# 第九章 Django模板系统\_静态文件处理和模板包含、继承

本节所讲内容：

9.1 Django 模板加载的优化方法

9.2 Django APP功能开发

9.3 Django 模板的包含和继承

9.4 Django模板静态文件加载

## 9.1 Django 模板加载的优化方法

在之前的学习过程当中讲解了类似的代码

修改【D:xuegod\_code/3-10/OurBlog/views.py】

插入：

from django.http import HttpResponse

from django.template.loader import get\_template

Navigation = [

{"id": 1, "label": "python", "href": "python", "parent\_id": 0},

{"id": 2, "label": "java", "href": "python", "parent\_id": 0},

{"id": 3, "label": "php", "href": "python", "parent\_id": 0},

{"id": 4, "label": ".net", "href": "python", "parent\_id": 0},

{"id": 5, "label": "django", "href": "python", "parent\_id": 1},

{"id": 6, "label": "flask", "href": "python", "parent\_id": 1},

{"id": 7, "label": "spring", "href": "python", "parent\_id": 2},

{"id": 9, "label": "xadmin", "href": "python", "parent\_id": 5},

{"id": 9, "label": "django\_ckeditor", "href": "python", "parent\_id": 5},

]

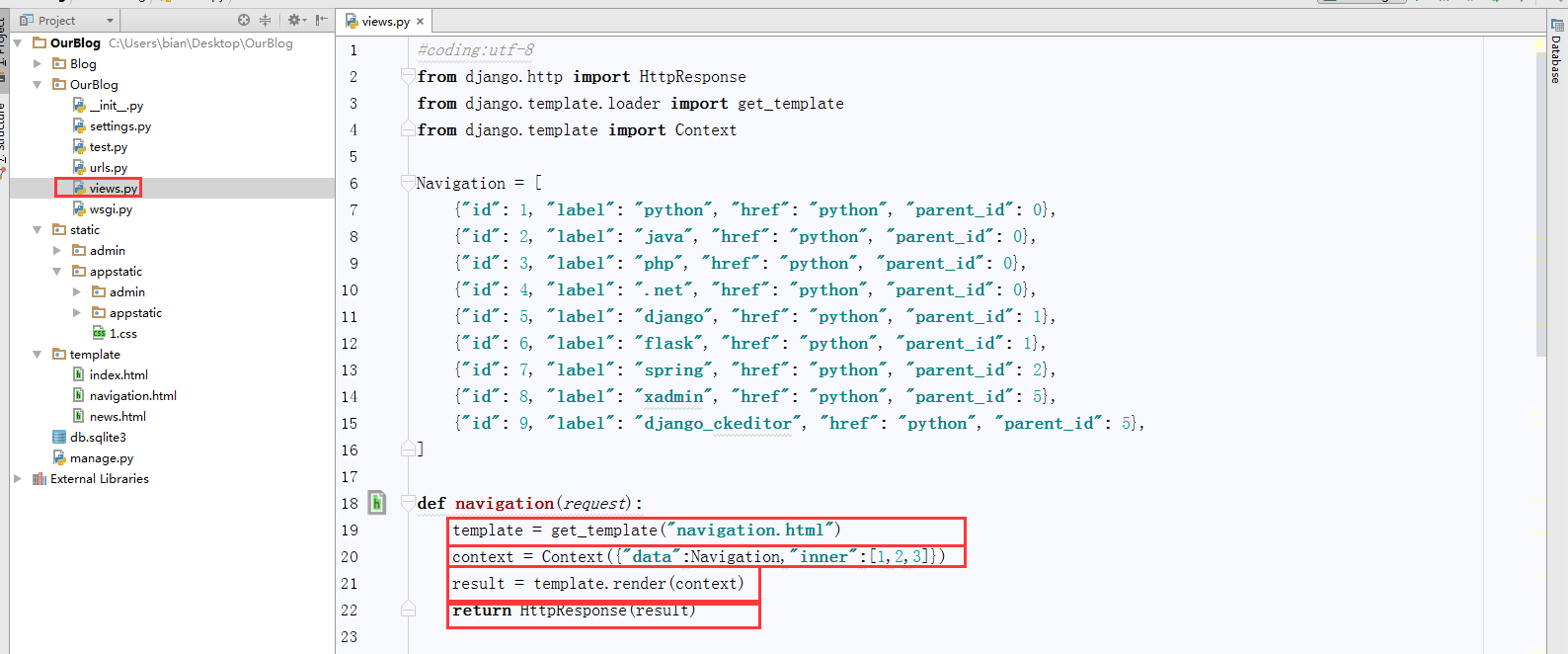
def navigation(*request*):

template = get\_template("navigation.html")

context = {"data":Navigation,"inner":[1,2,3]}

result = template.render(context)

return HttpResponse(result)

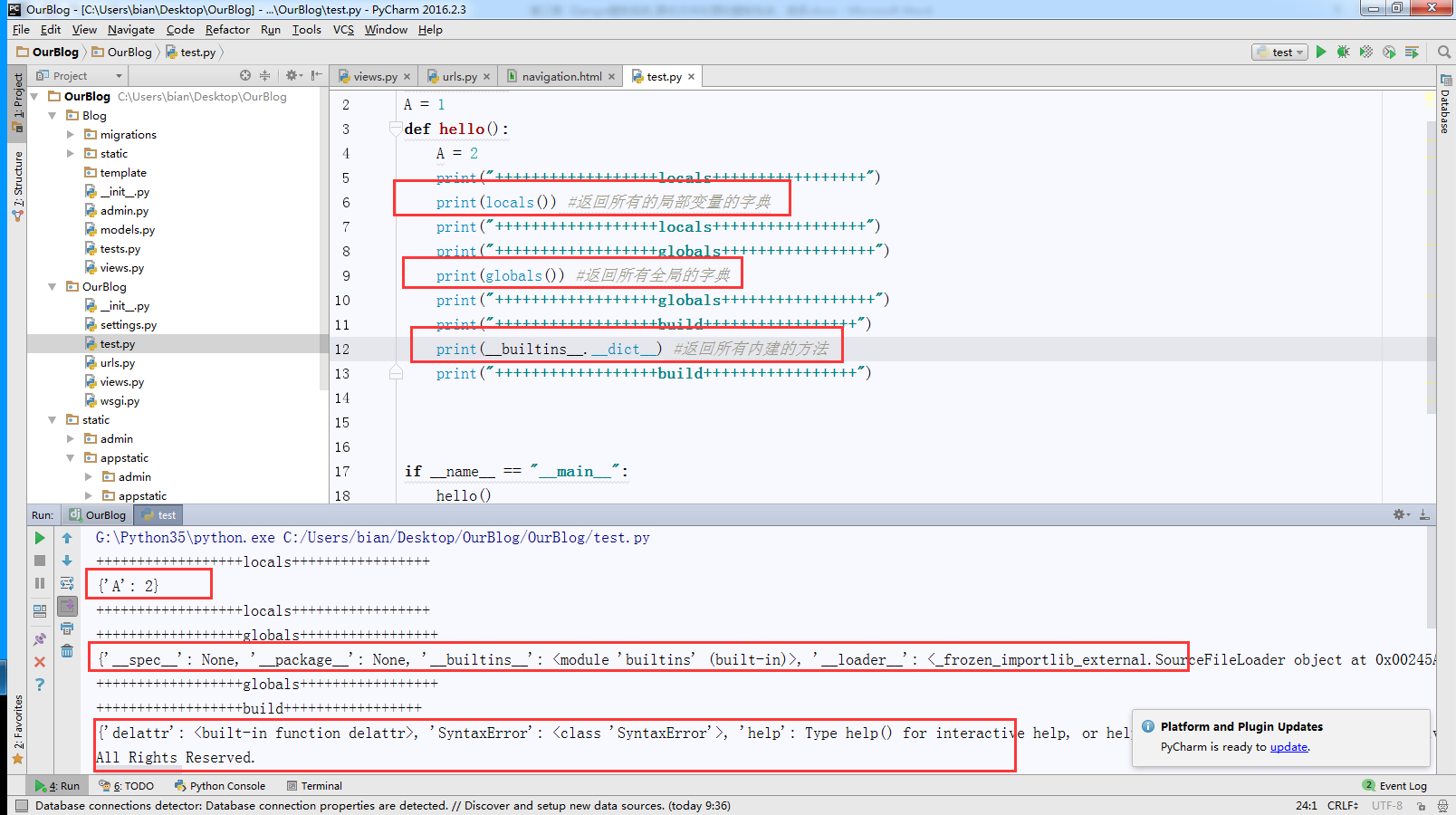


这几行代码实现了基本的前后端分离，但是在工作当中还有两种更加方便的方法，

在学习这两种方法之前，首先来看几个python内置的方法

修改【D:xuegod\_code/3-10/OurBlog/OurBlog/test.py】

插入：



local() 返回函数当中所有的局部变量

globals() 返回函数当中所有的全局变量

\_\_builtins\_\_ 返回函数当中所有的内置方法

render

from django.shortcuts import render

Navigation = [

{"id": 1, "label": "python", "href": "python", "parent\_id": 0},

{"id": 2, "label": "java", "href": "python", "parent\_id": 0},

{"id": 3, "label": "php", "href": "python", "parent\_id": 0},

{"id": 4, "label": ".net", "href": "python", "parent\_id": 0},

{"id": 5, "label": "django", "href": "python", "parent\_id": 1},

{"id": 6, "label": "flask", "href": "python", "parent\_id": 1},

{"id": 7, "label": "spring", "href": "python", "parent\_id": 2},

{"id": 9, "label": "xadmin", "href": "python", "parent\_id": 5},

{"id": 9, "label": "django\_ckeditor", "href": "python", "parent\_id": 5},

]

def navigation(*request*):

nav = Navigation

return render(*request*,"navigation.html",locals()， context = Context({"data":Navigation,"inner":[1,2,3]}))

这里render方法代替了上面的一大段方法，实现了同样的效果，使用render一定注意的点在于他的第一个参数必须是request也就是请求，也正是这个原因，如果在django当中发起post请求（后面form表单详细的讲），会使用render。

render\_to\_response

from django.shortcuts import render\_to\_response

Navigation = [

{"id": 1, "label": "python", "href": "python", "parent\_id": 0},

{"id": 2, "label": "java", "href": "python", "parent\_id": 0},

{"id": 3, "label": "php", "href": "python", "parent\_id": 0},

{"id": 4, "label": ".net", "href": "python", "parent\_id": 0},

{"id": 5, "label": "django", "href": "python", "parent\_id": 1},

{"id": 6, "label": "flask", "href": "python", "parent\_id": 1},

{"id": 7, "label": "spring", "href": "python", "parent\_id": 2},

{"id": 9, "label": "xadmin", "href": "python", "parent\_id": 5},

{"id": 9, "label": "django\_ckeditor", "href": "python", "parent\_id": 5},

]

def navigation(*request*):

nav = Navigation

return render\_to\_response("navigation.html",locals(),context = Context({"data":Navigation,"inner":[1,2,3]})

)

同样render\_to\_response也可以实现这样的效果，而且不用传入request，当然如果是form请求是不可以用的

## 9.2 Django APP功能开发

Django鼓励开发者采用松耦合的开发，就是将整个网站划分为几个独立的功能点进行独立开发，这样开发的好处就在于以下几点：

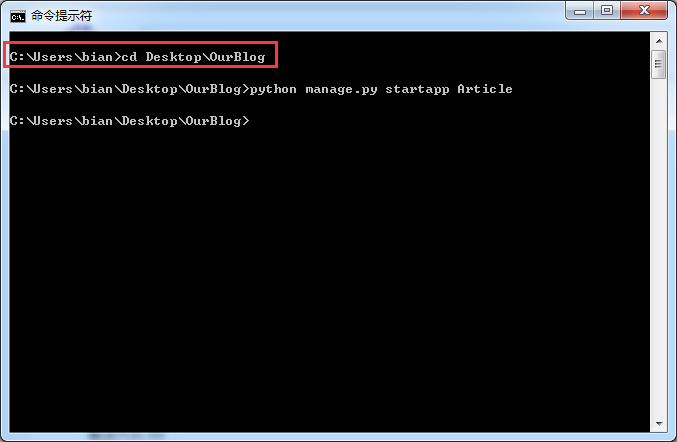
1、代码修改容易，不会出现 修改用户逻辑导致购物车出现bug这样的情况

2、便于功能的复用和迁移，在工作当中好多完整是有共性的，当开发者第二次或者更多次进行开发的时候，可以之间使用写好的功能，不必重新编写。

正是出于以上优点的设计，django把数据开发安排到了一个叫做APP的组件里面，而app就是django提供给开发者编写自己也为逻辑的组件。所以单单django项目的框架是没有办法实现django数据库开发的，进行django数据库开发首先要进行APP创建。

创建APP

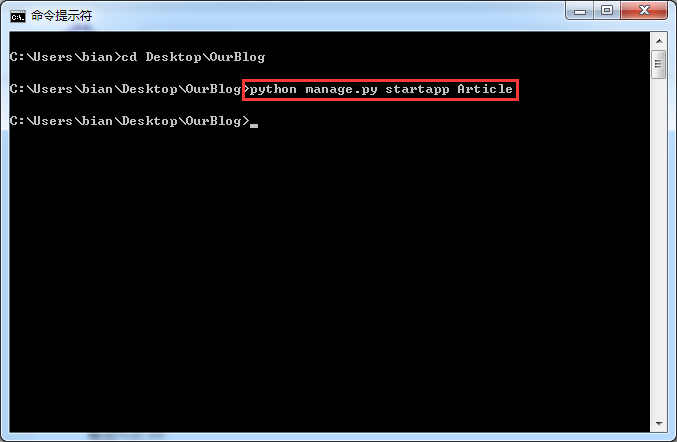
1. 首先在命令行当中切入到项目的根目录



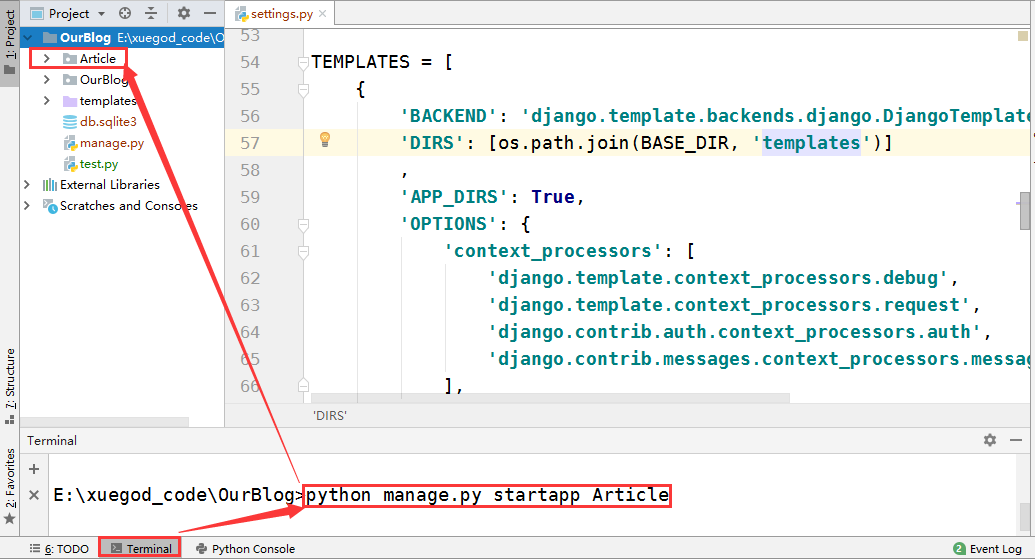
2. 然后在根目录先使用命令创建app

Python manage.py startapp Article

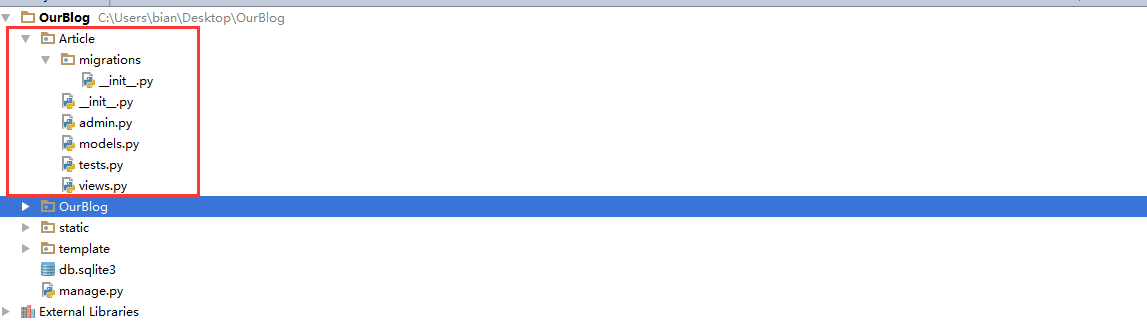
startapp是创建应用的命令 Article 是我们的app应用



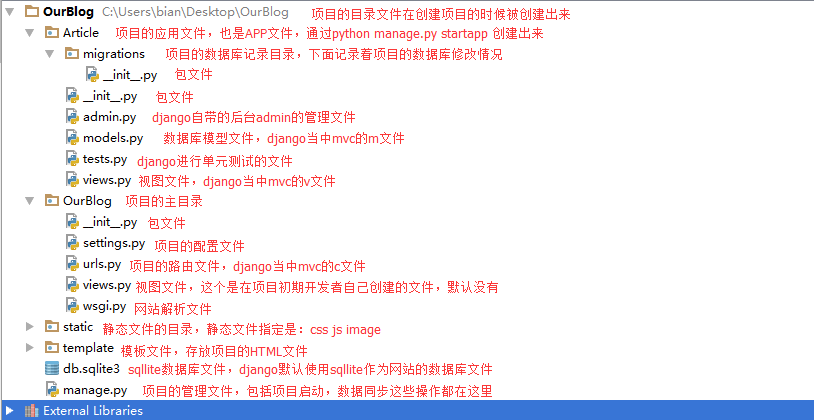
也可以在pycharm中的terminal中输入：python manage.py startapp Article 进行创建如下图所示：



然后看看django APP的目录



这里来认识一下当前项目的结构



到这里app就创建完成了，开发者就可以进行django的开发了。

9.2 Django app中模板文件加载

对于我们来说django app中的模板加载是必须要掌握的功能，其操作主要可总结成以下几个步骤。

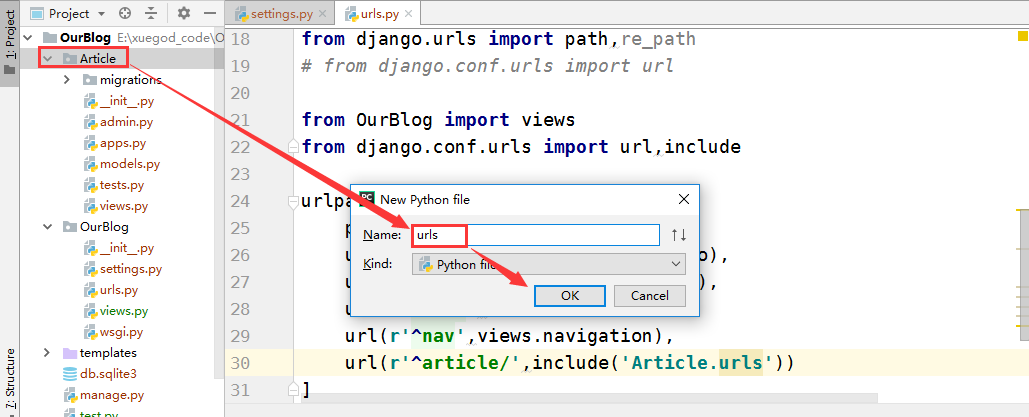
1. 生成app文件
2. 在settings中的INSTALL\_APPS中添加（这个为了方便orm模板映射）
3. 在总urls.py中配置下url让程序能够通过url路由找到这个app
4. 配置app 中的views.py 指向已经生成的HTML模板渲染

9.2.1 在settings中的INSTALL\_APPS中添加（这个为了方便orm模板映射）

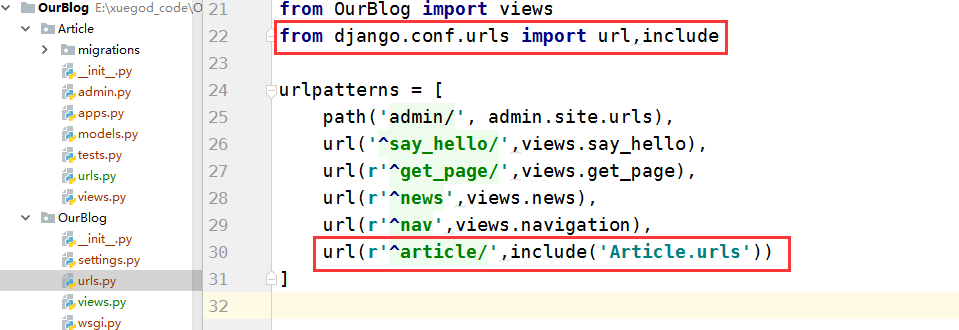


9.2.2 配置url

1、创建app中的urls



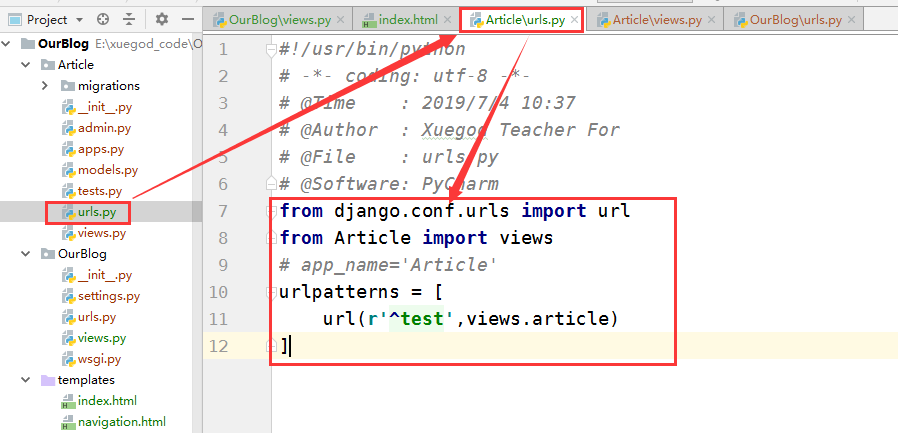
2、配置总路由



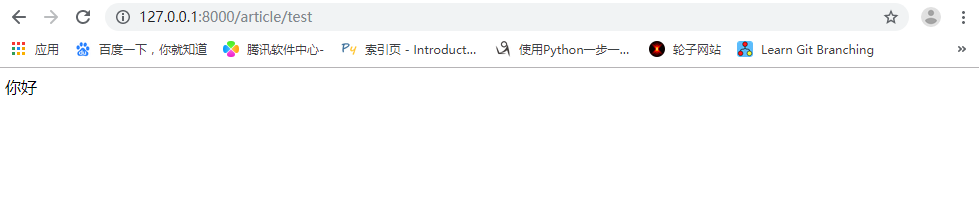
3、编写app中views.py



3、配置app路由



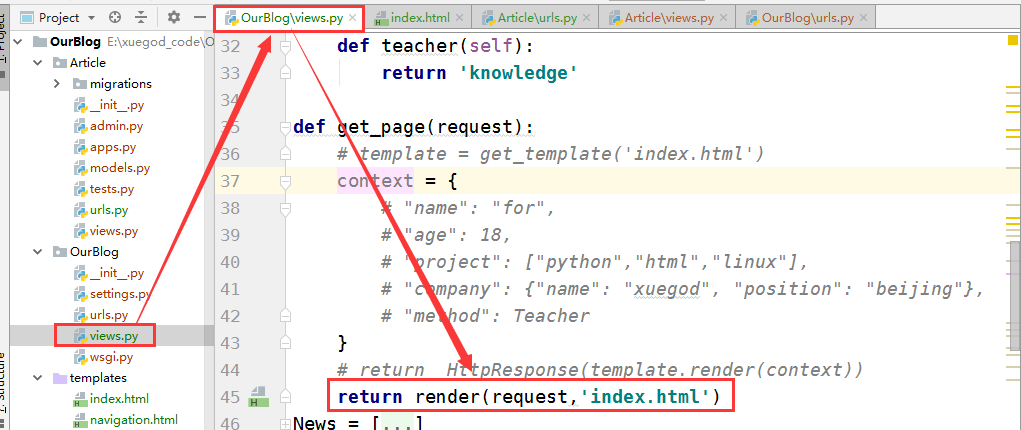
运行结果如下：



9.2.3 url反向解析

在HTML阶段我们学过a标签，a标签中的href引用，可以直接跳转的其他网站，接下来我们用一个简单的例子给大家看一下：

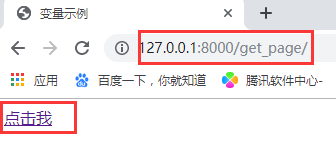
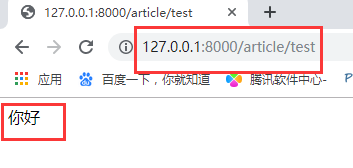
首先我在E:\xuegod\_code\OurBlog\OurBlog\views.py修改get\_page中的配置



在index.html中写个a标签如图所示

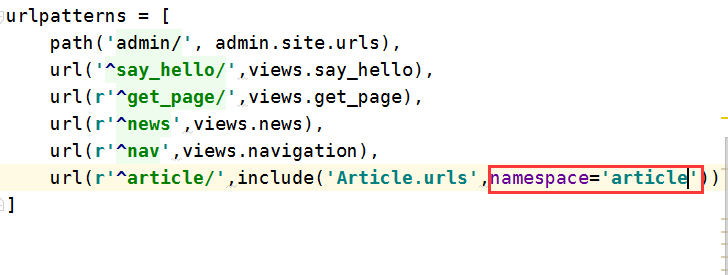


运行结果如下：

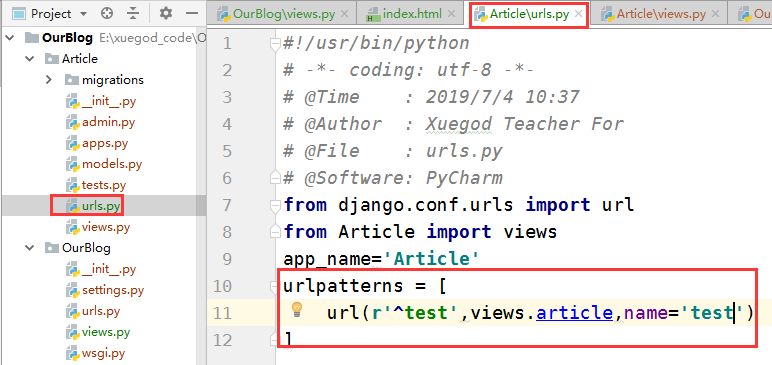
然后在生产环境下我们往往会使用url反向解析，什么叫url反向解析呢？请看如下示例：

首先在总路由中的include中加入namespace = ‘article’

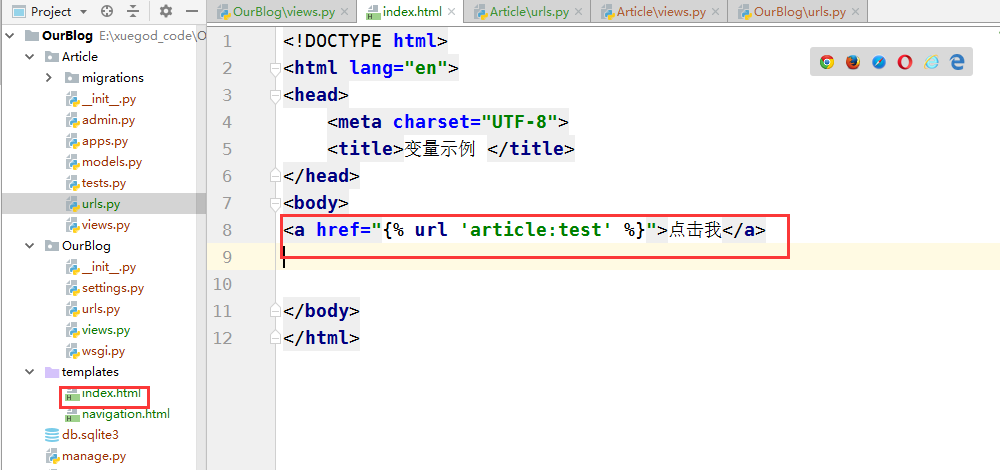


然后在app中的url路由中配置name

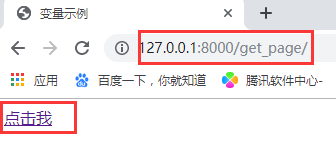
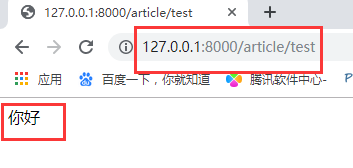
在urlpatterns上配置写个app\_name=’app名’



紧接着修改index.html文件中的a标签中的href="{% url 'article:test' %}"



运行结果如下：

## 9.3 Django 模板的包含和继承

上面的课程完成了django最基本的网站模板设置，但是，认真的去观察现在的网站，尤其是企业站和电商，会发现整个完整有n个页面，但是样式大同小异，比如说导航栏、比如页面的尾部，比如侧边栏。这样看来这些网页是有冗余的，为了代码的一致性和避免重复造轮子，django推出了两个标签

include 包含

指的是将一个网友加载到另一个网页当中

Views.py

from django.shortcuts import render\_to\_response

def framework\_page(*request*):

return render\_to\_response("framework\_page.html")

urls.py

from OurBlog.views import framework\_page

urlpatterns = [

url(r'^framework\_page', framework\_page),

]

include\_page.html

<!DOCTYPE html>

<html lang="en">

<head>

<meta charset="UTF-9">

<title>Title</title>

</head>

<body>

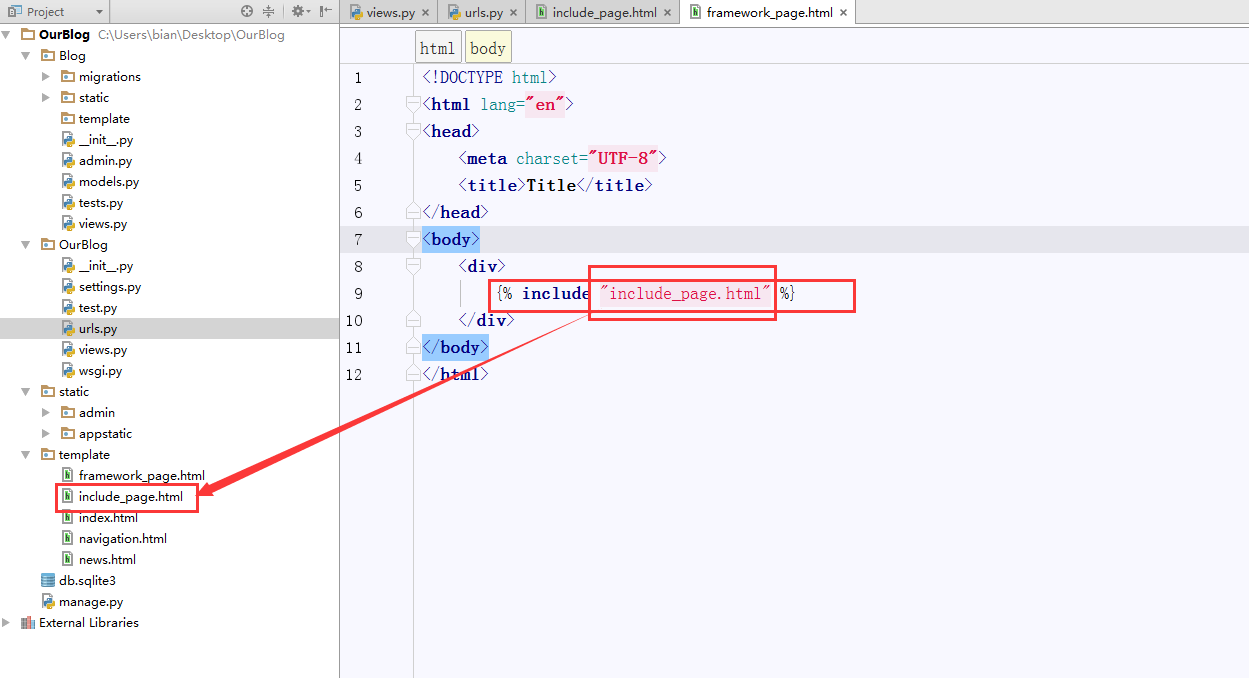
<div style = "background-color: green;">

</div>

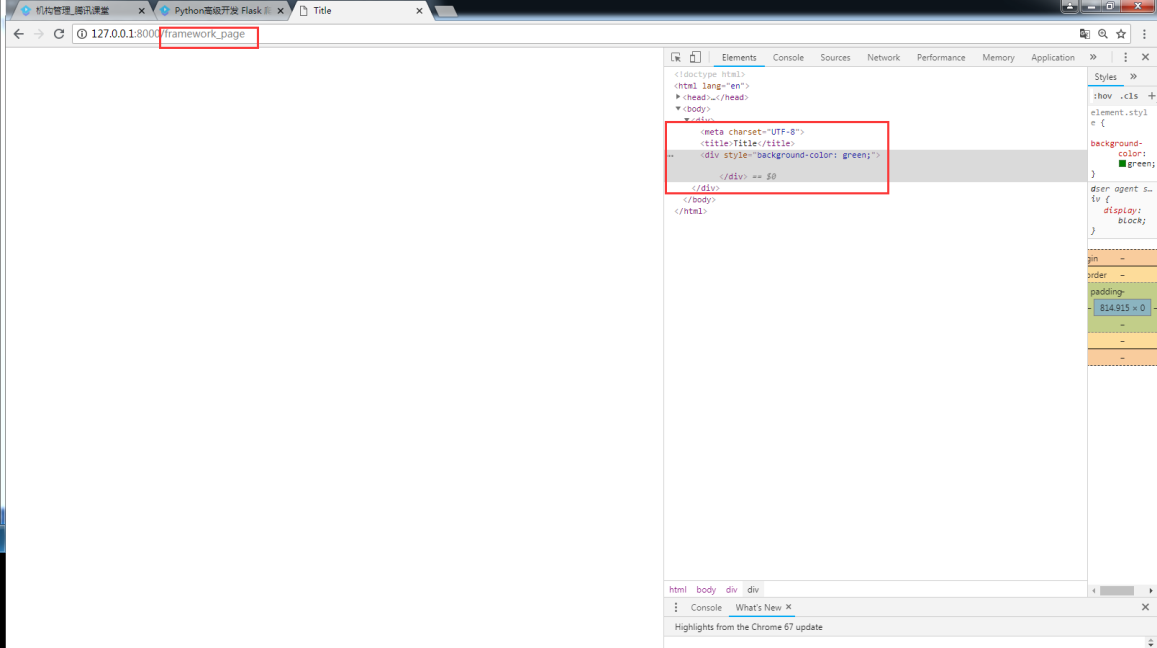
</body>

</html>

Framework\_page.html



这样就把include\_page.html的内容放到了framework\_page.html当中了，效果如下：



extends 继承

相比上面的包含，继承需要先设置好被继承的页面，通常命名为base.html，base.html当中编写了其他网页共性的东西，比如说抬头，比如说结尾，而其中其他页面需要修改，个性的东西被block标签包围，标签里的内容为默认内容。

Views.py

from django.shortcuts import render\_to\_response

def extend\_base(*request*):

return render\_to\_response("extend\_base.html")

urls.py

from OurBlog.views import extend\_base

urlpatterns = [

url(r'^extend\_base', extend\_base),

]

base.html

<!DOCTYPE html>

<html lang="en">

<head>

<meta charset="UTF-9">

<title>

{% block title %}

Title

{% endblock %}

</title>

</head>

<body style = "background-color: #0f74a9;color: white;">

{% block content %}

{% endblock %}

</body>

</html>

extend\_base.html

{% extends "base.html" %}

{% block title %}

extends base

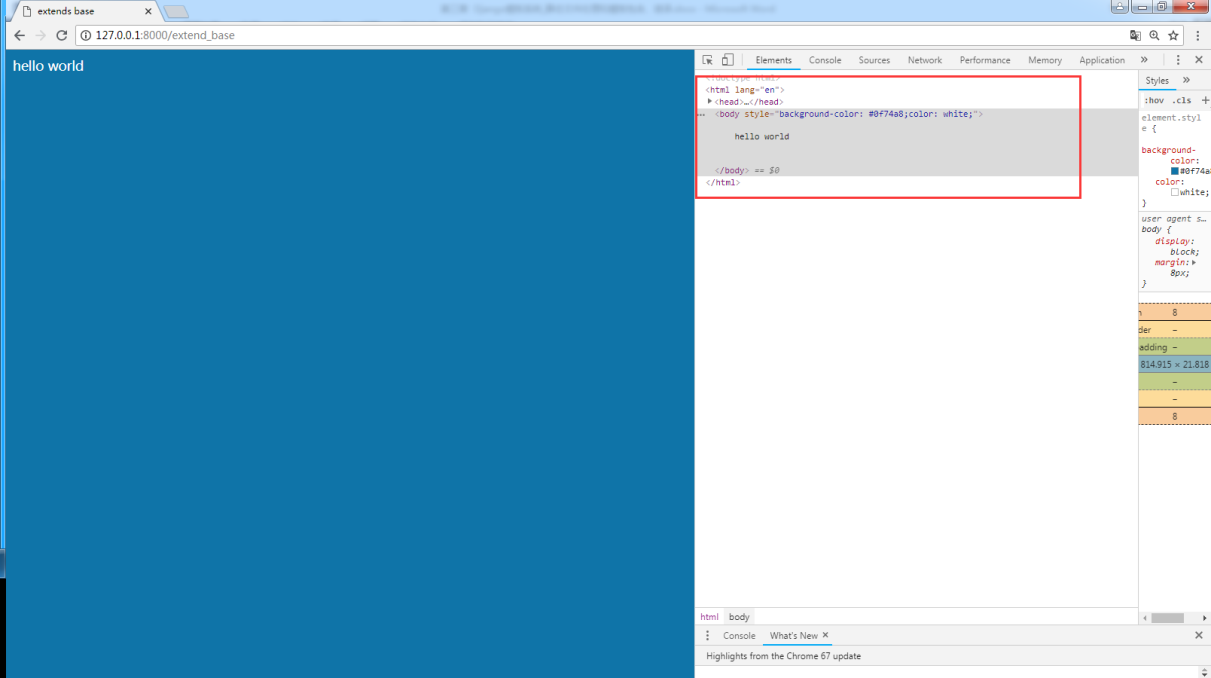
{% endblock %}

{% block content %}

hello world

{% endblock %}

效果



在这里，extend\_base.html继承了base.html，并且根据自己的需求修改了title和content块儿，这两个块儿在base页当中定义，在extend\_base.html也修改，其中title有默认值，加入在extend页面当中不进行修改，就采用默认值

## 9.4 Django模板静态文件加载

上一章的课程当中详细的讲解的讲解了Django模板的基础加载和Django模板的基础语法，现在又有问题出现了，在工作当中，网站的前端文件不单单有HTML语句，还需要：图片、Css脚本、JavaScript脚本、前端框架等等。

那么，在web开发的过程当中，统一的将这些文件称之为静态文件，当前端人员提交页面给django后端开发的时候，前端的开发是按照自己的开发环境添加的这些外部文件的链接的，Django后台开发人员开开心心的把前端开发提交文件存放在template下，却发现整个世界都变了如下：

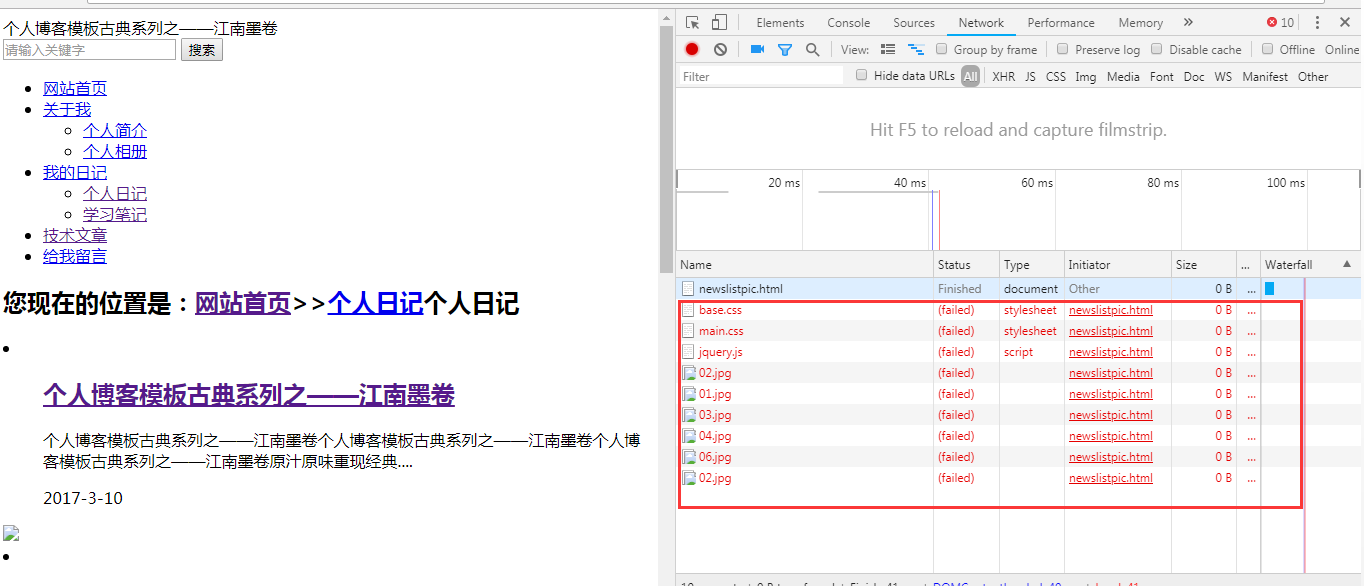
心中的页面：



实际的页面：



打开浏览器调试器



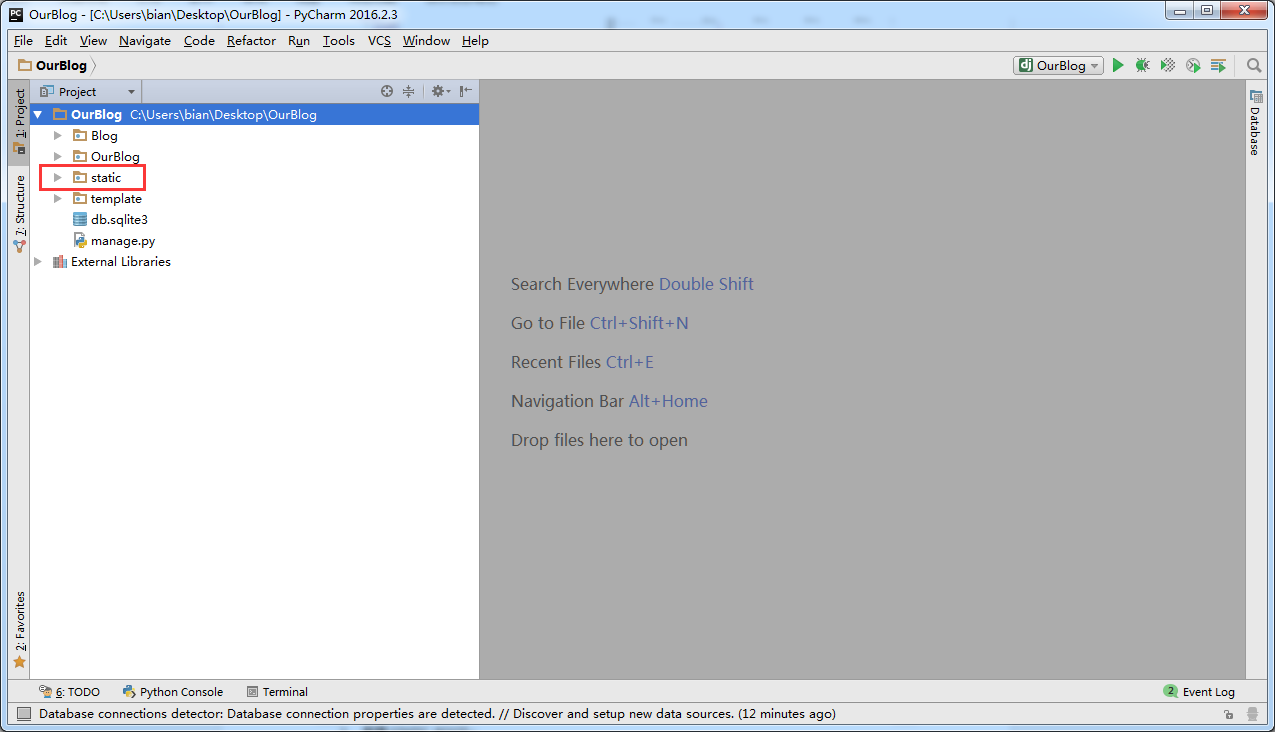
静态文件没有加载，由于路径的变化，之前的路径已经不存在了，所以需要重新配置静态文件，静态文件的配置方法在其他的地方介绍有多种，但在工作当中使用的通常有两种

1. 配置STATICFILES\_DIRS

在从django的1.1版本开始，django的配置文件当中就有一条配置叫做STATICFILE\_DIRS用来指出static文件的路径，配置如下

1. 创建static目录在项目的根目录下(也就是manage.py的目录下)

在【D:/xuegod/OurBlog/】下新建一个static目录



2、在settings当中配置STATICFILES\_DIRS配置项，

（这里有两个注意的点：1、STATICFILES\_DIRS 不可以写错，2.配置的参数通常是一个单元素的元组，元组单元素需要添加逗号）

修改【D:xuegod/OurBlog/OuBLog/settings.py/】

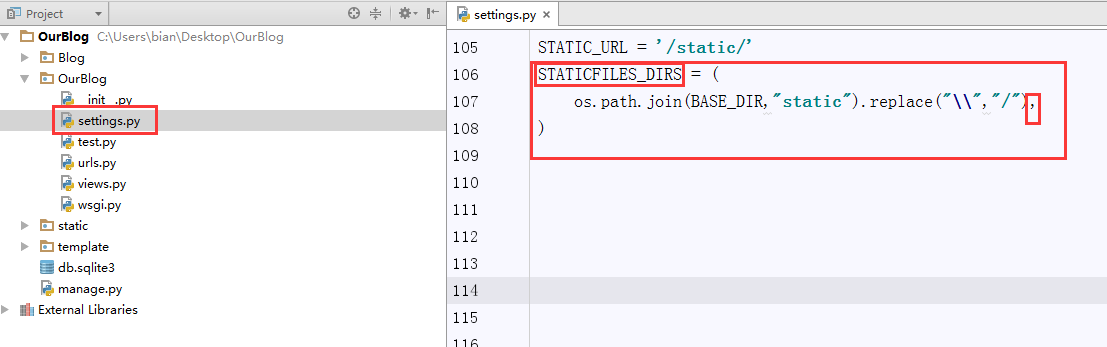
在第106行插入：

STATICFILES\_DIRS = (

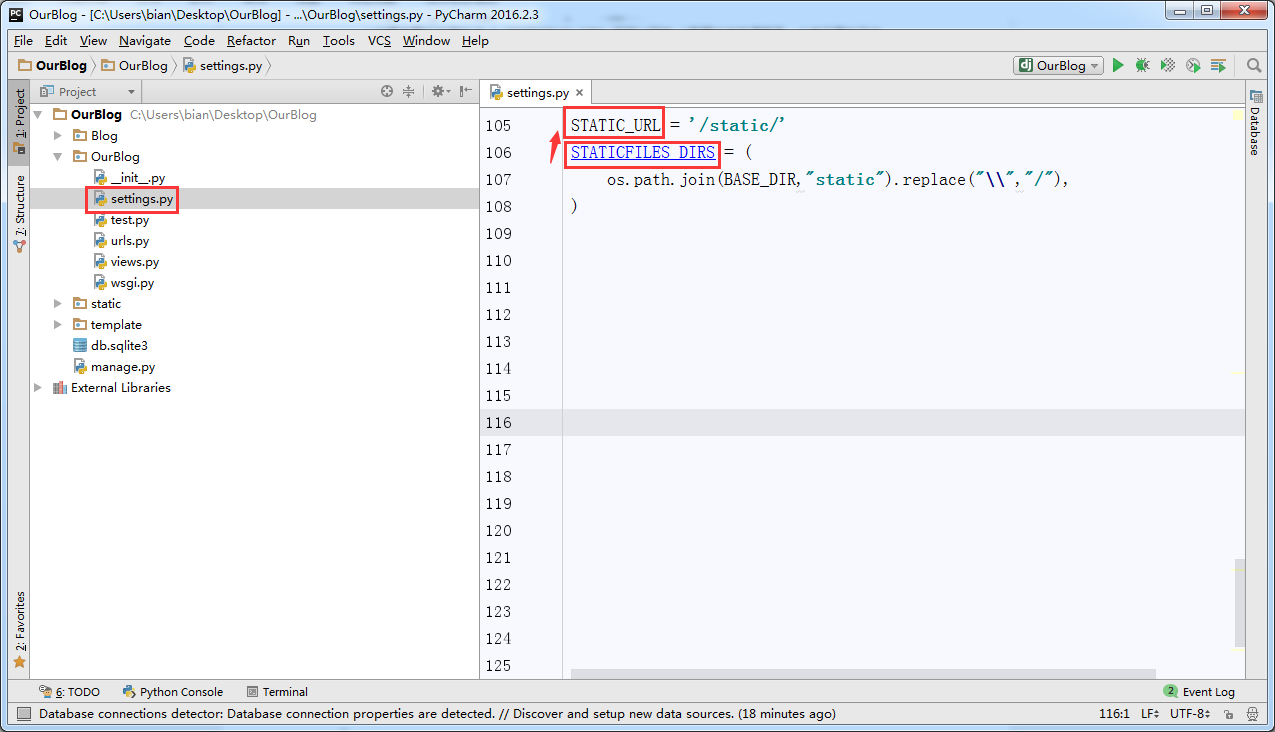
os.path.join(BASE\_DIR,"static").replace("\\","/"),

)

如下图：



2、django的settings当中STATIC\_URLS是搭配给静态文件使用的，可以理解为STATIC\_URLS是STATICFILES\_DIRS的路由，在HTML当中我们用STATIC\_URLS的配置项来指代STATICFILES\_DIRS当中指定的路径。



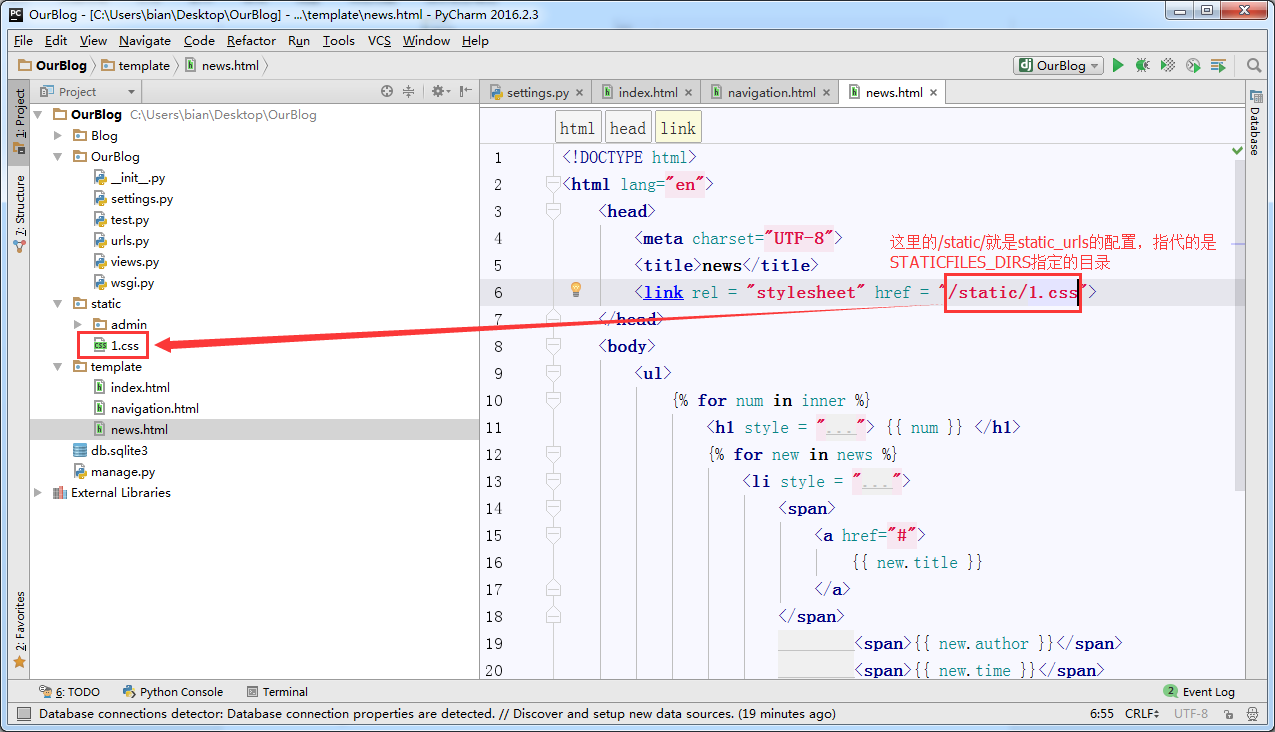
1. 前端按照配置拼接路径

修改【D:xuegod\_code/3-10/OurBlog/template/news.html】

第6行插入：

<link rel="stylesheet" href="/static/1.css">

效果如下图：



1. 配置STATIC\_ROOT

在django 1.3版本之后，官方推出了STATIC\_ROOT这条配置项，如同字面上的意思。保存在这个目录里面文件被当成静态文件处理。不过，千万不要把自己辛苦写的JavaScript、图片等静态文件放到里面去，接着看下去就知道了。与STATIC\_ROOT搭配使用的还有STATIC\_URL与STATICFILES\_DIRS两个变量。STATIC\_URL一般用默认的/static/，用于指定的静态目录的URL。STATICFILES\_DIRS则指定一个工程里面哪个目录存放了与这个工程相关的静态文件，它是一个列表。假定在STATICFILES\_DIRS有一个目录是"/dolphin/besteam/pages/static"，其中有一个文件是common.js。那么，在浏览器里面输入http://localhost:9000/static/common.js就可以访问这个JavaScript文件了。

更深层次的理解，这个东西其实是用来方便布署Django App的。考虑一下我们编写Django App的情况，经常会有一些静态的文件(JavaScript、图片等)。为了方便，把这些文件放置到App下面的static目录里面。每个App都有自己的子目录，所以在Django 1.3之前，布署的时候就麻烦了。有三种选择：

一是使用django.views.static.serve来处理文件。在App的urls.py里面加上一条：

url(r"^(?/static/P<path>.\*)$", "django.views.static.serve",\

{"document\_root" : "/path/to/project/app/static/"})

每个App都要加入这条纪录，这样做的话，静态经过了Django的处理，速度太慢。

二是让用户手动写Apache的映射。将/static/映射到/path/to/project/app/static。这种方法更麻烦，不仅URL不能冲突，而且破坏了App的代码独立原则。作为布署者，每个App的情况都要相当熟悉，不能遗落(可以强制每个App都使用static目录)。

三是由布署者将每个App内的static目录内的文件复制到一个指定的目录。然后重写Apache，将/static映射到这个目录。这个方案可以说兼有第一方案与第二方案的优点，已经接近staticfiles了。缺点是手动操作，还是太麻烦了一点。

新的Django提供了一个方法自动地将所有的静态文件放在一起。只要在写App的时候创建一个static子目录专门保存静态文件就行了。在开发阶段，不必费心去做映射，不需要配置urls.py。在布署到生产环境的时候，只需要配置Apache把/static/映射到STATIC\_ROOT。然后运行manage.py collectstatic，自动地STATICFILES\_DIRS列出的目录以及各个App下的static子目录的所有文件复制到STATIC\_ROOT。因为复制过程可能会覆盖掉原来的文件，所以，一定不能把我们辛苦做出来静态文件放这边！

在开发阶段，Django把/static映射到django.contrib.staticfiles这个App。staticfiles自动地从STATICFILES\_DIRS、STATIC\_ROOT以及各个App的static子目录里面搜索静态文件。一旦布署到开发环境上，settings.py不需要重新编写，只要在Apache的配置文件里面写好映射，/static将会被Apache处理。django.contrib.staticfiles虽然仍然存在，但因为不会接收到以/static/开始的路径，所以将不会产生作用。不必担心Django会使用处理速度变慢。另外，当settings.DEBUG is False的时候，staticfiles将自动关闭。

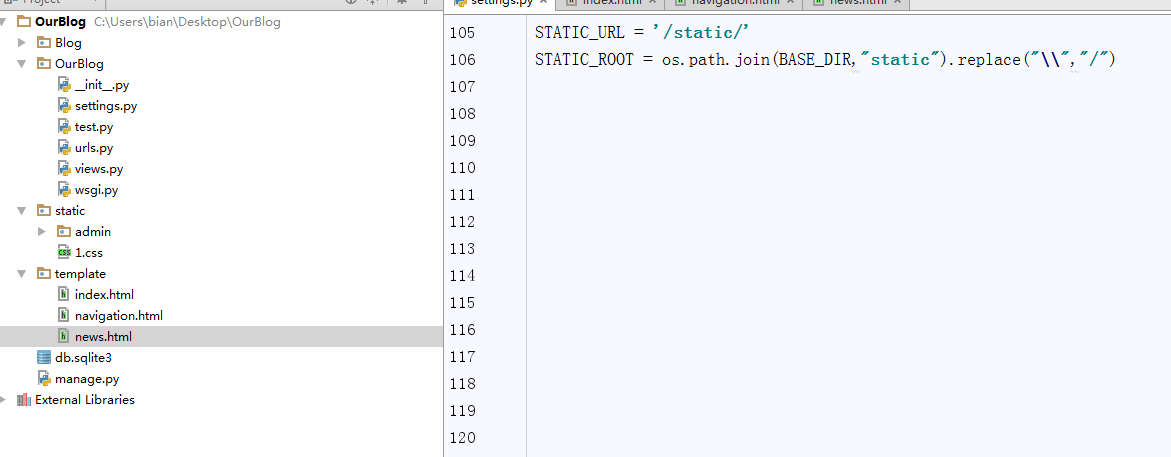
修改【D:xuegod\_code/3-10/OurBlog/OurBlog/settings.py】

在第105-106行插入：

STATIC\_URL = '/static/'

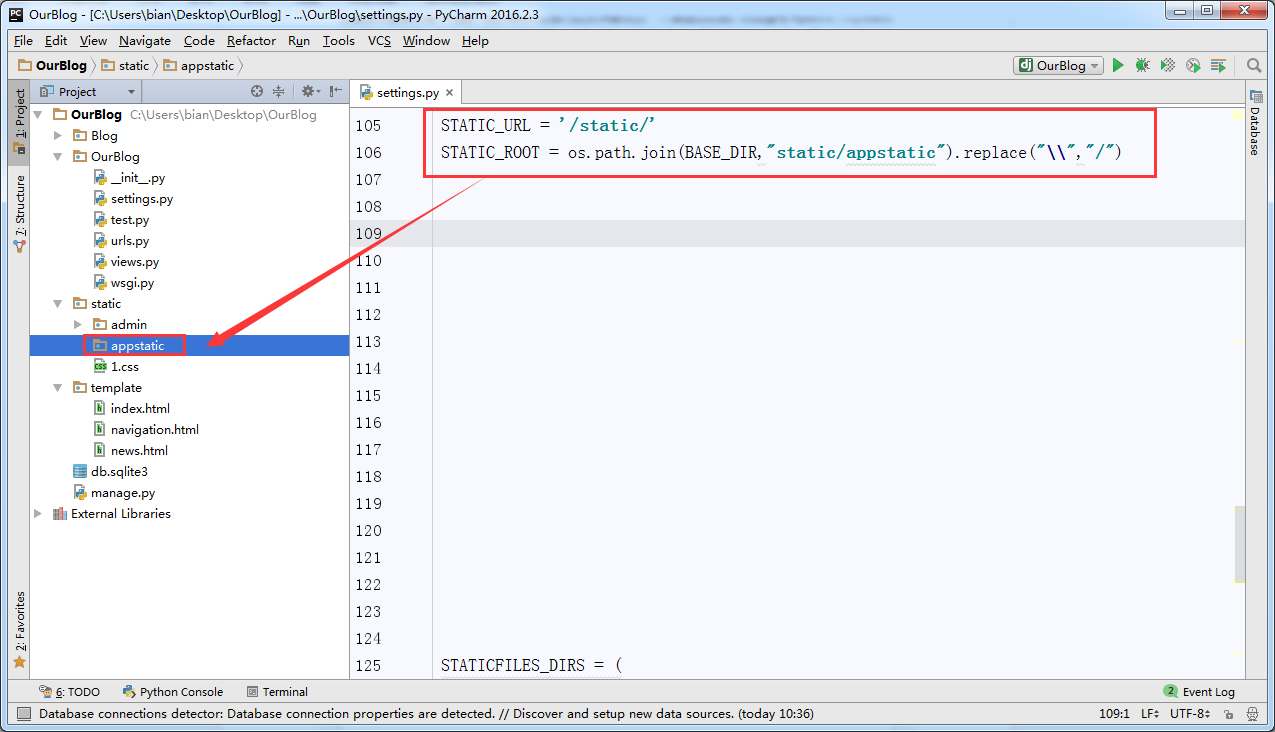
STATIC\_ROOT = os.path.join(BASE\_DIR, "static/appstatic").replace("\\", "/")

效果如下图：



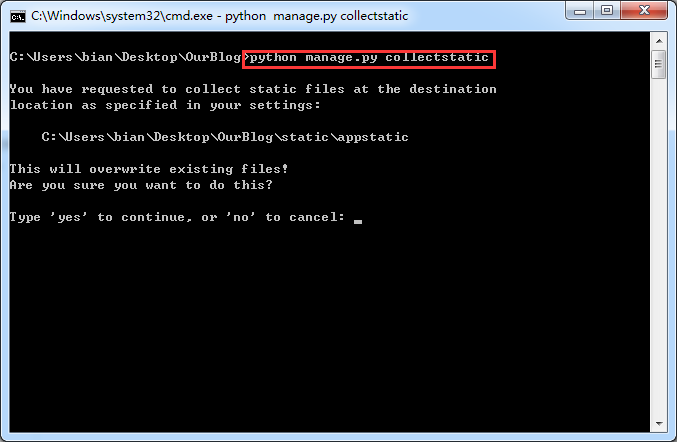
STATIC\_ROOT在web开发的过程当中需要使用标签加载使用，具体的使用方法如下

1. 配置STATIC\_ROOT的参数，通常指向static下的一个目录

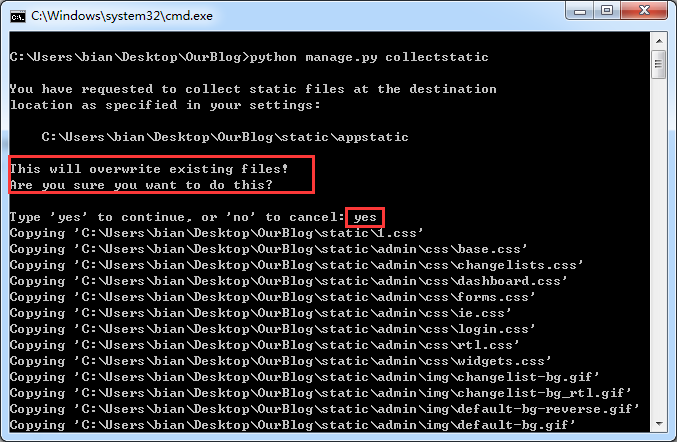


1. 在命令行收集静态文件

python manage.py collectstatic



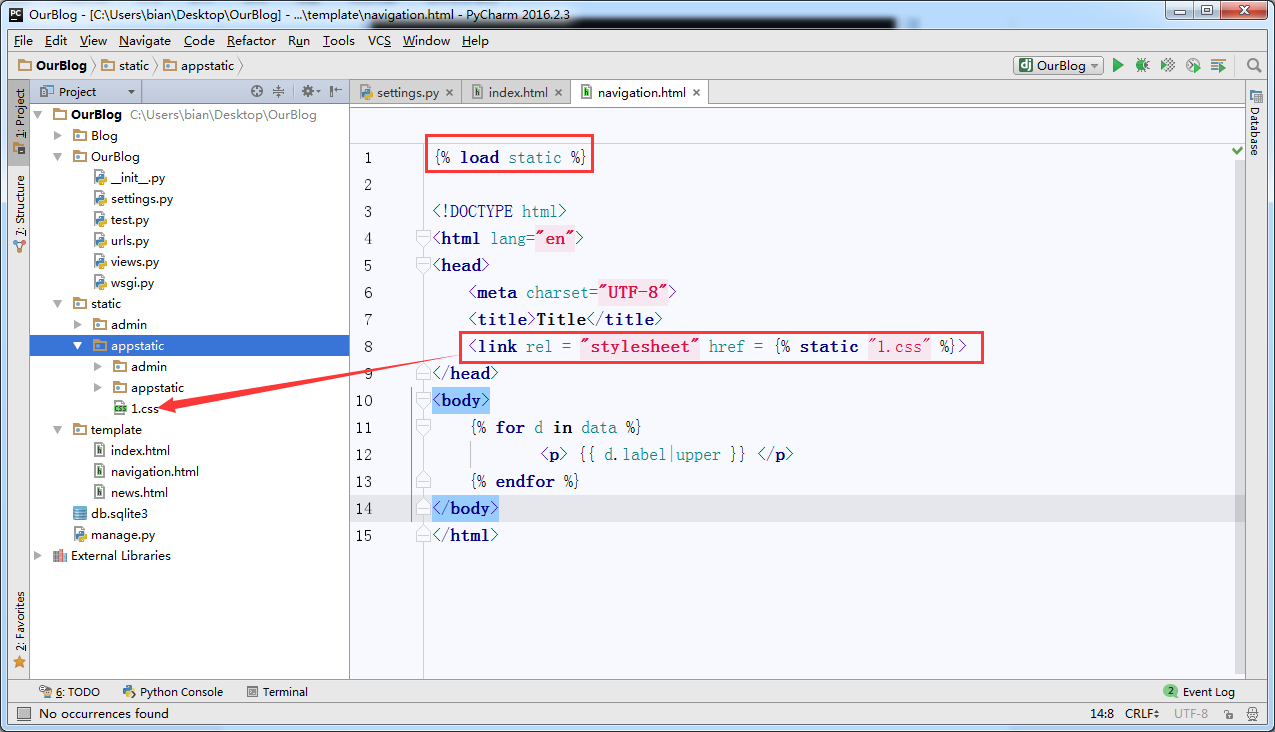
这里提示将要覆盖这个目录下面的文件，询问是否确定



1. 在前端部分加载static，然后使用

修改【D:xuegod\_code/3-10/OurBlog/template/navigation.html】

第9行插入：



比较两种方法，第一种配置静态的方法开发的效率比较高，第二种方法更利于松耦合的开发思路，选择哪种写法要根据自己的业务需求来定。

练习作业：

根据学习的模板系统知识修改个人博客的模板文件。（模板文件见blog目录）

总结：

9.1 Django模板静态文件加载

9.2 Django 模板加载的优化方法

9.3 Django 模板的包含和继承