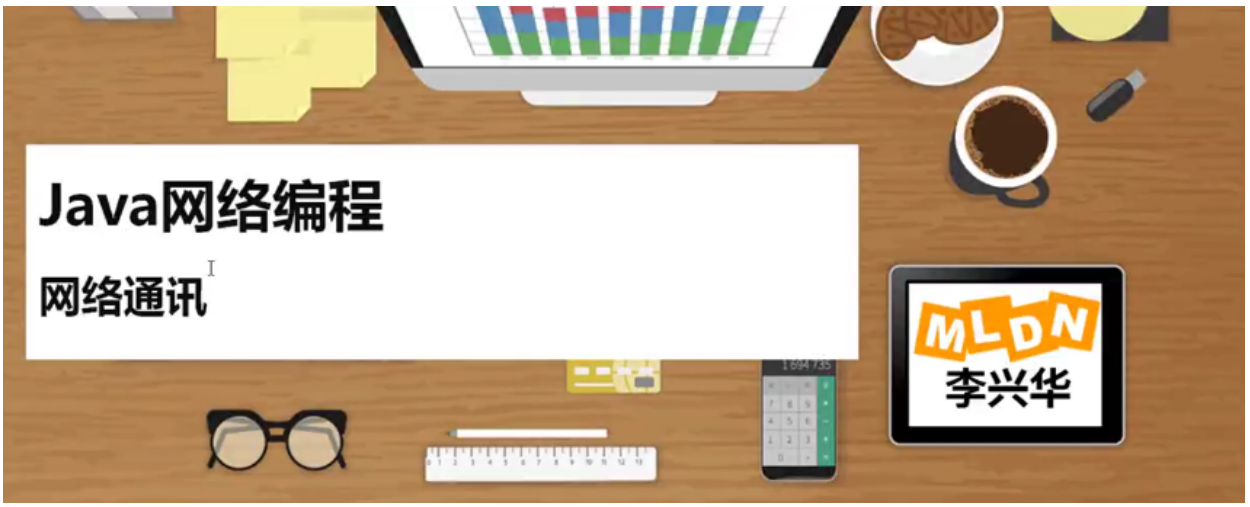


博客： https://www.cnblogs.com/HOsystem/p/14116443.html

2、具体内容

所谓的网络编程指的就是多台主机之间的数据通讯操作。

■网络编程简介

网络的核心定义在于：有两台以上的电脑就称为网络。实际上在世界上产生第一台电脑之后就有人去思考如何可以将更多的电脑生产出来并且将其进行有效的连接。

网络的连接的目的不仅仅是为了进行电脑的串联，更多的情况下是为了进行彼此之间的数据通信，包括现在所谓的网络游戏本质上还是网络通讯的问题，而在通讯的实现上就产生了一系列的处理协议：IP、TCP、UDP等等，也就是说所谓的网络编程实际上实现的就是一个数据的通讯操作而已，只不过这个通讯操作需要分为客户端与服务器端。

于是针对于网络程序的开发就有了两种模型：

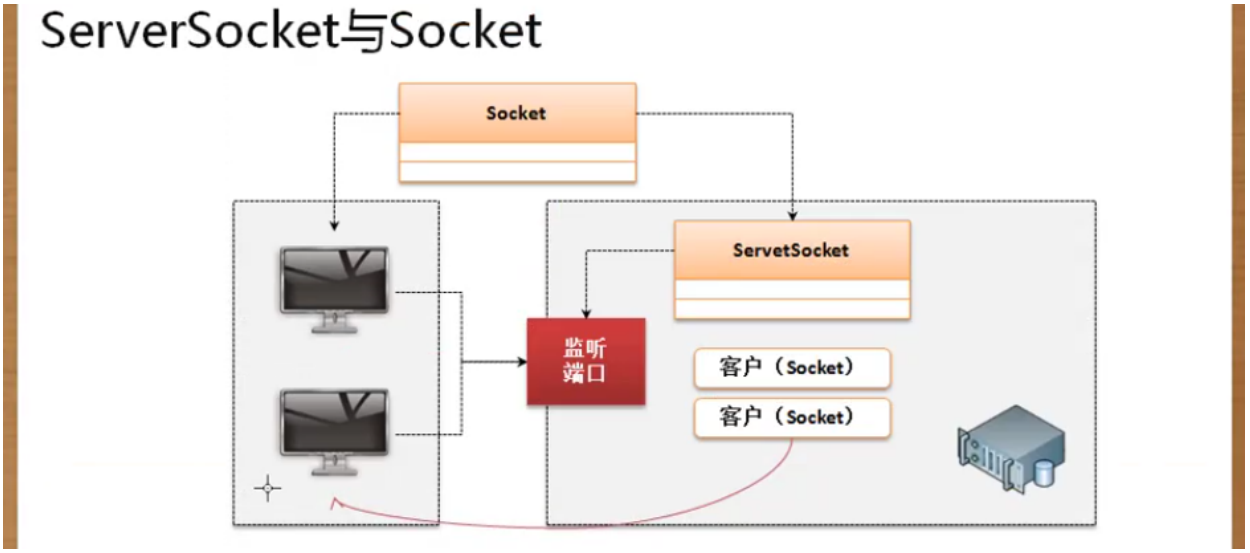
·**C/S(Client/Server、客户端与服务器端)**：要开发出两套程序，一套程序为客户端，另外一套程序为服务器端，如果现在服务器端发生了改变之后客户端也应该进行更新处理，这种开发可以由开发者自定义传输协议，并且使用一些比较私密的端口，所以安全性是比较高的，但是开发与维护成本比较高；

·**B/S(Browse/Server、浏览器与服务器端)**：只开发一套服务器端的程序，而后利用浏览器作为客户端进行访问，这种开发与维护的成本较低（只有一套程序），但是由于其使用的是公共的HTTP协议并且使用的公共80端口，所以其安全性相对较差，现在的开发基本上以“B/S”结构为主。

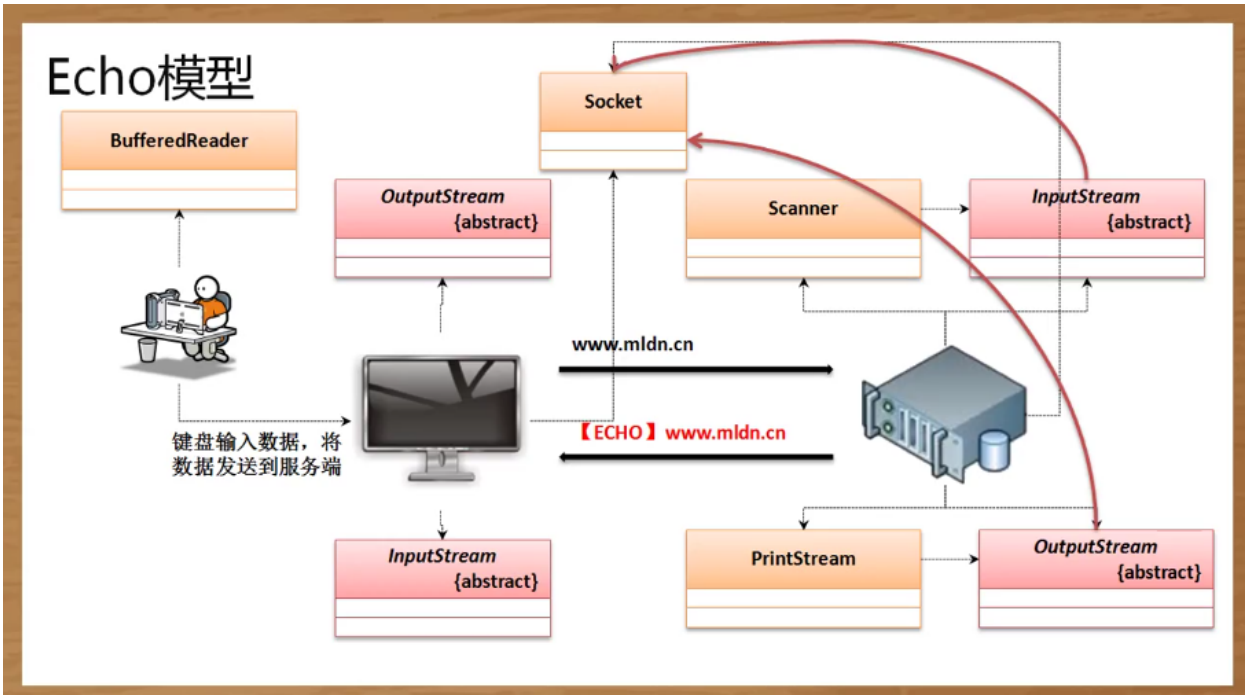
本次所要讲解的网络编程主要就是C/S程序模型，其分为两种开发：TCP(可靠的数据连接)、UDP(不可靠的数据连接)；

■TCP程序的基本实现

TCP的程序开发是网络程序的最基本的开发模型，其核心的特点是使用两个类实现数据的交互处理：ServerSocket(服务器端)、Socket(客户端)。



ServerSocket的主要目的是设置服务器的监听端口，而Socket需要指明要连接的服务器地址与端口。下面实现一个最简单的数据处理操作，即：Echo程序实现。

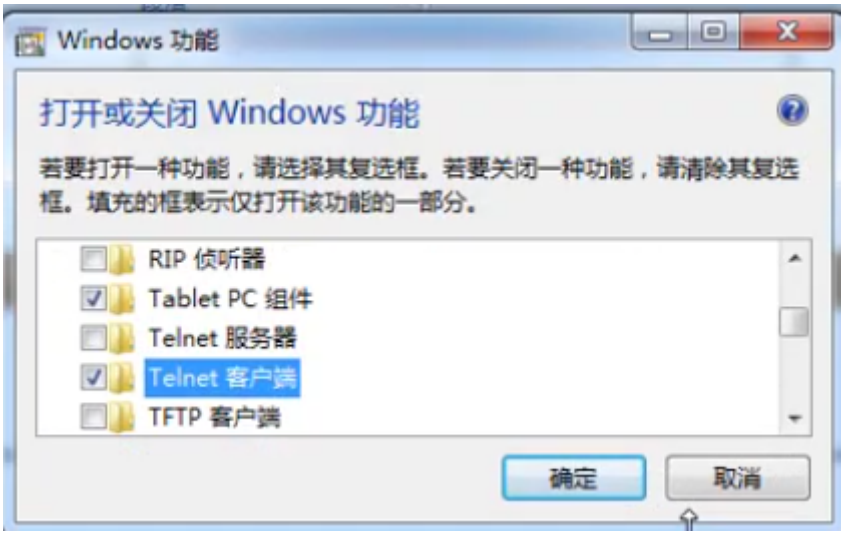


范例：实现服务器端的定义

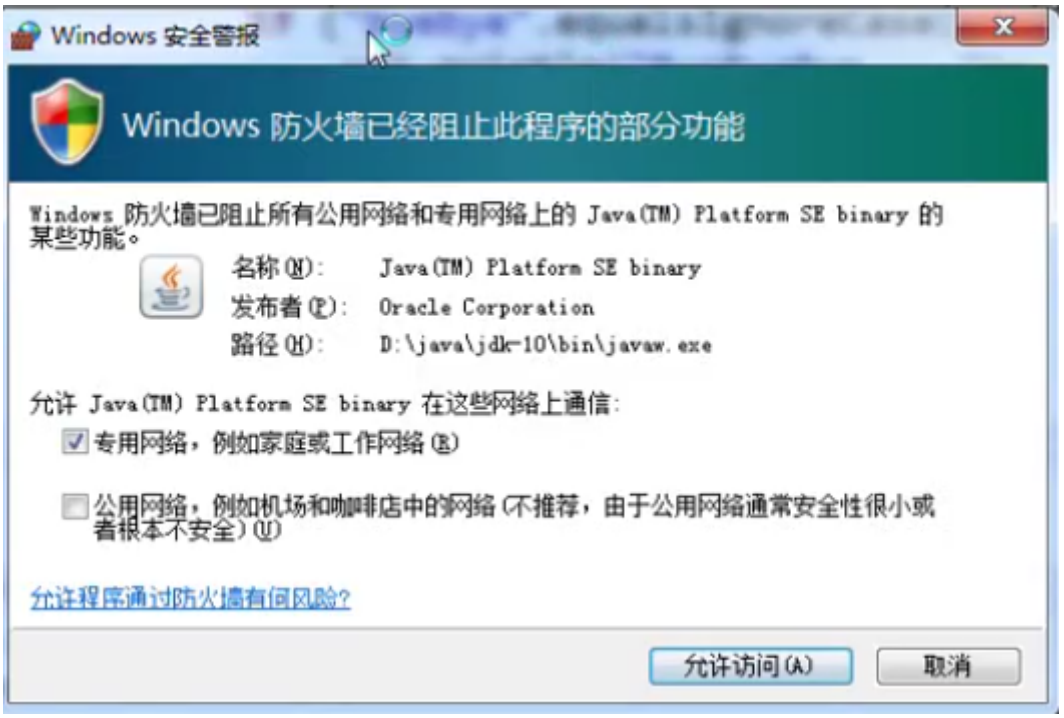
```
package cn.mldn.demo.server;
import java.io.PrintStream;
import java.net.ServerSocket;
import java.net.Socket;
import java.util.Scanner;
public class EchoServer {
	public static void main(String[] args) throws Exception {
		ServerSocket server = new ServerSocket(9999) ; // 设置服务器端的监听端口
		System.out.println("等待客户端连接.....");
```

```
            Socket client = server.accept() ; // 有客户端连接
            // 首先需要先接收客户端发送来的信息，而后才可以将信息处理之后发送回客户端
            Scanner scan = new Scanner(client.getInputStream()) ;        // 客户端输入流
            scan.useDelimiter("\n") ; // 设置分隔符
            PrintStream out = new PrintStream(client.getOutputStream()) ; // 客户端输出流
            boolean flag = true ; // 循环标记
            while (flag) {
                  if(scan.hasNext()) {  // 现在有数据发送
                        String val = scan.next().trim() ; // 接收发送的数据
                        if ("byebye".equalsIgnoreCase(val)) {
                              out.println("ByeByeBye....");
                              flag = false ; // 结束循环
                        } else {
                              out.println("【ECHO】" + val);
                        }
                  }
            }
            scan.close();
            out.close();
            client.close();
            server.close();
      }
}
```

如果此时需要对程序进行测试，最好的方法是直接使用telnet命令完成，但是此命令在Windows7之后已经变为了默认不开启的状态，所以如果要想使用则必须单独启用此命令。



在服务器端开启的情况下通过telnet指令输入：open localhost 9999



范例：实现客户端的定义

```
package cn.mldn.demo.client;
import java.io.BufferedReader;
import java.io.InputStreamReader;
import java.io.PrintStream;
import java.net.Socket;
import java.util.Scanner;
public class EchoClient {
	private static final BufferedReader KEYBOARD_INPUT = new BufferedReader(new
InputStreamReader(System.in)) ;
	public static void main(String[] args) throws Exception {
		Socket client = new Socket("localhost",9999) ; // 定义服务端的连接信息
		// 现在的客户端需要有输入与输出的操作支持，所以依然要准备出Scanner与PrintWriter
		Scanner scan = new Scanner(client.getInputStream()) ; // 接收服务器端的输入内容
		scan.useDelimiter("\n") ;
		PrintStream out = new PrintStream(client.getOutputStream()) ; // 向服务器端发送
内容
		boolean flag = true ;
		while(flag) {
			String input = getString("请输入要发送的内容：").trim() ;
			out.println(input); // 加换行
			if (scan.hasNext()) { // 服务器端有回应了
				System.out.println(scan.next());
			}
			if ("byebye".equalsIgnoreCase(input)) {
				flag = false ;
			}
		}
		scan.close();
		out.close();
		client.close();
	}
```

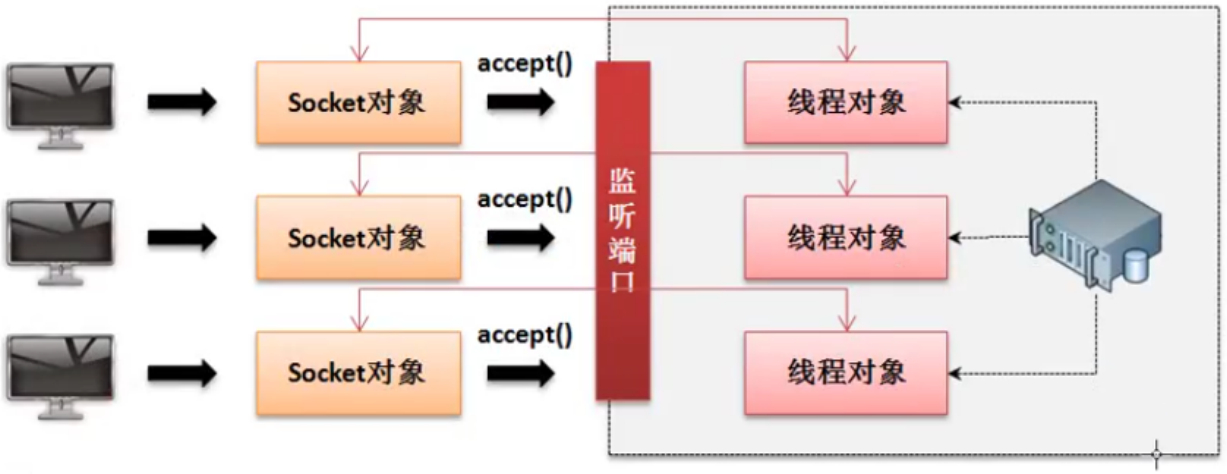
```
    public static String getString(String prompt) throws Exception {
        System.out.print(prompt);
        String str = KEYBOARD_INPUT.readLine() ;
        return str ;
    }
}
```

此时就实现了一个最基础的客户端与服务器端之间的数据通讯操作。

■多线程与网络编程

现在尽管已经实现了一个标准的网络程序开发，但是在整个的开发过程之中本程序有严重的性能缺陷，因为该服务器只能够为一个线程提供Ehco服务，如果说现在的服务器需要有多人进行连接访问的时候那么其它的使用者将无法连接(等待连接)。

所以现在就可以发现单线程的服务器开发本身就是一种不合理的做法，那么此时最好的解决方案将每一个连接到服务器上的客户端都通过一个线程对象来进行处理，即：服务器上启动多个线程，每一个线程单独为每一个客户端实现Echo服务支持。



范例：修改服务器端程序

```
package cn.mldn.demo.server;
import java.io.IOException;
import java.io.PrintStream;
import java.net.ServerSocket;
import java.net.Socket;
import java.util.Scanner;
public class EchoServer {
    private static class ClientThread implements Runnable {
        private Socket client = null ; // 描述每一个不同的客户端
        private Scanner scan = null ;
        private PrintStream out = null ;
        private boolean flag = true ; // 循环标记
        public ClientThread(Socket client) throws Exception {
            this.client = client ;
            this.scan = new Scanner(client.getInputStream()) ;       // 客户端输入流
            this.scan.useDelimiter("\n") ; // 设置分隔符
            this.out = new PrintStream(client.getOutputStream()) ; // 客户端输出流
```

```
        }
        @Override
        public void run() {
            while (this.flag) {
                if(scan.hasNext()) {  // 现在有数据发送
                    String val = scan.next().trim() ; // 接收发送的数据
                    if ("byebye".equalsIgnoreCase(val)) {
                        out.println("ByeByeBye....");
                        this.flag = false ; // 结束循环
                    } else {
                        out.println("【ECHO】" + val);
                    }
                }
            }
            try {
                scan.close();
                out.close();
                client.close();
            } catch (IOException e) {
                e.printStackTrace();
            }
        }
    }
    public static void main(String[] args) throws Exception {
        ServerSocket server = new ServerSocket(9999) ; // 设置服务器端的监听端口
        System.out.println("等待客户端连接.....");
        // 首先需要先接收客户端发送来的信息，而后才可以将信息处理之后发送回客户端
        boolean flag = true ; // 循环标记
        while (flag) {
            Socket client = server.accept() ; // 有客户端连接
            new Thread(new ClientThread(client)).start();
        }
        server.close();
    }
}
```

如果在这类的代码里面再追加一些集合的数据控制，实际上就可以实现一个80年代的聊天室。

■数据报发送与接收

之前所见到的都属于TCP程序开发范畴，TCP程序最大的特点是可靠的网络连接，但是在网络程序开发之中还存在有一种UDP程序，基于数据包的网络编程实现，如果要想实现UDP程序需要两个类：DatagramPacket(数据内容)、DatagramSocket(网络发送与接收)。数据报就好比发送的短信息一样，客户端是否接收到与发送者无关。

范例：实现一个UDP客户端

```
package cn.mldn.demo.client;
import java.net.DatagramPacket;
```

```
import java.net.DatagramSocket;
public class UDPClient {
	public static void main(String[] args) throws Exception { // 接收数据信息
		DatagramSocket client = new DatagramSocket(9999); // 连接到9999端口
		byte data[] = new byte[1024]; // 接收消息
		DatagramPacket packet = new DatagramPacket(data, data.length);// 接收数据
		System.out.println("客户端等待接收发送的消息.....");
		client.receive(packet); // 接收消息，所有的消息都在data字节数组之中
		System.out.println("接收到的消息内容为：" + new String(data, 0,
packet.getLength()));
		client.close();
	}
}
```

范例：实现UDP服务端

```
package cn.mldn.demo.server;
import java.net.DatagramPacket;
import java.net.DatagramSocket;
import java.net.InetAddress;
public class UDPServer {
	public static void main(String[] args) throws Exception {
		DatagramSocket server = new DatagramSocket(9000); // 连接到9999端口
		String str = "www.mldn.cn"; // 要发送的消息的内容
		DatagramPacket packet = new DatagramPacket(str.getBytes(), 0, str.length(),
InetAddress.getByName("localhost"),
				9999);// 发送数据
		server.send(packet); // 发送消息
		System.out.println("消息发送完毕.....");
		server.close();
	}
}
```

UDP发送的数据一定是不可靠的，但是TCP由于需要保证可靠的连接所以所需要的服务器资源就越多。