Lab 7: Content Delivery Network

181180050 孔孟荀

Task 1: Preparation

略

Task 2: DNS server

Step 1 : Load DNS Records Table

```
def parse_dns_file(self, dns_file):
    # ---------------------------------------------------
    # TODO: your codes here. Parse the dns_table.txt file
    # and load the data into self._dns_table.
    # ---------------------------------------------------
    with open(dns_file, "r") as f:

        for line in f:
            substr = line.split()

            li = []
            i = 2
            while i<len(substr):
                li.append(substr[i])
                i += 1
            item = DNSItem(domain=substr[0],type = substr[1],values = li)
            self._dns_table.append(item)

    ...
```

在这一步里要把文件里的内容抄到属性self._dns_table里去，这里借鉴了助教gg一个没删掉的结构DNSItem = namedtuple("DNSItem", ("domain", "type", "values"))，_dns_table实际上是这个

结构的一个列表，列表的第三项则是另一个列表。
实际的实现很像前面实验里完成过的forwarding_table
文件的样子如下

dnsServer > dns_table.txt

```
homepage.cncourse.org. CNAME home.cncourse.org.
*.cncourse.org. CNAME home.nasa.org.
*.netlab.org. CNAME home.nasa.org.
home.nasa.org. A 10.0.0.1 10.0.0.2 10.0.0.3
lab.nasa.org. A 10.0.0.4 10.0.0.5
*.localhost.computer A 127.0.0.1

```

Step 2 : Reply Clients' DNS Request

在这一步里要实现一个函数get_response,函数输入一个域名，返回一个查dns_table以后的值

```
def get_response(self, request_domain_name):
    response_type, response_val = (None, None)
    # -----------------------------------------------
    # TODO: your codes here.
    # Determine an IP to response according to the client's IP address.
    #       set "response_ip" to "the best IP address".
    client_ip, _ = self.client_address
    ...
    item = self.findDomain(request_domain_name)
    if item:
        if item[1] == "CNAME":
            response_type = "CNAME"
            response_val = str(item[2][0])
        else :
            response_type = "A"
            if len(item[2])>1:
                res = IP_Utils.getIpLocation(client_ip)
                if not res:
                    ra = random.randint(0,len(item[2])-1)
                    response_val = str(item[2][ra])
                else:
                    min = float('inf')
                    minip = 0
                    for key in item[2]:
                        p = IP_Utils.getIpLocation(key)
                        print(f"p={p}\n")
                        if self.calc_distance(p,res)<min:
                            min = self.calc_distance(p,res)
                            minip = key
                    response_val = str(minip)
            else :
                response_val = str(item[2][0])
    return (response_type, response_val)
```

程序逻辑如下，首先通过一个findDomain函数查一查table里有没有这个域名，这里本应该用正则表达式，但是奈何我不太熟悉相关的语法（在compile一个patten时如何区分输入的变量和常量）,于是偷了个懒做了个暴力的硬编码

```
def findDomain(self, reqDomain):
    for item in self.table:
        print(f"reqDomain:{reqDomain},have:{item.domain}\n")
        if item.domain[0]=='*' and item.domain[1]=='.':
            i = 0
            while reqDomain[i]!='.':
                i+=1
            i += 1
            sub = reqDomain[i:]
            print(f"sub:{sub},item.domain[2:]:{item.domain[2:]}\n")
            if sub == item.domain[2:] or sub == item.domain[2:]+'.':
                print("match1!\n")
                return item
        if item.domain == reqDomain:
            print("match2!\n")
            return item

    return None
```

这个硬编码一开始效果不错，让我过了test，但是在最后的test all让我找了好久的bug，发现在输入的域名经过了resolve domain函数以后最后会补一个'.'，直接导致了我一开始方案的失败

言归正传，如果findDomain函数找不到，返回两个None，如果找到了相关的表项，开始分类

- 1.如果类型是CNAME，直接返回另一个域名
- 2.如果类型是A，有两种可能：
  第一种，对应不止一个ip地址，这时候要根据地理位置来算，通过给的接口IP_Utils.getIpLocation可以完成找到地理位置的功能，然后比较哪个距离短即可。如果这个接口没有找到对应的地理位置，随机选择一个。

  ```
  def calc_distance(self, pointA, pointB):
     ''' TODO: too naive '''
     return math.sqrt((pointA[0]-pointB[0])**2 + (pointA[1]-pointB[1])**2)
  ```

  第二种，就一个ip地址，直接返回即可。

到这里task2就完成了，实现的不太好的地方就是没用正则表达式。
测试结果：

```
njucs@njucs-VirtualBox:~/networklab/lab-7-saltfishmx$ python3 test_entry.py dns
2021/06/02-20:23:35| [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 0.010s

OK
2021/06/02-20:23:36| [INFO] DNS server terminated
njucs@njucs-VirtualBox:~/networklab/lab-7-saltfishmx$
```

Task 3: Caching server

Step1: HTTPRequestHandler

httprequesthandler的任务顾名思义就是 process the HTTP requests，实验要求完成三个函数。理清了它们之间的关系以后发现是这么个意思：server的touch item函数根据自动解析的path能够返回给handler header和body，而handler把他们组装成一个http reply返回给client。do_Get和do_Head的区别是后者不需要返回body，除此之外两者功能相似，这两者都需要调用同样需要我们完成的sendheaders函数。

```
@trace
def do_GET(self):
    ''' Logic when receive a HTTP GET.
    Notice that the URL is automatically parsed and the path is stored in
    self.path.
    '''
    # TODO: implement the logic to response a GET.
    # Remember to leverage the methods in CachingServer.
    headers,body = self.server.touchItem(self.path)
    if headers:
        self.send_response(200)
        self.sendHeaders(headers)
        self.sendBody(body)
    else:
        self.send_error(HTTPStatus.NOT_FOUND, "'File not found'")
    ...

@trace
def do_HEAD(self):
    ''' Logic when receive a HTTP HEAD.
    The difference from self.do_GET() is that do_HEAD() only send HTTP
    headers.
    '''
    # TODO: implement the logic to response a HEAD.
    # Similar to do_GET()
    headers,body = self.server.touchItem(self.path)
    if headers:
        self.send_response(200)
        self.sendHeaders(headers)
    else:
        self.send_error(HTTPStatus.NOT_FOUND, "'File not found'")
    ...
```

不难发现实验已经给了我们接口send_header，说明我们已经能完成单个header的输入到缓冲区，所以只要遍历我们有的header列表（这里我改了下这个函数的参数），把每一条都调用接口send_header即可。
最后还要调用一次end_header接口，它的任务是用一行blank把header部分和body区分开来并且把缓冲区的东西发给客户

```
@trace
def sendHeaders(self,headers):
    ''' Send HTTP headers to client'''
    # TODO: implement the logic of sending headers
    for key in headers:
        self.send_header(key[0],key[1])
    self.end_headers()
    ...
```

做到这里handler已经完成了，在做这个的时候遇到第二个坑，就是一开始教程上没强调要用send_response返回code状态码，所以一开始一直fail，后来问了助教才知道。

Step2: Caching Server

上一步用到了touch item，这一步的任务就是完成这个函数

```
def touchItem(self, path: str):
    ''' Touch the item of path.
    This method, called by HttpHandler, serves as a bridge of server and
    handler.
    If the target doesn't exsit or expires, fetch from main server.
    Write the headers to local cache and return the body.
    '''
    if path in self.cacheTable:
        item = self.cacheTable[path]
        if self.cacheTable.expired(path):
            response = self.requestMainServer(path)
            if response:
                headers = response.getheaders()
                body = response.read()
                self.cacheTable.setHeaders(path,headers)
                self.cacheTable.appendBody(path,body)
                return headers,body
            else:
                return None,None
        else:
            headers = self.cacheTable.getHeaders(path)
            body = self.cacheTable.getBody(path)
            return headers,body
    else:
        response = self.requestMainServer(path)
        if response:
            headers = response.getheaders()
            body = response.read()
            self.cacheTable.setHeaders(path,headers)
            self.cacheTable.appendBody(path,body)
            return headers,body
        else:
            return None,None
```

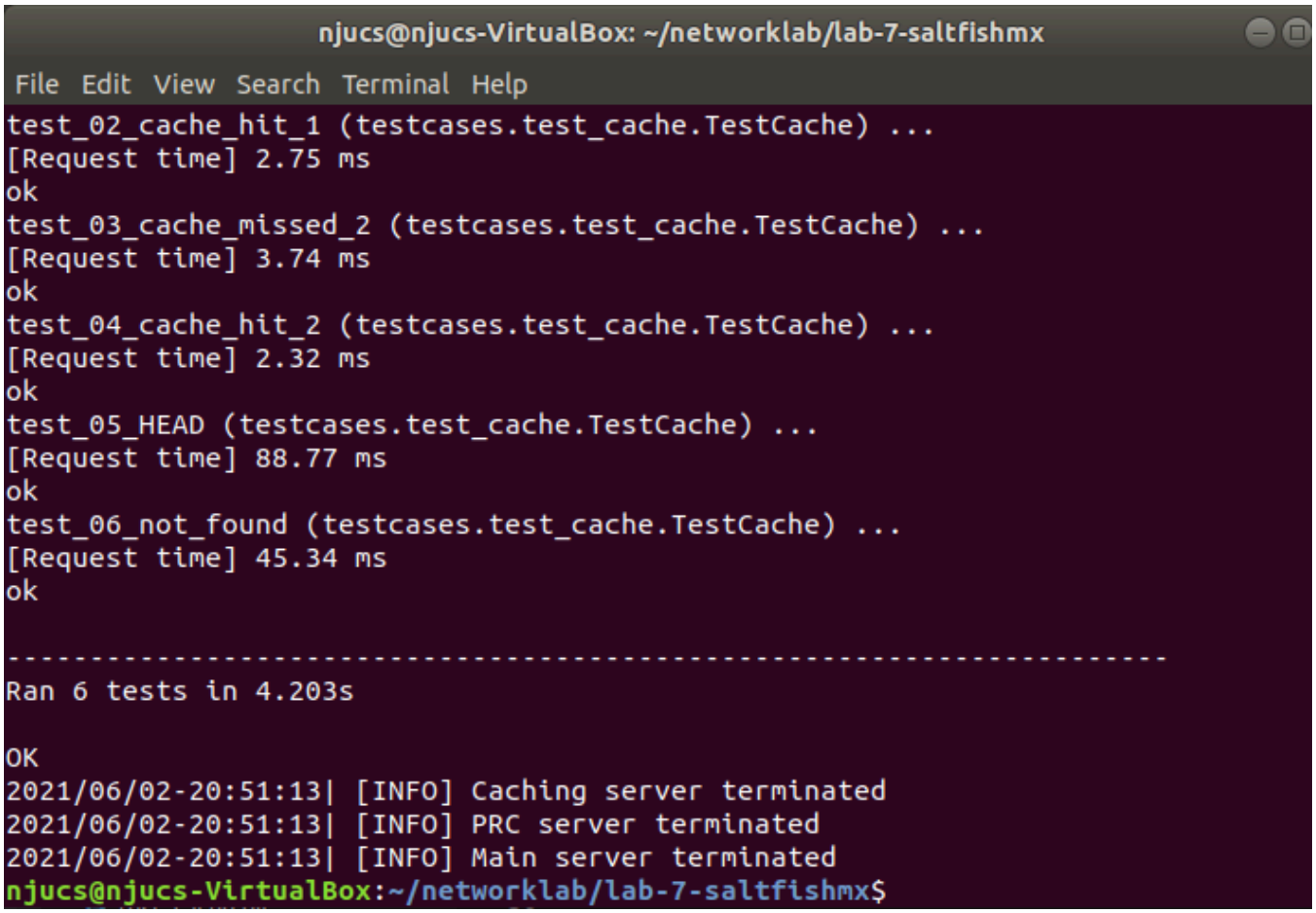
这个函数的任务就是根据一个已有的path去查字典cachetable，cachetable的每个key都是[headers，body，timestamp]，键是path。

- 如果说找不到，或者虽然找到了，但是已经超时了，都需要去remote main server找，然后把找到的结果重新存到cachetable里就行，如果连main server都没有，那只能返回None.
- 如果找到了而且没超时，那么直接返回cachetable的内容就行

Optional Step: Stream Forwarding

没写出来（菜

test



Task 4: Deployment

test

```
njucs@njucs-VirtualBox:~/networklab/lab-7-saltfishmx$ python3 test_entry.py all
2021/06/02-20:52:26| [INFO] DNS server started
2021/06/02-20:52:26| [INFO] Main server started
2021/06/02-20:52:26| [INFO] RPC server started
2021/06/02-20:52:26| [INFO] Caching server started
test_01_cache_missed_1 (testcases.test_all.TestAll) ...
[Request time] 51.28 ms
ok
test_02_cache_hit_1 (testcases.test_all.TestAll) ...
[Request time] 2.38 ms
ok
test_03_not_found (testcases.test_all.TestAll) ...
[Request time] 49.57 ms
ok

----------------------------------------------------------------------
Ran 3 tests in 1.833s

OK
2021/06/02-20:52:29| [INFO] DNS server terminated
2021/06/02-20:52:29| [INFO] Caching server terminated
2021/06/02-20:52:29| [INFO] PRC server terminated
2021/06/02-20:52:29| [INFO] Main server terminated
```

分析opennetlab的结果

Open ▾ 181180050_client.log ~/networklab/lab-7-saltfishmx/report

```
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 6.600s

OK

[Request time] 3880.67 ms

[Request time] 2.51 ms

[Request time] 980.83 ms
```

如图所示，fetched from main server需要3880.67ms，而hits cache只需要2.51ms，时间缩短为了原来的千分之六，效果非常明显。

总结

做完了计网的最后一次实验。在最后这次实验里把自己的代码上传到了真正的网络里，并且效果

很明显，感觉很开心。同时也体会到了这种 Content Delivery Network的效率：通过一个cache server既使得remote main server的压力减小，也提升了用户fetch的速度。

本学期计网实验一直很有趣，每次都刚开放就急着做完（导致两周以后验收的时候忘的差不多了），感谢助教gg们的付出！