

**day11【网络编程】**

**主要内容**

软件架构CS／BS

网络通信三要素

TCP通信

Socket套接字

ServerSocket

**教学目标**

能够辨别UDP和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

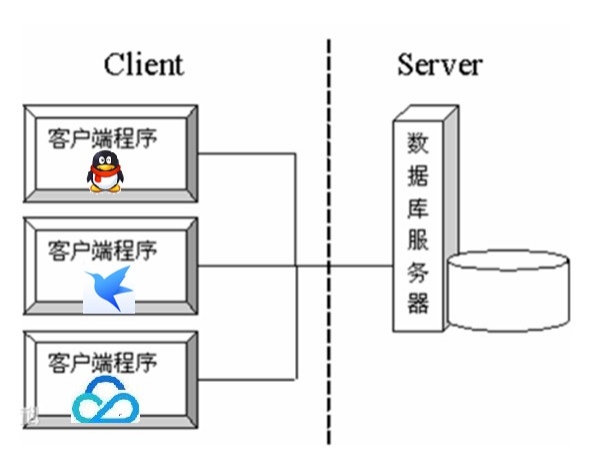
能够理解TCP协议下文件上传案例

能够理解TCP协议下案例2

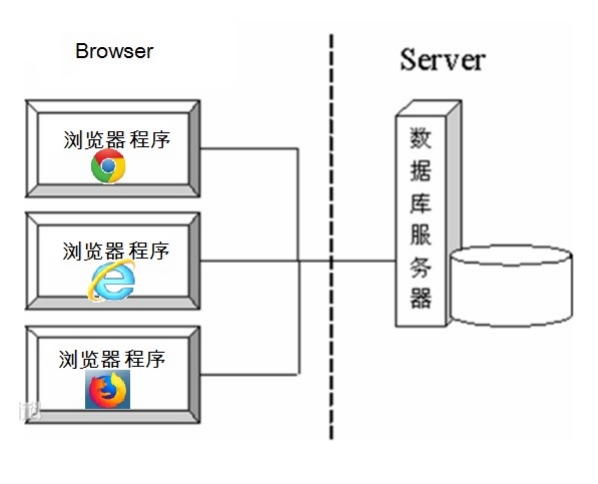
**第一章 网络编程入门**

**1.1软件结构**

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



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

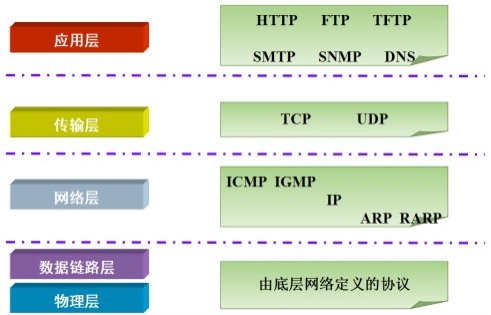


两种架构各有优势，但是无论哪种架构，都离不开网络的支持。**网络编程**，就是在一定的协议下，实现两台计算机 的通信的程序。

**1.2 网络通信协议**

**网络通信协议：**通信协议是对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：

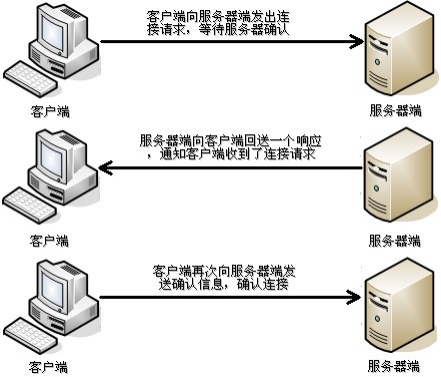
**TCP**：传输控制协议 (Transmission Control Protocol)。TCP协议是**面向连接**的通信协议，即传输数据之前， 在发送端和接收端建立逻辑连接，然后再传输数据，它提供了两台计算机之间可靠无差错的数据传输。

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。

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。

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。



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**



 ipconfig



 ping 空格 IP地址 ping 220.181.57.216



查看本机IP地址，在控制台输入：

检查网络是否连通，在控制台输入：

**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。

**第二章 TCP通信程序**

**2.1 概述**

TCP通信能实现两台计算机之间的数据交互，通信的两端，要严格区分为客户端（Client）与服务端（Server）。 **两端通信时步骤：**

1. 服务端程序，需要事先启动，等待客户端的连接。
2. 客户端主动连接服务器端，连接成功才能通信。服务端不可以主动连接客户端。

**在Java中，提供了两个类用于实现TCP通信程序：**

1. 客户端： java.net.Socket 类表示。创建 Socket 对象，向服务端发出连接请求，服务端响应请求，两者建 立连接开始通信。
2. 服务端： java.net.ServerSocket 类表示。创建 ServerSocket 对象，相当于开启一个服务，并等待客户端 的连接。

**2.2 Socket类**

Socket 类：该类实现客户端套接字，套接字指的是两台设备之间通讯的端点。

**构造方法**



 Socket client = new Socket("127.0.0.1" , 6666);



 ServerSocket server = new ServerSocket(6666);



public Socket(String host, int port) :创建套接字对象并将其连接到指定主机上的指定端口号。如果指 定的host是null ，则相当于指定地址为回送地址。

小贴士：回送地址(127.x.x.x) 是本机回送地址（Loopback Address），主要用于网络软件测试以及本 地机进程间通信，无论什么程序，一旦使用回送地址发送数据，立即返回，不进行任何网络传输。

构造举例，代码如下：

**成员方法**

public InputStream getInputStream() ： 返回此套接字的输入流。

如果此Scoket具有相关联的通道，则生成的InputStream 的所有操作也关联该通道。

关闭生成的InputStream也将关闭相关的Socket。

public OutputStream getOutputStream() ： 返回此套接字的输出流。

如果此Scoket具有相关联的通道，则生成的OutputStream 的所有操作也关联该通道。

关闭生成的OutputStream也将关闭相关的Socket。

public void close() ：关闭此套接字。

一旦一个socket被关闭，它不可再使用。

关闭此socket也将关闭相关的InputStream和OutputStream 。

public void shutdownOutput() ： 禁用此套接字的输出流。

任何先前写出的数据将被发送，随后终止输出流。

**2.3 ServerSocket类**

ServerSocket 类：这个类实现了服务器套接字，该对象等待通过网络的请求。

**构造方法**

public ServerSocket(int port) ：使用该构造方法在创建ServerSocket 对象时，就可以将其绑定到一个指 定的端口号上，参数port就是端口号。

构造举例，代码如下：

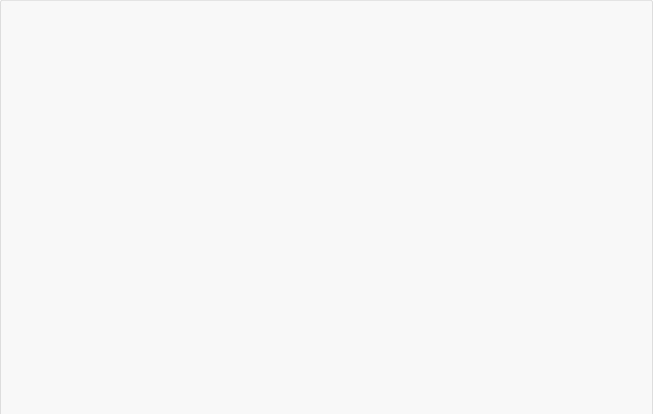
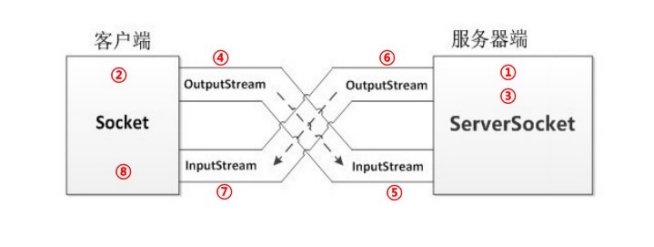
**成员方法**

public Socket accept() ：侦听并接受连接，返回一个新的Socket对象，用于和客户端实现通信。该方法 会一直阻塞直到建立连接。

**2.4 简单的TCP网络程序**

**TCP通信分析图解**

1. 【服务端】启动,创建ServerSocket 对象，等待连接。



1. 【客户端】启动,创建Socket对象，请求连接。
2. 【服务端】接收连接,调用accept方法，并返回一个Socket对象。
3. 【客户端】Socket对象，获取OutputStream，向服务端写出数据。 5. 【服务端】Scoket对象，获取InputStream，读取客户端发送的数据。

到此，客户端向服务端发送数据成功。

自此，服务端向客户端回写数据。

6. 【服务端】Socket对象，获取OutputStream，向客户端回写数据。 7. 【客户端】Scoket对象，获取InputStream，解析回写数据。

8. 【客户端】释放资源，断开连接。

**客户端向服务器发送数据**

**服务端实现：**

 public class ServerTCP {

    public static void main(String[] args) throws IOException  {         System.out.println ("服务端启动 ,等待连接 .... ");         // 1.创建 ServerSocket对象，绑定端口，开始等待连接         ServerSocket ss = new ServerSocket(6666);

        // 2.接收连接 accept 方法, 返回 socket 对象.

        Socket server = ss.accept();

        // 3.通过socket 获取输入流

        InputStream is = server.getInputStream ();

        // 4.一次性读取数据

        // 4.1 创建字节数组

        byte[] b = new byte[1024];

        // 4.2 据读取到字节数组中.

        int len = is.read(b)；

        // 4.3 解析数组,打印字符串信息

        String msg = new String(b, 0, len);

        System.out.println (msg);

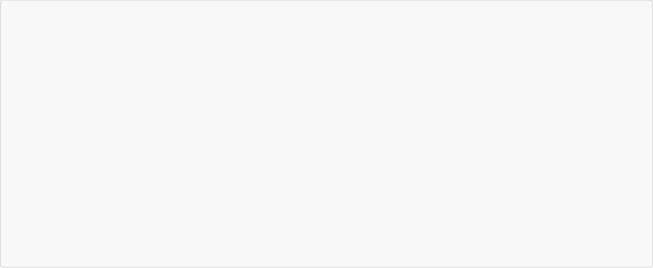
        //5.关闭资源.

        is.close();

        server.close();

    }

}



 public class ClientTCP {

    public static void main(String[] args) throws Exception {         System.out.println("客户端 发送数据");

        // 1.创建 Socket ( ip , port ) , 确定连接到哪里.         Socket client = new Socket("localhost" , 6666);         // 2.获取流对象 .输出流

        OutputStream os = client.getOutputStream ();         // 3.写出数据.

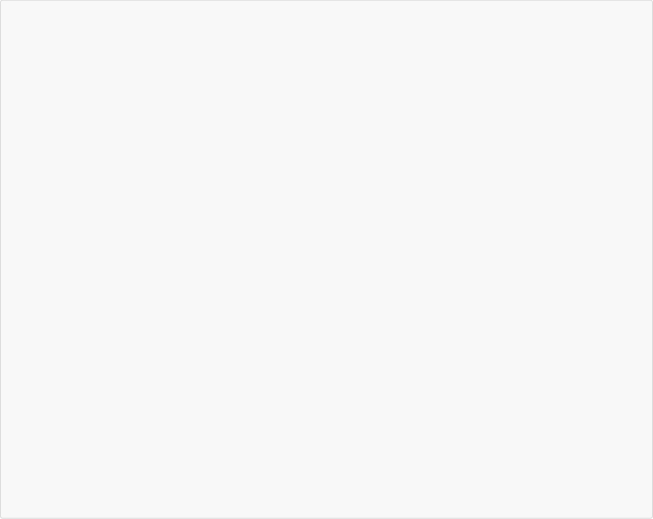
        os.write("你好么? tcp , 我来了".getBytes ());         // 4. 关闭资源 .

        os.close();

        client.close();

    }

}



**客户端实现：**

**服务器向客户端回写数据**

**服务端实现：**

 public class ServerTCP {

    public static void main(String[] args) throws IOException  {         System.out.println ("服务端启动 ,等待连接 .... ");         // 1.创建 ServerSocket对象，绑定端口，开始等待连接         ServerSocket ss = new ServerSocket(6666);

        // 2.接收连接 accept 方法, 返回 socket 对象.

        Socket server = ss.accept();

        // 3.通过socket 获取输入流

        InputStream is = server.getInputStream ();

        // 4.一次性读取数据

        // 4.1 创建字节数组

        byte[] b = new byte[1024];

        // 4.2 据读取到字节数组中.

        int len = is.read(b)；

        // 4.3 解析数组,打印字符串信息

        String msg = new String(b, 0, len);

        System.out.println (msg);

        // =================回写数据=======================         // 5. 通过 socket 获取输出流

         OutputStream out = server.getOutputStream ();

        // 6. 回写数据

         out.write("我很好,谢谢你".getBytes ());

        // 7.关闭资源.

        out.close();

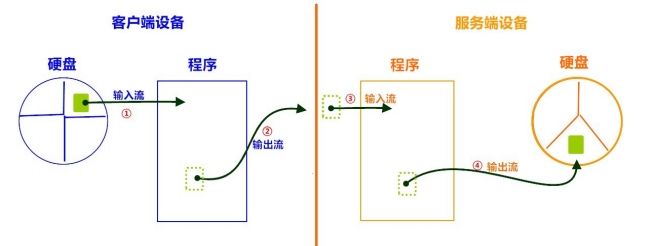
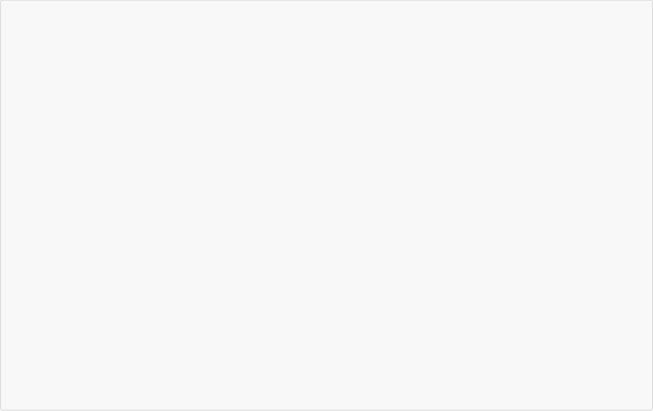
        is.close();

        server.close();

    }

}

**客户端实现：**



 public class ClientTCP {

    public static void main(String[] args) throws Exception {         System.out.println("客户端 发送数据");

        // 1.创建 Socket ( ip , port ) , 确定连接到哪里.         Socket client = new Socket("localhost" , 6666);         // 2.通过Scoket,获取输出流对象

        OutputStream os = client.getOutputStream ();         // 3.写出数据.

        os.write("你好么? tcp , 我来了".getBytes ());

        // ==============解析回写=========================         // 4. 通过Scoket,获取 输入流对象

        InputStream in = client.getInputStream ();

        // 5. 读取数据数据

        byte[] b = new byte[100];

        int len = in.read(b);

        System.out.println (new String(b, 0, len));

        // 6. 关闭资源 .

        in.close();

        os.close();

        client.close();

    }

}

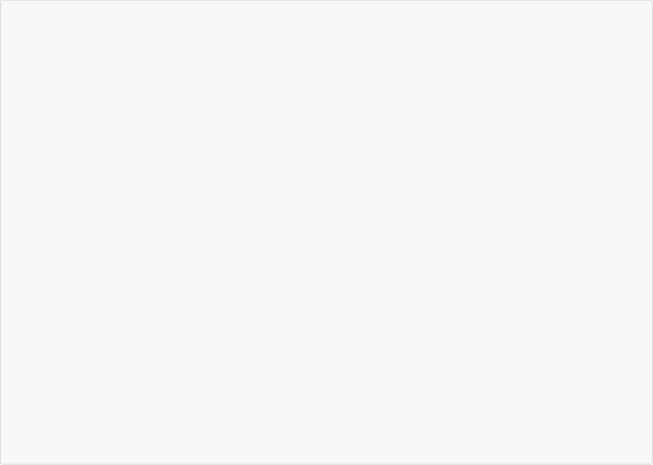
**第三章 综合案例**

**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。

**基本实现**



**服务端实现：**

 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. 建立连接

        Socket accept = serverSocket .accept();

        // 3. 创建流对象

        // 3.1 获取输入流,读取文件数据

        BufferedInputStream bis = new BufferedInputStream(accept.getInputStream());

        // 3.2 创建输出流,保存到本地 .

        BufferedOutputStream bos = new BufferedOutputStream(new FileOutputStream("copy.jpg" ));         // 4. 读写数据

        byte[] b = new byte[1024 \* 8];

        int len;

        while ((len = bis.read(b)) != ‐1) {

            bos.write(b, 0, len);

        }

        //5. 关闭 资源

        bos.close();

        bis.close();

        accept.close();

        System.out.println ("文件上传已保存");

    }

}

**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();

        }

        System.out.println ("文件发送完毕");

        // 3.释放资源

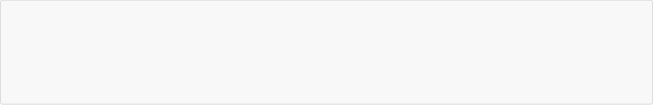
        bos.close();

        socket.close();

        bis.close();



 FileOutputStream fis = new FileOutputStream(System .currentTimeMillis()+".jpg") // 文件名称 BufferedOutputStream bos = new BufferedOutputStream(fis);



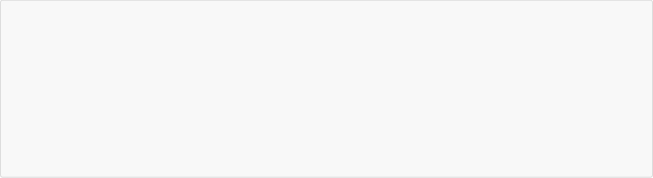
 // 每次接收新的连接,创建一个Socket

while（true）{

    Socket accept = serverSocket .accept();

    ......

}



 while（true）{

    Socket accept = serverSocket .accept();

    // accept 交给子线程处理.

    new Thread(() ‐> {

        ......

        InputStream bis = accept.getInputStream ();

        ......

    }).start();

}



        System.out.println ("文件上传完毕 ");

    }

}

**文件上传优化分析**

1. **文件名称写死的问题**

服务端，保存文件的名称如果写死，那么最终导致服务器硬盘，只会保留一个文件，建议使用系统时间优 化，保证文件名称唯一，代码如下：

2. **循环接收的问题**

服务端，指保存一个文件就关闭了，之后的用户无法再上传，这是不符合实际的，使用循环改进，可以不断 的接收不同用户的文件，代码如下：

3. **效率问题**

服务端，在接收大文件时，可能耗费几秒钟的时间，此时不能接收其他用户上传，所以，使用多线程技术优 化，代码如下：

**优化实现**

 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

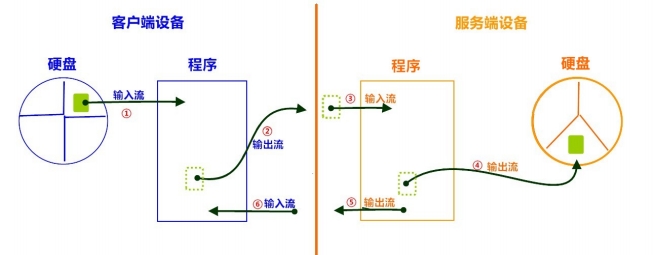
        ServerSocket serverSocket = new ServerSocket(6666);         // 2. 循环接收,建立连接

        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. 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();

        }

    }

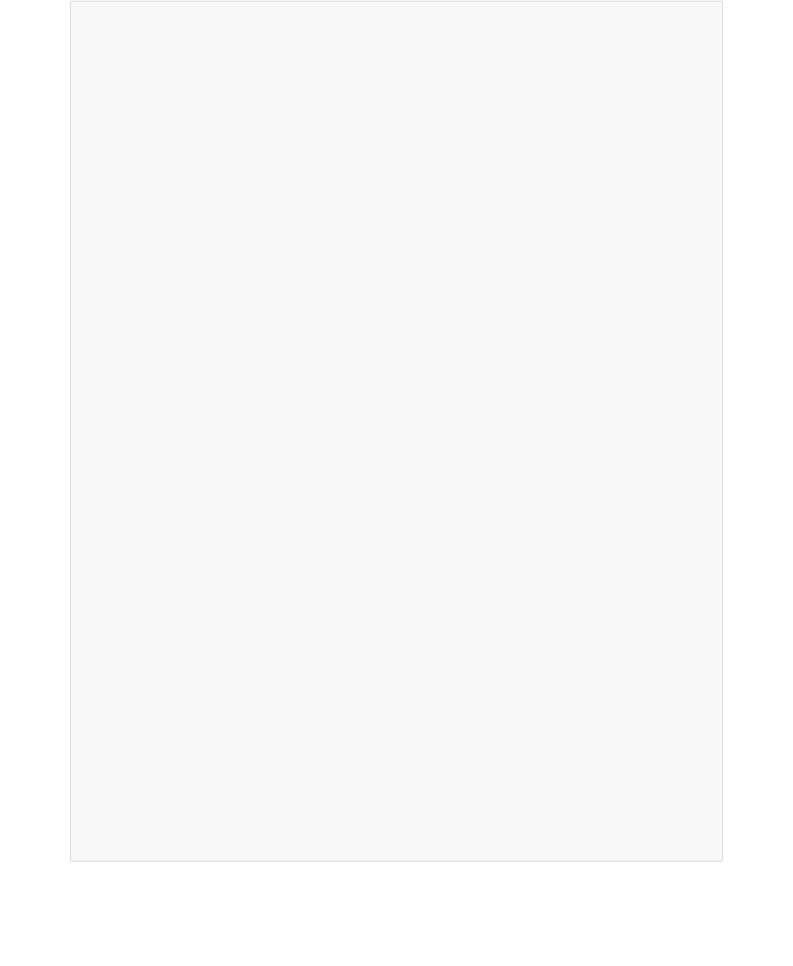
}

**信息回写分析图解**

前四步与基本文件上传一致.

1. 【服务端】获取输出流，回写数据。
2. 【客户端】获取输入流，解析回写数据。

**回写实现**



 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();

        }

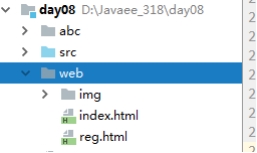
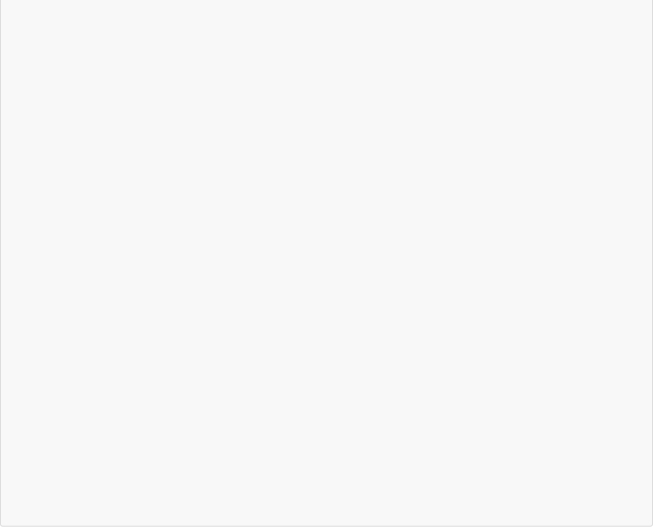
    }

}

**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);

        }

        // 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**

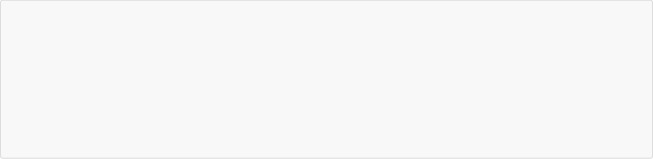
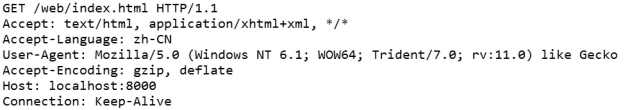
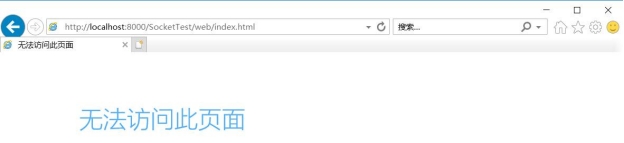
模拟网站服务器，使用浏览器访问自己编写的服务端程序，查看网页效果。

**案例分析**

1. 准备页面数据，web文件夹。

复制到我们Module 中，比如复制到day08中

2. 我们模拟服务器端，ServerSocket 类监听端口，使用浏览器访问



 //转换流,读取浏览器请求第一行

BufferedReader readWb = new BufferedReader(new InputStreamReader (socket.getInputStream())); String requst = readWb.readLine ();

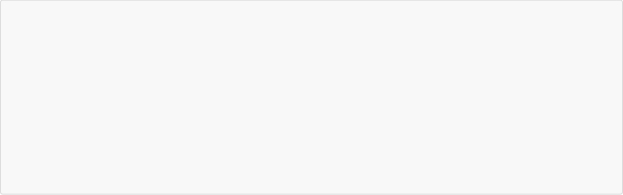
//取出请求资源的路径

String[] strArr = requst.split(" ");

//去掉web前面的/

String path = strArr[1].substring(1);

System.out.println(path);



 public static void main(String[] args) throws IOException  {         ServerSocket  server = new ServerSocket(8000);         Socket socket = server.accept();

        InputStream  in = socket.getInputStream ();

        byte[] bytes = new byte[1024];

        int len = in.read(bytes);

        System.out.println (new String(bytes,0,len));

        socket.close();

        server.close();

}

3. 服务器程序中字节输入流可以读取到浏览器发来的请求信息

GET/web/index.html HTTP/1.1 是浏览器的请求消息。/web/index.html 为浏览器想要请求的服务器端的资源,使用 字符串切割方式获取到请求的资源。

**案例实现**

服务端实现：

 public class SerDemo {

    public static void main(String[] args) throws IOException  {         System.out.println ("服务端  启动 ,等待连接 .... ");         // 创建ServerSocket 对象

        ServerSocket server = new ServerSocket(8888);         Socket socket = server.accept();

        // 转换流读取浏览器的请求消息

        BufferedReader readWb = new



        BufferedReader(new InputStreamReader(socket.getInputStream()));         String requst = readWb.readLine ();

        // 取出请求资源的路径

        String[] strArr = requst.split(" ");

        // 去掉web前面的/

        String path = strArr[1].substring(1);

        // 读取客户端请求的资源文件

        FileInputStream fis = new FileInputStream(path);

        byte[] bytes= new byte[1024];

        int len = 0 ;

        // 字节输出流,将文件写会客户端

        OutputStream out = socket.getOutputStream ();

        // 写入HTTP协议响应头,固定写法

        out.write("HTTP/1.1 200 OK\r\n".getBytes());

        out.write("Content‐Type:text/html\r\n" .getBytes ());

        // 必须要写入空行,否则浏览器不解析

        out.write("\r\n".getBytes());

        while((len = fis.read(bytes))!=‐1){

            out.write(bytes,0,len);

        }

        fis.close();

        out.close();

        readWb.close();

        socket.close();

        server.close();

    }

}

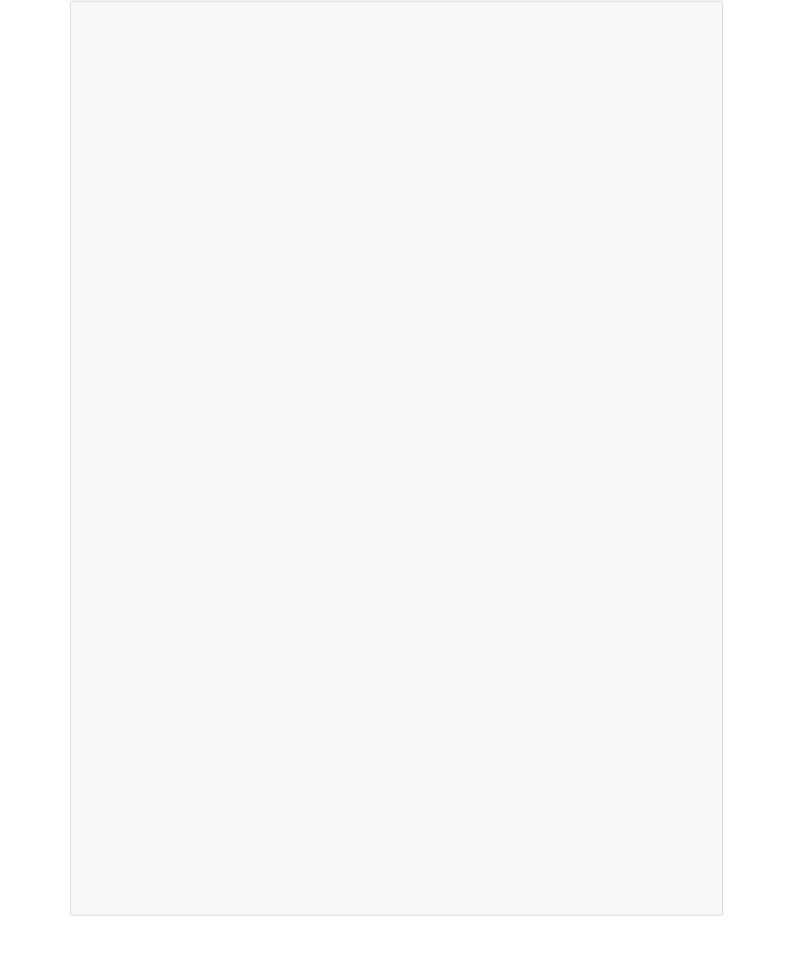
**访问效果**

**火狐**

小贴士：不同的浏览器，内核不一样，解析效果有可能不一样。

发现浏览器中出现很多的叉子,说明浏览器没有读取到图片信息导致。

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();

        }

    }

    static 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 requst = readWb.readLine ();

                //取出请求资源的路径

                String[] strArr = requst.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){

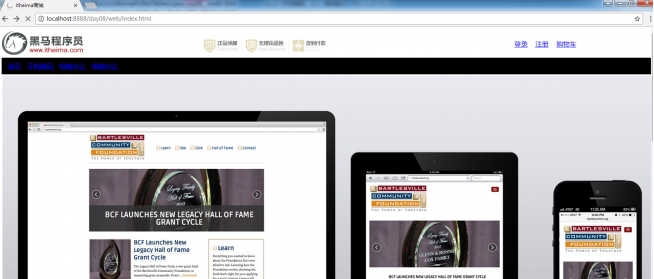
            }

        }

    }

}

**访问效果：**



图解：