第十六章 网络编程

16.1 介绍

硬件的客户 / 服务器架构 — 打印机，Network File System(NFS)
软件客户 / 服务器架构 —软件服务器提供的服务主要是程序的运行，数据的发送与接收，合并，升级或其它的程序或数据的操作。

> Web服务器，数据库服务器，窗口服务器(GUI)–Windows中的explorer.exe

16.2 套接字:通讯端点

套接字起源于 20 世纪 70 年代加利福尼亚大学伯克利分校版本的 Unix，即人们所说的 BSD Unix。
因此，有时人们也把套接字称为`伯克利套接字`或`BSD 套接字`。

基于文件型的套接字

一开始，套接字被设计用在同一台主机上多个应用程序之间的通讯。这也被称进程间通讯，或 `IPC`。
Unix套接字是第一个套接字家族。家族名为 `AF_UNIX` (在 POSIX1.g 标准中也叫 `AFLOCAL`)，表示地址家族。这两者基本等价。
Python自己仍然使用 `AF_UNIX`。

由于两个进程都运行在同一台机器上，而且这些套接字是基于文件的。所以，它们的底层结构是由文件系统来支持的。

基于网络型的套接字

另一种套接字是基于网络的，它有自己的家族名字： `AF_INET` ，或叫“地址家族： `Internet`。
还有一种地址家族 `AF_INET6` 被用于网际协议第 6 版（ IPv6 ）寻址上。
Python 只支持 `AF_UNIX` ， `AF_NETLINK` ，和 `AF_INET` 家族。

常用端口号列表
http://www.iana.org/assignments/port-numbers

16.2.3 面向连接与无连接

套接字的类型只有两种，面向连接的(TCP)和无连接的(UDP)。
这么看来，Socket套接字是对TCP/IP和UDP/IP这些底层协议的封装。
一种是面向连接的套接字，即在通讯之前一定要建立一条连接。
这种通讯方式也被称为“**虚电路**”或“**流套接字**”。

面向连接的通讯方式提供了**顺序的，可靠的，不会重复的数据传输**而且也不会被加上数据边界。
每一个要发送的信息，可能会被拆分成多份，每一份都会不多不少地正确到达目的地。

面向连接– `TCP/IP`

实现这种连接的主要协议就是传输控制协议(即 TCP)。
要创建 TCP 套接字就得在创建的时候，指定套接字类型为 `SOCK_STREAM`。

由于这些套接字使用 Internet 协议（IP）来查找网络中的主机，这样形成的整个系统，一般会由这两个协议（TCP 和 IP）来提及，即 `TCP/IP`。

无连接– `UDP/IP`

与虚电路完全相反的是数据报型的无连接套接字。

数据到达的顺序，可靠性及数据不重复性就无法保证了。
数据报会保留数据边界，这就表示，数据不会像面向连接的协议那样被拆分成小块。

由于面向连接套接字要提供一些保证，以及要维持虚电路接，这都是很重的额外负担。

实现这种连接的主要协议就是用户数据报协议–UDP。
要创建 UDP 套接字就得在创建的时候，指定套接字类型为 `SOCK_DGRAM`(来自于datagram–数据报)。

由于这些套接字使用 Internet 协议来查找网络中的主机，这样形成的整个系统，一般会由这两个协议（UDP 和 IP）来提及，即 `UDP/IP`。

16.3 Python中的网络编程

主要使用 socket 模块。模块中的 socket()函数被用来创建套接字。套接字也有自己的一套函数来提供基于套接字的网络通讯。

使用 `socket.socket()` 函数来创建套接字
`socket(socket_family,socket_type,protocol=0)`
创建一个 `TCP/IP` 的套接字
`tcpSock = socket(socket.AF_INET, sock.SOCK_STREAM)`
创建一个 `UDP/IP` 的套接字
`udpSock = socket.socket(socket.AF_INET, socket.SOCK_DGRAM)`

一个简单的（单线程的）服务器会调用 accept()函数等待连接的到来。默认情况下，`accept()`函数是阻塞式的，即程序在连接到来之前会处于挂起状态。

一旦接收到一个连接，accept()函数就会返回一个单独的客户的套接字用于后续的通讯。使用新的客户套接字就像把客户的电话转给一个客户服务人员。当一个客户打电话进来的时候，总机接了电话，然后把电话转到合适的人那里来处理客户的需求。

> 核心提示：创建线程来处理客户的请求。我们不打算在例子实现这样的功能。但是，创建一个新的线程或进程来完成与客户的通讯是一种非常常用的手段。SocketServer 模块是一个基于 socket 模块的高级别的套接字通讯模块，它支持在新的线程或进程中处理客户的请求。

16.3.6 创建一个`TCP/UDP`服务器/客户端

16.4 *SocketServer模块

SocketServer 是标准库中一个高级别的模块。用于简化网络客户与服务器的实现。

SocketServer 将很多繁杂的事情已经被封装好了，你不用再去关心那个样板代码了。

事件包含发送与接收数据两种。在类定义中只包含了接收客户消息的事件处理器。
其它的功能从我们所使用的 SocketServer 继承而来。

16.5 Twisted框架介绍

http://twistedmatrix.com
动手为主

16.6 相关模块

`select`模块通常在底层套接字程序中与 socket 模块联合使用。它提供的 select()函数可以同时管理多个套接字对象。
它提供了哪些套接字已经准备好可以开始读取的集合。

`async*`和 `SocketServer` 模块在创建服务器方面都提供了高层次的功能。由于是基于 socket 和（或）select 模块，封装了所有的底层的代码，它们使得你可以快速开发客户/服务器的系统。

`async*`是标准库提供的唯一的异步开发支持库。
`socket`–底层网络接口
`asyncore/asynchat`–为能异步处理客户请求的网络应用提供底层功能
`select`–在单线程网络服务器程序中，管理多个套接字连接
`SocketServer`–包含了写网络应用程序所需的高级别模块。提供完整的进程和线程的版本。

练习

16-1

```
面向连接:通讯之前在客户端/服务端建立连接，可提供顺序的，可靠的，不会重复的数据传输。实现这样连接的主要协议是TCP。
无连接:客户端/服务端无需建立连接，因此数据到达的顺序，可靠性和数据不重复得不到保证。
```

16-2

```
服务器:服务器端一直运行着程序，等到客户端的请求，当请求过来时进行处理，并将处理的结果(页面)返回给客户端(浏览器/App)
客户端:想服务器端发送请求，并等待接收服务器端发送过来的处理结果，并将结果显示在客户端(浏览器/App)
```

16-3

```
TCP
```

16-9

Server

```
from socket import *
from time import ctime
import threading

HOST = ''
PORT = 21567
BUFSIZE = 1024
ADDR = (HOST, PORT)

def Deal(sck, username):
    while True:
        data = sck.recv(BUFSIZE)
        print 'received from [',username,']--',data
        if data == "quit":
            del clients[username]
            sck.send(data)
            sck.close()
            break
        for i in  clients.iterkeys():
            if i != username:
                clients[i].send("[%s] %s: %s" %(ctime(), username, data))


chatSerSock = socket(AF_INET, SOCK_STREAM)
chatSerSock.bind(ADDR)
chatSerSock.listen(5)
#www.iplaypython.com

clients = {}

while True:
```

```
        print 'waiting for connection...'
        chatCliSock, addr = chatSerSock.accept()
        print "...connected romt: ", addr
        username = chatCliSock.recv(BUFSIZE).decode()
        print '用户',username,'登录成功'
        if clients.has_key(username):
            chatCliSock.send("reuse".encode())
            chatCliSock.close()
        else:
            chatCliSock.send("success".encode())
            clients[username] = chatCliSock
            # 以下是多线程部分
            t = threading.Thread(target=Deal, args=(chatCliSock, username))
            t.start()

    chatSerSock.close()
```

Client

```
from socket import *
from time import ctime
import threading
import random
from sys import argv, exit, stdout
from getopt import gnu_getopt, GetoptError

help_info = ["cs.py [ -h | --help | -u | --username] username",
    "\t-h or --help\t显示帮助信息",
    "\t-u or --username\t指定用户名"]
def help():
    for i in help_info:
        print(i)

def Send(sck, test):
    while True:
        data = raw_input('> ')
        sck.send(data)
        print 'received from server[',test,']--',data
        if  data == "quit":
            break

def Recieve(sck, test):
    while True:
        data = sck.recv(BUFSIZ)
        print 'received from server',data
        if data == "quit":
            sck.close()
            break
        str = "\n" + data + "\n>"
        stdout.write(str)
```

```
HOST = '127.0.0.1' #localhost
PORT= 21567
BUFSIZ = 1024
ADDR = (HOST, PORT)
threads = []

if __name__ == "__main__":
    # 解析命令行参数
    # try:
    opts, args = gnu_getopt(argv[1:], "hu:", ["help", "username="])
    # except GetoptError,err:
    #     print(str(err))
    #     help()
    #     exit(2)
    username = "join" # 测试用 join
    print username
    # for o, a in opts:
    #     if o in ("-h", "--help"):
    #         help()
    #         exit(0)
    #     elif o in ("-u", "--username"):
    #         username = a
    #     else:
    #         print "未知选项"
    #         help()
    #         exit(2)

    # if username == "":
    #     help()
    #     exit(2)
```

```
        chatCliSock = socket(AF_INET, SOCK_STREAM)
        chatCliSock.connect(ADDR)
        chatCliSock.send(username.encode())
        data = chatCliSock.recv(BUFSIZ).decode()
        if data == "reuse":
            print "用户已经",username,"登录"
            raw_input()
            exit(1)
        elif data == "success":
            print "用户",username,"成功登录"
            # 以下是多线程部分
            t = threading.Thread(target=Send, args = (chatCliSock, None))
            threads.append(t)
            t = threading.Thread(target=Recieve, args = (chatCliSock, None))
            threads.append(t)
            for i in range(len(threads)):
                threads[i].start()
            threads[0].join()
```

16-11

```python
from socket import *

def tcpCli():
    HOST = 'www.jd.com'
    PORT = 80
    BUFSIZE = 1024
    ADDR = (HOST, PORT)

    tcpCliSocket = socket(AF_INET, SOCK_STREAM) # 创建TCP客户端套接字
    tcpCliSocket.connect(ADDR) # 连接服务器

    while True:
        tcpCliSocket.send("GET /\n\n")
        data = tcpCliSocket.recv(BUFSIZE)
        if not data:
            break
        print data
    tcpCliSocket.close()

if __name__ == "__main__":
    tcpCli()
```