← Java 序列化 Java 发送邮件 →

Java 网络编程

网络编程是指编写运行在多个设备（计算机）的程序，这些设备都通过网络连接起来。

java.net 包中 J2SE 的 API 包含有类和接口，它们提供低层次的通信细节。你可以直接使用这些类和接口，来专注于解决问题，而不用关注通信细节。

java.net 包中提供了两种常见的网络协议的支持：

TCP：TCP 是传输控制协议的缩写，它保障了两个应用程序之间的可靠通信。通常用于互联网协议，被称 TCP / IP。

UDP：UDP 是用户数据报协议的缩写，一个无连接的协议。提供了应用程序之间要发送的数据的数据包。

本教程主要讲解以下两个主题。

Socket 编程：这是使用最广泛的网络概念，它已被解释地非常详细。

URL 处理：这部分会在另外的篇幅里讲，点击这里更详细地了解在 Java 语言中的 URL 处理。

Socket 编程

套接字使用TCP提供了两台计算机之间的通信机制。 客户端程序创建一个套接字，并尝试连接服务器的套接字。

当连接建立时，服务器会创建一个 Socket 对象。客户端和服务器现在可以通过对 Socket 对象的写入和读取来进行通信。

java.net.Socket 类代表一个套接字，并且 java.net.ServerSocket 类为服务器程序提供了一种来监听客户端，并与他们建立连接的机制。

以下步骤在两台计算机之间使用套接字建立TCP连接时会出现：

服务器实例化一个 ServerSocket 对象，表示通过服务器上的端口通信。

服务器调用 ServerSocket 类的 accept() 方法，该方法将一直等待，直到客户端连接到服务器上给定的端口。

服务器正在等待时，一个客户端实例化一个 Socket 对象，指定服务器名称和端口号来请求连接。

HTML / CSS
JavaScript
服务端
数据库
移动端
XML 教程
ASP.NET
Web Service
开发工具
网站建设

Socket 类的构造函数试图将客户端连接到指定的服务器和端口号。如果通信被建立，则在客户端创建一个 Socket 对象能够与服务器进行通信。

在服务器端，accept() 方法返回服务器上一个新的 socket 引用，该 socket 连接到客户端的 socket。

连接建立后，通过使用 I/O 流在进行通信，每一个socket都有一个输出流和一个输入流，客户端的输出流连接到服务器端的输入流，而客户端的输入流连接到服务器端的输出流。

TCP 是一个双向的通信协议，因此数据可以通过两个数据流在同一时间发送.以下是一些类提供的一套完整的有用的方法来实现 socket。

ServerSocket 类的方法

服务器应用程序通过使用 java.net.ServerSocket 类以获取一个端口,并且侦听客户端请求。

ServerSocket 类有四个构造方法：

序号	方法描述
1	**public ServerSocket(int port) throws IOException** 创建绑定到特定端口的服务器套接字。
2	**public ServerSocket(int port, int backlog) throws IOException** 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。
3	**public ServerSocket(int port, int backlog, InetAddress address) throws IOException** 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。
4	**public ServerSocket() throws IOException** 创建非绑定服务器套接字。

创建非绑定服务器套接字。 如果 ServerSocket 构造方法没有抛出异常，就意味着你的应用程序已经成功绑定到指定的端口，并且侦听客户端请求。

这里有一些 ServerSocket 类的常用方法：

序号	方法描述
1	**public int getLocalPort()** 返回此套接字在其上侦听的端口。
2	**public Socket accept() throws IOException** 侦听并接受到此套接字的连接。
3	**public void setSoTimeout(int timeout)** 通过指定超时值启用/禁用 SO_TIMEOUT，以毫秒为单位。

4	**public void bind(SocketAddress host, int backlog)** 将 ServerSocket 绑定到特定地址（IP 地址和端口号）。

Socket 类的方法

java.net.Socket 类代表客户端和服务器都用来互相沟通的套接字。客户端要获取一个 Socket 对象通过实例化 ，而 服务器获得一个 Socket 对象则通过 accept() 方法的返回值。

Socket 类有五个构造方法.

序号	方法描述
1	**public Socket(String host, int port) throws UnknownHostException, IOException.** 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。
2	**public Socket(InetAddress host, int port) throws IOException** 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。
3	**public Socket(String host, int port, InetAddress localAddress, int localPort) throws IOException.** 创建一个套接字并将其连接到指定远程主机上的指定远程端口。
4	**public Socket(InetAddress host, int port, InetAddress localAddress, int localPort) throws IOException.** 创建一个套接字并将其连接到指定远程地址上的指定远程端口。
5	**public Socket()** 通过系统默认类型的 SocketImpl 创建未连接套接字

当 Socket 构造方法返回，并没有简单的实例化了一个 Socket 对象，它实际上会尝试连接到指定的服务器和端口。

下面列出了一些感兴趣的方法，注意客户端和服务器端都有一个 Socket 对象，所以无论客户端还是服务端都能够调用这些方法。

序号	方法描述
1	**public void connect(SocketAddress host, int timeout) throws IOException** 将此套接字连接到服务器，并指定一个超时值。
2	**public InetAddress getInetAddress()** 返回套接字连接的地址。
3	**public int getPort()** 返回此套接字连接到的远程端口。

4	**public int getLocalPort()** 返回此套接字绑定到的本地端口。
5	**public SocketAddress getRemoteSocketAddress()** 返回此套接字连接的端点的地址，如果未连接则返回 null。
6	**public InputStream getInputStream() throws IOException** 返回此套接字的输入流。
7	**public OutputStream getOutputStream() throws IOException** 返回此套接字的输出流。
8	**public void close() throws IOException** 关闭此套接字。

InetAddress 类的方法

这个类表示互联网协议(IP)地址。下面列出了 Socket 编程时比较有用的方法：

序号	方法描述
1	**static InetAddress getByAddress(byte[] addr)** 在给定原始 IP 地址的情况下，返回 InetAddress 对象。
2	**static InetAddress getByAddress(String host, byte[] addr)** 根据提供的主机名和 IP 地址创建 InetAddress。
3	**static InetAddress getByName(String host)** 在给定主机名的情况下确定主机的 IP 地址。
4	**String getHostAddress()** 返回 IP 地址字符串（以文本表现形式）。
5	**String getHostName()** 获取此 IP 地址的主机名。
6	**static InetAddress getLocalHost()** 返回本地主机。
7	**String toString()** 将此 IP 地址转换为 String。

Socket 客户端实例

如下的 GreetingClient 是一个客户端程序，该程序通过 socket 连接到服务器并发送一个请求，然后等待一个响应。

GreetingClient.java 文件代码：

```
// 文件名 GreetingClient.java

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

public class GreetingClient
{
   public static void main(String [] args)
   {
      String serverName = args[0];
      int port = Integer.parseInt(args[1]);
      try
      {
         System.out.println("连接到主机：" + server
Name + "，端口号：" + port);
         Socket client = new Socket(serverName, po
rt);
         System.out.println("远程主机地址：" + clie
nt.getRemoteSocketAddress());
         OutputStream outToServer = client.getOutp
utStream();
         DataOutputStream out = new DataOutputStre
am(outToServer);

         out.writeUTF("Hello from " + client.getLo
calSocketAddress());
         InputStream inFromServer = client.getInpu
tStream();
         DataInputStream in = new DataInputStream(
inFromServer);
         System.out.println("服务器响应：  " + in.re
adUTF());
         client.close();
      }catch(IOException e)
      {
         e.printStackTrace();
      }
   }
}
```

Socket 服务端实例

如下的GreetingServer 程序是一个服务器端应用程序，使用 Socket 来监听一个指定的端口。

GreetingServer.java 文件代码：

```
// 文件名 GreetingServer.java

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

public class GreetingServer extends Thread
{
   private ServerSocket serverSocket;

   public GreetingServer(int port) throws IOExcept
ion
   {
      serverSocket = new ServerSocket(port);
      serverSocket.setSoTimeout(10000);
   }

   public void run()
   {
      while(true)
      {
         try
         {
            System.out.println("等待远程连接，端口号
为: " + serverSocket.getLocalPort() + "...");
            Socket server = serverSocket.accept();
            System.out.println("远程主机地址: " + s
erver.getRemoteSocketAddress());
            DataInputStream in = new DataInputStre
am(server.getInputStream());
            System.out.println(in.readUTF());
            DataOutputStream out = new DataOutputS
tream(server.getOutputStream());
            out.writeUTF("谢谢连接我: " + server.ge
tLocalSocketAddress() + "\nGoodbye!");
            server.close();
         }catch(SocketTimeoutException s)
         {
            System.out.println("Socket timed out!"
);
            break;
         }catch(IOException e)
         {
            e.printStackTrace();
            break;
         }
      }
   }
   public static void main(String [] args)
```

```
    {
        int port = Integer.parseInt(args[0]);
        try
        {
            Thread t = new GreetingServer(port);
            t.run();
        }catch(IOException e)
        {
            e.printStackTrace();
        }
    }
}
```

编译以上两个 java 文件代码，并执行以下命令来启动服务，使用端口号为 6066：

```
$ javac GreetingServer.java
$ java GreetingServer 6066
等待远程连接，端口号为：6066...
```

新开一个命令窗口，执行以上命令来开启客户端：

```
$ javac GreetingClient.java
$ java GreetingClient localhost 6066
连接到主机：localhost ，端口号：6066
远程主机地址：localhost/127.0.0.1:6066
服务器响应： 谢谢连接我：/127.0.0.1:6066
Goodbye!
```

← Java 序列化 Java 发送邮件 →

· XML 实例
· Java 实例

· HTML 实体符号
· HTML 拾色器
· JSON 格式化工具

· Go Channel 详解
· lua 中求 table ...
· Java transient ...

关注微信