Django路由反向解析与命名空间

在讲解 Django 的反向解析之前，我们首先要了解反向解析的使用场景以及为要将引入反向解析，然后我们方可探讨如何通过反向解析达到我们先想要的目的。下面我们介绍几个概念，在前面《Django模板继承精讲》一节中，我们已经提到过而且也已经做了使用，那就是可以转发路由的 include 函数。

1. 路由转发函数

在实际开发过程中，一个Django 项目会包含很多的 app ，这时候如果我们只在主路由里进行配置就会显得杂乱无章，所以通常会在每个 app 里，创建各自的 urls.py 路由模块，然后从根路由出发，将 app 所属的 url 请求，全部转发到相应的 urls.py 模块中。而这个从主路由转发到各个应用路由的过程叫做路由的分发，而它的实现是使用include() 函数来完成的，如下所示：

```
from django.urls import path,include
from BookStore import views
urlpatterns = [path('index/',include('index.urls'))]
```

从主路由的 urls.py 中使用 include 函数将其关联到 index 应用的路由模块。

在讲解 Django 的反向解析之前，我们首先要了解为什么要将引入反向解析，然后我们方可探讨如何通过反向解析达到我们先想要的目的。下面我们介绍几个概念，在前面《Django模板继承精讲》一节中，我们已经提到过而且也已经做了使用，那就是可以转发路由的 include 函数。

2. 什么是反向解析

我们知道每个视图函数都有一个和其相对应的路由，但是如果它们之间的匹配关系发生了变化，那么与之对应的访问地址也需要跟着发生改变，这是极其不方便的。因此我们可以用一种动态解析 url 的方法来避免。我们使用 Path 语法提供的 name 属性给对应路由起别名，从而让与之对应的链接或者跳转，会根据这个别名来动态解析 url，这个动态解析 url 路径的过程就是反向解析。

在处理业务需求的过程中，当我们遇到重定向或在模板中需要链接到其他的视图函数，在这两种场景下我们就会使用到 url 的反向解析，

而不使用硬编码的方式将 url 放在模板中。这样做的优势在于当 URL 发生变化后，也无须我们更改模板中的 URL。这一点我们在《Django url标签详解》已经做了讲解。

通过 name 参数，可以反向解析 URL、反向 URL 匹配、反向 URL 查询或者简单的 URL 反查。

3. 反向解析的应用

在需要解析 URL 的地方，Django 提供了不同的方式来实现 URL 反向解析：

- 在模板层使用 {% url %} 模板标签；
- 在视图函数的 Python 代码中：使用 reverse() 函数；
- 在处理模型 model 类实例时：使用 get_absolute_url() 方法。

在本节我们对前两种方法进行实例讲解，第三种方法了解即可。首先我们需要在 index/urls.py 路由文件中对 index 应用进行注册，使用如下方式来完成注册：

app_name='index' #写在开始位置即可

1) 使用url标签实现反向解析

我们使用《Django模板继承精讲》一节中的代码进行演示， 使用 include 函数关联 index 应用从而实现路的由分发，在index/urls.py 的路由列表中给 test/ 添加相应的 name 参数，如下所示：

```
from django.urls import path
from index import views
app_name='index'
urlpatterns=[
path('test/',views.index_html,name='detail_hello'),
....]
```

然后在 index/templates 目录下新建 newtest.html 并编写如下代码：

```
<!DOCTYPE html>
<html lang="en">
<head>
    <meta charset="UTF-8">
    <title>Title</title>
</head>
<body>
```

```
<a href=" {% url 'index:detail_hello' %}">点击继续</a>
<p>一起去C语言中文嗨翻天。</p>
</body>
</html>
```

最后在 index¥views.py 编写视图函数并配置相应路由映射，如下所示：

```
#视图函数
def redict_url(request):
    return render(request,'index/newtest.html')
#路由映射
urlpatterns=[path('redict/',views.redict_url),
...]
```

最终通过访问 http://127.0.0.1:8000/index/redict/ 可以实现反向解析页面跳转。

2) reverse()函数实现反向解析

reverse() 函数是在视图函数中实现反向解析的一种方式，它通常与配合 HttpResponseRedirect（http 重定向） 一起使用。我们再结合上面的例子来讲解，那么它理解与使用起来就会非常的简单，首先我们 index/views.py 中定义一个视图函数，如下所示：

```
#reverse函数实现反向解析重定向到我们想要的有页面
def test_to_reverse(request):
    return HttpResponseRedirect(reverse('index:detail_hello'))
#在index/urls.py中为视图函数配置路由
urlpatterns=[
path('test/',views.index_html,name='detail_hello'),
path('reverse/',views.test_to_reverse)]
```

在分发式路由中使用 index:detail_hello 也就是“应用名:url 别名”，如果不是分发式路由可以直接使用 url 别名，即 reverse('detail_hello')

最后访问 http://127.0.0.1:8000/index/reverse/ 可以直接重定向到 http://127.0.0.1:8000/index/test/。

3) reverse() 函数简析

上面我们使用 reverse 函数完成了视图函数的重定向，但是这里还要给大家简单介绍一下 reverse() 函数。在 Django中 reverse() 的定义如下所示：

reverse(viewname,urlconf=None,args=None,kwargs=None,current_app=None)

它只有一个必填参数，其他都是可选参数。其中 viewname 参数除了可以接受 url 路由 name 的别名以外，还可以接受可调用视图函数对象作为参数。示例如下：

```
from BookStore import views
def test_to_reverse(request):
    return HttpResponseRedirect(reverse(views.test_url))
```

其他参数说明如下：

- urlconf：这个属性用于决定当前的反向解析使用哪个 URLconf 模块,默认是根 URLconf；
- args：它用于传递参数，可以是元组或者列表，顺序填充 url 中的位置参数；
- kwargs：字典类型的传参，和 args 作用一样；
- current_app:它指定当前视图函数所在的 app，本例中是 index 应用。

4. 命名空间namespace

我们知道一个 Django 项目中可以创建多个应用，每个应用又可以定义很多的视图函数，所以就会有很多的 url 路由映射（简称 url 模式），在这种情况下给 url 命名就难免会发生命名冲突， Django 为了解决这一问题，为开发者提供了命名空间功能即 namespace。url 命名空间使得即使在不同的应用（app）中定义了相同的 url 名称，也能够正确的实现 url 反向解析。

URL 命名空间分为两个部分：第一，应用命名空间即使用 app_name 关联应用名字；第二，使用 namespace 用来标识一个应用的实例，主要功能是区分同一个应用中不同的实例。

下面为了让大家更好的理命名空间的概念，我们对本节的示例进行改写，首先在主路由 BookStore/urls.py 的 url 模式列表中，使用 namespace 参数给 index 应用添加命名空间。如下所示：

```
urlpatterns = [
path('index/',include('index.urls',namespace='first'))
]
```

在 index 应用的 urls.py 中添加（若之前已经添加则无需再重新添加）如下所示：

app_name="index"

最后改写 index 应用的 test_to_reverse()视图函数，如下所示：

```
def test_to_reverse(request):
    return HttpResponseRedirect(reverse
('index:detail_hello',current_app=request.resolver_match.namespace))
```

经过上面的设置我们就完成了 index 应用的命名空间操作，这样即使是不同的应用存在相同 url 名称，Django 也能精准的解析我们想要的 url 路径。

url 的反向解析是个重要的知识点，它可以帮助我们在不同应用之间实现页面的跳转以及视图的重定向，所以我们要熟练的掌握并应用它们。