路由层

> 路由是Web服务的入口，就好像办事大厅有各个服务窗口一样

> Django奉行DRY主义，提倡使用简洁、优雅的URL：
>
> 可以不用`.html`、`.php`或`.cgi`之类后缀
>
> 尽量不要单独使用无序随机数字这样无意义的东西
>
> 让你随心所欲设计你的URL，不受框架束缚

路由

urlpatterns

> urlpatterns是路由文件中的一个全局变量，用来存放路由及视图函数的映射关系
>
> 用户发起的请求`URL`都会首先进入主控制目录下的这个`urls.py`文件中进行查找匹配

1. 首先找到`urls.py`下的`urlpatterns`全局变量，这是一个路由规则实例的列表数据。
2. 按照先后定义顺序，进行路由匹配。
3. 找到第一个匹配项时停止匹配，执行匹配到的视图函数。
4. 遍历完全，未发现匹配，`django`进行异常处理

> 其中`urlpatterns`中的每一个路由映射规则可以由`path`或`re_path`进行构造

> **注意**：`Django`的路由不考虑`HTTP`请求方式，仅根据URL进行路由；即，只要`URL`相同，无论`POST`、`GET`等哪种请求方式都指向同一个操作函数

path

- `path(regex, view, kwargs=None, name=None)`

  > `regex`：一个匹配对应url地址的规则字符串。
  >
  > `view`：路由对应的视图函数，并且会自动封装HttpRequest作为第一个参数给这个视图函
  >
  > `kwargs`：视图函数的关键字参数。
  >
  > `name`：该路由的全局命名，可以让我们方便的在django项目中任意部分显示的使用，相当于为`url`取变量名，接下来全局使用该命名值即可；当对应`url`路由改变之后，结合路由反向解析使用的地方不需要更改路由
  >
  > 此外，`django`还提供了一个兼容老版本**url**路由配置函数的**re_path**函数；`re_path`：第一个参数部分为一个正则匹配规则，其他与path同

静态路由

> 静态路由用来映射对应视图函数，以下是一个简单的例子

```python
from django.http import HttpResponse

def index(request):
    return HttpResponse('Hello Worlds!')
```

```python
from django.urls import path,re_path
from urlapp import views
urlpatterns = [
    path('',views.index),
    re_path(r"^",views.index),
]
```

路由传参

> 有的时候，我们的路由设置不能一直维持一个一成不变的状态；
>
> 比如遇到一些内容翻页的场景，那么我们的连接可能是：`xx.com/airticle_list/1/`、`xx.com/airticle_list/2/`
>
> 那么这样的路由其实对应的都应该是一个视图函数，用以展示页面内容，那么如何设计这样的路由，就要涉及到动态路由及路由传参

```python
def index(request,x,y):
    content = "x:%s\ny:%s" % (x,y)
    return HttpResponse(content)
```

> 定义如上函数，将会接收连接中的后两部份`path`值作为参数，分别依次给到`x`和`y`

```python
from django.urls import path,re_path
from urlapp import views
urlpatterns = [
    path('<int:x>/<str:y>/',views.index),
    #指明类型
    path("<x>/<y>/",views.index)
    #不指明类型
    re_path(r"^(?P<x>\d+)/(?P<y>[a-zA-Z]+)/$"),
    # (?P<name>pattern) 正则分组
    re_path(r"^(\d+)/([a-zA-Z]+)/$"),
]
```

> 路由通过尖括号进行分组匹配，使用int以及str内置转换器将连接对应部分的值进行转换；并将匹配到的结果传递到视图函数对应的参数位置上；
>
> 访问：`http://127.0.0.1:8000/1/abc/`
>
> 其中`1`将作为x的参数值，`abc`将作为y的参数
>
> 但如果访问连接是：`http://127.0.0.1:8000/abc/abc/`，这会匹配到第二个路由，第二个路由没有对传递参数的类型进行限定

- 内置Path转换器：

```
str：匹配除了路径分隔符（`/`）之外的非空字符串，这是默认的形式
int：匹配正整数，包含0
slug：匹配字母、数字以及横杠、下划线组成的字符串
uuid：匹配格式化的uuid，如 075194d3-6885-417e-a8a8-6c931e272f00
path：匹配任何非空字符串，包含了路径分隔符
```

自定义转换器

> 除了以上`django`所提供的path转换器，如果还觉得无法实现我们想要的功能，我们可以通过编写一个类进行自定义`path`转换器

1. 定义转换器类，类名随意
2. 定义类中必须属性

   > `regex`：一个字符串形式的正则表达式，也是对应的路由规则
   >
   > `to_python(self, value)`：用于将匹配到的路由字符串转换为`Python`中的数据类型，并传递给视图函数，**如果转换失败，必须抛出ValueError**，路由映射视图函数时使用
   >
   > `to_url(self, value)`：将`Python`数据类型转换为一段url的方法，`to_python`方法的反向操作，反向解析时使用

3. 通过django.urls模块中的register_converter函数进行注册 函数第一个参数为转换器类 函数第二个参数为转换器别名

> 以下定义一个路由参数只能是三位字符的路由规则

```python
#先将转换器类定义
class ThreeChar:
    regex = "[a-zA-Z]{3}"
    def to_python(self,value):
        print("to_python")
        return str(value)

    def to_url(self,value):
        # 当通过反向路由解析时，将会调用该函数
        print('to_url')
        return str(value)[:3]
        #此处切片操作是为了当反向路由解析传参字符串长于3时，可以将其截断，符合转换器正则规则
#注册转换器
from django.urls import register_converter
register_converter(ThreeChar,'tc')
```

```python
urlpatterns = [
    path('<tc:x>/<tc:y>/',views.index)
]
#127.0.0.1:8000/aaa/bbb/
```

> 接下里，通过路由进行访问该视图映射时，一定是三个字符所组成的路由才可以，否则是访问不到的

```
#urls.py
app_name = "app"
path('<tc:x>/<tc:y>/',views.index, name="threechr")

#views.py
return redirect(reverse("app:threechr",args=('aaaa','bbbb')))
#此时会调用three路由规则中的tc转换器中的to_url反向合成路由，并切片只取参数前三位
```

路由分发

> 我们的路由编写都是在项目主要目录下的`urls.py`文件中，但是如果`app`有很多的话，这么多路由都写到一起，明显是一件很不方便管理的事情
>
> 其实在之前的练习中，我们使用的方式均是路由分发，每个子`app`都拥有自己独立的`urls.py`路由映射文件，而主控路由文件里只需要使用`include`函数导入子`app`下路由文件即可，这就是路由分发

```
from django.contrib import admin
from django.urls import path,include

urlpatterns = [
    path('admin/', admin.site.urls),
    path('',include("urlapp.urls")) # 使用include 实现路由分发，找到子app下的路由文件
]
```

> 路由分发为我们带来的好处有很多，可以让我们在多个`app`的项目中更加方便有效的管理每一个路由
>
> 并且也可以让我们的用户在访问时看到浏览器中的`URL`地址更加**赏心悦目**

路由反向解析

> 到了这里，思考一下，之前我们已经设置过了很多路由；
>
> 但是现在会出现一个问题，比如我们把其中某个路由规则进行了修改，把`aaa`换成了`aba`，那么现在我们需要回到每一个使用到这个路由的地方进行同步修改，这显然非常麻烦的，如果修改的路由更多，这甚至是一个灾难

> `django`也为我们提供了一个解决办法，通过为路由映射使用`name`参数，来为每一个路由映射设置一个独立唯一的变量名

```
path('left/<str:x>/',views.left, name="left"),
path('right/<int:x>/',views.right, name="right"),
# 通过正则命名分组方式
re_path(r'^left/([a-zA-Z]+)/$',views.left,name="left"),
re_path(r'^right/(?P<x>\d+)/$',views.right, name="right")
```

- 两个视图函数对应如下：

```python
def left(request,x):
    # x: str
    content = {
        'message':x,
    }
    return render(request, "left.html", content)

def right(request,x):
    # x: int
    content = {
        'message':x,
    }
    return render(request, "right.html",content)
```

- 两个HTML页面

```html
<p>我是左页面</p>
<p>路由参数: {{ message }}</p>
<a href="{% url 'right' 123 %}">右页面</a>
<!-- ------另一个页面------ -->
<p>我是右页面</p>
<p>路由参数: {{ message }}</p>
<a href="{% url 'left' 'abc' %}">右页面</a>
```

> 在模板页面中，对于已命名路由可以通过`{% url "name" "arg" %}`模板标签进行反向解析
>
> 参数以空格隔开，在标签后传入

- 视图函数反向解析

```python
def index(request):
    return redirect(reverse("left",args=('aaa',) ))
```

> 在视图函数中需要使用到路由命名时，进行反向解析需要我们通过`django.shortcuts`模块下的`reverse`函数

- `reverse(viewname,args=None,kwargs=None)`

参数介绍
viewname：视图函数、命名路由映射、或视图函数路径的字符串
args：元组形式路由传参。
kwargs：字典形式路由传参

命名空间

> 如果想在多个`app`下使用相同的`name`路由命名，那么我们可以通过路由分发过程中的include函数来指定不同`app`所属的命名空间

```
from django.contrib import admin
from django.urls import path,include

urlpatterns = [
    path('admin/', admin.site.urls),
    path('app1/',include(("app1.urls",'app1'))),
    #直接传递一个元祖，元祖第一个值为分发路由地址，第二个值为命名空间
    path('app2/',include(("app2.urls",'app2')))
]
```

> 当为每个`app`的路由分发映射设置了命名空间，接下来在模板页面以及视图函数对路由的反向解析将是如下所示的样子，路由解析前加冒号指明命名空间

```
def index(request):
    return redirect(reverse("app1:left"))
```

```
<a href="{% url 'app2:left' %}">app2:left</a>
```

应用命名空间：app_name

> 使用`app_name`指明命名空间，在子`app`的`urls.py`文件下配置全局变量`app_name`，这个值是唯一的
>
> 在这个路由文件中定义的其他映射关系，将具有命名空间`app1`

```
app_name = "app1" # 这个值应该是唯一的
urlpatterns = [
    ...
]
```

实例命名空间：namespace

> 当有多个子`app`同时引入同一个子路由映射文件，比如这样

```
from django.contrib import admin
from django.urls import path,include

urlpatterns = [
    path('admin/', admin.site.urls),
    path('app1/',include("app1.urls")),
    path('app2/',include("app1.urls"))
]
```

> 这就会出现一个问题，不同的路由访问在做路由反向解析时，会造成混淆，
>
> 此时需要给每一个路由分发的规则设置`namespace`属性，为实例进行命名空间

```python
from django.contrib import admin
from django.urls import path,include

urlpatterns = [
    path('admin/', admin.site.urls),
    path('app1/',include("app1.urls",namespace="app1")),
    path('app2/',include("app1.urls",namespace="app2"))
]
```

> 这样做的好处，可以在不同路由导向同一`app`下时，为他们的不同命名空间；
>
> 虽然看起来到最后执行的视图函数功能是一样的，但可以分清楚究竟是哪个路由引起视图函数在工作
>
> 接下来视图及模板页面中使用`namespace`的值

```html
<p>我是左页面</p>
<p>路由参数: {{ message }}</p>
<a href="{% url 'app1:right' 123 %}">app1的右页面</a>
<p>我是右页面</p>
<p>路由参数: {{ message }}</p>
<a href="{% url 'app1:left' 'abc' %}">app1的左页面</a>
<!-- ---------------------------------------- -->
<p>我是左页面</p>
<p>路由参数: {{ message }}</p>
<a href="{% url 'app2:right' 123 %}">app2的右页面</a>
<p>我是右页面</p>
<p>路由参数: {{ message }}</p>
<a href="{% url 'app2:left' 'abc' %}">app2的左页面</a>
```