Django 中的视图的概念是「一类具有相同功能和模板的网页的集合」

在 Django 中，网页和其他内容都是从视图派生而来。每一个视图表现为一个简单的 Python 函数（或者说方法，如果是在基于类的视图里的话）。Django 将会根据用户请求的 URL 来选择使用哪个视图（更准确的说，是根据 URL 中域名之后的部分）。

为了将 URL 和视图关联起来，Django 使用了 'URLconfs' 来配置。URLconf 将 URL 模式映射到视图。

本教程只会介绍 URLconf 的基础内容，你可以看看 [URL调度器](https://docs.djangoproject.com/zh-hans/2.1/topics/http/urls/) 以获取更多内容。

编写更多视图[**¶**](https://docs.djangoproject.com/zh-hans/2.1/intro/tutorial03/#writing-more-views)

现在让我们向 **polls/views.py** 里添加更多视图。这些视图有一些不同，因为他们接收参数：

polls/views.py**[¶](https://docs.djangoproject.com/zh-hans/2.1/intro/tutorial03/" \l "id1" \o "永久链接至代码)**

**def** detail(request, question\_id):

**return** HttpResponse("You're looking at question **%s**." % question\_id)

**def** results(request, question\_id):

response = "You're looking at the results of question **%s**."

**return** HttpResponse(response % question\_id)

**def** vote(request, question\_id):

**return** HttpResponse("You're voting on question **%s**." % question\_id)

把这些新视图添加进 **polls.urls** 模块里，只要添加几个 **[url()](https://docs.djangoproject.com/zh-hans/2.1/ref/urls/" \l "django.conf.urls.url" \o "django.conf.urls.url)** 函数调用就行：

polls/urls.py**[¶](https://docs.djangoproject.com/zh-hans/2.1/intro/tutorial03/" \l "id2" \o "永久链接至代码)**

**from** **django.urls** **import** path

**from** **.** **import** views

urlpatterns = [

*# ex: /polls/*

path('', views.index, name='index'),

*# ex: /polls/5/*

path('<int:question\_id>/', views.detail, name='detail'),

*# ex: /polls/5/results/*

path('<int:question\_id>/results/', views.results, name='results'),

# http://127.0.0.1:8000/polls/344152/vote/

path('<int:question\_id>/vote/', views.vote, name='vote'),

]

然后看看你的浏览器，如果你转到 "/polls/34/" ，Django 将会运行 **detail()** 方法并且展示你在 URL 里提供的问题 ID。再试试 "/polls/34/vote/" 和 "/polls/34/vote/" ——你将会看到暂时用于占位的结果和投票页。

当某人请求你网站的某一页面时——比如说， "/polls/34/" ，Django 将会载入 **mysite.urls** 模块，因为这在配置项 [**ROOT\_URLCONF**](https://docs.djangoproject.com/zh-hans/2.1/ref/settings/#std:setting-ROOT_URLCONF) 中设置了。然后 Django 寻找名为 **urlpatterns** 变量并且按序匹配正则表达式。在找到匹配项 **'polls/'**，它切掉了匹配的文本（**"polls/"**），将剩余文本——**"34/"**，发送至 'polls.urls' URLconf 做进一步处理。在这里剩余文本匹配了 **'<int:question\_id>/'**，使得我们 Django 以如下形式调用 **detail()**:

detail(request=<HttpRequest object>, question\_id=34)

**question\_id=34** 由 **<int:question\_id>** 匹配生成。使用尖括号“捕获”这部分 URL，且以关键字参数的形式发送给视图函数。上述字符串的 **:question\_id>** 部分定义了将被用于区分匹配模式的变量名，而 **int:** 则是一个转换器决定了应该以什么变量类型匹配这部分的 URL 路径。

## 写一个真正有用的视图[¶](https://docs.djangoproject.com/zh-hans/2.1/intro/tutorial03/#write-views-that-actually-do-something)

每个视图必须要做的只有两件事：返回一个包含被请求页面内容的 **[HttpResponse](https://docs.djangoproject.com/zh-hans/2.1/ref/request-response/" \l "django.http.HttpResponse" \o "django.http.HttpResponse)** 对象，或者抛出一个异常，比如 [**Http404**](https://docs.djangoproject.com/zh-hans/2.1/topics/http/views/#django.http.Http404) 。至于你还想干些什么，随便你。

你的视图可以从数据库里读取记录，可以使用一个模板引擎（比如 Django 自带的，或者其他第三方的），可以生成一个 PDF 文件，可以输出一个 XML，创建一个 ZIP 文件，你可以做任何你想做的事，使用任何你想用的 Python 库。

Django 只要求返回的是一个 **[HttpResponse](https://docs.djangoproject.com/zh-hans/2.1/ref/request-response/" \l "django.http.HttpResponse" \o "django.http.HttpResponse)** ，或者抛出一个异常。

因为 Django 自带的数据库 API 很方便，我们曾在 [教程第 2 部分](https://docs.djangoproject.com/zh-hans/2.1/intro/tutorial02/) 中学过，所以我们试试在视图里使用它。我们在 **index()** 函数里插入了一些新内容，让它能展示数据库里以发布日期排序的最近 5 个投票问题，以空格分割：

polls/views.py**[¶](https://docs.djangoproject.com/zh-hans/2.1/intro/tutorial03/" \l "id3" \o "永久链接至代码)**

**from** **django.http** **import** HttpResponse

**from** **.models** **import** Question

**def** index(request):

latest\_question\_list = Question.objects.order\_by('-pub\_date')[:5]

output = ', '.join([q.question\_text **for** q **in** latest\_question\_list])

**return** HttpResponse(output)

*# 这一段不用写，只是过渡的*

这里有个问题：页面的设计写死在视图函数的代码里的。如果你想改变页面的样子，你需要编辑 Python 代码。所以让我们使用 Django 的模板系统，只要创建一个视图，就可以将页面的设计从代码中分离出来。

首先，在你的 **polls** 目录里创建一个 **templates** 目录。Django 将会在这个目录里查找模板文件。

你项目的 [**TEMPLATES**](https://docs.djangoproject.com/zh-hans/2.1/ref/settings/#std:setting-TEMPLATES) 配置项描述了 Django 如何载入和渲染模板。默认的设置文件设置了 **DjangoTemplates** 后端，并将 [**APP\_DIRS**](https://docs.djangoproject.com/zh-hans/2.1/ref/settings/#std:setting-TEMPLATES-APP_DIRS) 设置成了 True。这一选项将会让 **DjangoTemplates** 在每个 [**INSTALLED\_APPS**](https://docs.djangoproject.com/zh-hans/2.1/ref/settings/#std:setting-INSTALLED_APPS) 文件夹中寻找 "templates" 子目录。这就是为什么尽管我们没有像在第二部分中那样修改 DIRS 设置，Django 也能正确找到 polls 的模板位置的原因。

在你刚刚创建的 **templates** 目录里，再创建一个目录 **polls**，然后在其中新建一个文件 **index.html**。换句话说，你的模板文件的路径应该是 **polls/templates/polls/index.html** 。因为 Django 会寻找到对应的 **app\_directories** ，所以你只需要使用 **polls/index.html** 就可以引用到这一模板了。

**模板命名空间**

虽然我们现在可以将模板文件直接放在 **polls/templates** 文件夹中（而不是再建立一个 **polls** 子文件夹），但是这样做不太好。Django 将会选择第一个匹配的模板文件，如果你有一个模板文件正好和另一个应用中的某个模板文件重名，Django 没有办法 区分 它们。我们需要帮助 Django 选择正确的模板，最简单的方法就是把他们放入各自的 命名空间 中，也就是把这些模板放入一个和 自身 应用重名的子文件夹里。

将下面的代码输入到刚刚创建的模板文件中：

polls/templates/polls/index.html[**¶**](https://docs.djangoproject.com/zh-hans/2.1/intro/tutorial03/#id4)

{% **if** latest\_question\_list %}

<**ul**>

{% **for** question **in** latest\_question\_list %}

<**li**><**a** href="/polls/{{ question.id }}/">{{ question.question\_text }}</**a**></**li**>

{% **endfor** %}

</**ul**>

{% **else** %}

<**p**>No polls are available.</**p**>

{% **endif** %}

然后，让我们更新一下 **polls/views.py** 里的 **index** 视图来使用模板：

polls/views.py**[¶](https://docs.djangoproject.com/zh-hans/2.1/intro/tutorial03/" \l "id5" \o "永久链接至代码)**

**from** **django.http** **import** HttpResponse

**from** **django.template** **import** loader

**from** **.models** **import** Question

**def** index(request):

latest\_question\_list = Question.objects.order\_by('-pub\_date')[:5]

template = loader.get\_template('polls/index.html')

context = {

'latest\_question\_list': latest\_question\_list,

}

**return** HttpResponse(template.render(context, request))

上述代码的作用是，载入 **polls/index.html** 模板文件，并且向它传递一个上下文(context)。这个上下文是一个字典，它将模板内的变量映射为 Python 对象。

用你的浏览器访问 "/polls/" ，你将会看见一个无序列表，列出了我们在 [教程第 2 部分](https://docs.djangoproject.com/zh-hans/2.1/intro/tutorial02/) 中添加的 “What's up” 投票问题，链接指向这个投票的详情页。

### 一个快捷函数： [render()](https://docs.djangoproject.com/zh-hans/2.1/topics/http/shortcuts/#django.shortcuts.render)[¶](https://docs.djangoproject.com/zh-hans/2.1/intro/tutorial03/#a-shortcut-render)

「载入模板，填充上下文，再返回由它生成的 **[HttpResponse](https://docs.djangoproject.com/zh-hans/2.1/ref/request-response/" \l "django.http.HttpResponse" \o "django.http.HttpResponse)** 对象」是一个非常常用的操作流程。于是 Django 提供了一个快捷函数，我们用它来重写 **index()** 视图：

polls/views.py**[¶](https://docs.djangoproject.com/zh-hans/2.1/intro/tutorial03/" \l "id6" \o "永久链接至代码)**

**from** **django.shortcuts** **import** render

**from** **.models** **import** Question

**def** index(request):

latest\_question\_list = Question.objects.order\_by('-pub\_date')[:5]

context = {'latest\_question\_list': latest\_question\_list}

**return** render(request, 'polls/index.html', context)

注意到，我们不再需要导入 [**loader**](https://docs.djangoproject.com/zh-hans/2.1/topics/templates/#module-django.template.loader) 和 **[HttpResponse](https://docs.djangoproject.com/zh-hans/2.1/ref/request-response/" \l "django.http.HttpResponse" \o "django.http.HttpResponse)** 。不过如果你还有其他函数（比如说 **detail**, **results**, 和 **vote** ）需要用到它的话，就需要保持 **HttpResponse** 的导入。

The [**render()**](https://docs.djangoproject.com/zh-hans/2.1/topics/http/shortcuts/#django.shortcuts.render) function takes the request object as its first argument, a template name as its second argument and a dictionary as its optional third argument. It returns an **[HttpResponse](https://docs.djangoproject.com/zh-hans/2.1/ref/request-response/" \l "django.http.HttpResponse" \o "django.http.HttpResponse)** object of the given template rendered with the given context.

## 抛出 404 错误[¶](https://docs.djangoproject.com/zh-hans/2.1/intro/tutorial03/#raising-a-404-error)

现在，我们来处理投票详情视图——它会显示指定投票的问题标题。下面是这个视图的代码：

polls/views.py**[¶](https://docs.djangoproject.com/zh-hans/2.1/intro/tutorial03/" \l "id7" \o "永久链接至代码)**

**from** **django.http** **import** Http404

**from** **django.shortcuts** **import** render

**from** **.models** **import** Question

*# ...*

**def** detail(request, question\_id):

**try**:

question = Question.objects.get(pk=question\_id)

**except** Question.DoesNotExist:

**raise** Http404("Question does not exist")

**return** render(request, 'polls/detail.html', {'question': question})

这里有个新原则。如果指定问题 ID 所对应的问题不存在，这个视图就会抛出一个 [**Http404**](https://docs.djangoproject.com/zh-hans/2.1/topics/http/views/#django.http.Http404) 异常。

我们稍后再讨论你需要在 **polls/detail.html** 里输入什么，但是如果你想试试上面这段代码是否正常工作的话，你可以暂时把下面这段输进去：

polls/templates/polls/detail.html[**¶**](https://docs.djangoproject.com/zh-hans/2.1/intro/tutorial03/#id8)

{{ question }}

### 一个快捷函数：[¶](https://docs.djangoproject.com/zh-hans/2.1/intro/tutorial03/#a-shortcut-get-object-or-404)[get\_object\_or\_404()](https://docs.djangoproject.com/zh-hans/2.1/topics/http/shortcuts/#django.shortcuts.get_object_or_404)

尝试用[**get()**](https://docs.djangoproject.com/zh-hans/2.1/ref/models/querysets/#django.db.models.query.QuerySet.get) 函数获取一个对象，如果不存在就抛出[**Http404**](https://docs.djangoproject.com/zh-hans/2.1/topics/http/views/#django.http.Http404)错误也是一个普遍的流程。Django也提供了一个快捷函数，下面是修改后的详情**detail()**视图代码：

polls/views.py [**¶**](https://docs.djangoproject.com/zh-hans/2.1/intro/tutorial03/#id9)

**from** **django.shortcuts** **import** get\_object\_or\_404, render

**from** **.models** **import** Question

*# ...*

**def** detail(request, question\_id):

question = get\_object\_or\_404(Question, pk=question\_id)

**return** render(request, 'polls/detail.html', {'question': question})

**#这里只是用**[**get\_object\_or\_404()**](https://docs.djangoproject.com/zh-hans/2.1/topics/http/shortcuts/#django.shortcuts.get_object_or_404)替换原有的捕获代码，所以效果不变

The [**get\_object\_or\_404()**](https://docs.djangoproject.com/zh-hans/2.1/topics/http/shortcuts/#django.shortcuts.get_object_or_404)function takes a Django model as its first argument and an arbitrary number of keyword arguments, which it passes to the [**get()**](https://docs.djangoproject.com/zh-hans/2.1/ref/models/querysets/#django.db.models.query.QuerySet.get)function of the model's manager. It raises [**Http404**](https://docs.djangoproject.com/zh-hans/2.1/topics/http/views/#django.http.Http404)if the object doesn't exist.

**设计哲学**

为什么我们使用辅助函数[**get\_object\_or\_404()**](https://docs.djangoproject.com/zh-hans/2.1/topics/http/shortcuts/#django.shortcuts.get_object_or_404)而不是自己捕获[**ObjectDoesNotExist**](https://docs.djangoproject.com/zh-hans/2.1/ref/exceptions/#django.core.exceptions.ObjectDoesNotExist)异常呢？还有，为什么模型API不直接抛出[**ObjectDoesNotExist**](https://docs.djangoproject.com/zh-hans/2.1/ref/exceptions/#django.core.exceptions.ObjectDoesNotExist)而是抛出[**Http404**](https://docs.djangoproject.com/zh-hans/2.1/topics/http/views/#django.http.Http404)呢？

因为这样做会增加模型层和视图层的耦合性。指导Django设计的最重要的思想之一就是要保证松散耦合。一些受控的耦合将会被包含在 **[django.shortcuts](https://docs.djangoproject.com/zh-hans/2.1/topics/http/shortcuts/" \l "module-django.shortcuts" \o "django.shortcuts: Convenience shortcuts that span multiple levels of Django's MVC stack.)**模块中。

也有[**get\_list\_or\_404()**](https://docs.djangoproject.com/zh-hans/2.1/topics/http/shortcuts/#django.shortcuts.get_list_or_404)函数，工作原理和[**get\_object\_or\_404()**](https://docs.djangoproject.com/zh-hans/2.1/topics/http/shortcuts/#django.shortcuts.get_object_or_404)一样，除了[**get()**](https://docs.djangoproject.com/zh-hans/2.1/ref/models/querysets/#django.db.models.query.QuerySet.get)函数被换成了[**filter()**](https://docs.djangoproject.com/zh-hans/2.1/ref/models/querysets/#django.db.models.query.QuerySet.filter)函数。如果列表为空的话会抛出[**Http404**](https://docs.djangoproject.com/zh-hans/2.1/topics/http/views/#django.http.Http404)异常。

## 使用模板系统[¶](https://docs.djangoproject.com/zh-hans/2.1/intro/tutorial03/#use-the-template-system)

回过头去看看我们的**detail()**视图。它向模板传递了上下文变量**question**。下面是**polls/detail.html**模板里正式的代码：

polls/templates/polls/detail.html [**¶**](https://docs.djangoproject.com/zh-hans/2.1/intro/tutorial03/#id10)

<**h1**>{{ question.question\_text }}</**h1**>

<**ul**>

{% **for** choice **in** question.choice\_set.all %}

<**li**>{{ choice.choice\_text }}</**li**>

{% **endfor** %}

</**ul**>

模板系统统一使用点符号来访问变量的属性。在示例中，首先Django尝试对 对象使用字典查找（也就是使用obj.get(str)操作），如果失败了就尝试属性查找（也就是obj.str操作），结果是成功了。如果这一操作也失败的话，将会尝试列表查找（也就是obj[int]操作）。**{{ question.question\_text}}question**

在 循环中发生的函数调用：被解释为Python代码，将会返回一个可迭代的对象，这一对象可以在 标签内部使用。**[{% for %}](https://docs.djangoproject.com/zh-hans/2.1/ref/templates/builtins/" \l "std:templatetag-for)question.choice\_set.allquestion.choice\_set.all()Choice**[**{% for%}**](https://docs.djangoproject.com/zh-hans/2.1/ref/templates/builtins/#std:templatetag-for)

查看[模板指南](https://docs.djangoproject.com/zh-hans/2.1/topics/templates/)可以了解关于模板的更多信息。

## 去除模板中的硬编码URL [¶](https://docs.djangoproject.com/zh-hans/2.1/intro/tutorial03/#removing-hardcoded-urls-in-templates)

还记得吗，我们在**polls/index.html**里编写投票链接时，链接是硬编码的：

<**li**><**a** href="/polls/{{ question.id }}/">{{ question.question\_text }}</**a**></**li**>

问题在于，硬编码和强耦合的链接，对于一个包含很多应用的项目来说，修改起来是十分困难的。然而，因为你在**polls.urls**的**[url()](https://docs.djangoproject.com/zh-hans/2.1/ref/urls/" \l "django.conf.urls.url" \o "django.conf.urls.url)**函数中通过name参数为URL定义了名字，你可以使用 标签代替它：**{%url %}**

<**li**><**a** href="{% **url** 'detail' question.id %}">{{ question.question\_text }}</**a**></**li**>

这个标签的工作方式是在**polls.urls**模块的URL定义中寻具有指定名字的条目。你可以回忆一下，具有名字'detail'的URL是在如下语句中定义的：

...

*# the 'name' value as called by the {% url %} template tag*

path('<int:question\_id>/', views.detail, name='detail'),

...

如果你想改变投票详情视图的URL，比如想改成**polls/specifics/12/**，你不用在模板里修改任何东西（包括其它模板），只要在**polls/urls.py**里稍微修改一下就行：

...

*# added the word 'specifics'*

path('specifics/<int:question\_id>/', views.detail, name='detail'),

...

## 为URL名称添加命名空间[¶](https://docs.djangoproject.com/zh-hans/2.1/intro/tutorial03/#namespacing-url-names)

教程项目只有一个应用，**polls**。在一个真实的Django项目中，可能会有五个，十个，二十个，甚至更多应用。Django如何分辨重名的URL呢？举个例子，**polls** 应用有**detail**视图，可能另一个博客应用也有同名的视图。Django如何知道标签到底对应哪一个应用的URL呢？**{% url %}**

答案是：在根URLconf中添加命名空间。在**polls/urls.py**文件中稍作修改，加上**app\_name**设置命名空间：

polls/urls.py [**¶**](https://docs.djangoproject.com/zh-hans/2.1/intro/tutorial03/#id11)

**from** **django.urls** **import** path

**from** **.** **import** views

app\_name = 'polls'

urlpatterns = [

path('', views.index, name='index'),

path('<int:question\_id>/', views.detail, name='detail'),

path('<int:question\_id>/results/', views.results, name='results'),

path('<int:question\_id>/vote/', views.vote, name='vote'),

]

现在，编辑**polls/index.html**文件，从：

polls/templates/polls/index.html [**¶**](https://docs.djangoproject.com/zh-hans/2.1/intro/tutorial03/#id12)

<**li**><**a** href="{% **url** 'detail' question.id %}">{{ question.question\_text }}</**a**></**li**>

修改为指向具有命名空间的详细视图：

polls/templates/polls/index.html [**¶**](https://docs.djangoproject.com/zh-hans/2.1/intro/tutorial03/#id13)

<**li**><**a** href="{% **url** 'polls:detail' question.id %}">{{ question.question\_text }}</**a**></**li**>

当你对你写的视图感到满意后，请阅读[教程的第4部分](https://docs.djangoproject.com/zh-hans/2.1/intro/tutorial04/)了解简单的表单处理和通用视图。