Programming Assignment 3

Goal **1**

Description **1**

Expression eval service 2

Cache 2

Web API server 4

http.server module 4

/api/evalexpression 4

/api/gettime 4

/status.html 5

Test client with http.client module 5

Config file 6

Grading policy **6**

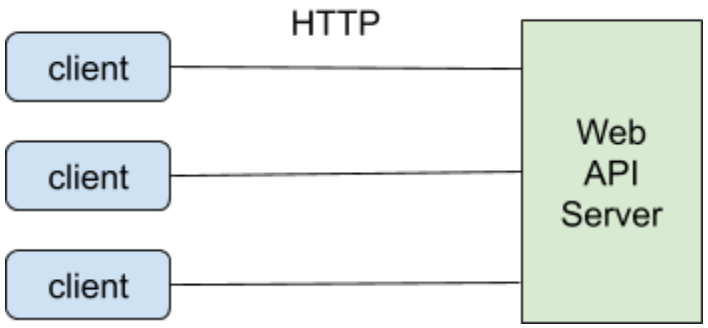
Submission instruction **6**

Goal

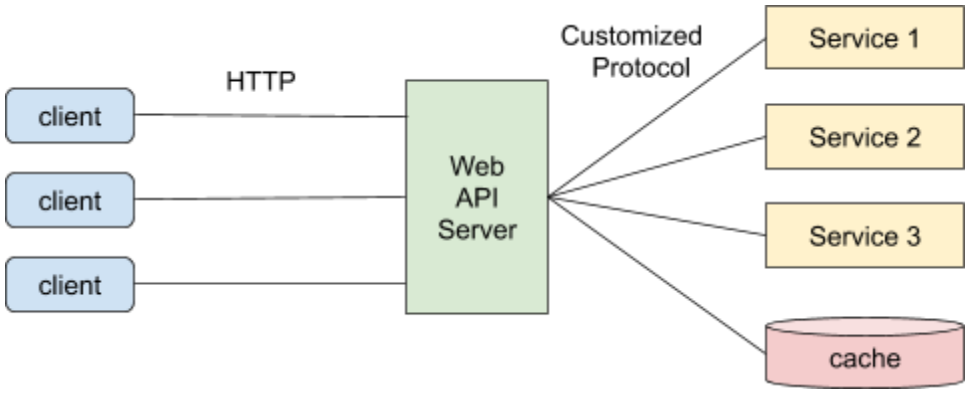
- Understanding the general structure of Web API system.
- Get familiar to program using higher level library (http module).
    - note: in this assignment, you are not allowed to use other high level libraries (e.g. urllib)
- Get familiar to use memcached as a cache service.

Description

In assignment 2, we gained experience on how to program an HTTP server (using socket API) to handle Web APIs. Because the logic is so simple, we programmed everything into a monolithic service inside HTTP server. The architecture looked like the picture below.

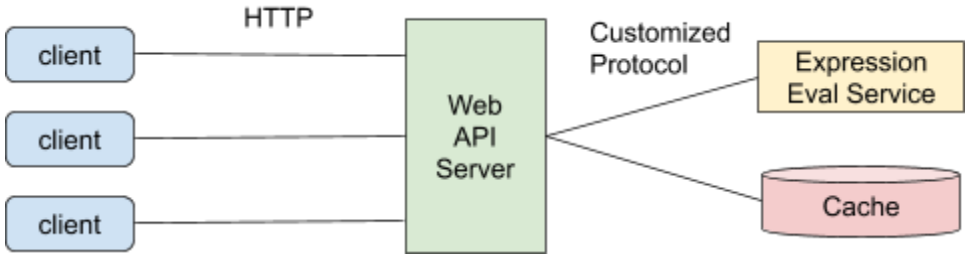


However, in the real world, the logic in Web APIs won't be that simple, and bundling everything into a monolithic service is not a good programming practice. Instead, what you often see is a collection of loosely coupled services, each is responsible for a small set of functionalities. And the Web API layer is just thin layer that fans out to call other services. The architecture usually looks like the picture below.



This architecture scales much better. Difference services can be hosted on different machines, or collocated on the same machine. And they all communicate with each other through network connections and speak predefined protocols (e.g. HTTP, or self-designed protocol in assignment 1). In this assignment, we will improve our toy example to adopt this structure. We will refactor our system into the following three components.

- Web API Server: similar to what's required in assignment 2, we will implement a Web API server that handles
  - `/api/evalexpression`
  - `/api/gettime`
  - `/status.html`
- Expression Eval Service: this will be exactly what you implemented in assignment 1.
- Cache: we will use memcached to store and update our counters on how many times each API is called, and use this to compose the status.html page.



Expression eval service

The specification for expression eval service is exactly the same as assignment 1. You can choose to reuse your solution, or rewrite it. Either way, it needs to follow the specification precise as defined here.

The requirements on expressions stay the same: (1) only integers; (2) only involves + and - operations; (3) don't need to consider overflow. Some examples are listed below

- 7+3-11+6
- 1+1+1+1+1+1+1+1+1+1+1+1+1+1+1+1+1+1+1+1+1+1+1+1+1+1+1+1+1+1+1+1+1+1+1+1+1+1+1+1
- 1-255-256

Cache

Cache is a commonly used technique when building services. In this assignment, we will get some experience on memcached, a popular key-value-pair distributed cache. Please follow the instruction to install it on your local computer. Once installed, you can start the cache service using the command below (e.g. make it listen on port 8182)

```
memcached -p 8182
```

After cache service is started, you can test it using telnet like below, to verify that your installation is correct.

```
telnet localhost 8182
Connected to localhost.
Escape character is '^]'.
get foo
END
set foo 1 0 11
Hello world
STORED
get foo
VALUE foo 1 11
Hello world
END
quit
Connection closed by foreign host.
```

If you can do the above successfully, it means your installation is correct, and memcached service is running as expected. You are ready to make you of it in your Web API service. To make your life easier to call memcached service, there are various Python client library you can use. In particular, you can use pymemcache. You need to fist install this library.

```
pip install pymemcache
```

Once installed, you can make use of the memcached service fairly easily. Sample code below.

```
cache = pymemcache.client.base.Client(('localhost', 8182))
cache.set('foo', '10')
print(cache.get('foo'))
cache.incr('foo', 2)
...
```

All the API description of pymemcache library can be found here. But in this assignment, you only need to learn the following three

- `set(key, value)`
- `get(key)`
- `incr(key, value)`

Web API server

The requirement on API server is essentially the same as assignment 2. But we won't program using the socket API any more. Instead, we will use the higher level http library provided in the Python standard library. To implement the Web server, we will make use of the http.server module.

http.server module

First, we will design a `Handler` class, which will inherit from `BaseHTTPRequestHandler`. This will hold the logic how we handle each URL path.

```
class Handler(BaseHTTPRequestHandler):
      def do_GET(self):
            # TODO: implement logic to handle GET request
            # of different URL path

      def do_POST(self):
            # TODO: implement logic to handle POST request
            # of different URL path
```

There is an attributed (inherited from `BaseHTTPRequestHandler`) you will find very useful, **`self.path`**, which will hold the URL path of the current request.

Once Handler is complete, you can start your server using the code below

```
s = http.server.ThreadingHTTPServer(('localhost', 8181), Handler)
s.serve_forever()
```

The benefit of using higher level library like http, is to free you from handling lower level things (e.g. TCP connections, how to parse HTTP protocol, etc.), and you can focus on the logic you care (aka how to handle each URL path).

/api/evalexpression

Whenever Web API server gets a request of `/api/evalexpression`, it will set up a TCP connection with the expression eval service, and send a request following the specification. Upon receiving the response, the Web API server will form a valid HTTP response with the result, and send back to the client.

Note:

- You can assume there is only one expression per request.

/api/gettime

We won't have a separate service to handle `/api/gettime`. Just copy over whatever you used in the previous assignment, and compose a valid HTTP response using the http.server module.

/status.html

When status.html page is requested, you only need to show the counter information of each API (as below) in a valid HTML page.

```
......
<h1>API count information</h1>
<h3>/api/evalexpression</h3>
<ul>
  <li>last ten minute: 2</li>
  <li>last hour: 10</li>
  <li>last 24 hours: 128</li>
  <li>lifetime: 314</li>
</ul>
<h3>/api/gettime</h3>
<ul>
  <li>last ten minute: 1</li>
  <li>last hour: 2</li>
  <li>last 24 hours: 2</li>
  <li>lifetime: 7</li>
</ul>
......
```

You are required to leverage the memcached service to store the count information. Remember, memcached is a simple key-value-pair cache. You need to be creative to design keys and what values to store.

Test client with http.client module

In the previous assignment, we listed a few different ways to test your Web API server (e.g. using Postman). In this assignment, you are required to implement a test client using http.client module, and show how you test and verify the correctness of you Web API server.

It's very easy to set GET and POST requests using http.client module. For example, you can use something like the following code to send HTTP GET request and read its response.

```
conn = http.client.HTTPConnection(IP, 80)
conn.request('GET', '/')
r = conn.getresponse()
r.read(BUFSIZE)
......
```

Config file

In this assignment, you have three different services running, and they communicate with each other using network TCP connections. To make it a bit easier to manage configurations (e.g. IP address and port numbers), you are required to have a file (named `config.py`) to hold the following settings

```
WEB_API_SERVER = 'localhost'
WEB_API_PORT = 8180
EXPRESSION_EVAL_SERVER = 'localhost'
EXPRESSION_EVAL_PORT = 8181
CACHE_SERVER = 'localhost'
CACHE_PORT = 8182
```

In your implementations of Web API server, expression eval server, you need to import this `config.py` and use the constant defined here to setup your servers.

Grading policy

100 points in total.

- [60 pts] Web API server implementation with http.server module.
    - [20 pts] Use memcached to store counter information, with clearly documented key design.
    - [20 pts] /api/evalexpression.
    - [5 pts] /api/gettime
    - [10 pts] /status.html
    - [5 pts] Return 404 for unsupported URLs
    - Note: max allowed BUFSIZE is 16 when using http module API to read data.
- [10 pts] Eval expression service. Adapt your assignment 1 solution.
    - Note: max allowed BUFSIZE is 16 when using socket module API to read data.
- [20 pts]Test client with http.client module.
    - Show your test cases on /api/evalexpression, /api/gettime, and /status.html.
- [10 pts] Make use of config.py file.
    - Define config.py file as a top level file in your solution directory.
    - Make use of config.py in Web API server and expression eval server to set up service.

Submission instruction

Please your code and report to all TAs and cc z.sun@northeastern.edu by the deadline.