基于 Java Socket 的多线程 HTTP 服务器实现

黄泽熙

（电子科技大学　英才实验学院　611731）

摘　要: 本文介绍了一个基于 Java Socket 的 HTTP 服务器的实现。该服务器支持 html、jpg、ico、css、js 与 pdf 文件响应报文的处理，从而保证了正常静态网页的传输，适用于诸如个人主页等网站平台的使用。特别地，利用 Java 中的 Thread 类，本服务器支持多线程同时处理，确保多用户同时浏览服务器内容时的正确响应。最后，本文通过一个个人主页的案例，展示了该服务器的实际可应用性。

关键词： Java Socket，HTTP 服务器，多线程

An Implementation of Multi-thread Capable HTTP Server Based on Java Socket

Zexi Huang

（Yingcai Experimental School　University of Electronic Science and Technology of China 611731）

Abstract: In this article, we introduce an easy implementation of HTTP server based on Java Socket. This server is capable of handling files of html, jpg, ico, css, js and pdf types, thus enabling the normal response of nearly any static websites, which is especially suitable for uses like personal homepage. In addition, taking advantage of the Thread class in Java, this server also supports multi-thread processing, providing correct responses when multiple users send queries concurrently. Finally, we demonstrate the applicability of our server by showing how it works properly for a personal homepage case.

Key words： Java Socket, HTTP server, Multi-thread

1. 简介

HTTP 协议作为应用层最为重要的协议之一，是现在互联网用户上网冲浪的重要途径。HTTP 服务器的实现既是现代互联网存在之必需，对于网络工程学习者而言又是一个很好的深入理解 HTTP 协议以及应用层其他协议和 Socket 编程的手段。本文介绍利用 Java Socket 来实现一个支持多线程的 HTTP 服务器。

本文的组织结构如下：第 1 节简单介绍本文内容；第 2 节详细介绍利用 Java Socket 实现多线程 HTTP 服务器；第 3 节展示一个该 HTTP 服务器应用的案例——一个简单的个人主页；最后在第 4 节中作出总结。

2. 具体实现

2.1. 整体架构

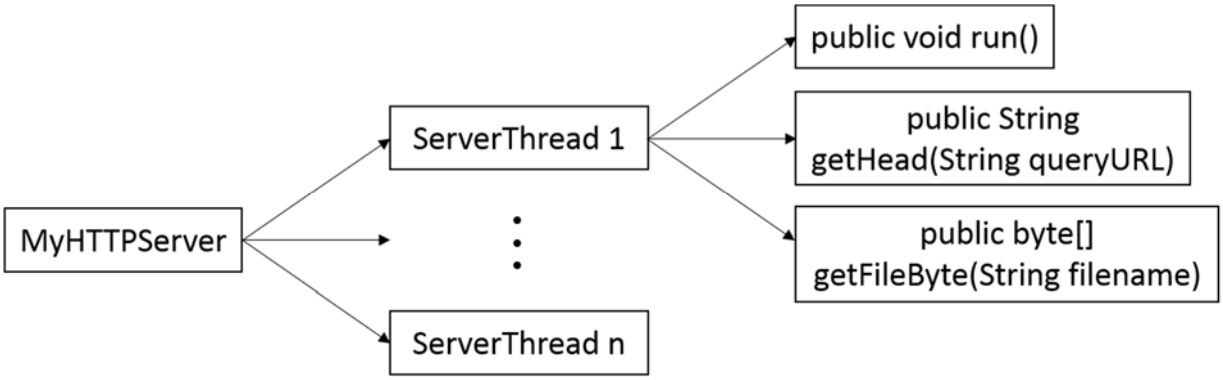

图 1 HTTP 服务器整体架构图

本文介绍的 HTTP 服务器整体架构图如图 1 所示。在 MyHTTPServer 类中为所有监听到的响应提供一个单独的 ServerThread 进行处理，ServerThread 类则定义了 run、getHead 和 getFileByte 三个方法，分别进行解析请求报文并响应，计算响应报文头与读取本地文件的功能。

2.2. MyHTTPServer

MyHTTPServer 类用于开启服务器的监听端口，并为每一个用户 Socket 分配一个线程进行响应处理，其实现代码如下：

```
/**
 * Java Implementation of a basic HTTP server.
 * Only Get method is implemented.
 * @author Eitima (Zexi Huang)
 * @version 1.0 Oct 16, 2016
 * Contact me via Eitima@163.com
```

```
 */
public class MyHTTPServer
{
    /**
     * Main
     * @param args
     * @throws IOException
     */
    public static void main(String[] args) throws IOException
    {
        final int port =80;//set default port 80.
        new MyHTTPServer().start(port);
    }
    /**
     * Create Server Socket and wait for clients.
     * @param port
     * @throws IOException
     */
    public void start(int port) throws IOException
    {
        ServerSocket welcomingSocket=new ServerSocket(port);
        while(true)
        {
            Socket connectionSocket=welcomingSocket.accept();
            ServerThread serverThread=new ServerThread(connectionSocket);
            serverThread.start();
        }
    }
}
```

其中，端口号由 port 变量指定，默认为 80。welcomingSocket 为用于监听客户请求的 Socket，并将客户的 connectionSocket 用于创建一个新进程进行处理。

2.3. ServerThread

ServerThread 为处理每个用户请求的线程，其实现代码如下：

```
/**
 * Thread to handle each query concurrently.
 * @author Eitima (Zexi Huang)
 * @version 1.0 Oct 16, 2016
 */
public class ServerThread extends Thread
{
    Socket client;
    public ServerThread (Socket connectionSocket)
    {
        client=connectionSocket;
```

```
    }
    public void run()
    {
        //...
    }
    public String getHead(String queryURL)
    {
        //...
    }
    public byte[] getFileByte(String filename) throws IOException
    {
        //...
    }
}
```

其中，构造体 ServerThread 用于将对应用户的 Socket 关联本线程.

2.4. public void run()

run()方法用于实现主要的 HTTP 响应功能，包括解析请求报文头以及生成响应报文。其中，解析请求报文头的实现如下：

```
BufferedReader inFromClient=new BufferedReader(new
InputStreamReader(client.getInputStream()));
client.setSoTimeout(60);//Set timeout to 60s.
String method="";
String queryURL="";
int state=0;
while(true)
{
    char nextChar=(char)inFromClient.read();
    boolean isSpace=Character.isWhitespace(nextChar);
    switch(state)
    {
        case 0://Skip invalid whitespaces before the method.
            if(isSpace)
                continue;
            else
                state=1;
        case 1://Read the method.
            if(isSpace)
            {
                state=2;
                continue;
            }
            else
```

```
                {
                    method=method+nextChar;
                    continue;
                }
            case 2://Skip invalid whitespaces between method and URL.
                if(isSpace)
                    continue;
                else
                    state=3;
            case 3://Read URL.
                if(isSpace)
                    break;
                else
                {
                    queryURL=queryURL+nextChar;
                    continue;
                }
        }
        break;
    }
```

其中，字符串 method 与 queryURL 分别用于存放解析出的请求报文的方法段与 URL 段。根据 HTTP 报文头利用空格分割逐个字段的特点，while 循环逐一遍历用户请求报文中的每一个字母，根据当前空格的位置状态（即 state 变量值）判断目前所处的字段，从而提取出请求报文中的 method 字段和 queryURL 字段。

生成响应部分的实现如下：

```
String head=getHead(queryURL);
OutputStream outFromServer=client.getOutputStream();
byte[] data=null;
//Generate the response message.
while(true)
{
    try
    {
        if(inFromClient.read()<0)
        break;
    }
    catch(InterruptedIOException e)
    {
        data=getFileByte("sources"+queryURL);
    }
    if(data!=null&&head!=null)//Object requested existed and legal to be
    transfered.
    {
        outFromServer.write(head.getBytes("utf-8"));
```

```
                outFromServer.write(data);
                outFromServer.close();
                break;
            }
        }
```

其中，字符串 head 利用 getHead()方法，获得所需传送的文件格式，从而生成正确的响应报文首部行。之后在完成对请求报文的解析后，利用 getFileByte()方法，从本地资源文件夹 sources 中读取请求所需的文件。最后将首部行以及文件中的数据封装成报文发送回客户端。

2.5. public String getHead(String queryURL)

getHead()方法实现了对 queryURL 的解析，代码如下：

```
    /**
     * Generate the head of the response message according to the filetype specified
by query URL.
     * @param queryURL
     * @return head of response message
     */
    public String getHead(String queryURL)
    {
        String filename="";
        int index=queryURL.lastIndexOf("/");
        filename=queryURL.substring(index+1);
        String[] filetypes=filename.split("\\.");
        String filetype=filetypes[filetypes.length-1];
        if(filetype.equals("html"))
        {
            return "HTTP/1.0 200 OK\n"+"Content-Type:text/html\n" + "Server:myserver\n"
            + "\n";
        }
        Else
        if(filetype.equals("jpg")||filetype.equals("gif")||filetype.equals("png"))
        {
            return "HTTP/1.0 200 OK\n"+"Content-Type:image/jpeg\n" +
            "Server:myserver\n" + "\n";
        }
        else if(filetype.equals("ico"))
        {
            return "HTTP/1.0 200 OK\n"+"Content-Type:image/x-icon\n" +
            "Server:myserver\n" + "\n";
        }
        else if(filetype.equals("css"))
        {
```

```
            return "HTTP/1.0 200 OK\n"+"Content-Type:text/css\n" + "Server:myserver\n"
            + "\n";
        }
        else if(filetype.equals("js"))
        {
            return "HTTP/1.0 200 OK\n"+"Content-Type:application/x-javascript\n" +
            "Server:myserver\n" + "\n";
        }
        else if(filetype.equals("pdf"))
        {
            return "HTTP/1.0 200 OK\n"+"Content-Type:application/pdf\n" +
            "Server:myserver\n" + "\n";
        }
        else return null;
    }
```

该方法提取 URL 最后的文件名部分的后缀名，并对典型的后缀名，如 html、jpg、ico、css、js 和 pdf 生成了对应的 HTTP 报文头。

2.6. public byte[] getFileByte(String filename)

getFileByte()方法根据所解析出的文件名读取本地文件，并返回为字节串方便封装进报文中，其实现如下：

```
    /**
     * Return the file specified by filename by bytes.
     * @param filename
     * @return bytes of file
     * @throws IOException
     */
    public byte[] getFileByte(String filename) throws IOException
    {
        ByteArrayOutputStream byteArrayOutputStream=new ByteArrayOutputStream();
        File file=new File(filename);
        FileInputStream fileInputStream=new FileInputStream(file);
        byte[] b=new byte[1024];
        int read;
        while((read=fileInputStream.read(b))!=-1)
        {
            byteArrayOutputStream.write(b,0,read);
        }
        fileInputStream.close();
        byteArrayOutputStream.close();
        return byteArrayOutputStream.toByteArray();
    }
```

3. 案例应用

3.1. 前端架构

案例为一个人主页，其前端架构如图 2 所示。

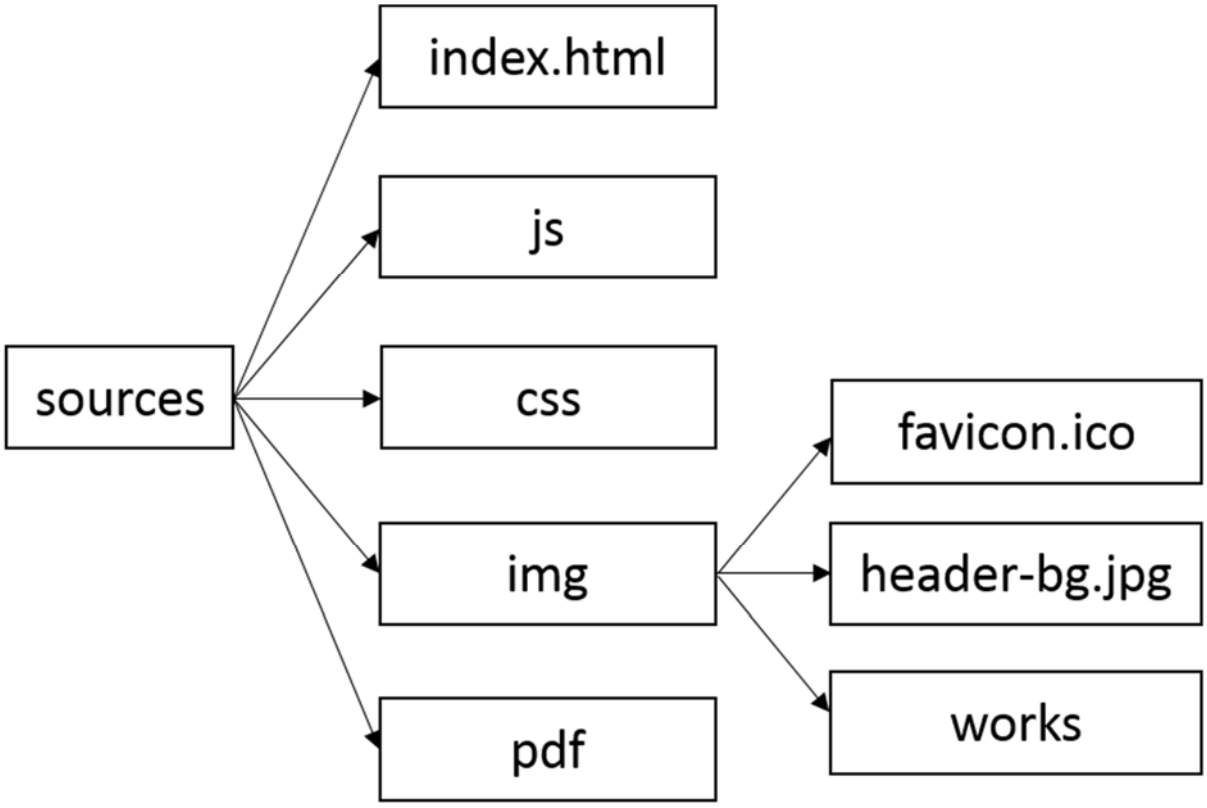

图 2 案例的个人主页前端架构图

案例为一个人主页，其前端架构如图 2 所示。其中，index.html 为主页文件，js 文件夹存放网页需要用到的 JavaScript 脚本，css 文件夹存放网页所需要的层叠样式表，img 文件夹存放包括网站图标 favicon.ico 与网站背景图片 header-bg.jpg 在内的网页所需的资源图片，pdf 文件夹中存放可供用户下载的包括个人简历在内的文件。

3.2. 测试结果

在 JVM 中编译运行对应的 java 文件后，在 Mozilla Firefox 49.0.1.6109 浏览器中输入地址 localhost/index.html，即可加载得到对应的个人主页。主页效果图如图 3、4、5、6、7 所示。

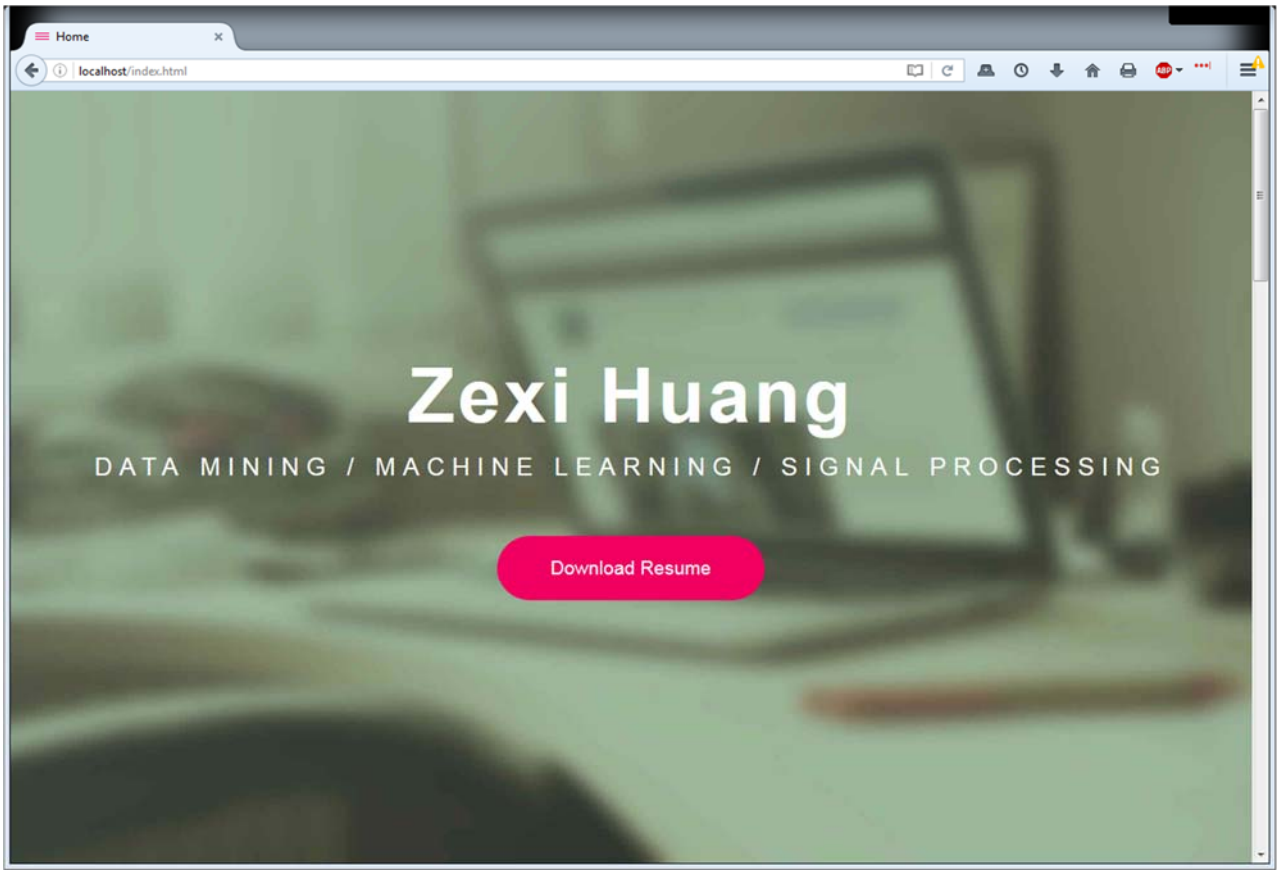

图 3 个人主页界面效果图(a)

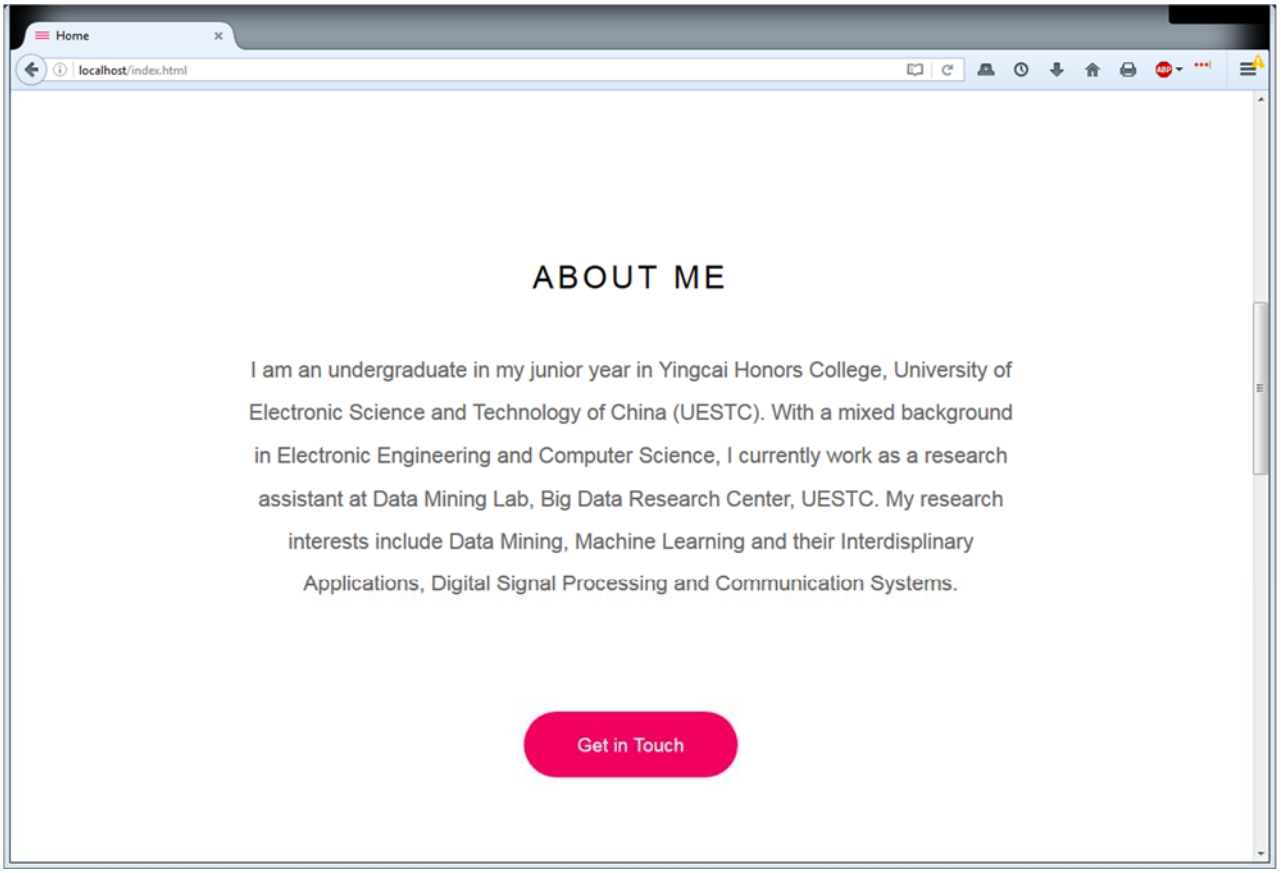

图 4 个人主页界面效果图(b)

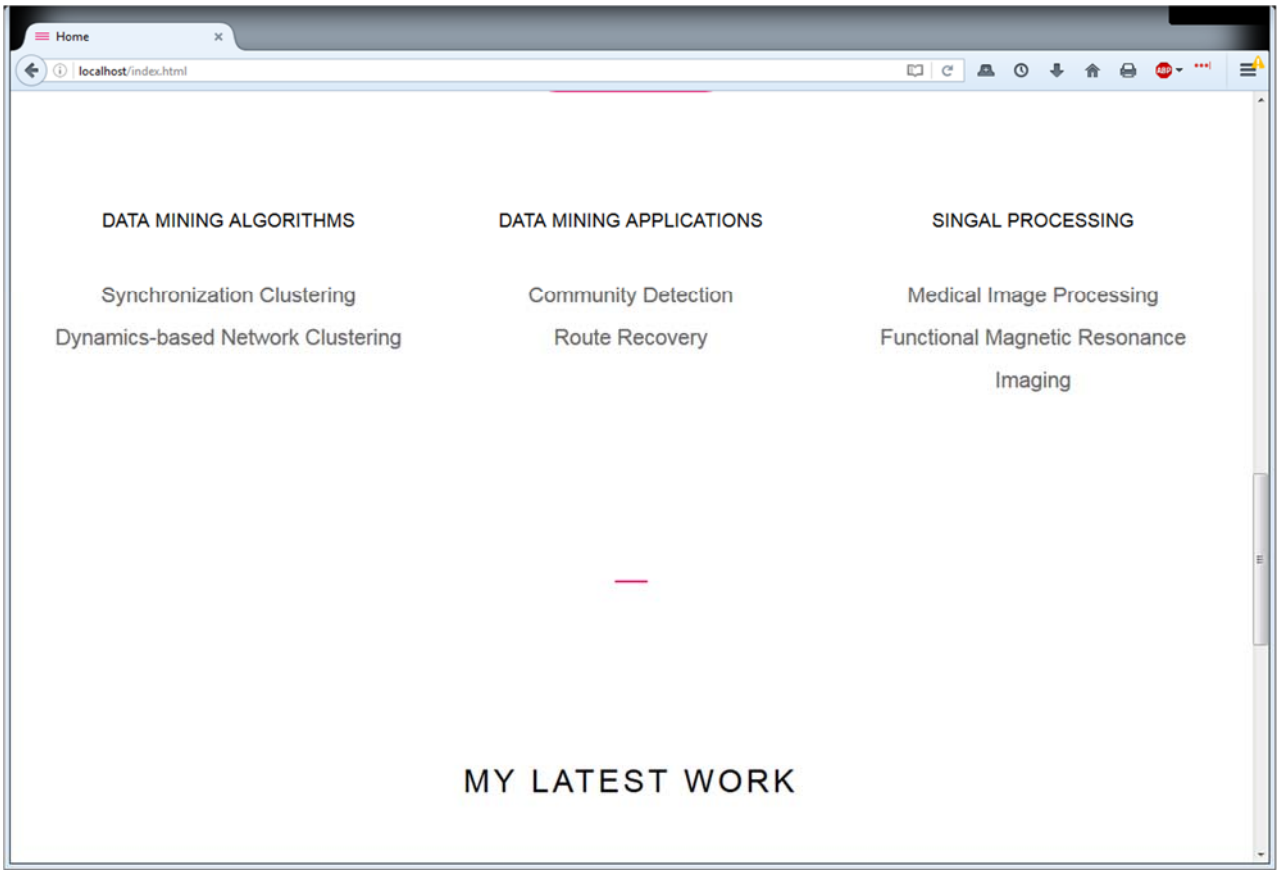

图 5 个人主页界面效果图(c)

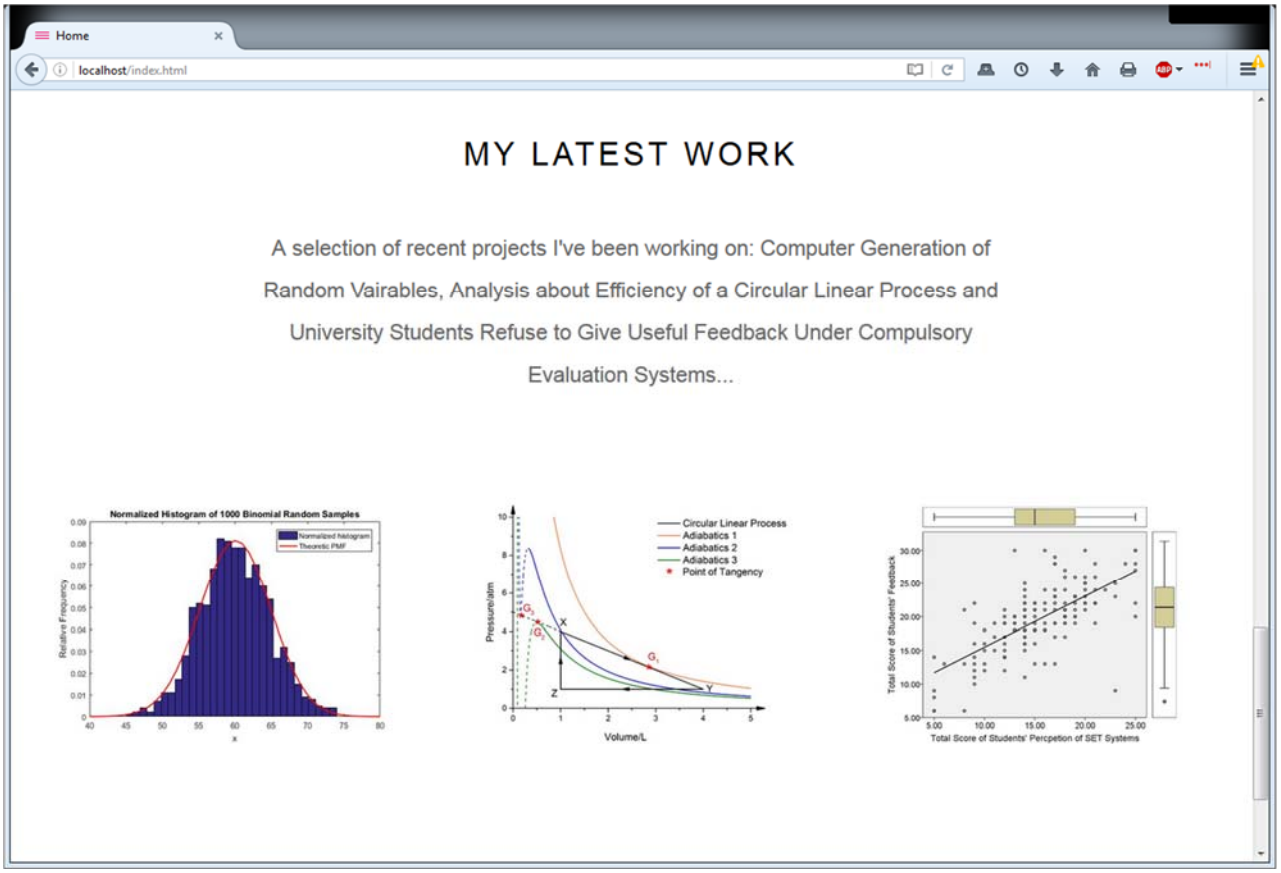

图 6 个人主页界面效果图(d)

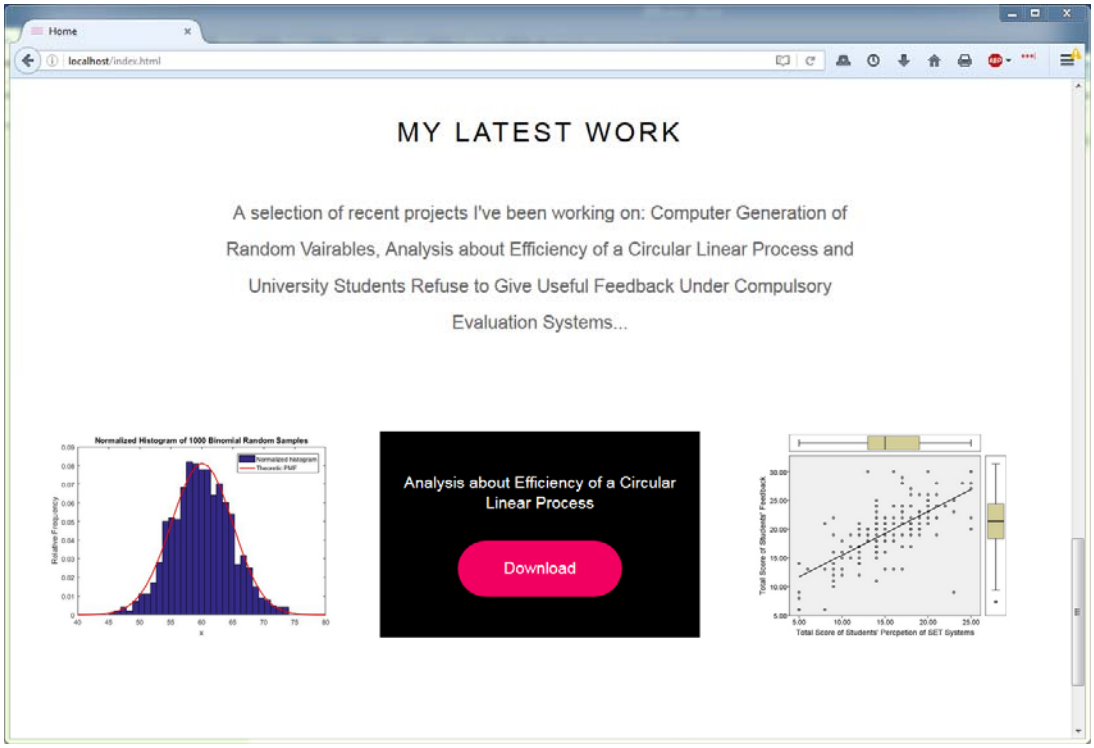

图 7 个人主页界面效果图(e)

由此可见，本服务器能够很好地完成包含多个种类的对象的较为复杂的个人网页的实现工作。并且，在图 3 的图中单击 Download Resume 按钮，将自动加载位于 pdf 文件夹中的个人简历，如图 8 所示.单击图 4 中 Get in Touch 按钮，可自动打开邮件客户端并向个人主页持有者发送邮件，如图 9 所示.将鼠标放在图 6 所示的页面下部的各个图片上，即会显示如图 7 所示的效果，此时点击 Download 按钮即可加载 pdf 文件夹中对应的文件，如图 10 所示。可见，本服务器还能很好地实现网页与客户之间的交互工作。

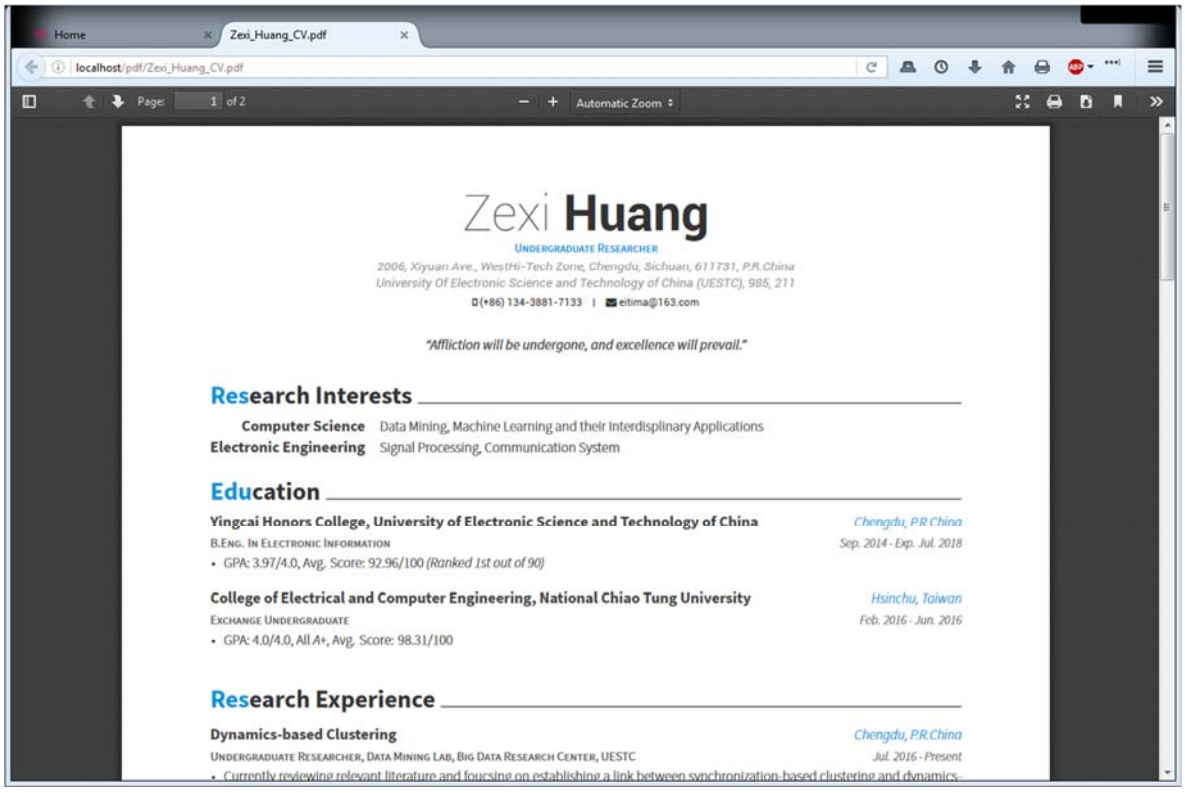

图 8 单击 Download Resume 自动加载个人简历示意图

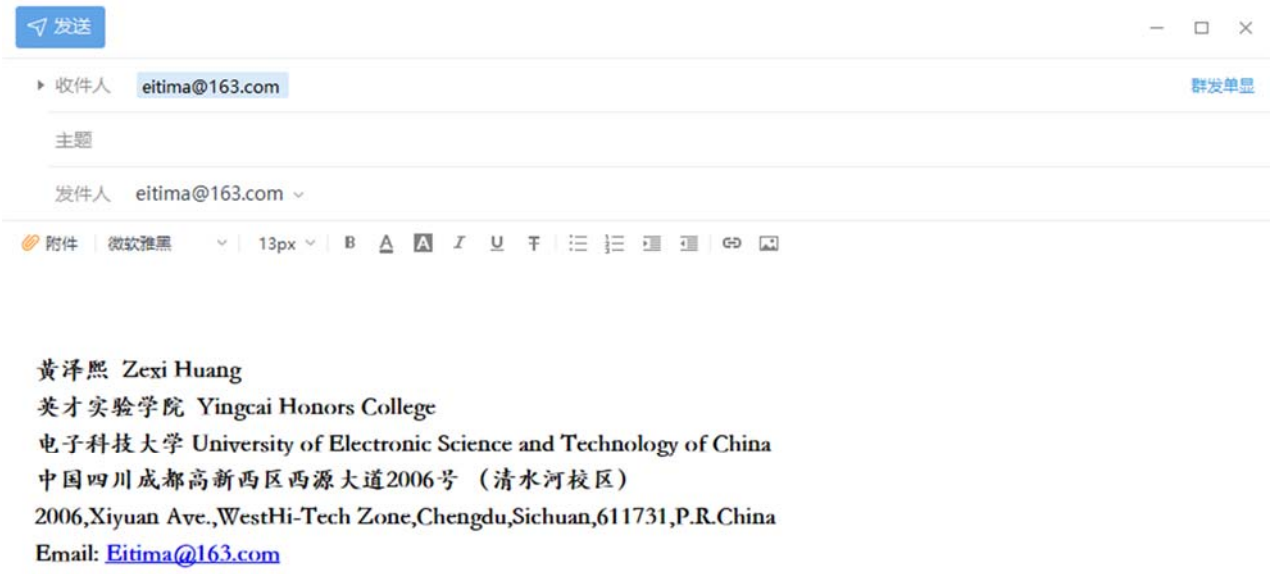

图 9 单击 Get in Touch 自动打开邮件客户端向个人主页所有者发送邮件示意图

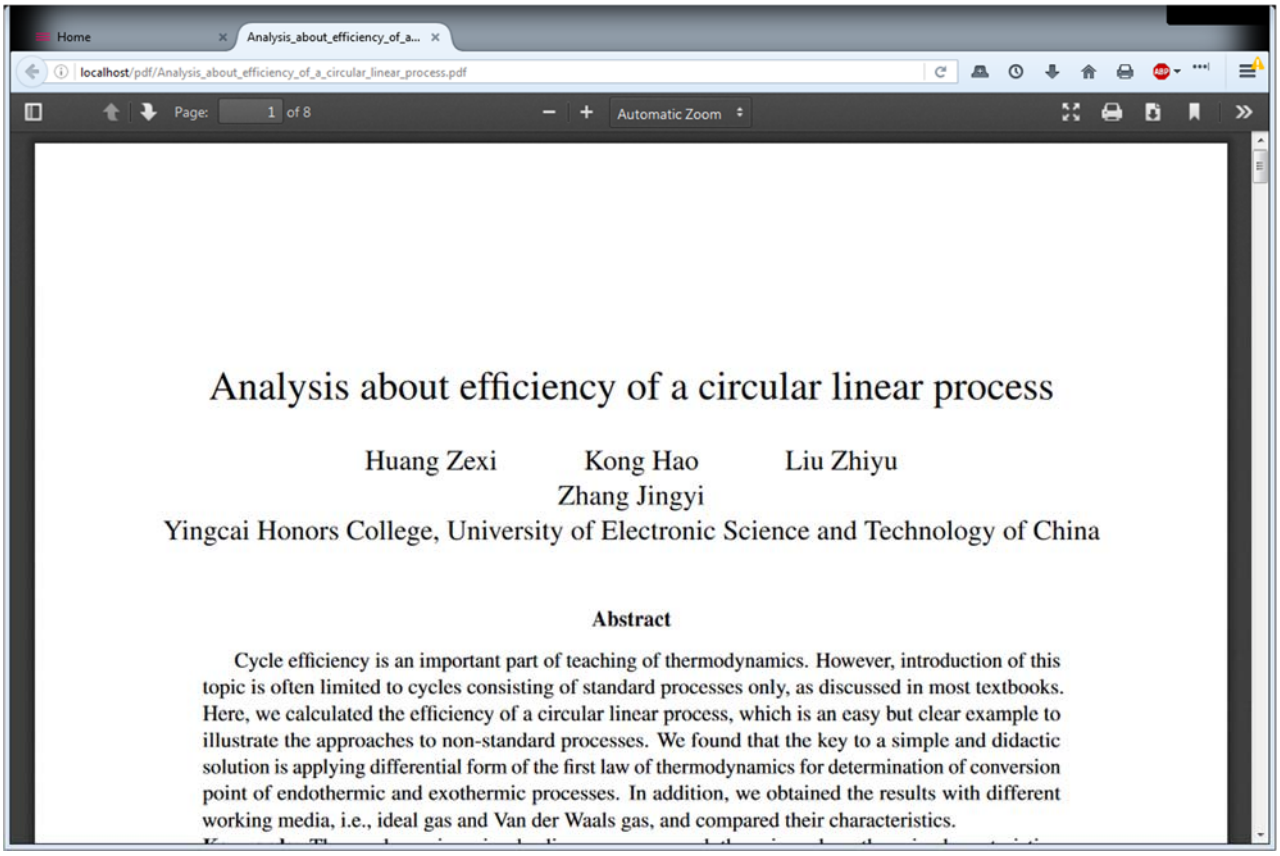

图 10 单击 Download 自动加载文件的示意图

4. 结论

本文介绍了一种基于 Java Socket 的多线程简易 HTTP 服务器，并通过一个个人主页的案

例展示了该服务器的良好的可应用性。

参考文献

[1] Kurose J F, Ross K. Computer Networking: A Top-Down Approach Featuring the Internet[J]. Network, 2009, 18(142):1–11.