相较于其他的主要开发语言，Go 语言对基于 Socket 的开发过程做了很多优化，使开发过程更加简单方便。

13.3 基于 TCP 的 Socket 客户端/服务器系统

Go 语言在标准库的 net 包中提供了比较统一的 Socket 开发所需的数据类型和函数，下面将针对服务器端和客户端分别进行典型实现方式的介绍。

13.4 典型的 Socket 服务器的例子

典型的 Socket 服务器程序需要进行下面几个步骤来完成一个服务器应用需要做的事情：

- 在某个网络端口上创建监听网络请求的监听器；
- 通过监听器接受客户端的连接请求；
- 为成功接入的客户端连接分配单独的并发线程（早期是进程）分别处理；
- 在单独的处理线程中接收客户端发送的数据，并在做相应处理后向客户端发送反馈信息；
- 在客户端完成通信或中断连接后，清理线程和网络连接等系统资源；
- 上述处理过程中各个客户端通信的单独线程都是并发运行的，监听客户端连接请求并分配单独处理线程的主线程也是和它们并发运行的；
- 在收到特殊的指令或满足指定的条件时服务器终止运行。

下面给出一个典型的 Socket 服务端（本书中大部分 Socket 服务端和 Socket 客户端都是基于 TCP，而基于 UDP 的则称为 UDP 服务端和 UDP 客户端，这也是一般开发者中常用的称呼方式）的代码样例，要运行该代码，需要在 src 目录下新建一个 socketserver 子目录，然后在该子目录下新建一个 socketserver.go 文件来输入代码 13-1。

```
package main

import (
  "bufio"
  "fmt"
  "net"
  "strings"
  "time"
)

// connectionHandler 是处理单个连接的函数
func connectionHandler(connectionA net.Conn) {
  // 确保连接最终会被关闭
  defer connectionA.Close()

  messageCountT := 0

  // 在连接上循环接收一行一行的文本字符串并作处理
  for {
      // 从连接读取字符串，每次接收一行（以"\n"换行符为分界）
      messageT, errT := bufio.NewReader(connectionA).ReadString('\n')

      if errT != nil {
          fmt.Printf("从连接读取数据时发生错误（连接将被关闭）：%v\n", errT.Error())
          return
      }

      // 去除收到字符串的首尾空白字符（包括最后的"\n"）
      messageT = strings.TrimSpace(messageT)

      // 根据收到的字符串进行处理
      switch messageT {
```