Django的FBV与CBV模式

FBV 是 **function based views** 的英文缩写，顾名思义代表的是基于函数的视图，在以前的我们编写的代码中，在 views.py 文件编写的都是这类视图函数，相信大家对这种编写视图函数的方式已经比较的熟悉。在本节我们将介绍另一种吧编写视图函数的方式，即 **CBV**（class based views）基于类的视图函数。

你可能会问为什么又突然介绍 CBV，刚熟悉了 FBV 的模式，那么我只能告诉你学习技术一个循序渐进的过程，CBV 的模式相比 FBV 还是略显复杂，所以只有掌握了 FBV 模式，才能够更好的理解 CBV模式。那么 这两者到底有什么不同之处呢，使用方法又有什么区别呢？下面就让我们一起来学习新知识吧。

1. CBV与FBV的区别

我们知道 Python 是一个门面向对象的编程语言。如果我们只用函数来编写视图函数，那么就会造成很多面向对象的优点无法利用起来，比如说封装、继承、多态等。这也是 Django 之所以加入了 CBV 模式的原因。它可以让开发者使用类的形式去编写 View 视图函数。对于使用 CBV 模式优势总结了如下几点：

- CBV 将整个视图函数的逻辑拆成了类下的多个函数，依靠函数调用来实现完整的逻辑；
- 提高代码的可复用性，更加灵活，让开发者使用面向对象的技术，比如多继承、多态等；
- 可以用不同的函数针对不同的 HTTP 方法处理，而不是通过很多 if 判断，提高代码可读性。

当然 CBV 也不是万能的，当继承关系变得很复杂，亦或是代码不是特别规整的时候，这时要去找某一个函数到底是被哪一个父类重载也是一个麻烦事。此时使用 FBV 模式就变的很方便，所以还是要理解它们两者的区别，在合适的场景选用合适的方法，不能把其中某一种模式视为唯一。总体上来说 CBV 的模式，在实际的开发工作中使用的相对较多，所以我们要掌握这种编写 view 视图函数的模式。

2. FBV与CBV实际应用

下面就让我们实际应用一下 CBV 这种基于类的视图函数，我们 FBV 与 CBV 的方式编写如下视图函数：

```
#使用FBV方式
def login_fbv(request):
    if request.method=="GET":
        return HttpResponse("登录成功")
    elif request.method=="POST":
        pass
#使用CBV方式
from django.views import View
class LoginView(View): #需要继承自View类
    def get(self,request):
        return HttpResponse("登录成功")
    def post(self,request):
        pass
```

1) as_view()方法创建类实例

首先 CBV 需要继承自 View 类所以需要使用如下方式进行导包

from django.views import View

在 FBV 模式中 Django 的 URL 将一个请求分配给可调用的函数的即 login_fbv()，那么基于类的视图函数这种方式就不再适用了，URL 无法将一个请求分发给类去处理。针对这个问题，CBV 提供了一个 as_view() 静态方法，调用这个方法就会创建一个类的实例。

2) dispatch()分发函数

通过实例自动调用 CBV 内置 dispatch() 方法，我们把 dispatch() 方法看做一个分发函数，它会根据不同的请求方法调用相应的get()或者post()来进行处理。同样 CBV 模式也需要接受一个 HttpRequest对象即 request，最终返回一个 response 。如果想了解 dispatch() 函数可以通过源码进行了解，它的主要作用就是实现请求的分发。

3) 配置CBV模式的路由

其实细心的小伙伴不难发现，其实在 Django 生成的配置文件中已经告诉我们，如何使用 CVB 方式进行路由的配置。在项目文件即 BookStore¥BookStore¥urls.py 中注释着如何使用的提示代码，如下所示：

```
Function views  #基于函数的视图配置url
    1. Add an import:  from my_app import views
    2. Add a URL to urlpatterns:  path('', views.home, name='home')
Class-based views #基于类的视图配置url
    1. Add an import:  from other_app.views import Home #第一步
    2. Add a URL to urlpatterns:  path('', Home.as_view(), name='home') #第二步
```

现在我们在 index¥urls.py 中配置路由，如下所示：

```
from index.views import LoginView
urlpatterns=[path('logincbv/',LoginView.as_view())]#使用as_view()方法创建类实例
```

3.CBV中类属性设置方法

类属性的设置方法有两种，一种可以是使用 Python 方法，即继承父类，重写其相应的属性，或者添加新的属性，第二种方式我们可以使用路由中的 as_view() 方法传递参数属性值，从而来指定类的属性。示例如下所示：

```
#第一种方式重写父类
class LoginView(View): #需要继承自View类
    username='xiaoli'
    def get(self,request):
        return HttpResponse("登录成功")
    def post(self,request):
        pass
class LoginViewChild(LoginView):
    #继承后重写类属性
    username = 'xiaowang'
#第二种方法也可以
urlpatterns = [
   path(r'logincbv/', LoginView.as_view(name="xiaowang"))
]
```

CBV 体现了 Python 面向对象这一语言特性。CBV 是通过类的方式来编写视图函数。这相比较于 function，更能利用面向对象中多态的特性，因此更容易将项目中比较通用的功能抽象出来。

CBV 的实现原理大家可以通过看 Django 的源码就进一步的理解，大概流程就是由 path 路由的 as_view() 创建一个类的实例由此关联到类视图，关联到类视图之后，通过 CBV 内部的 dispatch() 方法进行请求的分发处理，处理完成后并将 Response 返回。