11.2 创建TCP服务器¶

问题¶

你想实现一个服务器，通过TCP协议和客户端通信。

解决方案¶

创建一个TCP服务器的一个简单方法是使用 `socketserver` 库。例如，下面是一个简单的应答服务器：

```
from socketserver import BaseRequestHandler, TCPServer

class EchoHandler(BaseRequestHandler):
    def handle(self):
        print('Got connection from', self.client_address)
        while True:

            msg = self.request.recv(8192)
            if not msg:
                break
            self.request.send(msg)

if __name__ == '__main__':
    serv = TCPServer(('', 20000), EchoHandler)
    serv.serve_forever()
```

在这段代码中，你定义了一个特殊的处理类，实现了一个 `handle()` 方法，用来为客户端连接服务。 `request` 属性是客户端socket，`client_address` 有客户端地址。 为了测试这个服务器，运行它并打开另外一个Python进程连接这个服务器：

```
>>> from socket import socket, AF_INET, SOCK_STREAM
>>> s = socket(AF_INET, SOCK_STREAM)
>>> s.connect(('localhost', 20000))
>>> s.send(b'Hello')
5
>>> s.recv(8192)
b'Hello'
>>>
```

很多时候，可以很容易的定义一个不同的处理器。下面是一个使用 `StreamRequestHandler` 基类将一个类文件接口放置在底层socket上的例子：

```
from socketserver import StreamRequestHandler, TCPServer

class EchoHandler(StreamRequestHandler):
    def handle(self):
        print('Got connection from', self.client_address)
        # self.rfile is a file-like object for reading
        for line in self.rfile:
            # self.wfile is a file-like object for writing
            self.wfile.write(line)

if __name__ == '__main__':
    serv = TCPServer(('', 20000), EchoHandler)
    serv.serve_forever()
```

讨论¶

`socketserver` 可以让我们很容易的创建简单的TCP服务器。 但是，你需要注意的是，默认情况下这种服务器是单线程的，一次只能为一个客户端连接服务。 如果你想处理多个客户端，可以初始化一个 `ForkingTCPServer` 或者是 `ThreadingTCPServer` 对象。例如：

```
from socketserver import ThreadingTCPServer
```

```
if __name__ == '__main__':
    serv = ThreadingTCPServer(('', 20000), EchoHandler)
    serv.serve_forever()
```

使用fork或线程服务器有个潜在问题就是它们会为每个客户端连接创建一个新的进程或线程。 由于客户端连接数是没有限制的，因此一个恶意的黑客可以同时发送大量的连接让你的服务器奔溃。

如果你担心这个问题，你可以创建一个预先分配大小的工作线程池或进程池。 你先创建一个普通的非线程服务器，然后在一个线程池中使用 `serve_forever()` 方法来启动它们。

```
if __name__ == '__main__':
    from threading import Thread
    NWORKERS = 16
    serv = TCPServer(('', 20000), EchoHandler)
    for n in range(NWORKERS):
        t = Thread(target=serv.serve_forever)
        t.daemon = True
        t.start()
    serv.serve_forever()
```

一般来讲，一个 `TCPServer` 在实例化的时候会绑定并激活相应的 `socket` 。 不过，有时候你想通过设置某些选项去调整底下的 *socket*`，可以设置参数 `bind_and_activate=False` 。如下：

```
if __name__ == '__main__':
    serv = TCPServer(('', 20000), EchoHandler, bind_and_activate=False)
    # Set up various socket options
    serv.socket.setsockopt(socket.SOL_SOCKET, socket.SO_REUSEADDR, True)
    # Bind and activate
    serv.server_bind()
    serv.server_activate()
    serv.serve_forever()
```

上面的 `socket` 选项是一个非常普遍的配置项，它允许服务器重新绑定一个之前使用过的端口号。 由于要被经常使用到，它被放置到类变量中，可以直接在 `TCPServer` 上面设置。 在实例化服务器的时候去设置它的值，如下所示：

```
if __name__ == '__main__':
    TCPServer.allow_reuse_address = True
    serv = TCPServer(('', 20000), EchoHandler)
    serv.serve_forever()
```

在上面示例中，我们演示了两种不同的处理器基类（ `BaseRequestHandler` 和 `StreamRequestHandler` ）。 `StreamRequestHandler` 更加灵活点，能通过设置其他的类变量来支持一些新的特性。比如：

```
import socket

class EchoHandler(StreamRequestHandler):
    # Optional settings (defaults shown)
    timeout = 5                     # Timeout on all socket operations
    rbufsize = -1                   # Read buffer size
    wbufsize = 0                    # Write buffer size
    disable_nagle_algorithm = False  # Sets TCP_NODELAY socket option
    def handle(self):
        print('Got connection from', self.client_address)
        try:
            for line in self.rfile:
                # self.wfile is a file-like object for writing
                self.wfile.write(line)
        except socket.timeout:
            print('Timed out!')
```

最后，还需要注意的是绝大部分Python的高层网络模块（比如HTTP、XML-RPC等）都是建立在 `socketserver` 功能之上。 也就是说，直接使用 `socket` 库来实现服务器也并不是很难。 下面是一个使用 `socket` 直接编程实现的一个服务器简单例子：

```
from socket import socket, AF_INET, SOCK_STREAM

def echo_handler(address, client_sock):
```

```
    print('Got connection from {}'.format(address))
    while True:
        msg = client_sock.recv(8192)
        if not msg:
            break
        client_sock.sendall(msg)
    client_sock.close()

def echo_server(address, backlog=5):
    sock = socket(AF_INET, SOCK_STREAM)
    sock.bind(address)
    sock.listen(backlog)
    while True:
        client_sock, client_addr = sock.accept()
        echo_handler(client_addr, client_sock)

if __name__ == '__main__':
    echo_server(('', 20000))
```