Python套接字编程

摘要：使用Python基本套接字功能开发网络应用

1预备知识

1.1 关于本指导手册

Python是一种非常流行的面向对象的脚本语言，具有语法简洁的特点，并且拥有非常广泛的开发者基础。它是通用目的语言，能够在各种设定中使用。它在刚开始编程的初学者也非常受欢迎，就是20世纪70年代的BASIC语言那样。

本教程的侧重点是Python语言在网络编程方面的应用。 介绍了Python的基本套接字，除此之外，还提供异步套接字的一些Python的其他类。 我还详细介绍了Python的应用层协议类，展示了如何构建Web客户端，邮件服务器和客户端等等。

我还演示了一个简单的聊天服务器来说明Python套接字在套接字应用方面的强大。

在阅读本文前，你应该对标准BSD套接字APIs有一定的理解，且有GNU/Linux环境下工作的经验。熟悉面向对象编程也是非常有益的。

1.2 依赖条件

本教程和演示中的示例依赖于Python的2.4版本。 您可以从Python官方网站下载此版本（[请参阅参考资料中的链接](https://www.ibm.com/developerworks/linux/tutorials/l-pysocks/#resources)）。 要构建Python解释器，需要GNU C编译器（gcc）和configure / make工具（它们是任何一种标准GNU / Linux发行版的一部分）。

2介绍Python

首先，让我们体验一下Python。

2.1 什么是Python?

Python是一个通用的面向对象的脚本语言，可以应用于各种各样的问题。 它是在20世纪90年代初期在阿姆斯特丹的CWI创建的，其后在Python软件基金会下继续发展到今天。

Python有着不可思议的移植性，几乎可以在所有的操作系统上发现它的身影。

Python是解释型语言，且很容易扩展。你可以通过添加模块扩展Python，Python模块可以包含函数，变量，或者通过C/C++编译的函数类型。

您还可以轻松地将Python嵌入到C或C ++程序中，使您可以使用脚本功能扩展应用程序。

Python最有用的方面之一就是其拥有大量的扩展模块。这些模块不仅提供了诸如字符串或列表处理的标准功能，还提供了用于视频和图像处理，音频处理以及网络的应用层模块。

2.2 体验Python

现在让我来告诉你Python是什么吧！

作为一种解释型语言，使用Python轻松实现想法并快速建立原型软件非常简单。 Python程序可以整体解释，或一行一行解释。

您可以通过先运行Python来测试以下Python代码片段，然后逐个输入一行。 Python被调用后，出现一个提示符（>>>），允许您输入命令。 请注意，缩进在Python中很重要，因此一行中的前面的空格不能被忽略：

例1．尝试一些Python例子

# 打开一个文件, 逐行读取,然后打印出来

for line in open('file.txt'):

print line

# 创建一个新文件，写入内容

file = open("test.txt", "w")

file.write("test line\n")

file.close()

# 创建字典，包含名字和年龄，

family = {'Megan': 13, 'Elise': 8, 'Marc': 6}

# 读取8

family['Elise']

# 移除一个键值对

del family['Elise']

# 创建一个列表和函数，函数对列表中的每个元素求平方。

# 将函数与列表中的元素建立键值对关系。

arr = [1, 2, 3, 4, 5]

def double(x): return x\*x

map(double, arr)

# 通过继承创建一个类，并实例化，且调用它的方法

class Simple:

def \_\_init\_\_(self, name):

self.name = name

def hello(self):

print self.name+" says hi."

class Simple2(Simple):

def goodbye(self):

print self.name+" says goodbye."

me = Simple2("Tim")

me.hello()

me.goodbye()

2.3 为什么使用Python?

学习和使用Python的头号原因是它的受欢迎程度。其用户群的规模以及使用Python构建的应用程序数量不断增加，这是一项有价值的投资。

您可以在几个开发领域找到Python，它用于构建系统实用程序，以及作为程序集成，Internet应用程序和快速原型的粘合语言。

与其他脚本语言相比，Python也有一些优势。 它的语法简单，概念清晰，易于学习。 使用复杂的数据结构（如列表，字典和元组）时，Python也更容易且更具描述性。 Python也可以扩展语言，反过来也可以被语言扩展。

我发现Python的语法比Perl更具可读性和可维护性，但不如Ruby。Python相对于Ruby的优势在于可用的大量库和模块。 使用这些，您可以使用少量自定义代码来构建功能丰富的程序。

Python使用缩进来标识块范围可能相当烦人，但其简单性往往弥补了这个小缺陷。

现在，我们来深入Python中的套接字编程吧。

3 Python 套接字模块

3.1 基本Python 套接字模块

Python提供了两个基本的套接字模块。 首先，Socket提供了标准的BSD套接字API。 第2个，SocketServer提供了一个以服务器为中心的类，它简化了网络服务器的开发。 它以异步方式执行此操作，您可以在其中提供插件类来执行服务器的特定于应用程序的作业。 表1列出了本节涵盖的类和模块。

表1. Python类和模块

|  |  |
| --- | --- |
| 类/模块 | 描述 |
| Socket | 低级网络接口（根据BSD API） |
| SocketServer | 提供简化网络服务器开发的类 |

让我们来看看这些模块，了解他们可以为您做些什么。

3.2 Socket

Socket模块提供了UNIX程序员最熟悉的基本网络服务（也称为BSD API）。 这个模块拥有构建套接字服务器和客户端所需的一切。

这个API和标准C API的区别在于它的面向对象。 在C中，从套接字调用中检索套接字描述符，然后将其用作BSD API函数的参数。 在Python中，套接字方法返回套接字方法可应用的套接字对象。 表2提供了一些类方法，表3显示了实例方法的一个子集。

表2. Socket模块的类方法

|  |  |
| --- | --- |
| 类方法 | 描述 |
| Socket | 底层网络接口（BSD API） |
| socket.socket(family, type) | 创建并返回一个新的套接字对象 |
| socket.getfqdn(name) | 将字符串IP地址转换为完全限定的域名 |
| socket.gethostbyname(hostname) | 根据主机名得到IP地址字符串 |
| socket.fromfd(fd, family, type) | 根据现有的文件描述符创建一个套接字对象 |

表3. Socket模块的实例化方法

|  |  |
| --- | --- |
| 实例化方法 | 描述 |
| sock.bind((adrs, port) ) | 将套接字绑定到地址和端口 |
| sock.accept() | 返回一个客户端套接字（带有对等地址信息） |
| sock.listen(backlog) | 将套接字置于监听状态，能够挂起未完成的连接请求 |
| sock.connect((adrs, port) ) | 将套接字连接到定义的主机和端口 |
| sock.recv( buflen[, flags] ) | 从套接字接收数据，最多为buflen字节 |
| sock.recvfrom( buflen[, flags] ) | 从套接字接收数据，直到buflen字节，还返回数据来自的远程主机和端口 |
| sock.send( data[, flags] ) | 通过套接字发送数据 |
| sock.sendto( data[, flags], addr ) | 通过套接字发送数据，根据指定的地址 |
| sock.close() | 关闭套接字 |
| sock.getsockopt( lvl, optname ) | 获取指定套接字选项的值 |
| sock.setsockopt(lvl, optname, val ) | 设置指定套接字选项的值 |

类方法和实例方法的不同之处在于，实例方法需要执行一个套接字实例（从套接字返回），而类方法则不需要。

3.3 SocketServer

SocketServer模块是一个有趣的模块，可以简化套接字服务器的开发。 关于它的使用的讨论远远超出了本教程的范围，但是我将演示它的基本用法，点击[这个链接地址](https://www.ibm.com/developerworks/linux/tutorials/l-pysocks/#resources)，以获得更详细讨论的链接。

考虑例2所示的简单示例。在这里，我实现了一个简单的“Hello World”服务器，它在客户端连接时发出消息。 我首先创建一个继承了SocketServer.StreamRequestHandler类的请求处理程序。 我定义了一个称为handle的方法来处理服务器的请求。 服务器所做的一切必须在这个函数的上下文中处理（最后，套接字被关闭）。 这个过程很简单，但是你可以用这个类来实现简单的HTTP服务器。 在处理方法中，我发出我的招呼，然后退出。

现在连接处理程序已准备就绪，剩下的就是创建套接字服务器。我使用  
SocketServer.TCPServer类，提供地址和端口号（服务器将绑定到）和我的请求处理程序方法。得到的是一个TCPServer对象。 调用serve\_forevermethod启动服务器并使其可用于连接。

例2. 用SocketServer模块实现一个简单的服务器

import SocketServer

class hwRequestHandler(SocketServer.StreamRequestHandler):

def handle(self):

self.wfile.write("Hello World!\n")

server = SocketServer.TCPServer ( ("", 2525), hwRequestHandler )

server. serve\_forever()

就是这样！ Python允许在这个主题上有许多变体，包括UDPServers，进程和线程服务器。

4 Python套接字编程

在使用套接字的语言中，套接字通常是相同的 - 它是两个可以相互通信的应用程序之间的通道。

4.1 依赖条件

无论你是用Python，Perl，Ruby，Scheme还是其他有用的语言编写套接字应用程序（并且我的意思是有套接字接口的语言），套接字都是相同的。 这是两个应用程序之间的通道，可以相互通信（本地在单个机器上，或者在两个不同位置的机器之间）。

与像Python这样的语言编程的套接字的区别在于可以简化套接字编程的辅助类和方法。 在本节中，我将演示Python套接字API。 您可以使用脚本执行Python解释器，或者如果您自己执行Python，则可以一次一行地与它交互。 这样，你可以看到每个方法调用的结果。

以下示例说明了与Python解释器的交互。 在这里，我使用套接字类方法gethostbyname将完全限定的域名（www.baidu.com）解析为字符串四段的IP地址（'61.135.169.125'）：

[camus]$ python  
Python 2.4 (#1, Feb 20 2005, 11:25:45)  
[GCC 3.2.2 20030222 (Red Hat Linux 3.2.2-5)] on linux2  
Type "help", "copyright", "credits" or "license" for more  
information.  
>>> import socket  
>>> socket.gethostbyname('www.baidu.com')  
'61.135.169.125'  
>>>

在导入套接字模块后，我调用gethostbyname类方法将域名解析为IP地址。

Now, I'll discuss the basic socket methods and communicating through sockets. Feel free to follow along with your Python interpreter.

现在，我将讨论基本套接字方法和通过套接字进行通信。带着你的python自由跟随吧。

4.1创建和销毁套接字

To create a new socket, you use the socket method of the socket class. This is a class method because you don't yet have a socket object from which to apply the methods. The socket method is similar to the BSD API, as demonstrated in the creation of a stream (TCP) and datagram (UDP) socket:

要创建一个新的套接字，可以使用套接字类的套接字方法。 这是一个类方法，因为您还没有从中应用方法的套接字对象。 套接字方法与BSD API相似，如创建一个流（TCP）和数据报（UDP）套接字所示：

例4.创建流和数据报套接字

streamSock = socket.socket(socket.AF\_INET, socket.SOCK\_STREAM )  
dgramSock = socket.socket ( socket.AF\_INET, socket.SOCK\_DGRAM )

在每种情况下，返回一个套接字对象。 AF\_INET符号 - 参数1表示您正在请求Internet协议（IP）套接字，特别是IPv4。 第二个参数是传输协议类型（SOCK\_STREAM用于TCP套接字，SOCK\_DGRAM用于UDP套接字）。 如果您的底层操作系统支持IPv6，则还可以指定socket.AF\_INET6来创建IPv6套接字。

要关闭连接的套接字，请使用close方法：

streamSock.close()

最后，你可以用del语句删除一个套接字：

del streamSock

该语句永久删除套接字对象。 此后尝试引用套接字会产生错误。

4.2套接字地址

套接字的端点地址是由接口地址和端口号组成的元组。 因为Python可以很容易地表示元组，所以地址和端口也就很容易表示。这说明接口地址192.168.1.1和端口80的端点：

( '192.168.1.1', 80 )

您也可以在此处使用完全限定的域名，例如：

( 'www.ibm.com', 25 )

这个例子很简单，肯定会击败C中需要的sockaddr\_in操作。下面的讨论提供了Python中地址的例子。

4.3服务器套接字

服务器套接字通常是在网络上公开服务的套接字。由于服务器和客户端套接字是以不同的方式创建的，我将独立讨论它们。

创建套接字后，您使用bind方法将地址绑定到它，使用listen方法将其置于侦听状态，最后使用accept方法接受新的客户端连接。如下所示：

例5.使用服务器套接字

sock = socket.socket ( socket.AF\_INET, socket.SOCK\_STREAM )  
sock.bind(('', 2525) )  
sock.listen( 5 )  
newsock, (remhost, remport) = sock.accept()

对于这个服务器，使用地址（''，2525），这意味着通配符用于接口地址（''），允许来自主机上任何接口的传入连接。 您还绑定到端口号2525。

请注意，accept方法不仅返回表示客户端连接（newsock）的新套接字对象，还返回地址元组（套接字对端的远程地址和端口号）。 Python的SocketServer模块可以进一步简化这个过程，如上所示。

您也可以创建数据报服务器，但它们是无连接的，因此没有关联的接受方法。 以下示例创建一个数据报服务器套接字：

例6.创建一个数据报服务器套接字

sock = socket.socket( socket.AF\_INET, socket.SOCK\_DGRAM )  
sock.bind( ('', 2525) )

即将到来的关于套接字I/O的讨论显示了I/O如何适用于流和数据报套接字。

现在，我们来探讨客户端如何创建套接字并将其连接到服务器。

4.4 客户端套接字

创建和连接客户端套接字的机制与服务器套接字的设置相似。 在创建套接字时，需要一个地址 - 不要在本地绑定套接字（就像服务器一样），而是要确定套接字应该附加到哪里。 假设主机上有一台服务器，其接口IP地址为“192.168.1.1”，端口为2525.以下代码创建一个新的套接字并将其连接到定义的服务器：

例7.创建一个流套接字并连接到服务器

sock = socket.socket( socket.AF\_INET, socket.SOCK\_STREAM)  
sock.connect(('192.168.1.1', 2525))

对于数据报套接字，过程有点不同。 回想一下，数据报套接字本质上是断开的。 一种思考的方法如下：尽管流套接字是两个端点之间的管道，但是数据报套接字是基于消息的，能够同时与多个对等体进行通信。 这是一个数据报客户端的例子。

例8.创建一个数据报套接字并连接到服务器

sock = socket.socket( socket.AF\_INET, sock.sock\_DGRAM )  
sock.connect( ('192.168.1.1', 2525) )

这里的不同之处在于，即使我使用了连接方法，客户端和服务器之间也没有真正的连接。 这里的连接是对后面的I/O的简化。通常在数据报套接字中，您必须提供您要发送的数据的目标信息。通过使用连接，我已经与客户端缓存这个信息和发送方法可以发生很像流套接字版本（没有目标地址是必要的）。 您可以再次调用connect来重新指定数据报客户端消息的目标。

4.5 流套接字I / O

在Python中通过流套接字发送或接收数据很简单。存在几种通过流套接字移动数据的方法（如发送，接收，读取和写入）。

这第一个例子演示了一个服务器和客户端的流套接字。在这个演示中，服务器回应从客户端收到的任何内容。

echo服务器如例9所示。创建一个新的流套接字时，绑定一个地址（接受来自任何接口和端口45000的连接），然后调用listen方法来启用传入连接。echo服务器然后进入客户端连接的循环。 accept方法被调用，并阻塞（即不返回），直到新客户端连接，此时新客户端套接字与远程客户端的地址信息一起返回。使用这个新的客户端套接字，我调用recv从对端获取一个字符串，然后将这个字符串写回到套接字。然后，立即关闭了这次连接。

例9.简单的Python echo服务器

import socket

srvsock = socket.socket ( socket.AF\_INET, socket.SOCK\_STREAM )  
srvsock.bind( ('', 23000) )  
srvsock.listen( 5 )  
while 1:  
 clisock, (remhost, remport) = srvsock.accept()  
 str = clisock.recv(100)  
 clisock.send( str )  
 clisock.close()

例10显示了与例9中的服务器相对应的echo客户机。创建一个新的流套接字时，使用connect方法将此套接字连接到服务器。 当连接时（当连接方法返回时），客户端用send方法发出一个简单的文本消息，然后用recv方法等待回声。 print语句用于发出所读的内容。 完成后，执行close方法关闭套接字。

例10.简单的Python echo服务器

import socket

clisock = <strong>socket.socket</strong>( socket.AF\_INET, socket.SOCK\_STREAM )

clisock.<strong>connect</strong>( ('', 23000) )

clisock.<strong>send</strong>("Hello World\n")

print clisock.<strong>recv</strong>(100)

clisock.<strong>close</strong>()

Datagram sockets I/O

Datagram sockets are disconnected by nature which means that communication requires that a destination address be provided. Similarly, when a message is received through a socket, the source of the data must be returned. The recvfrom and sendto methods support the additional address information as you can see in the datagram echo server and client implementations.

Listing 11 shows the datagram echo server. A socket is first created and then bound to an address using the bind method. An infinite loop is then entered for serving client requests. The recvfrom method receives a message from the datagram socket and returns not only the message but also the address of the source of the message. This information is then turned around with the sendto method to return the message to the source.

Listing 11. Simple Python datagram echo server

import socket

dgramSock = socket.socket</strong>( socket.AF\_INET, socket.SOCK\_DGRAM )

dgramSock.<strong>bind</strong>( ('', 23000) )

while 1:

msg, (addr, port) = dgramSock.<strong>recvfrom</strong>( 100 )

dgramSock.<strong>sendto</strong>( msg, (addr, port) )

The datagram client is even simpler. After creating a datagram socket, I use the sendto method to send a message to a specific address. (Remember: Datagrams have no connection.) After sendto finishes, I await the echo response with recv, then print it. Note that I don't use recvfrom here because I'm not interested in the peer address information.

Listing 12. Simple Python datagram echo client

import socket

dgramSock = socket.socket</strong>( socket.AF\_INET, socket.SOCK\_DGRAM )  
dgramSock.<strong>sendto</strong>( "Hello World\n", ('', 23000) )

print dgramSock.<strong>recv</strong>( 100 )

dgramSock.<strong>close</strong>()

Socket options

Sockets default to a set of standard behaviors, but it's possible to alter the behavior of a socket using options. You manipulate socket options with the setsockopt method and capture them with the getsockopt method.

Using socket options is simple in Python, as demonstrated in Listing 13. In the first example, I read the size of the socket send buffer. In the second example, I get the value of the SO\_REUSEADDR option (reuse the address within the TIME\_WAIT period) and then enable it.

Listing 13. Using socket options

sock = <strong>socket.socket</strong>( socket.AF\_INET, socket.SOCK\_STREAM )

# Get the size of the socket's send buffer

bufsize = sock.<strong>getsockopt( socket.SOL\_SOCKET, socket.SO\_SNDBUF )

# Get the state of the SO\_REUSEADDR option

state = sock.<strong>getsockopt</strong>( socket.SOL\_SOCKET, socket.SO\_REUSEADDR )

# Enable the SO\_REUSEADDR option

sock.<strong>setsockopt</strong>( socket.SOL\_SOCKET, socket.SO\_REUSEADDR, 1 )

The SO\_REUSEADDR option is most often used in socket server development. You can increase the socket send and receive buffers for greater performance, but given that you're operating here in an interpreted scripting language, it may not provide you with much benefit.

Asynchronous I/O

Python offers asynchronous I/O as part of the select module. This feature is similar to the C select mechanism but has some simplifications. I'll first introduce select and then show you how to use it in Python.

The select method allows you to multiplex events for several sockets and for several different events. For example, you can instruct select to notify you when a socket has data available, when it's possible to write data through a socket, and when an error occurs on a socket; and you can perform these actions for many sockets at the same time.

Where C works with bitmaps, Python uses lists to represent the descriptors to monitor and also the return descriptors whose constraints are satisfied. Consider the following example in which you await some input from standard input:

Listing 14. Awaiting input from stdin

rlist, wlist, elist = <strong>select.select</strong>( [sys.stdin], [], [] )

print sys.stdin.read()

The arguments passed to select are lists representing read events, write events, and error events. The select method returns three lists containing the objects whose events were satisfied (read, write, exception). In this example, upon return rlist should be [sys.stdin], indicating that data are available to read on stdin. This data are then read with the read method.

The select method also works on socket descriptors. In the following example (see Listing 15), two client sockets are created and connected to a remote peer. The select method is then used to identify which socket has data available for reading. The data are then read and emitted to stdout.

Listing 15. Demonstrating the select method with multiple sockets

import socket

import select

sock1 = <strong>socket.socket</strong>( socket.AF\_INET, socket.SOCK\_STREAM )

sock2 = <strong>socket.socket</strong>( socket.AF\_INET, socket.SOCK\_STREAM )

sock1.<strong>connect</strong>( ('192.168.1.1', 25) )

sock2.<strong>connect</strong>( ('192.168.1.1', 25) )

while 1:

# Await a read event

rlist, wlist, elist = <strong>select.select</strong>( [sock1, sock2], [], [], 5 )

# Test for timeout

if [rlist, wlist, elist] == [ [], [], [] ]:

print "Five seconds elapsed.\n"

else:

# Loop through each socket in rlist, read and print the available data

for sock in rlist:

print sock.<strong>recv</strong>( 100 )

Building a Python chat server

A simple chat server

You've explored the basic networking APIs for Python; now you can put this knowledge to use in a simple application. In this section, you'll build a simple chat server. Using Telnet, clients can connect to your Python chat server and globally communicate with one another. Messages submitted to the chat server are viewed by others (in addition to management information, such as clients joining or leaving the chat server). This model is shown graphically in Figure 1.

Figure 1. The chat server uses the select method to support an arbitrary number of clients