Java-网络-Socket编程

Socket编程

> 以下内容转载自菜鸟教程

- 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Socket连接，而 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。
  - 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，可以使用Socket对象内中的流进行通信。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

- `public ServerSocket(int port) throws IOException`：创建一个服务器Socket，并制定在port端口，将会在此监听连接。
- `public Socket accept() throws IOException`：开始监听连接。运行至此这一线程将阻塞，直到有客户端的Socket尝试连接至此。成功建立连接后，该方法返回一个Socket对象，表示与该客户端的连接。

Socket类的方法

- 连接远程主机：Socket一般在创建时于构造方法中完成连接。常用的构造方法如下：
  - `public Socket(String host, int port) throws UnknownHostException, IOException`

        - 连接到指定主机的指定端口号。这里的host可以是ip地址，也可以是域名。
    - `public Socket(InetAddress host, int port) throws IOException`
        - 连接到指定IP地址的指定端口。
- 传输数据：在连接建立后，Socket使用输入输出流来传输数据。
    - `public InputStream getInputStream() throws IOException`：返回该Socket的输入流，用于接收远程主机传输的内容。
    - `public OutputStream getOutputStream() throws IOException`：返回该Socket的输出流，用于向远程主机发送内容
    - 区分什么时候用输入，什么时候用输出：这里的输入输出，是相对“我方”的操作来说的，即“我”能够从中读到东西的时输入流，能够往里面写入东西的是输出流。因此输入对应接收，输出对应发送。
- 在使用时，常常在上述用输入输出流初始化Reader和Writer对象，便于读取数据。

```
BufferedReader reader = new BufferedReader(
        new InputStreamReader(socket.getInputStream(), "UTF-8"));
PrintWriter writer = new PrintWriter(
        new OutputStreamWriter(socket.getOutputStream(), "UTF-8"));
// 读写示例
String readMessage = null;
readMessage = reader.readLine()
System.out.println(readMessage);
writer.println("server recive : " + readMessage);
writer.flush();
```

也可以用DataInputStream和DataOutputStream的相关方法。

```
DataInputStream in = new DataInputStream(socket.getInputStream());
System.out.println(in.readUTF());
DataOutputStream out = new DataOutputStream(socket.getOutputStream());
out.writeUTF("Hello!");
```

InetAddress

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。

序号	方法描述
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。

Datagram编程

DatagramSocket(UDP)简单示例
服务端:

```
public class Server {
    public static void main(String[] args) {
        try {
            DatagramSocket server = new DatagramSocket(5060);
            DatagramPacket packet = new DatagramPacket(new byte[1024], 1024);
            server.receive(packet);
            System.out.println(packet.getAddress().getHostName() + "(" +
packet.getPort() + "):" + new String(packet.getData()));
            packet.setData("Hello Client".getBytes());
            packet.setPort(5070);
            packet.setAddress(InetAddress.getLocalHost());
            server.send(packet);
            server.close();
        } catch (IOException e) {
            e.printStackTrace();
        }
    }
}
```

客户端:

```
public class Client {
    public static void main(String[] args){
        try {
            DatagramSocket client = new DatagramSocket(5070);
            DatagramPacket packet = new DatagramPacket(new byte[1024],1024);
            packet.setPort(5060);
            packet.setAddress(InetAddress.getLocalHost());
            packet.setData("Hello Server".getBytes());
            client.send(packet);
            client.receive(packet);
```

```
            System.out.println(packet.getAddress().getHostName() + "(" +
packet.getPort() + "):" + new String(packet.getData()));
            client.close();
        } catch (IOException e) {
            e.printStackTrace();
        }
    }
}
```