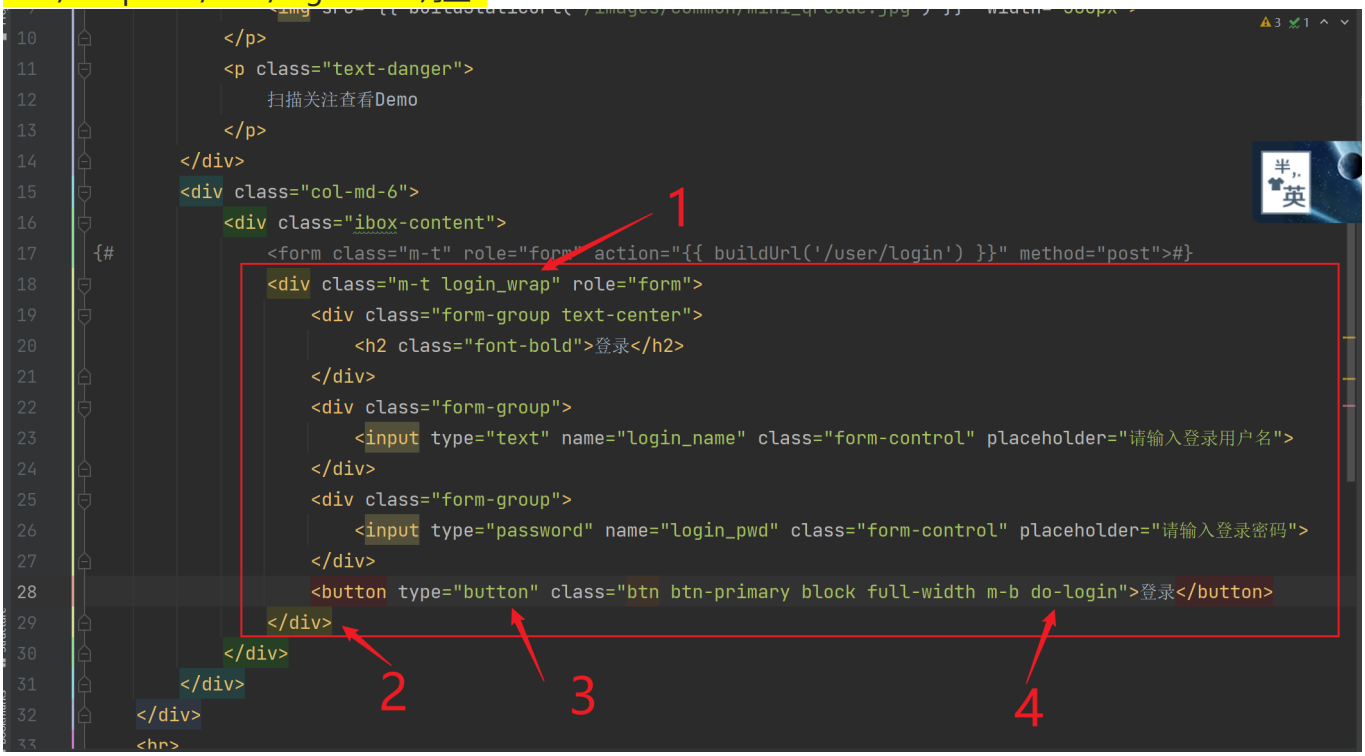
1.调整提交方式

前面我们完成了表单提交数据的方式

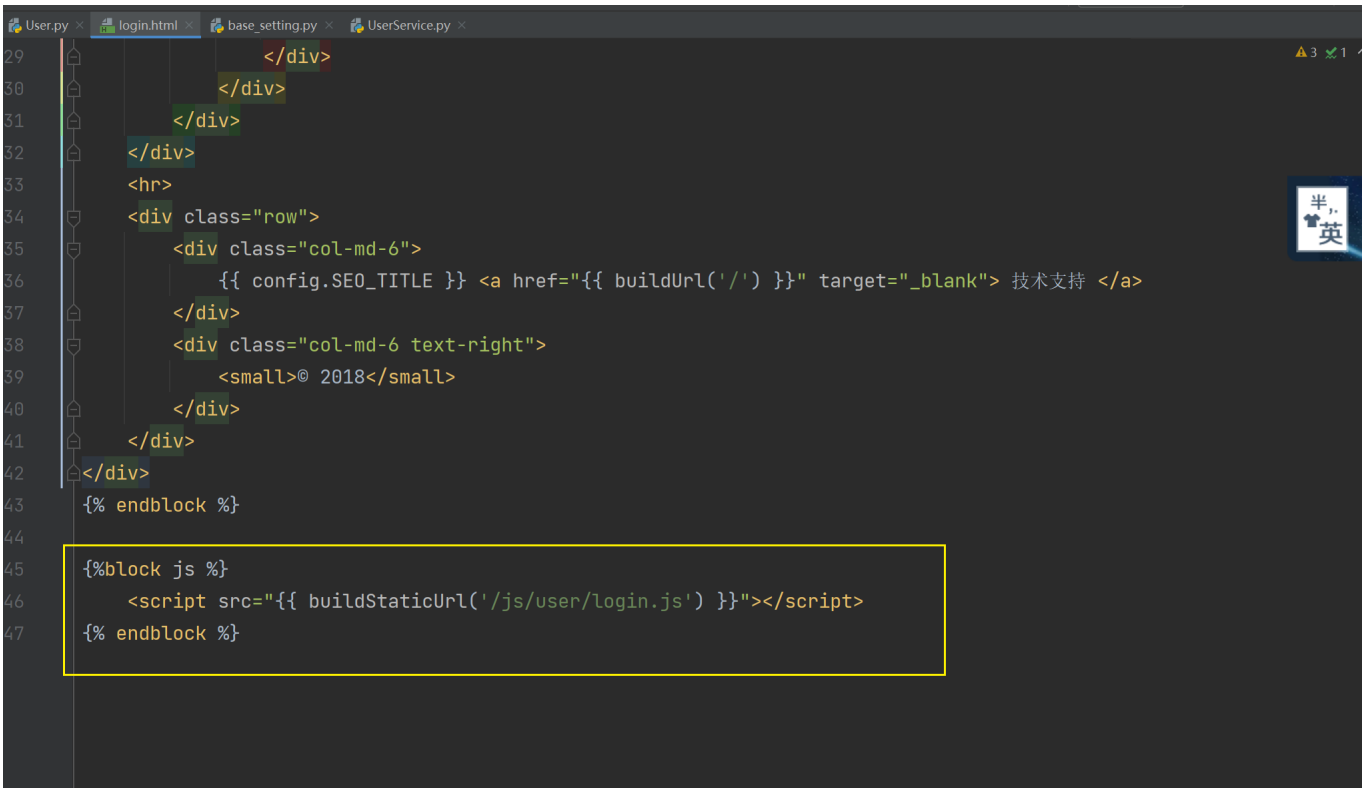
那么在实际开发中表单提交并不理想，我们采用ajax提交更为理想

web/templates/user/login.html调整1



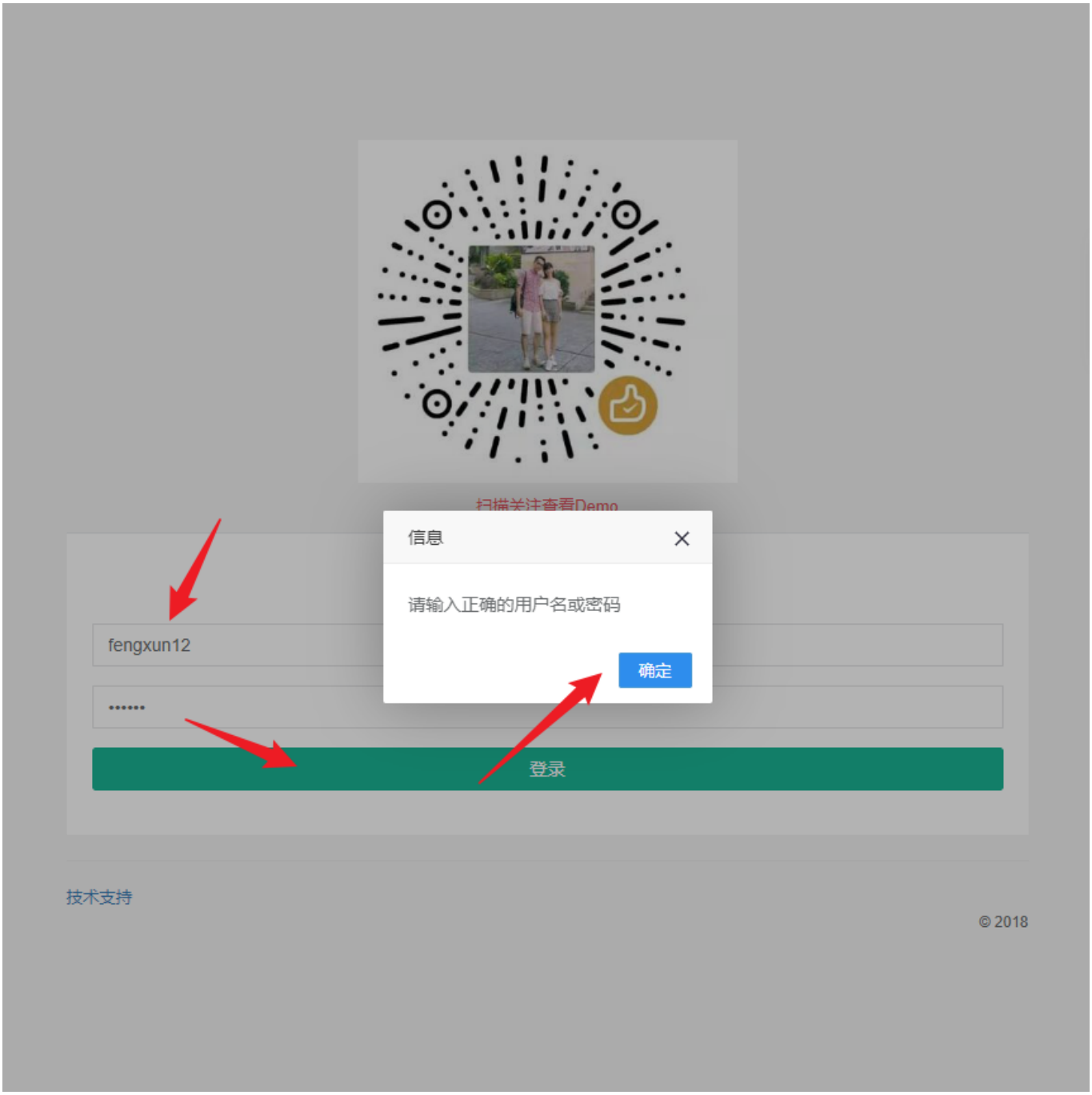

```
            </p>
            <p class="text-danger">
                扫描关注查看Demo
            </p>
        </div>
        <div class="col-md-6">
            <div class="ibox-content">
{#              <form class="m-t" role="form" action="{{ buildUrl('/user/login') }}" method="post">#}
                <div class="m-t login_wrap" role="form">
                    <div class="form-group text-center">
                        <h2 class="font-bold">登录</h2>
                    </div>
                    <div class="form-group">
                        <input type="text" name="login_name" class="form-control" placeholder="请输入登录用户名">
                    </div>
                    <div class="form-group">
                        <input type="password" name="login_pwd" class="form-control" placeholder="请输入登录密码">
                    </div>
                    <button type="button" class="btn btn-primary block full-width m-b do-login">登录</button>
                </div>
            </div>
        </div>
    </div>
    <hr>
```

web/templates/user/login.html调整2



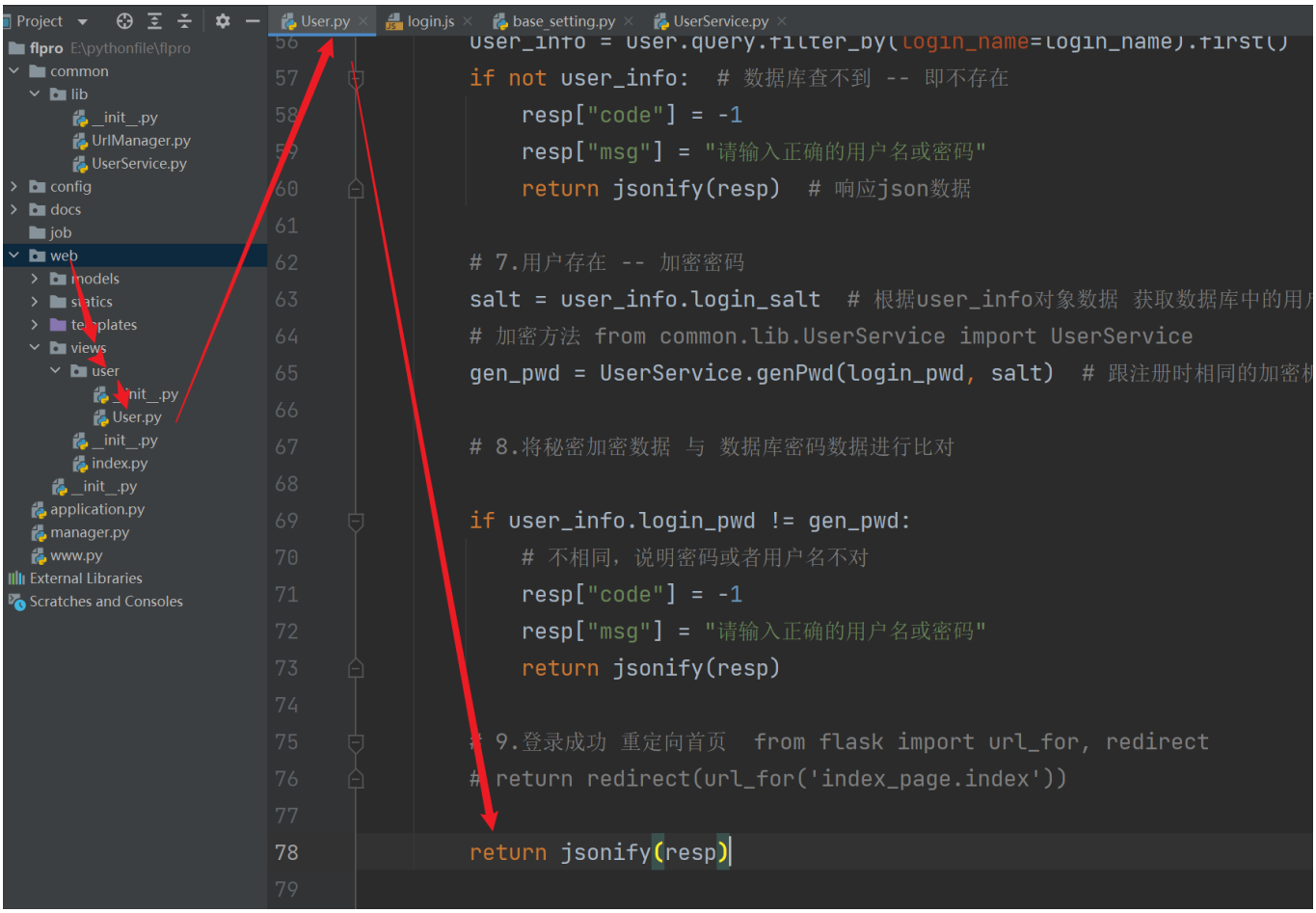
```
User.py  login.html  base_setting.py  UserService.py
                </div>
            </div>
        </div>
    </div>
    <hr>
    <div class="row">
        <div class="col-md-6">
            {{ config.SEO_TITLE }} <a href="{{ buildUrl('/') }}" target="_blank"> 技术支持 </a>
        </div>
        <div class="col-md-6 text-right">
            <small>© 2018</small>
        </div>
    </div>
</div>
{% endblock %}

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

调整后 - 启动项目

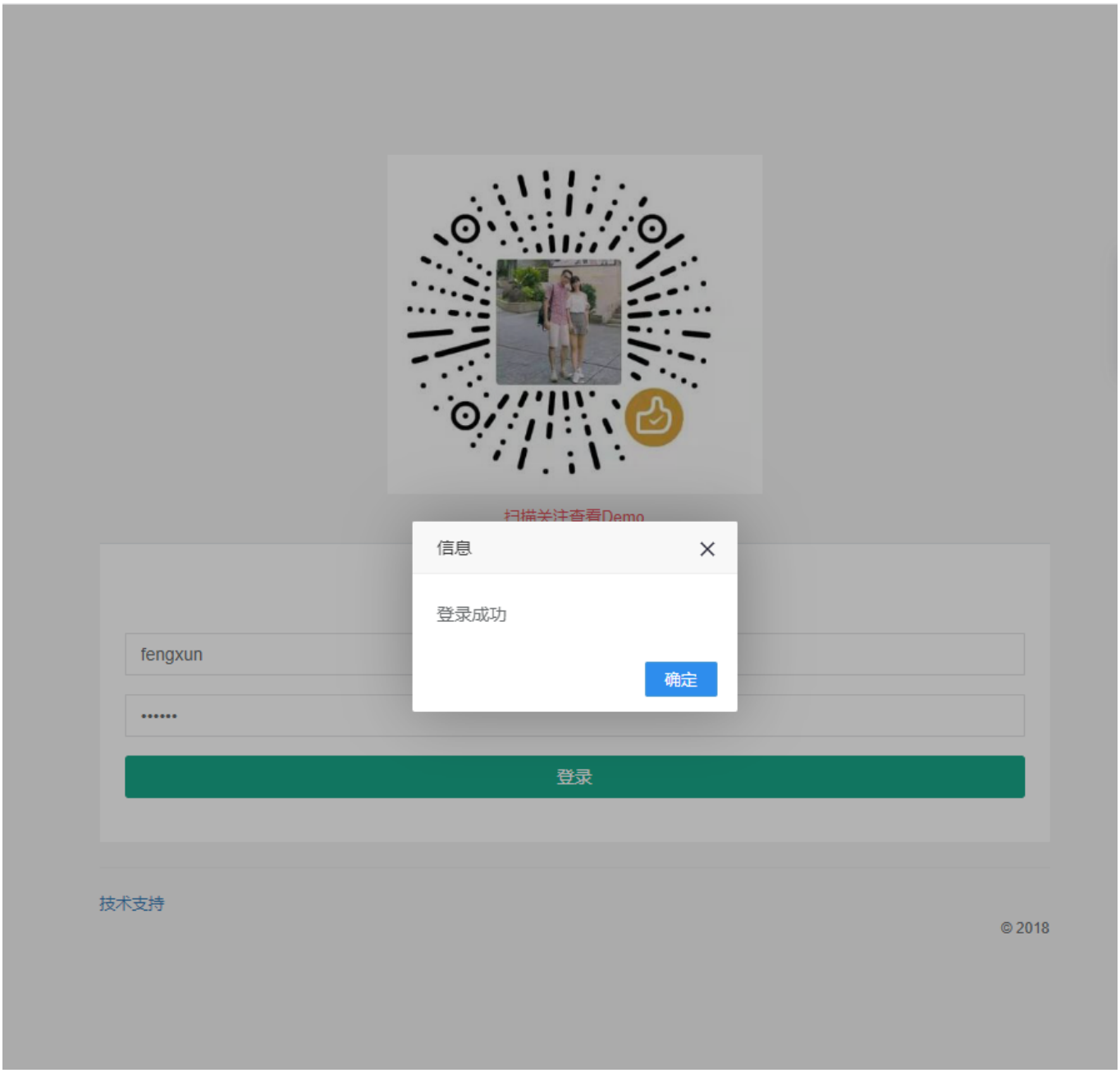


此时就有js的弹窗提示信息

登录成功没反应 --- 需要调整登录视图
因为此时ajax在等待响应，但是我们视图却重定向了，因此衔接不上


Project
flpro E:\pythonfile\flpro
common
lib
__init__.py
UrlManager.py
UserService.py
config
docs
job
web
models
statics
templates
views
user
__init__.py
User.py
__init__.py
index.py
__init__.py
application.py
manager.py
www.py
External Libraries
Scratches and Consoles
User.py
login.js
base_setting.py
UserService.py
user_info = User.query.filter_by(login_name=login_name).first()
if not user_info: # 数据库查不到 -- 即不存在
resp["code"] = -1
resp["msg"] = "请输入正确的用户名或密码"
return jsonify(resp) # 响应json数据
# 7.用户存在 -- 加密密码
salt = user_info.login_salt # 根据user_info对象数据 获取数据库中的用
# 加密方法 from common.lib.UserService import UserService
gen_pwd = UserService.genPwd(login_pwd, salt) # 跟注册时相同的加密
# 8.将秘密加密数据 与 数据库密码数据进行比对
if user_info.login_pwd != gen_pwd:
# 不相同，说明密码或者用户名不对
resp["code"] = -1
resp["msg"] = "请输入正确的用户名或密码"
return jsonify(resp)
9.登录成功 重定向首页 from flask import url_for, redirect
return redirect(url_for('index_page.index'))
return jsonify(resp)

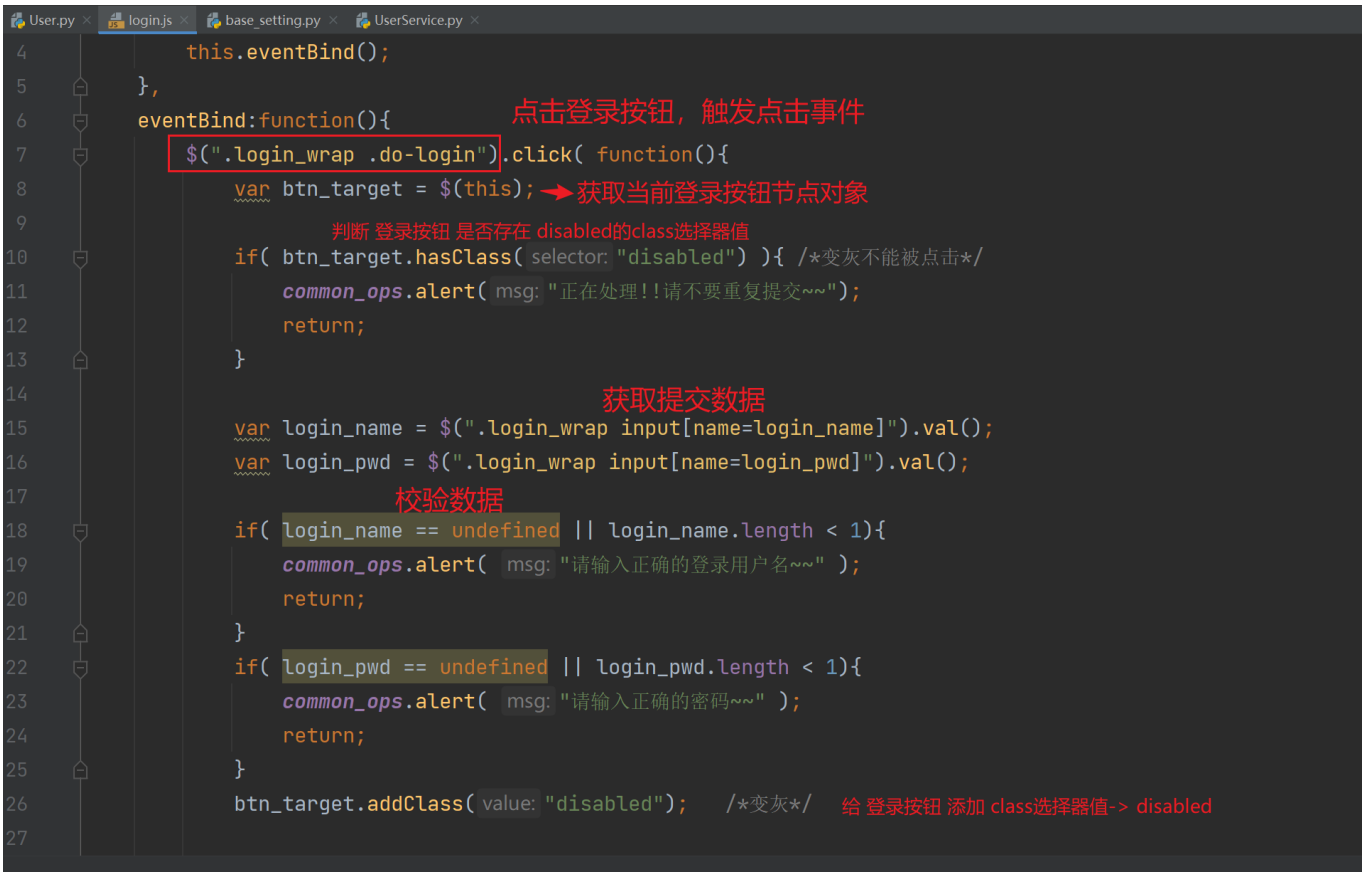
调整完后就好了

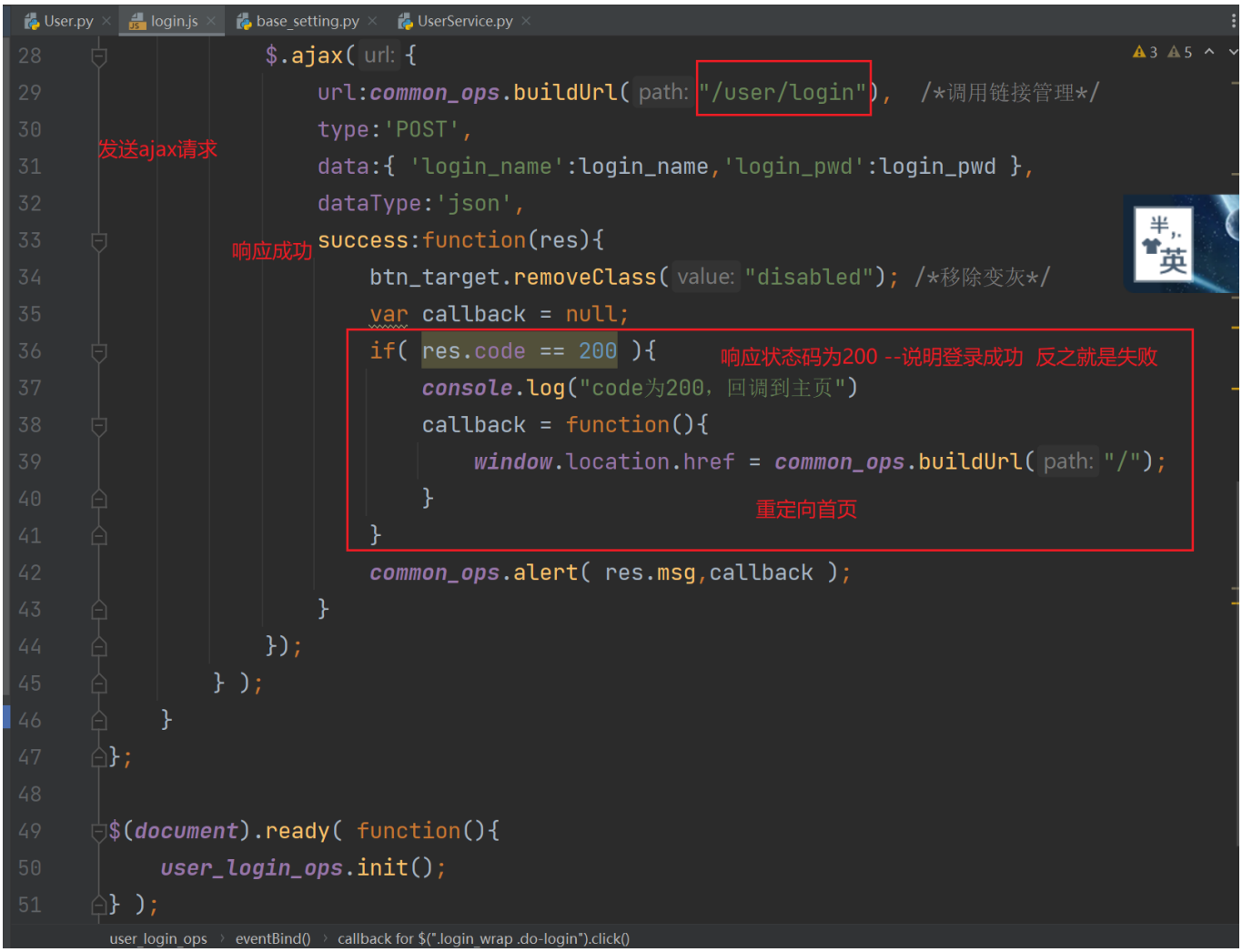


点击确定后 重定向到首页

2.分析js逻辑

web/statics/js/user/login.js





3.设置用户cookie分析

用户cookie数据是为了方便我们使用关键数据操作，减少频繁的数据库操作，
因此我们将用户的关键数据提取出来进行加密封装放入cookie内

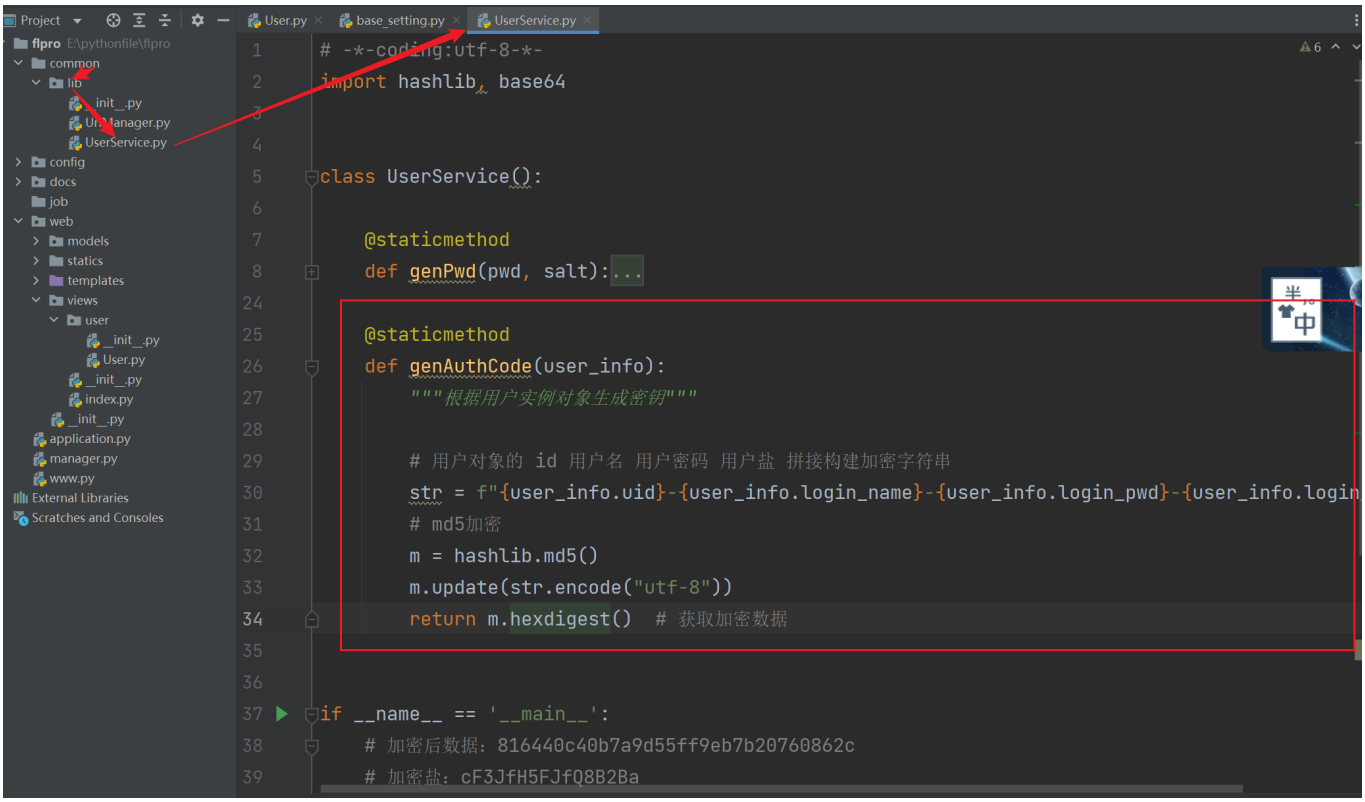
因此需要 设置数据加密基础方法

那么这里为什么一定需要设置用户cookie呢？

因为我们需要完成登录校验（是否处于登录状态）！！！

4.加密方法实现

操作文件 --- common/lib/UserService.py

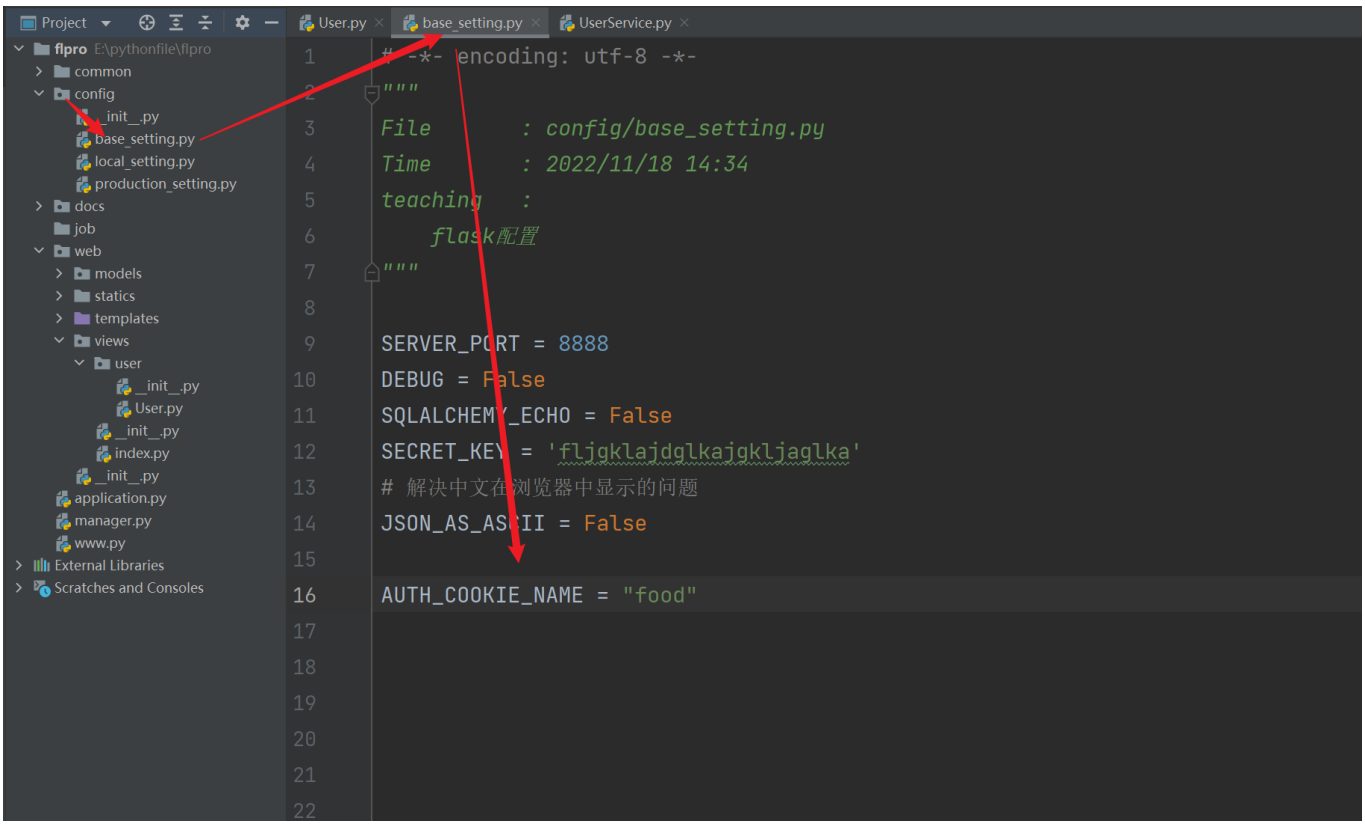



```
# -*-coding:utf-8-*-
import hashlib, base64


class UserService():

    @staticmethod
    def genPwd(pwd, salt):
        """
        密码加密
        :param pwd: 密码
        :param salt: 加密盐
        :return: 加密
        """

        # base64加密
        pwd_base64 = base64.encodebytes(pwd.encode("utf-8"))  # 将 密码 bs64加密
```

```
        data_str = f"{pwd_base64}-{salt}"  # 拼接 加密密码数据 与盐拼接

        # md5加密
        m = hashlib.md5()  # 实例md5加密对象
        m.update(data_str.encode("utf-8"))  # 加密
        return m.hexdigest()  # 获取加密数据

    @staticmethod
    def genAuthCode(user_info):
        """
        根据用户实例对象生成密钥
        :param user_info: 用户对象
        :return: 加密后的用户数据
        """

        # 用户对象的 id 用户名 用户密码 用户盐 拼接构建加密字符串
        # 至于用户密码，用户盐这类敏感数据就不建议进行放入cookie了，避免数据外泄，此处仅做教学演示
        str = f"{user_info.uid}-{user_info.login_name}-{user_info.login_pwd}-{user_info.login_salt}"
        # md5加密
        m = hashlib.md5()
        m.update(str.encode("utf-8"))
        return m.hexdigest()  # 获取加密数据
```

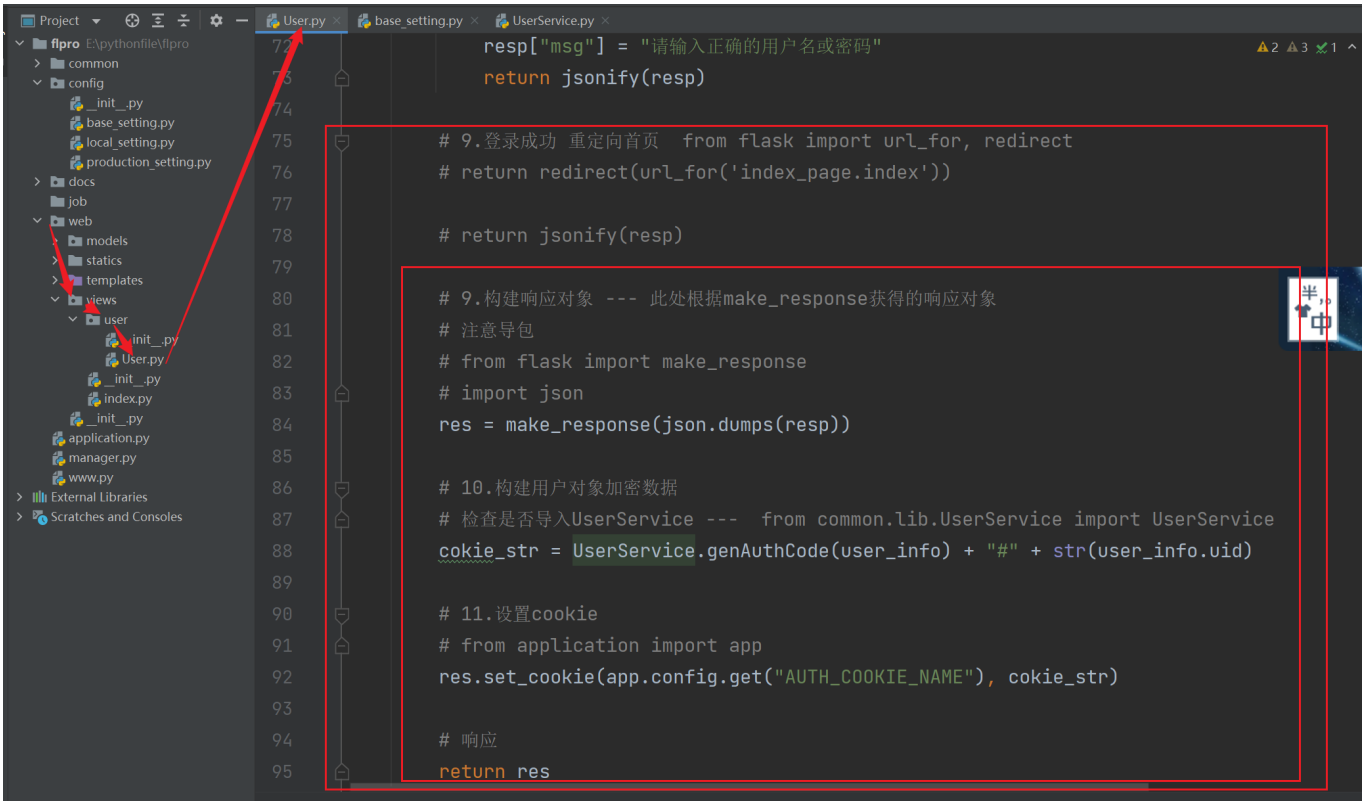
5.设置通用AUTH_COOKIE_NAME配置

操作文件位置：config/base_setting.py




```
AUTH_COOKIE_NAME = "food"
```

6.用户cookie设置




```
# -*- 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
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__)


# 登录视图
# 继承MethodView类可以重载，get/post/delete/put的请求方法。来实现针对某一类请求的视图接收。
class Login(views.MethodView):
    def get(self):
        return render_template("user/login.html")

    def post(self):

        # 1.注意导入请求上下文 request from flask import request

        # 2.获取form表单数据
        data = request.values

        # 3.获取用户名及密码
```

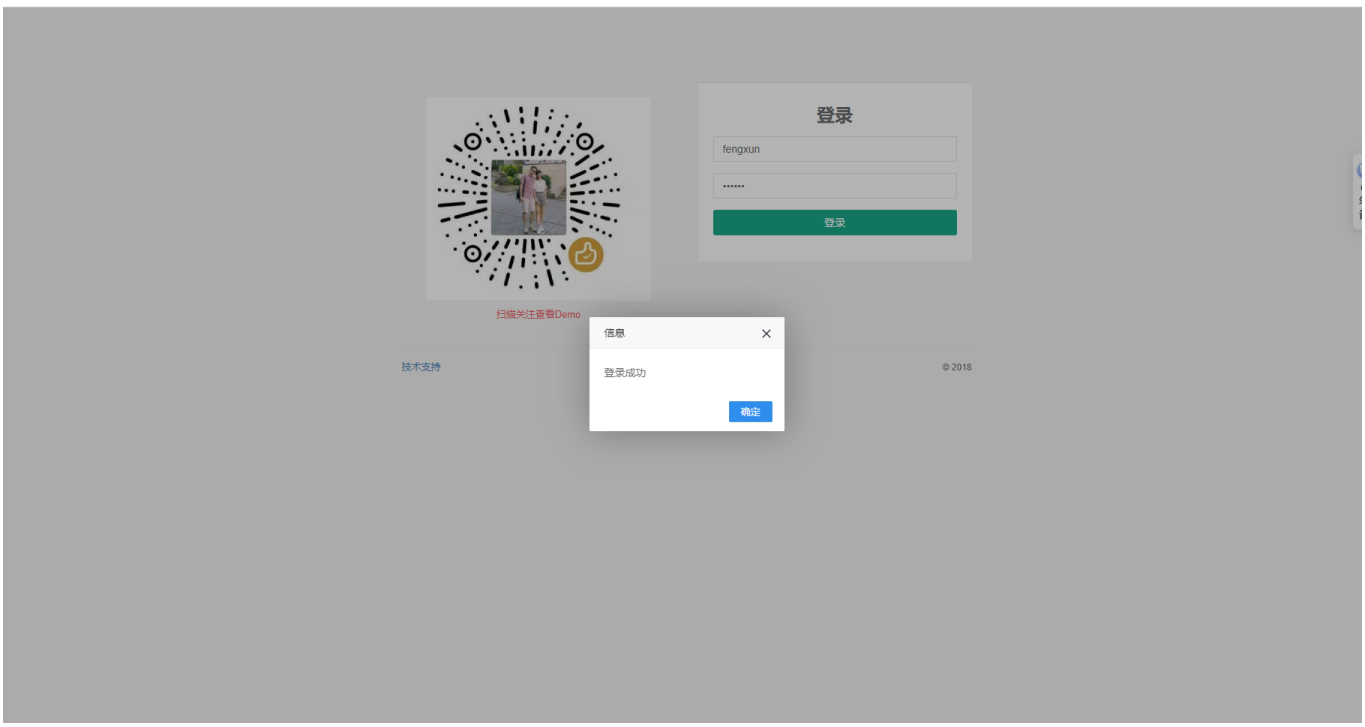
```
    login_name = data.get('login_name')
    login_pwd = data.get('login_pwd')

    # 4.构建响应数据对象
    resp = {
        "code": 200,  # 状态码
        "msg": "登录成功",  # 响应信息
        "data": {}  # 响应数据
    }

    # 5.判断数据是否获取及其是否满足要求
    if (not login_name) or len(login_name) < 1:
        # 用户名没有数据 或者 用户名长度小于1
        resp["code"] = -1
        resp["msg"] = "请输入正确的登录用户名"
        # 导入 jsonify --- from flask import jsonify
        return jsonify(resp)

    if (not login_pwd) or len(login_pwd) < 1:
        # 密码没有数据 或者 密码长度小于1
        resp["code"] = -1
        resp["msg"] = "请输入正确的密码"
        return jsonify(resp)  # 响应json数据

    # 6.根据用户名查询用户是否存在于数据库
    user_info = User.query.filter_by(login_name=login_name).first()
    if not user_info:  # 数据库查不到 -- 即不存在
        resp["code"] = -1
        resp["msg"] = "请输入正确的用户名或密码"
        return jsonify(resp)  # 响应json数据

    # 7.用户存在 -- 加密密码
    salt = user_info.login_salt  # 根据user_info对象数据 获取数据库中的用户密码 加密盐
    # 加密方法 from common.lib.UserService import UserService
    gen_pwd = UserService.genPwd(login_pwd, salt)  # 跟注册时相同的加密机制 -- 获得加密数据

    # 8.将秘密加密数据 与 数据库密码数据进行比对

    if user_info.login_pwd != gen_pwd:
        # 不相同，说明密码或者用户名不对
        resp["code"] = -1
        resp["msg"] = "请输入正确的用户名或密码"
        return jsonify(resp)

    # 9.登录成功 重定向首页  from flask import url_for, redirect
    # return redirect(url_for('index_page.index'))

    # return jsonify(resp)

    # 9.构建响应对象 --- 此处根据make_response获得的响应对象
    # 注意导包
    # from flask import make_response
    # import json
    res = make_response(json.dumps(resp))

    # 10.构建用户对象加密数据
    # 检查是否导入UserService ---  from common.lib.UserService import UserService
    cokie_str = UserService.genAuthCode(user_info) + "#" + str(user_info.uid)
```

```
        # 11.设置cookie
        # from application import app
        # app.config.get("AUTH_COOKIE_NAME") 获取配置中配置的通用AUTH_COOKIE_NAME cookie键名
        res.set_cookie(app.config.get("AUTH_COOKIE_NAME"), cokie_str)

        # 响应
        return res


# 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('/login', view_func=Login.as_view('login'))
```

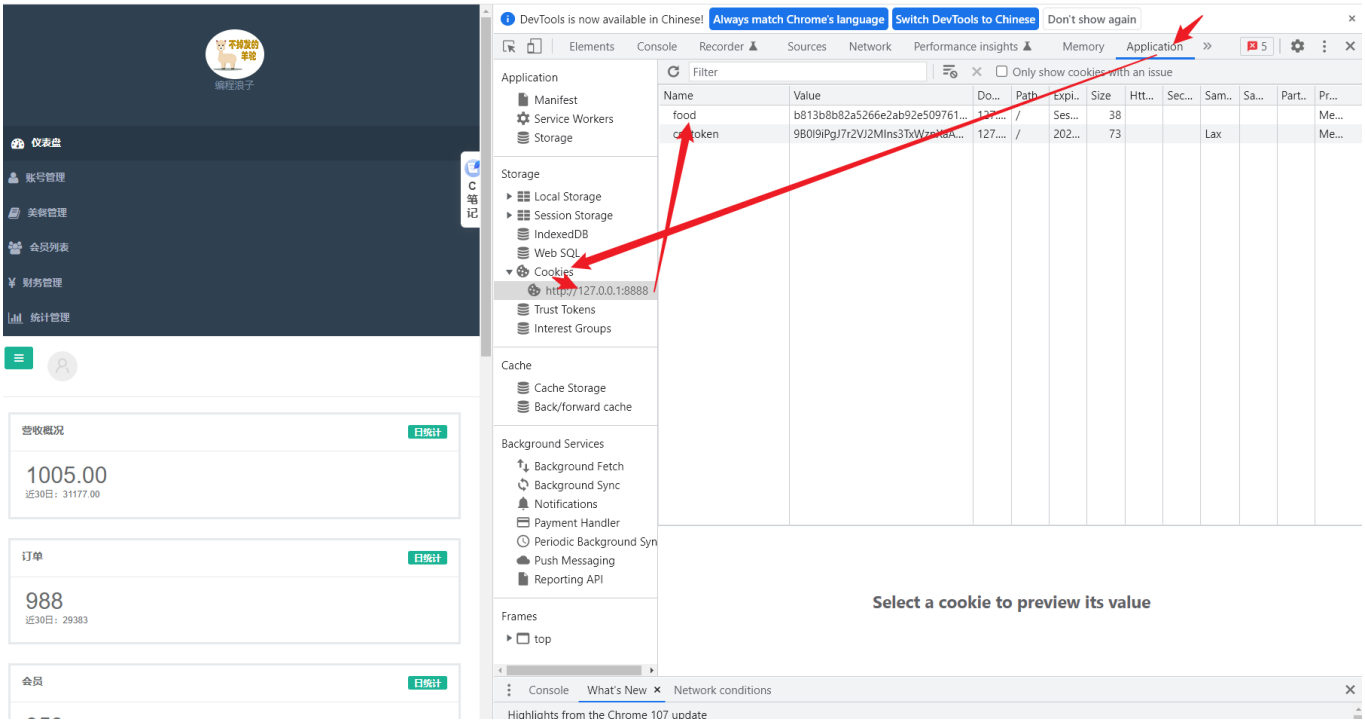
启动 --- 访问登录 --- 正确登陆成功

http://127.0.0.1:8888/user/login

用户：fengxun 密码：123456



点击确定 --- 回到首页 - 查看cookie

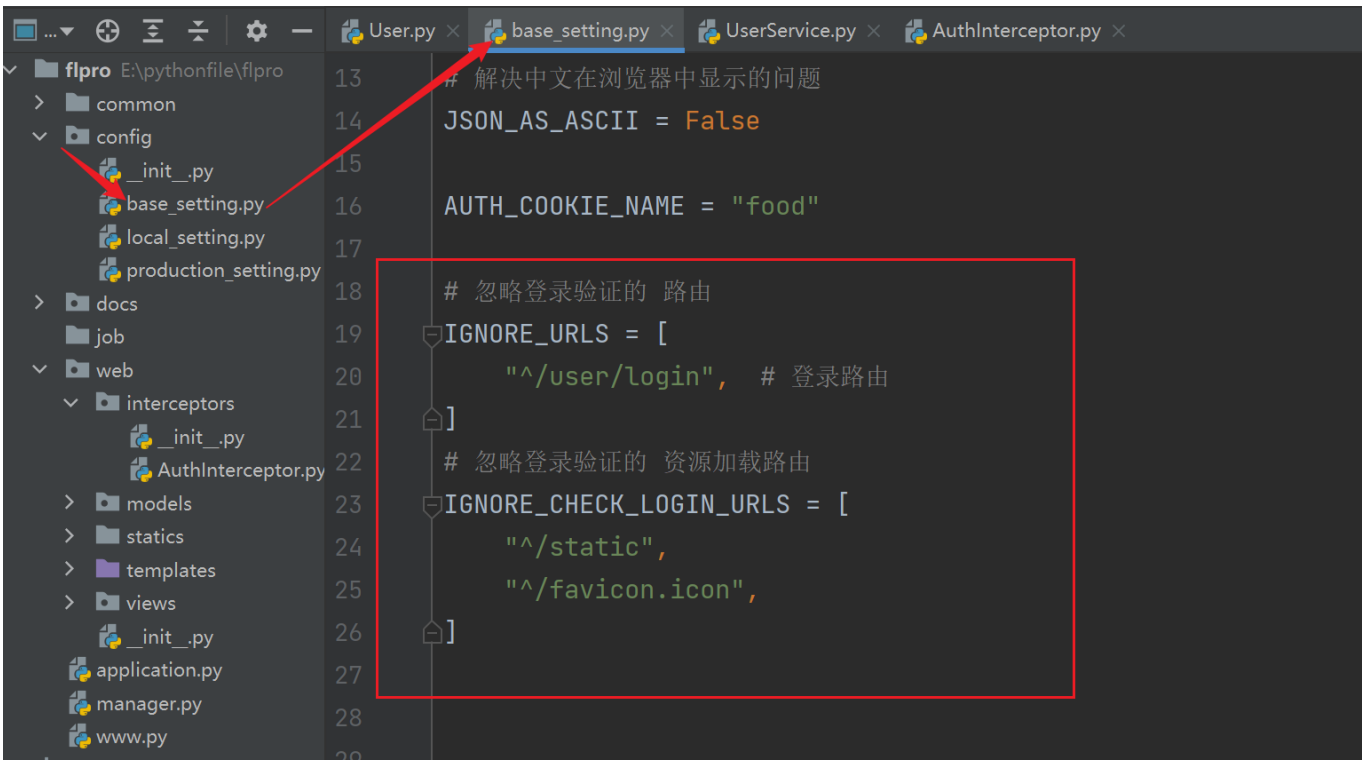


7.用户登录验证

验证是否处于登录状态

1.根据前面设置的用户cookie，对其数据进行比对校验
2.用户登录验证需要在每次请求都需要进行，也就代表着我们需要完成 中间件功能（flask的钩子函数完成 拦截器）

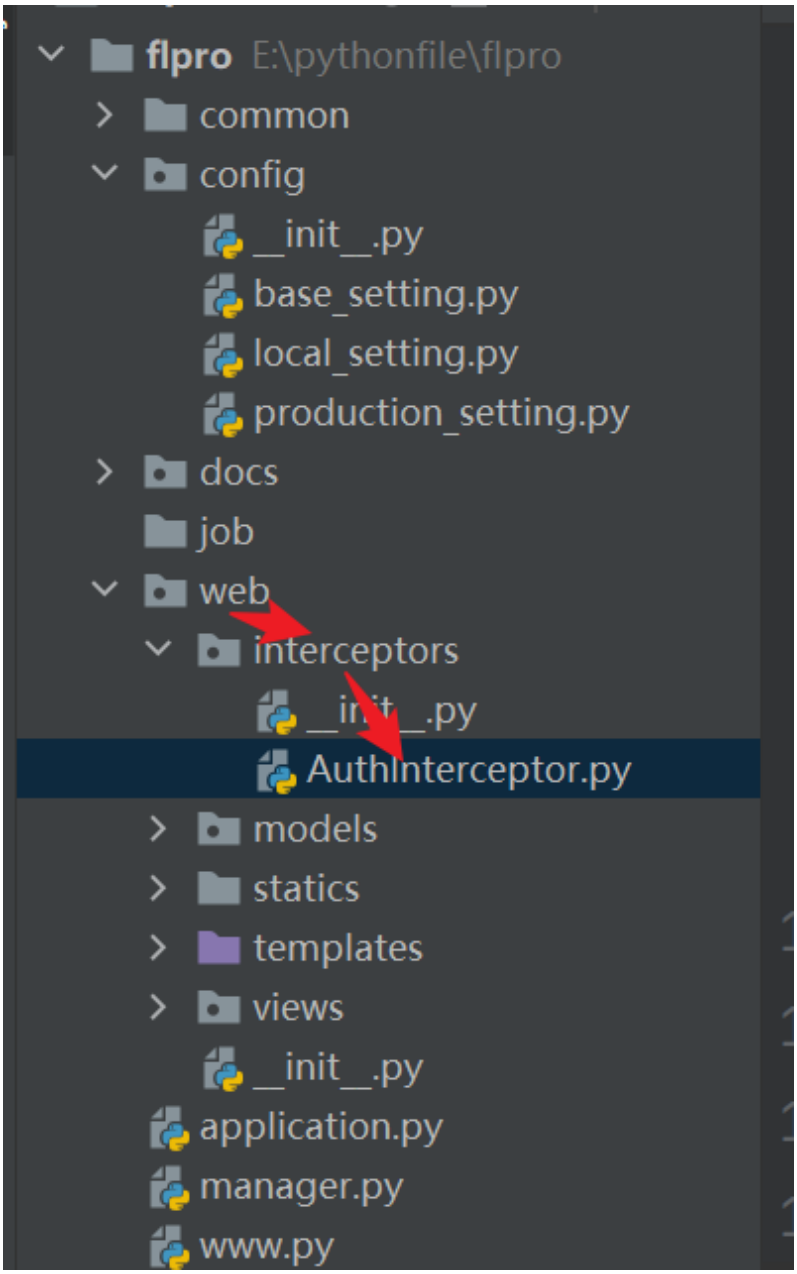
先设置忽略验证的配置




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

```
File       : config/base_setting.py
Time       : 2022/11/18 14:34
teaching   :
    flask配置
"""

SERVER_PORT = 8888
DEBUG = False
SQLALCHEMY_ECHO = False
SECRET_KEY = 'fljgklajdglkajgkljaglka'
# 解决中文在浏览器中显示的问题
JSON_AS_ASCII = False

AUTH_COOKIE_NAME = "food"

# 忽略登录验证的 路由
IGNORE_URLS = [
    "^/user/login",  # 登录路由
]
# 忽略登录验证的 资源加载路由
IGNORE_CHECK_LOGIN_URLS = [
    "^/static",
    "^/favicon.icon",
]
```

`web目录下建立interceptors目录， 在interceptors目录内建立AuthInterceptor.py文件`




```
# -*- encoding: utf-8 -*-
"""
File       : AuthInterceptor.py
teaching   :
    拦截器
"""
from application import app
from flask import request, redirect
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。
```

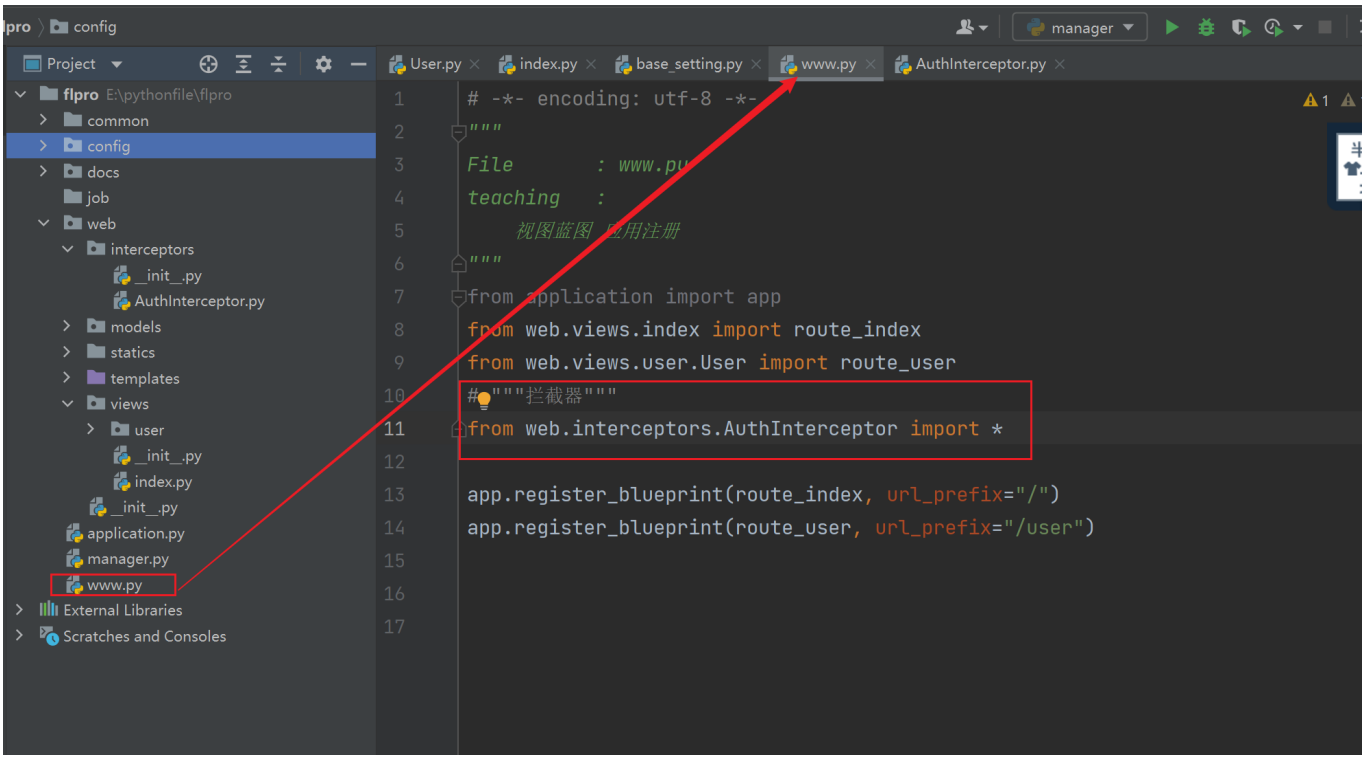
```
    # match() --- 向它传入要匹配的字符串，以及正则表达式，就可以检测这个正则表达式是否匹配字符串。
    # >>> import re
    # >>> a = ['123', 'hahhah', 'shuai']
    # >>> pattern = re.compile("|".join(a))
    # >>> pattern
    # >>> re.compile('123|hahhah|shuai')
    # >>> pattern.match('1')
    #
    # >>> pattern.match('123')
    #     <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

    # 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
```

Name	Value	Domain	Path	E
food	b813b8b82a5266e2ab92e50976126dc2#1	127.0.0.1	/	S

8.拦截器导入（钩子引用）

注意，拦截器（钩子）定义好后，在蓝图中我们需要在蓝图注册模块中导入定义的拦截器（钩子）模块

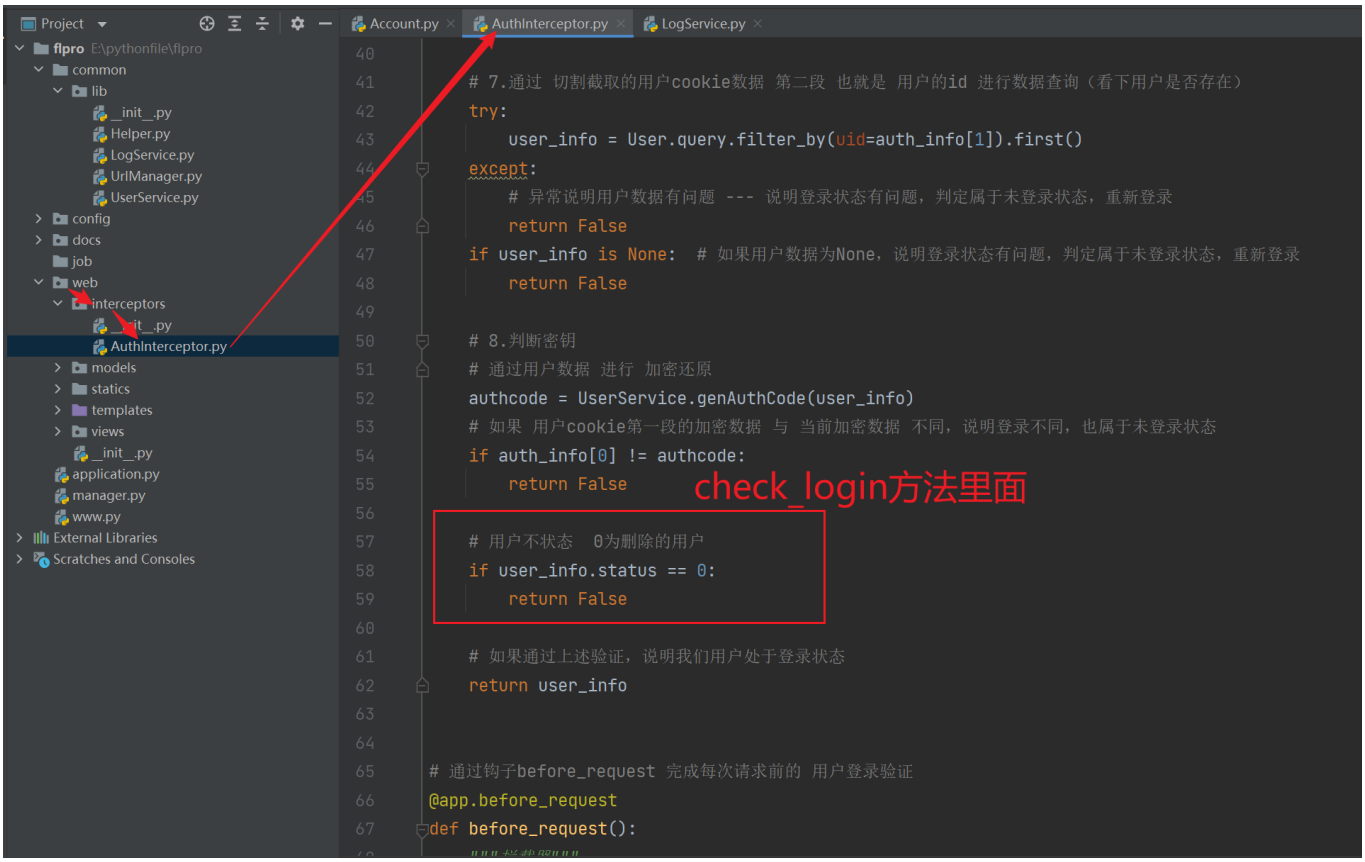
否者拦截器（钩子）不生效




```
# -*- encoding: utf-8 -*-
"""
File       : www.py
teaching   :
    视图蓝图 应用注册
"""
from application import app
from web.views.index import route_index
from web.views.user.User import route_user
# """拦截器"""
from web.interceptors.AuthInterceptor import *

app.register_blueprint(route_index, url_prefix="/")
app.register_blueprint(route_user, url_prefix="/user")
```

启动项目后，访问首页 `http://127.0.0.1:8888/`
在未登录状态（未设置用户cookie）情况下将会重定向到登录页

9.登录验证优化 -- 删除用户不能登录


Project
flpro E:\pythonfile\flpro
common
lib
__init__.py
Helper.py
LogService.py
UrlManager.py
UserService.py
config
docs
job
web
interceptors
__init__.py
AuthInterceptor.py
models
statics
templates
views
__init__.py
application.py
manager.py
www.py
External Libraries
Scratches and Consoles
Account.py
AuthInterceptor.py
LogService.py
# 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
check_login方法里面
# 用户不状态 0为删除的用户
if user_info.status == 0:
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():