类Flask框架实现

模板

目前在框架中返回的是字符串，这些字符串返回时会被webob包装成Response对象，这些就是HTTP响应的正文。这种方式有一个问题，就是数据和HTML混合在一起，也就是数据和数据的格式混在了一起。能否将数据和格式分离？

思路
request请求被handler处理后得到数据，设计一个HTML网页，其中设置一些特殊的字符，将数据插入到这个HTML网页中指定的位置，生成新的HTML，返回浏览器端。

设计首页HTML模板index.html

```
<!DOCTYPE HTML PUBLIC "-//W3C//DTD HTML 4.01 Transitional//EN"
        "http://www.w3.org/TR/html4/loose.dtd">
<html>
<head>
    <title>Magedu</title>
</head>
<body>
显示数据<br>
{{id}} {{name}} {{age}}
</body>
</html>
```

在网页中使用双花括号来表示占位符，例如`{{id}}`，以后只需要找到这个位置，从数据中找到对应名称的数据替换即可。
为此，提供一个字典`{'id':5, 'name':'tom', 'age':20}`，将这些数据插入对应的占位符中。

```
import re
from io import StringIO, BytesIO

d = {'id':5, 'name':'tom', 'age':20}

class Template:
    _pattern = '{{([a-zA-Z0-9_]+)}}'
    regex = re.compile(_pattern)

    @classmethod
    def render(cls, template, data:dict):
        html = StringIO()

        with open(template, encoding='utf-8') as f:
            for line in f:
                start = 0
                newline = ''
                for matcher in cls.regex.finditer(line):
```

```
                    newline += line[start:matcher.start()]
                    print(matcher, matcher.group(1))
                    key = matcher.group(1)
                    tmp = data.get(key, '')
                    newline += str(tmp)
                    start = matcher.end()
                else:
                    newline += line[start:]
                html.write(newline)

            print(html.getvalue())
        html.close()


# 模板渲染
filename = 'index.html'
Template.render(filename, d)
```

jinja2

基于Python的模板引擎。设计思想来自Django的模板引擎，和其非常相似。

文档
官网 http://jinja.pocoo.org/docs/2.10/
中文 http://docs.jinkan.org/docs/jinja2/

安装
pip install jinja2
pip install MarkupSafe

模板构建

在当前项目下，新建包webarch，其下新建一个目录templates，该目录下新建一个HTML模板文件index.html

```
<!DOCTYPE HTML PUBLIC "-//W3C//DTD HTML 4.01 Transitional//EN"
        "http://www.w3.org/TR/html4/loose.dtd">
<html>
<head>
    <title>Magedu</title>
    <meta http-equiv="Content-Type" content="text/html; charset=utf8">
</head>
<body>
显示数据<br>
<ul>
    {% for id,name,age in userlist %}
    <li>{{loop.index}} {{id}}， {{name}}， {{age}}</li>
    {% endfor %}
</ul>
总共{{usercount}}人
</body>
</html>
```

加载模板代码如下

```
from jinja2 import Environment, PackageLoader, FileSystemLoader

#env = Environment(loader=PackageLoader('webarch', 'templates')) # 包加载器
env = Environment(loader=FileSystemLoader('webarch/templates')) # 文件系统加载器

template = env.get_template('index.html') # 搜索模块

userlist = [
    (3, 'tom', 20),
    (5, 'jerry', 16),
    (6, 'sam', 23),
    (8, 'kevin', 18)
]

d = {'userlist':userlist, 'usercount':len(userlist)}
print(template.render(**d))
```

提供模板模块template.py

```
from jinja2 import Environment, PackageLoader, FileSystemLoader

#env = Environment(loader=PackageLoader('webarch', 'templates')) # 包加载器
env = Environment(loader=FileSystemLoader('webarch/templates')) # 文件系统加载器

def render(name, data:dict):
    """
    模板渲染
    :param name: 去模板目录搜索此模板名的文件
    :param data: 字典
    :return: HTML字符串
    """
    template = env.get_template(name) # 搜索模块index.html
    return template.render(**data)
```

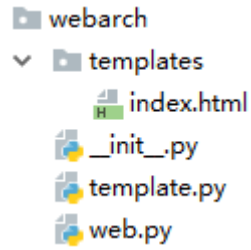
代码中增加

```
# 创建Router对象
idx = Router()
py = Router('/python')
user = Router('/user')

# 注册
App.register(idx, py)
App.register(user)

@user.get(r'^/?$')
def userhandler(request):
    userlist = [
```

```
        (3, 'tom', 20),
        (5, 'jerry', 16),
        (6, 'sam', 23),
        (8, 'kevin', 18)
    ]

    d = {'userlist': userlist, 'usercount': len(userlist)}
    return render('index.html', d)
```

模块化

将所有代码组织成包和模块。



包webarch
template.py模块移入此包
新建web.py模块，将AttrDict、Router、App类放入其中
将路由定义、注册代码、handler定义挪入 `webarch/__init__.py` 中

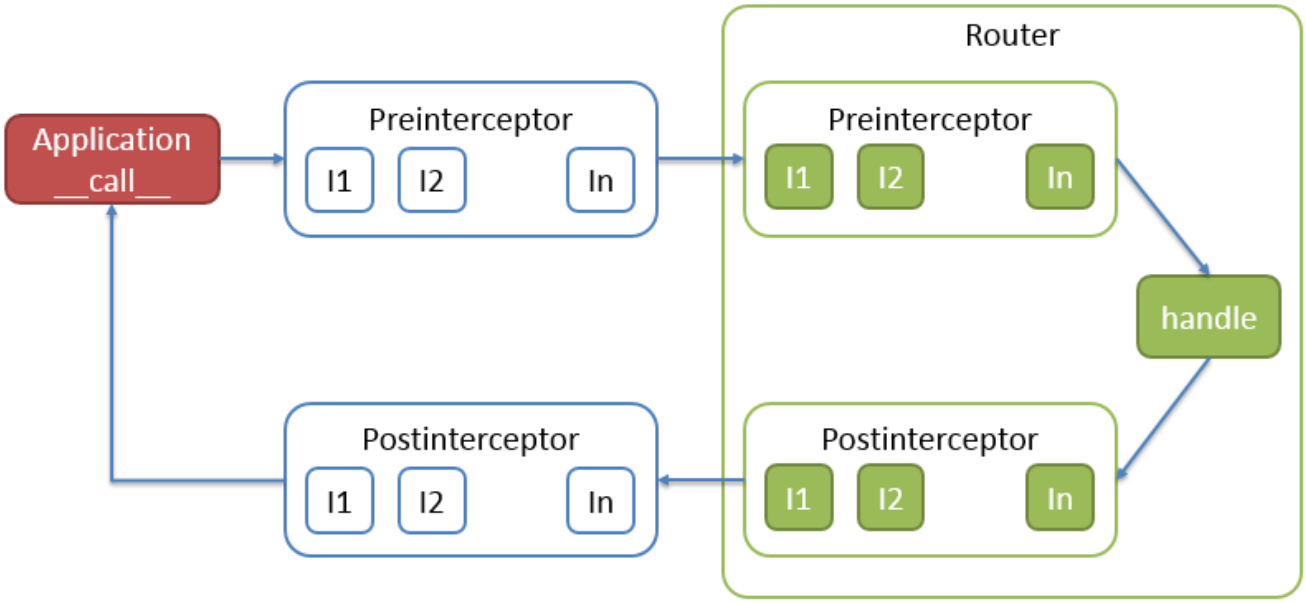
在项目根目录下，建一个app.py，里面放入启动server的代码

拦截器interceptor

拦截器，就是要在请求处理环节的某处加入处理，有可能是中断后续的处理。

根据拦截点不同，分为：
1、请求时拦截
2、响应时拦截

根据影响面分为：
1、全局拦截
在App中拦截
2、局部拦截
在Router中拦截



拦截器可以是多个，多个拦截器执行是有顺序的
数据response前执行的命名为preinterceptor，之后的命名为postinterceptor。

加入拦截器功能的方式
1、App和Router类直接加入
把拦截器的相关方法、属性分别添加到相关类中
实现简单
2、Mixin
App和Router类都需要这个拦截器功能，这两个类有没有什么关系，可以使用Mixin方式，将属性、方法组合进来
但是，App类拦截器适合使用第二种，但是Router的拦截器是每个实例不一样的，所以使用第一种方式实现

拦截函数fn的设计
fn不能影响数据继续向下一级的传递，也就是它是"透明"的。
handle之前，函数签名应该如下

```
def fn(request:Request) -> Request:
    pass
```

引入app，是为了以后从Application上获取一些全局信息。
handle执行之后，有了response对象，所以函数签名应该如下

```
def fn(request:Request, request:Response) -> Response:
    pass
```

拦截器代码实现，见本文最后的完整代码

IP拦截

`/python`只允许192开头的IP访问

```
# 创建Router对象
idx = Router()
py = Router('/python')
```

```
user = Router('/user')

# 注册
App.register(idx, py)
App.register(user)

def ip(request:Request):
    print(request.remote_addr, '~~~~~~~~~~~~~~~~')
    print(type(request.remote_addr))
    if request.remote_addr.startswith('192.'):
        return request
    else:
        return None

py.register_preinterceptor(ip)
```

Json支持

webob.response.Response支持json，如下

```
@py.get('^/{id}$')
def pythonhandler(request):
    userlist = [
        (3, 'tom', 20),
        (5, 'jerry', 16),
        (6, 'sam', 23),
        (8, 'kevin', 18)
    ]

    d = {'userlist': userlist, 'usercount': len(userlist)}
    res = Response(json=d)
    return res
```

测试`http://127.0.0.1:9999/python/123`，返回`{"userlist":[[3,"tom",20],[5,"jerry",16],[6,"sam",23],[8,"kevin",18]],"usercount":4}`

总结

1. 熟悉WSGI的编程接口
2. 强化模块化、类封装的思想
3. 增强分析业务的能力

这个框架基本具备了WSGI WEB框架的基本功能，其他框架都类似。

权限验证、SQL注入检测的功能使用拦截器实现。

完整代码

webarch/__init__.py

```
from webob import Response, Request
# 模板
from .template import render
from .web import Router, App


# 创建Router对象
idx = Router()
py = Router('/python')
user = Router('/user')

# 注册
App.register(idx, py)
App.register(user)

# ip拦截
def ip(request:Request):
    print(request.remote_addr, '~~~~~~~~~~~~~~~~')
    print(type(request.remote_addr))
    if request.remote_addr.startswith('192.'):
        return request
    else:
        return None # 返回None将截断请求

py.register_preinterceptor(ip)

@idx.get(r'^/$')
@idx.route(r'^/{id:int}$') # 支持所有方法访问
def indexhandler(request):
    id = ''
    if request.vars:
        id = request.vars.id
        print(type(id))
    return '<h1>马哥教育欢迎你{}. magedu.com</h1>'.format(id)

@py.get('^/{id}$')
def pythonhandler(request):
    userlist = [
        (3, 'tom', 20),
        (5, 'jerry', 16),
        (6, 'sam', 23),
        (8, 'kevin', 18)
    ]

    d = {'userlist': userlist, 'usercount': len(userlist)}
    res = Response(json=d)
    return res
```

```
@user.get(r'^/?$')
def userhandler(request):
    userlist = [
        (3, 'tom', 20),
        (5, 'jerry', 16),
        (6, 'sam', 23),
        (8, 'kevin', 18)
    ]

    d = {'userlist': userlist, 'usercount': len(userlist)}
    return render('index.html', d)
```

webarch/web.py

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


class AttrDict:
    def __init__(self, d:dict):
        self.__dict__.update(d if isinstance(d, dict) else {})

    def __setattr__(self, key, value):
        # 不允许修改属性
        raise NotImplementedError

    def __repr__(self):
        return "<AttrDict {}>".format(self.__dict__)

    def __len__(self):
        return len(self.__dict__)

class Router:
    ############## 正则转换
    __regex = re.compile(r'/{([^{}:]+):?([^{}:]*)}')

    TYPEPATTERNS = {
        'str': r'[^/]+',
        'word': r'\w+',
        'int': r'[+-]?\d+',
        'float': r'[+-]?\d+\.\d+',  # 严苛的要求必须是 15.6这样的形式
        'any': r'.+'
    }

    TYPECAST = {
        'str': str,
        'word': str,
        'int': int,
        'float': float,
```

```
        'any': str
    }

    def __parse(self, src: str):
        start = 0
        repl = ''
        types = {}

        matchers = self.__regex.finditer(src)
        for i, matcher in enumerate(matchers):
            name = matcher.group(1)
            t = matcher.group(2)

            types[name] = self.TYPECAST.get(matcher.group(2), str)

            repl += src[start:matcher.start()]  # 拼接分组前
            tmp = '/(?P<{}>{})'.format(
                matcher.group(1),
                self.TYPEPATTERNS.get(matcher.group(2), self.TYPEPATTERNS['str'])
            )
            repl += tmp  # 替换

            start = matcher.end()  # 移动
        else:
            repl += src[start:]  # 拼接分组后的内容

        return repl, types

    ############ 实例
    def __init__(self, prefix:str=''):
        self.__prefix = prefix.rstrip('/\\') # 前缀，例如/product
        self.__routetable = [] # 存四元组，列表，有序的

        # 拦截器
        self.pre_interceptor = []
        self.post_interceptor = []

    # 拦截器注册函数
    def register_preinterceptor(self, fn):
        self.pre_interceptor.append(fn)
        return fn

    def register_postinterceptor(self, fn):
        self.post_interceptor.append(fn)
        return fn

    def route(self, rule, *methods): # 注册路由，装饰器
        def wrapper(handler):
            # /student/{name:str}/xxx/{id:int}  =>
            # '/student/(?P<name>[^/]+)/xxx/(?P<id>[+-]?\\d+)'
            pattern, trans = self.__parse(rule) # 用户输入规则转换为正则表达式
            self.__routetable.append(
                (tuple(map(lambda x: x.upper(), methods)),
```

```
                re.compile(pattern), # 编译
                trans,
                handler)
            ) # (方法元组,预编译正则对象,类型转换,处理函数)
            return handler
        return wrapper

    def get(self, pattern):
        return self.route(pattern, 'GET')

    def post(self, pattern):
        return self.route(pattern, 'POST')

    def head(self, pattern):
        return self.route(pattern, 'HEAD')

    def match(self, request:Request):
        # 必须先匹配前缀
        if not request.path.startswith(self.__prefix):
            return None

        # 是此Router的请求，开始拦截，处理request
        for fn in self.pre_interceptor:
            request = fn(request)
            if not request:
                return None # 请求为None将不再向后传递，截止

        # 前缀匹配，说明就必须这个Router实例处理，后续匹配不上，依然返回None
        for methods, pattern, trans, handler in self.__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:
                # 前提已经是以__prefix开头了，可以replace，去掉prefix剩下的才是正则表达式需要匹配的路
径
                matcher = pattern.match(request.path.replace(self.__prefix, '', 1))
                if matcher: # 正则匹配
                    newdict = {}
                    for k,v in matcher.groupdict().items(): # 命名分组组成的字典
                        newdict[k] = trans[k](v)
                    # 动态为request增加属性
                    request.vars = AttrDict(newdict) # 属性化
                    response = handler(request)

                    # 依次拦截，处理响应
                    for fn in self.post_interceptor:
                        response = fn(request, response)
                    return response


class App:
    _ROUTERS = [] # 存储所有一级Router对象
    # 全局拦截器
    PRE_INTERCEPTOR = []
    POST_INTERCEPTOR = []
```

```
    # 全局拦截器注册函数
    @classmethod
    def register_preinterceptor(cls, fn):
        cls.PRE_INTERCEPTOR.append(fn)
        return fn

    @classmethod
    def register_postinterceptor(cls, fn):
        cls.POST_INTERCEPTOR.append(fn)
        return fn

    # 注册路由
    @classmethod
    def register(cls, *routers:Router):
        for router in routers:
            cls._ROUTERS.append(router)

    @wsgify
    def __call__(self, request:Request):
        # 全局拦截请求
        for fn in self.PRE_INTERCEPTOR:
            request = fn(request)

        # 遍历_ROTERS，调用Router实例的match方法，看谁匹配
        for router in self._ROUTERS:
            response = router.match(request)

            # 全局拦截响应
            for fn in self.POST_INTERCEPTOR:
                response = fn(request, response)

            if response: # 匹配返回非None的Router对象
                return response # 匹配则立即返回
        raise HTTPNotFound('<h1>你访问的页面被外星人劫持了</h1>')
```

webarch/template.py

```
from jinja2 import Environment, PackageLoader, FileSystemLoader

#env = Environment(loader=PackageLoader('webarch', 'templates')) # 包加载器
env = Environment(loader=FileSystemLoader('webarch/templates')) # 文件系统加载器

def render(name, data:dict):
    """
    模板渲染
    :param name: 去模板目录搜索此模板名的文件
    :param data: 字典
    :return: HTML字符串
    """
    template = env.get_template(name) # 搜索模块index.html
```

```
    return template.render(**data)
```

app.py

```
from wsgiref.simple_server import make_server
from webarch import App


if __name__ == '__main__':
    ip = '0.0.0.0'
    port = 9999
    server = make_server(ip, port, App())
    try:
        server.serve_forever() # server.handle_request() 一次
    except KeyboardInterrupt:
        server.shutdown()
        server.server_close()
```

发布

setup.py

```
from distutils.core import setup
import glob

# 导入setup函数并传参
setup(name='webarch',
      version='0.1.0',
      description='Python WSGI WEB Framework',
      author='wayne',
      author_email='wayne@magedu.com',
      url='https://www.magedu.com',
      #packages=['m', 'm.m1', 'm.m2', 'm.m2.m21'],
      packages=['webarch'],
      #data_files=['webarch/templates/index.html']
      data_files=glob.glob('webarch/templates/*') # 返回列表
     )
# name名字
# version 版本
# packages=[] 打包列表，
# packages=['magweb'],指定magweb，就会把magweb所有的非目录子模块打包
# description 描述信息
# author 作者
# author_email 作者邮件
# url 包的主页，可以不写
# data_files 配置文件、图片等文件列表。参看distutils.command.sdist.sdist#add_defaults
```

```
# 打包
$ python setup.py sdist

# 安装
$ pip install webarch-0.1.0.zip
```