

《Django 教程》

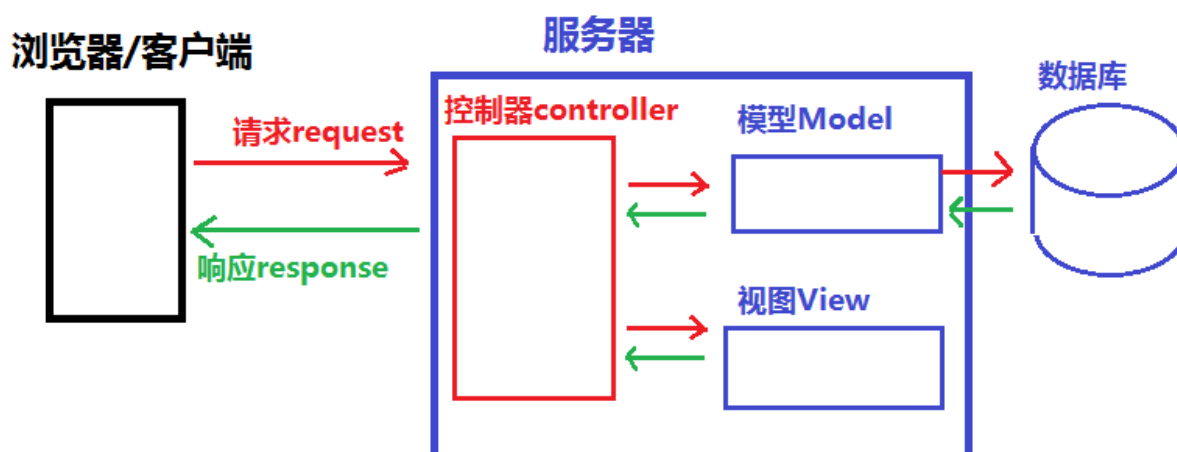
- 讲师: 魏明择
- 时间: 2019

目录

- Django的框架模式
- 模板 Templates
 - Django 模板语言: (The Django template language)
 - 模板的传参
 - 模板的变量
 - 模板的标签
 - 过滤器
 - 模板的继承
 - url 反向解析
- 静态文件
- Django中的应用 - app
 - 什么是应用(app)
 - 创建应用app
 - Django应用的结构组成

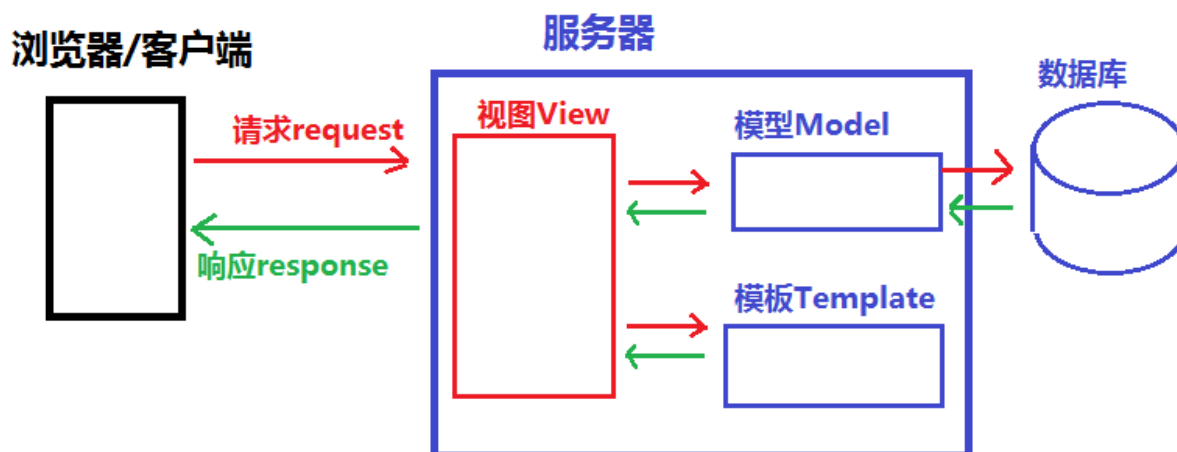
Django的框架模式

- MVC 设计模式
 - MVC 代表 Model-View-Controller (模型-视图-控制器) 模式。
 - 作用: 降低模块间的耦合度(解耦)
 - MVC
 - M 模型层(Model), 主要用于对数据库层的封装
 - V 视图层(View), 用于向用户展示结果
 - C 控制(Controller, 用于处理请求、获取数据、返回结果(重要))
 - 如图:



- MTV 模式 MTV 代表 Model-Template-View (模型-模板-视图) 模式。这种模式用于应用程序的分层开发
 - 作用:

- 降低模块间的耦合度(解耦)
- MTV
 - M -- 模型层(Model) 负责与数据库交互
 - T -- 模板层(Template) 负责呈现内容到浏览器
 - V -- 视图层(View) 是核心, 负责接收请求、获取数据、返回结果
- 如图:



模板 Templates

- 什么是模板
 1. 模板是html页面, 可以根据视图中传递的数据填充值相应的页面元素。
 2. 模板层提供了一个对设计者友好的语法用于渲染向用户呈现的信息。
- 模板的配置
 - 创建模板文件夹<项目名>/templates
 - 在 settings.py 中有一个 TEMPLATES 变量
 1. BACKEND : 指定模板的引擎
 2. DIRS : 指定保存模板的目录们
 3. APP_DIRS : 是否要在应用中搜索模板本
 4. OPTIONS : 有关模板的选项们
- 默认模块文件夹templates
- 修改settings.py文件, 设置TEMPLATES的DIRS值为 'DIRS': [os.path.join(BASE_DIR, 'templates')],

```

# file: settings.py
TEMPLATES = [
    {
        'BACKEND': 'django.template.backends.django.DjangoTemplates',
        # 'DIRS': [],
        'DIRS': [os.path.join(BASE_DIR, 'templates')], # 添加模板路径
        'APP_DIRS': True, # 是否索引各app里的templates目录
        ...
    },
]
  
```

3. 模板的加载方式

1. 通过 loader 获取模板,通过HttpResponse进行响应

```
from django.template import loader
# 1.通过loader加载模板
t = loader.get_template("模板名称")
# 2.将t转换成字符串
html = t.render()
# 3.响应
return HttpResponse(html)
```

■ 示例:

```
from django.template import loader

def render_page1(request):
    # 1.通过loader加载模板
    t = loader.get_template("page1.html")
    # 2.将t转换成字符串
    html = t.render()
    # 3.响应
    return HttpResponse(html)
```

2. 使用 render() 直接加载并响应模板

■ 示例:

```
from django.shortcuts import render
return render(request, '模板的名称')
```

■ 示例

```
def render_page2(request):
    from django.shortcuts import render
    return render(request, 'page1.html')
```

Django 模板语言: (The Django template language)

模板的传参

- 模板传参是指把数据形成字典, 传参给模板, 由模板渲染来填充数据

1. 使用 loader 加载模板

```
t = loader.get_template('xxx.html')
html = t.render(字典数据)
return HttpResponse(html)
```

2. 使用render加载模板

```
return render(request, 'xx.html', 字典数据)
```

模板的变量

1. 在模板中使用变量语法

- `{{ 变量名 }}`

1. 后端中必须将变量封装到字典中才允许传递到模板上

```
dic = {
    "变量1": "值1",
    "变量2": "值2",
}
```

模板的标签

1. 作用

- 将一些服务器端的功能嵌入到模板中

2. 标签语法

```
{% 标签 %}
...
{% 结束标签 %}
```

3. if 标签

```
{% if 条件表达式 %}
...
{% elif 条件表达式 %}
...
{% else %}
...
{% endif %}
```

4. if 标签里的布尔运算符

- if 条件表达式里可以用的运算符 ==, !=, <, >, <=, >=, in, not in, is, is not, not、and、or
- 在if标记中使用实际括号是无效的语法。如果您需要它们指示优先级，则应使用嵌套的if标记。

5. for 标签

1. 语法

```
{% for 变量 in 可迭代对象 %}
{% endfor %}
```

2. 内置变量 - forloop

变量	描述
forloop.counter	循环的当前迭代（从1开始索引）
forloop.counter0	循环的当前迭代（从0开始索引）
forloop.revcounter	循环结束的迭代次数（从1开始索引）
forloop.revcounter0	循环结束的迭代次数（从0开始索引）
forloop.first	如果这是第一次通过循环，则为真
forloop.last	如果这是最后一次循环，则为真
forloop.parentloop	对于嵌套循环，这是围绕当前循环的循环

6. for ... empty 标签

1. 语法

```
{% for 变量 in 可迭代对象 %}
{% empty %}
{% endfor %}
```

7. cycle 标签

- 循环从cycle 列表后的参数中进行取值,每次调用进行一次更换
- 这个标签经常用于循环中，如处理表格的隔行变色
- 语法:

```
{% for o in some_list %}
    <tr class="{% cycle 'row1' 'row2' %}">
        ...
    </tr>
{% endfor %}
```

- 示例:

```

<table>
  <tr>
    <th>姓名</th>
    <th>年龄</th>
    <th>成绩</th>
  </tr>
  <tr style="background: red;">
    <td>小张</td>
    <td>20</td>
    <td>100</td>
  </tr>
  <tr style="background: blue;">
    <td>小张</td>
    <td>20</td>
    <td>100</td>
  </tr>
  <tr style="background: red;">
    <td>小张</td>
    <td>20</td>
    <td>100</td>
  </tr>
  <tr style="background: blue;">
    <td>小张</td>
    <td>20</td>
    <td>100</td>
  </tr>
<!-- 以下可以用cycle 标签完成 -->
{% for s in students %}
  <tr style="background: {% cycle 'red' 'yellow' %};">
    <td>{{ s.name }}</td>
    <td>{{ s.age }}</td>
    <td>{{ s.score }}</td>
  </tr>
{% endfor %}
</table>

```

8. 注释 和 comment标签

- 以`{#` 开头, 以`#}` 结束范围内的文字信息将会被模板的渲染系统忽略掉
- 如:
 - `{# <h1>此处的文字不会被生成html文档</h1> #}`

4. comment 标签

- 在`{% comment %}` 和`{% endcomment %}`, 之间的内容会被忽略,
- 作用: 用于注释, 可以用此来记录代码被注释掉的原因。
- 注: comment 标签不能嵌套使用
- 例如:

```
<div>Rendered text with {{ pub_date|date:"c" }}</div>
{% comment "Optional note" %}
    <div>Commented out text with {{ create_date|date:"c" }}</div>
{% endcomment %}
comment标签不能嵌套使用。
```

过滤器

- 1. 作用
 - 在变量输出前对变量的值进行处理
 - 您可以通过使用 过滤器来改变变量的显示。
- 2. 语法
 - {{变量|过滤器1:参数值1|过滤器2:参数值2 ...}}
- 3. 有用的过滤器

过滤器	说明
default	如果value的计算结果为False，则使用给定的默认值。 否则，使用该value。
default_if_none	如果（且仅当） value为None，则使用给定的默认值。 否则，使用该value。
floatformat	当不使用参数时，将浮点数舍入到小数点后一位，但前提是要显示小数部分。
truncatechars	如果字符串字符多于指定的字符数量，那么会被截断。 截断的字符串将以可翻译的省略号序列（“...”）结尾。
truncatewords	在一定数量的字后截断字符串。
lower	将字符串转换为全部小写。
upper	将字符串转换为大写形式
...	

- 4. escape 示例:
 - escape 转义字符串的HTML。 具体来说， 它使这些替换：

```
< 转换为 &lt;
> 转换为 &gt;
' (单引号) 转换为 &#39;
" (双引号) 转换为 &quot;
& 转换为 &amp;
```

- 5. 文档参见:
 - https://iyijibooks.cn/xx/Django_1.11.6/ref/templates/builtins.html

- <https://docs.djangoproject.com/zh-hans/2.2/ref/templates/builtins/>

模板的继承

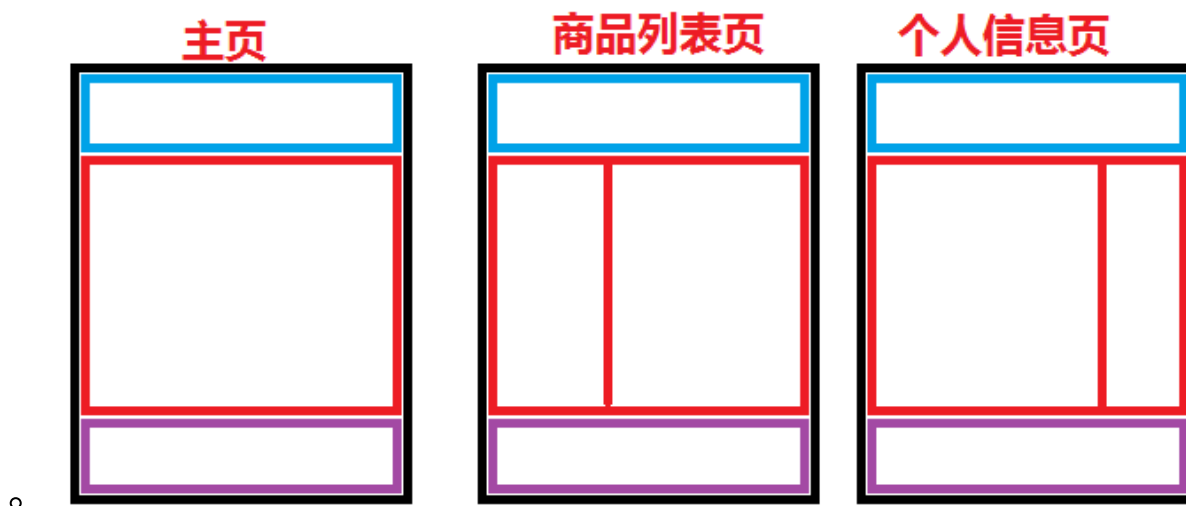
- 模板继承可以使父模板的内容重用,子模板直接继承父模板的全部内容并可以覆盖父模板中相应的块
- 定义父模板中的块 **block** 标签
 - 标识出哪些在子模块中是允许被修改的
 - block 标签: 在父模板中定义, 可以在子模板中覆盖

```
{% block block_name %}  
定义模板块, 此模板块可以被子模板重新定义的同名块覆盖  
{% endblock block_name %}
```

- 继承模板 **extends** 标签(写在模板文件的第一行)
 - 子模板继承语法标签
 - `{% extends '父模板名称' %}`
 - 如:
 - `{% extends 'base.html' %}`
 - 子模板 重写父模板中的内容块

```
{% block block_name %}  
子模板块用来覆盖父模板中 block_name 块的内容  
{% endblock block_name %}
```

- 重写的覆盖规则
 - 不重写,将按照父模板的效果显示
 - 重写,则按照重写效果显示
 - 注意
 - 模板继承时,服务器端的动态内容无法继承
- 参考文档
 - <https://docs.djangoproject.com/en/1.11/ref/templates/>
- 模板的继承示例:



```
# file : url.py
from django.conf.urls import url
from . import views
urlpatterns = [
    ...
    url(r'^index', views.index),
    url(r'^goods_list1', views.goods_list),
    url(r'^goods_list2', views.goods_list2),
    url(r'^user_info1', views.user_info1),
    url(r'^user_info2', views.user_info2),
]
# file: views.py
def index(request):
    return render(request, 'index.html')

def goods_list(request):
    return render(request, 'goods_list.html')

def goods_list2(request):
    return render(request, 'goods_list2.html')

def user_info1(request):
    return render(request, 'user_info_page1.html')

def user_info2(request):
    return render(request, 'user_info_page2.html')
```

o 基类模板

```
<!-- file : base.html -->
<!DOCTYPE html>
<html>
<head>
    <meta charset="utf-8">
    <title>{% block title %}我的达内{% endblock %}</title>
    <style>
```

```

/* 基础样式设置 */
body, h1, h2, h3, h4, h5, h6, p, ul, ol {
    margin: 0;
    padding: 0;
    border: 0;
    list-style: none;
}

/* 整体 */
#container {
    width: 1000px;
    margin: 0 auto;
}

/* 顶部 */
#top {
    width: 1000px;
    height: 100px;
    background: gray;
}

/* 主体 */
#main {
    display: flex;
    justify-content: space-between;
}

{% block head_style %}
{% endblock %}

/* 底部 */
#foot {
    width: 1000px;
    height: 100px;
    background: orange;
}

</style>
</head>
<body>
<!-- 整体 -->
<div id="container">
    <!-- 1.顶部 -->
    <div id="top">
        标题
    </div>
    <!-- 2.中间主体部分 -->
    <div id="main">
        {% block body_info %}
        {% endblock %}
    </div>
    <!-- 3.底部 -->
    <div id="foot">
        底部
    </div>

```

```

</div>
</body>
</html>

```

- 子类主页模板

```

{% extends "base.html" %}

{% block title %}首页{% endblock %}
{% block head_style %}
/* 主体左侧 */
#main #main_index {
    width: 1000px;
    height: 500px;
    background: pink;
}
{% endblock %}

{% block body_info %}
<!-- 2.1 主面中间的信息 -->
<div id="main_index">
    主页
</div>
{% endblock %}

```

- 子类商品列表模板

```

{% extends "base.html" %}

{% block title %}商品列表{% endblock %}
{% block head_style %}
/* 主体左侧 */
#main #left-type{
    width:300px;
    height:500px;
    background:pink;
}
/* 主体右侧 */
#main #right-goods{
    width:700px;
    height:500px;
    background:blue;
}
{% endblock %}

{% block body_info %}
<!-- 2.1 主面中间的信息 -->
<!-- 2.1主体左边 -->

```

```

<div id="left-type">
    <ul>
        <li>手机</li>
        <li>电脑</li>
        <li>户车</li>
    </ul>
</div>

{% block goods_info %}
<div id="right-goods">
    手机列表
</div>
{% endblock %}
{% endblock %}

```

```

{% extends "goods_list.html" %}

{% block goods_info %}
<div id="right-goods">
    电脑列表
</div>
{% endblock %}

```

```

{# 此模板继承自base.html #}
{% extends "base.html" %}

{% block title %}用户信息{% endblock %}
{% block head_style %}
    /* 主体左侧 */
    #main #left-list{
        width:300px;
        height:500px;
        background:pink;
    }
    /* 主体右侧 */
    #main #right-info{
        width:700px;
        height:500px;
        background:yellow;
    }
{% endblock %}

{% block body_info %}
<!-- 2.1 主面中间的信息 -->
<!-- 2.1主体左边 -->
<div id="left-list">
    <ul>
        <li>修改密码</li>
        <li>修改头像</li>
    </ul>

```

```

        <li>修改联系方式</li>
    </ul>
</div>

{% block user_detail %}
<div id="right-info">
    <div>原密码<input type="password"></div>
    <div>新密码<input type="password"></div>
    <div>新密码<input type="password"></div>
</div>
{% endblock %}
{% endblock %}

```

```

{# 此模板继承自user_info_page1.html #}
{% extends "user_info_page1.html" %}

{% block user_detail %}
<div id="right-info">
    <div>手机号<input type="text"></div>
    <div>家庭住址<input type="text"></div>
</div>
{% endblock %}

```

url 反向解析

- url()的name关键字参数
 - 作用:
 - 根据url 列表中的name=关键字传参给 url确定了个唯一确定的名字，在模板中，可以通过这个名字反向推断出此url信息
 - 语法
 - url(regex, views, kwargs=None, name="别名")
 - ex:
 - url(r'^user_login/\$', views.login, name="login")
 - 通过别名实现地址的反向解析 在模板中: {% url 'some-url-name' %} {% url '别名' %} {% url '别名' '参数值1' '参数值2' %}
 - 示例:

```

# file: url.py
from django.conf.urls import url
from . import views
urlpatterns = [
    ...
    url(r'^fav_list', views.fav_list, name="fav_list"),
    url(r'^fav_list/page/(\d+)', views.fav_list_page,
name="fav_list_page"),

```

```

]
# file: views.py
def fav_list(request):
    return render(request, 'fav_list.html')

def fav_list_page(request, page):
    return render(request, 'fav_list.html')

```

```

<html>
<head></head>
<body>
    <div><a href="/fav_list">我的收藏</a></div>
    <div><a href="{% url 'fav_list' %}">我的收藏</a></div>

    <a href="/fav_list/page/1">我的收藏第1页</a>
    <a href="/fav_list/page/2">我的收藏第2页</a>
    <a href="/fav_list/page/3">我的收藏第3页</a>
    <a href="{% url 'fav_list_page' '200' %}">我的收藏第4页</a>
</body>
</html>

```

静态文件

1. 什么是静态文件
 - 不能与服务器端做动态交互的文件都是静态文件
 - 如:图片,css,js,音频,视频,html文件(部分)
2. 静态文件配置
 - 在 settings.py 中配置一下两项内容:
 1. 配置静态文件的访问路径
 - 通过哪个url地址找静态文件
 - STATIC_URL = '/static/'
 - 说明:
 - 指定访问静态文件时是需要通过 /static/xxx或 127.0.0.1:8000/static/xxx
 - xxx 表示具体的静态资源位置
 2. 配置静态文件的存储路径
 - 静态文件在服务器端的保存位置
 - STATICFILES_DIRS=(os.path.join(BASE_DIR,'static'),)
 - 3. 示例:

```

# file: setting.py
STATICFILES_DIRS = [
    os.path.join(BASE_DIR, "static")
]

```

3. 访问静态文件


1. 使用静态文件的访问路径进行访问

- 访问路径: STATIC_URL=/static/
- 示例:

```


```

2. 通过 {% static %} 标签访问静态文件 {% static %} 表示的就是静态文件访问路径

1. 加载 static {% load static %}
 2. 使用静态资源时 语法:{% static '静态资源路径' %} 
- 示例:

```
# file: url.py
from . import views

urlpatterns = [
    url(r'^admin/', admin.site.urls),
    url(r'^show_image', views.show_image)
]
# file: views.py
from django.shortcuts import render

def show_image(request):
    return render(request, "show_image.html")
```

```
<html>
<head></head>
<body>
<h1>this is lena!</h1>

<h1>this is templates lena!</h1>
{% load static %}

</body>
</html>
```

○ 练习:

将之前课程阶段中的首页, 登录页, 购物车页

1. 127.0.0.1:8000 : 显示首页效果
2. 127.0.0.1:8000/login : 显示登录页
3. 127.0.0.1:8000/cart : 显示购物车页

处理好所有的静态文件

Django中的应用 - app

什么是应用(app)

- 应用在Django项目中是一个独立的业务模块,可以包含自己的路由,视图,... ..
- Django中,主文件夹是不处理用户具体请求的.主文件夹的作用是做项目的初始化以及请求的分发(分布式请求处理).具体的请求是由应用来进行处理的

创建应用app

- 创建应用的指令
 - python3 manage.py startapp 应用名称
 - 如:
 - python3 manage.py startapp music

Django应用的结构组成

1. `migrations` 文件夹
 - 保存数据迁移的中间文件
2. `__init__.py`
 - 应用子包的初始化文件
3. `admin.py`
 - 应用的后台管理配置文件
4. `apps.py`
 - 应用的属性配置文件
5. `models.py`
 - 与数据库相关的模型映射类文件
6. `tests.py`
 - 应用的单元测试文件
7. `views.py`
 - 定义视图处理函数的文件

- 配置安装应用
 - 在 settings.py 中配置应用, 让此应用能和整个项目融为一体

```
# file : settings.py
INSTALLED_APPS = [
    ... ..,
    '自定义应用名称'
]
```

- 如:


```
INSTALLED_APPS = [
    # ....
    'user', # 用户信息模块
    'music', # 收藏模块
]
```

- 应用的分布式路由
 - 使用include 函数让某个正则匹配后关联分支到某个app

```
# file : <项目名>/urls.py
from django.conf.urls import include

urlpatterns = [
    url(r'^admin/', admin.site.urls),
    url(r'^music/', include('music.urls')),
    url(r'^sport/', include('sport.urls')),
    url(r'^news/', include('news.urls')),
]

# file : <App名>/urls.py
from django.conf.urls import url
from . import views

urlpatterns = [
    # 购物车模块用到的路由
    url(r'^page1', views.page1),
    url(r'^page2', views.page2),
    url(r'^page3', views.page3),
    # ...
]
```

- 练习:

1. 创建三个应用
 1. 创建 music 应用, 并注册
 2. 创建 sport 应用, 并注册
 3. 创建 news 应用, 并注册
2. 创建分布式路由系统
 - 主路由配置只做分发
 - 每个应用中处理具体访问路径和视图
 - 1. 127.0.0.1:8000/music/index
交给 music 应用中的 index() 函数处理
 - 2. 127.0.0.1:8000/sport/index
交给 sport 应用中的 index() 函数处理
 - 3. 127.0.0.1:8000/news/index
交给 news 应用中的 index() 处理处理