

南京大学本科生实验报告

| | |
|---|--|
| 课程名称：计算机网络 | 任课教师：田臣 助教：方毓楚/于沛文/郑浩/陈伟/李国浩/李睿宸/杨溢... |
| 学院：计算机科学与技术系 | 专业(方向)：计算机科学与技术 |
| 学号：201220096 | 姓名：钟亚晨 |
| Email: 2991908515@qq.com | 开始/完成日期：2022.05.30-2022.05.30 |

1.实验名称

Content Delivery Network

完成清单：

- ✧ 完成了DNS服务器逻辑并通过对应的仿真测试和单元测试；
- ✧ 完成了 Caching Server 逻辑并通过对应的仿真测试和单元测试；
- ✧ 通过了最终的 a11 覆盖测试，并且提交至 OpenNetLab 完成了对应任务；
- ✧ 完成了流式cache的challenge并且通过了所有的测试用例；
- ✧ 完成了本篇实验报告；

2.实验目的

- ✧ 实现 DNS server 逻辑以及 Caching server 逻辑以构建一个CDN；
- ✧ 通过本地测试文件以及使用 OpenNetLab 获取日志进行分析；

3.实验内容

- ✧ 完成 DNS server 的逻辑，并通过手动测试和单元测试；
- ✧ 完成 Caching server 的逻辑，并通过手动测试和单元测试；
- ✧ 通过 a11 覆盖测试并提交至 OpenNetLab 获取日志对结果进行分析；
- ✧ 完成对应的实验报告；

4.实验结果

简单说明，以下对实验说明中的三个问题进行回答。

0x01 In the report, show how you implement the features of DNS server.

Step01.完成DNS记录表装载

首先是为 DNS server 装载DNS记录表，这里使用Python对文件IO即可完成：

```
26     def parse_dns_file(self, dns_file):
27         # -----
28         # TODO: your codes here. Parse the dns_table.txt file
29         # and load the data into self._dns_table.
30         # -----
31         with open(dns_file) as fp:
32             lines = fp.readlines()
33             for line in lines:
34                 tmp = line.split()
35                 if tmp != []:
36                     self._dns_table.append(tmp)
```

观察 dns_table.txt 文件可以知晓每一条表项的结构为：

```
1 | | regex_address | type | one or more destination addresses |
```

这里第31行根据传入的 dns_file 地址打开本地DNS记录的txt文件进行表项装载，使用 with open 的办法，Python将在文件不使用时自动关闭；

而后的32行到36行，以行为单位读取文件，一行对应于一个表项，而后使用 split 方法，将表项的各个部分进行分割，35行的判断是为了防止末尾回车导致的冗余空表项被载入；

最后，内存中的DNS记录表为一个列表，每个元素又为一个列表，而有如下结构：

```
1 self._dns_table[i][0]    表示DNS记录表中第i项的Domain Name;
2 self._dns_table[i][1]    表示DNS记录表中第i项的Record Type;
3 self._dns_table[i][2:]   表示DNS记录表中第i项的Record Values，可以有一项或多项；
```

Step02.完成对用户DNS Request的回复

依照实验手册说明，DNS服务器进行response响应的逻辑应当为：

使用请求的 domain_name 查找DNS表；

如果未查找到表项，则直接返回 (None, None) 元组；

如果找到了表项，则根据其不同的类型进行处理：

- 如果类型为CNAME，则使用格式(Domain Name, Record Value)返回对应表项；
- 如果类型为A，则分为以下情况进行考虑：
 - 如果Record Values仅包含一个项目，则同上直接返回表项；
 - 如果Record Values包含多个项目，则又分为如下状况进行考虑：
 - 如果无法找到 client_ip 的地址，则随机返回一个表项；
 - 如果可以找到 client_ip 的地址，则从这多个项目中找到一个距离最近的进行返回；

而实现代码依照上述逻辑来进行编写即可，实际上就为伪代码描述。

而此处需要注意两个细节：

- DNS记录表中的表项中 `Domain Name` 当中的星号很类似于正则表达式，可以和任何对应字段匹配：

这里的实现办法为将输入的 `request_domain_name` 和各个表项都通过 `split('.')` 来划分为列表，而后对于 `*` 进行特殊处理，并且首先核对长度是否相等以完成匹配：

```
89         flag = True
90         for i in range(len(tmp1)):
91             if tmp1[i] != '*' and tmp1[i] != tmp2[i]:
92                 flag = False
93                 break
94         if flag:
95             target_item = item
96     # If no matched item, return (None, None)
```

- 第二个细节在于DNS记录表中的一些表项末尾会包含多余的 `.`，从而导致 `all` 的覆盖性测试无法通过，因此需要对该细节进行特别处理，一个处理办法就是对匹配时划分成的两个列表末尾进行检测，如果表项为空，则直接弹出：

```
81         for item in self.table:
82             tmp1 = item[0].split('.')
83             if tmp1[-1] == '':
84                 tmp1.pop()
85             tmp2 = request_domain_name.split('.')
86             if tmp2[-1] == '':
87                 tmp2.pop()
```

```
(syenv) njucs@njucs-VirtualBox:~/My_files/tmp$ python
Python 3.6.9 (default, Dec  8 2021, 21:08:43)
[GCC 8.4.0] on linux
Type "help", "copyright", "credits" or "license" for more information.
>>> tmp = '127.1.1.0.'
>>> print(tmp.split('.'))
['127', '1', '1', '0', '']
>>>
```

最后执行测试脚本，通过了所有的单元测试，可以初步验证实实现逻辑的正确性：

```
njucs@njucs-VirtualBox: ~/My_files/tmp
File Edit View Search Terminal Help
(syenv) njucs@njucs-VirtualBox:~/My_files/tmp$ python3 test_entry.py dns
2022/05/30-11:33:11| [INFO] DNS server started
test_cname1 (testcases.test_dns.TestDNS) ... ok
test_cname2 (testcases.test_dns.TestDNS) ... ok
test_location1 (testcases.test_dns.TestDNS) ... ok
test_location2 (testcases.test_dns.TestDNS) ... ok
test_non_exist (testcases.test_dns.TestDNS) ... ok

-----
Ran 5 tests in 0.020s

OK
2022/05/30-11:33:12| [INFO] DNS server terminated
(syenv) njucs@njucs-VirtualBox:~/My_files/tmp$
```

0x02 ☒ In the report, show how you implement the features of caching server.

对 `Caching server` 逻辑的实现需要阅读一些代码，以了解整个 `Caching server` 当中各个类是如何协同工作完成其功能的。实验手册当中的实现顺序为 `HTTPRequestHandler` 到 `Caching Server`，而我的实现顺序为 `Caching Server` 到 `HTTPRequestHandler`，并且以自己实现的逻辑来讲述实现思路。

实现 Caching Server 的 touchItem 函数

```
97     def touchItem(self, path: str):
98         ''' Touch the item of path.
99         This method, called by HttpHandler, serves as a bridge of server and
100         handler.
101         If the target doesn't exist or expires, fetch from main server.
102         Write the headers to local cache and return the body.
103         '''
104         # TODO: implement the logic described in doc-string
105         if path not in self.cacheTable.data.keys() or self.cacheTable.expired(path) == True:
106             tmp = self.requestMainServer(path)
107             if tmp != None:
108                 headers_buf = tmp.getheaders()
109                 body_buf = tmp.read()
110                 self.cacheTable.setHeaders(path, headers_buf)
111                 self.cacheTable.appendBody(path, body_buf)
112                 return self.cacheTable.data[path]
113         if path in self.cacheTable.data.keys() and self.cacheTable.expired(path) == False:
114             return self.cacheTable.data[path]
115         else:
116             return None
```

首先需要阅读源代码了解到一些事实：

- ①阅读 `./cachingServer.py` 当中 `CachingServer` 类的 `requestMainServer` 方法，该方法用于获得远程服务器传来的回应，如果无法连接到或者是远程服务器无法获取到对应表项，则返回 `None`；
- ②阅读 `./cacheTable.py` 当中的各个类，可以知道 `HTTPCacheItem` 为对 HTTP header、body、timestamp 的封装体，而 `CacheTable` 主体为一个列表，每个表项就为 `HTTPCacheItem`，并且提供了一些增加 header、body 的方法以及对于过期表现的检验方法，如果表项过期，则调用 `expire` 方法会返回 `True` 并且自动地从表中删除过期表项。

而后对 `touchItem` 的实现逻辑就可以依照手册和提供的已有方法来进行实现：

- ①105到112行：如果请求的路径无法在自己的 `cacheTable` 当中找到，或者找到了但是过期了，那么则自动删除过期表项，并且尝试向远程服务器发送来查询。如果远程查找到，则将该表项添加到自己的 `cacheTable` 当中，并且将该 `HTTPCacheItem` 进行返回，没找到则什么也不做；
- ②113到116行：如果直接能够在本地查找到表项，则直接返回该对象；如果找不到则返回 `None`；

实现 CachingServerHttpHandler 类逻辑

首先通过阅读手册和源代码可以知道如下事实：

- ①可以通过 `self.server.touchItem(self.path)` 来对请求的路径进行查询，如果查询到了则会返回一个 `HTTPCacheItem` 对象，如果未查找到则会返回 `None`；
- ②可以通过手册中介绍的相关方法来发送 header、body 以及 response 状态码，而依照官方文档，可以通过 `HTTPStatue` 获取各种状态码；
- ③使用了 `send_header` 后要使用 `end_headers`；
- ④发送时依照的顺序为：状态码、header、body；

而后便可以依照实验手册和已知信息来编写代码：

1、首先来编写 `do_GET` 的逻辑：

```
174     @trace
175     def do_GET(self):
176         ''' Logic when receive a HTTP GET.
177         Notice that the URL is automatically parsed and the path is stored in
178         self.path.
179         '''
180         # TODO: implement the logic to response a GET.
181         # Remember to leverage the methods in CachingServer.
182         response = self.server.touchItem(self.path)
183         if response == None:
184             self.send_response(HTTPStatus.NOT_FOUND)
185             self.send_error(HTTPStatus.NOT_FOUND, "File not found")
186         else:
187             self.send_response(HTTPStatus.OK)
188             self.headers = response.headers
189             self.sendHeaders()
190             self.sendBody([response.body])
```

从 `server.touchItem(self.path)` 来查找是否存在对应的表项：

- 如果不存在表项，则返回状态码404，并且使用 `send_error` 方法返回一个错误消息；
- 如果存在表项，则返回状态码200，并且调用 `sendHeaders` 发送headers，再使用 `sendBody` 发送body；

这里以及以下都使用 `HTTPStatus` 来标识状态码；

2、由于上述引用了 `sendHeaders` 方法，因此这里来实现 `sendHeaders` 的逻辑：

在调用 `sendHeaders` 之前，我们是通过 `self.headers` 来指定了headers属性的，并且已经发送了状态码，因此这里直接通过循环遍历调用 `send_header` 函数发送即可：

```
158     @trace
159     def sendHeaders(self):
160         ''' Send HTTP headers to client'''
161         # TODO: implement the logic of sending headers
162         for header in self.headers:
163             self.send_header(header[0], header[1])
164         self.end_headers()
165
```

3、`do_HEAD` 的实现方法与 `do_GET` 一致，不同的在于其不发送body，因此将 `do_HEAD` 的代码复制粘贴并且将最后发送body的一句删除即可；

最后执行对应的单元测试脚本以及覆盖测试，通过了所有的测试用例，可以初步判断实现逻辑的正确性：

```
(syenv) njucs@njucs-VirtualBox:~/My_files/tmp$ python3 test_entry.py cache
2022/05/30-13:18:46| [INFO] Main server started
2022/05/30-13:18:46| [INFO] RPC server started
2022/05/30-13:18:46| [INFO] Caching server started
test_01_cache_missed_1 (testcases.test_cache.TestCache) ...
[Request time] 9.43 ms
ok
test_02_cache_hit_1 (testcases.test_cache.TestCache) ...
[Request time] 2.65 ms
ok
test_03_cache_missed_2 (testcases.test_cache.TestCache) ...
[Request time] 7.93 ms
ok
test_04_cache_hit_2 (testcases.test_cache.TestCache) ...
[Request time] 2.36 ms
ok
test_05_HEAD (testcases.test_cache.TestCache) ...
[Request time] 2.59 ms
ok
test_06_not_found (testcases.test_cache.TestCache) ...
[Request time] 6.32 ms
ok

-----
Ran 6 tests in 3.704s

OK
2022/05/30-13:18:51| [INFO] Caching server terminated
2022/05/30-13:18:51| [INFO] PRC server terminated
2022/05/30-13:18:51| [INFO] Main server terminated
(syenv) njucs@njucs-VirtualBox:~/My_files/tmp$
```

```
2022/05/30-13:18:51| [INFO] Main server terminated
(syenv) njucs@njucs-VirtualBox:~/My_files/tmp$ python3 test_entry.py all
2022/05/30-13:19:14| [INFO] DNS server started
2022/05/30-13:19:14| [INFO] Main server started
2022/05/30-13:19:14| [INFO] RPC server started
2022/05/30-13:19:14| [INFO] Caching server started
test_01_cache_missed_1 (testcases.test_all.TestAll) ...
[Request time] 10.50 ms
ok
test_02_cache_hit_1 (testcases.test_all.TestAll) ...
[Request time] 2.90 ms
ok
test_03_not_found (testcases.test_all.TestAll) ...
[Request time] 7.05 ms
ok

-----
Ran 3 tests in 1.671s

OK
2022/05/30-13:19:17| [INFO] DNS server terminated
2022/05/30-13:19:17| [INFO] Caching server terminated
2022/05/30-13:19:17| [INFO] PRC server terminated
2022/05/30-13:19:17| [INFO] Main server terminated
(syenv) njucs@njucs-VirtualBox:~/My_files/tmp$
```

Challenge: 为解决大文件的传输延时问题，实现流式的Cache缓存传输

依照实验手册，实现的原理实际上并不复杂，使用 `readinto(b)` 方法以及 `Iterator` 即Python当中的迭代器即可实现，而迭代器的用途实际上在给出的参考网站中以及有了很详尽的说明以及示例。

实现逻辑如下：


```

100     def helper(self, tmp, path):
101         body_buf = None
102         while True:
103             length = tmp.readinto(self.buffer)
104             if body_buf == None:
105                 body_buf = self.buffer[0:length]
106             else:
107                 body_buf += self.buffer[0:length]
108             yield self.buffer[0:length]
109             if length == 0:
110                 self.cacheTable.appendBody(path, body_buf)
111                 break
112
113     def touchItem(self, path: str):
114         ''' Touch the item of path.
115         This method, called by HttpHandler, serves as a bridge of server and
116         handler.
117         If the target doesn't exist or expires, fetch from main server.
118         Write the headers to local cache and return the body.
119         '''
120         # TODO: implement the logic described in doc-string
121         if path not in self.cacheTable.data.keys() or self.cacheTable.expired(path) == True:
122             tmp = self.requestMainServer(path)
123             if tmp != None:
124                 headers_buf = tmp.getheaders()
125                 # body_buf = tmp.read()
126                 self.cacheTable.setHeaders(path, headers_buf)
127                 # self.cacheTable.appendBody(path, body_buf)
128                 # return self.cacheTable.data[path]
129                 body_buf = self.helper(tmp, path)
130                 return [headers_buf, body_buf]
131             if path in self.cacheTable.data.keys() and self.cacheTable.expired(path) == False:
132                 return self.cacheTable.data[path]
133             else:
134                 return None

```

首先我们定义一个 `helper` 的 Generator Function，该函数返回一个 Iterator 迭代器，并且在迭代过程中会为本地 Cache 维护一个缓冲，当迭代结束时更新本地 Cache，同时在 `touchItem` 函数当中需要向远程服务器进行访问时调用该函数生成迭代器并和头部以列表形式进行返回；而后在处理阶段：

```

191     @trace
192     def do_GET(self):
193         ''' Logic when receive a HTTP GET.
194         Notice that the URL is automatically parsed and the path is stored in
195         self.path.
196         '''
197         # TODO: implement the logic to response a GET.
198         # Remember to leverage the methods in CachingServer.
199         response = self.server.touchItem(self.path)
200         if response == None:
201             self.send_response(HTTPStatus.NOT_FOUND)
202             self.send_error(HTTPStatus.NOT_FOUND, "File not found")
203         else:
204             self.send_response(HTTPStatus.OK)
205             if isinstance(response, list):
206                 self.headers = response[0]
207                 self.sendHeaders()
208                 while True:
209                     try:
210                         val = next(response[1])
211                         print(val)
212                         self.sendBody(val)
213                     except StopIteration:
214                         break
215             # self.sendBody(response[1])
216             elif isinstance(response, HTTPCacheItem):
217                 self.headers = response.headers
218                 self.sendHeaders()
219                 self.sendBody(response.body)

```

我们首先根据返回值的不同情况来进行不同分支的处理，如果是返回了列表，则说明所查找到的项目是访问了远程服务器的，这个时候就需要用到Python的错误处理（或者给next函数指定第二个参数防止迭代器迭代到尾部导致的报错），并且对迭代器进行迭代，在该过程当中，迭代器是一边将数据一个buffer一个buffer地写入缓存，并且进行响应，当数据迭代完成后整个响应也就完成了，同时本地Cache也会被更新。


至此我们完成了流式传输的实现，解决了大文件传输过程中不必要的中间时延。

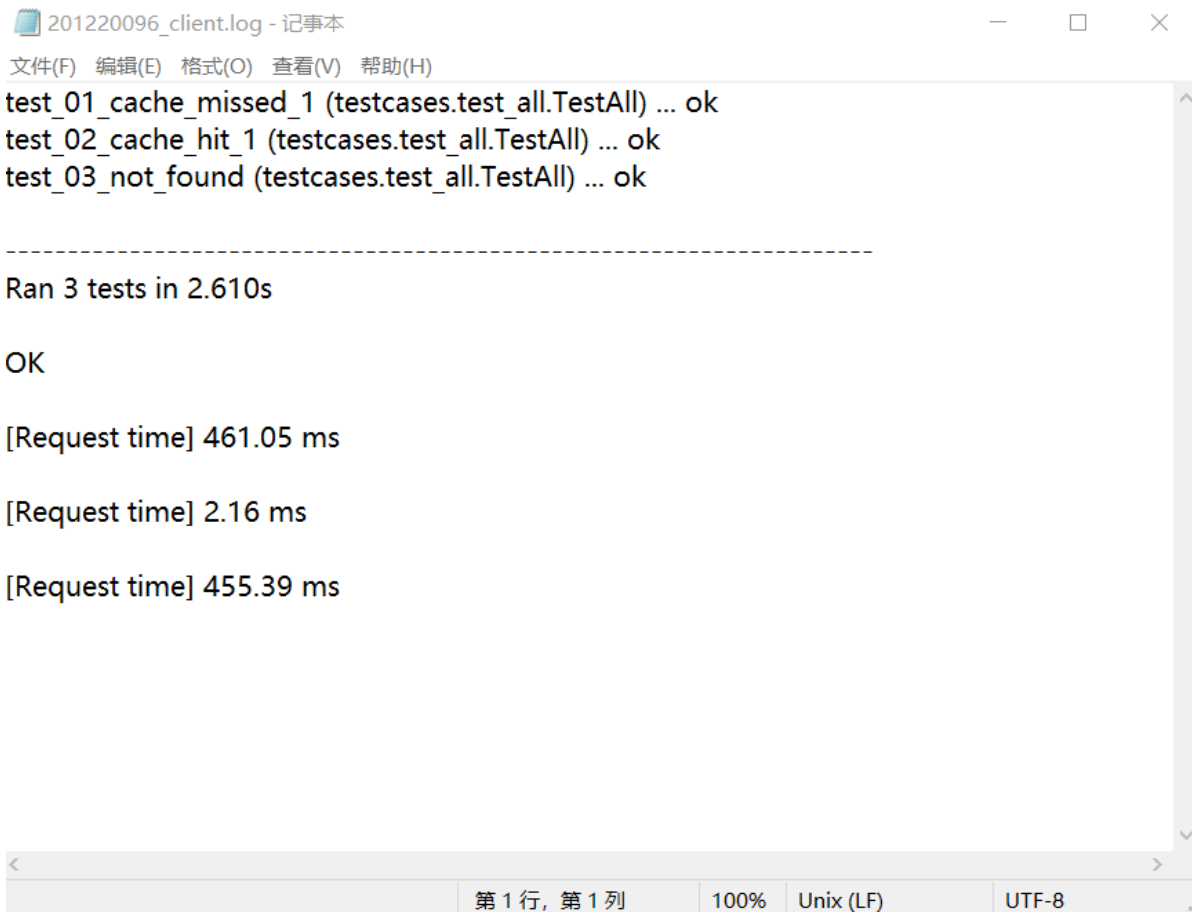
而后执行各个测试，可以通过所有的测试用例，因此可以基本验证逻辑实现的正确性：

```
(syenv) njucs@njucs-VirtualBox:~/My_files/tmp$ python3 ./test_entry.py cache
2022/05/30-22:28:05| [INFO] Main server started
2022/05/30-22:28:05| [INFO] RPC server started
2022/05/30-22:28:05| [INFO] Caching server started
test_01_cache_missed_1 (testcases.test_cache.TestCache) ...
[Request time] 6.60 ms
ok
test_02_cache_hit_1 (testcases.test_cache.TestCache) ...
[Request time] 2.69 ms
ok
test_03_cache_missed_2 (testcases.test_cache.TestCache) ...
[Request time] 8.89 ms
ok
test_04_cache_hit_2 (testcases.test_cache.TestCache) ...
[Request time] 2.71 ms
ok
test_05_HEAD (testcases.test_cache.TestCache) ...
[Request time] 3.19 ms
ok
test_06_not_found (testcases.test_cache.TestCache) ...
[Request time] 6.67 ms
ok
-----
Ran 6 tests in 3.579s

OK
2022/05/30-22:28:09| [INFO] Caching server terminated
2022/05/30-22:28:09| [INFO] PRC server terminated
2022/05/30-22:28:09| [INFO] Main server terminated
2022/05/30-22:28:09| [INFO] Main server terminated
(syenv) njucs@njucs-VirtualBox:~/My_files/tmp$ python3 ./test_entry.py all
2022/05/30-22:28:13| [INFO] DNS server started
2022/05/30-22:28:13| [INFO] Main server started
2022/05/30-22:28:13| [INFO] RPC server started
2022/05/30-22:28:13| [INFO] Caching server started
test_01_cache_missed_1 (testcases.test_all.TestAll) ...
[Request time] 6.47 ms
ok
test_02_cache_hit_1 (testcases.test_all.TestAll) ...
[Request time] 3.28 ms
ok
test_03_not_found (testcases.test_all.TestAll) ...
[Request time] 4.89 ms
ok
-----
Ran 3 tests in 1.722s

OK
2022/05/30-22:28:15| [INFO] DNS server terminated
2022/05/30-22:28:15| [INFO] Caching server terminated
2022/05/30-22:28:15| [INFO] PRC server terminated
2022/05/30-22:28:15| [INFO] Main server terminated
(syenv) njucs@njucs-VirtualBox:~/My_files/tmp$
```

0x03  In the report, show how much the CDN cache shortens the request time. Write the procedure and analysis in your report with screenshots.



```
201220096_client.log - 记事本
文件(F) 编辑(E) 格式(O) 查看(V) 帮助(H)
test_01_cache_missed_1 (testcases.test_all.TestAll) ... ok
test_02_cache_hit_1 (testcases.test_all.TestAll) ... ok
test_03_not_found (testcases.test_all.TestAll) ... ok

-----

Ran 3 tests in 2.610s

OK

[Request time] 461.05 ms

[Request time] 2.16 ms

[Request time] 455.39 ms

第 1 行, 第 1 列 100% Unix (LF) UTF-8
```

通过查看测试用例以及记录的[request time], 可以发现, 当cache缺失时, 需要从远程服务器来获取, 这时完成对请求回复的耗时为461.05ms, 而当cache命中时, 完成对请求的回应则仅需2.16ms。这说明使用cache能够大幅度缩短响应请求时间, 此处就缩短了458.89ms的响应请求时间。而cache运作的原理就在于将最近频繁请求所获得的响应存储在本地cache中, 当下次遇到了相同的请求并且本地cache中对应表项尚未过期, 则直接从本地进行发送响应, 这样就节省了大量从远程其它服务器获取响应所消耗的时间。

5.核心代码

```
1  ### dns_server.py ###
2  # class DNSServer
3  def parse_dns_file(self, dns_file):
4      # -----
5      # TODO: your codes here. Parse the dns_table.txt file
6      # and load the data into self._dns_table.
7      # -----
8      with open(dns_file) as fp:
9          lines = fp.readlines()
10         for line in lines:
11             tmp = line.split()
12             if tmp != []:
13                 self._dns_table.append(tmp)
14 # class DNSHandler
15 def get_response(self, request_domain_name):
```

```

16         response_type, response_val = (None, None)
17         # -----
18         # TODO: your codes here.
19         # Determine an IP to response according to the client's IP
address.
20         #         set "response_ip" to "the best IP address".
21         client_ip, _ = self.client_address
22         # Find matched item in DNS Table
23         target_item = None
24         # Compare each item and deal with point at end
25         for item in self.table:
26             tmp1 = item[0].split('.')
27             if tmp1[-1] == '':
28                 tmp1.pop()
29             tmp2 = request_domain_name.split('.')
30             if tmp2[-1] == '':
31                 tmp2.pop()
32             if len(tmp1) == len(tmp2):
33                 flag = True
34                 for i in range(len(tmp1)):
35                     if tmp1[i] != '*' and tmp1[i] != tmp2[i]:
36                         flag = False
37                         break
38                 if flag:
39                     target_item = item
40         # If no matched item, return (None, None)
41         if target_item == None:
42             return (response_type, response_val)
43         # If item type is CNAME, then return it
44         if target_item[1] == "CNAME":
45             response_type = target_item[1]
46             response_val = target_item[2]
47         # If item type is A, then...
48         elif target_item[1] == "A":
49             response_type = target_item[1]
50             # if only one items, return it
51             if len(target_item) == 3:
52                 response_val = target_item[2]
53             # if have more than one item, return the nearest
54             else:
55                 client_local = IP_Utils.getIpLocation(client_ip)
56                 # if can't find client, then random return
57                 if client_local == None:
58                     response_val = target_item[random.randint(2,
len(target_item))]
59                 # if can find client, then return the nearest
60                 else:
61                     min_dist = self.calc_distance(client_local,
IP_Utils.getIpLocation(target_item[2]))
62                     response_val = target_item[2]
63                     for item in target_item[2:]:

```

```

64         dist_tmp = self.calc_distance(client_local,
IP_Utils.getIpLocation(item))
65         if dist_tmp < min_dist:
66             min_dist = dist_tmp
67             response_val = item
68         # -----
69         return (response_type, response_val)
70
71 ### cachingServer.py ###
72 # class CachingServer
73 def touchItem(self, path: str):
74     ''' Touch the item of path.
75     This method, called by HttpHandler, serves as a bridge of
server and
76     handler.
77     If the target doesn't exist or expires, fetch from main
server.
78     Write the headers to local cache and return the body.
79     '''
80     # TODO: implement the logic described in doc-string
81     if path not in self.cacheTable.data.keys() or
self.cacheTable.expired(path) == True:
82         tmp = self.requestMainServer(path)
83         if tmp != None:
84             headers_buf = tmp.getheaders()
85             body_buf = tmp.read()
86             self.cacheTable.setHeaders(path, headers_buf)
87             self.cacheTable.appendBody(path, body_buf)
88             return self.cacheTable.data[path]
89     if path in self.cacheTable.data.keys() and
self.cacheTable.expired(path) == False:
90         return self.cacheTable.data[path]
91     else:
92         return None
93 # class CachingServerHttpHandler
94 def sendHeaders(self):
95     ''' Send HTTP headers to client'''
96     # TODO: implement the logic of sending headers
97     for header in self.headers:
98         self.send_header(header[0], header[1])
99     self.end_headers()
100 @trace
101 def do_GET(self):
102     ''' Logic when receive a HTTP GET.
103     Notice that the URL is automatically parsed and the path is
stored in
104     self.path.
105     '''
106     # TODO: implement the logic to response a GET.
107     # Remember to leverage the methods in CachingServer.
108     response = self.server.touchItem(self.path)
109     if response == None:

```

```

110         self.send_response(HTTPStatus.NOT_FOUND)
111         self.send_error(HTTPStatus.NOT_FOUND, "'File not found'")
112     else:
113         self.send_response(HTTPStatus.OK)
114         self.headers = response.headers
115         self.sendHeaders()
116         self.sendBody(response.body)
117
118     @trace
119     def do_HEAD(self):
120         ''' Logic when receive a HTTP HEAD.
121         The difference from self.do_GET() is that do_HEAD() only send
122         HTTP
123         headers.
124         '''
125         # TODO: implement the logic to response a HEAD.
126         # Similar to do_GET()
127         response = self.server.touchItem(self.path)
128         if response == None:
129             self.send_response(HTTPStatus.NOT_FOUND)
130             self.send_error(HTTPStatus.NOT_FOUND, "'File not found'")
131         else:
132             self.send_response(HTTPStatus.OK)
133             self.headers = response.headers
134             self.sendHeaders()

```

6.总结与感想

总结:

- 🧐操作系统的cache多级缓存模型拥有广泛的应用，用于减少不必要的远程资源请求，能够大幅度减少响应时间，尤其是对某一资源的频繁请求时；
- 🧐初探了HTTP相关知识，了解了一个网页请求中各个部分的含义，并且对于URL有了比以前更加清晰的认知；
- 🧐通过完成challenge学习了不少关于yield的知识和用途，其是一个很动态、灵活的工具，并且收获了一些有用的知识网站，温习了一些Python知识；

感想:

- 🧐本次实验的难度实际上并不算大，重要的是阅读源代码和官方文档，用到了许多已有模块当中的方法，多个模块、类对象协同工作来完成一个大的逻辑，虽然这样感觉耦合度实在很高，不过阅读源代码的能力实际上尚有欠缺；
- 🧐在继PA大实验之后又遇到了cache多级缓存结构，有不少共同之处；
- 🧐Real Python这个网站里写的东西太有用了，英文网站总有些小惊喜；

