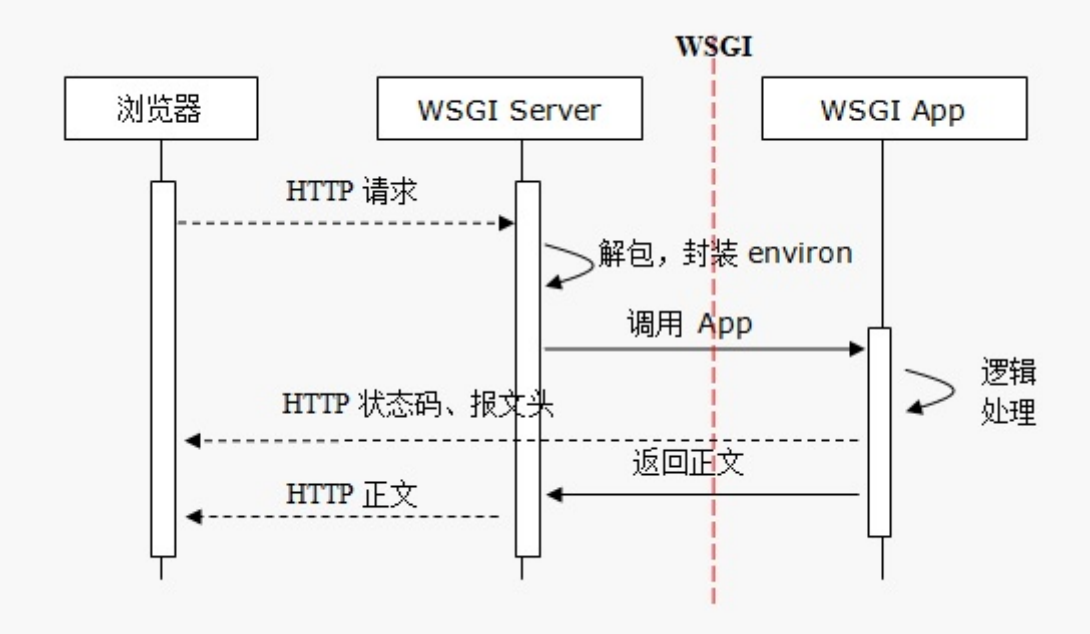
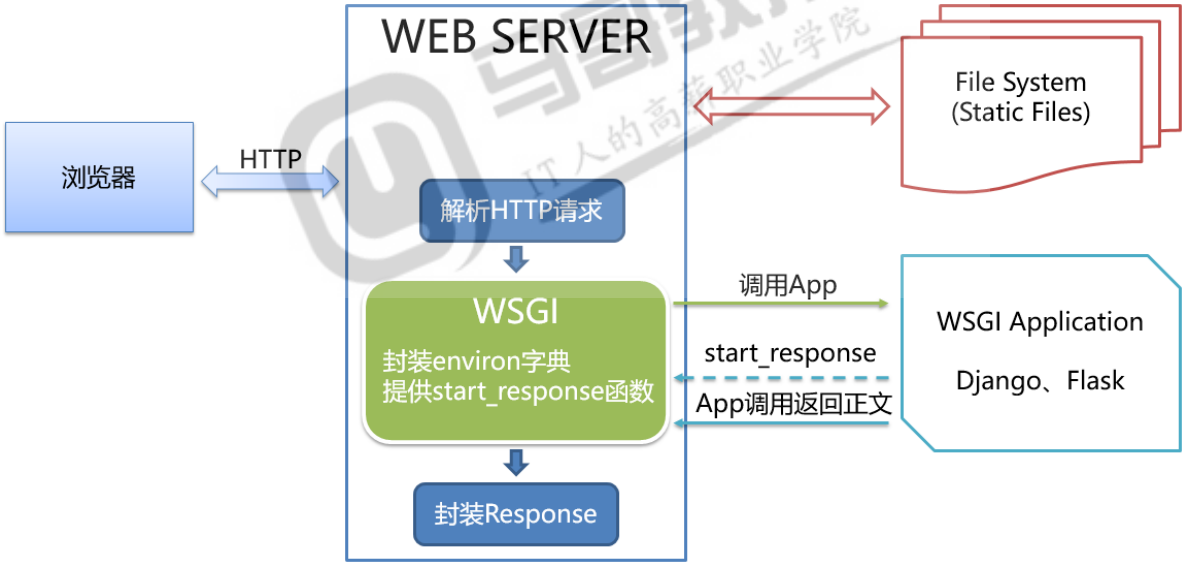
WSGI



WSGI（Web Server Gateway Interface）主要规定了服务器端和应用程序间的接口。

WEB Server主要负责HTTP协议请求和响应，但不一定支持WSGI接口访问。



- environ是简单封装的请求报文的字典
- start_response解决响应报文头的函数
- app函数返回响应报文正文，简单理解就是HTML

原理如下

```
from socketserver import ThreadingTCPServer, BaseRequestHandler
import webob

'''\
HTTP/1.1 200 OK
Content-Type: text/html; charset=UTF-8
Content-Length: 23
```

```
<h2>www.magedu.com</h2>\
'''

class HTTPHandler(BaseRequestHandler):
    def handle(self) -> None:
        # HTTP请求是不是接收到的数据
        environ = {}
        data:str = self.request.recv(1024).decode()
        print(data) # \r\n分割的多行，第一行分为三部分
        print('~' * 30)
        start = data.find('\r\n')
        method, path, protocol = data[:start].split(' ')
        environ['method'] = method
        environ['path'] = path
        environ['protocol'] = protocol
        end = data.find('\r\n\r\n')
        for entry in data[start+2:end].split('\r\n'):
            k, v = entry.split(': ', 1)
            environ[k] = v
        print(environ)
        print('=' * 30)

        html = '<h2>www.magedu.com</h2>' # application

        print('=' * 30)
        response = webob.response.Response(html)
        http_res = []
        http_res.append('HTTP/1.1 ' + response.status)
        http_res.extend(
            "{}: {}".format(k, v) for k, v in response.headerlist
        )
        http_res = "\r\n".join(http_res) + '\r\n\r\n'

        self.request.send(http_res.encode())
        print(http_res + response.body.decode())
        print('-' * 30)
        self.request.send(response.body)


server = ThreadingTCPServer(('127.0.0.1', 8080), HTTPHandler)
server.serve_forever()
```

剩下的问题就是谁来实现application和其内部的路径映射（路由）。

WSGI服务器——wsgiref

wsgiref是Python提供的一个WSGI参考实现库，不适合生产环境使用。

wsgiref.simple_server 模块实现一个简单的WSGI HTTP服务器。

```python
# 启动一个WSGI服务器
wsgiref.simple_server.make_server(host, port, app, server_class=WSGIServer, handler_class=WSGIRequestHandler)

# 一个两参数函数，小巧完整的WSGI的应用程序的实现
wsgiref.simple_server.demo_app(environ, start_response)
```

```python
# 返回文本例子
from wsgiref.simple_server import make_server, demo_app

ip = '127.0.0.1'
port = 9999
server = make_server(ip, port, demo_app) # demo_app应用程序，可调用
server.serve_forever() # server.handle_request() 执行一次
```

WSGI APP应用程序端

1、应用程序应该是一个可调用对象
Python中应该是函数、类、实现了`__call__`方法的类的实例

2、这个可调用对象应该接收两个参数

```python
# 1 函数实现
def application(environ, start_response):
    pass

# 2 类实现
class Application:
    def __init__(self, environ, start_response):
        pass

# 3 类实现
class Application:
    def __call__(self, environ, start_response):
        pass
```

3、以上的可调用对象实现，都必须返回一个可迭代对象

```python
res_str = b'www.magedu.com\n'

# 1 函数实现
def application(environ, start_response):
    start_response("200 OK", [('Content-Type', 'text/plain; charset=utf-8')])
    return [res_str]

# 2 类实现
class Application:
    def __init__(self, environ, start_response):
        self.env = environ
        self.start_response = start_response

    def __iter__(self): # 对象可迭代
```

```
        self.start_response('200 OK', [('Content-Type', 'text/plain;
charset=utf-8')])
        yield res_str
        #yield from [res_str]
        #return iter([res_str])


# 3 类实现，可调用对象
class Application:
    def __call__(self, environ, start_response):
        start_response('200 OK', [('Content-Type', 'text/plain; charset=utf-
8')])
        return [res_str]
```

environ和start_response这两个参数名可以是任何合法名，但是一般默认都是这2个名字。
应用程序端还有些其他的规定，暂不用关心

注意：第2、第3种实现调用时的不同

environ

environ是包含Http请求信息的dict字典对象

名称	含义
REQUEST_METHOD	请求方法，GET、POST等
PATH_INFO	URL中的路径部分
QUERY_STRING	查询字符串
SERVER_NAME, SERVER_PORT	服务器名、端口
HTTP_HOST	地址和端口
SERVER_PROTOCOL	协议
HTTP_USER_AGENT	UserAgent信息

```
CONTENT_TYPE = 'text/plain'
HTTP_HOST = '127.0.0.1:9999'
HTTP_USER_AGENT = 'Mozilla/5.0 (Windows; U; Windows NT 6.1; zh-CN)
AppleWebKit/537.36 (KHTML, like Gecko) Version/5.0.1 Safari/537.36'
PATH_INFO = '/'
QUERY_STRING = ''
REMOTE_ADDR = '127.0.0.1'
REMOTE_HOST = ''
REQUEST_METHOD = 'GET'
SERVER_NAME = 'DESKTOP-D34H5HF'
SERVER_PORT = '9999'
SERVER_PROTOCOL = 'HTTP/1.1'
SERVER_SOFTWARE = 'WSGIServer/0.2'
```

start_response

它是一个可调用对象。有3个参数，定义如下：
`start_response(status, response_headers, exc_info=None)`

参数名称	说明
status	状态码和状态描述，例如`200 OK`
response_headers	一个元素为二元组的列表，例如[('Content-Type', 'text/plain;charset=utf-8')]
exc_info	在错误处理的时候使用
start_response应该在返回可迭代对象之前调用，因为它是Response Header。返回的可迭代对象是Response Body。	

服务器端

服务器程序需要调用符合上述定义的可调用对象APP，传入environ、start_response，APP处理后，返回响应头和可迭代对象的正文，由服务器封装返回浏览器端。

```
# 返回网页的例子
from wsgiref.simple_server import make_server

def application(environ, start_response):
    status = '200 OK'
    headers = [('Content-Type', 'text/html;charset=utf-8')]
    start_response(status, headers)
    # 返回可迭代对象
    html = '<h1>马哥教育欢迎你</h1>'.encode("utf-8")
    return [html]

ip = '127.0.0.1'
port = 9999
server = make_server(ip, port, application)
server.serve_forever() # server.handle_request() 一次
```

simple_server 只是参考用，不能用于生产环境。

测试用命令

```
$ curl -I http://192.168.142.1:9999/xxx?id=5
$ curl -X POST http://192.168.142.1:9999/yyy -d '{"x":2}'
```

-I 使用HEAD方法
-X 指定方法，-d传输数据

到这里就完成了一个简单的WEB 程序开发。

总结

WSGI 服务器作用

1. 监听HTTP服务端口（TCPServer，默认端口80）接收浏览器端的HTTP请求，这是WWW Server的作用
2. 解析请求报文封装成environ环境数据
3. 负责调用应用程序app，将**environ**数据和**start_response**方法两个实参传入给Application
4. 利用app的返回值和start_response返回的值，构造HTTP响应报文
5. 将响应报文返回浏览器端

2、3、4要实现WSGI协议，该协议约定了和应用程序之间接口（参看PEP333，https://www.python.org/dev/peps/pep-0333/）

WSGI APP应用程序

- 遵从WSGI协议
- 本身是一个可调用对象
- 调用start_response，返回响应头部
- 返回包含正文的可迭代对象

Django、Flask都是符合WSGI协议且可以快速开发的框架，但本质上是编写Application，说白了，就是编写一个函数，这个函数签名为app(environ, start_response)，这不过在app函数内部调用非常复杂而已。比如要解决访问数据库、静态HTML页面读取、动态网页生成等。