注意到 UDPServer 的开始部分与 UDPClient 类似。它也是导入套接字模块，也将整数变量 serverPort 设置为 12000，并且也创建套接字类型 SOCK_DGRAM（一种 UDP 套接字）。与 UDPClient 有很大不同的第一行代码是：

```
serverSocket.bind(('', serverPort))
```

上面行将端口号 12000 与个服务器的套接字绑定（即分配）在一起。因此在 UDPServer 中，（由应用程序开发者编写的）代码显式地为该套接字分配一个端口号。以这种方式，当任何人向位于该服务器的 IP 地址的端口 12000 发送一个分组，该分组将指向该套接字。UDPServer 然后进入一个 while 循环；该 while 循环将允许 UDPServer 无限期地接收并处理来自客户的分组。在该 while 循环中，UDPServer 等待一个分组的到达。

```
message, clientAddress = serverSocket.recvfrom(2048)
```

这行代码类似于我们在 UDPClient 中看到的。当某分组到达该服务器的套接字时，该分组的数据被放置到变量 message 中，其源地址被放置到变量 clientAddress 中。变量 clientAddress 包含了客户的 IP 地址和客户的端口号。这里，UDPServer 将利用该地址信息，因为它提供了返回地址，类似于普通邮政邮件的返回地址。使用该源地址信息，服务器此时知道了它应当将回答发向何处。

```
modifiedMessage = message.upper()
```

此行是这个简单应用程序的关键部分。它获取由客户发送的行并使用方法 upper()将其转换为大写。

```
serverSocket.sendto(modifiedMessage, clientAddress)
```

最后一行将该客户的地址（IP 地址和端口号）附到大写报文上，并将所得的分组发送到服务器的套接字中。（如前面所述，服务器地址也附在分组上，尽管这是自动而不是显式地由代码完成的。）因特网则将分组交付到该客户地址。在服务器发送该分组后，它仍维持在 while 循环中，等待（从运行在任一台主机上的任何客户发送的）另一个 UDP 分组到达。

为了测试这对程序，可在一台主机上运行 UDPClient. py，并在另一台主机上运行 UDPServer. py。保证在 UDPClient. py 中包括适当的服务器主机名或 IP 地址。接下来，在服务器主机上执行编译的服务器程序 UDPServer. py。这在服务器上创建了一个进程，等待着某个客户与之联系。然后，在客户主机上执行编译的客户器程序 UDPClient. py。这在客户上创建了一个进程。最后，在客户上使用应用程序，键入一个句子并以回车结束。

可以通过稍加修改上述客户和服务器程序来研制自己的 UDP 客户 - 服务器程序。例如，不必将所有字母转换为大写，服务器可以计算字母 s 出现的次数并返回该数字。或者能够修改客户程序，使得收到一个大写的句子后，用户能够向服务器继续发送更多的句子。

2.7.2 TCP 套接字编程

与 UDP 不同，TCP 是一个面向连接的协议。这意味着在客户和服务器能够开始互相发送数据之前，它们先要握手和创建一个 TCP 连接。TCP 连接的一端与客户套接字相联系，另一端与服务器套接字相联系。当创建该 TCP 连接时，我们将其与客户套接字地址（IP 地址和端口号）和服务器套接字地址（IP 地址和端口号）关联起来。使用创建的 TCP 连接，当一侧要向另一侧发送数据时，它只需经过其套接字将数据丢给 TCP 连接。这与