



Python

实现简单的Web服务器

Python实现简单的Web服务器



Wayne Shi

Learn by doing

关注他

275 人赞了该文章

本课程将通过使用 Python 语言实现一个 Web 服务器，探索 HTTP 协议和 Web 服务的基本原理，同时学习 Python 如何实现 Web 服务请求、响应、错误处理及 CGI 协议，最后会使用 Python 面向对象思路进行重构。

本课程由 ekCit 发布在 [实验楼](#)，详细教程及在线练习地址：[Python - Python实现简单的Web服务器](#)

一、课程介绍

1. 课程来源

本课程核心部分来自《500 lines or less》项目，作者是 Mozilla 的 Greg Wilson。项目代码使用 MIT 协议，项目文档使用 [Creative Commons Legal Code](#) 协议。

课程内容在原文档基础上做了稍许修改，增加了部分

▲ 赞同 275 ▼

● 6 条评论

➤ 分享



互联网在过去20年里已经大大地改变了我们的生活方式，影响着社会。但是反观互联网，它的基础 - web的核心原理并没有改变多少。大部分web系统仍旧遵守 Tim Berners-Lee 20 多年前提出的 W3C 标准，大部分web服务器接收的信息格式与接收的方式与过去并无二致。

本课程将通过使用 Python 语言实现一个 Web 服务器，探索 HTTP 协议和 Web 服务的基本原理，同时学习 Python 如何实现 Web 服务请求、响应、错误处理及CGI协议，最后会根据项目需求使用 Python 面向对象思路对代码进行重构。

3. 课程知识点

本课程项目完成过程中，我们将学习：

1. HTTP 协议基本原理
2. 简单的 Web 服务器框架
3. Python 语言的网络开发
4. Web 服务请求，响应及错误处理的实现
5. CGI 协议的 Python 实现
6. 使用 Python 面向对象思想重构代码

二、实验环境

打开终端，进入Code目录，创建 web-server 文件夹, 并将其作为我们的工作目录。

```
$ cd Code
$ mkdir web-server && cd web-server
```

本实验使用httpie代替浏览器发送请求并在终端打印响应信息。

```
$ sudo apt-get install httpie
```

三、实验原理

想要了解http原理， [Introduction to HTTP](#) 是一个不错的选择，也可以参考它的中文翻译版：[HTTP 下午茶](#)

这里我们简单过一遍我们需要了解的部分。

一般我们的web程序都运行在 TCP/IP 协议上，程序之间使用 socket(套接字) 进行通信，它能够让计算机之间的通信就像写文件和读文件一样简单。一个 tcp socket 由一个IP地址和端口号组成。

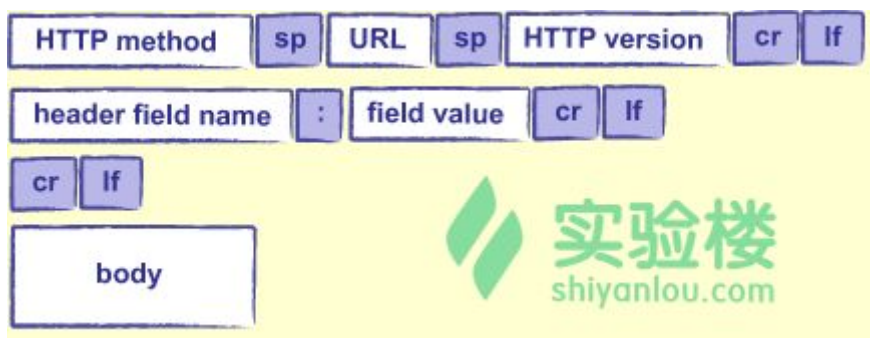
- IP地址是一个32位的二进制数，通常被分割为4

号进行标识。端口号0-1023 是保留给操作系统使用的，我们可以使用剩下的端口号。

超文本传输协议（HTTP）描述了一种程序之间交换数据的方法，它非常简单易用，在一个socket连接上，客户端首先发送请求说明它需要什么，然后服务器发送响应，并在响应中包含客户端的数据。响应数据也许是从本地磁盘上复制来的，也许是程序动态生成的。传输过程如图：



HTTP请求就是一段文本，任何程序都能生成一个http请求，就像生成文本一样简单。这段文本需要包含以下这些部分：



- HTTP method: HTTP请求方法。最常用的就是 GET(抓取数据)与POST(更新数据或者上传文件)
- URL: 通常是客户端请求的文件的路径，比如 /research/experiments.html，但是是否响应文件都是由服务器决定的。
- HTTP version: HTTP版本。通常是 HTTP/1.0 或 HTTP/1.1
- header field: HTTP头内的键值对，做一些基本设置，就像下面这样。

客户端接受的数据类型

Accept: text/html

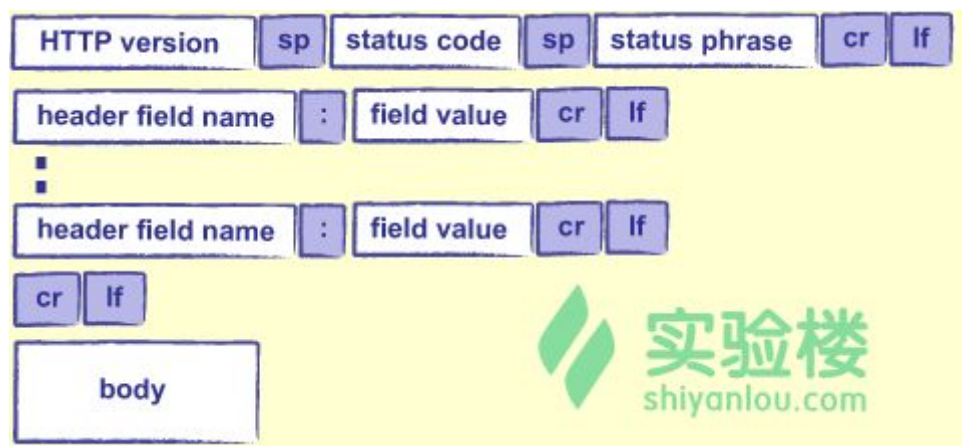


Accept-Language: en, tr

If-Modified-Since: 16-May-2005

- **body**: 一些与请求有关的负载数据了。比如在一个网站登陆的时候提交登陆表单, 那负载数据就是你的账号与密码信息了。

HTTP响应的结构类似于请求:



- **status code**: 状态码。请求成功响应200, 请求的文件找不到则响应404。
- **status phrase**: 对状态码的描述。

四、实验步骤

1.你好, web

现在就来写我们第一个web服务器吧, 基本概念非常简单:

1. 等待某个人连接我们的服务器并向我们发送一个HTTP请求
2. 解析该请求
3. 了解该请求希望请求的内容
4. 服务器根据请求抓取需要的数据 (从服务器本地文件中读或者程序动态生成)
5. 将数据格式化为请求需要的格式
6. 送回HTTP响应

步骤1, 2, 6的操作对所有web应用都是一样的, 这部分内容Python标准库中的BaseHTTPServer 模块可以帮助我们处理。我们只需要关注步骤3~5。

首先在工作目录下创建 server.py 文件



```
class RequestHandler(BaseHTTPServer.BaseHTTPRequestHandler):
    '''处理请求并返回页面'''

    # 页面模板
    Page = '''\
        <html>
        <body>
        <p>Hello, web!</p>
        </body>
        </html>
    '''

    # 处理一个GET请求
    def do_GET(self):
        self.send_response(200)
        self.send_header("Content-Type", "text/html")
        self.send_header("Content-Length", str(len(self.Page)))
        self.end_headers()
        self.wfile.write(self.Page)

#-----

if __name__ == '__main__':
    serverAddress = ('', 8080)
    server = BaseHTTPServer.HTTPServer(serverAddress, RequestHandler)
    server.serve_forever()
```

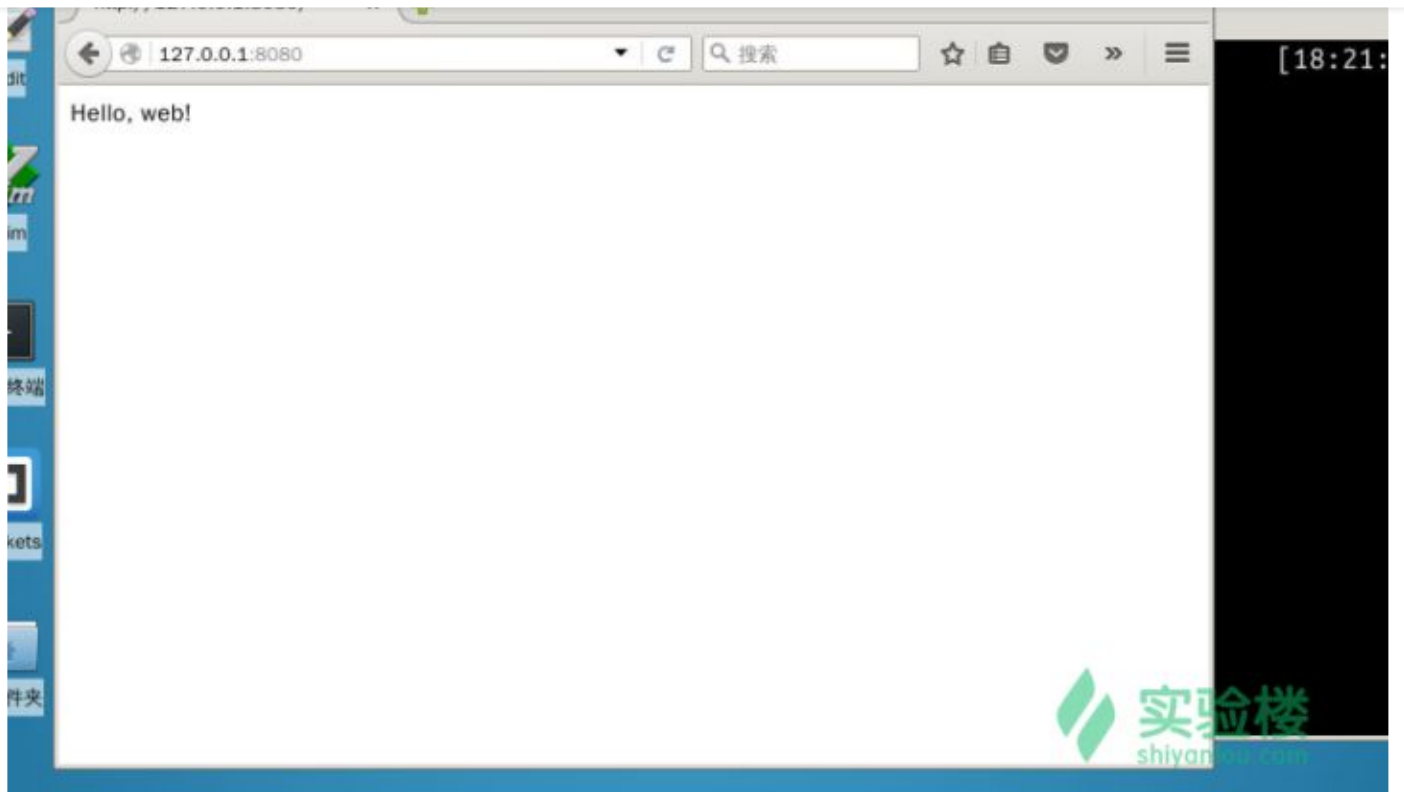
模块的 `BaseHTTPRequestHandler` 类会帮我们处理对请求的解析，并通过确定请求的方法来调用其对应的函数，比如方法是 `GET`，该类就会调用名为 `do_GET` 的方法。`RequestHandler` 继承了 `BaseHTTPRequestHandler` 并重写了 `do_GET` 方法，其效果如代码所示是返回 `Page` 的内容。`Content-Type` 告诉了客户端要以处理html文件的方式处理返回的内容。`end_headers` 方法会插入一个空白行，如之前的request结构图所示。

运行我们的第一个 web服务器

```
$ python server.py
```

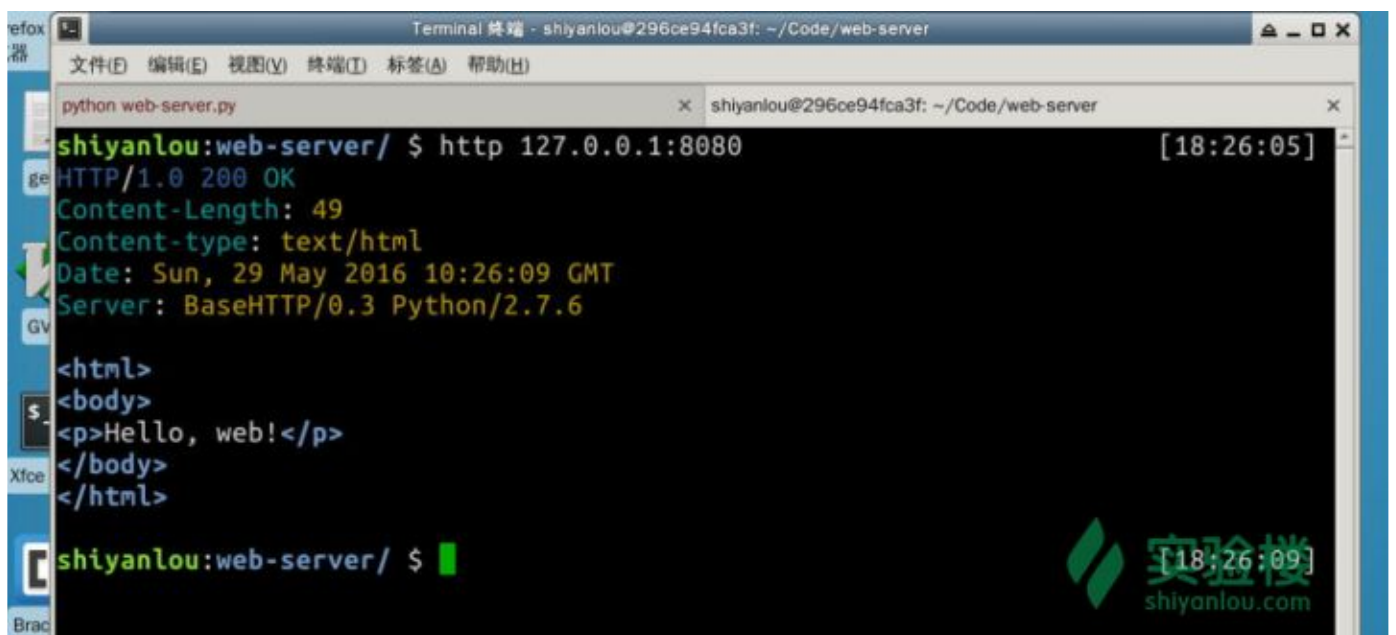
可以在浏览器地址输入 127.0.0.1:8080 进行查看





方便起见，还是让我们新开一个终端窗口，使用httpie来查看输出(之后都使用httpie来查看输出)

```
$ http 127.0.0.1:8080
```



httpie很贴心地显示了响应报文的全部内容。



修改之前的代码来显示请求的信息，同时重新整理一下代码：

```
class RequestHandler(BaseHTTPServer.BaseHTTPRequestHandler):

    # ...页面模板...
    Page = "...待设计..."

    def do_GET(self):
        page = self.create_page()
        self.send_content(page)

    def create_page(self):
        # ...待实现...

    def send_content(self, page):
        # ...待实现...
```

send_content 与之前 do_GET 内的代码一样：

```
def send_content(self, page):
    self.send_response(200)
    self.send_header("Content-type", "text/html")
    self.send_header("Content-Length", str(len(page)))
    self.end_headers()
    self.wfile.write(page)
```

设计页面模版

```
Page = '''\
<html>
<body>
<table>
<tr> <td>Header</td>          <td>Value</td>          </tr>
<tr> <td>Date and time</td>    <td>{date_time}</td>    </tr>
<tr> <td>Client host</td>      <td>{client_host}</td>  </tr>
<tr> <td>Client port</td>      <td>{client_port}</td> </tr>
<tr> <td>Command</td>          <td>{command}</td>      </tr>
<tr> <td>Path</td>             <td>{path}</td>          </tr>
</table>
</body>
</html>
...
'''
```



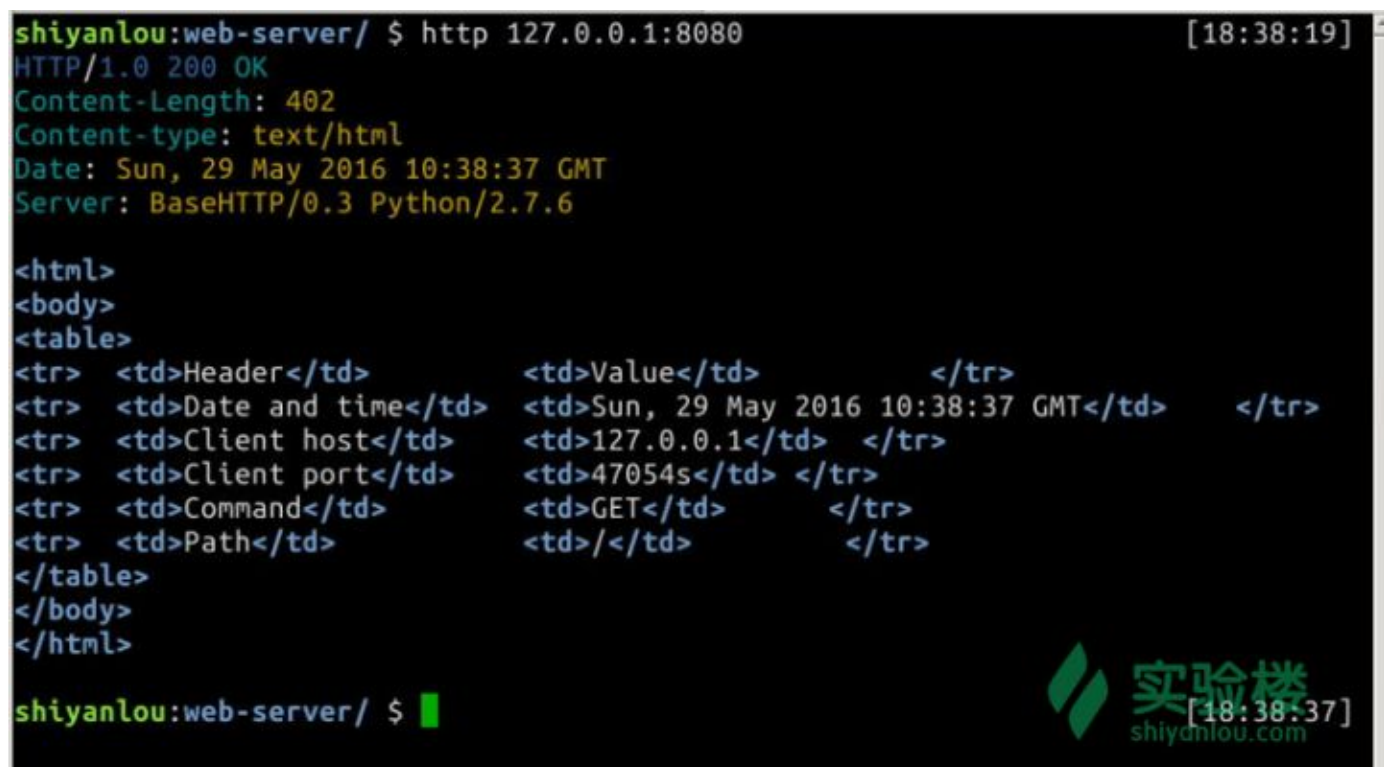
```
def create_page(self):
    values = {
        'date_time' : self.date_time_string(),
        'client_host' : self.client_address[0],
        'client_port' : self.client_address[1],
        'command' : self.command,
        'path' : self.path
    }
    page = self.Page.format(**values)
    return page
```

main中的内容不用去修改它。

运行看看

```
$ http 127.0.0.1:8080/something.html
```

效果图：



```
shiyanolou:web-server/ $ http 127.0.0.1:8080 [18:38:19]
HTTP/1.0 200 OK
Content-Length: 402
Content-type: text/html
Date: Sun, 29 May 2016 10:38:37 GMT
Server: BaseHTTP/0.3 Python/2.7.6

<html>
<body>
<table>
<tr> <td>Header</td> <td>Value</td> </tr>
<tr> <td>Date and time</td> <td>Sun, 29 May 2016 10:38:37 GMT</td> </tr>
<tr> <td>Client host</td> <td>127.0.0.1</td> </tr>
<tr> <td>Client port</td> <td>47054s</td> </tr>
<tr> <td>Command</td> <td>GET</td> </tr>
<tr> <td>Path</td> <td>/</td> </tr>
</table>
</body>
</html>

shiyanolou:web-server/ $
```

注意到它仍旧返回了200 OK而不是404 Not Found，即使 something.html 文件并不存在。那是因为我们现在的web服务器还没有实现找不到文件就返回404错误的功能。反过来说，只要我们能，可以通过编程实现任何我们想要的效果，像是随机返回一个维基百科的页面或是帮老王买一个披萨(并不会)。



3.响应静态页面

所以这一步就该处理静态页面了，处理静态页面就是根据请求的页面名得到磁盘上的页面文件并返回。

在当前目录下创建新文件 plain.html，这是我们测试用的静态页面

```
<html>
<head>
<title>Plain Page</title>
</head>
<body>
<h1>Plain Page</h1>
<p>Nothin' but HTML.</p>
</body>
</html>
```

在 server.py 中导入需要的库

```
import sys, os, BaseHTTPServer
```

为我们的服务器程序写一个异常类

```
class ServerException(Exception):
    '''服务器内部错误'''
    pass
```

重写 do_GET 函数：

```
def do_GET(self):
    try:

        # 文件完整路径
        full_path = os.getcwd() + self.path

        # 如果该路径不存在...
        if not os.path.exists(full_path):
            #抛出异常：文件未找到
            raise ServerException("{}' not found".format(self.path))

        # 如果该路径是一个文件
```



```
self.handle_file(full_path)
```

```
# 如果该路径不是一个文件
else:
    #抛出异常：该路径为不知名对象
    raise ServerException("Unknown object '{0}'".format(self.path))

# 处理异常
except Exception as msg:
    self.handle_error(msg)
```

首先看完整路径的代码，`os.getcwd()` 是当前的工作目录，`self.path` 保存了请求的相对路径，不要忘了 `RequestHandler` 继承自 `BaseHTTPRequestHandler`，它已经帮我们将请求的相对路径保存在 `self.path` 中了。

编写文件处理函数：

```
def handle_file(self, full_path):
    try:
        with open(full_path, 'rb') as reader:
            content = reader.read()
            self.send_content(content)
    except IOError as msg:
        msg = "'{0}' cannot be read: {1}".format(self.path, msg)
        self.handle_error(msg)
```

注意到我们是以二进制模式打开文件的，这样读文件的时候就不会对读取的内容做多余的处理。

接着，实现我们的错误处理函数并设计错误页面模板

```
Error_Page = """\
<html>
<body>
<h1>Error accessing {path}</h1>
<p>{msg}</p>
</body>
</html>
"""

def handle_error(self, msg):
    content = self.Error_Page.format(path=self.path, msg=msg)
    self.send_content(content)
```




```
$ http 127.0.0.1:8080/plain.html
```

效果图

```
shyanlou:web-server/ $ http 127.0.0.1:8080/plain.html [19:17:52]
HTTP/1.0 200 OK
Content-Length: 152
Content-type: text/html
Date: Sun, 29 May 2016 11:18:12 GMT
Server: BaseHTTP/0.3 Python/2.7.6

<html>
  <head>
    <title>Plain Page</title>
  </head>
  <body>
    <h1>Plain Page</h1>
    <p>Nothin' but HTML.</p>
  </body>
</html>
```




再测试一下错误的路径

```
$ http 127.0.0.1:8080/something.html
```

```
shyanlou:web-server/ $ http 127.0.0.1:8080/something.html [19:12:46]
HTTP/1.0 200 OK
Content-Length: 162
Content-type: text/html
Date: Sun, 29 May 2016 11:13:12 GMT
Server: BaseHTTP/0.3 Python/2.7.6

<html>
  <body>
    <h1>Error accessing /something.html</h1>
    <p>' /something.html' not found</p>
  </body>
</html>
```



确实返回了错误页面但同时注意到返回的是200状态码，我们希望它能够返回404，所以还需要修改一下 `handle_error` 与 `send_content` 函数

```
def handle_error(self, msg):
    content = self.Error_Page.format(path=self.path, msg=msg)
    self.send_content(content, 404)
```

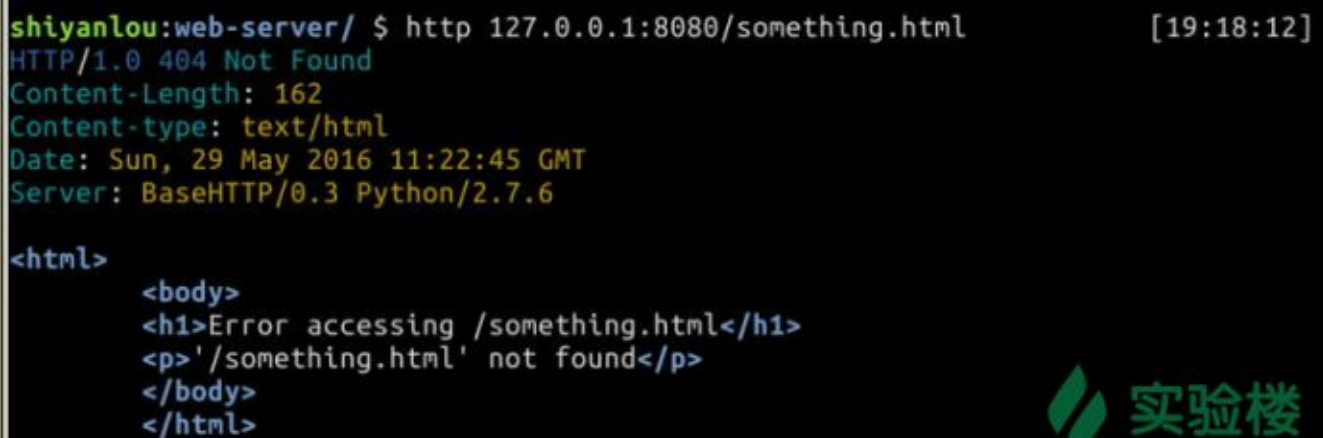


```
self.send_response(status)
self.send_header("Content-type", "text/html")
self.send_header("Content-Length", str(len(content)))
self.end_headers()
self.wfile.write(content)
```

测试看看

```
$ http 127.0.0.1:8080/something.html
```

这回就对了。



```
shyanlou:web-server/ $ http 127.0.0.1:8080/something.html [19:18:12]
HTTP/1.0 404 Not Found
Content-Length: 162
Content-type: text/html
Date: Sun, 29 May 2016 11:22:45 GMT
Server: BaseHTTP/0.3 Python/2.7.6

<html>
  <body>
    <h1>Error accessing /something.html</h1>
    <p>'something.html' not found</p>
  </body>
</html>
```



至今为止的代码：

```
#-*- coding:utf-8 -*-
import sys, os, BaseHTTPServer

class ServerException(Exception):
    '''服务器内部错误'''
    pass

class RequestHandler(BaseHTTPServer.BaseHTTPRequestHandler):

    Error_Page = """\
    <html>
    <body>
    <h1>Error accessing {path}</h1>
    <p>{msg}</p>
    </body>
```



```
def do_GET(self):
    try:

        # 文件完整路径
        full_path = os.getcwd() + self.path

        # 如果该路径不存在...
        if not os.path.exists(full_path):
            #抛出异常：文件未找到
            raise ServerException("'{}' not found".format(self.path))

        # 如果该路径是一个文件
        elif os.path.isfile(full_path):
            #调用 handle_file 处理该文件
            self.handle_file(full_path)

        # 如果该路径不是一个文件
        else:
            #抛出异常：该路径为未知对象
            raise ServerException("Unknown object '{}'.format(self.path))

    # 处理异常
    except Exception as msg:
        self.handle_error(msg)

def handle_error(self, msg):
    content = self.Error_Page.format(path=self.path, msg=msg)
    self.send_content(content, 404)

def send_content(self, content, status=200):
    self.send_response(status)
    self.send_header("Content-type", "text/html")
    self.send_header("Content-Length", str(len(content)))
    self.end_headers()
    self.wfile.write(content)

def handle_file(self, full_path):
    try:
        with open(full_path, 'rb') as reader:
            content = reader.read()
            self.send_content(content)
    except IOError as msg:
```



```
if __name__ == '__main__':
    serverAddress = ('', 8080)
    server = BaseHTTPServer.HTTPServer(serverAddress, RequestHandler)
    server.serve_forever()
```

4. 在根url显示首页内容

大部分时候我们都希望能够直接在127.0.0.1:8080/ 显示主页内容。要怎么做呢，也许我们可以在do_GET那冗长的if-elif-else判断里再加一个判断请求地址是不是根地址的分支，也许我们可以找到一个更加聪明的方法。

比如说把每一种情况都单独写成一个条件类

```
class case_no_file(object):
    '''该路径不存在'''

    def test(self, handler):
        return not os.path.exists(handler.full_path)

    def act(self, handler):
        raise ServerException("{} not found".format(handler.path))

class case_existing_file(object):
    '''该路径是文件'''

    def test(self, handler):
        return os.path.isfile(handler.full_path)

    def act(self, handler):
        handler.handle_file(handler.full_path)

class case_always_fail(object):
    '''所有情况都不符合时的默认处理类'''

    def test(self, handler):
        return True

    def act(self, handler):
        raise ServerException("Unknown object {}".format(handler.path))
```



将原先的if-elif-else分支替换成遍历所有的条件类来看一下区别。

替换前：

```
def do_GET(self):
    try:

        # 文件完整路径
        full_path = os.getcwd() + self.path

        # 如果该路径不存在...
        if not os.path.exists(full_path):
            #抛出异常：文件未找到
            raise ServerException("'{}' not found".format(self.path))

        # 如果该路径是一个文件
        elif os.path.isfile(full_path):
            #调用 handle_file 处理该文件
            self.handle_file(full_path)

        # 如果该路径不是一个文件
        else:
            #抛出异常：该路径为不知名对象
            raise ServerException("Unknown object '{}'.format(self.path))

    # 处理异常
    except Exception as msg:
        self.handle_error(msg)
```

替换后：

```
# 所有可能的情况
Cases = [case_no_file(),
         case_existing_file(),
         case_always_fail()]

def do_GET(self):
    try:

        # 文件完整路径
        full_path = os.getcwd() + self.path
```



```
handler = case()
#如果满足该类情况
if handler.test(self):
    #调用相应的act函数
    handler.act(self)
    break

# 处理异常
except Exception as msg:
    self.handle_error(msg)
```

这样每当我们需要考虑一个新的情况时，只要新写一个条件处理类然后加到 Cases 中去就行了，是不是比原先在if-elif-else中添加条件的做法看起来更加干净更加清楚呢，毕竟修改原有的代码是一件很有风险的事情，调试起来也非常麻烦。在做功能扩展的同时尽量不要修改原代码是软件开发过程中需要牢记的一点。

回到正题，我们希望浏览器访问根url的时候能返回工作目录下index.html的内容，那就需要再加一个条件判断啦。

写一个新的条件处理类

```
class case_directory_index_file(object):

    def index_path(self, handler):
        return os.path.join(handler.full_path, 'index.html')

    #判断目标路径是否是目录&&目录下是否有index.html
    def test(self, handler):
        return os.path.isdir(handler.full_path) and \
            os.path.isfile(self.index_path(handler))

    #响应index.html的内容
    def act(self, handler):
        handler.handle_file(self.index_path(handler))
```

加到Cases中

```
Cases = [case_no_file(),
          case_existing_file(),
          case_directory_index_file(),
          case_always_fail()]
```

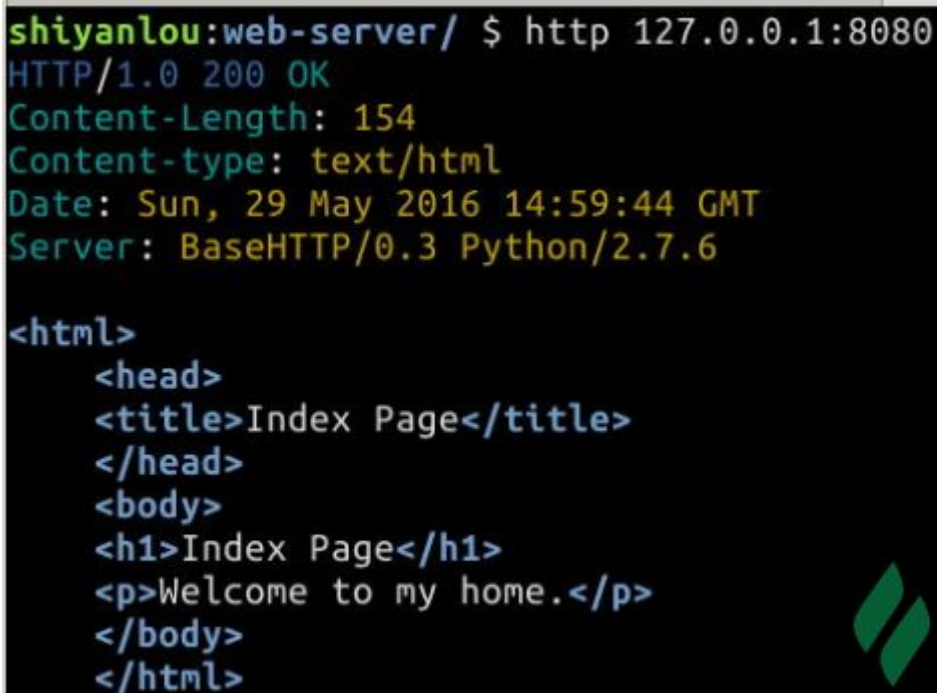


```
<html>
<head>
<title>Index Page</title>
</head>
<body>
<h1>Index Page</h1>
<p>Welcome to my home.</p>
</body>
</html>
```

测试一下

```
$ http 127.0.0.1:8080
```

效果图



5.CGI协议

当然，大部分人都不希望每次给服务器加新功能都要到服务器的源代码里进行修改。如果程序能独立在另一个脚本文件里运行那就再好不过了。什么是CGI? 本小节会实现CGI的效果。

接下来的例子中，我们会在html页面上显示当地时间。

创建新文件 time.py



```
<html>
<body>
<p>Generated {0}</p>
</body>
</html>''.format(datetime.now())
```

新建一个处理脚本文件的条件类:

```
class case_cgi_file(object):
    '''脚本文件处理'''

    def test(self, handler):
        return os.path.isfile(handler.full_path) and \
            handler.full_path.endswith('.py')

    def act(self, handler):
        ##运行脚本文件
        handler.run_cgi(handler.full_path)
```

实现运行脚本文件的函数

```
import subprocess

def run_cgi(self, full_path):
    data = subprocess.check_output(["python", handler.full_path])
    self.send_content(data)
```

不要忘了加到Cases中去

```
Cases = [case_no_file(),
          case_cgi_file(),
          case_existing_file(),
          case_directory_index_file(),
          case_always_fail()]
```

查看效果

```
$ http 127.0.0.1:8080/time.py
```



```
Content-type: text/html
Date: Tue, 31 May 2016 02:34:12 GMT
Server: BaseHTTP/0.3 Python/2.7.6

<html>
<body>
<p>Generated 2016-05-31 10:34:12.474388</p>
</body>
</html>
```



6.代码重构

回头看看我们的代码，注意到一个新的问题了吗？虽然条件判断已经被我们整理到几个类中去了，但是像run_cgi只有在路径为py文件的条件下才使用的函数是放在 RequestHandler下的，那以后再加几个新功能，但是这类函数都放到 RequestHandler下的话可想而知RequestHandler 会变的臃肿不堪。当然你会想这算什么问题嘛，把它放到各自的条件类下不就好了噢。

各自的代码归各自是个好办法，但有时候不同的条件类内可能会有功能相同的函数，这时候我们都知知道重复相同的代码是软件开发里很忌讳的一件事情，那么怎么处理重复的代码呢？

可以抽象出一个基类嘛，遇到重复的内容就放在基类的下面，所有的条件类都继承这个基类。

```
class base_case(object):
    '''条件处理基类'''

    def handle_file(self, handler, full_path):
        try:
            with open(full_path, 'rb') as reader:
                content = reader.read()
                handler.send_content(content)
        except IOError as msg:
            msg = "'{0}' cannot be read: {1}".format(full_path, msg)
            handler.handle_error(msg)

    def index_path(self, handler):
        return os.path.join(handler.full_path, 'index.html')
#要求子类必须实现该接口
    def test(self, handler):
        assert False, 'Not implemented.'

    def act(self, handler):
        assert False, 'Not implemented.'
```



```
class case_directory_index_file(base_case):

    def test(self, handler):
        return os.path.isdir(handler.full_path) and \
            os.path.isfile(self.index_path(handler))

    def act(self, handler):
        self.handle_file(handler, self.index_path(handler))
```

重构后的全部代码

```
#!/usr/bin/env python
#-*- coding:utf-8 -*-
import sys, os, BaseHTTPServer, subprocess

#-----

class ServerException(Exception):
    '''服务器内部错误'''
    pass

#-----

class base_case(object):
    '''条件处理基类'''

    def handle_file(self, handler, full_path):
        try:
            with open(full_path, 'rb') as reader:
                content = reader.read()
            handler.send_content(content)
        except IOError as msg:
            msg = "'{0}' cannot be read: {1}".format(full_path, msg)
            handler.handle_error(msg)

    def index_path(self, handler):
        return os.path.join(handler.full_path, 'index.html')

    def test(self, handler):
        assert False, 'Not implemented.'

    def act(self, handler):
        assert False, 'Not implemented.'
```




```
class case_no_file(base_case):
    '''文件或目录不存在'''

    def test(self, handler):
        return not os.path.exists(handler.full_path)

    def act(self, handler):
        raise ServerException("{}' not found".format(handler.path))
```

#-----

```
class case_cgi_file(base_case):
    '''可执行脚本'''

    def run_cgi(self, handler):
        data = subprocess.check_output(["python", handler.full_path])
        handler.send_content(data)

    def test(self, handler):
        return os.path.isfile(handler.full_path) and \
            handler.full_path.endswith('.py')

    def act(self, handler):
        self.run_cgi(handler)
```

#-----

```
class case_existing_file(base_case):
    '''文件存在的情况'''

    def test(self, handler):
        return os.path.isfile(handler.full_path)

    def act(self, handler):
        self.handle_file(handler, handler.full_path)
```

#-----

```
class case_directory_index_file(base_case):
    '''在根路径下返回主页文件'''

    def test(self, handler):
```



```
def act(self, handler):
    self.handle_file(handler, self.index_path(handler))

#-----

class case_always_fail(base_case):
    '''默认处理'''

    def test(self, handler):
        return True

    def act(self, handler):
        raise ServerException("Unknown object '{0}'".format(handler.path))

#-----

class RequestHandler(BaseHTTPServer.BaseHTTPRequestHandler):
    ...
    请求路径合法则返回相应处理
    否则返回错误页面
    ...

    Cases = [case_no_file(),
              case_cgi_file(),
              case_existing_file(),
              case_directory_index_file(),
              case_always_fail()]

    # 错误页面模板
    Error_Page = """\
    <html>
    <body>
    <h1>Error accessing {path}</h1>
    <p>{msg}</p>
    </body>
    </html>
    """

    def do_GET(self):
        try:

            # 得到完整的请求路径
```



```
# 遍历所有的情况并处理
for case in self.Cases:
    if case.test(self):
        case.act(self)
        break

# 处理异常
except Exception as msg:
    self.handle_error(msg)

def handle_error(self, msg):
    content = self.Error_Page.format(path=self.path, msg=msg)
    self.send_content(content, 404)

# 发送数据到客户端
def send_content(self, content, status=200):
    self.send_response(status)
    self.send_header("Content-type", "text/html")
    self.send_header("Content-Length", str(len(content)))
    self.end_headers()
    self.wfile.write(content)

#-----

if __name__ == '__main__':
    serverAddress = ('', 8080)
    server = BaseHTTPServer.HTTPServer(serverAddress, RequestHandler)
    server.serve_forever()
```

五、讨论

通过重构我们发现，真正实施行为(Action)的代码逻辑可以抽出来进行封装(封装成各种条件处理类)，而 RequestHandler类 或是 basecase类 提供了供条件处理类使用的接口，它们可以看作是一系列服务(Service)，在软件设计中我们常常会把业务代码进行分层，将行为与服务分开，降低耦合，更有利于我们开发维护代码。

通过统一接口，以及cgi程序，我们的代码功能扩展变的更加容易，可以专心于编写功能代码，而不用去关心其他部分。case 的添加虽然仍在server代码中，但我们也可以把它放到配置文件中，由server读取配置文件。

我们的 server 现在还是一个在新手村里打史莱姆的小菜鸡，你会给它添加什么功能让它成长广
么样子呢？



- [A Simple Web Server](#)
- [500 Lines or Less](#)
- [Introduction to HTTP](#)
- [HTTP 下午茶](#)

本课程可以在[实验楼](#)中在线完成，立即[【开始实验】](#)。

更多Python经典项目：[Python项目课](#)

注：转载需获授权，并保留课程地址：shianlou.com/courses/5...



实验楼
shianlou.com

专业的IT在线实训平台，
每天一个项目实战课程！



订阅微信：[实验楼（微信号：shianlou-com）](#)



关注微博：[@实验楼官方微博](#)



加入官方QQ群：[450412940](#)

编辑于 2016-06-15

[Python](#)

[Python 开发](#)

[Python教程](#)

文章被以下专栏收录

推荐阅读

跟黄哥学习Python系列文章。

跟黄哥学习Python第一章

<https://zhuanlan.zhihu.com/p/21>

跟黄哥学习python第二章

<https://zhuanlan.zhihu.com/p/21>

跟黄哥学习python第三章

<https://zhuanlan.zhihu.com/p/21>

黄哥

发表于通过pyt...



【Python】：初识while循环

Charles鱼鱼

6 条评论

⇌ 切换为时间排序

写下你的评论...



xmcp

2 年前

很好奇为什么不用flask之类的库，代码不知比BaseHTTPServer简洁多少啊

👍 赞



布啦豆

2 年前

哈哈，大boss在这哈

👍 赞



佐佐

2 年前

不错啊

👍 赞



Amnesian

1 年前

basehttpserver是不是不支持html5的canvas，用basehttprequest 显示不了本地html中的canvas

👍 1



李金金

1 年前



1. run_cgi实现错误,

```
def run_cgi(self, full_path):  
    data = subprocess.check_output(["python", handler.full_path])
```

2. full_path前少了self;

3. case居然还实例化;

文件完整路径

```
full_path = os.getcwd() + self.path
```

#遍历所有可能的情况

```
for case in self.Cases:
```

```
    handler = case()
```

#如果满足该类情况

```
    if handler.test(self):
```

#调用相应的act函数

```
    handler.act(self) break
```

👍 2



Lane 回复 李金金

4 个月前

谢谢补充。

👍 赞

💬 查看对话

