Chp15 网络编程

1 网络编程基础

1.1 网络协议的概念

要掌握网络编程，首先要掌握网络协议的概念。我们知道，网络主要是一种沟通的媒介。在不同机器上的不同程序，可以通过网络来交换数据，从而达到互联互通的功能。为了能够相互连通，首先要规定的是协议。所谓协议，可以理解为一些规则，这些规则规定了两个需要连通的点之间，应当如何进行数据交互。

举例来说，我们可以把语言当做一种协议。人在社会上需要交流，为了能够让两个人能够相互交流相互理解，我们定义了一种协议，这种协议就是语言。当两个需要沟通的人说同一种语言时，我们就说这两个人使用了同一种协议，这样就能够进行交流。而当两个人使用不同的语言时，就无法进行交流。

此外，协议是分层次的。对于不同的层次来说，规定的协议是不同的。例如，对于底层来说，为了连通网络必须要有网线。那么网线应该怎么做，水晶头的规格是什么，这是底层的协议。与之相对应的，ftp、http 等，就是在网络已经连通的情况下，不同的程序之间应当按照什么规则进行交互，这属于高层协议。

下面我们就介绍一下协议的分层。

1.2 OSI 七层结构和 TCP/IP 协议族

对于网络来说，有两种模型。一种是 OSI 七层模型，一种是 TCP/IP 协议族的 5 层模型。我们首先介绍 OSI 七层模型。

1.2.1 OSI 模型

OSI 七层模型如上图所示。对于这个模型，我们简单进行一下阐述。在这个模型中，物理层协议，规定的是网络连接中的一些电气特性，例如网线的电压电阻等等物理参数。数据链路层则规定了在网络中传输时，底层的包结构。这两个层次相对比较低层。

网络层规定了一个数据包从一台电脑（或者其他网络终端）发出之后，应当通过什么样的方式寻找路径，从而能够传输到另一台电脑上。传输层则规定了在能够找到终端之间的通路的情况下，应当如何收发数据包。这两层是网络通信软件的基础。

在之上的会话层、表示层和应用层，表示的是如何在网络已经能够收发数据包的情况下完成网络程序之间的通信。这部分内容我们不做进一步的具体阐述。

OSI 网络模型是一种标准化的网络模型。然而，这种网络模型在设计的时候过于理想化，

并且有些层次分的过细。在实际工业生产中，用的更多的是我们接下来要介绍的：TCP/IP 网络模型。

1.2.2 TCP 模型

上图是 TCP 网络模型。这个模型中，物理层、数据链路层与 OSI 网络模型中的两层相对应，而 IP 层对应着 OSI 模型中的网络层，传输层对应着 OSI 中的传输层。另外，TCP 模型中的应用层，对应着 OSI 模型中的会话层、表示层和应用层，可以说是把三层并为了一层。

对于 IP 层来说，在 TCP/IP 协议栈中，网络中某一台计算机的定位，靠的是 IP 地址这种方式。因此在 TCP/IP 模型中，把网络层也称之为 IP 层。如果那生活中的例子来比喻的话，那 IP 就可以当做是电话号码，通过电话号码能够唯一定位城市中的一台电话机。

而传输层，则可以认为是相互沟通的基础。从技术上说，传输层决定了一个数据包是如何从一台电脑传输到另一台电脑。传输层有两个协议：TCP 和 UDP 协议，这两个协议各有不同。TCP 协议是一个有连接、可靠的协议；而 UDP 协议是一个无连接，不可靠的协议。关于这两个协议的使用，我们在后面的编程中会进一步详细阐述。如何使用传输层进行网络通信，这是本章的重点。

应用层指的是应用程序使用底层的网络服务，典型的应用层协议包括 ftp、http 协议等。

事实上，大部分 Java 开发者，绝大多数时间都是在开发应用层方面的内容。而这一章介绍的内容，也许在你的 Java 程序员生涯中并不会经常接触。但是，掌握这些相对底层的技术，了解底层工作的原理，对以后理解 Web 编程是有非常大的帮助的。

下面我们就分别介绍传输层的两个协议。

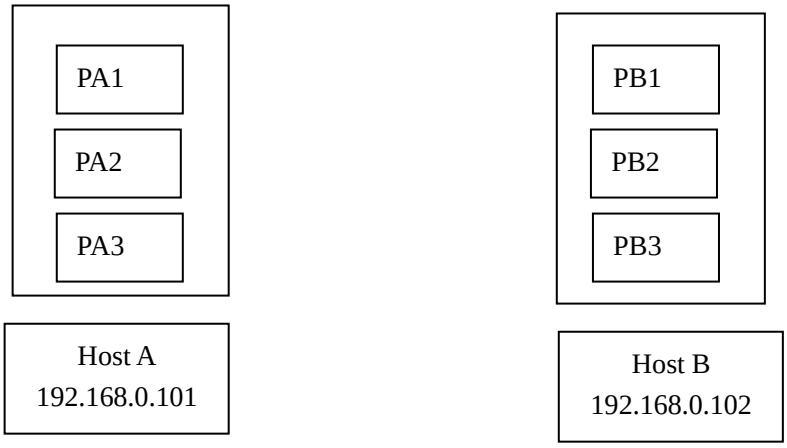
2 TCP 编程

2.1 TCP 协议简介

TCP 协议是传输层的协议，这个协议也是使用最广泛的协议之一。与 UDP 协议相比，TCP 协议的特点是：TCP 协议是一个有连接、可靠的协议。所谓有连接，指的是在进行 TCP 通信之前，两个需要通信的主机之间要首先建立一条数据通道，就好像打电话进行交流之前，首先要让电话接通一样。所谓可靠，指的是 TCP 协议能够保证：1、发送端发送的数据不会丢失；2、接收端接受的数据包的顺序，会按照发送端发送的包的顺序接受。也就是说，TCP 协议能够保证数据能够完整无误的传输。

此外，要进行 TCP 编程，还要理解端口号的概念。在网络通信的时候，我们连接服务器时，往往不仅仅要知道服务器的 ip 地址，还要知道服务器的端口号。那什么是端口号呢？

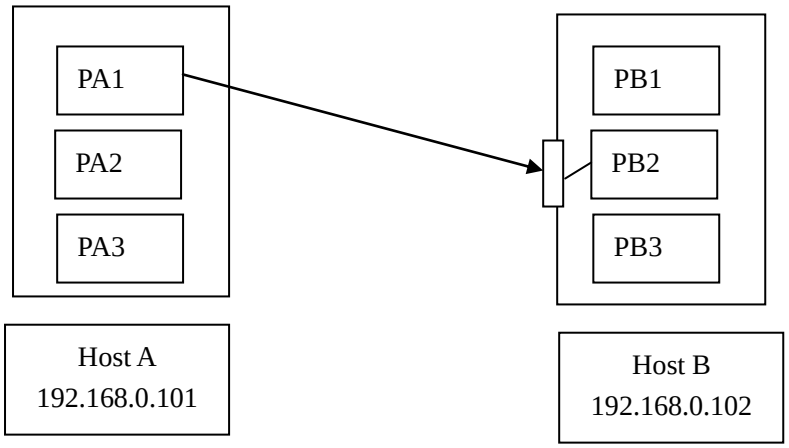
网络通信的本质，就是进程间通信。比如，两台电脑进行 qq 聊天，本质上就是两台电脑中的 qq 进程进行相互通信。对于这种情况的描述，如下图所示：



Host A 中运行了三个进程：PA1、PA2、PA3，Host B 中也运行了三个进程：PB1、PB2、PB3。所谓的网络通信，指的就是 HostA 中的某个进程和 HostB 中的某个进程进行通信。假设 PA1 要和 PB2 进行通信。

那么首先，PA1 要定位到网络中的某一台电脑，它可以使用 IP 地址，在本例中，使用 192.168.0.102，就能定位到 Host B。下一步，由于 HostB 中存在多个进程，那如何才能说明要跟哪一个进程通信呢？

HostB 中的每个进程会绑定在一个端口号上，假设 PB2 绑定了 9000 端口，因此，HostA 如果要通信的话，就可以连接到 HostB 的 9000 端口上。如下图：



可以这么来理解端口号：IP 地址就好比是一个单位的电话号码，而端口号则好比是某一部电话的分机号。通过电话号码找到某一个单位，而通过分机号则可以找到具体的个人；这就类似于通过 IP 地址可以定位某一台电脑，而通过端口号可以找到电脑中某一个的进程。

值得注意的是，在一台主机中，每个进程端口号上只能绑定唯一的一个进程。

2.2 TCP 编程的基本步骤

我们下面通过一个例子，来解释一下基本的 TCP 编程。TCP 编程又称为 Socket 编程，主要的类有两个：java.net.ServerSocket 和 java.net.Socket。

首先，从服务器端开始。在服务器端首先要做的是创建一个 ServerSocket 类型的对象。创建这个对象的时候，使用下面的方式：

```
ServerSocket ss = new ServerSocket(9000);
```

这样，就创建了一个 ServerSocket 对象，并把这个对象绑定到了 9000 端口。而 IP 地址，则是电脑当前的地址。在 Windows 中，可以使用 ipconfig 命令来查看当前地址，如下：

```
D:\book>ipconfig

Windows IP Configuration


Ethernet adapter 本地连接:

        Media State . . . . . . . . . . . : Media disconnected

Ethernet adapter 无线网络连接:

        Connection-specific DNS Suffix  . :
        IP Address. . . . . . . . . . . . : 150.236.56.101
        Subnet Mask . . . . . . . . . . . : 255.255.255.0
        Default Gateway . . . . . . . . . : 150.236.56.1
```

可以看到，当前地址为 150.236.56.101。

创建完 ServerSocket 对象之后，下一步就应该调用 ServerSocket 的 accept 方法。这个方法返回一个 Socket 对象。怎么来理解这个方法呢？我们可以理解为，这就是服务器端在等待连接的过程。如果没有一个客户端连接过来的话，则这个 accept 方法会一直不会返回。而一旦有一个客户端连接服务器，则这个方法就会返回。返回的 Socket 对象，我们可以比喻为一部电话机，很显然，通过一个电话号码和一个分机号，可以唯一的找到一台电话机。

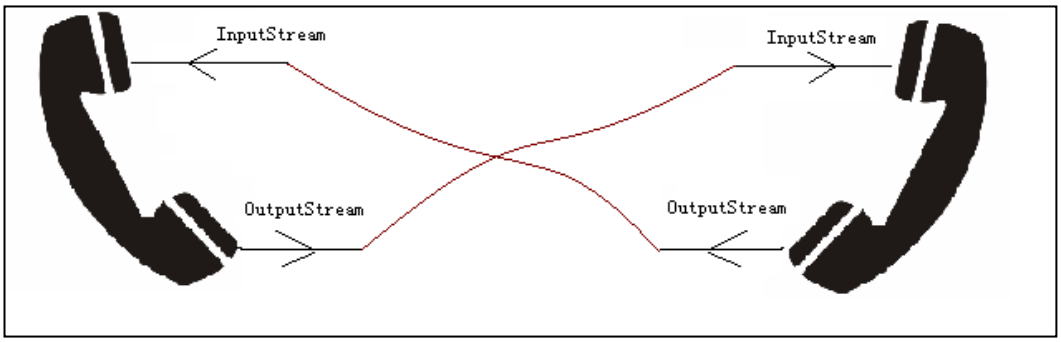
那客户端怎么连接服务器端呢？可以在客户端创建一个 Socket 对象连接服务端，如下：

```
Socket s = new Socket("150.236.56.101", 9000);
```

在创建 Socket 对象的时候，给出 IP 地址并给出端口号。这样就能创建一个 Socket 对象并连接服务器。

此时，客户端和服务器端都有一个 Socket 对象，并且保持连通，这就意味着两部电话已经接通了。接下来，就要进行数据传输了。

在数据传输的时候，需要调用 Socket 对象的 getInputStream 方法获得一个输入流，调用 getOutputStream 方法获得一个输出流。输入流就好比是电话机的听筒，能够听到从对方传来的声音，而输出流就好像是话筒，能够向对方说话。示意图如下：



上面的示意图说明，一端的 InputStream 连接在另一端的 OutputStream 上面，换句话说，在某一端的 Socket 上利用 OutputStream 写数据，就可以在另一端的 Socket 上用 InputStream 读数据。这就好比打电话的时候，向某一端的话筒说话，就能传到另一端的听筒。

获得 InputStream 和 OutputStream 之后，就能进行 I/O 操作了。当 I/O 操作完成之后，注意，不应该关闭 I/O 流，而应该调用 socket 对象的 close 方法，关闭 socket。这一点与之前的 I/O 不同。

总结一下通信的过程。服务器端：

1、创建 ServerSocket 对象（并绑定端口）

2、调用 accept 方法，等待来自客户端的连接

3、调用 getXXXStream 方法，进行 I/O

4、关闭 Socket

客户端：

1、创建 Socket 对象，并连接服务器

2、调用 getXXXStream 方法，进行 I/O

3、关闭 Socket

下面给出一个示例代码。在这个例子中，我们创建一个 TCP 服务器，和一个 TCP 客户端。客户端向服务器端发送一个“hello”字符串。服务端接受客户端发送的字符串，在后面增加一个“ from server”字符串，再返回给客户端。

服务器端代码：

```
import java.net.*;
import java.io.*;

public class TCPServer {
    public static void main(String[] args) throws Exception {
        //创建 ServerSocket 对象（并绑定端口）
        ServerSocket ss = new ServerSocket(9000);
        //调用 accept 方法
        Socket s = ss.accept();

        //调用 getXXXStream 方法，进行 I/O
        BufferedReader br =
            new BufferedReader(
                new InputStreamReader(s.getInputStream())
            );
        String line = br.readLine();

        PrintWriter pw = new PrintWriter(
                s.getOutputStream());
        pw.println(line + " from server");
        pw.flush();
        //关闭 Socket
        s.close();
    }
}
```

客户端代码：

```
import java.io.*;
import java.net.*;

public class TCPClient {
    public static void main(String[] args) throws Exception {
```

```
        //创建 Socket 对象（并连接服务器）
        Socket s = new Socket("150.236.56.101", 9000);
        //调用 getXXXStream 方法，进行 I/O
        PrintWriter pw = new PrintWriter(s.getOutputStream());
        pw.println("hello world");
        pw.flush();

        BufferedReader br = new BufferedReader(
               new InputStreamReader(s.getInputStream())
            );
        String line = br.readLine();
        System.out.println(line);
        //关闭 Socket
        s.close();
    }
}
```

2.3 多线程的服务器

上述的程序演示了基本的 TCP 服务器，但是这样的服务器是有很大的缺陷：启动服务器之后，只能让一个用户进行访问，访问之后服务器就会被关闭。这种服务器在现实生活中根本无法使用。

为了解决上面的问题，我们可以使用一个 while(true)的死循环，这样来保证服务器程序不会终止。这样，服务器端的步骤就改成：

创建 ServerSocket 对象（并绑定端口）
while(true){
    调用 accept 方法
    调用 getXXXStream 方法，进行 I/O
    关闭 Socket
}

注意，创建 ServerSocket 对象只需要一次，可以反复调用 accept 接听电话。

然而，这样的服务器依然不完善：这个服务器同时只能有一个客户端进行 I/O 操作。现实生活中，所有的服务器都应当是允许多用户同时浏览和访问的。如果一个网站做成这个样子：当一个用户浏览网站的时候，其他用户如果只能等待，那这样的网站是不会有人愿意访问的。

为了解决这个问题，我们可以把这个服务器设计成一个多线程的服务器。主线程只负责等待客户端的连接请求，一旦有客户端成功的连接过来，主线程负责创建并启动一个新的线程，由这个新线程负责与该客户端进行 I/O 操作，而此时的主线程，则继续去等待其他客户端的连接。修改成多线程的服务器基本工作步骤如下：

创建 ServerSocket 对象（并绑定端口）
while(true){
    调用 accept 方法
    创建并启动新线程进行 I/O 操作
}

我们可以看到，原有的 I/O 操作被挪到了新线程中，主线程只负责接受用户的连接。这

样，把速度较慢的 I/O 操作多线程处理，就能允许多个用户同时访问服务器。修改后的服务器代码如下：

```
import java.net.*;
import java.io.*;

class ServerThread extends Thread{
    private Socket s;

    public ServerThread(Socket s) {
        this.s = s;
    }

    public void run(){
        try{
            BufferedReader br =
                new BufferedReader(
                    new InputStreamReader(s.getInputStream())
                );
            String line = br.readLine();

            PrintWriter pw = new PrintWriter(
                    s.getOutputStream());
            pw.println(line + " from server");
            pw.flush();
        } catch (IOException e) {
            e.printStackTrace();
        }finally{
            try {
                s.close();
            } catch (IOException e) {
                e.printStackTrace();
            }
        }
    }
}

public class TCPServer {
    public static void main(String[] args) throws Exception {
        //创建 ServerSocket 对象（并绑定端口）
        ServerSocket ss = new ServerSocket(9000);
        while(true){
            //调用 accept 方法
            Socket s = ss.accept();
            //创建新线程进行 I/O
```

```
            Thread t = new ServerThread(s);
            t.start();
        }
    }
}
```

3 UDP 编程

3.1 UDP 协议简介

介绍完了 TCP 编程，我们开始介绍 UDP 编程。与 TCP 协议相比，UDP 是一个无连接，不可靠的协议。即：数据的发送方只负责将数据发送出去，数据的接受方只负责接受数据。发送方和接收方不会相互确认数据的传输是否成功。

使用 UDP 通信有点类似于写信，当我们寄信的时候，不需要想打电话一样事先准备一个连接，寄信人只知道把信寄了出去，但是对方有没有收到信，寄信人则一无所知。

相对于 TCP 而言，UDP 有一个优点：效率较高。因此，当我们在对数据传输的正确率不太关心，但是对传输效率要求较高的情况下，可以采用 UDP 协议。典型的使用 UDP 协议的是网络语音以及视频聊天应用。

下面我们来采用 UDP 协议进行 Java 网络编程。

3.2 UDP 编程基本步骤

UDP 编程所要用到的两个主要的类：DatagramSocket 和 DatagramPacket。我们可以把进行 UDP 通信比作收发传真。其中，DatagramSocket 可以当做是一台传真机，传真机既可以发传真，又可以收传真。而 DatagramPacket 则是需要传输的数据。

假设我们现在要从客户端向服务器端发送一个“hello server”，而服务器端回给客户端一个字符串：“hello client”。

首先看服务器端。先要创建一个 DatagramSocket 类型的对象，代码如下：

```
DatagramSocket socket = new DatagramSocket(9000);
```

此时，创建了一个绑定到端口号 9000 的 DatagramSocket。这样，就准备好了一个传真机。

再看客户端，客户端要发送数据，也得先创建一个 DatagramSocket，代码如下：

```
DatagramSocket socket = new DatagramSocket();
```

上面的代码没有指定客户端的端口，这样的话，系统会自动为客户端分配一个随机的端口号。

为什么服务器端的端口号不能让系统随机分配呢？因为服务器端的地址和端口号是必须要向外界公布，供客户端去访问的，如果一个网站向外公布：网站地址：xxxxxxxx，网站端口：随机，这样的话让用户究竟怎么访问你的网站？

因此，服务器端必须手动指定端口号。

客户端创建了一个 DatagramSocket 之后，就可以准备发送的数据了。首先应当准备一个 byte 数组，如下：

```
String str = "hello server";
byte[] data = str.getBytes();
```

这样，data 数组中保存的就是需要发送的数据内容。

然后，应当把 data 封装到一个 DatagramPacket 中，代码如下：

```
DatagramPacket packet = new DatagramPacket(
                data, 0, data.length,
                new InetSocketAddress("150.236.56.101", 9000)
            );
```

在创建 packet 的时候，给出了四个参数。前三个参数表示，发送的数据是 data 数组，从数组下标为 0 的位置开始发送，发送长度为 data.length 个字节。

第四个参数，是一个 InetSocketAddress 对象，这个对象表示数据要发送到哪个地址去。

当一切准备就绪之后，客户端就可以调用 socket 的 send 方法，发送 packet 对象。代码如下：

```
socket.send(packet);
```

这样，数据就从客户端发送到了服务器。

接下来，就是服务器如何接收了。在接收数据的时候，同样需要一个 DatagramPacket。这个 DatagramPacket 对象就好比是传真机上的纸，在收传真的时候，需要传真机里放上白纸，然后传真机根据发过来的内容，在白纸上打印出来。当传真接受完毕之后，这张纸上就记录了接受到的内容。

我们首先要创建一个空数组，这个数组就好像是白纸。

```
byte[] buf = new byte[100];
```

然后，根据这张白纸，创建一个 DatagramPacket：

```
DatagramPacket paper = new DatagramPacket(buf, 0, buf.length);
```

创建了 paper 对象之后，就可以调用 socket 对象的 receive 方法接受数据。

```
socket.receive(paper);
```

接受到数据之后，可以通过 paper 对象的下面几个方法获得相关信息：

paper.getSocketAddress()：获得发送者的地址。可以理解为获得发送传真的对方的号码，等一会儿回传真的时候，就可以用这个地址。

paper.getLength()：获得发送的数据的长度。虽然我们用一个长度为 100 的数据包去接受，但是接受到的数据长度有可能不满 100 个字节，可以通过调用这个 getLength 方法来获取实际接受的数据长度。

这样，我们就能获得客户端给服务器端发送的数据，代码如下：

```
String str = new String(buf, 0 , paper.getLength());
```

之后，我们就可以从服务器端向客户端发送数据。部分代码如下：

```
byte[] data = "hello client".getBytes();
DatagramPacket packet = new DatagramPacket(
                data, 0 ,data.length,
                paper.getSocketAddress()
            );
```

需要注意的是，由于是回信给客户端，因此，我们这次发送数据的收信人，就是 paper 数据包的发信人。所以，packet 对象的地址，就是 paper 对象的 getSocketAddress()方法的返回值。

最后，当两边完成通信之后，应当关闭 socket。

其余代码比较简单，不再赘述。完整代码如下：

服务器端：

```
import java.io.*;

public class UDPServer {
    public static void main(String[] args) throws Exception {
        //创建 socket
        DatagramSocket socket = new DatagramSocket(9000);

        //收数据
        byte[] buf = new byte[100];
        DatagramPacket paper = new DatagramPacket(
               buf, 0, buf.length);
        socket.receive(paper);
        String str = new String(buf, 0 , paper.getLength());
        System.out.println(str);

        //发送数据
        byte[] data = "hello client".getBytes();
        DatagramPacket packet = new DatagramPacket(
                  data, 0 ,data.length,
                  paper.getSocketAddress()
               );
        socket.send(packet);
        //关闭 socket
        socket.close();
    }
}
```

客户端：

```
import java.net.*;

public class UDPClient {
    public static void main(String[] args) throws Exception {
        DatagramSocket socket = new DatagramSocket();
        String str = "hello server";
        byte[] data = str.getBytes();
        DatagramPacket packet = new DatagramPacket(
               data, 0, data.length,
               new InetSocketAddress("150.236.56.101", 9000)
           );
        socket.send(packet);
```

```
            byte[] buf = new byte[100];
            DatagramPacket paper = new DatagramPacket(
                      buf, 0, buf.length
                   );
            socket.receive(paper);
            String msg = new String(buf, 0, paper.getLength());
            System.out.println(msg);
            socket.close();
        }
    }
```

4 URL 编程

最后，简单介绍一下 Java 中的 URL 编程。

URL 是统一资源定位符（Uniform Resource Locator）的简称，用于表示 Internet 上某一资源的地址。Internet 上的网络资源非常丰富，如常见的万维网和 FTP 站点上的各种文件、目录等。URL 的语法格式通常如下所示：

协议名 :// 主机名（或者 IP 地址）: 端口号 / 资源路径

如：http://localhost:8080/web/image/a.jpg 这就是一个 URL，指向了网络上的一个图片文件。（注意：localhost 是一个主机名，对应的 IP 地址是 127.0.0.1。这个地址指向的是本地主机，也就是当前这台计算机）

Java 语言同样提供了用 URL 来访问网络资源的编程方法：

Java 中的 URL 编程主要用到的类有两个：URL 和 URLConnection。

使用基本的 URL 编程非常简单：

1、创建 URL 对象

2、调用 URL 对象的 openConnection 方法，获得 URLConnection

3、调用 URLConnection 方法的 getInputStream，获得输入流，从而读取资源

4、I/O 操作

5、关闭 I/O 流

下面的代码能够读取新浪（http://www.sina.com.cn）首页的 html 源代码，并输出到屏幕上。

```
import java.net.*;
import java.io.*;

public class TestURL {
    public static void main(String[] args) throws Exception {
        //创建 URL 对象
        URL url = new URL("http://www.sina.com.cn");
        //调用 URL 对象的 openConnection 方法，获得 URLConnection
        URLConnection conn = url.openConnection();
        //调用 URLConnection 方法的 getInputStream
        InputStream in = conn.getInputStream();
        //I/O 操作
        BufferedReader br = new BufferedReader(
```

```
                new InputStreamReader(in));
            String line = null;
            while( (line=br.readLine()) != null ){
                System.out.println(line);
            }
            //关闭 I/O 流
            br.close();
        }
    }
```

该程序会显示一些“杂乱无章”的字符，不过不用担心，这些是一个 html 文件的源代码，这些代码如果在浏览器上显示出来，你就会看到一张漂亮的页面了！