**100071011 Computer Networks 2022-2023-2**

**Project-2**

**Building a CGI-Support Multi-Threaded Web Server**

**Specification**

|  |  |
| --- | --- |
| **学号 (Student ID)** | **1120200822** |
| **姓名 (Name)** | **郑子帆** |
| **班号 (Class No.)** | **07112002** |
| **授课教师 (Instructor)** | **郑宏** |

**School of Computer**

**Beijing Institute of Technology**

**April 12, 2023**

1. **Requirement Analysis**

本项目旨在构建一个多线程Web服务器，支持CGI测试。该服务器能够并行处理多个同时服务请求，并向Web浏览器提供静态和动态页面以进行测试。本次实验使用了Python、HTML、CSS等多种语言。项目的具体要求如下。

**1.1 服务器概述**

服务器能够并行处理多个同时的服务请求（0-20左右）。在主线程中，服务器会监听一个固定的端口。当它接收到一个TCP连接请求时，它会通过另一个端口建立一个TCP连接，并在一个单独的线程中为该请求提供服务。服务器需要支持静态、动态网页请求。

**1.2 HTTP 请求 & 响应**

使用HTTP1.0，服务器将为网页的每个组件发送单独的HTTP请求。多线程服务器能够处理来自客户端的GET、POST和HEAD请求。同时，服务器应能返回适当的状态代码，包括200、400、403和404，并包含相应消息的正文。

在实现时，一个关键点是如何处理请求和响应的返回。在http请求的处理中，我们接受socket套接字中传递过来的数据，将其转化为可以识别的UTF-8编码方式。然后根据Header判断请求的类型具体是GET、POST还是HEAD；判断完成后转到对应的处理方法中，根据对应的资源地址返回对应的响应数据。

**1.3 最大连接数和线程池**

服务器需要有最大连接数的限制。如果同时打开的连接总数超过Max Connections，服务器将开始关闭最早打开的连接，并终止与它们关联的工作线程。如果工作线程的数量多于活动请求，一些线程将被阻塞，等待新的HTTP请求的到来；如果请求多于工作线程，则需要缓冲这些请求，直到有可用的线程。

在实现时，对于连接，我在本次项目中采用了TCP连接，它是面向连接的，可靠性得以保证；同时收发两端都要有成对的socket。对于服务器端应用程序，我们应该对每个客户端请求做出响应，如果接受该请求，则应该为其创建一个新的线程，从而确保所有客户端任务可以并发执行。然而，考虑到服务器负载有限，我们不能无限制地创建线程。因此，我们需要释放长时间不使用的资源，以供后续用户进行服务访问。为了实现这一目标，我们设计了一个线程池，用于对所有客户线程进行统一管理。

对于线程池管理，它被作为一个独立的线程运行，我将其设置为“守护线程”线程。这是一种运行在后台的特殊进行，它可以周期性执行指定任务。在每个运行周期中，我们会对每个客户端线程进行创建，客户端线程创建时会将自身加入到线程队列当中。当客户端线程达到服务器最大相应序列时，我们会将线程队列中最早的客户端线程进行销毁，以便后续用户进行资源访问。

**1.4 CGI实现动态网页**

CGI（公共网关接口）提供了一种简单的方式来构建动态网页，它是Web服务器将Web用户的请求传递给应用程序并接收返回数据以转发给用户的标准方式。一个CGI程序是实时执行的，因此它可以输出动态信息。

在实现时，CGI接收来自网页的表单、主机名和端口信息。表单内容包括运算数和运算符或查询条件，这些内容由不同的模块进行处理。Calculator.py模块负责进行计算，根据运算符执行相应的运算，并将结果替换到网页上进行显示。Query.py模块负责从数据库中进行查询，根据查询条件编写SQL语句来检索所需的数据，最后将结果替换到网页上进行显示。

**1.5 日志文件**

服务器应提供一种记录服务器上发生的所有事件的机制。服务器日志是一个简单的文本文件，用于记录服务器上的活动。日志文件的每一行表示一个请求（命中），每行需要包含：发出请求的计算机（即访问者）的IP地址、发出请求的计算机的标识、访问者的登录ID、访问发生的日期和时间、请求方法、请求文件的位置和名称、HTTP状态码（例如，文件成功发送、文件未找到等）、请求文件的大小、引用此请求的网页。

在实现时，log file会随着服务器的每次启动而生成，日志文件中的每条记录均来自于socket.recv()函数所接收到的数据。

**1.6 性能分析**

使用负载/压力测试工具，如http\_load、webbench等，来测量Web服务器的吞吐量和响应时间。

**1.7 文件结构**

Web服务器根目录：/webroot

CGI程序：/webroot/CGI-bin/

日志文件：/webroot/Log/

静态网页：/webroot/

默认页面：/webroot/index.html

404页码：/webroot/404.html

1. **Design**

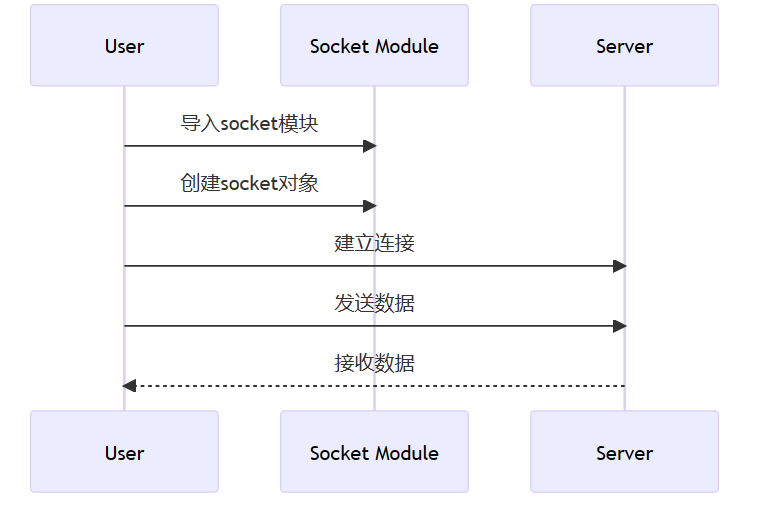
**2.1 TCP Socket连接**

（1）导入socket模块：在Python中，首先需要导入socket模块，它提供了进行网络通信的基本功能。

（2）创建socket对象：使用socket模块的socket()函数创建一个socket对象。需要指定地址族和套接字类型。

（3）建立连接：使用socket对象的connect()方法与客户端建立TCP连接。需要传入客户端的IP地址和端口号。

（4）发送和接收数据：连接建立后，可以使用send()方法发送数据到客户端，使用recv()方法接收客户端发送的数据。

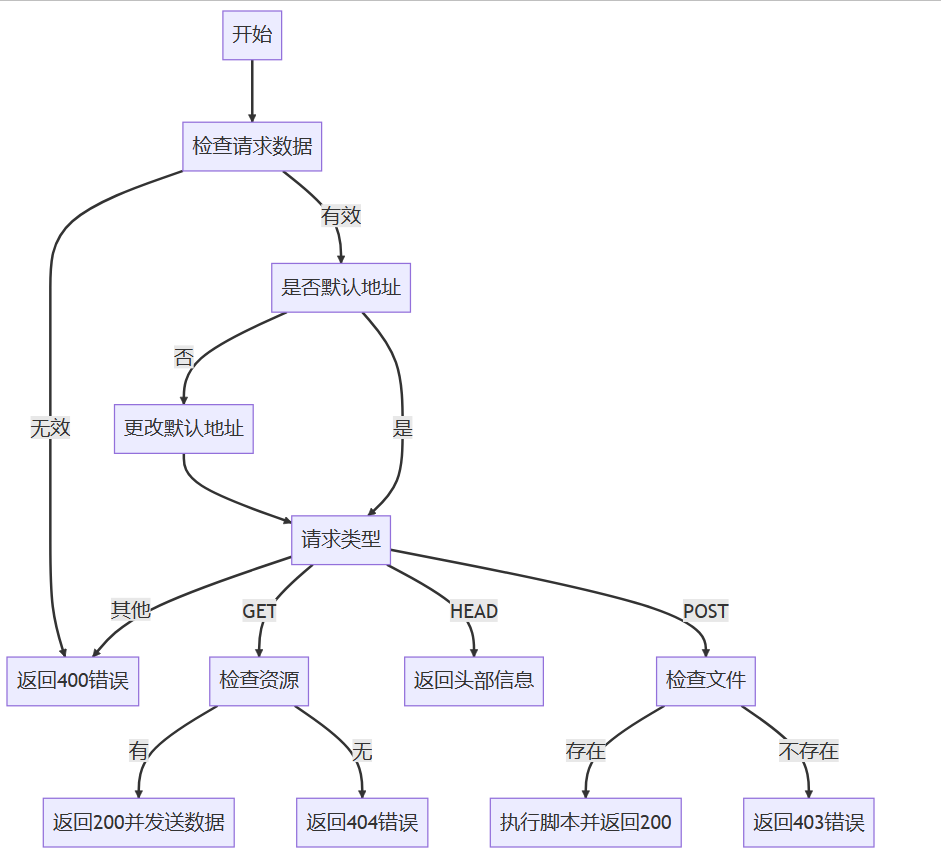


**2.2 HTTP 请求 & 响应**

对于HTTP的请求和响应，逻辑思路比较简单，我们首先需要查看是否接收到请求数据，然后判断请求数据是否有效。如果有效，那么判断请求资源地址是否是默认地址，如果不是则先对默认资源地址进行更改，然后根据报文中的报文类型字段分别判断该请求是GET请求、POST请求还是HEAD请求，对于对应的请求做相对应的处理。如果不是这三种请求，则返回错误状态码400。

GET请求是对资源的查询，如果服务器查询本地有请求中对应的资源，则包装好响应信息发送回去，若没有则说明本地没有请求需要的资源，返回404状态码；否则为OK状态码200。HEAD请求类似于GET请求，但只返回响应头部信息，而不返回响应体内容。所以HEAD请求处理包含于GET请求处理中，只不过在响应报文的包装上有一些区别。

POST用于请求服务器对数据进行处理，所以在POST请求中我们要做的处理是：=根据请求的文件名和参数，在服务器上执行相应的脚本，并将执行结果作为HTTP响应返回给客户端。如果文件不存在则返回错误状态吗403；否则为OK状态码200，然后执行完子进程后将标准输出包装进响应报文。

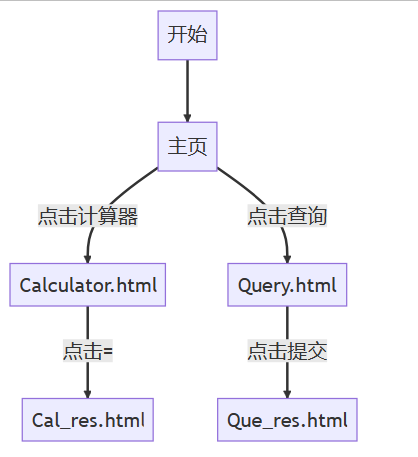


**2.3 线程池管理**

因为我们需要对服务端创建多个线程以并行处理多个客户端的请求。我们设计Worker类继承自Thread类，不同的Worker实例用于处理不同的客户端的需求。特殊地，我们需要处理对线程池进行管理，故设计继承类ThreadPoolManager继承自Thread类。具体地，我们用队列queue来记录当前所有Worker线程，ThreadPoolManager用于检查当前的线程数量是否已经达到设定的最大值，如果到达了，则销毁queue中第一个Worker线程，并释放资源。

**2.4 页面和数据库**

对于页面的设计，我们一共有5个html文件，首先主页为index.html，其中包含“计算器”和“查询”两个按钮。点击“计算器”则会跳转到Calculator.html，点击“查询”则会跳转到Query.html。在Calculator.html中，输入两个运算数和选择运算符号后，点击“=”，会跳转到Cal\_res.html。在Query.html中，输入要查询的学生的StudentID，点击“提交”则会跳转到Que\_res.html。



另外，我们还设计了3个错误状态码提示页面，分别为400.html、403.html、404.html。

对于数据库的设计，这里我们使用了csv文件搭建的数据库，装有一个关于学生信息的关系数据表，里面包含的键有Student ID、Name、Sex、Age和Class，其中主键为Student ID。

**2.5 CGI**

在本实验中我们需要完成两部分CGI的设计和实现，分别是计算器的CGI和数据库查询的CGI。

对于计算器的CGI，我们需要编写Calculator.py，从参数中取出要进行计算的数，然后判断要进行的运算类型，进行运算得到结果，将结果替换到Cal\_res.html中。

对于数据库查询的CGI，在查询时，我们通过数据Student ID以查询学生的完整信息。我们需要编写Query.py，从参数中取出要查寻的Student ID，并且在数据库中查找，最后将查找到的完整信息提换掉Que\_res.html中的$data。

1. **Development and Implementation**

本实验中的代码均在vs code中完成，下面将对于几个部分较为重要的代码进行展示和讲解。

**3.1 TCP Socket连接**

# 设置最大连接数和监听端口

# 创建服务器socket，绑定地址并开始监听

port = 8888

server\_socket = socket.socket(socket.AF\_INET, socket.SOCK\_STREAM)

host\_name = socket.gethostname()

host\_name = socket.gethostbyname(host\_name)

address = ("0.0.0.0", port)

server\_socket.setsockopt(socket.SOL\_SOCKET, socket.SO\_REUSEADDR, 1)

server\_socket.bind(address)

server\_socket.settimeout(60)

server\_socket.listen(max\_connection)

接收消息：

*self*.socket = tasks.get()

working\_thread.append(*self*)

sema.release()

message = *self*.socket.recv(8000).decode("utf-8")

**3.2 HTTP 请求 & 响应**

GET/HEAD请求处理：

*def* GET(*self*, *file\_name*, *is\_head*=False):     # 处理GET/HEAD请求

    # 判断请求的文件是否存在，设置响应的状态码和响应头

    # 如果文件存在，返回文件内容；如果文件不存在，返回404页面

    # 如果是HEAD请求，只返回响应头

    if (os.path.isfile(*file\_name*)):

        file\_suffix = *file\_name*.split('.')

        file\_suffix = file\_suffix[-1].encode()

        content = *b*"HTTP/1.1 200 OK\r\nContent-Type: text/" + file\_suffix + *b*";charset=utf-8\r\n"

*self*.status\_code = 200

    else:

        content = *b*"HTTP/1.1 404 Not Found\r\nContent-Type: text/html;charset=utf-8\r\n"

*file\_name* = "404.html"

*self*.status\_code = 404

    content += *b*'\r\n'

*self*.socket.sendall(content)

    file\_size = 0

    if not *is\_head*:

*self*.file\_handle = open(*file\_name*, "rb")

        for line in *self*.file\_handle:

*self*.socket.sendall(line)

        file\_size = os.path.getsize(*file\_name*)

*self*.write\_log(file\_size)

POST请求处理：

*def* POST(*self*, *file\_name*, *args*):    # 处理POST请求，执行python脚本

    command = 'python ' + *file\_name* + ' "' + *args* + '" "' + *self*.socket.getsockname(

    )[0] + '" "' + str(*self*.socket.getsockname()[1]) + '"'

*self*.proc = subprocess.Popen(command,

*shell*=True,

*stdout*=subprocess.PIPE)

*self*.proc.wait()

    file\_size = 0

    if (*self*.proc.poll() == 2):  ## 文件不存在时返回值为2

        content = *b*"HTTP/1.1 403 Forbidden\r\nContent-Type: text/html;charset=utf-8\r\n"

        page = *b*''

*self*.file\_handle = open("403.html", "rb")

        for line in *self*.file\_handle:

            page += line

        content += *b*'\r\n'

        content += page

*self*.status\_code = 403

    else:

        content = *b*"HTTP/1.1 200 OK\r\nContent-Type: text/html;charset=utf-8\r\n"

        content += *self*.proc.stdout.read()

        file\_size = os.path.getsize(*file\_name*)

*self*.status\_code = 200

*self*.socket.sendall(content)

*self*.write\_log(file\_size)

**3.3 线程池管理ThreadPoolManager**

*class* ThreadPoolManager(threading.Thread):  # 线程池管理类，负责创建Worker和管理线程池的大小

*def* \_\_init\_\_(*self*, *log\_name*):

        threading.Thread.\_\_init\_\_(*self*)

*self*.setDaemon(True)      # 设置为守护线程

*self*.start()

*self*.log\_name = *log\_name*

*def* run(*self*):

        # 创建指定数量的Worker

        # 不断检查线程池的大小，如果线程池已满，删除一个线程

        for i in range(max\_connection):

            Worker(log\_name)

        while True:

            for i in range(10):

                if (len(working\_thread) == max\_connection and max\_connection != 0 and (not tasks.empty())):

                    working\_thread[0].restart()

            sema.acquire(*timeout*=1)

**3.4 计算器脚本Calculator.py**

import sys

try:

    ini = sys.argv[1]

    ini = ini.split("&")

    a = ini[0].split("=")[1]

    b = ini[1].split("=")[1]

    c = ""

    c = ini[2].split("=")[1]

    res = ""

    with open("cgi-bin/Cal\_res.html", "r", *encoding*="utf-8") as f:

        for line in f:

            res += line

    # 替换$a和$b

    res = res.replace("$a", a)

    res = res.replace("$b", b)

    # 判断运算符号，进行运算

    if c == "mul":

        res = res.replace("$c","\*")

        res = res.replace("$res", str(float(a) \* float(b)))

    elif c == "div":

        res = res.replace("$c", "/")

        res = res.replace("$res", str(float(a) / float(b)))

    elif c == "add":

        res = res.replace("$c", "+")

        res = res.replace("$res", str(float(a) + float(b)))

    else:

        res = res.replace("$c", "-")

        res = res.replace("$res", str(float(a) - float(b)))

    # 替换socket和result

    res = res.replace("$hostname", sys.argv[2])

    res = res.replace("$port", sys.argv[3])

    print(res)

except:

    error\_page = ""

    with open("400.html", "r", *encoding*="utf-8") as f:

        for line in f:

            error\_page += line

    print(error\_page)

**3.5 数据库查询脚本Query.py**

import csv

import sys

ini = sys.argv[1]

hostname = sys.argv[2]

port = sys.argv[3]

student\_id = ini.split("=")[1]

csv\_file = './data/Student\_data.csv'

with open(csv\_file, 'r', *encoding*='utf-8') as f:

    reader = csv.reader(f)

    student\_data = *list*(reader)

Get\_res = 0

res = ""

with open("./cgi-bin/Que\_res.html", "r", *encoding*="utf-8") as f:

    for line in f:

        res += line

    for student in student\_data:

        if student[0] == student\_id:

            temp = "<tr>"

            for i in range(5):

                temp += "<th>" + student[i] + "</th>"

            temp += "</tr>\n"

            res = res.replace("$data", temp, 1)

            Get\_res = 1     # 说明找到了

if (not Get\_res):

    temp = "<p>Sorry, there's no correspoding data for your request.</p>\n"

    res = res.replace("$data", temp, 1)

print(res)

3.6 HTML页面

在本实验中我们实现了所需的功能，一共有8个html文件，由于篇幅原因，这里仅展示Calculator.html。

<html>

<title>Home</title>

</head>

<body>

    <form method="post" action="./Calculator.py">

        <div class="login">

            <h2>Calculator</h2>

            <div class="login-top">

                1st num: <input type="text" name="a" value="num1" onfocus="this.value = '';" onblur="if (this.value == '') {this.value = 'num1';}">

                <br>

                2nd num: <input type="text" name="b" value="num2" onfocus="this.value = '';" onblur="if (this.value == '') {this.value = 'num2';}">

                <div class="forgot">

                    +<input type = "radio" name = "ch" value = "add"/>

                    -<input type = "radio" name = "ch" value = "sub"/>

                    \*<input type = "radio" name = "ch" value = "mul"/>

                    /<input type = "radio" name = "ch" value = "div"/>

                    <input type="submit" value="=" />

                </div>

            </div>

            <div class="login-bottom">

            </div>

        </div>

    </div>

    </form>

</body>

</html>

1. **System Deployment, Startup, and Use**

**4.1 项目文件结构**

根目录下的文件结构如下：

cgi-bin文件夹：里面包含Cal\_res.html和Que\_res.html。

data文件夹：里面包含Student\_data.csv，即学生信息数据库。

images文件夹：里面包含网页中所用到的图片（静态网页），在本项目中只有一张乐学首页截图。

log文件夹：用于存放日志文件。在提交的项目文件中，只放了一个demo log文件用于展示。

Server.py：服务端主程序。

Worker.py：worker线程类的实现主程序。

Calculator.py：计算器功能实现的Python脚本。

Query.py：数据库查找功能实现的Python脚本。

Calculator.html：计算器功能的主页面。

Query.html：数据库查询功能的主页面。

400.html：400报错页面。

403.html：403报错页面。

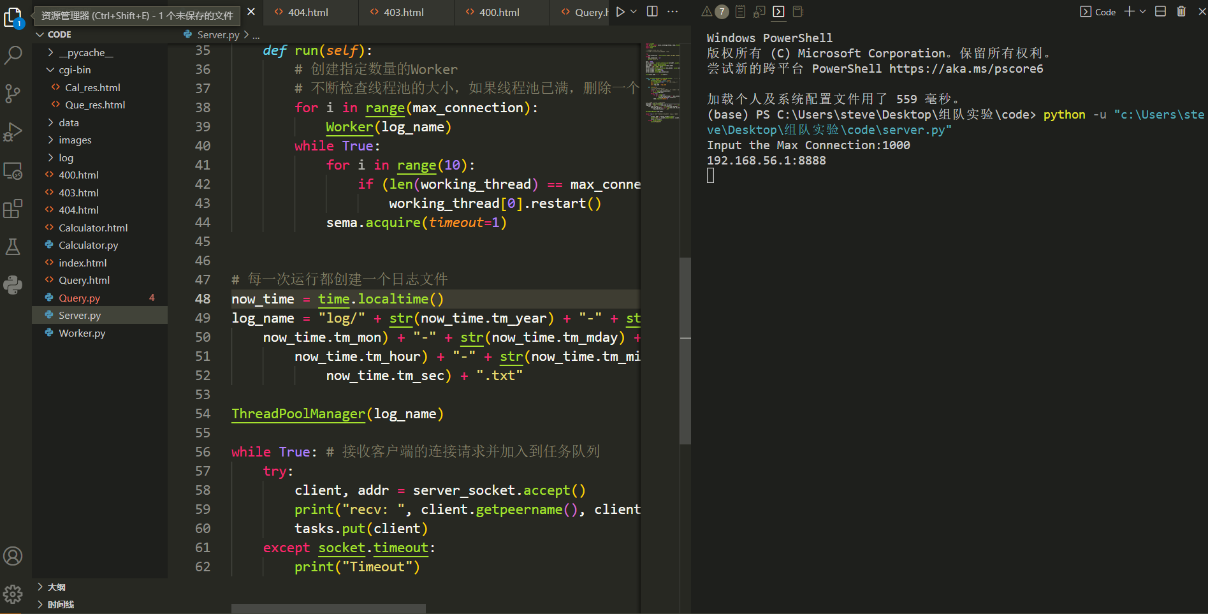
404.html：404报错页面。

**4.2 运行方法**

运行本项目程序需要安装配置Python3环境。可以通过命令行运行：python Server.py即可。

在运行时请不要开VPN、网络代理等，不然可能会影响实验的进行。

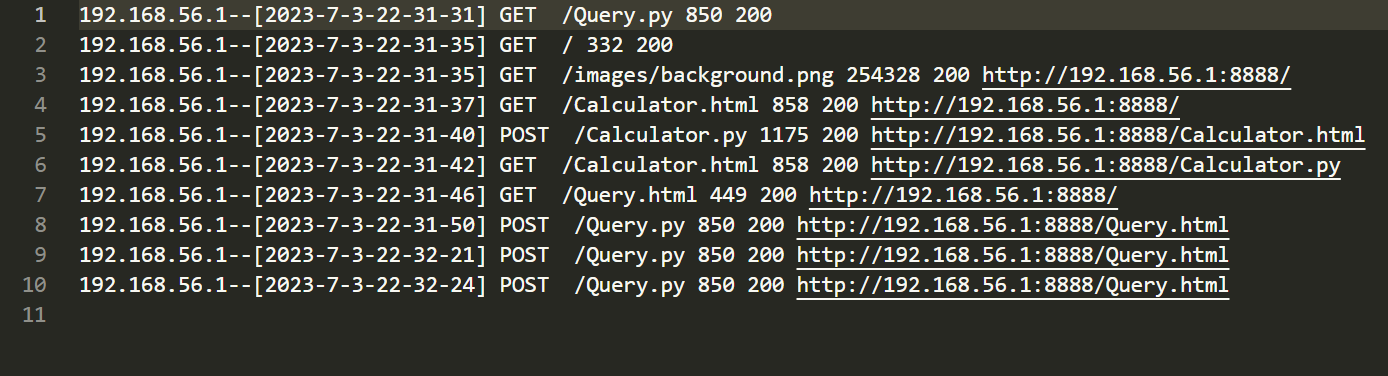
在运行Server.py并输入Max Connection后，如下图，将标准输出中的IP地址和端口复制粘贴到浏览器中打开。



1. **System Test**

**5.1 单元测试**

对于Server.py的测试，当我们正常进行连接后，日志文件能够正常记录GET/POST请求，如下图。

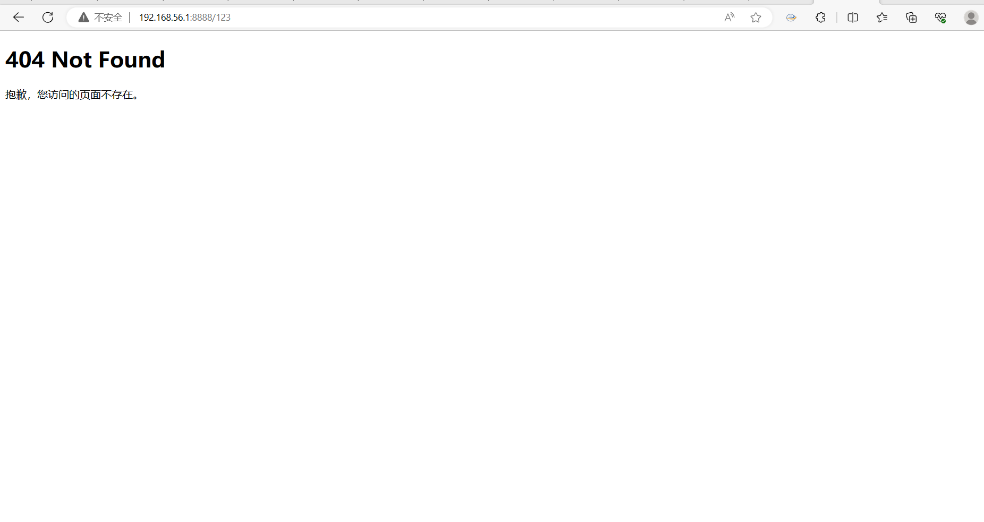


但当长时间没有响应时，会出现超时输出，如下图。



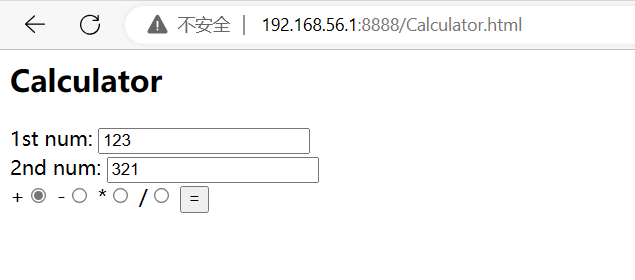
对于Worker的测试，我们主要测试日志输出部分，对于上图中的日志结果，经分析和Wireshark抓包结果相同，故测试通过。

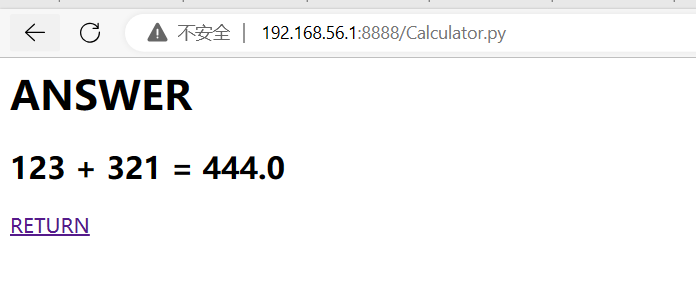
另外，测试GET请求，如果我们输入一个本地没有资源，则应该跳转到404错误界面，如下图。



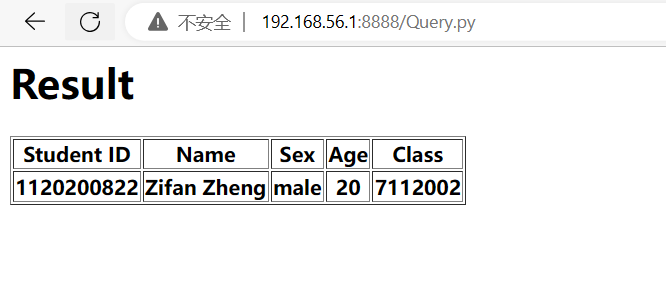
**5.2 集成测试**

首先测试计算器功能部分，当我们输入两个数且选择好运算符号后，点击“=”可以得到结果，按照等价类划分的方式，我们进行了一些测试，取加法测试结果如下：

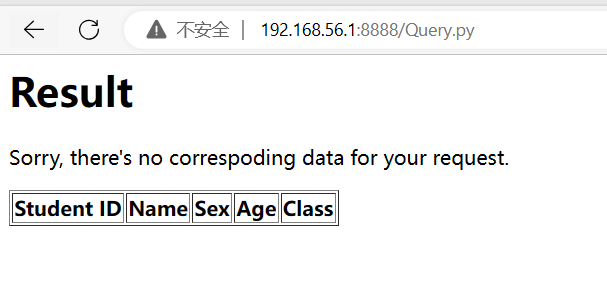




然后测试数据库查询，输入数据库中存在的Student ID，结果如下：



如果输入的Student ID在数据库中不存在，会进行提示：



1. **Performance and Analysis**

对于Server.py，在一开始我们会先输入一个Max Connection，显然，如果这个数越大，那么服务器所承受的“压力”上限就会越大。

对此，如果当前所有的worker线程都在忙，我们可以采用“阻塞”的方式，先阻塞一些线程，在服务器完成某些任务后，释放处理该任务的线程所占用的资源，以继续处理当前被阻塞的任务。

经过简单的测试，本项目所搭建的多线程服务器在一定范围内拥有较好的稳定性和较高的传输水平，即不会错过一些任务和高数据传输速率。

1. **Summary or Conclusions**

在这个项目中，我们研究了TCP Socket连接、HTTP的GET、HEAD、POST请求、并发连接，并成功构建了一个多线程服务器。通过使用线程池的方式，我们实现了对多线程服务器的搭建。此外，我们还使用Python语言完成了CGI模块，包括计算器和数据库查询两部分内容，成功构建了静态网页和动态网页。通过使用代码搭建多线程服务器的方式，我们对TCP连接、HTTP协议和CGI程序调用有了更深入的了解。

1. **References**

[1] Python的socket库: <https://blog.csdn.net/zsrwan/article/details/107082010>

[2] Python Socket模块的介绍与简单使用: [https://blog.csdn.net/qq\_42967398/article/details/105  
326616](https://blog.csdn.net/qq_42967398/article/details/105326616)

[3] Python实现多线程: <https://zhuanlan.zhihu.com/p/91601448>

[4] CGI实现简易网页加法计算器功能: [https://blog.csdn.net/zhyh1435589631/article/details/51  
530926](https://blog.csdn.net/zhyh1435589631/article/details/51530926)

[5] Vail, Cordell. "Stress, load, volume, performance, benchmark and base line testing tool evaluation and comparison." Tersedia: http://vcaa. com/tools/loadtesttoolevaluationchart-023. pdf (2005).

1. **Comments**

总的来说这次的项目的码量还是挺大的，但是认真做也会发现并没有那么难。我们在第7章学习了HTTP，这次也是对于理论知识的进一步探索与实践，我也通过这次实践对计算机网络的应用层，和网络编程有了更深的了解。

一学期时间匆匆飞逝，学期来到了尾声，这学期的计网课让我从对计网繁杂的抵触到好奇，再到感兴趣，也增进了自己英文阅读与写作的能力。最后，感谢郑宏老师和宿红毅老师一学期以来的倾囊相授，祝两位老师未来一切顺利！