2小时入门Django(DRF)+Vue前后端分离

本文链接：http://blog.cyru1s.com/posts/Django-vue.html

本文适用人群：Django 后端开发者，并希望入门前后端分离的工程化开发。

本文不适用人群：工程化前端开发者，本文的侧重点不在于前端的优雅开发而在于安全快速的构建前后端分离的工程化开发模式。

本文的前置技能点：对 Django 框架的工程结构较为熟悉，了解 Django REST Framework。了解 JS 和 CSS。本文标注【自行查找】的部分可以用搜索引擎方便找到，这里只提供无法查找到的解决方案。

本文不需要的前置技能点：不需要前端工程化开发经验，这个教程和 Django - template 开发差别不大，只是用了 Vue 作为载体。不需要使用 Nginx / Apache 等协助开发，我们的 Vue 和 Django 完全独立，所有静态文件都在 Vue，Django 只实现 API。

本文在 MacOS HighSierra 环境编写，在 Linux 系统上，如果你正常配置了 npm 环境，可以移植。

本文重点：

- 如何解决前后端非同源问题（0x01中）
- 如何在 Vue 引入第三方前端样式，如 iView（0x02中）
- 如何在 Vue template 中从后端获取 JSON 数据（0x03中）
- 如何在 Vue template 中向后端发送 JSON 数据（0x03中）
- 如何在后端使用 DRF 的 TokenAuthentication（0x04中）
- 如何在前后端进行 TokenAuthentication 认证（0x04中）

如果确认过眼神，确实是你想要的，不如现在就动手吧。

0x00 环境部署和目录结构

在Ubuntu工作环境配置不完全指南中我曾经提到，先装 nodejs 再装 npm，这个在 MacOS 不是必要的，但是建议这样做。npm 的被土嗇是打击多吐槽的，换 taobao 源是必要的。本文不注重于环境配置，我知道 npm 配置确实很费神，但是请自行查找。

多说一句，我们建议 npm 安装 Vue 相关组件的时候全部加上 -g 选项。

如何创建 Django 工程你应该是会的。Vue-init 也可以用来创建 Vue 工程。但是请遵循这样的目录结构。

```
+server root
|
|--+django root
|   |---setting.py
|   |---wsgi.py
|   |---// more //
|   +-------------
|
```

```
|--+django app
|  |
|  |--models.py
|  |--views.py
|  |--// more //
|  +------------
|
|  // run vue-init in server root //
|  // you will have vue root folder here //
|--+vue root
|  |
|  |--package.json
|  |--index.html
|  |--src folder
|  |--static folder
|  |--// more //
|  +------------
|
|---manage.py
|---// more //
+------------
```

（目录树真难画）

小结：

1. 在`manage.py`同级的地方运行`vue-init`
2. 装完记得`npm run build`（先有一个欢迎页面也行嘛）

0x01 Django 部分的配置

settings.py

我们先关注`settings.py`，我们需要修改许多部分：

1. 解决前后端非同源问题

   通常来说，前后端分离会架设在不同的端口上。这是非同源的（虽然我们在后面可以通过 build 到 Django 工程来避免此类问题）。我们用第三方插件解决这个问题。

   用`pip install django-cors-headers`来安装，并且在`MIDDLEWARE`部分添加如下（第三行）：

   ```
   MIDDLEWARE = [
       'django.middleware.security.SecurityMiddleware',
       'django.contrib.sessions.middleware.SessionMiddleware',
       'corsheaders.middleware.CorsMiddleware',
       'django.middleware.common.CommonMiddleware',
       # 'django.middleware.csrf.CsrfViewMiddleware',
       'django.contrib.auth.middleware.AuthenticationMiddleware',
   ```

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

CORS_ORIGIN_WHITELIST = (
     '127.0.0.1:8080', #Backend
     '127.0.0.1:8000', #Frontend on dev mode
)
```

之所以将整个部分贴出来，是因为 request 会从上至下通过 middleware 列表，然后经过 views 的处理，再从下至上通过 middleware 列表再返回。这其中 middleware 列表的返回也会影响执行顺序，详情参见 Ref 1。非同源访问模块应该尽早加载，这里被加载在 SessionMiddleware 之后，通用 Middleware 之前。虽然我们这个后端其实不用 Session 做用户认证（这一点后面会提到，可以先想一下为什么前后端分离的情况下不用？）。

2. AuthModel 部分

```
AUTH_USER_MODEL = 'info.UserInfo'
```

这是扩展 Model 使用的。

3. INSTALLED_APPS 部分

```
# 添加
INSTALLED_APPS = [
    'rest_framework',
]
```

这是添加 DRF 扩展。

4. TEMPLATES 部分

```
FRONTEND_ROOT = 'frontend/dist'

STATIC_URL = '/static/'

STATICFILES_DIRS = (
    os.path.join(BASE_DIR, FRONTEND_ROOT),
    os.path.join(BASE_DIR, FRONTEND_ROOT + '/static'),
)

TEMPLATES = [
    {
        'BACKEND': 'django.template.backends.django.DjangoTemplates',
        'DIRS': [FRONTEND_ROOT],
        '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',
            ],
        },
    },
]
```

前三个是添加的，然后在 TEMPLATES 中更改 DIRS。FRONTEND_ROOT 是 npm build 的目标目录

urls.py

此时，我们再关注一下`urls.py`部分。注意，这里是项目目录下的 urls，不是任何一个 APP 目录下的。

```
from django.views.generic import TemplateView
url(r'^$', TemplateView.as_view(template_name="index.html")),
```

这样我们就可以将根目录绑定到 build 好的 Vue 工程下了。此时访问 8000 端口，是不是不再是熟悉的 Django welcome，而是 Vue welcome了？

0x02 Vue 部分的配置

本文关注点并不在前端，这里简单说一下应该修改那些 Vue 工程中的文件。

- `src/router/indes.js`

  这里是路由配置。能在 Django 中只绑定一次 Template 的关键就在此。我们的页面跳转完全是由前端决定的，只是调用了后端的数据而已。路由是绑定前端页面和模板文件的关键。

  所以，一件很重要的事情是：**不要在后端写 Redirect，这回让前端路由不知所措。**

- `src/components`

  这里是 Vue 的一些自己写的组件。自行查找 VueJS 的文档吧。

- `src/main.js`

  引入的第三方插件和第三方前端样式都在这里。在此贴出文件内容以供参考。

```
// The Vue build version to load with the `import` command
// (runtime-only or standalone) has been set in webpack.base.conf with an
alias.
import Vue from 'vue'
import App from './App'
import router from './router'
import VueResource from 'vue-resource'
import iView from 'iview'
import 'iview/dist/styles/iview.css'
```

```
9
10 Vue.config.productionTip = false
11 Vue.use(VueResource)
12 Vue.use(iView)
13
14 /* eslint-disable no-new */
15 new Vue({
16   el: '#app',
17   router,
18   components: { App },
19   template: '<App/>'
20 })
21
```

我们引入了：

- vue-resource

  这个需要用 npm 进行安装。这是为了之后 http 请求更加方便。

  如果你想使用 Vue2.0 更加推荐的 vue-axios，可以参考 Ref 2

- iview

  是的，这个第三方前端样式也是需要进行 npm 安装的。使用如代码所示。

  经过这样设置，iView 可以直接使用`Col`和`Row`之类的标签，而不用使用`i-col`和`i-row`。

- `.eslintrc.js`

这个是根目录的一个文件，如果出现了一些奇奇怪怪的格式报错（空格个数不对，函数名和参数之间缺少空格，行末多了分号之类），请自行查找解决方案，在这里关闭报错就好了。否则你可以 run dev，但是不允许 run build。

这里在引入 iView 的时候，某些标签会被标记为不可关闭的。我是这样解决的：

```
1  module.exports = {
2    rules: {
3      // allow async-await
4      'generator-star-spacing': 'off',
5      // allow debugger during development
6      'no-debugger': process.env.NODE_ENV === 'production' ? 'error' :
   'off',
7      "vue/no-parsing-error": [2, { "x-invalid-end-tag": false }],
8      "indent": [0, 4]
9    }
10 }
```

`x-invalid-end-tag`是关闭不可关闭标签报错

`indent`是关闭空格数量不对报错（WebStorm 自动格式化的代码会报错 orz）

0x03 前后端数据交互

GET 请求后端数据

先看代码样例：

```
1  <template>
2      <div>
3          <p>
4              {{ recv_text }}
5          </p>
6      </div>
7  </template>
8  <script>
9      export default {
10         name: 'HelloWorld',
11         data () {
12             return {
13                 recv_url: 'http://localhost:8000/api/recv_api/',
14                 recv_text: ''
15             }
16         },
17         created () {
18             this.$http.get(this.recv_url).then(function (response) {
19                 this.recv_text = response.data
20             }, function (response) {
21                 this.recv_text = 'error'
22             })
23         }
24     }
25 </script>
```

和 Django template 一样，Vue 使用`{{value}}`双大括号进行数据绑定，同时提供全部 JS 表达式支持。

recv_url 要求后端返回一个数据。如果是 JSON 格式的，并不需要将接受数据的 recv_text 提前设置为 JSON 数据格式。你可以理解成和 Python 一样的弱类型。同时对于 JSON 格式的返回数据，可以直接在 Line 19 处使用形如 response.data.keyName 的表达方式。

关于 created() 是什么，请自行查找 VueJS 的文档吧。

POST 向后端发送数据

有了 GET 请求的实例，或许你觉得 POST 并没有什么问题，比如可以这样：

```
1  <template>
2      <div>
3          {{ page_title }}
```

```
        <Form ref="formLogin" method="post" :model="formLogin" :rules="ruleLogin">
            <FormItem prop="user">
                <Input type="text" v-model="formLogin.username" placeholder="Username" clearable>
                </Input>
            </FormItem>
            <FormItem prop="password">
                <Input type="password" v-model="formLogin.password" placeholder="Password" clearable>
                </Input>
            </FormItem>
            <FormItem>
                <Button type="primary" long @click="handleLogin('formLogin')">Sign in</Button>
            </FormItem>
        </Form>
    </div>
</template>

<script>
    export default {
        name: 'HelloWorld',
        data () {
            return {
                page_title: 'Login Please'
                login_url: 'http://localhost:8000/api/info/login/',
                formLogin: {
                    username: '',
                    password: ''
                },
                ruleLogin: {
                    username: [
                        {required: true, message: 'Please fill in the user name', trigger: 'blur'}
                    ],
                    password: [
                        {required: true, message: 'Please fill in the password.', trigger: 'blur'},
                        {type: 'string', min: 8, message: 'The password length cannot be less than 8 bits', trigger: 'blur'}
                    ]
                }
            }
        },
        methods: {
            handleLogin (formLogin) {
                this.$refs[formLogin].validate((valid) => {
                    if (valid) {
```

```
                        this.$http.post(this.login_url, this.formLogin)
                              .then(function (response) {
                                this.page_title = response.data.status
                            }, function (response) {
                                this.page_title = 'Login Error'
                            })
                    } else {
                        this.page_title = 'Wrong Input'
                    }
                })
            }
        }
    }
</script>
```

但是检查后端接收到的数据会发现，后端虽然给出了返回，实际上什么都没有收到！

检查前端页面的请求会发现，前端发送的数据其实是 application/json 格式的。我们 Django 后端通过 request 实际获取的是 application/x-www-form-urlencoded 的数据，无法直接处理 JSON 数据。（虽然 DRF 中是需要这样处理 JSON 数据的）

还记得`src/main.js`吗？我们一句话就可以解决这个问题：

```
Vue.http.options.emulateJSON = true
Vue.http.options.headers = {
  'Content-Type': 'application/x-www-form-urlencoded;charset=UTF-8',
}
```

这样我们就全局地在 http header 中增加了一个 Content-Type。检查后端，我们收到了请求。

0x04 在前端使用 DRF 的 Token 认证

我们在使用 POST 进行了登录和 GET 进行数据获取之后，会发现 GET 请求因为权限原因被拒绝了。检查后端接收到的 Session 我们会发现并没有发现任何数据。这是因为每次请求都是独立的，登录 POST 获得的 Session 在进行 GET 的时候早已经消失。

后端生成 Token 和认证 Token

我们需要一些准备工作：

首先`settings.py`的 INSTALLED_APPS 中增加 `'rest_framework.authtoken'`。并在之后增加：

```
REST_FRAMEWORK = {
    'DEFAULT_AUTHENTICATION_CLASSES': (
        'rest_framework.authentication.TokenAuthentication',
    )
}
```

为了在 admin 后台能看到全部 Token，我们可以添加在 AuthModel 对应的`admin.py`中增加以下部分：

```
from rest_framework.authtoken.admin import TokenAdmin
TokenAdmin.raw_id_fields = ('user',)
```

我们在用户登录的函数中，使用 Line 6 这样的语句：

```
# 提前导入
from rest_framework.authtoken.models import Token
# 增加语句
user = auth.authenticate(username=username, password=password) #验证用户名密码
if user is not None and user.is_active:
    # auth.login(request, user) # 这句是Session用的，只要给前端返回Token即可
    token, created = Token.objects.get_or_create(user=user)
    return JsonResponse({"status": "Authorized", "token": token.key})
else:
    return JsonResponse({"status": "Refused"})

```

如果没有 token 则新建，created 为 True，如果有则获取。这样可以防止对已有用户未分配 Token 的情况。实际工程中应该将这个新建 Token 的步骤放在注册。如果希望 Token 具有时效性，也可以在 AuthModel 中增加 Token 时间记录。

Token 在 HTTP_Header 中是`Token ffff…`的形式，注意这里的 token.key 没有 Token 前缀。

由于并不需要复杂逻辑，我直接自定义了一些装饰器。这里可以看出对于 Token 的使用方法。这是参考了 DRF 的源码的。

```
from . import conf
from django.contrib import auth
from django.contrib.auth.models import User
from django.http import HttpResponse, JsonResponse
from rest_framework.authtoken.models import Token

def need_login(func):
    def wrapper(request, *args, **kwargs):
        try:
            key = str(request.META['HTTP_AUTHORIZATION']).split()[1]
            request.user =
Token.objects.select_related('user').get(key=key).user
        except:
            return JsonResponse({"err": conf.NEED_LOGIN_HINT, "err_code":
conf.NEED_LOGIN_CODE})
        if request.user is not None:
            return func(request, *args, **kwargs)
        else:

```

```
17              return JsonResponse({"err": conf.NEED_LOGIN_HINT, "err_code":
    conf.NEED_LOGIN_CODE})
18
19      return wrapper
```

如果想检测后端是否正确启用了 Token 认证，可以用 Postman、http、curl 等确认。

前端通过 LocalStorage 保存 Token 和请求认证

这里相对后端会简单不少

在`main.js`中，增加 Line 3：

```
1  Vue.http.options.headers = {
2    'Content-Type': 'application/x-www-form-urlencoded;charset=UTF-8',
3    'Authorization': localStorage.getItem('user_token')
4  }
```

登录时：

```
1  var token = 'Token '
2  token = token + response.data.token
3  localStorage.setItem('user_token', token)
```

登出时：

```
1  localStorage.removeItem('user_token')
```

查看数据会自动带上 Token，后端如果需要验证可以自动获取 HTTP_Header 并验证。

0x05 TODO & Contact

大概这种前端 POST 请求方式会因为非同源而牺牲掉一些 CSRF 认证，好在 django-cors-headers 还不错。

大概如果考虑安全性，建议还是为 Token 设置一下时效性。这个算是 TODO 了。

很多文章中的一些步骤都有很多问题。如果你在应用本文时出现了问题，可以联系`fcs98#sina,com`。同时也希望更多中文母语者可以写出更多的中文相关资料。

0x06 Reference

- 理解django中的中间件机制和执行顺序
  https://blog.csdn.net/orangleliu/article/details/48316919
- Vue非最佳实践 - axios进行Ajax的小技巧

https://zhuanlan.zhihu.com/p/34715712

- Django REST framework api guide - Authentication
  http://www.django-rest-framework.org/api-guide/authentication/