

TCP代理的python实现（包括客户端/服务器/TCP代理三部分代码）

原创

eponia

2017-12-10 21:37:54

3730

收藏

9

文章标签：

python

socket

tcp

最近一直跟随《Python黑帽子》一书学习网络编程，在实现TCP代理功能的这一节，书中是直接使用FTP客户端和服务端进行测试，为了更深刻的理解通信过程，我们可以自己编写一个socket服务端和客户端来进行测试，同时需要对书中的代码进行一些改动，以适应我们自己编写的服务端和客户端。

1. TCP代理

用socket方式实现TCP代理，代理方作为客户端和服务端之间通信的桥梁，连接建立过程大致如下：

1. 服务端开始监听服务端的本机端口，等待来自于代理方的socket连接；
2. 代理方监听本机端口，等待来自客户端的socket连接；
3. 客户端创建socket与代理方进行连接；
4. 代理方收到来自客户端的链接后，创建一个socket与服务端进行连接；
5. 服务端收到来自代理端的连接；

链接建立后，通信过程如下：

1. 客户端发送请求给代理方；
2. 代理方对请求进行处理，将处理后的请求发送给服务端；
3. 服务端接收请求后，根据请求信息发送回复给代理方；
4. 代理方将收到的回复信息进行处理，将处理后的信息发送给客户端；

2. 本文TCP代理的具体功能

服务器发送字符串，客户端接收到相同的字符串；

客户端无需发送请求信息，仅接收从服务器发送过来的信息；

当没有信息传输时，不关闭连接，而是等待服务端发送新的字符串；

3. 具体实现

3.1 代理方代码(proxy.py)，[点击获取源代码](#)，注意下面函数的顺序是为了便于理解，如果使用下面的代码进行实验，需自行调整函数定义顺序

```
#!/usr/bin/python
import socket
import sys
import threading

#主函数：
def main():
    #简单地判断命令格式，如果参数个数不是5个则提示正确的格式
    if len(sys.argv[1:]) !=5:
        print "Usage: ./proxy.py [localhost] [localport] [remotehost] [remoteport] [receivefirst]"
        exit(0)

    #为监听客户端创建的socket
    local_host=sys.argv[1]
    local_port=int(sys.argv[2])
    #为连接服务端创建的socket
    remote_host=sys.argv[3]
    remote_port=int(sys.argv[4])
    #连接建立后是否由服务端先向客户端发送信息，True代表是
    if sys.argv[5] == "True":
        receive_first=True
    else:
        receive_first=False

    #开始代理过程
    server_loop(local_host,local_port,remote_host,remote_port,receive_first)

#定义上面的server_loop函数
def server_loop(local_host,local_port,remote_host,remote_port,receive_first):
    #创建一个TCP socket以监听客户端
    server=socket.socket(socket.AF_INET,socket.SOCK_STREAM)
    try:
        server.bind((local_host,local_port))
```

```

except:
    print "Failed to listen on %s: %d"%(local_host,local_port)
    assert False,"Please try other socket"
server.listen(5)
#开始监听后，等待来自客户端的连接
while True:
    print "waiting for local client connection"
    client_socket, addr= server.accept()
    print "[%s]Received connection from %s:%d"%(addr[0],addr[1])
    #收到来自客户端的连接后，就开启一个代理线程
    proxy_thread=threading.Thread(target=proxy_handler,args=(client_socket,remote_host,remote_port,receive_first))
    proxy_thread.start()

#定义代理线程
def proxy_handler(client_socket,remote_host,remote_port,receive_first):
    #创建一个socket以连接服务端
    remote_socket=socket.socket(socket.AF_INET,socket.SOCK_STREAM)
    remote_socket.connect((remote_host,remote_port))
    #如果需要服务端先向客户端发送信息
    if receive_first:
        remote_buffer=receive_from(remote_socket)
        print "[<=]Receive %d bytes from remote host receive first"%len(remote_buffer)
        #下面两行是对收到信息的处理，得到有用的信息（做十六进制处理和自定义的处理，如果不需要，可以省略）
        hexdump(remote_buffer)
        remote_buffer=response_handler(remote_buffer)
        #如果处理后仍有有用的信息，则发送给客户端
        if len(remote_buffer):
            client_socket.send(remote_buffer)
            print "[==>]Sending %d bytes to local hosts"%len(remote_buffer)
    while True:
        #从客户端接收请求信息，由于本文中的例子不需要客户端发送请求，所以注释掉这一段
        ...

        client_buffer=receive_from(client_socket)
        print "[<=]Receive %d bytes from localhost"%len(local_buffer)
        #下面两行是对收到信息的处理，得到有用的信息（做十六进制处理和自定义的处理，如果不需要，可以省略）
        hexdump(client_buffer)
        remote_buffer=request_handler(client_buffer)
        #如果处理后仍有有用的请求信息，则发送给服务端
        if len(client_buffer):
            remote_socket.send(client_buffer)
            print "[==>]Sending %d to remote"%len(local_buffer)
        ...

        remote_buffer=receive_from(remote_socket)
        print "[<=]Receive %d bytes from remote host receive first"%len(remote_buffer)
        #下面两行是对收到信息的处理，得到有用的信息（做十六进制处理和自定义的处理，如果不需要，可以省略）
        hexdump(remote_buffer)
        remote_buffer=response_handler(remote_buffer)
        #如果处理后仍有有用的信息，则发送给客户端
        if len(remote_buffer):
            client_socket.send(remote_buffer)
            print "[==>]Sending %d bytes to local hosts"%len(remote_buffer)
        #如果从服务端和客户端都不再收到信息了，就关闭连接（由于本文的例子需要等待服务端继续发送信息，所以不关闭连接，注释掉这一段）
        ...

        if not (len(local_buffer) or len(remote_buffer)):
            client_socket.close()
            remote_socket.close()
            print "[%s]No more data. Closing connections"
            break
        ...

#上面的代码中还有一些小的方法没有定义：
#由于是TCP代理，所以接收的信息块可能比较大，所以不直接用.recv()方法，而是写一个receive_from方法来获取发送的全部信息
def receive_from(connection):
    buffer=""
    #设置一个超时时间，如果超过2s没有接收到消息则进入异常处理（必须设置，如果不设置超时时间，则会阻塞，直到接收到消息为止，在本例中也许不会！）
    connection.settimeout(2)
    try:
        #将收到的消息拼接成buffer
        while True:
            data=connection.recv(4096)
            if not data:
                break

```

```

        buffer+=data
        #如果2s都没收到消息，就返回空的buffer

    except:
        pass
    return buffer

#对接收信息进行处理函数，如果不需要特殊处理，直接返回收到的信息即可
def request_handler(buffer):
    #此处添加具体的处理代码
    return buffer
def response_handler(buffer):
    #此处添加具体的处理代码
    return buffer

#十六进制处理函数，无需深究，如果不需十六进制转换，可以省略
def hexdump(src, length=16):
    result = []
    digits = 4 if isinstance(src, unicode) else 2
    for i in xrange(0, len(src), length):
        s = src[i:i + length]
        hexa = b''.join(["%0*X" % (digits, ord(x)) for x in s])
        text = b''.join([x if 0x20 <= ord(x) < 0x7F else b'.' for x in s])
        result.append(b"%04X %-*s %s" % (i, length * (digits + 1), hexa, text))
    print(b'\n'.join(result))

#运行主函数
main()

```

3.2 服务端测试代码 (echo_socket_server.py) , 点击获取源代码

```

#!/usr/bin/python
import socket
import sys
import threading

#初始化请求队列，以便同时对多个客户端进行应答
request_queue=[]
#使用提示
def usage():
    print "./echo_socket_server.py [listen_ip] [listen_port]"

#对请求列表进行操作，对请求队列中的socket连接进行处理
def client_handler(request_queue):
    #如果有已建立的socket连接，则提示用户输入想要发送的信息内容
    if request_queue:
        msg=raw_input("message:")
        #如果用户输入的是"exit"，就关闭所有的连接，否则就向所有已连接的客户端发送输入的内容
        if msg=="exit":
            for client_socket in request_queue:
                client_socket.close()
            break
        else:
            #由于本例中只是简单地发送消息，所以直接用send方法，如果是比较耗时的操作，可以创建对每个request创建一个线程
            for client_socket in request_queue:
                client_socket.send(msg)
            print "sending %s to proxy"%msg

#每1秒检测一次，是否有新的来自客户端的连接，如果有，就放进请求列表中去，然后对请求列表进行操作
def connection(server):
    global request_queue
    while True:
        server.settimeout(1)
        try:
            client_socket,addr=server.accept()
            request_queue.append(client_socket)
            print "[%s]Connection received from %s:%d"%(addr[0],addr[1])
        except:

```

```

pass | client_handler(request_queue)

#主函数
def main():
    global request_queue
    if "help" in sys.argv:
        usage()
    else:
        server=socket.socket(socket.AF_INET,socket.SOCK_STREAM)
        server_ip=host=sys.argv[1]
        server_port = int(sys.argv[2])
        server.bind((server_host,server_port))
        server.listen(5)
    #开启一个线程，检测连接并对现有连接进行操作
    connection_thread=threading.Thread(target=connection,args=(server,))
    connection_thread.start()

main()

```

3.3 客户端测试代码（异常简单，无需注释），[点击获取源代码](#)

```

#!/usr/bin/python
import sys
import socket
def receive_from(connection):
    buffer=""
    connection.settimeout(0.01)
    try:
        while True:
            data=connection.recv(4096)
            if not data:
                break
            buffer+=data
    except:
        #print "no data received !!!"
        pass
    return buffer

def main():
    server_ip=sys.argv[1]
    server_port=int(sys.argv[2])
    client=socket.socket(socket.AF_INET,socket.SOCK_STREAM)
    client.connect((server_ip,server_port))

    while True:
        data=receive_from(client)
        if data:
            print data

main()

```

4. 测试方法

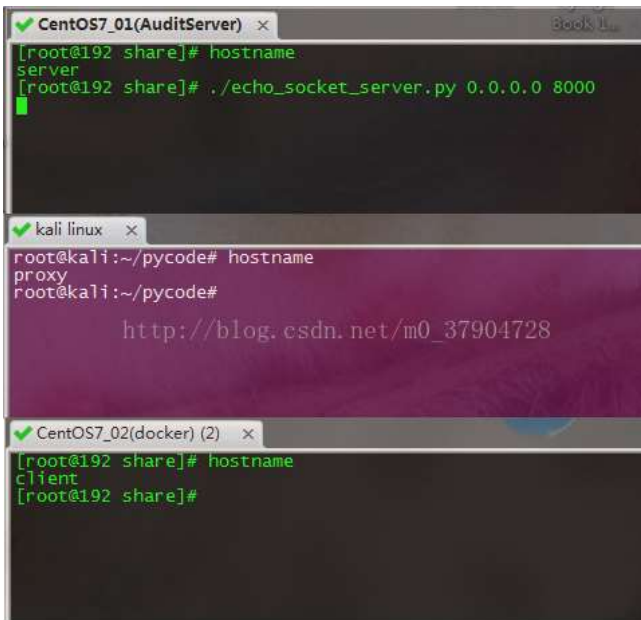
服务器主机ip地址为192.168.0.104;

客户端主机ip地址为192.168.0.102;

代理方主机ip地址为192.168.0.107;

1. 登录服务器主机，在echo_socket_server.py 文件同级目录下运行：（监听本地所有ip的8000端口，等待代理方连接）

```
./echo_socket_server.py 0.0.0.0 8000
```



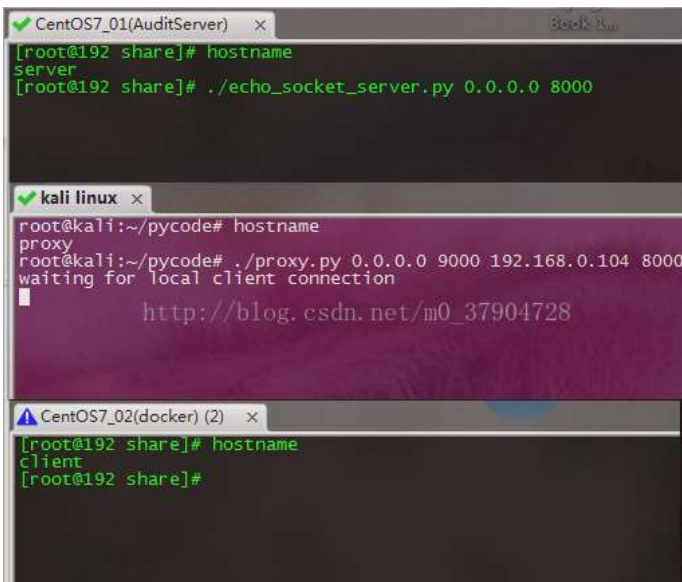
```
CentOS7_01(AuditServer) x
[root@192 share]# hostname
server
[root@192 share]# ./echo_socket_server.py 0.0.0.0 8000

kali linux x
root@kali:~/pycode# hostname
proxy
root@kali:~/pycode#
http://blog.csdn.net/m0_37904728

CentOS7_02(docker) (2) x
[root@192 share]# hostname
client
[root@192 share]#
```

2. 登录代理方主机，在proxy.py文件同级目录下运行：（监听本地所有ip的9000端口等待客户端连接，连接服务器的8000端口）

```
./proxy.py 0.0.0.0 9000 192.168.0.104 8000 True
```



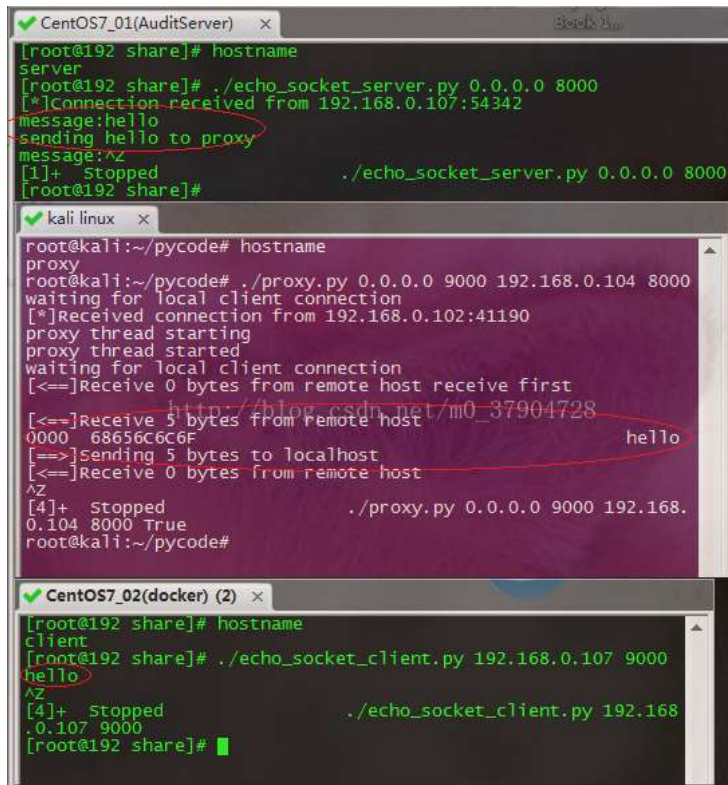
```
CentOS7_01(AuditServer) x
[root@192 share]# hostname
server
[root@192 share]# ./echo_socket_server.py 0.0.0.0 8000

kali linux x
root@kali:~/pycode# hostname
proxy
root@kali:~/pycode# ./proxy.py 0.0.0.0 9000 192.168.0.104 8000
waiting for local client connection
http://blog.csdn.net/m0_37904728

CentOS7_02(docker) (2) x
[root@192 share]# hostname
client
[root@192 share]#
```

3. 登录客户端主机，在echo_socket_client.py文件同级目录下运行：（连接代理方的9000端口）

```
./echo_socket_client.py 192.168.0.107 9000
```



```
CentOS7_01(AuditServer) x
[root@192 share]# hostname
server
[root@192 share]# ./echo_socket_server.py 0.0.0.0 8000
[*]Connection received from 192.168.0.107:54342
message:hello
sending hello to proxy
message:~Z
[1]+  Stopped                  ./echo_socket_server.py 0.0.0.0 8000
[root@192 share]#

kali linux x
root@kali:~/pycode# hostname
proxy
root@kali:~/pycode# ./proxy.py 0.0.0.0 9000 192.168.0.104 8000
waiting for local client connection
[*]Received connection from 192.168.0.102:41190
proxy thread starting
proxy thread started
waiting for local client connection
[<==]Receive 0 bytes from remote host receive first
[<==]Receive 5 bytes from remote host
0000 68656C66
[==>]Sending 5 bytes to localhost
[<==]Receive 0 bytes from remote host
~Z
[4]+  Stopped                  ./proxy.py 0.0.0.0 9000 192.168.
0.104 8000 True
root@kali:~/pycode#

CentOS7_02(docker) (2) x
[root@192 share]# hostname
client
[root@192 share]# ./echo_socket_client.py 192.168.0.107 9000
hello
~Z
[4]+  Stopped                  ./echo_socket_client.py 192.168
.0.107 9000
[root@192 share]#
```

此时可以看到客户端收到了来自服务器发送的"hello"消息，测试成功。