**3.1、网络编程（了解）**

网络编程指的就是通过网络进行程序数据操作，既然是网络开发，那么一定就分为用户和服务两端，而这两个端的开发实际上就有以下的两种不同的架构（**面试题：**请解释C/S和B/S的区别？）：

· C/S（Client / Server）：要开发两套程序，一套是服务器端，另外一套是与之对应的客户端，但是这种程序在日后进行维护的时候，是需要维护两套程序，而且客户端的程序更新也必须及时，此类程序安全；

· B/S（Browser / Server）：要开发一套程序，只开发服务器端的，客户端使用浏览器进行访问，这种程序在日后进行程序维护的时候只需要维护服务器端即可，客户端不需要做任何的修改，此类程序使用公共端口，包括公共协议，所以安全性很差。

如果从网络的开发而言，大的分类是以上的两类，可是从现在的开发来讲，更多的情况是针对于B/S程序进行的开发，或者可以这么理解：B/S程序的开发属于网络时代，而C/S程序的开发属于单机时代。而对于WebService的开发，实话而言，也属于B/S结构的程序（跨平台）。

而在日后学习Android开发的时候，如果要考虑安全性使用Socket，如果要考虑方便性，还是基于WEB的开发方便使用。而对于网络的开发在Java中也分为两种：TCP（传输控制协议，可靠的传输）、UDP（数据报协议），对于网络开发，本次只专注于TCP程序的实现。

**3.2、网络程序的基本实现（了解）**

如果要进行网络程序的开发，那么首先应该开发出的就是服务器端，本次的操作使用服务器端向客户端输出一个“Hello World.”的字符串信息，而如果要想完成服务器端的开发，则需要java.net包中的两个类：

· ServerSocket类：是一个封装支持TCP协议的操作类，主要工作在服务器端，用于接收客户端请求；

· Socket类：也是一个封装了TCP协议的操作类，每一个Socket对象都表示一个客户端。

而现在必须观察这两个类之中的操作方法：

· ServerSocket类的方法：

|  |  |  |  |
| --- | --- | --- | --- |
| **No.** | **方法名称** | **类型** | **描述** |
| 1 | public ServerSocket(int port) throws IOException | 构造 | 开辟一个指定的端口监听，一般使用5000以上 |
| 2 | public Socket accept() throws IOException | 普通 | 服务器端接收客户端请求，通过Socket返回 |
| 3 | public void close() throws IOException | 普通 | 关闭服务器端 |

· Socket类的方法：

|  |  |  |  |
| --- | --- | --- | --- |
| **No.** | **方法名称** | **类型** | **描述** |
| 1 | public Socket(String host, int port) throws UnknownHostException, IOException | 构造 | 指定要连接的主机（IP地址）和端口 |
| 2 | public OutputStream getOutputStream() throws IOException | 普通 | 取得指定客户端的输出对象，使用的时候肯定使用PrintStream装饰操作 |
| 3 | public InputStream getInputStream() throws IOException | 普通 | 从指定的客户端读取数据，使用Scanner操作 |

**范例：**完成一个服务器端程序代码

|  |
| --- |
| **package** cn.mldn.netdemo;  **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("服务开始启动...");  Socket client = server.accept(); // 接收客户端连接，进入到阻塞状态  PrintStream out = **new** PrintStream(client.getOutputStream());  out.println("Hello World ."); // 向客户端输出  out.close(); // 输出流的关闭  client.close(); // 关闭客户端  server.close(); // 关闭服务器端  System.*out*.println("服务器已关闭...");  }  } |

现在服务器端已经开发完成了，而现在的服务器端虽然是通过Java编写的，但是使用的是TCP协议，所以可以利用系统命令的telnet进行访问。

· 使用运行方式输入：telnet；

· 连接服务器：open ip地址 端口，open localhost 9999；

但是现在是利用了系统的工具完成的，那么在开发之中是不可能使用这工具，应该自己去编写客户端。

**范例：**编写一个客户端

|  |
| --- |
| **package** cn.mldn.netdemo;  **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);  Scanner scan = **new** Scanner(client.getInputStream());  scan.useDelimiter("\n") ;  **if** (scan.hasNext()) { // 有数据  System.*out*.println("服务器的回应数据：" + scan.next());  }  scan.close() ;  client.close() ;  }  } |

整个的操作就是一个输出和输入的过程，只是现在的输出和输入的来源在于网络，而不是像之前那样是通过文件操作了。

**3.3、网络开发的经典模型 —— ECHO程序**

这种开发模型的典型模式指的是客户端要输入信息，而后服务器端接收这端信息之后，前面增加一个“ECHO：”的信息再返回给客户端，也就是说现在的服务器端需要输入和输出数据。

**范例：**编写一个程序的基本模型

|  |
| --- |
| **package** cn.mldn.netdemo;  **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);  **boolean** flag = **true**;  System.*out*.println("服务器运行...");  Socket client = server.accept(); // 接收客户端请求  Scanner scan = **new** Scanner(client.getInputStream());  PrintStream out = **new** PrintStream(client.getOutputStream());  **while** (flag) {  **if** (scan.hasNext()) { // 有内容  String str = scan.next();  **if** ("byebye".equalsIgnoreCase(str.trim())) { // 程序结束  out.println("Bye Bye...");  flag = **false**; // 退出循环  }  out.println("ECHO：" + str.trim()); // 回应数据  }  }  System.*out*.println("服务器停止运行...");  server.close();  }  } |

这个时候程序实际上是一种单线程的运行状态，这样的程序运行起来只能够为一个用户进行服务，所以如果希望一个服务器可以同时处理多个客户的操作，那么就必须为其实现多线程的处理机制，让每一个客户端表示一个独立的线程对象，每个线程对象有自己独立的输入输出操作。

**范例：**为服务器端增加多线程机制，使用匿名内部类

|  |
| --- |
| **package** cn.mldn.netdemo;  **import** java.io.IOException;  **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);  **boolean** flag = **true**;  System.*out*.println("服务器运行...");  **while** (flag) {  **final** Socket client = server.accept(); // 接收客户端请求  **new** Thread(**new** Runnable() {  @Override  **public** **void** run() {  **boolean** runFlag = **true**;  **try** {  Scanner scan = **new** Scanner(client.getInputStream());  PrintStream out = **new** PrintStream(client.getOutputStream());  **while** (runFlag) {  **if** (scan.hasNext()) { // 有内容  String str = scan.next();  **if** ("byebye".equalsIgnoreCase(str.trim())) { // 程序结束  out.println("Bye Bye...");  runFlag = **false**; // 退出循环  }  out.println("ECHO：" + str.trim()); // 回应数据  }  }  } **catch** (IOException e) {  e.printStackTrace();  }  **try** {  client.close() ;  } **catch** (IOException e) {  e.printStackTrace();  }  }  }).start();  }  System.*out*.println("服务器停止运行...");  server.close();  }  } |

但是，在这个时候对于线程的控制也需要处理好，如果处理不好，可能就出现死锁问题了。

**4、总结**

知道什么叫网络编程就行了，暂时用不到。