Chapter3 Web Forms

1. Flask-WTF

Flask-WTF是用来处理程序中的form表单的扩展，基于 WTForms, 通过下列命令安装；

```
pip install flask-wtf
```

2. 配置

Flask提供灵活多变的配置方式，最基本的解决方法是在app.config中定义一个key-value的键值对，如:

```
# ...
app = FLASK(__name__)
app.config['SECRET_KEY'] = 'you-will-never-guess'
# ...
```

更好地方式是创建一个专门管理配置的文件：

```
# config.py
import os

class Config(object):
    SECRET_KEY = os.environ.get('SECRET_KEY') or 'you-will-never-guess'
```

上述代码配置了一个名为SECRET_KEY的变量，Flask与其他的拓展库就可以以该变量作为cryptographic key去生成签名或者令牌。Flask-WTF使用该配置去对抗Cross-Site Request Forgery or CSRF 攻击。

下列代码导入该配置：

```
# app/__init__.py: Flask configuration
from flask import Flask

from config import Config
# config.py    Config ==> class


app = Flask(__name__)
app.config.from_object(Config)

from app import routes
```

接下来，就可以在程序中使用该配置，例如在Python交互环境下

```
>>> from microblog import app
>>> app.config['SECRET_KEY']
'you-will-never-guess'
```

3. User Login Form

Flask-WTF使用Python类来表示web forms. 一个form类将form表单中的数据域定义为其类属性。

根据一直遵循的分离原则，创建一个单独*app/forms.py*模块类保存登陆表单:

```
# app/forms.py: Login from
from flask_wtf import FlaskForm
from wtforms import StringField, PasswordField, BooleanField, SubmitField
from wtforms.validators import DataRequired


class LoginForm(FlaskForm):
    username = StringField('用户名', validators=[DataRequired('请输入用户名')])
    password = PasswordField('密码', validators=[DataRequired('请输入密码')])
    remember_me = BooleanField('记住我')
    submit = SubmitField('登陆')

# validators参数：数据验证
# DataRequired验证器,进行简单的验证(必填项)
```

4. Form Templates

在页面上看到的输入表单代码写在HTML模板中，Flask-WTF可以将定义在*LoginForm*类中的属性渲染进HTML。创建 *app/templates/login.html*

```
<!-- app/templates/login.html -->
{% extends "base.html" %}

{% block content %}
    <h1>注册</h1>
    <form action="" method="post">
        {{ form.hidden_tag() }}
        <p>
            {{ form.username.label }}<br>
            {{ form.username(size=32) }}
        </p>
        <p>
            {{ form.password.label }}<br>
            {{ form.password(size=32) }}
        </p>
        <p>{{ form.remember_me() }} {{ form.remember_me.label }}</p>
```

```
        <p>{{ form.submit() }}</p>
    </form>
{% endblock %}
```

4.1 重用*base.html*

通过*extends*模板重用语句，重用(继承)*base.html*模板.

4.2 form 变量

*login.htm*与*LoginForm*之间的桥梁就是form变量，这个form变量哪里来的呢?是通过后续代码建立联系的。

4.3 *form.hidden_tag()*

form.hidden_tag() 生成一个隐藏的表单域，包含了一个token，用来抵御上文提到的CSRF攻击。

使用Flask-WTF解决CSRF问题只需要完成：

- 定义一个*SECRET_KEY* 配置
- 表单中通过*form.hidden_tag()* 生成一个隐藏域

4.4 *{{form.field_name.label...}}*

在页面代码中出现了 *{{form.field_name.label}}* , *{{form.<field_name>()}}* 等语句，与之前写*HTML form*完全不同。

这是因为Flask-WTF会在渲染时可以自动将*LoginForm*中定义的属性渲染为*form*表单的数据域。

5. Form Views

编写新的路由请求处理函数，以此来响应用户的登陆请求。用户访问 */login* 路径提交登陆请求

```
# app/routes.py

from flask import render_template
from app import app
from app.forms import LoginForm

# ...

@app.route('/login')
def login():
    form = LoginForm()
    return render_template('login.html', title='登陆', form=form)
    # 渲染模板时，将LoginForm类的实例传入到模板中
    # 建立了LoginForm类与前端页面模板的桥梁，login.html中就可以使用form变量
```

重构*base.html*页面，添加导航页面

```
    <div>
        微博客:
        <a href="/index">首页</a>
        <a href="/login">登陆</a>
    </div>
```

访问页面，查看效果

6. Receiving Form Data

点击*登陆* 按钮，肯定会得到一个错误。这是因为只是完成了页面渲染显示的代码，登陆的逻辑处理代码暂时还没有完成。下面就完成这部分工作。重构*app/routes.py* 代码:

```
# app/routes.py

from flask import render_template, flash, redirect
# ...

@app.route('/login', methods=['GET', 'POST']) # GET/POST请求，都是该函数处理
def login():
    form = LoginForm()
    if form.validate_on_submit():            # GET时，该函数值为false,直接返回模板渲染函数; # POST时为true
        flash('登陆的用户名为: {}'.format(form.username.data))
        return redirect('/index')
    return render_template('login.html', title='登陆', form=form)

# ...
```

*form.validate_on_submit() 函数完成所有*form表单数据处理工作：

- GET 请求时，该函数返回False
- POST请求时，该函数返回True, 调用*flash* 函数, 向前端模板发送一条提示信息。(前端页面不会神奇的直接显示该消息，需要编写代码)
- 注意，可以使用*form.username.data* 取到表单提交到server端的数据；
- *redirect* 函数，重定向, 登陆完成，跳转到首页 */index* .

7. Flashed messages

更新*base.html* , 增加显示登陆消息

```
<!--  app/templates/base.html: Flashed messages in base template -->
<html lang="en">
<head>
    <meta charset="UTF-8">
    <meta name="viewport" content="width=device-width, initial-scale=1.0">
    <meta http-equiv="X-UA-Compatible" content="ie=edge">
    {% if title %}
```

```
    <title>{{ title }} - 微博客</title>
    {% else %}
    <title>欢迎访问微博客!</title>
    {% endif %}
</head>
<body>
    <div>微博客:
        <a href="/index"">首页</a>
        <a href="/login">登陆</a>
    </div>
    <hr>

    {% with messages = get_flashed_messages() %}
    {% if messages %}
        <ul>
            {% for message in messages %}
                <li>
                    {{ message }}
                </li>
            {% endfor %}
        </ul>
    {% endif %}
    {% endwith %}

    {% block content %} {% endblock %}
</body>
</html>
```

- *get_flashed_messages()* 来自Flash框架，返回server端使用*flash()* 函数发送的消息list，使用with结构将该函数返回结果赋值给*messages* 。该函数有个特性，调用一次就可以将消息全部取出；
- 使用*ul li* 结构将flash message全部显示。

再次运行程序，查看结果。

8. Improving Field Validation

之前在*LoginForm* 上附加的数据验证器(*DataRequired*)并没有发挥作用。改验证器可以将错误输入数据显示在页面上以提醒用户。

如果*用户名, 密码* 表单域没有输入，直接点击*注册* 按钮，提示信息已经由数据验证器产生，只是没有显示出来而已。

更新前端模板页面*app/templates/login.html*

```
<!-- app/templates/login.html: Validation errors in login form template -->

{% extends "base.html" %}

{% block content %}

    <h1>注册</h1>
```

```
    <form action="" method="post">
        {{ form.hidden_tag() }}
        <p>
            {{ form.username.label }} <br>
            {{ form.username(size=32) }}
            {% for error in form.username.errors %}
                <span style="color:red">[{{ error }}]</span>
            {% endfor %}
        </p>
        <p>
            {{ form.password.label }} <br>
            {{ form.password(size=32) }}
            {% for error in form.password.errors %}
                <span style="color:red">[{{ error }}]</span>
            {% endfor %}
        </p>
        <p>
            {{ form.remember_me() }}
            {{ form.remember_me.label}}
        </p>
        <p>
            {{ form.submit() }}
        </p>
    </form>

{% endblock %}
```

- 在*username , passowrd* 表单域后面增加循环结构，将数据验证器产生的错误数据渲染在相应表单域后面;
- *form.<field_name>.errors* 属性保存了验证器产生的错误消息, 结构是一个*list*.

9. Generating Links

在*Templates*与*redirect*中使用链接的方式，不应该是硬编码，如下：

```
    <div>
        微博客:
        <a href="/index">Home</a>
        <a href="/login">Login</a>
    </div>
```

合理的链接生成方式应该使用Flask框架提供的*url_for()* 函数，例如：*url_for('login') 返回 /login, and url_for('index') 返回 '/index'*

更新*app/templates/base.html* 与*app/routes.py*

```
<!-- app/templates/base.html -->
    <div>微博客:
        <a href="{{ url_for('index') }}">首页</a>
```

```
        <a href="{{ url_for('login') }}">登陆</a>
    </div>
```

```
# app/routes.py

from flask import render_template, flash, redirect, url_for

@app.route('/login', methods=['GET', 'POST'])
def login():
    form = LoginForm()
    if form.validate_on_submit():
        flash('登陆的用户名为: {}'.format(form.username.data))
        return redirect(url_for('index'))
    return render_template('login.html', title='登陆', form=form)
```

10. enjoy it