视图层

一、基础

A、URL配置

1、URL调度器

Django 允许你任意设计你的URL,不受框架束缚。

关于URL应该保持和有意义的卓越论证：https://www.w3.org/Provider/Style/URI

为了给一个应用设计URL,你需要创建一个Python 模块,通常称为URLconf(URL configuration)。这个模块是纯粹的Python 代码,包含URL 模式(简单的正则表达 式)到Python 函数(你的视图)的简单映射。

2、Django如何处理一个请求

当一个用户请求Django 站点的一个页面,下面是Django 系统决定执行哪个Python 代码使用的算法:

1. Django 决定要使用的根 URLconf 模块。通常,这个值就是 ROOT_URLCONF 的设置,但是如果进来HttpRequest 对象具有一个 urlconf 属性(通过 中间件 request processing 设置),则使用这个值来替换 ROOT_URLCONF 设置。
2. Django 加载该Python 模块并寻找可用的 urlpatterns 。它是 django.conf.urls.url() 实例的一个列表。
3. Django依次匹配每个URL模式,在与请求的URL匹配的第一个模式停下来。
4. 一旦其中的一个正则表达式匹配上,Django 将导入并调用给出的视图,它是一个简单的Python 函数(或者一个基于类的视图)。视图将获得如下参数:
    - 一个 **HttpRequest** 实例。
    - 如果匹配的正则表达式没有返回命名的组,那么正则表达式匹配的内容将作为位置参数提供给视图。
    - 关键字参数由正则式匹配的命名组组成,但可以被 **django.conf.urls.url()** 的可选参数 **kwargs** 覆盖。
5. 如果没有匹配到正则表达式,或者如果过程中抛出一个异常,Django 将调用 一个适当的错误处理视图。

3、URLConf在什么上查找

URLconf 在请求的URL 上查找,将它当做一个普通的Python 字符串。不包括GET和POST参数以及域名。

```
在http://www.example.com/myapp/ 请求中,URLconf 将查找myapp/ 。
在 http://www.example.com/myapp/?page=3 请求中,URLconf 仍将查找myapp/ 。
```

URLconf 不检查请求的方法。换句话讲,所有的请求方法 —— 同一个URL的POST、GET、HEAD等等——都将路由到相同的函数。

4、捕获的参数永远是字符串

每个捕获的参数都作为一个普通的Python 字符串传递给视图,无论正则表达式使用 的是什么匹配方式。例如,下面这行URLconf 中:

```
url(r'^articles/(?P<year>[0-9]{4})/$', views.year_archive),
# views.year_archive() 的 year 参数将是一个字符串,即使 [0-9]{4} 值匹配整数字符串。
```

5、指定视图参数的默认值

有一个方便的小技巧是指定视图参数的默认值。 下面是一个URLconf 和视图的示 例:

```
# URLconf
from django.conf.urls import url
from . import views

urlpatterns = [
   url(r'^blog/$', views.page),
   url(r'^blog/page(?P<num>[0-9]+)/$', views.page),
]

# View (in blog/views.py)
def page(request, num="1"):
   # Output the appropriate page of blog entries, according to num.
   ...
```

在上面的例子中,两个URL模式指向同一个视图 views.page 。但第一个模式不会从URL 中捕获任何值。若第一个模式匹配, page() 函数将使用 num 参数的默认值"1"。若第二个模式匹配, page() 将使用正则表达式捕获的 num 值。

6、性能

urlpatterns 中的每个正则表达式在第一次访问它们时被编译。这使得系统相当快。

7、urlpatterns变量的语法

urlpatterns 应该是 url() 实例的一个Python 列表。

8、错误处理

当Django 找不到一个匹配请求的URL 的正则表达式时,或者当抛出一个异常时,Django 将调用一个错误处理视图。

这些情况发生时使用的视图通过4个变量指定。它们的默认值应该满足大部分项目,但是通过赋值给它们以进一步的自定义也是可以的。

这些值可以在你的根URLconf 中设置。在其它URLconf 中设置这些变量将不会生效果。

它们的值必须是可调用的或者是表示视图的Python 完整导入路径的字符串,可以方便地调用它们来处理错误情况。

> 这些值是:
>
> - handler404 —— 参见 django.conf.urls.handler404
> - handler500 —— 参见 django.conf.urls.handler500
> - handler403 —— 参见 django.conf.urls.handler403
> - handler400 —— 参见 django.conf.urls.handler400

8、包含其他的URLconfs

例如,下面是URLconf for the Django 网站自己的URLconf 中一个片段。它包含许多其它URLconf:

```
from django.conf.urls import include, url

urlpatterns = [
    # ... snip ...
    url(r'^community/', include('django_website.aggregator.urls')),
    url(r'^contact/', include('django_website.contact.urls')),
    # ... snip ...
]
```

另外一种包含其它URL 模式的方式是使用一个url() 实例的列表。例如,

```
from django.conf.urls import include, url
from apps.main import views as main_views
from credit import views as credit_views

extra_patterns = [
   url(r'^reports/(?P<id>[0-9]+)/$', credit_views.report),
   url(r'^charge/$', credit_views.charge),
]

urlpatterns = [
   url(r'^$', main_views.homepage),
   url(r'^help/', include('apps.help.urls')),
   url(r'^credit/', include(extra_patterns)),
]
```

在这个例子中, /credit/reports/ URL将被 credit.views.report() 这个Django 视图处理。

捕获的参数

被包含的URLconf 会收到来之父URLconf 捕获的任何参数,所以下面的例子是合法的:

```
# In settings/urls/main.py
from django.conf.urls import include, url

urlpatterns = [
    url(r'^(?P<username>\w+)/blog/', include('foo.urls.blog')),
]

# In foo/urls/blog.py
from django.conf.urls import url
from . import view

urlpatterns = [
   url(r'^$', views.blog.index),
   url(r'^archive/$', views.blog.archive),
]
```

在上面的例子中,捕获的" username "变量将被如期传递给包含的 URLconf。

9、嵌套的参数

正则表达式允许嵌套的参数,Django 将解析它们并传递给视图。当反查时, Django 将尝试填满所有外围捕获的参数,并忽略嵌套捕获的参数。考虑下面的 URL 模式,它带有一个可选的 page 参数:

```
from django.conf.urls import url

urlpatterns = [
    url(r'blog/(page-(\d+)/)?$', blog_articles),
    # bad
    url(r'comments/(?:page-(?P<page_number>\d+)/)?$', comments),
    # good
]
```

两个模式都使用嵌套的参数,其解析方式是:例如 blog/page-2/ 将匹配 blog_articles 并带有两个位置参数 page-2/ 和2。第二个 comments 的模 式将匹配 comments/page-2/ 并带有一个值为2 的关键字参数 page_number 。 这个例子中外围参数是一个不捕获的参数 (?:...) 。

blog_articles 视图需要最外层捕获的参数来反查,在这个例子中 是 page-2/ 或者没有参数,而 comments 可以不带参数或者用一 个 page_number 值来反查。

嵌套捕获的参数使得视图参数和URL 之间存在强耦合,正如 blog_articles 所 示:视图接收URL(page-2/)的一部分,而不只是视图感兴趣的值。这种耦合 在反查时更加显著,因为反查视图时我们需要传递URL 的一个片段而不只是page 的值。

10、传递额外的选项给视图函数

django.conf.urls.url() 函数可接收一个可选的第三个参数,它是一个字典,表示想传给视图函数的额外关键字参数。

在这个例子中,对于 /blog/2005/ 请求,Django 将调用 views.year_archive(request, year='2005', foo='bar') :

```
urlpatterns = [
   url(r'^blog/(?P<year>[0-9]{4})/$', views.year_archive, {'foo': 'bar'
]
```

> 处理冲突 URL 模式捕获的命名关键字参数和在字典中传递的额外参数有可能具有相同的 名称。当这种情况发生时,将使用字典中的参数而不是URL 中捕获的参数。

11、传递额外的选项给include（）

你传递额外的选项给include() 时,被包含的URLconf 的每一行将被传递这些额外的选项。

例如,下面两个URLconf 设置功能上完全相同:

设置一次:

```
# main.py
urlpatterns = [
    url(r'^url(r'^blog/', include('inner'), {'blogid': 3}),
]

# inner.py
urlpatterns = [
   url(r'^archive/$', views.archive),
   url(r'^about/$', views.about),
]
```

设置两次:

```
# main.py
urlpatterns = [
  url(r'^blog/', include('inner')),
]

# inner.py
urlpatterns = [
   url(r'^archive/$', views.archive, {'blogid': 3}),
   url(r'^about/$', views.about, {'blogid': 3})
]
```

注意,额外的选项将永远传递给被包含的URLconf 中的每一行,无论该行的视图实 际上是否认为这些选项是合法的。由于这个原因,该技术只有当你确定被包含的 URLconf 中的每个视图都接收你传递给它们的额外的选项。

12、URL的反响解析

在使用Django 项目时,一个常见的需求是获得URL 的最终形式,以用于嵌入到生 成的内容中(视图中和显示给用户的URL等)或者用于处理服务器端的导航(重定向等)。

> Django 提供一个办法是让URL 映射是URL 设计唯一的地方。你填充你的 URLconf,然后可以双向使用它:
>
> - 根据用户/浏览器发起的URL 请求,它调用正确的Django 视图,并从URL 中提取它的参数需要的值。
> - 根据Django 视图的标识和将要传递给它的参数的值,获取与之关联的URL。

在需要URL 的地方,对于不同层级,Django 提供不同的工具用于URL反查:

- 在模板中:使用url 模板标签。
- 在Python 代码中:使用 django.core.urlresolvers.reverse() 函数。
- 在更高层的与处理Django 模型实例相关的代码中:使用 get_absolute_url() 方法。

例子： URLconf:

```
from django.conf.urls import url
from . import views

urlpatterns = [
   #...
   url(r'^articles/([0-9]{4})/$', views.year_archive, name='news-year-archive'),
   #...
]
```

你可以在模板的代码中使用下面的方法获得它们:

```
<a href="{% url 'news-year-archive' 2012 %}">2012 Archive</a>
<ul>
{% for yearvar in year_list %}
<li><a href="{% url 'news-year-archive' yearvar %}">{{ yearvar }
} Archive</a></li>
{% endfor %}
</ul>
```

在Python 代码中,这样使用:

```
from django.core.urlresolvers import reverse
from django.http import HttpResponseRedirect

def redirect_to_year(request):
   # ...
   year = 2006
   # ...
   return HttpResponseRedirect(reverse('news-year-archive', args=(year,)))
```

如果出于某种原因决定按年归档文章发布的URL应该调整一下,那么你将只需要修 改URLconf 中的内容。

13、命名URL模式

为了完成上面例子中的URL 反查,你将需要使用命名的URL 模式。URL 的名称使用的字符串可以包含任何你喜欢的字符。不只限制在合法的Python 名称。

当命名你的URL 模式时,请确保使用的名称不会与其它应用中名称冲突。

14、URL命名空间

URL 命名空间允许你反查到唯一的命名URL 模式,即使不同的应用使用相同的 URL 名称。第三方应用始终使用带命名空间的URL 是一个很好的实践。一个应用的多个实例共享相同的命名URL,命名空间将提 供一种区分这些命名URL 的方法。

在一个站点上,正确使用URL 命名空间的Django 应用可以部署多次。例如, django.contrib.admin 具有一个 AdminSite 类,它允许你很容易地部署 多个管理站点的实例。

> 一个URL 命名空间有两个部分,它们都是字符串:
>
> - **应用命名空间**
>
> 它表示正在部署的应用的名称。一个应用的每个实例具有相同的应用命名空间。
> - **实例命名空间**
>
> 它表示应用的一个特定的实例。实例的命名空间在你的全部项目中应该是唯一的。 但是,一个实例的命名空间可以和应用的命名空间相同。它用于表示一个应用的默 认实例。

命名空间也可以嵌套。命名URL' sports:polls:index ' 将在命名空间' polls '中查找' index ',而 poll 定义在顶层的命名空间' sports ' 中。

反查带命名空间的URL

> 当解析一个带命名空间的URL(例如' polls:index ')时,Django 将切分名称为多个部分,然后按下面的步骤查找:
>
> 1. 首先,Django 查找匹配的应用的命名空间(在这个例子中为' polls ')。这将得到该应用实例的一个列表。
> 2. 如果有定义当前 应用,Django 将查找并返回那个实例的URL 解析器。当前 应 用可以通过请求上的一个属性指定。希望可以多次部署的应用应该设置正在处 理的 request 上的 current_app 属性。

> 当前应用还可以通过 reverse() 函数的一个参数手工设定。
>
> 1. 如果没有当前应用。Django 将查找一个默认的应用实例。默认的应用实例是 实例命名空间 与应用命名空间 一致的那个实例(在这个例子中, polls 的 一个叫做' polls ' 的实例)。
> 2. 如果没有默认的应用实例,Django 将该应用挑选最后部署的实例,不管实例 的名称是什么。
> 3. 如果提供的命名空间与第1步中的应用命名空间 不匹配,Django 将尝试直接将 此命名空间作为一个实例命名空间查找。如果有嵌套的命名空间,将为命名空间的每个部分重复调用这些步骤直至剩下视图 的名称还未解析。然后该视图的名称将被解析到找到的这个命名空间中的一个 URL。

例A:

```
# urls.py
from django.conf.urls import include, url

urlpatterns = [
    url(r'^author-polls/', include('polls.urls', namespace='author-polls',
app_name='polls')),
    url(r'^publisher-polls/', include('polls.urls', namespace='publisher-polls',
app_name='poll')),
]
```

例B:

```
# polls/urls.py
from django.conf.urls import url
from . import views

urlpatterns = [
   url(r'^$', views.IndexView.as_view(), name='index'),
   url(r'^(?P<pk>\d+)/$', views.DetailView.as_view(), name='detail'),
   ...
]
```

> 在基于类的视图的方法中:
>
> ```
> reverse('polls:index', current_app=self.request.resolver_match.namespace)
> ```
>
> 在模板中:
>
> ```
>  {% url 'polls:index' %}
> ```

URL 命名空间和被包含的URLconf

被包含的URLconf 的命名空间可以通过两种方式指定。 首先,在你构造你的URL 模式时,你可以提供 应用 和 实例的命名空间 给 include() 作为参数。例如:

```
url(r'^polls/', include('polls.urls', namespace='author-polls',
app_name='polls')),
```

请确保传递一个元组给 include() 。如果你只是传递3个参数: **include(polls_patterns, 'polls', 'author-polls')** ,Django 不会抛 出一个错误,但是根据 include() 的功能,' polls ' 将是实例的命名空间 而' author-polls ' 将是应用的命名空间,而不是反过来的。

B、视图函数

1、编写视图

2、把你的URL映射到视图

3、返回错误

4、Http404异常

5、自定义错误视图

C、快捷函数

1、render

```
render(request, template_name, context=None, content_type=None, status=None,
using=None)[source]
```

2、render_to_response

```
render_to_response(template_name, context=None,content_type=None, status=None,
using=None)[source]
```

3、redirect

```
redirect(to, permanent=False, *args, **kwargs)[source]
```

4、get_object_or_404

```
get_object_or_404(klass, *args, **kwargs)[source]
```

5、get_list_or_404

```
get_list_or_404(klass, *args, **kwargs)[source]
```

D、视图装饰器

1、允许的HTTP方法

2、可控制的视图处理

3、GZip视图压缩

4、Vary头部

二、参考

三、文件上传

四、基于类的视图

五、高级

六、中间件