

1. 渲染编辑页面

参考如下图

The screenshot shows the PyCharm IDE interface. On the left is the project tree for 'flpro' with 'User.py' selected. The main editor window shows 'User.py' with the following code:

```
# 2. 删除 AUTH_COOKIE_NAME COOKIE
response.delete_cookie(app.config.get("AUTH_COOKIE_NAME")) # 去掉 cookie
# 3. 响应
return response

class EditInfo(views.MethodView):
    def get(self):
        return render_template('user/edit.html')

# 使用add_url_rule方法来构造与视图关联的请求url
# 继承自模板的as_view方法可以将类转换为可以为路由注册的视图函数
# as_view('login') --- 传入的是站点名（标准操作是视图名小写 Login -- login）
# <Rule '/user/login' (GET, HEAD, OPTIONS) -> user_page.login>] user_page.login
route_user.add_url_rule('/edit', view_func=EditInfo.as_view('edit'))
route_user.add_url_rule('/login', view_func=Login.as_view('login'))
route_user.add_url_rule('/logout', view_func=Logout.as_view('logout'))
```

```
# -*- encoding: utf-8 -*-
"""
File      : User.py
teaching   :
    用户模块
"""

from flask import Blueprint, render_template, views, request, jsonify, make_response, url_for, redirect, g
from common.lib.UrlManager import UrlManager
from web.models.User import User # 模型类导出
from common.lib.UserService import UserService
from application import app
import json

# 构建蓝图对象
route_user = Blueprint('user_page', __name__)

class EditInfo(views.MethodView):
    def get(self):
        return render_template('user/edit.html')
```

```
# 使用add_url_rule方法来构造与视图关联的请求url  
# 继承自模板的as_view方法可以将类转换为可以为路由注册的视图函数  
# as_view('login') --- 传入的是站点名 (标准操作是视图名小写 Login -- login)  
#           <Rule '/user/login' (GET, HEAD, OPTIONS) -> user_page.login>] user_page.login中的login就是  
as_view指明的站点名  
route_user.add_url_rule('/edit', view_func=EditInfo.as_view('edit'))  
route_user.add_url_rule('/login', view_func=Login.as_view('login'))  
route_user.add_url_rule('/logout', view_func=Logout.as_view('logout'))
```

访问 <http://127.0.0.1:8888/user/edit> 或者如下图进入

127.0.0.1:8888/user/edit

2. 用户数据渲染问题及解决方案

用户数据 我们在视图中如何快速获取呢?

- 前面我们设置了用户对象cookie
- 在拦截器（钩子）中我们通过验证用户cookie数据来判断登录状态
- 根据这点，也就是说，我们可以通过应用上下文g对象在钩子与视图中传递用户对象数据

```

Project: E:\pythonfile\flipro
File: AuthInterceptor.py
Line: 104
Code: g.current_user = user_info

```

```

# 编译忽略验证的路由列表的正则表达式模式, 返回pattern对象。
pattern = re.compile("|".join(ignore_merge))

# 校验当前请求路径(路由)是否需要拦截
if pattern.match(path): # 在拦截数据中就忽略
    return None # return None不做拦截

# 运用g对象建立用户数据上下文数据 --- 方便视图直接操作用户数据
# from flask import g
g.current_user = None

if user_info:
    # 如果登录状态 添加用户对象数据
    # user_info看上述第二步骤 -- 调用check_login()方法进行登录验证, 验证成功返回
    g.current_user = user_info

# 7.不在或略拦截里面 就校验 是否登录
if not user_info: # 没有登录 重定向到登录页面登录
    return redirect(UrlManager.buildUrl("/user/login"))

# 8.上述情况均为满足书名已经登陆了 就不做拦截
return None # return None不做拦截

```

```

# -*- encoding: utf-8 -*-
"""
File      : AuthInterceptor.py
teaching   :
    拦截器
"""

from application import app
from flask import request, redirect, g
from web.models.User import User
from common.lib.UserService import UserService
from common.lib.UrlManager import UrlManager
import re


def check_login():
    """判断用户是否登录"""
    # 1.获取cookie数据
    cookie = request.cookies

    # 2.获取用户 AUTH_COOKIE_NAME 数据
    auth_key = app.config["AUTH_COOKIE_NAME"]

    # 3.根据 AUTH_COOKIE_NAME 获取用户cookie数据
    # --- 注意此处使用了三目运算, 不熟悉三目运算的同学请回顾if else的三目运算知识点
    auth_cookie = cookie[auth_key] if auth_key in cookie else ""
    # print(auth_cookie) # 7d267998aa49e9c8a130f6a6b23763dc#1

    # 4.判断用户cookie数据是否存在
    if auth_cookie is None:
        return False # False表示没有登录

    # 5.用户cookie数据存在, 那么根据 # 切割截取出两段关键数据, 如下图中可进行观察
    auth_info = auth_cookie.split("#")

    # 6.判断切割截取的用户cookie数据是不是两段

```

```
if len(auth_info) != 2:  
    # 如果切割截取的cookie数据不是两段，那么说明此用户cookie不符合我们设定规则  
    # 说明登录状态有问题，判定属于未登录状态，重新登录  
    return False  
  
# 7.通过 切割截取的用户cookie数据 第二段 也就是 用户的id 进行数据查询（看下用户是否存在）  
try:  
    user_info = User.query.filter_by(uid=auth_info[1]).first()  
except:  
    # 异常说明用户数据有问题 --- 说明登录状态有问题，判定属于未登录状态，重新登录  
    return False  
if user_info is None: # 如果用户数据为None，说明登录状态有问题，判定属于未登录状态，重新登录  
    return False  
  
# 8.判断密钥  
# 通过用户数据 进行 加密还原  
authcode = UserService.genAuthCode(user_info)  
# 如果 用户cookie第一段的加密数据 与 当前加密数据 不同，说明登录不同，也属于未登录状态  
if auth_info[0] != authcode:  
    return False  
  
# 如果通过上述验证，说明我们用户处于登录状态  
return user_info  
  
# 通过钩子before_request 完成每次请求前的 用户登录验证  
@app.before_request  
def before_request():  
    """拦截器"""  
    # 1.获取当前请求请求路径（路由）  
    path = request.path  
  
    # 2.调用check_login()方法进行登录验证  
    user_info = check_login() # 登录校验  
  
    # 3.获取配置的需要忽略 验证路由 -- list  
    ignore_urls = app.config["IGNORE_URLS"]  
  
    # 4.获取 配置的 需要忽略验证的 资源加载路由 -- list  
    ignore_check_login_urls = app.config["IGNORE_CHECK_LOGIN_URLS"]  
  
    # 5. 忽略验证的路由列表 -- list  
    ignore_merge = ignore_urls + ignore_check_login_urls # 所有的url  
  
    # 6. 判断当前请求路径（路由）是否需要拦截 -- 下列方法注解请同学们自己单独去验证，不要在项目中乱来  
    # re.compile --- 编译正则表达式模式，返回pattern对象。  
    # match() --- 向它传入要匹配的字符串，以及正则表达式，就可以检测这个正则表达式是否匹配字符串。  
    # >>> import re  
    # >>> a = ['123', 'hahhah', 'shuai']  
    # >>> pattern = re.compile("|".join(a))  
    # >>> pattern  
    # >>> re.compile('123|hahhah|shuai')  
    # >>> pattern.match('1')  
    #  
    # >>> pattern.match('123')  
    #     <re.Match object; span=(0, 3), match='123'>  
    # 编译忽略验证的路由列表的正则表达式模式，返回pattern对象。  
    pattern = re.compile("|".join(ignore_merge))  
    # 校验当前当前请求路径（路由）是否需要拦截
```

```

if pattern.match(path): # 在拦截数据中就忽略
    return None # return None不做拦截

# 运用g对象建立用户数据上下文数据 --- 方便视图直接操作用户数据
# from flask import g
g.current_user = None
if user_info:
    # 如果登录状态 添加用户对象数据
    # user_info看上述第二步骤 -- 调用check_login()方法进行登录验证，验证成功返回的从数据库中查询到的
    用户数据
    g.current_user = user_info

# 7.不在或略拦截里面 就校验 是否登录
if not user_info: # 没有登录 重定向到登录页面登录
    return redirect(UrlManager.buildUrl("/user/login"))

# 8.上述情况均为满足书名已经登陆了 就不做拦截
return None # return None不做拦截

```

```

退出登录
退出登录后将会回到登录页
:return:
"""
# 1.构建登录也响应用对象
# from common.lib.UrlManager import UrlManager
response = make_response(redirect(UrlManager.buildUrl("/user/login"))) # 重定向
# 2.删除 AUTH_COOKIE_NAME cookie
response.delete_cookie(app.config.get("AUTH_COOKIE_NAME")) # 删除cookie
# 3.响应
return response

class EditInfo(views.MethodView):
    def get(self):
        # 导入g对象 from flask import g
        # g对象的生命周期和request相同

# 1.建立渲染数据字典
context = {}
# 2.数据为空，添加用户对象数据
if not context:
    context['current_user'] = g.current_user
return render_template('user/edit.html', **context)

```

```

# -*- encoding: utf-8 -*-
"""

File      : User.py
teaching   :
    用户模块 -- 登录
"""

from flask import Blueprint, render_template, views, request, jsonify, make_response, url_for, redirect, g
from common.lib.UrlManager import UrlManager
from web.models.User import User # 模型类导出
from common.lib.UserService import UserService
from application import app, db
import json

```

```

# 构建蓝图对象
route_user = Blueprint('user_page', __name__)

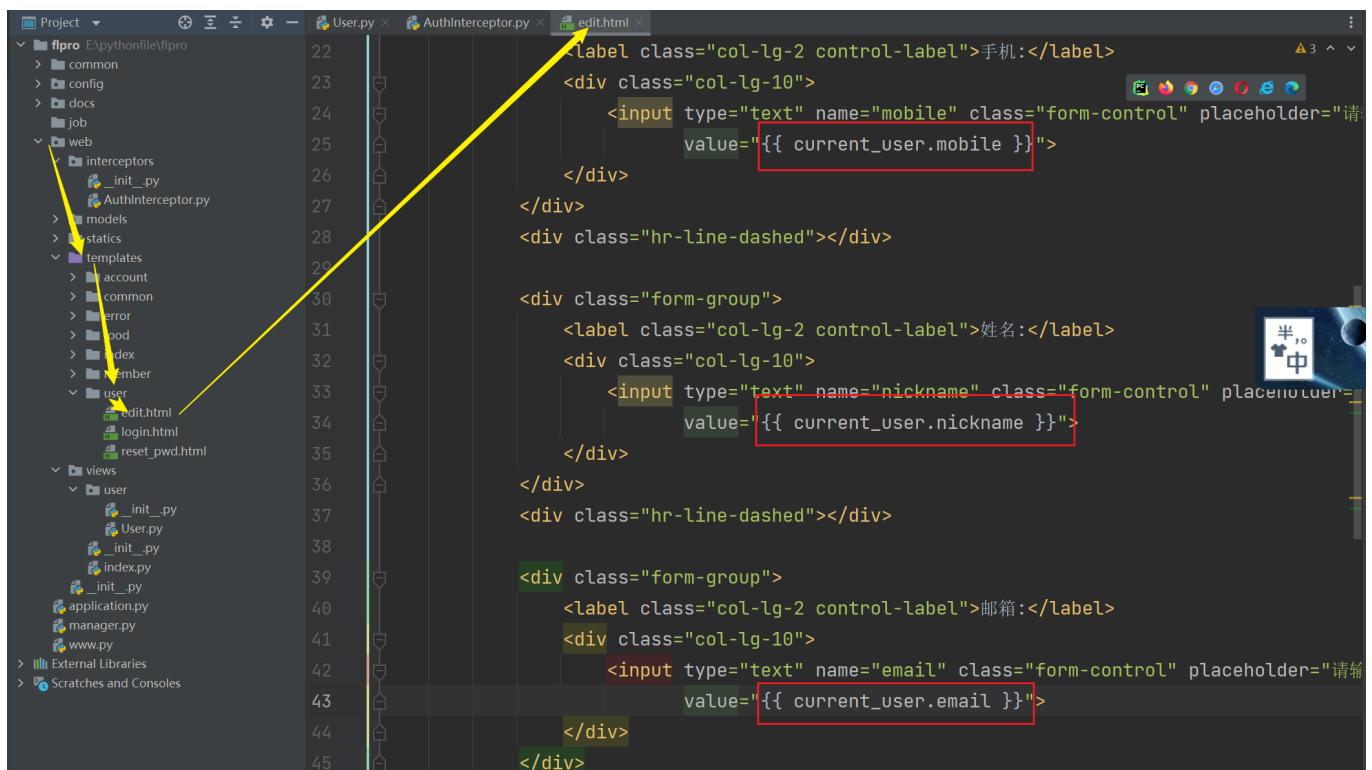
class EditInfo(MethodView):
    def get(self):
        # 导入g对象 from flask import g
        # g对象的生命周期和request相同

        # 1.建立渲染数据字典
        context = {}
        # 2.数据为空, 添加用户对象数据
        if not context:
            context['current_user'] = g.current_user
        return render_template('user/edit.html', **context)

# 使用add_url_rule方法来构造与视图关联的请求url
# 继承自模板的as_view方法可以将类转换为可以为路由注册的视图函数
# as_view('login') --- 传入的是站点名 (标准操作是视图名小写 Login -- login)
#           <Rule '/user/login' (GET, HEAD, OPTIONS) -> user_page.login>] user_page.login就是
# as_view指明的站点名
route_user.add_url_rule('/edit', view_func=EditInfo.as_view('edit'))
route_user.add_url_rule('/login', view_func=Login.as_view('login'))
route_user.add_url_rule('/logout', view_func=Logout.as_view('logout'))

```

3. 调整模板



```

label class="col-lg-2 control-label">手机:</label>
<div class="col-lg-10">
    <input type="text" name="mobile" class="form-control" placeholder="请输入手机号码" value="{{ current_user.mobile }}"/>
</div>
<div class="hr-line-dashed"></div>

<div class="form-group">
    <label class="col-lg-2 control-label">姓名:</label>
    <div class="col-lg-10">
        <input type="text" name="nickname" class="form-control" placeholder="请输入姓名" value="{{ current_user.nickname }}"/>
    </div>
</div>
<div class="hr-line-dashed"></div>

<div class="form-group">
    <label class="col-lg-2 control-label">邮箱:</label>
    <div class="col-lg-10">
        <input type="text" name="email" class="form-control" placeholder="请输入邮箱" value="{{ current_user.email }}"/>
    </div>
</div>

```

The screenshot shows the PyCharm IDE interface. On the left is the project tree with a 'fipro' project containing 'common', 'config', 'docs', 'job', 'web' (with 'interceptors', 'models', 'statics', 'templates' subfolders), 'views' (with 'user' folder), and 'External Libraries'. The right side shows the code editor with 'User.py' and 'AuthInterceptor.py' tabs open. The code in 'User.py' is highlighted with a red box around the JavaScript block. The status bar at the bottom right shows Chinese characters: 半屏 (Half Screen) and 中 (Middle).

```
44     44
45     45
46     46
47     47
48     48
49     49
50     50
51     51
52     52
53     53
54     54
55     55
56     56
57     57
58     58
59     59
60     60
61     61
62     62
```

```
</div>
<div class="hr-line-dashed"></div>
<div class="form-group">
    <div class="col-lg-4 col-lg-offset-2">
        <button class="btn btn-w-m btn-outline btn-primary save">保存</button>
    </div>
</div>
</div>
<% endblock %>
<% block js %>
<script src="{{ buildStaticUrl('/js/user/edit.js') }}></script>
<% endblock %>
```

```
{% extends "common/layout_main.html" %}
{% block content %}
<div class="row border-bottom">
    <div class="col-lg-12">
        <div class="tab_title">
            <ul class="nav nav-pills">
                <li class="current">
                    <a href="{{ buildUrl('/user/edit') }}>信息编辑</a>
                </li>
                <li>
                    <a href="{{ buildUrl('/user/edit') }}>修改密码</a>
                </li>
            </ul>
        </div>
    </div>
</div>
<div class="row m-t user_edit_wrap">
    <div class="col-lg-12">
        <h2 class="text-center">账号信息编辑</h2>
        <div class="form-horizontal m-t m-b">
            <div class="form-group">
                <label class="col-lg-2 control-label">手机:</label>
                <div class="col-lg-10">
                    <input type="text" name="mobile" class="form-control" placeholder="请输入手机
~~" readonly="" value="{{ current_user.mobile }}>
                </div>
            </div>
            <div class="hr-line-dashed"></div>

            <div class="form-group">
                <label class="col-lg-2 control-label">姓名:</label>
                <div class="col-lg-10">
                    <input type="text" name="nickname" class="form-control" placeholder="请输入姓名
~~" value="{{ current_user.nickname }}>
                </div>
            </div>
        </div>
    </div>
</div>
```

```

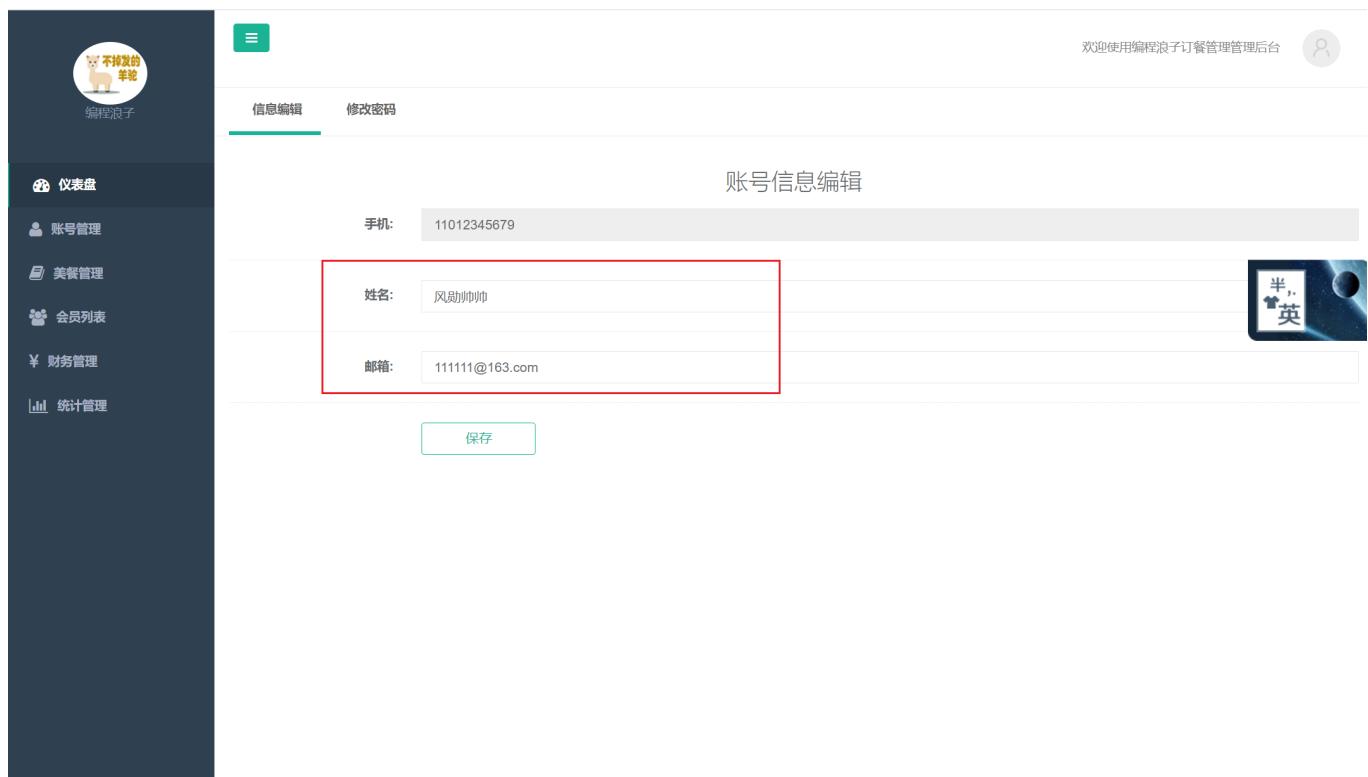
        </div>
        <div class="hr-line-dashed"></div>

        <div class="form-group">
            <label class="col-lg-2 control-label">邮箱:</label>
            <div class="col-lg-10">
                <input type="text" name="email" class="form-control" placeholder="请输入邮箱"
                value="{{ current_user.email }}>
            </div>
        </div>
        <div class="hr-line-dashed"></div>
        <div class="form-group">
            <div class="col-lg-4 col-lg-offset-2">
                <button class="btn btn-w-m btn-outline btn-primary save">保存</button>
            </div>
        </div>
        </div>
    </div>
    {% endblock %}

    {%block js %}
        <script src="{{ buildStaticUrl('/js/user/edit.js') }}></script>
    {% endblock %}

```

启动项目访问测试 <http://127.0.0.1:8888/user/edit>



4. 编辑修改实现

前端点击保存按钮后，触发js进行ajax提交

A screenshot of a code editor showing a project structure on the left and two files open on the right. The project structure includes 'flpro' with subfolders like 'common', 'docs', 'job', 'web' (which contains 'interceptors', 'models', 'statics', and 'views'). The 'views' folder under 'web' contains 'user' which has 'edit.js', 'login.js', 'reset_pwd.js', 'chartjs', 'common.js', and 'index.py'. The other file open is 'User.py'. A red arrow points from the 'edit.js' file in the project tree to the 'edit.js' file in the editor. A red box highlights a portion of the 'edit.js' code:

```
$ajax( url: {
    url: common_ops.buildUrl( path: "/user/edit" ),
    type: 'POST',
    data: data,
    dataType: 'json',
    success: function( res ) {
        btn_target.removeClass( value: "disabled" );
        var callback = null;
        if( res.code == 200 ){
            callback = function(){
                window.location.href = window.location.href;
            }
        } | common_ops.alert( res.msg, callback );
    }
});
```

ajax发送请求后，后端视图处理

A screenshot of a code editor showing a project structure on the left and a single Python file 'User.py' open on the right. The project structure is identical to the previous screenshot. A red arrow points from the 'User.py' file in the project tree to the 'User.py' file in the editor. A red box highlights a portion of the 'User.py' code:

```
class EditInfo(views.MethodView):
    def get(self):
        # 导入g对象 from flask import g
        # g对象的生命周期和request相同

        # 1.建立渲染数据字典
        context = {}

        # 2.数据为空，添加用户对象数据
        if not context:
            context['current_user'] = g.current_user
        return render_template('user/edit.html', **context)

    def post(self):
        """
        编辑信息提交保存
        :return:
        """

        # 1.构建响应数据
        resp = {"code": 200, "msg": "操作成功", "data": {}}

        # 2.获取提交数据
        req = request.values
```

```
# -*- encoding: utf-8 -*-
"""
File      : User.py
teaching   :
    用户模块 -- 登录
"""

# -
```

```
from flask import Blueprint, render_template, views, request, jsonify, make_response, url_for, redirect, g
from common.lib.UrlManager import UrlManager
from web.models.User import User # 模型类导出
from common.lib.UserService import UserService
from application import app, db
import json

# 构建蓝图对象
route_user = Blueprint('user_page', __name__)

class EditInfo(views.MethodView):
    def get(self):
        # 导入g对象 from flask import g
        # g对象的生命周期和request相同

        # 1.建立渲染数据字典
        context = {}
        # 2.数据为空，添加用户对象数据
        if not context:
            context['current_user'] = g.current_user
        return render_template('user/edit.html', **context)

    def post(self):
        """
        编辑信息提交保存
        :return:
        """
        # 1.构建响应数据
        resp = {"code": 200, "msg": "操作成功", "data": {}}

        # 2.获取提交数据
        req = request.values

        # 3.判断姓名，邮箱是否提交过来， 有就获取
        nickname = req["nickname"] if "nickname" in req else ""
        email = req["email"] if "email" in req else ""

        # 4.判断数据是否存在及是否符合要求（例如长度）
        if (not nickname) or len(nickname) < 1:
            resp["code"] = -1
            resp["msg"] = "请输入符合规范的姓名"
            return jsonify(resp)

        if (not email) or len(email) < 1:
            resp["code"] = -1
            resp["msg"] = "请输入符合规范的邮箱"
            return jsonify(resp)

        # 5.修改当前登录人姓名和邮箱
        # 导入g对象 from flask import g
        # g对象的生命周期和request相同
        user_info = g.current_user # 获取g对象中的用户对象数据
        user_info.nickname = nickname # 更改用户对象 用户名数据
        user_info.email = email # 更改用户对象 邮箱数据
```

```
# 6.提交保存到数据库    from application import app, db
db.session.add(user_info)
db.session.commit()

return jsonify(resp) # 响应成功

# 使用add_url_rule方法来构造与视图关联的请求url
# 继承自模板的as_view方法可以将类转换为可以为路由注册的视图函数
# as_view('login') --- 传入的是站点名 (标准操作是视图名小写 Login -- login)
#      <Rule '/user/login' (GET, HEAD, OPTIONS) -> user_page.login>] user_page.login中的login就是
as_view指明的站点名
route_user.add_url_rule('/edit', view_func=EditInfo.as_view('edit'))
route_user.add_url_rule('/login', view_func=Login.as_view('login'))
route_user.add_url_rule('/logout', view_func=Logout.as_view('logout'))
```

启动测试 <http://127.0.0.1:8888/user/edit> 修改提交

