Day 5 - 编写Web框架

542次阅读

在正式开始Web开发前，我们需要编写一个Web框架。

为什么不选择一个现成的Web框架而是自己从头开发呢？我们来考察一下现有的流行的Web框架：

Django：一站式开发框架，但不利于定制化；

web.py：使用类而不是更简单的函数来处理URL，并且URL映射是单独配置的；

Flask：使用@decorator的URL路由不错，但框架对应用程序的代码入侵太强；

bottle：缺少根据URL模式进行拦截的功能，不利于做权限检查。

所以，我们综合几种框架的优点，设计一个简单、灵活、入侵性极小的Web框架。

设计Web框架

一个简单的URL框架应该允许以@decorator方式直接把URL映射到函数上：

```
# 首页:
@get('/')
def index():
    return '<h1>Index page</h1>'

# 带参数的URL:
@get('/user/:id')
def show_user(id):
    user = User.get(id)
    return 'hello, %s' % user.name
```

有没有@decorator不改变函数行为，也就是说，Web框架的API入侵性很小，你可以直接测试函数show_user(id)而不需要启动Web服务器。

函数可以返回str、unicode以及iterator，这些数据可以直接作为字符串返回给浏览器。

其次，Web框架要支持URL拦截器，这样，我们就可以根据URL做权限检查：

```
@interceptor('/manage/')
def check_manage_url(next):
    if current_user.isAdmin():
        return next()
    else:
        raise seeother('/signin')
```

拦截器接受一个next函数，这样，一个拦截器可以决定调用next()继续处理请求还是直接返回。

为了支持MVC，Web框架需要支持模板，但是我们不限定使用哪一种模板，可以选择jinja2，也可以选择mako、Cheetah等等。

要统一模板的接口，函数可以返回dict并配合@view来渲染模板：

```
@view('index.html')
@get('/')
```

```
def index():
    return dict(blogs=get_recent_blogs(), user=get_current_user())
```

如果需要从form表单或者URL的querystring获取用户输入的数据，就需要访问request对象，如果要设置特定的Content-Type、设置Cookie等，就需要访问response对象。request和response对象应该从一个唯一的ThreadLocal中获取：

```
@get('/test')
def test():
    input_data = ctx.request.input()
    ctx.response.content_type = 'text/plain'
    ctx.response.set_cookie('name', 'value', expires=3600)
    return 'result'
```

最后，如果需要重定向、或者返回一个HTTP错误码，最好的方法是直接抛出异常，例如，重定向到登陆页：

```
raise seeother('/signin')
```

返回404错误：

```
raise notfound()
```

基于以上接口，我们就可以实现Web框架了。

实现Web框架

最基本的几个对象如下：

```
# transwarp/web.py

# 全局ThreadLocal对象：
ctx = threading.local()

# HTTP错误类:
class HttpError(Exception):
    pass

# request对象:
class Request(object):
    # 根据key返回value:
    def get(self, key, default=None):
        pass

    # 返回key-value的dict:
    def input(self):
        pass

    # 返回URL的path:
    @property
    def path_info(self):
        pass

    # 返回HTTP Headers:
    @property
    def headers(self):
        pass

    # 根据key返回Cookie value:
    def cookie(self, name, default=None):
        pass
```

```
# response对象:
class Response(object):
    # 设置header:
    def set_header(self, key, value):
        pass

    # 设置Cookie:
    def set_cookie(self, name, value, max_age=None, expires=None, path='/'):
        pass

    # 设置status:
    @property
    def status(self):
        pass
    @status.setter
    def status(self, value):
        pass

# 定义GET:
def get(path):
    pass

# 定义POST:
def post(path):
    pass

# 定义模板:
def view(path):
    pass

# 定义拦截器:
def interceptor(pattern):
    pass

# 定义模板引擎:
class TemplateEngine(object):
    def __call__(self, path, model):
        pass

# 缺省使用jinja2:
class Jinja2TemplateEngine(TemplateEngine):
    def __init__(self, templ_dir, **kw):
        from jinja2 import Environment, FileSystemLoader
        self._env = Environment(loader=FileSystemLoader(templ_dir), **kw)

    def __call__(self, path, model):
        return self._env.get_template(path).render(**model).encode('utf-8')
```

把上面的定义填充完毕，我们就只剩下一件事情：定义全局WSGIApplication的类，实现WSGI接口，然后，通过配置启动，就完成了整个Web框架的工作。

设计WSGIApplication要充分考虑开发模式（Development Mode）和产品模式（Production Mode）的区分。在产品模式下，WSGIApplication需要直接提供WSGI接口给服务器，让服务器调用该接口，而在开发模式下，我们更希望能通过app.run()直接启动服务器进行开发调试：

```
wsgi = WSGIApplication()
if __name__ == '__main__':
    wsgi.run()
else:
    application = wsgi.get_wsgi_application()
```

因此，WSGIApplication定义如下：

```
class WSGIApplication(object):
    def __init__(self, document_root=None, **kw):
        pass

    # 添加一个URL定义:
    def add_url(self, func):
        pass

    # 添加一个Interceptor定义:
    def add_interceptor(self, func):
        pass

    # 设置TemplateEngine:
    @property
    def template_engine(self):
        pass

    @template_engine.setter
    def template_engine(self, engine):
        pass

    # 返回WSGI处理函数:
    def get_wsgi_application(self):
        def wsgi(env, start_response):
            pass
        return wsgi

    # 开发模式下直接启动服务器:
    def run(self, port=9000, host='127.0.0.1'):
        from wsgiref.simple_server import make_server
        server = make_server(host, port, self.get_wsgi_application())
        server.serve_forever()
```

把WSGIApplication类填充完毕，我们就得到了一个完整的Web框架。
