类Flask框架实现

路由 route

什么是`路由`？
简单说，就是路怎么走。就是按照不同的路径分发数据。
URL代表对不同资源的地址的访问，可以认为请求不同路径对应的数据。对动态网页技术来说，不同的路径应该对应不同的应用程序来处理，返回数据，用户以为还是访问的静态的网页。
所以，代码中要增加对URL路径的分析处理。

不管是静态WEB服务器，还是动态WEB服务器，都需要路径和资源或处理程序的映射，最终返回HTML的文本。
静态WEB服务器，解决路径和文件之间的映射。
动态WEB服务器，解决路径和应用程序之间的映射。
所有的WEB框架都是如此，都有路由配置。

路由功能实现

路由类实现

路由功能，使用路由类实现。
路由类实现的功能主要就是完成path到handler函数的映射，使用字典保存最适合。
路由映射的建立需要提供一个方法register，它需要提供2个参数path和handler。

有以下的路由需求

路径	内容
/	返回欢迎内容
/python	返回Hello Python
其它路径	返回404

```
# 路由
# url = 'http://127.0.0.1:9999/python/index.html?id=5&name=wayne&age=19&age=20'
# path = '/python/index.html'

class Router:
    ROUTETABLE = {}

    def register(self, path, handler):
        self.ROUTETABLE[path] = handler

def indexhandler(request):
    return '<h1>马哥教育欢迎你. magedu.com</h1>'

def pythonhandler(request):
    return '<h1>Welcome to Magedu Python</h1>'
```

```
router = Router()
router.register('/', indexhandler)
router.register('/python', pythonhandler)
```

404处理

webob.exc提供了异常模块
https://docs.pylonsproject.org/projects/webob/en/stable/api/exceptions.html

使用webob.exc.HTTPNotFound表示路由表找不到对应的处理函数。

注册函数的改造

将注册函数改造装饰器

```
# 路由
# url = 'http://127.0.0.1:9999/python/index.html?id=5&name=wayne&age=19&age=20'
# path = '/python/index.html'

class Router:
    ROUTETABLE = {}

    @classmethod # 注册路由，装饰器
    def register(cls, path):
        def wrapper(handler):
            cls.ROUTETABLE[path] = handler
            return handler
        return wrapper

@Router.register('/')
def indexhandler(request):
    return '<h1>马哥教育欢迎你. magedu.com</h1>'

@Router.register('/python')
def pythonhandler(request):
    return '<h1>Welcome to Magedu Python</h1>'
```

将路由功能合并到App类中去。

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


class Router:
    ROUTETABLE = {}

    @classmethod # 注册路由，装饰器
    def register(cls, path):
```

```
        def wrapper(handler):
            cls.ROUTETABLE[path] = handler
            return handler
        return wrapper

@Router.register('/')
def indexhandler(request):
    return '<h1>马哥教育欢迎你. magedu.com</h1>'

@Router.register('/python')
def pythonhandler(request):
    return '<h1>Welcome to Magedu Python</h1>'


class App:
    _Router = Router

    @wsgify
    def __call__(self, request:Request):
        try:
            return self._Router.ROUTETABLE[request.path](request)
        except:
            raise HTTPNotFound('<h1>你访问的页面被外星人劫持了</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()
```

到目前为止，一个框架的雏形基本完成了。

App是WSGI中的应用程序，但是这个应用程序已经变成了一个路由程序，处理逻辑已经移到了应用程序外了，而这部分就是以后留给程序员完成的部分。

路由正则匹配

目前实现的路由匹配，路径匹配非常死板，使用正则表达式，可以更好的匹配路径。导入re模块。
注册的时候，存入不再是路径字符串，而是模式pattern。
`App的__call__`方法中实现模式和传入路径的匹配

```
compile 方法，编译正则表达式
match 方法，必须从头开始匹配，只匹配一次
search 方法，只匹配一次
fullmatch 方法，要完全匹配
findall 方法，从头开始找，找到所有匹配

分组捕获
'/(?P<biz>.*)/(?P<url>.*)' 贪婪
'/(?P<biz>.*?)/(?P<url>.*)' 非贪婪

@Application.register('^/$') # 只匹配根
@Application.register('/python$') # 只匹配/python
```

字典的问题
如果使用字典，key保存的是路径，普通字典遍历匹配的时候，是不能保证路径匹配的顺序的。
但是匹配过程应该是有顺序的。

正则表达式预编译
何时预编译正则表达式呢？
第一次使用的时候会影响用户体验，所以还是要在注册的时候编译。

综上，改用列表，元素使用二元组(编译后的正则对象, handler)

```python
from webob import Response, Request
from webob.dec import wsgify
from wsgiref.simple_server import make_server
from webob.exc import HTTPNotFound
import re

class Router:
    ROUTETABLE = [] # 列表，有序的

    @classmethod # 注册路由，装饰器
    def register(cls, pattern):
        def wrapper(handler):
            cls.ROUTETABLE.append((re.compile(pattern), handler)) # (预编译正则对象,处理函数)
            return handler
        return wrapper

@Router.register('^/$')
def indexhandler(request):
    return '<h1>马哥教育欢迎你. magedu.com</h1>'

@Router.register('^/python$')
def pythonhandler(request):
    return '<h1>Welcome to Magedu Python</h1>'


class App:
    _Router = Router

    @wsgify
```

```
    def __call__(self, request:Request):
        for pattern, handler in self._Router.ROUTETABLE:
            if pattern.match(request.path): # 正则匹配
                return handler(request)
        raise HTTPNotFound('<h1>你访问的页面被外星人劫持了</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()
```

正则表达式分组捕获

假设URL为`http://127.0.0.1:9999/11808`，路径为`/11808`，如何编写路径匹配的正则表达式？
`^/(?P<id>\d+)$`

```
@Router.register(r'^/$')
@Router.register(r'^/(?P<id>\d+)$')
def indexhandler(request):
    print(request.groups)
    print(request.groupdict)
    return '<h1>马哥教育欢迎你. magedu.com</h1>'

class App:
    _Router = Router

    @wsgify
    def __call__(self, request:Request):
        for pattern, handler in self._Router.ROUTETABLE:
            matcher = pattern.match(request.path)
            if matcher: # 正则匹配
                # 动态为request增加属性
                request.groups = matcher.groups() # 所有分组组成的元组，包括命名分组
                request.groupdict = matcher.groupdict() # 命名分组组成的字典
                return handler(request)
        raise HTTPNotFound('<h1>你访问的页面被外星人劫持了</h1>')
```

Request Method过滤

请求方法，一般来说即使是同一个URL，因为请求方法不同，处理方式也不同。
假设有一个URL，GET方法表示希望返回网页内容；POST方法表示浏览器提交数据过来需要处理并存入数据库，最终返回客户端存储成功或失败的信息。
换句话说，需要 请求方法、正则同时匹配才能决定执行 什么处理函数。

方法	含义
GET	请求指定的页面信息，并返回报头和正文
HEAD	类似于get请求，只不过返回的响应中没有具体的内容，用于获取报头
POST	向指定资源提交数据进行处理请求（例如提交表单或者上传文件）。数据被包含在请求正文中。POST请求可能会导致新的资源的建立或已有资源的修改
PUT	从客户端向服务器传送的数据取代指定的文档的内容
DELETE	请求服务器删除指定的内容

请求方法还有很多，这里不再赘述。

实现请求方法判断的方式有：
1、在register装饰器中增加参数

```
@classmethod # 注册路由，装饰器
def register(cls, method, pattern):
    def wrapper(handler):
        cls.ROUTETABLE.append((method.upper(), re.compile(pattern), handler)) # (预编译正则对象，处理函数)
        return handler
    return wrapper

@Application.register('GET', '^/$')
def handler(request):
    pass
```

2、将register分解成不同方法的装饰器

```
@classmethod
def get(cls, pattern):
    return cls.register('GET', pattern)
```

将register注册方法改名为route路由方法。
改造代码如下：

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

class Router:
    ROUTETABLE = [] # 列表，有序的

    @classmethod # 注册路由，装饰器
    def route(cls, method, pattern):
        def wrapper(handler):
```

```
            cls.ROUTETABLE.append((method.upper(), re.compile(pattern), handler)) # (预编译正则对象,处理函数)
            return handler
        return wrapper

    @classmethod
    def get(cls, pattern):
        return cls.route('GET', pattern)

    @classmethod
    def post(cls, pattern):
        return cls.route('POST', pattern)

    @classmethod
    def head(cls, pattern):
        return cls.route('HEAD', pattern)

@Router.get(r'^/$')
@Router.route('GET', r'^/(?P<id>\d+)$')
def indexhandler(request):
    print(request.groups)
    print(request.groupdict)
    return '<h1>马哥教育欢迎你. magedu.com</h1>'

@Router.get('^/python$')
def pythonhandler(request):
    res = Response()
    res.charset = 'utf-8'
    res.body = '<h1>Welcome to Magedu Python</h1>'.encode()
    return res


class App:
    _Router = Router

    @wsgify
    def __call__(self, request:Request):
        for method, pattern, handler in self._Router.ROUTETABLE:
            if request.method.upper() != method:
                continue
            matcher = pattern.match(request.path)
            if matcher: # 正则匹配
                # 动态为request增加属性
                request.groups = matcher.groups() # 所有分组组成的元组，包括命名分组
                request.groupdict = matcher.groupdict() # 命名分组组成的字典
                return handler(request)
        raise HTTPNotFound('<h1>你访问的页面被外星人劫持了</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()
```

改进

一个URL可以设定多种请求方法，怎么解决？

思路一
如果什么方法都不写，相当于所有方法都支持。
@Application.route('^/$') # 隐含一个默认值method=None
等价于@Application.route(None, '^/$')
如果一个处理函数handler需要关联多个请求方法method，如下：
@Router.route(('GET','PUT','DELETE'), '^/$')
@Router.route(['GET','PUT','POST'], '^/$')
@Router.route({'POST','PUT','DELETE'}, '^/$')

思路二
调整参数位置，把method放到后面变成可变参数
def route(cls, pattern, *methods):
    pass
那么methods有可能是一个空元组，表示匹配所有方法；非空表示，匹配指定方法。
@Router.route('^/$', 'POST','PUT','DELETE')
@Router.route('^/$')

选择思路二

完整代码如下

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

class Router:
    ROUTETABLE = [] # 列表，有序的

    @classmethod # 注册路由，装饰器
    def route(cls, pattern, *methods):
        def wrapper(handler):
            cls.ROUTETABLE.append(
                (tuple(map(lambda x: x.upper(), methods)),
                 re.compile(pattern), handler)) # (方法元组,预编译正则对象,处理函数)
            return handler
        return wrapper

    @classmethod
    def get(cls, pattern):
        return cls.route(pattern, 'GET')
```

```
    @classmethod
    def post(cls, pattern):
        return cls.route(pattern, 'POST')

    @classmethod
    def head(cls, pattern):
        return cls.route(pattern, 'HEAD')

@Router.get(r'^/$')
@Router.route(r'^/(?P<id>\d+)$') # 支持所有方法访问
def indexhandler(request):
    print(request.groups)
    print(request.groupdict)
    return '<h1>马哥教育欢迎你. magedu.com</h1>'

@Router.get('^/python$')
def pythonhandler(request):
    res = Response()
    res.charset = 'utf-8'
    res.body = '<h1>Welcome to Magedu Python</h1>'.encode()
    return res


class App:
    _Router = Router

    @wsgify
    def __call__(self, request:Request):
        for methods, pattern, handler in self._Router.ROUTETABLE:
            # not methods表示一个方法都没有定义，就是支持全部方法
            if not methods or request.method.upper() in methods:
                matcher = pattern.match(request.path)
                if matcher: # 正则匹配
                    # 动态为request增加属性
                    request.groups = matcher.groups() # 所有分组组成的元组，包括命名分组
                    request.groupdict = matcher.groupdict() # 命名分组组成的字典
                    return handler(request)
        raise HTTPNotFound('<h1>你访问的页面被外星人劫持了</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()
```

路由匹配从 URL => handler 变成了 method + URL => handler

马哥教育
IT人的高薪职业学院