----------- 4. Python Web框架 -----------  
一、 Django介绍  
 1. Django特点:  
 *# (1) 写得少，做得多  
 # (2) 做web项目  
 # (3)重量级框架，功能齐全 (json轻量级，使用Flask框架)  
 # (4补充)扩展性强，可开发自己的包，通过中间量与Django交互* 2. MVC设计模式(设计鼻祖) 目的: 为了解耦，要封装:高内聚(模块间有关系，功能要实现)、低耦合(模块和模块之间可以拆分)  
 *# Django: MVT设计模式，基于MVC  
 # View: 接收请求、处理请求、业务逻辑、处理数据  
 # Model: 数据模型、交互数据库  
 # Template: 模板、渲染、展示数据  
 #模式：* 客户端(浏览器)--(request)-->V--(data)-->M--(data)-->  
 V--(data)-->T--(html/css/js)-->V--(response)-->客户端(浏览器)  
  
二、工程搭建  
 1. django虚拟环境创建  
 *# 1-1. cd ~  
 # 1-2. workon 查看所有的虚拟环境  
 # 1-3. 创建虚拟环境：mkvirtualenv django\_py3\_1.11 -p python3 快捷键：mkv+tab键  
 # 删除虚拟环境：rmvirtualenv django\_py3\_1.11 快捷键：rmv+tab键  
 # 1-4. 安装软件 (django\_py3\_1.11 ) pip install django==1.11.11  
 # 需要安装的文件：Django==1.11.11 django-redis Jinja2 Pillow PyMySQL  
 # 1-5. 查看是否安装成功：pip list* 2. django基本操作  
 *# 2-1. 在django的程序下进入到桌面： cd ~/Desktop/  
 # 2-2. 查看当前所有虚拟环境：workon  
 # 2-3. 使用虚拟环境：workon zdemo  
 # 2-4. 查看是否有django框架：pip list  
 # 2-5. 创建项目：django-admin startproject zdemo  
 # 2-6. 进入到项目目录下：cd zdemo  
 # 2-7. 启动测试：python manage.py runserver # manage.py是管理文件 浏览器打开：http://127.0.0.1:8000/ 会出现It worked的界面  
 # 2-8. 写自己的功能: python manage.py startapp auser （auser自己起名）  
 # 2-9. pycharm ---导进桌面创建的项目--- 配置环境 django\_py3  
 # 2-10. 注册APP子应用:在apps.py文件内复制AuserConfig  
 # 到settings.py文件的INSTALLED\_APPS括号里下方输入：'auser.apps.AuserConfig’   
 # （找在auser下的apps中的AuserConfig类名称）  
 # 2-11. 写功能：在auser的views.py中，定义视图函数  
 # 导入包* **from** django.http.response **import** HttpResponse  
 *# 定义视图函数* **def** index(request):  
 *# 1.接收请求对象  
  
 # 2.返回响应对象* **return** HttpResponse(**'登录功能！'**)  
 *# 2-12. 让自己的文件找到配置路由(路径)：在zdemo文件的urls.py中，打开import ...  
 # 系统自带的路由，本身就有，不用管  
 # url(r'^admin/', admin.site.urls),  
 # 配置路由，通过路由找功能函数：（复制这个路由，copy到下边，修改后边站点）  
 # url(r'^login/', views.index), # 根据 views 找的 index 函数  
 # 在上边导包：from auser import views* 3. 流程顺序：  
 *# (1) 程序启动自动找到配置文件setting.py根路由的配置项ROOT\_URLCONF = 'zdemo.urls'  
 # (2) 找到zdemo的总路由urls  
 # (3) 通过include找子路由(子应用auser的子路由urls，没有则创建)  
 # (4) 根据子路由urls找到视图函数(views的index函数)* 4. 子路由的查找顺序,严格按照^$写,避免路由屏蔽  
 *# 4-1. 路由执行顺序：列表是从上到下的找的  
 # 4-2. 路由屏蔽:顺序问题、找到之后不往后找，格式不严谨*三、配置、静态文件与路由  
 1. 配置文件介绍：  
 *# 项目路径 BASE\_DIR # 配置、拼接路径  
 # 加密的key 加盐 加签 SECRET\_KEY  
 # 调试模式 DEBUG = True  
 # 允许的域名 ALLOWED\_HOSTS = [] #允许任何域名 ['']  
 # 注册子应用 Application definition INSTALLED\_APPS  
 # 中间件 MIDDLEWARE  
 # 根路由的配置项 ROOT\_URLCONF  
 # 模板配置 TEMPLATES  
 # WSGI协议 WSGI\_APPLICATION  
 # 数据库配置 DATABASES # 我们是用mysql，以后此处改为mysql  
 # 权限操作 AUTH\_PASSWORD\_VALIDATORS  
 # 国际化 Internationalization # 中文翻译 本地化  
 # 修改为中文：LANGUAGE\_CODE = 'zh-hans' TIME\_ZONE = 'Asia/shanghai' 下边三项可网上查修改，不改也可以，测试时把所有主路由注释  
 # 静态文件 Static files (CSS, JavaScript, Images) STATIC\_URL* 2. 静态文件  
 *# 2-1. 静态文件配置路由(静态文件放到服务器上有路由)  
 # 在项目下创建 directory文件夹static\_file(文件夹名随意)，并给此文件夹存数据，比如图片 tupian.jpg  
 # 2-2. 静态文件夹 配置django知道的路径  
 # STATICFILES\_DIRS=[os.path.join(BASE\_DIR, 'static\_file')]  
 # 2-3. 打开浏览器，输入 http://127.0.0.1:8000/static/tupian.jpg 验证打开图片即可* 3. 路由反向解析、命名空间、名字、斜杠说明  
 *# 3-1. 路由的正向解析: setting--root\_urlConfig--zdemo.rul.py--配置总路由auser-->子路由login-->views.index--  
 # 3-2. 路由的反向：index(视图函数)-->取出路由"auser/login"  
 # 3-3. 取出方法: reverse() 反向、颠倒  
 # 种类一：总路由有名字 namespace  
 # (1) 总路由起名字 namespace (命名空间)  
 # 在总路由auser.urls的include内部输 namespace='zongluyou' (一定要在内部输入，名字随便写，因为不管输什么，返回的都是auser/login)  
 # (2) 子路由起名字 name 名字  
 # 在子路由配置内输 name='zongluyou\_fenluyou' (也是要在内部写，名字随意)  
 # (3) 反向解析出总路由 resver('总路由名：子路由名')  
 # 在子应用的views的index视图函数内输 print(resver('zongluyou:zongluyou\_ziluyou'))  
 # 种类二：总路由没有名字：reverse('name')  
 # (1) 略过总路由有名字的第(1)步  
 # (2) 子路由起名字(子路由必须要起名字) name  
 # 在子路由配置内输 name='zongluyou\_fenluyou' (也是要在内部写，名字随意)  
 # (3) 反向解析出总路由 resver('子路由名')  
 # 在子应用的views的index视图函数内输 print(resver('zongluyou\_ziluyou'))* 路由反向解析的使用场景: 重定向 redirect (原来要去A网站, 后来重新定向到B网站)  
 重定向 优势：1. 避免手动写错； 2. 如果需求改变，重定向的代码不需要改动  
  
四、请求与响应  
 1.请求 Request  
 1-1. 解析参数 4种  
 (1) 拼接在路径里边 https://fanyi.baidu.com/*#zh/en/%E4%BA%BA --解决方式--> 正则表达式: 按照以前匹配* 匹配流程: 查询年份2019和城市beijing  
 ---配置子路由，使用正则表达式把数据拼接到路由内：url(**r'^joinparams/(\d{4})/([a-z]+)/$'**, views.joinParams),   
 *#年份四个数字，\d表示数字; 城市非空英文字母，加号表示非空任意字符* ---在子应用views文件内创建joinParams视图函数，除了接收请求对象(request参数)外，再加year和city两个参数，然后输出，函数为：  
 **def** joinParams(request,city,year):  
 print(**'年份：'**, year)  
 print(**'城市：'**, city)  
 **return** HttpResponse(**'拼接在路径里边'**)  
 情况: 如果该函数为位置传参，传参时两参数位置颠倒，结果年份和城市输出会颠倒顺序  
 优化: 通过给分组起别名解决: (?P<name>) 正则表达式匹配分组的分组起别名  
 方法: url(**r'^joinparams/(?P<year>\d{4})/(?P<city>[a-z]+)/$'**, views.joinParams),  
 结果: 将位置传参更改为以名字传参的方式传参，不需要考虑传递的顺序，解决了位置传参可能出现的颠倒错误。  
  
 (2) 查询参数 (跟在路径后边，参数前边加问号，不属于路由一部分) ？q=iphone --解决方式--> request.GET的属性:返回queryDict get() getlist()  
 *# 加问号则表示跟路由没关系，地址栏可以看到数据的永远都是GET方式* 匹配流程: ?a=10&b=20&a=30  
 ---配置子路由 url(**r'^check\_data/$'**, views.check\_data),  
 ---在子应用的views文件内定义视图函数，  
 **def** check\_data(request):  
 *# request.GET 属性 QueryDict 一键多值* check = request.GET *# 查看一下类型，是GET类型* print(check)  
 print(check.get(**'a'**)) *# 由于a里有两个值，因此只有get时只取最后一个值，即30* print(check.get(**'b'**))  
 print(check.getlist(**'a'**)) *#使用getlist可以取出所有的值* **return** HttpResponse(**'2. 查询参数'**)  
 ---如果使用POST, 需要借助第三方工具Poatman，选择POST方式，但要注释掉主路由settings.py文件MIDDLEWARE内的csrf  
  
 (3) 请求体body参数: POST(上传, 做增加) PUT(修改) DELETE(删除)   
 (3-1). form表单参数 --解决方式--> request.POST的属性: 返回queryDict get() getlist()  
 匹配流程: fromdata  
 --- 配置子路由 url(**r'^bodyformdata/$'**, views.bodyFormData),  
 --- 在子应用的views文件内定义视图函数  
 **def** bodyFormData(request):  
 *# 知道post类型，然后解析* form\_data = request.POST  
 print(form\_data)  
 print(form\_data.get(**'bodyformA'**))  
 print(form\_data.get(**'bodyformB'**))  
 print(form\_data.getlist(**'bodyformA'**))  
 **return** HttpResponse(**'请求体form表单'**)  
 ---使用Poatman选择POST，在Body中输入key和value值 bodyformA 1 bodyformB 20 bodyformA 666  
 点击send后即可在终端查看  
  
 (3-2). 非form表单参数 json xml javascript --解决方式--> request.body 属性: 返回二进制bytes数据，bytes.decode()-->str转成字符串  
 *# json可以转字典，xml不可转成字典* 匹配流程: **"a"**:1,**"b"**:2,**"c"**:20  
 --- 配置子路由 url(**r'^bodynotformdata/$'**, views.bodyNotFormData),  
 --- 在子应用的views文件内定义视图函数  
 **def** bodyNotFormData(request):  
 not\_form\_data = request.body  
 print(not\_form\_data) *# 类型是b 二进制* srt\_data = not\_form\_data.decode() *# 将二进制数据解码decode成字符串* dict\_data = json.loads(srt\_data) *# 由于在Postman内输入的是字典类型，因此将json字符串数据转成字典* print(type(dict\_data)) *# 类型改为字典dict类型* print(dict\_data[**'c'**]) *# 输出 20* **return** HttpResponse(**'请求体非form表单参数'**)  
 --- 使用Poatman选择Body，选择json，然后再raw内输json数据：  
 {  
 **"a"**:1,  
 **"b"**:2,  
 **"c"**:20  
 }   
  
 1-2. 获取请求头的内容  
 *# request.META: 返回字典  
 # request.META['CONTENT\_TYPE']* --方法: 配置子路由-->定义获取请求头的视图函数-->headers = request.META print(headers) 输出key的话:print(headers[**'HTTP\_HOST'**]) 输出127.0.0.1:8000  
  
 1-3. 其他的属性: PUT(修改)  
 *# request.method 请求方式: PUT  
 # request.user 用户（认证权限） 默认 匿名用户:AnonymousUser  
 # request.path 路径: /wang/bodynotformdata/ # 只能拿到自己的路径，不能做重定向  
 # request.encoding 编码格式: None 默认 utf-8* 2. 响应对象 Response  
 2-1. 响应对象的参数 3个: content content\_type status  
 -- 配置子路由和定义函数  
 content = **"操作响应对象"** content\_type = **""** status = 200  
 **return** HttpResponse(content=content,content\_type=content\_type,status=status)  
  
 2-2. 响应对象的属性 3个: content content\_type status\_code  
 -- 配置子路由和定义函数  
 response = HttpResponse()  
 response.content = **"响应对象的属性"** responses.status\_code = 200  
 **return** response  
  
  
 2-3. 子类 JsonResponse 1. 自动转换字典或者list--> 字符串 2. content\_type=**'application/json'** -- 配置子路由和定义函数  
 -- 字典 --  
 第一种: 需要转成字符串，三引号转成字符串，如果不加三引号的话会输出 abc  
 dict\_data1 = **"""{  
 'a':1,  
 'b':2,  
 'c':3  
 }"""  
 return** HttpResponse(dict\_data1)  
 第二种: JsonResponse自动把字典转成json字符串，把content\_type 设置成 application/json  
 导入包: 子类包 **from** django.http.response **import** HttpResponse,JsonResponse  
 dict\_data2 = {  
 **'a'**:1,  
 **'b'**:2,  
 **'c'**:3  
 }  
 **return** JsonResponse(dict\_data2)  
 *# 如果想传入列表，则在return后的括号内加入: safe=False* 3. Cookie 轻量级 Cookie就是个标签，记住用户名 *# 爬虫: 拿着cookie直接进入淘宝爬数据  
 # HTTP 无状态协议:第一次请求完，第二次请求不知道了  
 # 特点: 1. key:value 形式存储  
 # 2. 基于域名安全，不同域名的Cookie是不能互相访问的 (同源策略: 同源: IP:port 都一致)  
 # 作用: 1. 保持会话状态  
 # 2. 做缓存  
 # 数据存储位置: 浏览器，只能存储小型数据 # 浏览器的小饼干图标  
 # 操作流程: 浏览器--(用户第一次登录)-->服务器[cookie]--(向用户发送Cookie[标签]进行设置)-->浏览器(存储该标记)--(用户再次访  
 # 问时携带Cookie)-->服务器(读取客户端的Cookie进行验证)  
 # 弊端: 1. 不建议存储敏感信息，不安全  
 # 2. 长度有限制 4KB (大约几千个文字)  
 # 操作流程: (1) 切换到桌面进入workon工程，创建项目: django-admin startproject zcookie (项目名自定)  
 # (2) 创建并启动子应用: python manage.py startapp acookie (子应用名自定)  
 # (3) 启动测试: python manag.py runserver  
 # (4) 配置setting文件 --> 配置总路由[url(r'^')表示配置所有路由，子路由名必须写] --> 配置子路由 --> 定义视图函数* **from** django.http.response **import** HttpResponse  
 **def** index(request):  
 *# (后台给浏览器)设置cookie* response = HttpResponse(**'设置cookie'**)  
 *# max\_age 设置过期时间 以秒为单位，可不写，默认None即浏览器关闭则失效* response.set\_cookie(**'name'**, **'wang'**, max\_age=30)  
 *# (浏览器)读取/获取cookie* print(request.COOKIES[**'name'**]) *# 返回wang 如果没有['name']则返回字典{'name': 'wang'}  
 # 返回响应对象* **return** response  
  
 4. Session 重量级 (cookie与session是一对双胞胎)  
 *# 作用: 1. 缓存 2. 保持会话状态  
 # 操作流程: 浏览器--(用户第一次登录)-->服务器[session]--给客户端发送cookie进行设置sessionid-->浏览器(只有id没有值,值在服务器)--用户再次访  
 # 问携带sessionid-->服务器(读取客户端的cookie和后台的session验证)  
 # 优势: 可以存储敏感信息，安全可靠  
 # 弊端: 1. 必须依赖于cookie机制 2. session共享(多台服务器的问题，一台则不需要考虑)* 4-1. 存储和配置  
 1. 存储  
 *# 1. 默认存在服务器上的数据库: db  
 # 2. cache缓存  
 # 3. 混合使用 cache\_db  
 # --> 使用Redis作为缓存存储，以上3种存储配置成Redis # 数据库: mysql redis* 2. 配置: django自带session的应用和中间件(在settings.py文件内)  
 *# 1. pip install django-redis  
 # 2. 粘贴 课件里边的配置  
 # CACHES = {  
 # "default": {  
 # "BACKEND": "django\_redis.cache.RedisCache", # 后台 (django\_redis做完: 1.导redis 2.使用redis.StrictReids链接数据库 3.set get)  
 # "LOCATION": "redis://127.0.0.1:6379/1", # 本地 链接redis数据库的地址和库名，redis数据库有16个，默认是0号数据库，此处为1号  
 # "OPTIONS": {  
 # "CLIENT\_CLASS": "django\_redis.client.DefaultClient",  
 # }  
 # }  
 # }  
 # SESSION\_ENGINE = "django.contrib.sessions.backends.cache" # session的引擎  
 # SESSION\_CACHE\_ALIAS = "default" # session的缓存 default可以改，跟第二行一致* 4-2. session的操作  
   
 1. 设置 session  
 *# 以键值对的格式写session: request.session['键']=值  
 # 设置session的有效期: request.session.set\_expiry(value) 默认过期时间是两个星期，0的话关闭浏览器则过期  
 # 清除键读取值: request.session.clear()  
 # 清除session数据，在存储中删除session的整条数据: request.flush() # flush把django设置的sessionid删除  
 # 删除session中指定键及值，在存储中只删除某个键及对应的值: del request.session['键']* 2. 获取 session (string类型)  
 *# 根据键读取值: request.session.get('键',默认值)  
 # 第二次请求django自动操作两件事: 浏览器会自动携带sessionid到服务器; 后台(服务器)拿到sessionid到redis数据库取值* 总结: session 和 cookie对比:  
 *# 相同: 作用相同，都是保持会话状态，做缓存  
 # 区别:   
 # cookie 存储位置: 浏览器 (自动) 不安全，长度有限制  
 # session 存储位置: 服务器 安全，长度不限制*五、 类视图与中间件  
 1. 类视图  
 1-1. 类视图的优势  
 *# 1. 阅读性高 -- 自动判断  
 # 2. 代码复用度高 -- 类继承(子类继承父类super())* 1-2. 类视图的定义  
 *# 1. 创建 类 class classview(View)  
 # 2. 让Django路由 --> 类视图 --> as\_view() -->函数(注意点: 函数名字必须是请求方式的小写)* 子路由文件cla/urls.py:  
 **from** django.conf.urls **import** url, include  
 **from** . **import** views  
 urlpatterns = [  
 url(**r'^classview/$'**, views.classview.as\_view()),  
 ]  
 views.py文件:  
 **from** django.views **import** View  
 **class** classview(View):  
 **def** get(self, request):  
 **return** HttpResponse(**'classview---GET请求方式---登录页 input'**)  
 **def** post(self, request):  
 **return** HttpResponse(**'classview---POST---登录功能'**)  
  
 1-3. [理解]类视图 as\_View()原理  
 *# as\_view() --> 将类视图里面的函数 判断之后 返回来给路由用  
 # 1. 对象 login = LoginView()  
 # 2. if request.method else 10种左右 ==> getattr(login, 根据字符串名字返回对应的函数)  
 # response <== 函数(request,)* 1-4. 添加装饰器给类视图  
 装饰器的本质: 函数嵌套函数  
  
 *# 1. urls.py  
 # 1. 定义一个装饰器 my\_decorator  
 # 2. urls.py内配置子路由: url(r'^classview\_decorator/$', my\_decorator(views.classview.as\_view())),  
   
 # 2. @method\_decorator 给类视图直接添加装饰器 (依赖django)  
 # 1. 导包 from django.utils.decorators import method\_decorator  
 # 2.方式一: 把装饰器添加到类视图的头上，相当于装饰器作用于整个类视图:* @method\_decorator(my\_decorator, name=**'dispatch'**)  
 **class** classview(View):  
 *# 方式二: 把装饰器添加到函数头上，相当于装饰器只作用于下边的函数:* @method\_decorator(my\_decorator) *# name不用定义，默认是 get，如果函数是 post 则 name 默认是 post* **def** get(self, request):  
   
 *# 3. 扩展类Mixin: == 装饰器  
 # 具有添加装饰器功能的扩展类  
 # class abc(object):  
 # 写扩展的功能  
 # return 结果  
 # 代码:* **class** DecoratorMixin(object):  
 @classmethod  
 **def** as\_view(cls, \*\*kwargs):  
 *# 添加装饰器* view = super().as\_view(\*\*kwargs)  
 view = my\_decorator(view)  
 **return** view  
 **class** classview(DecoratorMixin, View): *# DecoratorMixin 一定要写到 View 前边，否则页面可以加载，但添加不上装饰器* ……  
  
 *# 类--类继承(多继承)  
 # 子类继承两个父类，两个父类都继承object，子类调用第一个父类的as\_view方法，直到object都没有，这时会到第二个父类找，第二个有则调用* 2. 中间件 middleware --- 装饰器函数  
 *# 1. 中间件的作用: middleware可以给所有 API 添加功能(校验IP, 校验登录)  
 # 2. 中间件的操作:   
 # 1. middleware.py  
 # 2. 写装饰器函数(get\_response)  
 # 3. settings # 注册中间件 'middleware.ipMiddleware'  
 # 3. 多个中间件的执行顺序:  
 # 在请求视图被处理前，中间件由上至下依次执行  
 # 在请求视图被处理后，中间件由下至上依次执行*六、模板  
 1. Django自带模板 template  
 1-1. 模板配置  
 *# 1. 项目目录--zdemo-->创建一个template文件夹  
 # 2. settings.py-->TEMPLATE-->配置文件夹的路径:  
 #'DIRS': [os.path.join(BASE\_DIR,'template')],* 1-2. 模板基本语法  
 *# 1. 在template下创建 index.html 文件  
 # 2. 在该文件内定义模板变量{{ 变量 }} 变量名必须由字母、数字、下划线（不能以下划线开头）和点组成。  
 # 在Django模板内的注释 {# 我是Django内HTML文件注释 #}* <h1>{{ content }}</h1>  
 *# 3. 去类视图 渲染模板变量(让数据传给模板，进行渲染) render: 渲染* **from** django.shortcuts **import** render  
 **from** django.views **import** View  
 **class** templateView(View):  
 **def** get(self, request):  
 *# 让 数据 传给 模板 进行渲染* data = {  
 **'content'**: **"让 数据 传给 模板 进行渲染"** }  
 **return** render(request, **'index.html'**, context=data)  
 *# 4. 模板语句  
 # 1. for {% for item in 列表 %} <h5> {{item}}</h5> {% endfor %}  
 # 2. if elif {% if ... %}  
 # 3. 比较运算符 == ！= > < >= <=  
 # 4. 布尔运算符 and or not  
 # 注意点: 符号左右必须有空格  
 # 解析多层使用. 不能使用中括号，比如模板变量data* data={  
 **"list"**:[1,2,4,**'A'**,**'B'**],  
 **"dict"**:{**"name"**:**"王二小"**,**"age"**:**"25"**}  
 }  
 *# 在index.html内渲染(取列表第二个值，字典的name):* <h3>{{ list.1 }}</h3> <h4>{{ dict.name }}</h4> 浏览器显示: 2 王二小  
 （1）使用for循环渲染  
 {% **for** i **in** list %}   
 <h5> {{item}}</h5>  
 {% endfor %}  
 如果取下标，中间行改成:<h5> {{forloop.counter}}---{{item}}</h5> *# counter默认从1开始  
 # <h5> {{forloop.counter0}}---{{item}}</h5> 则从0开始  
 # <h5> {{forloop.revcounter}}---{{item}}</h5> 则下标倒置，最后一个是1，元素位置不变  
 # <h5> {{forloop.revcounter0}}---{{item}}</h5> 则下标倒置，最后一个是0，元素位置不变* 判空: 如果没有内容，会显示空白页面，此时可以给用户友情提示  
 {% **for** i **in** list1 %} *# list1不存在* <h5> {{empty}}</h5>  
 <h1>友情提示:没有数据</h1>  
 {% endfor %}  
 （2）使用if条件判断 跟for循环一样使用，可以加入运算符(比较运算符和逻辑运算符)  
  
 1-3. 过滤器 本质: 函数  
 *# 1. Django自带过滤器 {{变量|过滤器:参数}}  
 # safe {{ safe\_date|safe }} "safe\_date":"<a href='#'>safe标签</a>" # a标签内的 # 表示自链接，即更新本页面  
 # length {{ list|length }} 查list的长度  
 # default {{ age|default:"年龄不详" }} 查年龄，默认值是 年龄不详 dict里的年龄是二层嵌套，不能被查询出来，只能一层可以  
 # date {{ data\_time|date:"Y-m-d" }} 或 {{ time|date:"Y年m月d日" }}  
 # from datetime import datetime  
 # time = datetime.now()  
 # "date\_time":time  
 # 2. Django自定义过滤器(7步)  
 # 1. 在自己的app下边创建templatetags的package包，名字不能更改  
 # 2. 在新创建的templatetags包下创建chengfa的py文件，文件名随便起 xx.py  
 # 3. 导包 from django import template  
 # 4. 实例化对象 register不能变 register = template.Library()  
 # 5. 给函数添加过滤器 (本质是 函数) 函数随便写 该函数是让传入的数据乘以100，输入为 3000  
 # @register.filter  
 # def fang(x):  
 # return x \* 100  
 # 6. 在index.html里导入自定义的过滤器文件(直接把文件导入方便调用本文件内其他的过滤器) {% load chengfa %}  
 # 7. 使用django自定义的模板 {{ dict.age | fang }} 即 {{ 变量 | 函数 }}  
 # 如果运行错误，则重新启动 python manage.py runserver，如果还错则检查是否注册app* 1-4. 模板继承  
 *# 作用: 让子模板复用  
 # 1. 在template文件夹内创建afather.html和bson.html文件  
 # 2. 在afather.html文件内写标签  
 # 3. 在bson.html文件内继承 {% extends "父html文件路径" %} 即 {% extends "afather.html" %}  
 # 4. views文件修改返回数据 return render(request, "bson.html")  
 # 5. 在afather.html文件写 {% block 名字 %} 父模板内容 {% endblock 名字 %}* {% block fatherTemplate %}  
 <h2>父模板内容</h2>  
 {% endblock fatherTemplate %}  
 *# 6. bson.html 重写 {% block 名字 %} 子模板内容 {% endblock 名字 %}  
 # 7. 继承 + 重写 在block和endblock内加入 {{ block.super }}* 2. Jinja2模板  
 优势: 跑得快、功能多  
 2-1. 安装包 pip install jinja2  
 2-2. 配置 settings.py --> TEMPLATE 改  
 *# 1.'BACKEND': 'django.template.backends.django.DjangoTemplates'  
 #1. django.DjangoTemplates 改成 jinja2.Jinja2  
 #2. 在OPTIONS里加入 'environment': 'jinja2\_env.environment',* 2-3. 创建 jinja\_env.py --> 创建jinja2环境  
 *# 复制讲义的代码* 2-4. for循环与Django有差异  
 *# 先定义函数:* **from** django.shortcuts **import** render  
 **def** index(request):  
 context = {  
 **"title"**: **"jinja2的模板变量"**,  
 **"jinja\_list"**: [**'A'**, **'B'**, **'C'**, **'D'**, **'E'**]  
 }  
 **return** render(request, **"jinja2\_index.html"**, context)  
 *# 获取下标 loop.index 从1开始 加0从0开始  
 # 倒置下标 loop.revindex  
 # 第一个是True，其他都是False loop.first  
 # 最后一个是True，其他都是False loop.last  
 # 项目数(长度) loop.length  
 # 间期取值 loop.cycle   
 # 比如{{loop.cycle('a','b')}} 结果为 a---A b---B a---C b---D a---E* 2-5. jinja2自定义过滤器  
 *# 1. 自定义过滤器 jinja2\_env.py 自定义函数* **def** jinja2\_filter(a):  
 **if** a == **"B"**:  
 **return "过滤器过滤到了结果：B"  
 else**:  
 **return** a  
 *# 2. 把过滤器添加到环境中，在environment函数内输入 jinja2\_en.py 注册过滤器 env.filters['过滤器名字']=函数名字* env.filters[**"jinja2\_filter"**] = jinja2\_filter  
 *# 3. 在jinja2\_index.html内使用该过滤器* {{ i | jinja2\_filter }} 结果为 A 过滤器过滤到了结果：B C D E  
  
 3. [理解]CSRF跨域攻击  
 *# 1. 攻击流程  
 # 用户老王访问安全网站A --> 网站A给老王的浏览器cookie --> 老王在cookie没有过期情况下访问骗子网站B --> B隐  
 # 藏发送访问A网站的请求，把该请求隐藏到"获取优惠券"之类的选项中 --> 老王点击 --> B间接通过老王访问到了A  
 # 2. 攻击防护  
 # 用户老王访问安全网站A --> 网站A给老王的浏览器cookie和csrf\_token(csrf\_token放到了页面内而不是cookie内)  
 # --> 老王在cookie没有过期情况下访问骗子网站B --> B隐藏发送访问A网站的ajax请求，把该请求隐藏到"获取优惠  
 # 券"之类的选项中 --> 老王点击 --> B间接通过老王访问A时只能拿到cookie但没有csrf\_token，A网站既要校验cookie  
 # 又要校验csrf\_token（双重校验），因此B网站访问不到A网站。*七、数据库  
 4. 数据库操作  
 1. ORM 增删改查  
 *# 1.增: save() create()  
 # 2.删：delete() 两种方法  
 # 3.改：save(), update()  
 # 4.查： get(1个) filter(querySet) count()* 2. 过滤查询  
 *# filter get exclude()  
 # exact  
 # lt gt gte lte  
 # contains startswith endswith  
 # isnull   
 # in []  
 # year* 3. F 对象 ---- 多属性 对比 F(属性名)  
 **from** django.db.models **import** F,Q  
 bcomment\_\_lt = F(**'bread'**)  
 Q 对象 --- 多条件 & | ~  
 Q()|Q()|Q()  
 4. 聚合查询 aggregate()  
 *# Sum Avg Max Min Count  
 # 排序 order\_by -* 5. 关联查询  
 *# 1. 关联查询  
 # 1:n 没有外键 关联对象的小写\_set realated\_name="abc"  
 # n:1 有外键 , n.外键属性  
  
 # 2. 关联过滤查询  
 # 1:n 关联对象 纯小写\_\_\_属性\_\_修饰=值  
 # n:1 外键属性\_\_属性\_\_修饰=值* 6. QuerySet 特点  
 两大特性:   
 *# 1.惰性执行 懒加载,, 当数据 使用的时候 才交互数据库, 没有 不交互  
 # 2.缓存, 有缓存 不交互数据  
 # 3.q[:2] 没有负数*八、admin站点  
  
 *# 本地化   
 # 1. django manage.py creatsuperuser  
 # 2. 注册 模型 admin.py admin.site.register()  
 # 3. 设置标题  
  
  
 # 1. fields = [] fieldset=()  
 # 2.关联对象, 编辑书 --关联英雄 一次添加  
 # 2.1 class abc() model = HeroInfo  
 # 2.2 inlines = [abc]  
  
  
# 掌握程度  
# 能够使用ORM实现数据新增、修改和删除数据  
# 能够使用ORM实现条件查询  
# 能够使用ORM实现F和Q对象查询  
# 能够使用ORM实现聚合函数查询  
# 能够使用ORM实现排序查询  
# 能够使用ORM实现关联查询  
# 能够知道查询集的特点  
# 能够使用命令创建管理员用户  
# 能够使用Admin站点管理数据  
# 能够使用Admin站点上传图片*