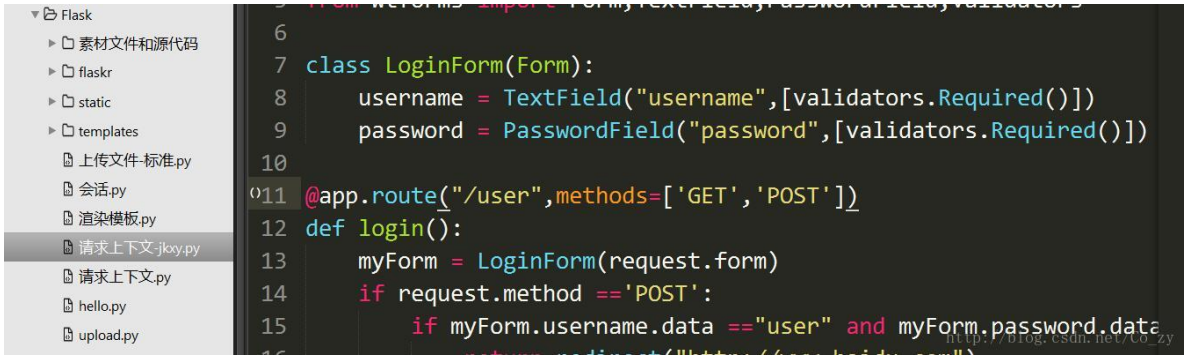
表单的处理

Python Flask-表单提交方式

这篇文章讲两种表单提交方式,先说一下目录树,下图左侧



templates文件夹放置html文件,
static文件夹放置css,js文件.

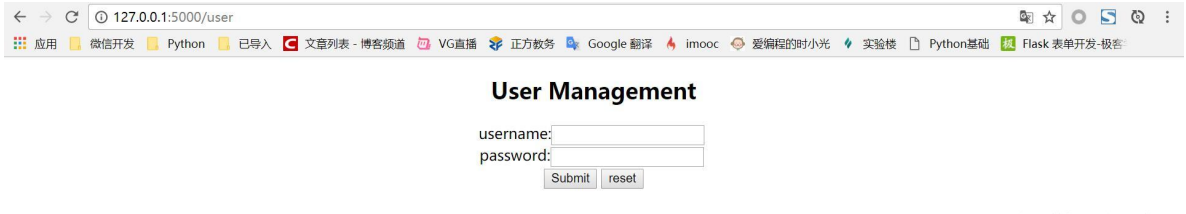
1.请求上下文

首先在templates文件夹新建一个login.html

```
<!DOCTYPE html>
<html lang="en">
<head>
    <meta charset="UTF-8">
    <title>Document</title>
</head>
<body>
<div align="center">
<h2>User Management</h2>
    {% if message %} {{message}} {% endif %}
    <form method="POST">

        <input type="text" name="username" placeholder="username">
        <br>
        <input type="password" name="password" placeholder="password">
        <br>
        <input type="submit" value="Submit">
        <input type="reset" value="reset">
    </form>
</div>

</body>
</html>
```



```
{% if message %} {{message}} {% endif %}用于输出登录失败时的错误信息,在form标签中添加提交方式<form method="POST">
```

然后新建一个login.py

```
from flask import Flask,request,render_template,redirect

app = Flask(__name__)
//绑定访问地址127.0.0.1:5000/user
@app.route("/user",methods=['GET','POST'])
def login():
    if request.method =='POST':
        username = request.form['username']
        password = request.form['password']
        if username =="user" and password=="password":
            return redirect("http://www.baidu.com")
        else:
            message = "Failed Login"
            return render_template('login1.html',message=message)
    return render_template('login1.html')

if __name__ == '__main__':
    app.run(debug=True)
```

2.WTForm

这里对上面login1.html和login.py修改

```
<!DOCTYPE html>
<html lang="en">
<head>
    <meta charset="UTF-8">
    <title>Document</title>
</head>
<body>
<div align="center">
<h2>User Management</h2>
    {% if message %} {{message}} {% endif %}
    <form method="POST">
        username:{{form.username}}
        <br>
        password:{{form.password}}
        <br>
        <input type="submit" value="Submit">
        <input type="reset" value="reset">
    </form>
</div>

</body>
</html>
```

这里直接去掉了输入的input标签,换成了

```
username:{{form.username}},
password:{{form.password}} (jinja2模板)
```

```
from flask import Flask,request,render_template,redirect
app = Flask(__name__)
from wtforms import Form,TextField,PasswordField,validators
class LoginForm(Form):
    username = TextField("username",[validators.Required()])
    password = PasswordField("password",[validators.Required()])
@app.route("/user",methods=['GET','POST'])
def login():
    myForm = LoginForm(request.form)
    if request.method =='POST':
        if myForm.username.data =="user" and myForm.password.data=="password"
and myForm.validate():
            return redirect("http://www.baidu.com")
        else:
            message = "Failed Login"
            return render_template('login1.html',message=message,form=myForm)
    return render_template('login1.html',form=myForm)
if __name__ == '__main__':
    app.run(debug=True)
```

(1)from wtforms import Form,TextField,PasswordField,validators
从wtforms 导入Form类,所有自定义的表单都需要继承这个类,比如

```
class LoginForm(Form):
```

(2)一系列的Field对应html的input标签控件,validators是验证器,用于验证用户输入的数据.

```
myForm.password.data
```

以上代码用于获取用户输入的密码,这里验证中多了myForm.validate():

(3)创建一个对象,[validators.Required()]表明这个值必须要输入

```
username = TextField("username",[validators.Required()])
```

(4)在最后还需将myForm传入form

```
return render_template('login1.html',form=myForm)
```

python轻量框架--Flask(web 表单详细版)

1.回顾

在上一章节中，我们定义了一个简单的模板，使用占位符来虚拟了暂未实现的部分，比如用户以及文章等。

在本章我们将要讲述应用程序的特性之一–表单，我们将会详细讨论如何使用 web 表单。

2.配置

为了能够处理 web 表单，我们将使用 Flask-WTF ， 该扩展封装了 WTForms 并且恰当地集成进 Flask 中。

许多 Flask 扩展需要大量的配置，因此我们将要在 microblog 文件夹的根目录下创建一个配置文件以至于容易被编辑。这就是我们将要开始的(文件 config.py):为了能够处理 web 表单，我们将使用 Flask-WTF ，该扩展封装了 WTForms 并且恰当地集成进 Flask 中。

许多 Flask 扩展需要大量的配置，因此我们将要在 *myproject* 文件夹的根目录下创建一个配置文件以至于容易被编辑。这就是我们将要开始的(文件 *config.py*):

十分简单吧，我们的 Flaks-WTF 扩展只需要两个配置。 CSRF_ENABLED 配置是为了激活 跨站点请求伪造 保护。在大多数情况下，你需要激活该配置使得你的应用程序更安全些。

SECRET_KEY 配置仅仅当 CSRF 激活的时候才需要，它是用来建立一个加密的令牌，用于验证一个表单。当你编写自己的应用程序的时候，请务必设置很难被猜测到密钥。十分简单吧，我们的 Flaks-WTF 扩展只需要两个配置。 *CSRF_ENABLED* 配置是为了激活 跨站点请求伪造 保护。在大多数情况下，你需要激活该配置使得你的应用程序更安全些。

```
CSRF_ENABLED = True
SECRET_KEY = 'you-will-never-guess'
```

SECRET_KEY 配置仅仅当 CSRF 激活的时候才需要，它是用来建立一个加密的令牌，用于验证一个表单。当你编写自己的应用程序的时候，请务必设置很难被猜测到密钥。

2.1既然我们有了配置文件，我们需要告诉 Flask 去读取以及使用它。

我们可以在 Flask 应用程序对象被创建后去做，方式如下(文件 app/**init**.py):

```
from flask import Flask
app = Flask(__name__)
app.config.from_object('config')
from app import views
```

2.2 用户登录表单

在 Flask-WTF 中，表单是表示成对象，Form 类的子类。一个表单子类简单地把表单的域定义成类的变量。

我们将要创建一个登录表单，**用户用于认证系统。**在我们应用程序中支持的登录机制不是标准的用户名/密码类型，我们将使用OpenID。。OpenIDs 的好处就是认证是由 OpenID 的提供者完成的，因此我们不需要验证密码，这会让我们的网站对用户而言更加安全。

OpenID 登录仅仅需要一个字符串，被称为 OpenID。我们将在表单上提供一个 'remember me' 的选择框，以至于用户可以选择在他们的网页浏览器上种植 cookie ，当他们再次访问的时候，浏览器能够记住他们的登录。

所以让我们编写第一个表单(文件 app/forms.py):

```
from flask.ext.wtf import Form
from wtforms import StringField, BooleanField
from wtforms.validators import DataRequired

class LoginForm(Form):
    openid = StringField('openid', validators=[DataRequired()])
    remember_me = BooleanField('remember_me', default=False)
```

我相信这个类不言而明。我们导入 Form 类，接着导入两个我们需要的字段类，TextField 和 BooleanField。

DataRequired 验证器只是简单地检查相应域提交的数据是否是空。在 Flask-WTF 中有许多的验证器，我们将会在以后看到它们。

3.表单模板

我们同样需要一个包含生成表单的 HTML 的模板。好消息是我们刚刚创建的 LoginForm 类知道如何呈现为 HTML 表单字段，所以我们只需要集中精力在布局上。这里就是我们登录的模板(文件 app/templates/login.html):

```
<!-- extend from base layout -->
{% extends "base.html" %}

{% block content %}
<h1>Sign In</h1>
<form action="" method="post" name="login">
    {{form.hidden_tag()}}
    <p>
        Please enter your OpenID:<br>
        {{form.openid(size=80)}}<br>
    </p>
    <p>{{form.remember_me}} Remember Me</p>
    <p><input type="submit" value="Sign In"></p>
</form>
{% endblock %}
```

在这时，还要在命令行安装:flask-wtf

> pip install flask-wtf

请注意，此模板中，我们重用了 base.html 模板通过 extends 模板继承声明语句。实际上，我们将在所有我们的模板中做到这一点，以确保所有网页的布局一致性。

form.hidden_tag() 模板参数将被替换为一个隐藏字段，用来是实现在配置中激活的 CSRF 保护。如果你已经激活了 CSRF，这个字段需要出现在你所有的表单中。

我们表单中实际的字段也将会被表单对象渲染，你只必须在字段应该被插入的地方指明一个 {{form.field_name}} 模板参数。某些字段是可以带参数的。在我们的例子中，我们要求表单生成一个 80 个字符宽度的 openid 字段。

因为我们并没有在表单中定义提交按钮，我们必须按照普通的字段来定义。提交字段实际并不携带数据因此没有必要在表单类中定义。

4. 表单视图

在我们看到我们表单前的最后一步就是编写渲染模板的视图函数的代码。

实际上这是十分简单因为我们只需要把一个表单对象传入模板中。这就是我们新的视图函数(文件 app/views.py):

```
from flask import render_template, flash, redirect
from app import app
from .forms import LoginForm

# index view function suppressed for brevity

@app.route('/login', methods = ['GET', 'POST'])
def login():
    form = LoginForm()
    return render_template('login.html',
        title = 'Sign In',
        form = form)
```

所以基本上，我们已经导入 LoginForm 类，从这个类实例化一个对象，接着把它传入到模板中。这就是我们渲染表单所有要做的。

让我们先忽略 flash 以及 redirect 的导入。我们会在后面介绍。

这里唯一的新的知识点就是路由装饰器的 methods 参数。参数告诉 Flask 这个视图函数接受 GET 和 POST 请求。如果不带参数的话，视图只接受 GET 请求。

这个时候你可以尝试运行应用程序，在浏览器上看看表单。在你运行应用程序后，你需要**在浏览器上打开 http://localhost:5000/login** 。

我们暂时还没有编写接收数据的代码，因此此时按提交按钮不会有任何作用。
如图：

5.接收表单数据

Flask-WTF 使得工作变得简单的另外一点就是处理提交的数据。这里是我们登录视图函数更新的版本，它验证并且存储表单数据 (文件 app/views.py):

```
@app.route('/login', methods = ['GET', 'POST'])
def login():
    form = LoginForm()
    if form.validate_on_submit():
        flash('Login requested for OpenID="' + form.openid.data + '",
remember_me=' + str(form.remember_me.data))
        return redirect('/index')
    return render_template('login.html',
        title = 'Sign In',
        form = form)
```

validate_on_submit 方法做了所有表单处理工作。当表单正在展示给用户的时候调用它，它会返回 False.

如果 validate_on_submit 在表单提交请求中被调用，它将会收集所有的数据，对字段进行验证，如果所有的事情都通过的话，它将会返回 True，表示数据都是合法的。这就是说明数据是安全的，并且被应用程序给接受了。

当 validate_on_submit 返回 True，我们的登录视图函数调用了两个新的函数，导入自 flask。

闪现的消息将不会自动地出现在我们的页面上，我们的模板需要加入展示消息的内容。我们将添加这些消息到我们的基础模板中，这样所有的模板都能继承这个函数。这是更新后的基础模板(文件 app/templates/base.html):

```
<html>
```

```
  <head>
    {% if title %}
    <title>{{title}} - microblog</title>
    {% else %}
    <title>microblog</title>
    {% endif %}
  </head>
  <body>
    <div>Microblog: <a href="/index">Home</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>
```

是到了启动应用程序的时候，测试下表单是如何工作的。确保您尝试提交表单的时候，OpenID 字段为空，看看 Required 验证器是如何中断提交的过程。

6. 加强字段验证

现阶段的应用程序，如果表单提交不合理的数据将不会被接受。相反，会返回表单让用户提交合法的数据。这确实是我们想要的。

然后，好像我们缺少了一个提示用户表单哪里出错了。幸运的是，Flask-WTF 也能够轻易地做到这一点。

当字段验证失败的时候， Flask-WTF 会向表单对象中添加描述性的错误信息。这些信息是可以在模板中使用的，因此我们只需要增加一些逻辑来获取它。

这就是我们含有字段验证信息的登录模板(文件 app/templates/login.html):

```
<!-- extend base layout -->
{% extends "base.html" %}

{% block content %}
  <h1>Sign In</h1>
  <form action="" method="post" name="login">
      {{ form.hidden_tag() }}
      <p>
          Please enter your OpenID:<br>
          {{ form.openid(size=80) }}<br>
          {% for error in form.openid.errors %}
            <span style="color: red;">[{{ error }}]</span>
          {% endfor %}<br>
      </p>
      <p>{{ form.remember_me }} Remember Me</p>
      <p><input type="submit" value="Sign In"></p>
  </form>
{% endblock %}
```

唯一的变化就是我们增加了一个循环获取验证 openid 字段的信息。通常情况下，任何需要验证的字段都会把错误信息放入 form.field_name.errors 下。在我们的例子中，我们使用 form.openid.errors 。我们以红色的字体颜色显示这些错误信息以引起用户的注意。

7.处理 OpenIDs

事实上，很多用户并不知道他们已经有一些 OpenIDs。一些大的互联网服务提供商支持 OpenID 认证自己的会员这并不是众所周知的。比如，如果你有一个 Google 的账号，你也就有了一个它们的 OpenID。

为了让用户更方便地使用这些常用的 OpenID 登录到我们的网站，我们把它们的链接转成短名称，用户不必手动地输入这些 OpenID。

我首先开始定义一个 OpenID 提供者的列表。我们可以把它们写入我们的配置文件中(文件 config):

```
CSRF_ENABLED = True
SECRET_KEY = 'you-will-never-guess'

OPENID_PROVIDERS = [
    { 'name': 'Google', 'url': 'https://www.google.com/accounts/o8/id' },
    { 'name': 'Yahoo', 'url': 'https://me.yahoo.com' },
    { 'name': 'AOL', 'url': 'http://openid.aol.com/<username>' },
    { 'name': 'Flickr', 'url': 'http://www.flickr.com/<username>' },
    { 'name': 'MyOpenID', 'url': 'https://www.myopenid.com' }]
```

现在让我们看看如何在我们登录视图函数中使用它们:

```
@app.route('/login', methods = ['GET', 'POST'])
def login():
    form = LoginForm()
    if form.validate_on_submit():
        flash('Login requested for OpenID="' + form.openid.data + '",
remember_me=' + str(form.remember_me.data))
        return redirect('/index')
    return render_template('login.html',
        title = 'Sign In',
        form = form,
        providers = app.config['OPENID_PROVIDERS'])
```

我们从配置中获取 OPENID_PROVIDERS，接着把它作为 render_template 中一个参数传入模板中。

我敢确信你们已经猜到了，我们还需要多做一步来达到目的。我们现在就来说明如何在登录模板中渲染这些提供商的链接(文件 app/templates/login.html):

```
<!-- extend base layout -->
{% extends "base.html" %}
{% block content %}
<script type="text/javascript">
function set_openid(openid, pr)
{
    u = openid.search('<username>')
    if (u != -1) {
        // openid requires username
        user = prompt('Enter your ' + pr + ' username:')
        openid = openid.substr(0, u) + user
    }
```

```
    form = document.forms['login'];
    form.elements['openid'].value = openid
}
</script>
<h1>Sign In</h1>
<form action="" method="post" name="login">
    {{ form.hidden_tag() }}
    <p>
        Please enter your OpenID, or select one of the providers below:<br>
        {{ form.openid(size=80) }}
        {% for error in form.openid.errors %}
          <span style="color: red;">[{{error}}]</span>
        {% endfor %}<br>
        |{% for pr in providers %}
          <a href="javascript:set_openid('{{ pr.url }}', '{{ pr.name }}');">{{
pr.name }}</a> |
        {% endfor %}
    </p>
    <p>{{ form.remember_me }} Remember Me</p>
    <p><input type="submit" value="Sign In"></p>
</form>
{% endblock %}
```

模板变得跟刚才不一样了。一些 OpenIDs 含有用户名，因此对于这些用户，我们必须利用 javascript 的魔力提示用户输入用户名并且组成 OpenIDs。当用户点击一个 OpenIDs 提供商的链接并且(可选)输入用户名，该提供商相应的 OpenID 就被写入到文本域中。

8. 结束语

尽管我们在登录表单上已经取得了很多进展，我们实际上没有做任何用户登录到我们的系统，到目前为止我们所做的是登录过程的 GUI 方面。这是因为在做实际登录之前，我们需要有一个数据库，那里可以记录我们的用户。

在下一章中，我们会得到我们的数据库并且运行它，接着我们将完成我们的登录系统。敬请关注后续文章。## 结束语

尽管我们在登录表单上已经取得了很多进展，我们实际上没有做任何用户登录到我们的系统，到目前为止我们所做的是登录过程的 GUI 方面。这是因为在做实际登录之前，我们需要有一个数据库，那里可以记录我们的用户。

在下一章中，我们会得到我们的数据库并且运行它，接着我们将完成我们的登录系统。敬请关注后续文章。