NSD Devweb DAY06

1 案例1：实现鉴权

1.1 问题

1. 编写登陆页面模板
2. 编写三个视图，分别用于登陆页、验证登陆以及受保护页面
3. 如果用户密码正确给出登陆成功，否则重定向到登陆页
4. 编写URLCONF，实现入口

1.2 方案

1.浏览器访问任意入口，检测session变量，如果没有设置就跳转/home入口，展示登陆页面

2.用户在登陆页面填写账号、密码信息，提交给/login入口，使用数据库鉴定是否是合法用户。如果合法，设置session变量为任意值，然后跳转到原始路径。

3.如果在任意入口检测logined变量存在，则正常显示页面。

1.3 步骤

实现此案例需要按照如下步骤进行。

步骤一：编写登陆页面模板

创建模板文件polls/templates/polls/home.html，将如下代码写入：

```
<form action="/polls/login/" method="post">      #表单提交后会转到login（验证用户是
{%csrf_token %}
    用户名：<input type="text" name="username"><br>
    密码：<input type="text" name="pwd">
<input type="submit" value="提交">
</form>
```

步骤二：编写三个视图，分别用于登陆页、验证登陆以及受保护页面

将三个视图写入前面案例创建项目中，在

Session 就是保存在后台数据或者缓存中的一个键值对,同样的存储着用户信息,为了更好的保护用户隐私,其实是对前端cookie的一个升级的保护措施.

当登录成功后,会向后台数据库与前端Cookie同时发放一段随机字符串,分别保存在后台的session中,前端写到用户浏览器中,用户下次登录时候 拿着浏览器存着的sessionID当做KEY去后台数据库中匹配进行验证登录即可拿到用户相关信息,可以防止敏感信息直接暴露在浏览器上。

Top

```
#登录页面
def home(request):
    return render(request, 'polls/home.html')

#验证用户是否登录成功
def login(request):
    if request.method == 'POST':
        #这里可以根据登录的用户显示不同的数据
        username = request.POST.get('username')
        pwd = request.POST.get('pwd')
    if username == 'zhangsan' and pwd == '123456':
        # 设置session
        request.session['IS_LOGIN'] = True
        return redirect('index')
    return redirect('home')

#受保护页面：已登录用户可以访问，如果没有登录重定向到登录页面
def protected(request):
        #从请求里获取username session
     #首先判断存不存在session，如果不存在跳转到home页面
    is_login = request.session.get('IS_LOGIN', False)
    if is_login:
        return HttpResponse('OK')
    return redirect('home')
```

步骤三：编写URLCONF，实现入口

修改polls/urls.py文件并更改URL，配对新视图

```
    url(r'^home/$', views.home, name='home'),
    url(r'^login/$', views.login, name='login'),
    url(r'^protected/$', views.protected, name = 'protected'),
```

步骤四：如果用户密码正确给出登陆成功，否则重定向到登陆页

当访问http://127.0.0.1:8000/polls/protected时，页面会跳转到如图-1所示：

Top

图-1

此时，登录账号密码输入错误时，页面自动重定向到登陆页，如图-2、图-3所示：

图-2

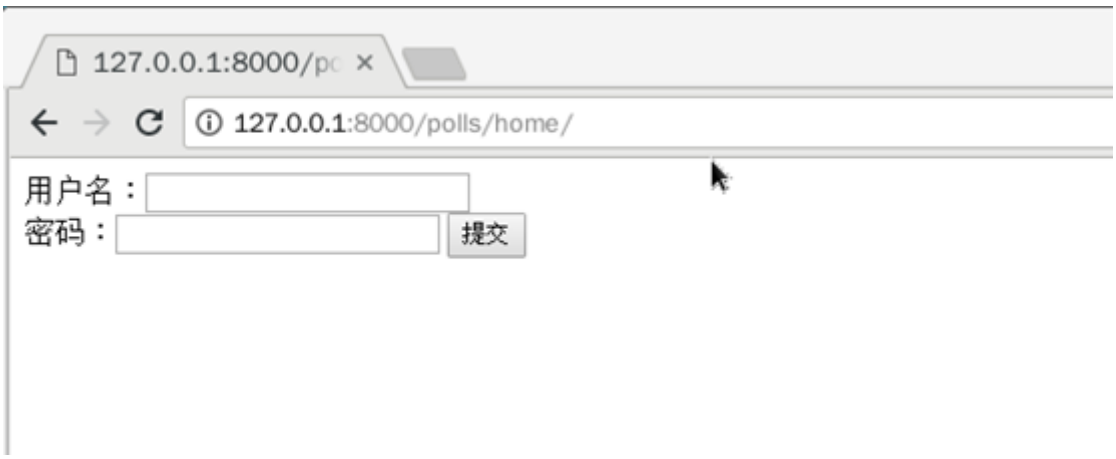

图-3

如果用户名密码输入正确，给出登录成功，输出结果如图-4、图-5所示：

Top

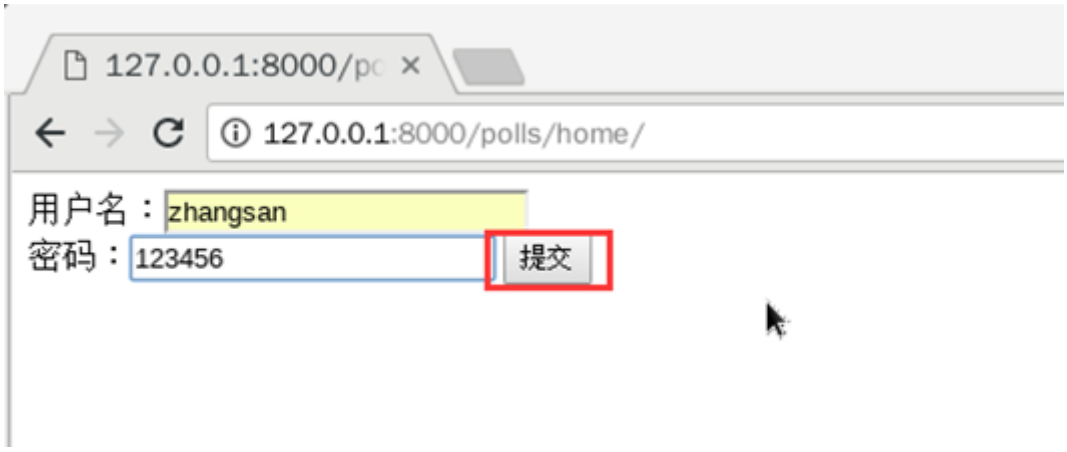

图-4

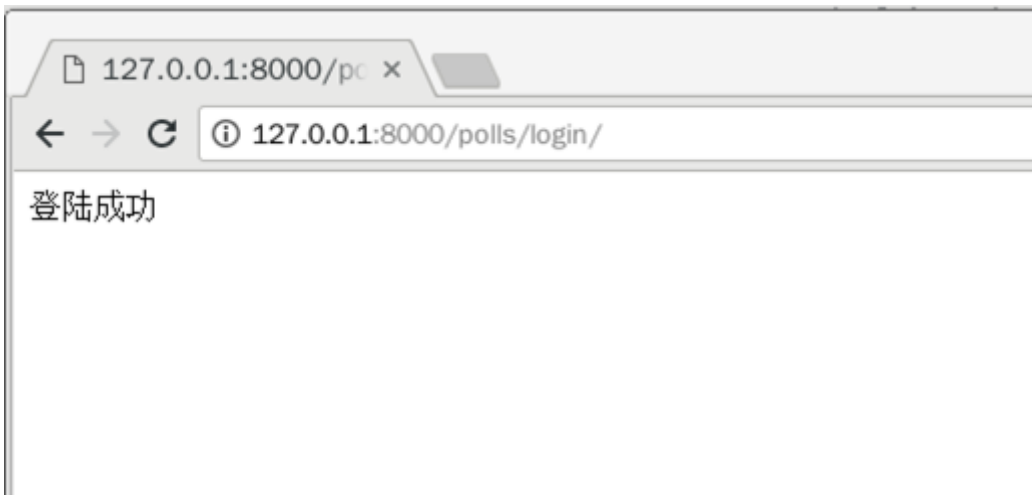

图-5

2 案例2：修改模板

2.1 问题

1. 为投票、投票结果、问题详情修改模板
2. 创建一个基础页面
3. 其他模板文件继承于基础页面

2.2 步骤

实现此案例需要按照如下步骤进行。

步骤一：创建一个基础页面：

在之前投票系统项目基础上，定义一个基础模板，该框架之后由子模板继承，

在templates文件下，创建基础模板base.html文件，为其他页面增加标题与页脚：

```
01.  <html lang="en">
02.  <head>
03.      <title>{ % block title %} { % endblock %} </title>
04.  </head>
05.  <body >
```

Top

```
06.     <h1>My helpful timestamp site</h1>
07.     {% block content %}{% endblock %}
08.     {% block footer %}
09.     <hr>
10.     <p>Thanks for visiting my site.</p>
11.     {% endblock %}
12. </body>
13. </html>
```

这个页面主要放公用部分代码，各个子页面都可以继承这个页面的样式

这个模版，它定义了一个可以用于两列排版页面的简单HTML骨架。“子模版”的工作是用它们的内容填充空的blocks。

在这个例子中， block 标签定义了三个可以被子模版内容填充的block。 block 告诉模版引擎： 子模版可能会覆盖掉模版中的这些位置。

步骤二：为投票、投票结果、问题详情修改模板

1)修改投票详情模板detail.html，让该模板继承base.html文件：

```
01. {% extends "polls/base.html" %}
02. {% block title %}投票详情{% endblock %}
03. {% block content %}
04.
05. <h1>{{ question.question_text }}</h1>
06. {% if error_message %}<p><strong>{{ error_message }}</strong></p>{% endif %}
07. <form action="/polls/{{ question.id }}/vote/" method="post">
08. {% csrf_token %}
09. {% for choice in question.choice_set.all %}
10.     <input type="radio" name="choice" id="choice{{ forloop.counter }}" value="{{ choice.i
11.     <label for="choice{{ forloop.counter }}">{{ choice.choice_text }}</label><br />
12. {% endfor %}
13. <input type="submit" value="投票" />
14. </form>
15.
16. {% endblock %}
```

2)修改投票结果模板results.html，让该模板继承base.html文件

Top

```
01. {% extends "polls/base.html" %}
02. {% block title %}投票结果{% endblock %}
```

```
{% block content %}

<h1>{{ question.question_text }}</h1>

<ul>
{% for choice in question.choice_set.all %}
    <li>{{ choice.choice_text }} : {{ choice.votes }}</li>
{% endfor %}
</ul>

<a href="/polls/">投票首页</a>

{% endblock %}
```

如上所示，{% extends ‘polls/base.html’ %}作为基础模板，必须放在第一行才可以识别。
{% block %}这个标签，告诉模板引擎，子模板可以重载这些
注意：修改results.html以及detail.html文件只需将以下代码加入即可

```
{% extends "polls/base.html" %}
{% block title %} #可替换内容{% endblock %}
{% block content %}

#子模板内容

{% endblock %}
```

步骤三：其他模板文件继承于基础页面结果显示

访问http://127.0.0.1:8000/polls/1/ ，现在启动服务器，在浏览器中查看效果，如图-6：

Top

图-6

访问http://127.0.0.1:8000/polls/1/ ，现在启动服务器，在浏览器中查看效果，如图：

图-7

如此两个html效果就显示出来了，同时也解释一下base.html中所起的作用，两个html中都使用了{% extends %}标记

这个就是继承base.html中的内容，在使用{ % block XXXXX %} {% endblock%}时，中间的内容便是插入在使用了base.html两个标签的

中间，由此便极大的避免了代码的冗余。每个模板只包含自己独一无二的代码，无需多余的部分，而如果想要进行站点级的设计修改，仅需

修改base.html，所有其他模板会立即反映出所做修改。

3 案例3：熟悉模型

Top

3.1 问题

1. 进入python shell
2. 导入模型
3. 对模型进行检索、增删改查操作

3.2 步骤

实现此案例需要按照如下步骤进行。

步骤一：编写Student类

1)在已创建的项目mysite中，创建Student模型，即修改mysite/polls/models.py文件内容如下：

```
01.  class Student(models.Model):
02.      id = models.IntegerField(primary_key=True)     #整型、主键
03.      cname = models.CharField(          #字符串字段
04.              unique=True,         #不允许重复
05.              max_length=128,
06.              blank=True,     #django的Admin中添加数据时是否可允许空值
07.              null=True        #数据库中字段是可以为空
08.              )
09.      cage = models.TextField(blank=True, null=True)         #字符串=longtext
10.
```

文件中每个class相当于单个数据库表，此时创建了Student表，每个属性也是这个表中的一个字段。 属性名就是字段名，它的类型（例如 CharField ）相当于数据库的字段类型。

步骤二：同步数据库

```
01.  (django_env) [root@localhost mysite]# python manage.py makemigrations
02.
03.  Migrations for 'polls':
04.    polls/migrations/0002_student.py
05.      - Create model Student
06.
07.  (django_env) [root@localhost mysite]# python manage.py migrate
08.
09.  Operations to perform:
10.    Apply all migrations: admin, auth, contenttypes, polls, sessions
11.  Running migrations:
12.    Applying polls.0002_student... OK
```

Top

步骤三：进入python shell

使用如下命令来调用Python shell：

```
(django_env) [root@localhost mysite]# python manage.py shell
```

步骤四：导入Student模型

```
>>> from polls.models import Student
```

步骤五：对模型进行检索、增删改查操作

1)增加数据：

通过create方法增加数据

```
>>>Person=Student.objects.create(cname='Tom',cage=12)     #新增name字段的值为tom
```

通过创建实例方法增加数据

```
>>>p = Student(cname="hanmeimei", cage=23)              #新增name字段的值为Tom，ag
>>>p.save()
```

2)删除数据：

直接删除

```
>>>student = Student()
>>>student.id = 13
>>>student.delete()

(0, {'polls.Student': 0})
```

查找对象后删除

```
>>>s=Student.objects.get(cage=12)
>>>s.delete()        #删除一条id=13的数据
```

Top

```
04.     (1, {'polls.Student': 1})
```

3)修改数据：

通过save方法修改记录

```
01.     >>>s = Student(id=1, cname='Tom', cage=12)
02.     >>>s.save()
```

通过update方法修改记录

```
01.     >>>Student.objects.filter(id=1).update(cname='Tom', cage=33)
02.
03.     1
```

4)查询数据：

```
01.     >>>Student.objects.get(id=1)  # 查询单条数据
02.     <Student: Tom>
03.
04.     >>> Student.objects.filter(cname='Tom')        # 查询匹配条件的多条数据
05.     <QuerySet [<Student: Tom>]>
06.
07.     >>>Student.objects.filter(cname__contains='Tom')  # 模糊查询；name为查询的字段名
08.     <QuerySet [<Student: Tom>]>
09.
10.     >>>Student.objects.filter(cname__contains=Tom)[0:5]  # [0]显示第一条 [0:2]会显示前
11.     <QuerySet [<Student: Tom>]>
```

获取表的所有记录：

```
01.     >>>Student.objects.all()
02.     <QuerySet [<Student: Tom>, <Student: hanmeimei>]>
```

Top

获取特定条件的记录：

```
>>>Student.objects.get(cname="hanmeimei")
<Student: hanmeimei>
```

获取前10条记录：

```
>>>Student.objects.all()[:10]
<QuerySet [<Student: Tom>, <Student: hanmeimei>]>
```

排序：

```
>>> Student.objects.order_by('cname')
<QuerySet [<Student: Tom>, <Student: hanmeimei>]>
```

Top