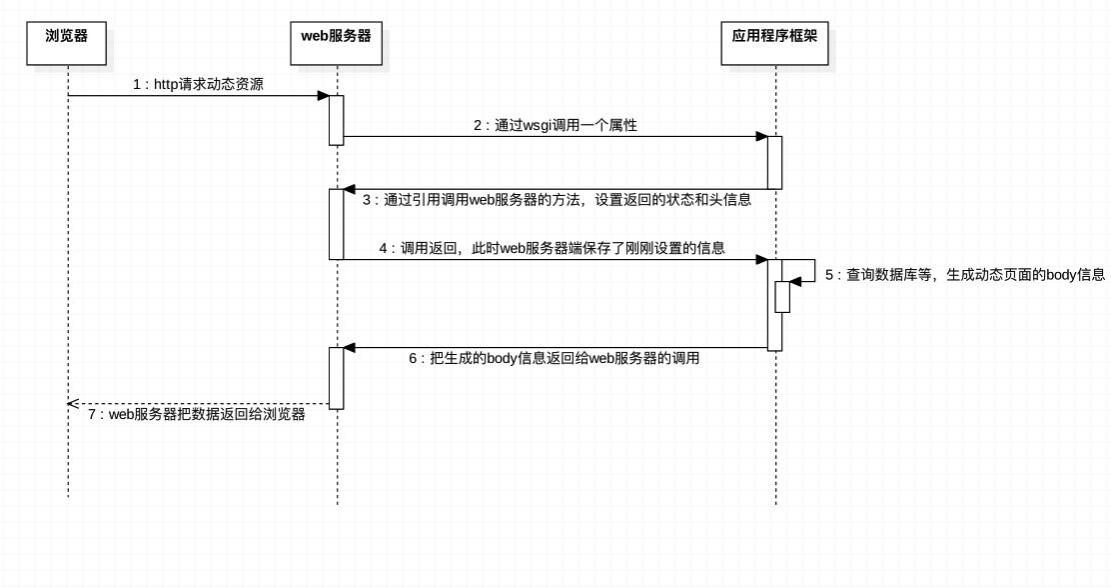
18.1 服务器动态资源请求

1. 浏览器请求动态页面过程

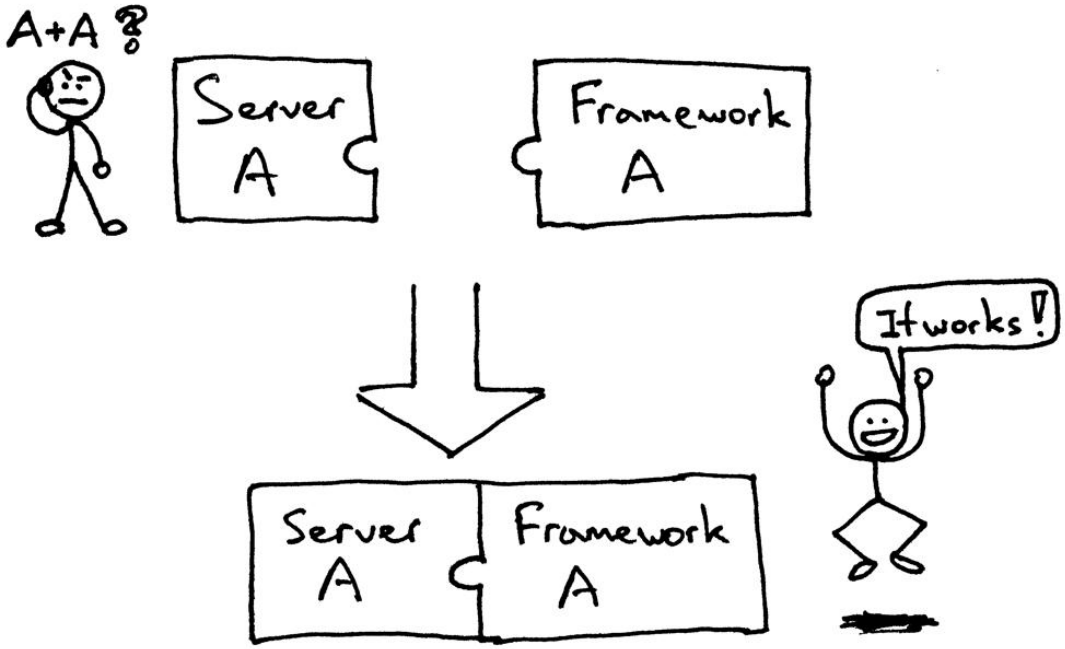


https://zhuanlan.zhihu.com/p/95942024

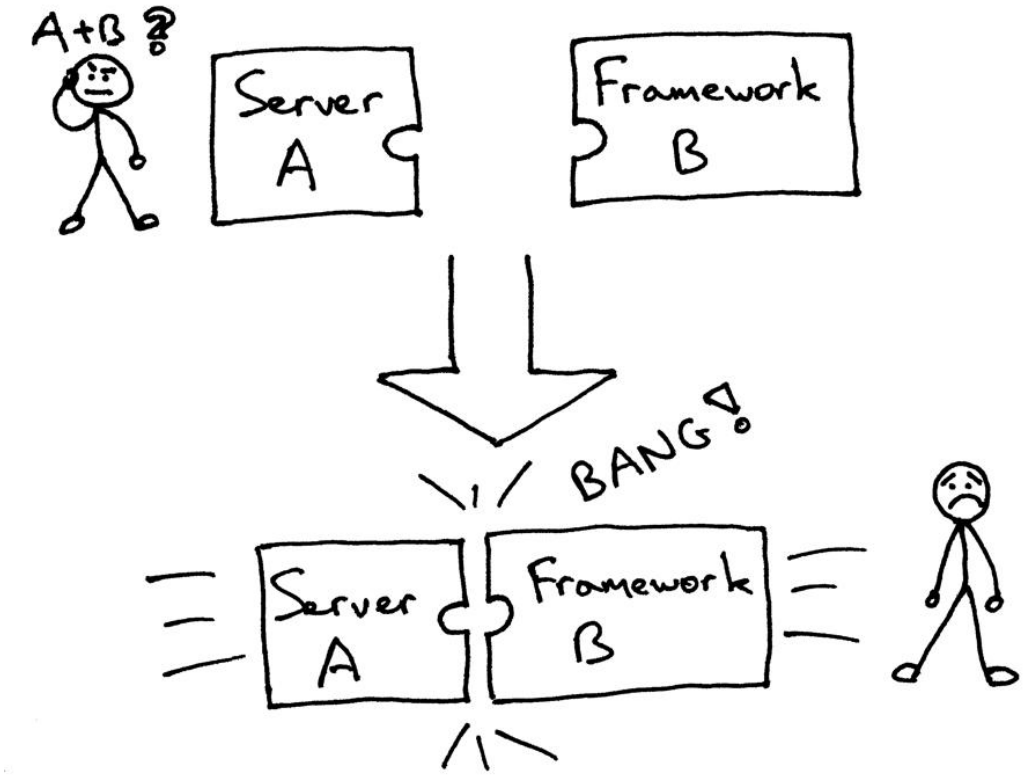
2. WSGI

怎么在你刚建立的 Web 服务器上运行一个 `Django` 应用和 `Flask` 应用，如何不做任何改变而适应不同的 web 架构呢？

在以前，选择 `Python web` 架构会受制于可用的 `web` 服务器，反之亦然。如果架构和服务器可以协同工作，那就好了：

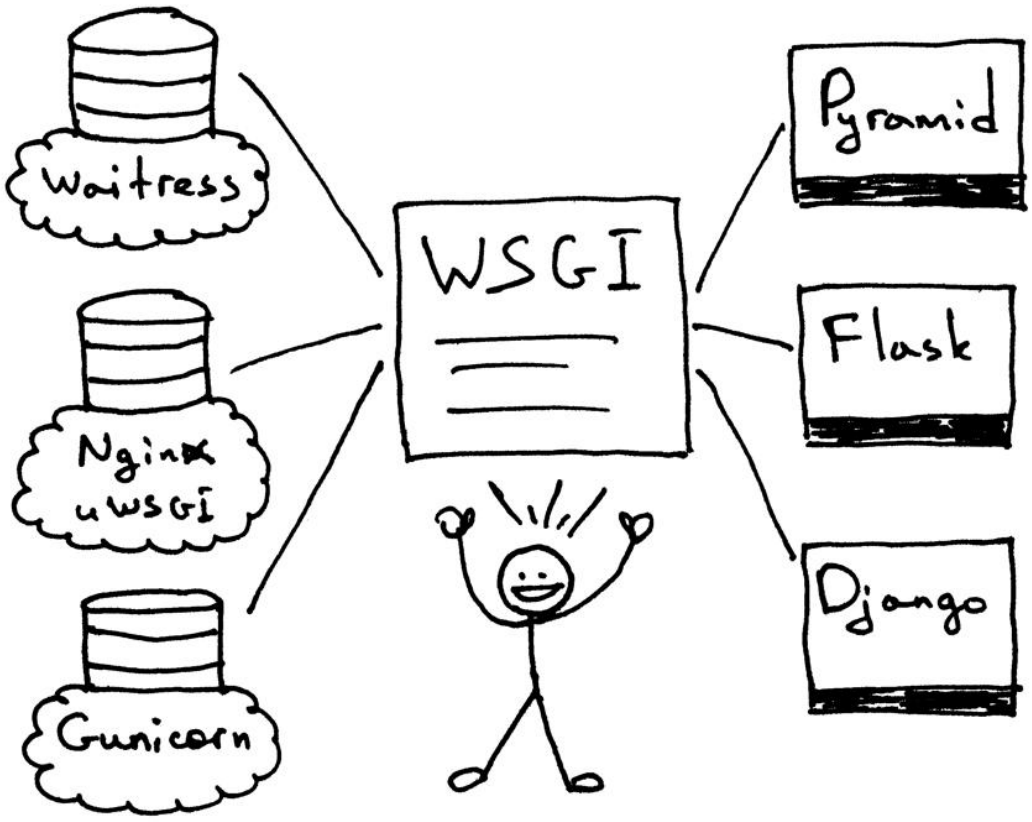


但有可能面对（或者曾有过）下面的问题，当要把一个服务器和一个架构结合起来时，却发现他们不是被设计成协同工作的：



那么，怎么可以不修改服务器和架构代码而确保可以在多个架构下运行 web 服务器呢？答案就是 Python Web Server Gateway Interface (或简称 WSGI，读作“wizgy”)。

WSGI 允许开发者将选择 web 框架和 web 服务器分开。可以混合匹配 web 服务器和 web 框架，选择一个适合的配对。比如,可以在 Gunicorn 或者 Nginx/uWSGI 或者 Waitress 上运行 Django, Flask, 或 Pyramid。真正的混合匹配，得益于 WSGI 同时支持服务器和架构：



web 服务器必须具备 WSGI 接口，所有的现代 Python Web 框架都已具备 WSGI 接口，它让你不对代码作修改就能使服务器和特点的 web 框架协同工作。

WSGI 由 web 服务器支持，而 web 框架允许你选择适合自己的配对，但它同样对于服务器和框架开发者提供便利使他们可以专注于自己偏爱的领域和专长而不至于相互牵制。其他语言也有类似接口：java 有 Servlet API，Ruby 有 Rack。

3.定义 WSGI 接口

WSGI 接口定义非常简单，它只要求 Web 开发者实现一个函数，就可以响应 HTTP 请求。我们来看一个最简单的 Web 版本的“Hello World!”：

```
def application(environ, start_response):
    start_response('200 OK', [('Content-Type', 'text/html')])
    return 'Hello World!'
```

上面的 application()函数就是符合 WSGI 标准的一个 HTTP 处理函数，它接收两个参数：

- environ：一个包含所有 HTTP 请求信息的 dict 对象；
- start_response：一个发送 HTTP 响应的函数。

整个 application()函数本身没有涉及到任何解析 HTTP 的部分，也就是说，把底层 web 服务器解析部分和应用程序逻辑部分进行了分离，这样开发者就可以专心做一个领域了

不过，等等，这个 application()函数怎么调用？如果我们自己调用，两个参数 environ 和 start_response 我们没法提供，返回的 str 也没法发给浏览器。

所以 application()函数必须由 WSGI 服务器来调用。有很多符合 WSGI 规范的服务器。而我们此时的 web 服务器项目的目的就是做一个既能解析静态网页还可以解析动态网页的服务器

4. web 服务器-----WSGI 协议---->web 框架 传递的字典

```
{
    'HTTP_ACCEPT_LANGUAGE': 'zh-cn',
    'wsgi.file_wrapper': <built-infunctionuwsgi_sendfile>,
    'HTTP_UPGRADE_INSECURE_REQUESTS': '1',
    'uwsgi.version': b'2.0.15',
    'REMOTE_ADDR': '172.16.7.1',
    'wsgi.errors': <_io.TextIOWrappername=2mode='w'encoding='UTF-8'>,
    'wsgi.version': (1,0),
    'REMOTE_PORT': '40432',
    'REQUEST_URI': '/',
    'SERVER_PORT': '8000',
    'wsgi.multithread': False,
    'HTTP_ACCEPT': 'text/html,application/xhtml+xml,application/xml;q=0.
9,*/*;q=0.8',
    'HTTP_HOST': '172.16.7.152: 8000',
    'wsgi.run_once': False,
    'wsgi.input': <uwsgi._Inputobjectat0x7f7faecdc9c0>,
    'SERVER_PROTOCOL': 'HTTP/1.1',
    'REQUEST_METHOD': 'GET',
    'HTTP_ACCEPT_ENCODING': 'gzip,deflate',
    'HTTP_CONNECTION': 'keep-alive',
    'uwsgi.node': b'ubuntu',
    'HTTP_DNT': '1',
    'UWSGI_ROUTER': 'http',
    'SCRIPT_NAME': '',
```

```
    'wsgi.multiprocess': False,
    'QUERY_STRING': '',
    'PATH_INFO': '/index.html',
    'wsgi.url_scheme': 'http',
    'HTTP_USER_AGENT': 'Mozilla/5.0(Macintosh;IntelMacOSX10_12_5)AppleW
ebKit/603.2.4(KHTML,likeGecko)Version/10.1.1Safari/603.2.4',
    'SERVER_NAME': 'ubuntu'
}
```

18.2 应用程序示例

```
import time

def application(environ, start_response):
    status = '200 OK'
    response_headers = [('Content-Type', 'text/html')]
    start_response(status, response_headers)
    return str(environ) + '==Hello world from a simple WSGI application!
--->%s\n' % time.ctime()
```

18.2 静态服务器回顾与改造

18.2.1 静态服务器

web_server.py

```
import socket

import re

import multiprocessing



class WSGIServer(object):

    def __init__(self):

        # 1. 创建套接字

        self.tcp_server_socket = socket.socket(socket.AF_INET,
socket.SOCK_STREAM)

        self.tcp_server_socket.setsockopt(socket.SOL_SOCKET,
socket.SO_REUSEADDR, 1)



        # 2. 绑定

        self.tcp_server_socket.bind(("", 7890))



        # 3. 变为监听套接字

        self.tcp_server_socket.listen(128)



    def service_client(self, new_socket):

        """为这个客户端返回数据"""
```

```
# 1. 接收浏览器发送过来的请求， 即 http 请求
# GET / HTTP/1.1
# .....
request = new_socket.recv(1024).decode("utf-8")
# print(">>>"*50)
# print(request)

request_lines = request.splitlines()
print("")
print(">"*20)
print(request_lines)

# GET /index.html HTTP/1.1
# get post put del
file_name = ""
ret = re.match(r"[^/]+(/[^ ]*)", request_lines[0])
if ret:
    file_name = ret.group(1)
    # print("*"*50, file_name)
    if file_name == "/":
        file_name = "/index.html"



# 2. 返回 http 格式的数据，给浏览器
```

```
try:
    f = open("./html" + file_name, "rb")
except:
    response = "HTTP/1.1 404 NOT FOUND\r\n"
    response += "\r\n"
    response += "------file not found-----"
    new_socket.send(response.encode("utf-8"))
else:
    html_content = f.read()
    f.close()
    # 2.1 准备发送给浏览器的数据---header
    response = "HTTP/1.1 200 OK\r\n"
    response += "\r\n"
    # 2.2 准备发送给浏览器的数据---boy
    # response += "hahahhah"

    # 将 response header 发送给浏览器
    new_socket.send(response.encode("utf-8"))
    # 将 response body 发送给浏览器
    new_socket.send(html_content)


# 关闭套接
new_socket.close()
```

```
        def run_forever(self):
            """用来完成整体的控制"""

            while True:
                # 4. 等待新客户端的链接
                new_socket, client_addr = self.tcp_server_socket.accept()

                # 5. 为这个客户端服务
                p = multiprocessing.Process(target=self.service_client,
args=(new_socket,))
                p.start()

                new_socket.close()


            # 关闭监听套接字
            self.tcp_server_socket.close()


    def main():
        """控制整体，创建一个 web 服务器对象，然后调用这个对象的
run_forever 方法运行"""
        wsgi_server = WSGIServer()
        wsgi_server.run_forever()


    if __name__ == "__main__":
        main()
```

18.2.2 解析动态请求

如果请求是不是 html，是动态请求，后缀是.py，那如何处理呢？

```
if not file_name.endswith(".py"):
    静态请求处理
else:
    # 2.2 如果是以.py 结尾，那么就认为是动态资源的请求
    header = "HTTP/1.1 200 OK\r\n"
    header += "\r\n"

    body = "hahahah %s " % time.ctime()

    response = header+body
    # 发送 response 给浏览器
    new_socket.send(response.encode("utf-8"))
```

18.2.3 web 服务器和逻辑处理代码分离

如果请求的是不同名字的 py，怎么办？

```
        header = "HTTP/1.1 200 OK\r\n"
        header += "\r\n"

        # body = "hahahah %s " % time.ctime()
        # if file_name == "/login.py":
        #     body = mini_frame.login()
        # elif file_name == "/register.py":
        #     body = mini_frame.register()
        body = mini_frame.application(file_name)

        response = header+body
```

```
            # 发送 response 给浏览器
            new_socket.send(response.encode("utf-8"))
```

mini_frame.py

```
    import time


    def login():
      return "i----login--welcome wangdao website.....time:%s" % time.ctime()


    def register():
      return "-----register---welcome wangdao website.....time:%s" %
time.ctime()


    def profile():
      return "-----profile----welcome wangdao website.....time:%s" %
time.ctime()


    def application(file_name):
        if file_name == "/login.py":
            return login()
        elif file_name == "/register.py":
            return register()
        else:
            return "not found you page...."
```

18.3 Web 动态服务器-基本实现

18.3.1 Web 服务器支持 WSGI

针对 18.2.3 中的改造，如果 web 服务器要给我们的 mini_frame 框架传参，同时上面我们改造的仅仅是 body，如果需要修改 header 怎么办？

针对 header 及 body 部分代码如下：

```
else:
    # 2.2 如果是以.py 结尾，那么就认为是动态资源的请求


    env = dict()
    body = mini_frame.application(env, self.set_response_header)


    header = "HTTP/1.1 %s\r\n" % self.status


    for temp in self.headers:
        header += "%s:%s\r\n" % (temp[0], temp[1])


    header += "\r\n"


    response = header+body
    # 发送 response 给浏览器
    new_socket.send(response.encode("utf-8"))
```

set_response_header 方法代码如下，是我们传递给 mini_frame.application 的一个行为，env 是我们传递的一个字典，env 暂时还未用到

```
def set_response_header(self, status, headers):
    self.status = status
    self.headers = [("server", "mini_web v8.8")]
    self.headers += headers
```

mini_frame.py

```
def application(environ, start_response):
    start_response('200 OK', [('Content-Type', 'text/html;charset=utf-8')])
    return 'Hello World! 我爱你中国....'
```

start_response 的第二个参数必须是一个列表，列表里边可以有多个元组，每个元组有两个部分，第一部分是冒号前面的内容，第二部分是冒号后面的内容

18.3.2 通过传递字典实现请求不一样的资源

使用 application 函数后，如何实现用户请求的 py 名字不同，显示不同的内容呢？首先 web_server 修改如下：

```
else:
    # 2.2 如果是以.py 结尾，那么就认为是动态资源的请求

    env = dict()  # 这个字典中存放的是 web 服务器要传递给 web 框架的数据信息
    env['PATH_INFO'] = file_name
    # {"PATH_INFO": "/index.py"}
    body = mini_frame.application(env, self.set_response_header)

    header = "HTTP/1.1 %s\r\n" % self.status

    for temp in self.headers:
        header += "%s:%s\r\n" % (temp[0], temp[1])

    header += "\r\n"

    response = header+body
    # 发送 response 给浏览器
    new_socket.send(response.encode("utf-8"))
```

然后 mini_frame.py 如下：

```
def index():
    return "这是主页"

def login():
    return "这是登录页面"

def application(env, start_response):
    start_response('200 OK', [('Content-Type', 'text/html;charset=utf-8')])

    file_name = env['PATH_INFO']
    # file_name = "/index.py"

    if file_name == "/index.py":
    return index()
    elif file_name == "/login.py":
    return login()
    else:
    return 'Hello World! 我爱你中国....'
```

18.3.3 让服务器指定端口和框架

文件结构

```
├── web_server.py
├── web
│   └── mini_frame.py
└── html
    └── index.html
    .....
```

web/mini_frame.py

```
import time

def application(environ, start_response):
    status = '200 OK'
    response_headers = [('Content-Type', 'text/html')]
    start_response(status, response_headers)
    return str(environ) + '==Hello world from a simple WSGI application!
--->%s\n' % time.ctime()
```

web_server.py

```
import select
import time
import socket
import sys
import re
import multiprocessing


class WSGIServer(object):
    """定义一个 WSGI 服务器的类"""

    def __init__(self, port, documents_root, app):

        # 1. 创建套接字
        self.server_socket = socket.socket(socket.AF_INET, socket.SOCK_
STREAM)
        # 2. 绑定本地信息
        self.server_socket.setsockopt(socket.SOL_SOCKET, socket.SO_REUS
EADDR, 1)
        self.server_socket.bind(("", port))
        # 3. 变为监听套接字
        self.server_socket.listen(128)

        # 设定资源文件的路径
        self.documents_root = documents_root

        # 设定 web 框架可以调用的函数(对象)
        self.app = app

    def run_forever(self):
        """运行服务器"""

        # 等待对方链接
        while True:
            new_socket, new_addr = self.server_socket.accept()
            # 创建一个新的进程来完成这个客户端的请求任务
```

```
            new_socket.settimeout(3)  # 3s
            new_process = multiprocessing.Process(target=self.deal_with
_request, args=(new_socket,))
            new_process.start()
            new_socket.close()

    def deal_with_request(self, client_socket):
        """以长链接的方式，为这个浏览器服务器"""

        while True:
            try:
                request = client_socket.recv(1024).decode("utf-8")
            except Exception as ret:
                print("========>", ret)
                client_socket.close()
                return

            # 判断浏览器是否关闭
            if not request:
                client_socket.close()
                return

            request_lines = request.splitlines()
            for i, line in enumerate(request_lines):
                print(i, line)

            # 提取请求的文件(index.html)
            # GET /a/b/c/d/e/index.html HTTP/1.1
            ret = re.match(r"([^/]*)([^ ]+)", request_lines[0])
            if ret:
                print("正则提取数据:", ret.group(1))
                print("正则提取数据:", ret.group(2))
                file_name = ret.group(2)
                if file_name == "/":
                    file_name = "/index.html"

            # 如果不是以 py 结尾的文件，认为是普通的文件
            if not file_name.endswith(".py"):

                # 读取文件数据
                try:
                    f = open(self.documents_root+file_name, "rb")
                except:
                    response_body = "file not found, 请输入正确的 url"

                    response_header = "HTTP/1.1 404 not found\r\n"
                    response_header += "Content-Type: text/html; charse
```

```
t=utf-8\r\n"
                    response_header += "Content-Length: %d\r\n" % (len
(response_body))
                    response_header += "\r\n"

                    response = response_header + response_body

                    # 将 header 返回给浏览器
                    client_socket.send(response.encode('utf-8'))

                else:
                    content = f.read()
                    f.close()

                    response_body = content

                    response_header = "HTTP/1.1 200 OK\r\n"
                    response_header += "Content-Length: %d\r\n" % (len
(response_body))
                    response_header += "\r\n"

                    # 将 header 返回给浏览器
                    client_socket.send(response_header.encode('utf-8')
+ response_body)

            # 以.py 结尾的文件，就认为是浏览需要动态的页面
            else:
                # 准备一个字典，里面存放需要传递给 web 框架的数据
                env = {}
                # 存 web 返回的数据
                response_body = self.app(env, self.set_response_headers)

                # 合并 header 和 body
                response_header = "HTTP/1.1 {status}\r\n".format(status
=self.headers[0])
                response_header += "Content-Type: text/html; charset=ut
f-8\r\n"
                response_header += "Content-Length: %d\r\n" % len(respo
nse_body)
                for temp_head in self.headers[1]:
                    response_header += "{0}:{1}\r\n".format(*temp_head)

                response = response_header + "\r\n"
                response += response_body

                client_socket.send(response.encode('utf-8'))
```

```
    def set_response_headers(self, status, headers):
        """这个方法，会在 web 框架中被默认调用"""
        response_header_default = [
            ("Data", time.ctime()),
            ("Server", "wangdao-python mini web server")
        ]

        # 将状态码/相应头信息存储起来
        # [字符串, [xxxxx, xxx2]]
        self.headers = [status, response_header_default + headers]


# 设置静态资源访问的路径
g_static_document_root = "./html"
# 设置动态资源访问的路径
g_dynamic_document_root = "./web"

def main():
    """控制 web 服务器整体"""
    # python3 xxxx.py 7890
    if len(sys.argv) == 3:
        # 获取 web 服务器的 port
        port = sys.argv[1]
        if port.isdigit():
            port = int(port)
        # 获取 web 服务器需要动态资源时，访问的 web 框架名字
        web_frame_module_app_name = sys.argv[2]
    else:
        print("运行方式如: python3 xxx.py 7890 my_web_frame_name:applica
tion")
        return

    print("http 服务器使用的 port:%s" % port)

    # 将动态路径即存放 py 文件的路径，添加到 path 中，这样 python 就能够找到这
个路径了
    sys.path.append(g_dynamic_document_root)

    ret = re.match(r"([^:]*):(.*)", web_frame_module_app_name)
    if ret:
        # 获取模块名
        web_frame_module_name = ret.group(1)
        # 获取可以调用 web 框架的应用名称
        app_name = ret.group(2)

    # 导入 web 框架的主模块
```

```
    web_frame_module = __import__(web_frame_module_name)
    # 获取那个可以直接调用的函数(对象)
    app = getattr(web_frame_module, app_name)

    # print(app)  # for test

    # 启动 http 服务器
    http_server = WSGIServer(port, g_static_document_root, app)
    # 运行 http 服务器
    http_server.run_forever()


if __name__ == "__main__":
    main()
```

运行

1. 打开终端，输入以下命令开始服务器

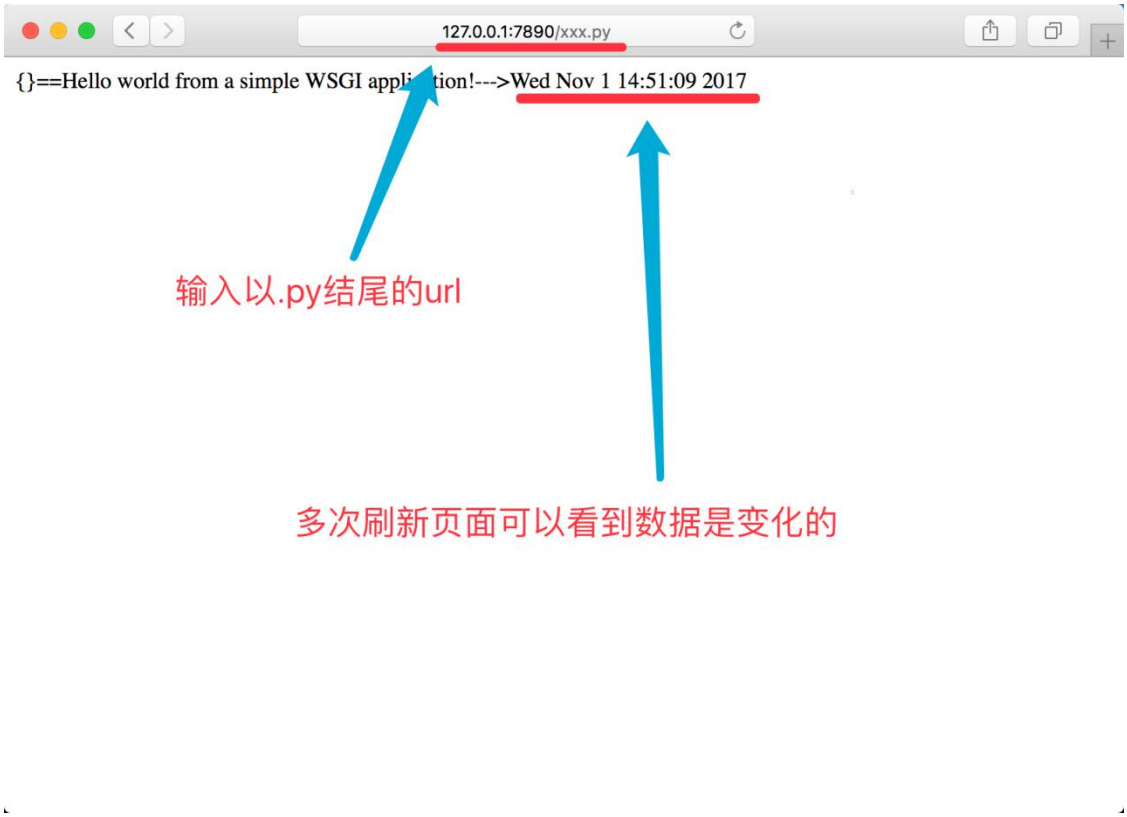
```
python3 web_server.py mini_frame:application
```

2.Pycharm 设置传参的方法

Run/Debug Configurations->Configurations->Script Parames

和 vs 类似，都不用输入程序名字，直接输入参数即可。

3. 打开浏览器


127.0.0.1:7890/xxx.py
{}==Hello world from a simple WSGI application!--->Wed Nov 1 14:51:09 2017
输入以.py结尾的url
多次刷新页面可以看到数据是变化的

18.4 mini web 框架-1-文件结构

文件结构

```
├── dynamic ---存放 py 模块
│   └── mini_frame.py
├── templates ---存放模板文件
│   ├── center.html
│   ├── index.html
│   ├── location.html
│   └── update.html
├── static ---存放静态的资源文件
│   ├── css
│   │   ├── bootstrap.min.css
│   │   ├── main.css
│   │   └── swiper.min.css
│   └── js
│       ├── a.js
│       ├── bootstrap.min.js
│       ├── jquery-1.12.4.js
│       ├── jquery-1.12.4.min.js
│       ├── jquery.animate-colors.js
│       ├── jquery.animate-colors-min.js
│       ├── jquery.cookie.js
│       ├── jquery-ui.min.js
│       ├── server.js
│       ├── swiper.jquery.min.js
│       ├── swiper.min.js
│       └── zepto.min.js
└── web_server.py ---mini web 服务器
```

mini_frame.py

```
def index():
    with open("./templates/index.html",encoding="utf-8") as f:
        content = f.read()
    return content

def center():
    with open("./templates/center.html",encoding="utf-8") as f:
        return f.read()
```

```
def application(env, start_response):
    start_response('200 OK', [('Content-Type',
'text/html;charset=utf-8')])

    file_name = env['PATH_INFO']
    # file_name = "/index.py"

    if file_name == "/index.py":
        return index()
    elif file_name == "/center.py":
        return center()
    else:
        return 'Hello World! 我爱你中国....'
```

web_server.py

```
import select
import time
import socket
import sys
import re
import multiprocessing


class WSGIServer(object):
    """定义一个 WSGI 服务器的类"""

    def __init__(self, port, documents_root, app):

        # 1. 创建套接字
        self.server_socket = socket.socket(socket.AF_INET,
socket.SOCK_STREAM)
        # 2. 绑定本地信息
        self.server_socket.setsockopt(socket.SOL_SOCKET,
socket.SO_REUSEADDR, 1)
        self.server_socket.bind(("", port))
        # 3. 变为监听套接字
```

```
        self.server_socket.listen(128)

        # 设定资源文件的路径
        self.documents_root = documents_root

        # 设定 web 框架可以调用的函数(对象)
        self.app = app

    def run_forever(self):
        """运行服务器"""

        # 等待对方链接
        while True:
            new_socket, new_addr = self.server_socket.accept()
            # 创建一个新的进程来完成这个客户端的请求任务
            new_socket.settimeout(3)  # 3s
            new_process =
multiprocessing.Process(target=self.deal_with_request,
args=(new_socket,))
            new_process.start()
            new_socket.close()

    def deal_with_request(self, client_socket):
        """以长链接的方式，为这个浏览器服务器"""

        while True:
            try:
                request = client_socket.recv(1024).decode("utf-
8")
            except Exception as ret:
                print("========>", ret)
                client_socket.close()
                return

            # 判断浏览器是否关闭
            if not request:
```

```
                client_socket.close()
                return

            request_lines = request.splitlines()
            for i, line in enumerate(request_lines):
                print(i, line)

            # 提取请求的文件(index.html)
            # GET /a/b/c/d/e/index.html HTTP/1.1
            ret = re.match(r"([^/]*)([^ ]+)", request_lines[0])
            if ret:
                print("正则提取数据:", ret.group(1))
                print("正则提取数据:", ret.group(2))
                file_name = ret.group(2)
                if file_name == "/":
                    file_name = "/index.html"

            # 如果不是以 py 结尾的文件，认为是普通的文件
            if not file_name.endswith(".py"):

                # 读取文件数据
                try:
                    f = open(self.documents_root+file_name,
"rb")
                except:
                    response_body = "file not found, 请输入正确
的 url"

                    response_header = "HTTP/1.1 404 not
found\r\n"
                    response_header += "Content-Type: text/html;
charset=utf-8\r\n"
                    response_header += "Content-
Length: %d\r\n" % (len(response_body))
                    response_header += "\r\n"
```

```
                    response = response_header + response_body

                    # 将 header 返回给浏览器
                    client_socket.send(response.encode('utf-8'))

                else:
                    content = f.read()
                    f.close()

                    response_body = content

                    response_header = "HTTP/1.1 200 OK\r\n"
                    response_header += "Content-
Length: %d\r\n" % (len(response_body))
                    response_header += "\r\n"

                    # 将 header 返回给浏览器

client_socket.send(response_header.encode('utf-8') +
response_body)

            # 以.py 结尾的文件，就认为是浏览需要动态的页面
            else:
                # 准备一个字典，里面存放需要传递给 web 框架的数据
                env = dict()
                env['PATH_INFO'] = file_name
                # 存 web 返回的数据
                response_body = self.app(env,
self.set_response_headers)

                # 合并 header 和 body
                header = "HTTP/1.1 %s\r\n" % self.status

                for temp in self.headers:
                    header += "%s:%s\r\n" % (temp[0], temp[1])
```

```
                header += "\r\n"

                response = header + response_body

                client_socket.send(response.encode('utf-8'))


    def set_response_headers(self, status, headers):
        """这个方法，会在 web 框架中被默认调用"""
        self.status = status
        self.headers = [
            ("Data", time.time()),
            ("Server", "wangdao-python mini web server")
        ]
        self.headers += headers


# 设置静态资源访问的路径
g_static_document_root = "./static"
# 设置动态资源访问的路径
g_dynamic_document_root = "./dynamic"

def main():
    """控制 web 服务器整体"""
    # python3 xxxx.py 7890
    if len(sys.argv) == 3:
        # 获取 web 服务器的 port
        port = sys.argv[1]
        if port.isdigit():
            port = int(port)
        # 获取 web 服务器需要动态资源时，访问的 web 框架名字
        web_frame_module_app_name = sys.argv[2]
    else:
        print("运行方式如: python3 xxx.py 7890
my_web_frame_name:application")
        return
```

```
    print("http 服务器使用的 port:%s" % port)

    # 将动态路径即存放 py 文件的路径，添加到 path 中，这样 python 就能够找到这个路径了
    sys.path.append(g_dynamic_document_root)

    ret = re.match(r"([^:]*):(.*)", web_frame_module_app_name)
    if ret:
        # 获取模块名
        web_frame_module_name = ret.group(1)
        # 获取可以调用 web 框架的应用名称
        app_name = ret.group(2)

    # 导入 web 框架的主模块
    web_frame_module = __import__(web_frame_module_name)
    # 获取那个可以直接调用的函数(对象)
    app = getattr(web_frame_module, app_name)

    # print(app)  # for test

    # 启动 http 服务器
    http_server = WSGIServer(port, g_static_document_root, app)
    # 运行 http 服务器
    http_server.run_forever()


if __name__ == "__main__":
    main()
```

18.5 mini web 框架-2-显示页面

如何在 index.html 和 center.html 中显示动态的数据呢？比如针对不同用户，从数据库中查询，显示不同的内容呢？通过修改 mini_frame.py 的两个函数接口，替换 html 页面中的{%content%}即可，代码如下：

dynamic/mini_frame.py (更新)

```
import re


def index():
    with open("./templates/index.html",encoding="utf-8") as f:
        content = f.read()

    my_stock_info = "哈哈哈哈 这是你的本月名称....."

    content = re.sub(r"\{%content%\}", my_stock_info, content)

    return content


def center():
    with open("./templates/center.html",encoding="utf-8") as f:
        content = f.read()

    my_stock_info = "这里是从 mysql 查询出来的数据。。。"

    content = re.sub(r"\{%content%\}", my_stock_info, content)

    return content
```

浏览器打开看效果

18.6 增加配置文件

如果目录名字不是 dynamic 和 static，那么我们就需要去 web_server 中，也就是我们的 web 服务器中去修改代码，这样显然是不合适的，所以我们需要增加配置文件，增加如下配置文件：

web_server.conf

```
{
    "static_path":"./static",
    "dynamic_path":"./dynamic"
}
```

然后在 web_server.py 中，修改代码如下

```
with open("./web_server.conf") as f:
    conf_info = eval(f.read())

# 此时 conf_info 是一个字典里面的数据为:
# {
#     "static_path":"./static",
#     "dynamic_path":"./dynamic"
# }


sys.path.append(conf_info['dynamic_path'])

print(conf_info['static_path'])
```