100071011 Computer Networks 2022-2023-2

Project-2

Building a CGI-Support Multi-Threaded Web Server

Specification

学号 (Student ID)	1120202944
姓名 (Name)	董若扬
班号 (Class No.)	07112005
授课教师 (Instructor)	胡琳梅

School of Computer
Beijing Institute of Technology
April 12, 2023

1. Requirement Analysis

a) 本项目要求构建一个支持 CGI 测试的多线程 Web 服务器，能够并行处理多个请求。该服务器提供静态和动态页面，用于测试 Web 浏览器。

b) HTTP1.0

使用HTTP1.0，服务器为网页中的每个组件发送单独的HTTP请求。该多线程服务器能够处理来自客户端的GET、POST和HEAD请求，并返回适当的状态代码，包括200、400、403和404，并带有返回消息的正文。在服务器中，处理请求和返回响应是关键。对于HTTP请求的处理，主要任务是接收套接字中传递的数据，并将其转换为可识别的UTF-8编码方式。接下来，根据报文头判断请求类型是GET、POST还是HEAD，然后将其转到相应的处理方法中，根据对应的资源地址返回相应的响应数据。

c) 并行处理 TCP 连接

该服务器可以并行处理多个请求。在主线程中，服务器监听一个固定的端口。当它接收到TCP连接请求时，它会通过另一个端口建立TCP连接，并在一个单独的线程中为该请求提供服务。在本项目中，客户端和服务器之间使用TCP连接，这种连接是面向连接的、面向流的，提供高可靠性服务。发送和接收两端都需要有一对一的套接字。TCP套接字是本项目中连接的底层原理。

d) 最大连接数和线程池

该服务器可以限制最大连接数。如果同时打开的连接总数超过maxConnections，服务器将开始关闭最旧的打开连接，并且它们关联的工作线程将终止。如果工作线程的数量多于活动请求，那么部分线程会被阻塞，等待新的HTTP请求到来；如果请求多于工作线程，则需要缓冲这些请求，直到有就绪线程。作为服务器端，我们应该针对每一个客户端请求进行响应，如果接受该请求，则应该为其创建一个新的线程，从而保证所有客户端任务并发执行。但是，考虑到服务器负载有限，我们不能无限制地创建线程，因此我们需要将长时间不使用的资源进行释放，以供后续用户进行服务访问。因此，我们设计了一个线程池来统一管理所有的客户线程。

线程池本身也作为一个独立的线程运行，但与客户端线程不同的是，我们将其设置为守护线程（setDaemon）。它可以周期性地执行指定任务。在每个运行周期中，我们会为每个客户端请求创建一个线程，并将其加入到线程队列中。当客户端线程达到服务器最大响应序列时，我们会销毁线程队列中最早的客户端线程，以便后续用户进行资源访问。

e) CGI

CGI提供了一种简单的方式来构建动态网页。使用CGI可以接收网页提交的表单、主机名和端口，并通过不同的模块处理这些表单内容。例如，Cal.py模块会根据运算符进行相应的计算，并将结果替换显示在网页上；Query.py模块则负责从数据库中查询数据，根据查询条件编写SQL语句，并使用cursor.execute()执行SQL语句从数据库中检索所需的数据，最后将结果替换显示在网页上。

f) 日志文件

为保持服务器的正常运行和管理，必须提供一种机制来记录服务器上发生的所有事情。这种机制是通过生成服务器日志来实现的，每个日志文件都是一个简单的文本文件，用于记录服务器上的活动。每行日志记录代表一个请求或命中，这使得日志文件成为了记录TCP事件的消息传递情况和错误事件的处理结果的重要工具。在本项目中，日志文件会在服务器启动时生成，并且每条记录都是从socket.recv()函数接收到的数据。日志功能在处理历史数据、诊断问题的追踪和理解系统活动方面发挥着至关重要的作用。

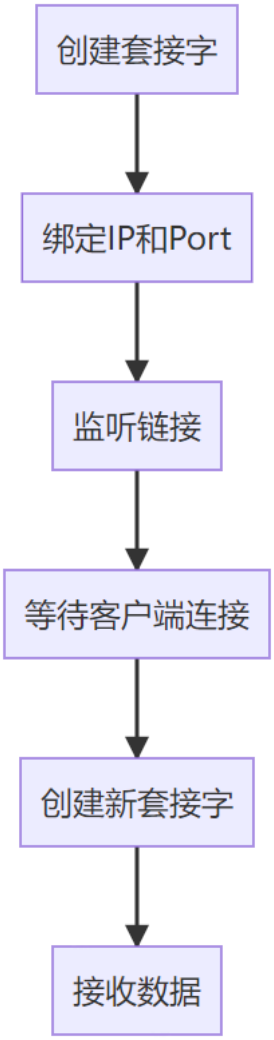
g) 性能测试分析

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

2. Design

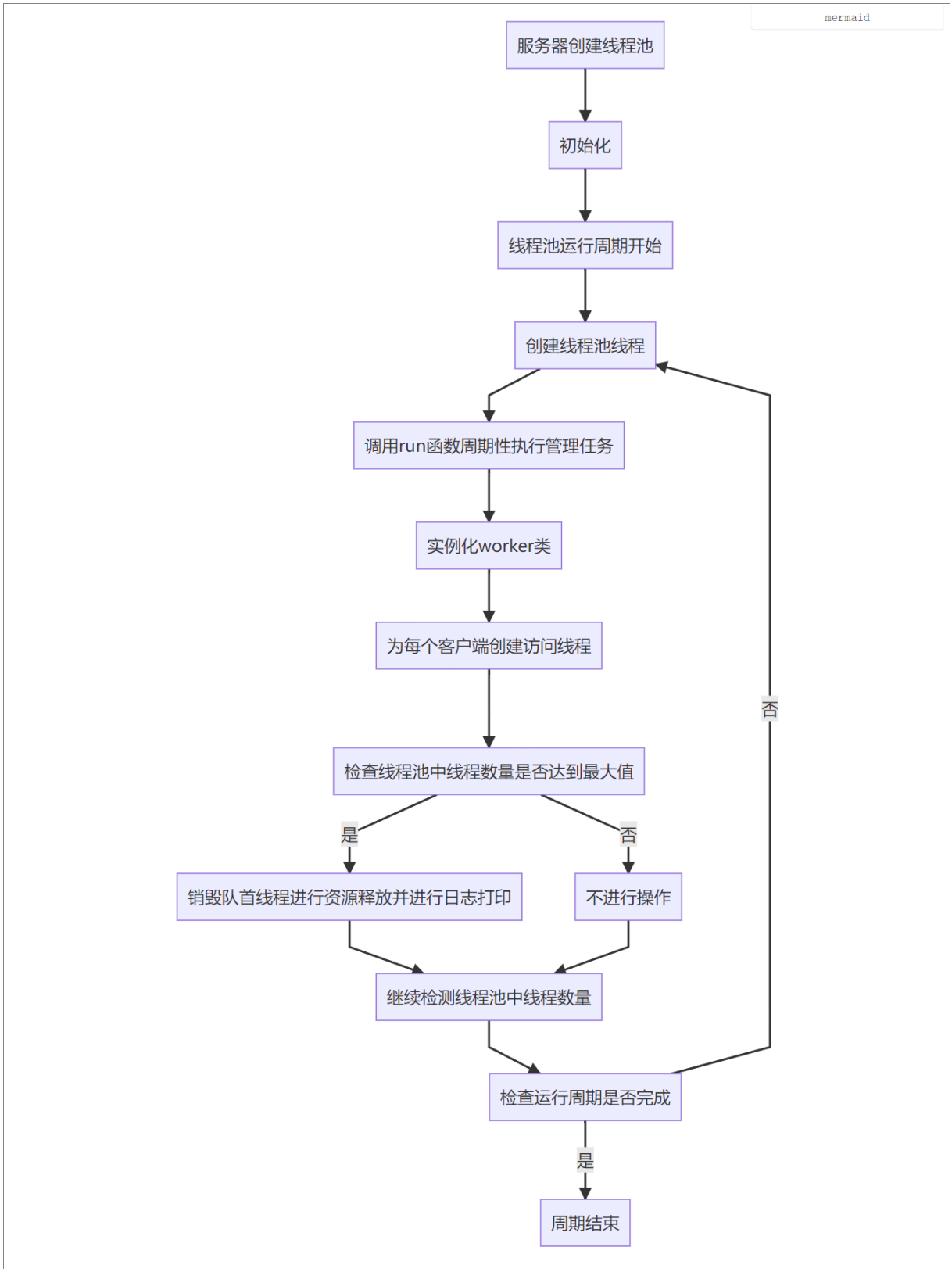
a) TCP Socket

建立TCP-socket连接的步骤如下：
1） 使用socket()创建套接字
2） 使用bind()绑定本地IP和端口信息
3） 使用listen()使套接字可以被动链接，并能够接收来自其他人的链接
4） 使用accept()等待客户端的连接。如果一个新的客户端连接到服务器，则会生成一个专门为该客户端服务的新套接字
5） 然后，使用recv()函数接受对方发送的数据



b) Thread Pool (Connection timeout handling)

线程池管理的一般流程如下：
（1）服务器创建线程池，自动调用__init__()函数进行线程池初始化，开始线程池运行周期。
（2）创建的线程池线程调用run()函数周期性开始执行管理任务。
（3）针对每个请求进行worker类的实例化，为每一个客户端创建访问线程。
（4）检查线程池中的线程数量是否到达最大值，如果到达最大值，则将队首线程进行销毁进行资源释放并进行相关日志打印，否则不进行操作。
（5）如果该运行周期完成，则该周期结束，否则继续进行上述的检测。



HTTP1.0的请求和响应流程如下：

(1) 在端口8888上创建一个服务器套接字。

(2) 输入一个无限循环：

A. 接受下一个连接。

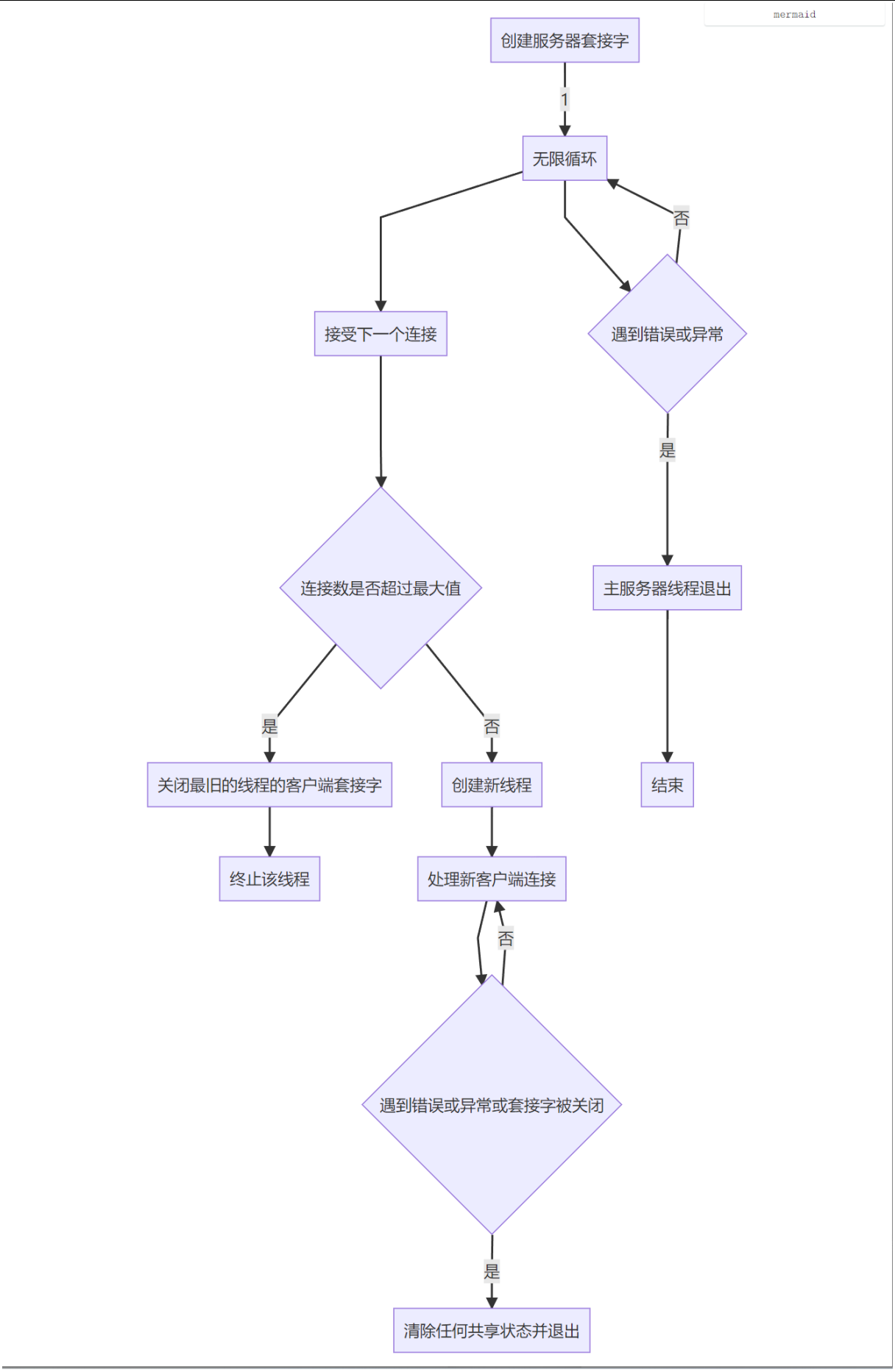
如果已经存在最大连接，请关闭最旧的线程的客户端套接字以终止该线程。这将导致工作线程在下次尝试从套接字读取或写入套接字时收到错误或异常。然后工作线程应该退出。如果同时打开的连接总数超过maxConnections，则服务器将开始关闭最旧的打开连接

连接及其关联的工作线程将失效。终止的工作线程应该清理任何共享状态，并在终止之前关闭其套接字端。请记住，连接也可以通过其他方式关闭。例如，客户端可以关闭连接。在这种情况下，关联的工作线程应该检测到套接字已关闭，清除任何共享状态，然后退出。

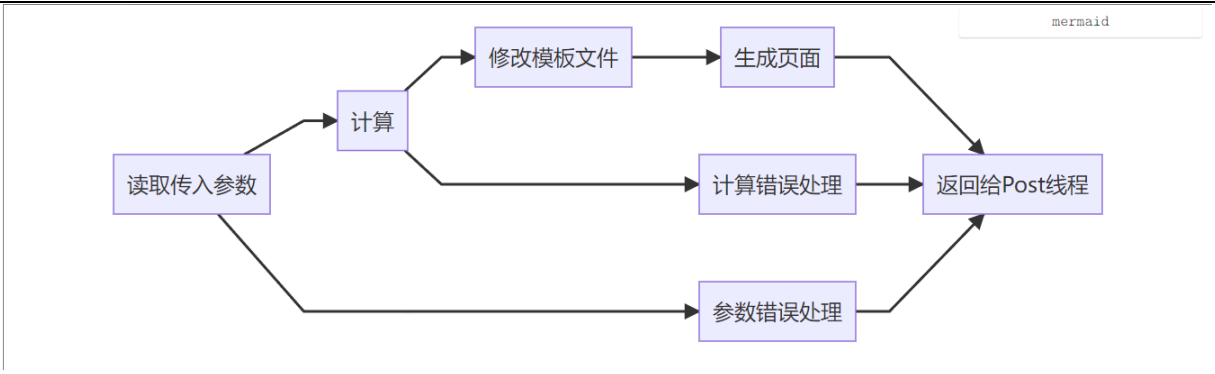
B. 创建一个新线程来处理新客户端的连接，并将accept返回的客户端套接字传递给它。

只有在遇到错误或异常时，主服务器线程才应该退出。

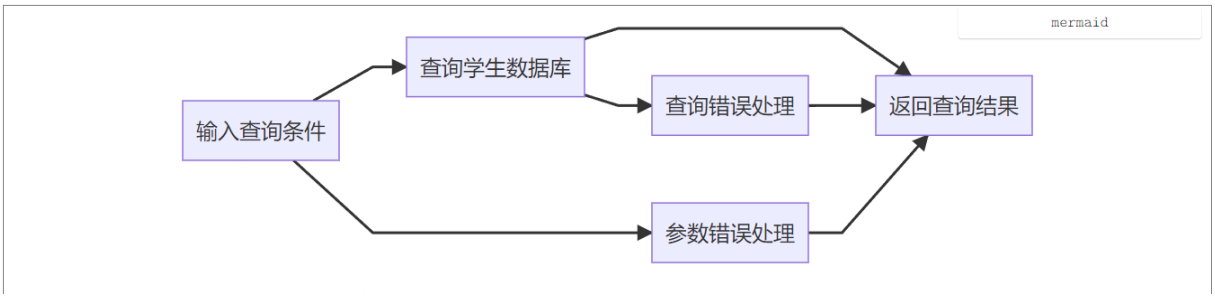
工作线程可以处于无限循环中，只有当它遇到错误或异常，或者套接字被主服务器线程或客户端关闭时，才会退出。否则，工作线程将继续处理来自客户端的HTTP请求。



计算器测试：采用python实现计算器的功能，读取传入的参数，相加后修改模板文件，将生成页面返回给Post线程。



数据库查询测试：采用python实现对学生数据库的查询。学生数据库中包含学生的姓名、学号、班级，可以支持三种属性的查询，也可以只使用其中的一种或者两种属性进行查询。



3. Development and Implementation

a) Web Server and TCP Socket

```
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)  # 允许地址重用
server_socket.bind(address)
server_socket.settimeout(60)  # 防止服务器长时间无响应
server_socket.listen(max_connection)  # 最大连接数
```

使用 recv 实现了一个 TCP 服务器的初始化操作，包括创建 socket 对象、绑定 IP 地址和端口号、设置 socket 选项、设置超时时间以及开始监听客户端连接等。具体来说，该服务器会在本地地址 0.0.0.0 上绑定一个指定端口，然后等待客户端的连接请求。当有新的客户端连接时，服务器将会接受并处理其请求，最大连接数由 max_connection 参数指定。同时，该服务器会设置 SO_REUSEADDR 选项，允许在 socket 关闭后立即重新使用该端口，而不需要等待一段时间。这段代码中的 socket 选项 SO_REUSEADDR 对于快速重启服务器或在同一端口上运行多个服务器实例非常有用。()函数接受对方发送的数据

```
self.socket = tasks.get()  # 从任务队列中获取 socket 对象
working_thread.append(self)  # 将当前线程添加到工作线程列表中
sema.release()  # 释放信号量
message = self.socket.recv(8000).decode("utf-8")  # 接收请求信息
message = message.splitlines()
```

接收客户端发送的请求信息并进行处理。其中，self.socket 变量表示当前线程正在处理的客户端socket对象，tasks 是一个任务队列，存储了需要处理的socket对象。由于多个客户端同时连接到服务器，因此需要使用多线程来并发处理请求。使用信号量可以确保每个线程都能够获取到一个任务并开始处理它。

b) Thread Pool Management

线程池管理部分我们使用python进行实现，使用pycharm作为开发工具。我们按照功能将该模块分为了四部分。关键代码及相关功能如下展示：

（1）线程池初始化。为实现该功能，我们在thread_pool下重写了__init__类，将线程池线程设置为守护线程，并将线程启动，初始化log_name。代码如下：

```
def __init__(self, log_name):
    threading.Thread.__init__(self)
    self.setDaemon(True)
    self.start()
    self.log_name = log_name
```

（2）线程池监测。每隔一段时间运行监控过程，如果此时工作线程池已满并且任务队列中有尚未处理的任务的时候，取出运行线程队列的队首线程，释放该线程所占用的资源，然后让该线程处理新的任务。代码实现如下：

```
# 循环检查任务队列和工作线程状态
while True:
    for i in range(10):
        if (len(working_thread) == max_connection
                and max_connection != 0 and (not tasks.empty())):
            print("shutdown")
            working_thread[0].restart()
    sema.acquire(timeout=1)
```

（3）关闭线程。关闭线程时调用down函数，将对应的文件句柄、套接字、线程全部关闭。代码实现如下：

```
# 重启方法，关闭相关资源
def restart(self):
    if self.file_handle is not None:
        self.file_handle.close()
        self.file_handle = None
    if self.socket is not None:
        try:
            self.socket.shutdown(2)
            self.socket.close()
        except Exception as e:
            print("socket error:", e)
        self.socket = None
    if self.proc is not None and self.proc.poll() is not None:
        self.proc.kill()
        self.proc = None
```

c) HTTP Request and Response

http 请求与响应处理在 worker.py 中，处理的流程可以简单分为两部分：一是对请求判断和预处理，二是对不同请求的处理。

请求判断和预处理的部分如下：

```
if message:
    key_mes = message[0].split()
else:
    self.restart()  # 如果请求信息为空，则重启线程
    continue
if len(key_mes) <= 1:
    self.restart()  # 如果请求信息格式不正确，则重启线程
    continue
file_name = "index.html"
if key_mes[1] != "/":
    file_name = key_mes[1][1:]
```

若没有获取到资源就取消处理进程，若解析的请求行正确，就判断当前请求行的请求资源地址，若是单纯“/”就转向首页地址，否则资源地址为请求行指向的资源地址。

对不同请求的处理如下：

```
try:
    if key_mes[0] == 'GET':
        self.get(file_name)
    elif key_mes[0] == 'POST':
        self.post(file_name, message[-1])
    elif key_mes[0] == 'HEAD':
        self.head(file_name)
    else:
        content = b"HTTP/1.1 400 Bad Request\r\nContent-Type: text/html\r\n"
        self.socket.sendall(content)
except Exception as e:
    print("reason:", e)  # 输出异常信息
```

这部分代码是总的判断部分，判断请求具体时间 get、post 还是 head，然后转向对应的处理，如果不是以上三种之一，那么返回 400 状态码以及对应的响应头信息。

Get 请求处理：

```
def get(self, file_name, is_head=False):  # 处理 GET 请求
    if os.path.isfile(file_name):  # 文件存在
        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  # 状态码为 200
    else:  # 文件不存在
        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  # 状态码为 404
    content += b'\r\n'
    self.socket.sendall(content)

    file_size = 0

    if not is_head:  # 不是 HEAD 请求，发送文件内容
        self.file_handle = open(file_name, "rb")
        for line in self.file_handle:
            self.socket.sendall(line)

        file_size = os.path.getsize(file_name)

    print(self.log_name)  # 输出日志文件名
    self.write_log(file_size)  # 写入日志
```

get 请求部分主要涉及的是对资源的查询，当请求的资源在本机资源中可以找到时，就包装好响应信息，将响应发送回去。若本地中并不存在资源，则返回 404 状态码和对应的 404 图片相应资源信息。Head 实质上是简化的 get，所以将 head 也在这里进行了处理。

Post 请求处理：

```
def post(self, file_name, args):  # 处理 POST 请求
    command = 'python ' + file_name + ' "' + args + '" "' + self.socket.getsockname(
    )[0] + '" "' + str(self.socket.getsockname()[1]) + '"'  # 执行请求的命令
    self.proc = subprocess.Popen(command,
                                 shell=True,
                                 stdout=subprocess.PIPE)
    self.proc.wait()

    file_size = 0

    if self.proc.poll() == 2:  # 执行命令失败
        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  # 状态码为 403
    else:  # 执行命令成功
        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  # 状态码为 200
    self.socket.sendall(content)
    print(self.log_name)  # 输出日志文件名
    self.write_log(file_size)  # 写入日志
```

Post 请求在本项目中作用是执行具体的处理流程，因而在处理 post 中主要做的是使用系统 shell 指令运行 python 程序，将运行得到的结果进行返回。

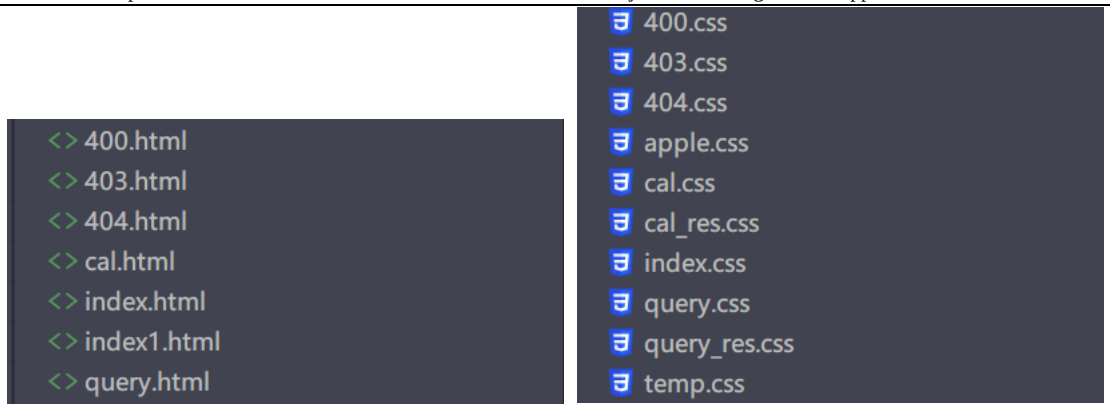
至此，http 的请求与响应处理部分结束。

Head 请求处理：

```
def head(self, file_name):  # 处理 HEAD 请求
    if os.path.isfile(file_name):  # 文件存在
        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  # 状态码为 200
    else:  # 文件不存在
        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)
    self.write_log(0)  # 传入文件大小为 0，表示只是响应头
```

d) Static and Dynamic Page

为网页界面编写了前端静态文件：



e) CGI

计算器测试：采用python实现计算器的功能，读取传入的参数与运算符，运算后修改模板文件，将生成页面返回给Post线程。

输入：传入的参数与运算符（表单的数据）

输出：运算结果

获取运算数与运算符：

```
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
    res = res.replace("$a", a)
    res = res.replace("$b", b)

    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)))

    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)
```

数据库查询测试：采用python实现对学生数据库的查询。学生数据库中包含学生的姓名、学号、班级，可以支持三种属性的查询，也可以只使用其中的一种或者两种属性进行查询。

输入：查询的条件

输出：拼接的sql语句以及sql语句运行的查询结果

获取查询条件：

```
import sqlite3

import sys

try:
    ini = sys.argv[1]
    hostname = sys.argv[2]
    port = sys.argv[3]

    ini = ini.split("&")
    student_id = ini[0].split("=")[1]
    student_name = ini[1].split("=")[1]
    student_class = ini[2].split("=")[1]

    db = sqlite3.connect('data\Student_data.db')
    cursor = db.cursor()

    sql = "SELECT * from student"

    id_flag = 1
    id_name = 1
    id_class = 1
```

```
if student_id == "":
    id_flag = 0
if student_name == "":
    id_name = 0
if student_class == "":
    id_class = 0

if id_flag == 0 and id_name == 0 and id_class == 0:
    pass
else:
    sql += " where"
    if id_flag == 1:
        sql += " id = " + student_id

    if id_flag == 0 and id_name == 1:
        sql += " name = \'" + student_name + "\' "
    elif id_flag == 1 and id_name == 1:
        sql += " and name = \'" + student_name + "\' "

    if id_flag == 0 and id_name == 0 and id_class == 1:
        sql += " class = \'" + student_class + "\' "
    elif id_class == 1:
        sql += " and class = \"" + student_class + "\" "

sql += ";"

try:
    cursor.execute(sql)
except:
    print(sql)

data = cursor.fetchall()
res = ""

# print()

with open("cgi-bin/query_res.html", "r", encoding="utf-8") as f:
    # pass
    print(res)

f.close()

with open("cgi-bin/query_res.html", "r", encoding="utf-8") as f:
```

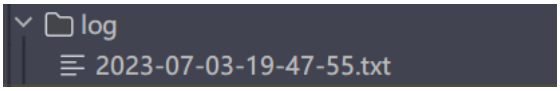
```
            for line in f:
                res += line

        student_data = ''
        for student in data:
            temp = "<tr>"
            temp += "<th>" + str(student[0]) + "</th>"
            temp += "<th>" + student[1] + "</th>"
            temp += "<th>" + student[2] + "</th>"
            temp += "</tr>\n"
            student_data += temp
        res = res.replace("$data", student_data)
        res = res.replace("$message", sql)
        res = res.replace("$test", res)

        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)
```

f) Log File

Log文件的生成在server.py文件中，每次server.py文件被执行就是服务器的一次启动过程，即服务器每次启动都会生成本此运行的日志文件，日志文件存储在”log”目录下，以当前服务器主机的时间命名 +“.txt”命名:

```
now_time = time.localtime()
log_dir = 'log'
if not os.path.exists(log_dir):
    os.makedirs(log_dir)
log_name = os.path.join(log_dir, time.strftime("%Y-%m-%d-%H-%M-%S.txt",
time.localtime()))
```

日志文件



日志详情

```
192.168.56.1--[2023-7-3-19-55-55] GET  / 1298 200
192.168.56.1--[2023-7-3-19-55-55] GET  /css/apple.css 1767 200 http://192.168.56.1:8888/
192.168.56.1--[2023-7-3-19-55-55] GET  /css/temp.css 1822 200 http://192.168.56.1:8888/
192.168.56.1--[2023-7-3-19-55-55] GET  /js/400.js 2194 200 http://192.168.56.1:8888/
192.168.56.1--[2023-7-3-19-55-56] GET  / 1298 200
192.168.56.1--[2023-7-3-19-55-56] GET  /css/temp.css 1822 200 http://192.168.56.1:8888/
192.168.56.1--[2023-7-3-19-55-56] GET  /css/apple.css 1767 200 http://192.168.56.1:8888/
192.168.56.1--[2023-7-3-19-55-56] GET  /js/400.js 2194 200 http://192.168.56.1:8888/
192.168.56.1--[2023-7-3-19-55-56] GET  / 1298 200
192.168.56.1--[2023-7-3-19-55-56] GET  /css/apple.css 1767 200 http://192.168.56.1:8888/
192.168.56.1--[2023-7-3-19-55-56] GET  /css/temp.css 1822 200 http://192.168.56.1:8888/
192.168.56.1--[2023-7-3-19-55-57] GET  / 1298 200
192.168.56.1--[2023-7-3-19-55-57] GET  /js/400.js 2194 200 http://192.168.56.1:8888/
192.168.56.1--[2023-7-3-19-55-57] GET  /css/apple.css 1767 200 http://192.168.56.1:8888/
192.168.56.1--[2023-7-3-19-55-57] GET  / 1298 200
192.168.56.1--[2023-7-3-19-55-57] GET  /css/apple.css 1767 200 http://192.168.56.1:8888/
192.168.56.1--[2023-7-3-19-55-57] GET  /css/temp.css 1822 200 http://192.168.56.1:8888/
192.168.56.1--[2023-7-3-19-55-57] GET  /js/400.js 2194 200 http://192.168.56.1:8888/
192.168.56.1--[2023-7-3-20-0-59] GET  /cal.html 1994 200 http://192.168.56.1:8888/
192.168.56.1--[2023-7-3-20-1-0] GET  /css/cal.css 1800 200 http://192.168.56.1:8888/cal.html
192.168.56.1--[2023-7-3-20-1-4] POST  /cgi-bin/cal.py 1082 200 http://192.168.56.1:8888/cal.html
192.168.56.1--[2023-7-3-20-1-4] GET  /css/cal_res.css 982 200 http://192.168.56.1:8888/cgi-bin/cal.py
192.168.56.1--[2023-7-3-20-1-7] GET  /query.html 905 200 http://192.168.56.1:8888/
192.168.56.1--[2023-7-3-20-1-7] GET  /css/query.css 1177 200 http://192.168.56.1:8888/query.html
192.168.56.1--[2023-7-3-20-1-13] POST  /cgi-bin/query.py 2504 200 http://192.168.56.1:8888/query.html
192.168.56.1--[2023-7-3-20-1-13] GET  /css/query_res.css 1274 200 http://192.168.56.1:8888/cgi-bin/query.py
192.168.56.1--[2023-7-3-20-46-37] GET  /images/calc.png 735 404 http://192.168.56.1:8888/
192.168.56.1--[2023-7-3-20-46-37] GET  /css/temp.css 1822 200 http://192.168.56.1:8888/
192.168.56.1--[2023-7-3-20-46-37] GET  /css/style.css 735 404 http://192.168.56.1:8888/cal.html
192.168.56.1--[2023-7-3-20-46-37] GET  /images/query.png 735 404 http://192.168.56.1:8888/
192.168.56.1--[2023-7-3-20-46-37] GET  /css/img/rocket.jpeg 735 404 http://192.168.56.1:8888/
192.168.56.1--[2023-7-3-20-46-37] GET  /css/img/beans.jpeg 735 404 http://192.168.56.1:8888/
192.168.56.1--[2023-7-3-20-46-37] GET  /css/img/p1.jpg 735 404 http://192.168.56.1:8888/
192.168.56.1--[2023-7-3-20-46-37] GET  /css/img/birds.jpeg 735 404 http://192.168.56.1:8888/
```

4. System Deployment, Startup, and Use

IDE：PyCharm
Python 3.10

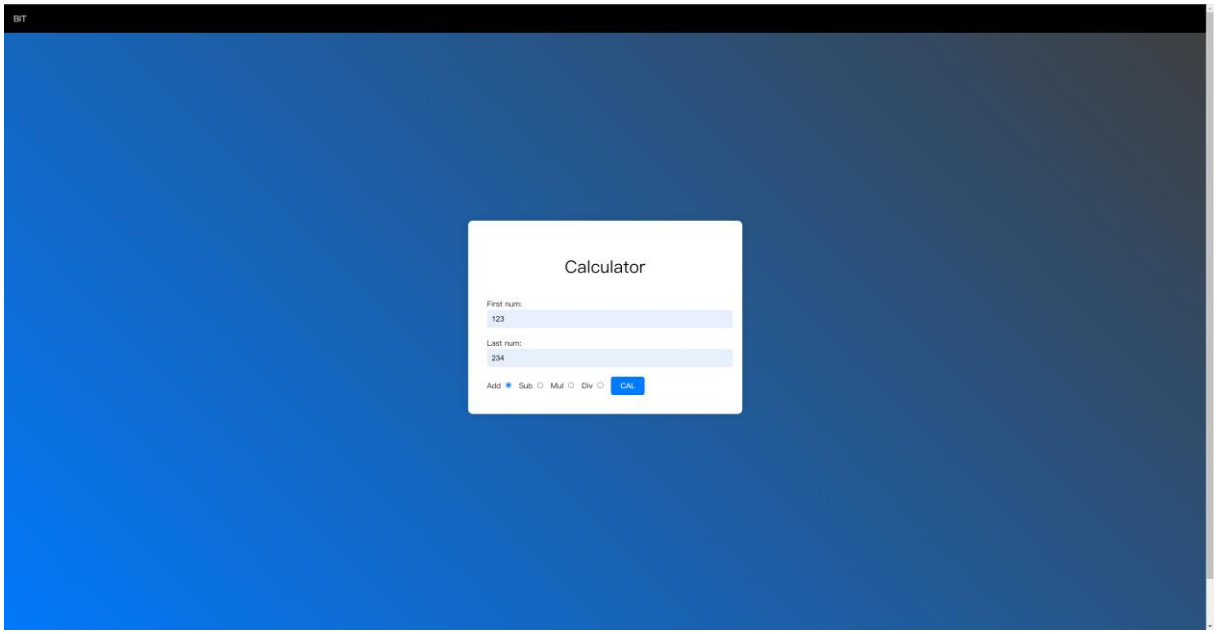
命令行调用：在CGI文件夹中打开命令行：输入python server.py，按照要求输入最大连接数，

```
终端  本地 ×  +  ⌄
Windows PowerShell
版权所有（C） Microsoft Corporation。保留所有权利。
安装最新的 PowerShell，了解新功能和改进！ https://aka.ms/PSWindows
(venv) PS C:\Users\DOCTORY\Desktop\CGI> py

                                    python .\server.py
请输入最大连接数：127
192.168.56.1:8888
C:\Users\DOCTORY\Desktop\CGI\server.py:37: DeprecationWarning: setDaemon() is deprecated, set the daemon attribute instead
  self.setDaemon(True)
```

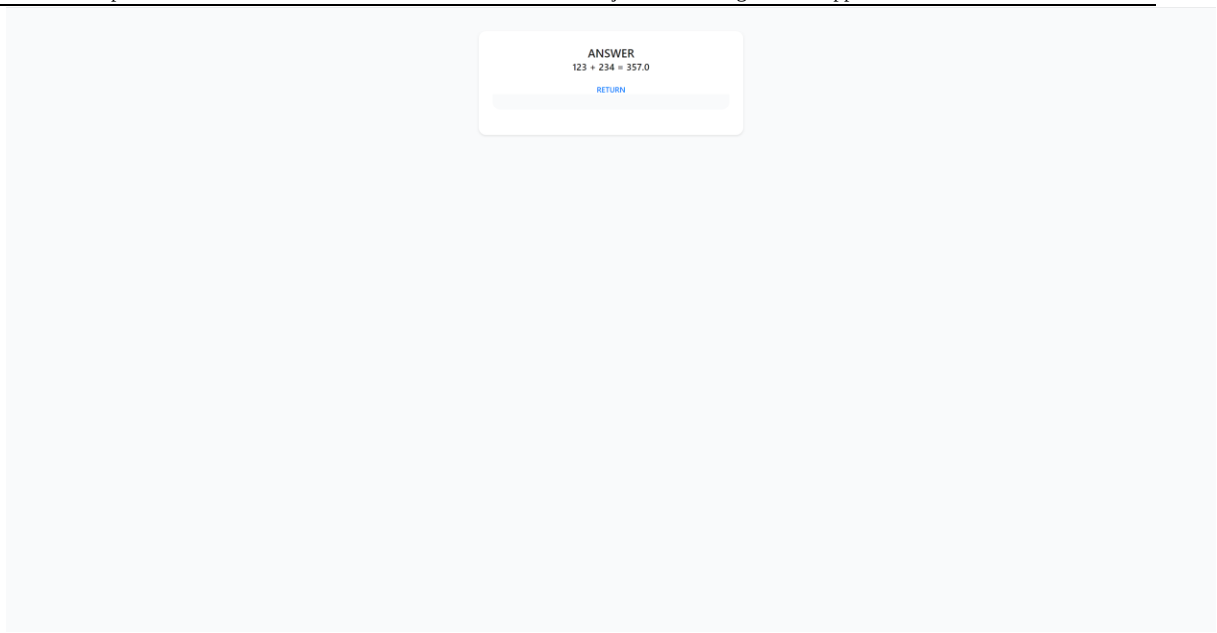
a) 系统的使用方法

在浏览器中打开

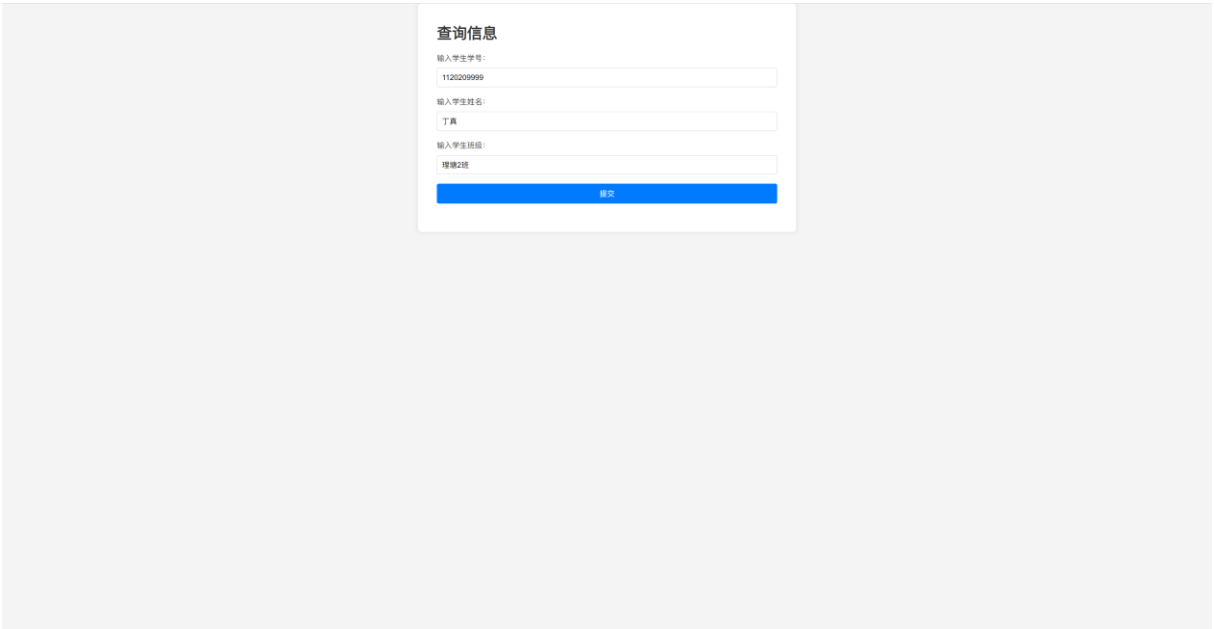
如图所示，点击两侧的按钮，调用CGI程序，左侧的计算器按钮，根据要求输入数字何计算方式，调用计算器程序cgi-bin/cal.py，如下所示：



输入两个数字，选择运算方法，点击CAL运行，具体显示结果位于测试中。在这个页面中，请按要求在num框中输入正确的数字，并选中一个运算方法，否则会返回到400 Bad Request界面。



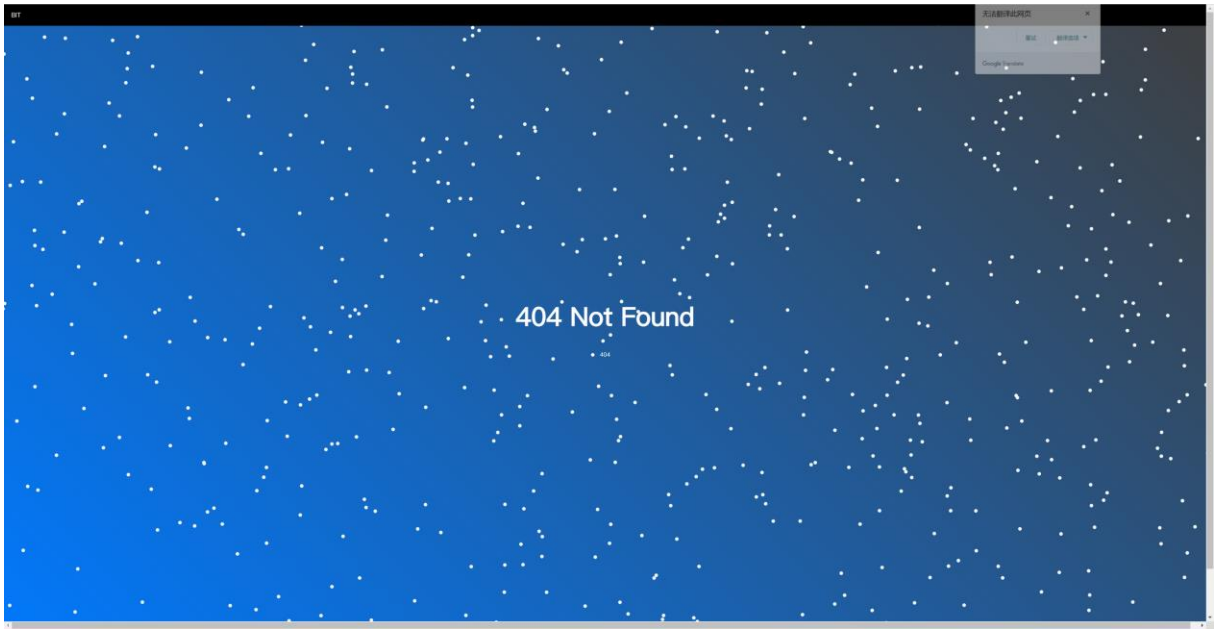
点击右侧的查询按钮，针对学生数据库进行信息查询，点击提交后调用查询程序cgi-bin/query.py，如下所示：



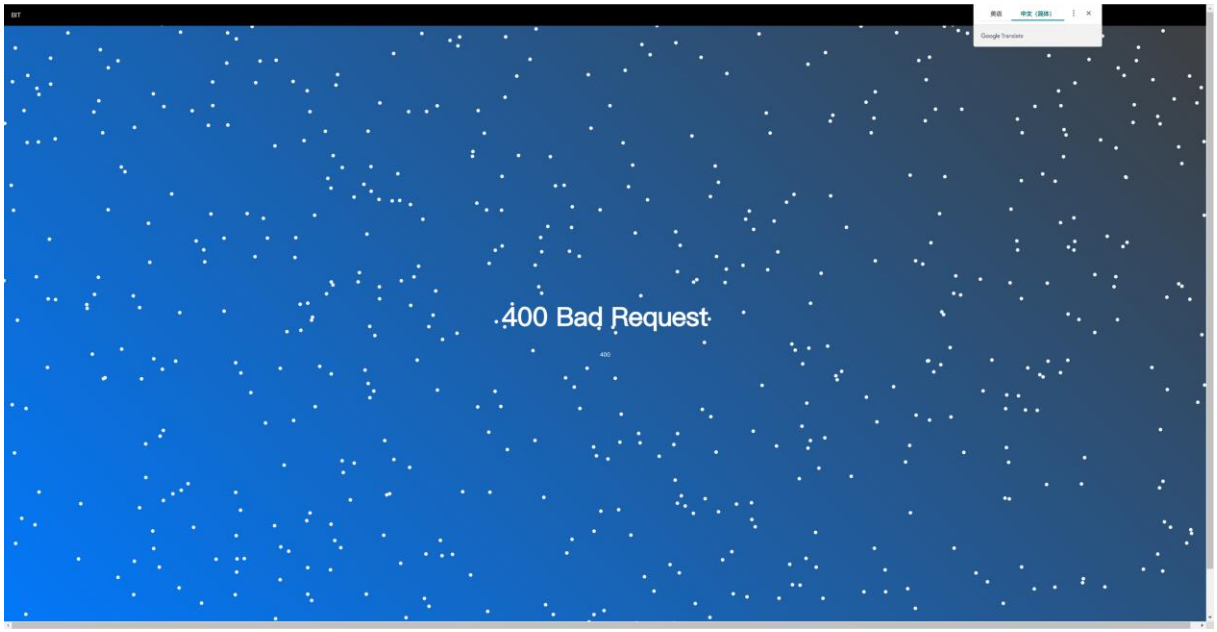
在进行查询时，在方框中输入查询的条件，如果输入为空，则无此查询条件。
同样，输入不符合要求或者sql语句调用错误时，会返回到400 Bad Request界面。

5. System Test

6. 当输入一个不存在的网址，会弹出 404 Not Found，例如输入 http://192.168.56.1:8888/cgi-bin



7. 当输入不符合规则，会弹出 400 Bad Request：



8. Performance and Analysis

Performance test and result analysis, etc. Show results in data sheet and figures.

9. Summary or Conclusions

该多线程Web服务器整体实现了以下功能:

1) 建立了基于 TCP 的套接字连接,可以与 Web 浏览器建立连接;
2) 实现了多线程处理,每个请求分配一个独立线程处理,从而支持并发处理多个请求;
3) 支持静态 HTML 页面和动态 CGI 页面,例如计算器和数据库查询脚本;
4) 可以处理 GET、POST 和 HEAD 请求,并返回相应的响应状态码和内容;

5) 使用线程池管理线程,能够有效利用线程资源,同时也可以限制最大连接数;
6) 生成日志记录服务器运行情况,方便问题排查和性能分析;

该系统实现了Web服务器的主要功能,但还需要进一步改进优化,以提高吞吐量和响应时间。

10. References

1) 王德福。Python 多线程服务器编程。《Python 实战》,2015。
2) 李述鹭。Web 服务器适用场景分析与简单实现。《轻量级 Web 服务器的开发》,2012。
3) 贾帅军等。基于 CGI 技术的基于 Python 的 GIS 信息服务系统设计与实现。《计算机系统与微电子》,2012。
4) 陈少平。基于 python 多线程技术的 web 服务器设计与实现。《电脑知识与技术》,2016。
5) 赵鹏程。基于 SQLAlchemy 的 python Web 应用程序设计实战。 机械工业出版社,2014.

11. Comments

任务量大，还是比较适合小组作业，或者把课程安排在大二比较有时间仔细完成不断完善。本次工程项目只是运行起来的最低要求，性能表现上与实际投入使用有较大的差距。不过借此实验也完善了计算机科学技术栈，对计算机网络的实操有了更全面的认知。