**题目：设计简单WEB服务管理系统**

**姓名：王怡恒**

**学号：2022300385**

**班号：10012201**

**时间：2024/12/20**

**计算机学院**

**目 录**

[一、实验目的和要求 5](#_Toc6518)

[（一）实验目的 5](#_Toc30593)

[（二）实验要求 5](#_Toc18475)

[二、相关工具和开发环境 5](#_Toc21182)

[（一）相关工具 5](#_Toc18052)

[（二）开发环境 5](#_Toc21798)

[三、实现原理及流程 6](#_Toc15889)

[（一）HTTP 协议基础 6](#_Toc3277)

[（二） 浏览器功能实现流程 6](#_Toc27732)

[（三）服务器端功能实现流程 8](#_Toc7667)

[四、系统设计 9](#_Toc24701)

[（一）浏览器设计 9](#_Toc4883)

[（二）服务器端设计 10](#_Toc1858)

[五、运行结果与分析 10](#_Toc13517)

[（一）运行结果 10](#_Toc17430)

[（二）结果分析 14](#_Toc25393)

[六、实验过程发现的新技术问题 14](#_Toc28621)

[七、参考文献 15](#_Toc10847)

**一、实验目的和要求**

**（一）实验目的**

本实验旨在设计并实现一个 Web 服务管理系统，深入理解 Web 服务中涉及的 HTTP 协议、浏览器与服务器的交互机制、动态网页和静态网页的处理流程，以及相关功能的实现技术，能够构建一个具备基本功能的 Web 服务环境，并通过测试验证系统的正确性和稳定性。

**（二）实验要求**

1. 实现一个包含浏览器和服务器端的 Web 服务管理系统，浏览器能够发送各种 HTTP 请求，服务器端能够正确响应并提供相应服务。
2. 服务器端需支持 GET、POST、HEAD、DELETE 等常见 HTTP 命令，对于 GET 请求，能够正确处理静态网页（如返回 index.html）和动态网页（如实现两个数相加结果）的获取，支持多进程通信用于动态网页处理。
3. 浏览器应具备用户输入 URL、显示服务器应答网页内容（包括静态和动态）的功能，能够正确解析和显示不同类型的 HTTP 响应状态码对应的结果。
4. 按照规范的实验报告格式撰写报告，详细阐述实验过程中的各个环节，包括相关工具和开发环境、实现原理及流程、系统设计、运行结果与分析、遇到的问题及建议、参考文献等内容。

**二、相关工具和开发环境**

**（一）相关工具**

1. 在浏览器设计阶段，运用了 Python 语言，借助requests库实现 HTTP 请求的发送和处理，tkinter库用于构建图形用户界面（GUI），BeautifulSoup库用于解析 HTML 页面，PIL（Python Imaging Library）库用于处理图像，socket库用于底层网络通信（如通过 hosts 文件获取 IP 地址时建立 TCP 连接），urllib.parse库用于处理 URL 相关操作，webbrowser模块用于在默认浏览器中打开 URL（处理静态网页时）。
2. 对于服务器端设计与实现，使用了 Python 的内置模块http.server来创建 HTTP 服务器，os模块用于文件和目录操作（如处理 DELETE 请求时删除文件），urllib.parse模块用于解析 URL 中的查询字符串。

**（二）开发环境**

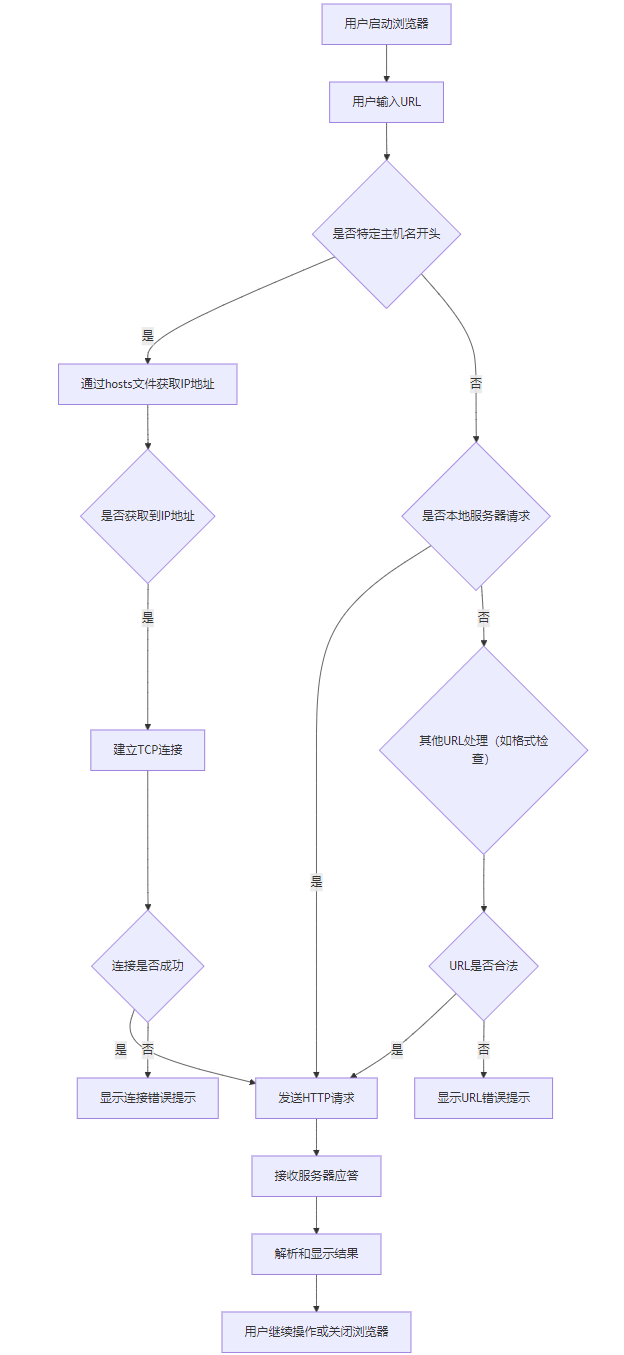
1. 本实验使用 Python 编程语言，开发环境可以是任何支持 Python 运行的环境，如 Python 自带的 IDLE 或者其他常用的集成开发环境（IDE），如 PyCharm 等。
2. 服务器运行在本地主机（[localhost](https://localhost/)）上，监听端口为 8000。

**三、实现原理及流程**

**（一）HTTP 协议基础**

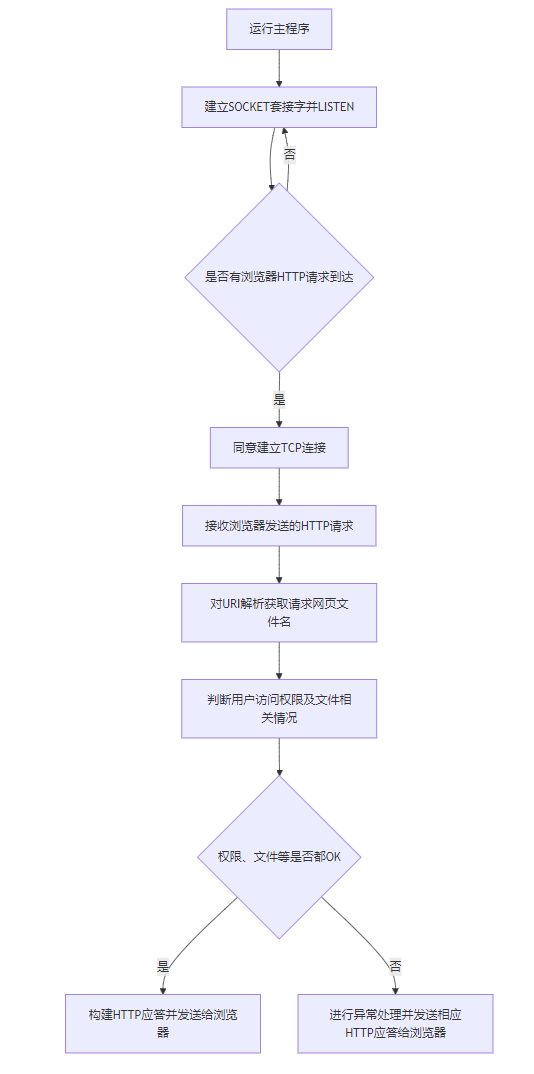
HTTP（超文本传输协议）是 Web 服务的核心协议，工作在 TCP/IP 协议体系中的 TCP 协议之上，允许 Web 服务器与浏览器之间进行超文本标记语言（HTML）文档及其他资源的传输。它定义了一系列请求方法（如 GET、POST、HEAD、PUT、DELETE、OPTIONS 等），每种方法具有特定的语义和用途。例如，GET 用于获取服务器上的资源，POST 用于向服务器提交数据，HEAD 用于获取资源的头部信息等。

1. **浏览器功能实现流程**



1. **用户输入 URL**：用户在浏览器的图形界面中输入要访问的 URL（统一资源定位符），如www.baidu.com或http://localhost:8000/index.html或http://localhost:8000/add?a=5&b=3等。
2. **获取服务器 IP 地址（针对特定主机）**：当输入的 URL 中包含特定主机名（如www.baidu.com）时，浏览器尝试从本地C:\Windows\System32\drivers\etc\hosts文件中获取对应的 IP 地址。若找到匹配项，则提取 IP 地址；若未找到，则显示错误提示。
3. **建立 TCP 连接（针对特定主机）**：对于需要从 hosts 文件获取 IP 地址的主机（如www.baidu.com），使用socket库建立 TCP 连接到服务器的指定端口（默认为 80）。
4. **发送 HTTP 请求**
   * 根据用户输入的 URL 和操作（如点击 GET、HEAD、DELETE 按钮），构建相应的 HTTP 请求报文。
   * 对于 GET 请求，如果 URL 指向静态网页（以.html或.htm结尾）且不是本地服务器（如www.baidu.com），使用webbrowser模块在默认浏览器中打开该网页；如果是本地服务器的静态网页请求或其他类型的 GET 请求（如动态网页请求http://localhost:8000/add?a=5&b=3），使用requests库发送 GET 请求。
   * 对于 HEAD 请求，使用requests库发送 HEAD 请求，获取服务器返回的头部信息。
   * 对于 DELETE 请求，使用requests库发送 DELETE 请求，尝试删除服务器上指定的资源。
5. **接收服务器应答**
   * 对于通过socket建立连接的请求（如www.baidu.com），直接接收服务器返回的应答数据并解码。
   * 对于使用requests库发送的请求，获取服务器返回的响应对象，根据响应状态码和内容进行处理。
6. **解析和显示结果**
   * 根据 HTTP 应答的状态码（如 200 表示成功、404 表示未找到资源等），判断请求是否成功。
   * 若成功，对于不同类型的请求进行相应处理。例如，对于 GET 请求显示网页内容、状态码和头部信息（如果是静态网页还可能显示部分正文内容预览）；对于 HEAD 请求仅显示格式化后的头部信息；对于 DELETE 请求显示操作结果信息。若失败，则将相应的错误信息显示给用户。

**（三）服务器端功能实现流程**



1. **启动服务器**：通过HTTPServer类创建一个 HTTP 服务器，绑定到本地地址localhost和端口 8000，指定自定义的请求处理类SimpleHandler来处理客户端请求。
2. **处理请求**
   * 当客户端发送请求时，SimpleHandler类中的相应方法（如do\_GET、do\_HEAD、do\_DELETE）被调用。
   * do\_GET**方法处理 GET 请求：**
     + 如果请求路径为/，将其映射为/index.html，然后尝试读取并返回对应的静态文件内容。
     + 如果请求路径以/add开头，解析查询字符串中的参数（假设为两个数字a和b），进行相加运算，将结果返回给客户端。
     + 如果文件不存在，返回 404 错误。
   * do\_HEAD**方法处理 HEAD 请求：**
     + 对于静态文件请求，获取文件的类型和长度，返回 200 状态码和头部信息（包括 Content-type 和 Content-Length）。
     + 对于动态请求（如/add），解析参数进行计算，返回 200 状态码和计算结果的头部信息（Content-type 为 text/plain）。如果参数解析错误，返回 400 错误。
   * do\_DELETE**方法处理 DELETE 请求：**
     + 根据请求路径尝试删除指定的文件。如果文件存在且删除成功，返回 200 状态码和成功信息；如果文件不存在，返回 404 错误；如果删除过程中出现其他错误，返回 500 错误。

**四、系统设计**

**（一）浏览器设计**

1. **界面设计**：使用tkinter库构建图形用户界面，包含一个用于输入 URL 的输入框，以及 GET、HEAD、DELETE 三个按钮用于发送不同类型的 HTTP 请求，还有一个文本框用于显示服务器的应答内容。
2. **功能模块**
   * **连接管理模块**：当需要与特定主机建立 TCP 连接时（如www.baidu.com），负责创建和管理socket连接，确保通信链路的正常建立和关闭。
   * **请求构建模块**：根据用户输入的 URL 和操作按钮点击，使用urllib.parse和字符串拼接等方式构建正确格式的 HTTP 请求报文，设置请求方法、URL、HTTP 版本、头部字段等信息。
   * **响应处理模块**：接收服务器返回的应答数据（通过socket接收或requests库获取响应对象），解析状态码、头部信息和正文内容。根据状态码进行相应处理，如显示成功信息、错误提示或格式化后的头部信息等，并将结果在文本框中展示给用户。对于静态网页请求（非本地服务器），调用webbrowser模块在默认浏览器中打开网页。

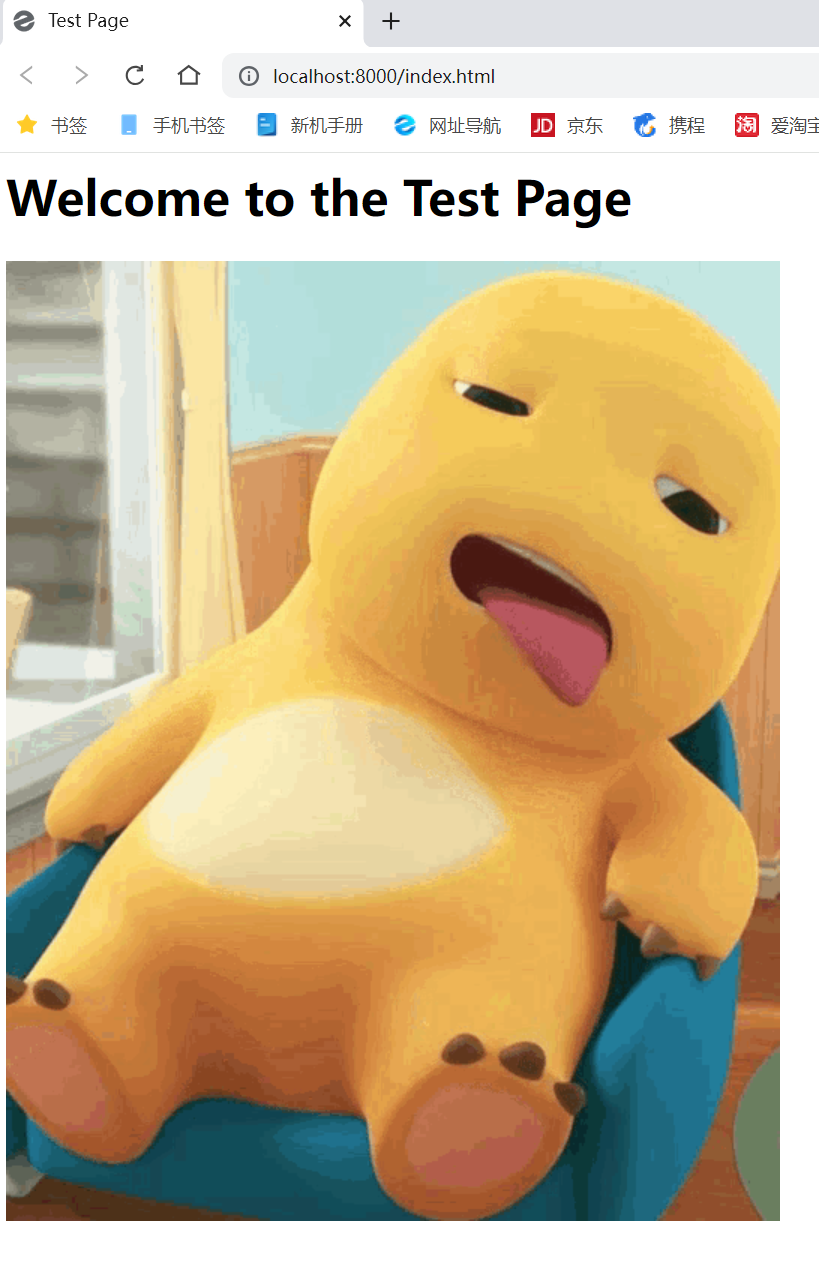
**（二）服务器端设计**

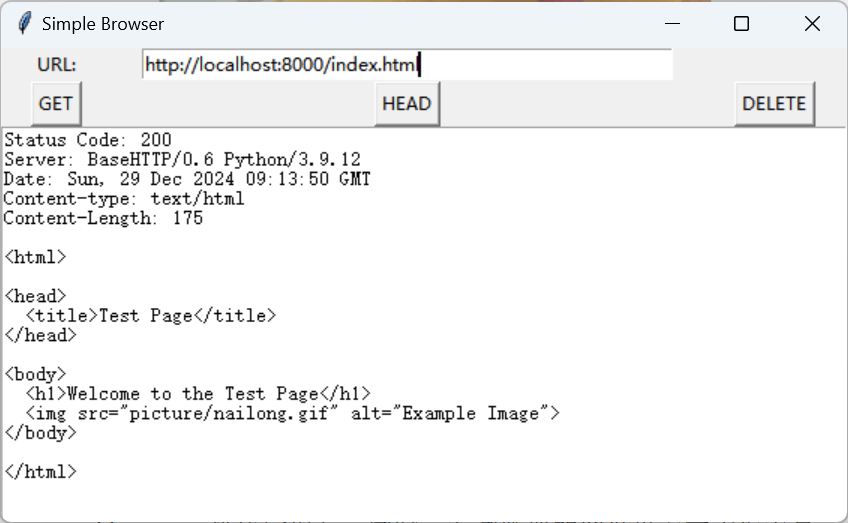
1. **主程序模块**：通过start\_server函数创建并启动 HTTP 服务器，设置服务器监听地址和端口，进入循环等待客户端请求。
2. **请求处理模块（SimpleHandler 类）**
   * 继承自BaseHTTPRequestHandler类，重写do\_GET、do\_HEAD、do\_DELETE等方法来处理不同类型的 HTTP 请求。
   * 在do\_GET方法中，根据请求路径判断是静态文件请求还是动态请求（如加法运算），分别进行相应处理。对于静态文件，读取文件内容并设置正确的 MIME 类型返回给客户端；对于动态请求，解析参数、执行运算并返回结果。
   * do\_HEAD方法根据请求路径处理静态文件或动态请求的头部信息返回。
   * do\_DELETE方法根据请求路径尝试删除文件，并根据操作结果返回相应的状态码和信息。

**五、运行结果与分析**

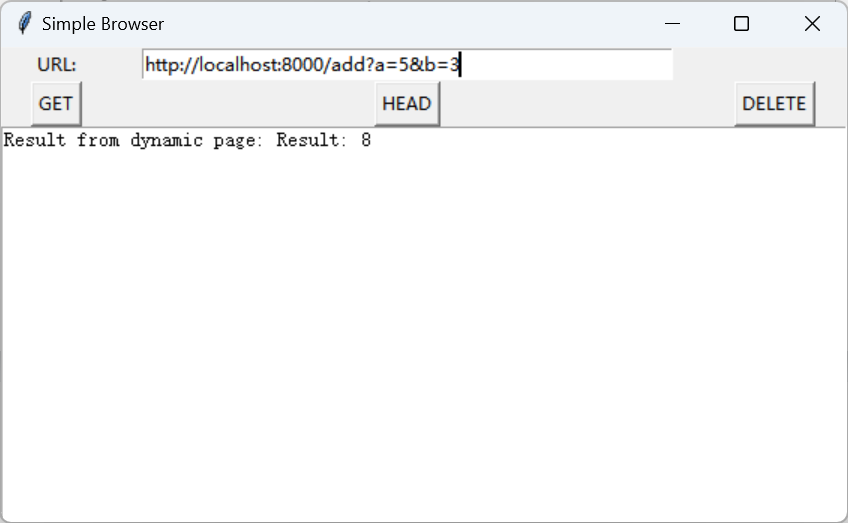
**（一）运行结果**

1. 启动服务器后，在浏览器输入http://localhost:8000/index.html，能够成功获取服务器端的网页内容并显示在默认浏览器中（通过webbrowser模块实现）。同时，在浏览器的图形界面文本框中显示与服务器建立 TCP 连接、发送请求和接收响应的相关信息，如 HTTP 状态码、响应头部信息等。

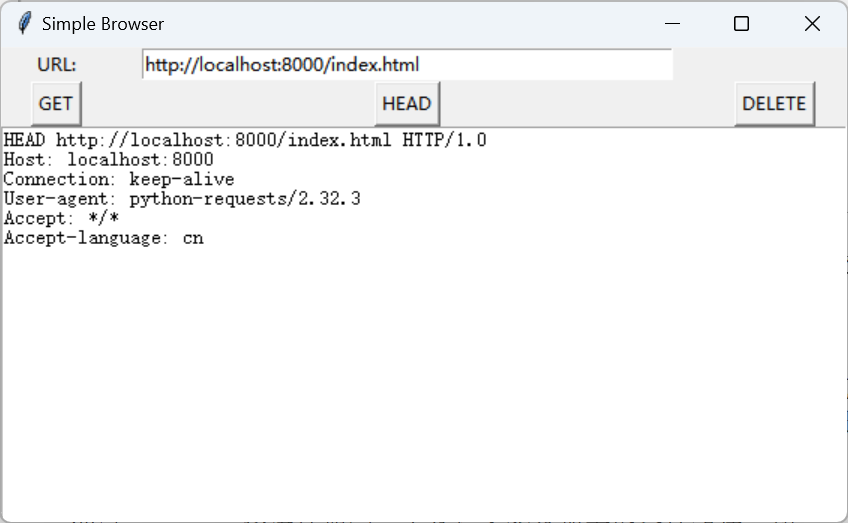




1. 输入http://localhost:8000/add?a=5&b=3，服务器正确计算两数之和（结果为 8）并返回，浏览器图形界面文本框中显示 “Result from dynamic page: 8”。



1. 点击 HEAD 按钮并输入http://localhost:8000/index.html，浏览器文本框中显示百度首页的 HEAD 信息，包括服务器类型、日期、内容长度、内容类型等头部字段。



1. 点击 DELETE 按钮并输入一个存在于服务器上的文件路径（如/test.txt，假设服务器上存在该文件），服务器成功删除文件，浏览器文本框中显示 “File /test.txt deleted”；若输入不存在的文件路径，显示 “File not found”。

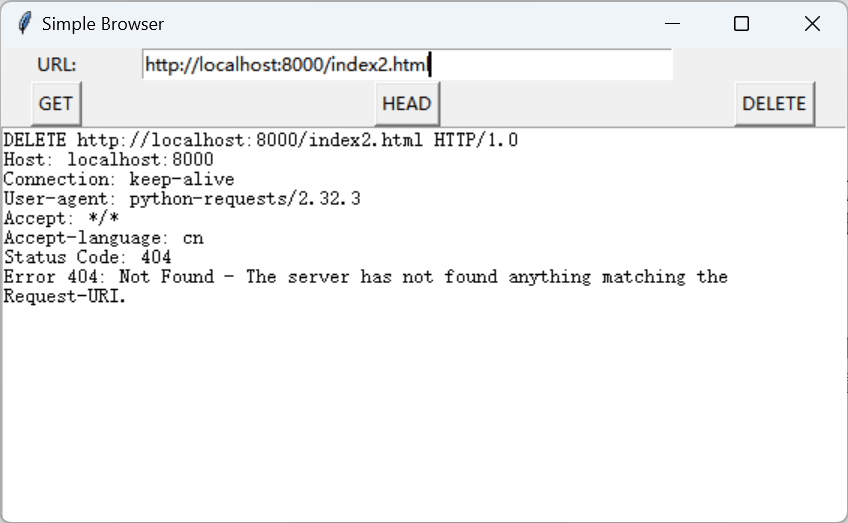


图 1 输入不存在的URL

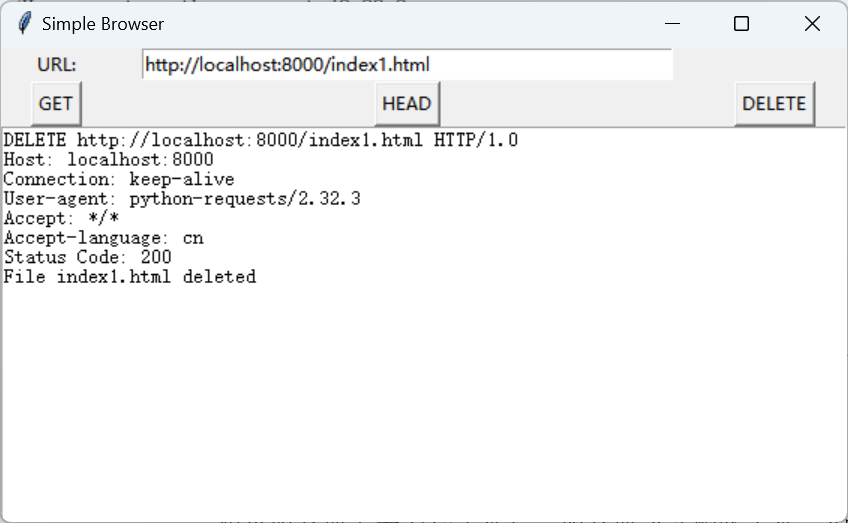


图 2 输入存在的URL

**（二）结果分析**

1. **功能正确性验证**
   * 对于不同类型的请求（GET 静态网页、GET 动态网页、HEAD、DELETE），服务器能够正确处理并返回预期的结果，浏览器能够正确解析和显示服务器的响应，表明系统的基本功能实现正确。例如，动态网页的加法运算结果准确，说明服务器端动态请求处理模块能够正确解析参数并执行计算；HEAD 请求返回的头部信息完整且正确，验证了服务器对 HEAD 请求的处理逻辑。
   * 浏览器能够根据用户输入的 URL 和操作，正确构建 HTTP 请求并发送到服务器，同时能够根据服务器的响应状态码进行相应的显示处理，反映了浏览器端请求构建和响应处理模块的有效性。
2. **性能和稳定性评估**
   * 在多次请求不同类型的资源和执行各种操作的过程中，系统表现稳定，未出现崩溃或异常响应的情况，说明系统的整体稳定性较好。
   * 然而，由于本实验主要关注功能实现，未对系统性能进行详细测试。在实际应用中，对于大量并发请求的处理能力、响应时间等性能指标可能需要进一步优化和评估，以满足更高的性能要求。例如，当多个客户端同时请求动态网页进行复杂计算时，可能需要考虑服务器的资源利用和响应效率。

**六、实验过程发现的新技术问题**

1. **URL 处理复杂性**
   * 在处理 URL 时，需要考虑多种情况，如不同的协议（http、https 等）、主机名解析（从 hosts 文件获取 IP 地址）、路径映射（如将/映射为/index.html）以及查询字符串解析（在动态网页请求中）。这增加了代码的复杂性，容易出现错误，尤其是在处理边界情况和特殊字符时。
2. **异常处理不完善**
   * 虽然在代码中对一些常见的异常情况进行了处理，如文件不存在（返回 404 错误）、参数解析错误（返回 400 错误）等，但可能存在一些未考虑到的异常情况。例如，网络连接突然中断、服务器过载导致响应超时等情况未进行全面的处理，可能导致用户体验不佳或系统不稳定。
3. **界面交互优化**
   * 浏览器的图形用户界面虽然能够实现基本功能，但在用户体验方面还有待提高。例如，界面布局可以更加美观和合理，对于较长的 URL 输入可能需要自动换行或滚动条支持；显示区域对于大量的服务器响应内容显示不够友好，可能需要分页或滚动条优化。

**七、参考文献**

1. CH5 设计并实现一个简单的WEB服务管理系统.pdf
2. 参考技术报告.docx
3. requests库官方文档：<https://docs.python-requests.org/>
4. tkinter库教程：<https://docs.python.org/3/library/tkinter.html>
5. http.server模块文档：<https://docs.python.org/3/library/http.server.html>
6. urllib.parse模块文档：<https://docs.python.org/3/library/urllib.parse.html>
7. BeautifulSoup库文档：<https://www.crummy.com/software/BeautifulSoup/bs4/doc/>
8. PIL（Python Imaging Library）库文档：<https://pillow.readthedocs.io/>