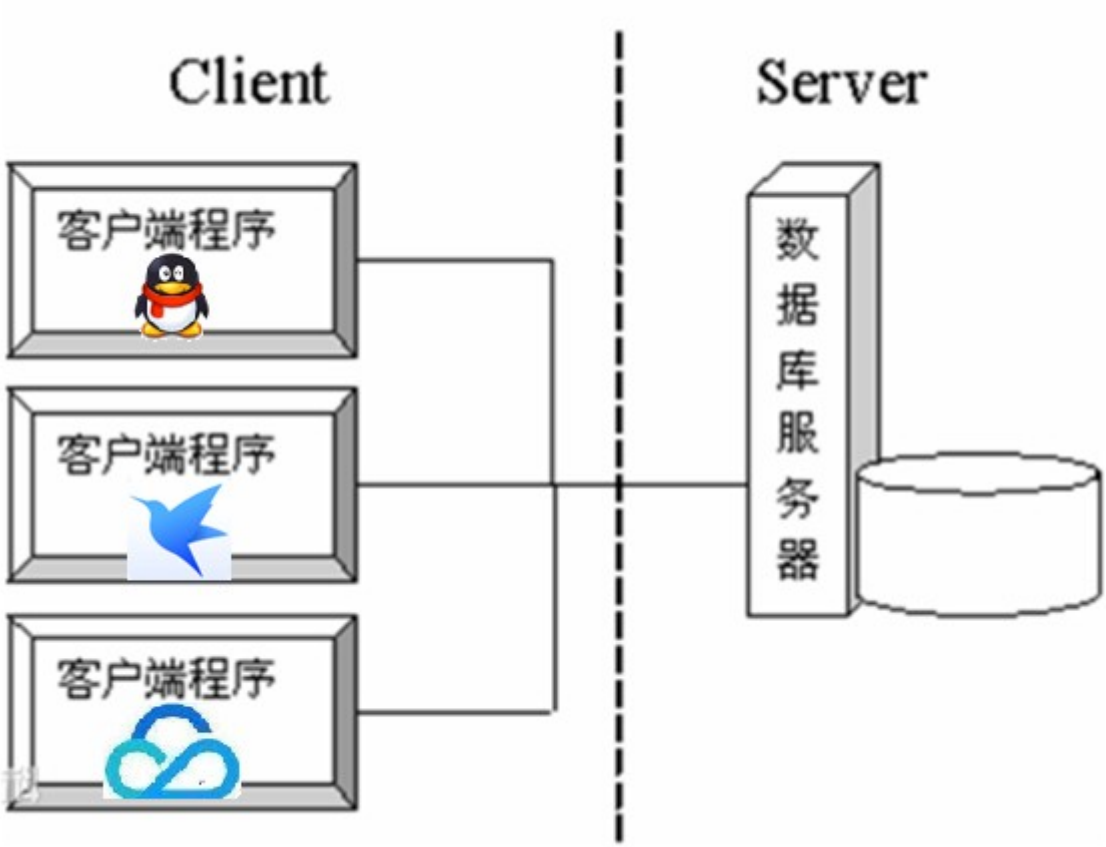
网络编程

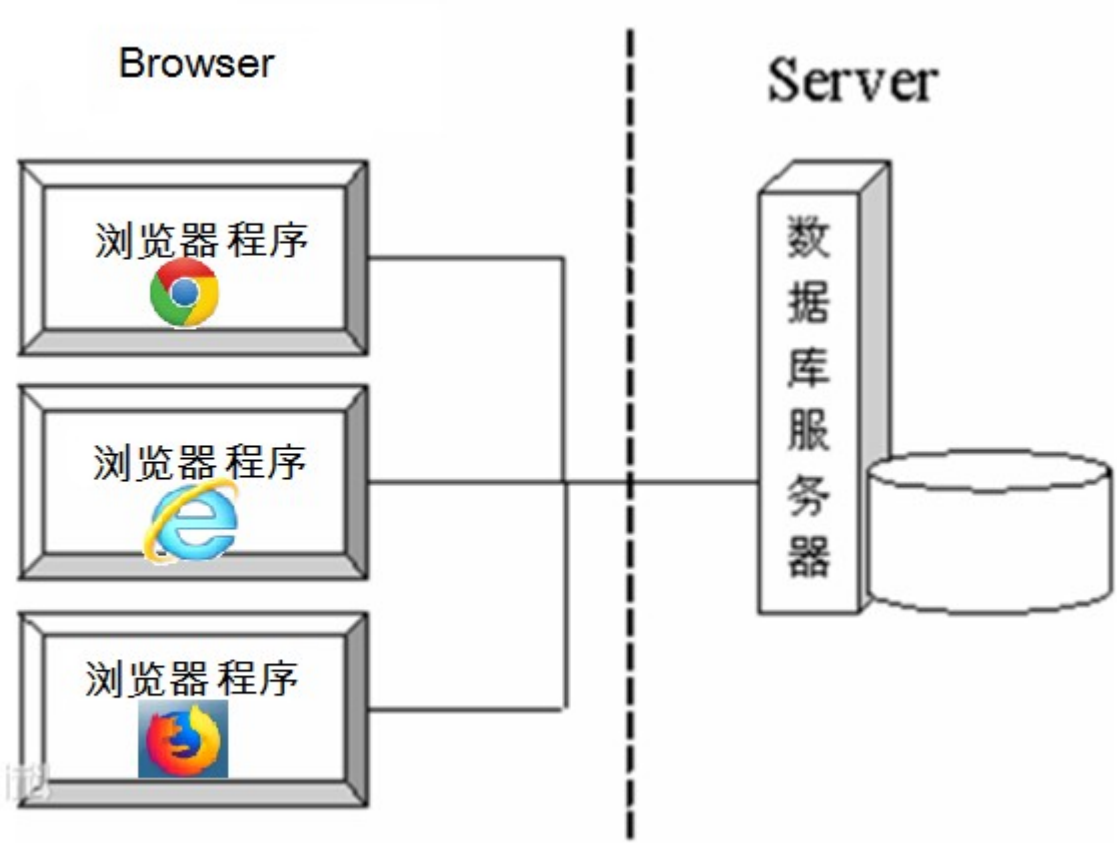
第一章 网络编程入门

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。



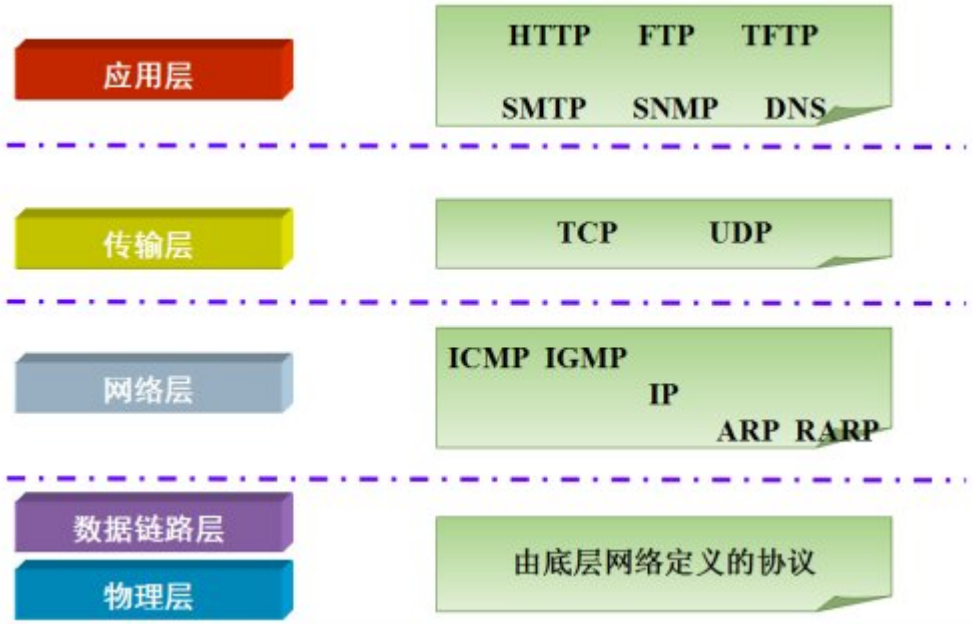
B/S结构： 全称为Browser/Server结构，是指浏览器和服务器结构。常见浏览器有谷歌、火狐等。



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：**通过计算机网络可以使多台计算机实现连接，位于同一个网络中的计算机在进行连接和通信时需要遵守一定的规则，这就好比在道路中行驶的汽车一定要遵守交通规则一样。在计算机网络中，这些连接和通信的规则被称为网络通信协议，它对数据的传输格式、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才能完成数据交换。
- **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。



上图中，TCP/IP协议中的四层分别是应用层、传输层、网络层和链路层，每层分别负责不同的通信功能。 链路层：链路层是用于定义物理传输通道，通常是对某些网络连接设备的驱动协议，例如针对光纤、网线提供的驱动。 网络层：网络层是整个TCP/IP协议的核心，它主要用于将传输的数据进行分组，将分组数据发送到目标计算机或者网络。 运输层：主要使网络程序进行通信，在进行网络通信时，可以采用TCP协议，也可以采用UDP协议。 应用层：主要负责应用程序的协议，例如HTTP协议、FTP协议等。

1.3 协议分类

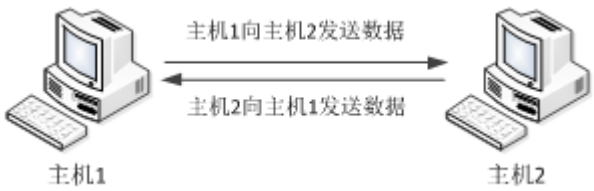
通信的协议还是比较复杂的，`java.net`包中包含的类和接口，它们提供低层次的通信细节。我们可以直接使用这些类和接口，来专注于网络程序开发，而不用考虑通信的细节。

`java.net`包中提供了两种常见的网络协议的支持：

- **UDP**：用户数据报协议(User Datagram Protocol)。UDP是无连接通信协议，即在数据传输时，数据的发送端和接收端不建立逻辑连接。简单来说，当一台计算机向另外一台计算机发送数据时，发送端不会确认接收端是否存在，就会发出数据，同样接收端在收到数据时，也不会向发送端反馈是否收到数据。

  由于使用UDP协议消耗资源小，通信效率高，所以通常都会用于音频、视频和普通数据的传输例如视频会议都使用UDP协议，因为这种情况即使偶尔丢失一两个数据包，也不会对接收结果产生太大影响。

  但是在使用UDP协议传送数据时，由于UDP的面向无连接性，不能保证数据的完整性，因此在传输重要数据时不建议使用UDP协议。UDP的交换过程如下图所示。



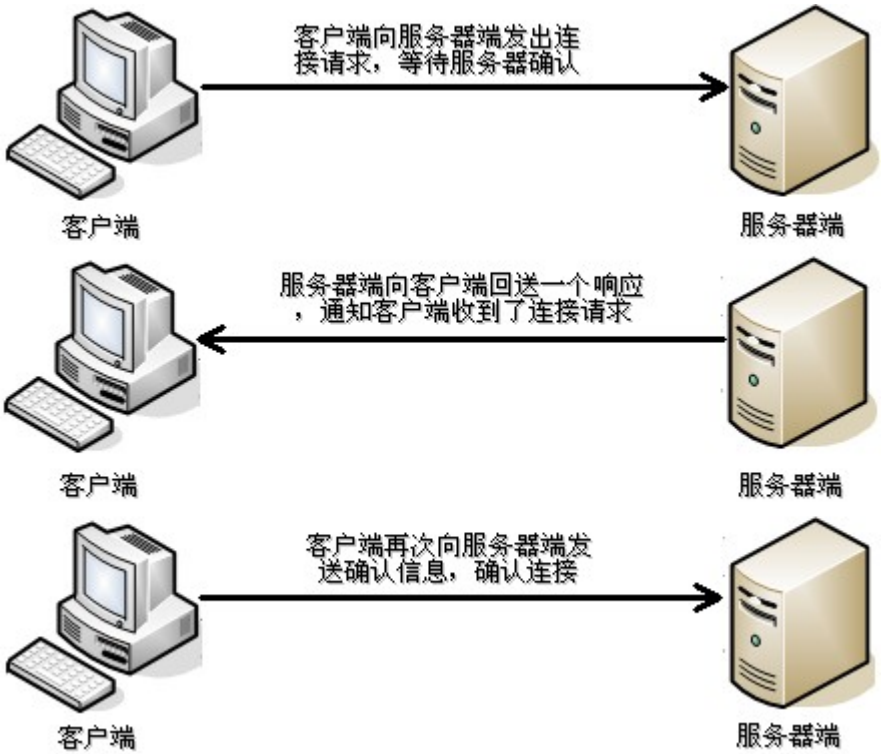
特点:数据被限制在64kb以内，超出这个范围就不能发送了。

数据报(Datagram):网络传输的基本单位

- **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。

在TCP连接中必须要明确客户端与服务器端，由客户端向服务端发出连接请求，每次连接的创建都需要经过“三次握手”。

- 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。

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。

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
```

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

- `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：

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

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：

```
ServerSocket server = new ServerSocket(6666);
```

成员方法

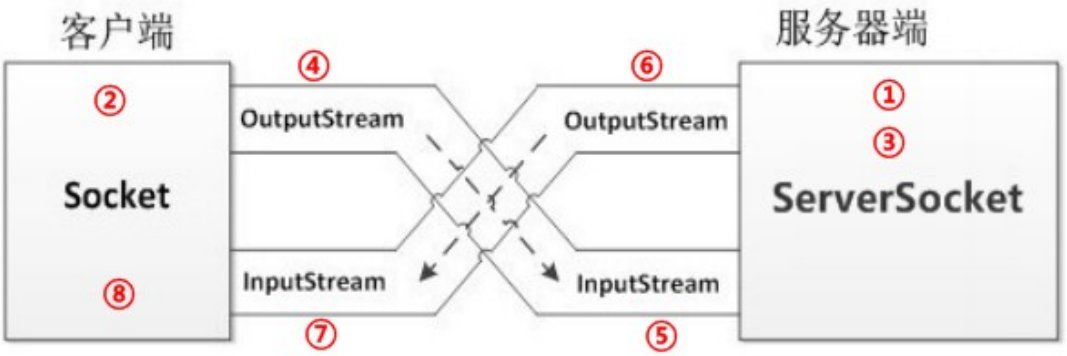
- `public Socket accept()` ：侦听并接受连接，返回一个新的Socket对象，用于和客户端实现通信。该方法会一直阻塞直到建立连接。

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。
2. 【客户端】启动,创建Socket对象，请求连接。
3. 【服务端】接收连接,调用accept方法，并返回一个Socket对象。
4. 【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();
    }
}
```

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.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();
    }
}
```

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);
```

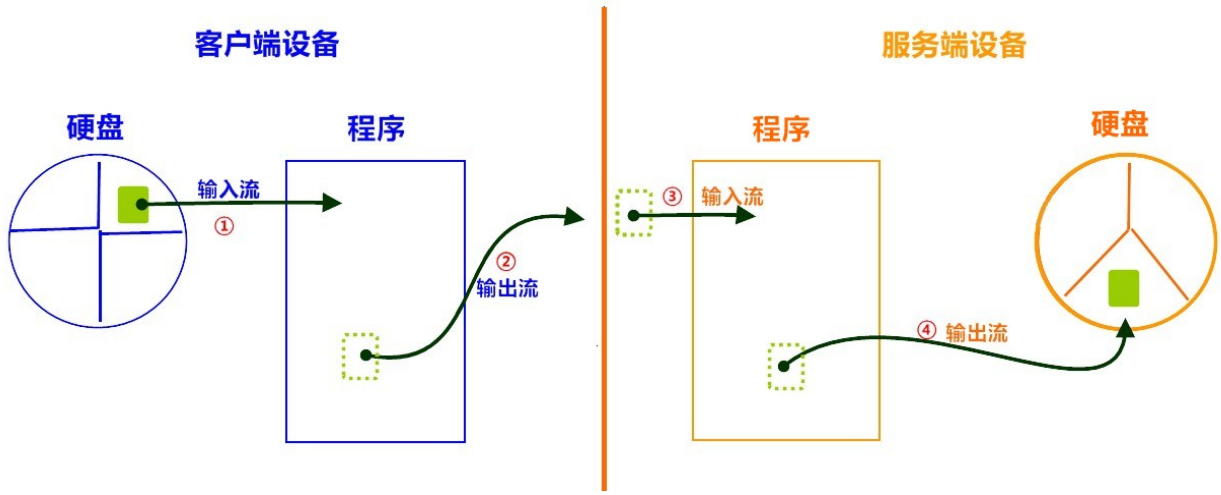
```
            // 2.通过Scoket,获取输出流对象
            OutputStream os = client.getOutputStream();
            // 3.写出数据.
            os.write("你好么? tcp ,我来了".getBytes());
            // ==============解析回写=========================
            // 4. 通过Scoket,获取 输入流对象
            InputStream in = client.getInputStream();
            // 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

TCP通信的文件上传案例

原理:客户端读取本地的文件,把文件上传到服务器,服务器在把上传的文件保存到服务器的硬盘上

1.客户端使用本地的字节输入流,读取要上传的文件
2.客户端使用网络字节输出流,把读取到的文件上传到服务器
3.服务器使用网络字节输入流,读取客户端上传的文件
4.服务器使用本地字节输出流,把读取到的文件,保存到服务器的硬盘上
5.服务器使用网络字节输出流,给客户端回写一个"上传成功"
6.客户端使用网络字节输入流,读取服务器回写的数据
7.释放资源

注意:

客户端和服务器和本地硬盘进行读写,需要使用自己创建的字节流对象(本地流)

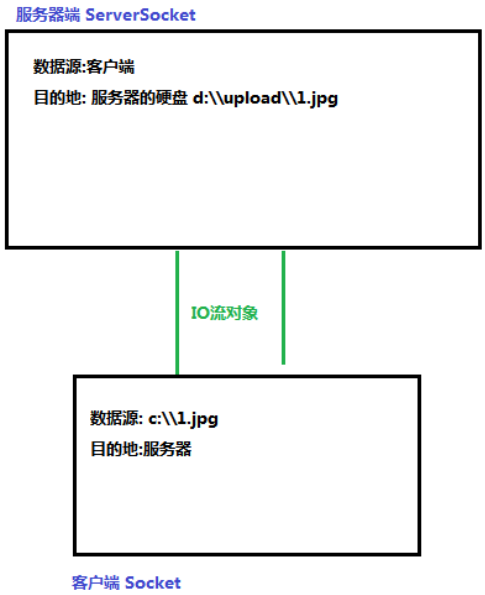
客户端和服务器之间进行读写,必须使用Socket中提供的字节流对象(网络流)

文件上传的原理,就是文件的复制

明确:

数据源

数据目的地



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

文件上传优化分析

```
public class TCPClient {
    public static void main(String[] args) throws IOException {
        //1.创建一个本地字节输入流FileInputStream对象,构造方法中绑定要读取的数据源
        FileInputStream fis = new FileInputStream( name: "c:\\1.jpg");
        //2.创建一个客户端Socket对象,构造方法中绑定服务器的IP地址和端口号
        Socket socket = new Socket( host: "127.0.0.1", port: 8888);
        //3.使用Socket中的方法getOutputStream,获取网络字节输出流OutputStream对象
        OutputStream os = socket.getOutputStream();
        //4.使用本地字节输入流FileInputStream对象中的方法read,读取本地文件
        int len = 0;                          fis.read(bytes) 读取本地文件,结束标记是读取到-1结束
        byte[] bytes = new byte[1024];
        while((len = fis.read(bytes))!=-1){   while循环里会读取-1吗? 不会 那么也不会把结束标记写个服务器
            //5.使用网络字节输出流OutputStream对象中的方法write,把读取到的文件上传到服
            os.write(bytes, off: 0,len);
        }                                     解决:上传完文件,给服务器写一个结束标记

        //6.使用Socket中的方法getInputStream,获取网络字节输入流InputStream对象
        InputStream is = socket.getInputStream();

        //7.使用网络字节输入流InputStream对象中的方法read读取服务回写的数据
        while((len = is.read(bytes))!=-1){  is.read 读取不到服务器回写的数据,进入到阻塞状态
            System.out.println(new String(bytes, offset: 0,len));
        }

        //8.释放资源(FileInputStream,Socket)
        fis.close();
        socket.close();
    }
}
```

```
public class TCPServer {
    public static void main(String[] args) throws IOException {
        //1.创建一个服务器ServerSocket对象,和系统要指定的端口号
        ServerSocket server = new ServerSocket( port: 8888);
        //2.使用ServerSocket对象中的方法accept,获取到请求的客户端Socket对象
        Socket socket = server.accept();
        //3.使用Socket对象中的方法getInputStream,获取到网络字节输入流InputStream对象
        InputStream is = socket.getInputStream();
        //4.判断d:\\upload文件夹是否存在,不存在则创建
        File file =  new File( pathname: "d:\\upload");
        if(!file.exists()){
            file.mkdirs();
        }
        //5.创建一个本地字节输出流FileOutputStream对象,构造方法中绑定要输出的目的地
        FileOutputStream fos = new FileOutputStream( name: file+"\\1.jpg");
        //6.使用网络字节输入流InputStream对象中的方法read,读取客户端上传的文件
        int len =0;
        byte[] bytes = new byte[1024];        is.read 读取客户端上传的文件,永远也读取不到文件的结束标记
        while((len = is.read(bytes))!=-1){read方法计入到阻塞状态,一直死循环等待结束标记
            //7.使用本地字节输出流FileOutputStream对象中的方法write,把读取到的文件保存到服务器的硬
            fos.write(bytes, off: 0,len);
        }
        8 9 10 代码就不会执行到 也不会给客户端回写上传成功
        //8.使用Socket对象中的方法getOutputStream,获取到网络字节输出流OutputStream对象
        //9.使用网络字节输出流OutputStream对象中的方法write,给客户端回写"上传成功"
        socket.getOutputStream().write("上传成功".getBytes());
        //10.释放资源(FileOutputStream,Socket,ServerSocket)
        fos.close();
        socket.close();
        server.close();
    }
}
```

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：

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

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：

```
// 每次接收新的连接,创建一个Socket
while (true) {
    Socket accept = serverSocket.accept();
    ......
}
```

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：

```
while (true) {
    Socket accept = serverSocket.accept();
    // accept 交给子线程处理.
    new Thread(() -> {
        ......
        InputStream bis = accept.getInputStream();
        ......
    }).start();
}
```

优化实现

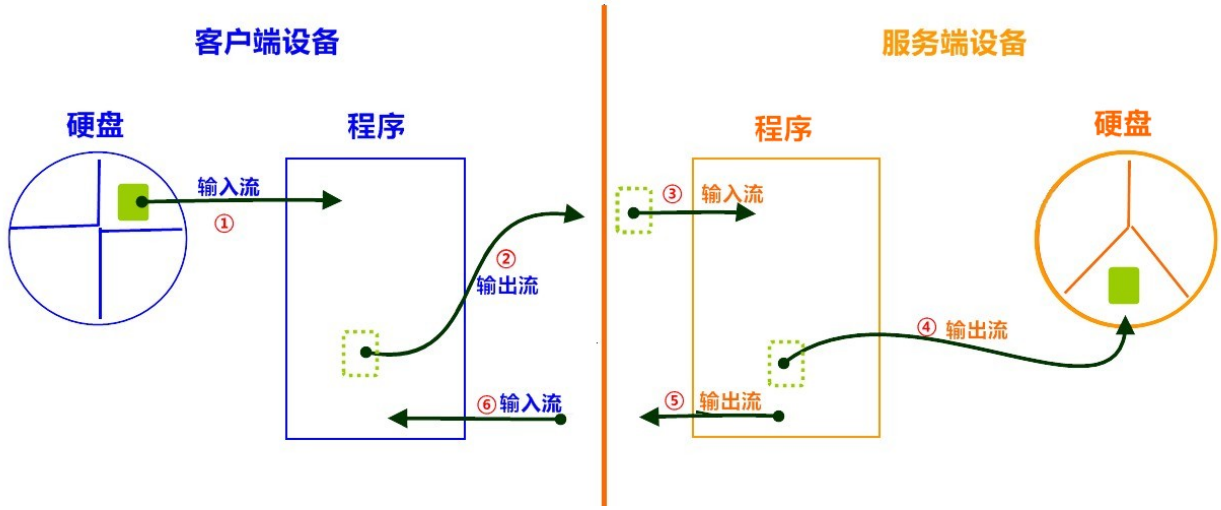
```
public class FileUpload_Server {
    public static void main(String[] args) throws IOException {
        System.out.println("服务器 启动.....  ");
        // 1. 创建服务端ServerSocket
        ServerSocket serverSocket = new ServerSocket(6666);
        // 2. 循环接收,建立连接
        while (true) {
            Socket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

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。



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();
    }
}
```

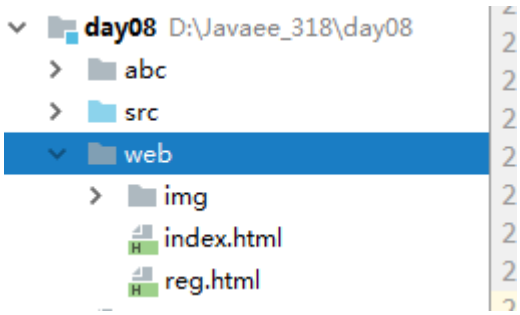
3.2 模拟B\S服务器(扩展知识点)

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

```
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();
}
```

无法访问此页面

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
Accept: text/html, application/xhtml+xml, */*
Accept-Language: zh-CN
User-Agent: Mozilla/5.0 (Windows NT 6.1; WOW64; Trident/7.0; rv:11.0) like Gecko
Accept-Encoding: gzip, deflate
Host: localhost:8000
Connection: Keep-Alive
```

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。

```
//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);
```

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**

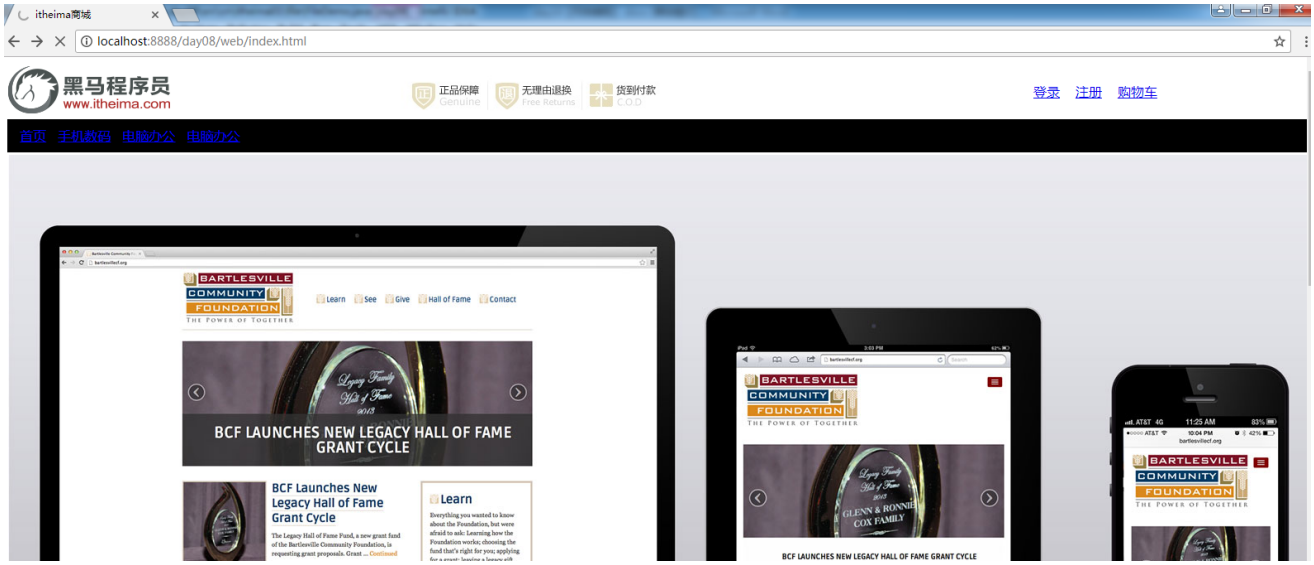
> 小贴士：不同的浏览器，内核不一样，解析效果有可能不一样。

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){

            }
        }
    }

}
```

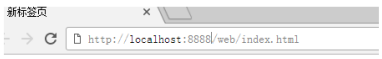
访问效果：



图解：

B\S通信图解

客户端

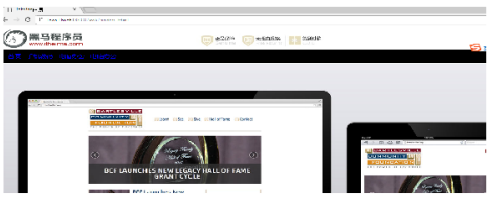


1： 输入访问地址回车，向服务器发送请求
如果服务器存在，可以正常连接那么就向服务器发送请求头

2： 下面就是客户端发给服务器的请求信息在客户端服务器之间的流中

```
GET /web/index.html HTTP/1.1
Accept: text/html, application/xhtml+xml, */*
Accept-Language: zh-CN
User-Agent: Mozilla/5.0 (Windows NT 6.1; WOW64; Trident/7.0; rv:11.0) like Gecko
Accept-Encoding: gzip, deflate
Host: localhost:8000
Connection: Keep-Alive
```

5：浏览器得到响应信息，就在页面进行数据的展示。



服务器端

对于服务器来说解析请求信息：
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。

3：因为在流中有用的信息在第一行，所以处理第一行数据

```
//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);
```

4：处理完之后 根据客户端想要访问的路径，读取资源文件，并写回去。