網頁編程入門

上期講到我們把斐波那契數列功能，改寫成了面向對象的版本，實現了一個終端接口。

server.py：

```
class BaseHandler:
  def handle(self, request:str):
    pass

class Server:
  def __init__(self, handlerClass):
    self.handlerClass = handlerClass

  def run(self):
    while True:
      request = input()
      self.handlerClass().handle(request)
```

fib_handle.py：

```
from fib import f
from server import BaseHandler, Server

class FibHandler(BaseHandler):
  def handle(self, request:str):
    n = int(request)
    print('f(n)=', f(n))
    pass

server = Server(FibHandler)
server.run()
```

簡單 Web 服務器

那如何改成 Web 接口呢。

我們把上面的 Server 換成 HTTP 協議的 Server 就行了。先來看看 Python 中的 HTTP 服務器是怎麼樣的。

Python 的標準庫中提供了一個網頁服務器。

```
python -m http.server
```

在終端中運行它。

```
$ python -m http.server
Serving HTTP on :: port 8000 (http://[::]:8000/) ...
```

在瀏覽器中打開便可以看到效果。

這把當前目錄列舉出來了。接著當瀏覽這個網頁時，再回去看終端。這會，很有意思。

```
$ python -m http.server
Serving HTTP on :: port 8000 (http://[::]:8000/) ...
::1 - - [07/Mar/2021 15:30:35] "GET / HTTP/1.1" 200 -
::1 - - [07/Mar/2021 15:30:35] code 404, message File not found
::1 - - [07/Mar/2021 15:30:35] "GET /favicon.ico HTTP/1.1" 404 -
::1 - - [07/Mar/2021 15:30:35] code 404, message File not found
::1 - - [07/Mar/2021 15:30:35] "GET /apple-touch-icon-precomposed.png HTTP/1.1" 404 -
::1 - - [07/Mar/2021 15:30:35] code 404, message File not found
::1 - - [07/Mar/2021 15:30:35] "GET /apple-touch-icon.png HTTP/1.1" 404 -
::1 - - [07/Mar/2021 15:30:38] "GET / HTTP/1.1" 200 -
```

這是網頁訪問日誌。其中 GET 表示網頁服務的一種數據訪問操作。HTTP/1.1 表示使用了 HTTP 的 1.1 版本的協議。

如何用它來打造我們的斐波那契數列服務。先網上找找樣例代碼，稍微改改，寫一個最簡單的 Web 服務器：

```
from http.server import SimpleHTTPRequestHandler, HTTPServer

class Handler(SimpleHTTPRequestHandler):
    def do_GET(self):
      self.send_response(200)
      self.send_header('Content-type', 'text')
```

```
      self.end_headers()
      self.wfile.write(bytes("hi", "utf-8"))

server = HTTPServer(("127.0.0.1", 8000), Handler)

server.serve_forever()
```

這些是不是很眼熟。幾乎跟上面我們使用 `Server` 是一樣的。注意到 `SimpleHTTPRequestHandler` 不是基礎類，還有一個叫 `BaseHTTPRequestHandler`。`SimpleHTTPRequestHandler` 相對於多處理了一些內容。這些加上斐波那契數列處理功能是容易的。

這裡的 `127.0.0.1` 表示本機的地址，`8000` 表示本機的端口。端口怎麼理解呢。就好像家裡的一個窗戶一樣，是家裡跟外界溝通的一個端口。`bytes` 表示把字符串變成字節。`utf-8` 是一種字符串編碼方式。`send_response`、`send_header` 和 `end_headers` 都是在輸出一些內容，來輸出 `HTTP` 協議所規定的內容，好能被瀏覽器所理解。這樣我們在網頁裡就看到了 `hi`。

接著試試再從請求從得到參數。

```
from http.server import SimpleHTTPRequestHandler, HTTPServer
from fib import f
from urllib.parse import urlparse,parse_qs

class Handler(SimpleHTTPRequestHandler):
    def do_GET(self):
      self.send_response(200)
      self.send_header('Content-type', 'text')
      self.end_headers()
      parsed = urlparse(self.path)
      qs = parse_qs(parsed.query)
      result = ""
      if len(qs) > 0:
        ns = qs[0]
        if len(ns) > 0:
          n = int(ns)
          result = str(f(n))
      self.wfile.write(bytes(result, "utf-8"))

server = HTTPServer(("127.0.0.1", 8000), Handler)
```

```
server.serve_forever()
```

有點複雜吧。這裡就是在解析一些參數。

```
self.path=/?n=3
parsed=ParseResult(scheme='', netloc='', path='/', params='', query='n=3', fragment='')
qs={'n': ['3']}
ns=['3']
n=3
```

遞歸進階

讓我們稍稍重構一下代碼。

```
from http.server import SimpleHTTPRequestHandler, HTTPServer
from fib import f
from urllib.parse import urlparse,parse_qs

class Handler(SimpleHTTPRequestHandler):

    def parse_n(self, s):
      parsed = urlparse(s)
      qs = parse_qs(parsed.query)
      if len(qs) > 0:
        ns = qs['n']
        if len(ns) > 0:
          n = int(ns[0])
          return n
      return None

    def do_GET(self):
      self.send_response(200)
      self.send_header('Content-type', 'text')
      self.end_headers()

      result = ""
```

```
        n = self.parse_n(self.path)
        if n is not None:
          result = str(f(n))

        self.wfile.write(bytes(result, "utf-8"))
        self.wfile.write(bytes(result, "utf-8"))

server = HTTPServer(("127.0.0.1", 8000), Handler)

server.serve_forever()
```

引入 `parse_n` 的函數來把從請求路徑中解析得到 `n` 封裝在一塊。

現在程序有這樣的問題。小王請求了斐波那契數列的第 10000 位，過了一些天，小明又請求了斐波那契數列的第 10000 位。兩次，小王和小明都等待了半天，才得到結果。我們該如何提高這個 `Web` 服務的效率呢。

注意到如果 `n` 相同，`f(n)` 的值總是一樣的。我們進行了一番實驗。

```
127.0.0.1 - - [10/Mar/2021 00:33:01] "GET /?n=1000 HTTP/1.1" 200 -
----------------------------------------
Exception occurred during processing of request from ('127.0.0.1', 50783)
Traceback (most recent call last):
    ...
    if v[n] != -1:
IndexError: list index out of range
```

原來數組不夠大，那就把 v 數組改成 10000 吧。

```
v = []
for x in range(10000):
   v.append(-1)
```

然而當 n 為 `2000` 時，出現了遞歸深度溢出錯誤：

```
127.0.0.1 - - [10/Mar/2021 00:34:00] "GET /?n=2000 HTTP/1.1" 200 -
----------------------------------------
Exception occurred during processing of request from ('127.0.0.1', 50821)
Traceback (most recent call last):
```

```
    ...
    if v[n] != -1:
RecursionError: maximum recursion depth exceeded in comparison
```

然而這一切都還挺快的。

為什麼。因為 `f(1)` 到 `f(1000)`，都只需要算一次。這意味著當在算 `f(1000)` 的時候，+ 運算也許只被執行了 1000 次左右。我們知道 `Python` 的遞歸深度大約在 1000 左右。這意味著我們可以這樣優化程序，如果要算 `2000`，那我先算 `1000` 的。不，這樣還是可能會出現遞歸深度溢出錯誤。如果要算 2000，先算 1200 吧。如果要算 1200，先算 400 吧。

這樣算完 400 和 1200 之後，再算 2000，遞歸深度大概在 800 左右，就不會出現遞歸深度溢出錯誤了。

```
v = []
for x in range(1000000):
   v.append(-1)

def fplus(n):
   if n > 800:
      fplus(n-800)
      return f(n)
   else:
      return f(n)

def f(n):
   if v[n] != -1:
      return v[n]
   else:
      a = 0
      if n < 2:
         a = n
      else:
         a = f(n-1) + f(n-2)
      v[n] = a
      return v[n]
```

增加了 `fplus` 函數。

然而不禁讓人想，`fplus` 被遞歸調用 `1000` 次怎麼樣。1000 * 800 = 800000。當我把 n 設為 80 萬之後，又出現遞歸深度錯誤了。繼續試探了一下，發現事情更複雜。然而這樣優化之後，算 2000 是非常輕鬆的了。

文件讀寫

似乎把話題岔開了。回到 Web 開發的話題上。第一次請求 `f(400)`，第二次請求 `f(600)`。那麼第二次請求時，第一次請求所產生的 `v` 數組的值，我們是能用上的。然而當我們把程序退出。再啟動就用不上了。按我們的方法，斐波那契數列計算是很快的。然而設想，如果很慢怎麼辦。尤其就如當我們沒有引入 v 數組的時候，有著大量重複的計算。這時我們希望能把好不容易得到的結果保存起來。

這時，就引入緩存的概念了。`v` 數組這裡就是一個緩存。不過它只存在於程序生命週期裡。程序關閉後，它就沒了。怎麼辦呢。很自然，我們會想到存到文件裡去。

如何把 v 數組保存到文件呢。

```
0 0
1 1
2 1
3 2
4 3
...
```

我們的 `v` 數組可以這樣保存。每一行保存為 `n f(n)`。既然 `n` 是自然增長的。或許我們可以只保存 `f(n)` 值。

```
0
1
1
2
3
...
```

來試試吧。

```
f = open("demofile2.txt", "a")
f.write("Now the file has more content!")
```

```
f.close()

#open and read the file after the appending:
f = open("demofile2.txt", "r")
print(f.read())
```

open 的第二個參數可以是 a，表示會加在文件末尾；或者是 w，表示會覆蓋掉文件。

```
file = open('fib_v', 'a')
file.write('hi')
file.close()
```

運行一下，果然有文件 fib_v。

```
fib_v:

hi
```

當我們再運行一次的時候，變成了這樣。

```
hihi
```

如何換行呢。

```
file = open('fib_v', 'a')
file.write('hi\n')
file.close()
```

這會打印一次，出現了 hihihi，沒看見換行呢。然而再打印一次，換行了。可見第一次已經打印了換行符，只是在末尾，看不見。

如何讀取呢。

```
file = open('fib_v', 'r')
print(file.read())

$ python fib.py
hihihi
hi
```

接下來，改改我們的斐波那契程序。

```
v = []
for x in range(1000000):
   v.append(-1)

def read():
   file = open('fib_v', 'r')
   s = file.read()
   if len(s) > 0:
      lines = s.split('\n')
      if (len(lines) > 0):
        for i in range(len(lines)):
           v[i] = int(lines[i])

def save():
   file = open('fib_v', 'w')
   s = ''
   start = True
   for vv in v:
      if vv == -1:
         break
      if start == False:
         s += '\n'
      start = False
      s += str(vv)
   file.write(s)
   file.close()

def fcache(n):
   x = fplus(n)
   save()
   return x

def fplus(n):
   if n > 800:
      fplus(n-800)
```

```
        return f(n)
    else:
        return f(n)

def f(n):
    if v[n] != -1:
        return v[n]
    else:
        a = 0
        if n < 2:
            a = n
        else:
            a = f(n-1) + f(n-2)
        v[n] = a
        return v[n]

read()
fcache(10)
save()
```

終於我們寫好程序了。程序運行後，fib_v 文件是這樣的。

```
fib_v:

0
1
1
2
3
5
8
13
21
34
55
```

看到上面的解析有點麻煩。\n 是換行符。有沒有更簡單統一的解析方式。人們發明了 JSON 這件數據格式。

JSON

JSON 的全名是 `JavaScript Object Notation`。以下是 `JSON` 的例子。

```
{"name":"John", "age":31, "city":"New York"}
```

以上這樣來表示一種映射。

JSON 有這樣基本元素：

1. 數字或字符串
2. 列表
3. 映射

而這些基本元素又可以任意嵌套。就是列表裡可以有列表。映射裡也可以有列表。等等

```
{
  "name":"John",
  "age":30,
  "cars":[ "Ford", "BMW", "Fiat" ]
}
```

寫成一行，和這樣寫得好看點是意義上的差別的。或許我們可以想像它們的計算圖。空格不會影響他們的計算圖。

接著我們要把 v 數組變成 `json` 格式的字符串。

```
import json

v = []
for x in range(1000000):
   v.append(-1)

def fplus(n):
   if n > 800:
      fplus(n-800)
      return f(n)
   else:
      return f(n)
```

```
def f(n):
    if v[n] != -1:
        return v[n]
    else:
        a = 0
        if n < 2:
            a = n
        else:
            a = f(n-1) + f(n-2)
        v[n] = a
        return v[n]


fplus(100)
s = json.dump(v)
file = open('fib_j', 'w')
file.write(s)
file.close()
```

當我們這麼寫的時候。報錯了。`TypeError: dump() missing 1 required positional argument: 'fp'`。在 `vscode` 上可以這樣來看到函數定義。

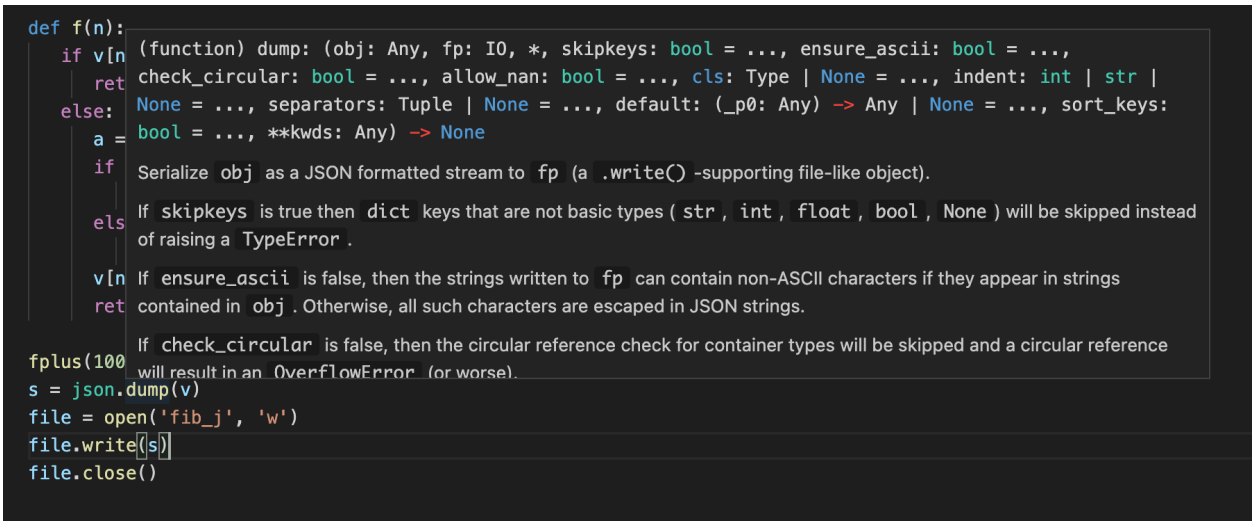

Figure 1: json

可以用鼠標移動到 `dump` 上就行。很方便吧。

```
fplus(10)
```

```
file = open('fib_j', 'w')
json.dump(v, file)
file.close()
```

計算到 100 顯示的數有點多，這裡改為 10。原來 dump 的第二個參數傳如 `file` 對象就行。

這樣可以看到文件：

```
[0, 1, 1, 2, 3, 5, 8, 13, 21, 34, 55, -1, -1, -1]
```

注意後面省略了很多`-1`。

```
def read():
    file = open('fib_j', 'r')
    s = file.read()
    sv = json.loads(s)
    for i in range(len(sv)):
        if sv[i] != -1:
            v[i] = sv[i]
def save():
    file = open('fib_j', 'w')
    json.dump(v, file)
    file.close()

read()

for vv in v:
    if vv!=-1:
        print(vv)
```

當這樣時，可見打印出了：

```
0
1
1
2
3
5
```

```
8
13
21
34
55
```

這幾個函數一起檢查一下：

```
def read():
    file = open('fib_j', 'r')
    s = file.read()
    sv = json.loads(s)
    for i in range(len(sv)):
        v[i] = sv[i]



def save():
    sv = []
    for i in range(len(v)):
        if v[i] != -1:
            sv.append(v[i])
        else:
            break
    file = open('fib_j', 'w')
    json.dump(sv, file)
    file.close()

read()
fplus(100)
save()
```

然後到文件查看，果然保存了正確的值，而且很整齊。

“ ‘json [0, 1, 1, 2, 3, 5, 8, 13, 21, 34, 55, 89, 144, 233, 377, 610, 987, 1597, 2584, 4181, 6765, 10946, 17711, 28657, 46368, 75025, 121393, 196418, 317811, 514229, 832040, 1346269, 2178309, 3524578, 5702887, 9227465, 14930352, 24157817