KylinBlog开发注解-Part4

Django函数视图

Django的视图(view)是处理业务逻辑的核心，它负责处理用户的请求并返回响应数据。

Django提供了两种编写视图的方式：基于函数的视图和基于类的视图。

在KylinBlog的编写过程中，我们主要使用的是基于函数的视图。

View (视图) 主要根据用户的请求返回数据，用来展示用户可以看到的内容(比如网页，图片)，也可以用来处理用户提交的数据，比如保存到数据库中。

Django的视图(`views.py`）通常和URL路由(URLconf)一起工作的。

工作原理

服务器在收到用户通过浏览器发来的请求后，会根据用户请求的url地址和`urls.py`里配置的url-视图映射关系，去执行相应视图函数或视图类，从而返回给客户端响应数据。

```
# views.py
from django.http import HttpResponse

def index(request):
    return HttpResponse("Hello， World!")
```

- 每个视图函数的第一个默认参数都必需是`request`, 它是一个全局变量。
- Django把每个用户请求封装成了`request`对象，它包含里当前请求的所有信息，比如请求路径`request.path`, 当前用户`request.user`以及用户通过POST提交的数据`request.POST`。

简易Blog视图讲解

urls.py

```
# blog/urls.py
from django.urls import path
from . import views

urlpatterns = [
    path('blog/', views.index, name='index'),
    path('blog/articles/<int:id>/', views.article_detail, name='article_detail'),
]
```

views.py

```
# blog/views.py
from django.shortcuts import render, get_object_or_404
from .models import Article

# 展示所有文章
def index(request):
    latest_articles = Article.objects.all().order_by('-pub_date')
    return render(request, 'blog/article_list.html', {"latest_articles": latest_articles})

# 展示所有文章
def article_detail(request, id):
    article = get_object_or_404(Article, pk=id)
    return render(request, 'blog/article_detail.html', {"article": article})
```

article_list.html

```
# blog/article_list.html

{% block content %}
{% for article in latest_articles %}
     {{ article.title }}
     {{ article.pub_date }}
{% endfor %}
{% endblock %}
```

article_detail.html

```
# blog/article_detail.html
{% block content %}
   {{ article.title }}
   {{ article.pub_date }}
   {{ article.body }}
{% endblock %}
```

工作机理

- 当用户在浏览器输入`/blog/`时，URL收到请求后会调用视图`views.py`里的`index`函数，展示所有文章。
- 当用户在浏览器输入`/blog/article/5/`时，URL不仅调用了`views.py`里的`article_detail`函数，而且还把参数文章id通过`<int:id>`括号的形式传递给了视图里的`article_detail`函数。。
- views.py里的index函数先提取要展示的数据对象列表`latest_articles`, 然后通过`render`方法传递给模板`blog/article_list.html`.。
- views.py里的`article_detail`方法先通过`get_object_or_404`方法和id调取某篇具体的文章对象article，然后通过render方法传递给模板`blog/article_detail.html`显示。

工作方法解释

`render()`和`get_object_or_404()`。

- `render`方法有4个参数。
    - 第一个是`request`,
    - 第二个是模板的名称和位置,
    - 第三个是需要传递给模板的内容, 也被称为`context object`。
    - 第四个参数是可选参数`content_type`（内容类型), 我们什么也没写。
- `get_object_or_404`方法

- 第一个参数是模型Models或数据集queryset的名字，
- 第二个参数是需要满足的条件（比如pk = id, title = 'python'）。
- 当需要获取的对象不存在时，给方法会自动返回Http 404错误。

复杂：视图处理用户提交的数据

视图View不仅用于确定给客户展示什么内容，以什么形式展示，而且也用来处理用户通过表单提交的数据。

views.py

```
from django.shortcuts import render, redirect, get_object_or_404
 from django.urls import reverse
 from .models import Article
 from .forms import ArticleForm

 # 创建文章
 def article_create(request):
     # 如果用户通过POST提交，通过request.POST获取提交数据
     if request.method == "POST":
         # 将用户提交数据与ArticleForm表单绑定
         form = ArticleForm(request.POST)
         # 表单验证，如果表单有效，将数据存入数据库
         if form.is_valid():
             form.save()
             # 创建成功，跳转到文章列表
             return redirect(reverse("blog:article_list"))
     else:
         # 否则空表单
         form = ArticleForm()
     return render(request, "blog/article_form.html", { "form": form, })

 # 更新文章
 def article_update(request, pk):
     # 从url里获取单篇文章的id值，然后查询数据库获得单个对象实例
     article = get_object_or_404(Article, pk=id)

     # 如果用户通过POST提交，通过request.POST获取提交数据
     if request.method == 'POST':
         # 将用户提交数据与ArticleForm表单绑定，进行验证
         form = ArticleForm(instance=article, data=request.POST)
         if form.is_valid():
             form.save()
             # 更新成功，跳转到文章详情
             return redirect(reverse("blog:article_detail", args=[pk,]))
     else:
         # 否则用实例生成表单
         form = ArticleForm(instance=article)

     return render(request, "blog/article_form.html", { "form": form, "object": article})
```

在创建和更新文章时我们向模板传递了`form`这个变量，模板会根据我们自定义的Form类自动生成表单。

使用了自定义的Form类对用户提交的数据(`request.POST`)进行验证,并将通过验证的数据存入数据库。

forms.py

```
#blog/forms.py
from .models import Article
from django import forms

class ArticleForm(forms.ModelForm):
     class Meta:
         model = Article
         fields = ['title', 'body']
```