通过前面的学习，我们实战过用tomcat作为服务器承载服务，也学习过Socket编程，为了更好地学习HTTP协议，本节我们用代码实现一个简易的web服务器，满足接收客户端请求、处理和响应的功能。

本文我使用JAVA来实现，当然了，读者朋友可以使用其他的语言和技术，比如nodejs来实现只需要几行代码。

老规矩，代码仓库见文末。

一、接收客户端请求

如何通过JAVA来实现一个最简易的web服务呢？并且我们打印下请求头等信息。代码如下：

```
public static void main(String[] args) throws IOException {
    //服务端监听在8888端接口号
    ServerSocket server = new ServerSocket(8888);
    System.out.println("服务器已经启动...正在监听8888端口，随时等待客户端连接");
    //服务端创建一个线程来处理客户端请求
    while (!Thread.interrupted()){
        //接收用户请求
        Socket client = server.accept();
        //获取输入输出流
        InputStream ins = client.getInputStream();
        OutputStream out = client.getOutputStream();
        //打印获取到的请求内容
        int len = 0;
        byte[] b = new byte[1024];
        while((len = ins.read(b)) != -1){
            System.out.println(new String(b,0,len));
        }
    }
}
```

此时我们访问地址： `localhost:8888` 打印出来的结果为：

```
服务器已经启动...正在监听8888端口，随时等待客户端连接
GET / HTTP/1.1
Host: localhost:8888
User-Agent: curl/7.55.1
Accept: */*
```

关注下第一行，由于我其实请求的是根路径，所以是/，如果我在这里请求 `localhost:8888/index.html` 那么就会显示 `GET /index.html HTTP/1.1` 这样的信息，使用的HTTP协议是HTTP/1.1。

但是上面的写法是存在很多问题的，不过不重要，我们先完善下功能，让他给客户端返回点什么。

二、接收+响应客户端请求

由于HTTP是超文本传输协议，我们本次返回一个典型的HTML页面。

首先我们得有资源才能展示，假设我们要展示 `index.html` ，我们将其暂时放在 `F:/webroot` 下。

里面的内容十分简单，就是借助HTML的标签显示一些文字，就像我们初次学习编程时打印"hello world"一样，我在此HTML中放置的内容是：

```
<!DOCTYPE html>
<html>
<head>
    <title>httpserver test page</title>
</head>
<body>
<h1><font color="green">hello world!</font></h1>
</body>
</html>
```

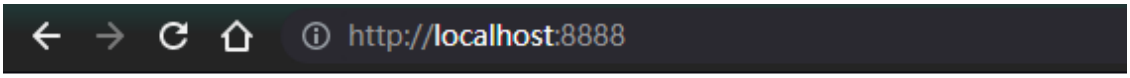
好了，素材准备了，下面就是升级代码，想办法将html的内容返回给浏览器。

服务端需要读取这个文件，然后以流的形式发送给客户端的浏览器上，浏览器再解析展示。

```
public static void main(String[] args) throws IOException {
    ServerSocket server = new ServerSocket(8888);
    System.out.println("服务器已经启动...正在监听8888端口，随时等待客户端连接");
    //服务端创建一个线程来处理客户端请求
    while (!Thread.interrupted()){
        //接收用户请求
        Socket client = server.accept();
        //获取输入输出流
        InputStream ins = client.getInputStream();
        OutputStream out = client.getOutputStream();
        //给用户响应
        //首先读取html文件流，准备返回给浏览器
```

```
        PrintWriter pw = new PrintWriter(out);
        InputStream i = new
FileInputStream("f:\\webroot\\index.html");
        BufferedReader br = new BufferedReader(new
InputStreamReader(i));
        //配置http协议需要的响应头信息，注意响应头结束后有一行空行
        pw.println("HTTP/1.1 200 OK");
        pw.println("Content-Type: text/html;charset=utf-8");
        pw.println("Content-Length:" + i.available());
        pw.println("Server:hello-server");
        pw.println("Date:"+new Date());
        pw.println("");
        //在空行之后就是返回响应体，即html给浏览器渲染展示
        String c = null;
        while ((c = br.readLine()) != null){
            pw.println(c);
        }
        pw.flush();
        System.out.println(" 本次请求处理结束");
    }
}
```

此时再在浏览器上去访问 `http://localhost:8888/` ，就会显示欢迎的信息啦！



hello world!

响应信息真的如我们代码设置的一样吗，我们抓包来验证下：

三、引入多线程

由于客户端往往是多个同时请求过来，服务端若只有一个线程在处理，整体会很慢，考虑引入多线程并行处理客户端请求，我们改进demo。

```
public static void main(String[] args) throws IOException {
    ServerSocket server = new ServerSocket(8888);
    System.out.println("服务器已经启动...正在监听8888端口，随时等待客户端连接");
    //服务端创建一个线程来处理客户端请求
    while (!Thread.interrupted()){
        //接收用户请求
        Socket client = server.accept();
        //新开一个线程去处理请求
        new Thread(new ServerThread(client)).start();
    }
    server.close();
}
```

更进一步地，可以使用线程池来分配线程去处理：

```
 public static void main(String[] args) throws IOException {
    ExecutorService pool = Executors.newCachedThreadPool();

    ServerSocket server = new ServerSocket(8888);
    System.out.println("服务器已经启动...正在监听8888端口，随时等待客户端连接");
    //服务端创建一个线程来处理客户端请求
    while (!Thread.interrupted()){
```

```
        //接收用户请求
        Socket client = server.accept();

        pool.execute(new ServerThread(client));
    }
    server.close();
}
```

下面核心的就是这个线程类的处理，其实跟之前是一样的：

```
public class ServerThread implements Runnable {
    private Socket client;
    InputStream ins;
    OutputStream out;

    public ServerThread(Socket client){
        this.client = client;
        init();
    }

    private void init(){
        try {
            ins = client.getInputStream();
            out = client.getOutputStream();
        } catch (IOException e) {
            e.printStackTrace();
        }
    }

    @Override
    public void run() {
        try {
            go();
        } catch (IOException e) {
            e.printStackTrace();
        }
    }

    private void go() throws IOException {
        //给用户响应
        PrintWriter pw = new PrintWriter(out);
        InputStream i = new
FileInputStream("f:\\webroot\\index.html");
        BufferedReader br = new BufferedReader(new
InputStreamReader(i));
        pw.println("HTTP/1.1 200 OK");
```

```
        pw.println("Content-Type: text/html;charset=utf-8");
        pw.println("Content-Length:" + i.available());
        pw.println("Server:hello-server");
        pw.println("Date:"+new Date());
        pw.println("");
        pw.flush();

        String c = null;
        while ((c = br.readLine()) != null){
            pw.println(c);
        }
        pw.flush();

        pw.close();
        br.close();
        i.close();
        client.close();
    }
}
```

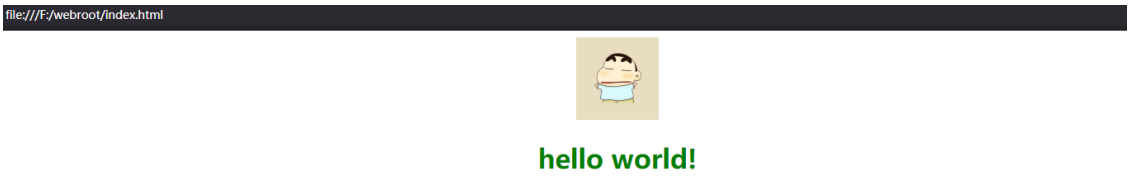
四、丰富服务端的处理逻辑

若我想在html中显示一张图片呢?

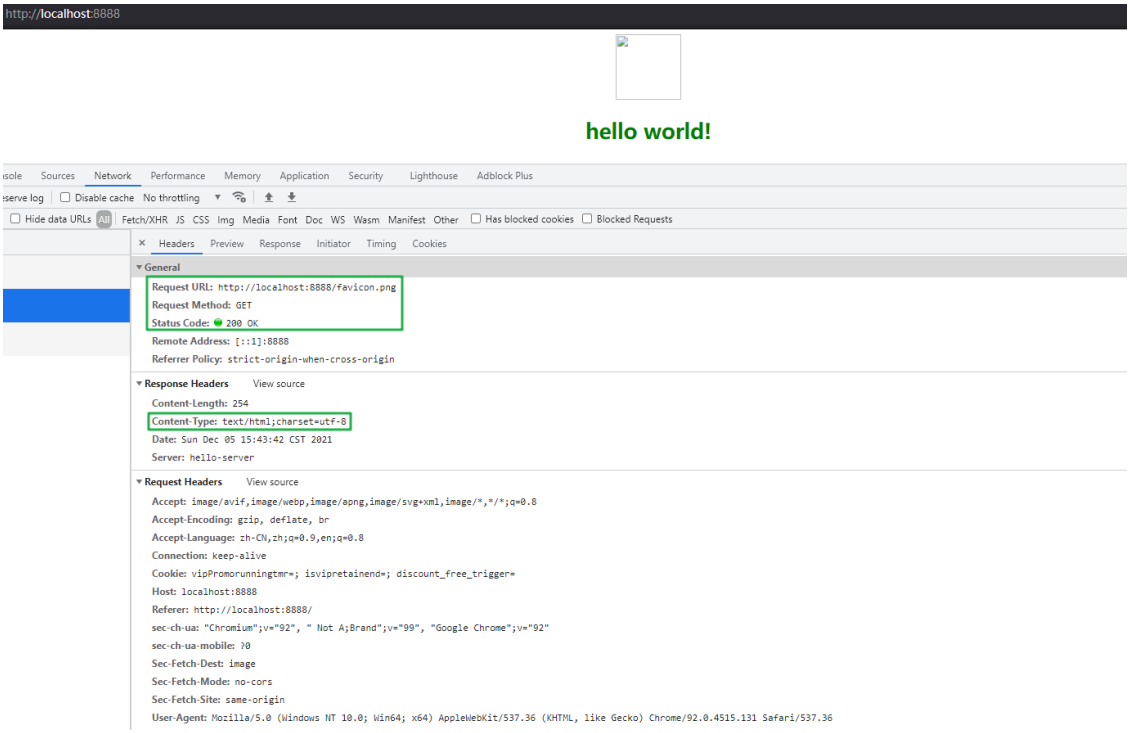
于是我快速修改了html代码:

```
<!DOCTYPE html>
<html>
<head>
  <title>httpserver test page</title>
</head>
<body>
<div align="center">

<img src="./favicon.png" width="100px" height="100px">
</br>
<h1><font color="green">hello world!</font></h1>

</div>

</body>
</html>
```

我希望得到的效果是:



但是当我启动代码，发现无法正常展示图片：



原因就在响应头中，所响应的content-type是text/html;charset=utf-8，这显然不是图片对应的类型，一看代码，原来我们是将content-type的类型写死了，导致浏览器侧无法正常渲染。

那么就需要优化代码，让其更加聪明，由于我们这里demo比较简单，可根据请求资源的后缀进行判断，如果是图片类型，则将响应头调整为图片的，我们设置了一个类型池子：

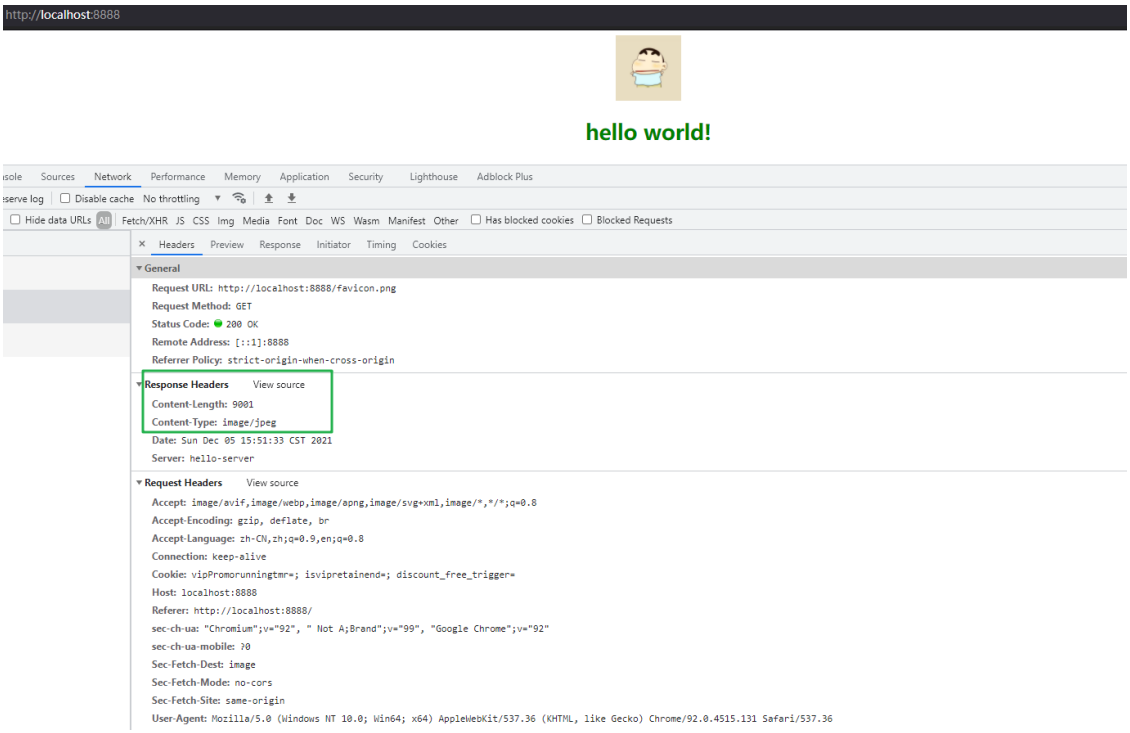
```
//存放类型，比如jpg对应的是image/jpeg，这是http协议规定的每种类型的响应格式
private static Map<String,String> contentMap =  new HashMap<>();
static {
    contentMap.put("html","text/html;charset=utf-8");
    contentMap.put("jpg","image/jpeg");
    contentMap.put("png","image/jpeg");
}
```

接下来就是根据资源的后缀类型去动态返回，核心代码是：

```
//1、获取请求资源名称
```

```
BufferedReader reader = new BufferedReader(new
InputStreamReader(ins));
String requestPath = reader.readLine().split(" ")
[1].replace("/","\\");
if(requestPath.equals("\\")){
    requestPath += "index.html";
}

//2、拼接起来就是资源的完整路径
File file = new File(webroot + requestPath);

//3、给用户响应
if (file.exists()) {
    //给用户响应
    PrintWriter pw = new PrintWriter(out);
    InputStream i = new FileInputStream(webroot + requestPath);
    //由于需要将图片也要传给前端，再用这个就不好办了，得用普通的文件输入流
    pw.println("HTTP/1.1 200 OK");
    //返回的类型是动态判断的，图片用图片的类型，文本用文本的类型
    String s =
contentMap.get(requestPath.substring(requestPath.lastIndexOf(".")+1
,requestPath.length()));
    System.out.println("返回的类型为："+ s);
    pw.println("Content-Type: " + s);
    ......
```

详细可见代码，通过这番调整后，我们终于顺利访问到了预期的页面：



五、结语

至此，我们基于JAVA完成了一个比较简单的web服务器的开发，读者朋友们，你们不妨也用自己擅长的语言来实现一下。

本文只是一个小demo，主要是为后续HTTP协议的深入学习做准备，本文后续行文也不会太突兀，若不实践也不影响后续学习。

代码仓库：`https://github.com/sunweiguo/httpserver`

PS：代码仅为简单演示使用，请忽略写法细节。