## django应用ajax

csrf的使用：

在模板文件中添加：

|  |
| --- |
| <**script**> **$**.ajaxSetup({  **data**: {**csrfmiddlewaretoken**: **'**{{ **csrf\_token** }}**'** }, });  </**script**> |

或者在view.py的函数中：

|  |
| --- |
| @csrf\_exempt **def** post(request): |

## 文件上传

首先必须在settings中添加：

|  |
| --- |
| *# 在url中通过 http://127.0.0.1:8000/media/ 这样的路径访问上传文件* MEDIA\_URL = **'/media/'** *# 真实的文件保存在物理目录，需要自己在项目根目录下创建 media 目录* MEDIA\_ROOT = os.path.join(BASE\_DIR, **'media'**) |

其次，需要在项目的urls中添加，而不是子应用中：

|  |
| --- |
| **from** django.conf **import** settings **from** django.conf.urls.static **import** static  urlpatterns = [  ] + static(settings.MEDIA\_URL, document\_root=settings.MEDIA\_ROOT) |

这个就是使用静态文件管理器将上传目录管理起来，只要这样，浏览器才能正确访问到

### admin管理

在模型中添加文件或者图片属性，并且使用admin模块进行管理

模型中有2个不同的field：

* ImageField
* FileField

首先在模型中添加：

|  |
| --- |
| *# upload\_to 属性，会将上传文件保存到 MEDIA\_ROOT/upload\_to/文件名* img = models.ImageField(upload\_to=**'chatroom'**, verbose\_name=**'图标'**, null=**True**) |

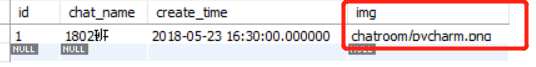
模型中还可以设置upload\_to 为一个函数:

|  |
| --- |
| **def** upload\_to(instance, filename):  *"""  上传图片路径* **:param** *instance: 模型对象实例* **:param** *filename: 文件名* **:return***: 保存的路径  """* **return** os.path.join(str(uuid.uuid4()), filename)  img = models.ImageField(upload\_to=upload\_to, verbose\_name=**'图标'**, null=**True**) |

在admin管理中会显示如下：



选择文件后保存，在数据库会保存为：



在模板中显示该图片，需要这样写：

|  |
| --- |
| *{# chat是模型的实例， img 是模型的ImageField属性名 img.url 是浏览器的url， img.name 是文件名 #}* <**img src="**{{ **chat**.**img**.**url** }}**"**>{{ **chat**.**img**.**name** }} |

### 自定义管理文件上传

#### form表单提交

模板页面中：

|  |
| --- |
| <**form action="**{% **url 'chatroom:fileupload\_submit'** %}**" method="post" enctype="multipart/form-data"**>  {% **csrf\_token** %}  <**input type="file" name="file\_upload"**>  <**input type="submit"**> </**form**> |

views后台：

|  |
| --- |
| @csrf\_exempt **def** fileupload\_submit(request):  obj = request.FILES.get(**'file\_upload'**)  *# 上传文件的文件名* print(obj.name)  **from** django.conf **import** settings  *# 事先在对应的 MEDIA\_ROOT 目录中创建 upload 目录* file\_path = os.path.join(**'upload'**, obj.name)   **"""  在这里可以进行数据的保存操作，将 file\_path.replace('\\', '/') 转换后，  保存到数据库中**  *# 保存数据库, img 对应的应该是 file 对象，而不是 file.name 这个字符串 # Chatroom.objects.create(chat\_name='上传聊天室', img=file, decription='上传时创建', create\_time=timezone.now())* **"""   with** open(os.path.join(settings.MEDIA\_ROOT, file\_path), **'wb'**) **as** f:  **for** chunk **in** obj.chunks():  f.write(chunk)   **return** HttpResponse(**'OK'**) |

#### ajax提交

模板中：

|  |
| --- |
| {% **load static** %} <**div**> <**input type="file" id="file\_upload"**> <**input type="button" value="上传" id="upload" onclick="***FileUpload*()**"**> </**div**> <**div id="msg"**></**div**> <**script src="**{% **static 'jquery-1.12.4.min.js'** %}**"**></**script**> <**script**>  *// 上传附件到方法* **function** *FileUpload*() {  **var** form\_data = **new *FormData***();  **var** file\_info = **$**(**'#file\_upload'**)[0].files[0];  form\_data.append(**'file\_upload'**, file\_info);   *// 提交ajax的请求* **$**.**ajax**({  **url**:**"**{% **url 'chatroom:fileupload\_submit'** %}**"**,  **type**:**'POST'**,  **data**: form\_data,  **processData**: **false**, *// tell jquery not to process the data* **contentType**: **false**, *// tell jquery not to set contentType* success: **function**(callback) {  **$**(**"#msg"**).html(**'上传成功'**);  **$**(**'#file\_upload'**).val(**""**);  }  }); *// end ajax* } </**script**> |

views中：

|  |
| --- |
| @csrf\_exempt **def** fileupload\_submit(request):  obj = request.FILES.get(**'file\_upload'**)  *# 上传文件的文件名* print(obj.name)  **from** django.conf **import** settings  *# 事先在对应的 MEDIA\_ROOT 目录中创建 upload 目录* file\_path = os.path.join(**'upload'**, obj.name)   **"""  在这里可以进行数据的保存操作，将 file\_path.replace('\\', '/') 转换后，  保存到数据库中  """   with** open(os.path.join(settings.MEDIA\_ROOT, file\_path), **'wb'**) **as** f:  **for** chunk **in** obj.chunks():  f.write(chunk)   **return** HttpResponse(**'OK'**) |

## 模板继承

Django模板引擎中最强大，也是最复杂的部分是模板继承。模板继承允许构建一个基本“骨架”模板，其中包含您网站的所有常见元素，并定义子模板可以覆盖的 blocks

如下，定义一个base.html， 如下：

|  |
| --- |
| <!DOCTYPE html>  <**html** lang="en">  <**head**>  <**link** rel="stylesheet" href="style.css" />  <**title**>{% **block** title %}My amazing site{% **endblock** %}</**title**>  </**head**>  <**body**>  <**div** id="sidebar">  {% **block** sidebar %}  <**ul**>  <**li**><**a** href="/">Home</**a**></**li**>  <**li**><**a** href="/blog/">Blog</**a**></**li**>  </**ul**>  {% **endblock** %}  </**div**>  <**div** id="content">  {% **block** content %}{% **endblock** %}  </**div**>  </**body**>  </**html**> |

上面这个模板中，定义了3个 block ：

* title
* sidebar
* content

block的使用方法：{% **block** title %} {% **endblock** %}

子页面detail.html继承上面这个base.html，内容如下：

|  |
| --- |
| {% **extends** "base.html" %}  {% **block** title %}My amazing blog{% **endblock** %}  {% **block** content %}  {% **for** entry **in** blog\_entries %}  <**h2**>{{ entry.title }}</**h2**>  <**p**>{{ entry.body }}</**p**>  {% **endfor** %}  {% **endblock** %} |

使用{% **extends** "base.html" %} 继承 base.html

本页面有2个block对应base.html的2个block，会在base.html中，使用本detail.html的对应内容进行替换，而block sidebar 并没有对应的block，因此继续使用base.html的 sidebar 的内容

继承的技巧：

* 如果在模板中使用，[**{% extends %}**](https://docs.djangoproject.com/en/2.0/ref/templates/builtins/#std:templatetag-extends)必须是该模板中的第一个模板标签。否则模板继承将不起作用。
* 基本模板中的越多[**{% block %}**](https://docs.djangoproject.com/en/2.0/ref/templates/builtins/#std:templatetag-block)标签越好。子模板不必定义所有父块，因此可以在多个块中填写合理的默认值
* 如果发现自己在多个模板中复制了内容，则可能意味着您应该将该内容移至父模板的 **{% block %} 中**
* 如果需要从父模板中获取块的内容，则使用**{{ block.super }}**。

|  |
| --- |
| {% **block sidebar** %}  {{ **block**.**super** }}  新的内容 {% **endblock** %} |

* 在使用模板标签语法之外创建的变量不能在块[**{% block% }**](https://docs.djangoproject.com/en/2.0/ref/templates/builtins/#std:templatetag-block)内使用。例如，该模板不会呈现任何内容
* {% **trans** "Title" **as** title %}
* {% **block** content %}{{ title }}{% **endblock** %}
* 为了增加可读性，您可以选择为您的标签*命名*。例如：**{% endblock** content **%}**

{% **block** content %}

{% **endblock** content %}

## session不记录用户登录状态

不记录登录状态就是关闭浏览器后，session失效，在views中，设置request.session.set\_expiry(0) 单位秒，0是关闭浏览器就失效

## 自定义模板过滤器

有些时候django自带的过滤器满足不了我们的需求，那么我需要编写自己的过滤器进行特殊处理

### 新建模块

在应用下新建templatetags源文件夹，名字不能修改

并且新建 my\_tags.py 文件

### 编写过滤器

|  |
| --- |
| **from** django **import** template *# 注册用哪个的对象* register = template.Library()   @register.filter **def** my\_range(value):  *"""  用于循环指定次数  5|my\_range* **:param** *value: 需要过滤的数据* **:return***: 可迭代 的 range 对象  """* **return** range(int(value))  @register.filter(name=**'get\_key'**) **def** get\_key(value, key):  *"""  用于value中获取指定的 key 的内容* **:param** *value: 需要被过滤的数据* **:param** *key: 过滤器后 ： 的参数* **:return***: 提取到的内容  """* **return** value.get(key) |

### 模板中应用

在模板的前面，导入自定义过滤器

|  |
| --- |
| {% **load my\_tags** %} |

在项目中需要的地方使用：

|  |
| --- |
| {% **for i in** 5|**my\_range** %}  {{ **i** }} {% **endfor** %} |

## admin管理自定义功能

如果我们要在admin管理页面中，增加自己的功能，那么我们需要做如下工作：

### admin允许覆盖的页面

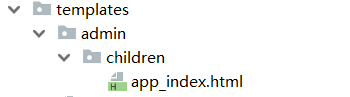
* app\_index.html
* change\_form.html
* change\_list.html
* delete\_confirmation.html
* object\_history.html
* popup\_response.html

### admin管理模板目录

contrib/admin/templates/admin目录，查看django自带的模板

### 建立目录

在settings中配置的templates目录下，建立admin目录，并且在admin目录下，建立mysite目录（mysite是你的应用名，假设mysite是children，那么目录结构如下图）



### 自定义模板

需要修改admin的哪个内置模板，则复制哪个模板，我们以app\_index.html为例：

原始模板是：

|  |
| --- |
| {% **extends "admin/index.html"** %} {% **load i18n** %}  {% **block bodyclass** %}{{ **block**.**super** }} app-{{ **app\_label** }}{% **endblock** %}  {% **if not is\_popup** %} {% **block breadcrumbs** %} <**div class="breadcrumbs"**> <**a href="**{% **url 'admin:index'** %}**"**>{% **trans 'Home'** %}</**a**> **&rsaquo;** {% **for app in app\_list** %} {{ **app**.**name** }} {% **endfor** %} </**div**> {% **endblock** %} {% **endif** %}  {% **block sidebar** %}{% **endblock** %} |

我们不需要修改所有的block，只需要在我们要添加功能的地方，进行继承，并且修改就行了，如下：

|  |
| --- |
| {% **extends "admin/app\_index.html"** %}  {% **block sidebar** %}  <**a href="**{% **url 'children:static\_html'** %}**"**>生成静态页面</**a**> {% **endblock** %} |

### 修改 urls.py

|  |
| --- |
| path(**'static\_html/'**, views.static\_html, name=**'static\_html'**), |

### 修改views.py

|  |
| --- |
| **def** static\_html(request):    *"""  编写你自己的自定义管理功能* **:param** *request:* **:return***:   """* **return** redirect(**'/admin/children/'**) |

## 模板页面静态化

通过在views中编写功能函数实现，一般配合上面的第6点，在admin管理中添加管理功能

功能函数示例如下：

|  |
| --- |
| **def** static\_html(request):  *"""  编写你自己的自定义管理功能* **:param** *request:* **:return***:  """* context = {  **'baby\_menu'**: get\_menu(),  }  html = render\_to\_string(**'base\_raw.html'**, context)  file\_path = os.path.join(settings.BASE\_DIR, **'children'**, **'templates'**, **'base.html'**)  **with** open(file\_path, **'w'**, encoding=**'utf-8'**) **as** f:  f.write(html)  **return** redirect(**'/admin/children/'**) |