

V1.0 版

Django2 萌新入门笔记

小楼一夜听春雨·作品

www.opython.com

2018 年 6 月 26 日

提示：如果文章中有错误或遗漏请参考原文链接，欢迎到网站留言指正！

Django2 : Web 项目开发入门笔记 (1)

在学习了 Python 入门教程之后，我们应该选择一个方向进行深入。

从这篇教程开始，我们一起学习 Python 的 Web 项目开发。

基于 Python 的 Web 开发框架有很多种，在这里不一一赘述。

Django 是这些框架中比较成熟并拥有庞大用户群体的一种。

与一些轻量级框架（例如 Flask）不同，Django 有丰富的内建功能，提供一站式解决方案，便于初学者快速实现一个 Web 项目。

当然，并不是说轻量级框架不如 Django，轻量级框架更适合灵活的定制化开发，但同时也意味着很多功能要由开发人员自己来实现。

这就好比想做一道菜。

轻量级框架就像蔬菜店，只提供基本的食材，这就要求顾客具有一定的烹饪知识和技巧，能够自己加工搭配食材，完成烹饪过程。

不过，这样的好处在于顾客能够随心所欲的进行搭配制作，做出各式各样的菜品。

而 Django 这样的框架则像一个半成品供应店，食材都已经加工好，并且按配方搭配好，顾客只需要按照一定流程，就能够完成菜品的制作。这对于烹饪知识和技巧有限或者希望快速做出一道菜的顾客来说是不错的选择。

但是，半成品的缺点在于限制了顾客的自由度，不利于顾客的自由发挥。

所以，作为一个入门级的 Python 开发人员，选择 Django 应该是个不错的决定。

这里需要注意的是，在之前的学习中，我们一直使用的是 PyCharm 的社区版，如果要进行基于 Django 的 Web 项目开发，我们需要使用 PyCharm 的专业版。

PyCharm 专业版，才能够直接创建 Django 的项目。

PyCharm 专业版下载地址：

<https://www.jetbrains.com/zh/pycharm/download/download-thanks.html>

注意：PyCharm 专业版只提供一定时间的试用，大家可以考虑购买，或者通过一些其他方式完成软件的激活，本站不提供相关的解决方案。

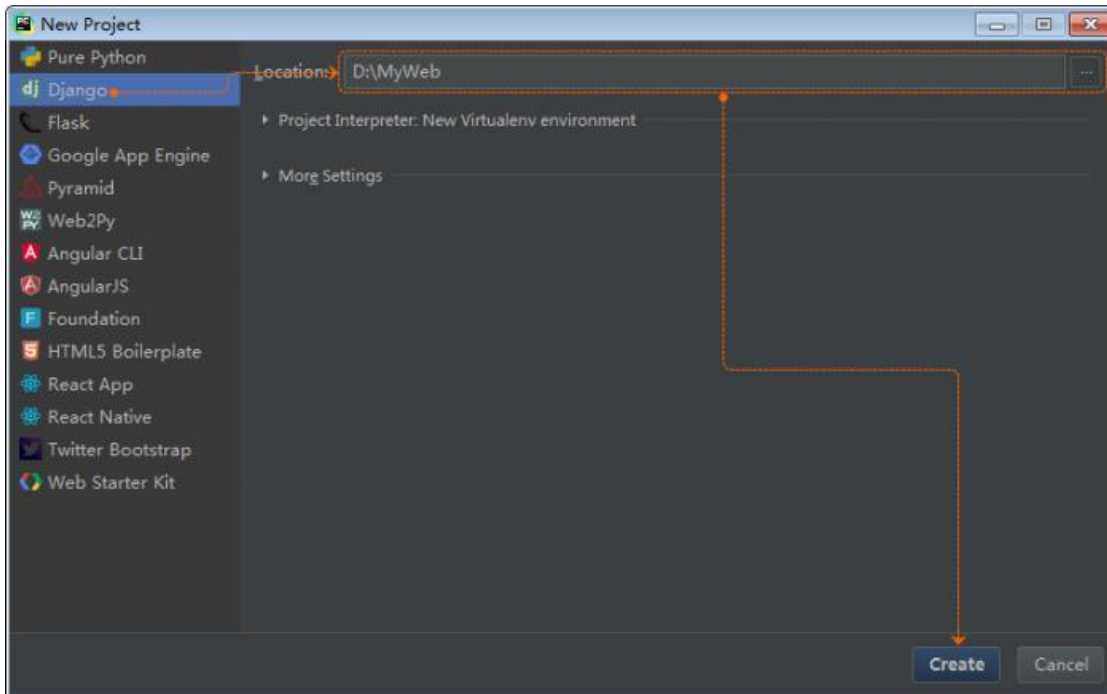
当我们完成 PyCharm 专业版的安装之后，将软件打开。

如果之前已经做过项目的话，此时可能会直接打开之前的项目。

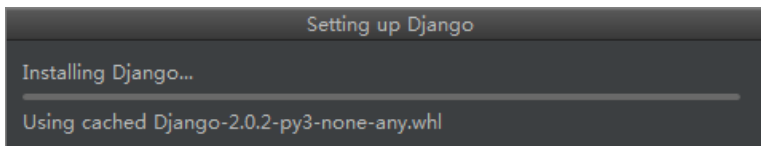
我们需要在文件（File）菜单中，选择关闭项目（Close Project）的选项，关闭当前项目。

关闭项目之后，会弹出创建项目的窗口，我们点击创建新项目（Create New Project）的选项，进入创建新项目的窗口。

在创建新项目的窗口中，左侧列表选择 Django，右侧选择一个本地已创建好的文件夹，然后点击创建（Create）按钮，就会开始 Django 项目的创建。



在 Django 项目的创建过程中，如果我们还没有安装 Django 的话，PyCharm 会自动帮助我们进行安装。

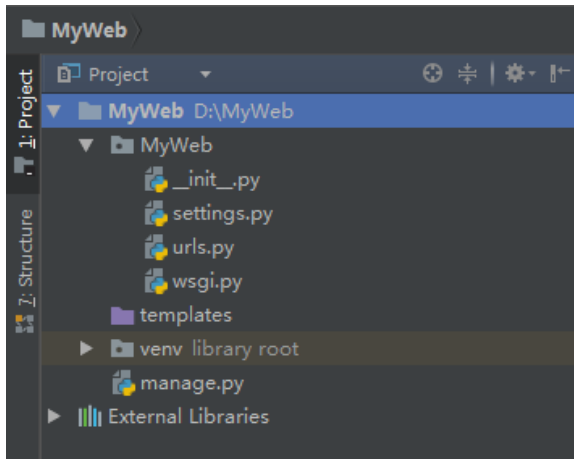


注意：PyCharm 会自动帮助我们安装 Django2.0 版本，这个版本不支持 Python2。

如果想自己安装 Django 的话，也可以使用 pip 命令进行安装。

`pip install Django`

当我们完成 Django 项目的创建，这个时候在 PyCharm 左侧的项目文件列表中会自动创建一些内容。



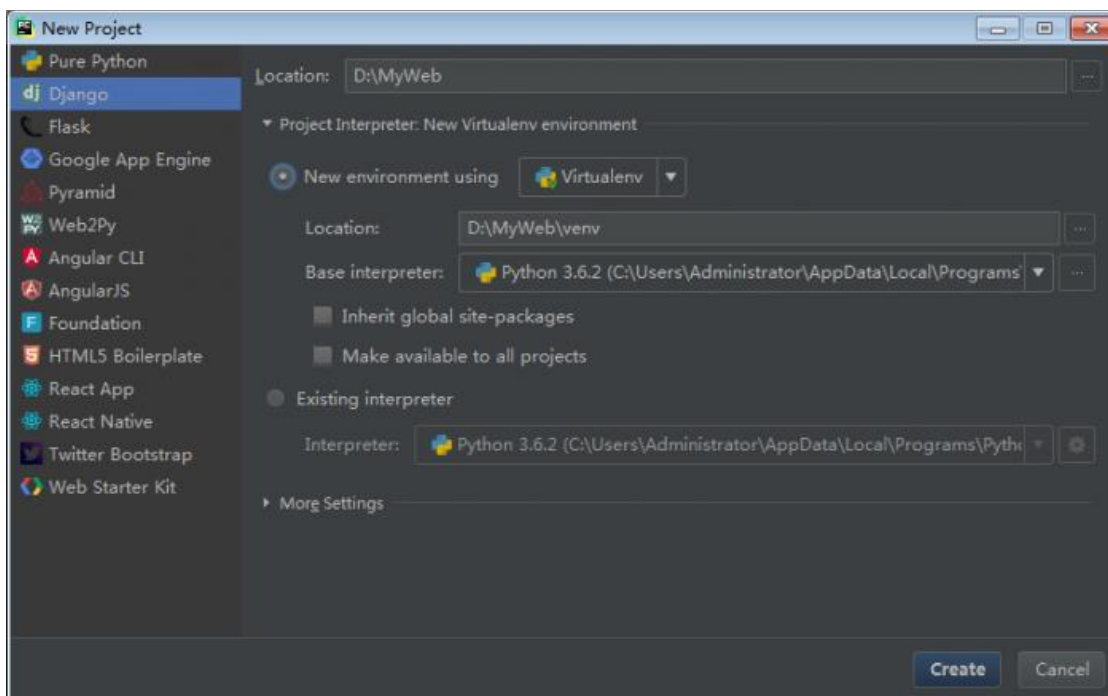
这些内容就是我们基于 Django 开发 Web 项目的关键内容。

至于这些内容如何使用，我们在之后的教程中逐步进行了解。

这里需要知道的是 `venv` 这文件夹中的内容。

这个文件夹内容的出现，是因为在创建 Django 项目时，同时创建了一个独立的 Python 运行环境。

在创建 Django 项目的窗口，Python 解释器设置中有一个默认选项，使用 Virtualenv 新建运行环境（New environment using Virtualenv）。



实际上，如果我们只开发一个应用的话，可以选择下方的现有翻译器（Existing interpreter）。

但是，如果同时开发多个应用的话，每个应用又需要独立的运行环境，就需要选择创建新的虚拟环境。

例如，系统中只安装了 Python2.7，安装第三方库的时候都会安装到同一目录。

但是同时需要开发两个项目，一个使用 Django1.7，另一个使用 Django1.9，这就产生了冲突。

所以，这个时候我们需要为不同的项目创建独立的运行环境，就需要用到 Virtualenv。

实际上 virtualenv 也是一个模块，通过 pip 命令进行安装，然后通过这个模块手动创建虚拟环境。

关于 virtualenv 模块的使用，有兴趣的同学可以自行搜索一下相关资料进行了解，在此不做赘述。

转载请注明：魔力 Python » Django2 : Web 项目开发入门笔记（1）

原文链接：<http://www.opython.com/855.html>

Django2 : Web 项目开发入门笔记（2）

在开始使用 Django2 之前，我们需要先了解一些关于 Web 框架的相关知识，这对我们了解 Django 的工作原理以及能够顺利的学习 Django 有很大的帮助。

首先，我们要了解的是设计模式和框架的区别。

设计模式是对在某种环境中反复出现的问题以及解决该问题的方案的描述（引自百度百科）。

上面对设计模式这个定义，看上去很难理解。

所以，我举一个常见的例子。

就拿盖房子（某种环境）来说，都要建造地基、墙体和屋顶（反复出现的问题），我们需要先打好地基，砌好墙体，再装上屋顶（解决问题的方案）。

这就是盖房子的设计模式。

而框架则是按照设计模式完成的半成品，在框架的基础上可以进行二次开发。

还是拿盖房子来说，框架就好像根据上面盖房子的设计模式所建造出来的只有地基、四面墙体和屋顶的毛坯房，如何增加新的墙体与装修都可以由房主自己再进行处理。

在编程领域，通常会听到 MVC 框架。

实际上，MVC 框架是基于 MVC 这种创建 Web 应用程序的设计模式所开发出来的框架。

那么，MVC 设计模式是什么样的呢？

先来看这个英文缩写每一个字母所代表的内容：

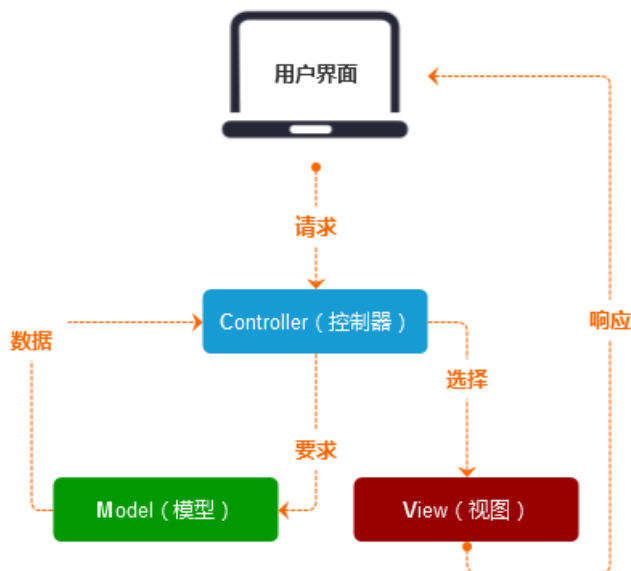
- M : Model（模型）
- V : View（视图）
- C : Controller（控制器）

Model（模型）是应用程序中用于处理应用程序数据逻辑的部分，通常模型对象负责在数据库中存取数据。

View（视图）是应用程序中处理数据显示的部分，通常视图是依据模型数据创建的。

Controller（控制器）是应用程序中处理用户交互的部分，通常控制器负责从视图读取数据，控制用户输入，并向模型发送数据。

大家可以通过下面这张图，了解 MVC 模式的组成结构与关系。



关于 MVC 设计模式以及基于 MVC 的框架，大家可以自行查找相关资料进行深入的了解。

这里之所以提及这个设计模式，是因为我们所使用的 Django 这个框架的设计模式与之非常相似。

Django 是标准的基于 MTV 设计模式的框架。

这里的 MTV 和我们所熟知的音乐电视(Music Television)没有任何关系。

这个英文缩写每一个字母所代表的内容，如下所示：

- M : Model (模型)
- T : Templates (模板)
- V : Views (视图)

大家看完上面的英文缩写所代表的内容，会发现多了一个 Template (模板) 却少了一个 Controller (控制器)。

实际上并没有缺少 Controller (控制器)，Django 框架本身承担了控制器的角色。

Django 处理请求的顺序如下：

1、通过 Web 服务器网关接口 (WSGI : Web Server Gateway Interface) 对 Web 服务器的 socket 请求进行处理；

提示：WSGI 是 Python 应用程序或框架和 Web 服务器之间的一种接口。

2、Django 框架 (控制器) 控制用户输入，进行 URL 匹配，通过映射列表将请求发送到合适的视图；

3、视图 (Views) 向模型 (Model) 和模板 (Templates) 发送或获取数据；

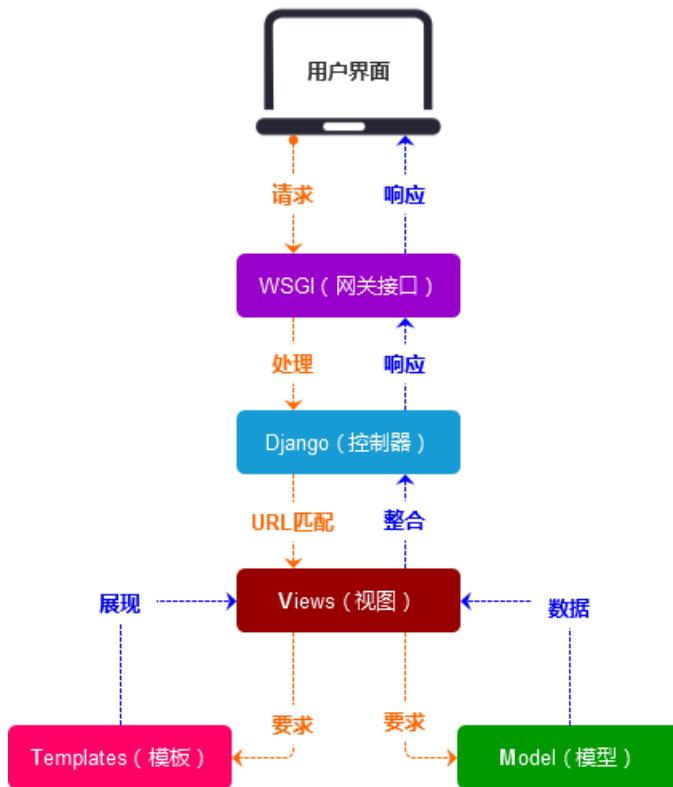
4、模型 (Model) 对数据库进行存取数据；

5、模板 (Templates) 用于将内容和展现分离，通过模板描述数据如何展现；

6、视图 (Views) 将模板和数据整合，形成最终页面；

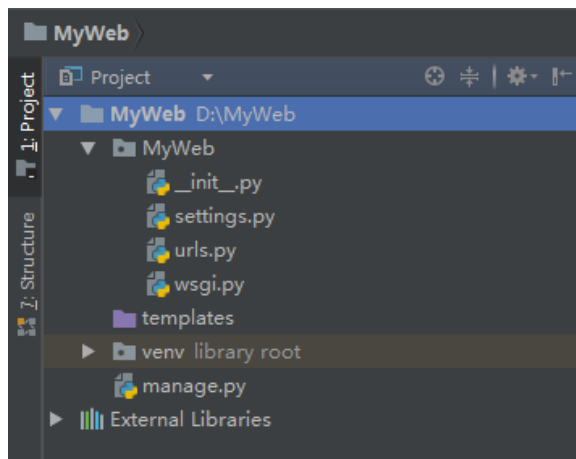
7、Django 框架 (控制器) 返回页面展示给用户。

大家可以通过下面这张图，了解 Django 处理请求的过程。



简单来说，Django 是将每个 URL 的页面请求分发给不同的 View（视图）进行处理，View（视图）再调用相应的 Model（模型）进行数据的保存或读取，并且 View（视图）也会调用相应 Template（模板），将 Model（模型）读取到的数据与 Template（模板）进行整合，形成最终页面后返回给控制器，展示给用户。

通过对上面所述内容的了解，我们再回到 PyCharm 中看一下，Django 项目自动生成的内容，就会基本明白其中一些内容的用途。



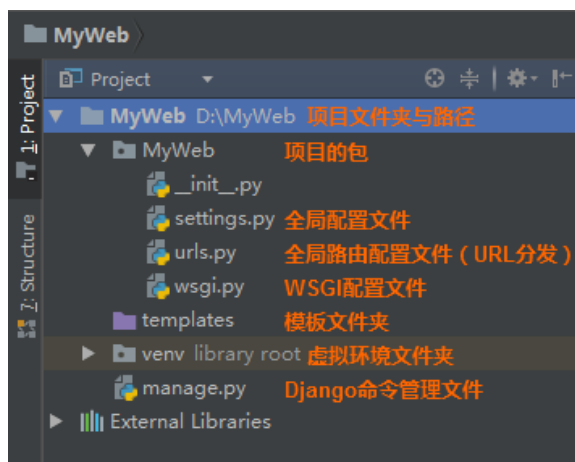
还有一些内容（例如 manage.py）在之后的教程中，我们再进行了解。

转载请注明：魔力 Python » Django2：Web 项目开发入门笔记（2）

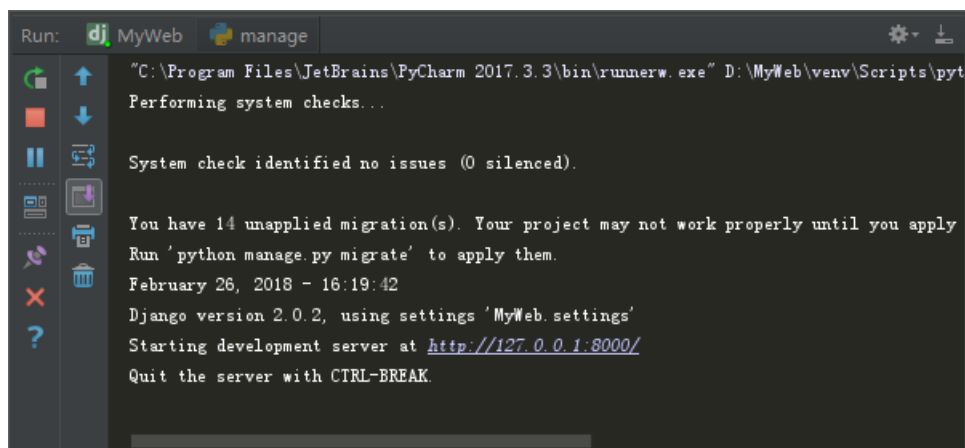
原文链接：<http://www.opython.com/875.html>

Django2：Web 项目开发入门笔记（3）

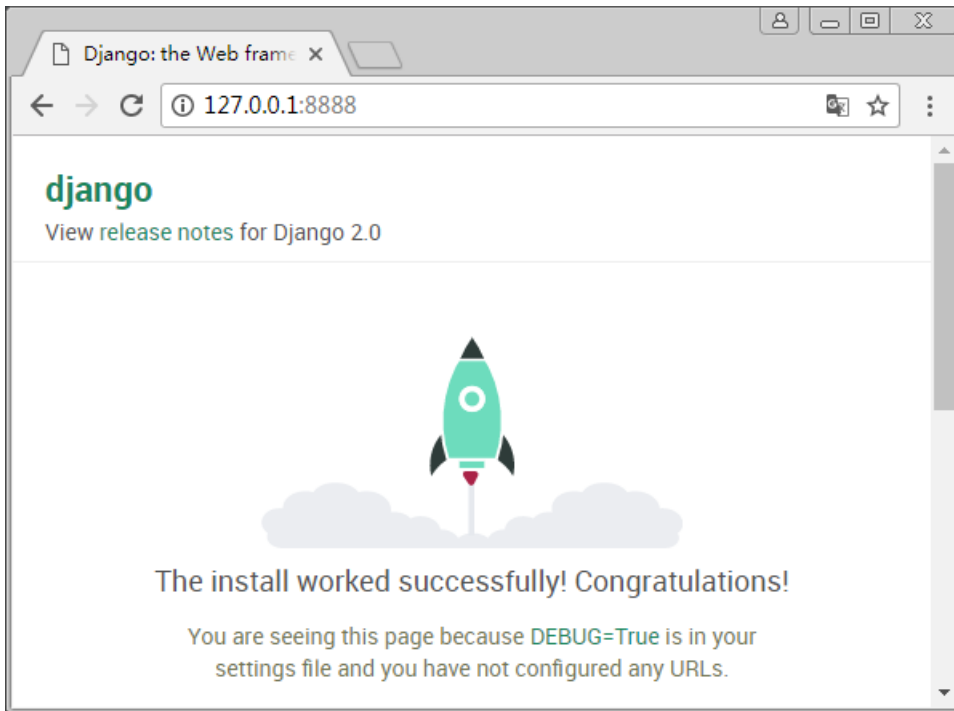
使用 PyCharm 创建 Django 的 Web 项目非常方便，在之前的教程中我们能够看到 PyCharm 已经帮我们创建了一些内容。



我们在列表中点击项目的包“MyWeb”，然后运行，就能够启动开发服务器。



此时，我们打开浏览器，输入地址“<http://127.0.0.1:8000/>”，就能够看到 Django 安装成功的提示页面。



不过，这仅仅是 Django 的页面内容，如果想用 Django 开发一个真正可以访问的 Web 应用，我们需要在项目中创建一个新的应用。

创建应用可以通过命令行进行创建。

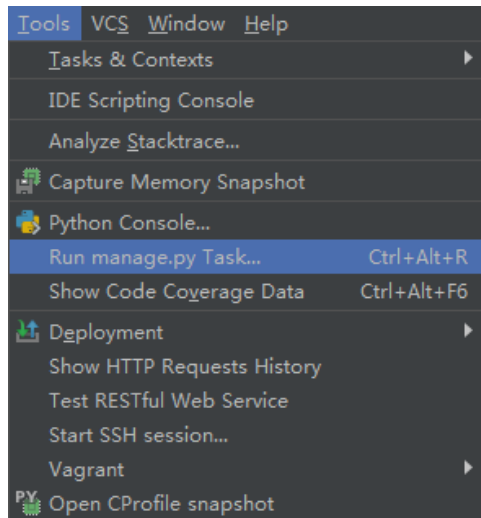
例如，创建一个名称为“MySite”的应用，命令为：

```
python manage.py startapp MySite
```

先别急！

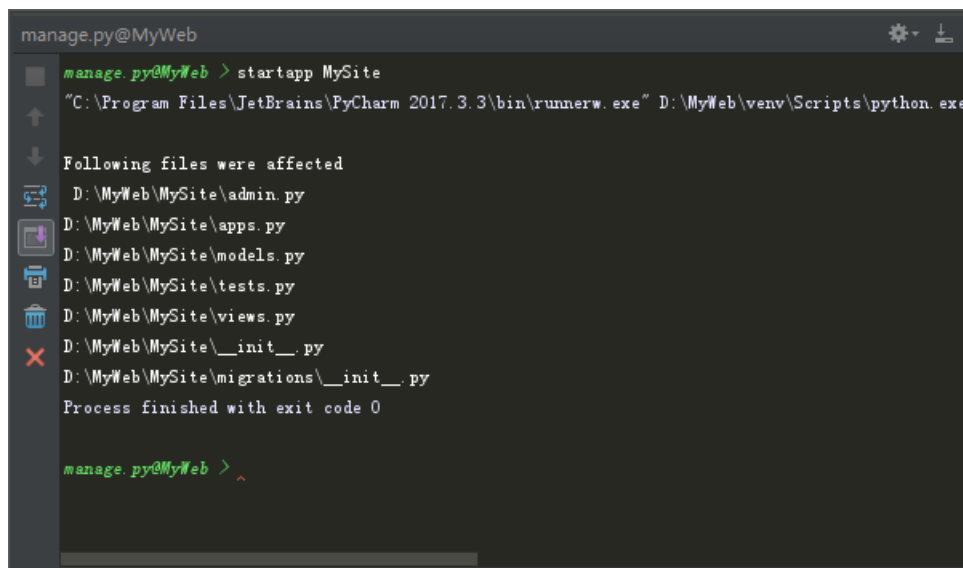
既然我们使用了 PyCharm 这个开发环境，创建应用还有另外一种方式。

在工具（Tools）菜单中，选择运行任务（Run manage.py Task）。



此时会启动命令行窗口。

在命令行窗口中，我们输入“startapp MySite”，回车之后即可完成应用的创建。



“manage.py”文件包含很多命令，不仅仅可以创建应用，还能对服务器、数据库以及会话等等进行相关操作。

如果了解这些命令，可以直接运行“manage.py”文件，就能够看到相关的帮助信息。

Available subcommands:

[auth]change_passwordcreatesuperuser

[contenttypes]remove_stale_contenttypes

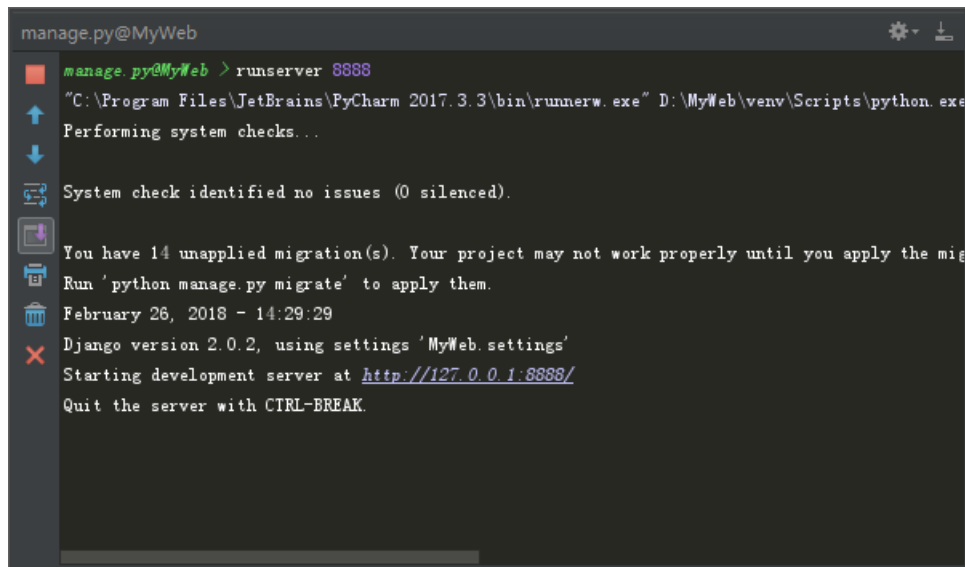
```
[django]check compilemessages createcachetable dbshell diffsettings dumpdata flush inspectdb  
loaddata makemessages makemigrations migrate sendtestemail shell showmigrationssqlflush  
sqlmigrate sqlsequencereset squashmigrations startapp startproject test testserver
```

```
[sessions]clearsessions
```

```
[staticfiles]collectstatic findstatic runserver
```

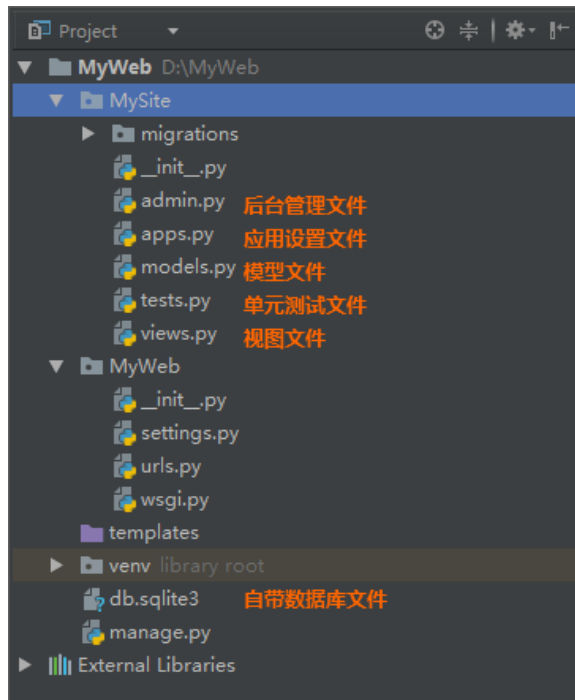
例如刚才启动开发服务器的操作，我们也可以通过“manage.py”文件来完成。

在命令行窗口，我们输入“runserver 端口号”就能够启动开发服务器。



```
manage.py@MyWeb  
manage.py@MyWeb > runserver 8888  
"C:\Program Files\JetBrains\PyCharm 2017.3.3\bin\runnerw.exe" D:\MyWeb\venv\Scripts\python.exe  
Performing system checks...  
System check identified no issues (0 silenced).  
You have 14 unapplied migration(s). Your project may not work properly until you apply the mig  
Run 'python manage.py migrate' to apply them.  
February 26, 2018 - 14:29:29  
Django version 2.0.2, using settings 'MyWeb.settings'  
Starting development server at http://127.0.0.1:8888/  
Quit the server with CTRL-BREAK.
```

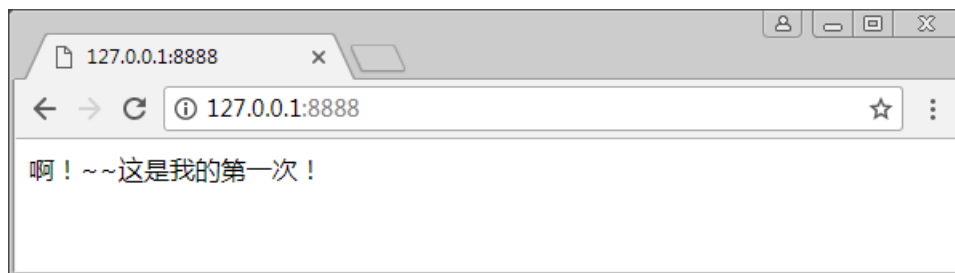
当我们完成应用的创建，此时在项目文件列表中又会增加一些新的内容。



到这里，我们终于看到了 MTV 框架中的模型（Models）和视图（Views）。

接下来，我们基于已经生成的内容，尝试着做一个首页。

内容不用很复杂，只需要一句话就可以。



从这个练习开始，我们逐步了解 Django 的使用。

1、新增视图函数。

视图函数用于返回响应内容，也就是用户看到的页面。

在“views.py”文件中添加新的代码（带注释部分），定义 `index(request)` 函数，参数 `request` 是必需的。

```
from django.shortcuts import render # 暂时没有作用
from django.http import HttpResponse # 从 http 模块中导入 HttpResponse 类
```

```
# Create your views here.
```

```
def index(request): # 定义站点首页视图函数
```

```
    return HttpResponse('啊！~~这是我的第一次！') # 返回响应内容对象
```

完成上方代码后，当调用 `index` 函数时，就能够将一个页面内容的对象返回给用户。

2、配置网址分发。

在“`urls.py`”文件中添加新的代码（带注释部分），将访问网站根目录的 `url` 交由视图中的 `index` 函数进行处理。

```
from django.contrib import admin
```

```
from django.urls import path
```

```
from MySite import views as siteviews # 从项目的包中导入视图模块
```

```
urlpatterns = [
```

```
    path("", siteviews.index), # 来自服务器的请求为网站根目录时，由视图中的 index 函数进行处理。
```

```
    path('admin/', admin.site.urls),
```

```
]
```

这样，当访问“`http://IP 地址:端口号`”时，会通过“`urls.py`”文件进行分发，调用视图中的 `index` 函数，得到返回的响应内容对象。

当我们完成以上两步，就可以启动开发服务器，设置端口号为“8888”，并且通过“`http://127.0.0.1:8888/`”访问了。

除此以外，大家还可以通过“`http://127.0.0.1:8888/admin/`”进行访问，是不是看到了 Django 自带的管理后台？

之所以能够打开后台页面，就是由上方代码中“`path('admin/', admin.site.urls),`”这一句进行分发的。吧

另外还要补充一点：如果项目中同时有多个应用的话，应该在每个应用的包中单独创建“`urls.py`”模块，然后在项目的“`urls.py`”模块中包含应用的 URL 分发配置。

示例代码：

```
from django.contrib import admin
```

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

```
urlpatterns = [
```

```
    path('应用的包名/', include('应用的包名.urls')),
```

```
path('admin/', admin.site.urls),  
]
```

这样就能够在用户访问某一个应用时，调用该应用的 URL 分发配置。

好了，这篇教程我们先学习到这里，谢谢大家的支持与鼓励！

转载请注明：魔力 Python » Django2：Web 项目开发入门笔记（3）

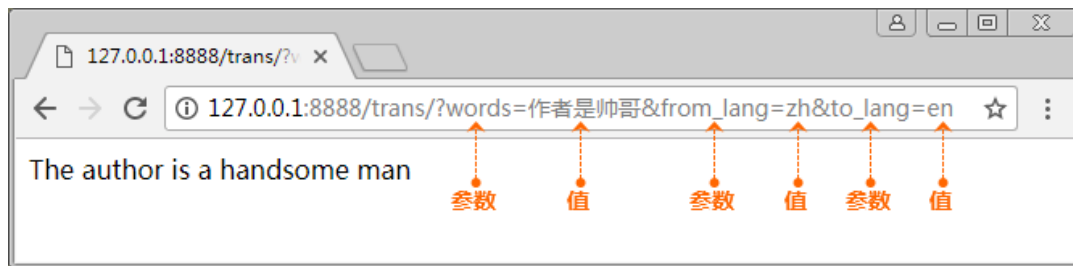
原文链接：<http://www.opython.com/880.html>

Django2：Web 项目开发入门笔记（4）

这一篇教程，我们通过 URL 分发（urls.py）和视图（views.py），做一个有意思的小功能。

通过 URL 请求中附带参数（问号“?”后方），进行翻译。

如下图所示：



如果想实现翻译功能，我们需要百度翻译的接口，并且通过下方代码实现百度翻译接口的调用。

示例代码：（trans.py）

```
from urllib.parse import quote  
from hashlib import md5  
from http import client  
import random  
import json
```

```
appid = '通过百度翻译开发者平台申请获取'  
secret_key = '通过百度翻译开发者平台申请获取'
```

```
def get_sign(salt, qurey): # 生成调用 API 的签名  
    sign = appid + qurey + str(salt) + secret_key  
    m = md5()  
    m.update(sign.encode('utf-8'))
```

```
return m.hexdigest()
```

```
def trans(quiry, from_lang='zh', to_lang='en'): # 实现翻译功能
    http_client = None
    salt = random.randint(12345, 67890)
    sign = get_sign(salt, query)
    myurl = '/api/trans/vip/translate' + '?appid=' + appid + '&q=' + quote(
        query) + '&from=' + from_lang + '&to=' + to_lang + '&salt=' + str(salt) + '&sign=' + sign
    try:
        http_client = client.HTTPConnection('api.fanyi.baidu.com') # 连接 API
        http_client.request('GET', myurl) # 发起请求
        response = http_client.getresponse() # 获取响应对象
        content = json.loads(response.read()) # 将调用 API 的返回结果转为字典
        return content['trans_result'][0]['dst'] # 返回翻译内容
    except Exception as e:
        return e
    finally:
        if http_client:
            http_client.close()
```

申请开发者 APP ID 和密钥流程：

- 1、进入百度翻译开放平台：<http://api.fanyi.baidu.com/api/trans/product/index>
- 2、打开页面后，点击【立即使用】按钮，申请开发者 APP ID 和密钥。
- 3、申请成功后，在开发者信息界面中即可看到 APP ID 和密钥。



当我们完成上方的“trans.py”模块，我们将其放入项目的包（MyWeb）中，并在视图模块（views.py）导入这个模块。

示例代码：

```
from MyWeb import trans # 导入翻译模块
```

然后，在视图模块（views.py）中我们添加一个视图函数。

通过 `request.GET['参数名称']` 获取 URL 中的参数值，并调用“trans.py”模块中的 `trans()` 函数，返回翻译结果。

特别提醒：一定要注意“request.GET”后面是方括号“[]”。

示例代码：（views.py）

```
def translate(request): # 定义视图函数
    from_lang = request.GET['from_lang'] # 获取 URL 中的参数
    to_lang = request.GET['to_lang'] # 获取 URL 中的参数
    text = request.GET['words'] # 获取 URL 中的参数
    return HttpResponse(trans.trans(text, from_lang, to_lang)) # 返回响应内容对象
```

接下来，我们再进行 URL 分发的配置。

打开“urls.py”模块，在 `urlpatterns` 列表中添加一个新的配置。

示例代码：（urls.py）

```
path('trans/', siteviews.translate), # 通过?传递参数进行处理
```

注意：因为新增加的配置是列表中的一项，代码的末尾要加上英文半角的逗号。

完成这一步之后，我们就可以启动开发服务器（如果忘记了，请参考上一篇教程），并在浏览器中输入 URL，查看效果。

例如：`http://127.0.0.1:8888/trans/?words=作者是帅哥&from_lang=zh&to_lang=en`

这是一种比较常见的 URL 传递参数的方式，例如百度搜索。

下面，我们再来看一种传递参数的方式。

这种方式不通过问号附带参数，而是以下图这种形式。



在我们之前学习自定义函数时，我们接触过关键字参数和位置参数两种。

如果说上一种附带参数的方式比较像关键字参数的话，当前这一种就比较像位置参数。

当前这种附带参数方式的参数位置都是固定的。

首先，我们还是在视图模块（`views.py`）中添加一个视图函数。

不过，这一次 URL 中参数值的获取是直接通过函数的参数进行获取。

示例代码：（`views.py`）

```
def translate2(request, words, from_lang, to_lang): # 定义视图函数
    return HttpResponse(trans.trans(words, from_lang, to_lang)) # 返回响应内容对象
```

然后，是 URL 分发的配置。

打开“`urls.py`”模块，在 `urlpatterns` 列表中添加一个新的配置。

示例代码：（`urls.py`）

```
path('trans/<str:from_lang>/<str:to_lang>/<str:words>', siteviews.translate2), # 通过定义
url 指定部分为参数进行处理
```

如上方代码所示，我们可以在配置中指定路径中的固定位置为不同的参数。

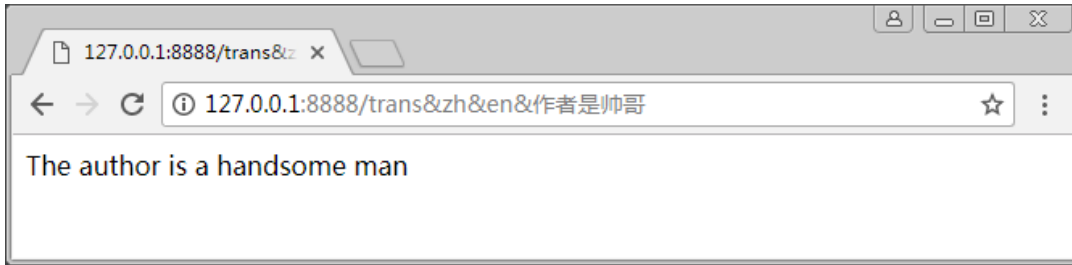
需要注意的是这里使用的是<类型：参数名称>的格式。

类型会限制参数的输入，共有 5 种类型：

- `int`：正整数，包含 0。
- `str`：除路径分隔符“/”之外的非空字符串。
- `slug`：由字母、数字、横线以及下划线其中一种或多种组成的字符串。
- `uuid`：通用唯一识别码（Universally Unique Identifier），包含 32 个 16 进制数字，以连字号分为五段，形式为 8-4-4-4-12 的 32 个字符。
- `path`：任何非空字符串，包含路径分隔符“/”。

另外，参数名称要和视图函数中的参数名称保持一致。

那么，如下图所示这种 URL，如何进行分发？



相信大家很快能得出答案！

示例代码：

```
path('trans<str:from_lang>&<str:to_lang>&<str:words>', siteviews.translate2), # 通过定义 url 指定部分为参数进行处理
```

最后，我们再来看一种 URL 分发的配置方法。

这一次的 URL 如下图所示：



还是使用上一个视图函数，我们在“urls.py”模块中导入一个新的方法。

示例代码：

```
from django.urls import re_path # 导入通过正则表达式处理路径的方法
```

然后，在配置中添加一行新的代码。

示例代码：

```
re_path('trans/(.+)&(.)-(.)$', siteviews.translate2), # 通过正则表达式获取 url 匹配部分为参数进行处理
```

如上方代码所示，这种方法是通过正则表达式获取路径中的参数，并传递给视图函数进行处理。

需要注意的是 URL 路径中的参数位置顺序必须与视图中的参数顺序相同。

本节练习源代码：[【点此下载】](#)

转载请注明：魔力 Python » Django2 : Web 项目开发入门笔记 (4)

原文链接：<http://www.opython.com/891.html>

Django2 : Web 项目开发入门笔记 (5)

前面的教程我们对 Django2 中的 URL 分发以及视图进行了初步的了解，这一篇教程我们接触模板的使用。



既然是创建 Web 项目，就少不了 HTML 代码。

每个呈现给用户的页面，都是由 HTML 代码所组成。

所以，对于 HTML 的相关知识，大家需要有一定的了解。

网上有很多关于 HTML 的教程，在此不做推荐。

另外一点，既然我们学习 Web 项目的开发，大家可以观察一下各类网站，页面是有分类的。

例如，一个新闻网站，会有不同种类新闻的列表页，还有各种新闻的内容页。

我们在制作网站的时候，肯定不会把每个 HTML 页面都写一遍，因为同类页面中的内容，往往只是有部分差异，大部分都是相同的。

例如，新闻列表页。

每一类新闻的列表页，只是列表内容不同，页面的其它部分往往是一样的。

那么，我们能不能把新闻列表页面只做一次，让里面的列表内容是可以动态加载的呢？

比如用户点击体育新闻就加载体育新闻的列表，点击经济新闻就加载经济新闻的列表。

当然是可以的，这就是模板的用途。

具体来说，模板可以是同一类页面的模板，也可以是不同页面中相同内容的模板。

也就是说，我们可以把同类页面只编写一次 HTML 代码，就可以通过动态载入数据呈现出不同的页面内容；也可以把多个页面中都存在的部分内容（例如页面导航和底部信息）只编写一次 HTML 代码，通过模板的嵌套加载到不同的页面中。

一、模板的使用

我们只需要创建包含 HTML 代码的模板文件，放置到模板文件夹（`templates`）中，然后在全局设置文件（`settings.py`）中添加相关配置，即可在视图中调用模板，并且通过视图函数可以将不同的数据与调用模板进行整合，形成不同的页面。

1、创建模板文件

在 PyCharm 项目文件列表的模板文件夹（`templates`）上点击鼠标右键，选择新建（New）一个 HTML 文件（HTML File）。

虽然，模板文件的后缀名称不要求必须是“.html”，但是在 PyCharm 中创建 HTML 文件会方便我们编写 HTML 代码（自动提示和代码补全）。

例如，我们创建一个名为“index.html”的文件，作为站点首页的模板，并写入 HTML 代码。

示例代码：

```
<!DOCTYPE html>
<html lang="zh">
<head>
  <meta charset="UTF-8">
  <title>首页</title>
</head>
<body>
<h4>这是我的第一个模板页面。</h4>
</body>
</html>
```

2、添加视图函数

打开视图模块“`views.py`”，添加视图函数调用模板。

示例代码：

```
from django.shortcuts import render # 创建应用时自动导入的模块
```

```
def index(request): # 定义站点首页视图函数
    return render(request, 'index.html') # 调用模板返回页面内容
```

3、配置 URL 分发

打开模块“urls.py”，在配置列表中添加新的配置。

示例代码：

```
from django.contrib import admin
from django.urls import path
from MySite import views as siteviews # 导入视图模块并创建别名

urlpatterns = [
    path('', siteviews.index), # 配置处理访问网站根目录的视图
    path('admin/', admin.site.urls),
]
```

完成上面的步骤后，启动开发服务器，通过浏览器访问当前项目，就能够看到页面内容了。



二、模板的嵌套

在很多页面都有的重复内容，我们可以做成模板，然后嵌套到页面中使用。

例如，我们新建两个 HTML 文件，一个作为顶部导航，一个作为底部信息，让 index 页面的模板中包含这两个模板的内容。

访问首页时效果如下图。



先来编写顶部导航内容的模板。

示例代码：（nav.html）

```
<h3>这里是顶部导航模板内容</h3>
<hr>
```

然后，编写底部信息内容的模板。

示例代码：（footer.html）

```
<hr>
<h3>这里是底部信息模板内容</h3>
```

最后，修改首页模板。

示例代码：（index.html）

```
<!DOCTYPE html>
<html lang="zh">
<head>
  <meta charset="UTF-8">
  <title>首页</title>
</head>
<body>
  {% include 'nav.html' %}
  <h4>这是我的第一个模板页面。 </h4>
  {% include 'footer.html' %}
</body>
</html>
```

大家能够看到，通过“{% include ‘模板名称’ %}”就能够把其它模板嵌套到当前模板内。

三、模板的复用

实际上通过嵌套，我们就实现了部分内容（顶部导航和底部信息）的复用。

不过，我们在开发 Web 项目的过程中，会有很多页面都包含重复的内容。

我们在每一类页面的模板中都进行重复内容的嵌套也是很麻烦。

所以，我们可以先做一个基本的模板，然后各类页面的模板复用这个基本的模板，只把页面中不同的部分进行重写。

1、创建基本模板

示例代码：（base.html）

```
<!DOCTYPE html>
<html lang="en">
<head>
  <meta charset="UTF-8">
  <title>{% block title %}默认{% endblock %} - 我的网站</title>
</head>
<body>
  {% include 'nav.html' %}
  {% block content %}这里是网站内容！{% endblock %}
  {% include 'footer.html' %}
</body>
</html>
```

在上方代码中，大家能够看到我们通过“{% block 名称 %}内容{% endblock %}”定义了一些内容，这些内容就能够在复用这个模板内容的模板中重写。

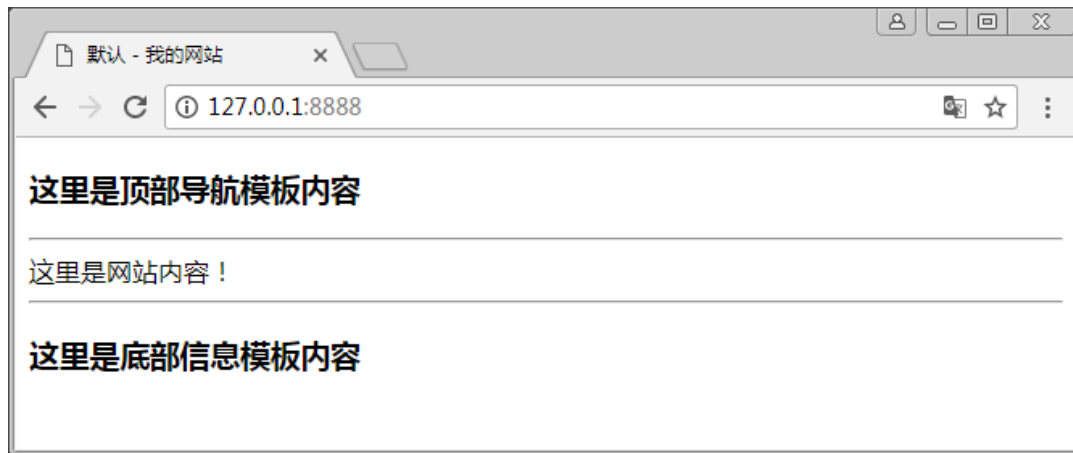
2、复用基本模板

接下来，我们在首页页面模板中复用上方的模板。

示例代码：（index.html）

```
{% extends 'base.html' %}
```

当完成上方代码中，大家通过浏览器访问当前项目，能够看到基本模板的内容。



但是，这不是我们想要的结果。

网站的标题和页面的内容都需要修改。

3、重写模板内容

我们在首页页面模板中继续添加内容，对复用的基本模板内容进行重写。

示例代码：（index.html）

```
{% extends 'base.html' %}
{% block title %}首页{% endblock %}
{% block content %}
    <h4>这是我的第一个模板页面。</h4>
{% endblock %}
```

通过上方代码，大家不难看出，“{% block 名称 %}内容{% endblock %}”能够对复用模板内容中相同名称的部分进行重写。

当完成上述内容，我们再次通过浏览器访问项目，就能够看到和之前一样的首页页面内容。

四、模板与数据整合

模板用于定义页面内容的展现，如果想动态的显示不同的内容，还需要与数据进行整合。

例如，访问不同的新闻列表页面，但是都是用相同的模板，模板中页面标题和新闻列表名称应该是可以改变的，而这些可变的数据应该由视图函数添加到模板定义的页面内容中。

1、创建模板

示例代码：（news_list.html）

```
{% extends 'base.html' %}
{% block title %}{{ news_type }}新闻{% endblock %}
```

```
{% block content %}  
    {{ news_type }}新闻列表 :  
{% endblock %}
```

上方代码中“{{ 变量名称 }}”能够获取视图函数中存入的变量数据。

2、定义视图函数

示例代码：

```
def news_list(request, news_type): # 定义新闻列表视图  
    news_dict = {'economic': '经济', 'sport': '体育'} # 创建参数字典  
    return render(request, 'news_list.html', {'news_type': news_dict[news_type]}) # 整合数据  
并返回页面内容
```

3、设置 URL 分发

示例代码：

```
path('news_list/<str:news_type>', siteviews.news_list),
```

通过以上步骤，我们就完成了想要的功能。

访问经济新闻的连接：http://127.0.0.1:8888/news_list/economic



访问体育新闻列表的连接：http://127.0.0.1:8888/news_list/sport



本节练习源代码：【[点此下载](#)】

转载请注明：魔力 Python » Django2：Web 项目开发入门笔记（5）

原文链接：<http://www.opython.com/899.html>

Django2：Web 项目开发入门笔记（6）

这一篇教程，我们继续了解 Django 中模板的使用。

主要内容如下：

- 内置标签和过滤器；
- 模板中使用循环；
- 模板中添加条件判断。

一、内置标签和过滤器

Django 中有很多的内置标签，例如之前已经使用的 `block` 和 `extends`。

关于内置标签，大家可以通过官方文档进行了解。【[点此查看](#)】

在官方文档的页面中，除了内置模板标签，还有过滤器。

过滤器实际上就是一些函数，帮助我们进行数据的处理。

我们可以通过管道符“|”使用过滤器，格式为“`{{ 变量|过滤器 1|过滤器 2 }}`”。

我们来做个练习，试用几个过滤器。

首先，我们在视图中先添加一个视图函数。

在视图函数中，我们添加两项数据和页面内容进行整合。

示例代码：

```
def fiter_test(request):  
    return render(request, 'filter.html', {'letters': 'abc', 'number': 1})
```

然后，我们添加 URL 分发配置。

示例代码：

```
path('filter/', siteviews.fiter_test),
```

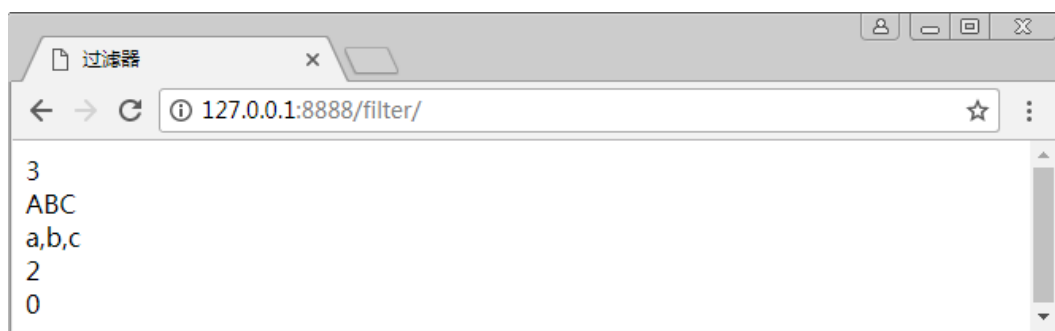
最后，我们创建一个模板文件'filter.html'，在模板文件中我们使用过滤器对数据进行处理。

示例代码：

```
<!DOCTYPE html>  
<html lang="en">  
<head>  
    <meta charset="UTF-8">  
    <title>过滤器</title>  
</head>  
<body>  
    {{ letters|length }}<br>  
    {{ letters|upper }}<br>  
    {{ letters|join:' ' }}<br>  
    {{ number|add:1 }}<br>  
    {{ number|add:-1 }}<br>  
</body>  
</html>
```

在模板代码中，我们通过过滤器对字符串'letters'分别进行了长度获取、转为大写以及逗号分隔的操作；对数字"number"分别进行了加 1 和减 1 的操作。

页面显示结果：



二、模板中使用循环

在一个模板页面中添加单个数据可以通过“`{{ 变量名 }}`”的方式来实现。

如果需要添加多个数据，则可以通过字典来实现。

例如，视图中向模板页面传入的数据为字典（例如：`{'person':{'name':'小楼','sex':'男'}}`），在模板中则可以通过“`{{ person.name }}`”和“`{{ person.sex }}`”获取到相应的数据。

但是，如果页面中我们需要呈现一个列表（例如新闻列表），这时会有大量的数据需要处理，该如何实现呢？

另外，如果列表中有个别的项需要处理（例如第一项为红色字体），如何处理呢？

这些需求的实现，需要在模板中使用循环和条件判断。

如果有新闻列表内容的话，我们需要能够在页面中呈现出来。



这就需要在视图函数中获取一个新闻列表的数据内容，并在模板中循环加载出每一个列表项。

因为，我们还没有接触过模型（Model），暂时还不能和数据库进行关联，在这里我们先手动写出一个新闻列表的内容数据。

1、定义视图函数

在之前编写好的视图函数中，我们写出一个新闻列表内容，然后整合到模板定义的页面内容中。

此处只以经济新闻为例。

示例代码：

```
def news_list(request, news_type):
    news_dict = {'economic': '经济', 'sport': '体育'}
    news_titles = []
    if news_type == 'economic':
        news_titles = [('12/5', '作者成为全国首富。'), ('12/4', '作者成为全省首富。'),
                        ('12/3', '作者成为全市首富。'), ('12/2', '作者成为镇里首富。'), ('12/1', '作者成
为村里首富。')]
    return render(request, 'news_list.html', {'news_type': news_dict[news_type], 'news_titles':
news_titles})
```

2、模板中添加循环

示例代码：

```
{% extends 'base.html' %}
{% block title %}{{ news_type }}新闻{% endblock %}
{% block content %}
    {{ news_type }}新闻列表：
    <ul>
        {% for date,title in news_titles %}
            <li>{{ title }} ({{ date }}) </li>
        {% endfor %}
    </ul>
{% endblock %}
```

注意，模板中的循环是由“{% for 变量 in 可迭代对象 %}”和循环的内容以及“{% endfor %}”组成。

当我们完成上方代码之后，访问经济新闻列表页面，就能看到想要的效果了。

三、模板中添加条件判断

通过添加循环的代码，我们也能够看出，在模板中添加逻辑代码其实非常简单，只要了解添加的规则和能够使用的功能就可以了。

接下来，我们就尝试将新闻列表第一项的用红色字体显示。

示例代码：

```
{% extends 'base.html' %}
{% block title %}{{ news_type }}新闻{% endblock %}
{% block content %}
    {{ news_type }}新闻列表：
```

```
<ul>
    {% for date,title in news_titles %}

        {% if forloop.first %}

            <li><font color="red">{{ title }}</font> ({{ date }}) </li>

        {% endif %}

        <li>{{ title }} ({{ date }}) </li>
    {% endfor %}
</ul>
{% endblock %}
```

上方代码中，红色部分是新增的代码。

通过条件判断，如果是循环第一项“forloop.first”，将“{{ title }}”的内容用“font”标签变为红色字体。

“forloop.first”是循环的一个变量，像这样的变量还有其他几个。

在 PyCharm 中，我们输入“forloop.”之后，就能够看到这些变量。



当我们写完“if”的语句块之后，不要忘了在最后添加“{% endif %}”。

不管是“for”还是“if”，在模板中他们都是标签，需要有开始标签和结束标签。

以上是我们对列表内容的一些处理。

不过，有时列表中没有内容的话，我们也需要进行处理。

在之前的示例中，不管是体育新闻还是经济新闻，我们都用的同一模板。

体育新闻，我们并没有任何列表数据，这时需要给用户一个提示。

那么，如何判断数据内容是空的，并给出相应提示呢？

示例代码：

```
{% extends 'base.html' %}
{% block title %}{{ news_type }}新闻{% endblock %}
{% block content %}
    {{ news_type }}新闻列表：
    <ul>
        {% for date,title in news_titles %}
            {% if forloop.first %}
                <li><font color="red">{{ title }}</font> ({{ date }}) </li>
            {% endif %}
            <li>{{ title }} ({{ date }}) </li>

        {% endfor %}
    </ul>
{% endblock %}
```

上方代码中，红色代码是新增部分。

通过在 for 循环中添加“{% empty %}”，就能够对数据内容为空进行判断，从而给出相应提示。

当我们完成本篇教程的所有代码，再次启动开发服务器，访问经济新闻列表和体育新闻列表，就能够看到我们想要的结果了。

经济新闻页面：



体育新闻页面：



本节练习源代码：[【点此下载】](#)

转载请注明：魔力 Python » Django2：Web 项目开发入门笔记（6）

原文链接：<http://www.opython.com/928.html>

Django2：Web 项目开发入门笔记（7）

这一篇教程，我们一起来学习 Django2 中模型的使用。



提示：为了更顺利的学习新的内容，建议大家在 PyCharm 中重新创建项目和应用。

在之前的教程中，我们已经知道模型（Model）是和数据库相关的。

Django 支持多种数据库，包括默认的 SQLite3 以及 MySQL 和 PostgreSQL 等数据库。

使用哪一种数据库需要在项目的 settings.py 中进行进行配置。

如果使用默认的 SQLite3 数据库，这个配置是 Django 创建项目时已经添加好的。

示例代码：

```
DATABASES = { # 数据库配置
    'default': { # 默认数据库
        'ENGINE': 'django.db.backends.sqlite3', # 数据库引擎设置
        'NAME': os.path.join(BASE_DIR, 'db.sqlite3'), # 数据库的路径与名称
    }
}
```

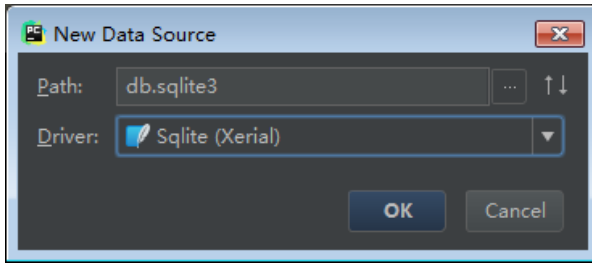
注意检查一下项目文件夹中是否包含“db.sqlite3”这个数据库文件。

如果没有的话，可以通过 PyCharm 进行添加。

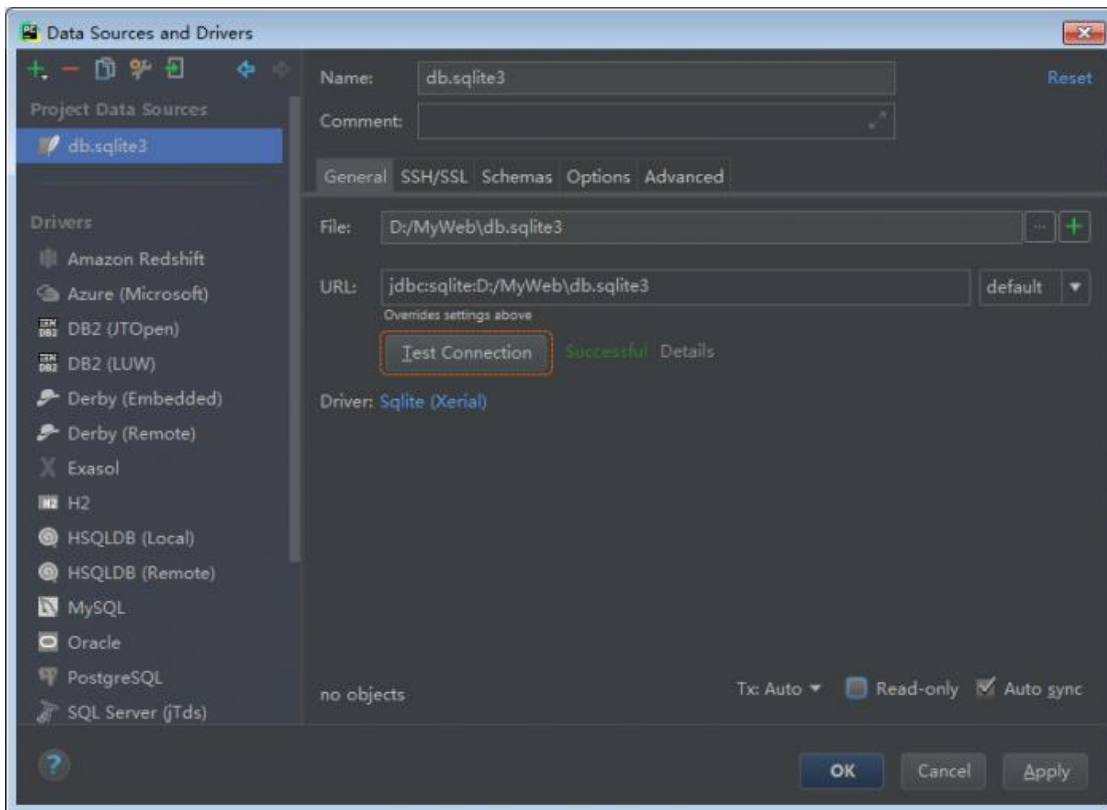
在项目文件夹上点击鼠标右键，菜单中选择新建（New）-数据源（Data Source），此时会弹出一个窗口。

在这个窗口中，我们可以选择数据库文件的存放路径（Path），直接填写名称即是存放在当前项目文件夹中。

数据库驱动（Driver）选择 sqlite（Xerial），然后点击确定（OK）按钮。



在新打开的窗口中，我们可以点击测试链接（Test Connection）测试一下数据库链接是否正常。



点击确定（OK）按钮后，数据库文件就创建好了。

接下来，我们编写模型文件（models.py）的代码。

这里值得一提的是，在更换数据库时，“models.py”中的代码不用做任何更改，只需要在“settings.py”文件中更改数据库的配置。

我们来看一下创建数据库表的流程，就知道为什么这么说了。

创建数据表的流程一共 3 步：

- 模型文件（models.py）中编写数据表所对应的类；
- 通过“manage.py”的“makemigrations”命令创建迁移项；

- 通过“manage.py”的“migrate”命令进行迁移。

也就是说，我们在模型文件（models.py）中创建了数据表对应的类，剩下的工作都通过“manage.py”的命令，让 Django 自动去执行。

一、编写模型文件（models.py）

假设我们创建一个商品信息表。

模型文件（models.py）中，需要创建类以及类的特性（变量），实际上类名对应的是数据库的表名，而特性（变量）对应的是数据表的列（字段）。

示例代码：

```
class Goods(models.Model):
    goods_name = models.CharField(max_length=30)
    goods_number = models.IntegerField()
    goods_price = models.FloatField()
```

在上方代码中，我们创建了一些不同类型的字段的对象，包括字符串类型（CharField）、整数类型（IntegerField）和浮点数类型（FloatField）。

字段类型有很多种，例如还有邮箱类型、URL 类型等等，大家可以参考官方文档进行使用。

<http://www.opython.com/files/docs/django2/ref/models/fields.html>

并且，大家能够看到，我们在创建字段对象时还可以输入参数，这些参数在上方文档中也有详尽的描述。

比较常见的参数有几种：

- max_length：该字段最大字符数量限制。
- null：如果为 True，则该字段默认为 null 值。
- blank：如果为 True，则该字段可以为空值。
- default：该字段的默认值。
- primary_key：该字段设置为主键。
- unique：该字段的值必须是当前数据列中唯一的。
- choices：该字段的值为二元数组或列表，例如：((1,2),(a,b))。

当我们编写完模型文件（models.py），接下来就可以进行数据迁移，将模型文件（models.py）中的数据迁移到数据库。

二、创建迁移项与执行迁移操作

在创建迁移项之前，我们需要先在全局配置文件（settings.py）中，将当前应用添加到“INSTALLED_APPS”列表。

```
INSTALLED_APPS = [  
    'django.contrib.admin',  
    'django.contrib.auth',  
    'django.contrib.contenttypes',  
    'django.contrib.sessions',  
    'django.contrib.messages',  
    'django.contrib.staticfiles',  
    'MySite.apps.MysiteConfig', # 添加此项  
    # 'MySite' # 也可以直接添加应用名称  
]
```

然后，在命令行终端中，通过命令创建迁移项。

```
python manage.py makemigrations
```

或者，我们在 PyCharm 的工具（Tools）中，运行“manage.py”的任务（Run manage.py Task），然后输入命令创建迁移项。

创建迁移项完毕后，会在项目文件夹下的“migrations”文件夹中出现迁移项文件，类似“0001_initial.py”。

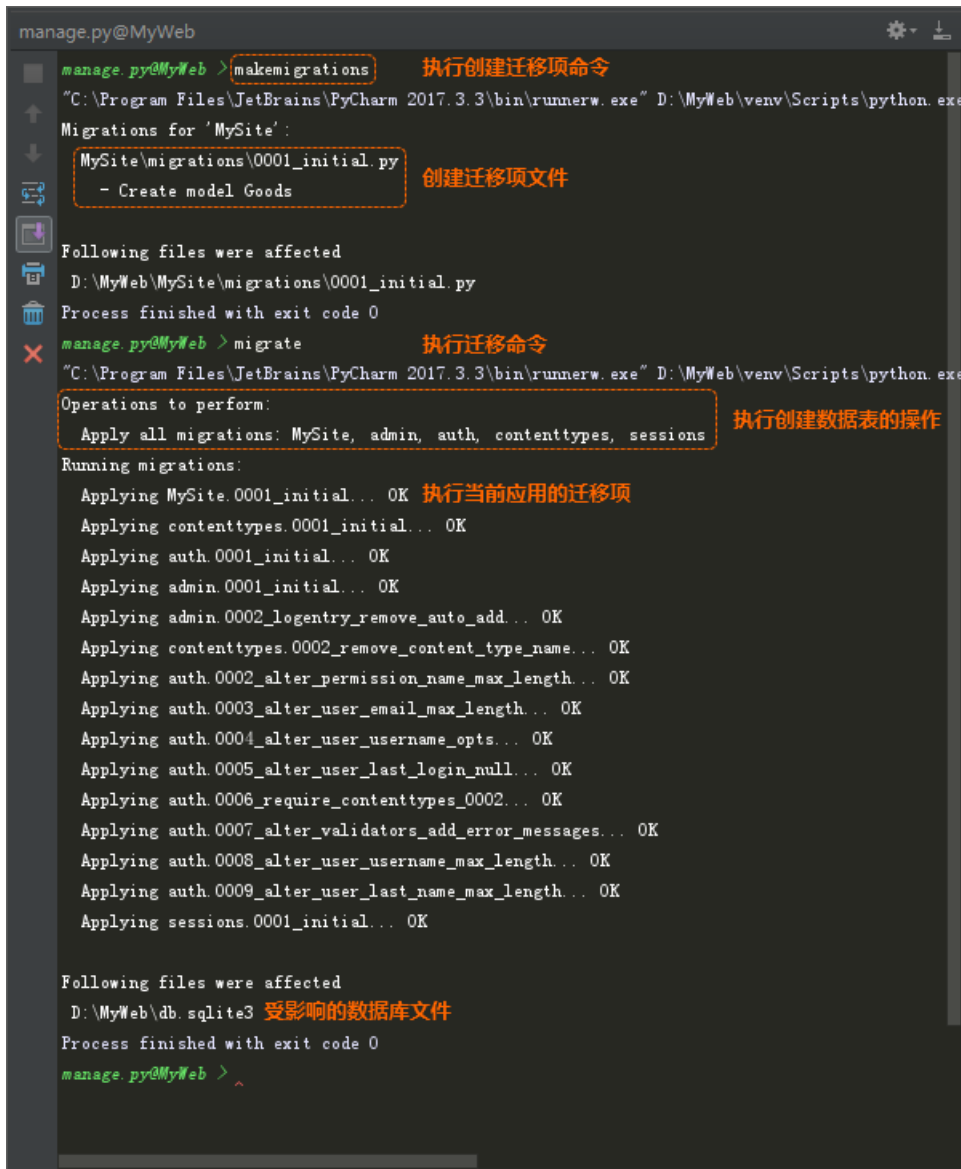
接下来，我们再通过“migrate”命令进行数据迁移操作。

```
python manage.py migrate
```

或者，在“manage.py”的任务终端中，执行“migrate”命令。

这个命令会将模型文件（models.py）中的数据同步到数据库，完成数据表的创建。

执行的命令与结果：



```
manage.py@MyWeb > makemigrations
"C:\Program Files\JetBrains\PyCharm 2017.3.3\bin\runnerw.exe" D:\MyWeb\venv\Scripts\python.exe
Migrations for 'MySite':
  MySite\migrations\0001_initial.py
    - Create model Goods
Following files were affected
  D:\MyWeb\MySite\migrations\0001_initial.py
Process finished with exit code 0
manage.py@MyWeb > migrate
"C:\Program Files\JetBrains\PyCharm 2017.3.3\bin\runnerw.exe" D:\MyWeb\venv\Scripts\python.exe
Operations to perform:
  Apply all migrations: MySite, admin, auth, contenttypes, sessions
Running migrations:
  Applying MySite.0001_initial... OK
  Applying contenttypes.0001_initial... OK
  Applying auth.0001_initial... OK
  Applying admin.0001_initial... OK
  Applying admin.0002_logentry_remove_auto_add... OK
  Applying contenttypes.0002_remove_content_type_name... OK
  Applying auth.0002_alter_permission_name_max_length... OK
  Applying auth.0003_alter_user_email_max_length... OK
  Applying auth.0004_alter_user_username_opts... OK
  Applying auth.0005_alter_user_last_login_null... OK
  Applying auth.0006_require_contenttypes_0002... OK
  Applying auth.0007_alter_validators_add_error_messages... OK
  Applying auth.0008_alter_user_username_max_length... OK
  Applying auth.0009_alter_user_last_name_max_length... OK
  Applying sessions.0001_initial... OK
Following files were affected
  D:\MyWeb\db.sqlite3
Process finished with exit code 0
manage.py@MyWeb >
```

当我们完成以上步骤，在数据库中就会出现一个名为“MySite_goods”（应用名称_小写类名）的数据表。

不过，在创建数据表的时候，我们可能会遇到这两种情况：

- “models.py”模块中的类遗漏了某个字段或者某个字段觉得没有用了；
- “models.py”模块中，类的某个字段写错了类型或者参数。

当发生这样的情况，创建的数据表就不符合要求，而不能正常使用，我们必须修改表的结构，或者更改字段的设置。

那么，如何解决呢？

在 Django1.7 版本之前，我们需要借助第三方库“South”进行表结构或者字段的修改操作。

而在 Django1.7 版本开始, “South”已经集成在“Django.core”中。

我们只需要修改“models.py”模块中类的内容, 然后再次运行“makemigrations”和“migrate”命令就能够完成修改。

不过要注意的是, 如果更改表结构时, 是为数据表添加新的字段, 在执行“makemigrations”命令时, 需要选择一个选项, 并输入该字段的默认值。



```
manage.py@MyWeb > makemigrations 创建迁移项
"C:\Program Files\JetBrains\PyCharm 2017.3.3\bin\runnerw.exe" D:\MyWeb\venv\Scripts\python.exe
You are trying to add a non-nullable field 'goods_sales' to goods without a default; we can't
Please select a fix:
1) Provide a one-off default now (will be set on all existing rows with a null value for this
2) Quit, and let me add a default in models.py
Select an option: 1 选择选项 (1: 提供一个默认值, 2: 退出)
Please enter the default value now, as valid Python
The datetime and django.utils.timezone modules are available, so you can do e.g. timezone.now
Type 'exit' to exit this prompt
>>> 0 输入默认值
Migrations for 'MySite':
  MySite\migrations\0005_goods_goods_sales.py
    - Add field goods_sales to goods

Following files were affected
  D:\MyWeb\MySite\migrations\0005_goods_goods_sales.py
Process finished with exit code 0
manage.py@MyWeb > migrate 执行迁移操作
"C:\Program Files\JetBrains\PyCharm 2017.3.3\bin\runnerw.exe" D:\MyWeb\venv\Scripts\python.exe
Operations to perform:
  Apply all migrations: MySite, admin, auth, contenttypes, sessions
Running migrations:
  Applying MySite.0005_goods_goods_sales... OK 迁移项成功执行

Following files were affected
  D:\MyWeb\db.sqlite3
Process finished with exit code 0
manage.py@MyWeb >
```

如果不想在运行命令时这么麻烦, 不妨在“models.py”模块中添加新的字段时, 直接写入默认值。

例如商品销量字段:

```
goods_sales = models.IntegerField(default=0)
```

当我们完成数据表的创建, 我们可以尝试进行数据的添加与读取。

大家还记得之前我们学习的 SQL 语句吗？

相信有很多人都已经记不清了吧！

别担心，我就是想吓你们一跳，喔嚯嚯嚯嚯嚯嚯嚯！

使用 Django 可以不用 SQL 语句，就能够实现数据库的增删改查。

不管执行什么数据操作，我们都需要先导入模型文件（models.py）中的类。

在命令行终端我们导入 Goods 类。

```
>>>from MySite.models import Goods
```

然后，通过类进行数据操作。

一、添加数据

添加数据有两种方式，我们分别演示。

第一种：通过 create()方法添加。

```
>>>Goods.objects.create(goods_name='铅笔',goods_number='10',goods_price='1.5')
<Goods: Goods object (1)>
```

第二种：通过 save()方法添加。

```
>>>g=Goods(goods_name='橡皮',goods_number='20',goods_price='0.5')
g.save()
```

或者使用另外一种写入参数的方式。

```
>>>g=Goods()
>>>g.goods_name='直尺'
>>>g.goods_number=15
>>>g.goods_price=2.4
>>>g.save()
```

二、查询数据

查询数据我们可以使用下列语句。

```
>>>Goods.objects.all()
<QuerySet [<Goods: Goods object (1)>, <Goods: Goods object (2)>, <Goods: Goods object (3)>]>
```

通过 all()方法，能够获取数据库中所有的数据对象。

那么，能不能看到数据的值呢？

通过 `values()` 方法，就能够看到数据库中所有数据的值。

```
>>>Goods.objects.values()
<QuerySet [{ 'id': 1, 'goods_name': '铅笔', 'goods_number': 10, 'goods_price': 1.5}, { 'id': 2,
'goods_name': '橡皮', 'goods_number': 20, 'goods_price': 0.5}, { 'id': 3, 'goods_name': '直尺',
'goods_number': 15, 'goods_price': 2.4}]>
```

关于数据库的操作，我们到此先告一段落，在下一篇教程中，我们结合 URL 分发、视图和模板，在浏览器的页面中进行一些对数据的操作。

转载请注明：魔力 Python » Django2 : Web 项目开发入门笔记（7）

原文链接：<http://www.opython.com/917.html>

Django2 : Web 项目开发入门笔记（8）

这一篇教程，我们一起来使用 Django 通过 HTML 页面进行数据库的访问。

首先，我们先来看一下页面中的功能。



这些功能包括：

- 查询数据库中所有商品数据；
- 查询数据库中指定名称的商品数据；
- 查询数据库中指定价格区间的商品数据；
- 查询数据库中的指定数据并排序。

在开始编写这些功能之前，我们先来准备数据库中的数据。

在上一篇教程的基础上，我们在已创建好的数据表中添加更多的数据。

铅笔,10,1.5
橡皮,20,0.5
直尺,15,2.4
作业本,18,2.5
笔记本,21,6.2
钢笔,35,9.9
铅笔盒,33,12.5
墨水,12,5.0
曲别针,9,1.0
订书器,18,8.0
订书钉,15,3.6
裁纸刀,12,7.5
签字笔,20,6.8
签字笔芯,50,3.5
自动铅笔,32,4.2
自动笔芯,36,2.1

这些数据如果通过命令行终端手动添加的话，实在是太累了。

我们可以打开 PyCharm，在应用文件夹中新建（New）一个文件（File），命名为“data”，把这些数据保存在文件中。

然后再新建（New）一个 Python 文件（Python File）编写一个脚本，进行数据导入。

示例代码：

```
from MySite.models import Goods
```

```
with open('data', 'r', encoding='utf-8') as file: # 打开文件创建文件对象（注意编码）
```

```
    for line in file: # 读取每一行
```

```
        lst = line.strip().split(',') # 将每一行中的商品信息转换为列表
```

```
        state = Goods.objects.create(goods_name=lst[0], goods_number=lst[1],
```

```
goods_price=lst[2]) # 添加数据到数据库
```

```
        print(state) # 显示输出添加数据的结果
```

然后，我们运行脚本代码。

这时候，大家会看到引发了一个异常：django.core.exceptions.AppRegistryNotReady: Apps aren't loaded yet.（应用程序没有载入）

这是因为，我们使用的是 Django 框架，进行数据库的操作，需要框架的支持。

而我们单独运行写好的脚本时，Django 框架没有被启动，这就会导致异常发生。

所以，在刚才脚本代码的开始，我们加入以下代码：

示例代码：

```
import os
import django

os.environ.setdefault("DJANGO_SETTINGS_MODULE", "MyWeb.settings") # 关联默认设置
django.setup() # 装载 Django
```

注意，程序代码的执行是从上至下的，所以这段代码一定要放在前一段代码的上方。

这样，在 Django 启动后，才会导入 Goods，不会引发异常。

当我们执行完上方代码，会看到多个类似“Goods object (编号)”的结果，这些结果表名数据添加是有效的，结果中的编号是因为我们创建数据表的时候没有指定主键，数据库自动给添加的 id 列作为主键。

这里给大家推荐一款小工具：sqlite studio。【[点此下载](#)】

这款工具，能够打开 SQLite 的数据文件，进行可视化操作。（注意不要把这个工具放在中文目录下，否则会报错，无法打开！）

添加好了数据之后，接下来，我们分步完成各个功能。

一、创建模板

1、首页模板

我们在模板文件夹中新建模板文件“index.html”，并在这个文件中添加 4 个表单。

示例代码：（index.html）

```
<!DOCTYPE html>
<html lang="en">
<head>
  <meta charset="UTF-8">
  <title>首页</title>
</head>
<body>
<h3>商品查询</h3>
<form action="/all/" method="GET">
  <input type="submit" value="查询全部">
</form>
```

```
<br>
<form action="/search_name/" method="GET">
  <input type="text" name="goods_name">
  <input type="submit" value="名称查询">
</form>
<br>
<form action="/search_price/" method="GET">
  <input type="text" name="min_price">
  <input type="text" name="max_price">
  <input type="submit" value="价格区间查询">
</form>
<br>
<form action="/search_sort/" method="GET">
  <input type="radio" name="sort" value="all_asc" checked>查询全部并升序排列
  <input type="radio" name="sort" value="all_desc">查询全部并降序排列
  <input type="radio" name="sort" value="result_asc">条件查询并升序排列
  <input type="submit" value="排序查询">
</form>
</body>
</html>
```

如上方代码所示，每个表单“form”都有不同“action”属性，这些属性就是点击提交“submit”按钮时，打开的 URL。

并且，在这些表单中添加了不同的“input”标签，形成了不同的表单内容。

“input”标签中的“name”属性就是 URL 中参数的名称，而“value”属性或文本框中输入的内容就是参数的值。

以第 2 个表单为例，当点击提交按钮时，就会打开 URL“http://IP:端口号（或域名）/search_name/? goods_name=文本框中的输入内容”

另外，还要注意的是第 4 个表单，里面“input”标签的“type”属性是“radio”，也就是单选按钮。

如果想让单选按钮默认选中，需要在标签属性中添加“checked”属性。

如果想让多个单选按钮只有一个能够被选中，需要将这些单选按钮的“name”属性添加相同的名称（编组操作）。

提交表单的时候，URL 中“name”属性值所对应的参数值就是被选中单选按钮的“value”值。

2、搜索结果页模板

在模板文件夹中新建搜索结果页面的模板“search_result.html”。

在这个模板所定义的页面内容中，我们要展示搜索结果的列表。

我们进行查询时，能够获得查询结果的数据对象（例如：goods_list），所以，在模板中我们对数据对象进行遍历，输出列表内容。

示例代码：（search_result.html）

```
<!DOCTYPE html>
<html lang="en">
<head>
  <meta charset="UTF-8">
  <title>搜索结果</title>
</head>
<body>
<h3>搜索结果：</h3>
<ol>
{% for goods_info in goods_list %}
  <li><label style="display:inline-block;width:120px;">名称：
{{ goods_info.goods_name }}</label>
  <label style="display:inline-block;width:90px;">数量：
{{ goods_info.goods_number }}</label>
  <label style="display:inline-block;width:90px;">价格：
{{ goods_info.goods_price }}</label>
{% empty %}
  没有查询结果！
{% endfor %}
</ol>
</body>
</html>
```

这里用到了“ol”标签，这个标签中的“li”子标签能够自动产生递增的数字编号。

然后，每一行的数据字段都使用“table”标签进行呈现，通过设置“table”标签的样式，让内容有更好的排列效果。

二、设置 URL 分发

我们创建模板时，已经定义好了提交的 URL，接下来就要在“urls.py”文件中进行分发配置。

先导入视图模块，然后在 urlpatterns 列表中，我们添加一些新的语句。

示例代码：

```
from MySite import views as siteviews

path('',siteviews.index),
path('all/', siteviews.searchall),
path('search_name/', siteviews.searchname),
path('search_price/', siteviews.searchprice),
path('search_sort/', siteviews.searchsort),
```

三、编写视图函数

根据 URL 分发设置中调用的函数名称，我们定义视图函数。

1、首页视图函数

示例代码：

```
def index(request):
    return render(request, 'index.html')
```

2、查询全部数据的视图函数

示例代码：

```
def searchall(request):
    goods_list = Goods.objects.all()
    return render(request, 'search_result.html', {'goods_list': goods_list})
```

通过模型对象调用 `all()` 方法，就能够获取到数据库中所有的数据。

我们将获取到的数据对象传入模板的页面内容中进行读取。

查询结果页面显示内容：



3、查询指定商品名称数据的视图函数

示例代码：

```
def searchname(request):
    goods_name = request.GET['goods_name']
    goods_list = Goods.objects.filter(goods_name=goods_name) # 完全匹配搜索关键字
    # goods_list = Goods.objects.get(goods_name=goods_name) # 查询满足条件的一个结果
    # (查询到多个结果时异常)
    # goods_list = Goods.objects.filter(goods_name__contains=goods_name) # 模糊匹配搜索
    # 关键字
    return render(request, 'search_result.html', {'goods_list': goods_list})
```

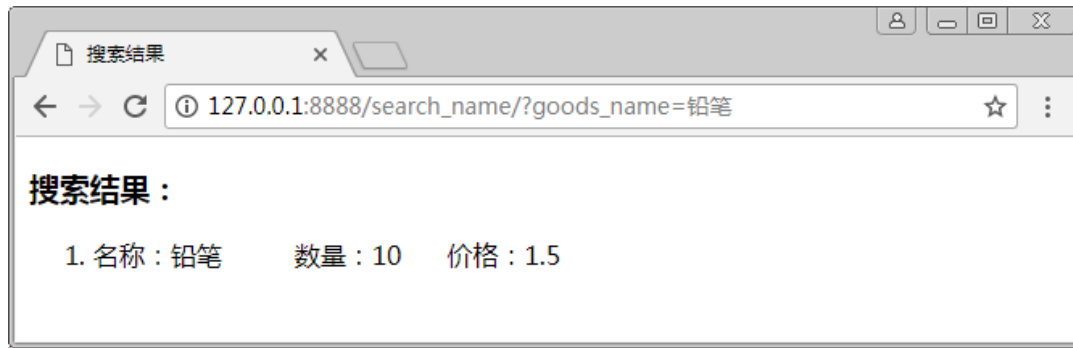
每个商品名称都是唯一的，在获取数据对象时可以使用 `get()` 方法（第 1 条被注释的语句），但是这个方法获取的数据对象不可迭代，会导致模板无法加载数据产生异常。

所以，在这里我们使用 `filter()` 方法来获取可迭代的数据对象。

如果 `filter` 方法的参数类似“(goods_name=goods_name)”，在查询时会进行精确匹配。

例如，查询商品名称为“铅笔”的数据。

查询结果页面显示内容：

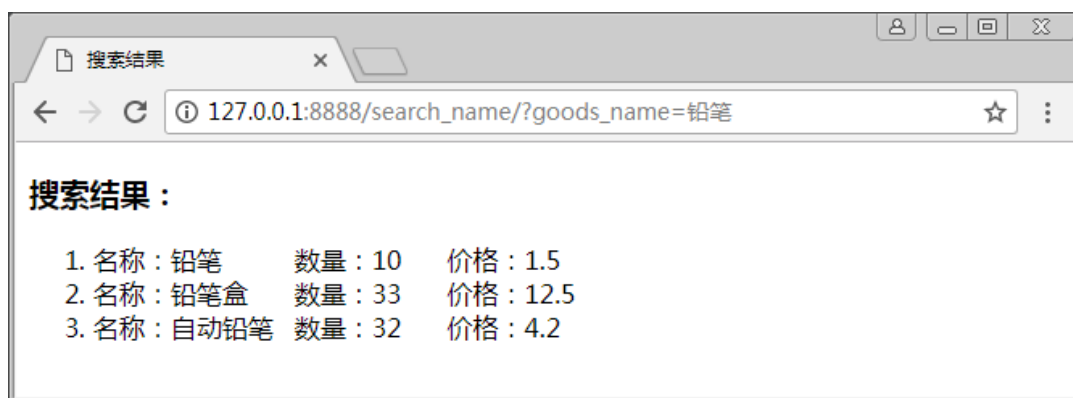


需要模糊查询的话，我们需要给参数添加查询的匹配规则，添加匹配规则就是在参数名称后方输入 2 个下划线和匹配规则的名称。

例如“contains”就是包含匹配，只要字段值中包含参数值即可查询出来，即模糊查询（第 2 条被注释的语句）。

例如，同样输入查询的商品名称为“铅笔”。

查询结果页面显示内容：



匹配规则有很多种，作用如下：（点击每个规则可查询详细文档）

- exact：精确匹配
- iexact：精确匹配（忽略大小写）
- contains：包含匹配
- icontains：包含匹配（忽略大小写）
- in：列表包含匹配
- gt：大于匹配
- gte：大于等于匹配
- lt：小于匹配
- lte：小于等于匹配
- startswith：开头匹配
- istartswith：开头匹配（忽略大小写）

- `endswith` : 结尾匹配
- `iendswith` : 结尾匹配 (忽略大小写)
- `range` : 范围匹配
- `date` : 日期匹配 (年, 月, 日)
- `year` : 年份匹配
- `month` : 月份匹配 (1-12)
- `day` : 天数的匹配 (1-31)
- `week` : 周数匹配 (1-52 或 53)
- `week_day` : 周几的匹配 (1-7)
- `quarter` : 季度的匹配 (1-4)
-
- `hour` : 小时匹配 (0-23)
- `minute` : 分钟匹配 (0-59)
- `second` : 秒数匹配 (0-59)
- `isnull` : 空值匹配 (True 或 False)
- `regex` : 正则表达式匹配
- `iregex` : 正则表达式匹配 (忽略大小写)

了解了以上内容, 相信大家能够找出更多的查询方式。

3、查询某一价格区间数据视图的视图函数

示例代码：

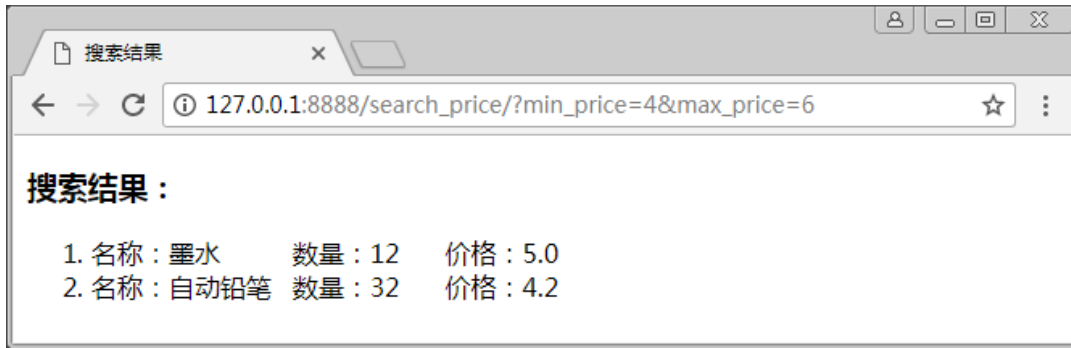
```
from django.db.models import Q

def searchprice(request):
    min_price = request.GET['min_price']
    max_price = request.GET['max_price']
    goods_list = Goods.objects.filter(goods_price__gt=min_price, goods_price__lt=max_price)
    # 满足全部多个条件
    # goods_list = Goods.objects.filter(Q(goods_price=0.5) | Q(goods_price=2.4)) # 满足任何一个条件
    return render(request, 'search_result.html', {'goods_list': goods_list})
```

在上方代码中大家能够看到, 当进行满足多个条件的查询时, 我们只需要在 `filter()` 方法中写入多个参数内容就可以了 (也可以使用下方介绍的“Q”类)。

例如, 查询价格大于 4 元并且小于 6 元的商品。

查询结果页面显示内容：



而如果满足任何一个条件时进行数据查询，我们需要使用 Django 数据模型模块中提供的“Q”类，多个条件需要创建多个“Q”的对象并用竖线“|”（或者）分隔。（被注释的语句）

关于“Q”类，它的作用是将过滤器参数封装为可以在逻辑上进行组合的对象，可以使用符号“&”（表示 AND）或“|”（表示 OR）进行组合。

4、对不同查询结果进行排序的视图函数

对于排序，如果查询全部数据并升序排列，我们只需要调用 `order_by()` 方法，并输入排序的字段名称为参数。

而查询全部数据并进行降序排列，同样调用 `order_by()` 方法，只是参数中的字段名称前面加上减号“-”即可。

而且，我们也可以通过 `filter()` 方法的查询结果，调用 `order_by()` 方法，并依据某一字段进行排序。

示例代码：

```
def searchsort(request):
    sort = {'all_asc': Goods.objects.order_by('goods_price'), # 查询全部结果后升序排列
           'all_desc': Goods.objects.order_by('-goods_price'), # 查询全部结果后降序排列
           'result_asc': Goods.objects.filter(goods_price__lt='5').order_by('goods_price') # 对某一查询结果排序
           }
    return render(request, 'search_result.html', {'goods_list': sort[request.GET['sort']]})
```

提示：上方代码并没有通过条件判断分别组织查询数据的语句进行数据查询，而是使用字典来完成。

以上就是通过查询数据操作对 Django 的数据库操作举了几个简单的例子。

实际上，Django 中对数据库的操作有很多，更多的数据库操作相关内容，还是建议大家多了解一下官方文档或者通过搜索引擎查阅相关的中文资料。

本节练习源代码：【[点此下载](#)】

转载请注明：魔力 Python » Django2 : Web 项目开发入门笔记 (8)

原文链接：<http://www.opython.com/956.html>

Django2 : Web 项目开发入门笔记 (9)

这一篇教程，我们来了解一些关于 JavaScript 的使用。

JavaScript 是网络脚本语言，它能够用来动态的改进设计（页面元素、内容以及交互效果等）、验证表单、检测浏览器、创建 cookie 等等。

所以，JavaScript 非常强大，也非常受欢迎，基本上我们日常访问的网站都有嵌入 JavaScript 代码。

但是，JavaScript 这门语言学起来并不复杂，它非常简单。

关于 JavaScript 这门语言的学习，大家可以参考：<http://www.w3school.com.cn/js/>

不过，在这里，我们要使用的是 JavaScript 的一个强大而知名的库：JQuery。

并不是了解了 JavaScript 就能够掌握 JQuery，就好像不是学完了 Python 就会使用 Django 一样。

JQuery 的使用同样需要进行相关的学习。

关于 JQuery 这个库的使用，大家可以参考：<http://www.w3school.com.cn/jquery/>

在这里，我们只做一些简单的了解，满足我们的实现目标。

一、调用 JQuery 库

JQuery 库可以到官网下载：<http://jquery.com/download/>

然后，将下载的文件放入项目中进行使用。

在这里，我们不采用下载这种方式，而是在线调用。

在模板文件的 HTML 代码中，只需要添加下方语句，即可实现在线调用百度静态资源公共库所提供的 JQuery 库文件。

```
<script type="text/javascript"
src="http://apps.bdimg.com/libs/jquery/2.1.4/jquery.min.js"></script>
```

二、相关语法

使用 JQuery 库，我们需要了解一些 JavaScript 的语法和 JQuery 的特定语法。

1、在 HTML 文件中，JavaScript 脚本的代码内容也要写在一对标签中。

```
<script>
.....
脚本代码
.....
</script>
```

2、需要使用 var 定义变量。

```
var name = '小楼';
```

3、语句以分号“;”结束。

```
$('#msg_user').html('请输入用户名！');
```

4、通过属性选择器“\$(‘#id’)”获取页面中的元素对象。（见上句代码）

5、语句块写在一大括号“{}”中。

```
function(){
...语句块...
}
```

6、条件写在小括号“[]”中。

```
if (name === ""){}
for (i in list) {}
```

7、条件判断的运算符使用“===”表示相等的关系；使用“!==”表示不相等的关系。

需要了解的语法就这么多，在下面的内容中我们会用到这些语法。

另外，我们还要了解一块内容：Ajax。

Ajax（Asynchronous Javascript And XML：异步 JavaScript 和 XML）是一种创建交互式网页应用的网页开发技术。

它能够无需重新加载整个网页而更新部分网页内容。

关于 Ajax 的相关资料，大家可以参考：<http://www.w3school.com.cn/ajax/>

那么，Ajax 如何使用呢？

在 JQuery 库中拥有完整的 Ajax 兼容套件。

我们只需要使用其中的函数和方法，就可以在不刷新浏览器的情况下从服务器加载数据。

下面，我们来看本篇教程要实现的案例。

首先，我们先来实现一个用户注册的功能。

这个注册功能需要的标签比较简单，只有一个用户名输入框、一个密码输入框以及一个提交注册的按钮。

要实现的具体功能包括：

- 当光标离开用户名输入框时，如果没有输入任何内容，则在用户名后方提示“请输入用户名！”
- 当光标离开用户名输入框时，数据库表中已存在相同的用户名，则在用户名后方提示“用户名已存在！”
- 当点击提交注册按钮时，如果用户名和密码都已输入，则将用户名和密码存入数据库中，并返回成功的状态编号；
- 根据返回的成功编号，在提交注册按钮后方提示“恭喜您，注册成功！”。

The image shows two side-by-side browser window screenshots of a web application titled "用户注册" (User Registration). Both windows have the address bar set to "127.0.0.1:8888/reg/".

The left window shows the registration form with the following state:

- Username input field: "aaa"
- Password input field: (empty)
- Registration button: "注册"
- Feedback message: "用户名已存在！" (Username already exists!)

The right window shows the registration form with the following state:

- Username input field: "aaaaaa"
- Password input field: "*****"
- Registration button: "注册"
- Feedback message: "恭喜您，注册成功！" (Congratulations, registration successful!)

以上这些功能，都是在页面无刷新的情况下完成的。

提示：一个完成的登录面板的验证与提示并不仅仅这么多，大家可以根据需求完成更多的功能。这里仅仅为讲解知识点，省略类似的其他功能。

一、创建模板文件（register.html）

示例代码：

```
<!DOCTYPE html>
<html lang="en">
<head>
  <meta charset="UTF-8">
  <title>注册</title>
</head>
<body>
<h3>用户注册</h3>
```

```
<input type="text" name="user_name" id="user_name">
<label id="msg_user"></label><br>
<input type="password" name="password" id="password">
<label id="msg_pass"></label><br><br>
<button type="button" id="register">注册</button>
<label id="msg_reg"></label>
</body>
</html>
```

在上方代码中，主体内容（body）的每个标签都和之后要写的脚本代码相关，所以全部加上“id”属性并设置属性值。

然后，我们添加 JavaScript 代码，代码可以添加在“head”或“body”中，这里我们添加在“head”中。

示例代码：

```
<head>
  <meta charset="UTF-8">
  <title>注册</title>
  //以上内容在上段代码中已存在，注意不要重复。
  <script type="text/javascript"
src="http://apps.bdimg.com/libs/jquery/2.1.4/jquery.min.js"></script>
  <script>
    $(document).ready(function () {
      ...代码块...（下文中的相关代码添加到这个位置）
    }
  </script>
  //以下内容在上段代码中已存在，注意不要重复。
</head>
```

在上方代码中，我们看到了“\$(document).ready(function)” ，这个事件函数能够将函数绑定到文档的就绪事件（当文档加载完成时）。

也就是说，当整个页面完成加载，我们就可以使用脚本中的功能。

1、添加检查用户名是否已注册的代码。

示例代码：

```
var registered = true;
$('#user_name').focusout(function () {
  var name = $('#user_name').val();
  if (name !== '') {
    $.getJSON("/check/", {'user_name': name}, function (result) {
```

```
        if (result === 100) {
            $('#msg_user').html('用户名已存在！');
            registered = true;
        } else if (result === 200) {
            $('#msg_user').html('');
            registered = false;
        }
    })
} else {
    $('#msg_user').html('请输入用户名！');
}
});
```

在上方代码中，变量“registered”用于记录用户名是否已注册的检查结果，变量“name”用于获取用户名文本框输入的内容。

当页面元素“（#user_name）”失去焦点“focusout”时，进行判断。

如果用户名不为空值，就调用 Ajax 函数“getJSON()”向服务器发起请求，并通过自定义变量“result”获取服务器返回的结果。然后，对返回结果进行判断，记录检查结果，并通过“html()”函数改变页面元素“（#msg_user）”显示的文字为相应提示内容。

否则，通过“html()”函数改变页面元素“（#msg_user）”显示的文字为“请输入用户名！”

这里需要注意，函数“getJSON()”的参数有 3 个，分别是：请求路径、参数字典和回调函数（function）。

2、添加提交注册的代码。

示例代码：

```
$('#register').click(function () {
    var name = $('#user_name').val();
    var pass = $('#password').val();
    if (registered === false && name !== "" && pass !== "") {
        $.getJSON("/register/", {'user_name': name, 'password': pass}, function (result) {
            if (result["status"] === 100) {
                $('#msg_user').html('抱歉，注册失败！');
            }
            if (result["status"] === 200) {
                $('#msg_reg').html('恭喜您，注册成功！');
            }
        })
    }
});
```

```
}  
});
```

在上方代码中，变量“name”用于获取用户名文本框输入的内容，变量“pass”用于获取密码文本框输入的内容。

当页面元素“(#register)”被鼠标点击“click”时，进行判断。

如果用户名未注册并且用户名和密码均不是空值，就调用 Ajax 函数“getJSON()”向服务器发起请求，并通过自定义变量“result”获取服务器返回的结果。然后，对返回结果进行判断，并通过“html()”函数改变页面元素“(#msg_user)”和“(#msg_reg)”显示的文字为相应提示内容。

二、配置 URL 分发

在“urls.py”文件中的“urlpatterns”列表中我们添加 3 行配置代码。

```
path('reg/', siteviews.reg), # 打开注册页面  
path('register/', siteviews.register), # 提交注册  
path('check/', siteviews.check), # 检查用户名是否注册
```

三、创建模型

在“models.py”模块中，我们定义一个用户类“Users”。

```
class Users(models.Model):  
    user_name = models.CharField(max_length=20, primary_key=True)  
    password = models.CharField(max_length=16)
```

然后，通过运行“makemigrations”和“migrate”命令完成数据库中数据表的创建。

四、定义视图函数

1、导入模块

示例代码：

```
from django.shortcuts import render, HttpResponse  
from MySite.models import Users  
import json
```

我们从视图向页面返回数据的时候，需要使用 json 对数据进行处理。

提示：JSON(JavaScript Object Notation：JS 对象标记) 是一种轻量级的数据交换格式，类似 XML，但是比 XML 更小、更快、更易解析。关于 JSON 大家可以参考：

<http://www.w3school.com.cn/json/>

2、打开注册页面的视图函数

示例代码：

```
def reg(request):  
    return render(request, 'register.html')
```

3、检查用户名是否注册的视图函数

```
def check(request):  
    user_name = request.GET['user_name']  
    user = Users.objects.filter(user_name=user_name)  
    if user:  
        status = 100 # 返回表示已注册的编号  
    else:  
        status = 200 # 返回表示未注册的编号  
    return HttpResponse(status)
```

检查用户名的结果，只需要通过“HttpResponse”返回，而无需通过“render”整合模板中的页面后返回。

4、实现用户注册的视图函数。

```
def register(request):  
    user_name = request.GET['user_name']  
    password = request.GET['password']  
    try:  
        user = Users(user_name=user_name, password=password)  
        user.save()  
        status = 200 # 返回注册成功的编号  
    except:  
        status = 100 # 返回注册失败的编号  
    return HttpResponse(json.dumps({'status': status}))
```

最后，我们再来看一个修改密码的功能。

127.0.0.1:8888/change/	127.0.0.1:8888/change/
<div><h3>修改密码</h3><div><input type="text" value="ccc"/></div><div>用户名不存在！</div><div><input type="text"/></div><div>提交</div></div>	<div><h3>修改密码</h3><div><input type="text" value="aaa"/></div><div><input type="text" value="..."/></div><div>提交 恭喜您，密码修改成功！</div></div>

通过上图，大家能够看到，当光标离开用户名文本框的时候，也会进行检查并给出提示。

也就是说，我们在视图中定义的函数“check()”依然可以在这里使用。

要实现的具体功能包括：

- 当光标离开用户名输入框时，如果没有输入任何内容，则在用户名后方提示“请输入用户名！”
- 当光标离开用户名输入框时，数据库表中不存在相同的用户名，则在用户名后方提示“用户名不存在！”
- 当点击提交修改按钮时，如果用户名和密码都已输入，则将用户名和密码更新到数据库中，并返回成功的状态编号；
- 根据返回的成功编号，在提交注册按钮后方提示“恭喜您，密码修改成功！”。

关于这些功能的实现，和注册用户的功能比较类似，大家可以通过以下示例代码进行理解。

一、模板文件（change.html）

示例代码：

```
<!DOCTYPE html>
<html lang="en">
<head>
  <meta charset="UTF-8">
  <title>修改密码</title>
  <script type="text/javascript"
src="http://apps.bdimg.com/libs/jquery/2.1.4/jquery.min.js"></script>
  <script>
    $(document).ready(function () {
      var registered = false;
      $('#user_name').focusout(function () {
        var name = $('#user_name').val();
        if (name === "") {
          $('#msg_user').html('请输入用户名！');
        }
        else {
          $.getJSON("/check/", {'user_name': name}, function (result) {
            if (result === 200) {
              $('#msg_user').html('用户名不存在！');
              registered = false;
            } else if (result === 100) {
              $('#msg_user').html("");
              registered = true;
            }
          })
        }
      })
    })
  </script>
</head>
<body>
  <div class="container">
    <div class="row">
      <div class="col-md-6">
        <div class="form-group">
          <input type="text" value="" class="form-control" id="user_name"/>
        </div>
        <div class="form-group">
          <input type="password" value="" class="form-control" id="password"/>
        </div>
        <div class="form-group">
          <input type="password" value="" class="form-control" id="password2"/>
        </div>
        <div class="form-group">
          <input type="button" value="提交" class="btn btn-primary"/>
        </div>
      </div>
      <div class="col-md-6">
        <div class="text-right">
          <div class="text" id="msg_user"></div>
        </div>
      </div>
    </div>
  </div>
</body>
</html>
```

```
        })
    }
});
$('#submit').click(function () {
    var name = $('#user_name').val();
    var pass = $('#password').val();
    if (registered && name !== '' && pass !== '') {
        $.getJSON("/changepass/", {'user_name': name, 'password': pass}, function
(result) {
            if (result["status"] === 200) {
                $('#msg_up').html('恭喜您, 密码修改成功!');
            } else {
                $('#msg_up').html('很抱歉, 密码修改失败!');
            }
        })
    }
});
</script>
</head>
<body>
<h3>修改密码</h3>
<input type="text" name="user_name" id="user_name">
<label id="msg_user"></label><br>
<input type="password" name="password" id="password"><br><br>
<button type="button" id="submit">提交</button>
<label id="msg_up"></label>
</body>
</html>
```

2、配置 URL 分发

示例代码：

```
path('change/', siteviews.change), # 打开修改密码页面
path('changepass/', siteviews.changepass), # 提交密码修改
```

3、定义视图函数

示例代码：

```
def change(request):
    return render(request, 'change.html')
```

```
def changepass(request):
    user_name = request.GET['user_name']
    password = request.GET['password']
    user = Users.objects.filter(user_name=user_name) # 查询已存在用户的数据对象
    try:
        user.update(password=password) # 通过数据对象更新数据库中的数据
        status = 200
    except:
        status = 100
    return HttpResponse(json.dumps({'status': status}))
```

上方代码中，使用“update()”方法实现了对数据库数据进行更新的操作。

提示：删除数据也可以通过查询到的数据对象（一个或多个）进行删除，使用“delete()”方法。

本节练习源代码：【[点此下载](#)】



转载请注明：魔力 Python » Django2 : Web 项目开发入门笔记 (9)

原文链接：<http://www.opython.com/967.html>

Django2 : Web 项目开发入门笔记 (10)

这一篇教程，我们把前面所学的内容做一下巩固练习，并且再接触一些新的知识内容，例如视图向 JS 返回数据时的处理以及 JQuery 中循环的使用等等。

先来看看，我们要实现的目标。



一、完成商品列表以及页面内容的呈现。

先不管添加和查询功能，我们先把上方呈现的页面实现出来。

1、创建模型（models.py）

这次我们创建的模型类，以商品名称为主键。

示例代码：

```
class GoodsInfo(models.Model):
    goods_name = models.CharField(max_length=30,
    primary_key=True
    )
    goods_number = models.IntegerField()
    goods_price = models.FloatField()
```

创建好模型类之后，通过“makemigrations”和“migrate”创建数据表，并将数据导入数据库。

提示：《Django2：Web 项目开发入门笔记（8）》中包含所需的数据内容和导入方法。

2、创建模板（goods_list.py）

在模板文件中我们写入 HTML 代码。

示例代码：

```
<!DOCTYPE html>
<html lang="en">
<head>
  <meta charset="UTF-8">
  <title>管理商品</title>
</head>
<body>
<h3>商品列表</h3>
价格查询：<input type="text" style="display:inline-block;width:100px;" id="min_price"> ~
<input type="text" style="display:inline-block;width:100px;" id="max_price">
<button type="button" id="search">确定</button>
<br>
<br>
<div id="list">
  {% for goods in goods_list %}
    <div id="{{ goods.goods_name }}">
      <label style="display:inline-block;width:120px;">名称：
      {{ goods.goods_name }}</label>
      <label style="display:inline-block;width:90px;">数量：
      {{ goods.goods_number }}</label>
      <label style="display:inline-block;width:90px;">价格：{{ goods.goods_price }}</label>
      <button class="delete" type="button" id="{{ goods.goods_name }}">删除</button>
      <br>
    </div>
  {% endfor %}
</div>
<br>
名称：<input type="text" style="display:inline-block;width:65px;" id="goods_name">
数量：<input type="text" style="display:inline-block;width:40px;" id="goods_number">
价格：<input type="text" style="display:inline-block;width:40px;" id="goods_price">
<button type="button" id="add">添加</button>
<label id="add_msg"></label>
```

```
</body>  
</html>
```

注意，上方代码中每个标签都有“id”属性，这个属性方便我们实现之后添加的功能。

特别是，商品列表中每个删除按钮的“id”，这里使用了每条商品信息中的商品名称做为属性值，实现删除功能时，就可以依据这个“id”属性的值进行删除。

另外，每一条商品信息的多个“label”标签和删除按钮的“button”标签，都写在一对“div”标签中。

这是因为，之后我们要对一整条记录进行删除操作，在页面中删除一条商品信息只需要移除对应的“div”就可以了。

还有，每一条商品信息对应的“button”标签中，需要添加“class”属性，属性值是自定义的，这样生成商品列表后，列表中所有的删除都属于同一个“class”，通过给这个“class”定义函数来实现每个按钮的删除功能。

至于，其它的代码，在之前的教程中都已接触过，在此不做详细解释。

3、配置 URL 分发

这里先添加一条访问页面的配置。

示例代码：

```
path('goods_list/', siteviews.goodslist),
```

4、定义视图函数

访问页面时，需要查询全部商品数据，与模板中的页面内容进行整合。

示例代码：

```
from django.shortcuts import render, HttpResponse # HttpResponse 在之后才会用到  
from MySite.models import Goods  
  
def goodslist(request):  
    result = Goods.objects.all()  
    return render(request, 'goods_list.html', {'goods_list': result})
```

当我们完成以上步骤，访问页面的时候，就能够看到前面展示的页面效果了。

接下来，在此基础上我们来分别完成商品添加、删除以及价格区间查询的功能。

这些功能的使用结果，全部是在页面无刷新的情况下呈现出来的。

二、完成添加商品并刷新列表的功能

当我们输入一条商品的商品信息（这里不做输入内容的验证），点击添加按钮，即可将这条商品数据添加到数据库，并且在页面上根据返回的代码，呈现提示信息“成功添加商品！”



当显示提示信息后，等待 1 秒钟，商品列表刷新，最后一条呈现新添加的商品信息。



1、模板中添加 JavaScript 脚本代码

实现这个功能，我们需要开始在模板文件（goods_list.html）中添加 JavaScript 代码。

在一对“head”标签中，我们先添加 JQuery 库文件和“script”标签以及一些固定的代码。

示例代码：

```
<script type="text/javascript"
src="http://apps.bdimg.com/libs/jquery/2.1.4/jquery.min.js"></script>
<script>
    $(document).ready(function () {
        ...之后的添加、删除、价格区间查询等功能代码...
    })
</script>
```


接下来，我们完成添加商品的功能代码。

示例代码：

```
$('#add').click(function () {  
    $('#add_msg').html("");  
    var name = $('#goods_name').val();  
    var price = $('#goods_price').val();  
    var number = $('#goods_number').val();  
    $.getJSON("/add/", {  
        'goods_name': name,  
        'goods_price': price,  
        'goods_number': number  
    }, function (result) {  
        if (result === 100) {  
            $('#add_msg').html('商品添加失败！');  
        }  
        if (result === 200) {  
            $('#add_msg').html('成功添加商品！');  
            setTimeout(function () {  
                location.reload()  
            }, 1000);  
        }  
    })  
});
```

在上方代码中，先通过定义 3 个变量获取 3 个商品信息文本框中输入的内容，然后通过“`getJSON()`”向服务器提交，并获取服务器响应的结果。

如果返回的结果是失败的编号，提示“商品添加失败！”；如果是成功的编号，则提示“成功添加商品！”。

然后，通过“`setTimeout()`”延时“1000”毫秒执行“`function()`”中的“`location.reload()`”，重载当前位置的内容，实现新商品信息的展示。

不过，“`location.reload()`”会向服务器再次发出请求获取全部商品数据，如果不想增加对服务器的访问，商品列表内容的增减也可以通过向列表中追加移除数据的方法来实现（参考下一篇教程中删除功能的再次实现）。

2、配置 URL 分发

继续添加一条新的配置语句，将“`getJSON()`”中请求的 URL 交由相应的视图处理。

示例代码：

```
path('add/', siteviews.add),
```

3、定义视图函数

示例代码：

```
def add(request):
    goods_name = request.GET['goods_name']
    goods_price = request.GET['goods_price']
    goods_number = request.GET['goods_number']
    isexist = Goods.objects.filter(goods_name=goods_name)
    try:
        if not isexist:
            goods = Goods()
            goods.goods_name = goods_name
            goods.goods_price = goods_price
            goods.goods_number = goods_number
            goods.save()
            result = 200
        else:
            result = 100
    except:
        result = 100
    return HttpResponse(result)
```

注意，在我们添加商品数据时，如果数据库的表中已经存在相同名称的商品，“save()”方法会更新已有数据，实现修改的效果。但是，很明显，我们当前不希望实现修改，而是没有商品时添加新的商品，已有商品的情况下不做任何改动。

所以，上方代码中，先进行商品查询，如果没有商品不存在，再进行保存，并返回成功的编号。

然后，无论是已存在商品还是保存发生异常，都返回失败的编号。

完成上述步骤后，我们就实现了添加商品的功能。

为避免篇幅过长，删除与价格区间查询的功能见下一篇教程《Django2：Web 项目开发入门笔记（11）》。

本节练习源代码：【见下一篇教程】



转载请注明：魔力 Python » Django2 : Web 项目开发入门笔记 (10)

原文链接：<http://www.opython.com/979.html>

Django2 : Web 项目开发入门笔记 (11)

这一篇教程，我们继续完成商品列表的删除和价格区间查询功能。

二、完成删除商品并从列表中移除的功能

当我们点击商品列表中任意一条商品信息后方的删除按钮，都能够将该条商品信息从商品列表中删除。

1、模板中添加 JavaScript 脚本代码

示例代码：

```
$('#button.delete').click(function () {  
    var name = $(this).attr('id');  
    $.getJSON("/del/", {'goods_name': name}, function (result) {  
        if (result === 200) {  
            location.reload()  
        }  
    })  
});
```

在上方代码中，第一行的语句表示这是一个页面中所有“class”属性为“delete”的“button”标签，在被点击（click）时，执行的脚本函数（function）。

第二句代码是创建了一个变量，获取当前（this）被点击标签的“id”属性值（attr）。

当获取到当前被点击按钮的“id”属性值（还记得吗？是商品名称），就可以通过“getJSON()”向服务器发起请求。

2、配置 URL 分发

示例代码：

```
path('del/', siteviews.delete),
```

3、定义视图函数

示例代码：

```
def delete(request):  
    goods_name = request.GET['goods_name']
```

```
goods = Goods.objects.filter(goods_name=goods_name)
try:
    goods.delete()
    result = 200
except:
    result = 100
return HttpResponse(result)
```

三、完成价格区间查询功能

当我们输入查询商品的最小价格和最大价格，点击确定按钮之后，页面中展示筛选后的商品列表。



1、定义视图函数

为了更好地理解视图和 JS 之间数据的传输，我们先来完成视图函数的定义。

示例代码：

```
def search(request):
    min_price = int(request.GET['min_price'])
    max_price = int(request.GET['max_price'])
    goods = Goods.objects.filter(goods_price__gte=min_price, goods_price__lte=max_price)
    try:
        if goods:
            result = json.dumps(serializers.serialize('json', goods))
        else:
            result = 100
    except:
```

```
    result = 100
    print(result)
    return HttpResponse(result)
```

在上方代码中，我们依据最小价格和最大价格进行数据库查询，如果未查询到结果或者发生异常，都返回错误的编号，否则返回一个经过序列化和格式化的结果。

其中，“serializers.serialize('json', goods)”是将查询结果（QuerySet 类型）序列化为 JSON 格式的字典。

而“json.dumps()”则是将字典序列化为 JSON 格式的字符串。

通过这两次序列化，我们就能够在 JS 中获取到查询结果的数据。

2、配置 URL 分发

```
path('search/', siteviews.search),
```

3、模板中添加 JavaScript 脚本代码

示例代码：

```
$('#search').click(function () {
    $('#list').empty();
    var min_price = $('#min_price').val();
    var max_price = $('#max_price').val();
    $.getJSON("/search/", {
        'min_price': min_price,
        'max_price': max_price
    }, function (result) {
        var labels = "";
        if (result === 100) {
            labels = '<table>没有查询结果！</table>';
        } else {
            var data = JSON.parse(result);
            for (i in data) {
                labels += '<div id="' + data[i]['pk'] + '">';
                labels += '<label style="display:inline-block;width:125px;">名称：' + data[i]['pk'] +
                '</label>' +
                '<label style="display:inline-block;width:95px;">数量：' +
                data[i]['fields']['goods_number'] + '</label>' +
                '<label style="display:inline-block;width:95px;">价格：' +
                data[i]['fields']['goods_price'] + '</label>' +
                '<button class="delete" type="button" id="' + data[i]['pk'] + '">删除
                </button><br>';
            }
        }
    });
});
```

```
        labels += '</div>';
    }
}
$('#list').append(labels)
}
});
```

在上方代码中，有一些需要注意的地方：

- (1) 第二句代码“`$('#list').empty();`”，表示将“id”属性值为“list”的元素所包含的子元素清空，即清空商品列表。
- (2) 变量“labels”负责组织一条商品信息包含的 HTML 代码，并且在这些代码中动态的嵌入服务器返回的数据。
- (3) 变量“data”保存通过“`JSON.parse()`”解析“result”数据后得到的 JSON 对象。
- (4) 使用“for”循环对“data”进行遍历，并生成每一条商品信息的 HTML 代码。
- (5) 我们从服务器获取到的一条商品数据类似“`{“model”: “MySite.goods”, “pk”: “\u81ea\u52a8\u7b14\u82af”, “fields”: {“goods_number”: 36, “goods_price”: 2.1, “goods_sales”: 0}}`”的格式，由模型、主键和字段 3 个部分组成，我们要获取到主键的值和各个字段的值添加到每条商品信息的 HTML 代码中。主键是商品名，通过“`data[i][‘pk’]`”获取，其他商品信息则通过“`data[i][‘fields’][‘字段名称’]`”获取。
- (6) 当所有的商品信息的 HTML 代码全部生成，通过“`$('#list').append(labels)`”添加到“id”属性为“list”的元素中，即形成新的商品列表。

提示，在浏览器中通过 JS 加载的商品信息可能会出现宽度发生改变，例如上方 JS 脚本代码中的每个“label”标签的宽度都增加了 5px。

四、删除的再次实现。

为什么要再次实现呢？

大家在进行价格区间的查询之后会发现，点击删除按钮没有任何反应。

这是因为这些查询后的商品信息是通过 JS 动态加载的，而不是页面上原有的 HTML 代码。

那么，如何让删除按钮生效，在点击的时候能够向服务器发起删除的请求，并将页面中相应的商品信息删除呢？

我们可以使用 `on()` 函数重写删除的 JS 代码。

示例代码：

```
$(document).on('click', 'button.delete', function () {  
    var name = $(this).attr('id');  
    $.getJSON("/del/", {'goods_name': name}, function (result) {  
        if (result === 200) {  
            $('div').remove('#' + name);  
            {#$('div').remove();#}{#也可以使用这一句#}  
        }  
    })  
});
```

在上方代码中，第一句会让页面文档（document）中，所有“class”属性值为“delete”的“button”标签被点击（on click）时，执行函数（function）的语句块。

语句块中，变量“name”保存获取到的当前（this）标签的“id”属性值（attr）；然后，当服务器返回成功的编号时，从所有“div”标签中移除“id”属性值与变量 name 相同的子元素。

提示：也可以像注释（{##}）中一样，写成“\$('div').remove();”，即将页面中所有“id”属性值与变量 name 相同的元素移除。

注意，这里不要用“location.reload()”，不仅仅是因为会再次向服务器发出请求，而是这样做的话商品列表中会显示所有商品信息，不再是查询结果删除某一项后剩余的商品信息。

本节练习源代码：【[点此下载](#)】



转载请注明：魔力 Python » Django2 : Web 项目开发入门笔记（11）

原文链接：<http://www.opython.com/986.html>

Django2 : Web 项目开发入门笔记（12）

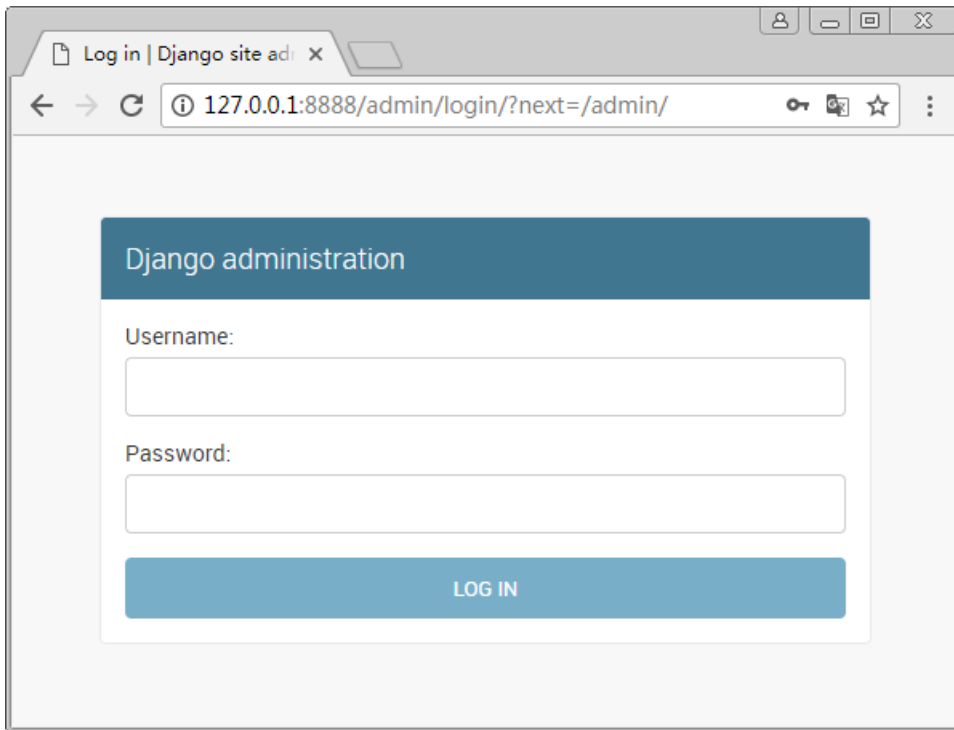
这一篇教程，我们一起来了解 Django 自带的后台。

Django 自带的后台功能已经比较完善，我们要了解的主要是如何使用它。

一、登录

当我们启动开发服务器，就可以通过“<http://127.0.0.1:端口号/admin>”进行访问了。

此时，会显示登录界面。



我猜你不知道用户名和密码，对吧？

嘿嘿，因为还没有创建管理员和密码。

接下来我们进行创建。

二、创建管理员

运行“manage.py”任务窗口，输入命令：`createsuperuser`（回车）

此时，就会要求输入用户名和两遍相同的密码，至于邮箱地址，不想输入的话，可以回车跳过。

如果设置的密码想修改的话，可以输入命令：`changepassword` 用户名（回车）

就可以进行密码的修改了。

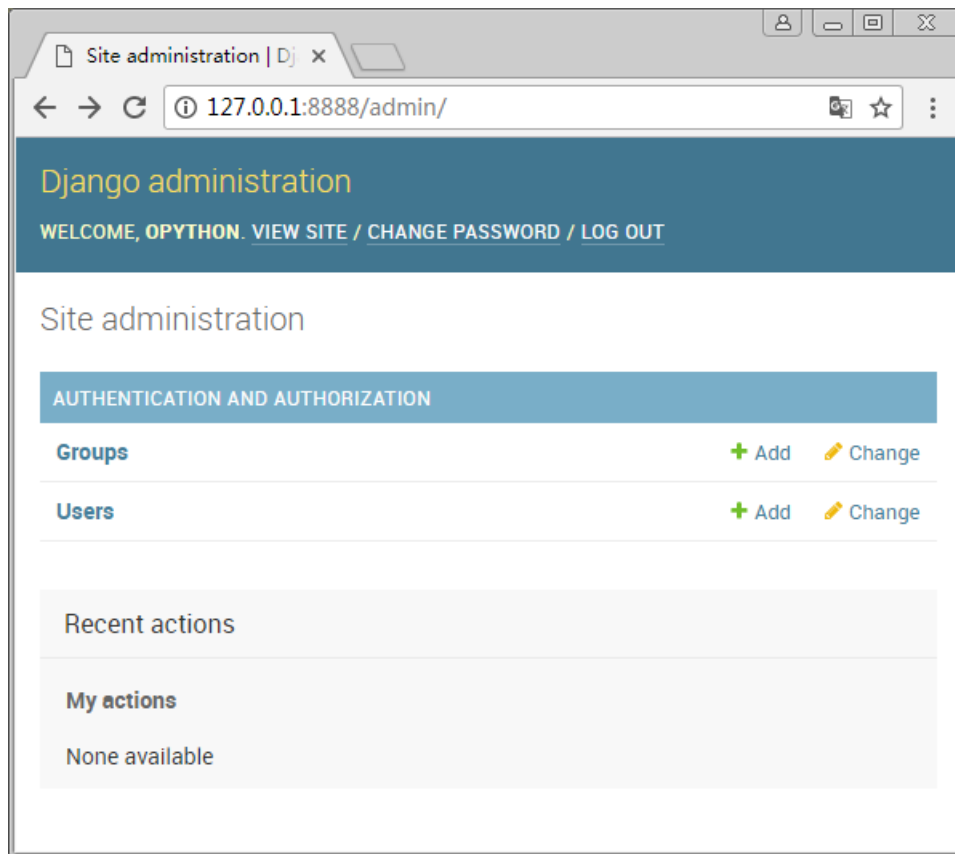

```
manage.py@MyWeb > createsuperuser          创建超级用户
"C:\Program Files\JetBrains\PyCharm 2017.3.3\bin\runnerw.exe" D:\MyWeb\venv\Scripts\python.exe
Username (leave blank to use 'administrator'): opython
Email address:                               邮箱可以不填（回车跳过）
Warning: Password input may be echoed.
Password: opython.com                        设置密码
Warning: Password input may be echoed.
Password (again): opython.com                重复密码
Superuser created successfully.

Following files were affected
D:\MyWeb\db.sqlite3
Process finished with exit code 0
manage.py@MyWeb > changepassword opython    修改用户密码
"C:\Program Files\JetBrains\PyCharm 2017.3.3\bin\runnerw.exe" D:\MyWeb\venv\Scripts\python.exe
Changing password for user 'opython'
Warning: Password input may be echoed.
Password: www.opython.com                   设置密码
Warning: Password input may be echoed.
Password (again): www.opython.com           重复密码
Password changed successfully for user 'opython'

Following files were affected
D:\MyWeb\db.sqlite3
D:\MyWeb\.idea\workspace.xml
Process finished with exit code 0

manage.py@MyWeb >
```

当我们完成超级用户（管理员）的创建，再次回到登录界面，就可以进行登录了。



打开之后，觉得很迷茫对吧？

不如，我们先把页面改成中文的看一看。

三、修改后台语言

打开文件“settings.py”，拉到末尾，我们会看到一项设置。

```
LANGUAGE_CODE = 'en-us'
```

Django 支持很多种语言，默认语言是美国英语。

我们可以打开 Django 库的路径：项目文件夹\venv\Lib\site-packages\django\contrib\admin\locale

这个路径下，每一个文件夹的名称代表了一种语言。

如果需要改成中文的话，我们把“LANGUAGE_CODE”的值改成“zh-Hans”，也就是中文汉字简体，“s”表示简体。

提示：繁体中文是“zh-Hant”，也就是中文汉字繁体，“t”表示台湾。

完成这项设置的修改之后，我们就能够看到中文界面了。



图中的“认证和授权”中，能够对用户和组进行管理。

用户：包括用户的创建、删除、信息修改、权限分配等。

组：创建不同的组别，给予相应的权限，就能够将用户划分到不同的组中，拥有相应的权限，适合大量用户的权限管理。

四、添加数据

一般来说，数据库中的全部或部分数据是需要后台进行管理的。

但是，当前后台中，并没有这些数据。

以我们之前已经创建好的模型“Goods”举例，如何把通过这个模型创建的数据表中的数据呈现在后台呢？

操作非常简单！

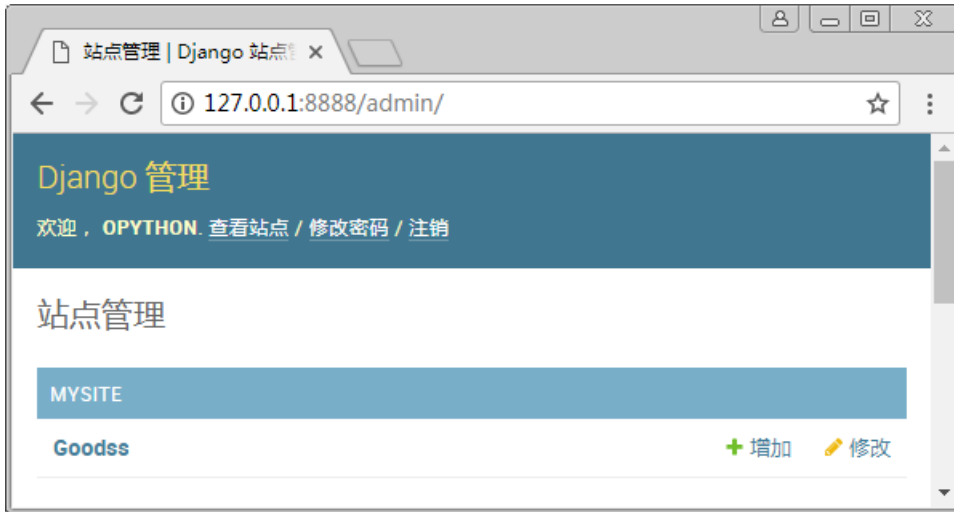
打开文件“admin.py”，在里面我们导入模型类“Goods”，并加入一行代码。

示例代码：

```
from django.contrib import admin
from MySite.models import Goods
```

```
admin.site.register(Goods)
```

这时，我们刷新后台页面看一看。



此时，在页面中我们看到了 Web 应用的名称和模型中类的名称，不过类的名称后面多了个 s，表示这里面可能包含多个数据对象。

不过，现在显示的站点名称和表名称都是英文，我们也可以给改成中文。

五、修改站点名称、表名称以及数据对象名称为中文

首先，打开文件“apps.py”，当前只有一行定义“name”变量的代码，我们在下方再添加一行代码。

示例代码：

```
verbose_name='我的商店'
```

这句代码就能够让站点名称显示为指定的名称。

然后，再打开模型文件“models.py”，在“Goods”类中，创建一个“Meta”类，并写入一行代码。

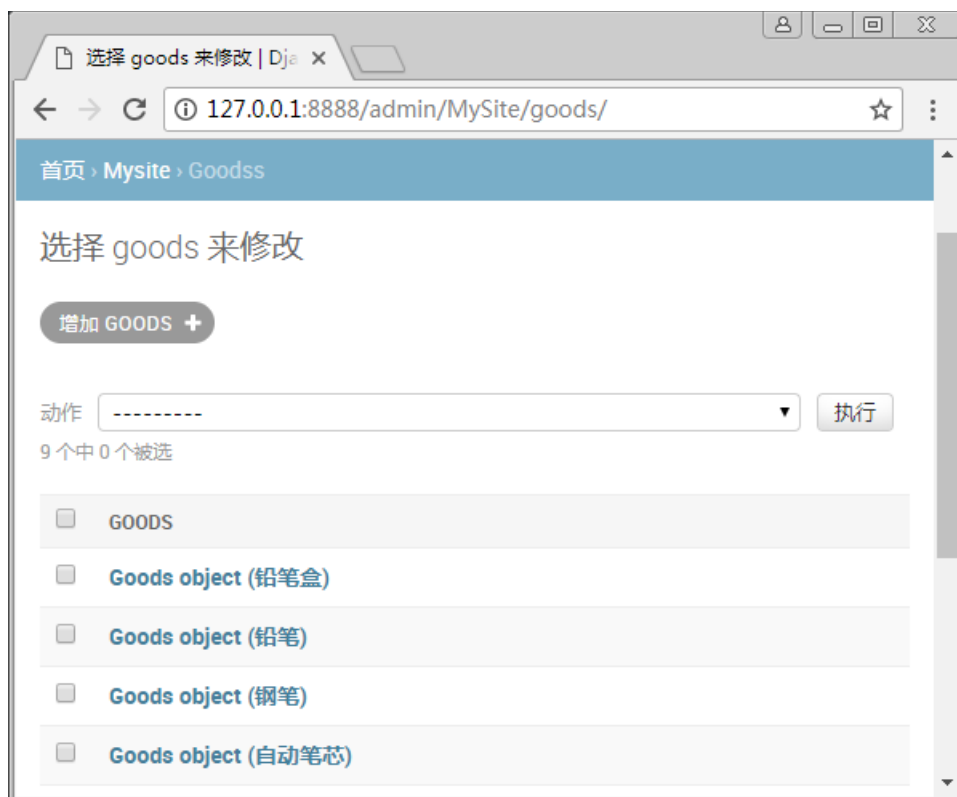
示例代码：

```
class Goods(models.Model):
    ...省略已有代码...
    class Meta:
        verbose_name_plural = "商品管理"
```

再次刷新页面，就可以看到我们要的结果了。



接下来，我们点击“商品管理”这个表名称看一看。



在打开的页面中，还有英文的“goods”或“GOODS”。

这些也需要改过来的话，我们在刚才的“meta”类中，再加入一行代码。

示例代码：

```
verbose_name = "商品"
```

页面刷新后，我们能够看到除了下方列表，中的“Goods object”，其余的“Goods”都变成了“商品”。

那么，列表中的“Goods object”怎么处理呢？

列表就是之前所说的多个数据对象的列表。

这些“Goods object”都是数据行标题，以 Goods object（商品名称）的形式呈现。

这样显示数据行标题不是很友好，我们应该比较期望这里是一条商品信息的数据内容。

解决这个问题，我们可以在“Goods”类中，再添加一段代码。

示例代码：

```
def __str__(self):  
    return self.goods_name
```

这一段代码的意思让数据行标题显示为商品名称。



但是，只有商品名称还不够，如果能够把其他的商品信息也显示出来就更好了。

六、显示更多数据字段

如果想显示更多的数据字段，我们需要打开文件“admin.py”，创建一个新的类。

示例代码：

```
class GoodsAdmin(admin.ModelAdmin):  
    list_display = ('goods_name','goods_number','goods_price','goods_sales')
```

变量“list_display”就是用来定义数据列表中显示哪些的字段。

然后，我们再将之前的语句：

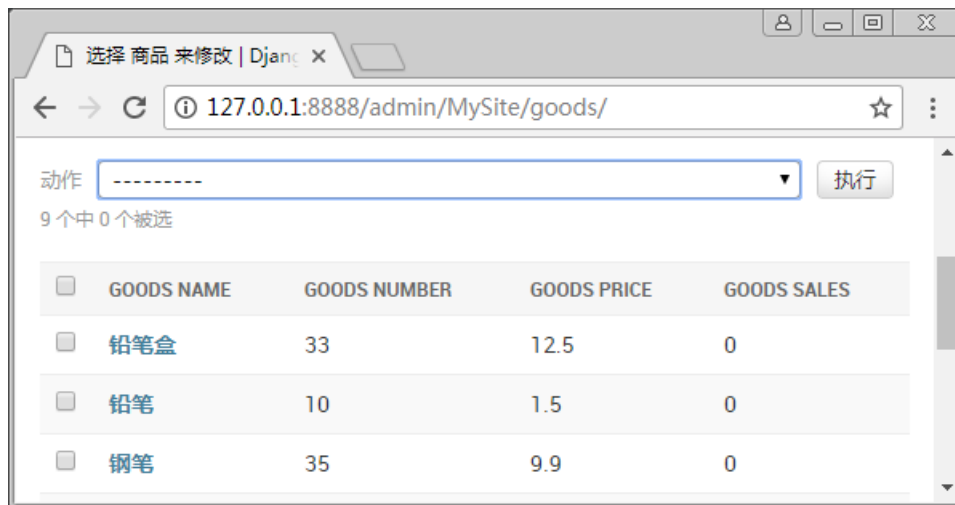
```
admin.site.register(Goods)
```

修改为：

```
admin.site.register(Goods,GoodsAdmin)
```

也就是将新增的类进行注册。

刷新页面，我们能够看到更多的数据字段了。



但是，新的问题又来了，列表中列名都是英文。

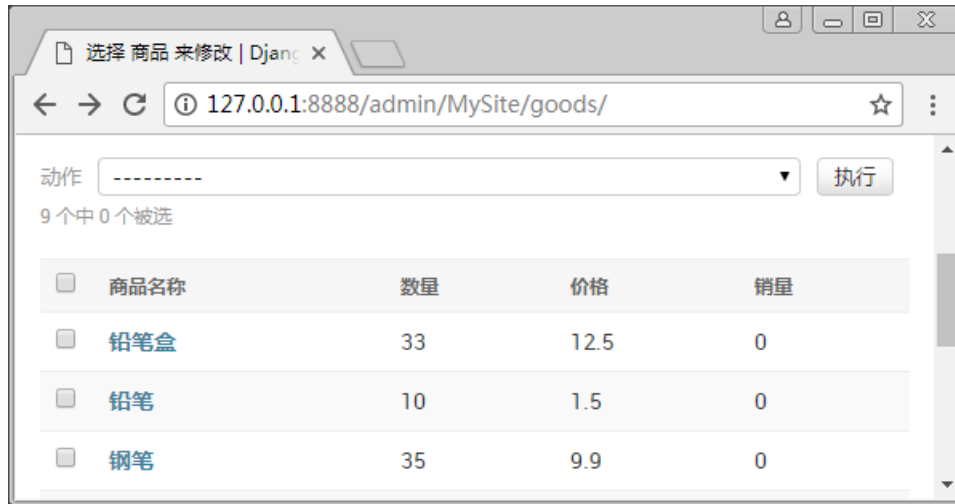
七、修改数据表的列名为中文。

这个很简单，我们把文件“models.py”中“Goods”类的每个变量赋值进行修改，在字段类型的参数中，第一个参数位置添加中文名称。

示例代码：

```
goods_name = models.CharField('商品名称', max_length=30, primary_key=True)  
goods_number = models.IntegerField('数量')  
goods_price = models.FloatField('价格')  
goods_sales = models.IntegerField('销量', default=0)
```

经过这样的修改，就能看到数据列表中所有的列名都变成了中文名称。



八、数据对象列表中添加非数据字段内容

假如，我们想在上方的数据表中，再添加一列内容，是销量的金额可不可以呢？

当然可以！

我们在 `Goods` 类中添加新的代码。

示例代码：

```
def sales_volume(self):  
    return self.goods_sales * self.goods_price
```

```
sales_volume.short_description = '销售额'  
goods_amount = property(sales_volume)
```

先通过一个方法“`sales_volume`”计算销售额，再将这个方法转为类的属性。

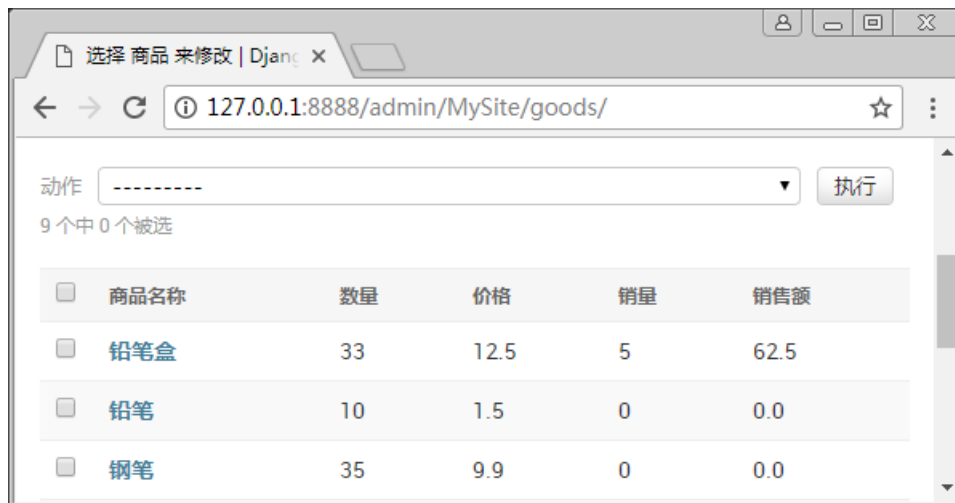
这个属性的名称可以通过“`sales_volume.short_description`”进行定义。

然后，我们在文件“`admin.py`”中将“`list_display`”修改为：

```
list_display = ('goods_name','goods_number','goods_price','goods_sales','goods_amount')
```

完成上面的操作之后，我们刷新页面，就能够在列表中看到新增的“销售额”这一列了。

提示：之前的商品销量都是“0”，可以修改之后查看销售额的变化。



以上就是关于 Django 后台的使用介绍。

本节练习源代码：【[点此下载](#)】



转载请注明：魔力 Python » Django2 : Web 项目开发入门笔记 (12)

原文链接：<http://www.opython.com/1002.html>

Django2 : Web 项目开发入门笔记 (13)

这一篇教程，我们一起来了解如何在 Ubuntu 系统中将 Django2 的 Web 项目部署到 Apache 服务器。

这里环境搭建的内容包括：

- 操作系统：Ubuntu16.04：
- Web 服务器：Apache2.4
- 解释器：Python3.6
- 框架：Django2.0.3

在进行正式操作之前，我先做下总结。

就这样一个组合，整整虐待了我三天半的时间，大大小小的坑层出不穷。

不过，正是因为有坑才能学到本领。

想学会跑，就不能怕摔倒，而且还要有勇气爬起来。

在这三天半的时间里，我参照网上的各种相关资料，几乎没有一个行得通。

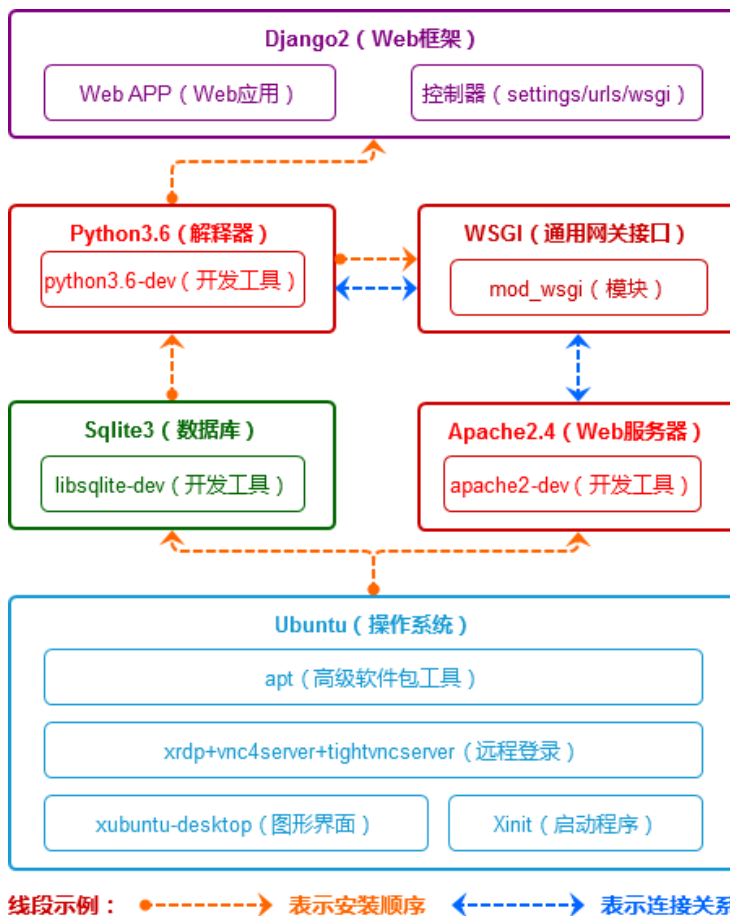
不是使用的 Python 或 Django 版本较低，就是缺乏实践验证的抄袭之作。

特别是有些错误，根本没有资料可查，全靠自己分析。

在接下来的教程中，我会对各种坑给出说明，希望能够为初学者以及开发人员提供帮助。

好了！接下来，进入正题。

首先，我们先来看一下使用的系统以及相关的软件程序。



在上图中，包含了我们要安装的所有系统支持程序和应用程序。

注意：橘黄色线段指出的是安装顺序，不按照顺序安装就会掉到坑里。

然后，我们分五个步骤来完成 Django 的部署。

- 添加系统支持
- 添加软件包支持

- 创建配置文件
- 使用虚拟环境
- 权限设置

提示：我会边做边写，所以可以保证这篇教程的有效性！

这一篇教程，我们先完成前两个步骤。

一、添加系统支持

1、安装系统

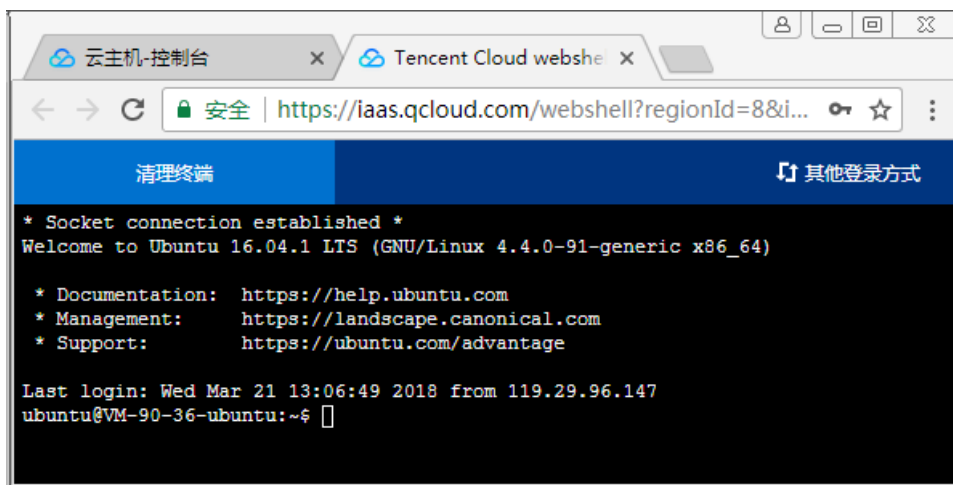
关于如何安装系统，这里省略，大家可以参考网上的一些资料，基本上都能安装完成。

安装系统有几种方式：

- 直接安装在电脑上。（应该很少有人这么干）
- 使用 VMware 或者 VirtualBox 安装在虚拟机中。（这个不错）
- 使用阿里云或者腾讯云的云主机（云服务器）。（有钱可以用这种，比如我，主要是重装系统省心省时间）

提示：腾讯云和阿里云都提供了面向个人用户的免费体验云主机（有时限）。

我用的是腾讯云服务器，安装完系统就能够进入命令行模式的界面了。



2、基本命令

既然是命令行模式，我们就要掌握一些基本命令：

- `sudo`：系统管理指令，是允许系统管理员让普通用户执行一些或者全部的 `root` 命令的一个工具。
- `sudo -i`：获取 `root` 权限。

- `rm -f [文件路径]`：删除文件指令。
- `rm -r [目录路径]`：删除非空文件夹指令。
- `mv [文件路径] [目标路径]`：移动文件命令。
- `cp [文件路径] [目标路径]`：复制文件指令。
- `cd [目标路径]`：进入目标文件夹指令。
- `cd /`：返回根目录指令。（注意空格）
- `cd ..`：进入上级目录。（注意空格）
- `cd -`：返回最近一次打开的目录。（注意空格）

了解了这些基础的操作指令，对于我们完成任务已经够用了。

3、安装启动程序

安装命令：`sudo apt install xinit`

命令中的 `apt` 是指高级软件包工具（Advanced Package Tool），本身集成了大量软件程序，能够快速帮助我们完成软件程序的安装。（也可以使用 `apt-get` 代替 `apt`，但是不建议这样做，`apt` 是友好版的 `apt-get`，提供进度条和彩色字符等功能）

而安装的“Xinit”通常用在启动 X（图形架构）时执行窗口管理器和其他程序。

第 1 个坑：这个如果不安装，后面的有些程序会安装失败。

4、安装图形界面

有些操作还是在图形界面中比较方便。

这里我选择安装的是 `Xubuntu-desktop`。

这个桌面环境基于桌面环境 `Xfce`，主要面向旧式电脑的用户和寻求更快捷的桌面环境的用户。

因为毕竟是服务器，图形界面会带来大量的资源消耗，没有必要使用比较华丽的界面。

另外，`Ubuntu16.04` 集成了 `LightDM`（桌面显示管理器），可以很好的支持 `Xubuntu-desktop`，所以就省略了安装与之对应的桌面显示管理器 `XDM`。

安装命令：`sudo apt-get install xubuntu-desktop`

安装过程中会提示：`Do you want to continue? [Y/n]`

输入“y”并按下回车键继续安装程序。

提示：很多安装程序都会出现这个提示，照此操作就可以了，后文不再赘述。

等到命令行再次出现光标闪烁，安装过程就结束了。

5、安装远程登录

安装远程登录是想通过 Windows 的远程桌面连接工具连接云主机，方便进行一些图形界面下的操作，如果是本机或虚拟机中安装的 Ubuntu 系统可以略过此步骤。

第 2 个坑：这里最好不要用“apt”，会出现“Sub-process /usr/bin/dpkg returned an error code (1)”错误。

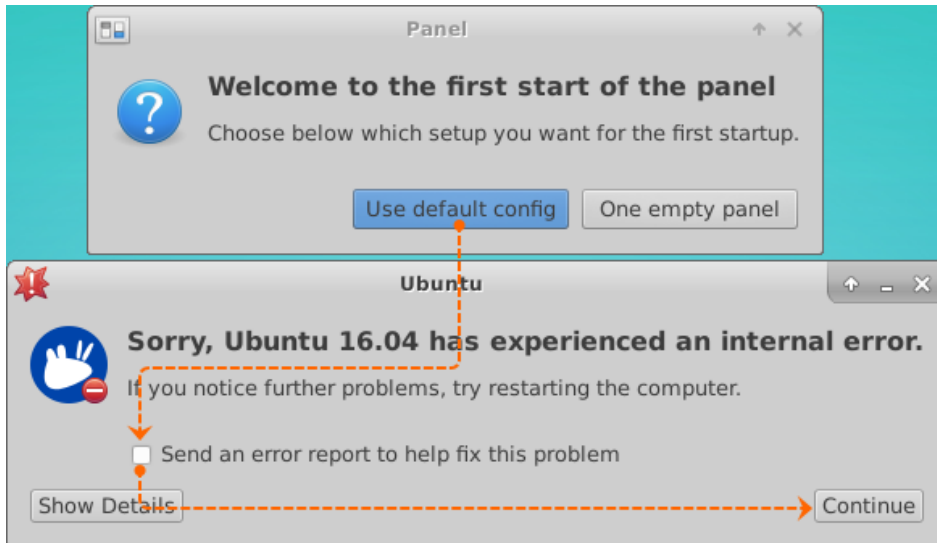
安装命令：`sudo apt-get install xrdpsudo apt-get install vnc4server tightvncserver` 完成安装之后，就可以打开 Windows 的远程桌面连接工具，输入云服务器的公网 IP 进行远程连接了。

第 3 个坑：如果远程登录输入密码后提示“`xrdp_mm_process_login_response: login failed`”，可以通过命令“`sudo service xrdp restart`”尝试重新启动 xrdp 来解决。

在打开的连接界面中，输入 Ubuntu 系统的用户名和密码，就进入到了云服务器主机的桌面中。

因为是第一次登录图形界面，会有一个提示，点击蓝色的按钮（使用默认配置）即可，另外一个按钮是创建一个空界面。

假如还有另外一个错误提示，不用管它，取消勾选后，点继续按钮即可。



图形界面支持中文，在桌面左上角所有应用程序【Applications】的设置【Settings】中，找到语言支持【Language Support】选项，此时会提示安装语言，点击安装按钮

（Install），输入系统密码后，按回车键进行安装。安装后即可在设置界面中安装/卸载语言【Install/Remove Languages】，右侧列表中找到简体中文【Chinese (simplified)】后勾选，并点击应用【Apply】。

此时，又会提示输入系统密码，输入密码并生效按钮【Authenticate】后，开始应用过程，简体中文语言名称【汉语（中国）】就会出现在语言列表中，用鼠标指针点住名称，拖动到默认的英文【English】上方后松开，然后点击应用到系统【Apply System-Wide】。

最后，在所有应用程序【Applications】菜单中选择注销【Log Out】，再选择重启系统【Restart】就能够使用中文图形界面了。

当然，也可以使用命令行重启。

执行命令：`sudo reboot`

6、更新高级软件包工具

提示：如果使用的云主机，接下来的操作建议在远程连接界面中使用命令行终端执行（桌面下方有命令行终端的图标）。

高级软件包工具包含了很多常用的软件程序，但是很多都已经不是新版本，或者有些我们需要的程序不在其中。

所以，我们要进行更新操作。

不过，我们后面再需要使用到的 `Python3.6-dev` 并不在已集成的程序中，所以我们需要添加软件更新源的地址。

执行命令：`sudo vi /etc/apt/sources.list`

此时会在命令行终端通过 `vi` 编辑器打开文件“`sources.list`”，我们按“`i`”键（注意别按多了）进入编辑模式，在文件的开始我们添加一行软件源的地址：`deb http://cz.archive.ubuntu.com/ubuntu bionic main`

第 4 个坑：添加这个软件源地址，是因为这个软件源中包含我们需要的软件包“`python3.6-dev`”，系统中没有集成，自带软件源中也没有，所以要添加这个软件源并同步更新高级软件包工具。

添加完毕后，按“`ESC`”键退出编辑模式，然后输入“`:wq`”（屏幕左下角会出现输入的内容）保存退出。

如果编辑错了，或者不想保留编辑的内容，想要退出的话可以输入“`:q!`”强制退出。

如果只是查看即关闭的话，命令是“`:q`”。

完成了软件更新源的设置，我们升级高级软件包工具。

第 5 个坑：这里最好不要用“`apt`”，会出现一个 `Python3-aptdaemon` 不完整程序包的错误，还是使用“`apt-get`”比较稳妥。

先同步一下更新源。

执行命令：`sudo apt-get update`

然后进行程序更新。

执行命令：`sudo apt-get upgrade`

提示：安装过程漫长，但不要离开，因为有些选项提示需要输入“y”或者“n”或者提示中的其他字母或数字。

第 6 个坑：在执行上方命令过程中，如果意外退出命令行终端，可能会导致执行其它 `apt` 执行命令无效，提示被锁定以及其他进程正在使用。

解决办法：

-
-
-

到这里需要的系统支持就操作完成了。

二、添加软件包支持

1、安装依赖文件 `sudo apt install openssl``sudo apt install libssl-dev``sudo apt install zlib`*第 7 个坑：如果不安装“`openssl`”和“`libssl.dev`”，在线安装一些第三方库（例如 Django）时，“`https://`”开头的下载地址无法连接，提示“SSL”相关的错误。

第 8 个坑：如果不安装“`zlib*`”，在安装之后的一些软件包时，软件包中的一些文件无法解压释放，导致安装失败。

2、安装 Sqlite3 和依赖

第 9 个坑：如果不先安装 Sqlite3，而是先安装 Python3.6，会导致找不到“`_sqlite3`”模块的错误。

安装命令：`sudo apt install sqlite3``sudo apt install libsqlite3-dev` 第 10 个坑：如果不安装 Sqlite3 的依赖包“`libsqlite3-dev`”，在 Python 代码中导入“`sqlite3`”模块，会出现错误“`ModuleNotFoundError: No module named '_sqlite3'`”。特别注意软件包名称是“`libsqlite3-dev`”不是“`libsqlite-dev`”，后者也能安装，但不支持 Python3.6。

3、安装 Apache2.4 和依赖

安装命令：`sudo apt install apache2``sudo apt install apache2-dev` 第 11 个坑：如果不安装 Apache2 的依赖包“`apache2-dev`”，后面安装“`mod_wsgi`”时会出现“`apxs: not found`”的错误。

4、安装 Python3.6 和依赖

首先，下载 Python3.6 的安装包（这里是 Python-3.6.5rc1.tgz），这里是放在 /home/ubuntu/Downloads/ 目录中。

如果是远程连接云主机，可以在本地下载后，使用软件 WinSCP 上传到云主机的 /home/ubuntu/Downloads/ 目录下。

如果是在图形界面中，可以在安装包上点击右键，选择解压缩到当前文件夹（Extract Here）。

也使用命令行。

执行命令：`cd /home/ubuntu/Downloads/sudo tar xzf Python-3.6.5rc1.tgz` 这里使用 `xfz` 命令，因为使用 `-xvzf` 命令释放的文件夹需要 `root` 权限才可以更改或者删除。

释放完成后，进入解压后的文件夹。

执行命令：`cd /home/ubuntu/Downloads/Python-3.6.5rc1` 然后，就能进行安装了。

首先，进行安装配置。

执行命令：`sudo ./configure --enable-shared --with-ssl=openssl` 第 12 个坑：此命令后面的参数一定要带上，如果不带“`--enable-shared`”参数，将导致“`mod_wsgi`”安装失败，如果不带“`--with-ssl=openssl`”，会导致使用 `pip` 命令在线安装下载地址为“`https://`”开头的第三方库时，无法连接的错误（SSL 错误）。

然后，创建安装文件。

执行命令：`sudo make`

第 13 个坑：如果之前已经进行过安装过程，这里可能会提示错误，可以通过执行“`sudo make clean`”先清空之前的安装残留，再执行“`sudo make`”命令。

最后，开始执行安装。

执行命令：`sudo make install`

接下来，安装依赖。

执行命令：`sudo apt install python3.6-dev` 第 14 个坑：如果没有安装这个依赖，下一步安装“`mod_wsgi`”，会出现“`src/server/wsgi_python.h:24:20: fatal error: Python.h: No such file or directory`”的错误。

5、安装 mod_wsgi 建立 python 与 apache 的连接

第 15 个坑：不能用 `pip` 命令安装 `mod_wdgi`，原因见下一个坑。

下载安装包 `mod_wsgi-4.6.2.tar.gz`，这个版本支持 Python3.6，解压后进入文件夹。

执行命令：`cd /home/ubuntu/Downloads/mod_wsgi-4.6.2` 然后，进行安装配置。

执行命令：`sudo ./configure --with-python=python3.6` 第 16 个坑：如果不加参数“`--with-python=python3.6`”，后果就是 `wsgi` 会连接系统默认的 `python2.7`，导致错误，错误提示中有“`include python2.7`”的内容。这是最大的一个坑，网上很难查到资料，我是尝试使用“`./configure --with-python3.6`”产生的帮助提示中找到的答案。

接下来，创建安装文件并进行安装。

执行命令：`sudo make` `sudo make install` 也可以写成：`sudo make && sudo make install`

第 17 个坑：这样写有时候会执行会出错，特别注意第二个“`sudo`”别漏写。

6、安装 Django2

可以使用 `pip` 命令安装。

第 18 个坑：不能直接执行 `pip` 命令，因为系统中自带 Python2.7 和 Python3.5 分别占用了“`python`”和“`python3`”这两个命令，如果想给 Python3.6 安装 Django（也只有 Python3.6 才能装 Django2），必须先进入 Python3.6 的“`site-packages`”目录。

执行命令：`cd /usr/local/lib/python3.6/site-packages/pip` `install django` 在线安装有些复杂，其实我们也可以通过安装包进行安装。

下载安装包 `Django-2.0.3.tar.gz`，解压后进入文件夹。

执行命令：`cd /home/ubuntu/Downloads/Django-2.0.3` `sudo python3.6 setup.py install`

第 19 个坑：`pytz` 是 Django 的依赖，在执行 Django 的安装时也会自动在线安装，但是有可能导致安装停止。可以参考在线安装的方法，先安装 `pytz`。

再次：为 Python3.6 安装第三方库，一定要先进入“`/usr/local/lib/python3.6/site-packages/`”目录，再执行 `pip` 安装命令。

最后，我们做一下测试，看一下是否一切正常。

执行命令：`python3.6>>>import django>>>import sqlite3`

如果没有任何错误出现，我们就完成了所需软件包的安装。

```
ubuntu@VM-90-36-ubuntu:~/Downloads/Python-3.6.5rc1$ python3.6
Python 3.6.5rc1 (default, Mar 21 2018, 18:52:18)
[GCC 5.4.0 20160609] on linux
Type "help", "copyright", "credits" or "license" for more information.
>>> import sqlite3
>>> import django
>>> |
```



转载请注明：魔力 Python » Django2：Web 项目开发入门笔记（13）

原文链接：<http://www.opython.com/1019.html>

Django2：Web 项目开发入门笔记（14）

这一篇教程，我们继续了解如何将 Django 项目部署到 Web 服务器。

既然是将项目部署到服务器，就需要有项目内容，例如把《Django2：Web 项目开发入门笔记（10）》和《Django2：Web 项目开发入门笔记（11）》部署到服务器上。

除此之外，如果拥有一个域名就更好了，将域名解析到主机的 IP 地址上，就能够通过域名进行访问了。








这里，我使用的域名是：www.qqtbb.com。

接下来，我们通过命令先给文件夹/var/www 赋予可读写的权限。

执行命令：`sudo chmod 777 /var/www`

然后，通过 WinSCP 将项目文件夹以及里面的内容上传到/var/www 这个文件夹中。

注意，项目文件夹的名称是“MyWeb”，在这个文件夹中还有一个同名文件夹“MyWeb”，不要把这两个搞混。

/var/www/MyWeb/					
名字	大小	已改变	权限	拥有者	
		2018-03-21 22:47:52	rw-rw-rw-r	root	
 templates		2018-03-21 22:47:55	rw-r-xr-x	ubuntu	
 static 用于存放js/css/图片等文件		2018-03-22 10:00:10	rw-rw-r-x	ubuntu	
 MyWeb		2018-03-21 23:39:31	rw-rw-r-x	ubuntu	
 MySite		2018-03-21 22:47:55	rw-r-xr-x	ubuntu	
 media 用于存放上传和下载的文件		2018-03-22 09:59:46	rw-rw-r-x	ubuntu	
 db.sqlite3	46 KB	2018-03-16 19:04:33	rw-rw-r--	ubuntu	

在上面的图中，大家能够看到新建了“static”和“media”两个文件夹。

这两个文件夹的使用，需要在“settings.py”文件中添加配置。

示例代码：

```
STATIC_URL = '/static/'  
STATIC_ROOT = os.path.join(BASE_DIR, 'static')
```

```
MEDIA_URL = '/media/'  
MEDIA_ROOT = os.path.join(BASE_DIR, 'media')
```

提示：在“settings.py”文件的末尾已经包含了第一句代码，我们只需要加上后面 3 句代码就可以了。

三、创建配置文件

执行命令：sudo vi /etc/apache2/sites-available/MyWeb.conf

在 vi 编辑器中写入配置文件内容。

配置文件内容：

```
<VirtualHost *:80>  
    ServerName qqtbb.com # 服务器名称  
    ServerAlias www.qqtbb.com # 服务器别名  
    ServerAdmin 4907442@qq.com # 管理员邮箱  
    WSGIScriptAlias / /var/www/MyWeb/MyWeb/wsgi.py # WSGI 文件路径  
    Alias /media/ /var/www/MyWeb/media/ # 媒体文件目录名称与路径  
    Alias /static/ /var/www/MyWeb/static/ # 静态文件目录名称与路径  
  
    <Directory /var/www/MyWeb/media> # 媒体文件目录  
        Require all granted # 目录权限：准许所有请求  
    </Directory>  
  
    <Directory /var/www/MyWeb/static> # 静态文件目录  
        Require all granted  
    </Directory>  
  
    <Directory /var/www/MyWeb/MyWeb/> # WAGI 文件目录  
        <Files wsgi.py> # 文件权限设置  
            Require all granted # 准许所有请求  
        </Files>  
    </Directory>  
    ErrorLog ${APACHE_LOG_DIR}/error-MyWeb.log # 错误日志存放位置与名称  
    CustomLog ${APACHE_LOG_DIR}/access-MyWeb.log combined # 指令设置日志文件存
```

放位置、名称与格式

```
</VirtualHost>
```

提示：使用时请删掉注释内容。

完成编辑后，按“ESC”键退出编辑模式，并输入“:wq”后按回车键保存退出。

接下来，我们让配置文件生效。

在目录/var/www/中默认有一个html文件夹，里面有一个文件“index.html”。

此时，我们通过域名或者ip地址访问站点，能够看到打开的是“Apache2 Ubuntu Default Page”这个页面，也就是“index.html”的文件内容。

那么，如何让我们的项目能够访问呢？

在/etc/apache2/sites-available/文件夹中包含了一个名为“000-default.conf”的配置文件，就是这个文件在起作用，所以访问时打开了“html”文件夹中的页面。

我们需要让我们刚刚添加的配置文件生效，并且禁用默认的配置文。

执行命令：`sudo service apache2 reload`
`sudo a2dissite 000-default.conf && sudo a2ensite MyWeb.conf`
`sudo service apache2 restart`

执行完上述命令，我们新建的配置文件就生效了。

这个时候，你会掉进一个大坑！

第20个坑：此时会报“Job for apache2.service failed because the control process exited with error code. See “systemctl status apache2.service” and “journalctl -xe” for details.”的错误，这个错误基本上搜索不到正确的解决方案。产生这个错误首先可能是配置文件内容有问题，导致Apache加载后不能正常工作；再有，就是没有添加一个必须的文件“wsgi.load”，不知道为什么，网上的教程都没提到这个文件的添。

接下来，我们添加“wsgi.load”文件。

执行命令：`sudo vi /etc/apache2/mods-available/wsgi.load`

在vi编辑器中，我们添加如下内容：`LoadModule wsgi_module`
`/usr/lib/apache2/modules/mod_wsgi.so`

添加完毕之后，我们保存退出。

然后，进入图形界面中，打开/etc/apache2/mods-available/文件夹，在“wsgi.load”文件上点击鼠标右键，发送到【Send To】桌面快捷方式【Desktop（Create Link）】。

回到命令行，我们把这个快捷方式放入/etc/apache2/mods-enabled/文件夹中。

执行命令：`cd /home/ubuntu/Desktop`
`sudo mv wsgi.load /etc/apache2/mods-enabled/wsgi.load`

完成快捷方式的创建，我们就可以再次尝试启动服务器。

执行命令：`sudo a2dissite 000-default.conf && sudo a2ensite MyWeb.conf`
`sudo service apache2 restart`

提示：如果添加了“wsgi.load”文件和快捷方式之后，重启 Apache 还报同样错误的话，那就需要检查配置文件是否内容有问题。

到这里，通过域名或者 ip 地址，就能够访问我们的项目了。



不过这个时候，如果访问后台的话，会发现后台没有样式。

这是因为样式文件都在 Django 的目录中，没有放到我们项目的目录中。

我们需要进行以下处理。

在项目目录下执行 python 命令：`python manage.py collectstatic`

这个命令能够把 Django 目录中的样式文件复制到“static”文件夹中。

然后，打开“urls.py”文件，添加新的 URL 分发配置，让请求静态文件的 URL 能够被正确处理。

示例代码：

```
from django.views.static import serve
```

```
re_path(r'^static/(?P<path>.*)$', serve, {'document_root': settings.STATIC_ROOT})
```

这样处理之后，Django 的后台界面就又变得很漂亮了。

四、使用虚拟环境

如果使用虚拟环境的话，我们需要的只是创建虚拟环境，并创建虚拟环境的配置文件，启用就可以了。

因为在实例环境中，我们已经为 Python3.6 安装了 Django2 等必须的库，创建虚拟环境时，我们只需要将这些内容原原本本的复制一份就可以了。

1、安装虚拟环境

进入“/usr/local/lib/python3.6/site-packages/”目录，执行 pip 命令安装“virtualenv”库。

执行命令：`cd /usr/local/lib/python3.6/site-packages/sudo python3.6 pip install virtualenv`

安装完成之后，我们进入项目目录，创建虚拟环境。

例如，创建一个名为“venv”的虚拟环境。

执行命令：`cd /var/www/MyWeb/sudo virtualenv -p /usr/local/bin/python3.6 venv`

如果只想要一个未安装第三方库的虚拟环境，需要在创建命令后方加上参数“-no-site-packages”。

进入虚拟环境需要使用“source”命令，例如进入刚刚创建的“venv”。

执行命令：`cd /var/www/MyWeb/venv/binsource activate`

此时的命令行会变成“(venv)”开头。如果要退出虚拟环境，直接在命令行输入“deactivate”即可。另外，在虚拟环境中只有一个 Python3.6，所以在虚拟环境的命令行下，可以直接使用“python”和“pip”命令。

第 21 个坑：虚拟环境的 Django 不能正常使用，需要重新安装！而且注意，虚拟环境安装第三方库时，需要先获取“root”权限，再进入虚拟环境。

执行命令：`sudo -isource /var/www/MyWeb/venv/bin/activatepip install pytzcd /home/ubuntu/Downloads/Django-2.0.3python setup.py install`

照例测试一下：

```
(venv) root@VM-90-36-ubuntu:/home/ubuntu/Downloads/Django-2.0.3# python
Python 3.6.5rc1 (default, Mar 21 2018, 18:52:18)
[GCC 5.4.0 20160609] on linux
Type "help", "copyright", "credits" or "license" for more information.
>>> import django
>>> import sqlite3
>>> █
```

2、创建虚拟环境的配置文件

配置文件内容：

```
<VirtualHost *:80>
    ServerName qqtbb.com
    ServerAlias www.qqtbb.com
    ServerAdmin 4907442@qq.com
    DocumentRoot /var/www/MyWeb/MyWeb # 设置站点根目录
    WSGIScriptAlias / /var/www/MyWeb/MyWeb/wsgi.py
    WSGIDaemonProcess www.qqtbb.com python-
path=/var/www/MyWeb/MyWeb:/var/www/MyWeb/venv/lib/python3.6/site-packages
    # 设置守护进程，指定 Python 路径。让虚拟环境的 python 解释器运行在单独的进程之
中，互不干扰。
    WSGIProcessGroup www.qqtbb.com # 设置 WSGI 进程组
    Alias /media/ /var/www/MyWeb/media/
    Alias /static/ /var/www/MyWeb/static/
    <Directory /var/www/MyWeb/media>
        Require all granted
    </Directory>
    <Directory /var/www/MyWeb/static>
        Require all granted
    </Directory>
    <Directory /var/www/MyWeb/MyWeb/>
        <Files wsgi.py>
            Require all granted
        </Files>
    </Directory>
    ErrorLog ${APACHE_LOG_DIR}/error-MyWeb.log
    CustomLog ${APACHE_LOG_DIR}/access-MyWeb.log combined
</VirtualHost>
```

3、激活配置文件并重启 Apache

执行命令：`sudo a2dissite MyWeb.conf && sudo a2ensite MyWeb_venv.conf`
`sudo service apache2 restart`

提示：可以修改一下模板文件，可以明显看出启动的是虚拟环境的站点。



五、权限设置

大家之前使用了一句命令来改变文件夹的权限：`sudo chmod 777 MyWeb`

那么，“777”是什么意思呢？

这个数字，每一位都代表一类用户。

- 第一位：拥有者
- 第二位：组
- 第三位：其它

而数字的大小，代表着权限组合的不同。

权限也分三种：

- R：读取权限
- W：写入权限
- X：执行权限

数字 0-7 表示 8 种不同的级别。

权限	读取 (R)	写入 (W)	执行 (X)
0			
1			✓
2		✓	
3		✓	✓
4	✓		
5	✓		✓
6	✓	✓	
7	✓	✓	✓

一般来说，网站项目只需要用到“777”、“755”、“644”这三种权限，目录权限使用“755”，文件权限使用“644”。

接下来，为我们项目中的所有文件设置“644”权限。

执行命令：`cd /var/www/sudo chmod -R 644 MyWeb`

提示：参数“-R”是指处理指定目录以及其子目录下的所有文件。

然后，再将所有的目录设置为“755”权限。

`sudo find MyWeb -type d | xargs chmod 755`

提示：参数“-type d”是指目录类型。

不过，当我们这么设置权限之后，会发现项目中对数据库的写入操作出现问题，无法使用类似商品添加删除的功能。

这是因为我们刚才的操作把数据库文件“db.sqlite3”的写入权限给取消了。

所以，我们还需要单独设置数据库文件的权限。

这里我们可以使用服务器中默认的用户“www-data”，具有比较好的安全性。

我们先把项目文件夹和“db.sqlite3”文件加入到“www-data”组。

执行命令：`cd /var/www/sudo chgrp www-data MyWeb`
`sudo chgrp www-data MyWeb/db.sqlite3`

然后，赋予项目文件夹和“db.sqlite3”文件写入权限。

执行命令：`sudo chmod g+w MyWeb`
`sudo chmod g+w MyWeb/db.sqlite3`

通过这样的调整项目中对数据库的写入操作就正常了。

到这里，我们就完成了使用 Ubuntu16.04 系统和 Apache2.4 服务器进行 Django2.0 项目的部署。



转载请注明：魔力 Python » Django2 : Web 项目开发入门笔记 (14)

原文链接：<http://www.opython.com/1027.html>

Django2 : Web 项目开发入门笔记 (15)

这一篇教程，我们一起来了解如何将 Django2 的 Web 项目部署到基于 IIS 的 Web 服务器。

很多学习 Python 的用户，都是在使用 Windows 系统，当使用 Django 完成一个 Web 项目，想在本机进行测试，又不想额外安装 Web 服务器的话，可以考虑使用 Windows 自带的 IIS 进行部署。

IIS (Internet Information Services : 互联网信息服务) 是 Web 服务组件，其中包括 Web 服务器、FTP 服务器、NNTP 服务器和 SMTP 服务器，分别用于网页浏览、文件传输、新闻服务和邮件发送等方面。

如果想使用 IIS 部署 Django 项目，需要先在 Window 系统中添加 IIS 功能并开启 CGI 应用程序支持，大家可以参考之前发布的教程《练习项目 11：在线编辑文件（上）》中的第一部分内容，进行添加与设置。

提示：教程中是以 Windows7 系统下的 IIS7.5 为例，不同版本的 Windows 或 IIS 的添加与设置都比较相近。

但是，仅有这些是不行的。

因为 IIS 支持 FastCGI，不支持 WSGI，而 Django2 全面使用 WSGI，不再使用 FastCGI。

MMP，这是有多大仇？

所以，要想让两个仇家能合作，需要有个中间人。

这个中间人是 Python 的第三方库 wfastcgi。

了解了以上内容，我们就可以开始部署工作了。

提示：本篇教程以 Window7 为例。

一、安装启用 wfastcgi

在系统目录中找到 CMD 命令行终端，路径为：C:\Windows\system32\cmd.exe

也可以在桌面左下角，点击开始按钮，在【搜索程序和文件】的输入框中输入 CMD 进行查找。

但是，不要急于打开，在程序图标上点击鼠标右键选择【以管理员身份运行】；否则，启用 wfastcgi 会失败。

执行命令：

```
pip install wfastcgi
```

```
wfastcgi-enable
```

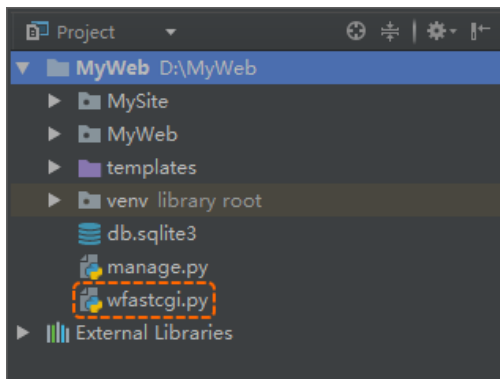
启用成功后会显示：

已经在配置提交路径“MACHINE/WEBROOT/APPHOST”向“MACHINE/WEBROOT/APPHOST”的“system.webServer/fastCgi”节应用了配置更

改“C:\Users\Administrator\AppData\Local\Programs\Python\Python36\python.exe|C:\Users\Administrator\AppData\Local\Programs\Python\Python36\lib\site-packages\wfastcgi-3.0.0-py3.6.egg\wfastcgi.py” can now be used as a FastCGI script processor

注意上面的标红的路径，是由 Python 解释器的路径和“|”以及“wfastcgi.py”文件路径组成，后面会用得到。

如果需要部署多个项目，大家可以把“wfastcgi.py”文件从上面的路径中复制到 Django 项目的根目录下。



另外，如果想卸载 wfastcgi，需要先禁用再卸载。

禁用命令：wfastcgi-disable

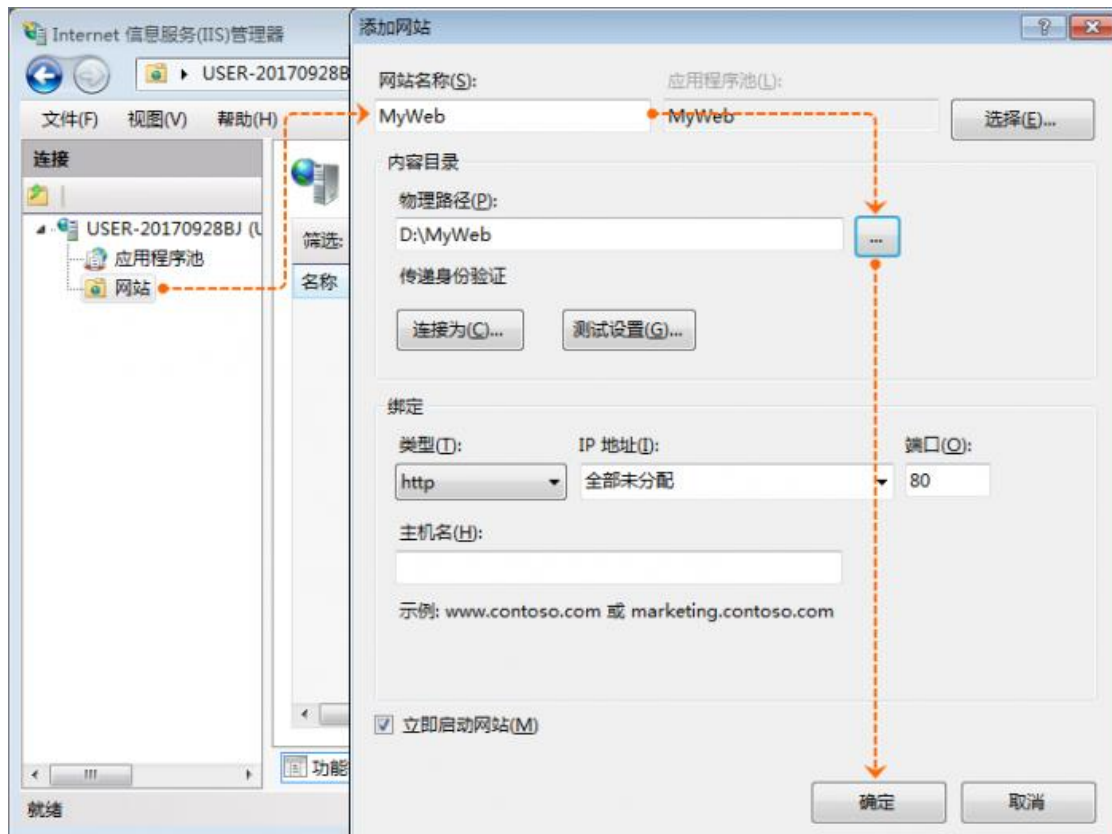
卸载命令：pip uninstall wfastcgi

二、创建网站

打开 Internet 信息服务(IIS)管理器，【控制面板】-【管理工具】-【Internet 信息服务(IIS)管理器】。

在窗口左侧的【网站】图标上点击鼠标右键，选择【添加网站】。

站点名称可以和项目名称保持一致，然后物理路径选择项目目录，绑定设置使用默认设置，主机名留空。



三、添加处理程序映射

点中我们新建的网站。

注意：不要点中左侧列表的根目录，根目录中的处理程序映射是对所有网站有效的。

窗口中部找到【处理程序映射】，双击打开之后，在窗口右侧找到【添加模块映射】，点击打开后依次进行设置：

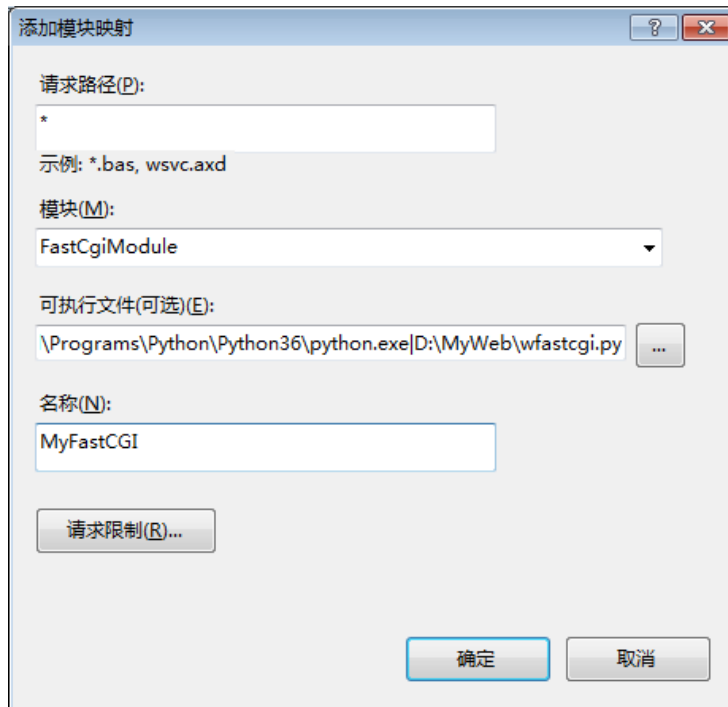
- 请求路径：*
- 模块：FastCgiModule

- 可执行文件：Python 解释器的路径|wfastcgi.py 文件路径
- 名称：MyFastCGI（自定义名称）

提示：可执行文件填写的内容就是之前标红的路径，如果将 wfastcgi.py 文件复制到了项目目录中的话，可以使用项目中的文件路径。

例如：

C:\Users\Administrator\AppData\Local\Programs\Python\Python36\python.exe|D:\MyWeb\wfastcgi.py



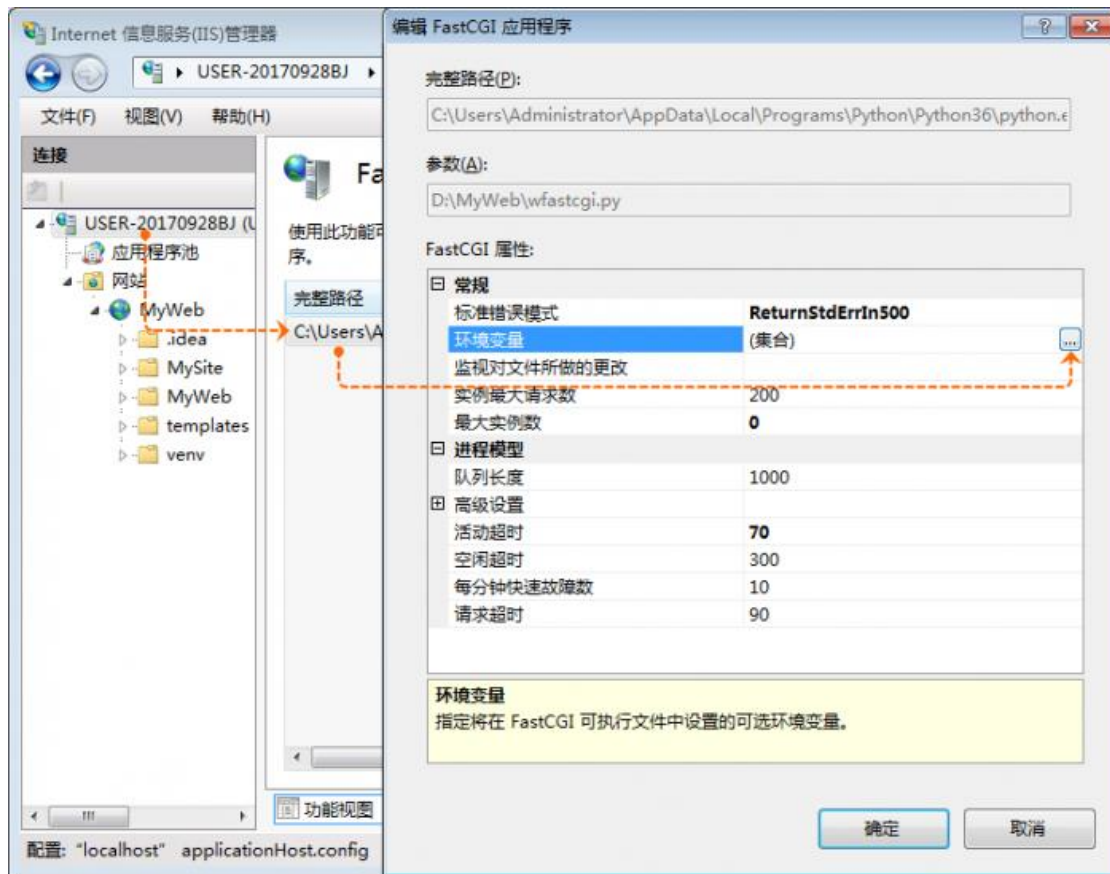
另外，还要点击上图中的【请求限制】按钮，打开的窗口中取消【仅当请求映射至以下内容时才调用处理程序】的勾选。

提示：IIS7.5 默认是不够选的，但是有些 IIS 版本会默认勾选，导致网站无法正常访问，例如：IIS10。

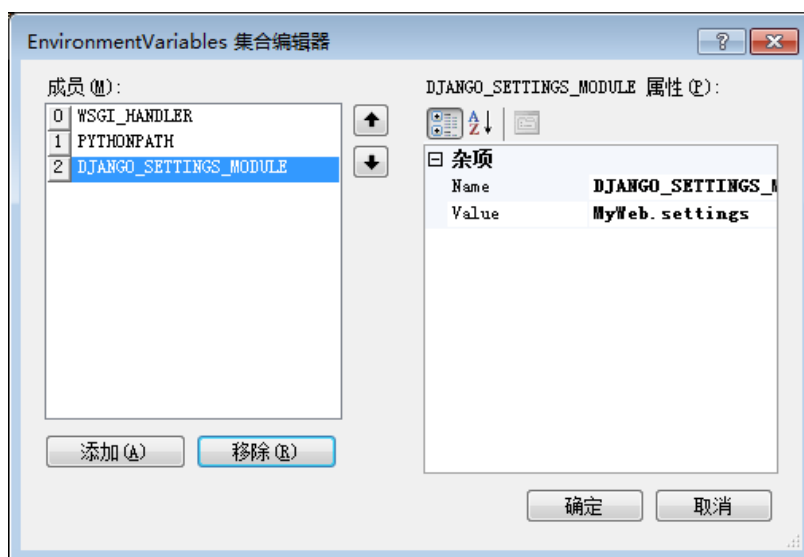
做完以上设置，点击【确定】，此时会弹出创建 FastCGI 应用程序的提示，点击【是】，完成创建。

四、添加应用程序

Internet 信息服务(IIS)管理器左侧点中根目录，窗口中部找到【FastCGI 设置】，双击进入。在打开的窗口中，双击我们创建的记录打开【编辑 FastCGI 应用程序】的窗口。



FastCGI 属性列表中，找到【环境变量】，点击后方的【...】按钮进入【EnvironmentVariables 集合编辑器】，左侧成员列表下方点击添加按钮，添加 3 个变量。



在 Django 项目的“wsgi.py”文件中有一句代码：`from django.core.wsgi import get_wsgi_application`

导入的方法“get_wsgi_application”，它的返回值就是一个 WSGIHandler()。

Name: WSGI_HANDLER

Value: django.core.wsgi.get_wsgi_application()

第 2 个变量是项目目录，Python 代码文件的所在路径。

Name: PYTHONPATH

Value: D:\MyWeb

第 3 个变量是项目中“settings.py”文件的路径。

Name: DJANGO_SETTINGS_MODULE

Value: MyWeb.settings

点击确定按钮，保存退出。

网上的很多教程到这里就会说：当我们完成以上步骤，重新启动网站，就能够访问了。

事实上还不行，继续完成下面的设置。

五、修改项目的设置文件

打开项目中的“settings.py”文件，找到“ALLOWED_HOSTS”，进行修改。

示例代码：

```
ALLOWED_HOSTS = ['*'] # *表示不限制，也可以在列表中添加站点绑定的域名
```

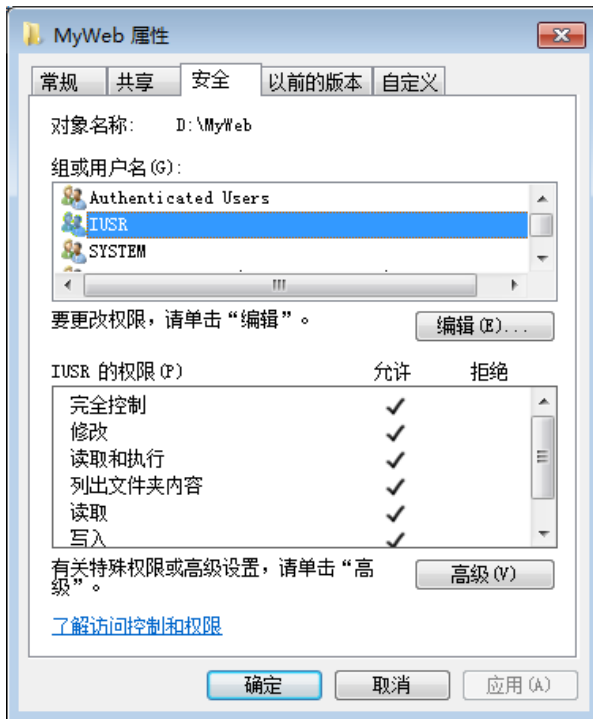
如果之前已经做过这项设置，此步骤可以略过。

六、添加目录权限

需要增加权限的目录包括 Python 安装目录和项目目录，缺一不可。

修改内容是将这些目录都添加指定用户组的控制权限。

这里，我们给指定用户组都赋予“完全控制”的权限，参考《练习项目 11：在线编辑文件（上）》中第二部分~第 6 点的内容。



提示：如果还不能访问，尝试将“IIS_IUSRS”用户组的权限也添加上。

到这里，就可以通过本机、局域网或者外网进行访问了。

打开浏览器，输入访问地址：

- 本机：127.0.0.1 或者 localhost
- 局域网：192.168.XX
- 外网：本机所在网络的外网 IP 地址

提示：如果本机通过路由器联网，需要路由器上做端口映射（转发），也可以通过启用 DMZ 来实现外网的访问。

问：如果局域网内可以访问，外网仍然无法访问，怎么办？

答：首先，找宽带服务商将 IP 地址转为公网 IP，作者是电信宽带，打完热线要求更改后不到 24 小时就可以访问了。

问：如果转成公网 IP 依旧无法访问本机服务器，怎么办？

答：可能存在很多原因，有可能是防火墙问题，也有可能是 IIS 问题等等，需要进行排查才能确定。Django 项目在 IIS 上部署，本来就是以测试为目的，我建议不要耗费太多精力在这个问题上，除非是真正在生产环境下需要如此进行部署。

最后，对于项目中的静态文件我们还需要进行相应的设置。七、处理静态文件 1、打开项目中的“settings.py”文件，找到“STATIC_URL”，进行修改。

示例代码：

```
STATIC_URL = 'static/' # 注意只有后面有"/"，否则会目录不正确。
```

2、打开项目中的“urls.py”文件，增加代码。

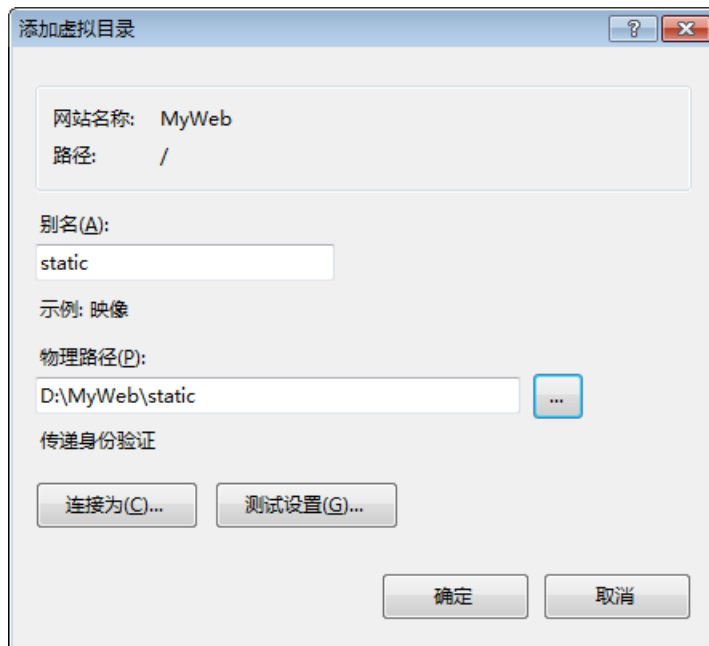
示例代码：

```
from django.conf import settings
from django.conf.urls.static import static
```

```
urlpatterns = [
    ...省略 URL 分发配置...
] + static(settings.STATIC_URL, document_root=settings.STATIC_URL)
```

3、在项目文件夹中添加静态文件的文件夹。例如：**static**

4、Internet 信息服务(IIS)管理器点中我们创建的网站，点击鼠标右键，选择添加虚拟目录，填写别名（例如：**static**），然后将这个虚拟目录指向静态文件目录的物理路径，点击确定保存。



到这里，我们就完成了 Django2 项目在 IIS 上的部署。

转载请注明：魔力 Python » Django2 : Web 项目开发入门笔记 (15)

原文链接：<http://www.opython.com/1043.html>

Django2 : Web 项目开发入门笔记 (16)

这一篇教程，我们一起来了解如何在 Ubuntu 系统中将 Django2 的 Web 项目部署到 Nginx 服务器。

首先，大家可以参考《Django2 : Web 项目开发入门笔记 (13)》先完成以下准备工作。

- 安装 Ubuntu 系统并更新 apt-get
- 安装 Python3.6 与依赖
- 安装 Django2.0.3
- 安装 Sqlite3 与依赖
- 安装 OpenSSL 等相关依赖

也就是说，除了 Apache2 和 mod_wsgi 之外全部都需要安装。

完成上述内容的安装之后，再将项目文件放入/var/www 目录中，我们就可以开始 Nginx 的安装了。

一、安装 Nginx 和依赖

执行命令：`sudo apt-get install nginx``sudo apt-get install libpcre3 libpcre3-dev`

二、安装 uwsgi

执行命令：`cd /usr/local/lib/python3.6/site-packages/``sudo python3.6 pip install uwsgi`

三、测试 uwsgi

先在/var/www 目录中创建一个测试文件。

执行命令：`sudo chmod 777 /var/www``cd /var/www``sudo vi mytest.py`

在 VI 编辑器中，我们输入测试文件的代码。

示例代码：

```
def application(env, start_response):
    start_response('200 OK', [('Content-Type','text/html')])
    return [b'UWSGI Test...']
```

代码输入完毕，按“ESC”键并键入“:wq”回车，保存测试文件后，进行测试。

执行命令：`sudo uwsgi --http :8888 --wsgi-file mytest.py`

通过本机浏览器访问“<http://127.0.0.1:8888>”或者“<http://localhost:8888>”进行测试，如果页面中显示“UWSGI Test...”字样，则说明测试成功，uwsgi 可以正常工作了。

如果，重复测试时，提示端口被占用，可以先将已启动的进程关闭。

先通过名称查询相关进程，然后通过“kill”命令将进程关闭。

执行命令：`sudo ps -ef|grep uwsgisudo kill -9 [端口号]`

也可以使用“killall”命令通过名称关闭全部相关进程。

`sudo killall -9 uwsgi`

如果命令无效，可以先获取“root”权限，再进行尝试。

四、配置 Nginx

首先，我们为项目添加配置文件。

执行命令：`sudo vi /etc/nginx/sites-available/MyWeb.conf`

在 VI 编辑器中写入配置文件内容。

```
server {
    listen 80; # 监听端口
    server_name www.qqtb.com; # 服务器名（多个域名用逗号分隔）
    charset utf-8; # 服务器字符集
    client_max_body_size 5M; # 上传文件最大限制
    location /media { # 媒体文件位置
        alias /var/www/MyWeb/media;
    }
    location /static { # 静态文件位置
        alias /var/www/MyWeb/static;
    }
    location / { # 为 uwsgi 协议提供支持（实现与 uWSGI 服务器通信）
        uwsgi_pass 127.0.0.1:8888; # 为 uWSGI 服务器设置监听地址（套接字或 sock 文件）
        # uwsgi_pass unix:///var/www/MyWeb/MyWeb.sock; # 使用 sock 文件
        include /etc/nginx/uwsgi_params; # 为 uwsgi 请求增加参数。
    }
}
```

内容输入完毕（使用时请先清除注释），按“ESC”键并键入“:wq”回车保存，然后将配置文件的快捷方式添加到/etc/nginx/sites-enabled/目录中，激活配置文件。

执行命令：`sudo ln -s /etc/nginx/sites-available/MyWeb.conf /etc/nginx/sites-enabled/MyWeb.conf`

然后，重启 Nginx 服务器或者让服务器重新加载配置文件。

重载配置命令：`sudo service nginx reload`

重启服务命令：`sudo service nginx restart`

如果重载配置文件或者重启服务器发生错误，需要检查配置文件是否有效。

语法检查命令：`sudo service nginx configtest`

注意，语法检查只是检查是否配置文件中语法全部正确，并不能排除配置项是否存在问题。

接下来，我们启动 uWSGI 服务器测试一下项目是否能够正常运行。

执行命令：`sudo uwsgi --http :8888 --chdir /var/www/MyWeb --module MyWeb.wsgi`

通过本机浏览器访问“`http://127.0.0.1:8888`”或者“`http://localhost:8888`”进行测试，如果项目页面访问正常，则说明配置文件

五、使用配置文件启动 uWSGI 服务器

uWSGI 服务器也可以通过配置文件进行启动，不但避免使用过长的命令，而且能增加更多的配置。执行命令：`cd /var/www/MyWeb``sudo vi uwsgi.ini` 配置文件内容：

[uwsgi]

```
socket = 127.0.0.1:8888 # 监听地址（套接字或 sock 文件）
# socket=/var/www/MyWeb/MyWeb.sock # 使用 sock 文件
chdir = /var/www/MyWeb/ # 项目根目录
wsgi-file = MyWeb/wsgi.py # wsgi 文件路径
processes = 3 # 开启的工作进程数
threads = 5 # 每个工作进程的线程数
chmod-socket = 664 # 客户端访问 MyWeb.sock 文件的权限
chown-socket = www-data # 客户端请求的所有者
pidfile= /var/www/MyWeb/MyWeb.pid # 保存进程文件的路径
vacuum = true # 服务器退出时自动删除 sock 文件和 pid 文件
```

内容输入完毕（使用时请先清除注释），按“ESC”键并键入“:wq”回车保存，然后就可以通过配置文件启动 uWSGI 服务器了。

启动命令：`sudo uwsgi --ini uwsgi.ini`

停止命令：`sudo uwsgi --stop MyWeb.pid`

重载配置：`uwsgi --reload uwsgi.ini`

六、使用 Supervisor 管理 uWSGI

Supervisor 是一个进程管理工具，能够方便我们对 uWSGI 服务器进行管理，并能够在 uWSGI 服务器意外关闭时，重新启动 uWSGI 服务器。

不过，目前 Supervisor 没有 Python3 的版本，我们需要使用 Ubuntu 系统自带的 Python2 进行安装。如果没有安装 pip，则需要先安装 pip。

执行命令：`sudo apt-get install python-pip`

接下来，安装 Supervisor。

执行命令：`sudo pip install supervisor`

安装完毕后，创建配置文件。

执行命令：`sudo echo_supervisord_conf > /etc/supervisord.conf`

打开配置文件，并添加配置。

执行命令：`sudo vi /etc/supervisord.conf`

配置内容：

```
[program:MyWeb] # 这里的“MyWeb”是启动/停止/重启项目时使用的名称
command=uwsgi --ini /var/www/MyWeb/uwsgi.ini # 启动 uWSGI 服务器的命令
directory=/var/www/MyWeb/ # 项目根目录
startsecs=10 # 进程持续运行多长时间认为启动成功
stopwaitsecs=10 # 向进程发出关闭信号后等待系统返回“SIGCHILD”信号的时间
stopasgroup=true # uwsgi.ini 中启动了多个进程时设置此项为 true，否则可以删除。
killasgroup=true # uwsgi.ini 中启动了多个进程时设置此项为 true，否则可以删除。
autostart=true # uWSGI 随 Supervisord 启动
autorestart=true # uWSGI 进程意外关闭后自动重启
```

内容输入完毕（使用时请先清除注释），按“ESC”键并键入“:wq”回车保存。

尝试启动 Supervisor。

执行命令：`sudo supervisord -c /etc/supervisord.conf`

此时，极有可能发生错误：`unix:///tmp/supervisor.sock no such file.`

这是因为配置文件中，默认将“supervisor.sock”文件存入/tmp/目录所导致的。

再次打开配置文件并修改以下内容：

- /tmp/supervisor.sock 修改为/var/run/supervisor.sock（有两处需要修改）
- /tmp/supervisord.log 修改为/var/log/supervisor.log
- /tmp/supervisord.pid 修改为/var/run/supervisor.pid

接下来，在修改后的目录中创建“supervisor.sock”文件，并给予读写权限。

执行命令：`sudo touch /var/run/supervisor.sock`
`sudo chmod 777 /var/run/supervisor.sock`

再次启动 Supervisor，又会发生错误：Unlinking stale socket /var/run/supervisor.sock

这次的错误是因为缺少依赖。

执行命令：`sudo apt-get install aptitude`
`sudo aptitude install python-meld3`

安装完依赖，继续尝试启动 Supervisor，此时还可能产生错误：Error: Another program is already listening on a port that one of our HTTP servers is configured to use. Shut this program down first before starting supervisord.

这是因为已经有“supervisor.sock”文件被链接（上一次启动 Supervisor 造成的）。

我们需要先查找哪个“supervisor.sock”文件被链接了。

执行命令：`sudo find / -name supervisor.sock`

结果中会显示链接的文件，例如：`/run/supervisor.sock`

然后，通过“unlink”命令取消查询到的链接。

执行命令：`sudo unlink /被链接文件的所在路径/supervisor.sock`

到这里，我们应该就能够正常启动 Supervisor 对 uWSGI 进行管理了。

启动项目命令：`sudo supervisorctl -c /etc/supervisord.conf start [配置文件中的项目名称]`

重启项目命令：`sudo supervisorctl -c /etc/supervisord.conf restart [配置文件中的项目名称]`

停止项目命令：`sudo supervisorctl -c /etc/supervisord.conf stop [配置文件中的项目名称]`

控制所有项目：`sudo supervisorctl -c /etc/supervisord.conf <start/restart/stop> all`



转载请注明：魔力 Python » Django2 : Web 项目开发入门笔记 (16)

原文链接：<http://www.opython.com/1069.html>

Django2 : Web 项目开发入门笔记 (17)

这一篇教程，我们一起来了解 Django 的表单 (Forms)。

这篇教程参考了官方文档，内容比较详细。

一、HTML 表单

首先，我们先更多的了解一些 HTML 中表单的相关知识。

在我们之前所接触的内容中，已经使用过 HTML 表单。

一个 HTML 表单包含一对<form>标签，并在这对标签中包含一个或多个表单元元素标签<input>以及其它可能需要的元素标签。

根据 Django 的官方文档描述，一个表单需要指定两件事：

- 什么位置：与用户输入相对应的数据返回到哪一个 URL 路径；
- 什么方法：采用哪一种 HTTP 方法返回的数据。

提示：这里的返回我理解为从客户端返回到服务器。

实际上就是给出表单的提交动作 (action 属性) 和提交的方法 (method 属性)。

1、action

action 的属性值可以是一个文件，例如 CGI 编程中属性值是一个 CGI 脚本文件的名称)；

它也可以是一个路径，例如《Django2 : Web 项目开发入门笔记 (8)》中我们就是这样使用。

2、method

method 的属性值可以是“GET”或者“POST”。

如果大家够细心的话，应该看出了这两种提交方法的区别。

简单来说，“GET”方法提交的表单数据会以参数形式呈现在浏览器的 URL 中,例如：

<https://www.baidu.com/s?wd=opython&nojc=1>；

而“POST”方法则看不到提交的数据内容。

看上去“POST”比“GET”更加安全 (在 HTTPS 中同样安全)。

那为什么不用“POST”而丢弃“GET”呢？

实际上，他们各有各的特点，能够应用于不同的目的。

“Post”通常会应用于：

- 改变系统状态：对数据库进行增、删、改的操作；
- 保护加密数据：类似注册、登录、支付等需要加密的表单数据，要避免将数据暴露在 URL 或者服务器日志中；
- 保护系统权限：避免攻击者模仿表单内容获取敏感信息。

“GET”通常会应用于：

- 数据查询搜索：查询结果能够比较方便的存为书签，进行共享或者再次提交。

另外，“GET”适合少量数据提交，不适合提交大量数据，类似图片的二进制数据。

二、Django 对于表单的作用

Django 能够将大部分表单相关的工作简单化以及自动化，并且比程序员自己写的代码有更高的安全性。

它承担三部分工作：

- 准备和重组数据准备渲染
- 创建 HTML 表单的数据
- 接收和处理从客户端提交的表单和数据

这些都不再需要自己写代码去完成。

三、Django 中的表单

在一个 Web 应用程序的上下文中，“form”并不单指 HTML 的<form>，也可能是 Django 的 Form 类的实例，或是提交时返回的结构化数据；在端对端工作中还可能是这些部分的集合。

Django 中表单的核心组件是 Forms 类。

和一个 Django 的模型描述一个对象的逻辑结构、行为以及各部分的表现方式类似，Forms 类描述了一个表单并决定它如何工作与呈现。

例如，一个模型类的字段映射到一个数据表的字段，一个表单类的字段映射到一个 HTML 表单的<input>元素。（一个 ModelForm 类可以通过 Form 映射一个模型类中的全部字段到 HTML 表单中相对应的 <input> 元素；Django admin 即是基于此实现。）

一个表单字段的类负责管理表单数据和表单被提交时的验证；`DateField` 类和 `FileField` 类处理不同种类的数据和执行不同的任务。

一个表单字段表示用户在浏览器中的一个 **HTML** 部件（用户界面的一部分），每一个字段类型都有一个对应的默认 **Widget**（部件）类，这些部件类可以被重写。

在我们所学过的内容中，我们知道在 **Django** 中将一个数据对象呈现的步骤如下：

- 在视图中获取数据对象（例如，从数据库中获取数据）；
- 将数据对象与模板整合；
- 使用模板变量将数据对象添加到 **HTML** 标记中。

在模板中渲染一个表单也是这样的形式，但是有一些重要的区别。

在模板中渲染表单可以是一个没有包含任何数据的模型实例，并且在模板中很少执行任何有用的操作；从另外的角度说，我们期望渲染一个未填充的表单，由用户去填充它。

所以，当我们在视图中进行模型实例化时，我们通常需要从数据库中进行检索；而当我们处理一个表单时，我们只是在视图中进行实例化。

当我们实例化一个表单时，可以选择留空或者预置内容，例如以下情形（预置内容）：

- 一个已保存的模型实例；（例如编辑后台信息时）
- 整理其它来源的数据；
- 从前一个 **HTML** 表单提交得到的数据。

四、使用 **Django** 构建表单

假设我们在一个页面获取用户输入的姓名，我们就需要一个表单。

示例代码：

```
<!DOCTYPE html>
<html lang="zh-cn">
<head>
  <meta charset="UTF-8">
  <title>表单示例</title>
</head>
<body>

  <form action="/user_name/" method="post">

    <label for="user_name">姓名：</label>

    <input id="user_name" type="text" name="user_name" value="{{ current_name }}">
```

```
<input type="submit" value="OK">

</form>

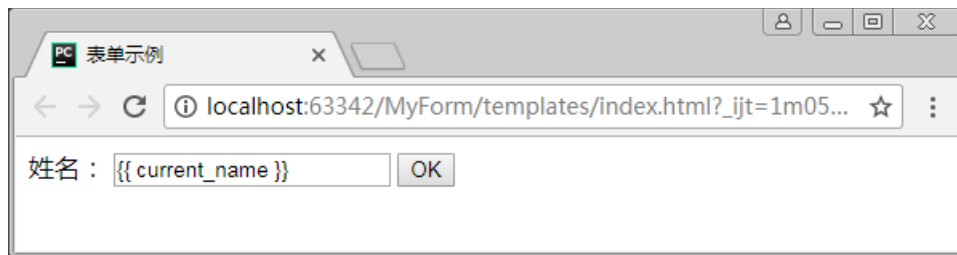
</body>
</html>
```

提示：`<table>`标签的“for”属性用于规定 label 与哪个表单元素绑定，点击`<table>`标签时，光标则会进入与其绑定的表单元素。

上方这段代码中，红色的部分就是表单内容，他告诉浏览器要使用“post”方法，将表单中的数据发送至“/user-name/”这个 URL 中。

如果模板上下文包含“current_name”变量,将会预填充“user_name”这个表单元素的文字。

现在，如果将模板文件用浏览器打开的话，会呈现下方的内容。



接下来，我们可以通过一个视图函数整合这个模板，并能够根据需求提供预填充的数据。

然后，在提交表单的时候，发送到服务器的 post 请求会包含表单中的数据。

当然，我们还需要一个与 URL“/user-name/”对应的视图函数，根据表单中的数据进行进一步处理。

以上，是我们之前的做法。

这是一个比较简单的表单，实际上，一个表单往往会包含更多的字段，可能是几个、几十个或者上百个。

而往往这些字段中的大部分字段都需要预填充，我们希望用户在完成操作前可以进行多次的编辑，进行提交（类似草稿）。

另外，我们有些时候需要在客户端（浏览器）中完成字段的验证，而不是提交到服务器之后再验证，这样能够为服务器缓解很多压力。

还有，我们可能需要一些更复杂的字段，允许用户进行像选择日历这样的操作。

类似以上这些需求，使用 Django 能够帮助我们完成大部分的工作。

接下来，我们来看使用 Django 构建一个表单。

1、定义表单类

我们在定义表单类时，都应该继承自“`django.forms.Form`”或者“`django.forms.ModelForm`”。

提示：“Form”类和“ModelForm”类都是继承自私有的“BaseForm”类。

在项目文件夹中，新建一个名为“`forms.py`”的文件。

示例代码：

```
from django import forms
```

```
class NameForm(forms.Form):
```

```
    user_name = forms.CharField(label='姓名', max_length=20)
```

在上方代码中，我们定义了一个名为“NameForm”的表单类，继承自 Django 的“Form”类。

在这个类中，我们定义了一个“user_name”的字符字段（CharField），并对这个字段进行了一些设置。

- `label`：与字段所在元素捆绑的<label>标签的文字；
- `max_length`：对字段字符数量的限制，浏览器会自动限制输入字符的数量，并且提交表单时 Django 会自动进行长度的验证。

关于“CharField”字段类的参数不仅仅这两种，大家可以打开“`fields.py`”文件查看这个字段类，路径是：`\Lib\site-packages\django\forms\fields.py`

每一个表单类的实例都有一个“`is_valid()`”方法，负责对表单中所有的字段进行验证，如果所有字段都通过验证，该方法的返回值为“True”，并且会将所有的表单字段数据存储在它的“`cleaned_data`”属性中。

我们刚刚定义的 Django 表单，呈现在浏览器中的时候，就像下方这样的 HTML 代码（实际上还会多出一些<tr>、<th>和<td>这样的表格标签）。

示例代码：

```
<label for="id_user_name">姓名:</label>
```

```
<input type="text" name="user_name" maxlength="20" required id="id_user_name"/>
```

注意：表单类所生成的 HTML 代码不包含<form>标签和提交（submit）按钮，我们需要在模板文件中自己编写。

2、定义视图函数

表单类需要在视图中实例化，并通过视图函数将空表单实例或者带有数据的表单实例与模板整合进行呈现。

示例代码：

```
from django.shortcuts import render
from django.http import HttpResponseRedirect # 导入重定向类
from .forms import NameForm # 导入表单类

def get_name(request):
    if request.method == 'POST': # 如果为“post”方法提交
        form = NameForm(request.POST) # 使用提交的数据实例化表单类
        if form.is_valid(): # 如果表单数据全部有效
            return HttpResponseRedirect('/thanks/') # 重定向到新的 URL
        else: # 如果是“get”方法提交，例如：http://127.0.0.1:8888/user_name/?user_name=aaa
            form = NameForm() # 实例化一个空表单
        return render(request, 'name.html', {'form': form}) # 将表单数据整合到模板并呈现

def thanks(request): # 表单提交有效时 URL 对应的视图
    return render(request, 'thanks.html')
```

3、定义模板内容

在模板中整合表单内容，和在模板中整合模型的数据非常相似。

只需要把表单实例通过名称添加到模板中相应的位置即可。

示例代码：

```
<!DOCTYPE html>
<html lang="zh-cn">
<head>
    <meta charset="UTF-8">
    <title>表单示例</title>
</head>
<body>
<form action="/user-name/" method="post">

    {% csrf_token %}

    {{ form }}

    <input type="submit" value="OK">
```

```
</form>
</body>
</html>
```

上方代码中，“{{ form }}”就是整合表单实例，而“{% csrf_token %}”是一个安全防御的标记，用来避免跨站伪造请求的攻击。

提示：如果只想呈现表单中的某一元素，可以通过类似“{{ form.user_name }}"的方式来获取这个元素。

4、配置 URL 分发

示例代码：

```
from django.contrib import admin
from django.urls import path
from FormTest import views as form_views

urlpatterns = [
    path('user_name/', form_views.get_name),
    path('thanks/', form_views.thanks),
    path('admin/', admin.site.urls),
]
```

完成以上的步骤，我们就可以启动开发服务器，通过“http://127.0.0.1:端口号/user_name/”或者“http://localhost:端口号/user_name/”进行测试了。

另外，大家也可以通过“http://127.0.0.1:8888/user_name/?user_name=aaa”这样的方式模拟“get”方法提交数据，结果是我们在视图中定义的空表单。

本节练习源代码：【[点此下载](#)】

转载请注明：魔力 Python » Django2：Web 项目开发入门笔记（17）

原文链接：<http://www.opython.com/1092.html>

Django2：Web 项目开发入门笔记（18）

这一篇教程，我们继续了解 Django 的表单。

在上一篇教程中，我们了解了 Django 表单的基本使用过程。

接下来，我们继续深入了解 Django 表单的其他功能，以及更多底层的内容。

一、使用更多的字段

我们可以在定义表单类时使用多个字段。

不过要注意，如果表单类中包含 `URLField`、`EmailField` 或任何整数类型的字段，Django 会使用 `HTML5` 的输入框类型。

默认情况下，浏览器会自动进行这些字段的验证，甚至比 Django 的验证更加严格。

如果要禁用浏览器的自动验证，需要在 `<form>` 标签上设置 `novalidate` 属性值为 `"novalidate"`，整个表单将会禁用验证功能。

示例代码：

```
<form action="/email/" method="post" novalidate="novalidate">
```

如果只想取消某个字段的验证，可以为字段指定为其他标签，比如 `"TextInput"`。

示例代码：

```
email = forms.EmailField(label='邮箱', widget=forms.TextInput)
```

接下来，我们来创建一个带有多个字段的表单类。

例如，一个发送邮件的表单，会包含标题、消息内容、发件人以及是否抄送给自己等等。

示例代码：

```
class ContactForm(forms.Form):
    subject = forms.CharField(label='主题', label_suffix='：',
                              widget=forms.TextInput(attrs={'style': 'width:440px', 'maxlength': '100'}))
    # 定义标题字段，指定文本框宽度并限定字符数量。
    message = forms.CharField(label='消息', label_suffix='：',
                              widget=forms.Textarea(attrs={'cols': '60', 'rows': '10'}))
    # 定义消息字段，指定使用文本域（多行文本框）标签，并设置标签的属性。
    sender = forms.EmailField(label='发送人', help_text='请输入正确的邮箱地址！',
                              label_suffix='：')
    # 定义发件人字段，指定为邮箱地址类型的字段。
    cc_myself = forms.BooleanField(label='是否抄送自己', label_suffix='：', required=False)
    # 定义抄送字段，指定为布尔类型（真假值）的字段。
```

在上方代码中，我们能够看到两个 `"CharField"` 类型的字段。

不同的是，第一个 `"CharField"` 类型的字段 `"subject"` 在网页上产生的 `HTML` 代码如下所示。

示例代码：

```
<label for="id_subject">主题 : </label>
<input type="text" name="subject" style="width:440px" maxlength="100" required
id="id_subject" />
```

而第二个“CharField”类型的字段“subject”在网页上产生的 HTML 代码则会另外一个是 `<textarea>` 标签。

示例代码：

```
<label for="id_message">消息 : </label>
<textarea name="message" cols="60" rows="10" required id="id_message"></textarea>
```

另外，在代码中我们看到了一些部件（widget）的参数。

其中，“attrs”参数是以字典的形式设置 HTML 标签的属性。

而“label_suffix”参数，是指 `<label>` 标签上文字后缀，默认是英文半角的“:”，如果想改成其他则需要设置这个参数的值，例如，这里改成了中文的“：”。

最后，大家应该也看到了“EmailField”带有一个“help_text”参数，这是表单元后方的文本提示信息。

二、表单数据整合到模板的多种方式

我们在视图文件“views.py”中实例化上方新建等等表单，并通过视图函数将它和模板进行整合。

示例代码：

```
from .forms import ContactForm

def write_email(request):
    form = ContactForm()
    return render(request, 'email.html', {'form':form})
```

我们在视图函数中将表单对象传递给“email.html”这个模板。

在“email”这个模板中，我们需要对表单对象进行处理。

如果只是通过“`{{ form }}`”的方式将表单添加到模板的话，所有的表单元会平铺在页面中，达不到我们的要求。

所以，我们需要进行一些处理。

1、使用表单输出选项

Django 给我们提供的表单输出选项有 3 种：

- `{{ form.as_table }}` : 以表格形式加载表单元素
-
- `{{ form.as_ul }}` : 以列表形式加载表单元素

以段落形式为例。

示例代码：

```
<!DOCTYPE html>
<html lang="zh-cn">
<head>
  <meta charset="UTF-8">
  <title>撰写邮件</title>
</head>
<body>
<form action="/send_email/" method="post">
```

```
  {{ form.as_p }}
```

```
    <input type="submit" value="发送">
</form>
</body>
</html>
```

采用这种形式，最终所呈现的页面，源代码中包含如下内容。

示例代码（页面部分源代码）：

```
<p><label for="id_subject">主题：</label> <input type="text" name="subject"
style="width:440px" maxlength="100" required id="id_subject" /></p>
<p><label for="id_message">消息：</label> <textarea name="message" cols="60"
rows="10" required id="id_message">
</textarea></p>
<p><label for="id_sender">发送人：</label> <input type="email" name="sender" required
id="id_sender" /> <span class="helptext">请输入正确的邮箱地址！</span></p>
<p><label for="id_cc_myself">是否抄送自己：</label> <input type="checkbox"
name="cc_myself" id="id_cc_myself" /></p>
```

显然，采用段落这种输出形式时，Django 自动将每个表单元素添加到了成对 `<p>` 标签中，表格和列表也类似于这种处理方式。

2、自定义输出表单元素

Django 给我们提供的输出选项，在很多时候并不能满足我们的要求。

所以，我们可以自己定义表单的加载。

之前已经提到过，通过模板标记“`{{ form.字段名称 }}`”可以获取表单元素。

那么，之前的表单内容，我们就可以通过这种模板标记来完成自定义输出。

示例代码：

```
<form action="/send_email/" method="post">

    <label for="{{ form.subject.id_for_label }}">标题：</label>

    {{ form.subject }}

    <input type="submit" value="发送">

</form>
```

采用这种形式，最终所呈现的页面，源代码中包含如下内容。

示例代码（页面部分源代码）：

```
<label for="id_subject">标题：</label>
<input type="text" name="subject" style="width:440px" maxlength="100" required
id="id_subject" />
```

每一个表单对象中的字段，我们都可以进行这样的处理，以满足我们的一些个性化需求（例如通过“class”属性指定元素的样式）。

另外，上方代码中<label>标签，我们通过“`{{ form.字段名称.属性名称_for_label }}`”获取到了这个标签“for”属性的值。

如果对于<label>标签没有任何处理的话，我们也可以通过“`{{ form.字段名称.label_tag }}`”直接加载这个标签。

示例代码：

```
<form action="/send_email/" method="post">

    {{ form.subject.label_tag }}

    {{ form.subject }}
```

```
<input type="submit" value="发送">
</form>
```

最终页面呈现的效果以及源代码是一样的。

表单相关的属性还有更多，这里给大家一一列出，并说明它们的用途。

- `{{ field.label }}`：字段对应的`<label>`标签的文字，例如“发件人”。
- `{{ field.label_tag }}`：字段对应的`<label>`标签。
- `{{ field.id_for_label }}`：字段的“id”属性值。
- `{{ field.value }}`：字段的值，例如标题的内容。
- `{{ field.html_name }}`：字段对应的 HTML 标签“name”属性的值。
- `{{ field.help_text }}`：字段的帮助文本。
-
- `{{ field.is_hidden }}`：字段是否隐藏字段，获取到的是布尔值。
-

3、循环输出表单元素

表单对象是一个可迭代对象。

这也就意味着，我们可以通过“for”循环遍历每一个表单元素，并进行相应的处理。

还是以之前的表单对象为例。

示例代码：

```
<form action="/send_email/" method="post">

    {% for field in form %}

        <div>

            {{ field.label_tag }}

            {{ field }}

            {% if field.help_text %}

                <span style="color: orange">{{ field.help_text }}</span>

            {% endif %}

        </div>

        <br>
```

```
{% endfor %}
```

```
<input type="submit" value="发送">
</form>
```

上方代码，在浏览器中呈现的内容如下。



另外，除了直接对表单对象进行循环遍历，Django 还支持对表单对象中的可见元素与隐藏元素分别进行遍历。

示例代码：

```
{% for hidden in form.hidden_fields %}
    {{ hidden }}
{% endfor %}
{% for field in form.visible_fields %}
    {{ field.label_tag }}
    {{ field }}
{% endfor %}
```

以上就是表单数据整合到模板的几种不同方式。

三、重用表单

在模板的教程中，我们知道一个模板中可以包含另外一个模板。

所以，我们可以这样来操作。

1、把表单单独创建一个模板。

示例代码：（form.html）

```
<form action="/send_email/" method="post">
  {% for field in form %}
    <div>
      {{ field.label_tag }}
      {{ field }}
      {% if field.help_text %}
        <span style="color: orange">{{ field.help_text }}</span>
      {% endif %}
    </div>
    <br>
  {% endfor %}
  <input type="submit" value="发送">
</form>
```

2、通过模板嵌套的方式，嵌套在其他模板中。

示例代码：（email.html）

```
<!DOCTYPE html>
<html lang="zh-cn">
<head>
  <meta charset="UTF-8">
  <title>撰写邮件</title>
</head>
<body>
```

```
{% include 'form.html' %}}
```

```
</body>
</html>
```

不过，在表单“form.html”模板中，我们是对名为“form”的表单对象进行处理；如果模板“email.html”获取的表单对象名称是“email_form”，此时就不能够正确的加载表单。

示例代码：（views.py）

```
def write_email(request):  
    email_form = ContactForm()  
    return render(request, 'email.html',  
  
{'email_form': email_form})
```

我们需要在嵌入表单模板时，为表单对象设置一个别名，与表单模板中的表单对象名称保持一致。

示例代码：（email.html）

```
{% include 'form.html' with form=email_form %}
```

最后，关于 Django 表单更加深入的内容还有很多，例如之后我们会学习通过模型生成表单。

关于更多的表单内容，大家可以参考 Django 官方文档进行深入的了解。

本节练习源代码：【[点此下载](#)】

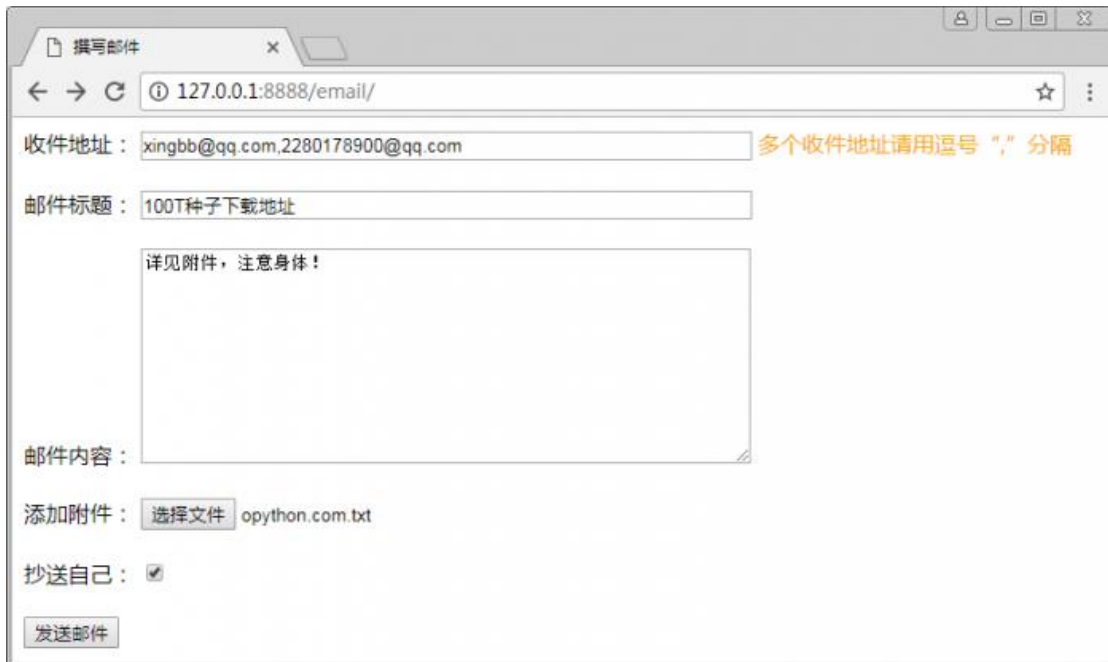
转载请注明：魔力 Python » Django2 : Web 项目开发入门笔记（18）

原文链接：<http://www.opython.com/1096.html>

Django2 : Web 项目开发入门笔记（19）

这一篇教程，我们一起来学习使用 Django 发送电子邮件。

首先，我们先来看一下撰写邮件的页面。



还有收到的邮件内容。



接下来，我们结合 Django 的表单完成上述功能。

一、创建表单类。

打开“forms.py”文件，根据截图中的页面元素，我们创建表单类“FileMailForm”，这个类包含邮件附件字段。

示例代码：（forms.py）

```
from django import forms
```

```
class FileMailForm(forms.Form):
    addressees = forms.CharField(label='收件地址', help_text='多个收件地址请用逗号","分隔',
label_suffix=' : ',
                                widget=forms.TextInput(attrs={'style': 'width:440px', 'maxlength': '100'}))
    # 定义收件地址字段，因为支持群发，所以未使用“EmailField”。
    subject = forms.CharField(label='邮件标题', label_suffix=' : ',
                                widget=forms.TextInput(attrs={'style': 'width:440px', 'maxlength': '100'}))
    # 定义邮件标题字段，指定文本框宽度并限定字符数量。
    message = forms.CharField(label='邮件内容', label_suffix=' : ',
                                widget=forms.Textarea(attrs={'cols': '60', 'rows': '10'}))
    # 定义邮件内容字段，指定使用文本域（多行文本框）标签，并设置标签的属性。
    file = forms.FileField(label='添加附件', label_suffix=' : ')
    # 定义邮件附件的字段。
    cc_myself = forms.BooleanField(label='抄送自己', label_suffix=' : ', required=False)
    # 定义抄送给发件人字段，指定为布尔类型（真假值）的字段。
```

提示：邮件地址字段可以使用“EmailField”，但不支持对多个邮件地址进行验证。

这里，我们还要考虑到发送邮件时会有两种情况：有附件和没有附件。

所以，我们创建一个“FileMailForm”类的子类“TextEmailForm”，子类中对“file”字段进行重写，当没有邮件附件时，通过这个类实例化表单。

示例代码：

```
class TextEmailForm(FileMailForm):
    file = None
```

二、创建模板

1、创建一个 HTML 文件“email.html”，用于撰写邮件。

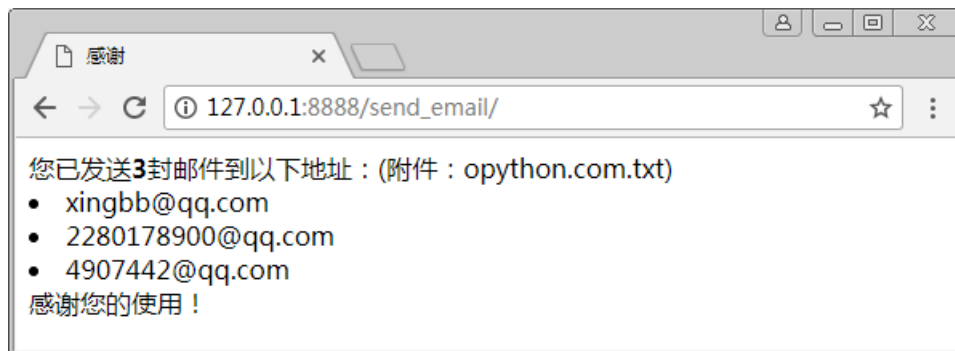
示例代码：（email.html）

```
<!DOCTYPE html>
<html lang="zh-cn">
<head>
    <meta charset="UTF-8">
    <title>撰写邮件</title>
</head>
<body>
<form action="/send_email/" method="post" novalidate="novalidate"
enctype="multipart/form-data">
    {% csrf_token %}
```

```
{% for field in email_form %}
<div>
    {{ field.label_tag }}
    {{ field }}
    {% if field.help_text %}
        <span style="color: orange">{{ field.help_text }}</span>
    {% endif %}
</div>
<br>
{% endfor %}
<input type="submit" value="发送邮件">
</form>
</body>
</html>
```

在上方代码中，<form>标签额外添加了两个属性，“novalidate”属性用于禁用浏览器的字段验证，“enctype”属性规定在发送到服务器之前，如何对表单数据进行编码，此处的设置“multipart/form-data”表示不进行编码，在使用包含文件上传元素的表单时，必须使用这个属性值。

2、创建一个 HTML 文件“thanks.html”，用于发送邮件后的提示。



示例代码：（thanks.html）

```
<!DOCTYPE html>
<html lang="zh-cn">
<head>
    <meta charset="UTF-8">
    <title>感谢</title>
</head>
<body>
    您已发送<b>{{ count }}</b>封邮件到以下地址：{{ file }}
    {% for email in to %}
        <li>{{ email }}</li>
```



```
{% endfor %}
感谢您的使用！
</body>
</html>
```

注意，在“thanks.html”模板中，包含三个变量“count”、“file”和“to”，分别为发送的邮件数量、附件名称和收件地址。

三、添加 Email 配置

打开配置文件“settings.py”，添加 Email 相关配置。

我们发送邮件需要使用某一个邮件服务器，这里以 QQ 邮箱的发件服务器为例。

在 QQ 邮箱的【设置】-【账户】中，我们开启 QQ 邮箱的【POP3/SMTP】或者【IMTP/SMTP】服务，并获取授权码。

因为我们只实现发送邮件的功能，所以使用这两个服务没有什么区别。



然后，大家可以点击相关的帮助，查看帮助内容。

在帮助内容的页面中，有如下内容：

如何设置IMAP服务的SSL加密方式？

使用SSL的通用配置如下：

接收邮件服务器：imap.qq.com，使用SSL，端口号993

发送邮件服务器：smtp.qq.com，使用SSL，端口号465或587

账户名：您的QQ邮箱账户名（如果您是VIP帐号或Foxmail帐号，账户名需要填写完整的邮件地址）

密码：您的QQ邮箱密码

电子邮件地址：您的QQ邮箱的完整邮件地址

上述内容中，就包含我们要在配置文件“settings.py”中添加的相关配置。

示例代码：

```
EMAIL_USE_SSL = True # 使用 SSL 加密方式
EMAIL_HOST = 'smtp.qq.com' # 发送邮件服务器地址
EMAIL_PORT = '465' # 发送邮件服务器端口
EMAIL_HOST_USER = '4907442@qq.com' # 发件邮箱地址
EMAIL_HOST_PASSWORD = 'limj*****cbaa' # 开启 STMP 服务时生成的授权码
DEFAULT_FROM_EMAIL = '小楼一夜听春雨<4907442@qq.com>' # 默认发件邮箱地址
```

四、定义视图函数

1、导入需要使用的模块

示例代码：

```
from django.shortcuts import render, HttpResponse
from .forms import TextEmailForm, FileMailForm # 定义的表单类
from django.core.mail import send_mail # 发送单封邮件
from django.core.mail import send_mass_mail # 发送多封邮件
from django.core.mail import EmailMultiAlternatives # 发送带有附件邮件
from MyForm import settings # 配置文件
import os
```

2、撰写邮件的视图函数

示例代码：

```
def write_email(request):
    email_form = FileMailForm()
    return render(request, 'email.html', {'email_form': email_form})
```

3、处理附件上传的函数

这个函数需要完成附件的上传（读取附件文件写入到服务器指定目录），并获得附件文件的路径。

示例代码：

```
def upload_handler(file, file_name):
    path = os.path.join(settings.BASE_DIR, 'uploads/') # 项目根目录下的附件上传目录
    if not os.path.exists(path): # 如果目录不存在
        os.makedirs(path) # 创建目录
    with open(path + file_name, 'wb+') as f: # 打开文件
        for chunk in file.chunks(): # 读取上传文件
            f.write(chunk) # 写入文件
    return path + file_name # 返回文件路径
```

4、发送带有附件邮件的函数

为了让发送邮件的视图函数更清晰，单独将发送带有附件邮件的功能定义为函数。

示例代码：

```
def file_mail_send(subject, message, sender, addressees, file):
    email = EmailMultiAlternatives(subject, message, sender, addressees) # 创建邮件对象
    file_path = upload_handler(file, str(file)) # 上传附件并获取文件路径
    email.attach_file(file_path) # 附加文件到邮件对象
    email.send() # 发送邮件
```

5、发送邮件的视图函数

发送邮件的视图函数中需要考虑以下内容：

- 表单提交是否使用“POST”方法提交，需要进行不同的处理；
- 邮件是否包含附件决定创建不同的表单对象，以及发送邮件的方法；
- 表单对象验证是否成功，需要进行不同的处理；
- 邮件发送是否成功，需要进行不同的处理；
- 邮件的数量（单封或多封）决定不同的发送邮件方法。

示例代码：

```
def send_email(request):
    if request.method == 'POST': # 如果表单为“POST”方法提交，则进行邮件发送。
        if request.FILES: # 如果包含附件
            email_form = FileMailForm(request.POST, request.FILES) # 创建包含附件的表单对象
            file = request.FILES['file'] # 获取附件信息
```

```
else:
    email_form = TextEmailForm(request.POST) # 创建不包含附件的表单对象
    file = None
    if email_form.is_valid(): # 如果验证有效
        addressees = email_form.cleaned_data['addressees'].split(',') # 获取所有收件地址的
列表
        subject = email_form.cleaned_data['subject'] # 获取邮件标题
        message = email_form.cleaned_data['message'] # 获取邮件内容
        cc_myself = email_form.cleaned_data['cc_myself'] # 获取是否抄送发件人
        if cc_myself: # 如果抄送发件人
            addressees.append(settings.EMAIL_HOST_USER) # 添加发件人的邮件地址到收件
地址列表
        count = len(addressees) # 获取发件数量
        email = [subject, message, settings.DEFAULT_FROM_EMAIL, addressees] # 保存所有
发件信息到变量
        try:
            if file: # 如果包含附件
                file_mail_send(*email, file) # 调用发送带有附件邮件的函数
                file_name = '(附件 : %s)' % str(file) # 发送成功信息中的附件信息
            else: # 如果不包含附件
                if count > 1: # 如果有多个收件地址
                    send_mass_mail((email,)) # 调用适合发送多封邮件的方法
                else: # 否则
                    send_mail(*email) # 调用适合发送单封邮件的方法
                file_name = ""
        except:
            # raise # 编写代码时获取异常信息
            return HttpResponse('发送失败！') # 如果发送邮件发生异常，则给出提示。
        return render(request, 'thanks.html',
            {'count': count, 'to': addressees, 'file': file_name}) # 正常发送邮件后，页面显
示发送邮件的相关信息。
    else: # 如果验证失败，则进行提示。
        return HttpResponse('验证失败！')
else: # 如果表单不是“POST”方法提交，则回到撰写邮件的页面。
    email_form = FileMailForm()
    return render(request, 'email.html', {'email_form': email_form})
```

在上方代码中，我们使用的“send_mail()”方法和“send_mass_mail()”方法实际上都可以发送单封或多封邮件，区别在于“send_mail()”发送多封邮件时，是一个一个的分开发送，而“send_mass_mail()”方法是多封邮件一起发送。

五、配置 URL 分发

最后，我们完成 URL 分发的配置，并启动开发服务器进行测试。

示例代码：（urls.py）

```
from django.urls import path
from FormTest import views as form_views

urlpatterns = [
    path('email/', form_views.write_email),
    path('send_email/', form_views.send_email),
]
```

注意：邮件附件不能够为 0 字节的空文件，否则会在发送邮件时验证失败。

本节练习源代码：【[点此下载](#)】

转载请注明：魔力 Python » Django2 : Web 项目开发入门笔记（19）

原文链接：<http://www.opython.com/1122.html>

Django2 : Web 项目开发入门笔记（20）

这一篇教程，我们一起来了解如何在 CentOS 系统中将 Django2 的 Web 项目部署到 Nginx 服务器。

CentOS 系统虽然和 Ubuntu 系统都是 Linux 系统，但是环境搭建和部署过程还是有一些区别。

整个流程分为几个部分：

- 安装图形桌面与远程登录
- 安装 Python3.6 及相关库文件
- 安装 Django2
- 安装 uWSGI
- 安装 Nginx
- 配置 Nginx
- 使用 ini 文件启动 uWSGI 服务器
- 使用 supervisor 管理 uWSGI 服务器

接下来，我们就逐一完成这些步骤。

一、安装图形界面

在此之前大家应该先完成 CentOS 系统的最小化安装。

然后，安装我们需要的图形界面和远程登录功能。

因为远程登录需要图形界面支持，所以从顺序上先进行图形界面的安装，再安装远程登录。

（一）使用 Gnome 桌面

GNOME 桌面比较耗费系统资源，在主机上我们更多是通过命令行进行操作，所以，我比较倾向于使用面向低性能硬件的 Xfce 桌面。

如果选择使用 Xfce 桌面，大家可以略过此步骤，直接浏览第（二）部分。

1、安装软件源

执行命令：`yum install epel* -y`

2、更新软件包

执行命令：`yum -y upgrade`

3、安装桌面支持

执行命令：`yum groupinstall "X Window System" "GNOME Desktop" -y` 或者：`yum -y groupinstall "Server with GUI"`

4、安装 xrdp 和 vnc

执行命令：`yum install tigervnc-server xrdp -y`

5、启动 xrdp 服务，并且设置为开机启动

执行命令：`systemctl start xrdp`

（二）使用 Xfce 桌面

1、安装软件源

执行命令：`yum install epel* -y`

2、更新软件包

执行命令：`yum -y upgrade`

3、安装桌面管理器

执行命令：`yum install lightdm -y`

4、安装桌面

执行命令：`yum groupinstall xfce -y`

5、安装远程服务

执行命令：`yum install tigervnc-server xrdp -y`

6、禁用 GDM 桌面管理器

执行命令：`systemctl disable gdm`

7、启用 LightDM 桌面管理器

执行命令：`systemctl enable lightdm`

8、配置 Xfce 为默认桌面

执行命令：`vim ~/.Xclients`

在打开的文件中，写入以下内容：

```
#!/bin/bash
XFCE="$(which xfce4-session 2>/dev/null)"
exec "$XFCE"
```

代码输入完毕，按“ESC”键并键入“:wq”回车，保存测试文件。

然后，执行命令，增加执行权限：`chmod +x ~/.Xclients`

9、启动或重启远程连接服务

执行命令：`systemctl start xrdp`

或者：`systemctl restart xrdp`

10、设置远程连接为开机启动

执行命令：`systemctl enable xrdp`

二、安装 Python3.6

CentOS 系统自带的是 Python2.7.5，可以通过输入“python”命令打开。

我们安装了 Python3.6 之后，需要使用命令“python3”启动 Python3.6 的 Shell。

1、安装相关库文件

执行命令：`yum -y install zlib*yum -y install gccyum -y install gcc-c++yum -y install opensslyum -y install openssl-develyum -y install sqliteyum -y install sqlite-develyum -y install readline readline-devel`

2、安装 Python3.6 与相关库文件

（一）安装 Python3.6

首先，创建一个用于保存下载文件的文件夹，并赋予权限。

执行命令：`mkdir /home/centos/Downloadschmod /home/centos/Downloadschmod 777 /home/centos/Downloads`

然后，下载 Python 的源码安装包。

下载地址：<https://www.python.org/downloads/source/>

如果是通过 Windows 远程登录，可以直接复制系统中下载好的 Python 源码安装包，粘贴到 CentOS 系统的文件夹中。

当然，也可以在 CentOS 系统中通过“wget”命令进行下载。

执行命令：`wget https://www.python.org/ftp/python/3.6.5/Python-3.6.5.tgz`

最后，解压缩软件安装包，进入解压缩后的目录进行安装。

执行命令：`tar xzf Python-3.6.5.tgzcd Python-3.6.5./configure --enable-shared --with-ssl=opensslmake && make install`

安装完毕后，启动 Python3.6。

执行命令：`python3`

此时，可能会出现错误。

`python3: error while loading shared libraries: libpython3.6m.so.1.0: cannot open shared object file: No such file or directory`

产生错误的原因是：配置文件添加了参数“--enable-shared”，Python3.6 运行时没有加载到文件“libpython3.6m.so.1.0”。

实际上我们在执行“make”命令时，已经编译了这个文件，解决问题的方法就是把编译好的文件复制到特定的目录中。

执行命令：`cd /home/centos/Downloads/Python-3.6.5cp libpython3.6m.so.1.0 /usr/local/lib64/cp libpython3.6m.so.1.0 /usr/lib/cp libpython3.6m.so.1.0 /usr/lib64/`

（二）安装相关库

我们需要安装“python36-devel”。

```
yum -y install python36-devel
```

如果正常完成安装，可直接跳转到下一部分“安装 Django2”。

如果这个库没有在系统默认源中，我们需要先添加一个安装源工具，通过下载 rpm 文件进行安装。

但是，直接下载 rpm 文件进行安装有可能会出错。

Warning: user mockbuild does not exist. using root

所以，我们需要先安装一个依赖库。

执行命令：yum install mock -y useradd -s /sbin/nologin mockbuild

然后，下载“rpmforge”的安装文件。

可以到“<http://repoforge.org/use/>”进行下载，或者通过“wget”命令进行下载。

执行命令：cd /home/centos/Downloads/wget

```
http://repository.it4i.cz/mirrors/repoforge/redhat/el7/en/x86_64/rpmforge/RPMS/rpmforge-release-0.5.3-1.el7.rf.x86_64.rpm
```

下载完成后进行安装。

执行命令：rpm -ivh rpmforge-release-0.5.3-1.el7.rf.x86_64.rpm

此时，可能会出现“NOKEY”的错误，是因为“yum”安装了旧版本的“GPG keys”造成的，解决办法是导入“gpg”。

执行命令：rpm --import /etc/pki/rpm-gpg/RPM*

接下来，就可以安装“python36-devel”了。

可以先通过命令，搜索 python3-devel 的可用版本。

执行命令：yum search python3 | grep devel

在搜索结果中可以看到一些关于 Python3.X 的文件名称，其中有一个为“Python36”。

执行命令：yum -y install python36-devel

这样就完成了“python36-devel”的安装。

三、安装 Django2

Python3.6 默认安装后，需要使用命令“pip3”安装第三方库。

先安装 Django 的依赖库“pytz”。

执行命令：pip3 install pytz

然后，安装 Django。

执行命令：pip3 install django

或者，下载 Django 的安装包，放入“Downloads”文件夹后进行安装。

执行命令：cd /home/centos/Downloads/tar xzf Django-2.0.3.tar.gzcd Django-2.0.3python3 setup.py install

最后，测试一下 Python3.6、Django 以及 Sqlite3 是否能够正常使用。

执行命令：python3>>>import django>>>import sqlite3

四、安装 uWSGI

注意：不要用“yum install uwsgi”进行安装，这样装完会关联系统中的 Python2.7，并且系统可能会自带 uwsgi，自带 uwsgi 的启动项在“/usr/sbin/”中，而我们通过“pip3”命令安装 uwsgi 的启动项在“/usr/local/bin/”目录中。

执行命令：pip3 install uwsgi

如果怕搞混，我们可以卸载系统自带的 uwsgi，然后将启动项复制到“/usr/sbin/”目录中。

执行命令：yum remove uwsgicp -f /usr/local/bin/uwsgi /usr/sbin/

或者，我们可以将自己安装的 uwsgi 启动项复制到“/usr/sbin/”目录中时，改名为“uwsgi3”。

执行命令：cp -f /usr/local/bin/uwsgi /usr/sbin/uwsgi3

接下来，我们测试一下 uwsgi 是否能够正常工作。

创建一个测试文件“mytest.py”。

执行命令：vi /var/www/mytest.py

写入内容：

```
#!/usr/bin/python3 # 也可以写成“#!/usr/bin/python3.6”。
def application(env, start_response):
```

```
start_response('200 OK', [('Content-Type', 'text/html')])  
return [b'UWSGI Test...']
```

代码输入完毕，按“ESC”键并键入“:wq”回车，保存测试文件。

然后，系统中如果没有自带浏览器的话，可以安装火狐浏览器。

执行命令：`yum install firefox -y`

最后，进行测试。

执行命令：`cd /var/www/uwsgi3 --http :8888 --wsgi-file mytest.py`

此时，通过本机浏览器访问“`http://127.0.0.1:8888`”或者“`http://localhost:8888`”进行测试，如果页面中显示“UWSGI Test...”字样，则说明测试成功，uwsgi 可以正常工作了。

五、安装 Nginx

首先，安装 Nginx 的依赖库“pcre”。

执行命令：`yum install -y pcre pcre-devel`

然后，安装 Nginx。

执行命令：`yum install -y nginx*`

安装完成后，就可以通过命令控制 Nginx 了。

启动命令：`/usr/sbin/nginx` 停止命令：`/usr/sbin/nginx -s stop` 退出命令：
`/usr/sbin/nginx -s quit` 重载命令：`/usr/sbin/nginx -s reload` 查询进程：`ps aux|grep nginx`

另外，我们还可以设置 Nginx 为开机自启动。

执行命令：`vi /etc/rc.local`

在打开的文件中，按“i”键进入编辑模式，添加一行内容。`/usr/local/nginx/sbin/nginx`

输入完毕，按“ESC”键并键入“:wq”回车，保存测试文件。

六、配置 Nginx

创建配置文件，并写入内容。

执行命令：`vi /etc/nginx/conf.d/MyWeb.conf`

写入内容：

```
server {  
    listen 80;  
    server_name www.qqtbb.com;  
    charset utf-8;  
    client_max_body_size 5M;  
  
    location /media {  
        alias /var/www/MyWeb/media;  
    }  
  
    location /static {  
        alias /var/www/MyWeb/static;  
    }  
  
    location / {  
        uwsgi_pass 127.0.0.1:8888;  
        include /etc/nginx/uwsgi_params;  
    }  
}
```

提示：配置内容的详细说明，可以参考《Django2：Web 项目开发入门笔记（16）》。

内容输入完毕（使用时请先清除注释），按“ESC”键并键入“:wq”回车保存，并让服务器重载配置。

执行命令：`nginx -s reload`

此时，可能发生错误。`nginx: [error] open() "/usr/local/var/run/nginx.pid" failed (2: No such file or directory)`

解决方法是找到“`nginx.conf`”的文件夹目录，然后运行“`nginx`”命令。

例如，“`nginx.conf`”文件在“`/etc/nginx/`”目录中。

执行命令：`nginx -c /etc/nginx/nginx.conf nginx -s reload`

如果发生 80 端口被占用的情况，可以先查询占用端口的进程，通过“`kill`”命令关闭进程。

执行命令：`lsof -i :80 kill -9 [进程 ID] nginx -c /etc/nginx/nginx.conf`

七、使用 ini 文件启动 uWSGI 服务器

首先，使用 ini 文件启动 uWSGI 服务器，需要安装依赖库。

执行命令：`yum -y install uwsgi-plugin-python3`

然后，在项目文件夹中创建 ini 文件。

例如，在“/var/www/MyWeb”中存放 Web 项目文件。

执行命令：vi /var/www/MyWeb/uwsgi.ini

在新建的文件中输入内容。

```
[uwsgi]
socket = 127.0.0.1:8888 # 因为要接收来自 Nginx 的 Socket，此处必须和 Nginx 的设置保持一致。
chdir = /var/www/MyWeb/
wsgi-file = MyWeb/wsgi.py # 完整路径是“/var/www/MyWeb/MyWeb/wsgi.py”
processes = 3 # 注意，此处启用了多进程，之后使用 supervisor 管理 uWSGI 时，需要增加配置项。
threads = 5
chmod-socket = 664
chown-socket = www-data
pidfile= /var/www/MyWeb/MyWeb.pid
vacuum = true
```

提示：配置内容的详细说明，可以参考《Django2：Web 项目开发入门笔记（16）》。

内容输入完毕（使用时请先清除注释），按“ESC”键并键入“:wq”回车保存，然后就可以通过配置文件启动 uWSGI 服务器了。

启动命令：uwsgi3 --ini /var/www/MyWeb/uwsgi.ini

停止命令：uwsgi3 --stop /var/www/MyWeb/MyWeb.pid

重载配置：uwsgi3 --reload uwsgi.ini

再次强调：启动时注意 Python 版本是否 Python3.6，并且不要使用系统自带的 uwsgi。

此时，通过域名就能够访问我们的 Web 项目了。

八、使用 supervisor 管理 uWSGI 服务器

supervisor 可以在程序意外关闭时自动重新启动，使用它管理 uWSGI 服务器非常不错。

不过，supervisor 只支持 Python2，我们需要通过 CentOS 自带的 Python2.7 进行安装。

首先，安装“pip”工具。

执行命令：cd /usr/lib/python2.7/site-packages/easy_install pip

然后，安装“supervisor”。`pip install supervisor`

接下来，进行配置。

执行命令：`echo_supervisord_conf > /etc/supervisord.conf`

在打开的文件末尾添加内容。

注意：语句前面不要有空格。

```
[program:MyWeb]
command=uwsgi3 --ini /var/www/MyWeb/uwsgi.ini
directory=/var/www/MyWeb/
startsecs=10
stopwaitsecs=10
stopasgroup=true
killasgroup=true
autostart=true
autorestart=true
```

提示：配置内容的详细说明，可以参考《Django2：Web 项目开发入门笔记（16）》。

这里特别需要注意的是，如果“uwsgi.ini”文件中开启了多进程，一定要加上下面两句。

```
stopasgroup = true # 用于停止进程组，即停止所有通过“uwsgi.ini”配置启动的进程。
killasgroup = true # 用于关闭进程组，即关闭所有通过“uwsgi.ini”配置启动的进程。
```

如果不添加这两句，`supervisorctl` 命令停止或关闭进程时，只会关闭其中 1 个进程，从而导致再次启动或重启 uWSGI 失败，出现“ERROR (spawn error)”的错误。

这是因为残留的孤儿进程，阻止了新的同类进程的开启。

当我们完成配置文件的修改之后，必须重新加载配置文件，才能使其生效。

执行命令：`supervisorctl reload`

此时，可能出现错误：`Unlinking stale socket /var/run/supervisor.sock`

这个的错误是因为缺少依赖，需要安装“python-meld3”。

执行命令：`yum -y install python-meld3`

最后，启动 `supervisord` 和 `uWSGI`。

执行命令：`/usr/bin/supervisord -c /etc/supervisord.conf` 或者：`supervisorctl -c /etc/supervisord.conf start MyWeb`

此时，可能会发生错误。Error: Another program is already listening on a port that one of our HTTP servers is configured to use. Shut this program down first before starting supervisord.

这是因为已经有“supervisor.sock”文件被链接（上一次启动 Supervisor 造成的）。

我们需要先查找哪个“supervisor.sock”文件被链接，并取消已有链接。

执行命令：`find / -name supervisor.sock` unlink /被链接文件的所在路径/supervisor.sock

如果出现错误“ERROR (spawn error)”，可能是 uWSGI 进程已存在所导致的。

执行命令：`ps -ef|grep uwsgi` kill -9 [端口号]

或者使用“killall”命令通过名称关闭全部相关进程。

`killall -9 uwsgi`

此时，再次启动项目就可以了。

另外，通过“supervisorctl”命令，可以方便的管理项目。

启动项目命令：`supervisorctl start [配置文件中的项目名称]` 重启项目命令：`supervisorctl restart [配置文件中的项目名称]` 停止项目命令：`supervisorctl stop [配置文件中的项目名称]`
控制所有项目：`supervisorctl <start/restart/stop> all`

注意：如果系统中出现了 Python3 的其他版本（例如：Python3.4.8），并且“python3”命令不能启动 Python3.6 时，可以通过创建新的链接解决。

执行命令：

`rm -f /usr/bin/python3`

`ln -s /usr/local/bin/python3 /usr/bin/python3`

转载请注明：魔力 Python » Django2 : Web 项目开发入门笔记（20）

原文链接：<http://www.opython.com/1134.html>

Django2 : Web 项目开发入门笔记（21）

这一篇教程，我们一起来学习如何使用 ModelForm，通过模型创建表单。

假设，我们完成一个用户注册的功能，用户注册的内容包括：邮箱、密码、姓名、年龄、生日。

提示：不要纠结这样的注册信息对用户是否友好，我们只是通过它们进行练习，熟悉字段的使用以及一些对字段的处理方法。

首先，在“settings.py”中，先将当前应用添加到“INSTALLED_APPS”的列表中。

然后，我们开始编写代码。

一、在“models.py”中创建模型类“UserModel”。

示例代码：

```
from django.db import models

class UserModel(models.Model):
    email = models.EmailField('邮箱')
    password = models.CharField('密码', max_length=256)
    name = models.CharField('姓名', max_length=20)
    age = models.IntegerField('年龄')
    birthday = models.DateField('生日')
```

创建完成后，使用“makemigrations”和“migrate”命令连接数据库创建数据表。

如果不知道上一句在说什么，请再次学习《Django2：Web 项目开发入门笔记（7）》。

二、在“forms.py”中创建表单类“RegisterForm”。

示例代码：

```
from django import forms
from FormTest import models # 导入模型模块

class RegisterForm(forms.ModelForm): # 继承 ModelForm 类
    class Meta:
        model = models.UserModel
        fields = '__all__'
```

在上方代码中，表单类“RegisterForm”里面写了一个内嵌类“class Meta”，它能够让我们定义 RegisterForm 类的元数据。

在这个内嵌类中，model 为表单类对应的模型类；fields 为表单字段中所包含的模型类字段，“__all__”表示表单字段包含所有的模型类字段。

实际上这段代码相当于下方代码：

```
from django import forms
```



```
class RegisterForm(forms.Form):
    email = forms.EmailField(label='邮箱')
    password = forms.CharField(label='密码', max_length=256)
    name = forms.CharField(label='姓名', max_length=20)
    age = forms.IntegerField(label='年龄')
    birthday = forms.DateField(label='生日')
```

那么，通过上方的代码，大家也能够看出模型字段和表单字段存在对应关系。

具体如下：

三、创建模板

模板内容比较简单，无需过多解释。

示例代码：（register.html）

```
<!DOCTYPE html>
<html lang="zh-cn">
<head>
    <meta charset="UTF-8">
    <title>用户注册</title>
</head>
<body>
<form action="/register/" method="post">
    {% csrf_token %}
    {{ form.as_p }}
    <input type="submit" value="注册">
</form>
</body>
</html>
```

四、添加 URL 分发配置

示例代码：

```
from django.urls import path
from FormTest import views as form_views

urlpatterns = [
    path('register/', form_views.register),
]
```

五、创建视图函数，将表单与模板整合。

示例代码：

```
from django.shortcuts import render, HttpResponseRedirect
from FormTest.forms import RegisterForm
```

```
def register(request):
    if request.method == 'POST':
        form = RegisterForm(request.POST)
        if form.is_valid():
            try:

                form.save()

            except:
                return HttpResponseRedirect('糟糕，注册失败了！！')
            return HttpResponseRedirect('恭喜您，注册成功！')
        form = RegisterForm()
        return render(request, 'register.html', {'form': form})
```

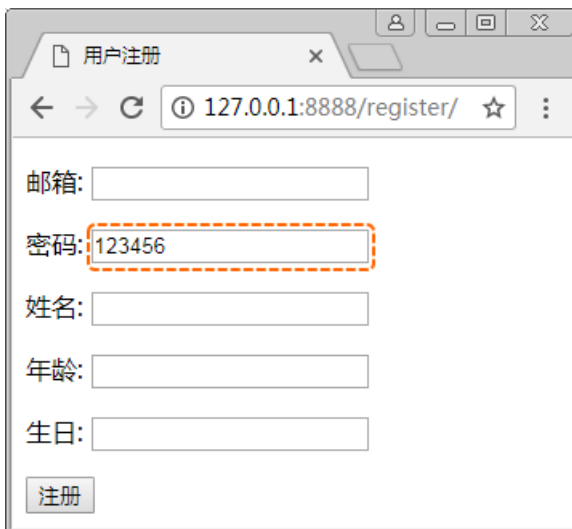
在上方代码中，大家注意标红的语句。

只需要这一句，我们就能够完成将表单数据写入数据库的操作。

这就是使用“ModelForm”的便捷之处。

不过，目前我们了解这些还不够。

例如，我们刚刚完成的注册界面，密码的输入能够看到内容。



The screenshot shows a web browser window with the title '用户注册' (User Registration). The address bar shows '127.0.0.1:8888/register/'. The form contains the following fields and labels:

- 邮箱:
- 密码: (This field is highlighted with a red dashed border)
- 姓名:
- 年龄:
- 生日:

At the bottom of the form is a button labeled '注册' (Register).

但是，这些表单中字段的控件都是自动生成的，怎么能够让密码变为加密类型呢？

在 `forms` 中有 `PasswordInput` 类，能够产生密码类型的表单控件。

我们可以单独定义“password”字段，使用“`PasswordInput`”生成密码框控件。

在代码中，我们添加一段内容。

示例代码：

```
class RegisterForm(forms.ModelForm): # 继承 ModelForm 类
    class Meta:
        model = models.UserModel
        fields = '__all__'

        widgets = {

            'password': forms.PasswordInput()

        }
```

上方代码，在“`widgets`”字典中，添加一个键值对，键为 `Model` 字段的名称，值为表单控件对象。

使用这种方法，我们也可以定义表单中“`label`”控件的文字。

示例代码：

```
labels = {
    'name': '芳名'
}
```

还有，定义帮助信息。

示例代码：

```
help_texts = {
    'birthday': ('日期格式：2018-2-28'),
}
```

当然，错误信息也可以照此操作。

示例代码：

```
error_messages = {
    'name': {
```

```
        'max_length': ("你那个太长了啦！."),  
    },  
}
```

另外，我们还可以指定字段使用其他类型（可以是自定义字段类型）。

因为没有自定义字段，下方代码使用 `forms` 中自带的字段进行演示。

示例代码：

```
field_classes = {  
    'age': forms.CharField  
}
```

除了上述这些，在“`class Meta`”中还有更多的功能。

大家肯定想到了，有时我们并不想把 `Model` 中的所有字段都生成表单控件。

这只需要把需要的字段赋值给“`fields`”。

示例代码：

```
fields = ['email', 'password', 'age']
```

或者，我们只有少数的字段想要排除在外。

也可以使用下方的代码。

示例代码：

```
exclude = ['birthday']
```

以上是关于“`class Meta`”的使用。

接下来，我们再一起了解一下“`save()`”方法的相关内容。

在前面的示例中，我们已经看到通过“`save()`”方法，可以将表单数据直接存入数据库。

不过，我们在使用过程中还会有其他的需求。

1、表单字段值修改后，存入数据库。

例如，我们把密码加密后存入数据库。

示例代码：

```
from django.shortcuts import render, HttpResponseRedirect
from FormTest.forms import RegisterForm
```

```
from hashlib import md5
```

```
def register(request):
    if request.method == 'POST':
        form = RegisterForm(request.POST)
        if form.is_valid():

            email = form.cleaned_data['email']

            password = form.cleaned_data['password']

            try:

                new_form = form.save(commit=False)

                m = md5()

                m.update((email + password).encode('utf-8'))

                new_form.password = m.hexdigest()

                new_form.save()

                form.save_m2m()

            except:
                # raise
                return HttpResponseRedirect('糟糕，注册失败了！！')
            return HttpResponseRedirect('恭喜您，注册成功！')
        form = RegisterForm()
        return render(request, 'register.html', {'form': form})
```

上方代码中，红色部分为新增代码。

2、一些字段无需用户填写，由系统自动给定。

例如，注册时无需用户填写生日，系统自动给定默认值，留作以后用户自行修改。

我们可以先在表单类中排除“birthday”字段。

示例代码：

```
class RegisterForm(forms.ModelForm):  
    class Meta:  
        model = models.UserModel
```

```
exclude = ['birthday']
```

然后，在视图函数中，创建一个 `Model` 对象，设置“birthday”字段的默认值，并添加到表单对象中。

示例代码：

...省略部分代码...

```
from FormTest.models import UserModel
```

```
def register(request):  
    if request.method == 'POST':  
  
        user = UserModel(birthday='1990-1-1')  
  
        form = RegisterForm(request.POST,instance=user)
```

```
        if form.is_valid():  
...省略部分代码...
```

以上就是关于将表单数据存入数据库的一些操作，另外还有“`save_m2m()`”方法，用于多对多（many to many）数据的保存，我们还没有接触过多对多的数据库操作，在此不做演示。有兴趣的朋友可以参考官方文档的相应示例。

最后一部分内容，关于使用 `ModelForm` 的工厂函数（factory function）。

之前，我们都是通过表单类完成一个表单的创建。

其实，我们还可以采用 `ModelForm` 的工厂函数方式创建表单。

例如下方这个表单。

示例代码：

```
from django import forms
from FormTest import models
class RegisterForm(forms.ModelForm):
    class Meta:
        model = models.UserModel
        fields = ['email', 'password', 'age']
```

使用“ModelForm factory function”实现，如下。

示例代码：

```
from django import forms
from FormTest import models
```

```
RegisterForm= forms.modelform_factory(models.UserModel,fields=('email', 'password',
'age'))
```

那么，如果需要对表单控件进行修改，或者添加帮助提示如何处理呢？

示例代码：

```
from django import forms
from FormTest import models
```

```
RegisterForm= forms.modelform_factory(models.UserModel,
    fields=('email', 'password', 'age'),
    widgets={'password':forms.PasswordInput},
    help_texts={'email': '请输入有效的邮箱地址！'})
```

结合本篇教程前面的内容，大家可以很快掌握 ModelForm 工厂函数的使用方法。

本节练习源代码：[【点此下载】](#)

转载请注明：魔力 Python » Django2 : Web 项目开发入门笔记 (21)

原文链接：<http://www.opython.com/1154.html>

Django2 : Web 项目开发入门笔记 (22)

这一篇教程，我们一起来学习关于 Django 表单的最后一部分内容表单集 (FormSets) 和通过模型创建表单集 (Model FromSets)。

表单集 (FormSets) 实际上就是多个表单的集合，通过表单集可以重复创建相同元素的表单。

按照官方文档中的说法，可以把它比作数据网格。

例如，一个文章信息的表单，包含文章标题（title）和发布日期（pub_date）。

示例代码：

```
# forms.py
from django import forms

class ArticleForm(forms.Form):
    title = forms.CharField(label='标题')
    pub_date = forms.DateField(label='日期')

# views.py
from django.shortcuts import render
from FormTest.forms import ArticleForm

def formlist(request):
    if request.method == 'POST':
        pass
    form = ArticleForm()
    return render(request, 'formlist.html', {'form': form})

# urls.py
from django.urls import path
from FormTest import views as form_views

urlpatterns = [
    path("", form_views.formlist),
]

# formlist.html
<!DOCTYPE html>
<html lang="zh-cn">
<head>
    <meta charset="UTF-8">
    <title>文章信息</title>
</head>
<body>
<form action="/" method="post" novalidate="novalidate">
    {% csrf_token %}
    {{ form }} <br>
    <input type="submit" value="保存">
</form>
```



```
</body>
</html>
```

完成以上代码之后，启动开发服务器，浏览器中显示如下内容。



如果，我们想显示多个标题和日期的组合就可以使用 `FormSets`。

在上述代码中进行修改。

示例代码：

```
# forms.py 末尾添加（注意不要添加到 class 中）
ArticleFormSet = forms.formset_factory(ArticleForm, extra=5) # 参数 extra 指定表单数量
```

```
# views.py 红色为修改部分
from FormTest.forms import ArticleForm,
ArticleFormSet
```

```
def formlist(request):
    if request.method == 'POST':
        pass
```

```
formset = ArticleFormSet()
```

```
    return render(request, 'formlist.html',
        {'formset': formset})
```

```
# formlist.html 红色为新增部分
```

```
<form action="/" method="post" novalidate="novalidate">
```

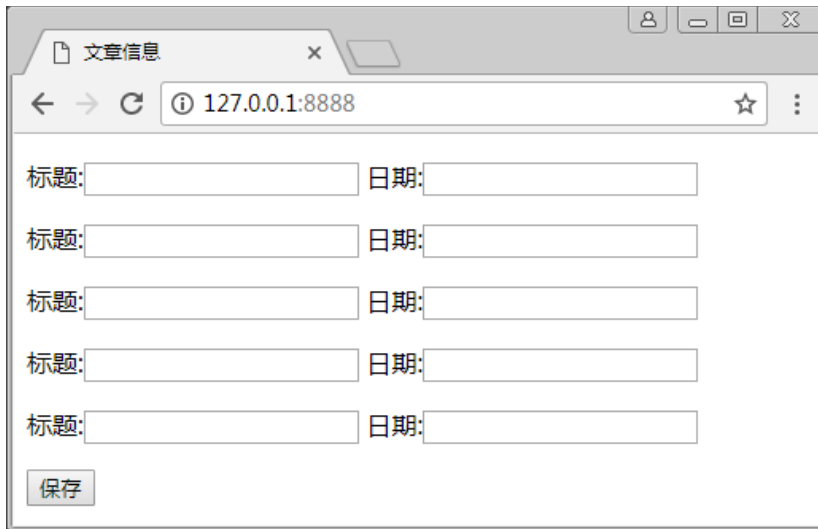
```
    {% for form in formset %}
```

```
<p> {{ form }} </p>
```

```
{% endfor %}
```

```
<input type="submit" value="保存">
```

当代码修改完毕后，浏览器中显示内容如下。



这里只做一个简单的举例，更多关于 **FormSets** 的内容大家可以参考官方文档。

接下来，我们学习如何通过模型创建表单集。

1、定义模型类并创建数据表，较之前无改动。（略）

2、定义表单集

这里我们先定义一个继承自 **BaseModelFormSet**，在构造方法中重写“**queryset**”，这是我们 从数据库中查询数据的结果，这个结果集决定页面上呈现的表单数量与内容。

然后，通过工厂函数“**modelformset_factory()**”定义表单集“**ArticleFormSet**”。

示例代码：

```
from django import forms
from django.forms import BaseModelFormSet
from FormTest.models import ArticleModel
```

```
class BaseArticleFormSet(BaseModelFormSet):
```

```
def __init__(self, *args, **kwargs):
    super().__init__(*args, **kwargs)
    self.queyset = ArticleModel.objects.all() # 默认查询请求
```

```
ArticleFormSet = forms.modelformset_factory(ArticleModel,
                                             fields='__all__',
                                             formset=BaseArticleFormSet,
                                             extra=2, # 额外的空表单数量
                                             max_num=6 # 最大表单数量
                                             )
```

3、定义视图函数

和表单类的用法一样，通过“request.POST”获取表单数据实例化为表单集对象，通过“save()”方法将表单内容存入数据库。

示例代码：

```
from django.shortcuts import render
from FormTest.forms import ArticleFormSet

def formlist(request):
    if request.method == 'POST':
        formset = ArticleFormSet(request.POST)
        if formset.is_valid():
            formset.save()
        formset = ArticleFormSet()
        return render(request, 'formlist.html', {'formset': formset})
```

如果想对数据进行处理，我们可以先不提交数据，对每个表单数据经过处理后，逐一通过“save()”方法保存到数据库。

示例代码：

```
instances = formset.save(commit=False)
for instance in instances:
    ...省略数据处理代码...
    instance.save()
```

4、URL 分发配置，较之前无改动。（略）

5、整合数据到模板

将视图函数传入模板的数据进行遍历，整合到页面内容中。

这里需要注意，使用表单集需要在遍历代码之前，添加标记“{{ 表单集名称.management_form }}”，否则会引发异常。

示例代码：

```
<!DOCTYPE html>
<html lang="zh-cn">
<head>
  <meta charset="UTF-8">
  <title>文章信息</title>
</head>
<body>
<form action="/" method="post" novalidate="novalidate">
  {% csrf_token %}
  {{ formset.management_form }}
  {% for form in formset %}
    <p> {{ form }} </p>
  {% endfor %}
  <input type="submit" value="保存">
</form>
</body>
</html>
```

完成上述代码之后，启动开发服务器，浏览器中就能够显示表单内容，并且可以输入保存表单数据。

文章信息

127.0.0.1:8888

标题: 小楼自传	日期: 2018-09-09
标题: 小楼如何成为了最帅的男人	日期: 2008-09-08
标题: 小楼这些年帅倒的迷妹	日期: 2018-09-07
标题: 如何成为像小楼一样帅的男人	日期: 2018-09-06
标题:	日期:
标题:	日期:

max_num=6

extra=2

保存

关于通过 Model 创建表单集的内容，在此就为大家分享这么多。

更多相关内容，请大家查阅官方文档。

本节练习源代码：【[点此下载](#)】

转载请注明：魔力 Python » Django2 : Web 项目开发入门笔记 (22)

原文链接：<http://www.opython.com/1158.html>

Django2 : Web 项目开发入门笔记 (23)

这一篇教程，我们一起来了解一些 Django 的 URL Name 和通用视图的内容。

一、URL Name

我们在做 URL 分发时，通过“`path()`”方法将一个路径匹配一个视图函数。

示例代码：

```
path('details<int:id>', shopviews.getgoodsdetails),
```

这样的路径，当我们访问类似“<http://127.0.0.1:8888/details1>”这样的网址时，就能够调用相应的视图进行处理。

不过，假如我们在完成了项目之后，想把这样的路径改为“<http://127.0.0.1:8888/details/1>”的格式，那就意味着所有使用了原有格式的路径都需要进行修改，这样的修改不能保证完全不会出现遗漏。

为了避免这种不安全的修改，我们在项目编写时，就可以主动去处理可能出现的问题。

在“`path()`”方法的参数中，可以再提供一个“`name`”参数的值，为“URL”设定一个名称。

然后，模板中使用“URL”时，不再写具体的路径，而是通过模板标记`{% url 'URL 名称' 更多参数 %}`自动产生相应的路径。

例如一个商品列表页面，点击某一商品时，能够进入该商品的详情页。

1、定义模型类并创建数据库表

大家可以参考《Django2 : Web 项目开发入门笔记 (8)》创建商品的模型类和数据库表。

2、定义视图函数

示例代码：

```
def getgoodslis(request):
    goodslist = GoodsInfo.objects.all()
    return render(request, 'goods_list.html', {'goodslist': goodslist})

def getgoodsdetails(request, id):
    goodsdetails = GoodsInfo.objects.filter(id=id)
    return render(request, 'goods_details.html', {'goodsdetails': goodsdetails})
```

3、配置 URL 分发

示例代码：

```
from MyShop import views as shopviews

path('', shopviews.getgoodslis),
path('details<int:id>', shopviews.getgoodsdetails,
name='details')
```

4、创建模板

示例代码：（goods_list.html）

```
<!DOCTYPE html>
<html lang="zh-CN">
<head>
    <meta charset="UTF-8">
    <title>商品列表</title>
</head>
<body>
<h3>商品列表</h3>
{% for goods in goodslist %}
    <p><a href="
{% url 'details' goods.id %}
">{{ goods.id }}.{{ goods.goods_name }}</a></p>
{% endfor %}
</body>
</html>
```

在上方代码中，模板标记“{% url ‘details’ goods.id %}”中的“‘details’”对应的就是 URL 分发配置中的 URL“details<int:id>”

参数“goods.id”对应的就是“<int:id>”。

示例代码：（goods_details.html）

```
<!DOCTYPE html>
<html lang="zh-CN">
<head>
    <meta charset="UTF-8">
    <title>商品详情</title>
</head>
<body>
{% for info in goodsdetails %}
    名称：{{ info.goods_name }} 数量：{{ info.goods_number }} 价格：{{ info.goods_price }}
{% endfor %}
</body>
</html>
```

完成上方代码中后，大家可以启动开发服务器，进行测试。

打开商品列表的页面后，点击商品名称就能够打开对应的商品详情页面。

此时，浏览器地址栏中的地址类似于“http://127.0.0.1:8888/details1”。

然后，大家可以把 URL 分发配置中的 URL 进行修改。

示例代码：

```
path('details/<int:id>', shopviews.getgoodsdetails, name='details'),
```

修改之后，刷新商品列表，并点击商品名称打开详情页面。

此时，浏览器地址栏中的地址类似于“http://127.0.0.1:8888/details/1”。

二、通用视图

1、TemplateView

按照我们之前所学内容，如果想让一个模板的内容呈现为页面，我们需要自定义一个视图函数。

哪怕只是一个没有数据加载的静态页面，也需要这么去做。

例如，我们有一个“hello.html”的模板。

示例代码：

```
<!DOCTYPE html>
<html lang="zh-CN">
<head>
    <meta charset="UTF-8">
```

```
<title>欢迎</title>
</head>
<body>
你好！欢迎访问小楼帅哥的个人网站！
</body>
</html>
```

这个模板内容与数据无关，呈现为页面之后，不会再有任何变化。

像这样的页面，实际上我们无需定义视图函数，可以直接使用 Django 给我们提供的通用模板视图类“TemplateView”。

在配置 URL 分发时，我们需要导入这个类，然后在“path()”方法中调用这个类的“as_view()”方法，并指定“as_view()”方法的“template_name”参数值为模板名称。

示例代码：

```
from django.views.generic.base import TemplateView

path('hello/', TemplateView.as_view(template_name='hello.html')),
```

这样结束之后，我们就可以启动开发服务器，访问“hello/”这个路径所打开的页面了。

2、ListView

“ListView”能够方便的帮助我们实现列表视图。

在‘views.py’文件中，我们不再定义视图函数，而是定义视图类继承“ListView”类。

示例代码：

```
from MyShop.models import GoodsInfo
from django.views.generic.list import ListView

class GoodsListView(ListView):
    model = GoodsInfo
    template_name = 'goods_list2.html'
```

在视图类中，我们只需要指定数据来源的模型类以及相应的模板文件。

然后，在 URL 分发配置中，我们添加新的配置。

示例代码：

```
path('list2/', shopviews.GoodsListView.as_view(),name='goods_list2'),
```

通过自定义视图类的“as_view()”方法，就能够将模型所有数据与模板进行整合。

最后，在视图中我们遍历数据对象，完成数据内容的填充。

示例代码：（goods_list2.html 的关键代码）

```
<h3>商品列表</h3>
{% for goods in
object_list

%}
  <p><a href="{% url 'details' goods.id %}">{{ goods.id }}.{{ goods.goods_name }}</a></p>
{% endfor %}
```

这里大家要注意，数据对象的名称为“object_list”，不再是自定义的名称。

上述操作是将全部商品数据在页面上呈现，如果我们只想呈现出符合某种条件的数据，该怎么做呢？

例如，我们只显示所有商品数据的前 5 条。

我们可以在自定义视图类中，重写“get_queryset()”方法。

示例代码：

```
# views.py
```

```
class GoodsList5View(ListView):
    template_name = 'goods_list3.html'

    def get_queryset(self):
        context = GoodsInfo.objects.all()[0:5]
        return context
```

```
#urls.py
```

```
path('list3/', shopviews.GoodsList5View.as_view(), name='goods_list3'),
```

```
#goods_list3.html
```

```
<h3>商品列表</h3>
```

```
{% for goods in object_list %}
<p><a href="{% url 'details' goods.id %}">{{ goods.id }}.{{ goods.goods_name }}</a></p>
{% endfor %}
```

3、DetailView

“DetailView”是通用详情视图。

我们可以通过继承“DetailView”类，完成商品详情的视图类。

然后，通过配置 URL 分发和创建相应的模板完成查看商品详情的功能。

示例代码：

```
#views.py
```

```
class GoodsDetailView(DetailView):
    model = GoodsInfo
    template_name = 'goods_details2.html'
```

```
#urls.py
```

```
path('goods_details/
<pk>
', shopviews.GoodsDetailView.as_view(), name='goods_details'),
```

```
#goods_details2.html
```

```
<body>
名称：{{ object.goods_name }} 数量：{{ object.goods_number }} 价格：
{{ object.goods_price }} 管理员：{{ manager }}
</body>
```

注意上方代码中，URL 分发配置中的 URL 参数要使用“<pk>”，也就是“Primary Key”（主键）的意思。

如果不是通过主键查询可以写成<slug:参数名称>。

在《Django2：Web 项目开发入门笔记（4）》中曾经提到，slug 类型是由字母、数字、横线以及下划线其中一种或多种组成的字符串。

我们除了能够查询到某个商品的详情，还可以在模型数据的基础上增加一些内容。

例如，把每个商品详情的末尾加上“管理员：小楼”。

示例代码：

#views.py

```
class GoodsDetailView(DetailView):
    model = GoodsInfo
    template_name = 'goods_details2.html'

    def get_context_data(self, **kwargs):
        context = super().get_context_data(**kwargs)

        context['manager'] = '小楼'

        return context

# goods_details2.html

<body>
名称：{{ object.goods_name }} 数量：{{ object.goods_number }} 价格：
{{ object.goods_price }}

管理员：{{ manager }}

</body>
```

以上代码中，红色部分为新增内容。

4、RedirectView

“RedirectView”是重定向视图。

例如，当用户访问“http://127.0.0.1:8888/django2”时，页面跳转到“http://www.opython.com/django2”。

示例代码：

```
from django.views.generic.base import RedirectView  
  
path('django2/', RedirectView.as_view(url='http://www.opython.com/django2/'), name='django2'),
```

这个视图我们也可以通过创建子类重写某些内容，实现更多的重定向功能。

最后，关于通用视图还有很多内容，大家可以参考官方文档相关内容：

- Use generic views: Less code is better
- Built-in class-based views API

本节练习源代码：【[点此下载](#)】

转载请注明：魔力 Python » Django2：Web 项目开发入门笔记（23）

原文链接：<http://www.opython.com/1171.html>

Django2：Web 项目开发入门笔记（24）

这一篇教程，我们一起来了解 Django 的上下文处理器（Context Processor）和中间件（Middleware）。

对于这两块内容，我们最好先了解概念和原理，才能比较好的去应用它们。

一、上下文处理器（Context Processor）

上下文处理器也称作上下文渲染器。

我个人认为处理比渲染更容易理解，而且英文中“Processor”的意思是处理，“Render”的意思是渲染。

但是称为上下文渲染器也有道理，渲染更接近于使用场景。

Django 中的 Context Processor 主要是应用于模板，完成页面的绘制的一些处理，也就是所说的页面渲染。

还有，上下文又是什么？

这个概念，感觉很难懂。

从语文的角度来说，上下文是语境；

同一句话，在不同的语境中，会有不同的意思。

例如，小明考试成绩 100 分，爸爸说“干得漂亮”，这是表扬的意思。

而小明考试成绩 0 分，爸爸说“干得漂亮”，这是讽刺的意思。

所以，同样的一句话，我们需要和上文联系到一起才能明白真正的意思。

从计算机编程角度来说，上下文是环境。

也就是说，同一段处理程序对于不同的环境，反馈出不同的处理结果。

借用网上的一个例子：用户访问站点的时候，站点的所有页面上都要能够显示这个用户自己的 IP 地址。

这样的功能，我们需要从请求（request）中获取到访问用户的 IP 地址，然后呈现到页面中。

也许大家能够想到，我们可以在每个页面对应的视图函数中进行这个处理，但是未免太过麻烦。

最好的方式是将处理过程定义一次，就能够在每个页面中使用。

此时，我们可以通过自定义一个上下文处理器帮助我们完成。

面对不同来源的用户通过同一个处理器完成页面上不同 IP 地址的渲染。

1、创建上下文处理器

在项目文件夹（“wsgi.py”所在的文件夹）新建一个文件“context_processor.py”。

在这个文件中添加获取用户 IP 地址的代码。

示例代码：

```
def getuserip(request):
    ip = request.META['REMOTE_ADDR']
    if ip == '127.0.0.1':
        return {'user_type': '本机用户'}
    return {'user_type': '外部用户(%s)' % ip}
```

注意：上下文处理器必须返回一个字典。

2、将上下文处理器添加到模板设置

打开文件“settings.py”，在模板设置中添加处理器中的函数。

```
TEMPLATES = [
    {
        ...省略部分代码...
        'OPTIONS': {
```

```
'context_processors': [  
    ...省略部分代码...  
  
    'MyProject.context_processor.getuserip'  
  
    ],  
},  
},  
]
```

上方代码中，红色部分为新增代码。

3、在页面模板中添加标记

```
<!DOCTYPE html>  
<html lang="zh-CN">  
<head>  
    <meta charset="UTF-8">  
    <title>首页</title>  
</head>  
<body>  
    当前访问用户 :  
  
    {{ user_type }}
```

```
</body>  
</html>
```

完成上述代码后，Django 调用模板时会先通过上下文处理器进行处理，并将处理后返回的数据字典传入模板，通过模板标记获取数据。

此时，我们就可以打开开发服务器，进行访问测试了。

注意：如果需要允许外部访问，需要监听所有 IP 地址，服务器启动命令为：`runserver 0.0.0.0:端口号`，也可以简写为 `runserver 0:端口号`。

本机访问结果：



外部访问结果：（例如手机）



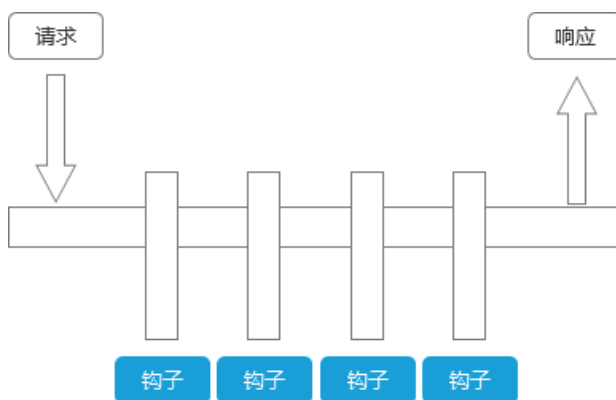
二、中间件（Middleware）

中间件，我们需要搞清楚它在什么中间，起到什么作用？

根据官方文档的说法，中间件是请求（request）与响应（response）之间的钩子（hooks）框架。

...又多了个钩子。

个人理解钩子实际上就是挂在主程序上的监控程序，当主程序运行中满足了钩子的触发条件时，执行钩子的处理程序。



在官方文档中，给出的中间件示例非常简单直观，并且包含了两种写法。

1、以函数的形式定义中间件。

示例代码：

```
def  
simple_  
middleware  
get_response
```

```
# One-time configuration and initialization.
```

```
def
```

```
    middleware
```

```
    request
```

```
# Code to be executed for each request before
```

```
# the view (and later
```

```
    middleware
```

```
) are called.
```

```
    response
```

```
    get_response
```

```
    request
```

```
# Code to be executed for each request/response after
```

```
# the view is called.
```



```
return
```

```
response
```

```
return
```

```
middleware
```

2、以类的形式定义中间件

```
class
```

```
Simple
```

```
Middleware
```

```
def
```

```
__init__
```

```
self
```

```
get_response
```

```
self
```

```
get_response
```

```
get_response
```

```
# One-time configuration and initialization.
```

```
def
```

```
_call_
```

```
self
```

```
request
```

```
# Code to be executed for each request before
```

```
# the view (and later
```

```
middleware
```

```
) are called.
```

```
response
```

```
self
```

```
get_response
```

```
request
```

```
# Code to be executed for each request/response after
```

```
# the view is called.
```

```
return
```

```
response
```

那么，中间件怎么使用？能起到什么样的作用呢？

依然通过举例说明。

例如，我们发布的项目只允许本机和指定 ip 访问，不允许外部其他地址的设备访问，就可以通过中间件来实现。

首先，创建一个中间件文件“middleware.py”，并定义访问限制的中间件。

这个文件可以创建一个新的 Package 进行存放，例如：MW。

示例代码：

```
from django.http import HttpResponseForbidden

class AaccessRestrictionMiddleware:

    def __init__(self, get_response):
        self.get_response = get_response
        self.wite_ip = ['127.0.0.1', '192.168.31.18'] # 初始化 ip 地址白名单

    def __call__(self, request):
        ip = request.META['REMOTE_ADDR'] # 获取访问用户的 ip
        if ip not in self.wite_ip: # 如果 ip 不在白名单中
            return HttpResponseForbidden('您被禁止访问！') # 返回响应
        response = self.get_response(request)
        return response
```

然后，我们打开文件“settings.py”，在中间件设置中添加中间件的类。

示例代码：

```
MIDDLEWARE = [
    'MW.middleware.AaccessRestrictionMiddleware',
    ...省略其他中间件...
]
```

通过以上操作，我们就可以限制只有 IP 白名单中的用户才能够访问我们的项目。

中间件比较像游戏的外挂程序，游戏主程序的执行过程中一旦满足触发外挂的情形出现，外挂就会执行相应的处理。但是，外挂并不会改变主程序，当我们不需要外挂的时候，只需要移除外挂程序，主程序依然能够正常的执行。

本节练习源代码：[【点此下载】](#)

转载请注明：魔力 Python » Django2 : Web 项目开发入门笔记 (24)

原文链接：<http://www.opython.com/1173.html>

Django2 : Web 项目开发入门笔记 (25)

这一篇教程，我们一起来了解 Cookie 与 Session。

Cookie 是由 Web 服务器保存在用户浏览器上的小文本文件，它可以包含有关用户的信息。

当 Web 服务器创建了 Cookies 后，只要在其有效期内（有效期可以由开发人员设定），当用户访问同一个 Web 服务器时，浏览器首先要检查本地的 Cookies，并将其原样发送给 Web 服务器。

这种状态信息称作“Persistent Client State HTTP Cookie”，简称为 Cookies。

以上是引用百度百科中对“Cookie”的描述。

对于“Cookie”的使用，每个人都应该体验过。

例如，我们成功登录百度网站，如果不主动退出，即便是隔一天再打开浏览器进入百度网站，依然能够保持登录状态。

这就是“Cookie”在起作用，它通过将一些信息保存在客户端，解决 HTTP 的会话保持问题。

提示：HTTP 是无状态的，所以无法保持会话状态。保持会话状态是指将同一用户多次进行 HTTP 连接所发出的不同请求形成关联。

在我们登录百度网站时，服务器创建了“Cookie”文件发送给浏览器进行保存。

当我们再次打开浏览器访问百度网站时，浏览器会先从本地保存的“Cookie”文件中查找与百度网站相匹配的“Cookie”文件，如果存在并有效则回送给服务器，服务器进行相应的验证之后，根据验证结果返回相应的状态到浏览器。

接下来，我们就使用“Cookie”完成一个保持登录状态功能。

1、创建一个简单的模板，添加一个表单。

表单中包含两个状态下的元素。

已登录状态包含一句问候语和一个退出链接。

未登录状态包含一个文本输入框和一个按钮。

示例代码：

```
<!DOCTYPE html>
<html lang="zh-CN">
<head>
  <meta charset="UTF-8">
  <title>首页</title>
</head>
<body>
<form action="/" method="post">
  {% csrf_token %}
  {% if username %}
    您好, {{ username }} !
    <a href="exit/">退出</a>
  {% else %}
    <input type="text" name="username">
    <input type="submit" value="登录">
  {% endif %}
</form>
</body>
</html>
```

通过这个模板，我们能够看出，页面呈现什么样的状态，取决于`{{ username }}`这个变量中是否存在数据内容。

2、添加 URL 分发配置

示例代码：

```
path('', siteviews.index),
path('exit/', siteviews.exit),
```

3、定义视图函数

在视图函数中，我们要对“Cookie”进行设置与读取的操作。

设置“Cookie”比较常用的参数包括：

- `key`：键
- `value`：值
- `max_age`：多少秒后过期
- `expires`：什么时间过期
- `path`：指定哪些 URL 可以访问当前“Cookie”文件（默认为“/”，表示所有）
- `domain`：指定哪些域名可以访问当前“Cookie”文件（默认为“None”，表示当前域名）
- `secure`：默认值为“False”（通过 HTTPS 传输时需设置为“True”）

- `httponly` : 默认值为“False”（如果设置为“True”，则只能通过 HTTP 传输，JS 无法获取和修改）

示例代码：

```
from django.shortcuts import render, HttpResponseRedirect

def index(request):
    if request.COOKIE.get('username', None): # 读取 Cookie 信息
        return render(request, 'index.html', {'username': request.COOKIE['username']}) # 读
    取 Cookie 信息
    if request.method == 'POST':
        username=request.POST['username']
        res =render(request,'index.html',{'username':username})
        res.set_cookie('username',username,30) # 设置 Cookie 信息
        return res
    else:
        return render(request,'index.html')

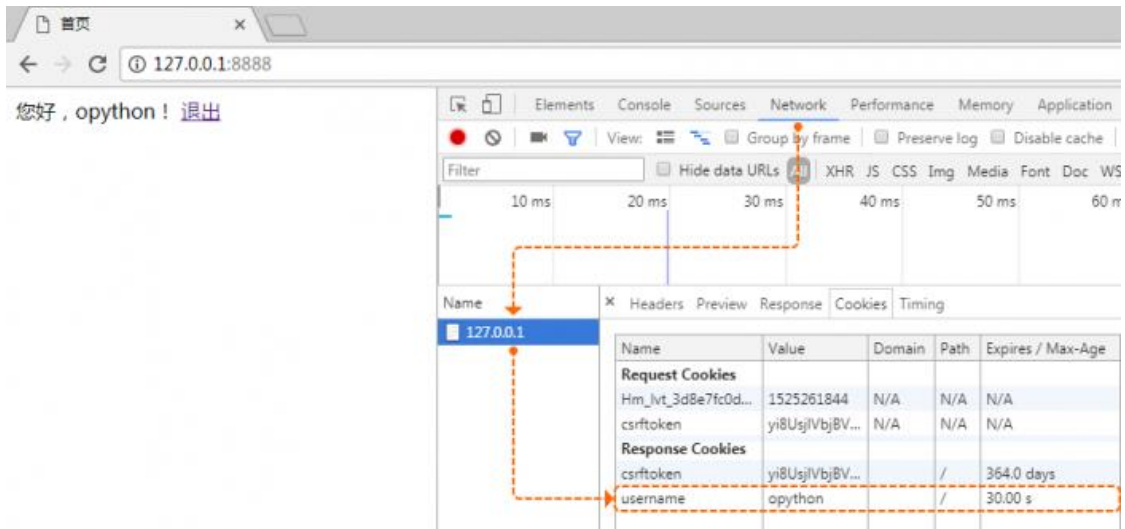
def exit(request):
    res = HttpResponseRedirect('/')
    res.delete_cookie('username') # 删除 Cookie 信息
    return res
```

完成上述代码后，大家运行开发服务器，在打开的页面中输入一个用户名，点击登录按钮，就能够切换为登录后的状态。

并且，在 30 秒内刷新当前页面，会持续保持登录状态。

而超过 30 秒之后，又会恢复未登录状态。

大家可以通过浏览器的开发者工具（F12），查看“Cookie”文件的内容。



在上图中，暴露了一个严重的问题，就是“Cookie”的内容是直接可见的。

所以，切记不要在“Cookie”中存储重要或者敏感的用户信息。

这样直接暴露的“Cookie”信息，非常容易被攻击者盗取并篡改。

但是，我们又需要通过“Cookie”持续保持会话状态。

有一种解决方法是“加盐”和“解盐”，也就是对“Cookie”信息设置时进行加密，读取时进行解密，以防止攻击者使用篡改过的信息欺骗服务器。

我们对之前的视图函数进行修改。

示例代码：

```
def index(request):
    if request.COOKIE.get('username', None):
        # return render(request, 'index.html', {'username': request.COOKIE['username']})

    return render(request, 'index.html', {'username': request.get_signed_cookie('username',
salt='用于加密的字符串')})

if request.method == 'POST':
    username = request.POST['username']
    res = render(request, 'index.html', {'username': username})
    # res.set_cookie('username', username, 30)

    res.set_signed_cookie('username', username, salt='用于加密的字符串')
```

```
    return res
else:
    return render(request, 'index.html')
```

上方代码中，注释的内容为修改前的语句，红色内容为修改后的语句。

当这样修改之后，在浏览器中查看到的“Cookie”文件信息，就包含了一段安全密钥。

Response Cookies				
csrftoken	yi8UsjlVbj8Vkc5UjF6P8w9MK4suW5trwmHAW9Vdld...		/	364.0 days
username	opython:1fE6qO:bqUgaEpQ2TAQr9TtzExZOxHfNzE		/	Session

当我们访问网站时，浏览器会将包含了密钥的“Cookie”信息回送到服务器，服务器对密钥进行解密验证。

这种解决方法虽然提高了安全性，但是用户的信息仍然是暴露的。

目前，最好的解决方式是通过“Cookie”结合“Session”来保持会话状态。

在“Cookie”文件中，只保存一个名为“sessionid”的随机字符串。

而与“sessionid”对应的用户信息全部保存在服务器端。

Django 中提供了 5 种类型的“Session”：

- 数据库（默认）
- 缓存
- 文件
- 缓存+数据库
- 加密 cookie

这里只为大家介绍默认的基于数据库的“Session”。

使用这种类型的“Session”，需要先完成数据库的设置，并通过“makemigrations”和“migrate”命令完成相关数据表的创建，

然后，就能够在项目中使用“Session”了。

示例代码：

```
def index(request):
    if request.session.get('username', None):
        return render(request, 'index.html', {'username': request.session['username']})
    if request.method == 'POST':
        username = request.POST['username']
```



```
request.session['username'] = username
return render(request, 'index.html', {'username': username})
else:
    return render(request, 'index.html')
```

```
def exit(request):
    del request.session['username']
    return HttpResponseRedirect('/')
```

通过上方代码，我们就完成和之前代码同样的功能。

此时，在浏览器中关于“Cookie”的信息，就不再有用户的信息，只有“sessionid”。

Response Cookies				
csrftoken	yi8UslVbjBVkc5UjF6P8w9MK4suW5trwmHAw9Vdld...	/		364.0 days
sessionid	7ohp39xdof9fxspbz7krt74vf7mvx3jr	/		14.0 days

在上图中，大家能够看到，默认“Session”的有效时间为 14 天，如果需要修改，我们可以在“settings.py”中进行修改。

示例代码：

```
SESSION_ENGINE = 'django.contrib.sessions.backends.db' # Session 的引擎（默认）
SESSION_COOKIE_NAME = 'mysession' # Session 的 Cookie 的 Key
SESSION_COOKIE_PATH = "/" # Session 的 Cookie 的可访问路径（默认）
SESSION_COOKIE_DOMAIN = None # Session 的 cookie 保存的域名（默认）
SESSION_COOKIE_SECURE = False # 是否 HTTPS 传输 Cookie（默认）
SESSION_COOKIE_HTTPONLY = True # 是否 Session 的 Cookie 只支持 HTTP 传输（默认）
SESSION_COOKIE_AGE = 604800 # Session 的 Cookie 的有效时长（秒）（默认 1209600）
SESSION_EXPIRE_AT_BROWSER_CLOSE = False # 是否关闭浏览器使得 Session 过期（默认）
SESSION_SAVE_EVERY_REQUEST = False # 是否每次请求都保存 Session（默认）
```

修改之后，在浏览器中能够看到信息的变化。

Response Cookies				
csrftoken	yi8UslVbjBVkc5UjF6P8w9MK4suW5trwmHAw9Vdld...	/		364.0 days
mysession	et11f7gmhmhcogz8t8toptim6w8kkpg6n	/		7.0 days

本节练习源代码：[【点此下载】](#)

转载请注明：魔力 Python » Django2：Web 项目开发入门笔记（25）

原文链接：<http://www.opython.com/1179.html>

Django2 : Web 项目开发入门笔记（26）

这一篇教程，我们一起来学习在 Django2 中使用 SummerNote 富文本编辑器。

这款编辑器基于 Bootstrap 和 JQuery，也就是说项目中要先准备好 Bootstrap 和 JQuery 相关文件，当然也可以在线调用。

相对于 CKEditor 我更喜欢 SummerNote，因为它样式很漂亮，而且使用也很简单，图片上传不用再自己编写代码。

提示：本教程基于 Django 项目，请先完成项目与应用的创建。

一、安装 SummerNote 库

通过“pip”命令就能够安装 SummerNote 库。

```
pip install django-summernote
```

二、添加静态文件

从 SummerNote 官网下载已编译的文件（Compiled）。

下载地址：<https://summernote.org/getting-started/#compiled-css-js>

下载之后，解压缩，将“dist”文件夹中的文件复制到项目根目录的“static/summernote”文件夹（自己创建）中。

三、添加设置

打开文件“settings.py”，添加相关设置。

示例代码：

DEBUG = False # 编辑器的粗体、斜体等功能，在执行“collectstatic”命令之前需要关闭 DEBUG 模式才能正常使用。

ALLOWED_HOSTS = ['*'] # DEBUG 模式关闭时，必须设置此项，生产环境中应为域名。

```
INSTALLED_APPS = [  
    ...省略部分代码...  
    'django_summernote',  
]  
STATIC_URL = '/static/'  
STATIC_ROOT = os.path.join(BASE_DIR, 'static')
```

```
MEDIA_URL = '/media/'
```

`MEDIA_ROOT = os.path.join(BASE_DIR, 'media')` # 官方文档为“media/”，实际测试中导致图片上传不显示。

提示：“MEDIA_ROOT”设置中，官方文档为“media/”是因为面向 Django1.11，因为没有 Django1.11 未做该环境中的验证。

四、添加 URL 配置

示例代码：（urls.py）

```
from django.contrib import admin
from django.urls import path, include
from summer import settings
from django.views.static import serve
from summerapp import views as view
```

```
urlpatterns = [
    path('summernote/', include('django_summernote.urls')),
    path('media/<path:path>', serve, {'document_root': settings.MEDIA_ROOT}), # 用于上传
    # 图片文件
    path('static/<path:path>', serve, {'document_root': settings.STATIC_ROOT}), # 用于加载
    # 静态文件
    path('admin/', admin.site.urls),
]
```

五、创建数据库表

SummerNote 图片上传需要在数据库中创建数据表“django_summernote_attachment”，执行“migrate”命令即可自动创建。

注意：先根据项目使用的数据库完成数据库的创建与配置。python manage.py migrate

到这里，我们就可以在 Django 后台中使用这个文本编辑器了。

六、测试

1、创建模型

示例代码：（models.py）

```
from django.db import models
from django_summernote.fields import SummernoteTextField
```

```
class SummerNoteTest(models.Model):  
    TextField = SummernoteTextField()
```

2、注册模型

示例代码：（admin.py）

```
from django.contrib import admin  
from .models import SummerNoteTest
```

```
admin.site.register(SummerNoteTest)
```

另外，如果想让一个模型的所有 `TextField` 字段都使用 `SummerNote`，可以在“admin.py”文件中进行设置。

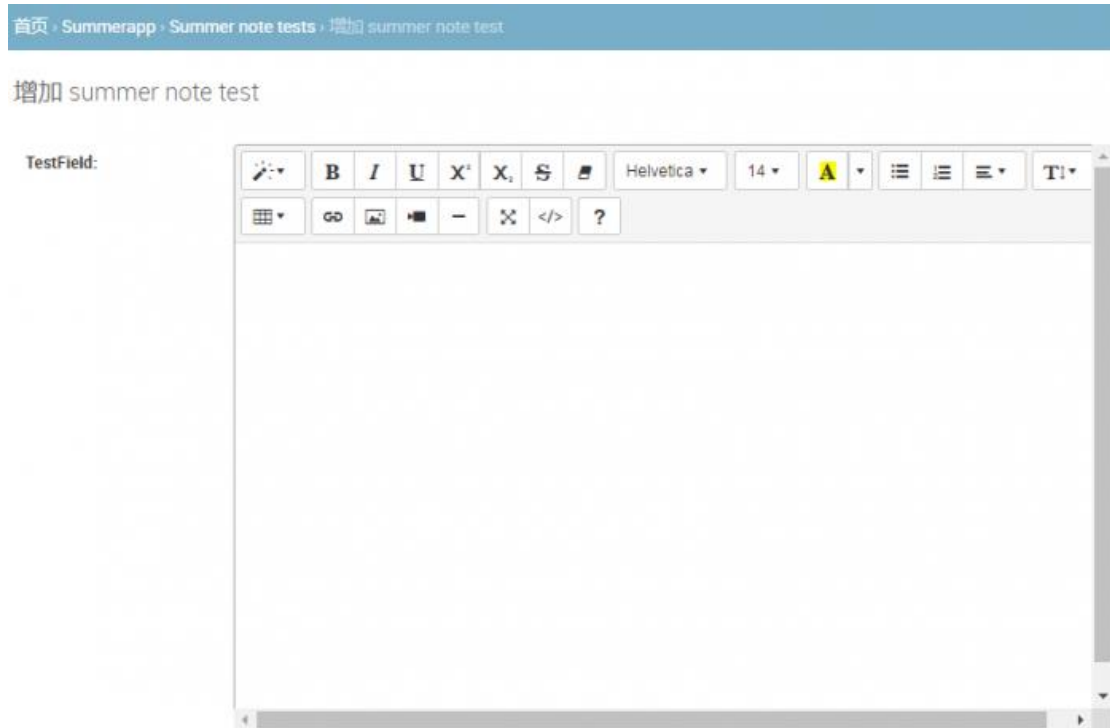
例如：

```
from django.contrib import admin  
from django_summernote.admin import SummernoteModelAdmin  
from .models import SummerNoteTest
```

```
class SummerNoteTestAdmin(SummernoteModelAdmin): # instead of ModelAdmin  
    summernote_fields = '__all__'
```

```
admin.site.register( SummerNoteTest,SummerNoteTestAdmin)
```

完成以上步骤之后，已经能够在 Django 后台编辑时使用富文本编辑器了。



不过，这个时候编辑器还是默认的设置，我们可以通过添加设置项进行更改。

七：设置

示例代码：（settings.py）

```
SUMMERNOTE_CONFIG = {
    # 是否使用 IFrame 的方式，不使用的必须加载 Bootstrap/jQuery。
    'iframe': True,
    # 以下为 SummerNote 自定义设置
    'summernote': {
        # 隐身模式
        'airMode': False,
        # 编辑器的尺寸
        'width': '75%',
        'height': '360',
        # 语言设置
        'lang': 'zh-CN',
        # 设置字体列表
        'fontNames':
            [
                'Arial',
                'Arial Black',
                'Comic Sans MS',
```

```
'Courier New',
'宋体', # 也可以写为“Songti”
'Microsoft YaHei' # 也可以写为“微软雅黑”
]
},
# 附加样式表
'css': (
    '/static/summernote/theme/paper.css', # 附加编辑器主题
    # '/static/summernote/theme/flatly.css', # 多个主题时默认加载最后一个，建议注释不
    # 使用的主题。
),
}
```

提示：主题文件需要下载后放入“/static/summernote/theme”文件夹（自己创建）中。

编辑器主题的下载地址：

-
-
-
-
-
-
-
-
-
-
-
-
-
-
-
-
-
-

编辑器主题效果预览：<https://summernote.org/examples/#themes-with-bootswatch>

以上只是部分设置内容，更多设置请参考：

<https://github.com/summernote/django-summernote>

https://github.com/summernote/django-summernote/blob/master/django_summernote/settings.py#L106-L133

八、收集静态文件

执行“python manage.py collectstatic”命令,让 Django 自动收集 Admin 以及 APP 中的静态文件到根目录的“static”文件夹下。

此时, 不管“setting.py”文件中的“DEBUG”设置为“True”还是“False”, 编辑器都能正常使用了。

第二部分：前台页面使用 SummerNote

一、添加静态文件

如果只是在前台页面使用 SummerNote, 无需安装 django-summernote。

下载已编译的 SummerNote 文件, 以及需要的主题文件, 添加到项目根目录的“static”文件夹中。

二、载入静态文件

在“settings.py”和“urls.py”文件中, 先完成“static”的相关设置 (参考第一部分内容)。

然后, 在模板文件 (例如命名为“summer_note.html”) 的<head>...</head>标签中载入静态文件。

示例代码：(summer_note.html)

```
<!DOCTYPE html>
<html lang="zh-CN">
<head>
  <meta charset="UTF-8">
  <title>编辑器测试</title>

  {% load static %}

  <link rel="stylesheet" href="{% static '/css/bootstrap.css' %}">

  <script src="{% static '/js/jquery-3.3.1.min.js' %}"></script>

  <script src="{% static '/js/bootstrap.js' %}"></script>

  <link rel="stylesheet" href="{% static '/summernote/summernote.css' %}">

  <script src="{% static '/summernote/summernote.js' %}"></script>

  <script src="{% static '/summernote/lang/summernote-zh-CN.js' %}"></script><!--加载
中文-->
```

```
<link rel="stylesheet" href="{% static '/summernote/theme/paper.css' %}"><!--加载主题-->
```

```
</head>
<body>
    ...后续代码在此处添加...
</body>
</html>
```

三、添加编辑器

添加编辑器有两种方式，通过<div>标签或者<textarea>都可以添加。

示例代码：

```
<body>

...此处添加后续代码...

<!--第 1 种添加方式-->
<div id="summernote">www.opython.com 原创教程</div>

<!--第 2 种添加方式：使用时取消注释-->
{#<form method="post">#}
{#  <textarea id="summernote" name="editordata"></textarea>#}{#</form>#}

</body>
```

四、设置

设置内容需要写在 JavaScript 脚本中。

编辑器默认功能不够完整，可以通过设置进行添加修改。

示例代码：

```
<script>
$(document).ready(function () {
    $('#summernote').summernote({#编辑器设置#}
    {
        width: 750,
        height: 360,
        focus: true, {#页面打开时光标自动进入编辑区#}
        lang: 'zh-CN',
    }
    });
});
```



```

toolbar: [{#工具栏加载项设置#}
    ['style', ['style']], {#文本样式#}
    ['font', ['bold', 'italic', 'underline', 'superscript', 'subscript',
        'strikethrough', 'clear']], {#字体样式#}
    ['fontname', ['fontname']], {#字体列表#}
    ['fontsize', ['fontsize']], {#字体尺寸#}
    ['color', ['color']], {#字体颜色#}
    ['para', ['ul', 'ol', 'paragraph']], {#文本列表#}
    ['height', ['height']], {#行高#}
    ['table', ['table']], {#表格#}
    ['insert', ['link', 'picture', 'video', 'hr']], {#插入项#}
    ['view', ['fullscreen', 'codeview']], {#视图#}
    ['help', ['help']] {#帮助#}
],
fontNames:
[
    'Arial',
    'Arial Black',
    'Comic Sans MS',
    'Courier New',
    '宋体',
    'Microsoft YaHei'
]
}
);
});
</script>

```

五、测试

1、添加 URL 配置

示例代码：

```
path("", view.index),
```

2、定义视图函数

示例代码：

```
from django.shortcuts import render
```

```
def index(request):
    return render(request, 'summer_note.html')
```

启动开发服务器，打开链接“[http://localhost:\[自定义端口号\]](http://localhost:[自定义端口号])”就能够看到编辑器了。

项目源代码下载：[【点此下载】](#)

转载请注明：魔力 Python » Django2：Web 项目开发入门笔记（26）

原文链接：<http://www.opython.com/1310.html>

Django2 练习项目：开发个人博客系统（1）

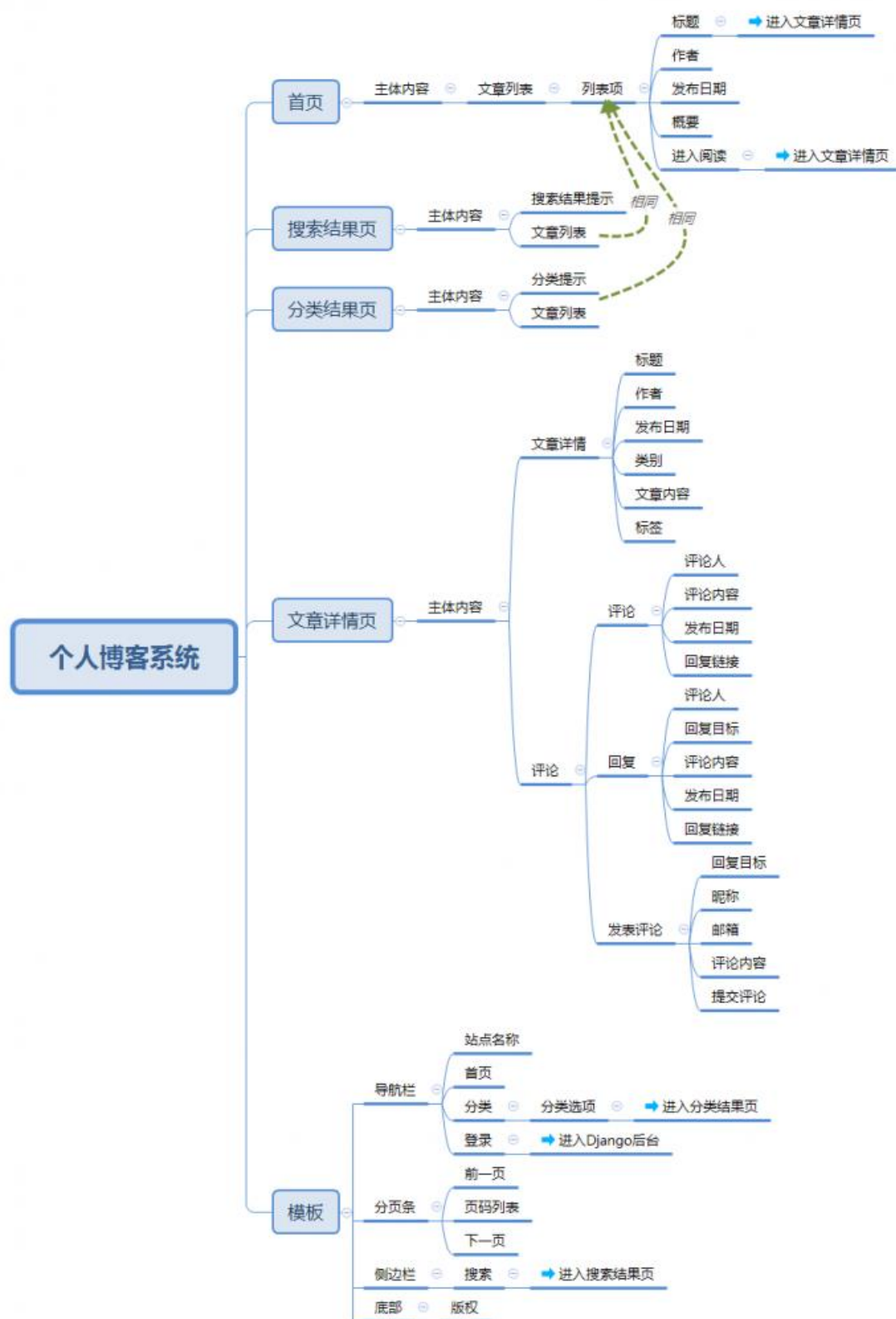
从这一篇教程开始，我们一起结合以往学过的知识内容，完成一个简单的基于 Django2 的个人博客系统。

当然，在开发的过程中，仍然会有更多的新鲜知识点的融入，让我们掌握更多的知识内容。

这里要开发的个人博客系统功能比较简单，主要包括文章、分类、标签、评论、搜索、分页以及侧边栏的实现。

还有就是，结合 **Bootstrap** 这个前端框架，让博客系统的界面美观，并且完成导航栏和页面底部内容。

按照教程的目标，我们的个人博客系统结构如下（使用 **Xmind** 构建）：



有了这张图作指导，我们就可以开始构建我们的博客系统了。

整体上我们需要完成数据模型、视图、URL 分发和模板。

这一篇教程，我们先完成数据模型。

首先，需要安装好 MySQL，并且在项目的“settings”文件中完成数据库配置。

提示：MySQL 的安装可以参考《MySQL5.7 版简易安装教程》。

示例代码：

```
DATABASES = {
    'default': {
        'ENGINE': 'django.db.backends.mysql', # 数据库引擎
        'NAME': 'myblog', # 数据库名称
        'HOST': '127.0.0.1', # 主机地址
        'PORT': '3306', # 主机端口
        'USER': 'root', # 数据库用户名
        'PASSWORD': 'Opython.com666', # 数据库密码
    }
}
```

然后，将 Web 应用也添加到“settings”文件的配置中。

示例代码：

```
INSTALLED_APPS = [
    ...省略部分代码...
    'blog.apps.BlogConfig',
]
```

这里大家能够看到，并不像以前直接添加应用的名称到配置中，而是添加了应用的配置类。

很明显这个配置类是在应用包的“apps.py”文件中。

示例代码：

```
from django.apps import AppConfig

class BlogConfig(AppConfig):
    name = 'blog' # 应用名称
    verbose_name = '我的博客' # Web 站点名称
```

最后，我们就可以着手创建数据模型。

数据模型的创建，需要仔细分析。

1、文章

一篇文章（对象）一般包括以下元素（特性）：

- 文章编号：唯一的数字。
- 文章标题：唯一的字符串，并且需要限制一定的长度。
- 文章作者：字符串，关联到用户。
- 发布时间：日期格式，本项目精确到哪一日。
- 文章内容：长文本。
- 文章标签：关联到标签，可具有多个标签，标签删除时，文章不受影响。
- 文章类别：关联到类别，但仅限一个类别，类别删除时，文章为未分类。

那么，基于文章的这些元素（对象的特性），我们就可以创建文章的模型类，也就是文章的数据模型。

这里要注意存在的关联关系。

- 文章作者：一篇文章对应一名作者，而一名作者可以发布多篇文章，这是多对一的关系。
- 文章标签：一篇文章可以有多个标签，而一个标签也可以对应多篇文章，这是多对多的关系。
- 文章类别：一篇文章对应一个类别，而一个类别可以对应多篇文章，这也是多对一的关系。

当有类似上述关系的存在时，就会涉及到一个数据模型与另外一个数据模型产生关联关系。

也就是说用户、文章类别和文章标签也需要有相应的模型类。

我们先不管这三个模型类的内容，但是可以先将这些类创建出来。

示例代码：

```
from django.db import models

class Category(models.Model): # 文章类别
    pass

class Tag(models.Model): # 文章标签
    pass

class Article(models.Model): # 文章
    pass
```

完成了类的创建，接下来我们对根据文章包含的元素，添加类的特性。

示例代码：

```
from django.contrib.auth.models import User # 使用 Django 自带的用户模型

class Article(models.Model): # 文章
    id = models.AutoField(primary_key=True)
    author = models.ForeignKey(User, on_delete=models.DO_NOTHING, verbose_name='作者')
    title = models.CharField('标题', max_length=50)
    content = models.TextField('内容')
    pub_time = models.DateField('日期', auto_now=True)
    category = models.ForeignKey(Category, on_delete=models.SET_DEFAULT, default=1,
    verbose_name='类别')
    tag = models.ManyToManyField(Tag, verbose_name='标签')

    class Meta:
        verbose_name_plural = verbose_name = '文章'

    def __str__(self):
        return self.title
```

添加的特性中有些是我们之前从未接触的内容，这里给大家做一下详细解释。

- **id**：文章编号，这个字段应该随着每一篇文章的发布，自动产生唯一的编号，所以这里使用“AutoField”，即自增长字段；同时，文章的编号具有唯一性，非常适合作为文章数据表的主键，所以在字段的参数中添加“primary_key=True”，指定这个字段为主键。
- **author**：文章作者，对应一个系统用户；这里我们不单独创建用户模型，而是使用 Django 自带的用户模型（注意导入）；因为文章和用户存在关联关系，这个关系通过外键字段“ForeignKey”建立，即文章从属于用户；因为这个从属关系，我们需要明确，当用户删除时，对文章如何处理，所以，在外键字段的参数中，第 1 个位置参数指定和哪一个模型存在关联关系，第 2 个关键字参数则是指定当外键指向的数据对象被删除“on_delete”时，如何处理，这里的值为“models.DO_NOTHING”，即不做任何处理；最后一个关键字参数“verbose_name”是在 Django 后台中显示的字段名称。
- **title**：文章标题，使用文本字段“CharField”；第 1 个位置参数是 Django 后台中显示的字段名称；第 2 个参数“max_length”指定最大字符数量，不可省略。
- **content**：文章内容，使用文本字段“TextField”；“TextField”字段不限定长度。
- **pub_time**：发布日期，使用日期字段“DateField”；第 2 个关键字参数“auto_now”表示是否自动使用当前日期。
- **category**：文章类别，使用外键字段；第 1 个位置参数指定文章类别的模型类；当某个文章类别被删除时，使用了这个类别的文章需要将类别改为默认的“未分类”类别，

所以，第 2 个关键字参数“on_delete”的值为“models.SET_DEFAULT”，即文章类别被删除时使用默认值；第 3 个关键字参数需要指定文章类别中默认类别的主键，这里的值为“1”，表示在最终创建好的文章类别数据表中需要有一个编号为“1”，名称为“未分类”的数据行。

- **tag**：文章标签，因为文章与文章标签为多对多的关系，所以需要使用多对多字段“ManyToManyField”；数据库的数据表创建之后，就会额外出现一个多对多的关系表，这个表中每一行都会包含文章和标签的主键，表示它们之间的关系。

除了特性，我们还可以通过内嵌类“Meta”和“__str__”方法对 Django 后台中显示的内容进行一些设定。

这些设定，大家可以参考《Django2：Web 项目开发入门笔记（12）》。

2、文章类别

文章的类别应该主要包含编号和类别名称。

示例代码：

```
class Category(models.Model): # 文章类别
    id = models.IntegerField(primary_key=True)
    name = models.CharField('类别', max_length=20, unique=True)

    class Meta:
        verbose_name_plural = verbose_name = '类别'

    def __str__(self):
        return self.name
```

在上方代码中，文章类别的编号“id”为主键。

不过这个编号，我们可能需要自己添加，所以不使用自增字段“AutoField”，而是使用整数字段“IntegerField”。

同时，还要注意文章类别不应该出现重复名称，所以类别字段的参数中需要添加唯一约束“unique=True”。

3、文章标签

文章标签只包含标签名称，当我们通过模型创建数据库表时，系统会自动添加一个主键列“id”。

当然，大家也可以考虑添加一个标签别名的字段，用于通过标签进行分类搜索。

因为标签别名可以使用清晰简短的英文短语，所以使用标签别名能够让搜索时的 URL 更加美观。

另外，仍然不要忘记，给标签名称字段添加唯一约束。

示例代码：

```
class Tag(models.Model): # 文章标签
    name = models.CharField('标签', max_length=20, unique=True)

    class Meta:
        verbose_name_plural = verbose_name = '标签'

    def __str__(self):
        return self.name
```

4、评论

以上都是文章主体内容相关的模型类。

而评论基于文章的扩展内容。

也就是说有文章才会有评论，删除一篇文章时，相应的评论也应该进行清除。

这个项目中关于评论功能包含的特性主要有评论编号、用户昵称、用户邮箱、评论内容、发布日期、所属文章、回复目标。

示例代码：

```
class Comment(models.Model):
    id = models.AutoField(primary_key=True)
    name = models.CharField('昵称', max_length=20)
    email = models.EmailField('邮箱')
    content = models.TextField('内容')
    publish = models.DateField('时间', auto_now=True)
    article = models.ForeignKey(Article, on_delete=models.CASCADE, verbose_name='文章')
    reply = models.ForeignKey('self', on_delete=models.DO_NOTHING, null=True, blank=True,
        verbose_name='回复')

    class Meta:
        verbose_name_plural = verbose_name = '评论'

    def __str__(self):
        return self.content
```


在上方代码中，需要特别说明特性是“article”和“reply”。

- **article**：所属文章，当文章删除时，文章的评论同步删除，这个关联表的删除操作无需我们编写代码，只需要将关键字参数“on_delete”的值设置为“CASCADE”；这样设置之后，当我们删除某一篇文章，Django 会自动帮助我们完成评论数据表中相关评论的删除。
- **reply**：回复目标，评论可以评论文章，也可以评论他人的评论，即回复；那么也就意味着评论需要和评论自身进行关联；在外键字段的参数中，第 1 个参数不要写“Comment”，这样是错误的；因为在定义当前特性时，“Comment”类也处于定义未完成的状态，所以，这里我们填写“self”来关联；另外，一条评论允许没有回复，关键字参数“null=True”即允许字段值为空值；但是仅设置这个参数，会导致 Django 后台中此项为空值时无法通过浏览器的验证，所以还要加上另外一个关键字参数“blank”，并设置为“True”。

到这里，我们就完成了全部模型类的创建。

接下来，我们创建相应的数据库和数据表。

Django2 中，SQLite 能够通过“migrate”命令自动完成数据库和相关数据表的创建。

但是，SQLite 以外的数据库则需要先进行数据库的创建。

因为，这里使用的数据库是 MySQL，所以，数据库的创建我们需要通过 MySQL 的 Shell 来完成。

以 Windows 系统为例。

以管理员身份打开命令行终端，输入命令：`mysql -u root -p`

然后，输入密码，回车后进入 MySQL 的 Shell，输入创建数据库“myblog”的命令。

```
mysql>CREATE DATABASE myblog DEFAULT CHARACTER SET utf8 COLLATE  
utf8_general_ci;
```

完成数据库的创建之后，就可以在项目文件夹下通过“makemigrations”和“migrate”命令进行数据表的创建。

最后，给大家推荐一款 MySQL 的可视化管理工具“Navicat for MySQL”，能够方便的进行 MySQL 数据库查看与相关操作。

转载请注明：魔力 Python » Django2 练习项目：开发个人博客系统（1）

原文链接：<http://www.opython.com/1187.html>

Django2 练习项目：开发个人博客系统（2）

这一篇教程，我们为创建好的数据库添加一些必要的测试内容。

站点的后台，我们使用 Django2 自带的后台。

主要操作如下：

前两步操作，大家可以参考《Django2：Web 项目开发入门笔记（12）》。

第三步操作，将模型注册到后台与之前的教程中略有不同。

示例代码：

```
from django.contrib import admin
from blog.models import * # 导入所有模型类

@admin.register(Article)
class ArticleAdmin(admin.ModelAdmin):
    list_display = ('title', 'category', 'pub_time') # 文章列表的显示项

admin.site.register((Category, Comment, Tag)) # 多个模块注册到后台
```

因为想在后台显示文章管理列表时，不仅仅显示一个文章的标题，所以，在上方代码中，文章模型注册到后台时需要指定这些项。

实现的方法是定义了一个文章管理的类“ArticleAdmin”继承自模块管理类“admin.ModelAdmin”，并通过装饰器“@admin.register”进行装饰，完成注册。

实际上“ArticleAdmin”类只是在定义一个管理列表，特性“list_display”是在这个列表中显示出来的所有列，至于这个列表是什么数据的列表，取决于装饰器参数中填入的模型类的名称“Article”，与类名“ArticleAdmin”并没有什么关系。

除此之外，其他的模型类，我们直接注册，无需进行更多处理。

这里的处理方法是将多个模型类放入了同一个元组中，变成了一个可迭代对象，并将这个可迭代对象作为“admin.site.register()”方法的参数，完成了多个模型的注册。

当我们完成上面的操作，就可以在 Django 的后台中添加测试数据了。

添加文章，是我们当前主要要添加的测试数据内容，添加分类以及添加相应的标签，可以在添加文章的同时进行添加。

不过，不要忘了先在文章分类添加一个分类编号为“1”，名称为“未分类”的默认分类。

如果大家找不到相应的测试文章，大家可以在本站复制文章内容添加到数据库中。

但是要注意，复制的文章内容需要带有样式。

具体的操作方法为：

- 1、打开需要复制的文章页面；
- 2、页面上点击鼠标右键选择查看网页源代码的选项；
- 3、在网页源代码中找到文章内容的一部分，将所有内容进行复制；
- 4、在 Django 后台添加一篇新的文章，将复制的内容粘贴到文章的内容编辑框中。

注意：在复制文章内容时，只应包含与段落、文本样式、图片、列表、代码相关的标签，不应包含其他标签，例如“<div>”层标签。

转载请注明：魔力 Python » Django2 练习项目：开发个人博客系统（2）

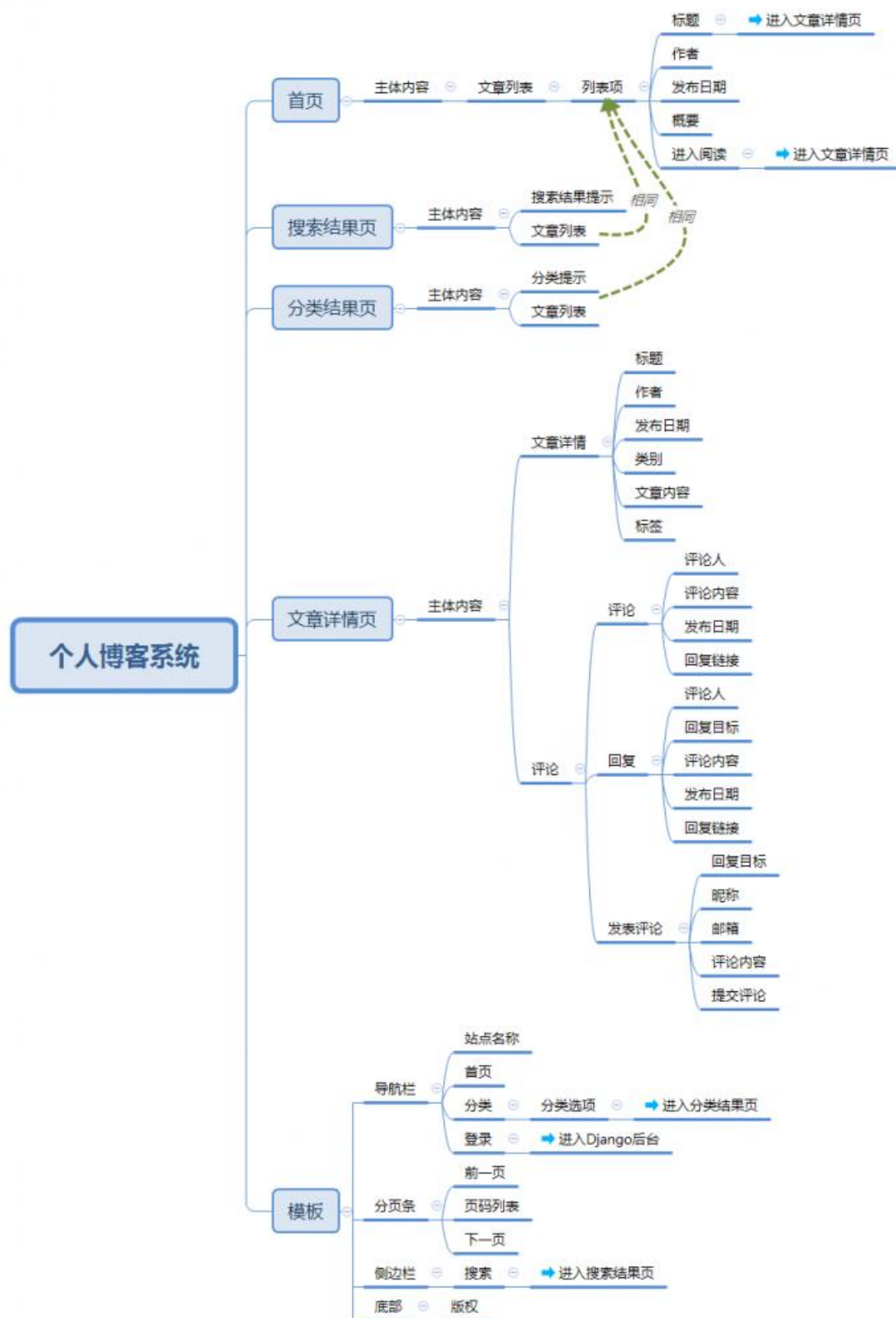
原文链接：<http://www.opython.com/1190.html>

Django2 练习项目：开发个人博客系统（3）

这一篇教程，我们一起为项目创建模板，也就是我们最终用来呈现数据的页面。

因为，我们是通过 Django2 搭建一个个人博客系统，首先要考虑的是个人博客所包含的页面以及页面的布局、功能与元素。

我们再来看一下我们的项目结构：

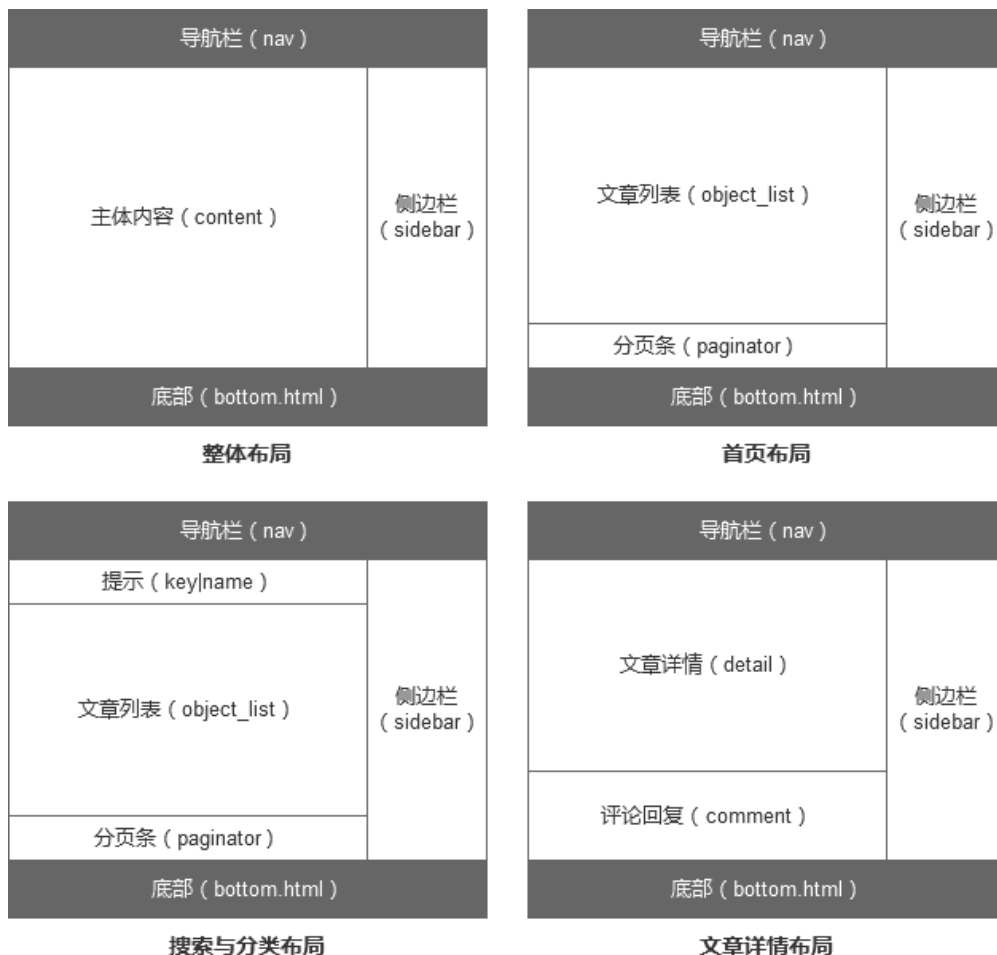


根据这张图中包含的页面与页面结构，我们进行页面布局，并将相同的部分抽象为模板。

1、布局规划

一个网站的外观应该有统一布局，整个站点才会显得协调一致。

根据上面结构图中列出的页面，我们规划整体的布局和每个页面的布局。



2、模板组成

有了上面的布局，我们把布局中的每一块通用内容做成页面的子模板。

这些模板包括：

- 导航栏：nav.html
- 底部：bottom.html
- 侧边栏：sidebar.html
- 分页条：paginator.html

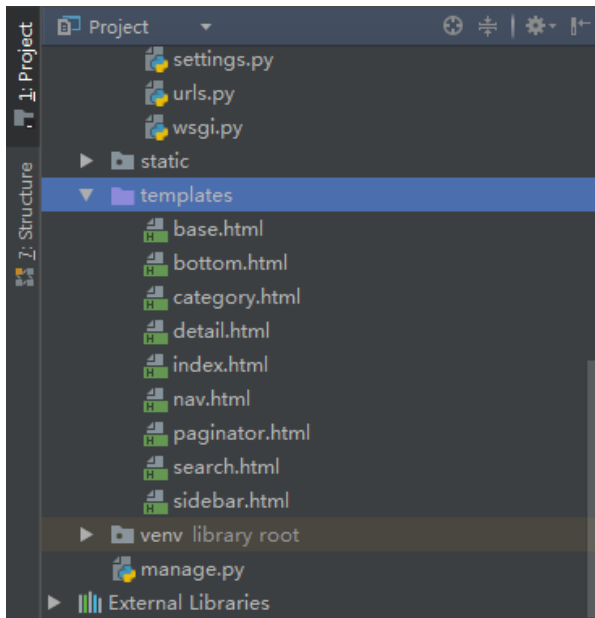
另外，为了不在每个页面模板中都重复一些相同的 HTML 代码以及包含这么多的模板，我们还需要创建一个基本模板。

- base.html

完成这些模板之后，我们再分别创建具体页面的模板：

- index.html
- search.html
- category.html
- detail.html

这些模板除了基本模板“base.html”，都可以把 PyCharm 自动生成的代码清空。



3、Bootstrap 框架

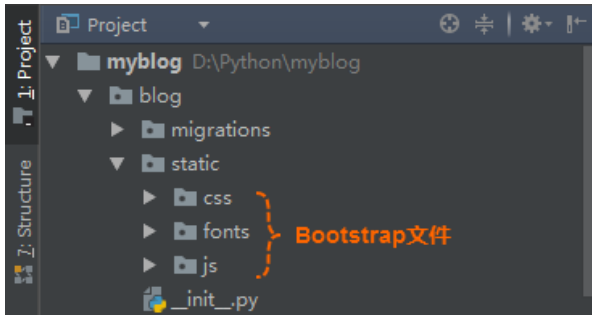
为了能够让我们的博客系统有一个漂亮的外观，这里我们使用 **bootstrap** 这个前端框架（包括 HTML、CSS 和 JS 框架）帮助我们完成视觉的优化。

首先，下载 Bootstrap V3.3.7 版，下载地址：<https://v3.bootcss.com/getting-started/#download>。

这里我们只需要下载用于生产环境的 Bootstrap，下载之后解压缩。

Django 项目的应用目录下新建“static”文件夹。

将解压缩之后的内容，也就是 css、js 和 fonts 三个文件夹复制到“static”文件夹中。



然后，在模板中使用 Bootstrap，我们可以通过以下代码实现。

示例代码：

```
<head>
<title>...</title>
{% load static %}

<link rel="stylesheet" href="{% static '/css/bootstrap.min.css' %}">

<script src="{% static '/js/jquery-3.3.1.min.js' %}"></script>

<script src="{% static '/js/bootstrap.min.js' %}"></script>

</head>
```

这段代码中，红色部分的代码就是加载 Bootstrap 样式表、脚本以及 JQuery 脚本的代码。

注意：因为 Bootstrap 的这些文件都存放在“static”目录中，我们需要通过“{% load static %}”引用。

完成了以上的准备工作之后，我们就可以在模板中添加代码了。

在这篇教程中，我们先根据整体布局，完成基本模板“base.html”。

示例代码：

```
<!DOCTYPE html>
<html lang="zh-CN">
<head>
  <meta charset="UTF-8">
  <meta name="viewport" content="width=device-width, initial-scale=1">
  <title>{% block title %}页面标题{% endblock %}</title>
  {% load static %}
  <link rel="stylesheet" href="{% static '/css/bootstrap.min.css' %}">
  <script src="{% static '/js/jquery-3.3.1.min.js' %}"></script>
```

```
<script src="{% static '/js/bootstrap.min.js' %}"></script>
</head>
<body>
{% include 'nav.html' %}
<!-- 顶部导航 -->
<div class="container"> <!-- 容器 -->
    <div class="row clearfix"><!-- 容器中的行 -->
        <div class="col-md-9 column"><!-- 行中的列：col-md-9 表示此列宽度为页面主体宽度
的 9/12。 -->
            <!--主要内容-->
            {% block content %}主要内容{% endblock %}
        </div>
        <div class="col-md-3 column hidden-xs hidden-sm">
            <!-- 行中的列：col-md-3 表示此列宽度为页面主体宽度的 3/12；hidden-xs 和
hidden-sm 表示在小屏幕和超小屏幕设备中打开页面时隐藏此列-->
            <!--侧边栏-->
            {% include 'sidebar.html' %}
        </div>
    </div>
</div>
<footer>
    <!--底部内容-->
    {% include 'bottom.html' %}
</footer>
</body>
</html>
```

在上方代码中，红色部分的代码用于支持响应式布局，加上之后才能够使用 **Bootstrap** 的响应式布局功能。

另外，代码中使用了一些 **Bootstrap** 属性，这些属性的作用，大家可以根据代码中注释进行理解。

最后，给大家一个建议。

很多朋友对 **HTML** 不是特别了解，如果想做出美观的页面还是比较有难度的。

我们可以借助一些现有的工具，降低开发难度。

例如，这个项目的教程中使用到的 **HTML** 代码，都是通过工具生成，部分代码略加改动。

这里给大家推荐，使用菜鸟教程的在线工具。

以导航栏为例：<http://www.runoob.com/bootstrap/bootstrap-navbar.html>

大家打开页面之后，就能够看到左侧的组件以及插件等列表，点击左侧右侧就会出现相应的示例和说明，并且可以通过【尝试一下】的功能修改代码查看效果。

当效果达到满意的程度，我们就可以将相应的代码复制到我们项目的模板文件中。

转载请注明：魔力 Python » Django2 练习项目：开发个人博客系统（3）

原文链接：<http://www.opython.com/1192.html>

Django2 练习项目：开发个人博客系统（4）

这一篇教程，我们继续进行 Django2 个人博客系统中所有模板的制作。

在上一篇教程中，我们已经完成了基本模板“base.html”的制作。

在这个模板中，我们通过“include”包含了一些子模板，分别是：

- 导航栏：nav.html
- 底部：bottom.html
- 侧边栏：sidebar.html

1、导航栏（nav.html）

示例代码：

```
<div class="container"> <!--容器-->
  <div class="row clearfix"><!--行-->
    <div class="col-md-12 column"><!--列-->
      <nav class="navbar navbar-inverse navbar-fixed-top" role="navigation"> <!--导航 反色 固定顶部-->
        <div class="navbar-header"> <!--标题-->
          <a class="navbar-brand" href="/">我的博客</a><!--navbar-header 和 navbar-brand 可以让标题文字大一号-->
        </div>
        <div class="collapse navbar-collapse" id="bs-example-navbar-collapse-1"><!--折叠-->
          <ul class="nav navbar-nav"> <!--导航列表-->
            <li class="active"> <!--激活的列表项-->
              <a href="/">首页</a>
            </li>
            <li class="dropdown"><!--下拉菜单-->
              <a href="#" class="dropdown-toggle" data-toggle="dropdown">分类<strong class="caret"></strong></a>
```

```
<ul class="dropdown-menu"><!-- 下拉菜单-->
  <li>
    <a href="{% url 'category' 2%}">Django</a>
  </li>
  <li>
    <a href="{% url 'category' 3%}">Django</a>
  </li>
</ul>
</li>
</ul>
<ul class="nav navbar-nav navbar-right " style="padding-right: 20px"><!-- 导航右侧-->
  <li class="active">
    <a href="admin/">登录</a>
  </li>
</ul>
</div>
</nav>
</div>
</div>
</div>
```

上方代码中，标签的“class”属性作用，大家可以结合注释理解。

另外，因为导航固定在顶部，页面主体内容会从页面顶部开始加载，这就会产生导航栏遮挡页面内容的情况。

为了解决这个问题，我们需要在基本模板“base.html”的“<head>...</head>”标签之间添加样式代码。

建议样式代码添加到其它内容的末尾，也就是“</head>”标签之前。

示例代码：

```
<style type="text/css">
  body {
    padding-top: 70px;
    padding-bottom: 50px;
  }
</style>
```

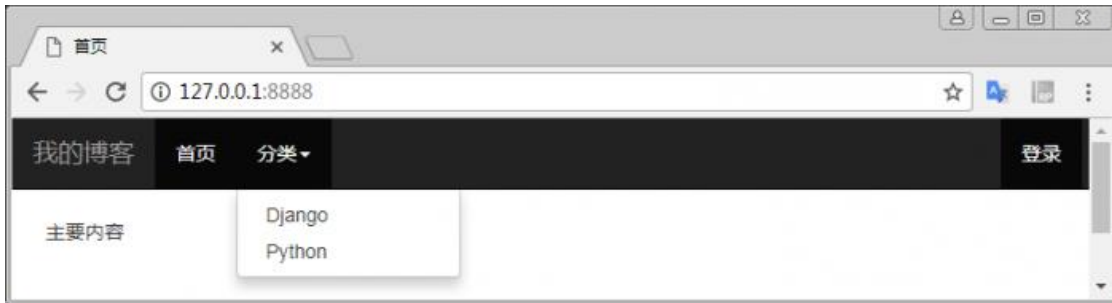
上方代码中，“body”表示对“<body>”标签的样式设置。

“padding-top: 70px;”表示“<body>”内容的上方填充 70 像素的空白。

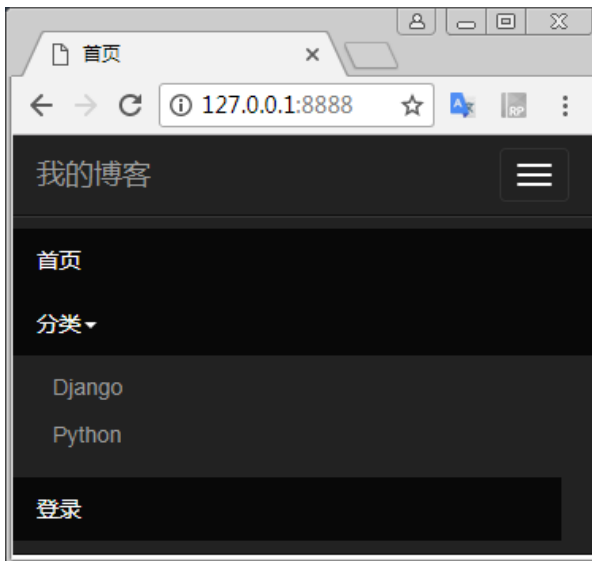
“padding-bottom: 50px;”表示“<body>”内容的下方填充 50 像素的空白。

通过这个两项样式设置，让页面主体内容与导航栏和底部产生间隔。

呈现效果：（PC 端）



呈现效果：（移动端）



2、底部（bottom.html）

示例代码：

```
<div class="copyright"> <!--版权-->
  <a href="http://www.opython.com " style="color: #9d9d9d">我的博客</a> 版权所有，
  保留一切权利 · 基于 Django 构建 · © 2017-2018
</div>
```

底部的代码比较简单，但是只有这些代码不能满足我们的样式需求。

因为协调统一的话，底部应该像导航一样，黑色并且水平铺满屏幕，同时页面内容过短时也应该在浏览器窗口的底部。

为了满足这样的需求，我们需要在基本模板的样式代码中继续添加对“<footer>”标签、class“copyright”以及 HTML 页面的样式设置。

```
html {  
    position: relative; # 相对定位（相对当前元素的位置）  
    min-height: 100%; # 页面最小高度为 100% 页面内容总高度  
}  
  
footer { # 定义“<footer>”标签的样式  
    position: absolute; # 绝对位置（父级元素的具体位置）  
    bottom: 0; # 标签内容底部与父级元素（本案例中父级元素为<html>）底部间隔，“0”表示无间隔，如有间隔需要用像素（px）表示。  
    width: 100%; # 100% 浏览器窗口宽度  
}  
  
.copyright { # 定义 class“<copyright>”标签的样式，注意以“.”开头。  
    background: #111; # 背景颜色  
    font-size: 13px; # 字体尺寸  
    text-align: center; # 字体水平居中对齐  
    color: #555; # 字体颜色  
    padding-top: 14px; # 顶部填充  
    padding-bottom: 14px; # 底部填充  
    border-top: 3px solid #337ab7; # 上边框的线条粗细、线型（当前为实线）和颜色  
}
```

为了方便大家理解样式的作用，在上方代码中我添加了相应的注释，使用时请将注释删除。

并且，为了帮助大家理解我对这些样式，再做一下简单的描述。

html：定义了以当前页面确定位置，所以最小高度为当前页面内容的 100% 高度；

footer：是“html”的子元素，位置通过 html 的边界确定，与底部无间隔，所以处于页面最底部，并且宽度为整体页面宽度；

copyright：所在的<div>标签是“footer”的子元素，宽高与“footer”一致。

提示：实际上，我们可以直接将“copyright”的样式设置，放在“footer”的样式设置中，无需单独设置。

呈现效果：



3、侧边栏（sidebar.html）

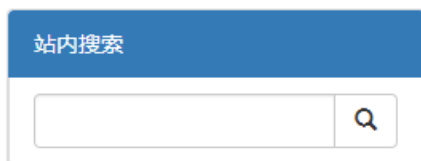
侧边栏内容是一个站内搜索框。

示例代码：

```
<div class="panel panel-default"> <!--面板 默认样式-->
  <a href="#" class="list-group-item active"><!--列表组合的项 激活状态-->
    站内搜索
  </a>
  <div class="list-group-item"><!--列表组合的项-->
    <form class="bs-example bs-example-form" role="search" action=" ../search/"
method="get"> <!--搜索表单-->
      <div class="input-group"> <!--输入框组合-->
        <input type="text" class="form-control" name="key" value="{{ key }}"> <!--输入框
默认值来自 key 变量-->
        <span class="input-group-btn"> <!--输入框组合的按钮-->
          <button class="btn btn-default" type="submit"> <!--按钮默认样式 类型为表单提
交按钮-->
            <span class="glyphicon glyphicon-search"></span><!--按钮中的字体图标-->
          </button>
        </span>
      </div>
    </form>
  </div>
```

上方代码中，同样添加了相应的注释，方便大家理解。

呈现效果：



以上就是基本模板“base.html”相关子模板的制作。

在之后的教程中，我们进行剩余模板的制作。

不过，剩余模板都是需要整合数据内容，比较好的实现过程是：视图>URL 配置>模板。

通过这样的过程，我们能够比较方便的完成每一个页面功能的实现。

因为有了视图，才能在 URL 配置中调用视图；有了 URL 配置，才能够在模板中添加请求路径（使用 URL Name）和使用带有数据的变量。

转载请注明：魔力 Python » Django2 练习项目：开发个人博客系统（4）

原文链接：<http://www.opython.com/1203.html>

Django2 练习项目：开发个人博客系统（5）

这篇教程我们一起来完成 Django2 个人博客系统首页相关的代码编写。

首先，根据我们系统结构，首页包含的数据内容是所有文章的列表。

那么，我们可以使用通用视图中的列表视图。

这里需要注意的问题是文章要按编号倒序排列，这样新发布的文章才会在列表首位出现。

示例代码：

```
from django.shortcuts import render
from django.views.generic import ListView
from .models import *

# Create your views here.
class Index(ListView):
    model = Article
    template_name = 'index.html'
    queryset = Article.objects.all().order_by('-id') # 获取到全部文章并按编号降序排列。
```

不过还没有结束，如果文章很多的话，肯定需要分页功能。

这里我们使用 Django 自带的分页功能，使用起来非常的简单。

在“Index”代码中我们再添加一行。

示例代码：

```
paginate_by = 5 # 设置分页时每页的文章数量
```

这样，我们就开启了分页的支持。

然后，我们进行 URL 配置。

示例代码：

```
from django.contrib import admin
from django.urls import path
from blog import views as blog_view
```

```
urlpatterns = [
    path("", blog_view.Index.as_view()),
    path('admin/', admin.site.urls),
]
```

接下来，我们就可以创建首页的模板内容了。

示例代码：（index.html）

```
{% extends 'base.html' %} <!--从基本模板扩展-->
{% block title %}首页{% endblock %} <!-- 首页标题-->
{% block content %} <!--重写内容块-->
    {% for article in page_obj.object_list %} <!--从分页的对象列表中遍历文章-->
        <div class="panel panel-default"> <!--显示文章的面板-->
            <div class="panel-heading"> <!--面板的头部-->
                <h3 class="panel-title"> <!--放置内容标题-->
                    <a href="{% url 'detail' article.id %}">{{ article.title }}</a>
                </h3>
            </div>
            <div class="panel-body"> <!--面板的主体-->
                <p>作者：{{ article.author }} 日期：{{ article.pub_time }}</p>
                <p>{{ article.content|truncatechars:200|striptags }}</p> <!--过滤器用于仅显示
200 字符和去除 HTML 标签-->
                <a class="btn" href="{% url 'detail' article.id %}">进入阅读 »</a>
            </div>
        </div>
    {% endfor %}
{% endblock %}
```

提示：上方代码中“truncatechars”也可以使用“truncatechars_html”，能够将被截断的 HTML 标签去除。

这里大家要注意一点。

因为，我们对数据内容做了分页设置，所以，在模板中我们不能直接遍历“object_list”，而是要遍历分页的页面对象“page_obj”中的“object_list”。

这样，我们页面上才会只显示一页的内容。

那么，如何进行翻页呢？

我们需要在模板中添加分页条功能。

不过，分页条功能不是首页独有的，例如搜索结果页和分类结果页也都是文章列表，同样需要分页条。

对于这种情况，我们就可以把分页条这个模块，单独做成模板，在需要的页面中去“include”。

提示：在 PyCharm 中，分页条的代码可以现在首页模板“index.html”中进行编辑后，再复制到分页条模板“paginator.html”文件中。因为，我们数据内容关联的是首页模板，直接在分页条模板中编写代码没有联想提示，不是特别方便。

示例代码：（paginator.html）

```
<ul class="pagination">
    {% if page_obj.has_previous %} <!--如果有前一页-->
        <li><a href="/?page={{ page_obj.previous_page_number }}">&laquo;</a></li> <!--链接到前一页页码-->
    {% else %}
        <li class="disabled"><a>&laquo;</a></li> <!--否则禁用前一页按钮-->
    {% endif %}
    {% for page_number in paginator.page_range %} <!--遍历页码范围-->
        {% if page_number != page_obj.number %} <!--如果页码与当前页页码不相同-->
            <li><a href="/?page={{ page_number }}">{{ page_number }}</a></li> <!--生成页码并添加链接-->
        {% else %}
            <li class="active"><a>{{ page_number }}</a></li> <!--否则，呈现激活样式-->
        {% endif %}
    {% endfor %}
    {% if page_obj.has_next %} <!--如果有下一页-->
        <li><a href="/?page={{ page_obj.next_page_number }}">&raquo;</a></li> <!--链接到下一页页码-->
    {% else %}
        <li class="disabled"><a>&raquo;</a></li> <!--否则禁用下一页按钮-->
    {% endif %}
</ul>
```


在上方的代码中，需要注意的是分页的链接为“/?page=页码”的格式。也就是说链接中需要添加名为“page”的参数。

最后，我们将写好的分页条模板，包含到首页的页面中。

示例代码：index.html

```
...省略部分代码...
```

```
{% endfor %}
```

```
<!--分页条-->
```

```
{% include 'paginator.html' %}
```

```
{% endblock %}
```

上方代码中，红色部分是新增代码。

到这里，我们就完成了首页的全部功能。



转载请注明：魔力 Python » Django2 练习项目：开发个人博客系统（5）

原文链接：<http://www.opython.com/1214.html>

Django2 练习项目：开发个人博客系统（6）

这篇教程我们一起来完成 Django2 个人博客系统分类页相关的代码编写。

在之前创建导航模板“nav.html”中，我们已经添加了两个分类“Django”和“Python”。

当在浏览器中点击这两个分类的时候，应该能够筛选出相应分类的文章，并且在浏览器中呈现出来。

不过，分类结果页和首页应该有些不同，就是在文章列表上方要有提示当前是什么分类内容的消息。

这个相对于首页额外多处来的消息，需要我们在视图进行处理。

示例代码：

```
class CategoryList(ListView):
    model = Article
    template_name = 'category.html'
    paginate_by = 5

    def get_queryset(self): # 定义通过分类查询的 QuerySet
        return Article.objects.filter(category=self.kwargs['category']).order_by('-id') # 按参数
        传入的分类 id 进行查询并按文章编号降序排序

    def get_context_data(self, **kwargs): # 增加额外要传递给模板的数据
        context = super().get_context_data(**kwargs)
        category = Category.objects.get(id=self.kwargs['category']) # 通过分类 id 查询分类对
        象
        context['category'] = category.name # 将分类对象的名称存入传递给模板的数据中
        return context
```

然后，添加 URL 配置。

示例代码：

```
path('category/<int:category>', blog_view.CategoryList.as_view(), name='category'),
```

接下来，定义模板内容。

这个模板内容可以从首页模板“index.html”中直接复制，然后添加一些内容。

示例代码：

```
{% extends 'base.html' %}
{% block title %}分类{% endblock %}
```

{% block content %}

<h4> 以下是【{{ category }}】类别的文章：</h4>

```
{% for article in page_obj.object_list %}
<div class="panel panel-default">
  <div class="panel-heading">
    <h3 class="panel-title">
      <a href="{% url 'detail' article.id %}">{{ article.title }}</a>
    </h3>
  </div>
  <div class="panel-body">
    <p>作者：{{ article.author }} 日期：{{ article.pub_time }}</p>
    <p>{{ article.content|truncatechars:200|striptags }}</p>
    <a class="btn" href="{% url 'detail' article.id %}">进入阅读 »</a>
  </div>
</div>
{% endfor %}
<!--分页条-->
{% include 'paginator.html' %}
{% endblock %}
```

上方代码中，标红的部分是新增代码。

当我们完成上述内容，就可以在浏览器中通过分类进行查询了。



转载请注明：魔力 Python » Django2 练习项目：开发个人博客系统（6）

原文链接：<http://www.opython.com/1216.html>

Django2 练习项目：开发个人博客系统（7）

这篇教程我们一起来完成 Django2 个人博客系统搜索结果页相关代码的编写。

搜索栏功能在模板“sidebar.html”中，在之前的教程中我们已经完成。

结合前面的教程，我们知道搜索的路径是“search/”，并且通过“GET”方法，将输入框“key”的值进行传递。

在视图中，我们就需要根据“key”的值获取搜索结果。

那么，我们通过关键词搜索时，往往是需要进行模糊搜索。

这里，我们要实现将标题或内容中包含关键词的文章查询出来。

另外，还有一点需要实现，就是在搜索结果页的搜索框中要保留关键字。

示例代码：

```
from django.db.models import Q # 帮助完成查询条件设置

class Search(ListView):
    model = Article
    template_name = 'search.html'
    paginate_by = 5

    def get_queryset(self):
        key = self.request.GET['key'] # 获取查询关键字
        if key:
            return Article.objects.filter(Q(title__icontains=key) |
            Q(content__icontains=key)).order_by('-id')
            # 查询标题或者内容包含关键字的数据对象
        else:
            return None

    def get_context_data(self, **kwargs):
        context = super().get_context_data(**kwargs)
        context['key'] = self.request.GET['key'] # 获取关键字存入传入模板的数据中
        return context
```

然后，我们进行 URL 设置。

示例代码：

```
path('search/', blog_view.Search.as_view(), name='search'),
```

最后，定义搜索结果页面的模板内容。

依然可以从首页模板复制内容进行增加。

提示：其实大部分相同的页面结构，也可单独为这些页面创建父级模板，以便减少模板中的代码量。

示例代码：（search.html）

```
{% extends 'base.html' %}
{% block title %}搜索结果{% endblock %}
{% block content %}
    <h4> 以下是 【{{ key }}】 的查询结果： </h4>
    {% for article in object_list %}
        <div class="panel panel-default">
            <div class="panel-heading">
                <h3 class="panel-title">
                    <a href="{% url 'detail' article.id %}">{{ article.title }}</a>
                </h3>
            </div>
            <div class="panel-body">
                <p>作者：{{ article.author }} 日期：{{ article.pub_time }}</p>
                <p>{{ article.content|truncatechars:200|striptags }}</p>
                <a class="btn" href="{% url 'detail' article.id %}">进入阅读 »</a>
            </div>
        </div>
    {% empty %}
        <h4> 没有查询结果！ </h4>
    {% endfor %}
    <!--分页条-->
    {% include 'paginator.html' %}
{% endblock %}
```

到这里，我们就可以正常使用搜索功能了。



转载请注明：魔力 Python » Django2 练习项目：开发个人博客系统（7）

原文链接：<http://www.opython.com/1245.html>

Django2 练习项目：开发个人博客系统（8）

这一篇教程，我们一起完成 Django2 个人博客系统的文章详情页面的内容部分。

文章详情页面包含文章主体内容、评论内容以及评论发布功能。

因为内容比较多，我们在这一篇先完成视图部分。

代码很长，不能一次全放进来，那样会引起不适。

所以，我一段一段的慢慢深入，大家一起感受这个过程。

首先，我们需要导入 Django 提供的 `DetailView` 类，这是详情通用视图。

示例代码：

```
from django.views.generic import ListView,
```

```
DetailView
```

上方代码中，红色部分是新增内容。

然后，我们定义文章详情类继承自 `DetailView` 类，并设置文章详情的文章主体内容部分。

示例代码：

```
class ArticleDetail(DetailView):
    model = Article
    template_name = 'detail.html'
```

实际上，通过上方的代码，我们就可以结合 URL 设置以及模板呈现文章主体内容了。

不过，在文章详情页还需要显示评论内容。

评论内容的展现效果如下：

评论

小楼一夜听春雨：镜楼图不错！(2018年5月7日) [回复]
小白 >>> 小楼一夜听春雨：我也觉得不错 (2018年5月9日) [回复]
小糖糖 >>> 小白：你们两个好没节操！(2018年5月9日) [回复]
小白 >>> 小糖糖：嘿嘿！都是跟小楼学的！(2018年5月19日) [回复]
梦儿：小楼，棒棒大！(2018年5月9日) [回复]
小希 >>> 梦儿：你怎么知道的？(2018年5月9日) [回复]
梦儿 >>> 小希：当然是看过才知道呀！(2018年5月12日) [回复]
吃瓜群众：好热闹啊！(2018年5月12日) [回复]

我们分析一下。

这样的评论列表，并不是完全按评论的发布顺序排序。

顶级评论是按发布顺序，顶级评论下的回复是按照层级进行排序。

评论文章是顶级评论，回复一条文章的评论是二级评论，回复一条回复内容是三级评论，以此类推。

而评论数据存储到数据库时并不是按照层级存储，所以我们需要将这些评论按照层级关系重新排列顺序。

具体的思路如下：

递归的过程不太容易理解，简单为大家描述一下。

调用递归函数时，存储参数中传入的顶级评论到最终返回的评论列表，再通过这个顶级评论的 id 从字典中获取它的所有回复评论，然后进行递归；此时再次调用递归函数，存储顶级评论的第一个回复评论，在通过这个回复评论的 id 获取这个回复评论的所有回复评论，再次递归，以此类推，直到找不到新的一层回复评论，结束递归。

综上所述，我们需要在文章详情类（ArticleDetail）中定义相关的函数。

示例代码：

```
def comment_sort(self, comments): # 评论排序函数
    self.comment_list = [] # 排序后的评论列表
    self.top_level = [] # 存储顶级评论
    self.sub_level = {} # 存储回复评论
    for comment in comments: # 遍历所有评论
        if comment.reply == None: # 如果没有回复目标
            self.top_level.append(comment) # 存入顶级评论列表
        else: # 否则
            self.sub_level.setdefault(comment.reply.id, []).append(comment) # 以回复目标（父级评论）id 为键存入字典
    for top_comment in self.top_level: # 遍历顶级评论
        self.format_show(top_comment) # 通过递归函数进行评论归类
    return self.comment_list # 返回最终的评论列表

def format_show(self, top_comment): # 递归函数
    self.comment_list.append(top_comment) # 将参数评论存入列表
    try:
        self.kids = self.sub_level[top_comment.id] # 获取参数评论的所有回复评论
    except KeyError: # 如果不存在回复评论
        pass # 结束递归
    else: # 否则
        for kid in self.kids: # 遍历回复评论
            self.format_show(kid) # 进行下一层递归
```

完成了这两个函数之后，我们就可以从数据库中获取当前文章的所有评论内容，进行归类排序，并整合到发送给模板的数据中。

示例代码：

```
def get_context_data(self, **kwargs):
    context = super().get_context_data(**kwargs)

    comments = Comment.objects.filter(article=self.kwargs['pk']) # 通过文章 id 查询评论内容

    context['comment_list'] = self.comment_sort(comments) # 将排序归类后的文章列表存入传送到模板的数据中

    return context
```

在上方代码中，标红部分就是关于评论的显示内容。

另外，在文章详情页，我们还需要提供评论的功能。

评论功能的表单我们可以通过“**ModelForm**”进行创建。

提示：实际上，在之后的教程中，因为使用了 **JQuery** 提交评论数据，这个表单并没有进一步的作用，之所以在这里这么做，是为了给大家介绍一些更多的内容。

在模板中我们结合了 **Bootstrap** 和 **JQuery**，当我们使用 **Django** 表单时，如何让自动生成的表单元素使用 **Bootstrap** 的样式呢？如何让 **JQuery** 能够关联表单的元素呢？

新建一个表单文件“**forms.py**”，然后编写代码。

示例代码：

```
from django import forms
from .models import Comment
```

```
class CommentForm(forms.ModelForm):
    class Meta:
        model = Comment
        fields = ['name', 'email', 'content'] # 需要呈现在页面上的表单元素
        widgets = { # 定义字段对应的表单元素以及属性
            'name': forms.TextInput(attrs={'id': 'name', 'class': 'form-control', 'placeholder': '请输入昵称'}),
            'email': forms.EmailInput(attrs={'id': 'email', 'class': 'form-control', 'placeholder': '请输入邮箱'}),
            'content': forms.Textarea(attrs={'id': 'content', 'class': 'form-control', 'placeholder': '请输入评论内容'}),
        }
```

在上方代码中，我们通过“**widgets**”定义了字段对应的表单元素，并且设置了属性。

属性中定义的“**id**”，是为了方便 **JQuery** 获取以及控制元素的属性与值。

而属性中定义的“**class**”则是用来调用 **Bootstrap**。

完成上方代码之后，我们就可以在视图中使用这个表单了。

示例代码：

```
from .forms import CommentForm
```

```
def get_context_data(self, **kwargs):  
    ...省略部分代码...
```

```
comment_form = CommentForm() # 创建评论表单对象  
  
context['comment_form'] = comment_form # 将表单对象传送到模板的数据中  
  
return context
```

这篇教程就到这里，在下一篇教程中，我们将基于这个视图进行模板的创建。

转载请注明：魔力 Python » Django2 练习项目：开发个人博客系统（8）

原文链接：<http://www.opython.com/1224.html>

Django2 练习项目：开发个人博客系统（9）

这篇教程我们一起来完成 Django2 个人博客系统的文章详情模板。

在定义模板内容之前，我们先完成 URL 配置。

示例代码：

```
path('detail/<int:pk>', blog_view.ArticleDetail.as_view(), name='detail'),
```

然后，我们开始模板的定义。

一、引入基本模板

引入基本模板之后，我们需要定义页面的标题。

使用 `DetailView` 会向模板传递一个基于 `Model` 的数据对象“object”，当然这个名称可以在视图中重写。

在模板中我们就可以通过“object.title”获取属性，另外，也可以通过类名获取属性，例如“article.title”。

示例代码：（detail.html）

```
{% extends 'base.html' %}  
{% block title %}{{ object.title }}{% endblock %} <!--定义文章标题内容-->  
{% block content %}
```

...此处添加后续代码...

```
{% endblock %}
```

二、定义页面内容

文章详情页面主要包含以下内容：

- 文章内容（名称、作者、日期、类别、正文）
- 文章标签
- 评论列表（评论文章、回复评论）
- 发布评论（昵称、邮箱、内容、评论目标）

1、文章内容

示例代码：

```
<div>
  <h3>{{ object.title }}</h3>
  <p>作者：{{ object.author }} 日期：{{ object.pub_time }} 类别：<a
    href="{% url 'category' article.category.id %}">{{ article.category }}</a></p>
  {{ object.content | safe }} <!--过滤器 safe 能够让文章内容中的 HTML 标签生效-->
</div>
```

2、文章标签

一篇文章可能具有多个标签，所以需要对文章的所有标签进行遍历，并呈现在页面中。

另外，标签之间通过“|”进行分隔。

示例代码：

```
<div>
  <h3>标签</h3>
  {% for tag in object.tag.all %} <!--遍历所有标签-->
    {{ tag }} <!--显示标签到页面-->
    {% if not forloop.last %} <!--如果不是循环的最后一项-->
      | <!--显示一条竖线-->
    {% endif %}
  {% endfor %}
</div>
```

3、评论列表

评论列表需要按顶级评论分块。

因为 HTML 的列表标签“...”是成对的，每一组评论内容都要写在这两个标签中间，所以要注意标签的闭合。

这里存在一些逻辑。

如果不是第一条顶级评论，需要在评论内容前面加上“”标签，让前一组评论的标签闭合。

如果是顶级评论，需要在评论内容前加上“”标签。

如果不是顶级评论，需要在评论中加上“>>>”，表示对哪一个评论人的回复。

如果是最后一条评论，需要在评论内容末尾加上“”标签，让当前一组标签闭合。

示例代码：

```
<div>
  <h3>评论</h3>
  {% for comment in comment_list %} <!--遍历评论数据列表-->
    {% if comment.reply == None %} <!--如果没有回复目标（即顶级评论）-->
      {% if not forloop.first %} <!--如果不是循环遍历的第一项-->
        </ul>
      {% endif %}
      <ul class="list-group"> <!--顶级评论-->
        <li class="list-group-item active">{{ comment.name }} : {{ comment.content }}
        ({{ comment.publish }})
        <a href="#publish" onclick="reply('{{ comment.name }}',{{ comment.id }})"
          style="color: white;align-self: right">[回复]</a>
        <!--onclick 的属性中调用了一个 js 函数 函数的参数为评论人和评论 id 此函数在之后实现-->
        </li> <!--class 的属性值“active”能够让列表项样式为蓝色-->
      {% else %}
        <li class="list-group-item"> <!--回复评论-->
          {{ comment.name }} >>> {{ comment.reply.name }} : {{ comment.content }}
          ({{ comment.publish }})
          <a href="#publish" onclick="reply('{{ comment.name }}',{{ comment.id }})">[回
          复]</a>
          <!--“#publish”能够让页面滚动到发表评论的位置-->
        </li>
      {% endif %}
    {% if forloop.last %} <!--如果是循环遍历的最后一项-->
```

```

    </ul>
    {% endif %}
    {% empty %} <!--如果数据列表是空的-->
        暂无评论！
    {% endfor %}
</div>

```

4、发表评论

发表评论中主要是表单元素以及样式的设置。

发表评论

示例代码：

```

<div>
    <h3 id="publish">发表评论</h3>
    <input type="hidden" id="reply" name="reply" value="0"> <!--隐藏元素 用于记录回复的
    目标-->
    <div class="input-group"> <!--class 中调用 Bootstrap 文本框组的样式-->
        <span class="input-group-addon glyphicon glyphicon-
        user"></span>{{ comment_form.name }}
        <!--span 标签的 class 属性中调用 Bootstrap 文本框组的样式以及文字图标-->
        <!--span 标签的后方通过表单字段生成页面元素-->
    </div>
    <div class="input-group" style="margin-top: 10px"> <!--样式的作用是设置和上方的元素
    间隔-->
        <span class="input-group-addon glyphicon glyphicon-
        envelope"></span>{{ comment_form.email }}
    </div>
    <div class="input-group" style="margin-top: 10px">
        <span class="input-group-addon glyphicon glyphicon-

```

```
edit"></span>{{ comment_form.content }}
</div>
<button id="submit_comment" type="button" class="btn btn-default" style="margin-top:
10px"><span
    class="glyphicon glyphicon-hand-up"></span> 回复
</button> <!--button 标签的 class 属性中调用 Bootstrap 按钮样式并通过 span 标签添加
了文字图标-->
<label id="comment_message" hidden style="margin-top: 10px;vertical-align:
middle;color: green"></label>
<!--lable 标签用于显示评论的提示 默认隐藏 样式中定义了与上方元素的间距 与左侧按钮
的垂直对齐方式 还有默认文字颜色-->
</div>
```

到这里，我们的文章详情页就能够正常的显示内容了。

大家可以在 Django 自带的后台中添加一些评论内容，检验评论是否显示正常。

转载请注明：魔力 Python » Django2 练习项目：开发个人博客系统（9）

原文链接：<http://www.opython.com/1237.html>

Django2 练习项目：开发个人博客系统（10）

这一篇教程，我们结合 JQuery 完成 Django2 个人博客系统的发布评论的功能。

如果要发布评论，我们需要先定义一个发布评论的视图函数。

在这个视图函数中，将请求中的数据通过模型存储到数据库。

这里要注意，评论的所属文章和回复的目标评论都不是用户填写的表单内容，而是需要通过请求中的相关参数查询到数据对象添加到评论对象中，再将评论对象和用户提交的表单数据结合到一起创建表单数据对象进行校验与保存。

示例代码：

```
from django.shortcuts import HttpResponseRedirect
from .forms import CommentForm

def pub_comment(request): # 发布评论函数
    if request.method == 'POST': # 如果是 post 请求
        comment = Comment() # 创建评论对象
        comment.article = Article.objects.get(id=request.POST.get('article')) # 设置评论所属的文章
        if request.POST.get('reply') != '0': # 如果回复的不是文章而是他人评论
```

```
comment.reply = Comment.objects.get(id=request.POST.get('reply')) # 设置回复的目标评论
```

```
form = CommentForm(request.POST, instance=comment) # 将用户的输入和评论对象结合为完整的表单数据对象
```

```
if form.is_valid(): # 如果表单数据校验有效
```

```
    try:
```

```
        form.save() # 将表单数据存入数据库
```

```
        result = '200' # 提交结果为成功编码
```

```
    except: # 如果发生异常
```

```
        result = '100' # 提交结果为失败编码
```

```
else: # 如果表单数据校验无效
```

```
    result = '100' # 提交结果为失败编码
```

```
    return HttpResponseRedirect(result) # 返回提交结果到页面
```

```
else: # 如果不是 post 请求
```

```
    return HttpResponseRedirect('非法请求！') # 返回提交结果到页面
```

接下来，我们添加 URL 配置。

示例代码：

```
path('comment/', blog_view.pub_comment, name='comment'),
```

最后，我们来完成模板内容。

首先，在我们之前已经完成的模板内容中，评论内容后方“[回复]”的元素属性中都包含了如下属性：

```
onclick="reply('{{ comment.name }}',{{ comment.id }})"
```

这个属性指定了当“[回复]”被点击时执行“reply()”函数。

所以，我们需要在模板内容中添加一个“reply()”函数，它的作用是点击“[回复]”时记录回复目标的 id 以及在评论内容输入框中显示被回复的评论人。

示例代码：

```
<script>
    function reply(comment_name, comment_id) {
        $('#content').attr('placeholder', '回复' + comment_name + '的内容：'); // 设置内容输入框的提示
        $('#reply').val(comment_id) //设置隐藏元素的值为评论目标的 id
    }
</script>
```

在上方代码中，通过“\$('#元素 id)’对元素的属性值进行了更改，当点击“[回复]”的时候就能够呈现我们需要的效果。

然后，我们继续添加回复按钮的函数。

点击回复按钮时，我们做一个简单的非空验证，如果所有内容均已填入，我们将内容提交，否则给出检查输入内容的提示。

当内容正常提交之后，还要根据返回的结果给出“评论成功”或者“评论失败”的提示。

另外，这里还要注意一点，我们的评论表单并不是放在了一对“<form>...</form>”标签中，所以，没有添加防止跨域攻击的代码。

这会导致提交评论时发生异常。

我们在编写 Javascript 代码的时候，需要添加防止跨域攻击的代码。

示例代码：

```
<script>
$(document).ready(function () {
    $('#submit_comment').click(function () { // 定义回复按钮点击时调用的函数
        $.ajaxSetup({ // 添加防止跨域攻击的代码
            data: {csrfmiddlewaretoken: '{{ csrf_token }}'}
        });
        var reply = $('#reply').val(); // 将评论目标 id 存入变量
        var name = $('#name').val(); // 将评论人昵称存入变量
        var email = $('#email').val(); // 将评论人邮箱地址存入变量
        var content = $('#content').val(); // 将评论内容存入变量
        if (name && email && content) { // 如果所有内容都已填写
            $.post('% url 'comment' %', { // 用 post 方法提交请求
                'article': {{ article.id }}, // 请求中包含的参数字典
                'reply': reply,
                'name': name,
```



```
        'email': email,
        'content': content
    }, function (result) { // 回调函数获取返回结果
        if (result === '200') { // 如果返回评论成功编码
            $('#comment_message').css({color: "green"}).html('评论成功！'); // 设置提示
            元素的样式与文字
        } else {
            $('#comment_message').css({color: "red"}).html('评论失败！');
        }
    });
    $('#comment_message').removeAttr('hidden'); // 去除提示元素的隐藏属性将提示
    显示在页面
    setTimeout(function () { // 设置超时后执行的函数
        location.reload() // 重载页面内容
    }, 1000); // 设置超时时长
} else { // 如果不是所有内容都已填写
    $('#comment_message').removeAttr('hidden').html('请检查填写的内容！
    ').css({color: "red"});
    // 去除提示元素的隐藏属性将提示显示在页面，同时设置提示的样式与文字
}
});
});
</script>
```

到这里我们就完成发表评论的功能。

转载请注明：魔力 Python » Django2 练习项目：开发个人博客系统（10）

原文链接：<http://www.opython.com/1247.html>

Django2 练习项目：开发个人博客系统（11）

这一篇教程，我们通过 Django2 的 Session，让个人博客系统中用户的昵称和邮箱能够根据以往的输入内容自动填入，并且当评论失败时也能够保留评论内容。

我们需要完成的功能如下：

- 用户提交评论时，需要将填入的昵称和邮箱存入 session。
- 用户提交评论成功时，session 中存储的评论内容为空，否则将未成功提交的评论内容存入 session。
- 请求页面时，需要将 session 中的内容添加到传送到模板的数据中。
- 页面加载时，获取视图中传来的 session 数据，分别写入不同表单元素的值。

我们就按上述顺序完成个个功能的添加。

1、在提交评论的视图函数中添加新的代码，向 session 中存储数据。

示例代码：

```
def pub_comment(request):
    if request.method == 'POST':

        request.session['name'] = request.POST.get('name') # 将请求中的昵称存入 session

        request.session['email'] = request.POST.get('email') # 将请求中的邮箱存入 session

        ...省略部分代码...
        if form.is_valid():
            try:
                ...省略部分代码...

                request.session['content'] = "" # 发布成功时 session 中存储的内容数据为空值

            except:
                ...省略部分代码...

                request.session['content'] = request.POST.get('content') # 发布失败时将请求中的
                内容存入 session

        else:
            ...省略部分代码...
```

以上代码中，标注为红色的代码为新增内容。

2、在页面详情视图中获取 session 的内容并存入传送给模板的数据中。

示例代码：

```
class ArticleDetail(DetailView):
    ...省略部分代码...

    def get_context_data(self, **kwargs):
```

...省略部分代码...

```
try:

    context['session'] = {

        'name': self.request.session['name'],

        'email': self.request.session['email'],

        'content': self.request.session['content']

    }

    # 将 session 数据存入传送到模板的数据中

except: # session 读取异常时不做处理

    pass
```

...省略部分代码...

以上代码中，标注为红色的代码为新增内容。

3、在文章详情的模板中添加一个函数，负责页面载入时为指定的表单元素赋值。

示例代码：

```
{% block content %}
<script>
    function reply(comment_name, comment_id) {
        ...省略部分代码...
    }

    function onload() {

        $('#name').val('{{ session.name }}'); // 设置昵称为 session 中的昵称

        $('#email').val('{{ session.email }}'); // 设置邮箱为 session 中的邮箱

        $('#content').val('{{ session.content }}') // 设置内容为 session 中的内容

    }

</script>
```

...省略部分代码...
{% endblock %}

以上代码中，标注为红色的代码为新增内容。

4、在页面载入时需要执行“onload()”函数，这个触发我们需要房子基本模板“base.html”的“<body>”标签中。

示例代码：

```
<body  
onload="onload()"
```

以上代码中，标注为红色的代码为新增内容。

到这里我们就实现了想要的功能。

那么，使用 Django2 开发个人博客系统的教程到这里也就结束了。

感谢大家对魔力 Python 教程的支持！

这个项目并不尽善尽美，只是通过这个项目帮助大家之前学习的常用知识内容进行串联，熟悉 Django 常用功能的使用方法。

对于这个项目，我们还能够进行更多的优化。

例如：

- 自定义后台模块，为用户添加权限，不同用户具备不同的后台功能；
- 添加支持第三方登录的功能；
- 添加根据文章标签筛选的功能；
- 添加前一篇和后一篇文章的快捷链接；
- 限制用户登录之后才能够发布评论；
- 解决其它更多的需求。

这里的教程只是抛砖引玉，希望大家更够做出具有更加丰富功能的博客系统。

只有更多的练习与实战，才能够将所学知识融会贯通，早日成为 Django2 的高手。

项目源代码下载：[【点此下载】](#)

转载请注明：魔力 Python » Django2 练习项目：开发个人博客系统（11）

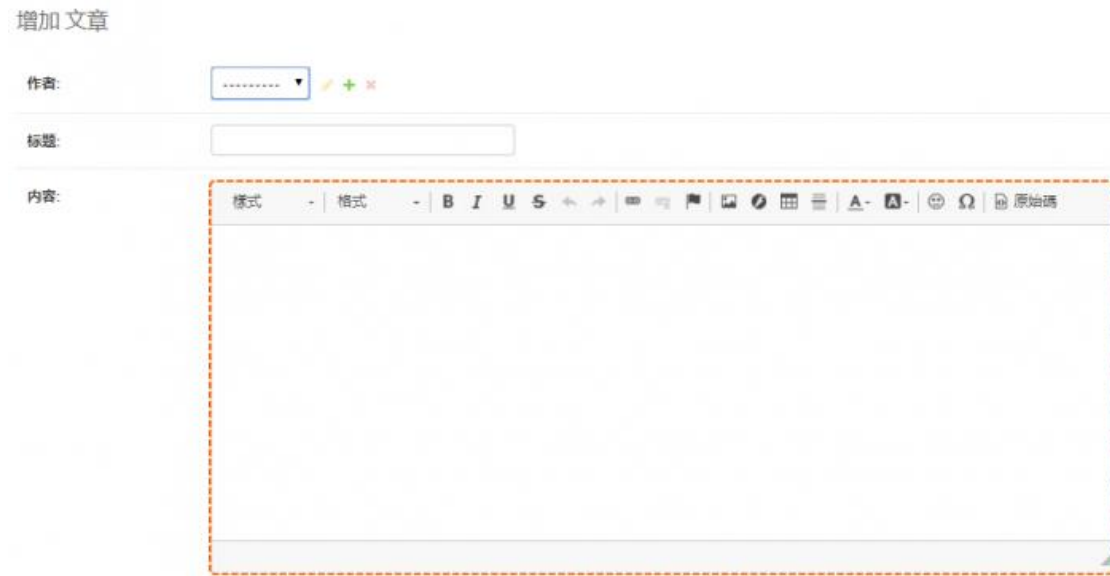
原文链接：<http://www.opython.com/1252.html>

Django2 练习项目：开发个人博客系统（12）

这一篇教程，我们一起学习如何在 Django2 项目中使用富文本编辑器 Django-CKEditor。

提示：另一款富文本编辑器“SummerNote”的使用教程请参考《Django2：Web 项目开发入门笔记（26）》。

例如，我们在 Django 后台中编辑文章内容时，就需要富文本编辑器。



一、安装

如果使用 Django-CKEditor，我们需要先安装依赖库 Pillow。

```
pip install pillow
```

然后，安装 Django-CKEditor。

```
pip install django-ckeditor
```

二、设置

完成 Django-CKEditor 的安装后，打开文件“settings.py”，添加如下设置。

1、“INSTALLED_APPS”中添加“ckeditor”和“ckeditor_uploader”。

示例代码：

```
INSTALLED_APPS = (  
    ...省略部分代码...  
    'ckeditor',
```

```
'ckeditor_uploader'  
)
```

2、添加 Media 相关配置

示例代码：

```
MEDIA_URL = '/media/'  
MEDIA_ROOT = os.path.join(BASE_DIR, 'media')
```

3、添加 CKEditor 文件上传路径。

示例代码：

```
CKEDITOR_UPLOAD_PATH = "upload/" # 注意只有右侧带有“/”
```

4、添加 CKEditor 加载配置。

示例代码：

```
CKEDITOR_CONFIGS = {  
    'default': { # 添加默认配置  
        },  
    'mycfg': { # 添加自定义配置  
        'skin': 'moono', # 设置皮肤  
        'toolbar': 'full', # 设置工具栏加载全部功能  
        'width': '100%', # 设置宽度  
    },  
}
```

可以预先定义多种配置（例如上方代码中的“mycfg”），以便在不同的应用场景中调用不同的配置。

更多配置项请参考：<https://pypi.org/project/django-ckeditor/#example-ckeditor-configuration>

关于 CKEditor 的简体中文语言设置：

作者电脑上显示默认繁体中文，检查后发现原因在于 CKEditor 会使用“settings.py”文件中的语言设置。

Django 使用简体中文时，我们的设置为“LANGUAGE_CODE = ‘zh-Hans’”，这是因为 Django2 语言文件夹中没有“zh-cn”文件夹，简体中文是“zh-Hans”文件夹，直接设置“LANGUAGE_CODE = ‘zh-cn’”会导致异常。

但是这样的设置又会导致 CKEditor 找不到名为“zh-Hans.js”的语言文件，当找不到文件时，CKEditor 会找到第一个名称包含“zh”的 js 文件去调用。

所以，解决方案是修改 Django 语言文件夹或者 CKEditor 语言文件的名称，让他们保持一致。

不过 CKEditor 的语言文件的名称在 PyCharm 中提示不能更改，最终只好修改 Django 的简体中文语言文件夹名称为“zh-cn”，并将“settings.py”文件中的语言设置改为“LANGUAGE_CODE = ‘zh-cn’”。

提示：Django 语言文件夹所在路径为：(venv)\Lib\site-packages\django\conf\locale

另外一种解决方案是将繁体中文的语言文件“zh.js”内容替换成了“zh-cn.js”中的简体内容。

示例代码：

```
CKEDITOR.lang[
    'zh'
]={"editor": "所见即所得编辑器"...省略后方代码...
```

注意上方代码中标红的部分，要在替换了内容之后改回“zh”。

5、URL 配置

打开文件“urls.py”，在“urlpatterns”列表中，包含 CKEditor 的 URL 配置。

示例代码：

```
path('ckeditor/', include('ckeditor_uploader.urls')),
```

三、使用

打开文件“models.py”，导入 ckeditor 模块中的字段类，修改文章类“Article”中的“content”字段。

示例代码：

```
from ckeditor.fields import RichTextField

class Article(models.Model):
    ...省略部分代码...
    content = RichTextField('内容')
    ...省略部分代码...
```

在上方代码中，使用了“RichTextField”字段，默认会调用“settings.py”中的“default”设置，如果需要使用我们添加的自定义设置“mycfg”，可以在“RichTextField”的参数中进行指定。

示例代码：

```
content = RichTextField('内容',config_name='mycfg')
```

通过以上步骤的设置，我们就可以在 Django 后台中使用富文本编辑器了。

如果在我们自己编写的页面中使用 CKEditor，需要在模板中（例如“base.html”）添加如下代码。

示例代码：

```
<script src="{% static '/ckeditor/ckeditor/ckeditor.js' %}"></script>
<script src="{% static '/ckeditor/ckeditor-init.js' %}"></script>
```

不过，通过以上的操作，我们还不能够在富文本编辑器中使用图片上传的功能。

四、设置图片上传

首先，打开文件“\venv\Lib\site-packages\ckeditor\static\ckeditor\ckeditor\config.js”（本案例使用了虚拟环境）。

在打开的文件中添加配置。

示例代码：

```
CKEDITOR.editorConfig = function (config) {

    config.filebrowserImageUploadUrl = "/upload/";

};
```

上方代码中，红色部分为新增代码。

添加这一段代码之后，在富文本编辑器中点击图片的图标会出现新的标签“上传”。

然后，我们还需要在视图文件“views.py”中编写一段上传文件的代码。

但是，因为 Django 自带防止跨域攻击的检查功能，还会导致上传会失败，所以，我们为这一段代码禁用跨域攻击检查。

示例代码：


```
import time
from django.shortcuts import Http404 # 导入 404 异常类
from django.views.decorators.csrf import csrf_exempt # 导入禁用跨域攻击检查的装饰器

@csrf_exempt # 装饰上传图片的视图函数
def image_upload(request): # 定义上传图片的视图函数
    if request.method == 'POST':
        callback = request.GET.get('CKEditorFuncNum') # 获取客户端上传事件的标记
        try:
            path = "media/upload/" + time.strftime("%Y%m%d%H%M%S", time.localtime())
            f = request.FILES["upload"] # 获取上传文件的对象
            file_name = path + "_" + f.name # 组织文件存储路径与名称
            with open(file_name, "wb+") as file: # 创建文件
                for chunk in f.chunks(): # 读取上传文件
                    file.write(chunk) # 写入文件
        except Exception as e:
            print(e)
            res = "<script>window.parent.CKEDITOR.tools.callFunction(" + callback + "," + "/" +
file_name + "," + ");</script>"
            # 将上传文件的路径通过上传事件的标记写回浏览器客户端
            return HttpResponse(res) # 返回响应内容
        else:
            raise Http404() # 抛出异常
```

添加完上述代码之后，我们进行 URL 配置。

示例代码：

...省略部分代码...

```
from django.urls import re_path
from . import settings
from django.views.static import serve
```

```
urlpatterns = [
    ...省略部分代码...
    path('upload/', blog_view.image_upload),
    re_path(r'^media/(?P<path>.*)$', serve, {'document_root': settings.MEDIA_ROOT})
]
```

最后，在项目根目录下添加“media”文件夹并在“media”文件夹下添加“upload”文件夹，以保证文件能够正常上传。

当然，“upload”文件夹也可以在图片上传的视图函数中通过代码创建。

到这里，我们就能够正常的使用图片上传功能了。

转载请注明：魔力 Python » Django2 练习项目：开发个人博客系统（12）

原文链接：<http://www.opython.com/1257.html>