Open System Interconnect,开放式系统互连）模型和TCP/IP （Transmission Control Protocol/Intemet Protocol,传输控制协议 / 网际协议）模型。

OSI由国际标准协会ISO制定,共分为七层,由上而下依次为应用层、表示层、会话层、 传输层、网络层、数据链路层和物理层，虽然OS][由ISO制定，但其实用性较差，并未得到广 泛应用。

在OSI诞生时，因特网已实现了全世界的基本覆盖，因此市面上应用最广泛的体系结构为

因特网中使用的TCP/IP体系结构，该结构包含四层，分别为应用层、传输层、网际层和网络

接口层。

此外，在计算机网络中通常以一种包含五 层协议的体系结构来讲解各层之间的功能与联 系,这种体系结构结合OSI和TCP/IP的优点*，* 分为应用层、传输层、网络层、数据链路层和 物理层。

以上三种体系结构中各层的对应关系如 图18-2所示。

五层协议体系结构中各层的功能分别如下：

（1 ）物理层。物理层是计算机体系结构的

| OSI | | TCP/IP | 五层协议 | |
| --- | --- | --- | --- | --- |
| 7应用层 |  |  |  |  |
| *6*表示层 |  | 应用层 |  | 5应用层• |
| 5会话层 |  |  |  |  |
| 4传输层 |  | 传输层 |  | 4传输层 |
| 3网络层 |  | 网际层 |  | 3网络层 |
| 2数据链路层 |  | 网络接口层 |  | 2数据链路层 |
| 1物理层 |  |  | 1物理层 |

图18-2计算机网络体系结构

最底层，它为设备之间的数据传输提供可靠的环境。

（2）数据链路层。数据链路层可简称为链路层，该层将从网络层获取的数据报组装成帧, 在网络结点之间以帧为单位传输数据。

（3）网络层。为分组交换网上的不同主机提供通信服务，在进行通信时，将从传输层获 取的报文段或数据报封装成分组或包，数据或包通常称为数据报。

1. 传输层。为应用进程提供连接服务，实现连接两端进程的会话。
2. 应用层。为应用进程提供服务，定义了应用进程间通信和交互的观则。 .

计算机网络通信基于 TCP/IP ( Transmission Control Protocol/Intemet Protocol,传输控制协 议/网际协议)，TCP/IP实际上是协议簇，它由体系结构中各层使用的多种协议构成，包括 TCP协议、UDP协议、IP协议等等。其中，TCP、UDP协议应用在传输层；IP协议应用在网络层。

们 TCPf»

TCP协议即传输控制协议(Transmission Control Protocol) 9使用该协议的传输层会接收 由应用层传输而来的、使用8位字节表示的数据流，然后根据协议规则，将数据流分为多个报 文段，并为每个报文段添加本层的控制信息，生成传递给网络层的数据单元。

TCP协议是一种面向连接的、可靠的、基于字节流的传输协议，在传递数据之前，收发双 方会先通过一种被称为“三次握手”的协商机制使通信双方建立连接，为数据传输做好准备。 为了防止报文段丢失，TCP会给每个数据段一个序号，接收端应按序号顺序接收数据，若接收 端正常接收到报文段，便会向发送端发送一个确认信息；若发送端在一定的时延后未接收到确 认信息，便假设报文段已丢失，并重新向接收端发送对应报文段。此外，TCP协议中还定义了 一个校验函数，用于检测发送和接收的数据，防止产生数据错误。

通信结束后，通信双方经过“四次握手”关闭连接，因为TCP连接是全双工的(全双工 指的交换机在发送数据的同时也能够接收数据，两者同步进行，这好像我们平时打电话一样， 说话的同时也能够听到对方的声音)，因此每个方向必须单独关闭连接，即连接的一端需先发 送关闭信息到另一端，当关闭信息发送后，发送关闭信息的一端不会再发送信息，但另一端仍 可向该端发送信息。

2 UDP协议

UDP协议即用户数据报协议(User Datagram Protocol) 9使用UDP协议的传输层中传输 的数据是按UDP协议封装成的数据报，每个数据报的前8个字节用来存储报头信息，其余字 节用来存储需传输的数据。

UDP是一种无连接的传输层协议，因为UDP的收发双方并不存在连接关系，按照UDP协 议传输数据时，发送方使用套接字文件发送数据报给接收方，之后可立即使用同一个套接字发 送其他数据报给另一个接收方；同样的，接收方也可以通过相同的套接字接收由多个发送方发 来的数据。

UDP不对数据报进行编号，它不保证接收方以正确的顺序接收到完整的数据，但UDP会 将数据报的长度随数据发送给接收方。虽然UDP面向无连接的通信，不能如TCP般很好地保 证数据的完整性和正确性，但UDP处理速度快，耗费资源少，因此在对数据完整性要求低、 对传输效率要求高的应用中一般使用UDP协议传输数据。

& IPWX

IP协议的两个基本功能为寻址和分段。传输层的数据封装完成后没有直接发送到接收方， 而是先递达网络层；网络层又在原数据报前添加IP首部，封装成IP数据报，并解析数据报中

的目的地址，为其选择传输路径乎将数据报发送到接收方，頂协议中这种选择道路的功能也被 称为路由功能。此外，厦协议可重新组装数据报，改变数据报的大小，以适应不同网络对包大 小的要求。需要说明的是，IP协议不提供端到端或结点到结点的确认，只检测报头中的校验码， 不提供可靠的传输服务。

虽然各层使用的协议互不相同，但协议通常都由如下3个部分组成：

（1） 待交互数据的结构和格式；

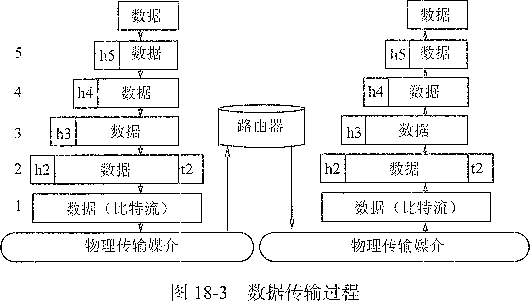
（2） 进行交互的方式，包括数据的类型、对数据的处理动作等；

（3） 事件实现顺序的说明。

一组完整的协议不仅需要考虑通信双方在正常情况下的动作，还应考虑到通信时可能出现 的异常,并对异常情况下通信双方的动作做出规定。

18.2.3數骚億籲赢複

在数据传输的过程中，除物理层之外的其他各层都会向原始数据中添加控制信息，若接收 双方通过同一个路由器连接，那么数据在传输过程中的变化将如图18；所示。



由图1&3可知，两个进程间进行通信时，数据传输流程以及数据的变化情况如下：

（1）来自应用进程1的数据到达应用层，应用层根据本层协议在其头部添加相应控制信 息h5,之后数据被传向传输层。

（2 ）传输层接收到来自应用层的信息，经传输协议TCP或UDP添加TCP首部或UDP首 部后，作为数据段或数据包被传向网络层。

（3） 网络层收到来自传输层的数据后，为其添加IP首部并封装为IP数据报，传递到链 路层。

（4） 链路层接收到来自网络层的IP数据报，在其头尾分别添加控制信息，封装成数据帧 并传递到物理层。

（5） 物理层接收到来自链路层的帧数据，将其转化为由0、1代码组成的比特流，再传送 到物理传输媒介。

（6 ）物理传输媒介中的比特流经路由转发，首先到达应用程序2所在的物理传输媒介中， 之后按照TCP/IP协议簇中的协议，先将比特流格式的数据转换为数据帧，再依次去除链路层、

网络层、传输层和应用层添加的控制信息，最后将原始数据递送给应用程序2。

到此，两个进程完成了在网络间的一次数据传递。

由以上数据传输过程可知，体系结构中各层的实现建立在其下一层所提供的服务上贸且本 层继续向上层提供服务，各层之间的常用协议以及层级关系如图18・4所示。

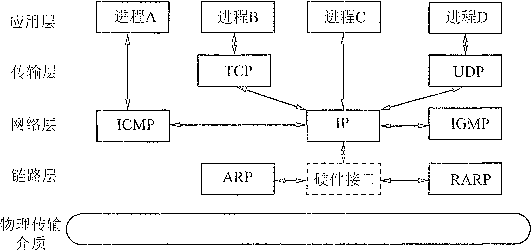
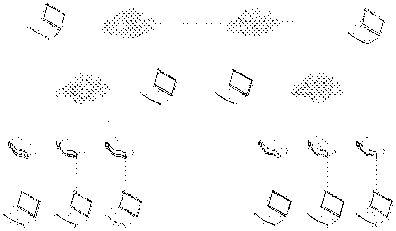


图18-4协议与体系结构

网络架构分为C/S架构和B/S架构,其中C/S架构即客户端(client) /服务器(server)模式, 这种架构需要在进行通信的两端分别架设客户机和服务器，常见的基于C/S架构的有银行系统 中的ATM机、打印店中的计算机与打印机等;B/S架构是浏览器(browser) /服务器(server) 架构，这是万维网(World Wide Web )兴起后的一种网络架构，客户机只需安装浏览器，便可 与服务器进行交互，常见的B/S架构如百度、谷歌等浏览器建设、大多学校使用的校园网，以 及公司内网等。

实际上，这两种网络架构都符合客户端/服务器模式，它们最大的区别在于客户端是否需 要特定的硬件支持。

网络架构中的服务器通常用于提供服务，客户端则用于发出请求。客户端/服务器之间可 以是一对一、多对一，也可以是多对多的关系。服务器是被动的，客户端是主动的。服务器一 般需要永久运行，以便能随时接收并处理用户请求；而客户端可以只在需要时启动，但它应主 动向服务器发起请求。

18O4 IP地址

ip地址用于在网络上唯一标记一台计算 机。如图18＜所示的网络中，包含多个小型 的网络与众多主机，假设其中的pci要向pc2 发送信息,那么pci必须能在这个网络中找到 pc2,这要求pc2在整个网络中有一个唯一标识, 这个每台主机在网络中的唯一标识就是IP地址。

目前较通用的IP地址是网际协议的第四版地址，即IPv4。IPv4由4个字段和3个分割字 段的“了组成，每个字段的取值范围为0〜255,即0〜2七如“12700.1”就是一个标准的IPv4 格式的地址，使用这种方式表示的地址叫作“点分十进制”地址。1P地址中的字段可以使用二 进制表示，如 “127.0Q1” 也可表示为"11111111J0000000J0000000.00000001",此地址是本 机回送地址(LoopbackAddress ),可用于网卡在本机内部的访问。

IPv4地址共分为5类,依次为A类IP地址、B类IP地址、C类IP地址、D类IP地址 和E类IP地址。其中A、R、。类IP地址在逻辑上又分为两个部分：第一部分标识网络，第 二部分标识网络中的主机，如IP地址"92J684331”，该地址的前3个字段标识网络号为 ”92」68.43.0”，最后一个字段“31”标识该网络中的主机，具体如图1强6所示。

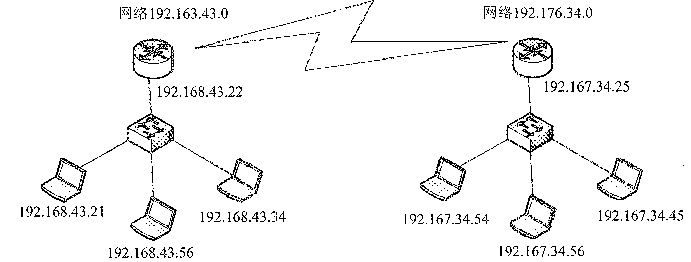


图184 IP地址图示

由图18-6可知，处于同一网络中的主机由最后一个字段区分。

图18・6中所示的IP地址都是C类IP地址」P地址根据取值范围分类，具体如图1&7所示 网络号7位 主机号24位

r~: ~1 r~ ~~~ ~~ ……二二二…一……….…1L

a类

网络号14位 主机号16位

I 5 I !

b类 bIUBBBBB

网络号21位 主机号8位

I B i \*

c类 iiiiimm BBB8BB8B

d类 iHIOSSe 88BB0B88 BOBB®BBB 8®BBBBOB  
e类 BOBBBBBB

图18-7 IP地址的分类

网络号相同的IP地址处于同一个网段，A〜E类IP地址的取值范围及可用IP数量分别如下：

(1 ) A类地址

A类IP地址由1字节的网络号和3字节的主机号组成，网络号的最高位必须是“0” o A 类IP地址的范围为:1.0.0.1~126.255.255.2549可用A类网络共有20 即126个;每个网络 的可用IP地址有224-2，即1 677 214个。

（2） B类地址

B类IP地址由2字节的网络号和2字节的主机号组成，网络号的最高位必须是”0” o B 类IP地址的范围为:128.1.0.1〜191.255.255.254,可用R类网络有（214-2 ）个,即16 384个， 每个网络的可用IP地址有（216-2 ）个,即65 534个。

（3） C类地址

C类IP地址由3个字节的网络号和一个字节的主机号组成，网络号的最高位必须是“110” o C类IP地址的范围为:192.0.L1〜223.255.255.254。每个C类网络中可用IP地址有（28~2 ）个, 即254个。

（4） D类地址

D类IP地址不分网络号和主机号，它固定以“1110”开头，取值范围为：22400」〜 239255255254O D类IP地址并不指向特定的网络，目前这一类地址被用在多播中。

（5） E类地址

E类地址不分网络号和主机号,它固定以"11110"开头,取值范围为:240.0.0.1〜 255.255255.254。E类IP地址仅在实验和开发中使用。

A、B、C类IP地址每个网络号中的可用IP地址总是为某类IP地址的网络号位数）， 这是因为，主机号从0开始，但第一个编号％”与网络号一起表示网络号（如C类][P地址的 第一个网络号为127。0°0）,最后一个编号“255”与网络号一起作为广播地址存在（如C类 IP地址的第一个广播地址为127.0.0.255 ） o

此外，每个网段中都有一部分IP地址是供给局域网使用的9这类IP地址也称为私有地址， 它们的范围如下：

。10.0.0.0 〜10.255.255.255。

® 172.16.0.0 〜17231.255.255o

© 192.168.0.0 〜192.168.255.255o

由于使用四个字段表示的IP地址难以阅读和记忆，人们发明了域名系统，域名系统中的 每个域名都对应唯一一个IP地址，即使用域名或者与域名对应的IP地址可以访问网络上的同 一台主机,例如,使用域名“www.baidu.com”或者IP地址“202.108.22.5”都能访问百度的主机。

域名和IP地址也被称为主机名（hostname ）。

随着智能设备的普及，IPv4格式的地址数量已不能满足人类的需求，为此IETF小组设计 了新的IP地址格式，即IPv6o IPv6的地址长度为128 bit,是IPv4地址长度的4倍，通常使用 点分十六进制表示,如“2001:DB8:0:23:8:800:200C:417A” ,若地址的一个字段全部为0,可 将其压缩为“::”，如％80::384a:22bc:2c98:71e5”,“在一个地址中只能出现一次。

如今网络中使用率较高的地址仍为IPv4,对IPv6有兴趣的读者可查阅相关资料自行学习。

18.4.3

子网掩码又称为地址掩码，它用于划分IP地址中的网络号与主机号，网络号所占的位用“1” 标识,主机号所占的位用“0”标识膏因为A、B. C类IP地址网络号和主机号的位置是确定的3 因此子网掩码的取值也是确定的，分别如下：

(1 ) 255.0.0.0,等同于 11111111.00000000.00000000.00000000,用于匹配 A 类地址;

1. 255.255.0.0,等同于 11111111.11111111.00000000.00000000,用于匹配 R 类地址;
2. 255255.255.0,等同于 1111111L11111111.11111111.00000000,用于匹配 C 类地址。

子网掩码通常应用于网络搭建中，申请到网络号之后，用户可利用子网掩码将该网络号标 识的网络划分为多个子网，假设申请到了一个C类网络，网络号为“193°93.54甘’，这个网络 中的可用IP地址有254个，若想将这个网络等分为4个子网，则可将子网掩码第四个字段的前 两位设置为］,得到子网掩码 ulllllllLlllllllLlllllllL11000000n ,即 “255.255.255.192”。 此时得到的四个子网的IP地址取值范围分别如下：

•网络号:192.93.54.0o IP 范围:192.93.54.1 〜192.93.54.62。

•网络号:192.93.54.64。IP 范围:192.193.5465〜192.93.54J26。

•网络号:192.93.54.128o IP 范围:192.193.54.129〜192.93.54.190。

•网络号:192.93.54.192o IP 范围:192.193.54.193〜192.93.54.254。

以上示例将C类网络193.93.54.0等分为了 4个子网,这4个子网使用相同的子网掩码。 若要将该网络划分为不等长的子网，那么子网的子网掩码将各不相同。

例如将C类子网192.9354J划分为三个不等长的子网，要求第一个子网需能容纳100台 主机，第二个子网需能容纳50台主机，第三个子网需能容纳25台主机，那么第一个子网至少 应有126个主机位,第二个子网至少应有62个主机位，第三个子网至少应有30个主机位。假 设将C类网络192J68540依次分给这三个子网，那这三个子网的网络号、子网掩码依次如下：

•网络号:192.93.54.0。子网掩码:255.255.255.128。

•网络号:192.93.54.128o 子网掩码:255.255.255.192。

•网络号:192.93.54.192。子网掩码:255.255.255.224。

这三个子网的IP地址范围依次为:

•子网\_： 192.93.54.1〜192.93.54J26。

* 子网二:192.93.54.129 〜192.93.54.190。
* 子网三:192.93.54.193〜192.93.54.222。

18o 5嘏口1号

IP地址只能确定网络中的主机，要确定主机中的进程，还需用到端口号(port) o在计算 机网络中，端口号是一台主机中进程的唯一标识，因此一个进程在向另一个进程发送数据时， 要使用“IP地址+端口号”唯一确定网络中的唯一进程。

端口号的最大取值为65 535,其中0〜1024号端口一般是由系统进程占用，用户可到www。 ianaxom上查看由国际因特网地址分配委员会维护的官方已分配的端口列表。用户在编写自己 的服务器时，可以选择一个大于1 024、小于65 535的端口号对其进行标记，但要注意选择空 闲端口号，避免与其他服务器产生冲突。

膾顼厕络编程嬲述

网络编程的实质是两台设备中的进程通过网络进行数据交换，即进程间的网络通信。当今 的网络编程一般都基于请求/响应方式，即一个进程向另外一个进程发送请求，接收请求的进 程对请求进行处理，再将处理结果返回到发送请求的进程中。

在网络编程中，用于发起连接请求的进程被称为客户端，等待其他程序连接的进程被称为 服务器。客户端程序可以在需要的时候启动，但为了保证能随时处理客户端请求，原则上服务 器应保证一直运行。

R&7套接宇

套接字即socket, socket是进程间通信方式的一种，其本意为“插座”，常被称为套接字。 socket起源于UNIX,在UNIX和Linux中,socket被具象化为一种文件，当使用socket进行通 信时，进程会先生成一个socket文件，之后再通过socket文件进行数据传递。

Python中有一个名为socket的模块，该模块包含了网络编程的基本方法、函数和类等。 socket模块中包含了一个同名类,通过该类可以实例化出一个socket对象。socket类 init 方法的声明如下：

init (selfz family=<AddressFamily.AF\_INET: 2>z

type=<SocketKind o SOCK\_STREAM: *1>r* proto^O z fileno=None)

其中的参数family用于指定地址簇，默认值为AF\_\_INET,表示可用于地址为IPv4格式的 进程间通信，也可以使用AFJNET6和AF\_UNIX为family赋值：

° AF\_INET6：表示可用于地址为IPv6格式的进程间通信。

° AF\_UNIX：只能用于单一的UNIX系统进程间通信。

参数type用于指定socket的类型，该参数决定socket的通信方式，其取值及代表的含义 分别如下：

* SOCK\_STREAM： type的默认值，表示流式套接字，用于TCP通信中。
* SOCK\_DGRAM：数据报式套接字，用于UDP通信中。
* SOCK\_RAW：原始套接字，用于处理][CMP、IGMP等网络报文，或需用户构造IP头部 的通信中。

参数pmto用于指定与特定的地址簇相关的协议，其默认值为0,表示由系统根据地址格 式和套接字类型自动选择合适的协议。

参数fileno用于为套接字文件设置文件描述符，默认设置为None,表示由系统分配。

socketQ函数的使用方法如下所示。

»> import socket

>» socket\_tcp = socket» socket(socket oAF\_INETr socket» SOCK\_STREAM)

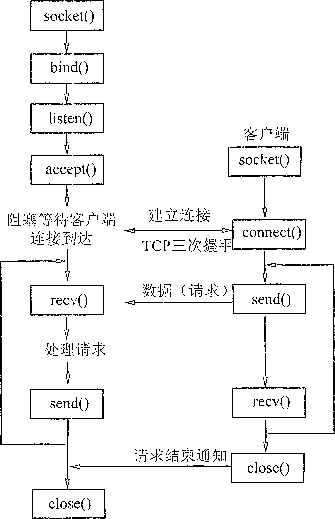
»> socket\_tdp = socket.socket(socket.AF\_INET,socket.SOCK\_DGRAM)

使用如上的两条语句，分别可创建一个基于TCP通信的流式套接字socketjcp,和一个基 于UDP协议的数据报式套接字socket\_udp, socket方法创建的套接字默认为一个主动套接字， 即当需要与其他进程通信时，该套接字应主动向目标进程发送数据。

网络通信中的程序通常分为服务器程序和客户端程序，这两种程序中都会创建socket对象, 但针对socket的操作流程各不相同。

18O8 soctet WOfS

根据socket的类型，网络通信又分为基于TCP协议的、面向连接的通信，以及基于UDP的、 面向无连接的通信。

面向连接的通信类似日常生活中的电话服务：在电话服务中，接电话的一方需要保持手机 畅通，以便可随时接收他人发送的通话请求;在电话连通之后，通话的双方可以开始交换数据。 面向连接的socket通信流程如图18-8所示。

服务器

图18-8面向连接的socket通信

结合图18-8,在面向连接的通信中，服务器的工作流程如下：

(1 )调用socket()函数创建socket0

(2 )调用bind()函数将服务器进程与端口地址绑定。

(3 )调用listenQ函数开启服务器监听，等待客户端的连接请求。

(4 )当有客户端请求递达时，调用accept()函 数尝试与客户端进行连接。

1. 若连接成功，则处理客户端请求，并将处 理结果反馈给客户端。
2. 继续等待客户端请求并进行处理。 (7 )通信结束后，调用close()函数关闭socketo

需要说明的是，由于服务器需要一直保持运行， 所以除非有特殊情况，否则服务器端永远不会关闭监 听的 socketo

结合图18-8,在面向连接的通信中，客户端进程 的工作流程如下：

(1 )调用socket()函数创建socketo

(2 )调用connectQ函数向服务端发起连接。

(3 )连接建立后须调用send()函数向服务器发 送数据。

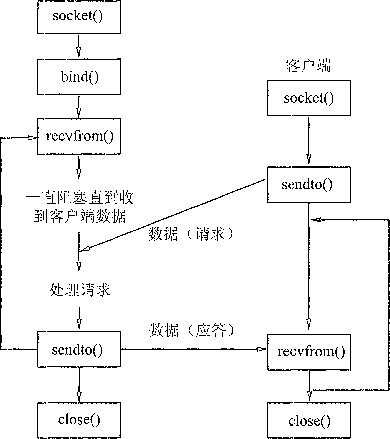
(4 )接收服务器反馈的处理结果。

(5 )若仍有需求则继续进行数据发送和接收操 作，否则关闭socket向服务器发送结束通知o

1Ba@a2顾廠罪建遥齣瞩僱 •

面向非连接的socket通信与生活中的邮件投 递类似，接收邮件的一方无须一直等待发送邮件 的一方发起连接请求；发送邮件的一方只需知道 接收方的收件地址，便可直接投递邮件。面向非 连接的通信流程如图18成所示。

图18™9中所示的面向非连接的socket通信流 程与图18^8中所示的面向连接的socket通信流程 大致相同，区别在于面向非连接的通信中，客户 端不再发起连接请求，而是使用sendto()函数向 指定的接收方发送数据；服务器则使用recvfromQ 函数接收数据，并可以使用sendto()函数将请求 的处理结果反馈到客户端。

18o9 socket

服务器

图1&9

socket模块中为socket对象定义了一些内建方法,通过这些内建方法,可以对套接字进行 操作，常用的方法如表18』所示。

表18-1 socket的内建方法

|  |  |
| --- | --- |
| 函数/方法 | 说 明 |
| s.bind(address) | 将套接字s与通信地址address绑定 |
| s.listen(backlog) | 将套接字变为监听套接字，并设置允许等待接受的最大连接数 |
| s. setblocking(bool) | 设置套接字的阻塞状态，默认为True表示阻塞，False表示非阻塞 |
| s.connect(address) | 向地址为address的进程发起连接请求 |
| s.accept() | 接受连接请求 |
| s. send()/s .sendto(adress) | 使用套接字s发送数据/向adress发送数据 |
| s .rec v()/s .recvfrom() | 使用套接字s接收数据 |
| s.closeQ . | 关闭套接字s |

18.9J bmd—BOtttaSB# .. .

bind方法一般由服务器端的socket调用。为了确保客户端能准确地找到服务器进程，服务 器进程的地址(hostname+port)通常是固定的，服务器程序在创建socket后，通常会调用bind 方法将服务器socket与服务器地址绑定。bind方法的声明如下:

bind(address)

其中的参数address是一个形如"(hosfuaigport)”的元组，address元组中的第一个元素

hostname本质是一个字符串,表示主机地址;第二个元素port的本质是整数,表示进程端口号。 当元组中的hostname为空字符串时，bind将会允许使用本机的任意IP地址作为目标地址。

假设当前服务器端的socket对象为socket\_server,主机名为“192J684331”，端口号为 “3456” ,那么bind方法的用法如下所示。

socket\_server.bind(('192。168。43。31',3546))

客户端的socket也可以调用bind方法绑定一个地址，但因为客户端进程存活时间相对较短， 且客户端套接字为主动套接字，服务器在接收数据时动态地获取客户端地址已能满足需求，因 此客户端一般不进行此项操作。客户端进程的端口号通常由系统随机分配。

套接字默认是主动发送数据的，但服务器程序的套接字应被动地监听客户端的连接请求。 调用lists方法将使一个套接字由主动套接字变为被动套接字，等待接收其他程序的连接请求。 listen方法的声明如下:

listen([backlog])

listen方法中的参数backlog用于指定在拒绝新连接之前，系统允许的未完成连接数。内核 •为监听套接字维护了两个队列：已连接队列和未连接队列，完成三次握手过程的客户端对应的 套接字将被添加到已连接队列中，处于半连接状态的客户端对应的套接字将被添加到未连接队 列中，若未连接队列存满，再有新的客户端发起连接请求歹该连接请求将被直接拒绝。

需要说明的是，在3.5版本的Pythoii中，backlog是一个可选参数，若指定该参数，则其 值至少为0;若该参数缺省，则系统会选择一个合理的默认值为其赋值。

listen方法的用法如下所示。

socket\_server.listen(5)

18n9B3 還豆堵臘翳器齣麗潑

connect方法由客户端的socket调用，其功能为向服务器发起连接请求。connect方法的声 明如下：

connect(address)

参数address是一个形如"(hostnam勺port)”的元组，用于指定服务器的地址。若连接出错， 将返回socket.error错误。

假设客户端的套接字对象为client socket,贝!j connect方法的用法如下所示。

client\_socket.connect((\* 192 o168.43.31 \*, 354 6))

该示例的功能为：客户端套接字clientsocket向主机名为192」6&43°31,端口号为3456的 进程发起连接请求。

18S.4 accept—「

acc顷方法由服务器端的socket调用，其功能为处理客户端发起的连接请求。accept方法 的声明如下：

accept()

若调用accept方法前没有客户端连接请求到达，accept方法将会使服务器进程阻塞，直到 有客户端请求连接到达时才会返回；否则accept方法立即返回一个形如(conn?address)的元组， 其中conn是新的套接字对象，用于与相应客户端进行数据交互；address是客户端进程地址， 其本质为一个形如(hostname,port)的元组。

accept方法的用法如下所示。

client\_socketz address = socket\_server » accept()

18a9Q5寧郭础偈0肮砒®)—褰送歡H

send, sendto方法用于向目标进程发送数据，它们的声明分别如下：

send(string)

sendto(string^ address)

院nd方法由流式套接字调用；sendto方法一般由数据报式套接字调用，参数string用于设 置要发送的数据，需要注意的是，Python 3中要求曲ing为字节码格式，因此用户需使用如下 方式传入数据：

1. 以ub?string,n的形式传入参数，例如传递的字符串为"hello ithcima”，则send方法 的用法为sendCb'hello itheima1) ?这种方式只能转换ASCII字符。

(2 )通过encodeQ函数对字符串进行转码，encode()函数中需传入一个表示字节码格式 的参数，如将字符串转为gb2312格式(针对的是Windows系统)的字节码，则应使用语句 send(?hello itheima\encode(ngb2312"))。

sendto方法中的参数address本质为一个形如(hostname,port)的元组*，*用于指定目标进程的 地址。

send, sendto方法调用成功都会返回所发送字符串的字节数，它们的用法如下所示。

send(5hello world5)

sendto(!hello world\*, ( ?192„168o12.32 \ 4567)) .

pb篡辺『詹©wf［過海~護臘饑骚

recv、wcvfiDm方法用于接收数据，其声明分别如下：

recv(bufsize)

recvfrom(bufsize)

冀cv方法的功能是接收数据，其参数bufsize用于设置可接收的最大数据量。若方法调用 成功，返回接收到的数据。

recvfrom方法的功能是接收数据，其参数bufsize同样用于设置可接收的最大数据量。若方 法调用成功,recvfrom方法将返回一个形如(data,address)的元组,其中data是接收到的数据, address是发送数据的套接字地址。

‘ 需要注意的是，这两个方法在python 3中返回的数据都是字节码，若要以字符串形式打印，

需先使用decode()函数对其进行解码，decode()函数的参数与encodeQ函数的参数相同，都表 示数据的格式。

假设接收的数据存储在recv info中,则使用decode方法将recv\_info转为gb2312格式的 语句如下:

recv\_info.decode(\* gb2312 5)

18.9.7 ◎瞄岛(8——美衝套簷掌 ：

close方法用于关闭套接字。类似于文件，套接字也是系统中的一种资源，因此使用完毕 的套接字应及时关闭。此外，一台主机中端口的数量是有限的，系统同样应关闭空闲的套接字， 避免端口浪费。

close方法的用法如下所示。

client\_socket.close() server\_socket。close()

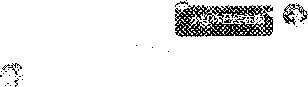
当客户端终止后，服务器中用于与此客户端交互的套接字也应关闭。一般情况下9若套接 字接收到的数据为空，则说明客户端进程已经关闭连接，因此可通过判断接收到的数据长度判 断是否关闭与客户端交互的套接字。

ISolO厕鑑编程实例

为帮助读者理解和掌握网络编程，本节将以案例的形式分别实现UDP聊天室，和TCP数 据转换服务器。

18J0J ODPIBSB

聊天室是早期网络交流常用的软件之一，也是现如 今大多通信软件配备的功能之一，一般情况下，聊天室 可接收聊天室中不同成员发送的信息并显示在聊天窗口 中，如图1820所示，就是一个简单的聊天室界面。

图18-10所示的聊天室界面主要分为两部分，第一 部分是一个聊天窗口，用于显示接收到的不同聊天室成 员发送的消息；第二部分是一个编辑框，用户可在此框 中编辑消息，发送到聊天室中。虽然这两部分出现在同 一个界面中，但实际上他们是聊天室的两个功能，需要 由不同的程序实现。

诲有狞也舞曲搏方留

聊天室一般基于UDP协议，以上所示的聊天室中， 聊天窗口是一个基于UDP协议的服务器，编辑框则是 一个基于UDP协议的客户端。聊天窗口应可接收编辑框发送来的数据，并将发送数据的地址 以及数据打印到聊天室中，下面通过简单代码来展示UDP聊天室的实现方式。

—Ill—

图1840聊天室界面示例

作为服务器的UDP聊天窗口实现如下。

import socket def main():

* 1.创建 socket

server\_\_socket = socket.socket(socket.AF\_INET^ socket.SOCK\_DGRAM)

* 2 . Mil：绑定

server\_socket.bind((""z 3456))

* 3•接&数据并打印

print (" UDP 聊天室 °)

while True:

recv\_info = server\_socket 8 recvfrom(1024)

address = recv\_info [ 1 ] [0] + \* :: f + str (recv\_info [ 1 ] [1]) print("%s° % address)

print (?'%sn % recv\_info [0] .decode ("gb2312°)) # 4 .关闭聊天室套接字server\_socket server\_socket.close()

if name == ' main \* :

main()

执行此段代码，结果如下所示：

一——一-UDP聊天室

此时服务器已启动，为测试服务器接收与处理客户端数据的功能，下面实现一个实现消息 编辑发送功能的客户端。客户端的具体实现如下：

import socket

def main():

* 1.创建客户端socket

client\_socket = socket.socket(socket.AF\_\_INETZ socket.SOCK\_DGRAM)

* 2。向i艮务器发送数据 — —

print ( ? —输■入•框. ?)

while True:

data = input()

client\_socket。sendto(data.encode("gb2312 °f

(°172.16.43.31,\ 3456))

if data == ? 88 ?: break

* 3 .关闭 client\_socket client\_socket.close()

if name == 5 main ?:

main()

此时服务器已启动，为测试服务器接收与处理客户端数据的功能，下面实现一个实现消息 编辑发送功能的客户端。客户端的具体实现如下：

from socket import \*

def main():

#1.创建客户端socket cliSock = socket (AF\_\_INETZ SOCK\_DGRAM)

#2 •向服务器发送数据 — ~

print ( ? 孝俞才匡———?)

while True: s = input()

cliSock.sendto(s.encode(ngb2312n), ("172.16.43.31°,3456))

if s == x887:  
break

#3 .关闭 cliSock cliSock.close()

if name == J \_main \*: main()

客户端程序中的IP地址"72.16.4331”为服务器所用主机的IP地址,即本机的IP地址, 读者在进行测试时，需传入个人主机的IP地址，个人主机的EP地址可在命「令行中使用ipconfig 命令查看(Linux系统中釆用ifconfig命令查看)。

执行客户端程序，客户端的执行结果如下：

糸曾入框—-—

此时可通过客户端向聊天室中发送数据，示例如下：

输入框

你好

之后聊天室中打印的信息如下:

UDP聊天室

172.16.43.31:50873

你好

观察打印结果，其中新增了两行数据，第一行"172464331:50873"为客户端的IP地址 和端口号508739该端口号由系统随机分配；第二行为客户端发送的数据。

用户可使用一个客户端发送多条数据，亦可启动多个客户端向聊天室发送数据。启动位于 不同主机上的多个客户端向聊天室中发送数据，聊天室中打印的信息如下：

观察打印结果，聊天室中可接收并打印多个客户端发送的信息，可知UDP聊天室实现成功。 需要说明的是，在多台计算机中进行测试时，计算机中的防火墙可能会过滤掉来自其他主 机的UDP客户端发送的数据包，为了保证测试成功，可先使用"service iptables stop"命令关 闭防火墙，测试完成后,再通过"service iptables start"命令重启防火墙。

UDP聊天室

172.16.43.31:50873 # 用户 1

你好

172.16.43.33:64632 # 用户 *2*

你们知道传智播客吗？

172.16.43.37:65200 # 用户 3

知道

172.16.43.51:58017 # 用户 4

知道

172.16.43.33:64632 # 用户 2

那你们知道传智专修学院吗？

172.16.43.37:65200 # 用户 3

那是什么？

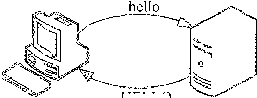
172.16.43.51:58017 # 用户 4

是传智播客创办的大学

TCP数据转换程序位于服务器端，如图18』1所示，数据 转换程序可以接收客户端发来的字符，将其转换为大写字符后 返回给客户端。

1BJ0.2 TCPB»B

TCP数据转换程序的具体实现如下:

import socket

HELLO

客户端

服务器

图18』1 TCP数据转换

def main():

#「创建套接字 server\_socket

server\_socket = socketosocket (socket., AF\_INETf socket« SOCK\_STREAM)

* 2 .绑定地址

server\_socket.bind((n,\ 5678))

* 3 . 最大连接数

server\_socket.listen(5)

* 4。创谴连接

client\_socketz address = server\_socket o accept() print ( ? TCP 数据转换器-----?)

while True:

* 5 .接收数据

recv\_inf o = client\_\_socket. recv (102 4) . decode ( \* gb2 312 5 ) string\_address = address[0] + !:? + str(address[1]) print(string\_address)

print ( n 待处理数据:%s ° % recv\_inf o)

* 6.数据处理 一

if recv\_info:

data = recv\_info »upper() client\_socket。send(data.encode(1gb2312 \*)) print ("处理结果:%s°%data)

else :

print(¥ exit')

client\_socket。close()

break

* 7.关闭套接字 server\_\_socket server\_socket。close()

if name == ?\_main\_':

main ()

为了测试该程序的功能，我们需实现一个用于发送数据和接收数据处理结果的客户端程序, 客户端程序代码如下。

import socket

def main():

* 1.创建套接字 client\_socket

client\_socket = socket <, socket (socket OAF\_INET, socket. SOCK\_\_STREAM)

* 2。请求连接

client\_socket» connect((1172.16^43.31', 5678))

* 3。发送数据

while True:

data = input("

待处理数据

\")

client\_\_socket. send (data。encode ( 1 gb2312'5 ))

recv\_info = client\_socket.recv(102 4) decode(\* gb2312 \*)

print ( n 处理结果 \n%s" % recv\_info)

# 4 .关闭套接字 client\_socket client\_\_socket. close ()

if name == ? main ':

main ()

启动TCP数据转换程序，终端并没有数据打印。之后启动TCP客户端程序，TCP数据转 换器和TCP客户端程序的终端中打印的信息分别如下：

(1 ) TCP数据转换程序终端

TCP数据转换器——-

(2 ) TCP客户端程序的终端

待处理数据一一一

此时TCP客户端程序处于等待用户输入数据的状态，TCP数据转换程序处于阻塞等待 客户端发送数据的状态，向客户端的终端中输入uhelio itheima"后，两端显示的信息分别 如下：

(1 ) TCP数据转换程序终端

172.16.43.31:13142

待处理数据:hello itheimao

处理结果:HELLO ITHEIMAo

1. TCP客户端程序的终端

待处理数据

hello itheima

处理结果

HELLO ITHEIMA

再次向客户端终端中输入数据，数据转换程序将继续处理新的数据，并将处理结果返回到 客户端中。由此可知，基于TCP的数据转换功能实现成功。

*以；多学一*招：端口保留

服务器在网络中的地址是唯一的，因此在为其设置端口号时，必须使用主机中的空闲端口 号。但有时我们明确使用的是空闲端口号，却还是会遇到如下所示的问题：

OSError : [WinError 10048 ]通常每个套接字地址(协议/网络地址/端口)只允许使用一次。

这是因为，默认情况下内核会在进程终止的两分钟内，保留进程的端口号。若想解决这一 问题，可以在创建套接字后，执行如下语句：

server\_socket•setsockopt(socket。SOL\_SOCKET7 socket.SO\_REUSEADDR^ 1)

以上调用的方法setsockopt用于设置套-接字的选项，当其中的第三个参数被设置为1时， 服务器终止后在两分钟内重新启动，可重复使用同一个端口。

:18oll TCP »IK# 器

计算机中的一个处理器于某个时刻只能处理一个程序，但因为CPU釆用分时复用技术， 在几个连续的时间片中处理不同的程序，又因为CPU的时间片非常短暂，所以从感官上人们 会认为一个处理机可同时处理多个程序，像这种处理程序的方式被称为并发处理。类似地，在 一个极短的时间段中可以为多个客户端服务的服务器称为并发服务器。

UDP服务器是无连接的，客户端的情况不会对服务器造成影响，服务器可以随时接收并处 理客户端发来的数据，如18.104小节中的UDP聊天室。

TCP服务器是基于连接的，连接请求和收发数据都在同一个进程中处理，一个可实现并发 操作的TCP服务器，应能建立并保留多个连接，每个连接又应该可以连续接收数据，由此， TCP服务器的大体框架应该如下:

while True:

#在循环中处理连接请求

new\_socketz address = server\_socket.accept() while True:

#保持不断接收数据

recv\_\_data = new\_socket. recv (102 4)

但若仅以此框架对18402小节中实现的TCP服务器进行调整，会发现服务器仍然无法实 现并发。这是因为，若在调用心顷方法时恰好有连接请求到达，那么连接会被创建，且新的 套接字new\_socket被返回，之后new\_socket通过冀cv方法等待接收客户端发送的数据，但客 户端若尚未发送数据，程序将在此处阻塞，此时若有新的连接请求到达，accept显然会因此次 阻塞而无法对连接请求做出处理。同样地，若程序在new\_socket调用accept方法后，因没有新 的连接请求到达而阻塞，服务器也无法处理已建立连接的客户端发送的数据。

总的来说，单进程服务器中的连接处理工作和数据交互工作都可能造成程序阻塞，为了避 免因一项阻塞影响程序其他流程的执行，TCP服务器中通常采用解阻塞或创建子进程、创建新 线程的方式来实现并发操作。

18J11 MISSBBIKSI '

单进程非阻塞服务器通过解阻塞的方式实现并发操作。Python中的套接字默认以阻塞的方 式处理数据，若套接字调用accept, recv等用于接收数据的方法时没有接收到数据，套接字就 会阻塞等待数据递达。用户可以通过套接字中提供的setblocking方法将套接字设置为非阻塞模 式，如此即便套接字中没有数据递达，套接字调用的方法也会立刻返回。

setblocking的使用方法如下:

server\_socket.setblocking(False)

其中False是一个实参，用于将套接字设置为非阻塞模式。setblocking方法的默认参数为 True,套接字默认工作在阻塞模式°

经过分析我们知道，若要使服务器可以与多个客户端建立连接，需要保证在调用accept > 法时不会产生阻塞，即服务器套接字wrveLsocket应被设置为非阻塞；若要使每个连接中的客

户端都可随时向服务器中发送数据，则需保证在调用冀cv方法时不会产生阻塞身即新套接字 new\_socket应被设置为非阻塞。 •

下面将以18J02小节的TCP数据转换服务器为基础，实现一个TCP单进程非阻塞并发服 务器，具体代码如下：

1. import socket
2. def main():
3. # 1.创建 socket
4. server\_socket = socket.socket(socket.AF\_INETZ socket.SOCK\_STREAM) ■
5. #重复後用绑定的信息 一 一
6. server\_socket.setsockopt(socket.SOL\_SOCKET, socket.SO\_REUSEADDR, 1)
7. local\_addr = (f s 7 6789)
8. # 2 •绑定本地工P以及port
9. server\_socket.bind(local\_addr)
10. # 3。将socket变为监听(被話)套接字
11. server\_socket.listen(3)
12. # 4 .将服务器套接字server\_socket设置为非阻塞
13. server\_socket.setblocking(False)
14. #用来保存所有已经连接的客户端的信息
15. client\_address\_list =[]
16. while True:
17. #等待一个新的客户端到来
18. try:
19. new\_socketclient\_address = server\_socket.accept()
20. except:
21. pass
22. else:
23. print (° —个新的客户端到来:%sn%str (client\_address))
24. #将新套接字new\_socket设置为非阻塞
25. new\_socket.setblocking(False)
26. #将本次建立连接后获取的套接字添加到已连接客户端列表中
27. client\_\_address\_list.append((new\_socketz client\_address))
28. for client\_socketz client\_address in client\_ad.dress\_list:
29. try:
30. recv\_info = client\_socket.recv(1024) .decode(? gb2312?)
31. except:
32. pass
33. else:
34. if len(recv\_info)>0:
35. #数据处逾
36. print ( 1 待处理数据:%s 1 %recv\_info)
37. data = recv\_info.upper()
38. client\_socket. send (data . encode ( 5 gb2.312 !))
39. print ( f 处理结果:%sJ%data)
40. else:
41. #断开连接
42. client\_socket.close()
43. #将套接字从已连接客户端列表移除
44. client address list o remove((client socket^

client\_address)) print ( °%s 已断开连接 °%str (client\_address)) server\_socket。close()

45

46

47

48

49

name == ? main \*:

\_name\_  
main ()

第15行代码创建了一个列表client\_address\_list,用于存储每次建立连接后获得的客户端信 息，由于每次连接都会产生新的套接字和客户端地址，若不将这些信息进行存储，那么这些信 息将会被下次建立连接后获取到的套接字和客户端地址信息覆盖。’

需要说明的是，在此段代码中为accept方法和gv方法的调用设置了异常处理，这是因为, 若非阻塞套接字调用这两个方法时没有接收到数据，将会抛出如下所示的异常：

BlockinglOError : [WinError 10035]无法立即完成一个非阻止性套接字操作。

当客户端发送的数据长度为0字节时，表示客户端将断开连接，这种情况下关闭服务器端 与该客户端进行数据交互的套接字,并将套接字从已连接客户端列表中移除,因此第34行代 码对接收到的数据长度进行了判断，根据判断的结果决定数据处理方式。

为了测试该程序的功能，需实现一个用于发送数据和接收数据处理结果的客户端程序，客 户端程序代码如下：

import socket def main():

#「创建套接字 server\_socket

server\_socket = socket.socket(socket.AF\_INETZ socket.SOCK\_STREAM)

* 2.请求连接

client\_socket.connect((f172.16.43 o 31f z 6789))

* 3.发遑数据 while True:

data = input (n 待处理数据——~--\nn)

10

11

12

13

14

15

16

client\_socket.send(data。encode(\* gb2312 ,)) recv\_\_inf o = client\_socket „ recv (1024) . decode ( 1 gb2312 ?)

print -处理结果 —-\n%s° % recv\_info)

# 4 .关闭聊天室套接字server\_socket server\_socket.close()

if name == 5 main main()

启动服务器程序，在多个控制台窗口中分别执行客户端程序，服务器端中将依次打印建立 连接的客户端地址信息，如下所示。

一个新的客户端到来：(?172.16»43.31\ 7836)

一个新的客户端到来：(\*172.16.43.31 S 7844)

一个新的客户端到来：(S172O16.43.31\ 7852)

一个新的客户端到来：(v172.16.43.31 \ 7861)

由此可知，服务器程序可同时与多个客户端建立连接。

使用客户端向服务器发送数据，客户端1、2与服务器端打印的信息分别如下。 客户端L

-一一待处理数据一一一

hello itehima

处理结果—— —— —

HELLO ITEHIMA

\_\_\_\_待处理数据

客户端2：

待处理数据一一一

hello world

处理结果

HELLO WORLD

待处理数据一——

服务器：

一个新的客户端到来 一个新的客户端到来 一个新的客户端到来 一个新的客户端到来

(,172.16.43.31 \ 7836) ('172.16.43.31 \ 7844) (f172.16.43.31 \ 7852) (f172.16.43.31 \ 7861)

待处理数据：hello itehima 处理结果：HELLO ITEHIMA 待处理数据：hello world 处理结果：HELLO WORLD 由此可知，服务器程序可处理多个客户端请求。

综上所述，单进程非阻塞并发服务器实现成功。

18J1.2爹谴糧舞饑贓鰐SI

在多进程并发服务器中，主进程用于处理客户端的连接请求，当有新的客户端与服务器建 立连接后，月艮务器会创建一个子进程，由子进程完成数据的交互工作。

以可将小写转大写的服务器为例，多线程并发服务器的实现如下:

1. import socket
2. import multiprocessing
3. #与客户端进行交互

def deal\_with\_client(new\_socketf client\_address):

4

5

6

7

8

9

10

11

12

13

while True:

recv\_data = new\_socket o recv(1024) » decode(v gb2312 \*) if len(recv\_data) > 0:

print (1 待处理数据:%s: %s1 % (str (client\_address) t recv\_data)) data = recv\_data »upper()

new\_socket。send(data。encode(\* gb2 312 ?))

else :

print ( \* 客户*端*［%s ］已关闭'% str (client\_address))

break

1. new\_socket.close ()
2. def main():
3. #创建服务器套接字server\_socket
4. server\_socket = socket.socket(socket»AF\_INET7 socket s SOCK\_STREAM)
5. #端口瓯速重启用
6. server\_socket <. setsockopt (socket. SOL\_SOCKETZ socket. SO\_REUSEADDRf 1)
7. local address = (5 ? f 8081)

server\_socket,bind(local\_address)

21

22

23

24

25

26

27

28

29

30

31

32

33

34

35

3 6

37

38

39

server\_socket.listen(5)

print ( s 月艮务器 5 )

try:

while True:

#处理客户端连接请求

new\_socketf client\_address = server\_socket.accept() print (? 一个新的客户端到达.[%s] \* % str (client\_address)) #创建子进程与客户端进行交互 —

client = multiprocessing.Process(target=deal\_with\_clientz args=(new\_socketz client\_address))

client.start()

#关闭父进程中的套接字new\_socket new\_socket.close()

finally:

#关闭服务器套接字

server\_socket.close()

\_name == ? main ?:

main()

'以上服务器的大体实现流程与18J02小节中TCP数据转换服务器相同，只是这个服务器 在与客户端建立连接后创建了子进程，用于与本次连接的客户端进行数据交互。

为了测试此段服务器代码，我们需实现一个可向服务器发送数据，并接收服务器反馈数据 的客户端。

import socket

客户端代码实现如下:

1

2

3

4

5

6

7

8

9

10

11

12

13

14

15

16

def main():

* 1.创建套接字 server\_socket

server\_socket = socket.socket(socket.AF\_INETf socket.SOCK\_STREAM)

* 2 •屛连接 ~ —

client\_socket。connect ((! 172.16.. 43.31? z 8081))

* 3.压数据

while True:

data = input (" 待处理数据 -\n")

client\_socket*.send*(data.encode(\*gb2312,)) recv\_info = client\_socket.recv(1024).decode('gb2312\*)

print (" 处理结果 -\n%s" % recv\_info)

# 4.关闭聊天室套接字server\_socket server\_socket。close()

\_name\_ == ; main\_?:

main ()

启动服务器，服务器中会显示如下信息：

一-一服务器-

在多个终端中执行客户端进程,服务器端逐个打印每个客户端的地址信息:

一个新的客户端到达[(U72.16.43.31 S 7866)]

一个新的客户端到达[(U72.16.43.31 S 78 68)]

一个新的客户端到达[(,172.16.43.31,/ 7 8 69)]

一个新的客户端到达[(?172.16»43.31\ 7873)]

由此可知，本小节实现的多进程并发服务器可同时与多个客户端进程建立连接。使用其中 一个客户端向服务器中发送信息，客户端中打印的数据如下所示。

待处理数据

hello itheima

处理结果一一一-

HELLO ITHEIMA

待处理数据一

由以上数据可知，服务器成功接收到客户端发送的数据，并将数据转换为大写后返回到客 户端。

综上所述，多进程并发服务器成功实现。

18J13參爨糧舞慝膽麝器 .. . .「 .

由于每创建一个进程都会耗费系统资源，当进程较多时，服务器的效率会因系统内存的减少 而降低，为解决这一问题，人们考虑使用耗费资源较少的线程代替进程，搭建多线程并发服务器。

对18JL2小节的多进程并发服务器进行修改，多线程并发服务器的代码实现如下。

1. import socket
2. import threading
3. #与客户端进行交互
4. def deal\_with\_client(new\_socketz client\_address):
5. while True:
6. recv\_data = new\_socket 8 recv(1024) .decode(\* gb2312 5)
7. if len(recv\_data) > 0:
8. print ('待处理数据 %s : %s ! % (str (client\_address) , recv\_data))
9. data = recv\_data«upper()
10. new\_socket。send(data。encode(® gb2312 ?))
11. else:
12. print ( 1 客户端［%s］已关闭'%str (client\_address))
13. break
14. new\_socket.close()
15. def main():
16. server socket = socket.socket(socket.AF\_INETr socket.SOCK\_STREAM)
17. server socket.setsockopt(socket.SOL\_SOCKETe socket.SO\_REUSEADDRf 1)
18. local\_address = (1 ? f 8082)
19. server\_socket.bind(local\_address)
20. server\_socket.listen(5)
21. print (f 月艮务器一 \*)
22. try:
23. . while True:
24. new\_socketf client\_address = server\_socket o accept()
25. print (s 一个新的客户端到达［%s］ , % str (client\_\_address))
26. #创建线程，.与客户端进行交互 ~
27. client = threading.Thread(target=deal\_with\_clientz
28. args=(new\_socket, client\_address))
29. client.start()
30. finally:
31. server socket.close()
32. if name == ' main 1 :
33. main()

多线程服务器与多进程服务器基本相同，需要说明的是，多进程服务器在创建子进程后， 会调用close方法关闭服务器进程的new\_socket,这是因为在创建子进程时new\_socket已被复 制给了子进程，服务器进程中的iww\_sockef不再使用；但在多线程服务器中，主线程和新线程 共享new\_socket,若在服务器线程中关闭new\_socket9新线程将无法再使用new\_socket与客户 端进程交互。

与进程相比，虽然线程消耗的资源较少，但大量线程占据的内存仍不可小觑，此外，线程 的稳定性较差，因此多进程和多线程服务器通常只能使用在规模较小、对性能要求较低的场合。

Mo 12 I/O

I/O多路转接服务器是一类单进程服务器，虽然这类服务器中的套接字一般也为非阻塞模 式，但单进程非阻塞服务器中的连接创建和数据处理功能只能在双重循环中交替执行，I/O多 路转接服务器却可以同时监听用来处理客户端连接请求的服务器套接字server\_socket和用来处 理客户端数据的套接字new\_socket0

常见的I/O多路转接服务器有selects poll和epoll9下面将从select入手对DO多路转接服 务器进行讲解。

Python中通过select模块实现select模式,该模块中包含了一个select()函数。selectQ函数 的声明如下：

select (rlistz wlistz xlist[f timeout])

根据套接字中数据的分类，套接字有三种状态，即可读状态、可写状态和异常状态。其中， 处于可读状态的套接字表示接收到了其他套接字发送到的数据，处于可写状态的套接字表示已 准备好向其他套接字发送数据，处于异常状态的套接字表示获取了套接字使用过程中产生的异 常信息。

selectQ函数可同时监测套接字的可读、可写和异常状态，该函数中的前三个参数都是列 表，其中Hist表示等待读就绪的套接字列表；wlisf表示等待写就绪的套接字列表；xlist表示等 待异常出现的套接字列表。若s曲ct模式不检测某个套接字列表，可将对应参数设置为“口” o selectQ函数中的第四个timeout是一个可选参数，用于指定等待时长，通常用以秒为单位 的浮点数表示，若该参数缺省，则等待永不会超时。

selectQ函数调用成功将会返回三个列表，依次为读就绪套接字列表、写就绪套接字列表和 出现异常的套接字列表。用户可在程序中分别遍历这三个列表，获取每个套接字接收到的数据、 要发送的数据和出现的异常信息。

下面将搭建一个基于select模式，可实现小写转大写功能的TCP服务器，具体代码如下:

1. import select
2. import socket
3. import sys
4. def main():
5. server\_socket = socket e socket(socket„AF\_INET, socket.SOCK\_STREAM)
6. server\_socket.bind((! J 7 7788))
7. server\_socket.setsockopt(socket.SOL\_SOCKETf socket.SO\_REUSEADDRf 1)
8. server\_socket alisten(5)
9. #将服务器套接字加入等待读就绪的套接字列表
10. inputs = [server\_socket]
11. while True:
12. #调用select ()函数,阻塞等待
13. readable, writeablez exceptional = select»select(inputsf □/ 口)
14. #数据抵达，循环
15. for temp\_socket in readable:
16. #监疏到有新的连接
17. if temp\_socket == server\_socket:
18. new\_socket f client\_address = server\_\_socket accept ()
19. print (° 一个新的客户端到来:%s" % str (client\_address))
20. # se丄㊀ct 监听的 temp\_socket
21. inputs o append(new\_socket)
22. #有数据到达 —
23. else:
24. #读取客户端连接发送的数据
25. data = temp\_socket.recv(1024)
26. #若有数据递五，对数据进行处理
27. if data:
28. data = data.upper()
29. temp\_socket o send(data)
30. #若未接收到数据，断开连接
31. else:
32. #从监听列表中移除对应的temp\_socket
33. inputs.remove(temp\_socket)
34. temp\_socket.close()
35. server\_socket close()
36. if name == 1 main 1 :
37. main()

此段程序搭建的服务器只对读就绪套接字进行了处理：

程序中的第10行代码创建了等待读就绪的套接字列表，并使用用于监测客户端连接请求 的套接字server\_socket初始化该列表。客户端的连接请求相当于客户端发送到服务器套接字 server\_socket中的数据，若有客户端发送了连接请求，服务器套接字中将会接收 到数据，并转变为读就绪状态。

程序中的第13行代码调用龄1心监测服务器端各种套接字列表的状态，经此步骤后，read^ able列表中将存储读就绪的套接字，writeable列表中将存储写就绪的套接字，exceptional列表 中将存储捕获到异常的套接字。

之后程序对可读套接字列表中的套接字逐个进行处理：若可读套接字列表中包含服务器套 接字server\_socket9说明有新的客户端连接请求到达，此时调用accept方法对其进行处理，并 将accept方法返回的套接字new\_socket添加到等待读就绪的套接字列表inputs中；若可读套接 字列表中含有非服务器套接字的其他套接字，说明有已建立连接的客户端程序向服务器发送了 数据，此时服务器应通过该套接字接收客户端程序发送的数据，并根据数据的长度执行不同的 操作。

为了测试该程序的功能，我们需实现一个用于发送数据和接收数据处理结果的客户端程序， 客户端程序代码如下：

1. import socket
2. def main():
3. #「创建套接字 server\_socket
4. client\_socket = socket.socket(socket.AF\_INETZ socket.SOCK\_STREAM)
5. # 2•癖连接
6. client\_socket.connect((,172.16.43.31,z 7788))
7. # 3.点数据
8. while True:
9. data = input ( 待处理数据一 -\n° )
10. client\_socket.send(data.encode(\* gb2312'))
11. recv\_inf o = client\_socketrecv (1024)。decode ( ' gb2312 1 )
12. print ("一 处理结果 ™--\n%sn%recv\_info)
13. # 4 .关闭聊天室套接字server\_socket
14. server\_\_socket. close ()
15. if name == ? main ':
16. main()

启动服务器之后再启动客户端，服务器中将打印客户端的地址信息，本次测试共启动了4

个客户端，客户端都启动后，服务器中打印的信息如下:

一个新的客户端到来:(?172.16.43.31', 9668)

一个新的客户端到来:('172.16^43.31\ 10269)

一个新的客户端到来:(,172.16.43.31 \ 10275)

一个新的客户端到来:(?172.16.43.31 \ 10281)

由此可知，本小节搭建的基于select模型的TCP服务器可与多个客户端建立连接。 使用客户端向服务器中发送数据，客户端中打印的信息如下所示。

待处理数据

hello itheima

一一一处理结果

HELLO ITHEIMA

一一-待处理数据一一一

由此可知，本小节搭建的基于select模型的TCP服务器可成功接收客户端发送的数据，并 将数据的处理结果返回给客户端。

综上所述，基于select模型的TCP并发服务器成功实现。

sdLect会先在遍历中获取到所有已就绪套接字列表，之后对列表中的套接字进行操作，只 有当本次获取到的所有已就绪套接字处理完毕后，select模式的服务器才会再次遍历所有连接。 与单进程非阻塞服务器相比，若服务器中的不活跃连接较多，select可以避免大量无用轮询， 以此提高服务器的效率。

需要说明的是，select服务器能监测的套接字数量相当有限，在Linux系统中，其上限一

般为1 024,用户可以通过修改宏定义、重新编译内核的方式打破这一限制，但select 器的

效率又会随着其监测连接数量的增多而降低。

18J2.2 ®p©B! 、

单进程非阻塞服务器和select服务器都以轮询的方式获取就绪套接字，当服务器程序中的 连接数较多时，一趟轮询便会耗费大量的时间，月艮务器的效率也因此而逐渐降低。

在信息科技飞速发展的今天，一个小型网站的并发量已远远不止1 024,显然单进程非阻 塞服务器和select服务器都很难满足服务器搭建的实际需求。为了克服select服务器连接数量 的限制，人们研发了一种名为poll的服务器，然而须这种服务器除了能解决连接上限的问题外， 其他方面与select基本没有区别，它的工作效率同样会随着连接数量的增高而越低。

那么是否有连接数量与效率兼得的服务器呢？答案是肯定的。

©poll服务器是Linux系统中常用的一种高效服务器，这种服务器采用事件通知机制，事 先为要建立连接的socket注册事件，一旦该socket就绪，注册事件将被触发，socket将被加入 epoll的就绪套接字列表，而服务器无须主动监测所有套接字状态，只需直接获取就绪套接字列 表，对其中的套接字进行处理即可。

Python中的©poll模式定义在select模块中，select中包含一个名为epoll的类，用户可先 在程序中创建epoll对象，再通过epoll对象的方法实现epoll模式。

在程序中定义epoll对象的方法如下:

import select

epoll = select.epoll()

©poll模式中包含了两个重要操作，一是事件注册，二是就绪套接字获取，这两个重要操作 分别通过epoll对象的register方法和poll方法实现。

1. register 方法

register方法的功能是为其参数fd创建注册事件，该方法的声明如下:

register(fd[7 eventmask])

register方法中的第一个参数fd是一个文件描述符。实际上epoll并非针对套接字，而是针 对套接字文件描述符的注册事件，在处理数据之前，叩。11获取的也并非套接字，而是文件描述 符的集合。

文件描述符是Linux操作系统中定义的、用于在进程或主机中标识唯一确定的一个已打开 文件的符号。在形式上文件描述符是一个非负整数，当程序打开一个文件时，内核就会向进程 返回一个文件描述符。

Limix将套接字视为一种特殊文件，Python中的套接字对象包含一个文件描述符属性。在 Python中创建一个套接字对象后，可观察到套接字对象中的fd属性，用户可通过套接字方法 fileno获取其中的fd属性,示例如下:

>>> demo socket = socket.socket(socket.AF\_INETZ socket.SOCK\_STREAM)

>>> socket

<socket. socket f d=3 r f ami 1 y：=AddressFami 1 y. AF\_INET7

type==SocketKindo SOCK STREAM, proto=0, laddr= ( 1 0.0.0.0 5 , 0) >

>>> demo\_socket .fileno ()

3

一个进程可打开的文件描述符数量是有限的，默认该上限为1024,在程序启动的同时，进 程会打开3个标准文件(标准输入文件、标准输出文件、标准错误文件)，这3个标准文件又 占用0、1、2这3个文件描述符，因此用户在程序中打开文件的文件描述符最小为3。

register方法的第二个参数eventmask是一个可选参数，该参数用于设置epoll要监控的事 件和©poll的工作模式，事件是由耶)11常量组成的位，其默认值为EPOLLIN | EPOLLOUT | EPOLLPRL表示同时监控fd的读事件、写事件和紧急可读事件。此外，该参数可用的©poll 常量及其表示的含义分别如下:

。EPOLLERR表示监控fd的错误事件。

◎ EPOLLHUP表示监控fd的挂断事件。

。EPOLLET表示将©poll设置为边缘触发(EdgeTriggered)模式。

。EPOLLONESHOT表示只监听一次事件，当此次事件监听完成后，若要再次监听该fd, 需将其再次添加到©poll队列中*。*

register方法没有返回值，epoll中有一个register相对的方法 unregister,这个方法用于

将文件描述符从监听列表中移除。

1. poll方法

poll方法的功能是查询epoll对象，判断是否有©poll关注的事件被触发。poll方法的声明 如下：

poll([timeout=-l[7 maxevents=-l]])

poll中的参数都可缺省，其中血wout用于设置等待时长，其默认值-1表示无限等待。若 在调用p血方法前已有©poll监测的事件发生，此次查询会立刻返回一个列表，该列表中的元 素为形如(fd,event code)的元组。

下面以大小写转换为例，搭建基于epoll模式的TCP服务器，服务器的具体代码如下:

1. import socket
2. import select
3. def main():
4. #创建套接字
5. server\_socket = socket.socket(socket.AF\_INET, socket.SOCK\_STREAM)
6. #绑定末机信息 ~ —
7. server\_socket.bind((""z 8080))
8. #重复夜用绑定的信息
9. server\_socket.setsockopt(socket.SOL\_SOCKETZ socket.SO\_REUSE：ADDRZ 1)
10. #变为萩动 *一 ~*
11. server\_socket.listen(10)
12. #设置套接字为非阻塞模式
13. server\_socket.setblocking(False)
14. #创建一个epo丄丄对象
15. epoll = select.epoll()
16. #为服务器端套接字server\_socket的文件描述符注册事件
17. epoll。register (server\_socket .fileno () z

selecteEPOLLIN | select.EPOLLET) new\_socket\_list = {} client\_\_address\_list = { )

18

19

20

21

22

23

24

25

#循环等待数据到决

while True:

#检测并获取epoll监控的已触发事件 epoll\_list = epollOpoll()

#对事律进行处理

|  |  |  |
| --- | --- | --- |
| 26 | for fdz events in epoll\_list: | |
| 27 | # | 如果有新的连接请求递达 |
| 28 | if | fd == server\_socket .fileno (): |
| 29 |  | new\_socket, client\_address = server\_\_socket.accept() |
| 30 |  | print ('有新的客户端到来 %s 5 %str (client\_address)) |
| 31 |  | #保存新客户端的套接字信息和地址信息 一 |
| 32 |  | new\_socket\_list [new\_\_socket .fileno () ] = new\_socket |
| 33 |  | client\_address\_list[new\_socket.fileno()] = client\_address |
| 34 |  | #为新套骚字的文存描述符注际读事件 — |
| 35 |  | epoll. register (new\_socket ofileno () f |
| 36 |  | select.EPOLLIN | select.EPOLLET) |
| 37 | elif events == select.EPOLLIN: | |
| 38 |  | #从new\_socket触发的事件 |
| 39 |  | recv\_data = new\_socket\_list[fd].recv(1024) |
| 40 |  | #若薮据长度大于0；处理数羸 |
| 41 |  | if len(recv\_data) > 0: |
| 42 |  | print ( 1 待处理数据 %s \* %recv\_data . decode ( f gb2312 5 )) |
| 43 |  | recv\_data = recv\_data s upper() |
| 44 |  | new\_socket.send(recv\_data) |
| 45 |  | #若数据R度为o,关闭连接 一 |
| 46 |  | else : |
| 47 |  | #从epoll中移除fd |
| 48 |  | epoll.unregister(fd) |
| 49 |  | #关闭服务器端为该连接创建的套接字 |
| 50 |  | new\_socket\_list[fd].close() |
| 51 |  | print ("%s——offline—% str (client\_address\_list [fd])) |
| 52 if | name == | \* main 5: |

53 main()

第19、20两行代码创建了字典new\_socket\_list和client\_address\_list 9分另0用于存储与客户 端交互的套接字信息和客户端地址；第32、33两行代码以套接字的文件描述符作为key值， 将新客户端的套接字信息和地址信息存储到T new socket list和client address list中。

为了测试该程序的功能，我们需实现一个可向服务器发送数据并接收服务器反馈信息的客 户端程序，客户端程序代码如下：

1. from socket import \*
2. def main():
3. client\_socket = socket (AF\_\_INET, SOCK\_STREAM)
4. client\_socket.connect((\* 192 »168 o 255 o144 ? r 8 0 8 0))
5. while True:
6. data = input ( ?-——一待处理数据一 ")
7. client socket.send(data.encode(\* gb2312 ?))
8. recv\_info = client\_socket.recv(1024) » decode(1gb2312 f)

1. print ("一 处理结果 —\n%s ! %recv\_info)
2. server\_\_socket s close ()
3. if name == ' main \* :
4. main()

启动服务器之.后再启动客户端，服务器中将打印客户端的地址信息，本次测试共启动了 4 个客户端，客户端都启动后，服务器中打印的信息如下：

有新的客户端到来:(f192.168.255.144 \ 10268) 有新的客户端到来:(U92.168^255.144 \ 10269)

有新的客户端到来:(9192.168.255O144\ 10275)

有新的客户端到来:(?192.168.255.144 \ 10281)

由此可知，本小节搭建的基于epoll模型的TCP服务器可同时与多个客户端建立连接。 使用客户端向服务器中发送数据，客户端中打印的信息如下所示。

待处理数据-一一-

hello itheima

处理结果

HELLO 工THE工MA

-一一待处理数据-一---

由此可知，本小节搭建的基于epoll模型的TCP服务器可成功接收客户端发送的数据，并 将数据的处理结果返回给客户端。

综上所述，可知基于epoll模型的TCP并发服务器成功实现。