Django+Vue.js构建项目

润润万睡
你去乾坤四海问一问，我是历代驰名第一妖！

118 人赞同了该文章

本文已发表于知乎专栏文章润润万睡：Django+Vue.js构建项目

本文主要讲述如何从0开始，用Django和Vue.js构建一个项目。文章提要：

Django与vue.js整合开发原理
从头新建一个Django项目
新建一个前端页面，使用vue应用
在Django中设计api
在vue应用中调用api获取数据，并展示到用户界面

几年前曾接触过Django Web开发框架，对其留下了深刻的印象。后来做数据可视化过程中，愈发觉得不久的将来，可视化领域必是JavaScript的天下。机缘巧合之下，接触到了vue.js，便为其设计之美所惊叹，能不能用Django和Vue.js做一个前后端分离的网站呢？

说干就干！

本文目标是利用Django和Vue.js做一个前后端分离的**最简可用**站点，并没有实现任何有实际意义的功能。

Django与Vue.js是如何结合起来？

推荐的Django+Vue.js整合开发，大家可见参考可视化Scrapy爬虫开发应用**Gerapy**的组织方式。（Gerapy的作者是

@崔庆才丨静觅，他也是静觅博客的作者，著有《Python3网络爬虫开发实战》，Gerapy项目地址：

Gerapy/Gerapy

github.com

)

Gerapy项目的结构：

```
.
├── client/  # 前端（客户端），vue项目。该文件夹不包含在上线项目中。
│   ├── build/
│   ├── server/
│   ├── src/  # 前端源码
│   │   ├── App.vue
│   │   ├── components/  # 组件
│   │   ├── favicon.ico
│   │   ├── index.html  # 入口html文件
│   │   ├── main.js
│   │   ├── pages/
│   ├── static/
│   ├── theme/
│   └── twistd.pid
├── cmd/
├── downloadermiddlewares/  # Gerapy内置的Scrapy下载器中间件
├── __init__.py
├── pipelines/  # Gerapy内置的Scrapy pipeline
├── server/  # Django项目
│   ├── core  # Django应用（app）
│   │   ├── admin.py
│   │   ├── apps.py
│   │   ├── build.py
│   │   ├── config.py
│   │   ├── encoder.py
│   │   ├── __init__.py
│   │   ├── middlewares.py
│   │   ├── migrations/
│   │   ├── models.py
│   │   ├── parser.py
│   │   ├── response.py
│   │   ├── scheduler.py
│   │   ├── templates  # 前端模板文件；从vue项目中构建来的前端文件
│   │   │   ├── favicon.ico
│   │   │   ├── index.html
│   │   │   └── static
│   │   │       ├── css
│   │   │       ├── fonts
│   │   │       ├── images
│   │   │       └── js
│   │   ├── tests.py
│   │   ├── time.py
│   │   ├── urls.py
│   │   ├── utils.py
│   │   └── views.py
│   ├── __init__.py
│   ├── manage.py
│   └── server
│       ├── __init__.py
│       ├── settings.py
│       ├── urls.py
│       └── wsgi.py
├── spiders/
├── templates/  # Gerapy内置的爬虫模板
└── VERSION
```

观察以上结构，最重要的两个部分，一个是client文件夹，一个是server文件夹。前者是vue单页面应用项目，它提供一个入口页面，页面中有一系列取数和数据组织逻辑。后者是一个Django项目，它管理数据库和api行为。

下面是server/core/urls.py中的部分代码：

```
# in server/core/urls.py
urlpatterns = [
    url(r'^$', views.index, name='index'),
    url(r'^api/index/status/$', views.index_status, name='index_status'),
    url(r'^api/client/$', views.client_index, name='client_index'),
    url(r'^api/client/create', views.client_create, name='client_create'),
    url(r'^api/client/(\d+)/$', views.client_info, name='client_info'),
    url(r'^api/client/(\d+)/status', views.client_status, name='client_status'),
    ... ...
]
```

可以看出来，它除了一个url指向返回入口页面的视图，其它url都指向api视图。

以下是server/core/views.py的部分代码：

```
# in server/core/views.py
def index(request):
    """
    render index page
    :param request: request object
    :return: page
    """
    return render(request, 'index.html')

... ...

def client_info(request, client_id):
    """
    get client info
    :param request: request object
    :param id: client id
    :return: json
    """
    if request.method == 'GET':
        return JsonResponse(model_to_dict(Client.objects.get(id=client_id)))
... ...
```

可以看到，index视图返回一个渲染的入口页面的响应；client_info视图返回一个JSONResponse响应：Client.objects.get(id=client_id))，从Client模型对应的表中取出特定的记录；model_to_dict(Client.objects.get(id=client_id))，将记录（模型实例）转换为字典dict；JsonResponse(model_to_dict(Client.objects.get(id=client_id)))，返回一个JSONResponse响应。JSONResponse响应决定了响应头的content-type是application/json。

Vue是View的法语说法，也就是说，Vue自打出生以来，就是为View而生。选择vue.js进行前端渲染，就放弃了Django内置的模型引擎。从功能模块上来说，Django将利用其强大的ORM功能，继续负责对Model的管理和操作；Django的view，将负责1）返回vue的入口文件，2）成为api-view，负责对数据的序列化与反序列化，返回JSON（等能被机器取数的）响应；Django的URLconf，将负责后端路由，指向vue入口文件和api。vue负责前端视图逻辑和前端模型，以及前端路由。

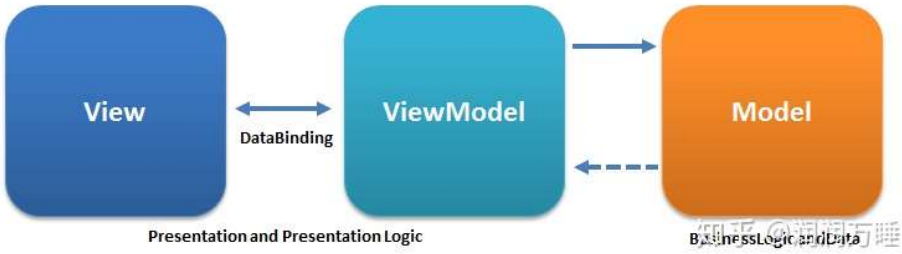
> Web开发真正困难的地方在于编写前端页面。前端页面需要混合HTML、CSS和JavaScript，如果对这三者没有深入地掌握，编写的前端页面将很快难以维护。
> 更大的问题在于，前端页面通常是动态页面，也就是说，前端页面往往是由后端代码生成的。生成前端页面最早的方式是拼接字符串；显然这种方式完全不具备可维护性，所以有第二种模板方式。
> 如果在页面上大量使用JavaScript（事实上大部分页面都会），模板方式仍然会导致JavaScript

> 代码与后端代码绑得非常紧密，以至于难以维护。其根本原因在于负责显示的HTML DOM模型与负责数据和交互的JavaScript代码没有分割清楚。
> 要编写可维护的前端代码绝非易事。和后端结合的MVC模式已经无法满足复杂页面逻辑的需要了，所以，新的MVVM：Model View ViewModel模式应运而生。
> MVVM最早由微软提出来，它借鉴了桌面应用程序的MVC思想。在前端页面中，把Model用纯JavaScript对象表示；View是纯HTML。
> 由于Model表示数据，View负责显示，两者做到了最大限度的分离。
> 把Model和View关联起来的就是ViewModel。ViewModel负责把Model的数据同步到View显示出来，还负责把View的修改同步回Model。
> ViewModel如何编写？需要用JavaScript编写一个通用的ViewModel，这样，就可以复用整个MVVM模型了。
> 双向绑定是MVVM框架最大的作用。借助于MVVM，我们把复杂的显示逻辑交给框架完成。由于后端编写了独立的REST API，所以，前端用AJAX（vue推荐的是axios）提交表单非常容易，前后端分离得非常彻底。

引用自

@廖雪峰 的文章

Day 11 - 编写日志创建页
www.liaoxuefeng.com



图来自维基百科

新建Django项目

如果你不知道如何新建一个Django项目，可以去Django官网查看一直文档，现在已经有官方中文文档了。

打开项目文件夹，比如我的是my_first_project

```
my_first_project$pip install django==2.1
my_first_project$django-admin startproject approot
```

可以看到项目文件夹下多了一个approot的文件夹，我把这个文件夹重命名为source，当然也可以不必重命名。

在source同级文件夹，新建一个database文件夹，用来存放数据库文件。再在source同级文件夹，新建一个static文件夹，用来存放静态文件（本博文内容中，并没有用到这个文件夹）。

一般教程中，会让你进入source文件夹中新建一个app，但是我们的目标是做一个最简可用的站点，所以我们甚至不需要一个app。

此时项目结构如下：

```
├── database
└── source
    ├── approot
    │   ├── __init__.py
```

```
    ├── settings.py
    ├── urls.py
    └── wsgi.py
└── manage.py
```

迁移数据库

```
source$python manage.py makemigrations
source$python manage.py migrate
```

运行Django开发服务器

```
source$python manage.py runserver
```

正常情况下，可以看到django的默认页。

添加前端内容

在source文件夹下，新建一个frontend文件夹，用来存放前端文件。

在frontend文件夹下新建一个new_file.html文件，文件中所有代码如下：

```
<!DOCTYPE html>
<html>
<head>
<meta charset="utf-8">
<title></title>
</head>
<body>
<div id="app">
改编不是乱编，{% verbatim %}{{ words_from_api }}{% endverbatim %}
</div>
<script src="https://cdn.jsdelivr.net/npm/vue@latest/dist/vue.js"></script>
<script src="https://cdn.jsdelivr.net/npm/axios@0.18.0/dist/axios.min.js"></script>
<script>
var app = new Vue({
    el: '#app',
    data () {
        return {
            string_from_api: '要向全国人民谢罪的！'
        }
    },
mounted () {
    axios
    .get('api/666')
    .then(response => (this.string_from_api = response.data))
    }
})
</script>
</body>
</html>
```

解释一下代码，

```
<div id="app">
{% verbatim %}{{ string_from_api }}{% endverbatim %}
</div>
```

这个div是用来展示数据的容器，vue应用将会挂载此处。

{{ string_from_api }}是前端应用从后端api获得的数据。

为什么要在{{ string_from_api }}前后加上{% verbatim %}和{% endverbatim %}？这是因为，Django本来也有自己有模板语言，它也要处理视图，但是我们此处用vue来管理我们的视图，它们之间会产生冲突。所以就要禁用掉Django的模板语言。关于解决Django模板语言和vue的冲突的方法还有很多，也以上网查询。

```
<script src="https://cdn.jsdelivr.net/npm/vue@latest/dist/vue.js"></script>
<script src="https://cdn.jsdelivr.net/npm/axios@0.18.0/dist/axios.min.js"></script>
```

引入两个js文件，vue.js是我们的前端框架（实际上是视图框架）；axios.js是尤雨溪推荐的用来前端与后端打交道的工具，它用来向后端api请求数据。

```
var app = new Vue
```

新建一个Vue的实例。

```
el: '#app',
```

与Html中的div容器绑定。

```
data () {
        return {
            string_from_api: '要向全国人民谢罪的！'
        }
    },
```

设定string_from_api的数据。这里也可以不设，但是由于加载各js并从后端拿到数据须要一定的时间，所以放个字符串在这里占位也不错。

```
mounted () {
    axios
        .get('api/666')
        .then(response => (this.string_from_api = response.data))
    }
})
```

挂载成功后执行的行为。这里是用axios向api/666发起get请求，再将获得的数据给data中的string_from_api。

在Django中设计一个api

在前文中，我们可以看到，axios向api/666发起get请求，所以我们要在Django项目中设计这个api返回的数据。

在/source/approot/urls.py中写入以下代码

```
from django.contrib import admin
from django.urls import path
from django.conf.urls import include, url
from django.views.generic.base import TemplateView
from django.http import HttpResponse
from django.shortcuts import render,render_to_response

urlpatterns = [
    path('admin/', admin.site.urls),
```

```
    url('api/666', view=lambda request: HttpResponse('戏说不是胡说'))
]
```

解释下上面的代码，Django用

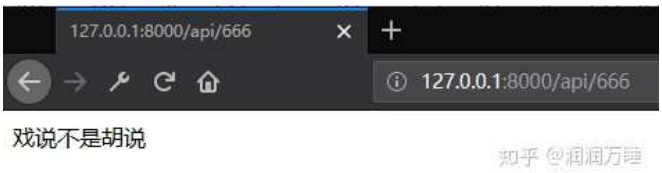
```
urlpatterns = [
    path('admin/', admin.site.urls),
    url('api/666', view=lambda e: HttpResponse('戏说不是胡说'))
]
```

来管理路由，访问站点时，Django会根据此处的urlpatterns决定将请求的url分发给谁来处理。一般的Django站点，都应该至少有一个应用，django将url分发给相应的应用处理，相应的应用返回视图。但是我们这个项目中，没有应用，所以我们只好就地返回一个视图。

```
url('api/.*', view=lambda request: HttpResponse('戏说不是胡说'))
```

可以看到，用户访问以api/666的url时，我们就返回lambda request: HttpResponse('戏说不是胡说')这个视图。这个视图直接返回一个字符串。

运行Django开发服务器，打开127.0.0.1:8000/api/666，可以看到如下结果：



api/666的返回内容

说明我们设计的API是有效的，能正确返回相应内容。

在Django中返回一个页面

前面我们有了一个前端页面了，还有了一个api了。现在的问题是如何访问到这个页面。很简单，在刚刚的urls.py中再添加一个路由就好了。

```
urlpatterns = [
    path('admin/', admin.site.urls),
    url(regex='^$', view=lambda request:
TemplateView.as_view(template_name='daily/new_file.html')),
    url('api/666', view=lambda e: HttpResponse('戏说不是胡说'))
```

可以看到，用户访问站点时，会返回给他一个new_file.html的页面，就是我们之前写过的那个含有vue应用的页面了。

但是到目前为止，这个应用还不能正常使用。

首先我们要解决跨域访问问题，我们用一个插件django-cors-headers处理这个问题。

```
$pip install django-cors-headers
```

在settings.py中进行如下设置：

```
MIDDLEWARE = [
    'django.middleware.security.SecurityMiddleware',
    'django.contrib.sessions.middleware.SessionMiddleware',
    'corsheaders.middleware.CorsMiddleware',  # 新加的插件
    'django.middleware.common.CommonMiddleware',
    'django.middleware.csrf.CsrfViewMiddleware',
```

```
    'django.contrib.auth.middleware.AuthenticationMiddleware',
    'django.contrib.messages.middleware.MessageMiddleware',
    'django.middleware.clickjacking.XFrameOptionsMiddleware',
]

INSTALLED_APPS = [
    'django.contrib.admin',
    'django.contrib.auth',
    'django.contrib.contenttypes',
    'django.contrib.sessions',
    'django.contrib.messages',
    'django.contrib.staticfiles',
    'corsheaders',  # 新加的应用
]

CORS_ORIGIN_ALLOW_ALL = True  # 新增的跨域访问设置
```

除了进行跨域访问设置外，还要进行静态文件路径设置。

```
TEMPLATES = [
    {
        'BACKEND': 'django.template.backends.django.DjangoTemplates',
        'DIRS': ['frontend'],  # 修改的行
        'APP_DIRS': True,
        'OPTIONS': {
            'context_processors': [
                'django.template.context_processors.debug',
                'django.template.context_processors.request',
                'django.contrib.auth.context_processors.auth',
                'django.contrib.messages.context_processors.messages',
            ],
        },
    },
]

STATIC_URL = '/frontend/'  # 修改的行
STATICFILES_DIRS = [
    os.path.join(BASE_DIR, 'frontend')  # 修改的行
,]
```

运行服务器查看结果

访问站点返回结果

最后的项目结构

```
.
├── database
│   └── db.sqlite3
└── source
    ├── approot
    │   ├── __init__.py
    │   ├── settings.py
    │   ├── urls.py
    │   └── wsgi.py
    ├── frontend
```

```
│   └── daily
│       └── new_file.html
└── manage.py
```

写在最后

无论是Django项目的组织还是vue应用的组织，都有更好的方式，本文仅仅展示了一个最简可用的“Django管理api，vue应用访问api”的前后端分离的web站点开发方式。

文章中介绍的项目结构，不适用于有前端路由的前端应用，所以推荐的方式还是前端后端分别部署。