**Django 表单**

HTML表单是网站交互性的经典方式。 本章将介绍如何用Django对用户提交的表单数据进行处理。

**HTTP 请求**

HTTP协议以"请求－回复"的方式工作。客户发送请求时，可以在请求中附加数据。服务器通过解析请求，就可以获得客户传来的数据，并根据URL来提供特定的服务。

**GET 方法**

我们在之前的项目中创建一个 search.py 文件，用于接收用户的请求：

# -\*- coding: utf-8 -\*-

from django.http import HttpResponse

from django.shortcuts import render\_to\_response

# 表单

def search\_form(request):

return render\_to\_response('search\_form.html')

# 接收请求数据

def search(request):

request.encoding='utf-8'

if 'q' in request.GET:

message = '你搜索的内容为: ' + request.GET['q'].encode('utf-8')

else:

message = '你提交了空表单'

return HttpResponse(message)

在模板目录template中添加 search\_form.html 表单：

<html>

<head>

<meta charset="utf-8" />

<title>Search - w3cschool.cc</title>

</head>

<body>

<form action="/search/" method="get">

<input type="text" name="q">

<input type="submit" value="Search">

</form>

</body>

</html>

urls.py 规则修改为如下形式：

from django.conf.urls import \*

from HelloWorld.view import hello

from HelloWorld.testdb import testdb

from HelloWorld import search

urlpatterns = patterns("",

('^hello/$', hello),

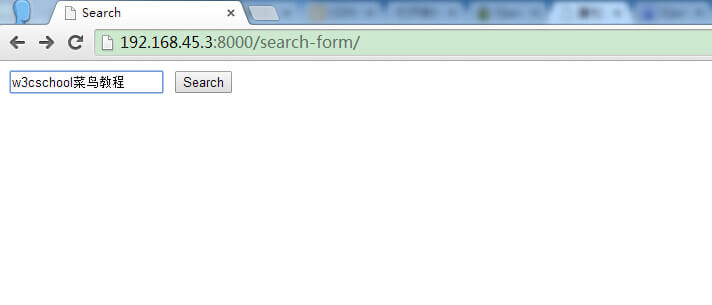
('^testdb/$', testdb),

(r'^search-form/$', search.search\_form),

(r'^search/$', search.search),

)

访问地址：http://192.168.45.3:8000/search-form/并搜索，结果如下所示:



**POST 方法**

上面我们使用了GET方法。视图显示和请求处理分成两个函数处理。

提交数据时更常用POST方法。我们下面使用该方法，并用一个URL和处理函数，同时显示视图和处理请求。

我们在tmplate 创建 post.html：

<html>

<head>

<meta charset="utf-8" />

<title>Search - w3cschool.cc</title>

</head>

<body>

<form action="/search-post/" method="post">

{% csrf\_token %}

<input type="text" name="q">

<input type="submit" value="Submit">

</form>

<p>{{ rlt }}</p>

</body>

</html>

在模板的末尾，我们增加一个rlt记号，为表格处理结果预留位置。

表格后面还有一个{% csrf\_token %}的标签。csrf全称是Cross Site Request Forgery。这是Django提供的防止伪装提交请求的功能。POST方法提交的表格，必须有此标签。

在HelloWorld目录下新建 search2.py 文件并使用 search\_post 函数来处理 POST 请求：

# -\*- coding: utf-8 -\*-

from django.shortcuts import render

from django.core.context\_processors import csrf

# 接收POST请求数据

def search\_post(request):

ctx ={}

ctx.update(csrf(request))

if request.POST:

ctx['rlt'] = request.POST['q']

return render(request, "post.html", ctx)

urls.py 规则修改为如下形式：

from django.conf.urls import \*

from HelloWorld.view import hello

from HelloWorld.testdb import testdb

from HelloWorld import search

from HelloWorld import search2

urlpatterns = patterns("",

('^hello/$', hello),

('^testdb/$', testdb),

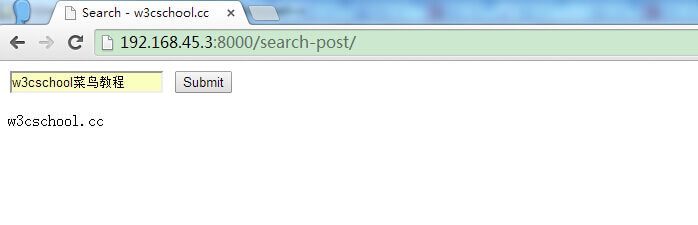
(r'^search-form/$', search.search\_form),

(r'^search/$', search.search),

(r'^search-post/$', search2.search\_post),

)

访问 http://192.168.45.3:8000/search-post/ 显示结果如下：



完成以上实例后，我们的目录结构为：

HelloWorld

|-- HelloWorld

| |-- \_\_init\_\_.py

| |-- \_\_init\_\_.pyc

| |-- models.pyc

| |-- search.py

| |-- search.pyc

| |-- search2.py

| |-- search2.pyc

| |-- settings.py

| |-- settings.pyc

| |-- testdb.py

| |-- testdb.pyc

| |-- urls.py

| |-- urls.pyc

| |-- view.py

| |-- view.pyc

| |-- wsgi.py

| `-- wsgi.pyc

|-- TestModel

| |-- \_\_init\_\_.py

| |-- \_\_init\_\_.pyc

| |-- admin.py

| |-- models.py

| |-- models.pyc

| |-- tests.py

| `-- views.py

|-- manage.py

`-- templates

|-- base.html

|-- hello.html

|-- post.html

`-- search\_form.html

3 directories, 29 files

**Request 对象**

每个view函数的第一个参数是一个HttpRequest对象，就像下面这个hello()函数:

from django.http import HttpResponse

def hello(request):

return HttpResponse("Hello world")

HttpRequest对象包含当前请求URL的一些信息：

|  |  |
| --- | --- |
| **属性** | **描述** |
| path | 请求页面的全路径,不包括域名—例如, "/hello/"。 |
| method | 请求中使用的HTTP方法的字符串表示。全大写表示。例如:  if request.method == 'GET':     do\_something() elif request.method == 'POST':     do\_something\_else() |
| GET | 包含所有HTTP GET参数的类字典对象。参见QueryDict 文档。 |
| POST | 包含所有HTTP POST参数的类字典对象。参见QueryDict 文档。  服务器收到空的POST请求的情况也是有可能发生的。也就是说，表单form通过HTTP POST方法提交请求，但是表单中可以没有数据。因此，不能使用语句if request.POST来判断是否使用HTTP POST方法；应该使用if request.method == "POST" (参见本表的method属性)。  注意: POST不包括file-upload信息。参见FILES属性。 |
| REQUEST | 为了方便，该属性是POST和GET属性的集合体，但是有特殊性，先查找POST属性，然后再查找GET属性。借鉴PHP's $\_REQUEST。  例如，如果GET = {"name": "john"} 和POST = {"age": '34'},则 REQUEST["name"] 的值是"john", REQUEST["age"]的值是"34".  强烈建议使用GET and POST,因为这两个属性更加显式化，写出的代码也更易理解。 |
| COOKIES | 包含所有cookies的标准Python字典对象。Keys和values都是字符串。参见第12章，有关于cookies更详细的讲解。 |
| FILES | 包含所有上传文件的类字典对象。FILES中的每个Key都是<input type="file" name="" />标签中name属性的值. FILES中的每个value 同时也是一个标准Python字典对象，包含下面三个Keys:   * filename: 上传文件名,用Python字符串表示 * content-type: 上传文件的Content type * content: 上传文件的原始内容   注意：只有在请求方法是POST，并且请求页面中<form>有enctype="multipart/form-data"属性时FILES才拥有数据。否则，FILES 是一个空字典。 |
| META | 包含所有可用HTTP头部信息的字典。 例如:   * CONTENT\_LENGTH * CONTENT\_TYPE * QUERY\_STRING: 未解析的原始查询字符串 * REMOTE\_ADDR: 客户端IP地址 * REMOTE\_HOST: 客户端主机名 * SERVER\_NAME: 服务器主机名 * SERVER\_PORT: 服务器端口   META 中这些头加上前缀HTTP\_最为Key, 例如:   * HTTP\_ACCEPT\_ENCODING * HTTP\_ACCEPT\_LANGUAGE * HTTP\_HOST: 客户发送的HTTP主机头信息 * HTTP\_REFERER: referring页 * HTTP\_USER\_AGENT: 客户端的user-agent字符串 * HTTP\_X\_BENDER: X-Bender头信息 |
| user | 是一个django.contrib.auth.models.User 对象，代表当前登录的用户。  如果访问用户当前没有登录，user将被初始化为django.contrib.auth.models.AnonymousUser的实例。  你可以通过user的is\_authenticated()方法来辨别用户是否登录：  if request.user.is\_authenticated():  # Do something for logged-in users.  else:  # Do something for anonymous users.  只有激活Django中的AuthenticationMiddleware时该属性才可用 |
| session | 唯一可读写的属性，代表当前会话的字典对象。只有激活Django中的session支持时该属性才可用。 参见第12章。 |
| raw\_post\_data | 原始HTTP POST数据，未解析过。 高级处理时会有用处。 |

Request对象也有一些有用的方法：

|  |  |
| --- | --- |
| **方法** | **描述** |
| \_\_getitem\_\_(key) | 返回GET/POST的键值,先取POST,后取GET。如果键不存在抛出 KeyError。  这是我们可以使用字典语法访问HttpRequest对象。  例如,request["foo"]等同于先request.POST["foo"] 然后 request.GET["foo"]的操作。 |
| has\_key() | 检查request.GET or request.POST中是否包含参数指定的Key。 |
| get\_full\_path() | 返回包含查询字符串的请求路径。例如， "/music/bands/the\_beatles/?print=true" |
| is\_secure() | 如果请求是安全的，返回True，就是说，发出的是HTTPS请求。 |

**QueryDict对象**

在HttpRequest对象中, GET和POST属性是django.http.QueryDict类的实例。

QueryDict类似字典的自定义类，用来处理单键对应多值的情况。

QueryDict实现所有标准的词典方法。还包括一些特有的方法：

|  |  |
| --- | --- |
| **方法** | **描述** |
| \_\_getitem\_\_ | 和标准字典的处理有一点不同，就是，如果Key对应多个Value，\_\_getitem\_\_()返回最后一个value。 |
| \_\_setitem\_\_ | 设置参数指定key的value列表(一个Python list)。注意：它只能在一个mutable QueryDict 对象上被调用(就是通过copy()产生的一个QueryDict对象的拷贝). |
| get() | 如果key对应多个value，get()返回最后一个value。 |
| update() | 参数可以是QueryDict，也可以是标准字典。和标准字典的update方法不同，该方法添加字典 items，而不是替换它们:  >>> q = QueryDict('a=1')  >>> q = q.copy() # to make it mutable  >>> q.update({'a': '2'})  >>> q.getlist('a')  ['1', '2']  >>> q['a'] # returns the last  ['2'] |
| items() | 和标准字典的items()方法有一点不同,该方法使用单值逻辑的\_\_getitem\_\_():  >>> q = QueryDict('a=1&a=2&a=3')  >>> q.items()  [('a', '3')] |
| values() | 和标准字典的values()方法有一点不同,该方法使用单值逻辑的\_\_getitem\_\_(): |

此外, QueryDict也有一些方法，如下表：

|  |  |
| --- | --- |
| **方法** | **描述** |
| copy() | 返回对象的拷贝，内部实现是用Python标准库的copy.deepcopy()。该拷贝是mutable(可更改的) — 就是说，可以更改该拷贝的值。 |
| getlist(key) | 返回和参数key对应的所有值，作为一个Python list返回。如果key不存在，则返回空list。 It's guaranteed to return a list of some sort.. |
| setlist(key,list\_) | 设置key的值为list\_ (unlike \_\_setitem\_\_()). |
| appendlist(key,item) | 添加item到和key关联的内部list. |
| setlistdefault(key,list) | 和setdefault有一点不同，它接受list而不是单个value作为参数。 |
| lists() | 和items()有一点不同, 它会返回key的所有值，作为一个list, 例如:  >>> q = QueryDict('a=1&a=2&a=3')  >>> q.lists()  [('a', ['1', '2', '3'])] |
| urlencode() | 返回一个以查询字符串格式进行格式化后的字符串(e.g., "a=2&b=3&b=5"). |