# 作业 1 构造一个简单的 Web 服务器

2015220103022 信工三班 张健顺

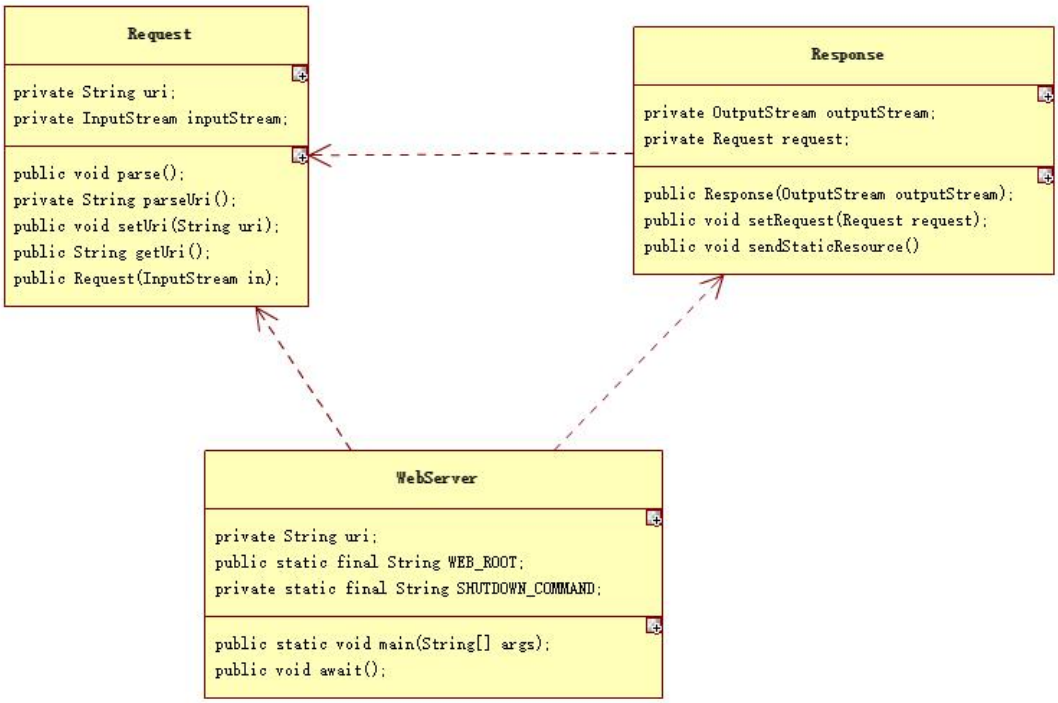
**目标：**

客户机基本基本流程：1、仅处理一个 HTTP 请求。① 接受请求；② 解析头部；③ 从服务器文件系统中得到所请求的文件；④ 生成 HTTP 请求报文，首部文件；⑤ 向客户机发送请求报文。2、安装服务器后可以在其他主机上使用浏览器，向服务器发送一个请求文件。

服务器基本流程：① 创建套接字； ② 获取 HTTP 请求，并解析 HTTP 请求报文；③ 显示请求报文各个字段字段名和值，并进行说明；④ 根据 HTTP 请求报文获得对象的文件路径名；⑤ 根据路径名打开本地文件；⑥ 封装本地文件到 HTTP 响应报文；⑦ 使用套接字发送 HTTP 响应报文。

**设计思想：**

根据客户端和服务交互过程中最重要的两类对象请求报文和响应报文分别设计对应的类 request 和 response，其中各自包含对报文段中内容的提取与处理，在设计出相应的 WebServer 类来实现整个服务器的过程。




**设计步骤：**

**1.Request 类**

① 明确 request 请求的实质，实质为输入流；对于 WebServer 一方，请求发送到 WebServer，WebServer 给出相应的反馈信息，request 即为输入流，对于存在于服务器上的某一个文件的请求输入流。故采用构造方法中传入输入流作为参数来对 Request 对象进行创建。

② 明确 request 请求对于 WebServer 最为主要的功能就是发送请求报文，并提取

出相关统一资源标识符 uri，交付给 WebServer，故产生了 parse()和 parseUri()方法以及 uri 的成员变量。而其中 parse()方法主要是用于打印输出 Request 的报文信息，在其中调用限定符为 private 的 parseUri()方法来打印出对应的 uri 信息（服务器上所要请求的文件资源的路径）

③ parse() 方法主要是需要将输入流 inputStream 中包含的信息给解析出来，打印成相应的字符串，又因为 inputStream 本身存储相关信息是字节码，而不是易读的字符串类型，所以需要利用字符串缓冲 StringBuffer 和 字节数组 Buffer[] 来将 inputStream 中的信息读取出来，再利用强制转换以及字符串拼接将字节数组转换成字符串。

④ 而 parseUri() 则是在 parse() 的基础上，利用请求报文格式的特点，uri 位于第一行的方法之后与方法间隔了一个空格 eg. GET /somedir/page.html HTTP/1.1，采用通过求子串的方法，根据 uri 前后的两个空格来提取 request 中的 uri

**2.Response 类**

① 与 Request 相对应的就能知道 Response 相对于 WebServer 而言其本质为输出流。对于接收到请求 request，webServer 需要做出相应的反馈即 response 来进行输出，又因为 webServer 需要根据 request 中相应的 uri 的信息来输出 response，故在 response 将 request 作为了相应的属性。

② Response 最主要的功能则是将 request 发来的请求中 uri 的信息拿来并创建相应的文件对象，若文件存在，则将文件中的信息写入到输出流 outputStream 中，并将数据段和报文头进行封装。若不存在相应的文件信息，也要给出相应的报文并打印。在将信息写入到输出流的过程中，对于报文头段，则是通过已知的报文格式，静态输出打印，然后在其后添加数据段，数据段则采用文件相应的输入流、输入缓冲 BufferedReader 以及相应的字节数组 byte[] 来将文件中存储的信息给打印输出到控制台。

**3.WebServer 类**

WebServer 主要是要在客户端和服务器端建立起相应的通信来，并对输入输出流的（也就是相应的 request 和 response）发送与接收进行相应的控制，建立通信则主要是利用套接字编程的相关知识。创建相应的服务器端套接字，并绑定相应的端口号以及主机地址，创建完成以后，循环等待客户端发来的 request 请求，连接成功以后，根据服务器端套接字接收到的信息创建一个新的套接字 socket = serverSocket.accept();并根据该套接字获得相应的输入输出流，再利用 Request 对象解析出输入流对应的请求报文，调用 request 中相应的方法并打印输出 uri，利用 Response 对象解析输出流对应的响应报文，并调用 response 中相应的方法打印输出相应报文的相关信息。

**程序源码：**

**Request.java**

```
public class Request {

    private InputStream inputStream;

    private String uri;

    // 从 InputStream 中读取 request 信息并打印，并从 request 中获取 uri 值
    public void parse() {
```

```
            // 创建字符串缓冲
            StringBuffer request = new StringBuffer(2048);
            //
            int i;
            // 创建字节数组
            byte[] buffer = new byte[2048];
            try {
        // 读取数据填充到字节数组 buffer，并返回读取的字节数 public int read(byte[] r)
                i = inputStream.read(buffer);
            } catch (IOException e) {
                e.printStackTrace();
                i = -1;
            }
            // i 为读取的字节数，
            for (int j = 0; j < i; j++) {
                // 将字节数组转换成相应的字符串，即为 request
                request.append((char) buffer[j]);
            }
            // 打印出 request 的信息
            System.out.print(request.toString());
            // 从 request 中获取 uri 的值
            uri = parseUri(request.toString());
        }
    // 根据 HTTP 请求报文的内容获取相应的 uri
    // uri 位于第一行的方法之后 与方法间隔了一个空格 eg. GET /somedir/page.html
    HTTP/1.1
        // 可以采用通过求子串的方法，根据 uri 前后的两个空格来提取 request 中的 uri
        private String parseUri(String requestString) {
            // 声明索引
            int index1, index2;
            index1 = requestString.indexOf(' ');
            if (index1 != -1) {
                index2 = requestString.indexOf(' ', index1 + 1);
                if (index2 > index1)
                    return requestString.substring(index1 + 1, index2);
            }
            return null;
        }
        public void setUri(String uri) {this.uri = uri;}
        public String getUri() {return uri;}
        public Request(InputStream inputStream) {
            this.inputStream = inputStream;
        }
    }
```

**Response.java**

```
public class Response {
    private static final int BUFFER_SIZE = 1024;
    Request request = null;
    OutputStream output = null;
    public Response(OutputStream output) {this.output = output;}
    public void setRequest(Request request) {this.request = request;}
    public void sendStaticResource() throws IOException {
        // 创建字节数组
        byte[] bytes = new byte[BUFFER_SIZE];
        // 文件输入流
        FileInputStream fis = null;
        try {
            //将 uri 对应的要访问的 web 文件写入到 OutputStream 字节流中
            File file = new File(WebServer.WEB_ROOT, request.getUri());
            if (file.exists()) {
                fis = new FileInputStream(file);
                BufferedReader bufferedReader = new BufferedReader(new
InputStreamReader(fis));
                String readData = null;
                System.out.println("响应报文：\r\n");
                while ((readData = bufferedReader.readLine()) != null)  {
                    System.out.println(readData);
                }
 // ch 为读取到的字节数，循环每次读取 1024/8=128 个字节数据到相应的文件输入流中
                int ch = fis.read(bytes, 0, BUFFER_SIZE);
                while (ch != -1) {
                    output.write(bytes, 0, ch);
                    ch = fis.read(bytes, 0, BUFFER_SIZE);
                }
            } else {
        // 如果要访问的文件不存在，则将错误信息对应的响应报文写入到输出流中
       String errorMessage = "HTTP/1.1 404 File Not Found\r\n" + "Content-Type:
text/html\r\n" + "Content-Length: 23\r\n" + "\r\n" + "<h1>File Not Found</h1>";
                   output.write(errorMessage.getBytes());
            }
        } catch (Exception e) {
            System.out.println(e.toString());
        } finally {
            // 最后关闭相应的文件输入流
            if (fis != null)
                fis.close();
        }
    }
```

**WebServer.java**

```
public class WebServer {
    public static final String WEB_ROOT = System.getProperty("user.dir") + File.separator +
"WebContent";
    // 关闭服务命令
    private static final String SHUTDOWN_COMMAND = "/SHUTDOWN";
    private String uri;
    public static void main(String[] args) {
        WebServer server = new WebServer();
        server.await(); // 等待连接请求
    }
    public void await() {
        // 声明服务器端套接字
        ServerSocket serverSocket = null;
        // 设置端口号
        int port = 8080;
        try {
      // 创建服务器套接字对象 ServerSocket(int port, int backlog, InetAddress bindAddr)
        serverSocket = new ServerSocket(port, 1, InetAddress.getByName("127.0.0.1"));
        } catch (IOException e) {
            e.printStackTrace();
            System.exit(1);
        }
        // 循环等待一个请求
        while (true) {
            Socket socket = null;
            InputStream input = null;
            OutputStream output = null;
            try {
  // 等待连接，连接成功后，返回一个 Socket 对象    Socket java.net.ServerSocket.accept()
                socket = serverSocket.accept();
                input = socket.getInputStream();
                output = socket.getOutputStream();
    // 创建 Request 对象并解析相应的请求报文   Request(InputStream inputStream)
                Request request = new Request(input);
                request.parse();
                uri = request.getUri();
                // 检查是否是关闭服务命令
                if(uri != null){
                    if (uri.equals(SHUTDOWN_COMMAND)) {
                        break;
                    }
                }
```

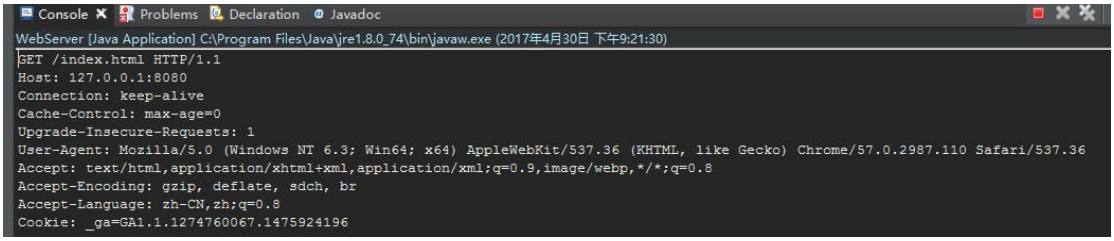
```
                    // 创建 Response 对象
                    Response response = new Response(output);
                    response.setRequest(request);
                    response.sendStaticResource();
                    // 关闭 socket 对象
                    socket.close();
                } catch (Exception e) {
                    e.printStackTrace();
                    continue;
                }
            }
        }
}
```

**程序结果截图：**

① 浏览器发送请求


② 控制台输出

**请求报文**




**响应报文数据段**