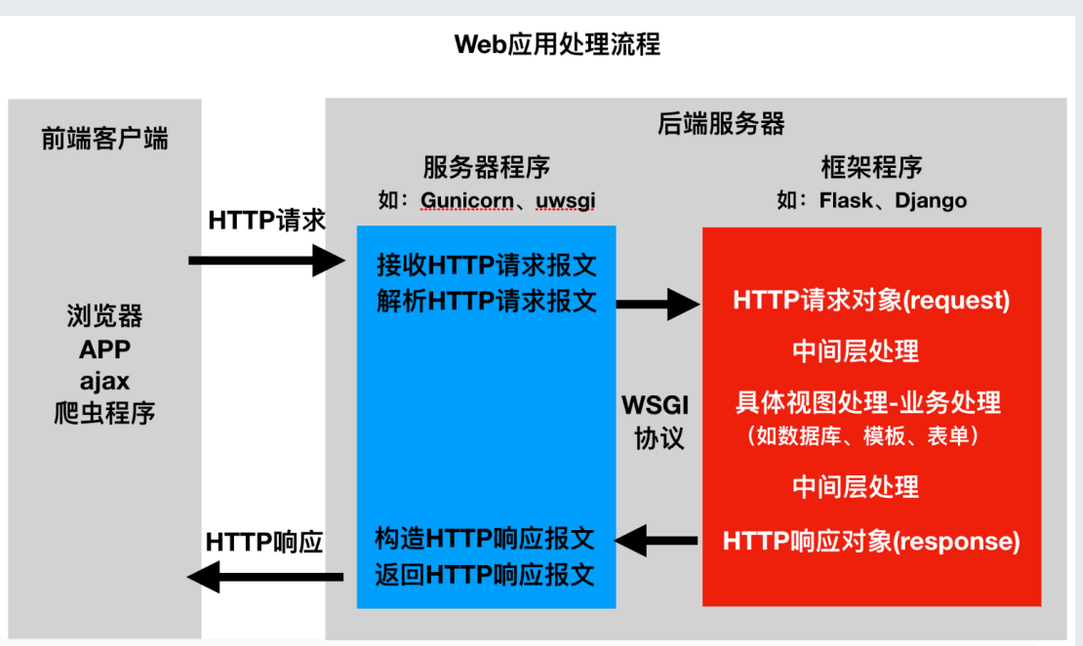
Django框架基础-day01

- Web框架要点
  - 1.Web应用程序处理流程

  - 2.Web框架的意义
    - 1.用于搭建Web应用程序
    - 2.免去不同Web应用相同代码部分的重复编写，只需关心Web应用核心的业务逻辑实现
  - 3.Web应用程序的本质
    - 1.接收并解析HTTP请求，获取具体的请求信息
    - 2.处理本次HTTP请求，即完成本次请求的业务逻辑处理
    - 3.构造并返回处理结果——HTTP响应
  - 4.如何学习Web框架说明
    - 如何搭建工程程序
      - 工程的组建
      - 工程的配置
      - 路由定义
      - 视图函数定义
    - 如何获取请求数据（操作request对象）
    - 如何构造响应数据（构造response对象）
    - 如何使用中间层
    - 框架提供的其他功能组件的使用
      - 数据库
      - 模板
      - admin
- Django框架介绍
  - 1.简介和特点
    - 1.简介
      - 是用python语言写的开源web开发框架，并遵循MVC设计。
      - Django的**主要目的是简便、快速的开发数据库驱动的网站。**
    - 2.特点
      - 遵循MVC设计
      - 重量级的Web框架，功能齐全
  - 2.设计模式
    - MVC
      - **MVC，其核心思想是分工、解耦，让不同的代码块之间降低耦合，增强代码的可扩展性和可移植性，实现向后兼容。**
      - M全拼为Model，主要封装对数据库层的访问，对数据库中的数据进行增、删、改、查操作。
      - V全拼为View，用于封装结果，生成页面展示的html内容。
      - C全拼为Controller，用于接收请求，处理业务逻辑，与Model和View交互，返回结果。
    - MVT
      - M全拼为Model，与MVC中的M功能相同，负责和数据库交互，进行数据处理。
      - V全拼为View，与MVC中的C功能相同，接收请求，进行业务处理，返回应答。
      - T全拼为Template，与MVC中的V功能相同，负责封装构造要返回的html。
  - 3.相关文档
    - 官方网站 https://www.djangoproject.com/
    - GitHub源码 https://github.com/django/django
    - 中文版文档 https://yiyibooks.cn/xx/Django_1.11.6/index.html
- 搭建Django工程
  - 1.安装Django框架
    - 创建虚拟环境 mkvirtualenv 虚拟环境名字 -p python3
    - 安装框架 pip install django==1.11.11
  - 2.创建Django工程目录
    - 命令 django-admin startproject 工程名称
    - 步骤
      - 进入要创建工程的目录
      - django-admin startproject demo
    - 启动测试服务器 python manage.py runserver
  - 3.创建Django子应用
    - 说明 **Django的视图编写是放在子应用中的。**
    - 命令 `python manage.py startapp 子应用名称`
    - 步骤
      - 进入到项目目录中
      - `python manage.py startapp users`
    - 安装子应用

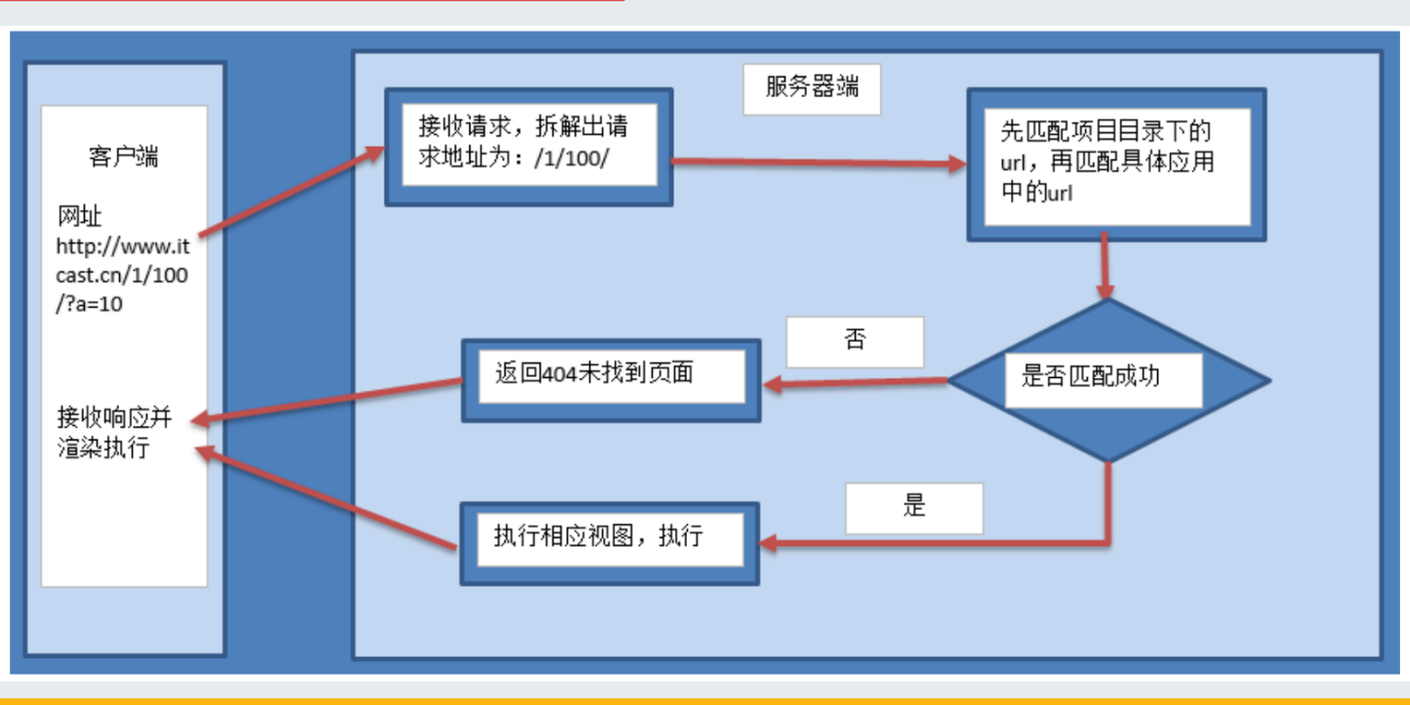
      ```
      INSTALLED_APPS = [
          'django.contrib.admin',
          'django.contrib.auth',
          'django.contrib.contenttypes',
          'django.contrib.sessions',
          'django.contrib.messages',
          'django.contrib.staticfiles',
          'users.apps.UsersConfig', # 安装users应用
      ]
      ```
  - 4.创建视图并定义路由
    - 1.总路由入口 `ROOT_URLCONF = 'demo.urls'`
    - 2.总路由定义

      ```
      urlpatterns = [
          url(r'^admin/', admin.site.urls),
          # 将users应用中的所有路由包含进来
          url(r'^users/', include('users.urls')),
      ]
      ```
      - demo.urls
    - 3.子路由定义

      ```
      urlpatterns = [
          # url(r'^路径$', views.视图),
          url(r'^index/$', views.index),
      ]
      ```
      - Users.urls
    - 4.视图定义

      ```
      users.views
      def index(request):
          """
          定义django函数视图
          :param request: 传入到函数视图的请求对象
          :return: 响应对象
          """

          return HttpResponse('hello world')
      ```
- 配置文件、静态文件、路由说明
  - 1.路由的使用
    - 1.Django解析路由的流程

    - 2.路由定义位置
      - 1.总路由+子路由
        - 子路由 `url(r'^index/$', views.index),`
        - 总路由 `url(r'^users/', include('users.urls')),`
      - 2.总路由 `url(r'^users/index/$', views.index),`
      - 3.子路由
        - 总路由 `url(r'^', include('users.urls')),`
        - 子路由 `url(r'^users/index/$', views.index),`
    - 3.路由解析的顺序
      - 1.urlpatterns里面的路由解析的顺序

        ```
        urlpatterns = [
            # url(r'^路径$', views.视图),
            url(r'^index/$', views.index, name='index'),

            # 以下代码演示路由自上而下匹配规则
            # url(r'^say', views.say), # 这个会拦截下面的路由
            # url(r'^sayhello', views.say_hello),

            url(r'^say$', views.say), # 定义路由要指明开始和结束
            url(r'^sayhello$', views.say_hello),
        ]
        ```
        - 自上而下
      - 2.结论 **子路由需要有开始和结尾**
    - 4.路径以斜线结尾说明
      - 1.正则匹配路由结尾带斜线
        - 当找不到该路由时 浏览器会自动的重定向到带斜线的路由
        - 如果用户输入的URL路径没有斜线
      - 2.正则匹配路由结尾不带斜线
        - 如果用户输入的URL路径有斜线
        - 直接报404错误
  - 2.settings.py配置文件说明
  - 3.静态文件使用
    - 1.配置静态文件的访问

      ```
      # 静态文件访问的路由
      STATIC_URL = '/static/'
      # 配置静态文件加载目录
      STATICFILES_DIRS = [
          os.path.join(BASE_DIR, 'static_files')
      ]
      ```
    - 2.准备静态文件1
      - 项目根路径/静态文件夹/静态文件
      - 项目根路径/static_files/index.html
      - http://127.0.0.1:8000/static/index.html
    - 3.准备静态文件2
      - 项目根路径/static_files/goods/detail.html
      - http://127.0.0.1:8000/static/goods/index.html
- 请求对象
  - 1.利用HTTP协议向服务器传参的几种途径
    - 请求行中的路径
    - 查询字符串
    - 请求体
    - 请求头
  - 2.使用正则提取URL中参数
    - 1.位置参数

      ```
      # url(r'^weather1/([a-z]+)/(\d{4})/$', views.weather),
      def weather1(request, city, year):
          """读取正则中的组提取的参数"""

          print('city=%s' % city)
          print('year=%s' % year)

          return HttpResponse('OK')
      ```
    - 2.关键字参数

      ```
      # url(r'^weather2/(?P<city>[a-z]+)/(?P<year>\d{4})/$', views.weather2),
      def weather2(request, year, city):
          """读取正则中的关键字提取的参数"""

          print('city=%s' % city)
          print('year=%s' % year)

          return HttpResponse('OK')
      ```
  - 3.提取查询字符串参数
    - 1.需求
      - 提取问号后面的查询字符串
      - /get_query_str/***?a=10&b=20&a=30***
    - 2.属性 GET
      - request.GET
      - 返回QueryDict类型的对象
    - 3.QueryDict类型
      - 可以存储一键一值和一键多值
      - 提供get()方法 读取一键一值
      - 提供getlist()方法 读取一键多值
    - 4.代码演练

      ```
      # /get_query_str/?a=10&b=20&a=30
      def weather3(request):
          """提取查询字符串参数"""

          a = request.GET.get('a')
          b = request.GET.get('b')
          a_list = request.GET.getlist('a')
          print(a,b,a_list)

          return HttpResponse('OK')
      ```
  - 4.提取请求体中的参数
    - 1.提取请求体中的表单数据
      - 1.属性
        - request.POST
        - POST
        - 返回QueryDict类型的对象
      - 2.代码演练

        ```
        # url(r'^get_body_form/$', views.get_body_form),
        def get_body_form(request):
            """演示获取请求体中的表单"""

            a = request.POST.get('a')
            b = request.POST.get('b')
            alist = request.POST.getlist('a')
            print(a)
            print(b)
            print(alist)
            return HttpResponse('OK')
        ```
    - 2.提取请求体中的非表单数据
      - 1.属性
        - body
        - request.body
      - 2.代码演练

        ```
        # url(r'^get_body_json/$', views.get_body_json),
        # {"a": 1, "b": 2}
        def get_body_json(request):
            """演示获取请求体中的非表单数据"""

            print(request.META['CONTENT_TYPE'])
            print(request.method, request.path)

            json_str = request.body
            json_str = json_str.decode()
            req_data = json.loads(json_str)
            print(req_data['a'])
            print(req_data['b'])
            return HttpResponse('OK')
        ```
  - 5.提取请求头中的信息
    - 1.提取方案

      可以通过**request.META**属性获取请求头headers中的数据，**request.META为字典类型。**

      常见的请求头如：

      - `CONTENT_LENGTH` – The length of the request body (as a string).
      - `CONTENT_TYPE` – The MIME type of the request body.
      - `HTTP_ACCEPT` – Acceptable content types for the response.
      - `HTTP_ACCEPT_ENCODING` – Acceptable encodings for the response.
      - `HTTP_ACCEPT_LANGUAGE` – Acceptable languages for the response.
      - `HTTP_HOST` – The HTTP Host header sent by the client.
      - `HTTP_REFERER` – The referring page, if any.
      - `HTTP_USER_AGENT` – The client's user-agent string.
      - `QUERY_STRING` – The query string, as a single (unparsed) string.
      - `REMOTE_ADDR` – The IP address of the client.
      - `REMOTE_HOST` – The hostname of the client.
      - `REMOTE_USER` – The user authenticated by the Web server, if any.
      - `REQUEST_METHOD` – A string such as `"GET"` or `"POST"`.
      - `SERVER_NAME` – The hostname of the server.
      - `SERVER_PORT` – The port of the server (as a string).
    - 2.代码实现 `print(request.META['CONTENT_TYPE'])`
  - 6.其他请求报文信息
    - **method**：一个字符串，表示请求使用的HTTP方法，常用值包括：'GET'、'POST'。
    - **user：请求的用户对象。**
    - path：一个字符串，表示请求的页面的完整路径，不包含域名和参数部分。
    - encoding：一个字符串，表示提交的数据的编码方式。
      - 如果为None则表示使用浏览器的默认设置，一般为utf-8。
      - 这个属性是可写的，可以通过修改它来修改访问表单数据使用的编码，接下来对属性的任何访问将使用新的encoding值。
    - FILES：一个类似于字典的对象，包含所有的上传文件。
    - `print(request.method, request.path)`
- 响应对象
  - 1.HttpResponse
    - 1.常规用法

      ```
      # HttpResponse(content=响应体, content_type=响应体数据类型, status=状态码)
      # 注意点：响应体参数是必传的

      return HttpResponse('OK', content_type='text/html', status=200)
      ```
    - 2.自定义响应头

      ```
      # 创建响应对象
      response = HttpResponse('OK', content_type='text/html', status=200)
      # 自定义响应头：键值对；自定义响应头键 Itcast, 值 Python
      response['Itcast'] = 'Python'

      return response
      ```
  - 2.JsonResponse

    ```
    # GET /response_json/ (表示用户向我们索取json)
    def response_json(request):
        """响应json数据"""

        # 准备字典:json字符串的key，需要双引号
        json_dict = {"name":"chao", "age":2.5}

        # 默认会设置状态码
        # 默认会将数据类型设置为 Content-Type: application/json
        # 默认会将json字典转json字符串,再将json字符串转二进制响应给用户
        return JsonResponse(json_dict)
    ```
  - 3.重定向

    ```
    # GET /response_redirect/
    def response_redirect(request):
        """重定向"""

        # 需求：重定向到users应用的index视图
        # 使用最原始的方式
        # return redirect('/users/index/')

        # 使用反向解析实现重定向
        return redirect(reverse('users:index'))
    ```
    - 路由命名空间和reverse反解析
      - 1.定义命名空间
        - `url(r'^users/', include('users.urls', namespace='users')),`
        - `url(r'^index/$', views.index, name='index'),`
      - 2.使用反向解析
        - `'users/index/'`
        - `url = reverse('users:index')`
  - 4.render 响应模板数据的，后续讲解模板时会使用