

Lab 7: Content Delivery Network

191220138 杨飞洋

实验目的

了解CDN的工作原理，通过代码进行DNS和cache的模拟。

背景知识

[BaseHTTPRequestHandler](#)

[python源码参考](#)

[List of HTTP status codes - Wikipedia](#)

[socketserver](#)

[HTTPResponse](#)

实现逻辑

DNS server

首先需要从dns_table.txt中读取数据进 `self._dns_table` 中，可以用python自带的 `readline` 和 `split` 方法来实现

我的 `_dns_table` 的数据结构是一个数组，每一个元素也是一个小list，形如[Domain name, Record type, Record values]

如果 `Record values` 不只一个，那么这个元素的长度就 > 3

`parse_dns_file()` 代码：

```
def parse_dns_file(self, dns_file):
    f = open('./dnsServer/dns_table.txt')
    line = f.readline()
    while(line):
        line = line[:len(line)-1]
        line = line.split(' ')
        print(line)
        self._dns_table.append(line)
        line = f.readline()
    f.close()
```

接下来实现 `DNSHandler` 类中的 `get_response` 函数

由于`.`和`*`的存在, 首先需要自定义一个字符串匹配函数, 如果后面有`.`删去, 如果开头有`, 比较第一个`.`之后的部分

匹配函数:

```
def match(str1, str2):
    temp1 = str1
    temp2 = str2
    if str1[len(str1)-1] == '.':
        temp1 = str1[:len(str1)-1]
    if str2[len(str2)-1] == '.':
        temp2 = str2[:len(str2)-1]
    print(temp1, temp2)
    if temp1[0] != '*':
        return temp1 == temp2
    else:
        idx = temp2.find('.')
        return temp1[1:] == temp2[idx:]
```

在`get_response()`函数中, 遍历`dns_table`, 如果有匹配的, 读取`response_type`, `response_val`, `response_type`就是元素的第二个位置。

如果对应的`Record values`只有一个, 直接返回。

如果有多个, 返回和`client_ip`位置最近的, 如果`client_ip`无法获取位置, 随机抽取。

可以用`IP_Utils.getIpLocation(ipstr)`函数来得到经纬度`(int, int)`

`calc_distance`函数:

```
```python
def calc_distance(self, pointA, pointB):
 ''' TODO: calculate distance between two points '''
 return (pointA[0]-pointB[0])**2 + (pointA[1]-pointB[1])**2
```

get\_response 函数:

```
for item in self.table:
 if match(item[0], request_domain_name):
 response_type = item[1]
 if len(item) == 3:
 response_val = item[2]
 elif len(item) > 3:
 candidate = item[2:len(item)]
 if IP_Utils.getIpLocation(client_ip) == None:
 response_val = choice(candidate)
 else:
 ip_paddr = IP_Utils.getIpLocation(client_ip)
 distance =
self.calc_distance(ip_paddr, IP_Utils.getIpLocation(candidate[0]))
 response_val = candidate[0]
 for item in candidate[1:]:
```

```

 if
self.calc_distance(ip_paddr, IP_Utils.getIpLocation(item)) < distance:
 distance =
self.calc_distance(ip_paddr, IP_Utils.getIpLocation(item))
 response_val = item
 break
 return (response_type, response_val)

```

这样DNS server的逻辑就实现了。

## Caching server

接下来实现Caching serve的逻辑。

### touchItem

首先看 CachingServer 类中的 touchItem 函数实现，这可以算是核心逻辑。

这个函数传入的参数是 path:str，对应 cachetable 类中的 key

先判断这个 path 是否在表中存在，如果存在判断是否过期。

如果存在且不过期，利用 getHeaders 和 getBody 函数读取 headers 和 body，返回一个 tuple: (header,body)，注意 headers 用 \_filterHeaders 函数进行一轮筛选，去除不需要的。

如果不存在或者过期了，利用 requestMainServer 函数向上层询问，返回一个 response，利用 HTTPResponse 类中的 getheaders 和 read 函数读取 headers 和 body，同样过滤一遍 headers，向 cacheTable 中加入这两个信息，此处注意必须先调用 setHeaders 来存储 headers，再调用 appendBody 来存储 body

如果 response 也没有，返回None

代码如下：

```

def touchItem(self, path: str):
 if self.cacheTable.exist(path):#exist
 if not self.cacheTable.expired(path): # timeout
 headers = self.cacheTable.getHeaders(path)
 headers = self._filterHeaders(headers)
 body = self.cacheTable.getBody(path)
 return (headers,body)
 response = self.requestMainServer(path)
 if response:
 headers = response.getheaders()
 headers = self._filterHeaders(headers)
 body = response.read()
 self.cacheTable.setHeaders(path,headers)
 self.cacheTable.appendBody(path,body)
 return (headers,body)
 return None

```

## sendHeaders

从 `_filterHeaders` 函数的注释中可以得知 `headers` 的类型, `headers: List[Tuple[str, str]]` 每个 tuple 存储的是 (keyword,value), 那么遍历 `headers`, 再用 `send_header` 函数发出, 最后 `end_headers` 结束, 就好了。

代码如下:

```
def sendHeaders(self, headers):
 for item in headers:
 self.send_header(item[0], item[1])
 self.end_headers()
```

## do\_GET

`do_GET` 和用户直接交互, 用 `touchItem` 函数来找 `headers` 和 `body`。

如果找到了, 发送一个response为ok, 并且调用 `sendHeaders` 和 `sendBody` 函数发出去。

如果没找到, 回复一个错误信息。

代码如下:

```
def do_GET(self):
 info = self.server.touchItem(self.path)
 if info:
 self.send_response(HTTPStatus.OK)
 self.sendHeaders(info[0])
 self.sendBody(info[1])
 else:
 self.send_error(HTTPStatus.NOT_FOUND, "File not found")
```

## do\_HEAD

和 `do_GET` 逻辑一致, 只是只发出 `headers`。

代码如下:

```
def do_HEAD(self):
 info = self.server.touchItem(self.path)
 if info:
 self.send_response(HTTPStatus.OK)
 self.sendHeaders(info[0])
 #self.sendBody(info[1])
 else:
 self.send_error(HTTPStatus.NOT_FOUND, "File not found")
```

## 测试结果

---

### python3 test\_entry.py dns

```
2021/06/15-20:55:29| [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 0.008s

OK
```

### python3 test\_entry.py cache

```
test_01_cache_missed_1 (testcases.test_cache.TestCache) ... url: http://127.0.0.1:1222/doc/
[Request time] 5.92 ms
ok
test_02_cache_hit_1 (testcases.test_cache.TestCache) ... url: http://127.0.0.1:1222/doc/

[Request time] 89.88 ms
ok
test_03_cache_missed_2 (testcases.test_cache.TestCache) ... url: http://127.0.0.1:1222/doc/success.jpg

[Request time] 47.04 ms
ok
test_04_cache_hit_2 (testcases.test_cache.TestCache) ... url: http://127.0.0.1:1222/doc/success.jpg

[Request time] 43.61 ms
ok
test_05_HEAD (testcases.test_cache.TestCache) ... url: http://127.0.0.1:1222/doc/success.jpg

[Request time] 87.30 ms
ok
test_06_not_found (testcases.test_cache.TestCache) ... url: http://127.0.0.1:1222/noneexist

[Request time] 45.35 ms
ok

Ran 6 tests in 4.520s

OK
```

### python3 test\_entry.py all

```
[Request time] 49.25 ms
ok
test_02_cache_hit_1 (testcases.test_all.TestAll) ... url: http://stfw.localhost.computer:1222/doc/success.jpg


[Request time] 47.00 ms
ok
test_03_not_found (testcases.test_all.TestAll) ... url: http://stfw.localhost.computer:1222/noneexist

[Request time] 46.36 ms
ok

Ran 3 tests in 2.119s

OK
```

**OpenNetLab**

Open  191220138\_client.log  
~/network/lab-7-fengling-aat/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 3.133s

OK

[Request time] 718.80 ms

[Request time] 2.52 ms

[Request time] 718.17 ms
```

查看client的log文件，可以看到第二次的速度比第一次快很多，这就是cache的优势。

手动测试

% Total		% Received		% Xferd		Average Speed		Time	Time	Time	Current
						Dload	Upload	Total	Spent	Left	Speed
100	2125	100	2125	0	0	691k	0	--:--:--	--:--:--	--:--:--	691k

实验心得

通过代码模拟学习了CDN的工作原理。