[python网络编程学习初步](http://www.cppblog.com/lai3d/archive/2008/02/19/42919.html)

一、套接字

套接字是为特定网络协议（例如TCP/IP，ICMP/IP，UDP/IP等）套件对上的网络应用程序提供者提供当前可移植标准的对象。它们允许程序接受并进行连接，如发送和接受数据。为了建立通信通道，网络通信的每个端点拥有一个套接字对象极为重要。

套接字为BSD UNIX系统核心的一部分，而且他们也被许多其他类似UNIX的操作系统包括Linux所采纳。许多非BSD UNIX系统（如ms-dos，windows，os/2，mac os及大部分主机环境）都以库形式提供对套接字的支持。

三种最流行的套接字类型是:stream,datagram和raw。stream和datagram套接字可以直接与TCP协议进行接口，而raw套接字则接口到IP协议。但套接字并不限于TCP/IP。

二、套接字模块

套接字模块是一个非常简单的基于对象的接口，它提供对低层BSD套接字样式网络的访问。使用该模块可以实现客户机和服务器套接字。要在python 中建立具有TCP和流套接字的简单服务器，需要使用socket模块。利用该模块包含的函数和类定义，可生成通过网络通信的程序。一般来说，建立服务器连接需要六个步骤。

第1步是创建socket对象。调用socket构造函数。

socket=socket.socket(familly,type)

family的值可以是AF\_UNIX(Unix域，用于同一台机器上的进程间通讯)，也可以是AF\_INET（对于IPV4协议的TCP和 UDP），至于type参数，SOCK\_STREAM（流套接字）或者 SOCK\_DGRAM（数据报文套接字）,SOCK\_RAW（raw套接字）。

第2步则是将socket绑定（指派）到指定地址上，socket.bind(address)

address必须是一个双元素元组,((host,port)),主机名或者ip地址+端口号。如果端口号正在被使用或者保留，或者主机名或ip地址错误，则引发socke.error异常。

第3步，绑定后，必须准备好套接字，以便接受连接请求。

socket.listen(backlog)

backlog指定了最多连接数，至少为1，接到连接请求后，这些请求必须排队，如果队列已满，则拒绝请求。

第4步，服务器套接字通过socket的accept方法等待客户请求一个连接：

connection,address=socket.accept()

调用accept方法时，socket会进入'waiting'（或阻塞）状态。客户请求连接时，方法建立连接并返回服务器。accept方法返回 一个含有俩个元素的元组，形如(connection,address)。第一个元素（connection）是新的socket对象，服务器通过它与客 户通信；第二个元素（address）是客户的internet地址。

第5步是处理阶段，服务器和客户通过send和recv方法通信（传输数据）。服务器调用send，并采用字符串形式向客户发送信息。send方法 返回已发送的字符个数。服务器使用recv方法从客户接受信息。调用recv时，必须指定一个整数来控制本次调用所接受的最大数据量。recv方法在接受 数据时会进入'blocket'状态，最后返回一个字符串，用它来表示收到的数据。如果发送的量超过recv所允许，数据会被截断。多余的数据将缓冲于接 受端。以后调用recv时，多余的数据会从缓冲区删除。

第6步，传输结束，服务器调用socket的close方法以关闭连接。

建立一个简单客户连接则需要4个步骤。

第1步，创建一个socket以连接服务器 socket=socket.socket(family,type)

第2步，使用socket的connect方法连接服务器 socket.connect((host,port))

第3步，客户和服务器通过send和recv方法通信。

第4步，结束后，客户通过调用socket的close方法来关闭连接。

三、一个简单的服务器和客户端通信的例子

服务器：

import socket  
s=socket.socket()  
s.bind(('xxx.xxx.xxx.xxx',xxxx))    #ip地址和端口号  
s.listen(5)  
cs,address = s.accept()  
print 'got connected from',address  
cs.send('byebye')  
ra=cs.recv(512)  
print ra  
cs.close()

客户端：

import socket  
s=socket.socket()  
s.connect(('xxx.xxx.xxx.xxx',xxxx))   #与服务器程序ip地址和端口号相同  
data=s.recv(512)  
s.send('hihi')  
s.close()  
print 'the data received is',data

运行：

在本机测试（windows环境下，可以将ip地址改为本机ip，端口号在1024以上，windows将1024以下的为保留），运行--CMD--进入命令行模式

先python 服务器程序，后python 客户端程序即可。

或者启动服务器程序后，用telnet ip地址 端口号，也可以得到同样结果。

让server持续接受连接

server.py

import socket  
s=socket.socket()  
s.bind(('192.168.43.137',2000))  
s.listen(5)  
  
while 1:  
    cs,address = s.accept()  
    print 'got connected from',address  
    cs.send('hello I am server,welcome')  
    ra=cs.recv(512)  
    print ra  
    cs.close()

测试两个一个程序中两个socket并存是否可行  
  
client.py

import socket  
s=socket.socket()  
s.connect(('192.168.43.137',2000))     
data=s.recv(512)  
print 'the data received is\n    ',data  
s.send('hihi I am client')  
  
sock2 = socket.socket()  
sock2.connect(('192.168.43.137',2000))  
data2=sock2.recv(512)  
print 'the data received from server is\n   ',data2  
sock2.send('client send use sock2')  
sock2.close()  
  
s.close()

[主题：Python网络编程的一般步骤](http://www.javaeye.com/topic/492649)

Python提供流套接字(tcp)和数据报套接字(udp)。urlparse模块提供了用于解析url的函数，以及用于url处理的函数。   
  
1. 服务器端   
要在Python中建立具有TCP和流套接字的简单服务器，需要使用socket模块。利用该模块包含的函数和类定义，可生成通过网络通信的程序。建立这个连接需要6个步骤：   
a). 第一步是创建socket对象。调用socket构造函数。如：socket = socket.socket(family, type)   family参数代表地址家族，可为AF\_INET或AF\_UNIX。AF\_INET家族包括Internet地址，AF\_UNIX家族用于同一台机器上 的进程间通信。type参数代表套接字类型，可为SOCK\_STREAM(流套接字)和SOCK\_DGRAM(数据报套接字)。   
  
b). 第二步是将socket绑定到指定地址。这是通过socket对象的bind方法来实现的：socket.bind(address)   由AF\_INET所创建的套接字，address地址必须是一个双元素元组，格式是(host,port)。host代表主机，port代表端口号。如果端口号正在使用、主机名不正确或端口已被保留，bind方法将引发socket.error异常。   
  
c). 第三步是使用socket套接字的listen方法接收连接请求。socket.listen(backlog)   backlog指定最多允许多少个客户连接到服务器。它的值至少为1。收到连接请求后，这些请求需要排队，如果队列满，就拒绝请求。   
  
d). 第四步是服务器套接字通过socket的accept方法等待客户请求一个连接。connection, address = socket.accept()   调用accept方法时，socket会进入“waiting”状态。客户请求连接时，方法建立连接并返回服务器。accept方法返回一个含有两个元素 的元组(connection,address)。第一个元素connection是新的socket对象，服务器必须通过它与客户通信；第二个元素 address是客户的Internet地址。   
  
e). 第五步是处理阶段，服务器和客户端通过send和recv方法通信(传输数据)。服务器调用send，并采用字符串形式向客户发送信息。send方法返回 已发送的字符个数。服务器使用recv方法从客户接收信息。调用recv 时，服务器必须指定一个整数，它对应于可通过本次方法调用来接收的最大数据量。recv方法在接收数据时会进入“blocked”状态，最后返回一个字符 串，用它表示收到的数据。如果发送的数据量超过了recv所允许的，数据会被截短。多余的数据将缓冲于接收端。以后调用recv时，多余的数据会从缓冲区 删除(以及自上次调用recv以来，客户可能发送的其它任何数据)。   
  
f). 第六步，传输结束，服务器调用socket的close方法关闭连接。   
  
2. 客户端   
在Python中建立一个简单客户端需要4个步骤：   
a). 第一步是创建一个socket以连接服务器：socket = socket.socket(family, type)   
  
b). 第二步是使用socket的connect方法连接服务器。对于AF\_INET家族,连接格式如下：socket.connect(host, port)   host代表服务器主机名或IP，port代表服务器进程所绑定的端口号。如连接成功，客户就可通过套接字与服务器通信，如果连接失败，会引发 socket.error异常。   
  
c). 第三步是处理阶段，客户和服务器将通过send方法和recv方法通信。   
  
d). 第四步，传输结束，客户通过调用socket的close方法关闭连接。

Python socket 模块

Python 提供了两个基本的 socket 模块。第一个是 Socket，它提供了标准的 BSD Sockets API。第二个是 SocketServer，它提供了服务器中心类，可以简化网络服务器的开发。Python 使用一种异步的方式来实现这种功能，您可以提供一些插件类来处理服务器中应用程序特有的任务。表 1 列出了本节所涉及的类和模块。

|  |  |
| --- | --- |
| *表 1. Python 类和模块* | |
| **类/模块** | **说明** |
| Socket | 低层网络接口（每个 BSD API） |
| SocketServer | 提供简化网络服务器开发的类 |

让我们来看一下这些模块，以便理解它们是如何工作的。

|  |
| --- |
|  |

|  |
| --- |
|  |

Socket 模块提供了 UNIX® 程序员所熟悉的基本网络服务（也称为 *BSD API*）。这个模块中提供了在构建 socket 服务器和客户机时所需要的所有功能。

这个 API 与标准的 C API 之间的区别在于它是面向对象的。在 C 中，socket 描述符是从 socket 调用中获得的，然后会作为一个参数传递给 BSD API 函数。在 Python 中，socket 方法会向应用 socket 方法的对象返回一个 socket 对象。表 2 给出了几个类方法，表 3 显示了一部分实例方法。

|  |  |
| --- | --- |
| *表 2. Socket 模块的类方法* | |
| **类方法** | **说明** |
| Socket | 低层网络接口（每个 BSD API） |
| socket.socket(family, type) | 创建并返回一个新的 socket 对象 |
| socket.getfqdn(name) | 将使用点号分隔的 IP 地址字符串转换成一个完整的域名 |
| socket.gethostbyname(hostname) | 将主机名解析为一个使用点号分隔的 IP 地址字符串 |
| socket.fromfd(fd, family, type) | 从现有的文件描述符创建一个 socket 对象 |

|  |  |
| --- | --- |
| *表 3. Socket 模块的实例方法* | |
| **实例方法** | **说明** |
| sock.bind( (adrs, port) ) | 将 socket 绑定到一个地址和端口上 |
| sock.accept() | 返回一个客户机 socket（带有客户机端的地址信息） |
| sock.listen(backlog) | 将 socket 设置成监听模式，能够监听 *backlog* 外来的连接请求 |
| sock.connect( (adrs, port) ) | 将 socket 连接到定义的主机和端口上 |
| sock.recv( buflen[, flags] ) | 从 socket 中接收数据，最多 buflen 个字符 |
| sock.recvfrom( buflen[, flags] ) | 从 socket 中接收数据，最多 buflen 个字符，同时返回数据来源的远程主机和端口号 |
| sock.send( data[, flags] ) | 通过 socket 发送数据 |
| sock.sendto( data[, flags], addr ) | 通过 socket 发送数据 |
| sock.close() | 关闭 socket |
| sock.getsockopt( lvl, optname ) | 获得指定 socket 选项的值 |
| sock.setsockopt( lvl, optname, val ) | 设置指定 socket 选项的值 |

*类方法* 和 *实例方法* 之间的区别在于，实例方法需要有一个 socket 实例（从 socket 返回）才能执行，而类方法则不需要。

1、服务端-echoserver.py

#!/usr/bin/python

"USAGE: echoserver.py <port>"

from SocketServer import BaseRequestHandler, TCPServer

import sys, socket

class EchoHandler(BaseRequestHandler):

def handle(self):

print "Client connected:", self.client\_address

self.request.sendall(self.request.recv(2\*\*16))

self.request.close()

if len(sys.argv) != 2:

print \_\_doc\_\_

else:

TCPServer(('', int(sys.argv[1])), EchoHandler).serve\_forever()

2、客户端-echoclient.py

#!/usr/bin/python

"USAGE: echoclient.py <server> <word> <port>"

from socket import \*

import sys

if len(sys.argv) != 4:

print \_\_doc\_\_

sys.exit(0)

sock = socket(AF\_INET, SOCK\_STREAM)

sock.connect((sys.argv[1], int(sys.argv[3])))

message = sys.argv[2]

messlen, received = sock.send(message), 0

if messlen != len(message):

print "Failed to send complete message"

print "Received: ",

while received < messlen:

data = sock.recv(32)

sys.stdout.write(data)

received += len(data)

print

sock.close()

**python网络编程**

一、套接字套接字是为特定网络协议（例如TCP/IP，ICMP/IP，UDP/IP等）套件对上的网络应用程序提供者提供当前可移植标准的对象。它们允许程序接受并进行连接，如发送和接受数据。为了建立通信通道，网络通信的 ...

一、套接字

套接字是为特定网络协议（例如TCP/IP，ICMP/IP，UDP/IP等）套件对上的网络应用程序提供者提供当前可移植标准的对象。它们允许程序接受并进行连接，如发送和接受数据。为了建立通信通道，网络通信的每个端点拥有一个套接字对象极为重要。

套接字为BSD UNIX系统核心的一部分，而且他们也被许多其他类似UNIX的操作系统包括Linux所采纳。许多非BSD UNIX系统（如ms-dos，windows，os/2，mac os及大部分主机环境）都以库形式提供对套接字的支持。

三种最流行的套接字类型是:stream,datagram和raw。stream和datagram套接字可以直接与TCP协议进行接口，而raw套接字则接口到IP协议。但套接字并不限于TCP/IP。

二、套接字模块

套接字模块是一个非常简单的基于对象的接口，它提供对低层BSD套接字样式网络的访问。使用该模块可以实现客户机和服务器套接字。要在python 中建立具有TCP和流套接字的简单服务器，需要使用socket模块。利用该模块包含的函数和类定义，可生成通过网络通信的程序。一般来说，建立服务器连接需要六个步骤。

第1步是创建socket对象。调用socket构造函数。

socket=socket.socket(familly,type)

family的值可以是AF\_UNIX(Unix域，用于同一台机器上的进程间通讯)，也可以是AF\_INET（对于IPV4协议的TCP和 UDP），至于type参数，SOCK\_STREAM（流套接字）或者 SOCK\_DGRAM（数据报文套接字）,SOCK\_RAW（raw套接字）。

第2步则是将socket绑定（指派）到指定地址上，socket.bind(address)

address必须是一个双元素元组,((host,port)),主机名或者ip地址+端口号。如果端口号正在被使用或者保留，或者主机名或ip地址错误，则引发socke.error异常。

第3步，绑定后，必须准备好套接字，以便接受连接请求。

socket.listen(backlog)

backlog指定了最多连接数，至少为1，接到连接请求后，这些请求必须排队，如果队列已满，则拒绝请求。

第4步，服务器套接字通过socket的accept方法等待客户请求一个连接：

connection,address=socket.accept()

调用accept方法时，socket会进入'waiting'（或阻塞）状态。客户请求连接时，方法建立连接并返回服务器。accept方法返回 一个含有俩个元素的元组，形如(connection,address)。第一个元素（connection）是新的socket对象，服务器通过它与客 户通信；第二个元素（address）是客户的internet地址。

第5步是处理阶段，服务器和客户通过send和recv方法通信（传输数据）。服务器调用send，并采用字符串形式向客户发送信息。send方法 返回已发送的字符个数。服务器使用recv方法从客户接受信息。调用recv时，必须指定一个整数来控制本次调用所接受的最大数据量。recv方法在接受 数据时会进入'blocket'状态，最后返回一个字符串，用它来表示收到的数据。如果发送的量超过recv所允许，数据会被截断。多余的数据将缓冲于接 受端。以后调用recv时，多余的数据会从缓冲区删除。

第6步，传输结束，服务器调用socket的close方法以关闭连接。

建立一个简单客户连接则需要4个步骤。

第1步，创建一个socket以连接服务器 socket=socket.socket(family,type)

第2步，使用socket的connect方法连接服务器 socket.connect((host,port))

第3步，客户和服务器通过send和recv方法通信。

第4步，结束后，客户通过调用socket的close方法来关闭连接。

三、一个简单的服务器和客户端通信的例子

服务器：

import socket  
s=socket.socket()  
s.bind(('xxx.xxx.xxx.xxx',xxxx))    #ip地址和端口号  
s.listen(5)  
cs,address = s.accept()  
print 'got connected from',address  
cs.send('byebye')  
ra=cs.recv(512)  
print ra  
cs.close()

客户端：

import socket  
s=socket.socket()  
s.connect(('xxx.xxx.xxx.xxx',xxxx))   #与服务器程序ip地址和端口号相同  
data=s.recv(512)  
s.send('hihi')  
s.close()  
print 'the data received is',data

运行：

在本机测试（windows环境下，可以将ip地址改为本机ip，端口号在1024以上，windows将1024以下的为保留），运行--CMD--进入命令行模式

先python 服务器程序，后python 客户端程序即可。

或者启动服务器程序后，用telnet ip地址 端口号，也可以得到同样结果。

--------------------------------------------------------------------------------  
让server持续接受连接

server.py

import socket  
s=socket.socket()  
s.bind(('192.168.43.137',2000))  
s.listen(5)

while 1:  
    cs,address = s.accept()  
    print 'got connected from',address  
    cs.send('hello I am server,welcome')  
    ra=cs.recv(512)  
    print ra  
    cs.close()

测试两个一个程序中两个socket并存是否可行

client.py  
import socket  
s=socket.socket()  
s.connect(('192.168.43.137',2000))     
data=s.recv(512)  
print 'the data received is\n    ',data  
s.send('hihi I am client')

sock2 = socket.socket()  
sock2.connect(('192.168.43.137',2000))  
data2=sock2.recv(512)  
print 'the data received from server is\n   ',data2  
sock2.send('client send use sock2')  
sock2.close()

s.close()

python 编写server的步骤：

第一步是创建socket对象。调用socket构造函数。如：

socket = socket.socket( family, type )

family参数代表地址家族，可为AF\_INET或AF\_UNIX。AF\_INET家族包括Internet地址，AF\_UNIX家族用于同一台机器上的进程间通信。  
type参数代表套接字类型，可为SOCK\_STREAM(流套接字)和SOCK\_DGRAM(数据报套接字)。

第二步是将socket绑定到指定地址。这是通过socket对象的bind方法来实现的：

socket.bind( address )

由AF\_INET所创建的套接字，address地址必须是一个双元素元组，格式是(host,port)。host代表主机，port代表端口号。如果端口号正在使用、主机名不正确或端口已被保留，bind方法将引发socket.error异常。

第三步是使用socket套接字的listen方法接收连接请求。

socket.listen( backlog )

backlog指定最多允许多少个客户连接到服务器。它的值至少为1。收到连接请求后，这些请求需要排队，如果队列满，就拒绝请求。

第四步是服务器套接字通过socket的accept方法等待客户请求一个连接。

connection, address = socket.accept()

调 用accept方法时，socket会时入“waiting”状态。客户请求连接时，方法建立连接并返回服务器。accept方法返回一个含有两个元素的 元组(connection,address)。第一个元素connection是新的socket对象，服务器必须通过它与客户通信；第二个元素 address是客户的Internet地址。

第五步是处理阶段，服务器和客户端通过send和recv方法通信(传输 数据)。服务器调用send，并采用字符串形式向客户发送信息。send方法返回已发送的字符个数。服务器使用recv方法从客户接收信息。调用recv 时，服务器必须指定一个整数，它对应于可通过本次方法调用来接收的最大数据量。recv方法在接收数据时会进入“blocked”状态，最后返回一个字符 串，用它表示收到的数据。如果发送的数据量超过了recv所允许的，数据会被截短。多余的数据将缓冲于接收端。以后调用recv时，多余的数据会从缓冲区 删除(以及自上次调用recv以来，客户可能发送的其它任何数据)。  
传输结束，服务器调用socket的close方法关闭连接。  
python编写client的步骤：

创建一个socket以连接服务器：socket = socket.socket( family, type )  
使用socket的connect方法连接服务器。对于AF\_INET家族,连接格式如下：

socket.connect( (host,port) )

host代表服务器主机名或IP，port代表服务器进程所绑定的端口号。如连接成功，客户就可通过套接字与服务器通信，如果连接失败，会引发socket.error异常。

处理阶段，客户和服务器将通过send方法和recv方法通信。  
传输结束，客户通过调用socket的close方法关闭连接。  
下面给个简单的例子：  
server.py

python 代码

|  |  |
| --- | --- |
| 1 2 3 4 5 6 7 8 9 10 11 12 13 14 15 16 17 | if \_\_name\_\_ == '\_\_main\_\_':        import socket        sock = socket.socket(socket.AF\_INET, socket.SOCK\_STREAM)        sock.bind(('localhost', 8001))        sock.listen(5)        while True:            connection,address = sock.accept()            try:                connection.settimeout(5)                buf = connection.recv(1024)                if buf == '1':                    connection.send('welcome to server!')                else:                    connection.send('please go out!')            except socket.timeout:                print 'time out'            connection.close() |

client.py

|  |  |
| --- | --- |
| 1 2 3 4 5 6 7 8 9 | if \_\_name\_\_ == '\_\_main\_\_':        import socket        sock = socket.socket(socket.AF\_INET, socket.SOCK\_STREAM)        sock.connect(('localhost', 8001))        import time        time.sleep(2)        sock.send('1')        print sock.recv(1024)        sock.close() |