类Flask框架实现

从现在开始，我们将一步步完成一个WSGI的WEB框架，从而了解WEB框架的内部机制。

WSGI请求environ处理

WSGI服务器程序会帮我们处理HTTP请求报文，但是提供的environ还是一个用起来不方便的字典

```
http://127.0.0.1:9999/python/index.html?id=1234&name=tom
('SERVER_PROTOCOL', 'HTTP/1.1')
('wsgi.url_scheme', 'http')
('HTTP_HOST', '127.0.0.1:9999')
('SERVER_PORT', '9999')
('REMOTE_ADDR', '127.0.0.1')

('REQUEST_METHOD', 'GET')
('CONTENT_TYPE', 'text/plain')
('PATH_INFO', '/python/index.html')
('QUERY_STRING', 'id=1234&name=tom')
('HTTP_USER_AGENT', 'Mozilla/5.0 (Windows NT 6.1) AppleWebKit/537.36 (KHTML, like Gecko)
Maxthon/5.0 Chrome/55.0.2883.75 Safari/537.36')
```

QUERY_STRING 查询字符串的解析

WSGI服务器程序处理过HTTP报文后，返回一个字典，可以得到查询字符串`('QUERY_STRING', 'id=1234&name=tom')`。这个键值对用起来不方便。

1、编程序解析

```
# id=5&name=wayne
qstr = environ.get('QUERY_STRING')
print(qstr)
if qstr:
    for pair in qstr.split('&'):
        k, _, v = pair.partition('=')
        print("k={}, v={}".format(k,v))
```

```
# id=5&name=wayne
querystr = environ.get('QUERY_STRING')
if querystr:
    querydict = {k:v for k,_,v in map(lambda item: item.partition('='), querystr.split('&'))}
print(querydict)
```

2、使用cgi模块

```
# id=5&name=wayne
qstr = environ.get('QUERY_STRING')
print(qstr)
print(parse_qs(qstr))
# {'name': ['wayne'], 'id': ['5']}
```

可以看到使用这个库，可以解析查询字符串，请注意value是列表，为什么？
这是因为同一个key可以有多个值。

cgi模块过期了，建议使用urllib

3、使用urllib库

```
# http://127.0.0.1:9999/?id=5&name=wayne&age=&comment=1,a,c&age=19&age=20
qstr = environ.get('QUERY_STRING')
print(qstr)
print(parse.parse_qs(qstr)) # 字典
print(parse.parse_qsl(qstr)) # 二元组列表

# 运行结果
id=5&name=wayne&age=&comment=1,a,c&age=19&age=20
{'name': ['wayne'], 'age': ['19', '20'], 'id': ['5'], 'comment': ['1,a,c']}
[('id', '5'), ('name', 'wayne'), ('comment', '1,a,c'), ('age', '19'), ('age', '20')]
```

parse_qs函数，将同一个名称的多值，保存在字典中，使用了列表保存。

comment=1,a,c 这不是多值，这是一个值。
age 是多值。

environ的解析——webob库

环境数据有很多，都是存在字典中的，字典的存取方式没有对象的属性访问方便。
使用第三方库webob，可以把环境数据的解析、封装成对象。

webob简介

Python下，可以对WSGI请求进行解析，并提供对响应进行高级封装的库。

```
$ pip install webob
```

官网文档 `docs.webob.org`

webob.Request对象

将环境参数解析并封装成request对象

GET方法，发送的数据是URL中Query string，在Request Header中。
request.GET就是一个字典MultiDict，里面就封装着查询字符串。

POST方法，"提交"的数据是放在Request Body里面，但是也可以同时使用Query String。
request.POST可以获取Request Body中的数据，也是个字典MultiDict。

不关心什么方法提交，只关心数据，可以使用request.params，它里面是所有提交数据的封装。

```
    request = webob.Request(environ)
    print(request.headers) # 类字典容器
    print(request.method)
    print(request.path)
    print(request.query_string) # 查询字符串
    print(request.GET) # GET方法的所有数据
    print(request.POST) # POST方法的所有数据
    print('params = {}'.format(request.params)) # 所有数据，参数
```

MultiDict

MultiDict允许一个key存了好几个值。

```
from webob.multidict import MultiDict

md = MultiDict()

md.add(1, 'magedu')
md.add(1, '.com')
md.add('a', 1)
md.add('a', 2)
md.add('b', '3')
md['b'] = '4'

for pair in md.items():
    print(pair)

print(md.getall(1))
#print(md.getone('a')) # 只能有一个值
print(md.get('a')) # 返回一个值
print(md.get('c')) # 不会抛异常KeyError，返回None
```

webob.Response对象

```
res = webob.Response()
print(res.status)
print(res.headerlist)
start_response(res.status, res.headerlist)
# 返回可迭代对象
html = '<h1>马哥教育欢迎你</h1>'.encode("utf-8")
return [html]
```

如果一个Application是一个类的实例，可以实现`__call__`方法。
我们来看看webob.Response类的源代码

```
def __call__(self, environ, start_response):
    """
    WSGI application interface
    """
    if self.conditional_response:
        return self.conditional_response_app(environ, start_response)

    headerlist = self._abs_headerlist(environ)

    start_response(self.status, headerlist)
    if environ['REQUEST_METHOD'] == 'HEAD':
        # Special case here...
        return EmptyResponse(self._app_iter)
    return self._app_iter
```

由此可以得到下面代码

```
def application(environ:dict, start_response):
    # 请求处理
    request = webob.Request(environ)
    print(request.method)
    print(request.path)
    print(request.query_string)
    print(request.GET)
    print(request.POST)
    print('params = {}'.format(request.params))

    # 响应处理
    res = webob.Response() # [('Content-Type', 'text/html; charset=UTF-8'), ('Content-Length',
'0')]
    res.status_code = 200 # 默认200
    print(res.content_type)
    html = '<h1>马哥教育欢迎你</h1>'.encode("utf-8")
    res.body = html
    return res(environ, start_response)
```

webob.dec 装饰器

wsgify装饰器
文档
https://docs.pylonsproject.org/projects/webob/en/stable/api/dec.html

```
class webob.dec.wsgify(func=None, RequestClass=None, args=(), kwargs=None, middleware_wraps=None)
```

要求提供类似下面的可调用对象，以函数举例：

```
from webob.dec import wsgify

@wsgify
def app(request:webob.Request) -> webob.Response:
    res = webob.Response('<h1>马哥教育欢迎你. magedu.com</h1>')
    return res
```

> wsgify装饰器装饰的函数应该具有一个参数，这个参数是webob.Request类型，是对字典environ的对象化后的实例。
>
> 返回值
> 可以是一个webob.Response类型实例
> 可以是一个bytes类型实例，它会被封装成webob.Response类型实例的body属性
> 可以是一个字符串类型实例，它会被转换成bytes类型实例，然后会被封装成webob.Response类型实例的body属性
> 总之，返回值会被封装成webob.Response类型实例返回

由此修改测试代码，如下

```
from wsgiref.simple_server import make_server
import webob

from webob.dec import wsgify

# application函数不用了，用来和app函数对比
def application(environ:dict, start_response):
    # 请求处理
    request = webob.Request(environ)
    print(request.method)
    print(request.path)
    print(request.query_string)
    print(request.GET)
    print(request.POST)
    print('params = {}'.format(request.params))

    # 响应处理
    res = webob.Response() # [('Content-Type', 'text/html; charset=UTF-8'), ('Content-Length',
'0')]
    res.status_code = 200 # 默认200
    print(res.content_type)
    html = '<h1>马哥教育欢迎你</h1>'.encode("utf-8")
    res.body = html
    return res(environ, start_response)

@wsgify
def app(request:webob.Request) -> webob.Response:
    print(request.method)
    print(request.path)
    print(request.query_string)
    print(request.GET)
    print(request.POST)
    print('params = {}'.format(request.params))
```

```
    res = webob.Response('<h1>马哥教育欢迎你. magedu.com</h1>')
    return res

if __name__ == '__main__':
    ip = '127.0.0.1'
    port = 9999
    server = make_server(ip, port, app)
    try:
        server.serve_forever() # server.handle_request() 一次
    except KeyboardInterrupt:
        server.shutdown()
        server.server_close()
```

将上面的app函数封装成类

```
from webob import Response, Request
from webob.dec import wsgify
from wsgiref.simple_server import make_server

class App:
    @wsgify
    def __call__(self, request:Request):
        return '<h1>马哥教育欢迎你. magedu.com</h1>'


if __name__ == '__main__':
    ip = '127.0.0.1'
    port = 9999
    server = make_server(ip, port, App())
    try:
        server.serve_forever() # server.handle_request() 一次
    except KeyboardInterrupt:
        server.shutdown()
        server.server_close()
```

上面的代码中，所有的请求，都有这个App类的实例处理，需要对它进行改造。