第 5 章 网 络 编 程

Java 语言提供了丰富的网络编程类库。查看 Android API 文档会发现，Android 支持 Java 提供的所有网络编程方式，所以本章通过具体实例对 Java 网络编程进行讲解，循序渐进地让读者理解网络编程。5.2 节将从最简单的控制台程序开始学起，掌握最简单的 IO 操作。从实际教学情况看，可以发现几乎每个学过 Java 的同学都知道 System.out.println(…)，但并不是每个同学都知道到怎么实现由键盘读入。可以发现，从键盘、显示器的默认输入输出开始，一旦建立起网络连接，接下来网络两端的读写操作和本地读写是一样的，本章的教学顺序、教学方法在实际教学中取得了较好的效果。

5.1 什么是网络编程

什么是网络编程？这是我们遇到的第一个问题。也是我在本科学习阶段一直很困惑的问题。

首先简单地回答网络为什么要分层这个问题。当一个问题比较复杂的时候，常见有两种方式处理。一是横向分。例如下课后我们要办个班级小聚会。张三同学带着几个同学准备布置场地、李四同学和几个同学去买吃的等。这样的协作形式就是横向的。落实到软件领域，例如做一个简单的网站，各个模块之间没有耦合或耦合很少，大家横向分工，各自之间互不干扰。二是纵向分，例如在淘宝上网购，从下订单、商家发货、物流、买家收货、淘宝确认等，这一系列的过程是承前启后、彼此连贯的，这就是分层。这里就不再扩展了。当然，在软件工程领域还有其他的一些方法，如原型法等，这里也不扩展了。

一般计算机网络课程中是按照 5 层的形式教学的，即物理层、数据链路层、网络层、传输层、应用层。每一层都会提供一些函数供我们调用。这在现实世界是一样的，例如去物流快递公司寄包裹，那么我们要填一个单子，这就是接口，或者说是函数调用。

我们学习的是 Android 应用程序开发，因此站在应用层。试想我们站在应用层，那么看到的有两层：传输层和应用层本身。因此，本章重点讲两种编程：一是调用传输层提供的 Socket 接口进行网络编程；二是调用应用层的函数、类接口进行网络编程。在 Socket 编程中，一般有两种：TCP 和 UDP，本章重点介绍 TCP。在应用层网络编程中有很多协议，重点学习最常使用的 HTTP。

5.1.1 Socket 通信

在 TCP/IP 网络应用中，通信的两个进程间相互作用的主要模式是客户/服务器模式（Client/Server model），即客户向服务器发出服务请求，服务器接收到请求后，提供相应的服务。TCP 通信中与 Socket 通信有两个相关的类：一个是代表客户端套接字的 Socket 类，另一个是代表服务器端的套接字的 ServerSocket 类。

1．客户端

Socket 是对 TCP/IP 协议的封装，它本身并不是协议，而是一个调用接口（API），客户端通过构造一个 Socket 类对象建立与服务器的 TCP 连接。Socket 类的构造方法有如表 5-1 所示的 6 种。

表 5-1 Socket 类的构造方法

构 造 方 法	功 能 说 明
Socket()	创建套接字，不请求任何连接
Socket(InetAddress address, int port)	创建一个流套接字并将其连接到指定 IP 地址的指定端口号
Socket(InetAddress address, int port, InetAddress localAddr, int localPort)	创建一个套接字并将其连接到指定远程端口上的指定远程地址
Socket(Proxy proxy)	不管其他设置如何，都应使用的指定代理类型（如果有），根据创建一个未连接的套接字
Socket(SocketImpl impl)	创建带有用户指定的 SocketImpl 的未连接 Socket
Socket(String host, int port)	创建一个流套接字并将其连接到指定主机上的指定端口号

5.2 节讲解的实例中用到的是 Socket(String host, int port)这个构造方法建立连接，如 Socket（"localhost","8008"），这意味着向本机发送请求通过 8008 端口连接，如果客户端发出的请求被服务器拒绝，Socket 构造方法就会抛出 ConnectionException。成功创建 Socket 对象后，通过 Socket 类提供的一系列方法，与服务器进行交互，Socket 类的主要方法如表 5-2 所示。

表 5-2 Socket 类的主要方法

方 法 名	功 能 说 明
void close()	关闭 Socket 连接
InputStream getInputStream()	获取 Socket 输入流
OutputStream getOutputStream()	获取 Socket 输出流
int getPort()	获取远程主机端口号
InetAddress getLocalAddress()	获取本地主机的 Internet 地址

客户端的实例将在 5.2 节详细讲述。

2．服务器端

ServerSocket 类用来在服务器端监听所有来自指定接口的连接，并为每个新的连接创建一个 Socket 对象，客户端和服务器端就可以通过 Socket 对象进行通信了。ServerSocket 的构造方法如表 5-3 所示。

表 5-3　Server Socket 的构造方法

构 造 方 法	功 能 说 明
ServerSocket()	创建非绑定服务器套接字
ServerSocket(int port)	创建绑定到特定端口的服务器套接字
ServerSocket(int port, int backlog)	利用指定的 backlog 创建服务器套接字并将其绑定到指定的本地端口号
ServerSocket(int port, int backlog, InetAddress bindAddr)	使用指定的端口、侦听 backlog 和要绑定到的本地 IP 地址创建服务器

在 5.2 节的服务器端实例中，采用了 ServerSocket(8008)这个构造方法开启 8008 端口监听客户端连接，并且通过 ServerSocket 类的 accept()方法侦听并接收来自客户端的连接请求，返回 Socket 对象；如果没有客户端连入，则服务器端一直在 accept()方法这里处于阻塞状态。

5.1.2　HTTP 通信

在 Java 中使用 HTTP 通信的客户端需要用到 java.net 包中的 HttpURLConnection 类，每个 HttpURLConnection 实例都可用于生成单个请求，请求完成后在 HttpURLConnection 的 InputStream 或 OutputStream 上调用 close() 方法以释放与此实例关联的网络资源。表 5-4 是 HttpURLConnection 的常用方法。

表 5-4　HttpURLConnection 的常用方法

方　　法	功 能 说 明
HttpURLConnection(URL u)	构造方法
String getRequestMethod(String method)	获取请求方法
int getResponseCode()	从 HTTP 响应消息获取状态码

在 HttpURLConnection 构造方法中通过 URL 对象指明要访问的 URL 地址。URL 在 java.net 包中，表 5-5 是 URL 的常用方法。

表 5-5　URL 的常用方法

方　　法	功 能 说 明
URL(String spec)	根据 String 表示形式创建 URL 对象
String getHost()	获得此 URL 的主机名
URLConnection openConnection()	返回一个 URLConnection 对象，它表示到 URL 所引用的远程对象的连接
InputStream openStream()	打开到此 URL 的连接并返回一个用于从该连接读入的 InputStream

服务器端通过 Servlet 接收并响应客户端的请求，详细示例见 5.2.7 节。

5.2　客户/服务器模式

本节采用 7 个实例介绍从最简单的控制台输入输出到网络上通过 Socket 建立 C/S 模式的通信。C/S 模式采用由简单到复杂的思路：一个客户端和一个服务器端进行一次通

信、一个客户端和一个服务器端进行多次通信、多个客户端和一个服务器端进行通信。最后讲解客户端和服务器端通过 HTTP 协议进行通信。

5.2.1 控制台上的简单输入输出

本节实现了一个在控制台上输入并且输出结果的示例，代码在工程 Chap5.2.1 中，运行本工程，在控制台中输入 hello 字符，如图 5-1 所示。

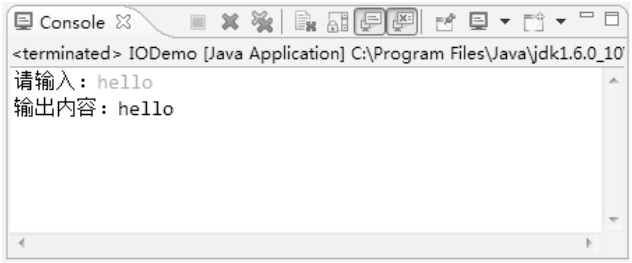

图 5-1 Chap5.2.1 执行结果

Chap5.2.1 代码如下所示：

```
1.  package io;
2.  import java.util.Scanner;
3.
4.  public class IODemo {
5.
6.      public IODemo() {
7.          System.out.print("请输入：");
8.          Scanner scanner = new Scanner(System.in);
9.          String str = scanner.next();
10.         System.out.println("输出内容："+str);
11.     }
12.
13.     public static void main(String[] args) {
14.         new IODemo();
15.     }
16.
17. }
```

在第 8 行通过 java.util 包中的 Scanner 类实现控制台的输入输出，在第 9 行通过 Scanner 类的 next()方法得到输入到控制台中的字符串，并在第 10 行通过 System.out 输出字符串到控制台。

5.2.2 控制台上的循环输入输出

本节的工程实现在控制台上的循环输入输出，在 Chap5.2.1 工程的基础上加入了一

个 while 循环，使程序可以循环输入输出。当输入 exit 字符串时，程序退出循环。运行本工程，在控制台中输入 hello 字符，效果如图 5-2 所示。

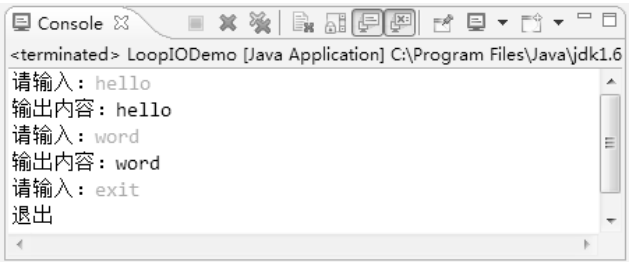

图 5-2　Chap5.2.2 执行结果

工程 Chap5.2.2 代码如下所示：

```
1.  package io;
2.  import java.util.Scanner;
3.
4.  public class LoopIODemo {
5.
6.      public LoopIODemo() {
7.          while (true) {
8.              System.out.print("请输入：");
9.              Scanner scanner = new Scanner(System.in);
10.             String str = scanner.next();
11.             if (str.equals("exit")) {
12.                 System.out.println("退出");
13.                 break;
14.             } else {
15.                 System.out.println("输出内容：" + str);
16.             }
17.         }
18.     }
19.
20.     public static void main(String[] args) {
21.
22.         new LoopIODemo();
23.     }
24.
25. }
```

本程序与 Chap5.2.1 相比，在第 7 行添加了一个 while 循环实现在控制台循环输入的功能，每次在控制台上输入的数据都会在第 11～13 行的条件语句中进行判断，如果输入的字符是 exit，则执行第 12、13 行，退出 while 循环，终止程序。

5.2.3 一个客户端和一个服务器端一次通信

从本节起开始介绍客户端与服务器端的 Socket 通信。首先是最简单的一个客户端和一个服务器端进行一次通信。工程 Chap5.2.3 中有两个包：client 包和 server 包。在 client 包中是客户端 SocketClient，在 server 包中是服务器端 SocketServer。首先运行服务器端的 SocketServer 类，没有连接客户端时的效果如图 5-3 所示。

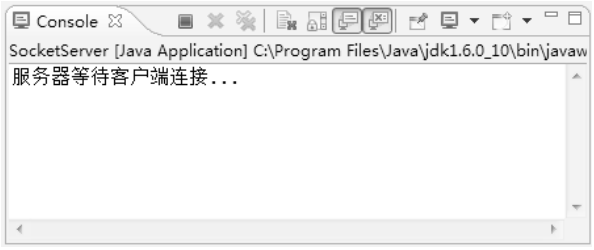

图 5-3 服务器未连客户端时

此时没有客户端连入，所以服务器端一直等待客户端的连入。当运行 SocketClient 类时，服务器端的控制台运行效果如图 5-4 所示，客户端的控制台运行效果如图 5-5 所示。

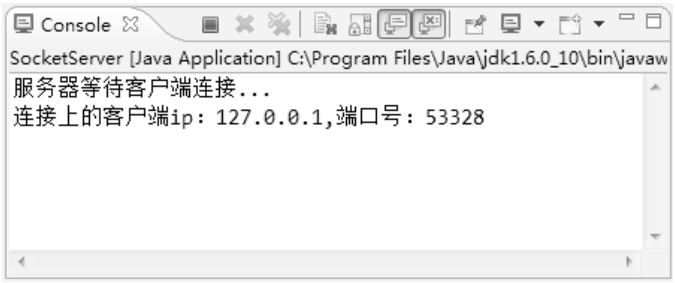

图 5-4 服务器端

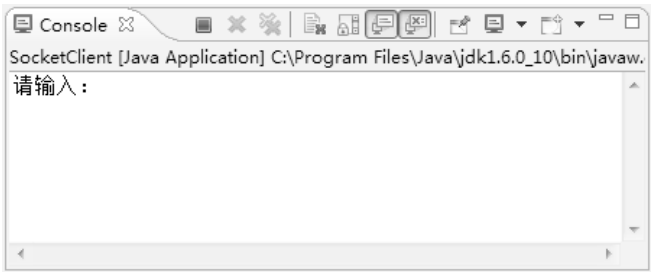

图 5-5 客户端

在客户端的控制台中输入如图 5-6 所示的数据并发送给服务器端，服务器端接收数据并返回一个字符串，然后断开与客户端的连接，如图 5-7 所示。

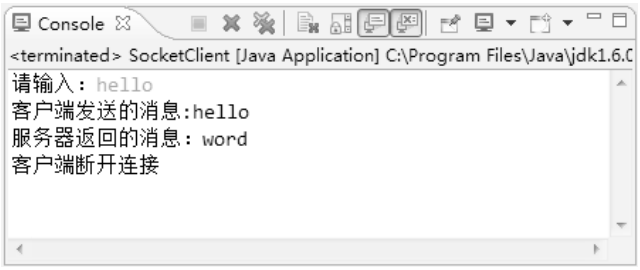

图 5-6　客户端发送消息

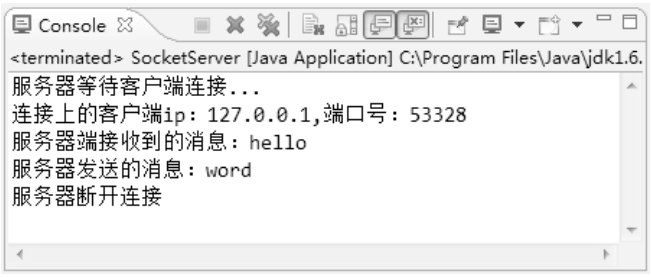

图 5-7　服务器端接收消息并返回

下面讲解 Chap5.2.3 的代码。首先是客户端 SocketClient，代码如下所示：

```
1.  package client;
2.  import java.io.IOException;
3.  import java.io.InputStream;
4.  import java.io.OutputStream;
5.  import java.net.Socket;
6.  import java.net.UnknownHostException;
7.  import java.util.Scanner;
8.
9.  public class SocketClient {
10.     public SocketClient() {
11.         Socket socket = null;
12.         OutputStream out = null;
13.         InputStream in = null;
14.         try {
15.             socket = new Socket("localhost", 8008);
16.             out = socket.getOutputStream();
17.             System.out.print("请输入：");
18.             Scanner scanner = new Scanner(System.in);
19.             String mes = scanner.next();
20.             System.out.println("客户端发送的消息:" + mes);
21.             out.write(mes.getBytes());
22.             in = socket.getInputStream();
```

```
23.             byte[] buffer = new byte[1024];
24.             int index = in.read(buffer);
25.             String receive = new String(buffer, 0, index);
26.             System.out.println("服务器返回的消息: " + receive);
27.             System.out.println("客户端断开连接");
28.             in.close();
29.             out.close();
30.             socket.close();
31.         } catch (UnknownHostException e) {
32.             e.printStackTrace();
33.         } catch (IOException e) {
34.             e.printStackTrace();
35.         }
36.     }
37.
38.     public static void main(String[] args) {
39.         new SocketClient();
40.
41.     }
42.
43. }
```

客户端 SocketClient 在包 client 中，首先在第 15 行通过 Socket 连接本地监听端口号为 8008 的服务器端，在第 16 行和第 22 行得到 Socket 的输出输入流，通过得到的输出对象 out 向服务器端发送控制台中输入的消息，然后再通过输入对象 in 接收来自服务器端的返回消息，最后在第 28～30 行关闭各个连接。

下面来看一看服务器端 SocketServer 是如何实现的，代码如下所示：

```
1.  package server;
2.  import java.io.IOException;
3.  import java.io.InputStream;
4.  import java.io.OutputStream;
5.  import java.net.ServerSocket;
6.  import java.net.Socket;
7.
8.  public class SocketServer {
9.
10.     public SocketServer() {
11.         Socket socket = null;
12.         OutputStream out = null;
13.         InputStream in = null;
14.         try {
15.             ServerSocket serverscoket = new ServerSocket(8008);
16.             System.out.println("服务器等待客户端连接...");
```

```
17.             socket = serverscoket.accept();
18.             String ip = socket.getLocalAddress().getHostAddress();
19.             int port = socket.getPort();
20.             System.out.println("连接上的客户端 ip：" + ip + ",端口号：" + port);
21.             in = socket.getInputStream();
22.             byte[] buffer = new byte[1024];
23.             int index = in.read(buffer);
24.             String receive = new String(buffer, 0, index);
25.             System.out.println("服务器端接收到的消息：" + receive);
26.             out = socket.getOutputStream();
27.             String mes = "word";
28.             out.write(mes.getBytes());
29.             System.out.println("服务器发送的消息：" + mes);
30.             System.out.println("服务器断开连接");
31.             in.close();
32.             out.close();
33.             socket.close();
34.             serverscoket.close();
35.
36.         } catch (IOException e) {
37.             e.printStackTrace();
38.         }
39.
40.     }
41.
42.     public static void main(String[] args) {
43.         new SocketServer();
44.     }
45.
46. }
```

服务器端 SocketServer 在包 server 中，首先在第 15 行通过 ServerSocket 监听 8008 端口，通过 accept()方法接收客户端的连接，如果没有客户端接入，则程序一直阻塞在第 17 行，如图 5-3 所示。当客户端接入后，开始执行下面的代码通过 Socket 获取客户端的 IP 地址和端口号，并且得到与客户端进行通信的 Socket 输入输出流，完成和客户端之间的通信。服务器端在第 24 行得到了客户端发送的消息，立即在第 28 行返回一个字符串给客户端，然后关闭各个流，完成一次客户端和服务器端的通信。

本节讲述了客户端和服务器端的一次对话，在后面将介绍客户端与服务器端的进一步交互。

5.2.4 一个客户端和一个服务器端多次通信

5.2.3 节介绍了一个客户端和一个服务器端的一次通信，本节将对 5.2.3 节的功能进

行扩充，让客户端能与服务器多次通信，直到发送退出消息结束通信。Chap5.2.4 在 Chap5.2.3 上进行了改动，修改了客户端 SocketClient，在 SocketClient 中增加了一个 while 循环，使客户端可以一直发送消息给服务器端，直到客户端发送退出消息。在服务器端的 SocketServer 中也增加了一个 while 循环，用于一直读取客户端发送的消息。下面来看一看 Chap5.2.4 的运行效果，首先分别运行服务器端和客户端，服务器端如图 5-8 所示。

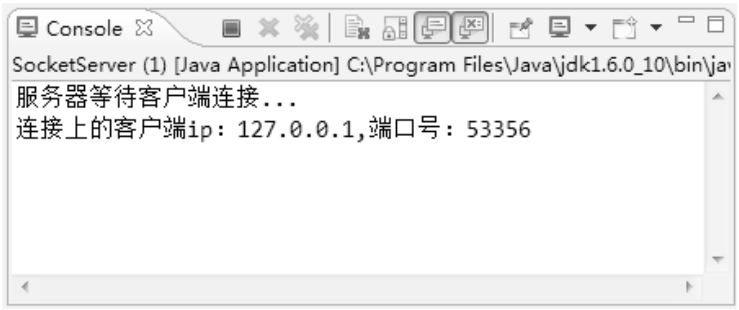

图 5-8　服务器端

然后在客户端的控制台输入 hello 消息并发送给服务器端，客户端的控制台如图 5-9 所示。

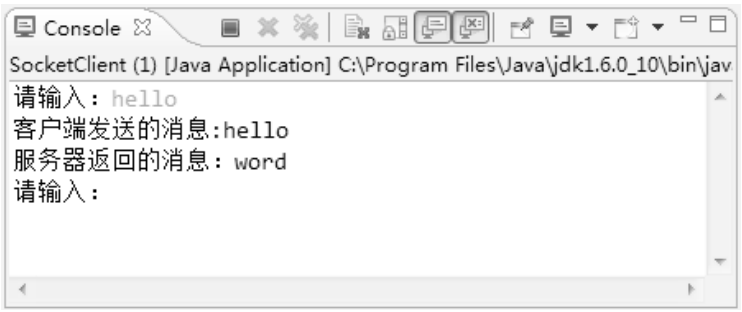

图 5-9　客户端发送一条消息

由图 5-9 可以看出，客户端还可以继续输入消息，那么再输入一条消息 hi，此时客户端的控制台如图 5-10 所示，服务器端如图 5-11 所示。

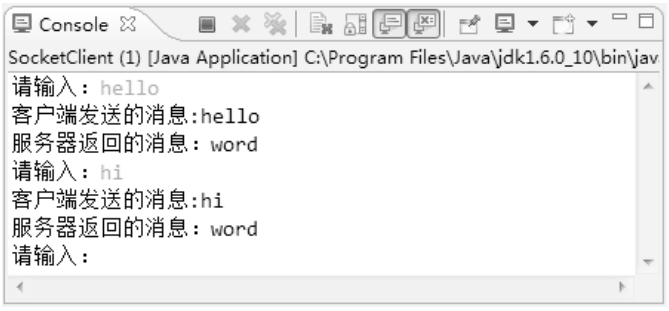

图 5-10　客户端发送第二条消息

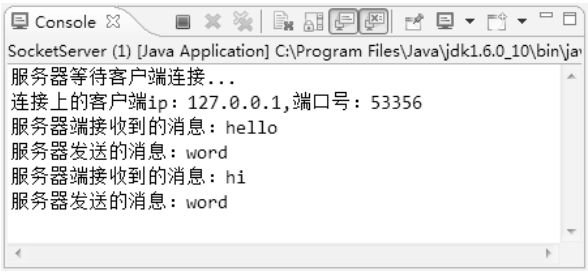

图 5-11　服务器端接收了两条消息

最好当客户端发送 exit 消息时告诉服务器断开连接，客户端如图 5-12 所示，服务器端如图 5-13 所示。

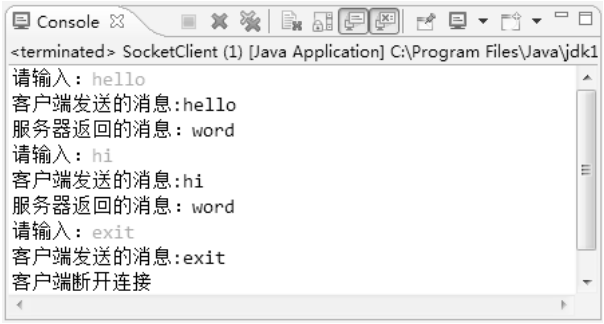

图 5-12　客户端发送退出消息

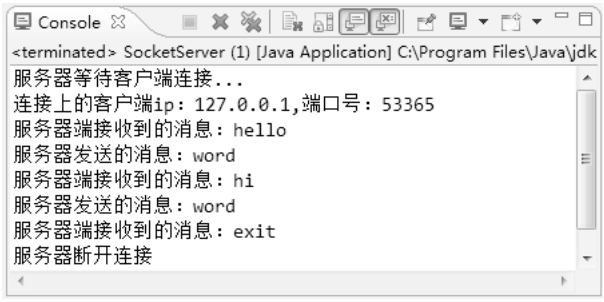

图 5-13　服务器端接收到退出消息并退出

从以上各图中可以发现客户端与服务器端进行了多次通信，下面就来讲解如何实现此功能。首先讲解客户端 SocketClient，代码如下所示：

```
1.  package client;
2.
3.  import java.io.IOException;
4.  import java.io.InputStream;
5.  import java.io.OutputStream;
6.  import java.net.Socket;
```

```
7.  import java.net.UnknownHostException;
8.  import java.util.Scanner;
9.
10. public class SocketClient {
11.     public SocketClient() {
12.         InputStream in =null;
13.         OutputStream out =null;
14.         Socket socket = null;
15.         try {
16.              socket=new Socket("localhost",8008) ;
17.              out = socket.getOutputStream();
18.             while(true)
19.             {
20.             System.out.print("请输入：");
21.             Scanner scanner = new Scanner(System.in);
22.             String mes = scanner.next();
23.             System.out.println("客户端发送的消息:"+mes);
24.             out.write(mes.getBytes());
25.             if(mes.equals("exit"))break;
26.              in = socket.getInputStream();
27.             byte[] buffer = new byte[1024];
28.             int index = in.read(buffer);
29.             String receive = new String(buffer, 0, index);
30.             System.out.println("服务器返回的消息："+receive);
31.             }
32.             System.out.println("客户端断开连接");
33.             in.close();
34.             out.close();
35.             socket.close();
36.         } catch (UnknownHostException e) {
37.             e.printStackTrace();
38.         } catch (IOException e) {
39.             e.printStackTrace();
40.         }
41.     }
42.
43.     public static void main(String[] args) {
44.         new SocketClient();
45.
46.     }
47.
48. }
```

客户端 SocketClient 在包 client 中，将本代码和 Chap5.2.3 工程中的 SocketClient 对

比，可以发现，我们将第 20～30 行（客户端与服务器进行消息发送和接收）这段代码放入了一个 while 循环中，这样客户端就可以一直发送消息和获取消息，然后在第 25 行对客户端发送的消息进行判断，如果客户端发送了 exit（退出）消息，则客户端跳出循环，结束程序。

下面来看一看服务器端 SocketServer，代码如下所示：

```
1.  package server;
2.  import java.io.IOException;
3.  import java.io.InputStream;
4.  import java.io.OutputStream;
5.  import java.net.ServerSocket;
6.  import java.net.Socket;
7.
8.  public class SocketServer {
9.
10.     public SocketServer() {
11.         InputStream in =null;
12.         OutputStream out =null;
13.         Socket socket = null;
14.         try {
15.             ServerSocket serverscoket = new ServerSocket(8008);
16.             System.out.println("服务器等待客户端连接...");
17.             socket = serverscoket.accept();
18.             String ip = socket.getLocalAddress().getHostAddress();
19.             int port = socket.getPort();
20.             System.out.println("连接上的客户端 ip：" + ip + ",端口号："
                + port);
21.             while(true)
22.             {
23.             in = socket.getInputStream();
24.             byte[] buffer = new byte[1024];
25.             int index = in.read(buffer);
26.             String receive = new String(buffer, 0, index);
27.             System.out.println("服务器端接收到的消息：" + receive);
28.             if(receive.equals("exit"))break;
29.             out = socket.getOutputStream();
30.             String mes = "word";
31.             out.write(mes.getBytes());
32.             System.out.println("服务器发送的消息："+mes);
33.             }
34.             System.out.println("服务器断开连接");
35.             in.close();
36.             out.close();
```

```
37.             socket.close();
38.             serverscoket.close();
39.
40.         } catch (IOException e) {
41.             e.printStackTrace();
42.         }
43.
44.     }
45.
46.     public static void main(String[] args) {
47.         new SocketServer();
48.     }
49.
50. }
```

服务器端 SocketServer 在 server 包中，同样，我们把本代码和 Chap5.2.3 的 SocketServer 对比，可以发现，本代码只是在 Chap5.2.3 的 SocketServer 收发消息这一部分增加了一个 while 循环，使服务器端可以不停地接收和响应客户端信息。在第 28 行对收到的消息进行判断，如果收到的是客户端的退出消息，则服务器端也退出循环并且关闭连接。

本节实现了一个客户端和一个服务器端的多次通信，那么多个客户端和一个服务器端应该如何实现呢？

5.2.5 多个客户端和一个服务器端串行通信

5.2.4 节讲述了一个客户端和一个服务器端的多次通信，本节将在 5.2.4 节的 Chap5.2.4 的基础上对服务器端代码进行修改，再加入一个 while 循环，使服务器端接收多个客户端的消息。

首先运行服务端，并且运行一个客户端 client_1，服务器端连入一个客户端，如图 5-14 所示。

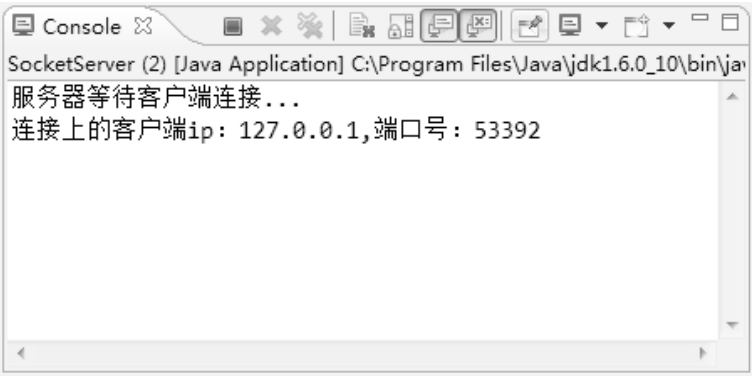

图 5-14 服务器端连入一个客户端

然后客户端 client_1 发送一条消息给服务器端，服务器端如图 5-15 所示。

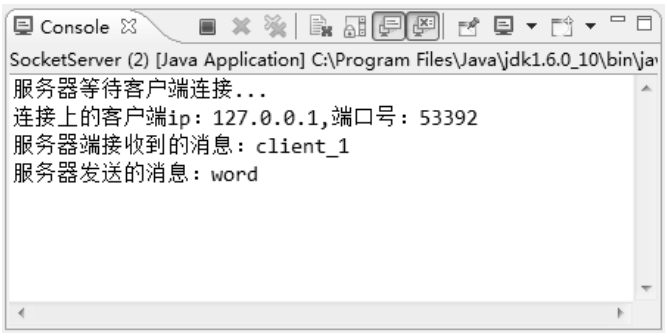

图 5-15　服务器端

此时再次运行一个客户端 client_2，此时服务器端如图 5-16 所示。

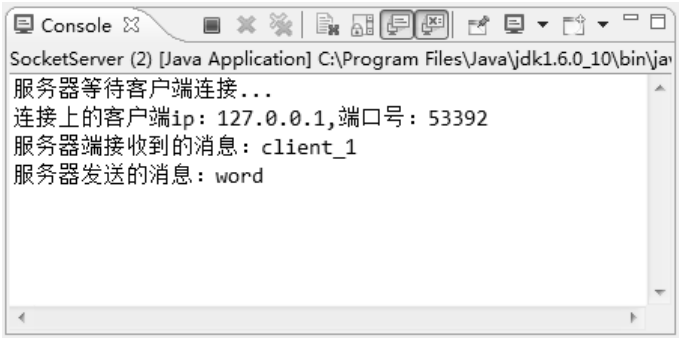

图 5-16　启动 client_2 后

可以发现客户端 client_2 没有和服务器端建立连接。为了进一步验证，用客户端 client_2 服务器端发送一条消息，客户端 client_2 如图 5-17 所示。

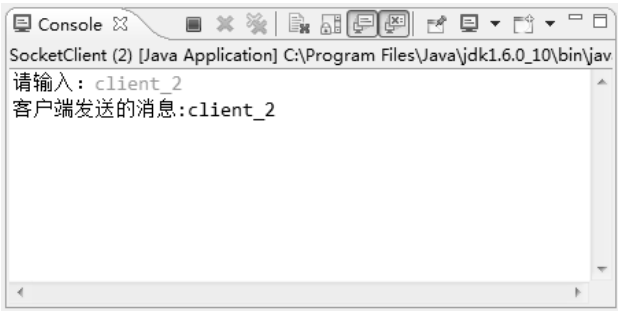

图 5-17　客户端 client_2

可以看到，客户端 client_2 发送消息后并没有收到服务器端的响应信息，证明了客户端 client_2 确实没有和服务器端建立连接。下面将断开客户端 client_1 与服务器端的连接，如图 5-18 所示。

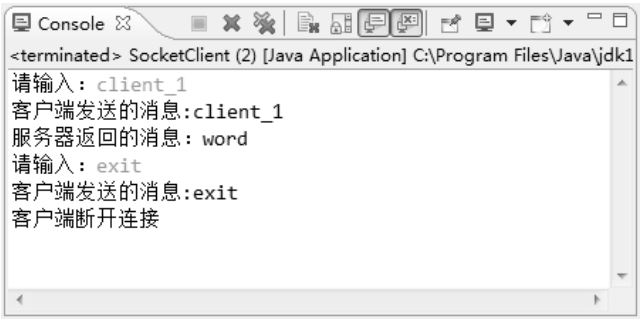

图 5-18 client_1 断开连接

此时服务器端和客户端client_2发生了变化，服务器端如图5-19所示，客户端client_2如图 5-20 所示。

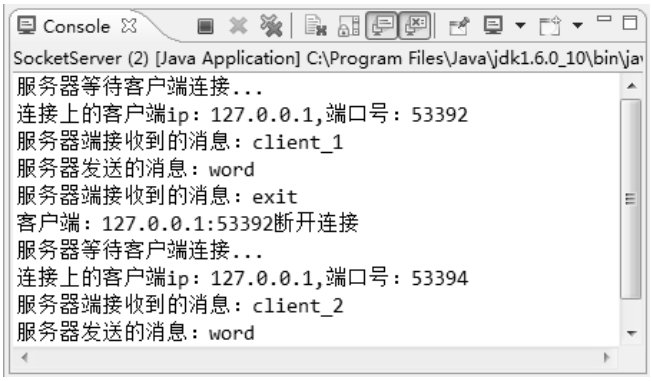

图 5-19 服务器端

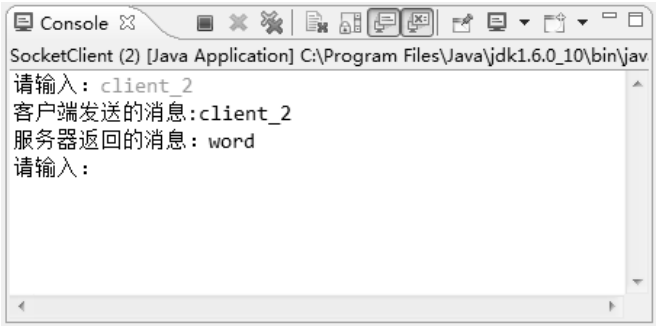

图 5-20 客户端 client_2

可以看到，当客户端 client_1 断开与服务器端的连接后，服务器端立即与客户端client_2 建立连接，收到客户端 client_2 的消息并返回响应消息。

服务器端 SocketServer 代码如下所示：

```
1.  package server;
```

```
2.  import java.io.IOException;
3.  import java.io.InputStream;
4.  import java.io.OutputStream;
5.  import java.net.ServerSocket;
6.  import java.net.Socket;
7.
8.  public class SocketServer {
9.
10.     public SocketServer() {
11.         InputStream in =null;
12.         OutputStream out =null;
13.         Socket socket = null;
14.         try {
15.             ServerSocket serverscoket = new ServerSocket(8008);
16.         while(true)
17.         {
18.             System.out.println("服务器等待客户端连接...");
19.             socket = serverscoket.accept();
20.             String ip = socket.getLocalAddress().getHostAddress();
21.             int port = socket.getPort();
22.             System.out.println("连接上的客户端 ip：" + ip + ",端口号："
                + port);
23.             while(true)
24.             {
25.             in = socket.getInputStream();
26.             byte[] buffer = new byte[1024];
27.             int index = in.read(buffer);
28.             String receive = new String(buffer, 0, index);
29.             System.out.println("服务器端接收到的消息：" + receive);
30.             if(receive.equals("exit"))break;
31.             out = socket.getOutputStream();
32.             String mes = "word";
33.             out.write(mes.getBytes());
34.             System.out.println("服务器发送的消息："+mes);
35.             }
36.             System.out.println("客户端："+ip+":"+port+"断开连接");
37.             in.close();
38.             out.close();
39.         }
40.
41.         } catch (IOException e) {
42.             e.printStackTrace();
43.         }
44.
```

```
45.     }
46.
47.     public static void main(String[] args) {
48.         new SocketServer();
49.     }
50.
51. }
```

服务器端 SocketServer 在 server 包中，在 Chap5.2.4 的基础上在第 16 行新增加了一个 while 循环，将服务器端等待客户端的连接这段代码放入 while 循环中。通过以上实例可以看出，服务器端是没有办法在与客户端 client_1 通信的时候同时等待其他客户端的连接和通信的，只有当客户端 client_1 退出后，服务器端才能与客户端 client_2 进行通信。

那么怎么才能实现服务器端一边等待其他客户端的通信，一边和已连上的客户端进行通信呢？5.2.6 节将对此做介绍。

5.2.6 多个客户端和一个服务器端并行通信

5.2.5 节想要实现多个客户端和一个服务器端多次通信，但遇到了一个问题：服务器端不能在等待其他客户端的连入的同时和已连入的客户端通信，本节通过 Java 多线程解决这个问题。

首先运行 Chap5.2.6 的服务器端 SocketServer 和客户端 client_1，如图 5-21 所示。

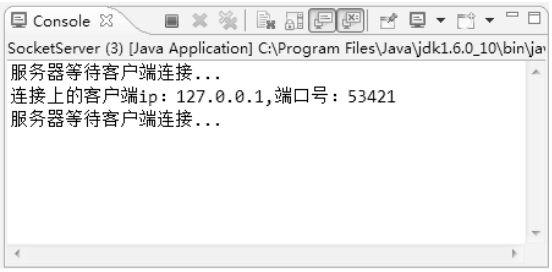

图 5-21 接入一个客户端

再次打开一个客户端 client_2，此时观察服务器端的控制台，如图 5-22 所示。

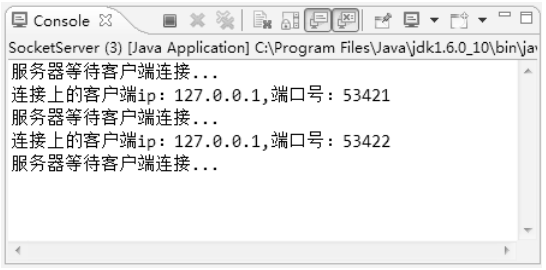

图 5-22 接入两个客户端

和 5.2.5 节的 Chap5.2.5 运行结果相比较，可以发现本工程的服务器端可以接受两个客户端同时连接，也可以与客户端 client_1 和客户端 client_2 进行通信，而不需要关闭其中一个客户端，如图 5-23 所示。

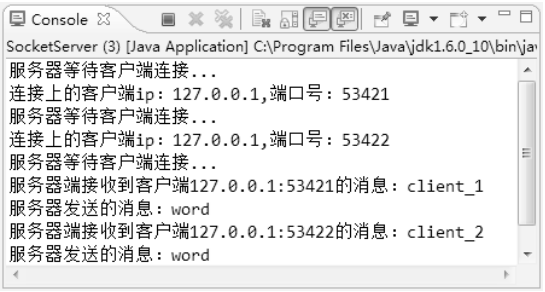

图 5-23　与不同客户端通信

下面来看一下如何实现本工程。本工程在 Chap5.2.4 的基础上对服务器端进行了修改，当每次有客户端连入时，都会在服务器端为这个客户端开一个独立的线程收发消息。服务器端 SocketServer 代码如下所示：

```
1.  package server;
2.  import java.io.IOException;
3.  import java.net.ServerSocket;
4.  import java.net.Socket;
5.
6.  public class SocketServer {
7.
8.      public SocketServer() {
9.          Socket socket = null;
10.         try {
11.             ServerSocket serverscoket = new ServerSocket(8008);
12.         while(true)
13.         {
14.             System.out.println("服务器等待客户端连接...");
15.             socket = serverscoket.accept();
16.             String ip = socket.getLocalAddress().getHostAddress();
17.             int port = socket.getPort();
18.             System.out.println("连接上的客户端 ip：" + ip + ",端口号："
                + port);
19.             new ServerThread(socket).start();
20.         }
21.
22.         } catch (IOException e) {
23.             e.printStackTrace();
24.         }
```

```
25.
26.     }
27.
28.     public static void main(String[] args) {
29.         new SocketServer();
30.     }
31. }
```

与 Chape5.2.4 的 SocketServer 对比，在第 19 行新增加了一个开启线程的功能，服务器端通过第 12 行的 while 循环不断接收客户端的连入请求，当客户端连入后，在第 19 行为连入的客户端开启一个新的线程用于与客户端进行通信，此时服务器端又回到第 15 行继续等待下一个客户端的连入请求。我们就是通过开启新线程的方式解决了工程 Chap5.2.5 服务器端不能同时接收多个客户端的连入请求问题的。线程 ServerThread 代码如下所示：

```
1.  package server;
2.  import java.io.IOException;
3.  import java.io.InputStream;
4.  import java.io.OutputStream;
5.  import java.net.Socket;
6.
7.  public class ServerThread extends Thread {
8.      private Socket socket;
9.
10.     public ServerThread(Socket socket) {
11.         this.socket = socket;
12.     }
13.
14.     public void run() {
15.         InputStream in = null;
16.         OutputStream out = null;
17.         String ip = socket.getLocalAddress().getHostAddress();
18.         int port = socket.getPort();
19.         try {
20.             while (true) {
21.                 in = socket.getInputStream();
22.                 byte[] buffer = new byte[1024];
23.                 int index = in.read(buffer);
24.                 String receive = new String(buffer, 0, index);
25.                 System.out.println("服务器端接收到客户端"+ip+":
                    "+port+"的消息：" + receive);
26.                 if (receive.equals("exit"))
27.                     break;
28.                 out = socket.getOutputStream();
```

```
29.                 String mes = "word";
30.                 out.write(mes.getBytes());
31.                 System.out.println("服务器发送的消息：" + mes);
32.             }
33.             System.out.println("客户端："+ip+":"+port+"断开连接");
34.             in.close();
35.             out.close();
36.             socket.close();
37.         } catch (IOException e) {
38.             e.printStackTrace();
39.         }
40.
41.     }
42. }
```

ServerThread 在 server 包中，ServerThread 是一个线程类，因为它继承了 Thread（注意：实现了 Runnable 接口的类也是线程类），并且覆写了 run 方法，在 run 方法中实现了与客户端之间的通信，实现通信的方法和 Chap5.2.4 是一样的，此处不再赘述。

5.2.7 客户端与服务器端 HTTP 通信

前面几节介绍了客户端与服务器端的 TCP 通信方式，通过 Socket 连接，本节讲述 HTTP 通信，Chap5.2.7 使用 tomcat 作为服务器。下面运行 Chap5.2.7 的服务器端 HttpServer 和客户端 HttpClient，并且在客户端输入 hello 字符串，客户端如图 5-24 所示，服务器端如图 5-25 所示。

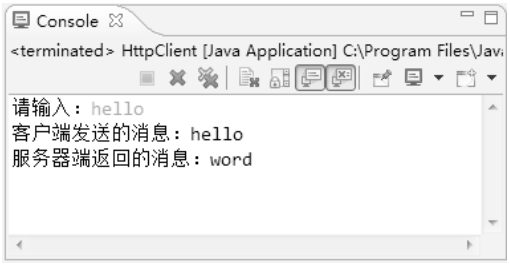

图 5-24 HTTP 客户端

图 5-25 HTTP 服务器端

HTTP 和 TCP 的区别是：HTTP 在每次请求结束后都会主动释放连接，因此 HTTP 连接是一种“短连接”，要保持客户端程序的在线状态，需要不断地向服务器发起连接请求；而 TCP 连接是一种“长连接”，连接并不会主动关闭，后续的读写操作会继续使用这个连接。下面讲解 Chap5.2.7 中的 HttpClient 和 HttpServer 的实现过程，首先是 HttpClient，代码如下所示：

```
1.  package client;
2.  import java.io.IOException;
3.  import java.io.InputStream;
4.  import java.io.OutputStream;
5.  import java.net.HttpURLConnection;
6.  import java.net.URL;
7.  import java.util.Scanner;
8.
9.
10.
11. public class HttpClient{
12.
13.     public HttpClient() {
14.         System.out.print("请输入：");
15.         Scanner scanner = new Scanner(System.in);
16.         String mes = scanner.next();
17.         String urlStr = "http://localhost/Chap5.2.7/HttpServer";
18.         URL url;
19.         try {
20.             url = new URL(urlStr);
21.             HttpURLConnection connection = (HttpURLConnection) url
22.                     .openConnection();
23.             connection.setDoOutput(true);
24.             connection.setRequestMethod("POST");
25.             OutputStream out = connection.getOutputStream();
26.             System.out.println("客户端发送的消息："+mes);
27.             out.write(mes.getBytes());
28.             InputStream in = connection.getInputStream();
29.             byte[] buffer = new byte[1024];
30.             int index = in.read(buffer);
31.             String receive = new String(buffer, 0, index);
32.             System.out.println("服务器端返回的消息："+receive);
33.             in.close();
34.             out.close();
35.         } catch (IOException e) {
36.             e.printStackTrace();
37.         }
```

```
38.
39.     }
40.
41.     public static void main(String[] args) {
42.                 new HttpClient();
43.
44.                 }
45.
46. }
```

HttpClient 在 client 包中，第 17 行定义了 URL 地址，第 20 行定义了 URL 对象，第 21 行得到 HttpURLConnection 对象，第 23 行将 DoOutput 标志设置为 true，指示应用程序要将数据写入 URL 连接，在第 24 行设定请求的方式是 POST 请求，第 25 行和第 28 行得到输出流和输入流，然后与服务器端交互。

接下来是 HttpServer，代码如下所示：

```
1.  package server;
2.
3.  import java.io.IOException;
4.  import javax.servlet.ServletInputStream;
5.  import javax.servlet.ServletOutputStream;
6.  import javax.servlet.http.HttpServlet;
7.  import javax.servlet.http.HttpServletRequest;
8.  import javax.servlet.http.HttpServletResponse;
9.
10.
11. public class HttpServer  extends HttpServlet{
12.
13.     public void doPost(HttpServletRequest req, HttpServletResponse resp)
14.     {
15.           try {
16.               ServletInputStream  in = req.getInputStream();
17.               ServletOutputStream out = resp.getOutputStream();
18.                  int len = req.getContentLength();
19.                  byte[] buffer = new byte[len];
20.                  int index = in.read(buffer);
21.                  String receive = new String(buffer, 0, index);
22.               System.out.println("服务器端接收到的消息："+receive);
23.               String mes="word";
24.               out.write(mes.getBytes());
25.               System.out.println("服务器端发送的消息："+mes);
26.                  in.close();
27.                  out.close();
28.         } catch (IOException e) {
```

```
29.             e.printStackTrace();
30.         }
31.
32.     }
33. }
```

HttpServer 在 server 包中，HttpServer 继承了 HttpServlet，并且覆写了 doPost()方法，第 16 行和第 17 行得到了输入输出流，和客户端进行交互。客户端发送 post 请求，tomcat 将 HttpServer 放入 tomcat 中运行，并且根据请求的方式执行请求的 post 方法，然后断开连接。

5.3 通 信 协 议

5.3.1 什么是协议，为什么需要协议

协议就是一组规则。在玩游戏的时候，有游戏的规则，我们需要遵守这个规则才能进行下去。例如打麻将，首先规定了麻将的规则，是成都麻将还是重庆麻将，四个人使用相同的规则我们才能玩，如果两个人打成都麻将，两个人打重庆麻将，那么这场牌就没有办法打下去。再如插座与插头，两孔插座不能插入三头的插头，插座与插头需要接口对应才能完全吻合，这是插座与插头的规则。以上这些例子说的就是生活中潜在的协议，那么计算机之间的通信也是有协议的。落实到计算机网络，通信双方必须遵循相同的协议才能互联互通，否则无法通信。例如，我们用 QQ 客户端不能登录 MSN 的服务器。

在 5.2 节所讲的客户端和服务器端的通信也是遵守了协议的，例如，如果客户端发送 hello，则服务器端返回 word；如果客户端发送 good，则服务器端返回 thanks，这一组规则就是协议，而这些协议是我们自定义的协议。

5.3.2 如何实现协议

1．协议定义

协议有两种形式：一种是基于文本的，另一种是基于二进制的。为了方便理解，本书采用的是基于文本的协议。协议既可以是自定义的，也可以是参照格式的文本，例如 Http RFC，读者可以在 Google 上搜索 Http RFC，出现的第一条记录就是 Http 协议。5.5 节通过自定义协议使客户端与服务器端可以通信，后面将会看到。协议的定义又分请求和响应两部分，例如 5.2 节中客户端发送 hello，服务器端返回 word，就是一组请求和响应。

2．协议处理

上面讲了协议的定义。协议是由请求和响应组成的，那么如何对协议进行处理呢？首先发送方需要将消息通过指定的协议打包发送给接收方，接收方收到消息根据协议对消息进行解析，对消息的打包和解析就是对协议的处理。

5.4 Handler 机制

Handler 机制是 Android 中的一种消息异步处理机制。当应用程序启动时，Android 首先会开启一个主线程（也就是 UI 线程），主线程为管理界面中的 UI 控件，如果需要一个耗时的操作，例如通过网络连接读取数据库，如果把耗时操作放在主线程中，界面会出现假死现象。如果 5s 还没有完成的话，会收到 Android 系统的一个错误提示“强制关闭”。为了解决这个问题，需要把耗时操作放在一个子线程中，由于子线程涉及 UI 更新，所以 Android 主线程是线程不安全的，也就是说，更新 UI 只能在主线程中更新，在子线程中操作是危险的。这时就使用 Handler 对 UI 进行更新，图 5-26 为子线程通过 Handler 更新 UI（主线程）的示意图。

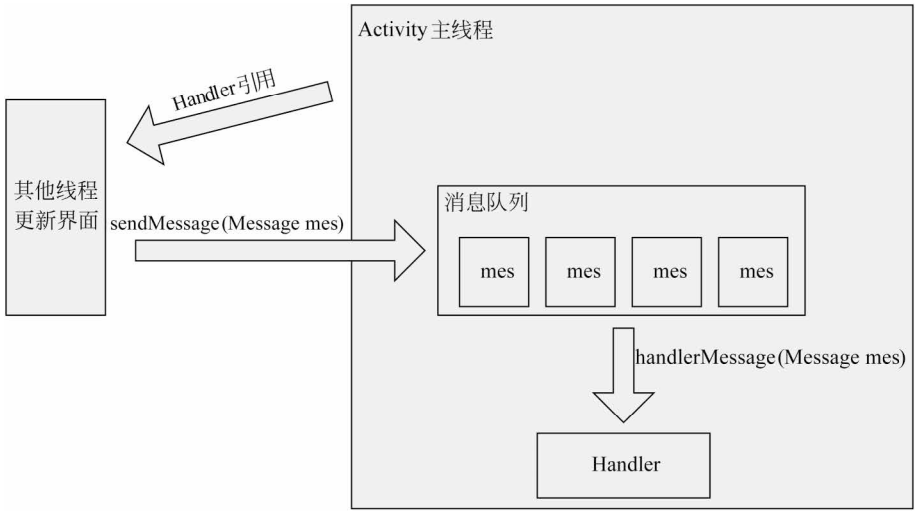

图 5-26　子线程与主线程

从图 5-26 可以看到，首先主线程将 Handler 对象的引用传递给子线程，子线程进行一些操作后要对主线程界面进行更新。此时在子线程中调用 Handler 的 sendMessage 方法，将封装到 Message 对象中数据发送到主线程的消息队列中，主线程的 Handler 从消息队列中取出消息，通过 handlerMessage 方法对取出的消息进行处理，Handler 类还有其他的一些方法，读者可以查阅 Android API 文档。Android 中对界面的更新和 Java 中的 Swing 机制有相似之处，读者可以参考《Java 核心技术卷一》的第 14 章。

5.5 联网的图书管理系统

本章学习了 Android 的 TCP 和 HTTP 编程，对 Handler 机制也做了介绍，下面通过两个实例——使用 TCP Socket 的图书管理系统和使用 HTTP 的图书管理系统讲述本章所

学内容的应用。

5.5.1 定义协议

Android 客户端需要与服务器端进行通信，首先需要自定义客户端和服务器端的通信协议，协议如下所示：

插入图书

客户端：

operate:insert
content:图书

服务器端：

operate:insert
content:图书列表
result: 插入成功/插入失败，已有此图书

删除图书

客户端：

operate:delete
content:图书名称

服务器端：

operate:delete
content:图书列表
result: 删除成功/删除失败，没有此图书

修改图书

客户端：

operate:set
content:图书

服务器端：

operate:set
content:图书列表
result: 修改成功/修改失败，没有此图书

查询图书

客户端：

operate:select
content:

服务器端：

operate:select
content:图书列表
result: 查询成功/查询失败

退出

客户端：

operate:exit

content

我们自定义了增、删、改、查 4 种协议消息。每次客户端发送增加图书的请求时，就会将 operate:insert 和 content:新增加的图书打包成一个大的字符串发送到服务器端，服务器端接收到这个消息时，对此消息进行解析并响应，服务器端在响应客户端时，首先打包消息如：

operate:insert

content:图书列表

result:插入成功/插入失败，已有此图书

客户端和服务器端就通过这样的协议进行通信。

5.5.2 使用 TCP Socket 的图书管理系统

Chapter05_tcp 工程是在 Chapter03 工程上改进的，增加了 TCP 网络连接的功能，所以本节主要讲解 TCP 网络连接和更新 UI 界面。Chapter05_tcp 工程有 5 个包，分别是 control.cqupt、model.cqupt、net.cqupt、ui.cqupt、util.cqupt。与 Chapter03 相比，增加了 net.cqupt 包、util.cqupt 包并在 model.cqupt 包中增加了 Response 类，net.cqupt 包的功能是建立 TCP 网络连接，util.cqupt 包的功能是对发送和接收的数据进行打包和解析， Response 类的功能是存储服务器发送的所有信息，Chapter05_tcp 包结构如图 5-27 所示。

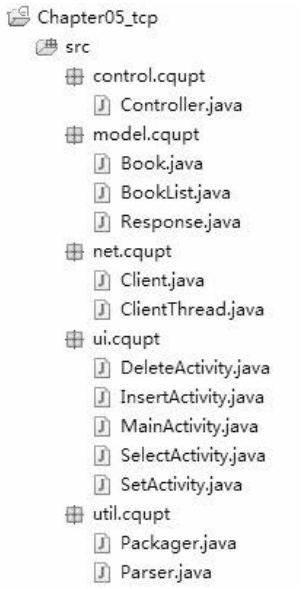

图 5-27 包结构

下面来看看 Chapter05_tcp 各个类之间的关系，如图 5-28 所示。

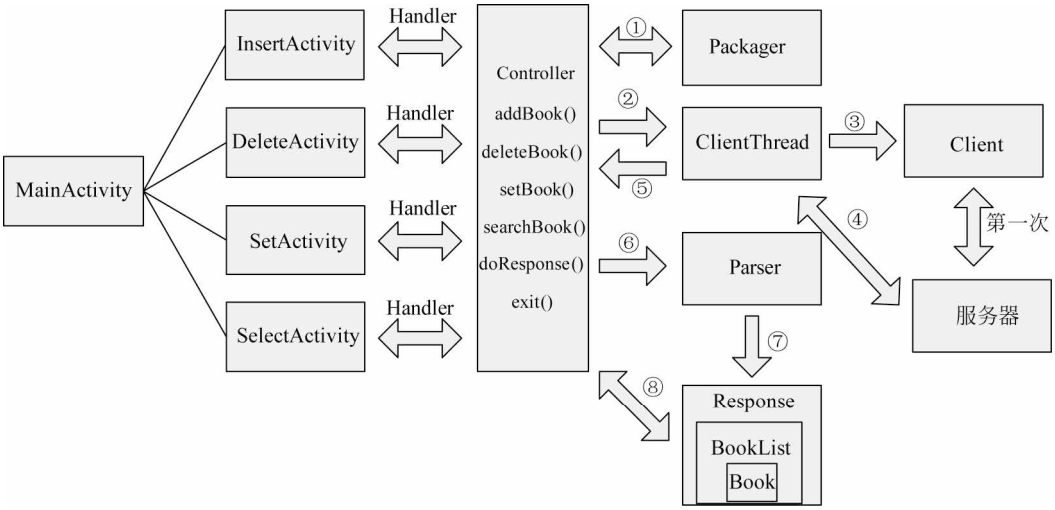

图 5-28　各个类之间的关系

下面开始按照图 5-28 标记的步骤讲解各个类。我们主要针对增加图书的流程做详细讲解，删除、修改和查询图书的实现和增加图书基本相同。Controller 的代码如下所示：

```
1.  package control.cqupt;
2.  import java.io.IOException;
3.  import java.io.OutputStream;
4.  import android.os.Bundle;
5.  import android.os.Handler;
6.  import android.os.Message;
7.  import net.cqupt.Client;
8.  import net.cqupt.ClientThread;
9.  import util.cqupt.Packager;
10. import util.cqupt.Parser;
11. import model.cqupt.Response;
12.
13. public class Controller {
14.     private Handler handler;
15.
16.     public Controller() {
17.
18.     }
19.
20.     public Controller(Handler handler) {
21.         this.handler = handler;
22.     }
23.
24.     public void addBook(String id,String name,String price) {
25.         Packager packager = new Packager();
```

```
26.         String message = packager.addPackage(id,name,price);
27.         new ClientThread(this, message).start();
28.
29.     }
30.
31.     public void searchBook() {
32.         Packager packager = new Packager();
33.         String message = packager.searchPackage();
34.         new ClientThread(this, message).start();
35.
36.     }
37.
38.     public void deleteBook(String name) {
39.         Packager packager = new Packager();
40.         String message=packager.deletePackage(name);
41.         new ClientThread(this, message).start();
42.     }
43.
44.     public void setBook(String id,String name,String price) {
45.         Packager packager = new Packager();
46.         String message = packager.setPackage(id,name,price);
47.         new ClientThread(this, message).start();
48.
49.     }
50.
51.     public void doResponse(String message) {
52.         Parser parser=new Parser();
53.         Response response=parser.parserResponse(message);
54.         String operate = response.getOperate();
55.         String result = response.getResult();
56.         Message msg = new Message();
57.         Bundle bundle = new Bundle();
58.         if (operate.equals("insert")) {
59.             bundle.putString("result", result);
60.             msg.setData(bundle);
61.             handler.sendMessage(msg);
62.         } else if (operate.equals("delete")) {
63.             bundle.putString("result", result);
64.             msg.setData(bundle);
65.             handler.sendMessage(msg);
66.         }else if (operate.equals("set")) {
67.             bundle.putString("result", result);
68.             msg.setData(bundle);
69.             handler.sendMessage(msg);
```

```
70.         }else if (operate.equals("select")) {
71.             bundle.putString("result", result);
72.             msg.setData(bundle);
73.             handler.sendMessage(msg);
74.         }
75.     }
76.
77.     public void exit() {
78.         Packager packager = new Packager();
79.         String message = packager.exitPackage();
80.         OutputStream out = null;
81.         try {
82.             out = Client.getSocket().getOutputStream();
83.             out.write(message.getBytes("GBK"));
84.             out.flush();
85.             out.close();
86.         } catch (IOException e) {
87.             e.printStackTrace();
88.         } finally {
89.             Client.close();
90.         }
91.
92.     }
93. }
```

Controller 在第 14 行声明了一个私有的成员变量 handler 对象，第 20～22 行通过构造数传递的参数为 handler 对象赋值，传递的 handler 对象是在 UI 界面定义的。关于增加图书功能，InsertActivity 将 handler 对象传递给 Controller 构造函数，并且调用 addBook 成员方法，Controller 类中第 24～29 行是 addBook 方法的实现，第 25、26 行打包数据并返回打包后的字符串。在第 27 行创建一个发送线程 ClientThread 并且启动线程，把打包后的数据和 Controller 类对象作为参数传递到线程中。传递 Controller 类对象的 this 指针是因为在 ClientThread 接收到服务器的返回值需要调用 Controller 类的 doResponse 方法，这些步骤都会在后面讲到。deleteBook 方法、setBook 方法、searchBook 方法用了同样的实现功能，不同的是打包方式。第 77～92 行是退出程序时执行的方法 exit，在 exit 方法中直接打包和发送信息给服务器，没有开新的线程，第 78、79 行打包数据并返回打包好的字符串，第 82～85 行得到 Socket 网络连接，并发送信息，第 84 行的 flush 方法是刷新缓存，在第 85 行关闭输出流，最后在第 89 行调用 Client 中的 close 方法关闭 Socket 连接。

下面讲解图 5-28 的第①步 util.cqupt 中的 Packager 类，Packager 类的作用是打包发送的数据。打包数据是按照自己规定的协议来打包的。Packager 的代码如下所示：

```
1.  package util.cqupt;
```

```
2.
3.  public class Packager {
4.      public String addPackage(String id, String name, String price) {
5.          StringBuffer mes = new StringBuffer("");
6.          mes.append("operate:insert" + "\n");
7.          String content = BookPackage(id, name, price);
8.          mes.append("content:" + content + "\n");
9.          return mes.toString();
10.     }
11.
12.     public String searchPackage() {
13.         StringBuffer mes = new StringBuffer("");
14.         mes.append("operate:select" + "\n");
15.         mes.append("content:" + "\n");
16.         return mes.toString();
17.
18.     }
19.
20.     public String deletePackage(String name) {
21.         StringBuffer mes = new StringBuffer("");
22.         mes.append("operate:delete" + "\n");
23.         mes.append("content:" + name + "\n");
24.         return mes.toString();
25.     }
26.
27.     public String setPackage(String id, String name, String price) {
28.         StringBuffer mes = new StringBuffer("");
29.         mes.append("operate:set" + "\n");
30.         String content = BookPackage(id, name, price);
31.         mes.append("content:" + content + "\n");
32.         return mes.toString();
33.     }
34.
35.     public String exitPackage() {
36.         StringBuffer mes = new StringBuffer("");
37.         mes.append("operate:exit" + "\n");
38.         mes.append("content:" + "\n");
39.         return mes.toString();
40.     }
41.
42.     public String BookPackage(String id, String name, String price) {
43.         StringBuffer str = new StringBuffer("");
44.         str.append(id + "#" + name + "#" + price);
45.         return str.toString();
```

```
46.     }
47.
48. }
```

Packager 类是对数据打包的工具类，实现了增、删、改、查、退出程序和图书信息打包的功能。这里主要讲解增加图书的打包方法。在第 4～10 行的 addPackage 方法中，在第 5 行新建一个 StringBuffer 对象，第 6 行在 StringBuffer 对象中按照自定义的协议加入操作的方法，在第 7 行对整本通过 BookPackage 打包图书并返回打包后的字符串，BookPackage 方法是在第 42～46 行实现的，对图书的打包也是根据协议完成的。在 addPackage 方法的第 9 行返回打包后的数据。deletePackage 方法、setPackage 方法、searchPackage 方法和 exitPackage 方法的实现与 addPackage 方法的实现相同，不同的是打包的内容。读者可自行阅读这些方法的实现，此处不再赘述。

图 5-28 的第②步每次向服务器发送数据都新开一个 ClientThread 线程，net.cqupt 包中的 ClientThread 的代码如下：

```
1.  package net.cqupt;
2.
3.  import java.io.IOException;
4.  import java.io.InputStream;
5.  import java.io.OutputStream;
6.  import java.net.Socket;
7.
8.  import control.cqupt.Controller;
9.
10. public class ClientThread extends Thread {
11.     private InputStream in = null;
12.     private OutputStream out = null;
13.     private static final int SIZE = 1024;
14.     private String mes;
15.     private Controller controller;
16.
17.     public ClientThread(Controller controller, String mes) {
18.         Socket socket = Client.getSocket();
19.         this.controller = controller;
20.         this.mes = mes;
21.         try {
22.             out = socket.getOutputStream();
23.             in = socket.getInputStream();
24.         } catch (IOException e) {
25.             e.printStackTrace();
26.         }
27.     }
28.
```

```
29.     public void run() {
30.
31.         try {
32.             send();
33.             byte[] buffer = new byte[SIZE];
34.             int index = in.read(buffer);
35.             String message = new String(buffer, 0, index, "GBK");
36.             controller.doResponse(message);
37.         } catch (IOException e) {
38.             e.printStackTrace();
39.         }
40.
41.     }
42.
43.     public void send() {
44.         try {
45.             out.write(mes.getBytes("GBK"));
46.             out.flush();
47.         } catch (IOException e) {
48.             e.printStackTrace();
49.         }
50.     }
51. }
```

ClientThread 继承了 Thread 类，并且在 run 方法中实现了发送和接收数据的功能。ClientThread 类在第 11、12 行声明了输入输出流的，在第 13 行定义了 Buffer 数组的大小，第 14 行声明了一个私有成员变量 mes 存放 Controller 类传递过来的打包后的字符串。在第 17～27 行的构造函数中，首先通过 Client 获得 socket 连接，这是图 5-28 的第③步，Client 的代码如下：

```
1.  package net.cqupt;
2.
3.  import java.io.IOException;
4.  import java.net.Socket;
5.  import java.net.UnknownHostException;
6.
7.  public class Client {
8.      private static Socket socket;
9.
10.     private Client() {
11.         try {
12.             socket = new Socket("192.168.1.101", 8002);
13.         } catch (UnknownHostException e) {
14.             e.printStackTrace();
```

```
15.         } catch (IOException e) {
16.             e.printStackTrace();
17.         }
18.     }
19.
20.     public static Socket getSocket() {
21.         if (socket == null)
22.             new Client();
23.         return socket;
24.     }
25.
26.     public static void close() {
27.         if (socket != null)
28.             try {
29.                 socket.close();
30.                 socket = null;
31.             } catch (IOException e) {
32.                 e.printStackTrace();
33.             }
34.     }
35.
36. }
```

Client 类和 Chapter03 中的 BookList 类一样都用了单例模式，关于单例模式在这里就不做详细讲解了。Client 类在第 10～18 行的私有构造函数中连接服务器，第 12 行 Socket 构造函数的第一个参数为服务器的 IP 地址（注意：由于用的是 Android 开发，所以不能用 localhost 代表服务器的 IP 地址），第二个参数为服务器的端口号。在 Client 中还有一个 close 方法，用于关闭 Socket 连接，并且释放 Socket 对象。

下面回到 ClientThread 类中，在 ClientThread 构造函数中获得 Socket 连接后，接着为成员变量 controller 和 mes 赋值，第 22、23 行获得 Socket 连接的输入输出管道。ClientThread 有一个 send 方法（第 43～50 行），这个方法实现了向服务器发送数据，在第 45 行通过 OutputStream 的 write 方法发送 mes 数据，在 write 方法中把 mes 通过 getBytes 方法变成 byte 数组，并且指定了编码为 GBK。在第 46 行调用 flush 方法刷新缓冲区。现在讲解 ClientThread 的核心方法 run。一旦线程启动，就开始执行 run 方法（第 29～41 行），run 方法中首先调用 send 方法发送数据，第 33～35 行实现接收服务器返回的信息，将得到的数据读入 byte 数组中，然后转换成编码格式为 GBK 的字符串，这是图 5-29 的第④步。在第 36 行调用 Controller 类中的 doResponse 方法对服务器返回的数据进行处理，这是图 5-28 的第⑤步，Controller 类中 doResponse 方法的代码如下：

```
1.  public void doResponse(String message) {
2.          Parser parser=new Parser();
3.          Response response=parser.parserResponse(message);
```

```
4.          String operate = response.getOperate();
5.          String result = response.getResult();
6.          Message msg = new Message();
7.          Bundle bundle = new Bundle();
8.          if (operate.equals("insert")) {
9.              bundle.putString("result", result);
10.             msg.setData(bundle);
11.             handler.sendMessage(msg);
12.         } else if (operate.equals("delete")) {
13.             bundle.putString("result", result);
14.             msg.setData(bundle);
15.             handler.sendMessage(msg);
16.         }else if (operate.equals("set")) {
17.             bundle.putString("result", result);
18.             msg.setData(bundle);
19.             handler.sendMessage(msg);
20.         }else if (operate.equals("select")) {
21.             bundle.putString("result", result);
22.             msg.setData(bundle);
23.             handler.sendMessage(msg);
24.         }
25.     }
```

doResponse 方法是对服务器返回的数据进行处理，第 2、3 行是对服务器返回数据的分析并将数据存储在 Response 类中，这是图 5-28 的第⑥步和第⑦步。那么首先来看一下 Parser 类的代码：

```
1.  package util.cqupt;
2.
3.  import model.cqupt.Book;
4.  import model.cqupt.BookList;
5.  import model.cqupt.Response;
6.
7.  public class Parser {
8.
9.      public Response parserResponse(String mes) {
10.         String[] message = mes.toString().split("\n");
11.         int length = message.length;
12.         String operate = message[0].substring(8, message[0].length());
13.         String result = message[length - 1].substring(7,
14.                 message[length - 1].length());
15.         Response response = Response.getResponse();
16.         response.setOperate(operate);
17.         response.setResult(result);
```

```
18.         BookList booklist = new BookList();
19.         for (int i = 1; i < message.length - 1; ++i) {
20.             String str = null;
21.             if (i == 1) {
22.                 str = message[1].substring(8, message[1].length());
23.             } else {
24.                 str = message[i];
25.             }
26.             Book book = parseBook(str);
27.             booklist.add(book);
28.         }
29.         response.setBooklist(booklist);
30.         return response;
31.     }
32.
33.     public Book parseBook(String str) {
34.         String[] mes = str.split("#");
35.         String id = mes[0];
36.         String name = mes[1];
37.         String price = mes[2];
38.         return new Book(id, name, price);
39.     }
40.
41. }
```

Parser 类中第 9～31 行为成员方法 parserResponse。在这个方法中实现了解析数据，将数据存放到 Response 类中并且返回 Response 对象。现在详细分析 parserResponse 方法。在第 10 行过滤掉数据的换行符，将内容放于 message 数组中，在第 11 行得到 message 数组长度，在第 12～14 行根据协议提取 message 数组中的内容，第 15～29 行把在 message 数组中提取的信息放入 Response 类中保存，parseBook 方法的作用是解析图书的信息。model.cqupt 包中 Response 类的代码如下：

```
1.  package model.cqupt;
2.
3.  public class Response {
4.      private static Response response;
5.      private BookList booklist;
6.      private String result;
7.      private String operate;
8.
9.      private Response() {
10.     }
11.
12.     public static Response getResponse() {
```

```
13.         if (response == null)
14.             response = new Response();
15.         return response;
16.     }
17.
18.     public void setOperate(String operate) {
19.         this.operate = operate;
20.     }
21.
22.     public void setResult(String result) {
23.         this.result = result;
24.     }
25.
26.     public void setBooklist(BookList booklist) {
27.         this.booklist = booklist;
28.     }
29.
30.     public String getOperate() {
31.         return operate;
32.     }
33.
34.     public String getResult() {
35.         return result;
36.     }
37.
38.     public BookList getBookList() {
39.
40.         return booklist;
41.     }
42. }
```

Response 类与 Client 类有相似之处，都只能被创建一次。Response 类有 4 个成员变量：response、operate、result 和 booklist，由名字可以知道这是服务器返回的全部信息，Response 就是一个存放返回内容的结果类。Response 定义了一个私有构造函数，防止外部调用。第 12～16 行 getResponse 方法的作用是得到 Response 对象。第 18～41 行通过 set 和 get 方法设置和得到成员服务器返回结果的信息，如 operate、result 和 booklist。

现在回到 Parser 类中的 parserResponse 方法，在 parserResponse 方法将所有解析后的结果都放入到 Response 中后，返回了 Response 对象给 Controller 类的 doResponse 方法，这是图 5-28 的第⑧步。接下来继续讲解 doResponse 方法中的剩余操作，在 doResponse 方法的第 4、5 行得到返回的方法和结果，在第 6、7 行定义一个 Message 对象和 Bundle 对象，在第 8～24 行通过判断在第 4 行得到的 operate 来执行不同的操作，例如第一个 insert 操作，在第 9 行将返回的结果存入 Bundle 中，在第 10 行将 bundle 对象放入 Message

对象里，在第 11 行，通过前面所学的 handler 用法，调用 handler 的 sendMessage 方法将数据发送到 InsertActivity 界面的消息队列中，InsertActivity 中定义的 handler 从消息队列中取出数据，通过 handlerMessage 方法对取出的数据进行处理，对界面进行更新操作。InsertActivity 定义 handler 的代码如下所示：

```
1.  private Handler handler = new Handler() {
2.          public void handleMessage(Message msg) {
3.              super.handleMessage(msg);
4.              Bundle b = msg.getData();
5.              String result = b.getString("result");
6.              buildDialog(result);
7.          }
8.      };
```

这是个匿名内部类，在第 2～7 行的 handleMessage 方法对 InsertActivity 界面更新，在第 4 行从 Message 对象中得到传递到消息队列中的数据，在第 5 行由 Bundle 的 getString 方法通过名称取得存放在 Bundle 的数据，然后通过调用 buildDialog 创建对话框并将结果显示到对话框中。在 doResponse 方法的其他判断中，都采用了同样的思想，此处不再赘述。

至此就完成了对使用 TCP Socket 的图书管理系统的讲解，下面继续讲解本节中所用服务器的实现。

5.5.3 使用 TCP Socket 的图书管理系统的服务器

本节讲述 5.4.1 节使用的服务器。首先运行 Server，运行效果如图 5-29 和图 5-30 所示。

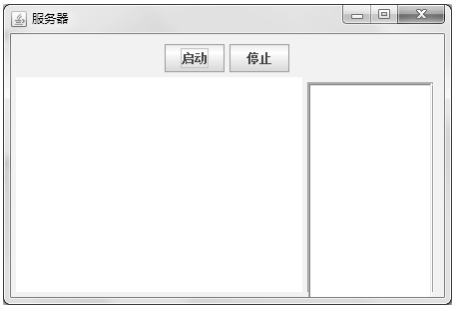

图 5-29 Server 运行效果

图 5-30 单击“启动”按钮

服务器端是用纯 Java 编写的，图 5-31 为 Server 工程的包结构。

ui.cqupt 包中是服务器的界面，model.cqupt 是模型，control.cqupt 包起到控制作用，控制 model.cqupt 中的 BookList 和 ui.cqupt 包的界面。util.cqupt 是工具包，对服务器接收和发送的数据打包和解析，net.cqupt 包实现 TCP 网络连接，data.cqupt 包实现 MySQL 数据库的连接。

下面将按照服务器端执行顺序进行讲解。首先是 ui.cqupt 包中的 MainClass，这个类是 Server 的界面，代码如下：

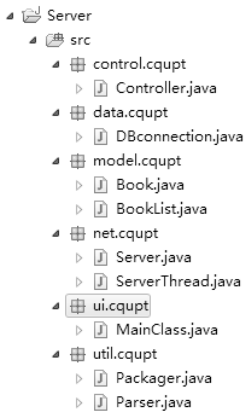

图 5-31　包结构

```
1.  package ui.cqupt;
2.
3.  import java.awt.BorderLayout;
4.  import java.awt.EventQueue;
5.  import java.awt.FlowLayout;
6.  import java.awt.List;
7.  import java.awt.event.ActionEvent;
8.  import java.awt.event.ActionListener;
9.
10. import javax.swing.JButton;
11. import javax.swing.JFrame;
12. import javax.swing.JOptionPane;
13. import javax.swing.JPanel;
14. import javax.swing.JTextArea;
15. import javax.swing.border.EmptyBorder;
16. import javax.swing.JScrollPane;
17.
18. import net.cqupt.Server;
19.
20. public class MainClass extends JFrame implements ActionListener {
21.
22.     private JTextArea textArea = new JTextArea();
23.     private JButton start = new JButton("启动");
24.     private JButton stop = new JButton("停止");
25.     private List list = new List(35);
26.     private Server server;
27.
28.     public static void main(String[] args) {
29.         EventQueue.invokeLater(new Runnable() {
30.             public void run() {
31.                 try {
32.                     MainClass frame = new MainClass();
33.                     frame.setVisible(true);
34.                 } catch (Exception e) {
35.                     e.printStackTrace();
36.                 }
37.             }
38.         });
39.     }
40.
41.     public MainClass() {
```

```
42.         setTitle("服务器");
43.         setDefaultCloseOperation(JFrame.EXIT_ON_CLOSE);
44.         setBounds(100, 100, 450, 300);
45.         JPanel contentPane = new JPanel();
46.         JPanel right = new JPanel();
47.         contentPane.setBorder(new EmptyBorder(5, 5, 5, 5));
48.         setContentPane(contentPane);
49.         contentPane.setLayout(new BorderLayout(0, 0));
50.
51.         JPanel panel = new JPanel();
52.         contentPane.add(panel, BorderLayout.NORTH);
53.         panel.setLayout(new FlowLayout(FlowLayout.CENTER, 5, 5));
54.
55.         panel.add(start);
56.         start.addActionListener(this);
57.
58.         panel.add(stop);
59.         stop.addActionListener(this);
60.         contentPane.add(textArea, BorderLayout.CENTER);
61.         right.add(new JScrollPane(list));
62.         getContentPane().add(right, BorderLayout.EAST);
63.         list.addActionListener(new ActionListener() {
64.
65.         public void actionPerformed(ActionEvent e) {
66.                 String item = list.getSelectedItem();
67.                 if (JOptionPane.showConfirmDialog(null, "是否断开与
                    当前客户端的连接") == JOptionPane.OK_OPTION) {
68.                     server.deleteThread(item);
69.                     deleteList(item);
70.                     textArea.append("客户端" + item + "连接中断" + "\n");
71.
72.                 }
73.
74.             }
75.
76.         });
77.
78.     }
79.
80.     public void actionPerformed(ActionEvent e) {
81.         if (e.getSource() == start) {
82.             server = new Server(this);
83.             server.start();
84.             textArea.append("服务器启动" + "\n");
```

```
85.             start.setEnabled(false);
86.         }
87.         if (e.getSource() == stop) {
88.             if (server != null)
89.                 server.interrupt();
90.             System.exit(0);
91.         }
92.     }
93.
94.     public void addList(String client) {
95.         list.add(client);
96.     }
97.
98.     public void deleteList(String item) {
99.         list.remove(item);
100.        }
101.
102.        public void setTextArea(String str) {
103.            textArea.append(str + "\n");
104.        }
105.
106.    }
```

MainClass 类用到了 Java 图形设计和事件监听的知识，下面对这段代码进行分析。MainClass 首先继承了 JFrame 和实现了 ActionListener。在第 41～78 行的构造函数中，我们构建了整个图形化界面，在第 56 行为启动按钮添加了监听器，在第 58 行为停止按钮添加了监听器。第 63～76 行是为 list 组件添加事件监听，并且实现监听器，list 组件显示所有连入的客户端，第 66 行通过 getSelectedItem 方法得到 list 上的条目名称，双击 list 上的条目就会执行第 67～72 行，生成一个判断框，当单击“确定”按钮时执行第 68～70 行，第 68 行删除此线程，在第 69 行调用在第 98～100 行定义的 deleteList 方法删除 list 组件中的这个条目。第 70 行在 Server 界面的 textArea 组件上显示客户端断开的信息。在第 80～92 行是启动和停止按钮的监听器，第 81～86 行是单击启动按钮执行的代码，单击启动按钮，首先实例化 Server 对象并且把整个 MainClass 对象传递给 Server，Server 是一个线程类，接收客户端的连接，后面将会详细讲解。第 83 行启动 Server 线程，第 84、85 行对界面操作，在 textArea 显示有客户端连接，把启动按钮设置成不可再次单击。第 87～91 行是单击退出按钮执行的代码，首先判断 Server 对象是否为 null，如果不是，则中断 Service 线程，然后退出界面；否则直接退出界面。第 94～96 行是 addList 对 list 的增加操作，每次连入一个客户端，在 Server 中调用就这个方法。第 102～104 是在 textArea 上显示信息上调用的。这就是 MainClass 的全部内容。

下面开始讲解 net.cqupt 包中的 Server 和 ServerThread。Server 类的作用是等待客户端的连接，一旦有新的客户端连接上，就会为这个客户端新建一个收发线程

ServerThread，服务器在 ServerThread 中接收客户端数据，并且对之解析、打包，最后发送回客户端。Server 类代码如下：

```
1.  package net.cqupt;
2.
3.  import java.io.IOException;
4.  import java.net.ServerSocket;
5.  import java.net.Socket;
6.  import java.util.HashMap;
7.
8.  import ui.cqupt.MainClass;
9.
10. import control.cqupt.Controller;
11.
12. public class Server extends Thread {
13.     private ServerSocket serverscoket;
14.     private HashMap<String, ServerThread> threadPool = new HashMap
        <String, ServerThread>();
15.     private Controller controller;
16.
17.     public Server(MainClass mainClass) {
18.         controller = new Controller(mainClass);
19.     }
20.
21.     public void run() {
22.         try {
23.             serverscoket = new ServerSocket(8002);
24.             while (!Server.interrupted()) {
25.                 Socket socket = serverscoket.accept();
26.                 String name = socket.getLocalAddress().
                    getHostAddress() + "::"
27.                         + socket.getPort();
28.                 controller.addClientView(name);
29.                 ServerThread st = new ServerThread(name, socket,
                    this);
30.                 threadPool.put(name, st);
31.                 st.start();
32.             }
33.         } catch (IOException e) {
34.             e.printStackTrace();
35.         } finally {
36.             close();
37.         }
38.
```

```
39.     }
40.
41.     public void deleteThread(String name) {
42.         ServerThread thread = threadPool.get(name);
43.         thread.interrupt();
44.         threadPool.remove(name);
45.     }
46.
47.     public Controller getController() {
48.         return controller;
49.     }
50.
51.     public void close() {
52.         if (serverscoket != null) {
53.             try {
54.                 serverscoket.close();
55.             } catch (IOException e) {
56.                 e.printStackTrace();
57.             }
58.         }
59.     }
60. }
```

Server 类继承了 Thread 类，在第 17～19 行的构造函数中，接收 MainClass 对象作为参数，在第 18 行实例化 control 对象并且把 MainClass 对象作为参数传递给 Controller。21～39 行是线程执行的 run 方法，第 23 行使用 ServerSocket(int port)实例化一个 ServerSocket 对象，port 参数传递端口号，这个端口就是服务器监听连接请求的端口，如果在这时出现错误，将抛出 IOException 异常对象并且执行在第 51～59 行定义的 close 方法关闭 ServerSocket 连接，否则将实例化 ServerSocket 对象并开始准备接收连接请求。接下来中第 24～32 行服务程序进入无限循环中，无限循环从第 25 行调用 ServerSocket 的 accept 方法开始，在调用开始后 accept 方法将导致调用线程阻塞直到连接建立。在建立连接后 accept 返回一个最近创建的 Socket 对象，该 Socket 对象绑定了客户程序的 IP 地址或端口号。在第 26 行得到客户端的 IP 地址和端口号，在第 28 行调用 Controller 类中的 addClientView 方法在界面显示客户端连接上的信息，Controller 类将在后面介绍。第 29 行为客户端新开一个线程，将客户端的名称、Socket 对象和 Server 对象作为参数传递给 ServerThread 线程类。在第 30 行将这个 ServerThread 线程添加到在第 14 行定义的 threadPool 成员变量中保存，然后调用 start 运行线程，threadPool 的类型是 HashMap，以客户端名称作为 key，ServerThread 对象作为 value 存储。在第 41～45 行 deleteThread 方法是删除在 threadPool 中的线程并且在第 43 行调用线程的 interrupt 方法终止线程。第 47～49 行的 getController 是得到 control 对象，这个方法将在 ServerThread 中用到。现在就完成了对 Server 的讲解。

下面开始讲解 ServerThread 类，这是个接收和发送数据的类，和 5.4 节所讲的 ClientThread 有相同的功能。ServerThread 代码如下所示：

```
1.  package net.cqupt;
2.
3.  import java.io.IOException;
4.  import java.io.InputStream;
5.  import java.io.OutputStream;
6.  import java.net.Socket;
7.
8.  import util.cqupt.Parser;
9.
10. import control.cqupt.Controller;
11.
12. public class ServerThread extends Thread {
13.     private Socket socket;
14.     private String name;
15.     private InputStream in;
16.     private OutputStream out;
17.     private Server server;
18.
19.     private static final int SIZE = 1024;
20.
21.     public ServerThread(String name, Socket socket, Server server) {
22.         this.name = name;
23.         this.server = server;
24.         this.socket = socket;
25.         try {
26.             in = socket.getInputStream();
27.             out = socket.getOutputStream();
28.         } catch (IOException e) {
29.             e.printStackTrace();
30.         }
31.     }
32.
33.     public void run() {
34.         try {
35.             while (!this.isInterrupted()) {
36.                 byte[] buffer = new byte[SIZE];
37.                 int index = in.read(buffer);
38.                 String message = new String(buffer, 0, index, "GBK");
39.                 Parser parser = new Parser();
40.                 String operate = parser.getOperate(message);
41.                 Controller controller = server.getController();
```

```
42.                 if (operate.equals("exit")) {
43.                     server.deleteThread(name);
44.                     controller.deleteClientView(name);
45.                     break;
46.                 }
47.                 String response = controller.doResponse(message);
48.                 send(response);
49.             }
50.
51.         } catch (IOException e) {
52.             e.printStackTrace();
53.
54.         } finally {
55.             close();
56.         }
57.     }
58.
59.     public void send(String mes) {
60.         try {
61.             out.write(mes.getBytes("GBK"));
62.             out.flush();
63.         } catch (IOException e) {
64.             e.printStackTrace();
65.         }
66.
67.     }
68.
69.     private void close() {
70.         try {
71.             if (in != null)
72.                 in.close();
73.             if (out != null)
74.                 out.close();
75.             if (socket != null)
76.                 socket.close();
77.         } catch (IOException e) {
78.             e.printStackTrace();
79.         }
80.     }
81.
82. }
```

ServerThread 是一个继承了 Thread 的线程类，在 ServerThread 声明了 5 个私有成员变量：name 代表客户端的名称，socket 代表客户端，server 是 Server 对象的引用，in 和

out 是输入输出流。在第 21～31 行的构造函数里为各个成员变量赋值，并且实例化 in 和 out 流，通过 socket 的 getInputStream 和 getOutputStream 的客户端与服务器之间的通道。第 33～57 行的 run 方法实现了服务器与客户端的交互功能，首先在第 35 行判断线程是否被中断，如果没有中断，则在这个 while 循环里与客户端进行交互，在第 36 行定义一个 byte 数据，长度为 SIZE，在第 19 行定义了常量 SIZE 并复制为 1024。在第 37 行通过 InputStream 的 read 方法将客户端发来的数据读入到 byte 数组中，值得注意的是，read 方法是阻塞的，也就是说，如果客户端没有数据发送到服务器，那么程序就会阻塞在第 37 行，直到接收到了数据，才继续执行下面的代码。在第 38 行将读到的数据以 GBK 的形式转换成字符串。然后在第 39、40 行对转换后的字符串解析，并且返回解析后客户端的请求操作，在第 42～46 行是判断客户端的请求操作是否为退出，如果是退出操作，则执行第 43～45 行，第 43 行在 Server 对象中的 threadPool 中删除此线程，第 54 行在 MainClass 界面上删除此客户端信息，跳出循环执行 finally 中的 close 方法关闭流，close 方法在第 69～80 行定义，作用是关闭输入输出流和 socket 连接。如果客户端发送的是其他的请求操作，则执行第 47、48 行，第 47 行调用 controller 对象的 doResponse 方法，返回打包好的服务器响应数据，在第 48 行调用在第 59～67 行定义的 send 方法发送数据到客户端。send 方法里实现了服务器的发送功能，在第 61 行通过 write 方法以 Bytes 字节，编码方式为 GBK 发送到客户端，第 62 行刷新缓冲区。

下面讲解控制类 Controller，代码如下：

```
1.  package control.cqupt;
2.
3.  import ui.cqupt.MainClass;
4.  import util.cqupt.Packager;
5.  import util.cqupt.Parser;
6.  import model.cqupt.Book;
7.  import model.cqupt.BookList;
8.
9.  public class Controller {
10.     private MainClass mainclass;
11.     private String result;
12.
13.     public Controller(MainClass mainclass) {
14.         this.mainclass = mainclass;
15.     }
16.
17.     public String doResponse(String request) {
18.         Parser parser = new Parser();
19.         String operate = parser.getOperate(request);
20.         String content = parser.getContent(request);
21.         String response = null;
22.         if (operate.equals("insert")) {
```

```
23.             addBook(content);
24.             Packager packager = new Packager();
25.             response = packager.addPackage(operate, result);
26.         } else if (operate.equals("delete")) {
27.             deleteBook(content);
28.             Packager packager = new Packager();
29.             response = packager.deletePackage(operate, result);
30.         } else if (operate.equals("set")) {
31.             setBook(content);
32.             Packager packager = new Packager();
33.             response = packager.setPackage(operate, result);
34.         } else if (operate.equals("select")) {
35.             getBookList();
36.             Packager packager = new Packager();
37.             response = packager.searchPackage(operate, result);
38.         }
39.         return response.toString();
40.     }
41.
42.     public void addBook(String content) {
43.         Parser parser = new Parser();
44.         Book book = parser.parseBook(content);
45.         BookList booklist = new BookList();
46.         if (booklist.insert(book)) {
47.             result = "插入成功";
48.         } else {
49.             result = "插入失败,已有此图书";
50.         }
51.     }
52.
53.     public void deleteBook(String content) {
54.         BookList booklist = new BookList();
55.         if (booklist.delete(content)) {
56.             result = "删除成功";
57.         } else {
58.             result = "删除失败,没有此图书";
59.         }
60.     }
61.
62.     public void setBook(String content) {
63.         BookList booklist = new BookList();
64.         Parser parser = new Parser();
65.         Book book = parser.parseBook(content);
66.         if (booklist.set(book)) {
```

```
67.             result = "修改成功";
68.         } else {
69.             result = "修改失败，没有此图书";
70.         }
71.     }
72.
73.     public void getBookList() {
74.         BookList booklist = new BookList();
75.         if (booklist != null) {
76.             result = "查询成功 ";
77.         } else {
78.             result = "查询失败";
79.         }
80.
81.     }
82.
83.     public void deleteClientView(String name) {
84.         mainclass.setTextArea("客户端" + name + "退出");
85.         mainclass.deleteList(name);
86.     }
87.
88.     public void addClientView(String name) {
89.         mainclass.addList(name);
90.         mainclass.setTextArea(name + "客户端连接上");
91.     }
92.
93. }
```

Controller 类实现的功能是操作 BookList 类和 MainClass 类更改界面。首先在第 13～15 行的构造函数里为 MainClass 对象赋值，MainClass 对象在第 83～86 行的 deleteClientView 方法中对 MainClass 界面的 TextArea 和 List 对象操作，实现删除客户端。MainClass 对象在第 88～91 行的 addClientView 方法中同样对 MainClass 界面的 TextArea 和 List 对象操作，实现的是增加客户端的操作。

下面讲解 Controller 类在 ServerThread 中调用的 doResponse 方法，第 17～40 行自定义了 doResponse 方法，实现对服务器端收到的消息的解析，根据客户端的请求操作 BookList，并且将响应的信息打包返回。第 18～20 行是用工具类 Parser 解析消息，并且取得解析后的请求和内容。第 22～38 行根据解析后的请求执行相应的增、删、改、查操作。这里选取增加操作进行讲解，当请求操作为 insert 时，执行第 23～25 行，第 23 行调用 addBook 方法，并把内容传递到 addBook 方法中，addBook 方法是在第 42～51 行定义的，第 43、44 行解析内容并返回 book 对象，在第 45 行创建 BookList 对象，在第 46 行调用 BookList 对象中的 insert 方法插入图书，根据成功与否，对 result 变量赋值。在 doResponse 方法的第 24、25 行完成对响应信息的打包，最后在第 39 行将打包后的数

据转换成的字符串并返回。删、改、查的操作流程和增加操作是一样的，此处不再赘述。

下面讲解 util.cqupt 包中的 Parser 类和 Packager 类，这两个类是工具类，实现对数据的解析和打包功能。

```
1.  package util.cqupt;
2.
3.  import model.cqupt.Book;
4.
5.  public class Parser {
6.
7.      public String getOperate(String request) {
8.          String[] message = request.split("\n");
9.          String operate = message[0].substring(8, message[0].length());
10.         return operate;
11.     }
12.
13.     public String getContent(String request) {
14.         String[] message = request.split("\n");
15.         String content = message[1].substring(8, message[1].length());
16.         return content;
17.     }
18.
19.     public Book parseBook(String str) {
20.         String[] mes = str.split("#");
21.         String id = mes[0];
22.         String name = mes[1];
23.         String price = mes[2];
24.         return new Book(id, name, price);
25.     }
26. }
```

Parser 类中实现了对客户端发送过来的请求操作以及内容和图书信息的解析。第 7～11 行 getOperate 方法是对请求操作的解析，通过 split 函数过滤掉换行符；第 13～17 行 getContent 是对内容的解析，同样是用 split 过滤掉换行符；第 19～24 行 parseBook 是对图书信息的解析。这 3 个方法解析数据都是根据自定义的协议进行解析的。

Packager 类是对服务器响应信息按照协议进行打包的工具类。Packager 类的代码如下：

```
1.  package util.cqupt;
2.
3.  import model.cqupt.Book;
4.  import model.cqupt.BookList;
5.
6.  public class Packager {
```

```
7.      public String addPackage(String operate, String result) {
8.          BookList booklist = new BookList();
9.          StringBuffer mes = new StringBuffer("");
10.         String books = booklist.getBookList();
11.         mes.append("operate:" + operate + "\n");
12.         mes.append("content:" + books + "\n");
13.         mes.append("result:" + result + "\n");
14.
15.         return mes.toString();
16.     }
17.
18.     public String searchPackage(String operate, String result) {
19.         BookList booklist = new BookList();
20.         StringBuffer mes = new StringBuffer("");
21.         String books = booklist.getBookList();
22.         mes.append("operate:" + operate + "\n");
23.         mes.append("content:" + books + "\n");
24.         mes.append("result:" + result + "\n");
25.         System.out.println(mes.toString());
26.         return mes.toString();
27.     }
28.
29.     public String deletePackage(String operate, String result) {
30.         BookList booklist = new BookList();
31.         StringBuffer mes = new StringBuffer("");
32.         String books = booklist.getBookList();
33.         mes.append("operate:" + operate + "\n");
34.         mes.append("content:" + books + "\n");
35.         mes.append("result:" + result + "\n");
36.
37.         return mes.toString();
38.     }
39.
40.     public String setPackage(String operate, String result) {
41.         BookList booklist = new BookList();
42.         StringBuffer mes = new StringBuffer("");
43.         String books = booklist.getBookList();
44.         mes.append("operate:" + operate + "\n");
45.         mes.append("content:" + books + "\n");
46.         mes.append("result:" + result + "\n");
47.
48.         return mes.toString();
49.     }
50.
```

```
51.     public String BookPackage(Book book) {
52.         StringBuffer str = new StringBuffer("");
53.         str.append(book.getId() + "#" + book.getName() + "#" +
            book.getPrice());
54.         return str.toString();
55.     }
56.
57. }
```

Packager 类实现了对图书信息的增、删、改、查以及打包功能。Packager 打包方法是根据我们定义的协议打包的。读者可以根据定义的协议来学习代码，这里简单地介绍一下增加操作对应的打包功能。在第 7～16 行定义了 addPackage 方法，第 8 行实例化 BookList 对象，第 10 行通过 getBookList 方法以字符串的方式返回所有的图书，接下来将服务器响应的信息放到一个大的字符串里，并返回这个字符串。删、改、查的方法类似，此处不再赘述。

接下来去看看 model 包中的 BookList 类。BookList 类存储图书并且对 MySQL 数据库进行访问，BookList 的代码如下：

```
1.  package model.cqupt;
2.
3.  import java.sql.Connection;
4.  import java.sql.ResultSet;
5.  import java.sql.SQLException;
6.  import java.sql.Statement;
7.  import java.util.ArrayList;
8.
9.  import util.cqupt.Packager;
10.
11. import data.cqupt.DBconnection;
12.
13.
14. public class BookList extends ArrayList<Book> {
15.
16.
17.     private static final long serialVersionUID = 1L;
18.
19.     public BookList() {
20.         super();
21.         String sql = "SELECT id,name,price FROM books";
22.         DBconnection d = new DBconnection();
23.         Connection con = null;
24.         Statement sta = null;
25.         ResultSet re = null;
```

```
26.         try {
27.             con = d.getConnect();
28.             sta = con.createStatement();
29.             re = sta.executeQuery(sql);
30.             while (re.next()) {
31.                 add(new Book(re.getString(1), re.getString(2),
                    re.getString(3)));
32.             }
33.         } catch (SQLException e) {
34.             e.printStackTrace();
35.         } finally {
36.             d.close(con, sta, re);
37.         }
38.     }
39.
40.     public boolean insert(Book book) {
41.         if (checkId(book.getId())) {
42.             String id = book.getId();
43.             String name = book.getName();
44.             String price = book.getPrice();
45.             add(book);
46.             String sql = "INSERT INTO books(id,name,price)" +
                "VALUES('" + id
47.                     + "','" + name + "','" + price + "')";
48.             DBconnection d = new DBconnection();
49.             Connection con = null;
50.             Statement sta = null;
51.             try {
52.                 con = d.getConnect();
53.                 sta = con.createStatement();
54.                 sta.executeUpdate(sql);
55.             } catch (SQLException e) {
56.                 e.printStackTrace();
57.             } finally {
58.                 d.close(con, sta);
59.             }
60.             return true;
61.         } else {
62.             return false;
63.         }
64.     }
65.
66.     public boolean delete(String name) {
67.         if (checkName(name)) {
```

```
68.             String sql = "DELETE FROM books WHERE name='" + name + "'";
69.             DBconnection d = new DBconnection();
70.             Connection con = null;
71.             Statement sta = null;
72.             try {
73.                 con = d.getConnect();
74.                 sta = con.createStatement();
75.                 sta.executeUpdate(sql);
76.             } catch (SQLException e) {
77.                 e.printStackTrace();
78.             } finally {
79.                 d.close(con, sta);
80.             }
81.             return true;
82.         } else {
83.
84.             return false;
85.         }
86.     }
87.
88.     public boolean set(Book book) {
89.         if (!checkId(book.getId())) {
90.             String id = book.getId();
91.             String name = book.getName();
92.             String price = book.getPrice();
93.             int index = getIndex(id);
94.             set(index, book);
95.             String sql = "UPDATE books SET name='" + name + "'," +
                "price='"
96.                     + price + "' WHERE id='" + id + "'";
97.             DBconnection d = new DBconnection();
98.             Connection con = null;
99.             Statement sta = null;
100.                try {
101.                    con = d.getConnect();
102.                    sta = con.createStatement();
103.                    sta.executeUpdate(sql);
104.                } catch (SQLException e) {
105.                    e.printStackTrace();
106.                } finally {
107.                    d.close(con, sta);
108.                }
109.                return true;
110.            } else {
```

```
111.                return false;
112.            }
113.        }
114.
115.        public String getBookList() {
116.            StringBuffer books = new StringBuffer("");
117.            Packager packager = new Packager();
118.            for (int i = 0; i < this.size(); ++i) {
119.                if (i == this.size() - 1) {
120.                    String s = packager.BookPackage(this.get(i));
121.                    books.append(s);
122.                } else {
123.                    Book book = this.get(i);
124.                    String str = packager.BookPackage(book);
125.                    books.append(str + "\n");
126.                }
127.            }
128.            return books.toString();
129.        }
130.
131.        public boolean checkId(String id) {
132.            for (int i = 0; i < this.size(); ++i) {
133.                Book book = this.get(i);
134.                String bookid = book.getId();
135.                if (bookid.equals(id))
136.                    return false;
137.            }
138.            return true;
139.        }
140.
141.        public boolean checkName(String name) {
142.            for (int i = 0; i < this.size(); ++i) {
143.                Book book = this.get(i);
144.                if (book.getName().equals(name)) {
145.                    remove(i);
146.                    return true;
147.                }
148.            }
149.            return false;
150.        }
151.
152.        public int getIndex(String bookid) {
153.            int i = 0;
154.            for (; i < this.size(); ++i) {
```

```
155.                Book book = this.get(i);
156.                String id = book.getId();
157.                if (id.equals(bookid)) {
158.                    break;
159.                }
160.            }
161.            return i;
162.        }
163.
164.    }
```

BookList 类继承了 ArrayList，所以可以当作 ArrayList 来使用。在第 19～38 行 BookList 的构造方法中，通过访问 MySQL 数据库，将数据库中的数据存入 BookList 对象中，相当于查询数据库的功能。第 21 行是查询数据库的 SQL 语句，第 22 行实例化 DBconnection 对象，在第 23 行声明数据库连接，在第 27 行得到连接。第 24 行是声明数据库操作，并在第 28 行通过 createStatement 方法得到 Statement 对象。第 25 行声明结果集 ResultSet，在第 29 行通过 executeQuery 执行第 21 行的 SQL 语句得到 ResultSet 对象。在第 30～32 行通过 ResultSet 对象的 next 方法遍历数据库，在第 31 行将得到的数据存放到 BookList 对象中，完成对数据库的查询功能，在构造方法里查询数据库是为了使数据库和 BookList 对象同步。在 BookList 类中主要有 4 个方法对 MySQL 数据库进行操作：增加（insert）、删除（delete）、修改（set）和查询（getBookList），这些方法的实现和构造方法差不多，读者可自行学习。

最后来看一看 data.cqupt 包中的 DBconnection 类，这个类完成了 Java 连接 MySQL 数据库的功能，代码如下所示：

```
1.  package data.cqupt;
2.  import java.sql.Connection;
3.  import java.sql.DriverManager;
4.  import java.sql.ResultSet;
5.  import java.sql.SQLException;
6.  import java.sql.Statement;
7.
8.  public class DBconnection {
9.      public static final String DBURL = "jdbc:mysql://localhost:3306/
        books";
10.     public static final String DBUSER = "root";
11.     public static final String DBPASS = "1111";
12.     public static final String DBRIVER = "org.gjt.mm.mysql.Driver";
13.
14.     static {
15.         try {
16.             Class.forName(DBRIVER);
17.         } catch (ClassNotFoundException e) {
```

```
18.             e.printStackTrace();
19.         }
20.     }
21.
22.     public Connection getConnect() throws SQLException {
23.         return DriverManager.getConnection(DBURL, DBUSER, DBPASS);
24.     }
25.
26.     public void close(Connection con, Statement sta, ResultSet re) {
27.         try {
28.             re.close();
29.             if (con != null && re != null) {
30.                 close(con, sta);
31.             }
32.         } catch (SQLException e) {
33.             e.printStackTrace();
34.         }
35.
36.     }
37.
38.     public void close(Connection con, Statement sta) {
39.         if (con != null && sta != null) {
40.             try {
41.                 sta.close();
42.                 con.close();
43.
44.             } catch (SQLException e) {
45.                 e.printStackTrace();
46.             }
47.         }
48.     }
49. }
```

DBconnection 完成与 MySQL 数据库的连接，首先第 9 行定义 MySQL 数据库驱动程序，第 10 行定义 MySQL 数据库连接的用户名，第 11 行定义 MySQL 数据库连接的密码，第 12 行定义 MySQL 数据库的连接地址。在第 14～20 行是一个静态代码块，静态代码块只在第一次创建 DBconnection 对象的时候执行一次，此处的作用是通过 Class.forName 加载驱动。第 22～24 行代码是连接数据库并且返回 Connection 对象的引用。第 26～28 行是两个重载的 close 方法，作用是关闭操作，按照先打开后关闭的顺序执行关闭操作。

这里完成了对 TCP 的服务器端的讲解。5.5.4 节将介绍 HTTP 的应用。

5.5.4 使用 HTTP 的图书管理系统

由本章学习的网络编程知识，可知 HTTP 的底层就是 TCP，所以本节采用 HTTP 实现的 Chapter05_http 是在 Chapter05_tcp 的 net.cqupt 包上做了少量修改。Chapter05_http 的包结构如图 5-32 所示。

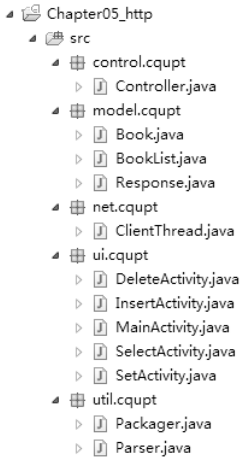

图 5-32 Chapter05_http 包结构

可以看到，net.cqupt 包中少了 Client 类，只有 ClientThread 类，ClientThread 的代码如下所示：

```
1.  package net.cqupt;
2.
3.  import java.io.IOException;
4.  import java.io.InputStream;
5.  import java.io.OutputStream;
6.  import java.net.HttpURLConnection;
7.  import java.net.URL;
8.
9.  import control.cqupt.Controller;
10.
11. public class ClientThread extends Thread {
12.     private InputStream in;
13.     private OutputStream out;
14.     private static final int SIZE = 1024;
15.     private String mes;
16.     private Controller controller;
17.
18.     public ClientThread(Controller controller, String mes) {
19.         this.controller = controller;
20.         this.mes = mes;
21.
22.     }
23.
24.     public void run() {
25.
26.         String urlStr = "http://192.168.1.100:8080/FiveHttpServer/
            server";
27.         URL url;
28.         try {
29.             url = new URL(urlStr);
30.             HttpURLConnection connection = (HttpURLConnection) url
31.                     .openConnection();
32.             connection.setDoOutput(true);
33.             connection.setRequestMethod("POST");
```

```
34.             out = connection.getOutputStream();
35.             send();
36.             in = connection.getInputStream();
37.             byte[] buffer = new byte[SIZE];
38.             int index = in.read(buffer);
39.             String message = new String(buffer, 0, index, "GBK");
40.             controller.doResponse(message);
41.         } catch (IOException e) {
42.             e.printStackTrace();
43.         }
44.
45.     }
46.
47.     public void send() {
48.         try {
49.             out.write(mes.getBytes("GBK"));
50.             out.flush();
51.             out.close();
52.         } catch (IOException e) {
53.             e.printStackTrace();
54.         }
55.     }
56. }
```

与 Chapter05_tcp 的 ClientThread 相比，这里对 run 方法做了修改，Chapter05_tcp 采用的是长连接，而此处是采用短连接。第 26 行定义了服务器的地址（注意：不能为 localhost），第 27 行声明了 URL 对象并在第 29 行实例化，第 30、31 行打开 HTTP 连接，第 32 行设置是否向 httpUrlConnection 输出，因为在第 33 行设置定了请求方法为 POST，参数要放在 HTTP 正文内，因此需要设为 true，默认情况下是 false。然后客户端就开始发送数据了，注意在 send 方法中，每次发送了数据都执行了第 51 行关闭输出流的操作。这就是 HTTP 图书管理系统，下面讲解 HTTP 图书管理系统的服务器端代码。

5.5.5 使用 HTTP 的图书管理系统的服务器

与 TCP 图书管理系统的服务器相比较，HTTP 的服务器少了界面，并使用了 tomcat 发布。Server_http 的包结构如图 5-33 所示。

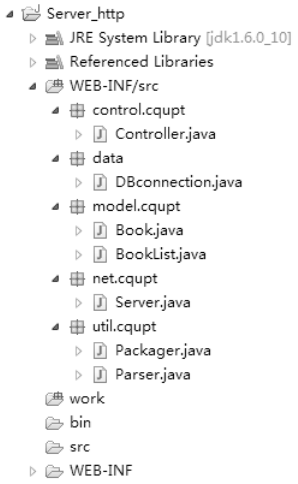

图 5-33 Server_http 包结构

这是一个 tomcat 工程，服务器通过 tomact 部署 Web 应用程序。Server_http 只在 TCP 图书管理系统的服务器上修改了 net.cqupt 包，Server 类代码如下所示：

```
1.  package net.cqupt;
2.
3.  import java.io.IOException;
4.  import control.cqupt.Controller;
5.  import javax.servlet.ServletInputStream;
6.  import javax.servlet.ServletOutputStream;
7.  import javax.servlet.http.HttpServlet;
8.  import javax.servlet.http.HttpServletRequest;
9.  import javax.servlet.http.HttpServletResponse;
10.
11. public class Server extends HttpServlet {
12.
13.     public void doPost(HttpServletRequest req, HttpServletResponse
        resp) {
14.         ServletInputStream in;
15.         ServletOutputStream out;
16.         try {
17.             in = req.getInputStream();
18.             out = resp.getOutputStream();
19.             int len = req.getContentLength();
20.             byte[] buffer = new byte[len];
21.             int index = in.read(buffer);
22.             String message = new String(buffer, 0, index, "GBK");
23.             Controller controller = new Controller();
24.             String response = controller.doResponse(message);
25.             out.write(response.getBytes("GBK"));
26.             in.close();
27.             out.close();
28.         } catch (IOException e) {
29.             e.printStackTrace();
30.         }
31.     }
32.
33. }
```

Server 继承了 HttpServlet 抽象类，并且覆盖了它的 doPost 方法。每当 Servlet 容器接收到客户端的请求，Servlet 容器就会解析 Web 客户的 HTTP 请求，然后创建一个 HttpRequest 对象，在这个对象中封装 HTTP 请求信息，创建一个 HttpResponse 对象。由于客户端是 post 请求，所以调用 HttpServlet 的 doPost 方法，把 HttpRequest 和 HttpResponse 对象作为 doPost 方法的参数传给 HttpServlet 对象。HttpServlet 调用 HttpRequest 的有关方法，获取 HTTP 请求信息，调用 HttpResponse 的有关方法，生成响应数据并传递给客户端。

第 14、15 行声明了 HTTP 的输入输出流，并在第 17、18 行实例化。其他操作和 TCP 图书管理系统的服务器端执行一样，此处不再赘述。值得注意的是，每次发送完数据都会执行第 26、27 行的 close 方法关闭输入输出流，因为 HTTP 是短连接的。