**Computer Network Course Practice**

**Part 2: Building a CGI-Support Multi-Threaded Web Server**

**Report**

|  |  |  |  |
| --- | --- | --- | --- |
| **No.** | **Student ID** | **Name** | **Role** |
| 1 | 1120191600 | 张驰 | Leader |
| 2 | 1120181123 | 李彧泽 | Member |
| 3 | 1120181124 | 张洪璨 | Member |
| 4 | 1120191548 | 于松涛 | Member |
| 5 | 1120191586 | 李鹏 | Member |
| 6 | 1120191866 | 魏慧聪 | Member |
| 7 | 1120192050 | 傅裕嵘 | Member |

**School of Computer**

**Beijing Institute of Technology**

**October, 2022**

1. **Requirements**

在本项目中，要求构建一个支持CGI测试的多线程Web服务器。本项目构建一个能够并行处理多个同时服务请求的多线程Web服务器。该Web服务器能够向Web浏览器提供静态和动态页面用于测试。

* 1. **HTTP**

使用HTTP1.0，服务器为网页每个组件发送单独的HTTP请求。该多线程服务器能处理来自客户端的get post head请求。同时，请求应该能够返回适当的状态代码，包含200、400、403、404，并且带有返回消息的正文。

* 1. **Post and Address**

服务器将能够并行处理多个同时的服务请求。在主线程中，服务器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。

* 1. **Max Connections and Thread pool**

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。

* 1. **CGI**

CGI提供了一个简单的方式用于构建动态网页。

* 1. **Log File**

服务器应该提供一种机制来记录服务器上发生的所有事情。 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。日志文件中的每一行代表一个请求（命中）。

* 1. **Performance Analysis**

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。

1. **Analysis(分析模块主要的完成内容 对该模块的理解)**

Understanding of Web server, static and dynamic page, cgi, and HTTP, TCP. Main issues, key functions, etc.

* 1. **TCP Socket**

在本次项目中，我们的client和server之间使用的连接为TCP连接，他是面向连接的，面向流的，提供高可靠性服务。收发两端都要有一一成对的socket。TCP socket是本次项目中连接的底层原理。

* 1. **Thread Pool Management**

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，所以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对所有的客户线程进行统一管理。

线程池本身同样也作为一个独立的线程运行，但不同与客户端线程的是，我们将其设置为守护线程（setDaemon）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. **HTTP Request and Response**

在服务器中，一个关键点是请求的处理和响应的返回。在http请求的处理中，主要需要做的事情是，接受socket套接字中传递过来的数据，将之转化为可以识别的utf-8编码方式。之后根据报文头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. **Static and Dynamic Page**

这一部分主要是网页的CSS和HTML实现，是整个系统的用户界面。

重点模块是网页，计算器界面和数据库查询界面。

选用CSS和HTML语言完成网页界面的设计

* 1. **CGI**

CGI接收网页提交的表单、主机名与端口，表单内容包括运算数与运算符或者查询条件，分别由不同的模块处理。Cal.py负责运算，根据运算符进行相应的运算后将结果替换显示到网页上。Quary.py负责从数据库中查询，根据查询条件编写sql语句，cursor.execute()执行sql语句从数据库中查到所需数据，最后将结果替换显示到网页上。

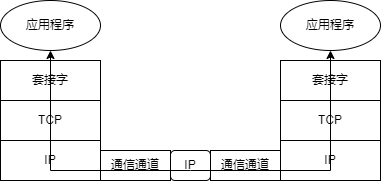
* 1. **Log File**

在一个服务器中日志功能是必不可少的，日志文件用于记录TCP事件的消息传递情况和错误事件的处理结果，具有处理历史数据，诊断问题的追踪以及理解系统的活动等重要作用。本项目中，日志文件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()函数从连接的套接字接收的数据。

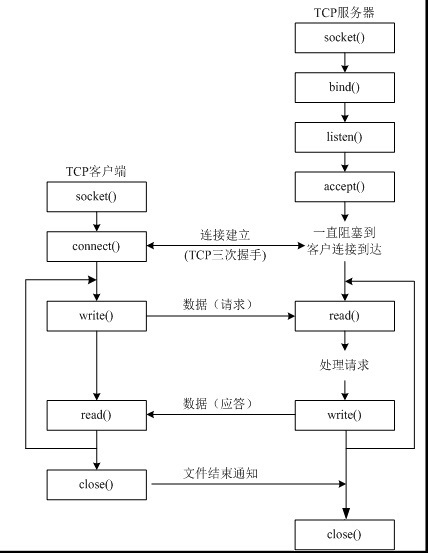
1. **Design**
   1. **TCP Socket**

建立TCP-socket连接的步骤如下：

1. 使用socket()创建一个套接字
2. 使用bind()绑定本地的IP和Port信息
3. 使用listen()使套接字变为可以被动链接，可以接收别人的链接
4. 使用accept()等待客户端的连接，如果有新的客户端来链接服务器，就产生一个新的套接字专门为这个客户端服务
5. 之后使用recv()函数接受对方发送的数据

****

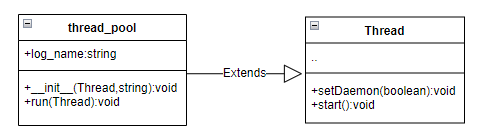
模型图



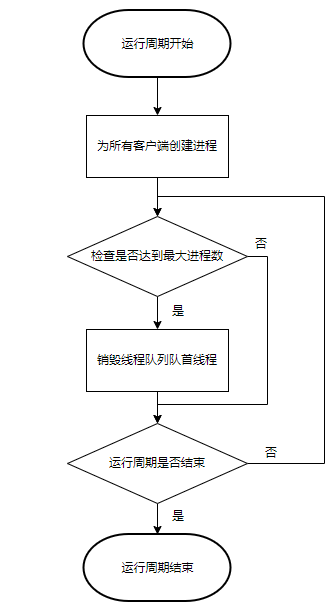
流程图

* 1. **Thread Pool (Connection timeout handling)**

线程池管理的相关过程的类图和流程图如下：

****

**线程池类图**

****

**线程池管理流程图**

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。

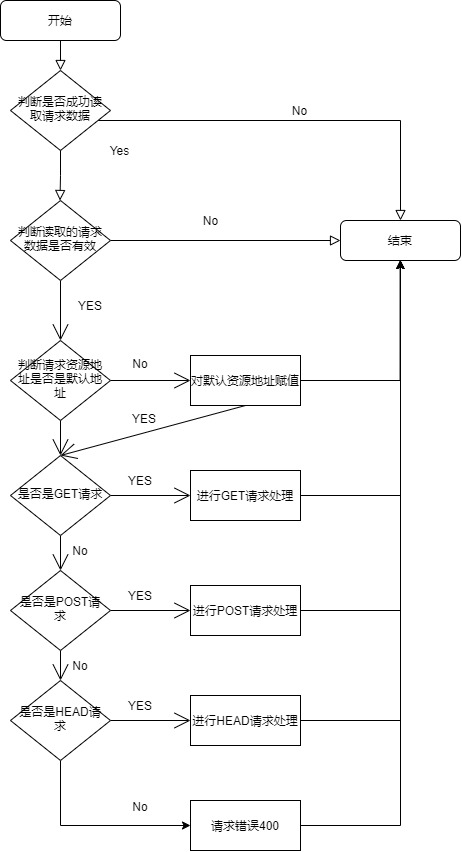
（3）针对每个请求进行worker类的实例化，为每一个客户端创建访问线程。

（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。

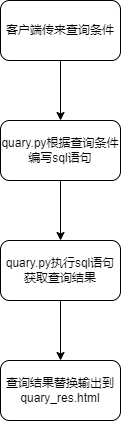
* 1. **HTTP Request and Response**

该部分流程图如下所示：

****

* 1. **CGI**

计算器的CGI和数据库查询的CGI，流程图如下所示：



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。

1. **Development and Implementation**
   1. **Web Server and TCP Socket**

该模块使用python，基于pycharm进行编写，关键代码及其流程注释展示如下：

1. server\_socket = socket.socket(socket.AF\_INET, socket.SOCK\_STREAM)
2. host\_name = socket.gethostname()
3. host\_name = socket.gethostbyname(host\_name)
4. server\_socket.setsockopt(socket.SOL\_SOCKET, socket.SO\_REUSEADDR, 1)

使用socket()创建WebServer的服务器套接字

1. address = ("0.0.0.0", port)
2. server\_socket.bind(address)

绑定本地IP和Port信息。

1. server\_socket.settimeout(60)
2. server\_socket.listen(max\_connection)

使用listen()使套接字变为可以被动链接，可以接收别人的链接

1. while True:
2. try:
3. client, addr = server\_socket.accept()
4. print("recv: ",client.getpeername(),client.getsockname())
5. tasks.put(client)
6. except socket.timeout:
7. print("main server timeout")

当socket接收到网络请求时，调用accept函数会得到新的用于和该地址交互的socket套接字，将该套接字放入到我们的任务队列中，交由worker.py中的解析函数对请求进行解析以及后续的处理。

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

使用recv()函数接受对方发送的数据

* 1. **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：

1. def \_\_init\_\_(self, log\_name):
2. threading.Thread.\_\_init\_\_(self)
3. self.setDaemon(True)
4. self.start()
5. 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：

1. while True:
2. for i in range(10):
3. if (len(working\_thread) == max\_connection
4. and max\_connection != 0 and (not tasks.empty())):
5. print("shutdown")
6. working\_thread[0].restart()

（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：

1. def restart(self):
2. if (self.file\_handle != None):
3. self.file\_handle.close()
4. self.file\_handle = None
5. if (self.socket != None):
6. try:
7. self.socket.shutdown(2)
8. self.socket.close()
9. except Exception as e:
10. print("socket error:", e)
11. self.socket = None
12. if (self.proc != None and self.proc.poll() != None):
13. self.proc.kill()
14. self.proc = None

（4）日志更新。在管理线程池过程中，对当前线程池状态、线程关闭、套接字出错等信息进行进行日志的更新。代码实现如下。

1. if (len(working\_thread) == max\_connection
2. and max\_connection != 0 and (not tasks.empty())):
3. print("shutdown")
4. print("now working thread: " + str(working\_thread\_cnt) +
5. " ; free thread: " +
6. str(max\_connection - working\_thread\_cnt) +
7. " ; now waiting request: " + str(tasks.qsize()))
8. try:
9. self.socket.shutdown(2)
10. self.socket.close()
11. except Exception as e:
12. print("socket error:", e)
    1. **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：

1. if (message):
2. key\_mes = message[0].split()
3. else:
4. self.restart()
5. continue
6. if (len(key\_mes) <= 1):
7. self.restart()
8. continue
9. file\_name = "index.html"
10. if (key\_mes[1] != "/"):
11. 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：

1. if (key\_mes[0] == 'GET'):
2. self.get(file\_name)
3. elif (key\_mes[0] == 'POST'):
4. self.post(file\_name, message[-1])
5. elif (key\_mes[0] == 'HEAD'): *# 轻量版get*
6. self.get(file\_name, True)
7. else:
8. content = b"HTTP/1.1 400 Bad Request\r\nContent-Type: text/html\r\n"
9. self.socket.sendall(content)

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：

1. def get(self, file\_name, is\_head=False):
2. if (os.path.isfile(file\_name)):
3. file\_suffix = file\_name.split('.')
4. file\_suffix = file\_suffix[-1].encode()
5. content = b"HTTP/1.1 200 OK\r\nContent-Type: text/" + file\_suffix + b";charset=utf-8\r\n"
6. self.status\_code = 200
7. else:
8. content = b"HTTP/1.1 404 Not Found\r\nContent-Type: text/html;charset=utf-8\r\n"
9. file\_name = "404.html"
10. self.status\_code = 404
11. content += b'\r\n'
12. self.socket.sendall(content)
13. file\_size = 0
14. if not is\_head:
15. self.file\_handle = open(file\_name, "rb")
16. for line in self.file\_handle:
17. self.socket.sendall(line)
18. file\_size = os.path.getsize(file\_name)

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。

Get请求处理：

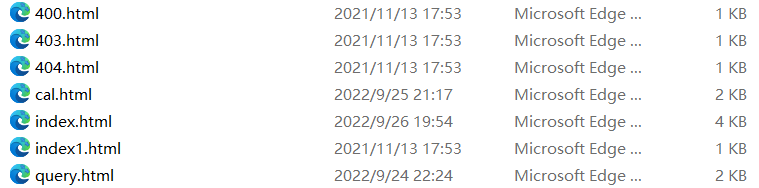
1. def post(self, file\_name, args):
2. command = 'python ' + file\_name + ' "' + args + '" "' + self.socket.getsockname(
3. )[0] + '" "' + str(self.socket.getsockname()[1]) + '"'
4. self.proc = subprocess.Popen(command,
5. shell=True,
6. stdout=subprocess.PIPE)
7. self.proc.wait()
8. file\_size = 0
9. if (self.proc.poll() == 2):  *## 文件不存在时返回值为2*
10. content = b"HTTP/1.1 403 Forbidden\r\nContent-Type: text/html;charset=utf-8\r\n"
11. page = b''
12. self.file\_handle = open("403.html", "rb")
13. for line in self.file\_handle:
14. page += line
15. content += b'\r\n'
16. content += page
17. self.status\_code = 403
18. else:
19. content = b"HTTP/1.1 200 OK\r\nContent-Type: text/html;charset=utf-8\r\n"
20. content += self.proc.stdout.read()
21. file\_size = os.path.getsize(file\_name)
22. self.status\_code = 200
23. self.socket.sendall(content)

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。

* 1. **Static and Dynamic Page**

为网页界面编写了前端静态文件：





CSS设计：

CSS是一种可以为结构化文档添加各种样式的语言，在本次实践当中我们使用CSS对主要网页，计算器界面，数据库查询界面等进行渲染和视觉风格的统一。

代码（部分）：

1. html,body,div,span,applet,object,iframe,h1,h2,h3,h4,h5,h6,p,blockquote,pre,a,abbr,acronym,address,big,cite,code,del,dfn,em,img,ins,kbd,q,s,samp,small,strike,strong,sub,sup,tt,var,b,u,i,dl,dt,dd,ol,nav ul,nav li,fieldset,form,label,legend,table,caption,tbody,tfoot,thead,tr,th,td,article,aside,canvas,details,embed,figure,figcaption,footer,header,hgroup,menu,nav,output,ruby,section,summary,time,mark,audio,video{margin:0;padding:0;border:0;font-size:100%;font:inherit;vertical-align:baseline;}
2. article, aside, details, figcaption, figure,footer, header, hgroup, menu, nav, section {display: block;}
3. ol,ul{list-style:none;margin:0;padding:0;}
4. blockquote,q{quotes:none;}
5. blockquote:before,blockquote:after,q:before,q:after{content:'';content:none;}
6. table{border-collapse:collapse;border-spacing:0;}
7. */\* start editing from here \*/*
8. a{text-decoration:none;}
9. .txt-rt{text-align:right;}*/\* text align right \*/*
10. .txt-lt{text-align:left;}*/\* text align left \*/*
11. .txt-center{text-align:center;}*/\* text align center \*/*
12. .float-rt{float:right;}*/\* float right \*/*
13. .float-lt{float:left;}*/\* float left \*/*
14. .clear{clear:both;}*/\* clear float \*/*

HTML部分的代码过于冗长，不再展示。

* 1. **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：

1. ini = sys.argv[1]
2. hostname = sys.argv[2]
3. port = sys.argv[3]
4. ini = ini.split("&")
5. a = ini[0].split("=")[1]
6. b = ini[1].split("=")[1]
7. c = ""
8. c = ini[2].split("=")[1]

根据运算符判断运算类型并进行运算：

1. if c == "mul":
2. res = res.replace("$c","\*")
3. res = res.replace("$res", str(float(a) \* float(b)))
4. elif c == "div":
5. res = res.replace("$c", "/")
6. res = res.replace("$res", str(float(a) / float(b)))
7. elif c == "add":
8. res = res.replace("$c", "+")
9. res = res.replace("$res", str(float(a) + float(b)))
10. else:
11. res = res.replace("$c", "-")
12. res = res.replace("$res", str(float(a) - float(b)))

输出运算结果：

1. res = res.replace("$hostname", hostname)
2. res = res.replace("$port", port)
3. print(res)

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：

1. ini = sys.argv[1]
2. hostname = sys.argv[2]
3. port = sys.argv[3]
4. ini = ini.split("&")
5. student\_id = ini[0].split("=")[1]
6. student\_name = ini[1].split("=")[1]
7. student\_class = ini[2].split("=")[1]

判断采用了哪些查询条件：

1. id\_flag = 1
2. name\_flag = 1
3. class\_flag = 1
4. if student\_id == "":
5. id\_flag = 0
6. if student\_name == "":
7. name\_flag = 0
8. if student\_class == "":
9. class\_flag = 0

根据查询条件编写sql语句：

1. if id\_flag == 0 and id\_name == 0 and class\_flag == 0:
2. pass
3. else:
4. sql += " where"
5. if id\_flag == 1:
6. sql += " id = " + student\_id
7. if id\_flag == 0 and name\_flag == 1:
8. sql += " name = \'" + student\_name + "\' "
9. elif id\_flag == 1 and name\_flag == 1:
10. sql += " and name = \'" + student\_name + "\' "
11. if id\_flag == 0 and name\_flag == 0 and class\_flag == 1:
12. sql += " class = \'" + student\_class + "\' "
13. elif class\_flag == 1:
14. sql += " and class = \"" + student\_class + "\" "

执行sql语句：

1. cursor.execute(sql)

调整查询结果的格式，输出查询结果与sql语句：

1. with open("cgi-bin/query\_res.html", "r", encoding="utf-8") as f:
2. for line in f:
3. res += line
4. student\_data = ''
5. for student in data:
6. temp = "<tr>"
7. temp += "<th>" + str(student[0]) + "</th>"
8. temp += "<th>" + student[1] + "</th>"
9. temp += "<th>" + student[2] + "</th>"
10. temp += "</tr>\n"
11. student\_data += temp
12. res = res.replace("$data", student\_data)
13. res = res.replace("$message", sql)
14. print(res)
    1. **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:

1. *# 每一次运行都创建一个日志文件*
2. now\_time = time.localtime()
3. log\_name = "log/" + str(now\_time.tm\_year) + "-" + str(
4. now\_time.tm\_mon) + "-" + str(now\_time.tm\_mday) + "-" + str(
5. now\_time.tm\_hour) + "-" + str(now\_time.tm\_min) + "-" + str(
6. now\_time.tm\_sec) + ".txt"
7. thread\_pool(log\_name)

日志记录的数据来源于socket.recv()接收的数据，存储在message变量中，message消息的来源如下：

1. message = self.socket.recv(1024).decode("utf-8")
2. message = message.splitlines()
3. self.msg = message

获取到的message消息如下所示（以GET消息为例）：

GET /css/img/birds.jpeg HTTP/1.1

Host: 192.168.92.1:8888

Connection: keep-alive

User-Agent: Mozilla/5.0 (Windows NT 10.0; Win64; x64) AppleWebKit/537.36 (KHTML, like Gecko) Chrome/106.0.0.0 Safari/537.36

Accept: image/avif,image/webp,image/apng,image/svg+xml,image/\*,\*/\*;q=0.8

Referer: http://192.168.92.1:8888/

Accept-Language: zh-CN,zh;q=0.9

Accept-Encoding: gzip, deflate

对消息处理的过程如下：

1. content = self.msg[1].split(":")[1].replace(" ", "")    *#得到主机地址*
2. content = content + "--"
3. *#记录消息的时间*
4. content = content + "[" + str(time.localtime().tm\_year) + "-" + str(
5. time.localtime().tm\_mon) + "-" + str(
6. time.localtime().tm\_mday) + "-" + str(
7. time.localtime().tm\_hour) + "-" + str(
8. time.localtime().tm\_min) + "-" + str(
9. time.localtime().tm\_sec) + "]"
10. content = content + " " + self.msg[0].split("/")[0].replace(" ", "") + " "  *#请求的方法*
11. content = content + " " + self.msg[0].split(" ")[1].replace(" ", "") + " "  *#请求的URL*
12. content = content + str(file\_size) + " "    *#返回的数据块大小*
13. content = content + str(self.status\_code) + " "     *#返回的状态码*
14. for i in self.msg:  *#发送指向的地址*
15. if (i.split(" ")[0] == "Referer:"):
16. content = content + i.split(" ")[1].replace(" ", "")
17. content = content + "\n"
18. with open(self.log\_name, "a") as f:
19. f.write(content)    *#写回日志文件*

以上述GET类型的消息为例，对消息进行处理后，筛选出其中的主机地址，访问时间，请求方法（GET、POST或是HEAD），请求的资源和返回的状态码后，得到如下日志信息：

192.168.56.1--[2022-9-29-16-41-15] GET  /css/img/birds.jpeg 147048 200 http://192.168.56.1:8888/

最后将得到下消息与使用wireshark抓包结果对比，结果一致，日志记录正确。

图形用户界面, 文本, 应用程序

描述已自动生成

1. **System Deployment, Startup, and Use（系统部署、启动、使用教程，这一块我来写）**
   1. **系统所需环境：**

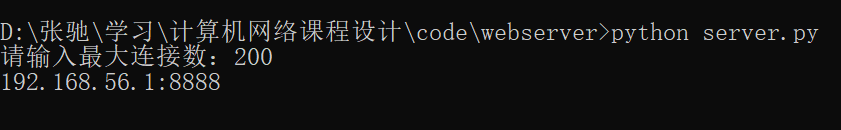
所需环境：python>=3.6，所需python下的库：socket, threading, time, os, sys, queue, subprocess。

使用的编译器：VSCode 或者 Pycharm；

调用方法：编译器中使用 或者 命令行调用。

* 1. **系统的启动：**

**命令行调用**：在webserver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，如下所示：



此时，多线程服务器已启动。

**编译器调用：**使用编译器运行server.py文件即可。同样按照要求输入最大连接数，多线程服务器即可启动。

* 1. **系统的使用方法**

将运行后提示的IP地址和端口’192.168.56.1:8888’复制到浏览器中，打开，可以看到主界面，如下图所示：



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：

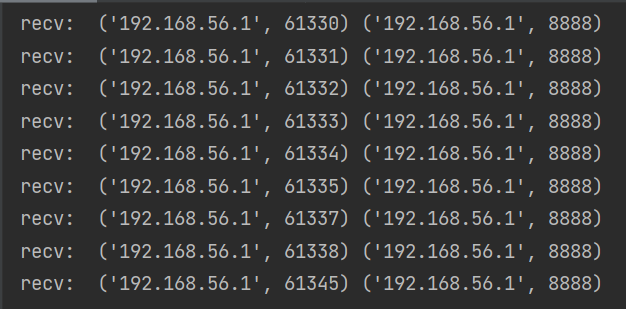


在进行查询时，在方框中输入查询的条件，如果输入为空，则无此查询条件。

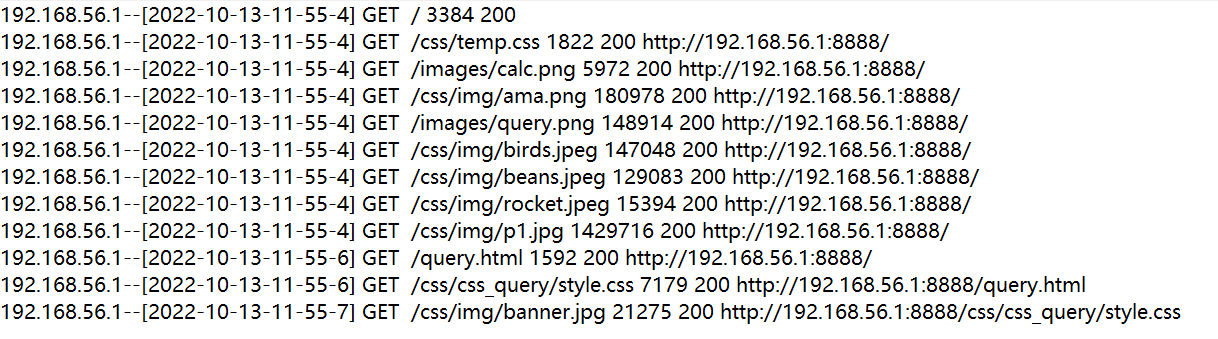
同样，输入不符合要求或者sql语句调用错误时，会返回到400 Bad Request界面。

1. **System Test**
   1. **Unit Test**

Server.py: 负责服务器的socket连接，以及调用线程池，能接收到如下的信息，说明信息发送与接收成功；并且，当服务器超过时间未响应时，会弹出main server timeout



Worker.py：负责HTTP请求处理与相应，记录操作并生成日志文件，日志文件如下所示：



同时，针对以上两个模块的测试，也写了相应的测试代码，位于test\get.py，测试代码如下所示：

1. class RQ\_get(threading.Thread):
2. """docstring for RQ\_get."""
3. def \_\_init\_\_(self):
4. super(RQ\_get, self).\_\_init\_\_()
5. def run(self):
6. try:
7. rq = requests.get("http://192.168.56.1:8888/index.html")
8. print(rq.headers)
9. print(rq.content.decode("utf8"))
10. except:
11. print("提前关闭了链接，因为要释放")

RQ\_head 以及 RQ\_post 类与之类似，test.py运行结果如下：



可以成功输出相应的HTML文本。

CGI测试放在集成测试中，不在进行单元测试。

* 1. **Integrated Test**

网页如下所示：



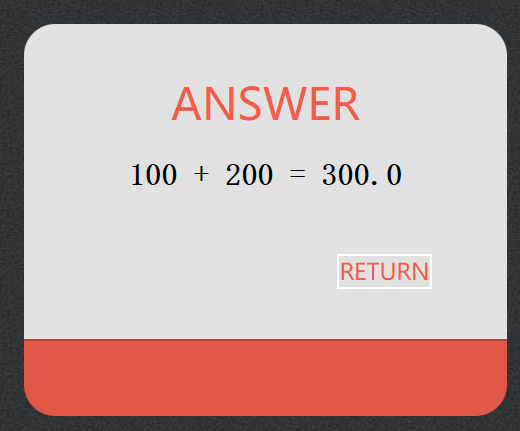
当输入一个不存在的网址，会弹出404 Not Found，例如输入http://192.168.56.1:8888/cgi-bin



当输入不符合规则，会弹出400 Bad Request：



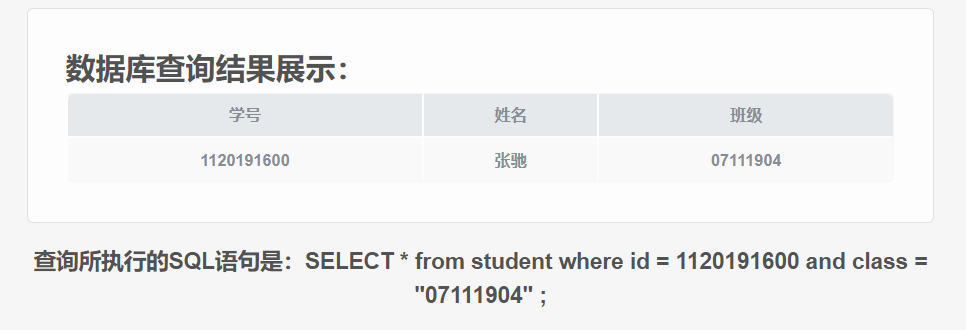
当输入正确，则返回：



对于学生信息查询的界面，在文本框中输入任何均可，查询结果如下所示：



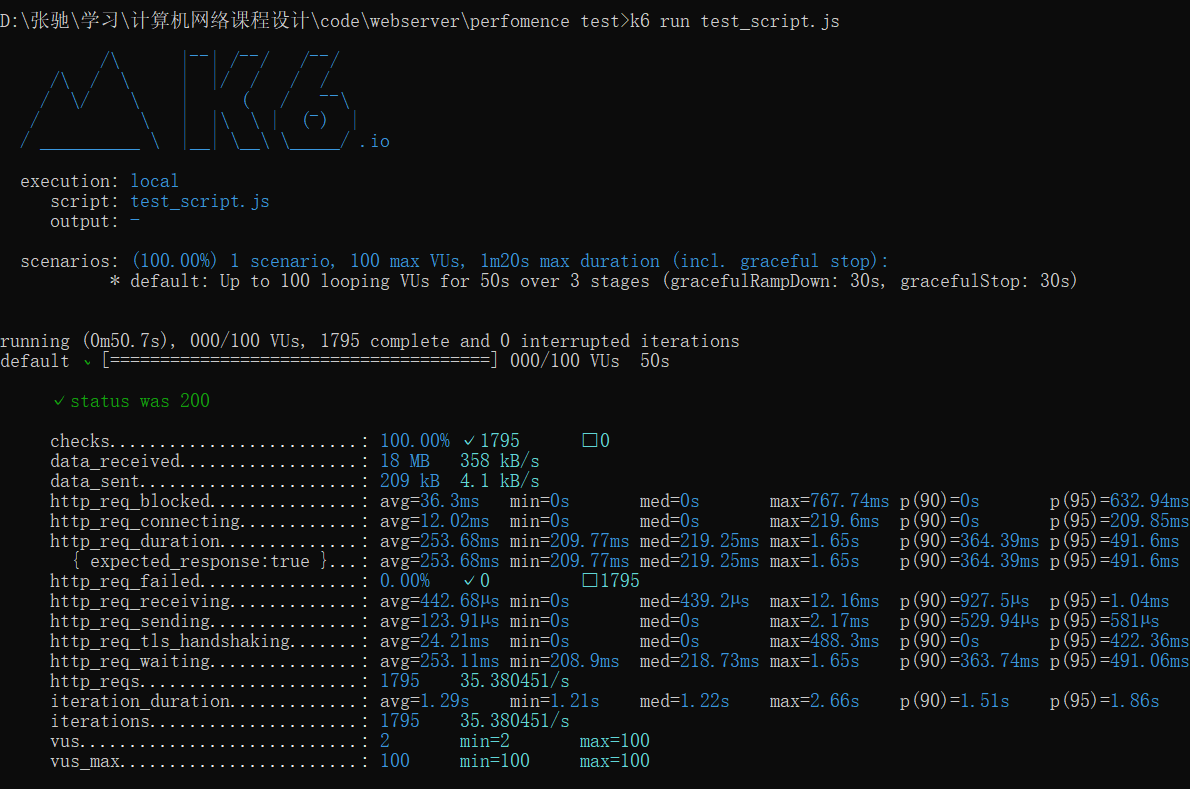
查询后会显示查询结果，以及查询所执行的SQL语句：



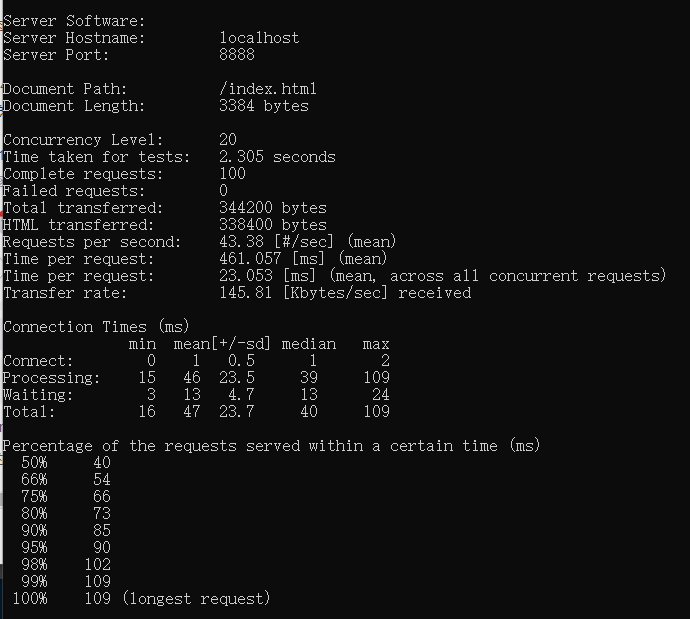
1. **Performance**

针对多线程服务器，我们使用了压力测试与负载测试软件为：K6

在命令行端运行测试脚本test\_script.js, 得到结果如下所示：



同时在Apache Bench上进行压力测试，测试结果如下图所示：



请求数目增长，并发数目不变，各项内容如下表所示：

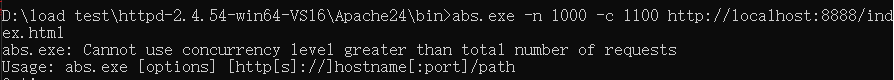
|  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- |
| 请求数目 | 100 | 200 | 300 | 500 | 1000 |
| 并发数目 | 20 | 20 | 20 | 20 | 20 |
| 测试持续时间(sec) | 2.305 | 2.393 | 2.521 | 2.551 | 3.342 |
| 完成请求数量 | 100 | 200 | 300 | 500 | 1000 |
| 失败请求数量 | 0 | 0 | 0 | 0 | 0 |
| 网络传输量(bytes) | 344200 | 688400 | 1032600 | 1721000 | 3442000 |
| 平均每秒传输流量(Kbytes/sec) | 145.81 | 280.91 | 399.96 | 658.87 | 1005.74 |

做请求数目与平均每秒传输流量的折线图，如下所示：

请求数目不变，并发数目增长，各项内容如下表所示：

|  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- |
| 请求数目 | 1000 | 1000 | 1000 | 1000 | 1000 |
| 并发数目 | 50 | 100 | 200 | 500 | 1000 |
| 测试持续时间(sec) | 3.246 | 3.229 | 3.329 | 4.119 | 4.085 |
| 完成请求数量 | 1000 | 1000 | 1000 | 1000 | 1000 |
| 失败请求数量 | 0 | 0 | 0 | 0 | 0 |
| 网络传输量(bytes) | 3442000 | 3442000 | 3442000 | 3442000 | 3442000 |
| 平均每秒传输流量(Kbytes/sec) | 1035.43 | 1041.09 | 1009.76 | 816.01 | 822.78 |

同时，并发数目最大为1000，当并发数大于1000时，会报错，如下所示：



做并发数目与平均每秒传输流量的折线图，如下所示：

1. **Summary or Conclusions**

本项目分析了TCP Socket连接，HTTP的GET、HEAD、POST请求，通过构建线程池的方式，完成了多线程服务器的构建。通过Python语言完成了CGI测试模块，包含计算器和数据查询两部分的内容，成功完成了静态网页和动态网页的构建。通过使用代码搭建多线程服务器的方式，对于TCP连接、HTTP协议、CGI程序调用，有了进一步的了解。同时，对网站惊醒了压力测试，网站的系统性能尚可，能够承载1000个并发客户的访问，能够满足小部分人同时访问的要求。

1. **References**

<https://blog.csdn.net/ouguangqian/article/details/116096273?spm=1001.2101.3001.6650.3&utm_medium=distribute.pc_relevant.none-task-blog-2%7Edefault%7ECTRLIST%7ERate-3-116096273-blog-115374812.pc_relevant_3mothn_strategy_and_data_recovery&depth_1-utm_source=distribute.pc_relevant.none-task-blog-2%7Edefault%7ECTRLIST%7ERate-3-116096273-blog-115374812.pc_relevant_3mothn_strategy_and_data_recovery&utm_relevant_index=5>

<https://blog.csdn.net/yongcto/article/details/12683653>

<https://blog.csdn.net/qq_42967398/article/details/105326616>

<https://blog.csdn.net/zsrwan/article/details/107082010>

https://www.cnblogs.com/luyuze95/p/11289143.html

1. **Comments**

张驰：本次计网课程设计中，我一方面负责统筹的工作，在经过前期了解后，决定了整个多线程服务器的架构，决定使用python作为编写该方面的语法。同时，在本次代码方面，后端方面我主要负责了数据库查询方面的内容，完成了通过在网页端输入查询条件后，能够对数据库进行查询，并且显示sql语句。同时，针对前端的网页，我又编写了一些css文件，渲染了整个界面，使得整个界面更加美观。通过本次计网课设，我了解了网络通过连接来传递信息，同时通过CGI的编写，了解了前端和后端的数据传输，收获颇丰。

傅裕荣：我完成的部分是server与client之间TCP-socket连接模块的实现。对于此部分知识，之前只在计算机网络的课程上从理论的角度进行了学习。此次实践，经过多番资料查阅，最后选用功能简便，结构清晰，有丰富库函数的python对此进行了实现。此次课设加深了我对计算机网络这门课的了解与兴趣。

张洪璨：运算模块可以丰富一部分内容，比如支持其他的运算或者支持采用按钮的方式进行输入；数据库查询模块可以实现其他的一些功能，比如把向数据库中插入内容，在查询前要求用户验证身份。验证通过才能进行查询或者查询到全部的数据。

李鹏：在本次课设中，我完成的是http的请求处理和响应部分，对我而言，收获最大的部分是学会了如何从二进制数据中解析报文数据，再者就是如何根据http协议进行响应的封装和返回。加深了我对http协议的理解，提高了我的编码能力和对python编程的能力。收获颇深。

魏慧聪：我完成了服务端日志文件的记录工作，通过对日志文件的处理，加深了我对TCP通信模块信息交换的了解。日志文件本身的记录工作不难，通过抓包等方式去验证和了解TCP报文的格式和传递的消息内容非常有趣。

于松涛：此次实验我编写了线程池管理模块，使用python作为编程语言。整体来讲实现过程比较容易，但是改进空间仍然很大。当前线程池管理我们使用了一个简单的队列，当超出最大线程数量时，将最早的线程进行销毁，但是这样并不符合实际应用场景。更符合逻辑的实现方法是检测线程的活跃程度，将长时间不操作的线程进行销毁，如果所有的线程均活跃，则拒绝新提出的用户请求或使其等待。这个过程中我对服务器管理过程有了更深入的了解，也对python语言运用的更加熟练。

李彧泽： 在本次项目当中进行了相当多的编码工作，积累了代码工作的经验，同时也加深了对计算机网络和网页搭建以及线程操作等知识的理解。在编码当中遇到的问题也都在与队伍之间的交流中得到了解决。对于一些抽象的知识，名词和概念也有了更深的感受。