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/李文中
  助教：

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	201220102	姓名	武雅琛
Email	201220102@smail.nju.edu.cn	开始/完成日期	2022.6.1

1. 实验名称

Content Delivery Network

2. 实验目的

- **学习应用层DNS服务器的工作原理和提供的具体服务**
- **了解CDN(Content Delivery Network)在互联网通信中发挥的作用**

3. 实验内容

task 1:Preparation

task 2:DNS server

- 这一步要求实现DNS server的部分功能：**建立DNS缓存**和**回应DNS请求**。

step 1:Load DNS Records Table

- **读取文本文件**

  文件列举DNS记录如下：

Domain name	Record Type	Record Values
homepage.cncourse.org.	CNAME	home.cncourse.org.
...	...	...

  只需要使用python提供的文件操作readlines逐行读取，然后以空格为标志spilit每条记录为list即可。

- **处理表项内容**
  - **Domain name**

    域名处理需要得到在查询DNS表时便于查找和匹配的形式。

由于我在接下来采用了**正则表达式**来实现域名匹配，所以在这里我将**域名转换为表示正则表达式的字符串**存入DNS表。

- 对于根域名标志`.`的处理，由于本实验不做区分，所以我的处理是在读入时**统一处理为最后不带点**的格式。
- 对于通配符`*`的处理，我将域名首部的通配符去掉加入了前缀：`[a-zA-Z0-9.]*`，表示在主域名下可以匹配由大小写字母数字和符号`.`间隔的子域名。
- 同时在最后都加入了`$`**符号**确保匹配到域名的结尾。

- **Record Type**

  按照**字符串**存入DNS表。

- **Record Values**

  因为可能匹配一个域名可能有多个ip地址，所以这里存入的数据类型为**list**。

- 程序中保存的DNS表结构如下：

WildCard（通配符）	Domain name	Record Type	Record Values
True or False	[a-zA-Z0-9.]*nju.com$（正则表达式）	CNAME or A（str）	192.168.1.1 or home.org（str）
...	....	...	...

step 2:Reply Clients' DNS Request

- 这一步要求查询step1构造的DNS表并且返回一个元组表示查询结果。主要需要完成**DNS表的查询**和**最优ip地址的选择**。
  - **DNS表的查询**

    这一部分的大部分工作在step1通过将正则表达式放入DNS表完成，只需要遍历表项后使用**正则匹配**：

```
#DNSserver.py
#eg. [a-zA-Z0-9.]*edu.cn, nju.edu.cn = True

if re.match(record[1], match_domain_name):
```

  - 在CNAME类型记录中只需**要返回唯一的域名**即可，但是在A类型记录中可能提供了多个ip地址。
    - 如果只有一个ip地址与之对应，直接返回该地址。
    - 如果表中有多于一个ip地址，首先计算客户端ip的位置。调用API`**IP_Utils.getIpLocation`返回一个**表明对应IP地址的经纬度的元组**。

```
#DNSserver.py
geographicaladdr = IP_Utils.getIpLocation(client_ip)
```

      如果返回得到None，说明地址不可获取，那么随机在表项中选择一个ip地址返回来实现简单的负载均衡。

      如果不是None，那么遍历表项提供的ip地址获取其地址，用简单的距离公式计算客户端和服务器之间的距离，选择最小的服务器ip地址返回。

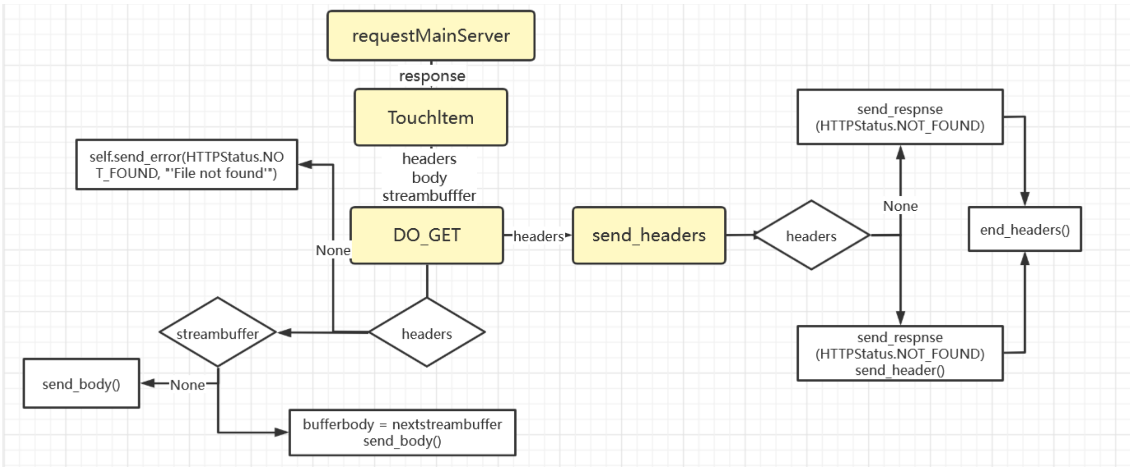
```
dst_dist = (dst_geographicaladdr[0] - geographicaladdr[0])**2 + (dst_geographicaladdr[1] - geographicaladdr[0])**2
```

task 3:Cache server

- 在本实验中Cache server通过将远程Main server提供的http信息在一段时间内储存在Cache Table中实现相应的功能。

  主要包括查询Cache Table， 更新Cache Table，向客户端发送httpheaders和body。在touchItem，sendheaders, DO_GET，DO_HEAD函数中实现。

  函数之间的调用关系以及传递参数关系如下：

  
  requestMainServer
response
TouchItem
headers
body
streambufffer
self.send_error(HTTPStatus.NOT_FOUND, "'File not found'")
None
DO_GET
headers
send_headers
headers
send_respnse (HTTPStatus.NOT_FOUND)
None
end_headers()
send_respnse (HTTPStatus.NOT_FOUND) send_header()
streambuffer
headers
send_body()
None
bufferbody = nextstreambuffer send_body()

  - **touchItem**

    ```
    def touchItem(self, path: str):
        ...
        return headers, body, streambuffer
        ...
    ```

    对于一个path可能会出现以下情况：**从未在Cache中缓存，缓存已超时，缓存且尚未超时。**

    在Cache Table中信息以path为key的字典存储，只需要通过path判断是否在Table中。如果已经在Table中，需要调用提供的函数接口`self.cacheTable.expired(path)`来判断是否超时。

    缓存且尚未超时的情况非常容易，只需要查询http回复的headers和body并返回即可。

    而另外两种情况可以统一为**需要从主服务器获取response更新Cache Table**的情形。

    利用主服务器接口`response = self.requestMainServer(path)`获得一条对应path的response。

    - headers：由于headers通常字节数较少，所以不采用流控制。

      ```
      #touchItem
      headers = response.getheaders()
      self._filterHeaders(headers)
      self.cacheTable.setHeaders(path, headers)
      ```

    - body：为了可以实现**流控制**，定义了**generator函数来获得一个buffer生成器**，从response中不断得读出body的内容。在touchItem函数得最后返回了一个由response生成generator对象。

```python
def StreambufferGen(self,response,path):
...
buffer = bytearray(buffersize)
...
readbytes = response.readinto(buffer)
```

- **sendHeaders**

  ```python
  def sendHeaders(self, headers):
      ...
      if headers != None:
      ...
  ```

  首先发送状态码：send_response

  如果headers为None，说明Cache Server没有收到对应得http回复，返回状态码`404`。

  否则则认为正常接收，发送状态码`200`

  然后逐个发送头部信息：send_header

  遍历传入得list发送即可。

  最后注意调用：end_headers

- **DO_GET**

  对于headers的处理通过sendHeaders完成。同sendHeaders相同，通过headers的内容判断是否收到回复。

  如果未收到回复，返回错误信息`self.send_error(HTTPStatus.NOT_FOUND, "'File not found'")`。

  否则通过`streambuffer`判断是否已经全部读入缓存：

  ```python
  if streambuffer == None:
  ...#已经读入缓存
  else:
  ...#未读入缓存，使用geneator对象生成buffer逐个发送
  ```

- **DO_HEAD**

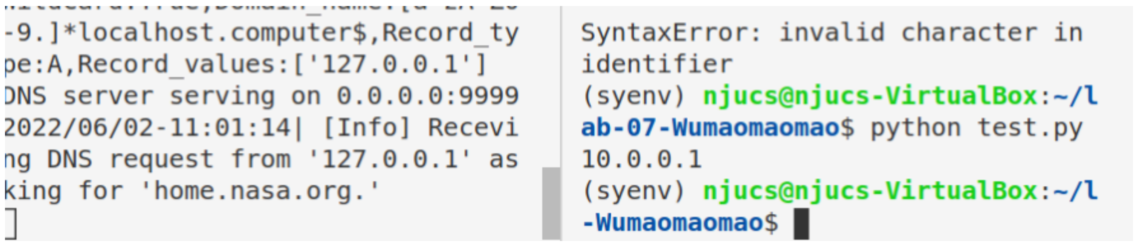
  和DO_GET相近，只需要注意**同样需要使用*genertor*对象确保缓存内容填写到了*Cache Table***，但无需发送给客户端。

4.实验结果

Testcase

task 2:DNS server

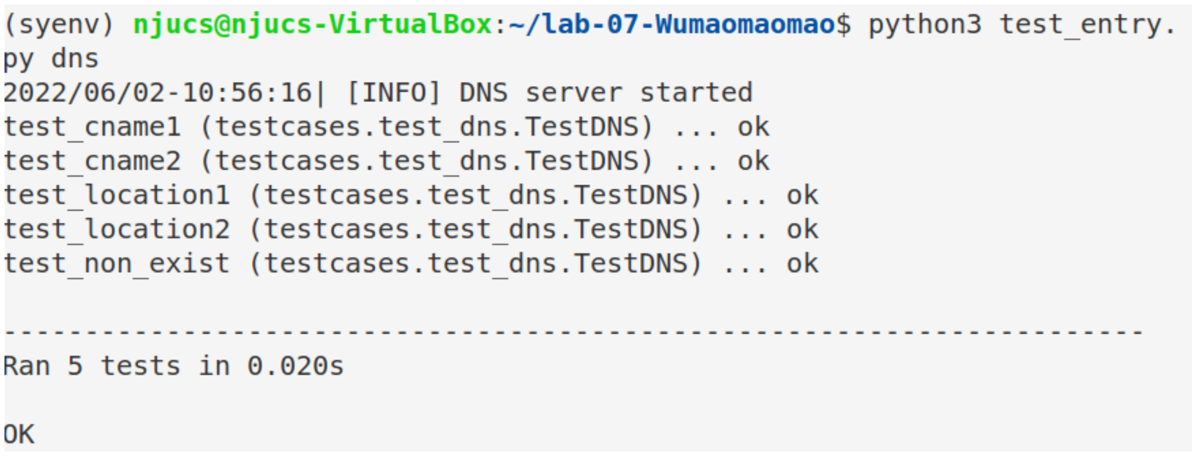
- Manually



```
-9.]*localhost.computer$,Record_ty
pe:A,Record_values:['127.0.0.1']
DNS server serving on 0.0.0.0:9999
2022/06/02-11:01:14| [Info] Recevi
ng DNS request from '127.0.0.1' as
king for 'home.nasa.org.'
]
```

```
SyntaxError: invalid character in
identifier
(syenv) njucs@njucs-VirtualBox:~/l
ab-07-Wumaomaomao$ python test.py
10.0.0.1
(syenv) njucs@njucs-VirtualBox:~/l
-Wumaomaomao$ 
```

- Testcase



```
(syenv) njucs@njucs-VirtualBox:~/lab-07-Wumaomaomao$ python3 test_entry.
py dns
2022/06/02-10:56:16| [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
```

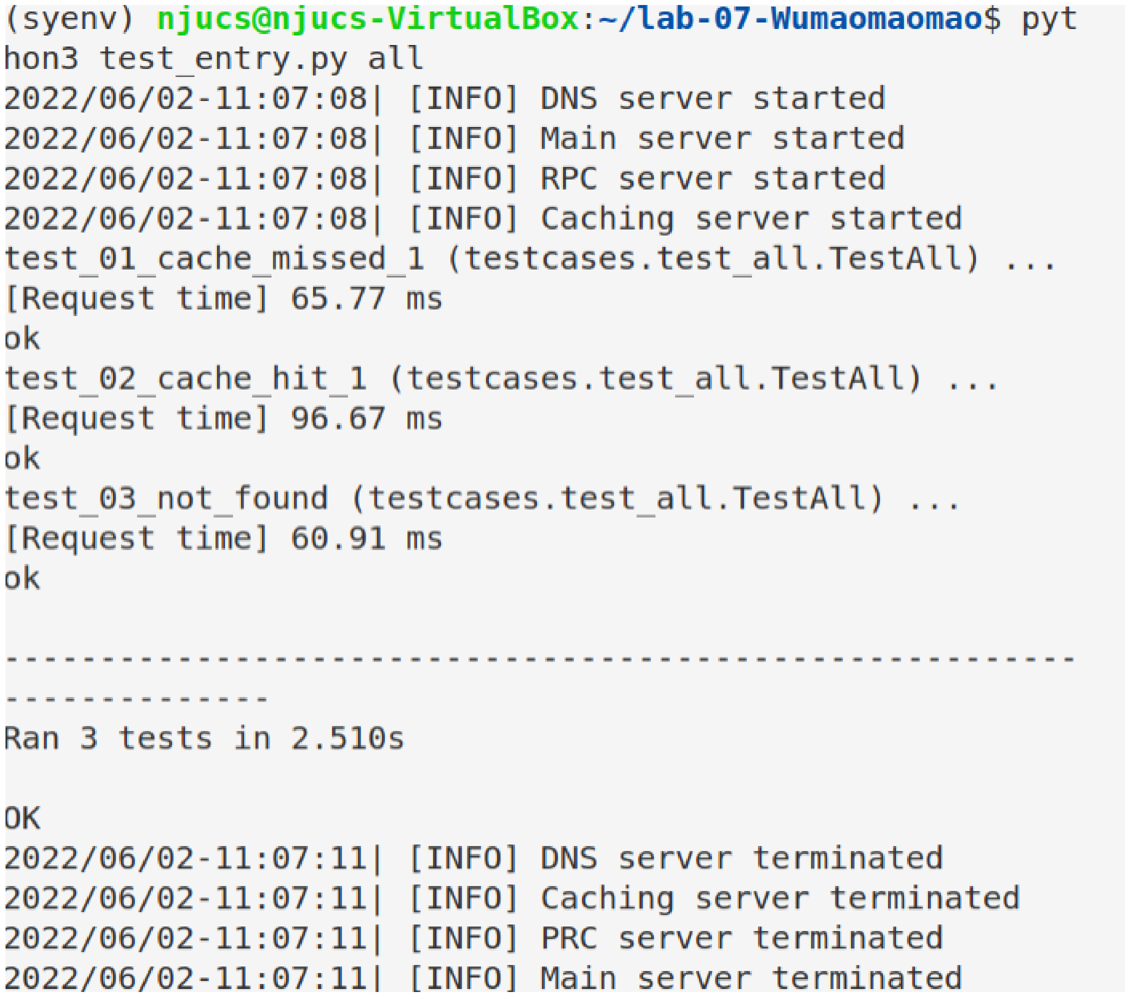
task 3: Cache server

- Testcase

  (没有Manually是因为基本一致略去)

```
ok
test_02_cache_hit_1 (testcases.test_cache.TestCache) ...

[Request time] 93.50 ms
ok
test_03_cache_missed_2 (testcases.test_cache.TestCache)
...
[Request time] 63.39 ms
ok
test_04_cache_hit_2 (testcases.test_cache.TestCache) ...

[Request time] 99.31 ms
ok
test_05_HEAD (testcases.test_cache.TestCache) ...
[Request time] 122.99 ms
ok
test_06_not_found (testcases.test_cache.TestCache) ...
[Request time] 66.21 ms
ok

-------------------------------------------------------
--------------
Ran 6 tests in 5.131s

OK
2022/06/02-11:05:48| [INFO] Caching server terminated
```

task2 & task3



```
(syenv) njucs@njucs-VirtualBox:~/lab-07-Wumaomaomao$ pyt
hon3 test_entry.py all
2022/06/02-11:07:08| [INFO] DNS server started
2022/06/02-11:07:08| [INFO] Main server started
2022/06/02-11:07:08| [INFO] RPC server started
2022/06/02-11:07:08| [INFO] Caching server started
test_01_cache_missed_1 (testcases.test_all.TestAll) ...
[Request time] 65.77 ms
ok
test_02_cache_hit_1 (testcases.test_all.TestAll) ...
[Request time] 96.67 ms
ok
test_03_not_found (testcases.test_all.TestAll) ...
[Request time] 60.91 ms
ok

--------------------------------------------------------
--------------
Ran 3 tests in 2.510s

OK
2022/06/02-11:07:11| [INFO] DNS server terminated
2022/06/02-11:07:11| [INFO] Caching server terminated
2022/06/02-11:07:11| [INFO] PRC server terminated
2022/06/02-11:07:11| [INFO] Main server terminated
```

Deploying

- 见附件

5.核心代码

task 2:DNS server

- **读取本地DNS记录**

```python
dns_src_file = open(dns_file,'r')
        text = dns_src_file.readlines()
        #for each row record
        for entry in text:
            cells = entry.split()
            #exist wildcark mask, append a flag
            if cells[0][-1] == '.':#omit root domain name flag
                cells[0] = cells[0][0:-1]
            if cells[0][0] == '*':
                wildCard = "True"
                Domain_name = cells[0][2:]
                Domain_name = "[a-zA-Z0-9.]*" + Domain_name + "$"
            else:
                wildCard = "False"
                Domain_name = cells[0]
                Domain_name = Domain_name + "$"

            #if cells[1] == "A":
             #   geographicaladdr = IP_Utils.getIpLocation(cells[2][0])
            #  print(f"geographicaladdr:{geographicaladdr},type:{type(geographicaladdr)}")
            Record_type = cells[1]
            cells.pop(0)
            cells.pop(0)
            Record_values = cells;
            self._dns_table.append([WildCard,Domain_name,Record_type,Record_values])
            print(f"WildCard:{WildCard},Domain_name:{Domain_name},Record_type:{Record_type},Record_values:{Record_values}")


```

- **查询DNS表返回元组**

```
client_ip, _ = self.client_address
        match_record = None
        if request_domain_name[-1] == '.':
            match_domain_name = request_domain_name[0:-1]
        else:
            match_domain_name = request_domain_name
        for record in self.table:
            if re.match(record[1], match_domain_name):
                match_record = record
                break
        if match_record != None:
            if match_record[2] == "CNAME":
                response_type = "CNAME"
                response_val = match_record[3][0]
            else:
                recordcnt = len(match_record[3])
                if recordcnt == 1:#only one ip address
                    response_type = "A"
                    response_val = match_record[3][0]
                else:
                    geographicaladdr = IP_Utils.getIpLocation(client_ip)

                    #print(f"geographicaladdr:{geographicaladdr},type:{type(geographicaladdr)}")
                    if geographicaladdr == None:
                        index = random.randint(0,recordcnt)
                        response_type = "A"
                        response_val = match_record[3][index]
                    else:
                        dist = -100#initialize the dist with a min value
                        for ipaddress in match_record[3]:
                            dst_geographicaladdr = IP_Utils.getIpLocation(ipaddress)
                            dst_dist = (dst_geographicaladdr[0] - geographicaladdr[0])**2 + (dst_geographicaladdr[1] - geographicaladdr[0])**2
                            if dist == -100:
                                dist = dst_dist
                                response_type = "A"
                                response_val = ipaddress
                            else:
                                if dst_dist < dist:
                                    dist = dst_dist
                                    response_type = "A"
                                    response_val = ipaddress



        # -------------------------------------------------
        return (response_type, response_val)
```

task 3: Cache server

- **touchItem**

```
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.
        '''
        # TODO: implement the logic described in doc-string

        #headers = self.cacheTable.getHeaders(path)
        headers = None
        body = None
        streambuffer = None
        if path not in self.cacheTable.data:#if not exist in cachetable
            print(f"path:{path} not exist cache!")
            response = self.requestMainServer(path)

            if response:#if receive response from Mainserver
                print(f"get response from main server for the path:{path}")
                headers = response.getheaders()
                #body = response.read()
                self._filterHeaders(headers)
                self.cacheTable.setHeaders(path,headers)
                #self.cacheTable.appendBody(path,body)
                print(response)
                streambuffer = self.StreambufferGen(response, path)
        else:
            ret = self.cacheTable.expired(path)
            if ret == False:
                print(f"get data from main server for the path:{path}")
                headers = self.cacheTable.getHeaders(path)
                body = self.cacheTable.getBody(path)
            else:
                print(f"path:{path} cache timeouts!")
                response = self.requestMainServer(path)
                headers = response.getheaders()
                #print(f"get headers:{headers}")
                #body = response.read()
                if response != None:#if receive response from Mainserver
                    self._filterHeaders(headers)

                    self.cacheTable.setHeaders(path, headers)
                    #self.cacheTable.appendBody(path, body)
                    streambuffer = self.StreambufferGen(response,path)
        #print(f"return headers:{headers}, body:{body}")
        return headers,body,streambuffer
```

- **send_Headers**

```
def sendHeaders(self, headers):
        ''' Send HTTP headers to client'''
        # TODO: implement the logic of sending headers
        if headers != None:
            self.send_response(HTTPStatus.OK)
            for keyword,value in headers:
                self.send_header(keyword, value)
        else:
            self.send_response(HTTPStatus.NOT_FOUND)
            #print("404 not found!\n")
        self.end_headers()
        ...
```

- **DO_GET**

```
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.
        #self.server.touchItem(self.path)
        headers, body, streambuffer = self.server.touchItem(self.path)
        self.sendHeaders(headers)
        if headers != None:
            if streambuffer == None:
                self.sendBody(body)
            else:
                bufferbody = next(streambuffer)
                while bufferbody != None:
                    self.sendBody(bufferbody)
                    bufferbody = next(streambuffer)

        else:
            self.send_error(HTTPStatus.NOT_FOUND, "'File not found'")
        ...
```

- **DO_HEAD**

  与DO_GET相近，不赘述。

6.总结与感想

- 最后一次实验啦，本学期~~迫害助教哥哥~~和各位助教老师一起学习计网实验的经历就要到此结束啦，感谢各位助教哥哥的辛勤付出。
- 可能因为线上课的原因，感觉这个学期过得超级快~还记得刚开学的时候一星期速成git、python、switchyard的惊险操作，还有某次腾讯会议被助教哥哥抓到好几个错误的痛苦经历，以及本人两次反复纠结要不要写bonus，最后还是选择在验收前一天熬夜爆肝写了的无语操作。
- 总之还是非常非常感谢各位助教哥哥的辛勤付出，如果有机会的话，后会有期😘