1、课程名称：网络编程

2、知识点

2.2、本节预计讲解的知识点

1、 TCP、UDP 程序的实现
2、 ServerSocket 和 Scoket 类的使用
3、 URL、URLConnection 的使用

3、具体内容

3.1、网络编程的概述（了解）

什么是计算机网络：

把分布在不同地理区域的计算机与专门的外部设备用通信线路互连成一个规模大、功能强的网络系统，从而使众多的计算机可以方便地互相传递信息，共享硬件、软件、数据信息等资源。

计算机网络的主要功能

资源共享
信息传输与集中处理
均衡负荷与分布处理
综合信息服务(www/综合业务数字网络 ISDN)等

网络通信协议

要使计算机连成的网络能够互通信息，需要对数据传输速率、传输代码、代码结构、传输控制步骤、出错控制等制定一组标准，这一组共同遵守的通信标准就是网络通信协议，不同的计算机之间必须使用相同的通讯协议才能进行通信。

网络通信接口

为了使两个结点之间能进行对话，必须在它们之间建立通信工具(即接口)，使彼此之间能进行信息交换。接口包括两部分：

硬件装置:实现结点之间的信息传送
软件装置:规定双方进行通信的约定协议

TCP/IP

传输控制协议/因特网互联协议，又叫网络通讯协议，这个协议是 Internet 最基本的协议、Internet 国际互联网络的基础，简单地说，就是由网络层的 IP 协议和传输层的 TCP 协议组成的。

IP 地址：网络中每台计算机的一个标识号，本地 IP：127.0.0.1　　localhost

端口号(POST)：端口号的范围：0~65535 之间，0~1023 之间的端口数是用于一些知名的网络服务和应用

网络编程主要是指完成 C/S 程序的开发，程序的开发结构有两种：

- C/S（客户端/服务器），开发两套程序，两套程序需要同时维护，例如：QQ。CS 程序一般比较稳定
- B/S（浏览器/服务器），开发一套程序，客户端使用浏览器进行访问，例如：各个论坛。BS 程序一般稳定性较差，而且安全性较差。

但是，C/S 的程序开发在实际的 Java 应用中毕竟很少了，而且整个 java 基本上都是以 B/S 为主。

C/S 程序主要可以完成以下两种程序的开发：

- TCP：（Transmission Control Protocol）传输控制协议，采用三方握手的方式，保证准确的连接操作。
- UDP：（User Datagram Protocol）数据报协议，发送数据报，例如：手机短信或者是 QQ 消息。

所有的开发包都保存在 java.net 包中

TCP、UDP 的数据帧格式简单图例：

协议类型	源 IP	目标 IP	源端口	目标端口	帧序号	帧数据

其中协议类型用于区分 TCP、UDP

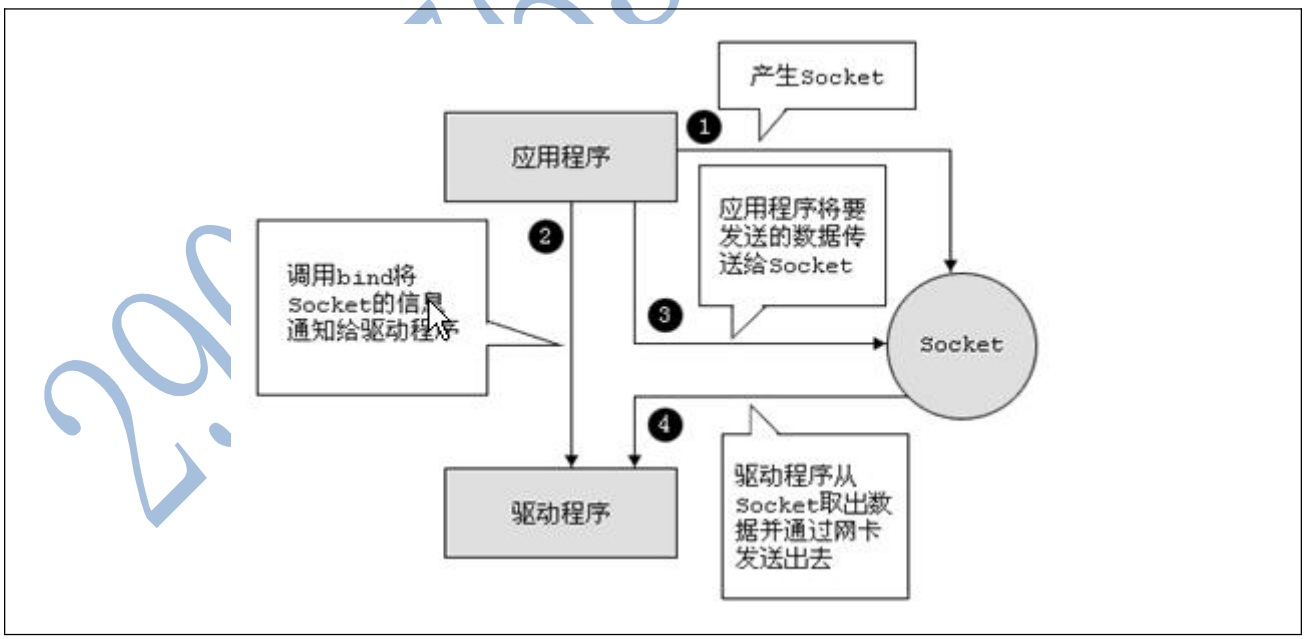
3.2、TCP 程序（了解）

如果要想实现 TCP 程序，则需要编写服务器和客户端，服务器使用 ServerSocket 和 Socket 两个类完成，客户端使用 Socket 类完成。

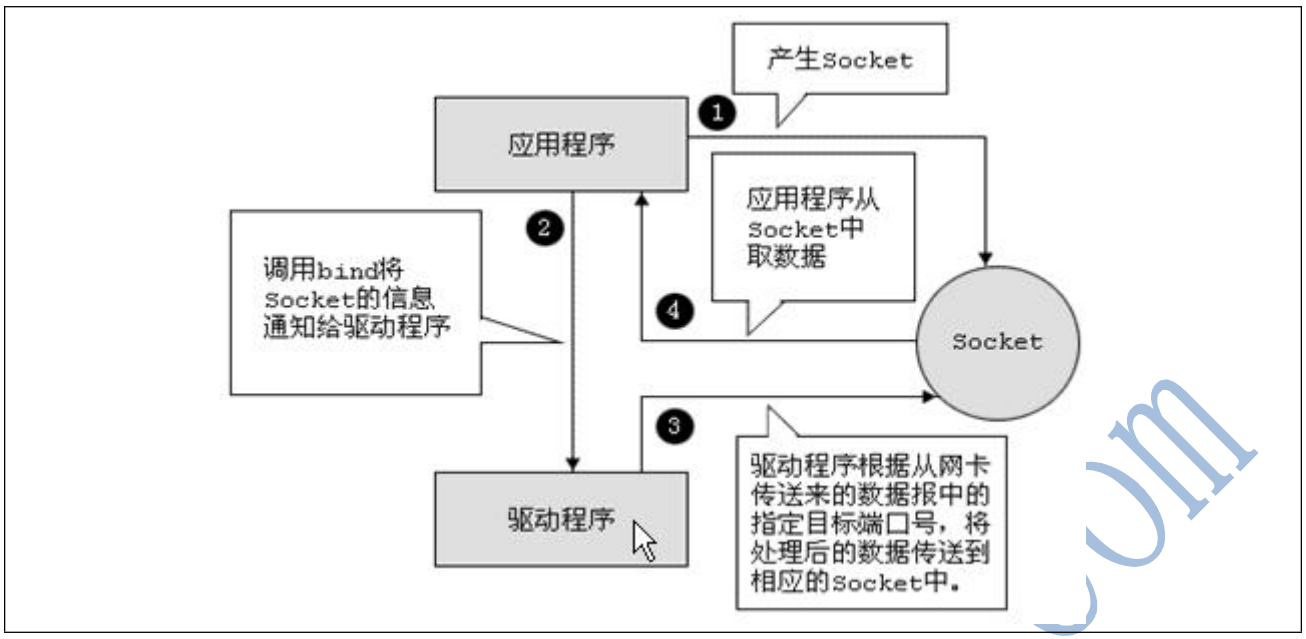
Socket：是网络驱动层提供给应用程序编程的接口和一种机制。

Socket 在应用程序中创建，通过一种绑定机制与驱动程序建立关系，告诉自己所对应的 IP 和 POST

Socket 数据发送过程：



Socket 数据接收过程：



Java 中的网络编程类：

位于.net 包中

DatagramSocket 类 用于 UDP 通信

ServerSocket 类 用于 TCP 通信的服务端

Socket 类用于 TCP 通信的服务器和客户端

3.2.1、编写服务器端

编写服务器端需要使用 ServerSocket 类，此类的常用方法如下：

No.	方法名称	类型	描述
1	public ServerSocket(int port) throws IOException	构造	构造的时候传入监听的端口号
2	public Socket accept() throws IOException	普通	接收客户端的连接

在以上的操作中使用 accept()方法接收客户端的连接，每一个客户端对服务器来讲都是一个 Socket 对象。Socket 类中的常用方法如下：

No.	方法名称	类型	描述
1	public Socket(String host,int port) throws UnknownHostException, IOException	普通	指明要连接的服务器名称和端口号
2	public InputStream getInputStream() throws IOException	普通	得到客户端的输入流
3	public OutputStream getOutputStream() throws IOException	普通	得到客户端的输出流

在 Socket 类中可以进行 IO 操作，如果服务器现在想向客户端输出的话，则必须使用 getOutputStream()方法取得客户端的输出流，如果客户端要是想接收服务器端输入的内容，则使用 getInputStream()方法得到。

下面先完成一个服务器程序，此服务器程序的主要功能，就是向客户端打印“Hello World!!!”。

范例：编写服务器端

```
package org.tcpdemo.hello;
import java.io.PrintStream;
import java.net.ServerSocket;
import java.net.Socket;
```

```
public class HelloServer {
    public static void main(String[] args) throws Exception {
        ServerSocket server = new ServerSocket(9999); // 在9999端口监听
        System.out.println("等待客户端连接。。。。。。。");
        Socket client = server.accept();// 程序执行到此位置进行等待
        // 得到客户端的输出流，准备输出
        PrintStream out = new PrintStream(client.getOutputStream());
        out.print("Hello World!!!");
        out.close();
        client.close();
        server.close();
    }
}
```

写完之后，运行此程序。服务器端在等待着客户端的连接。

那么使用 telnet 命令测试服务器端是否可以正常的使用。

- open localhost 9999

连接之后，服务器端停止执行，同时在 telnet 屏幕上显示了最终的内容。

网络编程中，进行开发的时候只管协议，而不管具体使用何种语言进行开发。

3.2.2、编写客户端

如果要想编写一个客户端，则可以使用 Socket 类完成，Socket 在实例化的时候要指定连接的主机和端口号。

范例：客户端开发

```
package org.tcpdemo.hello;
import java.io.InputStream;
import java.net.Socket;
import java.util.Scanner;
public class HelloClient {
    public static void main(String[] args) throws Exception {
        Socket client = new Socket("localhost", 9999);
        InputStream input = client.getInputStream();// 得到输入流
        Scanner scan = new Scanner(input);
        scan.useDelimiter("\n") ;
        System.out.println(scan.next());
        scan.close();
        client.close();
        input.close();
    }
}
```

此时，内容已经可以正确的接收到了。

3.3、ECHO 程序（了解）

ECHO 程序是在网络编程中一个比较经典的开发程序，客户端任意输入内容，之后服务器端在输入的内容前加上

“ECHO”返回给客户端。

那么这个时候服务器端要接收客户端输入的内容，也要向客户端发送数据，所以，这个时候就必须明确这样一点：

- 服务器端的输出流就是客户端的输入流
- 客户端的输出流就是服务器端的输入流

3.3.1、服务器端

在服务器端要保证可以一直接收用户的发送的数据，直到用户不想发送为止，所以，在服务器端接收客户端内容的操作应该采用循环的方式完成。

```
package org.tcpdemo.echo;
import java.io.BufferedReader;
import java.io.InputStreamReader;
import java.io.PrintStream;
import java.net.ServerSocket;
import java.net.Socket;
public class EchoServer {
    public static void main(String[] args) throws Exception {
        ServerSocket server = new ServerSocket(9999); // 在9999端口监听
        System.out.println("等待客户端连接。。。。。。");
        boolean flag = true;
        while (flag) { // 服务器可以接收到多个用户请求
            Socket client = server.accept();// 程序执行到此位置进行等待
            // 得到客户端的输出流，准备输出
            PrintStream out = new PrintStream(client.getOutputStream());// 客户端输出流
            BufferedReader buf = new BufferedReader(new InputStreamReader(
                    client.getInputStream()));
            boolean temp = true;
            while (temp) { // 要一直接收客户端的请求
                String str = buf.readLine(); // 接收内容
                if (str == null || "".equals(str)) {// 没有输入内容
                    temp = false;// 不再接收
                } else {
                    if ("bye".equals(str)) {
                        temp = false; // 不再接收
                    } else {
                        out.println("ECHO: " + str); // 将内容送回客户端
                    }
                }
            }
            out.close();
            buf.close();
            client.close();
        }
        server.close();
    }
```

```
}
```

此时，一个 Echo 程序的服务器端就完成了，服务器端完成之后，下面继续开发客户端。

3.3.2、客户端开发

客户端要向服务器端发送信息，还要接收服务器端的回应信息，还要从键盘接收数据。

```
package org.tcpdemo.echo;
import java.io.BufferedReader;
import java.io.InputStreamReader;
import java.io.PrintStream;
import java.net.Socket;
public class EchoClient {
    public static void main(String[] args) throws Exception {
        Socket client = new Socket("localhost", 9999);
        BufferedReader input = new BufferedReader(new InputStreamReader(
                System.in));// 从键盘接收数据
        BufferedReader echoInput = new BufferedReader(new InputStreamReader(
                client.getInputStream()));// 从服务器端接收数据
        PrintStream out = new PrintStream(client.getOutputStream());// 向服务器端打印信息
        boolean flag = true;
        while (flag) {
            System.out.print("请输入要发送的内容: ");
            String str = input.readLine();
            if (str == null || "".equals(str)) {
                flag = false;
            } else {
                if (str.equals("bye")) {
                    flag = false;
                } else {
                    out.println(str);
                    System.out.println(echoInput.readLine()) ;
                }
            }
        }
        client.close();
    }
}
```

此时已经完成了客户端的开发，但是这样的程序本身存在一个极大的漏洞，此程序的服务器端每次只能有一个客户端连接。

3.3.3、为服务器端加入多线程

之前的程序中，所有的操作都是单线程的处理操作，所以，每次只能有一个客户端进行连接，如果现在要想进行多个客户端的连接，则必须加入多线程的处理机制，将每一个连接的客户端都创建一个新的线程对象。

范例：编写线程操作类

```
package org.tcpdemo.echothread;
import java.io.BufferedReader;
import java.io.InputStreamReader;
import java.io.PrintStream;
import java.net.Socket;
public class ThreadEcho implements Runnable {
    private Socket client = null;
    public ThreadEcho(Socket client) {
        this.client = client;
    }
    public void run() {
        try {
            PrintStream out = new PrintStream(client.getOutputStream());// 客户端输出流
            BufferedReader buf = new BufferedReader(new InputStreamReader(
                    client.getInputStream()));
            boolean temp = true;
            while (temp) { // 要一直接收客户端的请求
                String str = buf.readLine(); // 接收内容
                if (str == null || "".equals(str)) {// 没有输入内容
                    temp = false;// 不再接收
                } else {
                    if ("bye".equals(str)) {
                        temp = false; // 不再接收
                    } else {
                        out.println("ECHO: " + str); // 将内容送回客户端
                    }
                }
            }
            out.close();
            buf.close();
            client.close();
        } catch (Exception e) {
        }
    }
}
```

每一个 Socket 的客户端都会创建一个线程的对象，那么，此时服务器端修改如下：

```
package org.tcpdemo.echothread;
import java.net.ServerSocket;
public class EchoServer {
    public static void main(String[] args) throws Exception {
        ServerSocket server = new ServerSocket(9999); // 在9999端口监听
        System.out.println("等待客户端连接。。。。。。。");
        boolean flag = true;
        while (flag) { // 服务器可以接收到多个用户请求
            // 得到客户端的输出流，准备输出
```

```
            new Thread(new ThreadEcho(server.accept())).start();
        }
        server.close();
    }
}
```

此时，此服务器就可以同时处理多个客户端的连接了。

所以说，Socket + Thread + IO 可以完成服务器的编写开发。

3.4、UDP 程序（理解）

UDP 属于不可靠的连接，使用数据报协议进行发送，主要使用以下的两个类：

- 数据报：DatagramPacket
- 数据报的 Socket：DatagramSocket

范例：编写服务器端

```
package org.udpdemo.echo;
import java.net.DatagramPacket;
import java.net.DatagramSocket;
import java.net.InetAddress;
public class HelloUDPServer {
    public static void main(String[] args) throws Exception {
        String info = "Hello World";// 要发送的信息
        // 将信息封装成数据报
        DatagramPacket dp = new DatagramPacket(info.getBytes(), 0, info
            .length(), InetAddress.getByName("localhost"), 6000);// 客户端在6000端口监听
        DatagramSocket server = new DatagramSocket(3000);// 服务器的端口
        server.send(dp);// 发送数据报
        server.close();
    }
}
```

此时，服务器端已经开发完成了，在服务器端指明了服务器的监听端口是 3000 端口，之后所有的数据报要向 6000 端口发送。

范例：编写客户端程序

```
package org.udpdemo.echo;
import java.net.DatagramPacket;
import java.net.DatagramSocket;
public class HelloUDPClient {
    public static void main(String[] args) throws Exception {
        byte b[] = new byte[1024];// 接收内容
        DatagramPacket dp = new DatagramPacket(b, b.length);// 接收内容
        DatagramSocket client = new DatagramSocket(6000);// 客户端在6000端口等待
        client.receive(dp); // 接收内容
        System.out.println(new String(dp.getData(), 0, dp.getLength()));
        client.close();
    }
}
```

以上表示接收服务器端发送来的信息。

但是执行的时候首先要运行客户端，之后再运行服务器端。

3.5、URL 与 URLConnection

URL：统一资源定位符（URL，英语 Uniform / Universal Resource Locator 的缩写）也被称为网页地址，是因特网上标准的资源的地址（Address），是用于完整地描述 Internet 上网页和其他资源的地址的一种标识方法

URL 的一般格式为(带方括号[]的为可选项)：

protocol :// hostname[:port] / path / [;parameters][?query]#fragment

URLConnection：

代表应用程序和 URL 之间的通信链接。此类的实例可用于读取和写入此 URL 引用的资源。

4、总结

1、 了解 Java 可以完成 TCP 和 UDP 两种程序的开发

2、 ServerSocket 和 Socket 可以完成 TCP 程序的开发

3、 掌握 URL、URLConnection 的使用

5、作业

1. 务必完成本节所有示例代码。