

Django框架进阶



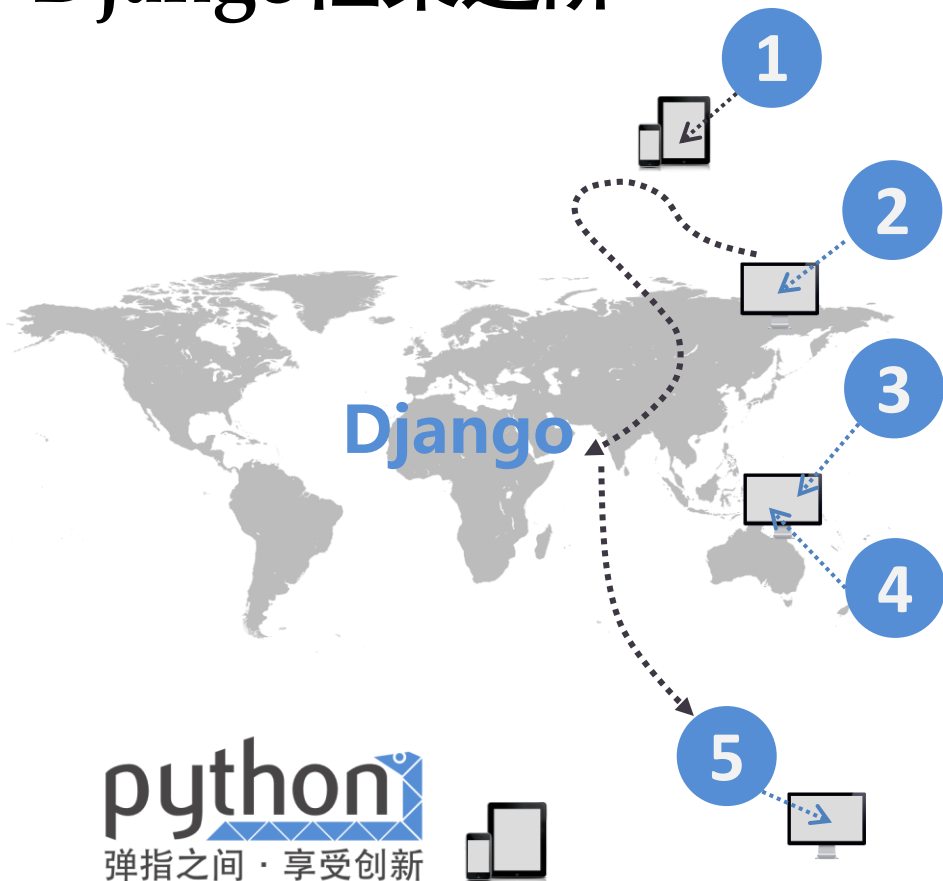
嵩 天
北京理工大学





单元开篇

Django框架进阶



Django的URL路由机制

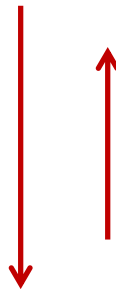
Django的视图响应类型

Django的视图流式响应机制

Django的模板引擎与模板使用

Django模板语言入门

URL



views



Django的URL路由机制

URL → 定义、转换、传参、命名等 → views

Django的URL路由配置

路由是关联URL及其处理函数关系的过程

settings.py文件中`ROOT_URLCONF`变量指定全局路由文件名称

```
ROOT_URLCONF = '<工程名称>.urls' # 默认对应工程目录下urls.py文件
```

Django使用`urlpatterns` 变量表示路由（urls.py），该变量是列表类型，由 `path()` 或 `re_path()` 作为元素组成

```
urlpatterns = [  
    path('msggate/', include('msgapp.urls')),  
    path('admin/', admin.site.urls),  
]
```

Django的URL路由流程

路由是关联URL及其处理函数关系的过程

- (1) Django查找全局 `urlpatterns` 变量 (`urls.py`)
- (2) Django按照先后顺序，对URL逐一匹配`urlpatterns`每个元素
- (3) 在找到第一个匹配时，停止查找，根据匹配结果执行对应处理函数
- (4) 如果没有找到匹配或出现异常，Django进行错误处理

Django的URL路由流程

路由是关联URL及其处理函数关系的过程

注意：

- Django的路由不考虑HTTP请求方式，仅根据URL进行路由，即，
- 只要URL相同，无论POST、GET等哪种请求方式都指向同一个操作函数

路由：path() 和 re_path()

path()处理字符串路由，re_path()处理正则表达式路由

URL字符串

path(*route*, *view*)

re_path(*route*, *view*)

对应的处理函数

正则表达式

正则表达式可以看作是字符串的模式

path(*route*, *view*, *kwargs=None*, *name=None*)

路由函数

- *route* : URL或URL模式, Django定义的URL转换语法
- *view* : 处理(视图)函数的名称, 或include()类
- *kwargs*: 向处理函数提供的额外参数, 以字典形式表示
- *name* : 给URL模式的命名

route的格式和转换

实例：

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

urlpatterns = [
    path('articles/2003/', views.special_case_2003),
    path('articles/<int:year>', views.year_archive),
    path('articles/<int:year>/<int:month>', views.month_archive),
    path('articles/<int:year>/<int:month>/<slug:slug>', views.article_detail),
]
```

*route*的格式和转换

Django支持三种方式表达*route*

- (1) 精确字符串格式：`articles/2003/`
- (2) Django的转换格式：`<类型:变量名>`，`articles/<int:year>/`
- (3) 正则表达式格式：`articles/(?P<year>[0-9]{4})/`

*route*的格式和转换

(1) 精确字符串格式

- 一个精确URL匹配一个操作函数
- 最简单形式，适合对静态URL的响应
- URL字符串不以 / 开头，但要以 / 结尾

`articles/2003/`

route的格式和转换

(2) Django的转换格式

- 一个URL模板，匹配URL同时在其中获得一批变量作为参数
- 常用形式，目的是通过URL进行参数获取和传递
- 采用 **<类型:变量名>** 格式获取参数

articles/2018/  articles/<int:year>/  匹配
请求URL Django的转换格式 year: 2018

route的格式和转换

(2) Django的转换格式

转换格式类型	说明
str	匹配除分隔符 (/) 外的非空字符串，默认类型 <year> 等价于 <str:year>
int	匹配0和正整数
slug	匹配字母、数字、横杠、下划线组成的字符串，str的子集
uuid	匹配格式化的uuid，如： 075194d3-6885-417e-a8a8-6c931e272f00
path	匹配任何非空字符串，包括路径分割符，是全集

route的格式和转换

(2) Django的转换格式

```
urlpatterns = [  
    path('articles/2003/', views.special_case_2003),  
    path('articles/<int:year>/', views.year_archive),  
    path('articles/<int:year>/<int:month>/', views.month_archive),  
    path('articles/<int:year>/<int:month>/<slug:msg>/', views.article_detail),  
]
```

articles/2018/01/learn-django-mooc

请求URL



第四行



提取带类型的参数

views.article_deail(request, year=2018, month=1, msg="learn-django-mooc")

*route*的格式和转换

(3) 正则表达式格式

- 借助正则表达式丰富语法表达一类URL（不是一个）
- 可以通过<>提取变量作为处理函数的参数，高级用法
- 使用re_path()函数

`articles/([0-9]{4})/`

*route*的格式和转换

(3) 正则表达式格式

- 使用re_path()函数，两种具体形式：
 - 简单形式 (pattern)，不提取参数：articles/([0-9]{4})/
 - 命名形式 (?P<name>pattern)，提取参数，统一为str类型：

articles/(?P<year>[0-9]{4})/

route的格式和转换

(3) 正则表达式格式

```
urlpatterns = [  
    re_path('articles/(?P<year>[0-9]{4})/', views.year_archive),  
    re_path('articles/(?P<year>[0-9]{4})/(?P<month>[0-9]{2})/', views.month_archive),  
    re_path('articles/(?P<year>[0-9]{4})/(?P<month>[0-9]{2})/(?P<msg>[\w-]+)/', \  
                                                    views.article_detail),  
]
```

articles/2018/01/learn-django-mooc

请求URL



第三行



提取参数，统一字符串类型

views.article_deail(request, year="2018", month="01", msg="learn-django-mooc")

path(*route*, *view*, *kwargs=None*, *name=None*)

路由函数

- *route* : URL或URL模式, Django定义的URL转换语法
- *view* : 处理(视图)函数的名称, 或include()类
- *kwargs*: 向处理函数提供的额外参数, 以字典形式表示
- *name* : 给URL模式的命名

view的使用

View包括两种类型：处理函数和include()函数

- 处理函数：views.py中处理URL的对应函数，URL处理的归宿


```
# appA/urls.py
urlpatterns = [
    path('help/', views.help),
    path('sub/', views.sub),
]
```

view的使用

View包括两种类型：处理函数和include()函数

- include()函数：包含其他路由信息的函数，分段路径组合形成总路径

```
# urls.py
urlpatterns = [
    path('entry/', include('appA.urls')),
]
```



```
# appA/urls.py
urlpatterns = [
    path('help/', views.help),
    path('sub/', views.sub),
]
```

http://127.0.0.1:8000/entry/help/ --> views.help()

http://127.0.0.1:8000/entry/sub/ --> views.sub()

view的使用

include()用法：1) 附加本地路由；2) 路径去重

```
urlpatterns = [  
    path('<page_slug>-<page_id>/history/', views.history),  
    path('<page_slug>-<page_id>/edit/', views.edit),  
    path('<page_slug>-<page_id>/discuss/', views.discuss),  
    path('<page_slug>-<page_id>/permissions/', views.permissions),  
]
```

路径去重



```
urlpatterns = [  
    path('<page_slug>-<page_id>/', include([  
        path('history/', views.history),  
        path('edit/', views.edit),  
        path('discuss/', views.discuss),  
        path('permissions/', views.permissions),  
    ])),  
]
```

URL根目录的处理

http://127.0.0.1:8000/ 的响应

```
# 全局urls.py文件
urlpatterns = [
    path('', <根目录处理函数>),
    re_path('^$', <根目录处理函数>), # 或者，二选一
]
```




Django的视图响应类型

视图函数编写原则

视图函数接受HTTP请求并返回响应，可以放在任何地方，可以是任何功能

- 视图函数可以返回Web文本、页面、重定向、错误、图片等任何内容
- 视图函数通过HttpResponse、JsonResponse等类表达并返回响应
- 按约定，视图函数放在对应app中的views.py文件中

Django的响应类型

`django.http`包含所有响应类型

- `HttpResponse`类及子类 （共10个）
- `JsonResponse`类
- `StreamingHttpResponse`类
- `FileResponse`类

Django的响应类型

(1) HttpResponse类及子类

类型	说明
<code>HttpResponse</code>	主要反馈类型，父类，HTTP状态码默认为200
<code>HttpResponseRedirect</code>	重定向，HTTP状态码为302
<code>HttpResponsePermanentRedirect</code>	永久重定向，HTTP状态码为301
<code>HttpResponseNotModified</code>	网页无改动，该类型无任何参数，HTTP状态码为304
<code>HttpResponseBadRequest</code>	不良响应，HTTP状态码为400

Django的响应类型

(1) HttpResponse类及子类

类型	说明
HttpResponseForbidden	禁止访问，HTTP状态码为403
HttpResponseNotAllowed	不被允许，HTTP状态码为405
HttpResponseGone	HTTP状态码为410
HttpResponseServerError	服务器错误，HTTP状态码为500
HttpResponseNotFound	404错误，HTTP状态码为404

Django的响应类型

(1) HttpResponse类及子类

`HttpResponse(content, content_type=None, status=200, charset=None)`

- *content* : 拟返回的字符串
- *content_type* : MIME格式的返回内容类型
- *status* : 响应状态码
- *charset* : 响应的字符集

Django的响应类型

(1) HttpResponseRedirect类及子类

```
from django.contrib import admin
from django.urls import include, path
from msgapp import views as msgviews

urlpatterns = [
    path('msggate/', include('msgapp.urls')),
    path('admin/', admin.site.urls),
    path('', msgviews.homeproc),
]
```

与老师一起动动手吧

修改cloudms的根页面

```
def homeproc(request):
    return HttpResponseRedirect("<h1>这是首页，具体功能请访问<a href='./msggate'>这里</a></h1>")
```

Django的响应类型

(1) HttpResponseRedirect类及子类

```
def homeproc(request):  
    response = HttpResponseRedirect()  
    response.write("<h1>这是首页，具体功能请访问<a href='./msggate'>这里</a></h1>")  
    response.write("<h1>这是第二行</h1>")  
    return response
```

与老师一起动动手吧

Django的响应类型

(2) JsonResponse类

`JsonResponse(data)`

- *data* : 字典类型，返回的JSON类型数据

```
def homeproc1(request):  
    response = JsonResponse({'key1': 'value1'})  
    return response
```

与老师一起动动手吧

Django的响应类型

(3) StreamingHttpResponse类

`StreamingHttpResponse(streaming_content)`

- *streaming_content* : 内容的迭代器形式，以内容流的方式响应

流式响应

Django的响应类型

(4) FileResponse类

`FileResponse(stream)`

- *stream* : 以流形式打开后的文件

```
def homeproc2(request):  
    cwd = os.path.dirname(os.path.dirname(os.path.abspath(__file__)))  
    response = FileResponse(open(cwd + "/msgapp/templates/PyLogo.png", "rb"))  
    response['Content-Type'] = 'application/octet-stream'  
    response['Content-Disposition'] = 'attachment;filename="pylogo.png"'  
    return response
```

与老师一起动手吧

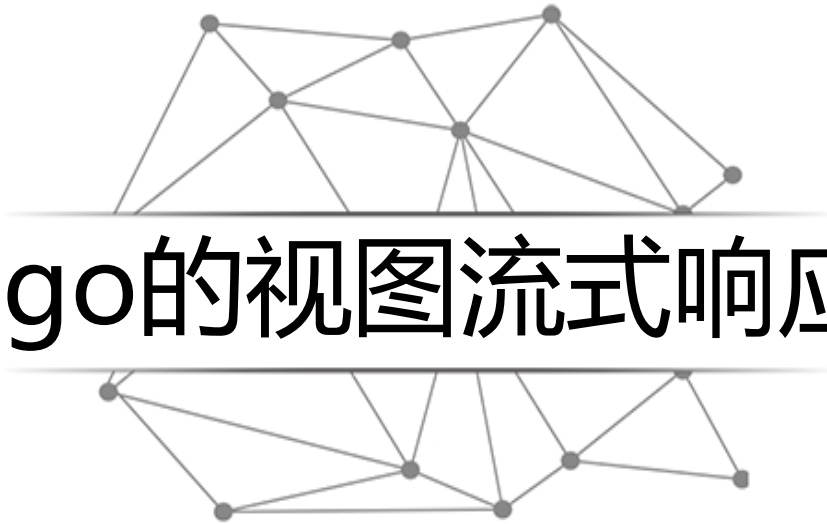
Django请求类型的判断

`django.views.decorators.http`

- 路由不能判断HTTP请求的类型
- Django通过decorators在视图函数前进行基本判断，格式如下：

```
from django.views.decorators.http import require_http_methods

@require_http_methods(["GET", "POST"])
def my_view(request):
    # I can assume now that only GET or POST requests make it this far
    # ...
    pass
```



Django的视图流式响应机制

Django的响应类型

一次性响应 vs 流式响应

- HttpResponseRedirect类及子类
- JsonResponse类
- StreamingHttpResponse类
- FileResponse类

一次性响应

流式响应

大文本文件传输

大二进制文件传输

实例：文件下载

(1) HttpResponse方式

```
def file_download(request):  
    # do something...  
    with open('data.txt') as f:  
        c = f.read()  
    return HttpResponse(c)
```

文件内容一次性响应，仅适合小文件

实例：文件下载

(2) StreamingHttpResponse方式

- 将文件分段，每次传输一部分，分段大小可调
- 利用Python的迭代器产生分段
- 可以是文件，也可以是任何大规模数据响应

实例：文件下载


(2) StreamingHttpResponse方式

```
from django.http import StreamingHttpResponse

def big_file_download(request):
    # do something...
    def file_iterator(file_name, chunk_size=512):
        with open(file_name) as f:
            while True:
                c = f.read(chunk_size)
                if c:
                    yield c
                else:
                    break

    fname = "data.txt"
    response = StreamingHttpResponse(file_iterator(fname))
    return response
```

yield关键字

yield  生成器

包含yield语句的函数是一个生成器（迭代器的一种）

生成器每次产生一个值（yield语句），函数被冻结，被唤醒后再产生一个值

生成器是一个不断产生值的函数

实例：文件下载

(3) FileResponse方式

- FileResponse是StreamingHttpResponse的子类
- 自动分段、自动迭代，适合二进制文件传输

```
def homeproc2(request):  
    cwd = os.path.dirname(os.path.dirname(os.path.abspath(__file__)))  
    response = FileResponse(open(cwd + "/msgapp/templates/PyLogo.png", "rb"))  
    response['Content-Type'] = 'application/octet-stream'  
    response['Content-Disposition'] = 'attachment;filename="pylogo.png"'  
    return response
```

实例：文件下载

MIME标记

```
response['Content-Type'] = 'application/octet-stream'  
response['Content-Disposition'] = 'attachment;filename="pylogo.png"'
```

- Content-Type用于指定文件类型
- Content-Disposition用于指定下载文件的默认名称
- 这两者是MIME类型的标准定义

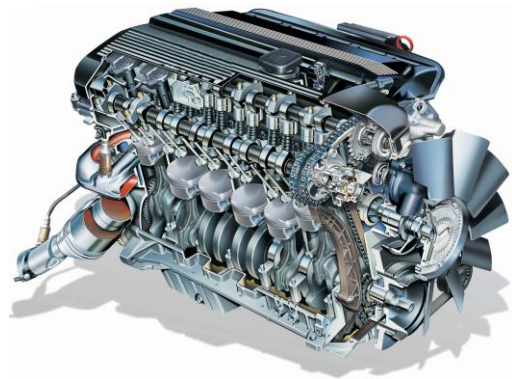


Django的模板引擎与模板使用

Django的模板引擎

模板引擎是模板响应的后端

响应



模板引擎: **django** , Jinja2

模板引擎的配置

settings.py文件中对模板引擎进行配置 (BACKEND)

```
TEMPLATES = [  
    {  
        'BACKEND': 'django.template.backends.django.DjangoTemplates',  
        'DIRS': [os.path.join(BASE_DIR, "msgapp/templates")],  
        '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',  
            ],  
        },  
    },  
]
```

引擎配置

模板引擎的配置

模板目录在列表中的顺序是搜索模板的顺序

```
TEMPLATES = [  
    {  
        'BACKEND': 'django.template.backends.django.DjangoTemplates',  
        'DIRS': [os.path.join(BASE_DIR, "msgapp/templates")],  
        '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',  
            ],  
        },  
    },  
]
```

模板目录

模板的查找

`get_template(template_name)` -> Template类

在模板目录列表中，依次查找某个模板，直到找到并返回Template类

如果未找到模板，则抛出TemplateDoesNotExist异常

- *template_name* : 待查找模板的名称

模板的查找

`select_template(template_name_list)` -> Template类

与`get_template()`相似，依次按照模板列表查找某个模板，直到找到第一个并返回Template类，未找到则抛出`TemplateDoesNotExist`异常

- *template_name_list* : 待查找模板名称列表

模板的渲染

`Template.render(context, request)` -> HTML字符串

模板对象的`.render()`方法用于将模板结合内容渲染成HTML字符串

- *context* : 字典类型，用于加载到模板中的内容
- *request* : HTTP请求

渲染render()

将模板和内容整合到一起，返回HTML字符串

```
try:
    tpl = get_template("MsgSingleWeb.html")
except:
    return HttpResponseNotFound("<h1>自定义的404</h1>")
html = tpl.render({"data":datalist}, request)
return HttpResponse(html)
```



模板使用步骤

步骤1：指定Template：定义一个Template类

步骤2：进行渲染：通过Context类或字典类

```
def pgproc(request):  
    template = Template("<h1>这个程序的名字是 {{ name }}</h1>")  
    context = Context({"name" : "实验平台"})  
    return HttpResponse(template.render(context))
```

与老师一起设立一个playground/路径



Django模板语言入门

模板语言



模板语言：指导模板加载数据方式的工具

DTL: Django Template Language

<https://docs.djangoproject.com/en/2.0/ref/templates/language/>

模板语言

- 注释 `comment`
- 变量 `variable`
- 标签 `tags`
- 过滤器 `filter`

模板语言

- 注释

单行注释：`{# 这是单行注释 #}`

多行注释：`{% comment %}`

这是多行注释第一行

这是多行注释第二行

`{% endcomment %}`

模板语言

- 变量

```
{{ name }}
```

如果变量本身是字典类型、列表类型或对象，用`.`获取元素

```
{{ adict.key }}    {{ alist.0 }}
```

```
{{ aobject.attribute }}
```

模板语言

- 标签

`{% 关键字引导的程序逻辑 %}`

标签中的关键字包括

`for`, `endfor`, `block`, `endblock`, `if`, `elif`, `else`,
`endif`, `in`, `trans`, `as`, `with`, `extends` 等

<https://docs.djangoproject.com/en/2.0/ref/templates/language/>

模板语言

- 标签

```
{% for line in data %}  
<tr>  
  <td>{{ line.time }}</td>  
  <td align="center">{{ line.userA }}</td>  
  <td>{{ line.msg }}</td>  
</tr>  
{% endfor %}
```

<https://docs.djangoproject.com/en/2.0/ref/templates/language/>

模板语言

- 过滤器

`{{ name|过滤标签 }}` `{{ name|f1|f1 }}`

过滤器对变量的值进行修饰

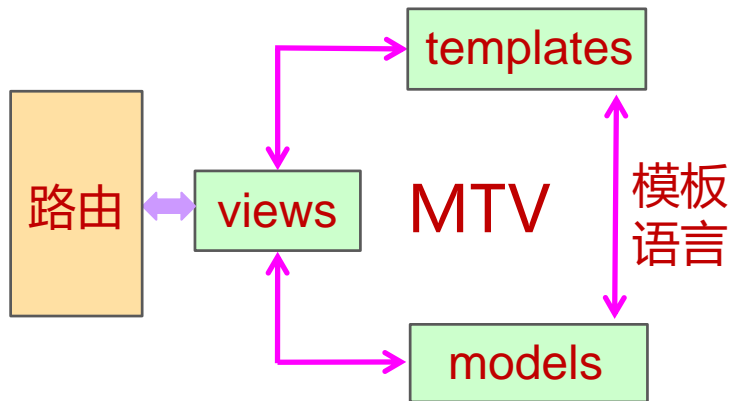
`lower`, `escape`, `linebreaks`, `date`, `length`等

```
{{ name|lower }}  
{{ my_date|date:"Y-m-d" }}
```



单元小结

Django框架进阶



Django的URL路由机制

Django的视图响应类型

Django的视图流式响应机制

Django的模板引擎与模板使用

Django模板语言入门