



中南大學
CENTRAL SOUTH UNIVERSITY

计算机网络 实验报告

A3: 简单 Web 服务器端程序实现

学 院: 计算机学院

专业班级: _____

学生姓名: R

学 号: _____

指导教师: _____

年 月 日

目录

一、实验目的和要求.....	1
二、实验内容与实现原理.....	2
1. Socket 编程接口.....	2
2. HTTP 传输协议.....	4
三、实验具体设计实现及结果.....	5
UML.....	5
关键代码说明.....	7
运行结果.....	12
四、实验设备与实验环境.....	14
五、实验总结.....	15
六、附录:html 文件.....	15

一、实验目的和要求

目的

本实验要求学生实现一个简单的 Web 服务器端程序，该程序监听 TCP 80 端口，能够接受传入的 HTTP 连接请求并进行解析，并且能够正确的响应请求，回送相关的网页。为简单起见，仅要求正确解析常用的 get 请求，并只需要支持一个 HTTP 连接。

要求

Web 服务器的基本功能是接受并解析客户端的 HTTP 请求，然后从服务器的文件系统获取所请求的文件，生成一个由头部和响应文件内容所构成成的 HTTP 响应消息，并将该响应消息发送给客户端。如果请求的文件不存在于服务器中，则服务器应该向客户端发送“404 Not Found”差错报文。具体的过程和步骤分为：

1. 当一个客户（浏览器）连接时，创建一个连接套接字；
2. 从这个连接套接字接收 HTTP 请求；
3. 解释该请求以确定所请求的特定文件；
4. 从服务器的文件系统获得请求的文件；
5. 创建一个由请求的文件组成的 HTTP 响应报文，报文前面有首部行；
6. 经 TCP 连接向请求浏览器发送响应；
7. 如果浏览器请求一个在该服务器中不存在的文件，服务器应当返回一个“404 Not Found”差错报文。

二、实验内容与实现原理

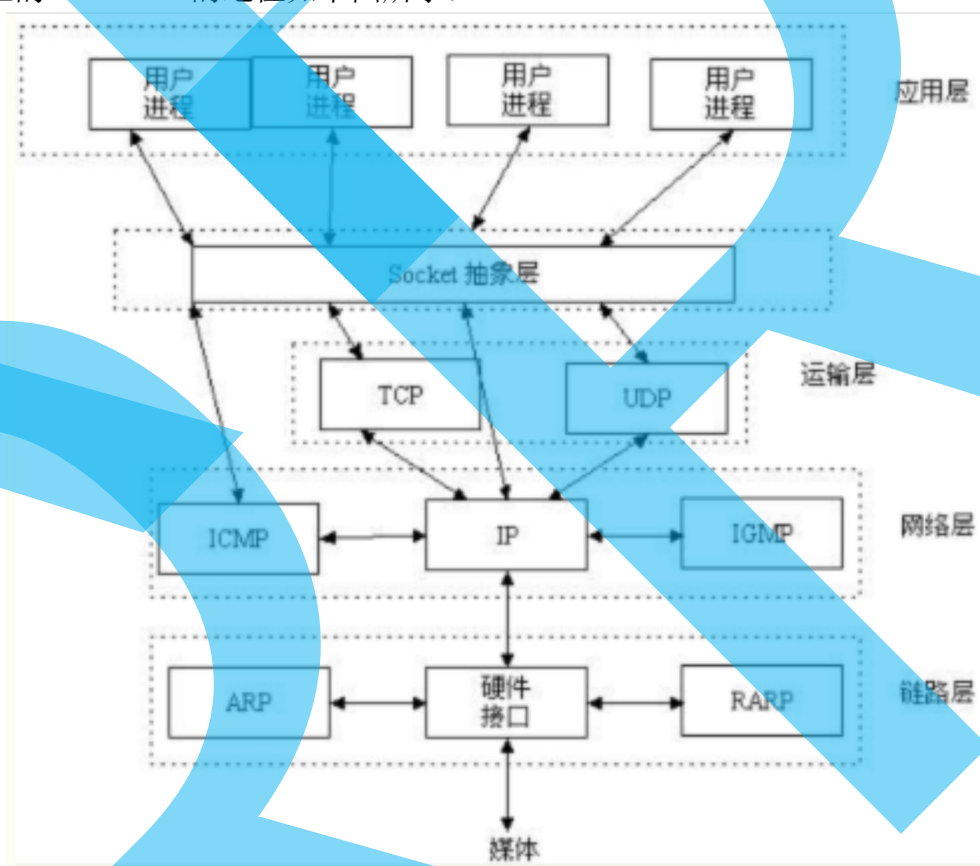
实验内容

首先学习面向 TCP 连接的套接字编程基础知识：如何创建套接字，将其绑定到特定的地址和端口，以及发送和接收数据包。其次还将学习 HTTP 协议格式的相关知识。在此基础上，本实验开发一个简单的 Web 服务器，它仅能处理一个 HTTP 连接请求。

实现原理

1. Socket 编程接口

要实现 Web 服务器，需使用套接字 Socket 编程接口来使用操作系统提供的网络通信功能。Socket 是应用层与 TCP/IP 协议族通信的中间软件抽象层，是一组编程接口。它把复杂的 TCP/IP 协议族隐藏在 Socket 接口后面，对用户来说，一组简单的接口就是全部，让 Socket 去组织数据，以符合指定的协议。使用 Socket 后，无需深入理解 TCP/UDP 协议细节（因为 Socket 已经为我们封装好了），只需要遵循 Socket 的规定去编程，写出的程序自然就是遵循 TCP/UDP 标准的。Socket 的地位如下图所示：

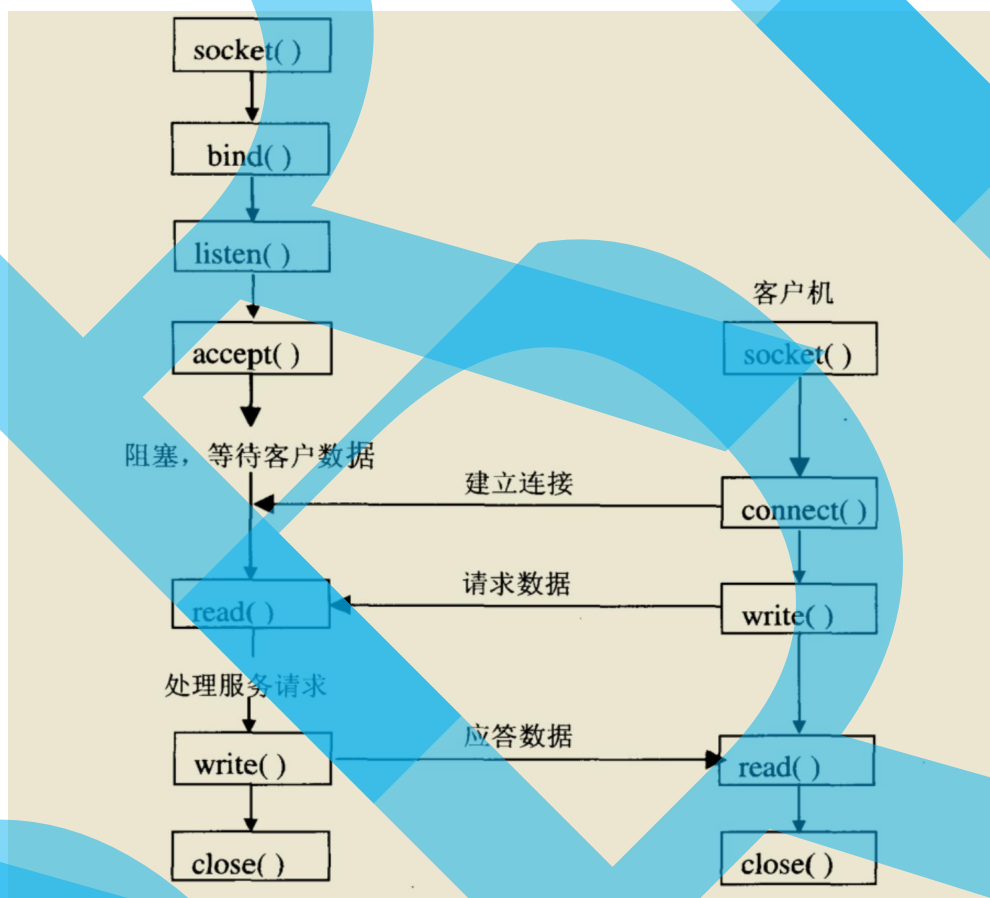


从某种意义上说，Socket 由地址 IP 和端口 Port 构成。IP 是用来标识互联

网中的一台主机的位置,而 Port 是用来标识这台机器上的一个应用程序,IP 地址是配置到网卡上的,而 Port 是应用程序开启的,IP 与 Port 的绑定就标识了互联网中独一无二的一个应用程序。

套接字类型 流式套接字 (SOCK_STREAM): 用于提供面向连接、可靠的数据传输服务。数据报套接字 (SOCK_DGRAM): 提供了一种无连接的服务。该服务并不能保证数据传输的可靠性,数据有可能在传输过程中丢失或出现数据重复,且无法保证顺序地接收到数据。原始套接字 (SOCK_RAW): 主要用于实现自定义协议或底层网络协议。

在本 WEB 服务器程序实验中,采用流式套接字进行通信。其基本模型如下图所示:



其工作过程如下: 服务器首先启动,通过调用 `socket()` 建立一个套接字,然后调用绑定方法 `bind()` 将该套接字和本地网络地址联系在一起,再调用 `listen()` 使套接字做好侦听连接的准备,并设定的连接队列的长度。客户端在建立套接字后,就可调用连接方法 `connect()` 向服务器端提出连接请求。服务器端在监听到连接请求后,建立和该客户端的连接,并放入连接队列中,并通过调用 `accept()` 来返回该连接,以便后面通信使用。客户端和服务器连接一旦建立,就可以通过调用接收方法 `recv()` / `recvfrom()` 和发送方法 `send()` / `sendto()` 来发送和接收数据。最后,待数据传送结束后,双方调用 `close()` 关闭套接字。

2. HTTP 传输协议

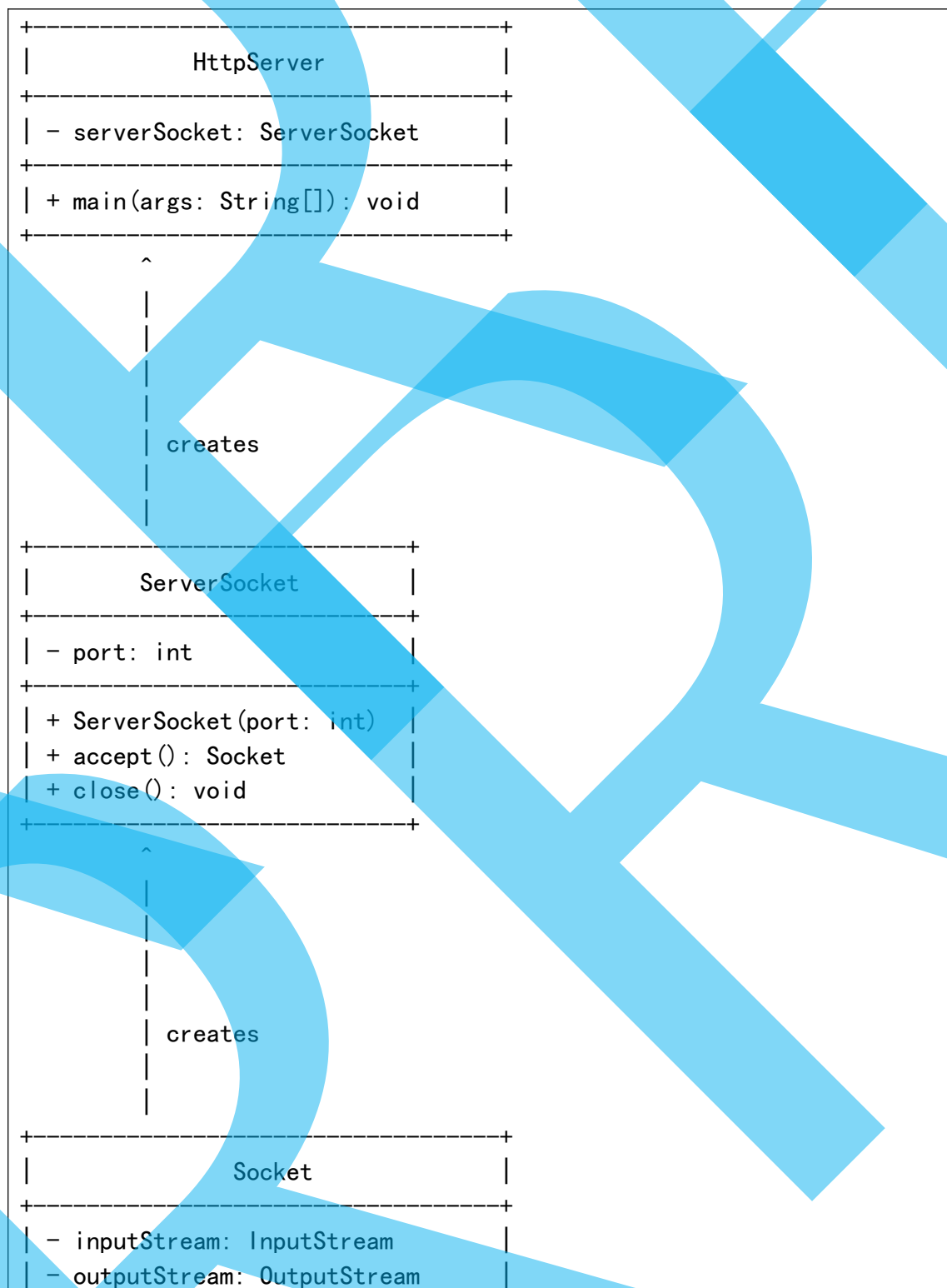
超文本传输协议 (HTTP) 是用于 Web 上进行通信的协议：它定义 Web 浏览器如何从 Web 服务器请求资源以及服务器如何响应。为简单起见，在该实验中将处理 HTTP 协议的 1.0 版。HTTP 通信以事务形式进行，其中事务由客户端向服务器发送请求，然后读取响应组成。请求和响应消息共享一个通用的基本格式：

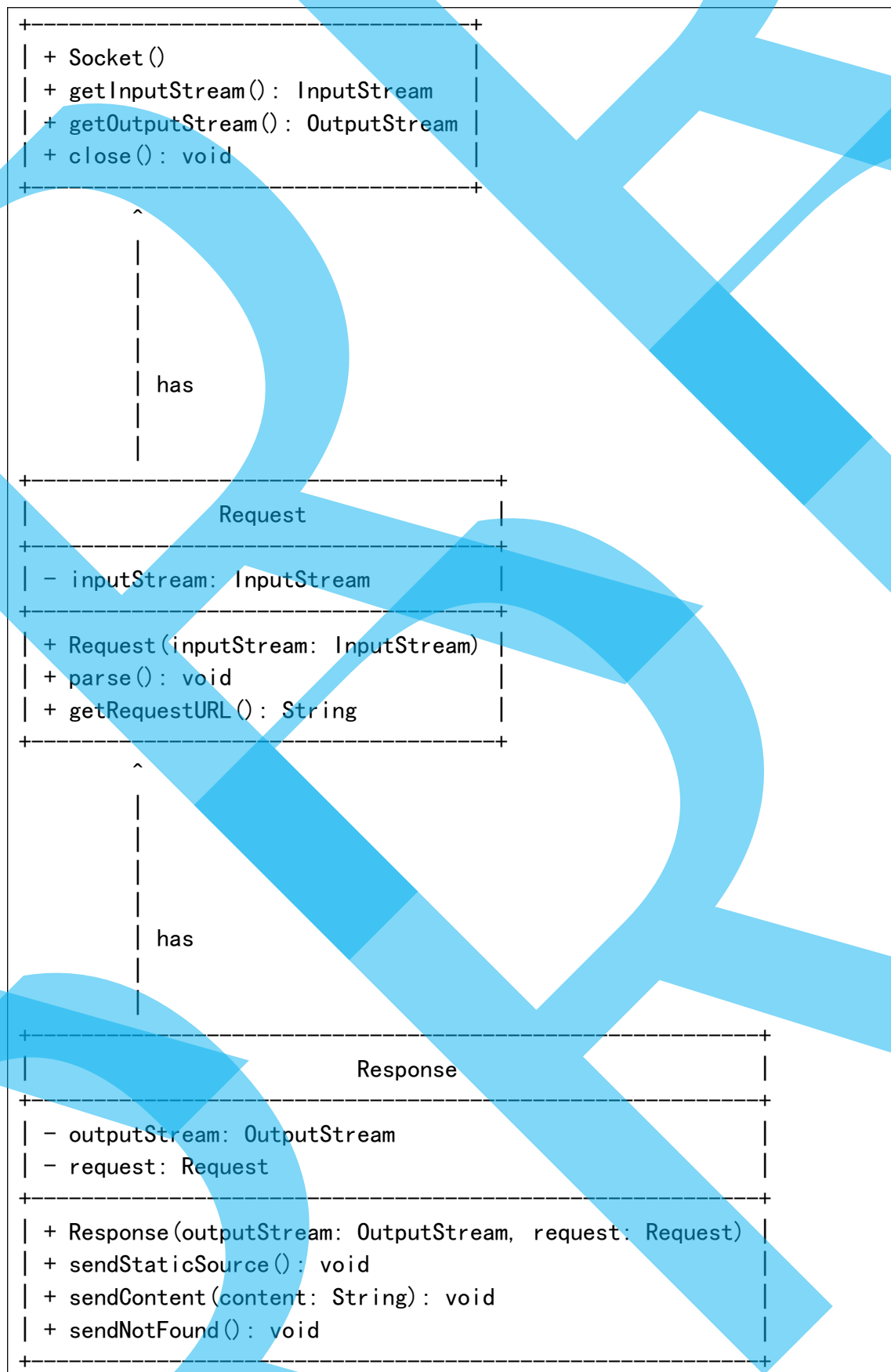
- 初始行（请求或响应行）
- 零个或多个头部行
- 空行（CRLF）
- 可选消息正文。

对于大多数常见的 HTTP 事务，协议归结为一系列相对简单的步骤：首先，客户端创建到服务器的连接；然后客户端通过向服务器发送一行文本来发出请求。这请求行包 HTTP 方法（比如 GET, POST、PUT 等），请求 URI（类似于 URL），以及客户机希望使用的协议版本（比如 HTTP/1.0）；接着，服务器发送响应消息，其初始行由状态线（指示请求是否成功），响应状态码（指示请求是否成功完成的数值），以及推理短语（一种提供状态代码描述的英文消息组成）；最后一旦服务器将响应返回给客户端，它就会关闭连接。

三、实验具体设计实现及结果

UML





关键代码说明

工作原理:

1. 创建服务器套接字并绑定到指定的端口（这里是 8080）。
2. 进入无限循环，接受客户端的连接请求。
3. 当有客户端连接时，创建与该客户端通信的套接字。
4. 获取套接字的输入流和输出流，用于接收客户端发送的 HTTP 请求和向客户端发送 HTTP 响应。
5. 实例化一个 Request 对象，将套接字的输入流传递给它。
6. 实例化一个 Response 对象，将套接字的输出流和 Request 对象传递给它。调用 Request 对象的 `parse()` 方法解析 HTTP 请求报文，提取请求的 URI（请求的文件名）。
7. 根据请求的 URI 判断请求的文件是否存在。
8. 如果文件存在，发送 HTTP 响应报文，报文中包含 200 OK 的状态码和文件内容。
9. 如果文件不存在，发送 HTTP 响应报文，报文中包含 404 File Not Found 的状态码和错误信息。
10. 关闭套接字。
11. 循环回到第 3 步，等待下一个客户端连接。

简要地说，该程序通过监听指定端口，接受客户端连接，并根据客户端发送的 HTTP 请求来发送相应的 HTTP 响应，实现了一个简单的静态文件服务器。

HttpServer.java

```
import java.net.ServerSocket;
import java.net.Socket;
import java.io.IOException;
import java.io.InputStream;
import java.io.OutputStream;

public class HttpServer {
    public static void main(String[] args) {
        ServerSocket serverSocket = null;
        try {
            // 创建服务器套接字
            serverSocket = new ServerSocket(8080);
        } catch (IOException ioe) {
            ioe.printStackTrace();
            System.exit(0);
        }
    }
}
```

```

Socket socket = null;
InputStream inputStream = null;
OutputStream outputStream = null;
while (true) {
    try {
        // 接受套接字请求并创建相应的服务器套接字来启动监听
        socket = serverSocket.accept();
        // 获取输入流
        inputStream = socket.getInputStream();
        // 获取输出流
        outputStream = socket.getOutputStream();
        // 实例化请求类
        Request request = new Request(inputStream);
        // 实例化应答类
        Response response = new Response(outputStream,
request);

        // 解析请求
        request.parse();
        // 根据请求做响应应答
        response.sendStaticSource();
        // 关闭套接字
        socket.close();
    } catch (IOException ioe) {
        ioe.printStackTrace();
    }
}
}

```

这是一个简单的 HTTP 服务器，它使用了 Java 的 Socket 编程来接受客户端请求并发送响应。主要包含以下几个部分：

1. **ServerSocket**：通过创建 **ServerSocket** 对象来监听指定端口（这里是 8080），等待客户端的连接请求。
2. **Socket**：一旦有客户端连接请求到达，**ServerSocket** 将会接受该请求并创建一个新的 **Socket** 对象，用于与客户端进行通信。
3. **输入流和输出流**：通过获取 **Socket** 对象的输入流和输出流，服务器可以接收来自客户端的请求数据，并向客户端发送响应数据。
4. **Request 类**：该类负责解析客户端请求的输入流数据，并提供相应的方法来获取请求的相关信息，如请求 URL。

5. Response 类：该类负责构建服务器的响应，并发送给客户端。它接收 Request 对象作为参数，可以根据请求的内容生成相应的响应。

6. 主函数：主函数中的循环部分用于持续监听客户端的连接请求。每当有新的连接请求到达时，就创建一个新的 Socket 对象，并使用它来处理客户端的请求和响应。

我这个简单的 HTTP 服务器是单线程的，它在一个循环中接受连接、处理请求和发送响应。当有新的连接到达时，服务器会创建一个新的 Socket 对象，处理完请求后关闭该连接，然后再继续监听下一个连接请求。

Request.java

```
import java.io.InputStream;
import java.io.IOException;
import java.io.InputStreamReader;
import java.io.BufferedReader;

public class Request {
    private InputStream inputStream = null;
    private String uri = null;

    public Request(InputStream inputStream) {
        // 获取输入流
        this.inputStream = inputStream;
    }

    // 解析请求报文
    public void parse() throws IOException {
        BufferedReader br = new BufferedReader(new
        InputStreamReader(inputStream));
        String msg = br.readLine();
        // 在终端打印报文
        System.out.println(msg);
        // 解析请求文件名
        this.uri = msg.split("/") [1].substring(1);
        while ((msg = br.readLine()) != null) {
            System.out.println(msg);
            if (msg.length() == 0) {
                break;
            }
        }
    }
}
```

```

    }
    // 获取请求文件名
    public String getUri() {
        return this.uri;
    }
}

```

Request 类用于解析 HTTP 请求报文并提取请求的文件名 (URI)。主要包含以下几个部分:

1. parse() 方法: 该方法通过传入一个输入流 (InputStream), 使用 BufferedReader 逐行读取请求报文的内容。首先, 它读取报文的第一行, 即请求行 (如 "GET /index.html HTTP/1.1")。然后, 它将请求行拆分, 并提取第二部分作为请求的文件名 (URI), 通过去除开头的斜杠来获取最终的文件名。接下来, 它继续读取报文的头部字段 (header fields), 并在终端打印每一行。当读取到一个空行时, 表示头部字段的结束, 解析过程结束。

2. getUri() 方法: 该方法返回解析后得到的请求文件名 (URI)。

这个 Request 类的作用是解析 HTTP 请求报文, 提取出请求的文件名。它通过读取输入流中的报文内容, 并按照 HTTP 协议的格式进行解析。然后, 可以调用 getUri() 方法获取解析得到的请求文件名。

Response.java

```

import java.io.OutputStream;
import java.io.InputStream;
import java.io.BufferedOutputStream;
import java.io.File;
import java.io.FileInputStream;
import java.io.FileNotFoundException;
import java.io.IOException;

public class Response {
    private OutputStream outputStream;
    private Request request;

    public Response(OutputStream outputStream, Request request) {
        this.outputStream = outputStream;
        this.request = request;
    }
}

```

```

    public void sendStaticSource() {
        BufferedOutputStream bos = new
BufferedOutputStream(outputStream);
        BufferedInputStream bis = null;
        try {
            // 如果请求的文件存在
            if (new File(this.request.getUri()).exists()) {
                try {
                    // 获取文件输入流
                    bis = new BufferedInputStream(new
FileInputStream(this.request.getUri()));
                } catch (FileNotFoundException fnfe) {
                }
                byte[] data = new byte[1024];
                int length = 0;
                // 发送可以发送文件的报文
                String msg = "HTTP/1.1 200 OK \r\n" +
                    "content-type: text/html; charset=utf-8
\r\n\r\n";

                // 发送文件数据
                bos.write(msg.getBytes());
                while ((length = bis.read(data)) != -1) {
                    bos.write(data, 0, length);
                }
            } else {
                // 发送未找到文件报文
                String errmsg = "HTTP/1.1 404 File Not Found\r\n" +
                    "Content-Type:text/html\r\n" +
                    "Content-Length:23\r\n" +
                    "\r\n" +
                    "<h2>File Not Found</h2>";
                bos.write(errormsg.getBytes());
            }
            // 关闭输出流
            bos.close();
            // 关闭输入流
            if (bis != null) {
                bis.close();
            }
        } catch (IOException ioe) {
            ioe.printStackTrace();
        }
    }
}

```

```
}
```

Response 类用于处理服务器的响应并发送给客户端。主要包含以下几个部分：

1. 构造函数：通过传入一个输出流（outputStream）和 Request 对象（request），初始化 Response 类的实例。

2. sendStaticSource() 方法：该方法用于发送静态资源的响应。首先，它创建了一个 BufferedOutputStream 对象，用于将响应数据写入到输出流中。然后，它检查请求的文件是否存在，如果存在，则尝试创建一个 BufferedInputStream 对象来读取该文件的内容。接下来，它构建响应报文，并写入输出流中，包括一个成功的响应头部以及响应内容（文件数据）。如果文件不存在，则构建一个包含“404 File Not Found”的错误响应报文。最后，关闭输出流和输入流。

这个 Response 类的作用是**根据请求的文件名，发送对应的响应给客户端**。它根据请求的文件是否存在来决定发送成功的响应还是错误的响应。对于存在的文件，它会读取文件的内容并发送给客户端；对于不存在的文件，它会发送一个包含错误信息的响应报文。

运行结果



图 1: 谷歌浏览器请求 index.html

图 2: 浏览器请求报文

图 2: 谷歌浏览器请求不存在的文件

四、实验设备与实验环境

操作系统

Windows 11

开发工具

IntelliJ IDEA Community Edition 2023.1.2

编程语言

Java

五、实验总结

六、附录：html 文件

index.html

```
<html>

<head>
  <link href="favicon.ico" rel="shortcut icon">
  <meta content="text/html" charset="UTF-8">
  <title>实验服务器</title>
</head>

<body style="background: linear-gradient(135deg, #f7347a, #ffffff,
#45a3e5);
      background-size: 600% 600%;
      animation: gradientAnimation 10s ease infinite;
      text-align: center; position: relative;">

  <div style="position: relative; margin-top: 20px;">
    <h1 style="font-weight: bold; font-size: 72px; color: #fff;
      font-family: 'Quicksand', sans-serif;
      text-shadow: 0 0 10px rgba(255, 255, 255, 0.6);">
      实验测试界面
    </h1>
  </div>

  <div style="position: absolute; bottom: 20px; right: 20px;">
    <p style="font-size: 24px; color: #fff; font-family:
'Quicksand', sans-serif;
      text-shadow: 0 0 6px rgba(255, 255, 255, 0.8);">
      R
    </p>
  </div>

  <script>
    function gradientAnimation() {
      var body = document.querySelector('body');
      var currentPosition = 0;
      var gradientInterval = setInterval(function() {
        currentPosition += 1;
        body.style.backgroundPosition =
currentPosition + '% 50%';
        if (currentPosition >= 100) {
```

```
        currentPosition = 0;
    }
    }, 50);
}
gradientAnimation();
</script>
</body>
</html>
```