**实 验 报 告**

（四学时）

|  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- |
| **课 程** | 计算机网络基础 | **实验项目** | Socket程序设计 | **成 绩** |  |
| **专业班级** | **计算机科学与技术2202** | **学 号** | **202231060435** | **批阅日期** |  |
| **姓 名** | **王磊** | **实验日期** | 2023/12/28 | **指导教师** | 张恒汝 |

**一【实验目的】**

1. 掌握基于多线程的复杂网络程序的开发方法和开发流程；
2. 按照要求设计实现软件，以培养应用复杂计算系统开发的能力，并在设计和开发过程中体现出创新意识。

**二【实验内容】**

1-1 假设Tom和Jerry利用Java UDP进行聊天，请为他们编写程序。具体如下：

（1）、Tom和Jerry聊天的双方都应该具有发送端和接收端；

（2）、利用DatagramSocket与DatagramPacket；

（3）、实现 java.lang.Runnable类，重写 run()方法。

1-2 利用Java Socket编写一个简单的Web服务器，具体如下：

（1）、使用 ServerSocket 监听某一端口，然后等待连接获取 Socket对象；

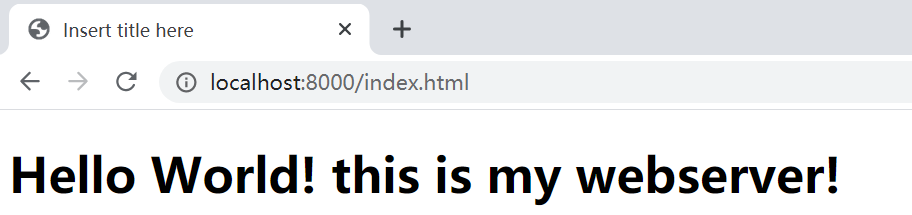
（2）、创建一个类 HttpServer 继承 java.lang.Thread 类，重写 run()方法，执行浏览器请求；

（3）、获得浏览器请求，解析资源文件路径；

（4）、读取资源文件，响应给浏览器；

（5）、浏览器地址栏输入： <http://localhost:8000/index.html>；

（6）、效果如下：



**三【实验步骤】**

1. 设计思路

通过流程图或者伪代码描述程序设计思路

设计思路：

1. Tom和Jerry聊天程序

Tom的发送端

+ 创建一个DatagramSocket对象，指定端口号和IP地址。

+ 创建一个DatagramPacket对象，设置发送的数据内容和目标IP和端口。

+ 使用DatagramSocket对象的send()方法发送数据。

Tom的接收端

+ 创建一个DatagramSocket对象，指定端口号。

+ 创建接收数据的DatagramPacket对象。

+ 使用DatagramSocket对象的receive()方法接收数据。

Jerry的发送和接收端

+ 与Tom类似，但需要设置不同的端口号。

线程处理

+ 为Tom和Jerry的发送和接收端分别创建一个线程，实现Runnable接口并重写run()方法。

2. 简单的Web服务器

监听端口

+ 使用ServerSocket监听8000端口。

处理连接请求

+ 当有客户端（如浏览器）请求连接时，获取Socket对象。

创建线程处理请求

+ 创建一个HttpServer类，继承Thread类，并重写run()方法。

解析资源文件路径

+ 从请求中获取资源文件路径。

读取并响应资源文件

+ 根据资源文件路径读取文件内容。

+ 将文件内容作为响应发送给客户端。

3. 浏览器请求

在浏览器地址栏输入：<http://localhost:8000/Web/index.html>

1. 实现过程

给出主要步骤

Tom和Jerry聊天程序实现步骤

1. 创建Tom的发送端

导入java.net包中的DatagramSocket和DatagramPacket类。

创建一个DatagramSocket对象，并指定端口号和IP地址。

创建一个DatagramPacket对象，设置要发送的数据内容和目标IP和端口。

使用DatagramSocket对象的send()方法发送数据。

2. 创建Tom的接收端

创建一个DatagramSocket对象，并指定端口号。

创建接收数据的DatagramPacket对象。

使用DatagramSocket对象的receive()方法接收数据。

3. 创建Jerry的发送和接收端

与Tom类似，但需要设置不同的端口号。

4. 线程处理

为Tom和Jerry的发送和接收端分别创建一个线程，实现Runnable接口并重写run()方法。这样可以使聊天在单独的线程中运行，不阻塞主线程。

5. 测试

在Tom和Jerry各自的Java虚拟机中运行上述代码。

确保Tom和Jerry可以相互发送和接收消息。

简单Web服务器实现步骤

1. 监听端口

导入java.net包中的ServerSocket类。

使用ServerSocket类监听8000端口。

2. 处理连接请求

当有客户端（如浏览器）请求连接时，获取Socket对象。

3. 创建线程处理请求

创建一个HttpServer类，继承Thread类，并重写run()方法。在run()方法中处理HTTP请求。

4. 解析资源文件路径

从请求中获取资源文件路径。这通常从URL中获取。

5. 读取并响应资源文件

根据资源文件路径读取文件内容。可以使用java.io包中的类如FileReader或BufferedReader等。

将文件内容作为响应发送给客户端。响应需要遵循HTTP协议的格式，包括状态码、响应头和响应体。

6. 测试

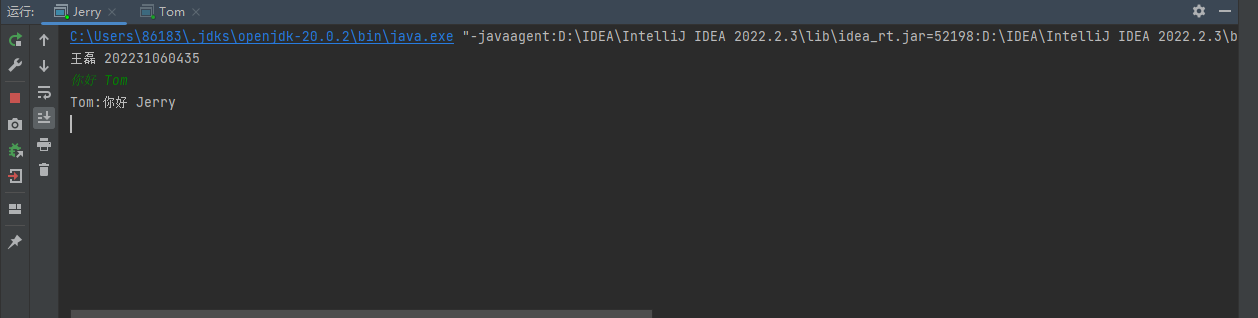
在命令行或终端中启动服务器。例如：java HttpServer。

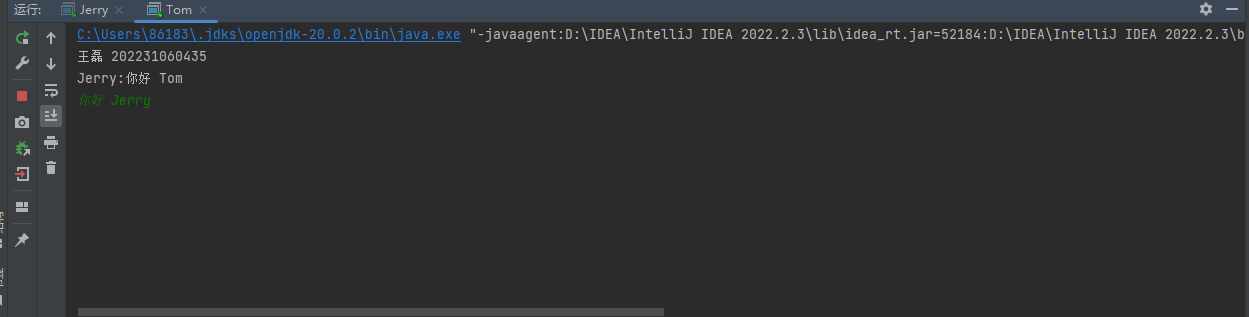
在浏览器中输入http://localhost:8000/index.html进行测试，确保服务器能够正确响应请求并显示页面内容。

**四【实验结果】**

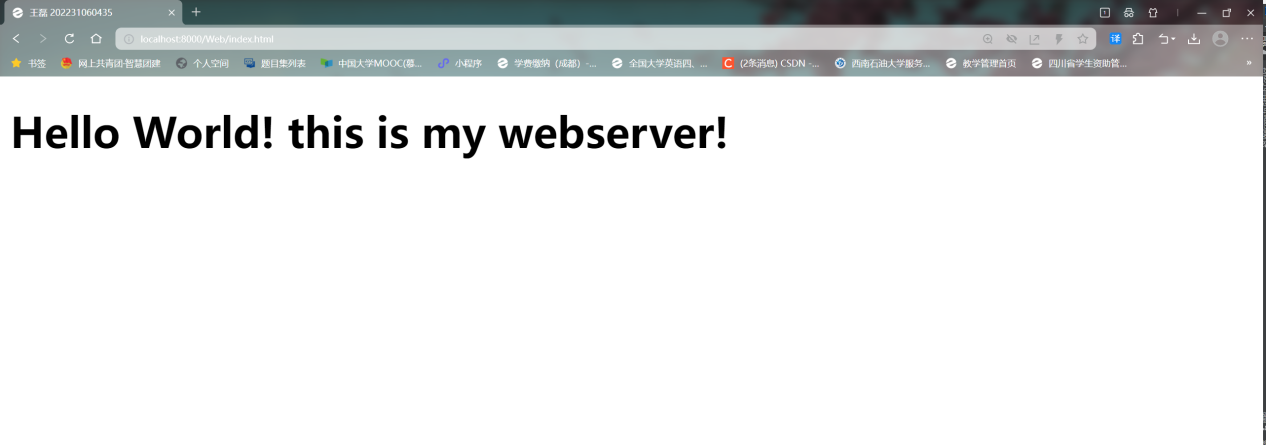
主要运行结果及说明

实验一：





实验二：



**五【总结】**

可以包括：

1. 问题及解决方法
2. UDP、TCP协议对比
3. 服务器编程要点

总结：

实验通过实现Tom和Jerry的聊天程序以及一个简单的Web服务器，对网络编程中的UDP和TCP协议、服务器编程等知识点进行了实践。

在Tom和Jerry的聊天程序中，使用了UDP协议进行通信。为了实现这一功能，我们首先创建了DatagramSocket对象来发送和接收数据包，并设置了目标IP和端口。然后，使用DatagramPacket对象来封装发送的数据内容。发送端和接收端的代码逻辑清晰，并且通过线程处理使聊天在单独的线程中运行，确保主线程不会被阻塞。最后，测试结果表明Tom和Jerry可以成功地相互发送和接收消息。

在简单Web服务器的实现中，我们使用了TCP协议。首先，使用ServerSocket监听8000端口以等待客户端的连接请求。当有请求时，获取Socket对象，并创建一个新的线程来处理该请求。接着，解析资源文件路径并读取相应的文件内容，然后将文件内容作为HTTP响应发送给客户端。测试结果表明，服务器能够正确响应请求并显示页面内容。

通过本次实验，我们深入了解了UDP和TCP协议的特点和应用场景。UDP协议适用于需要快速传输少量数据的场景，而TCP协议则适用于需要可靠、有序传输大量数据的场景。此外，我们还掌握了服务器编程的基本要点，如监听端口、处理连接请求、创建线程等。

在实验过程中，我们遇到了一些问题，如数据包丢失、连接超时等。针对这些问题，我们采取了相应的解决方法，如增加重试机制、优化代码逻辑等。

总之，本次实验加深了我们对网络编程的理解和实践能力。通过实际操作，我们掌握了UDP和TCP协议的应用、服务器编程的基本要点以及解决实际问题的能力。这将为我们后续的学习和工作打下坚实的基础。