Django框架

# 请求与响应

View创建响应

Urls创建请求

# 二、模型与admin站点

配置数据库

创建模型

创建admin

# 三、视图与模板

首先分析要建立的网站需要哪些视图：

比如一个博客需要：

博客首页——展示最近的几项内容。

内容“详情”页——详细展示某项内容。

以年为单位的归档页——展示选中的年份里各个月份创建的内容。

以月为单位的归档页——展示选中的月份里各天创建的内容。

以天为单位的归档页——展示选中天里创建的所有内容。

评论处理器——用于响应为一项内容添加评论的操作。

比如投票应用需要：

问题索引页——展示最近的几个投票问题

问题详情页——展示某个问题和选项列表

问题结果页——展示某个投票的结果

投票处理器——用于响应用户为某个问题的特定选项投票的操作

分析出视图要求后，就可以在app的view中创建视图了。

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。

**备注**：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，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。

**快捷函数**

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。

注意：使用render函数，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。

**抛出错误**

就是在视图函数中，使用try except

**去除模板中的硬编码url**

硬编码：指的是在模板中写入页面的路径

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

非硬编码方式：

**备注：**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 {% url %} 标签代替它：<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。

**为url名称添加命名空间**

Django如何分辨重名的URL呢？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：

# 四、表单与通用视图

**编写一个简单表单**

简要说明：

上面的模板在 Question 的每个 Choice 前添加一个单选按钮。 每个单选按钮的 **value** 属性是对应的各个 Choice 的 ID。每个单选按钮的 **name** 是 **"choice"** 。这意味着，当有人选择一个单选按钮并提交表单提交时，它将发送一个 POST 数据 **choice=#** ，其中# 为选择的 Choice 的 ID。这是 HTML 表单的基本概念。

我们设置表单的 **action** 为 **{% url 'polls:vote' question.id %}** ，并设置 **method="post"** 。使用 **method="post"``（与其相对的是 ``method="get"`）是非常重要的，因为这个提交表单的行为会改变服务器端的数据。 无论何时，当你需要创建一个改变服务器端数据的表单时，请使用``method="post"** 。这不是 Django 的特定技巧；这是优秀的网站开发技巧。

**forloop.counter** 指示 [**for**](https://docs.djangoproject.com/zh-hans/2.1/ref/templates/builtins/#std:templatetag-for) 标签已经循环多少次。

由于我们创建一个 POST 表单（它具有修改数据的作用），所以我们需要小心跨站点请求伪造。 谢天谢地，你不必太过担心，因为 Django 已经拥有一个用来防御它的非常容易使用的系统。 简而言之，所有针对内部 URL 的 POST 表单都应该使用 [**{% csrf\_token %}**](https://docs.djangoproject.com/zh-hans/2.1/ref/templates/builtins/#std:templatetag-csrf_token) 模板标签。

创建视图来处理提交的数据

**使用通用视图：代码越少越好**

这些视图反映基本的 Web 开发中的一个常见情况：根据 URL 中的参数从数据库中获取数据、载入模板文件然后返回渲染后的模板。 由于这种情况特别常见，Django 提供一种快捷方式，叫做“通用视图”系统。

通用视图将常见的模式抽象化，可以使你在编写应用时甚至不需要编写Python代码。

让我们将我们的投票应用转换成使用通用视图系统，这样我们可以删除许多我们的代码。我们仅仅需要做以下几步来完成转换，我们将：

转换 URLconf。

删除一些旧的、不再需要的视图。

基于 Django 的通用视图引入新的视图。

**备注：**一般来说，当编写一个 Django 应用时，你应该先评估一下通用视图是否可以解决你的问题，你应该在一开始使用它，而不是进行到一半时重构代码。本教程目前为止是有意将重点放在以“艰难的方式”编写视图，这是为将重点放在核心概念上。

**通用视图：**

两个通用视图： [ListView](https://docs.djangoproject.com/zh-hans/2.1/ref/class-based-views/generic-display/" \l "django.views.generic.list.ListView" \o "django.views.generic.list.ListView) 和 [DetailView](https://docs.djangoproject.com/zh-hans/2.1/ref/class-based-views/generic-display/" \l "django.views.generic.detail.DetailView" \o "django.views.generic.detail.DetailView) 。这两个视图分别抽象“显示一个对象列表”和“显示一个特定类型对象的详细信息页面”这两种概念。

每个通用视图需要知道它将作用于哪个模型。 这由 **model** 属性提供。

[**DetailView**](https://docs.djangoproject.com/zh-hans/2.1/ref/class-based-views/generic-display/#django.views.generic.detail.DetailView) 期望从 URL 中捕获名为 **"pk"** 的主键值，所以我们为通用视图把 **question\_id** 改成 **pk** 。

# 五、测试

在复杂的应用程序中，组件之间可能会有数十个复杂的交互。在更加复杂的应用中，各种组件之间的交互可能会及其的复杂。改变其中某一组件的行为，也有可能会造成意想不到的结果。判断「代码是否正常工作」意味着你需要用大量的数据来完整的测试全部代码的功能，以确保你的小修改没有对应用整体造成破坏——这太费时间了。

**测试的优点：**

测试不仅能发现错误，而且能预防错误

测试使你的代码更有吸引力

测试有利于团队协作

**基础测试策略：**

# 六、静态文件

除了服务端生成的 HTML 以外，网络应用通常需要一些额外的文件——比如图片，脚本和样式表——来帮助渲染网络页面。在 Django 中，我们把这些文件统称为“静态文件”。

**静态文件命名空间**

虽然我们 可以 像管理模板文件一样，把 static 文件直接放入 **polls/static** ——而不是创建另一个名为 **polls**的子文件夹，不过这实际上是一个很蠢的做法。Django 只会使用第一个找到的静态文件。如果你在 其它 应用中有一个相同名字的静态文件，Django 将无法区分它们。我们需要指引 Django 选择正确的静态文件，而最简单的方式就是把它们放入各自的 命名空间 。也就是把这些静态文件放入 另一个 与应用名相同的目录中。

# 七、自定义admin站点

Django框架的关键结构

# 一、模型层

# 二、视图层

# 三、模板

# 四、表单

# 五、开发进程

# 六、管理

# 七、安全

# 八、国际化与本地化

# 九、性能与优化

# 十、地理框架

# 十一、常用的web应用工具

# 十二、其它核心功能

# 十三、django开源项目