讲师：李振良（阿良）

今天课题：《Django入门与进阶》上

学院官网：www.ctnrs.com

阿良微信

添加微信好友

DevOps技术栈

关注微信公众号

Django 入门与进阶（上）

- Django基本使用
- Django路由系统
- Django视图
- Djanog模板系统

Django 基本使用

- Django 是什么
- Django 发展历程
- 开发环境准备
- 创建项目
- 牛刀小试：第一个页面、第二个页面
- Django工作流程

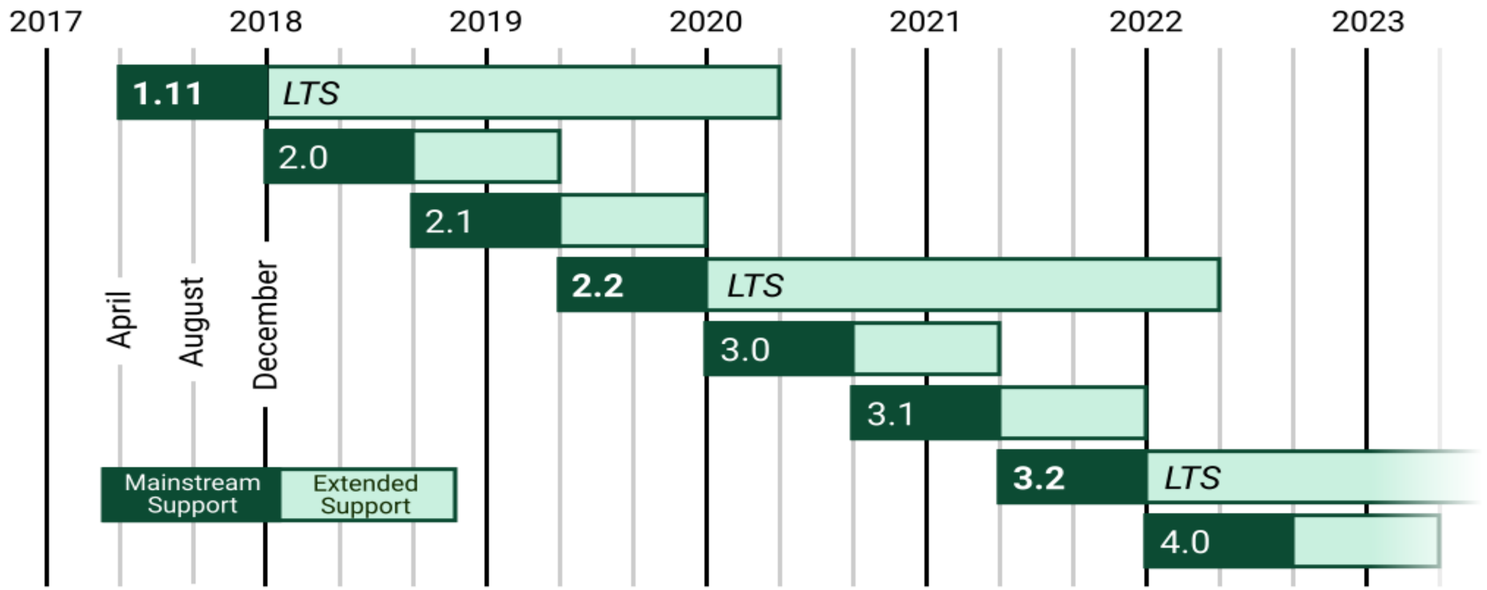
Django是什么

Django是Python的一个主流Web框架，提供一站式解决方案，开发成本低，内建ORM、数据管理后台、登录认证、表单、RESTAPI等功能，适合开发中大型项目。

其他Web框架：

- Flask（轻量级）
- Tornado（异步）

Django发展历程



目前最新版本3.0， Python版本推荐3.6、3.7、3.8

官方文档：https://docs.djangoproject.com/zh-hans/3.1/

开发环境准备

软件	安装方式
Python3.8	官方网站下载安装程序：https://www.python.org/downloads/windows/
Django3.0	pip install django==3.0.5
Pycharm Pro	官方网站下载安装程序
PyMySQL	pip install pymysql
MySQL5.7	https://dev.mysql.com/downloads/mysql/

创建项目

1、创建项目

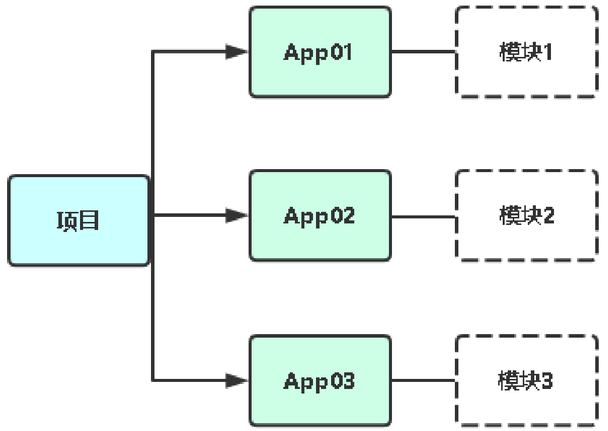
django-admin startproject devops

2、创建应用

python manage.py startapp myapp

3、运行项目

python manage.py runserver 0.0.0.0:8888



牛刀小试：第一个页面

1、添加URL路由

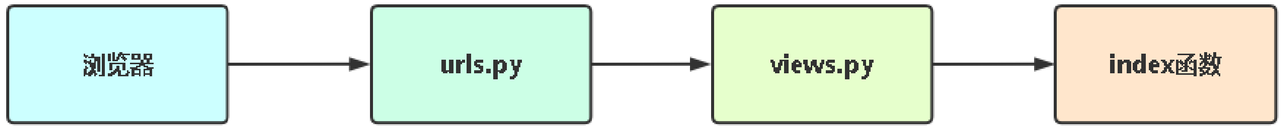
```
# devops/urls.py
from django.contrib import admin
from django.urls import path
from devops import views

urlpatterns = [
    path('index', views.index),          # /index访问
    path('admin/', admin.site.urls),
]
```

2、添加视图

```
# devops/views.py
from django.http import HttpResponse # 导入处理响应模块

def index(request):
    return HttpResponse("首页")
```



牛刀小试：第二个页面

1、添加URL路由

```
# devops/urls.py
from django.contrib import admin
from django.urls import path
from devops import views

urlpatterns = [
    path('index', views.index),
    path('logs', views.logs),
    path('admin/', admin.site.urls),
]
```

2、添加视图

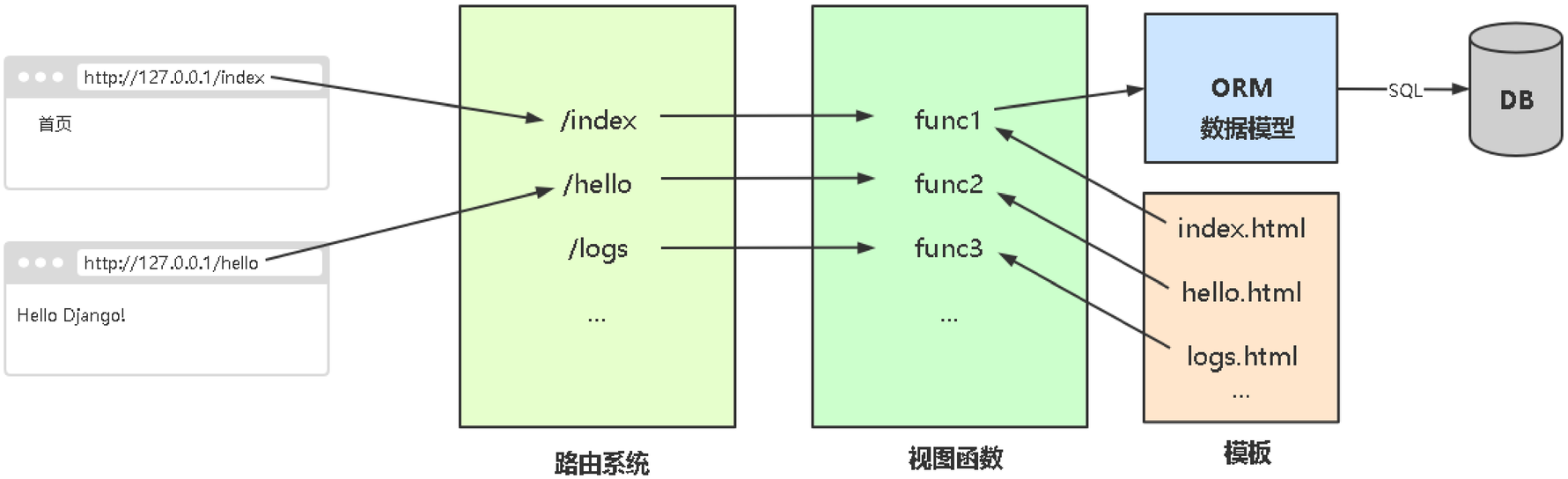
```
# devops/views.py
from django.shortcuts import render
from django.http import HttpResponse

def index(request):
    return HttpResponse("首页")
def logs(request):
    import os
    current_dir = os.path.dirname(os.path.abspath(__file__))
    with open(current_dir + '\\access.log') as f:
        result = f.read()
    return render(request, "logs.html", {"result": result})
```

3、创建HTML模板

```
# devops/templates/logs.html
<!DOCTYPE html>
<html lang="en">
<head>
    <meta charset="UTF-8">
    <title>日志</title>
</head>
<body>
<h1>Nginx访问日志</h1>
<pre>{{ result }}</pre>
</body>
</html>
```

Django工作流程



工作流程图

Django URL路由系统

- URL路由系统是什么
- URL配置
- URL正则表达式匹配
- URL名称

URL路由系统是什么

简而言之，路由系统就是URL路径和视图函数的一个对应关系，也可以称为转发器。

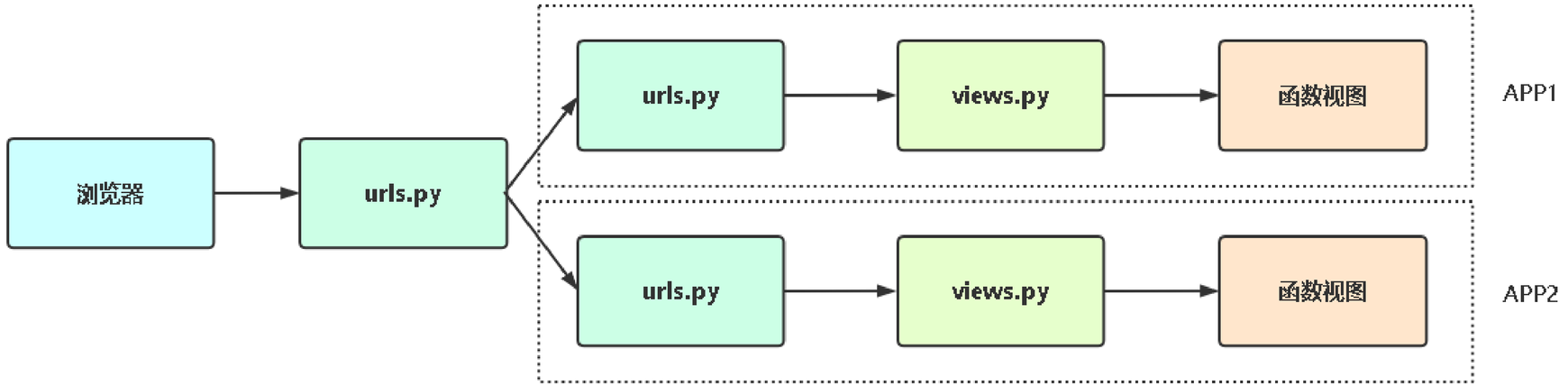
URL配置

URL路由系统格式：

```
# devops/urls.py
urlpatterns = [
        path(regex, view, kwargs=None, name=None)
]
```

- urlpatterns：一个列表，每一个path()函数是一个元素，对应一个视图。
- regex：一个字符串或者正则表达式，匹配URL。
- view：对应一个函数视图或者类视图（as_view()的结果），必须返回一个HttpResponse对象，Django将这个对象转换成一个HTTP响应。
- kwargs：可选，字典形式数据传递给对应视图。
- name：可选，URL名称

URL路由分发



URL路由分发好处：urls配置解耦，方便管理

URL路由分发

示例：

```
# devops/urls.py
urlpatterns = [
    path('index', views.index),
    path('myapp/', include('myapp.urls')),
]

# myapp/urls.py
urlpatterns = [
    path("hello", views.hello)
]
```

访问地址：http://127.0.0.1:8000/myapp/hello

URL正则表达式匹配

URL路径也可以使用正则表达式匹配，re_path()替代path()

示例：博客文章归档访问形式

```
from django.urls import re_path
from devops import views

urlpatterns = [
    re_path('articles/2020/$', views.specified_2020),
    re_path('^articles/([0-9]{4})/$', views.year_archive),
    re_path('^articles/([0-9]{4})/([0-9]{2})/$', views.month_archive),
    re_path('^articles/([0-9]{4})/([0-9]{2})/([0-9]+)/$', views.article_detail),
]
```

视图：

```
def specified_2020(request):
    return HttpResponse("指定2020年 文章列表")
def year_archive(request, year):
    return HttpResponse("%s年 文章列表" % year)
def month_archive(request, year, month):
    return HttpResponse("%s年/%s月 文章列表" % (year, month))
def article_detail(request, year, month, id):
    return HttpResponse("%s年/%s月 文章ID: %s" %(year, month, id))
```

URL正则表达式匹配

命名分组语法：(?P<name>pattern) 其中name是名称，pattern是匹配的正则表达式

示例：博客文章归档访问形式

```
from django.urls import re_path
from devops import views

urlpatterns = [
    re_path('articles/2020/$', views.specified_2020),
    re_path('^articles/(?P<year>[0-9]{4})/$', views.year_archive),
    re_path('^articles/(?P<year>[0-9]{4})/(?P<month>[0-9]{2})/$', views.month_archive),
    re_path('^articles/(?P<year>[0-9]{4})/(?P<month>[0-9]{2})/(?P<id>[0-9]+)/$', views.article_detail),
]
```

URL名称

在前端代码里经常会指定URL，例如超链接，提交表单等，这时用URL反查就方便多了。

之前：

```
<a href="/hello">你好</a>
```

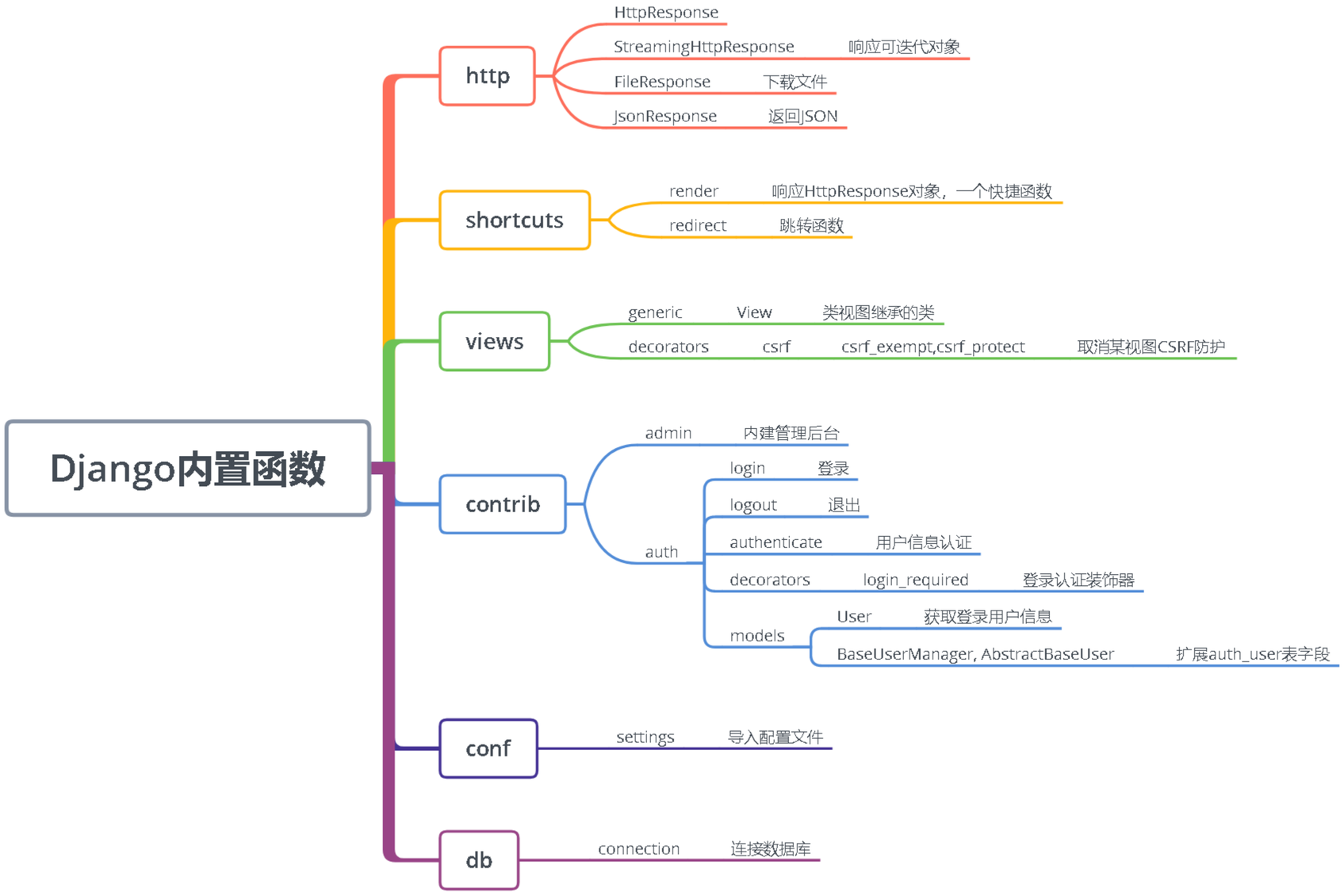
之后：

```
<a href="{% url 'hello' %}">你好</a>
```

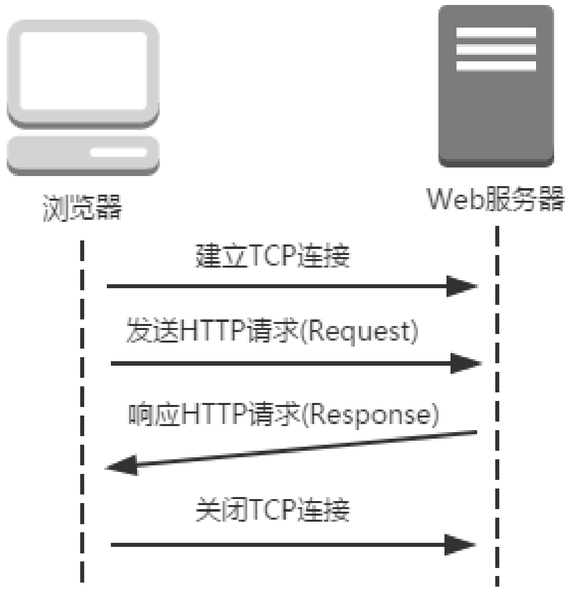
Django 视图

- Django内置函数
- HttpRequest对象
- HttpRespace对象

Django内置函数

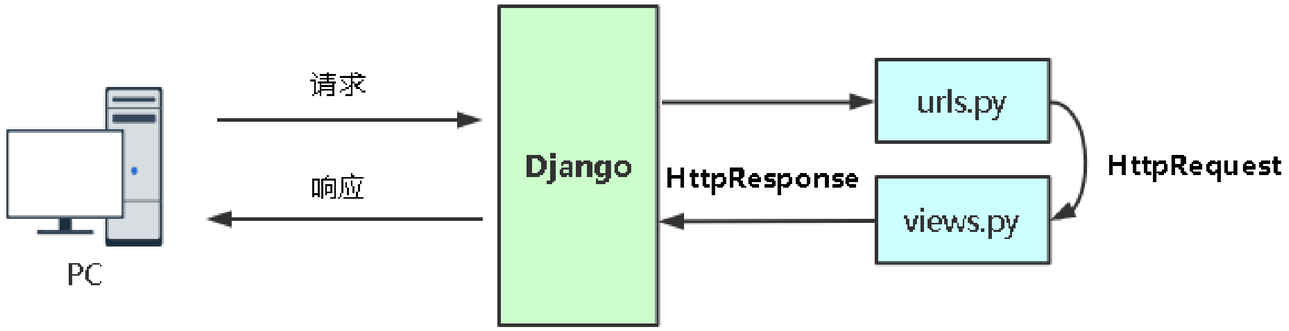

Django内置函数
http
HttpResponse
StreamingHttpResponse 响应可迭代对象
FileResponse 下载文件
JsonResponse 返回JSON
shortcuts
render 响应HttpResponse对象，一个快捷函数
redirect 跳转函数
views
generic View 类视图继承的类
decorators csrf csrf_exempt,csrf_protect 取消某视图CSRF防护
contrib
admin 内建管理后台
auth
login 登录
logout 退出
authenticate 用户信息认证
decorators login_required 登录认证装饰器
models
User 获取登录用户信息
BaseUserManager, AbstractBaseUser 扩展auth_user表字段
conf
settings 导入配置文件
db
connection 连接数据库

HTTP请求流程



HTTP工作流程图

HTTP请求流程



HttpRequest对象：常用属性

Django会建立一个包含请求源数据的HttpRequest对象，当Django加载对应的视图时，HttpRequest对象将作为函数视图的第一个参数（request），每个视图负责返回一个HttpResponse对象。

例如：

```
def index(request):
    return HttpResponse("首页")
```

属性	描述
request.scheme	表示请求协议的字符串（http或https）
request.body	原始HTTP请求正文
request.path	一个字符串，请求页面的完整路径，不包含域名
request.method	一个字符串，请求的HTTP方法，比如GET/POST等
request.GET	GET请求所有参数，返回QueryDict类型，类似于字典
request.POST	POST请求所有参数，返回QueryDict类型
request.COOKIES	以字典格式返回Cookie
request.session	可读写的类似于字典的对象，表示当前的会话
request.FILES	所有上传的文件
request.META	返回字典，包含所有的HTTP请求头。比如客户端IP，Referer等

HttpRequest对象：常用方法

方法	描述
request.get_host()	服务器主机地址和端口
request.get_port()	服务器端口
request.get_full_path()	请求页面完整路径和查询参数
request.get_raw_uri()	请求页面URL所有信息，包括主机名、路径和参数

HttpRequest对象：接收URL参数

URL参数形式：http://www.ctnrs.com/demo/?id=1&value=100

```
def url_args(request):
    args1 = request.GET[‘a’]
    args2 = request.GET[‘b’]
    return HttpResponse(int(args1) + int(args2))
```

HttpRequest对象：QueryDict对象

request.GET和request.POST返回的都是一个QueryDict对象，类似于字典。

```
def index(request):
    req = request.GET
    print(type(req))
    return HttpResponse("首页")
```

方法	描述
req.get(key,default)	返回key的值，如果key不存在返回default
req.items()	返回迭代器，键值
req.values()	返回迭代器，所有键的值
req.keys()	返回所有键
req.getlist(key,deafult)	返回key的值作为列表，如果key不存在返回default
req.lists()	返回迭代器，所有键的值作为列表
req.dict()	返回字典

HttpRequest对象：示例

示例1：表单GET提交，例如搜索页面

示例2：表单POST提交，例如登录页面

示例3：上传文件，例如修改头像

HttpRequest对象：小结

request.GET应用场景：

- 获取客户端信息、请求页面情况
- 接收客户端上传的数据，例如文件
- 根据客户端特定信息做相应的处理，例如根据请求方法

HttpResponse对象：HttpResponse函数

HttpResponse函数：给浏览器返回数据。

语法：HTTPResponse(content=响应体，content_type=响应体数据类型，status=状态码)

示例：返回HTML内容

```
from django.http import HttpResponse
def hello(request):
    return HttpResponse("<h1>Hello Django!</h1>")
```

示例：设置响应头

```
from django.http import HttpResponse
def hello(request):
    res = HttpResponse("Hello APP!")
    res['name'] = "aliang"
    res.status_code = 302
```

HttpResponse对象：render函数

render指定模板，返回一个渲染后的HttpResponse对象。

语法：render(request, template_name, context=None, content_type=None, status=None, using=None)

- request：固定参数，django封装的请求
- template_name：返回html模板
- context：传入模板中的内容，用于渲染模板，默认空字典

示例：

```
from django.shortcuts import render
from datetime import datetime
def current_datetime(request):
    now = datetime.now()
    return render(request, 'demo.html', {'datetime': now})
```

HttpResponse对象：redirect函数

redirect函数：重定向，发起第二次请求

语法：redirect(to, *args, **kwargs)

参数可以是：

- 一个视图
- 一个绝对的或者相对的URL
- 一个模型，对象是重定向的URL

示例：

```
from django.shortcuts import redirect
def test_redirect(request):
    return redirect('https://www.baidu.com')
```

HttpResponse对象：StreamingHttpResponse函数

StreamingHttpResponse函数：流式响应可迭代对象

HttpResponse对象：StreamingHttpResponse函数

示例：下载文件

URL路由：

```
re_path('^download/$', views.download),
re_path(r'^down_file/(?P<filename>.*)$', views.down_file, name="down_file")
```

视图：

```
from django.http import StreamingHttpResponse
import os
def download(request):
    file_list = os.listdir('upload')
    return render(request, "download.html", {'file_list': file_list})
def down_file(request, filename):
    file_path = os.path.join('upload', filename)
    response = StreamingHttpResponse(open(file_path, 'rb'))
    response['Content-Type'] = 'application/octet-stream'
    response['Content-Disposition'] = 'attachment; filename=%s' %(os.path.basename(file_path))
   return response
```

模板：

```
<!DOCTYPE html>
<html lang="en">
<head>
    <meta charset="UTF-8">
    <title>文件列表</title>
</head>
<body>
   {% for i in  file_list %}
      <p><a href="{% url 'down_file' i  %}">{{ i }}</a></p>
   {% endfor %}
</body>
</html>
```

HttpResponse对象： FileResponse函数

FileResponse函数：如果提供文件下载建议方法

示例：下载文件

```
def down_file(request, filename):
    file_path = os.path.join('upload', filename)
    response = FileResponse(open(file_path, 'rb'))
    response['Content-Type'] = 'application/octet-stream'
    response['Content-Disposition'] = 'attachment; filename=%s' %(os.path.basename(file_path))
    return response
```

HttpResponse对象： JsonResponse函数

JsonResponse函数：响应一个JSON对象

示例：下载文件

```
from django.http import JsonResponse
def test_response(request):
    res = { ‘foo’ : ‘bar’ }
    return JsonResponse(res)
```

Django 模板系统

- 模板是什么
- 变量
- 标签
- 常用过滤器
- 注释
- 模板继承
- 模板导入
- 引用静态文件

模板系统是什么

Django模板系统：用于自动渲染一个文本文件，一般用于HTML页面。模板引擎渲染的最终HTML内容返回给客户端浏览器。

模板文件有两部分组成：

- 静态部分，例如html、css、js
- 动态部分，django模板语言，类似于jinja语法

变量：介绍

变量定义：在函数视图render中的context传入，类似于字典对象。

变量在模板中引用，格式：{{ key }}

变量：示例

示例：

```
def hello(request):
    user = {'name': '阿良', 'property': {'sex': '男', 'age': 30}}
    return render(request, 'user.html', {'user': user})
```

```
<!DOCTYPE html>
<html lang="en">
<head>
    <meta charset="UTF-8">
    <title>用户信息</title>
</head>
<body>
传递过来的字典: {{ user }}<br>
姓名: {{ user.name }}<br>
性别: {{ user.property.sex }}<br>
年龄: {{ user.property.age }}<br>
</body>
</html>
```

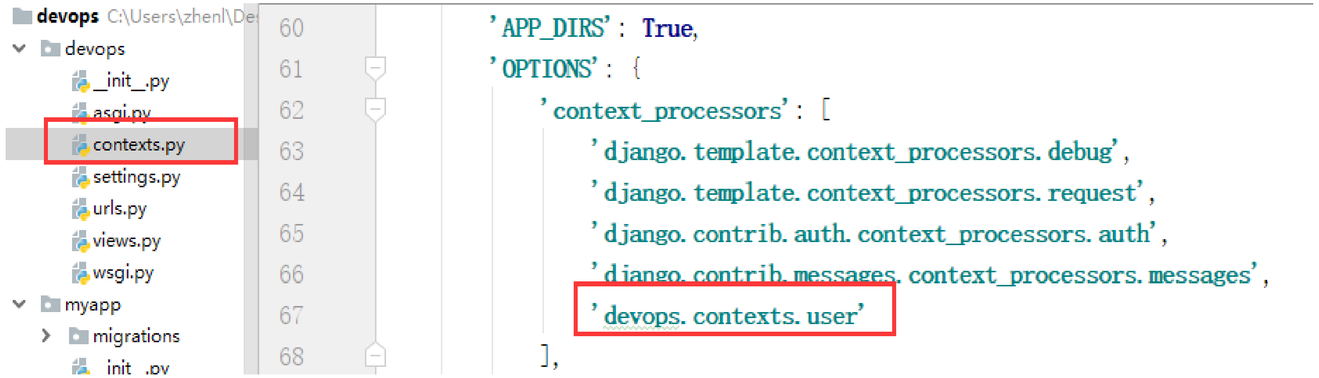
变量：设置全局变量

示例：设置全局变量

1、在项目目下创建contexts.py文件

```
def user(request):
    username = request.session.get('username')
    return {'username': username}
```

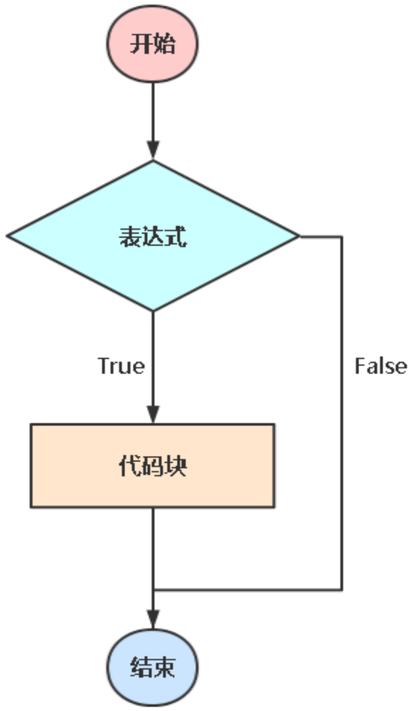
2、在settings.py文件中添加你的上下文处理器



模板：

```
<h3>欢迎：{{ username }}</h3>
```

标签：条件判断



if条件判断：判定给定的条件是否满足（True或False），根据判断的结果决定执行的语句。

语法：

```
{% if <表达式> %}
    <内容块>
{% elif <表达式> %}
    <内容块>
{% else %}
    <内容块>
{% endif %}
```

标签：条件判断

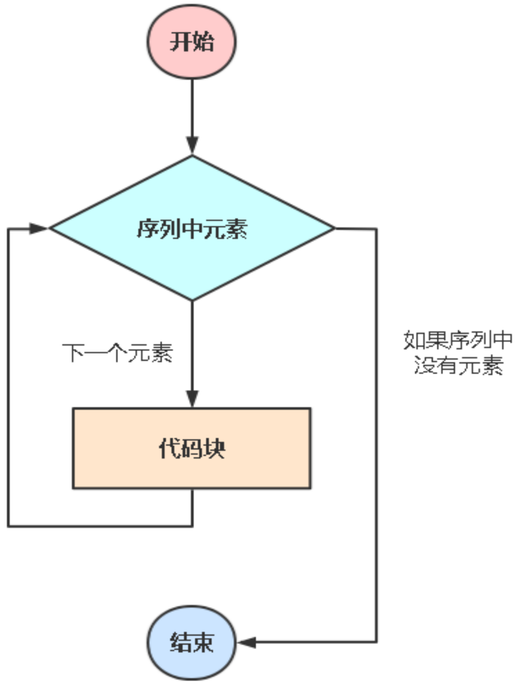
```
# 相等执行内容块
{% ifequal <值1> <值2> %}
    <内容块>
{% endifequal %}

# 不相等执行内容块
{% ifnotequal <值1> <值2> %}
    <内容块>
{% endifnotequal %}
```

标签：操作符

类型	操作符
比较操作符	• == 等于 • != 不等于 • > 大于 • < 小于 • >= 大于等于 • <= 小于等于
逻辑操作符	• and 与 • or 或
成员操作符	• not 逻辑否定 • in 包含在内

标签：循环



for循环：一般用于遍历数据类型的元素进行处理，例如列表。

语法：

```
{% for <变量> in <序列> %}
    <内容块>
{% endfor %}
```

标签：forloop变量

forloop是在{% for %}标签中生成的变量，用于获取当前循环进展信息。

变量	描述
forloop.counter	循环计数器，当前循环的索引从1开始
forloop.counter0	循环计数器，当前循环的索引从0开始
forloop.revcounter	当前循环倒数计数，最后一次循环为1，反向计数
forloop.revcounter0	当前循环倒数计数，最后一次循环为0，反向计数
forloop.first	当前循环为第一个循环时，该变量为True
forloop.last	当前循环为最后一个循环时，该变量为True
forloop.parentloop	再嵌套循环中，指向当前循环的上级循环

标签：for empty

for...empty 当循环的序列为空时，执行empty下面的内容。

语法：

{% for <变量> in <序列> %}

    <遍历>

{% empty %}

    <代码块>

{% endfor %}

常用过滤器

过滤器：在变量被显示前修改值的一种方法。

语法：{{ value | 过滤器:参数 }}

过滤器	说明	示例
add	将两个值转换为整数相加	{{ 11 | add:"6" }} 结果 17
cut	切除字符。从给定字符串中删除arg的所有值。	{{ "hello world" | cut:"w" }} 结果 hello orld
default	如果值的计算结果为 False，则使用给定的默认值。否则，使用该值。	{{ "" | default:"hello world" }} 结果 hello world
first	返回第一个元素	{{ "hello world" | first }} 结果 h
last	返回最后一个元素	{{ "hello world" | last }} 结果 d
join	使用字符串连接列表，如Python的 str.join(list)	{{ abc | join:"," }} 结果 1,2,3 # abc = [1,2,3]
length	返回值的长度。这适用于字符串和列表	{{ "hello world" | length }} 结果 11
lower	将字 符串转换为小写	{{ "AAA" | lower }} 结果 aaa
upper	将字符串转换为大写	{{ "aaa" | upper }} 结果 AAA
slice	切片, 类似于Python中的切片操作。	{{ "hello world" | slice:"2:" }} 结果 llo world
title	所有单词首字母大写	{{ "aaa" | title }} 结果 Aaa
truncatechars	如果长度大于指定的字符数，则截断字符串。截断的字符串将以可翻译的省略号序列（“…”）结束	{{ "hello world" | truncatechars:2 }} 结果 h…
filesizeformat	将该值格式化为“人类可读”文件大小（即 ‘13 KB ‘，‘4.1 MB ‘， ‘102 bytes ‘等）。	{{ 10000 | filesizeformat }} 结果 9.8 KB
floatformat	当不带参数时，将一个浮点数舍入到小数点后一位，但前提是要显示一个小数部分。	{{ 1.33333333 | floatformat }} 结果 1.3 floatformat:2 指定保留的小数位数

参考资料：https://docs.djangoproject.com/zh-hans/3.1/ref/templates/builtins/

常用过滤器：自定义过滤器

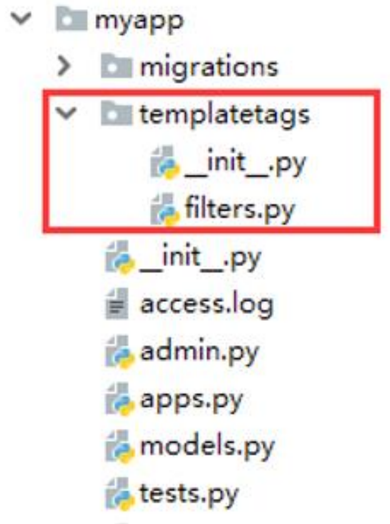
1、在app下创建templatetags目录

2、自定义过滤器函数

```
from django.template import  Library
register = Library()  # 注册过滤器对象
@register.filter   # 通过装饰注册自定义过滤器
def func(n):
    return n / 2
```

3、在模板中使用

```
{% load filters %}
{{ 123 | func }}
```



标签：注释

注释：

{# 注释内容 #}

模板继承

模板继承主要是为了提高代码重用，减轻开发人员的工作量。

典型应用：网站的头部、尾部信息。

1、定义一个基础模板，也称为母板，这个页面存放整个网站共用的内容

templates/base.html

2、在子模板继承这个母版

{% extends ‘base.html’ %}

3、在基础模板预留子模板差异化内容

{% block 名称 %} 预留区域 {% endblock %}

4、在子模板里同样语法引用并填充预留区域内容

模板导入

模板导入：导入一个模板（一般是某个网页功能）到当前模板

将一个功能创建为模板：

```
# templates/hello.html
<style>
    .hello {
        background-color: red;
    }
</style>
<div class="hello">
    子模板
</div>
```

模板导入：

```
{% extends 'base.html' %}
{% block title %}首页{% endblock %}
{% block context %}
    <h1>这是首页！</h1>
    {% include "hello.html" %}
{% endblock %}
```

引用静态文件

- STATICFILES_DIRS：告诉Django哪个目录是“静态文件的文件夹”
- STATIC_ROOT：指出浏览器访问静态文件“根路径”

1、在settings.py配置

```
BASE_DIR = os.path.dirname(os.path.dirname(os.path.abspath(__file__)))
STATICFILES_DIRS = (
    os.path.join(BASE_DIR, 'static'),
)
STATIC_URL = '/static/'
```

2、在模板文件引用静态文件

```
<link rel="stylesheet" href="/static/main.css">
```

或者

```
<link rel="stylesheet" href="{% static 'main.css' %}">
```

谢谢

阿良微信

添加微信好友

DevOps技术栈

关注微信公众号

DevOps实战学院：www.ctnrs.com