

day23 【网络编程】

今日内容

- 网络编程三要素
- TCP通信
- 文件上传
- 模拟B/S

教学目标

- ☐ 能够说出TCP协议特点
- ☐ 能够说出TCP协议下两个常用类名称
- ☐ 能够编写TCP协议下字符串数据传输程序
- ☐ 能够理解TCP协议下文件上传案例
- ☐ 能够理解TCP协议下BS案例

第一章 网络编程入门

1.1 软件结构

- **C/S结构**：全称为Client/Server结构，是指客户端和服务端结构。常见程序有QQ、迅雷等软件。



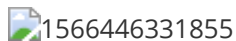
B/S结构：全称为Browser/Server结构，是指浏览器和服务端结构。常见浏览器有谷歌、火狐等。



两种架构各有优势，但是无论哪种架构，都离不开网络的支持。**网络编程**，就是在一定的协议下，实现两台计算机的通信的程序。

1.2 网络通信协议

- **网络通信协议**：通信协议是计算机必须遵守的规则，只有遵守这些规则，计算机之间才能进行通信。这就好比在道路中行驶的汽车一定要遵守交通规则一样，协议中对数据的传输格式、传输速率、传输步骤等做了统一规定，通信双方必须同时遵守，最终完成数据交换。
- **TCP/IP协议**：传输控制协议/因特网互联协议(Transmission Control Protocol/Internet Protocol)，是Internet最基本、最广泛的协议。它定义了计算机如何连入因特网，以及数据如何在它们之间传输的标准。它的内部包含一系列的用于处理数据通信的协议，并采用了4层的分层模型，每一层都呼叫它的下一层所提供的协议来完成自己的需求。



1.3 协议分类

通信的协议还是比较复杂的，`java.net` 包中包含的类和接口，它们提供低层次的通信细节。我们可以直接使用这些类和接口，来专注于网络程序开发，而不用考虑通信的细节。

`java.net` 包中提供了两种常见的网络协议的支持：

- **TCP**：传输控制协议 (Transmission Control Protocol)。TCP协议是**面向连接**的通信协议，即传输数据之前，在发送端和接收端建立逻辑连接，然后再传输数据，它提供了两台计算机之间可靠无差错的数据传输。
 - 三次握手：TCP协议中，在发送数据的准备阶段，客户端与服务器之间的三次交互，以保证连接的可靠。
 - 第一次握手，客户端向服务器端发出连接请求，等待服务器确认。服务器你死了吗？
 - 第二次握手，服务器端向客户端回送一个响应，通知客户端收到了连接请求。我活着啊！！
 - 第三次握手，客户端再次向服务器端发送确认信息，确认连接。整个交互过程如下图所示。我知道了！！



完成三次握手，连接建立后，客户端和服务器就可以开始进行数据传输了。由于这种面向连接的特性，TCP协议可以保证传输数据的安全，所以应用十分广泛，例如下载文件、浏览网页等。

- **UDP**：用户数据报协议 (User Datagram Protocol)。UDP协议是一个**面向无连接**的协议。传输数据时，不需要建立连接，不管对方端服务是否启动，直接将数据、数据源和目的地都封装在数据包中，直接发送。每个数据包的大小限制在64k以内。它是不可靠协议，因为无连接，所以传输速度快，但是容易丢失数据。日常应用中，例如视频会议、QQ聊天等。

1.4 网络编程三要素

协议

- **协议**：计算机网络通信必须遵守的规则，已经介绍过了，不再赘述。

IP地址

- **IP地址**：指互联网协议地址 (**Internet Protocol Address**)，俗称IP。IP地址用来给一个网络中的计算机设备做唯一的编号。假如我们把“个人电脑”比作“一台电话”的话，那么“IP地址”就相当于“电话号码”。

IP地址分类

- IPv4：是一个32位的二进制数，通常被分为4个字节，表示成 `a.b.c.d` 的形式，例如 `192.168.65.100`。其中a、b、c、d都是0~255之间的十进制整数，那么最多可以表示42亿个。
- IPv6：由于互联网的蓬勃发展，IP地址的需求量愈来愈大，但是网络地址资源有限，使得IP的分配越发紧张。有资料显示，全球IPv4地址在2011年2月分配完毕。

为了扩大地址空间，拟通过IPv6重新定义地址空间，采用128位地址长度，每16个字节一组，分成8组十六进制数，表示成 `ABCD:EF01:2345:6789:ABCD:EF01:2345:6789`，号称可以为全世界的每一粒沙子编上一个网址，这样就解决了网络地址资源数量不够的问题。

常用命令

- 查看本机IP地址，在控制台输入：

```
ipconfig
```

- 检查网络是否连通，在控制台输入：

```
ping 空格 IP地址
ping 220.181.57.216
ping www.baidu.com
```

特殊的IP地址

- 本机IP地址：127.0.0.1、localhost。

端口号

网络的通信，本质上是两个进程（应用程序）的通信。每台计算机都有很多的进程，那么在网络通信时，如何区分这些进程呢？

如果说IP地址可以唯一标识网络中的设备，那么端口号就可以唯一标识设备中的进程（应用程序）了。

- **端口号：用两个字节表示的整数，它的取值范围是0~65535。**其中，0~1023之间的端口号用于一些知名的网络服务和应用，普通的应用程序需要使用1024以上的端口号。如果端口号被另外一个服务或应用所占用，会导致当前程序启动失败。

利用 协议 + IP地址 + 端口号 三元组合，就可以标识网络中的进程了，那么进程间的通信就可以利用这个标识与其它进程进行交互。

在Java中，可以使用java.net.InetAddress类来表示一个IP地址：

```
/**
 * InetAddress类概述
 * 一个该类的对象就代表一个IP地址对象。
 *
 * InetAddress类成员方法
 * static InetAddress getLocalHost()
 *     获得本地主机IP地址对象
 * static InetAddress getByName(String host)
 *     根据IP地址字符串或主机名获得对应的IP地址对象
 *
 * String getHostName();获得主机名
 * String getHostAddress();获得IP地址字符串
 */
public class InetAddressDemo01 {
    public static void main(String[] args) throws Exception {
        // 获得本地主机IP地址对象
        InetAddress inet01 = InetAddress.getLocalHost();
        // pkxingdeMacBook-Pro.local/10.211.55.2
        // 主机名/ip地址字符串
        System.out.println(inet01);
        // 根据IP地址字符串或主机名获得对应的IP地址对象
        // InetAddress inet02 = InetAddress.getByName("192.168.73.97");
        InetAddress inet02 = InetAddress.getByName("baidu.com");
        System.out.println(inet02);

        // 获得主机名
        String hostName = inet01.getHostName();
        System.out.println(hostName);
        // 获得IP地址字符串
        String hostAddress = inet01.getHostAddress();
        System.out.println(hostAddress);
    }
}
```

```
        System.out.println(hostAddress);
    }
}
```

第二章 TCP通信程序

2.1 TCP协议概述

- TCP协议是面向连接的通信协议，即在传输数据前先在客户端和服务端建立逻辑连接，然后再传输数据。它提供了两台计算机之间可靠无差错的数据传输。TCP通信过程如下图所示：



TCP ==> Transfer Control Protocol ==> 传输控制协议

TCP协议的特点

- * 面向连接的协议
- * 只能由客户端主动发送数据给服务器端，服务器端接收到数据之后，可以给客户端响应数据。
- * 通过三次握手建立连接，连接成功形成数据传输通道。
- * 通过四次挥手断开连接
- * 基于IO流进行数据传输
- * 传输数据大小没有限制
- * 因为面向连接的协议，速度慢，但是是可靠的协议。

TCP协议的使用场景

- * 文件上传和下载
- * 邮件发送和接收
- * 远程登录

TCP协议相关的类

- * **Socket**
 - * 一个该类的对象就代表一个客户端程序。
- * **ServerSocket**
 - * 一个该类的对象就代表一个服务器端程序。

Socket类构造方法

- * **socket(String host, int port)**
 - * 根据ip地址字符串和端口号创建客户端Socket对象
 - * 注意事项：只要执行该方法，就会立即连接指定的服务器程序，如果连接不成功，则会抛出异常。

如果连接成功，则表示三次握手通过。

Socket类常用方法

- * **OutputStream getOutputStream()**; 获得字节输出流对象
- * **InputStream getInputStream()**; 获得字节输入流对象

2.2 TCP通信案例

2.2.1 客户端向服务器发送数据

```
/*
```

TCP客户端代码实现步骤

- * 创建客户端Socket对象并指定服务器地址和端口号
- * 调用Socket对象的getOutputStream方法获得字节输出流对象
- * 调用字节输出流对象的write方法往服务器端输出数据
- * 调用Socket对象的getInputStream方法获得字节输入流对象
- * 调用字节输入流对象的read方法读取服务器端返回的数据
- * 关闭Socket对象断开连接。

```
*/  
// TCP客户端代码实现  
public class TCPClient {  
    public static void main(String[] args) throws Exception{  
        // 要发送的内容  
        String content = "你好TCP服务器端，约吗";  
        // 创建Socket对象  
        Socket socket = new Socket("192.168.73.99",9999);  
        // System.out.println(socket);  
        // 获得字节输出流对象  
        OutputStream out = socket.getOutputStream();  
        // 输出数据到服务器端  
        out.write(content.getBytes());  
  
        // 获得字节输入流对象  
        InputStream in = socket.getInputStream();  
        // 创建字节数组：用来存储服务器端发送来的数据  
        byte[] buf = new byte[1024];  
        // 读取服务器端返回的数据  
        int len = in.read(buf);  
        System.out.println("len = " + len);  
        System.out.println("服务器端返回的内容 = " + new String(buf,0,len));  
  
        // 关闭socket对象  
        socket.close();  
    }  
}
```

2.2.2 服务器向客户端回写数据

```
/**  
    TCP服务器端代码实现  
  
    ServerSocket类构造方法  
        * ServerSocket(int port) 根据指定的端口号开启服务器。  
  
    ServerSocket类常用方法  
        * Socket accept() 等待客户端连接并获得与客户端关联的Socket对象  
  
    TCP服务器端代码实现步骤  
        * 创建ServerSocket对象并指定端口号(相当于开启了一个服务器)  
        * 调用ServerSocket对象的accept方法等待客户端连接并获得对应Socket对象  
        * 调用Socket对象的getInputStream方法获得字节输入流对象  
        * 调用字节输入流对象的read方法读取客户端发送的数据  
        * 调用Socket对象的getOutputStream方法获得字节输出流对象  
        * 调用字节输出流对象的write方法往客户端输出数据  
        * 关闭Socket和ServerSocket对象  
  
*/  
public class TCPServer {  
    public static void main(String[] args) throws Exception{
```

```

// 创建服务器socket对象
ServerSocket serverSocket = new ServerSocket(9999);
// 等待客户端连接并获得与客户端关联的Socket对象
Socket socket = serverSocket.accept();
// 获得字节输入流对象
InputStream in = socket.getInputStream();
// 创建字节数组：用来存储读取到客户端发送的数据
byte[] buf = new byte[1024];
// 读取客户端发送过来的数据
int len = in.read(buf);
System.out.println("len = " + len);
System.out.println("客户端发送的数据 = " + new String(buf,0,len));

// 获得字节输出流对象
OutputStream out = socket.getOutputStream();
// 往客户端输出数据
out.write("约你妹".getBytes());

// 关闭socket
socket.close();
// 关闭服务器（在实际开发中，服务器一般不会关闭）
serverSocket.close();
}
}


```

第三章 综合案例

3.1 文件上传案例

文件上传分析图解

1. 【客户端】输入流，从硬盘读取文件数据到程序中。
2. 【客户端】输出流，写出文件数据到服务端。
3. 【服务端】输入流，读取文件数据到服务端程序。
4. 【服务端】输出流，写出文件数据到服务器硬盘中。
5. 【服务端】获取输出流，回写数据。
6. 【客户端】获取输入流，解析回写数据。

1566446548503

案例实现

服务器端实现：

```

public class FileUpload_Server {
    public static void main(String[] args) throws IOException {
        System.out.println("服务器 启动..... ");
        // 1. 创建服务端ServerSocket
        ServerSocket serverSocket = new ServerSocket(6666);
        // 2. 循环接收,建立连接
        while (true) {
            Socket accept = serverSocket.accept();
            /*

```

3. socket对象交给子线程处理,进行读写操作

Runnable接口中,只有一个run方法,使用lambda表达式简化格式

```
*/
new Thread(() -> {
    try (
        //3.1 获取输入流对象
        BufferedInputStream bis = new
BufferedInputStream(accept.getInputStream());
        //3.2 创建输出流对象, 保存到本地 .
        FileOutputStream fis = new
FileOutputStream(System.currentTimeMillis() + ".jpg");
        BufferedOutputStream bos = new BufferedOutputStream(fis);
    ) {
        // 3.3 读写数据
        byte[] b = new byte[1024 * 8];
        int len;
        while ((len = bis.read(b)) != -1) {
            bos.write(b, 0, len);
        }

        // 4.=====信息回写=====
        System.out.println("back .....");
        OutputStream out = accept.getOutputStream();
        out.write("上传成功".getBytes());
        out.close();
        //=====

        //5. 关闭 资源
        bos.close();
        bis.close();
        accept.close();
        System.out.println("文件上传已保存");
    } catch (IOException e) {
        e.printStackTrace();
    }
}).start();
}
}
```

客户端实现:

```
public class FileUpload_Client {
    public static void main(String[] args) throws IOException {
        // 1.创建流对象
        // 1.1 创建输入流,读取本地文件
        BufferedInputStream bis = new BufferedInputStream(new
FileInputStream("test.jpg"));
        // 1.2 创建输出流,写到服务端
        Socket socket = new Socket("localhost", 6666);
        BufferedOutputStream bos = new
BufferedOutputStream(socket.getOutputStream());

        //2.写出数据.
        byte[] b = new byte[1024 * 8 ];
        int len ;
        while (( len = bis.read(b))!=-1) {
```

```

        bos.write(b, 0, len);
        bos.flush();//这里必须实时刷新
    }
    // 关闭输出流,通知服务端,写出数据完毕
    socket.shutdownOutput();
    System.out.println("文件发送完毕");
    // 3. =====解析回写=====
    InputStream in = socket.getInputStream();
    byte[] back = new byte[20];
    in.read(back);
    System.out.println(new String(back));
    in.close();
    // =====
    // 4.释放资源
    socket.close();
    bis.close();
}
}

```

3.2 模拟B/S服务器

模拟网站服务器，使用浏览器访问自己编写的服务端程序，查看网页效果。

案例分析

1. 准备页面数据，web文件夹。
2. 我们模拟服务器端，ServerSocket类监听端口，使用浏览器访问，查看网页效果

案例实现

浏览器工作原理是遇到图片会开启一个线程进行单独的访问,因此在服务器端加入线程技术。

```

public class ServerDemo {
    public static void main(String[] args) throws IOException {
        ServerSocket server = new ServerSocket(8888);
        while(true){
            Socket socket = server.accept();
            new Thread(new Web(socket)).start();
        }
    }
}

```

```

class Web implements Runnable{
    private Socket socket;

    public Web(Socket socket){
        this.socket=socket;
    }

    public void run() {
        try{
            //转换流,读取浏览器请求第一行
            BufferedReader readWb = new

```



```

        BufferedReader(new
InputStreamReader(socket.getInputStream()));
        String request = readWb.readLine();
        //取出请求资源的路径
        String[] strArr = request.split(" ");
        System.out.println(Arrays.toString(strArr));
        String path = strArr[1].substring(1);
        System.out.println(path);

        FileInputStream fis = new FileInputStream(path);
        System.out.println(fis);
        byte[] bytes= new byte[1024];
        int len = 0 ;

        //向浏览器 回写数据
        OutputStream out = socket.getOutputStream();
        out.write("HTTP/1.1 200 OK\r\n".getBytes());
        out.write("Content-Type:text/html\r\n".getBytes());
        out.write("\r\n".getBytes());
        while((len = fis.read(bytes))!=-1){
            out.write(bytes,0,len);
        }
        fis.close();
        out.close();
        readWb.close();
        socket.close();
    }catch(Exception ex){


    }
}
}

```

访问效果:

1566446578300

图解:

1566446643154