# **python进阶之socket详解**

获取资料和职业规划请加qq:1900529093

Socket的英文原义是“孔”或“插座”。作为BSD UNIX的进程通信机制，通常也称作"套接字"，用于描述IP地址和端口，是一个通信链的句柄，可以用来实现不同虚拟机或不同计算机之间的通信。

网络上的两个程序通过一个双向的通信连接实现数据的交换，这个连接的一端称为一个socket。

建立网络通信连接至少要一对端口号(socket)。socket本质是编程接口(API)，对TCP/IP的封装，TCP/IP也要提供可供程序员做网络开发所用的接口，这就是Socket编程接口；HTTP是轿车，提供了封装或者显示数据的具体形式;Socket是发动机，提供了网络通信的能力。

下面来说一下python的socket。

1.socket模块

获取资料和职业规划请加qq:1900529093

要使用socket.socket()函数来创建套接字。其语法如下：

socket.socket(socket\_family,socket\_type,protocol=0)

socket\_family可以是如下参数：

　　socket.AF\_INET IPv4（默认）

　　socket.AF\_INET6 IPv6

　　socket.AF\_UNIX 只能够用于单一的Unix系统进程间通信

socket\_type可以是如下参数:

　　socket.SOCK\_STREAM　　流式socket , for TCP （默认）

　　socket.SOCK\_DGRAM　　 数据报式socket , for UDP

　　socket.SOCK\_RAW 原始套接字，普通的套接字无法处理ICMP、IGMP等网络报文，而SOCK\_RAW可以；其次，SOCK\_RAW也可以处理特殊的IPv4报文；此外，利用原始套接字，可以通过IP\_HDRINCL套接字选项由用户构造IP头。

　　socket.SOCK\_RDM 是一种可靠的UDP形式，即保证交付数据报但不保证顺序。SOCK\_RAM用来提供对原始协议的低级访问，在需要执行某些特殊操作时使用，如发送ICMP报文。SOCK\_RAM通常仅限于高级用户或管理员运行的程序使用。

　　socket.SOCK\_SEQPACKET 可靠的连续数据包服务

protocol参数：

　　0　　（默认）与特定的地址家族相关的协议,如果是 0 ，则系统就会根据地址格式和套接类别,自动选择一个合适的协议

### 2.套接字对象内建方法

服务器端套接字函数

s.bind()　　　绑定地址(ip地址,端口)到套接字,参数必须是元组的格式例如：s.bind(('127.0.0.1',8009))

s.listen(5)　　开始监听，5为最大挂起的连接数

s.accept()　　被动接受客户端连接，阻塞，等待连接

客户端套接字函数

s.connect()　　连接服务器端，参数必须是元组格式例如：s.connect(('127,0.0.1',8009))

公共用途的套接字函数

s.recv(1024)　　接收TCP数据，1024为一次数据接收的大小

s.send(bytes)　　发送TCP数据，python3发送数据的格式必须为bytes格式

s.sendall()　　完整发送数据，内部循环调用send

s.close()　　关闭套接字

实例1.简单实现socket程序

获取资料和职业规划请加qq:1900529093

server端

|  |  |
| --- | --- |
| 1  2  3  4  5  6  7  8  9  10  11  12  13  14  15  16  17  18  19  20  21 | #!/usr/bin/env python  # \_\*\_ coding:utf-8 \_\*\_  **import** socket  **import** time  IP\_PORT **=** ('127.0.0.1',8009)  BUF\_SIZE **=** 1024    tcp\_server **=** socket.socket()  tcp\_server.bind(IP\_PORT)  tcp\_server.listen(5)    **while** True:  **print**("waiting for connection...")      conn,addr **=** tcp\_server.accept()  **print**("...connected from:",addr)  **while** True:          data **=** tcp\_server.recv(BUF\_SIZE)  **if** **not** data:**break**          tcp\_server.send('[%s] %s'**%**(time.ctime(),data))    tcp\_server.close() |

以上代码解释：

1~4行

第一行是Unix的启动信息行，随后导入time模块和socket模块

5~10行

IP\_PORT为全局变量声明了IP地址和端口，表示bind()函数绑定在此地址上，把缓冲区的大小设定为1K，listen()函数表示最多允许多少个连接同时进来，后来的就会被拒绝掉

11~到最后一行

在进入服务器的循环后，被动等待连接的到来。当有连接时，进入对话循环，等待客户端发送数据。如果消息为空，表示客户端已经退出，就跳出循环等待下一个连接到来。得到客户端消息后，在消息前面加一个时间戳然后返回。最后一行不会执行，因为循环不会退出所以服务端也不会执行close()。只是提醒不要忘记调用close()函数。

client端

|  |  |
| --- | --- |
| 1  2  3  4  5  6  7  8  9  10  11  12  13  14  15  16  17  18  19  20  21 | #!/usr/bin/env python  # \_\*\_ coding:utf-8 \_\*\_  **import** socket    HOST **=** '127.0.0.1'  PORT **=** 8009  BUF\_SIZE **=** 1024  ADDR **=** (HOST,PORT)    client **=** socket.socket()  client.connect(ADDR)    **while** True:      data **=** input(">>> ")  **if** **not** data:**break**      client.send(bytes(data,encoding**=**'utf-8'))      recv\_data **=** client.recv(BUF\_SIZE)  **if** **not** recv\_data:**break**      print(recv\_data.decode())    client.close() |

5~11行

HOST和PORT变量表示服务器的IP地址与端口号。由于演示是在同一台服务器所以IP地址都是127.0.0.1，如果运行在其他服务器上要做相应的修改。端口号要与服务器端完全相同否则无法通信。缓冲区大小还是1K。

客户端套接字在10行创建然后就去连接服务器端

13~21行

客户端也无限循环，客户端的循环在以下两个条件的任意一个发生后就退出：1.用户输入为空的情况或者服务器端响应的消息为空。否则客户端会把用户输入的字符串发送给服务器进行处理，然后接收显示服务器返回来的带有时间戳的字符串。

运行客户端程序与服务端程序

以下是客户端的输入与输出

|  |  |
| --- | --- |
| 1  2  3 | [root@pythontab]# python client.py  >>> hello python  [Thu Sep 15 22:29:12 2016] b'hello python' |

以下是服务端输出

|  |  |
| --- | --- |
| 1  2  3 | [root@pythontab]# python server.py  waiting **for** connection...  ...connected from: ('127.0.0.1', 55378) |

### 3.socketserver模块

socketserver是标准库中的一个高级别的模块。用于简化实现网络客户端与服务器所需要的大量样板代码。模块中已经实现了一些可以使用的类。

实例1：使用socketserver实现与上面socket()实例一样的功能

获取资料和职业规划请加qq:1900529093

服务端程序代码

|  |  |
| --- | --- |
| 1  2  3  4  5  6  7  8  9  10  11  12  13  14  15  16  17  18  19  20  21 | #!/usr/bin/env python  # \_\*\_ coding:utf-8 \_\*\_  **import** socketserver  **import** time    HOST **=** '127.0.0.1'  PORT **=** 8009  ADDR **=** (HOST,PORT)  BUF\_SIZE **=** 1024    **class** Myserver(socketserver.BaseRequestHandler):  **def** handle(self):  **while** True:              print("...connected from:",self.client\_address)              data **=** self.request.recv(BUF\_SIZE)  **if** **not** data:**break**              self.request.send(bytes("%s %s"**%**(time.ctime(),data)))    server **=** socketserver.ThreadingTCPServer(ADDR,Myserver)  print("waiting for connection...")  server.serve\_forever() |

11~17行

主要的工作在这里。从socketserver的BaseRequestHandler类中派生出一个子类，并重写handle()函数。

在有客户端发进来的消息的时候，handle()函数就会被调用。

19~21行

代码的最后一部分用给定的IP地址和端口加上自定义处理请求的类(Myserver)。然后进入等待客户端请求与处理客户端请求的无限循环中。

客户端程序代码

|  |  |
| --- | --- |
| 1  2  3  4  5  6  7  8  9  10  11  12  13  14  15  16  17  18 | **import** socket  HOST **=** '127.0.0.1'  PORT **=** 8009  ADDR **=** (HOST,PORT)  BUF\_SIZE **=** 1024    client **=** socket.socket()  client.connect(ADDR)    **while** True:      data **=** input(">>> ")  **if** **not** data:**continue**      client.send(bytes(data,encoding**=**'utf-8'))      recv\_data **=** client.recv(BUF\_SIZE)  **if** **not** recv\_data:**break**      print(recv\_data.decode())    client.close() |

执行服务端和客户端代码

下面是客户端输出

|  |  |
| --- | --- |
| 1  2  3  4  5 | [root@pythontab]# python socketclient.py  >>> hello python  Thu Sep 15 23:53:31 2016 b'hello python'  >>> hello pythontab  Thu Sep 15 23:53:49 2016 b'hello pythontab' |

下面是服务端输出

|  |  |
| --- | --- |
| 1  2  3  4  5  6 | [root@pythontab]# python socketserver.py  waiting **for** connection...  ...connected from: ('127.0.0.1', 55385)  ...connected from: ('127.0.0.1', 55385)  ...connected from: ('127.0.0.1', 55385)  ...connected from: ('127.0.0.1', 55385) |

获取资料和职业规划请加qq:1900529093