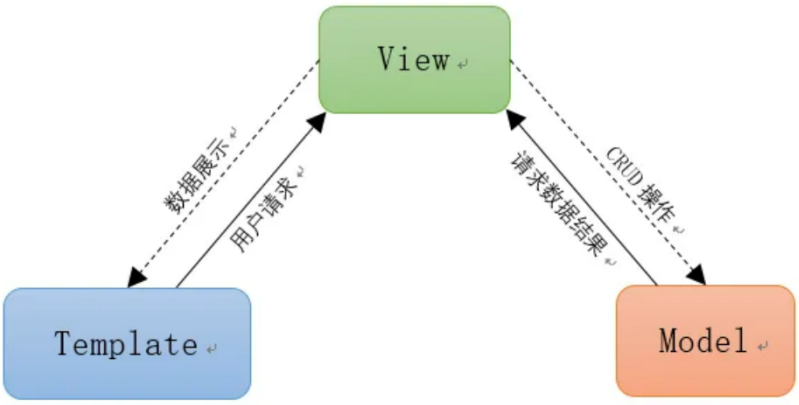
一、Django快速入门

Django来源

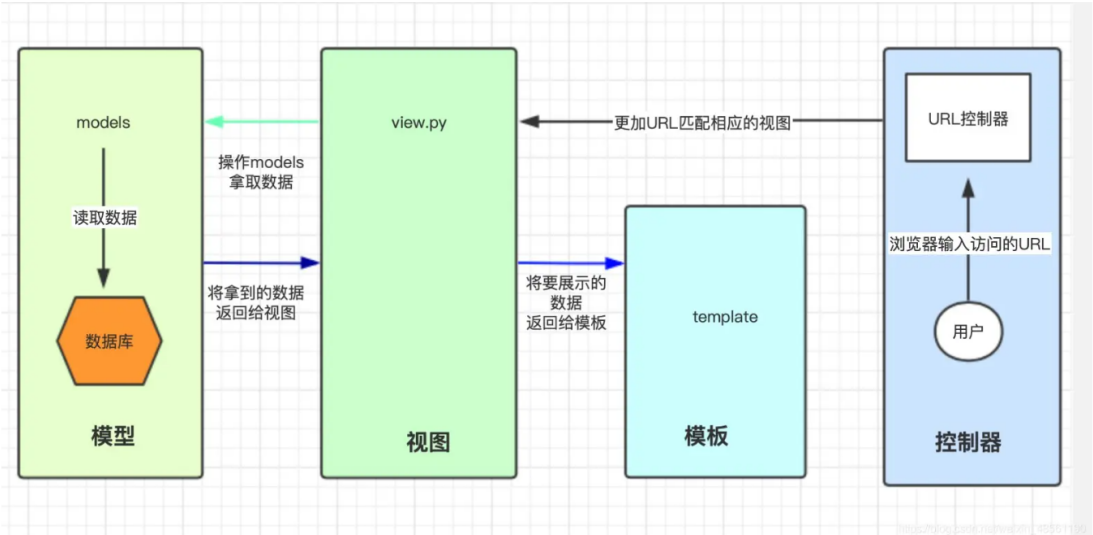
Django是劳伦斯出版集团的开发人员为开发新闻内容网站而设计出来的一个软件，它遵循MVC思想，但是有自己的一个名词，叫做**MVT**。Django遵循快速开发和DRY原则。Do not repeat yourself.不要自己去重复一些工作。

MVT介绍



- **Model，模型和数据库进行交互。**
- **V: View, 视图接收请求，进行处理，与M和T进行交互，返回应答。**
- **T: Template,模板，产生html页面。**

工作流程图



1. 项目创建

1. 创建Django项目

命令：`django-admin startproject 项目名`

注意：创建应用必须先进入虚拟环境。

项目目录如下：

```
manage.py
    mysite/
        __init__.py
        settings.py
        urls.py
        asgi.py
        wsgi.py
```

- `manage.py`一个让你用各种方式管理 Django 项目的命令行工具。
- `_init_.py`一个空文件，告诉 Python 这个目录应该被认为是一个 Python 包
- `settings.py`是项目的整体配置文件。
- `urls.py`是项目的URL配置文件。
- `wsgi.py`：作为你的项目的运行在 WSGI 兼容的Web服务器上的入口。
- `asgi.py`：作为你的项目的运行在 ASGI 兼容的 Web 服务器上的入口

2 创建Django应用

一个项目由很多个应用组成的，每一个应用完成一个功能模块。

创建应用的命令如下:`python mange.py startapp 应用名称`

- `_init.py_`是一个空文件，表示当前目录news可以当作一个python包使用。
- `tests.py`文件用于开发测试用例，在实际开发中会有专门的测试人员，这个事情不需要我们来做。
- `models.py`文件跟数据库操作相关。
- `views.py`文件跟接收浏览器请求，进行处理，返回页面相关。
- `admin.py`文件跟网站的后台管理相关。
- `migrations`文件夹之后给大家介绍。

3. 应用注册

应用创建成功后，需要注册才可以使用，也就是建立应用和项目之间的关联，在项目`/settings.py`中INSTALLED_APPS下添加应用的名称就可以完成注册。

初始项目的INSTALLED_APPS如下图：

```
INSTALLED_APPS = (
    'django.contrib.admin',
    'django.contrib.auth',
    'django.contrib.contenttypes',
    'django.contrib.sessions',
    'django.contrib.messages',
    'django.contrib.staticfiles',
    'news',
)
```

4. 启动项目

运行开发web服务器命令：

```
python manage.py runserver
```

2. 模型类

1.模型设计

django中内嵌了ORM框架，不需要直接面向数据库编程，而是定义模型类，通过模型类和对象完成数据表的增删改查操作。

- **ORM框架**

> O是object，也就**类对象**的意思，
>
> R是relation，翻译成中文是关系，也就是关系数据库中**数据表**的意思，
>
> M是mapping，是**映射**的意思。

下面我们以保存图书信息为例来给大家介绍Django中进行数据库开发的整个流程。

2.定义模型类

模型类定义在models.py文件中，继承自models.Model类。

> 说明：不需要定义主键列，在生成时会自动添加，并且值为自动增长。

- **设计新闻类**

> - 标题：title
> - 新闻内容：content
> - 发布日期：b_date

- **模型类的设计**

> 根据设计，在models.py中定义模型类如下：
>
> ```
> from django.db import models
> class NewsInfo(models.Model):
>   title = models.CharField(max_length=20)
>     content = models.TextField()
>     b_date = models.DateField()
> ```

3.激活模型

- 1.生成迁移文件：根据模型类生成创建表的迁移文件。

```
python manage.py makemigrations
```

- 2.执行迁移：根据第一步生成的迁移文件在数据库中创建表。

```
python manage.py migrate
```

Django默认采用sqlite3数据库，上图中的db.sqlite3就是Django框架帮我们自动生成的数据库文件。sqlite3是一个很小的数据库，通常用在手机中，它跟mysql一样，我们也可以通过sql语句来操作它。

生成的数据表的默认名称为：`应用名字_类名`

```
例：
news_newsinfo
```

4.数据操作

完成数据表的迁移之后，下面就可以通过进入项目的shell，进行简单的API操作。如果需要退出项目，可以使用ctrl+d快捷键或输入quit()。

进入项目shell的命令：`python manage.py shell`

首先引入news.models 中的类：

```
from news.models import NewsInfo
```

查询所有体育新闻信息：

```
>>> NewsInfo.objects.all()
```

因为当前并没有数据，所以返回空列表

```
>>> NewsInfo.objects.all()
[]
```

新建新闻对象：

```
>>> b = newsInfo()
>>> b.title = '斯坦科维奇杯：中国男篮红队胜突尼斯队'
>>> b.content = '8月14日，在广东深圳举行的斯坦科维奇杯洲际篮球赛的首日比赛中，中国男篮红队以77比72战胜突尼斯队'
b.b_date=date(2018,8,15)
b.save()
```

再次查询所有信息：

```
>>> NewsInfo.objects.all()
```

查找该条新闻信息并查内容：

```
>>> b=NewsInfo.objects.get(id=1)
>>> b
<NewsInfo: NewsInfo object>
>>> b.id
1
>>> b.title
'斯坦科维奇杯：中国男篮红队胜突尼斯队'
>>> b.content
'8月14日，在广东深圳举行的年斯坦科维奇杯洲际篮球赛的首日比赛中，中国男篮红队以77比72战胜突尼斯队'
>>> b.b_date
datetime.date(2018, 8, 15)
```

修改该条新闻信息：

```
>>> b.b_date=date(2018,8,1)
>>> b.save()
>>> b.b_date
datetime.date(2018, 8, 1)
```

删除该条新闻信息：

```
>>> b.delete()
>>> newsInfo.objects.all()
[]
```

3. 后台管理

使用Django的管理模块，需要按照如下步骤操作：

- 1.管理界面本地化
- 2.创建管理员
- 3.注册模型类
- 4.自定义管理页面

1.管理界面本地化

本地化是将显示的语言、时间等使用本地的习惯，这里的本地化就是进行中国化，中国大陆地区使用简体中文，时区使用亚洲/上海时区，注意这里不使用北京时区表示。

打开test1/settings.py文件，找到语言编码、时区的设置项，将内容改为如下：

```
LANGUAGE_CODE = 'zh-hans' #使用中国语言
TIME_ZONE = 'Asia/Shanghai' #使用中国上海时间
```

2.创建管理员

创建管理员的命令如下，`python manage.py createsuperuser`

按提示输入用户名、邮箱、密码。

这里我设置的账号是，密码是123qwe,邮箱随便输入一个

```
(dj_py3) python@ubuntu:~/Desktop/djproject3/demo1$ python manage.py
createsuperuser
Username (leave blank to use 'python'): python
Email address: 123@qq.com
Password:
Password (again):
Superuser created successfully.
(dj_py3) python@ubuntu:~/Desktop/djproject3/demo1$
```

接下来启动服务器：`python manage.py runserver 7777`

打开浏览器，在地址栏中输入如下地址后回车。

```
http://192.168.128.132:7777/admin/
```

3.注册模型类

登录后台管理后，默认没有我们创建的应用中定义的模型类，需要在自己应用中的admin.py文件中注册，才可以在后台管理中看到，并进行增删改查操作。

打开news/admin.py文件，编写如下代码：

```
from django.contrib import admin
from .models import newsInfo
admin.site.register(newsInfo)
```

4.自定义管理页面

在列表页只显示出了NewsInfo object，对象的其它属性并没有列出来，查看非常不方便。 Django提供了自定义管理页面的功能，比如列表页要显示哪些值。

```
from django.contrib import admin
from .models import newsInfo

class newsInfoAdmin(admin.ModelAdmin):
    # list_display表示要显示哪些属性
    list_display = ['id','title','b_date']

admin.site.register(newsInfo,newsInfoAdmin)
```

4. 视图

后台管理页面做好了，接下来就要做公共访问的页面了。当我们刚刚在浏览器中输入 http://192.168.128.132:7777/admin之后，浏览器显示出了后台管理的登录页面，那有没有同学想过这个服务器是怎么给我们找到这个页面并返回呢？/admin/是我们想要请求的页面，服务器在收到这个请求之后，就一定对应着一个处理动作，这个处理动作就是帮我们产生页面内容并返回回来，这个过程是由**视图**来做的。

对于django的设计框架MVT，用户在URL中请求的是视图，视图接收请求后进行处理，并将处理的结果返回给请求者。

使用视图时需要进行两步操作：

```
* 1.定义视图函数
* 2.配置URLconf
```

1.定义视图

视图就是一个Python函数，被定义在views.py中。

视图的必须有一个参数，一般叫request，视图必须返回HttpResponse对象，HttpResponse中的参数内容会显示在浏览器的页面上。

打开news/views.py文件，定义视图index如下

```
from django.http import HttpResponse

def index(request):
    return HttpResponse('这个是index页面')
```

2.配置URL

查找视图的过程

请求者在浏览器地址栏中输入url，请求到网站后，获取url信息，然后与编写好的URLconf逐条匹配，如果匹配成功则调用对应的视图函数，如果所有的URLconf都没有匹配成功，则返回404错误。

一条URLconf包括url规则、视图两部分：

- url规则使用正则表达式定义。
- 视图就是在views.py中定义的视图函数。

需要两步完成URLconf配置：

- 1.在应用中定义URLconf
- 2.包含到项目的URLconf中

在news/应用下创建urls.py文件，定义代码如下：

```
from django.conf.urls import url
from news import views
urlpatterns = [
    url(r'^index', views.index),
]
```

包含到项目中：打开demo1/urls.py文件，为urlpatterns列表增加项如下：

```
url(r'^',include('news.urls')),
```

demo1/urls.py文件完整代码如下：

```
from django.conf.urls import include, path
from django.contrib import admin

urlpatterns = [
    path(r'^admin/', include(admin.site.urls)),
    path(r'^',include('news.urls')),
]
```

3.请求访问

视图和URLconf都定义好了，接下来在浏览器地址栏中输入网址：

```
http://192.168.128.132:8000/
```

网页显示效果如下图，视图被成功执行了。

4.返回模型类定义的数据

在news/views.py的index函数中先把模型类里面的数据查询出来，然后返回到页面上，

修改views.py中的代码，如下

```
from django.shortcuts import render

# Create your views here.
from django.http import HttpResponse
from .models import newsInfo

def index(request):
    s_info = newsInfo.objects.all()
    content = [[item.title,item.content] for item in s_info]
    return HttpResponse(content)
```

5. 模板

- **如何向请求者返回一个漂亮的页面呢?**

肯定需要用到html、css，如果想要更炫的效果还要加入js，问题来了，这么一堆字段串全都写到视图中，作为HttpResponse()的参数吗？这样定义就太麻烦了吧，因为定义字符串是不会出任何效果和错误的，如果有一个专门定义前端页面的地方就好了。

解决问题的技术来了：**模板。**

在Django中，将前端的内容定义在模板中，然后再把模板交给视图调用，各种漂亮、炫酷的效果就出现了。

1.配置模板根目录

为应用news下的视图index创建模板index.html，在项目目录下新建一个名为tamplates的文件夹，在该文件夹下面新建一个和应用同名的文件夹，结构目录如下：

设置查找模板的路径：打开`settings.py`文件，设置TEMPLATES的DIRS值

```
'DIRS': [BASE_DIR / 'tamplates']
```

2.定义模板

打开templates/news/index.html文件，定义代码如下：

```
<html>
<head>
    <title>首页</title>
</head>
<body>
<h1>{{title}}</h1>
    {%for i in list%}
    {{i}}<br>
    {%endfor%}
</body>
</html>
```

在模板中输出变量语法如下，变量可能是从视图中传递过来的，也可能是在模板中定义的。

```
{{变量名}}
```

在模板中编写代码段语法如下：

```
{%代码段%}
```

3.视图调用模板

视图调用模板都要执行以上三部分，于是Django提供了一个函数render封装了以上代码。 方法render包含3个参数：

- 第一个参数为request对象
- 第二个参数为模板文件路径
- 第三个参数为字典，表示向模板中传递的上下文数据

打开news/views.py文件，调用render的代码如下：

```
from django.shortcuts import render

def index(request):
    context={'title':'新闻列表','list':range(10)}
    return render(request,'news/index.html',context)
```

二、模型

1. ORM介绍

> **O(objects):类和对象。**
>
> **R(Relation):关系，关系数据库中的表格。**
>
> **M(Mapping):映射。**

Django ORM框架的功能：

> a) 建立模型类和表之间的对应关系，允许我们通过面向对象的方式来操作数据库。
>
> b) 根据设计的模型类生成数据库中的表格。
>
> c) 通过方便的配置就可以进行数据库的切换。

2. 数据库配置

1 .创建数据库:

- django框架,只能生产数据表，不会自动帮我们生成mysql数据库，所以我们需要自己去创建。

```
create database test2 charset=utf8
```

2. Django配置使用mysql数据库

修改settings.py中的DATABASES。

```
DATABASES = {
    'default': {
        # 'ENGINE': 'django.db.backends.sqlite3',
        # 'NAME': os.path.join(BASE_DIR, 'db.sqlite3'),
        'ENGINE': 'django.db.backends.mysql',  # 设置使用mysql数据库
        'NAME':'news',                          # 指定数据库
        'USER':'root',                            # 用户名
        'PASSWORD':'mysql',                    #数据库所在主机的ip
        'PORT':3306                               #端口号
        'HOST':'localhost'
    }
}
```

3.安装依赖

- **django框架使用mysql：需要安装依赖mysqlclient**

  ```
  pip install mysqlclient
  ```

3. 模型类字段

django会为表创建自动增长的主键列，每个模型只能有一个主键列，如果使用选项设置某属性为主键列后django不会再创建自动增长的主键列。

默认创建的主键列属性为id，可以使用pk代替，pk全拼为primary key。

1.字段命名限制

1）不能是python的关键字。

2）不允许使用连续的下划线，django的查询方式有连续下划线的语法。

3）定义字段时需要指定字段类型，通过字段类型的参数指定选项，语法如下：

```
属性名=models.字段类型(选项)
```

2.常用字段类型

使用时需要引入django.db.models包，字段类型如下：

类型	描述
AutoField	自动增长的IntegerField，通常不用指定，不指定时Django会自动创建属性名为id的自动增长属性。
BooleanField	布尔字段，值为True或False。
CharField(max_length=最大长度)	字符串。参数max_length表示最大字符个数。
TextField	大文本字段，一般超过4000个字符时使用。
IntegerField	整数
DecimalField(max_digits=None, decimal_places=None)	十进制浮点数。参数max_digits表示总位。参数decimal_places表示小数位数。
FloatField	浮点数。参数同上
DateField：([auto_now=False, auto_now_add=False])	日期。1)参数auto_now表示每次保存对象时，自动设置该字段为当前时间，用于"最后一次修改"的时间戳，它总是使用当前日期，默认为false。2) 参数auto_now_add表示当对象第一次被创建时自动设置当前时间，用于创建的时间戳，它总是使用当前日期，默认为false。3)参数auto_now_add和auto_now是相互排斥的，组合将会发生错误。
TimeField	时间，参数同DateField。
DateTimeField	日期时间，参数同DateField。
FileField	上传文件字段。
ImageField	继承于FileField，对上传的内容进行校验，确保是有效的图片。

3.字段选项

通过选项实现对字段的约束，选项如下：

选项名	描述
default	默认值。设置默认值。
primary_key	若为True，则该字段会成为模型的主键字段，默认值是False，一般作为AutoField的选项使用。
unique	如果为True, 这个字段在表中必须有唯一值，默认值是False。
db_index	若值为True, 则在表中会为此字段创建索引，默认值是False。
db_column	字段的名称，如果未指定，则使用属性的名称。
null	如果为True，表示允许为空，默认值是False。
blank	如果为True，则该字段允许为空，默认值是False。
verbose_name	字段的一个人类可读名称（admin字段显示的名称）
help_text	额外的"帮助"文本

对比：null是数据库范畴的概念，blank是后台管理页面表单验证范畴的。

经验:

当修改模型类之后，如果添加的选项不影响表的结构，则不需要重新做迁移，商品的选项中default和blank不影响表结构。

参考文档:

```
http://python.usyiyi.cn/translate/django_182/index.html
```

4.关系字段类型

关系型数据库的关系包括三种类型：

- **ForeignKey：一对多，将字段定义在多的一端中。**

  ```
  # 一个多对一的关系。需要两个位置参数：模型相关的类和 on_delete 选项。
  goods = models.ForeignKey('Goods', on_delete=models.CASCADE)
  ```

- **ManyToManyField：多对多，将字段定义在任意一端中。**

  ```
  user = models.ManyToManyField('User',on_delete=models.CASCADE)
  ```

- **OneToOneField：一对一，将字段定义在任意一端中。**

  ```
  user = models.OneToOneField('User', on_delete=models.CASCADE)
  ```

- **自关联关系 使用'self'指定**

  ```
  # 一对多的自关联
  bid = models.ForeignKey('self',null=True,blank=True)
  # 多对多的自关联
  bid= models.ManyToManyField('self',null=True,blank=True)
  ```

> **on_delete的作用：**
>
> 当一个引用的对象被删除时，Django 将模拟 `on_delete` 参数所指定的 SQL 约束的行为
>
> **常用的值：**
>
> - CASCADE：级联删除
>
> Django 模拟了 SQL 约束 ON DELETE CASCADE 的行为，也删除了包含 ForeignKey 的对象
> - PROTECT：防止删除被引用对象

5.元选项

> 在模型类中定义类Meta，用于设置元信息，如使用db_table自定义表的名字。
>
> **数据表的默认名称为：应用名称_模型类名称小写**

```
#定义图书模型类BookInfo
class BookInfo(models.Model):

 #定义元选项
 class Meta:
    db_table='bookinfo' #指定BookInfo生成的数据表名为bookinfo
    verbose_name = '地区表'
```

6.objects属性

> **objects：管理器，是models.Manager类型的对象，用于与数据库进行交互**

4. 查询基础语法

1.数据准备

> 1、在models.py中定义一个新闻和新闻类型的模型类

```
class NewsInfo(models.Model):
 """新闻表"""
 title = models.CharField(max_length=100, help_text='新闻标题',
verbose_name='新闻标题', blank=True, default='')
 content = models.TextField(help_text='内容', verbose_name='内容',
blank=True, default='')
 b_date = models.DateField(help_text='日期', verbose_name='日期')
 read = models.IntegerField(help_text='阅读量', verbose_name='阅读量',
blank=True, default=0)
 good = models.IntegerField(help_text='点赞数量', verbose_name='点赞数量',
blank=True, default=0)

class TypeInfo(models.Model):
 """新闻类型"""
 type = models.CharField(max_length=20, help_text='新闻类型',
verbose_name='新闻类型')
```

> 2、再次生成迁移：`python manage.py makemigrations`
>
> 3、执行迁移：`python manage.py migrate`
>
> 4、登录Django后台，添加一些练习数据

2.查询函数

通过模型类.objects属性可以调用如下函数，实现对模型类对应的数据表的查询。

函数名	功能	返回值	说明
get	返回表中满足条件的一条且只能有一条数据。	返回值是一个模型类对象。	参数中写查询条件。1) 如果查到多条数据，则抛异常MultipleObjectsReturned。2)查询不到数据，则抛异常：DoesNotExist。
all	返回模型类对应表格中的所有数据。	返回值是QuerySet类型	查询集
filter	返回满足条件的数据。	返回值是QuerySet类型	参数写查询条件。
exclude	返回不满足条件的数据。	返回值是QuerySet类型	参数写查询条件。
order_by	对查询结果进行排序。	返回值是QuerySet类型	参数中写根据哪些字段进行排序。

- **Django的交互调试环境**

> 启动项目，进入交互环境
>
> `python manage.py shell`
>
> 导入models模块，

```
from news.models import *
```

> **查询单条数据：get**

```
# 查询id为3的新闻信息
NewsInfo.objects.get(id=3)
```

> **所有数据：all**

```
# 查询所有新闻数据
NewsInfo.objects.all()
```

> **过滤：filter**

```
# 查询新闻阅读量为9999的新闻数据：
NewsInfo.objects.filter(read=9999)
```

不等于：exclude

```
#查询所有id不等于1的新闻数据
NewsInfo.objects.exclude(id=1)
```

排序：roder_by

```
#查询所有新闻数据并且按id进行排序（从小到大）
NewsInfo.objects.all().order_by('id')
```

```
#查询所有新闻数据并且按id进行排序（从大到小）
NewsInfo.objects.all().order_by('-id')
```

3.查询集

`all, filter, exclude, order_by`调用这些函数会产生一个查询集（QuerySet）,查询集可以继续调用上面的所有函数。可以通过exists判断一个查询集中是否有数据。

- **查询集特性**

  - **惰性查询：只有在实际使用查询集中的数据的时候才会发生对数据库的真正查询。**
  - **缓存：当使用的是同一个查询集时，第一次使用的时候会发生实际数据库的查询，然后把结果缓存起来，之后再使用这个查询集时，使用的是缓存中的结果。**
  - **索引取值**

    ```
    TypeInfo.objects.all()[0]
    ```

  - **切片操作**

    ```
    TypeInfo.objects.all()[0:3]
    ```

4.模糊条件查询

```
#条件语法格式
模型类.objects.filter(模型类属性名__条件名=值)
```

包含：contains

```
# 查询名称里包含'娱乐'的新闻类别。
TypeInfo.objects.filter(new_type__contains='娱乐')
```

开头：starswith

```
# 查询以'国'开头的新闻类别
TypeInfo.objects.filter(new_type__startswith='国')
```

结尾：endswith

```
# 查询以'资讯'结尾的新闻类别
TypeInfo.objects.filter(new_type__endswith='资讯')
```

范围查询：in

```
# 查询id为1或3或5的新闻类别
TypeInfo.objects.filter(id__in=[1,3,5])
```

5.比较查询：

- **大于:gt**

```
# Demo：查询id大于3的数据
TypeInfo.objects.filter(id__gt=3)
```

- **小于:lt**

```
# Demo:查询id小于3的数据
TypeInfo.objects.filter(id__lt=3)
```

- **大于等于：gte**

```
#Demo:查询id大于等于5的数据
TypeInfo.objects.filter(id__gte=5)
```

- **小于等于：lte**

```
# Demo:查询id小于等于3的数据
TypeInfo.objects.filter(id__lte=3)
```

6.空查询： isnull

isnull值为True:表示查询为空的数据，值为False查询不为空的数据

```
Demo:查询标题不为空的新闻.
NewsInfo.objects.filter(title__isnull=False)
```

5. F对象和Q对象

1、F对象

- **之前的查询都是对象的属性与常量值比较，两个属性怎么比较呢?**

F对象的作用：用于类属性之间的比较。

```
# 使用之前需要先导入
from django.db.models import F
```

```
# 查询阅读量大于等于评论量的新闻
NewsInfo.objects.filter(read__gte=F('comment'))
```

可以在F对象上使用算数运算。

```
# 查询阅读量大于2倍评论量的图书
NewsInfo.objects.filter(read__gt=F('comment') * 2)
```

2. Q对象

> **作用：用于查询时条件之间的逻辑关系。 and or not，可以对Q对象进行& | ~ 操作。**

```
# 使用之前需要先导入
from django.db.models import Q
```

逻辑与(and)： &

```
# 查询阅读量大于300且评论量大于10的新闻的数据
# 不使用Q对象
NewsInfo.objects.filter(read__gt=300,comment__gt=10)

# 使用Q对象
NewsInfo.objects.filter(Q(read__gt=3)&Q(comment__gt=10))
```

逻辑或(or)：|

```
# 查询id大于3或者阅读量大于30的新闻的信息
NewsInfo.objects.filter(Q(read__gt=3)|Q(comment__gt=10))
```

逻辑非(not)：~

```
# 查询id不等于2的新闻信息
NewsInfo.objects.filter(~Q(id=2))
```

6. 聚合函数

- **作用：对查询结果进行聚合操作(等同数据库的聚合函数)。**

函数	作用
sum	求和
count	计数
avg	平均值
max	最大值
min	最小值

1.直接使用

直接使用聚合函数查询返回值是一个数字

> **计数：count**

```
# 统计id大于3的新闻类别数
TypeInfo.objects.count()
```

> **求和：所有新闻的阅读总数**

```
TypeInfo.objects.all.sum('read')
```

2.使用aggregate

使用aggregate方法来使用聚合聚合函数, 返回值是一个字典

```
# 使用前需先导入聚合类
from django.db.models import Sum,Count,Max,Min,Avg
```

计数：Count

```
#查询所有新闻类别的总数
TypeInfo.objects.all().aggregate(Count('id'))
# 注意点aggregate里面的函数，第一个字母要大写，属性项也要用引号括起来
```

求和：Sum

```
# 查询所有新闻阅读量的总和
NewsInfo.objects.all().aggregate(Sum('read'))
```

参考文档:

```
http://python.usyiyi.cn/translate/django_182/ref/models/querysets.html
```

7. 关联查询

1.数据准备

1、创建应用books

2、`setting.py`中注册应用

3、定义模型类

```
#定义图书模型类Book
class Book(models.Model):
 title = models.CharField(max_length=20,verbose_name='图书名称')
 read = models.IntegerField(default=0,verbose_name='阅读量')
 comment = models.IntegerField(default=0,verbose_name='评论量')

#定义人物表 Person
class Person(models.Model):
 name = models.CharField(max_length=20,verbose_name='人物姓名')
 gender = models.BooleanField(default=True,verbose_name='性别')
 book = models.ForeignKey('BookInfo',verbose_name='所属图书')

# 人物与图书表的关系为一对多
```

4、激活模型

- 生成迁移：`python manage.py makemigrations`
- 执行迁移：`python manage.py migrate`

5、Admin注册

> **6、数据准备**
>
> 登录Admin后台添加一些测试数据

2. 通过对象执行关联查询

> **由一到多的访问语法：**

```
# 一对应的模型类对象.多对应的模型类名小写_set
# 查询编号为1的图书。
book=Book.objects.get(id=1)
# 获得book图书的所有人物。
book.person_set.all()
```

> **由多到一的访问语法：**

```
#多对应的模型类对象.多对应的模型类中的关系类属性名
p = Person.objects.get(id=1)
p.book
```

> **访问一对应的模型类关联对象的id语法：**

```
#多对应的模型类对象.关联类属性_id
p = Person.objects.get(id=1)
p.book_id
```

3 .通过模型类实现关联查询

由一模型类条件查询多模型类数据：

> **语法如下：**

```
关联模型类名小写__属性名__条件运算符=值
```

> 如果没有"__运算符"部分，表示等于，结果和sql中的inner join相同。

```
# 查询图书，要求图书中人物的描述包含'八'。
list = Book.objects.filter(persono__content__contains='八')
```

由多模型类条件查询一模型类数据：

> 语法如下：

```
一模型类关联属性名__一模型类属性名__条件运算符=值
```

```
# 查询书名为“天龙八部”的所有人物。
list = Person.objects.filter(book__title='天龙八部')
```

8.增删改操作

1、添加数据

> 通过模型类往数据库添加数据

```
Book.objects.create(title='新闻001')
```

2、删除数据，使用delete方法

> 通过模型类删除数据

```
obj = Book.objects.all()[3]
obj.delete()
```

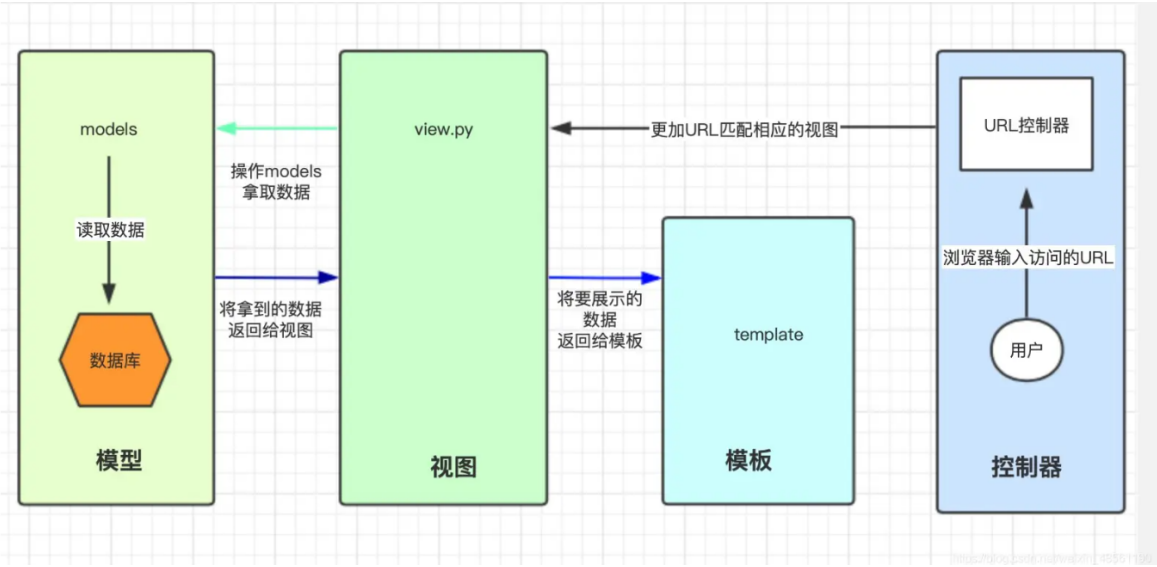
3、修改数据

> 通过模型类修改数据

```
obj = Book.objects.all()[3]
obj.title = '修改之后的标题'
obj.save()
```

三、视图

1、视图的基本使用



1.项目准备

> 1、创建项目
>
> 2、创建应用
>
> 3、注册应用

```
INSTALLED_APPS = (
 'django.contrib.admin',
 'django.contrib.auth',
 'django.contrib.contenttypes',
 'django.contrib.sessions',
 'django.contrib.messages',
 'django.contrib.staticfiles',
 'news',
)
```

4、配置数据库

```
DATABASES = {
 'default': {
     'ENGINE': 'django.db.backends.mysql',
     'NAME': 'demo3', # 指定数据库名字
     'USER':'root',
     'PASSWORD':'mysql',
     'PORT':3306,
     'HOST':'localhost'
 }
}
```

5、创建模型类

在应用目录下的`models.py`定义如下代码

```
class NewsInfo(models.Model):
 title = models.CharField(max_length=30)
 content = models.TextField()
```

6、生成迁移

7、执行迁移

8、配置模板路径

在项目目录下新建tamplates的文件夹

设置settings.py文件，设置TEMPLATES的DIRS值

```
'DIRS': [BASE_DIR/'templates']
```

2.定义视图

在`views.py`中定义视图

- 视图函数必须定义一个参数,参数名推荐使用request
- 返回的是一个HttpRequest类型的对象。

```
from django.shortcuts import render

# Create your views here.
from django.http import  HttpResponse

def index(request):
    return HttpResponse('<h1>个是新定义的视图index<h1>')
```

3.配置路由

> 1、在应用中创建一个`urls.py`文件
>
> 2、在项目的urls文件中包含具体应用的urls文件

```
from django.contrib import admin
from django.urls import path, include, re_path

urlpatterns = [
    path('admin/', admin.site.urls),
    # 匹配到news开头的路径，则交给news.urls去处理
    re_path(r'^news', include('news.urls'))
]
```

> 3、在应用的`urls.py`配置具体的路由

```
from django.urls import path, re_path #导入url函数
from .views import *
# 路由访问规则定义在名为urlpatterns的列表中
urlpatterns = [
 path(r'^$',index), #建立url和index视图函数的关联
]
```

4.启动项目，

```
python manage.py runserver
```

2、路由

用户通过在浏览器的地址栏中输入网址请求网站，对于Django开发的网站，由哪一个视图进行处理请求，是由url匹配找到的。

1、基本配置

> 1、路由规则是由`settings.py`中通过ROOT_URLCONF指定url配置，默认指定的项目项目下的`urls.py`。

```
ROOT_URLCONF = 'demo3.urls'
```

> 2、打开项目中的`urls.py`可以看到默认配置

```
from django.contrib import admin
from django.urls import path, include, re_path

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

3、建议在项目目录下的urls.py中进行包含配置应用，在各自应用中创建具体配置路由规则，以便维护和管理。

```
urlpatterns = [
 path('admin/', admin.site.urls),
 # 包含应用中的url
 re_path(r'^',include('应用.urls'))
]
```

2、路由配置函数

- **说明**

请求的url被看做是一个普通的python字符串，进行匹配时不包括域名、get或post参数。 如请求地址如下：

```
http://172.0.0.1:8000/index1/
```

路由匹配部分

```
index/
```

- **path**

用法

```
path('路由路径','视图函数')
```

案例

```
from django.urls import path
from .views import *
urlpatterns = [
 path('/index', views.index),
 path('/list', views.list1)
]
```

- **re_path**

用法

```
re_path(正则表达式,'视图函数名称')
```

案例

```
from django.urls import re_path
from .views import *
urlpatterns = [
 url(r'^$',index),
 url(r'^index$',index)
]
```

说明1：正则匹配推荐使用r，表示字符串不转义，这样在正则表达式中使用\只写一个就可以。

说明2：不能在开始加反斜杠，推荐在结束加反斜杠。

```
正确：index/
正确：index
错误：/index
错误：/index/
```

3、url参数

可以在匹配过程中从url中捕获参数，每个捕获的参数都作为一个普通的python字符串传递给视图。

- **方式一：位置参数**

直接使用小括号，通过位置参数传递给视图。

1、参数匹配

```
re_path(r'^index/(\d+)/$',views.show),
```

2、视图中获取参数

```
def show(request,num):
 return HttpResponse('这个是show返回的页面,index后面的num:%s'%num)
```

- **方式二：关键字参数**

1、参数匹配

其中`?P<name1>`表示为这个参数定义的名称为name1，可以是其它名称，起名做到见名知意。

```
re_path(r'^index/(?P<name1>\d+)/$',show),
```

2、视图中获取参数

视图函数中接受的参数名要与匹配时定义的参数名一致

```
def show(request,name1):
 return HttpResponse('这个是show返回的页面,index后面的num:%s'%name1)
```

- **注意：两种参数的方式不要混合使用，在一个正则表达式中只能使用一种参数方式。**

3、错误视图

Django内置处理HTTP错误的视图，主要错误及视图包括：

- 404错误：url匹配失败，找不到页面
- 500错误：server error视图

如3果想看到错误视图而不是调试信息，需要修改`setting.py`文件的DEBUG项。

```
# 一般线上环境才会开启
DEBUG = False
ALLOWED_HOSTS = ['*']
```

1.404错误视图配置

> 将请求地址进行url匹配后，没有找到匹配的正则表达式，则调用404视图，这个视图会调用404.html的模板进行渲染。视图传递变量request_path给模板，表示导致错误的URL。
>
> **1、在templates中创建404.html。**
>
> ```
> <html>
> <head>
>  <title></title>
> </head>
> <body>
>  找不到该页面
>  <hr/>
>  {{request_path}}
> </body>
> </html>
> ```
>
> 2、在浏览器中访问没有定义的路由：
>
> ```
> http://127.0.0.1:8000/test/
> ```

2.500错误视图配置

> 在视图中代码运行报错会发生500错误，调用内置错误视图，使用`templates/500.html`模板渲染。
>
> **1、定义错误的视图**
>
> ```
> def show(request,num):
>  print(a)   #  a 没定义  代码这里会报错
>  return HttpResponse('这个是show返回的页面,index后面的num:%s'%num)
> ```
>
> 访问上面的视图，调用show视图函数出错，则返回500页面

4、Reqeust

服务器接收到http协议的请求后，会根据报文创建HttpRequest对象，这个对象不需要我们创建，由Django构造好传给我们的视图函数。

1、request对象的属性

>

- path：一个字符串，表示请求的页面的完整路径，不包含域名和查询参数部分。
- method：一个字符串，表示请求使用的HTTP方法，常用值包括：'GET'、'POST'。
- encoding：一个字符串，表示提交的数据的编码方式。
- GET：QueryDict类型对象，类似于字典，包含get请求方式的所有参数。
- POST：QueryDict类型对象，类似于字典，包含post请求方式的所有参数。
- FILES：一个类似于字典的对象，包含所有的上传文件。
- COOKIES：一个标准的Python字典，包含所有的cookie，键和值都为字符串。
- session：一个既可读又可写的类似于字典的对象，表示当前的会话，只有当Django 启用会话的支持时才可用，详细内容见"状态保持"。

2、案例演示

1、编写视图

```
def index(request):
 str1 = 'paht:{}---method:{}----encoding:
{}'.format(request.path,request.method,request.encoding)
 return HttpResponse(str1)
```

2、浏览器访问，则可以看相关信息

3、QueryDict对象

HttpRequest对象的属性GET、POST都是QueryDict类型的对象,与python字典不同，QueryDict类型的对象用来处理同一个键带有多个值的情况

1、方法get()：

- 根据键获取值，如果一个键同时拥有多个值将获取最后一个值
- 如果键不存在则返回None值，可以设置默认值进行后续处理

```
dict.get('键',默认值)
可简写为
dict['键']
```

2、方法getlist()：

- 根据键获取值，值以列表返回，可以获取指定键的所有值
- 如果键不存在则返回空列表[]，可以设置默认值进行后续处理

```
dict.getlist('键',默认值)
```

4、GET属性

用来获取请求参数

案例

```
# 请求格式：在请求地址结尾使用?，之后以"键=值"的格式拼接，多个键值对之间以&连接。
https://www.baidu.com/s?wd=python&a=1&b=2
# 其中的请求参数为：
a=1
b=2
wd=python
```

- 分析请求参数，键为'a'、'b'、'wd'，值为'10'、'20'、'python'。
- 在Django中可以使用HttpRequest对象的GET属性获得get方方式请求的参数。
- GET属性是一个QueryDict类型的对象，键和值都是字符串类型。

5、POST属性

POST属性接收post请求传递的参数，POST属性是一个QueryDict类型的对象。

案例

1）编写视图函数

```
#接收请求参数
def methos_show(request):
 if request.method == 'GET':
     a = request.GET.get('a') #获取请求参数a
     b = request.GET.get('b') #获取请求参数b
     c = request.GET.get('c') #获取请求参数c
     content = {'a':a,'b':b,'c':c}
     return render(request, 'app1/get.html', content)
 else:
     name = request.POST.get('id') #获取账号
     password = request.POST.get('pw') #获取密码
     return render(request, 'app1/post.html', locals())
```

2）编写模板`index.html`

```
<html>
<head>
 <title>首页</title>
</head>
<body>
<h2>提交数据的两种方式：<h2><br>
get方式:<br/>
<a href="/method_show/?a=99&b=88&c=python">get方式提交数据</a><br/>

post方式:<br/>
<form method="post" action="/method_show/">
 账号：<input type="text" name="name"><br/>
 密码：<input type="password" name="password">

 <input type="submit" value="提交">
</form>
<br/>
</body>
</html>
```

3、编写模板get.html

```
<html>
<head>
 <title>GET方式提交数据</title>
</head>
<body>
<h1>get提交的数据</h1>
a:{{ a }}<br/>
b:{{ b }}<br/>
c:{{ c }}<br/>
</body>
</html>
```

4、编写模板post.html:

```
<html>
<head>
 <title>post</title>
</head>
<body>

<h1>post方式提交数据</h1>

提交的账号:{{ name }}<br/>
提交的密码:{{ password }}<br/>
</body>
</html>
```

5、Response

视图在接收请求并处理后，必须返回HttpResponse对象或子对象。

1、初始化参数

- **content：表示返回的内容字符串。**
- **charset：表示response采用的编码字符集，默认为utf-8。**
- **status_code：返回的HTTP响应状态码。**
- **content-type：指定返回数据的的MIME类型，默认为'text/html'。**

2、常用方法

- **set_cookie：**

设置Cookie信息。

```
set_cookie(key, value='', max_age=None, expires=None)
```

- **max_age是一个整数，表示在指定秒数后过期。**
- **expires是一个datetime或timedelta对象，会话将在这个指定的日期/时间过期。**
- **max_age与expires二选一。**

> - **如果不指定过期时间，在关闭浏览器时cookie会过期。**
> - write：向响应体中写数据。

- **delete_cookie：**

> **删除指定的key的Cookie**
>
> ```
> delete_cookie(key)
> ```

3、JsonResponse

JsonResponse是HttpResponse的子类，用来返回json数据，JsonResponse对象返回的content-type为'application/json'。

- **案例**

> **1）定义视图函数**
>
> ```
> from django.http import JsonResponse
>
> def json1(request):
>  return render(request,'app1/json1.html')
>
> def json2(request):
>  return JsonResponse({'h1':'hello','h2':'world'})
> ```
>
> **2）配置url**
>
> ```
>  re_path(r'^json1/$', views.json1),
>  re_path(r'^json2/$', views.json2),
> ```

4、重定向

当一个逻辑处理完成后，不需要向客户端呈现数据，而是转回到其它页面，如添加成功、修改成功、删除成功后显示数据列表，而数据的列表视图已经开发完成，此时不需要重新编写列表的代码，而是转到这个视图就可以，此时就需要模拟一个用户请求的效果，从一个视图转到另外一个视图，就称为重定向。Django中提供了HttpResponseRedirect对象实现重定向功能，这个类继承自HttpResponse，返回的状态码为302。

示例

> **1）定义视图**
>
> ```
> from django.http import HttpResponseRedirect
> ...
> # 定义重定义向视图，转向首页
> def red1(request):
>  return HttpResponseRedirect('/')
> ```

2）配置url

```
 url(r'^red1/$', views.red1),
```

简写redirect

```
from django.shortcuts import render,redirect
...
def red1(request):
 return redirect('/')
```

6. 登录案例

url	视图	模板文件
/login	login	login.html

1、 显示出登录页面

- 配置路由

```
re_path(r'^login$',login)
```

- 编写视图

```
def login(request):
    return render(request,'app1/login.html')
```

- 编写模板

```
<!DOCTYPE html>
<html lang="en">
<head>
    <meta charset="UTF-8">
    <title>登陆页面</title>
</head>
<body>
<form method="post" action="/login_check">
    账号：<input type="text" name="user"><br>
    密码：<input type="password" name="password"><br>
    <input type="submit" value="登陆">
</form>
</body>
</html>
```

2、 登录校验功能

- 配置路由

```
re_path(r'login_check$',login_check),
```

- 编写视图函数

```
def login_check(request):
    dic = {'python': '123456'}
    user = request.POST.get('user')
    password = request.POST.get('password')
    print(user)
    print(password)
    if dic.get(user)==password:
        return HttpResponseRedirect('/')
    else:
        return HttpResponseRedirect('/login')
```

7. Cookie和Session

有时需要保存下来用户浏览的状态，比如用户是否登录过，浏览过哪些商品等。 实现状态保持主要有两种方式：

- 在客户端存储信息使用Cookie。
- 在服务器端存储信息使用Session。

1、Cookie

cookie是由服务器生成，存储在浏览器端的一小段文本信息。

cookie的特点：

- 以键值对方式进行存储。
- 通过浏览器访问一个网站时，会将浏览器存储的跟网站相关的所有cookie信息发送给该网站的服务器。request.COOKIES
- cookie是基于域名安全的。www.baidu.com www.tudou.com
- cookie是有过期时间的，如果不指定，默认关闭浏览器之后cookie就会过期。

设置Cookie

1）编写视图

```
#不设置过期时间
def cookie_set(request):
    response = HttpResponse("<h1>设置Cookie，请查看响应报文头</h1>")
    response.set_cookie('h1', '9999')
    return response

# 设置过期时间为了14天
def cookie_set(request):
    response = HttpResponse("<h1>设置Cookie，请查看响应报文头</h1>")
    from datatime import *  #导入日期模块
    d_time=datetime.now()+timedelta(days=14) #获取当前时间 再加上14天
    response.set_cookie('h1', '9999', expires=d_tim)
```

2）配置路由

```
url(r'^cookie_set/$',views.cookie_set),
```

读取Cookie

Cookie信息被包含在请求头中，使用request对象的COOKIES属性访问。

1）打开views.py文件，创建视图cookie_get。

```
def cookie_get(request):
    str1 = '<h1>读取cookie数据,数据如下：</h1><br>{}'.format(request.COOKIES)
   return HttpResponse(str1)
```

2）打开urls.py文件，配置url。

```
    url(r'^cookie_get/$',views.cookie_get),
```

删除cookie

```
def cookie_del(request):
    response = HttpResponse('删除cookie')
   # 删除cookie
    response.delete_cookie(name)
   return response
```

2、Session

session存储在服务器端。

1、session的特点：

- 1) session是以键值对进行存储的。
- 2) session依赖于cookie。唯一的标识码保存在sessionid cookie中。
- 3) session也是有过期时间，如果不指定，默认两周就会过期。

2、Django项目Session配置。

打开settings.py文件，在项MIDDLEWARE_CLASSES中启用Session中间件。

```
MIDDLEWARE_CLASSES = (
 'django.contrib.sessions.middleware.SessionMiddleware',  #默认启用的
 'django.middleware.common.CommonMiddleware',
 # 'django.middleware.csrf.CsrfViewMiddleware',
 'django.contrib.auth.middleware.AuthenticationMiddleware',
 'django.contrib.auth.middleware.SessionAuthenticationMiddleware',
 'django.contrib.messages.middleware.MessageMiddleware',
 'django.middleware.clickjacking.XFrameOptionsMiddleware',
 'django.middleware.security.SecurityMiddleware',
)
```

禁用Session：将Session中间件删除。

存储方式

打开settings.py文件，设置SESSION_ENGINE项指定Session数据存储的方式，可以存储在数据库、缓存、Redis等。

1）存储在数据库中，如下设置可以写，也可以不写，这是默认存储方式。

```
SESSION_ENGINE='django.contrib.sessions.backends.db'
```

2）存储在缓存中：存储在本机内存中，如果丢失则不能找回，比数据库的方式读写更快。

```
SESSION_ENGINE='django.contrib.sessions.backends.cache'
```

3）混合存储：优先从本机内存中存取，如果没有则从数据库中存取。

```
SESSION_ENGINE='django.contrib.sessions.backends.cached_db'
```

4）如果存储在数据库中，需要在项INSTALLED_APPS中安装Session应用。

```
INSTALLED_APPS = (
 'django.contrib.admin',
 'django.contrib.auth',
 'django.contrib.contenttypes',
 'django.contrib.sessions', # 默认注册好了
 'django.contrib.messages',
 'django.contrib.staticfiles',
 'app1',
)
```

5）迁移后会在数据库中创建出存储Session的表。

```
mysql> show tables;
+----------------------------+
| Tables_in_demo3            |
+----------------------------+
| app1_newsinfo              |
| auth_group                 |
| auth_group_permissions     |
| auth_permission            |
| auth_user                  |
| auth_user_groups           |
| auth_user_user_permissions |
| django_admin_log           |
| django_content_type        |
| django_migrations          |
| django_session             |
+----------------------------+
11 rows in set (0.00 sec)
```

6）表结构如下图。

```
mysql> desc django_session;
+--------------+-------------+------+-----+---------+-------+
| Field        | Type        | Null | Key | Default | Extra |
+--------------+-------------+------+-----+---------+-------+
| session_key  | varchar(40) | NO   | PRI | NULL    |       |
| session_data | longtext    | NO   |     | NULL    |       |
| expire_date  | datetime(6) | NO   | MUL | NULL    |       |
+--------------+-------------+------+-----+---------+-------+
3 rows in set (0.00 sec)
```

> 由表结构可知，操作Session包括三个数据：键，值，过期时间。

3、session的操作

通过HttpRequest对象的session属性进行会话的读写操作。

1） 以键值对的格式写session。

```
request.session['键']=值
```

2）根据键读取值。

```
request.session.get('键',默认值)
```

3）清除所有session，在存储中删除值部分。

```
request.session.clear()
```

4）清除session数据，在存储中删除session的整条数据。

```
request.session.flush()
```

5）删除session中的指定键及值，在存储中只删除某个键及对应的值。

```
del request.session['键']
```

6）设置会话的超时时间，如果没有指定过期时间则两个星期后过期。

```
request.session.set_expiry(value)
```

- 如果value是一个整数，会话将在value秒没有活动后过期。
- 如果value为0，那么用户会话的session将在用户的浏览器关闭时过期。
- 如果value为None，那么会话永不过期。

4、案例

写session

1）编写视图：

```
def sesion_set(request):
 request.session['num']=10000
 return HttpResponse('把num写入session中')
```

2）配置url

```
re_path(r'^session_set/$',views.session_set),
```

读session

1）编写视图：

```
def session_get(request):
 h1=request.session.get('num')
 return HttpResponse(num)
```

2) 配置url

```
re_path(r'^session_get',session_get),
```

删除

```
def session_del(request):
 del request.session['h1']
 return HttpResponse('ok')
```

8、类视图案例

Django提供基于类的视图，与基于函数的视图相比具有如下优势

- 不同的 HTTP 方法（`GET`, `POST`, 等等）关联不同的方法来处理
- 面向对象技术可用于将代码分解，提供代码的重用性。

需求:

- 给`/cls/book/`这个路由，实现get请求，返回所有新闻数据
- 给`/cls/book/`这个路由，实现post请求，添加新闻数据，添加成功和失败均返回对应提示
  - 参数
    - title:"新闻标题" （必填）
    - content:"新闻内容"（必填）

步骤一：定义视图

```
class NewsView(View):
    def get(self, request):
        """get登录方法,返回所有的新闻信息"""
        # 获取所有的新闻数据
        ns = NewsInfo.objects.all()
        # 2、转换为python的列表
        result = []
        for i in ns:
            # 获取新闻数据
            item = dict(id=i.id,title=i.title,content=i.content,)
            result.append(item)
        # 3、返回给前端
        return JsonResponse(result, safe=False)

    def post(self, request):
        """post方法,添加新闻信息"""
        title = request.POST.get('title')
        content = request.POST.get('content')
        if title and content:
            # 添加到数据库
```

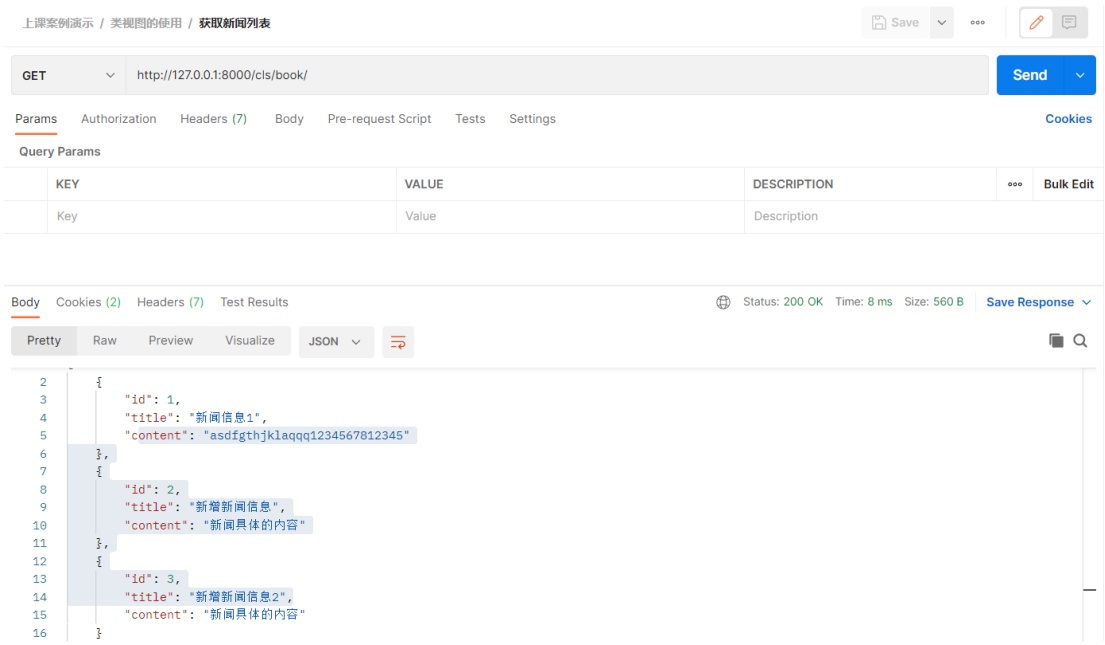
```
            NewsInfo.objects.create(title=title, content=content)
            return JsonResponse({'message': "添加成功"})
        else:
            return JsonResponse({'error': "添加失败，参数title和content均不能为空！"})
```

步骤二：配置路由

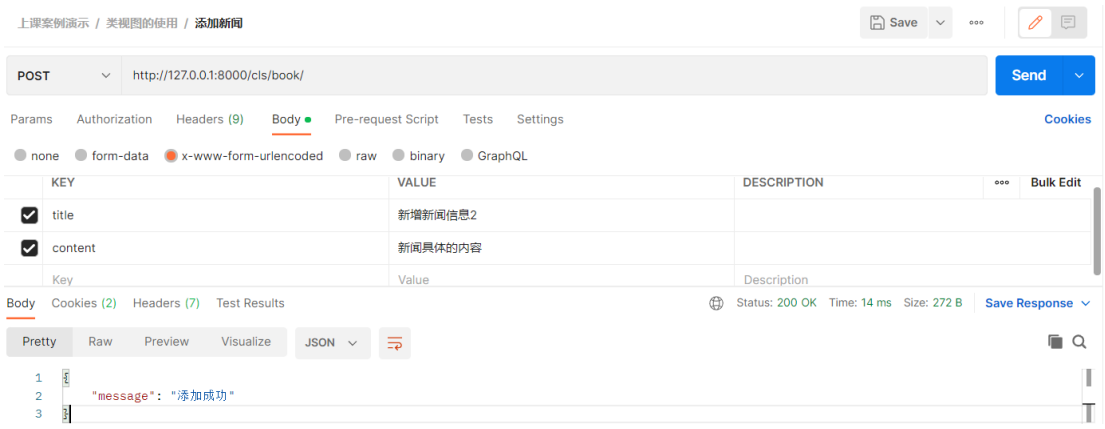
```
from django.urls import  re_path
from . import views

urlpatterns = [
    re_path(r'book/$', views.NewsView.as_view())
]
```

步骤三：postman调试接口

- 获取数据



- 添加数据



四、模板

为了减少开发人员重复编写加载、渲染的代码，Django提供了简写函数render，用于调用模板。

模板包含两部分：

- 静态部分，包含html、css、js。
- 动态部分，就是模板语言。

> **模板配置**
>
> 创建项目后，在"项目名称/settings.py"文件中定义了关于模板的配置

```
TEMPLATES = [
 {
     'BACKEND': 'django.template.backends.django.DjangoTemplates',
     # 修改模板存放的根路径
     'DIRS': [BASE_DIR / 'tamplates'],
     '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',
         ],
     },
 },
]
```

> DIRS定义一个目录列表，模板引擎按列表顺序搜索这些目录以查找模板文件，通常是在项目的根目录下创建templates目录。

Django处理模板分为两个阶段：

- 1.加载：根据给定的路径找到模板文件，编译后放在内存中。
- 2.渲染：使用上下文数据对模板插值并返回生成的字符串。

1、创建示例项目

1.项目准备

> **1、创建项目**
>
> **2、创建应用**
>
> **3、注册应用**

```
INSTALLED_APPS = (
'django.contrib.admin',
'django.contrib.auth',
'django.contrib.contenttypes',
'django.contrib.sessions',
'django.contrib.messages',
'django.contrib.staticfiles',
'news',
)
```

4、配置数据库

```
DATABASES = {
'default': {
  'ENGINE': 'django.db.backends.mysql',
  'NAME': 'demo3', # 指定数据库名字
  'USER':'root',
  'PASSWORD':'mysql',
  'PORT':3306,
  'HOST':'localhost'
}
}
```

5、配置模板路径

在项目目录下新建tamplates的文件夹

设置settings.py文件，设置TEMPLATES的DIRS值

```
'DIRS': [BASE_DIR/'templates']
```

6、在应用目录下创建`urls.py`，配置`url`。

```
from django.conf.urls import url
from .views import *
urlpatterns=[
 url(r'^$',index),
]
```

7、打开`views.py`文件，定义视图index。

```
from django.shortcuts import render
def index(request):
 return render(request,'app1/index.html')
```

8、在`templates/news目录下创建文件`index.html`，代码如下：

```
<html>
<head>
 <title>首页</title>
</head>
<body>
</body>
</html>
```

9、打开`app1/models.py`文件，定义模型类`NewsInfo`

```
from django.db import models

class NewsInfo(models.Model):
 news_title = models.CharField(max_length=20)
 news_content = models.TextField()
 news_date = models.DateField()
 isDelete = models.BooleanField(default=False)
```

10、生成迁移、执行迁移

11、配置Django后台

- 设置管理界面本地化

```
LANGUAGE_CODE = 'zh-hans' #使用中国语言
TIME_ZONE = 'Asia/Shanghai' #使用中国上海时间
```

- 创建admin管理员，命令如下

```
python manage.py createsuperuser
```

- admin.py中注册模型类，启动服务器，登录admin后台，添加几条初始数据

```
from .models import *
admin.site.register(NewsInfo)
```

2、模板语言

模板语言包括4种类型，分别是：

- **变量**
- **标签**
- **过滤器**
- **注释**

2.1、模板变量

模板变量的作用是计算并输出，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。

语法如下：

```
{{变量}}
```

当模版引擎遇到点如dict.title，会按照下列顺序解析：

- 1.字典dict['title']
- 2.先属性后方法，将dict当作对象，查找属性title，如果没有再查找方法title()
- 3.如果是格式为dict.0则解析为列表dic[0]

> 如果变量不存在则插入空字符串''。

在模板中调用方法时不能传递参数。

案例

1、编写视图

```
def temp(request):
 a = 'aaaa'
 b = {'b1':'b1111','b2':'b2222'}
 c = [1,2,3]
 news = NewsInfo()
 news.title = 'new的属性'
 return render(request,'app1/temp.html',locals())
```

2、配置url。

```
url(r'^temp/$', views.temp),
```

3、修改在templates/news下创建temp.html。

```
<!DOCTYPE html>
<html lang="en">
<head>
 <meta charset="UTF-8">
 <title>temp</title>
</head>
<body>
<h2>
a变量：{{a}}<br>
字典属性值b.b1：{{b.b1}},<br>
列表对象的下标c.0的值：{{c.0}}<br>
对象news的tltle属性：{{news.title}}<br>
</h2>
</body>
</html>
```

2.2、模板标签

标签在渲染过程中提供了任意逻辑。例如，标签可以输出内容，或用作控制结构如 “if” 语句和 “for” 循环，或从数据库中读取内容，甚至可以访问其他模板标签。

- **标签语法**

```
{% 代码段 %}
```

1、for标签的使用

语法如下：

```
{%for item in 列表%}
循环逻辑
{{forloop.counter}}表示当前是第几次循环，从1开始
{%empty%}
列表为空或不存在时执行此逻辑
{%endfor%}
```

案例

1、编写视图

```
from .models import NewsInfo
def temp2(request):
 news_list = NewsInfo.object.all()
 return render(request,'news/temp2.html',locals())
```

2、配置url。

```
 url(r'^temp2$',temp2),
```

3、创建temp2.html。

```
<!DOCTYPE html>
<html lang="en">
<head>
 <meta charset="UTF-8">
 <title>标签</title>
</head>
<body>

<h1>新闻列表如下：</h1>
<ul>
 {%for new in news_list%}
     <h2><li>{{new.news_title}}</li></h2>
 {%endfor%}
</ul>

</body>
</html>
```

- **if标签**

if标签语法如下：

```
{%if ...%}
逻辑1
{%elif ...%}
逻辑2
{%else%}
逻辑3
{%endif%}
```

案例

更改for标签案例代码：

```
<!DOCTYPE html>
<html lang="en">
<head>
 <meta charset="UTF-8">
 <title>标签</title>
</head>
<body>
<h1>新闻列表如下：</h1>
<ul>
```

```
 {%for new in new_list%}
     {%if new.id < 2%}
         <h2><li style="background: red">{{new.news_title}}</li></h2>
     {%elif new.id < 4%}
         <h2><li style="background: gold">{{new.news_title}}</li></h2>
     {%else%}
         <h2><li style="background: grey">{{new.news_title}}</li></h2>
     {%endif%}
 {%empty%}
     <h2 style="color: red">没有获取到数据</h2>
 {%endfor%}
</ul>
</body>
</html>
```

- **比较运算符使用**

> 注意：运算符左右两侧不能紧挨变量或常量，必须有空格。

```
{% a == 9 %}
{% a != 9 %}
{% a < 9 %}
{% a > 9 %}
{% a <= 9 %}
{% a >= 9 %}
```

- **逻辑运算符使用**

```
{% a == 9 and b == 8 %}
{% a != 9 or b >9 %}
{% not a < 9 %}
```

2.3、过滤器

- **内建过滤器：https://docs.djangoproject.com/zh-hans/3.2/ref/templates/builtins/#ref-templates-builtins-filters**

语法如下

> - 使用管道符号 | 来应用过滤器，用于进行计算、转换操作，可以使用在变量、标签中。
> - 如果过滤器需要参数，则使用冒号:传递参数。
> - 语法：{{变量|过滤器}}，例如{{name|lower}}，表示将变量name的值变为小写输出
>
> ```
> 变量|过滤器:参数
> ```
>
> - 使用管道符号(|)来应用过滤器
> - 通过使用过滤器来改变变量的计算结果
> - 可以在if标签中使用过滤器结合运算符
>
> ```
> if list1 | length > 1
> ```
>
> - 过滤器能够被“串联”，构成过滤器链

```
name|lower|upper
```

- 过滤器可以传递参数，参数使用引号包起来

```
list|join:", "
```

- default：如果一个变量没有被提供，或者值为false或空，则使用默认值，否则使用变量的值

```
value|default:"什么也没有"
```

- 日期date，用于对日期类型的值进行字符串格式化，常用的格式化字符如下：

```
value|date:'Y-m-d'
```

- Y表示年，格式为4位，y表示两位的年。
- m表示月，格式为01,02,12等。
- d表示日, 格式为01,02等。
- j表示日，格式为1,2等。
- H表示时，24进制，h表示12进制的时。
- i表示分，为0-59。
- s表示秒，为0-59。

- **案例**

1、编写视图

```
def temp3(request):
 #模板过滤器
 new_list = NewsInfo.objects.all()
 return render(request,'news/temp3.html',locals())
```

2、配置url。

```
 url(r'^temp3/$', temp3),
```

3、创建模板

```
<!DOCTYPE html>
<html lang="en">
<head>
 <meta charset="UTF-8">
 <title>过滤器</title>
</head>
<body>
<h1>新闻数量为:{{new_list|length}}</h1>  //通过过滤器获取新闻总数
{%for new in new_list%}
 {%if new.id > 3%}
 <h2><li style="color: red">
 {{new.news_title}}
 ---原来的日期格式:{{new.news_date}}</li></h2>
```

```
 {%else%}
 <h2><li style="color: gold">
 {{new.news_title}}
 ---过滤器格式化的日期:
 {{new.news_date|date:"y-m-j"}}</li></h2>
 {%endif%}
{%endfor%}

</body>
</html>
```

2.4、注释

在模板中使用如下模板注释，这段代码不会被编译，不会输出到客户端；html注释只能注释html内容，不能注释模板语言。

> **1）单行注释语法如下：**
>
> ```
> {#...#}
> ```
>
> 注释可以包含任何模版代码，有效的或者无效的都可以。
>
> ```
> {# { % if foo % }bar{ % else % } #}
> ```
>
> **2）多行注释使用comment标签，语法如下：**
>
> ```
> {%comment%}
> ...
> {%endcomment%}
> ```

3、模板继承

模板继承和类的继承含义是一样的，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。

3.1、父模板

如果发现在多个模板中某些内容相同，那就应该把这段内容定义到父模板中。

标签block：用于在父模板中预留区域，留给子模板填充差异性的内容，名字不能相同。 为了更好的可读性，建议给endblock标签写上名字，这个名字与对应的block名字相同。父模板中也可以使用上下文中传递过来的数据。

```
{%block 名称%}
预留区域，可以编写默认内容，也可以没有默认内容
{%endblock  名称%}
```

3.2、子模板

标签extends：继承，写在子模板文件的第一行。

```
{% extends "父模板路径"%}
```

子模版不用填充父模版中的所有预留区域，如果子模版没有填充，则使用父模版定义的默认值。

填充父模板中指定名称的预留区域。

```
{%block 名称%}
实际填充内容
{{block.super}}用于获取父模板中block的内容
{%endblock 名称%}
```

> **案例**
>
> **1、定义视图**
>
> ```
> def temp4(request):
>  #模板过滤器
>  title = '模板继承'
>  new_list = NewsInfo.objects.all()
>  return render(request,'news/temp3.html',locals())
> ```
>
> **2、配置url。**
>
> ```
>  url(r'^temp4$',temp4),
> ```
>
> **3、创建父模板`bash.html`**
>
> ```
> <!DOCTYPE html>
> <html lang="en">
> <head>
>  <meta charset="UTF-8">
>  <title>{{title}}</title>
> </head>
> <body>
> <h1>------------------网页头部------------------</h1>
> <hr>
> {% block quyu1%}
> 这个是区域1,请添加内容
> {%endblock quyu1%}
> <hr>
> {%block quyu2%}
> 这个是区域2,请添加内容
> {%endblock quyu2%}
> <hr>
> <h1>------------------网页尾部--------------------</h1>
> </body>
> </html>
> ```
>
> **4、创建子模版`temp3.html`**

```
{% extends 'base.html'%}  <!--继承bash.html页面-->

{% block quyu2%}      <!--添加预留区域的内容-->-->

{%for new in new_list%}
<h3>
 <li>{{new.news_title}}</li>
</h3>
{%endfor%}

{%endblock quyu2%}
```

4、HTML转义

1、基本使用

模板对上下文传递的字符串进行输出时，会对以下字符自动转义。

```
小于号 < 转换为 &lt;
大于号 > 转换为 &gt;
单引号 ' 转换为 &#39;
双引号 " 转换为 &quot;
与符号 & 转换为 &amp;
```

案例

1、创建视图

```
#模板过滤器
content = '<h1>新闻列表</h1>'
return render(request,'app1/temp_html.html',locals())
```

2、配置url。

```
 url(r'^temp_html$',temp_html),
```

3、创建模板 `temp_html.html`

```
<!DOCTYPE html>
<html lang="en">
<head>
 <meta charset="UTF-8">
 <title>HTML转义</title>
</head>
<body>
content: {{content}}
</body>
</html>
```

2、关闭转义

过滤器escape可以实现对变量的html转义，默认模板就会转义，一般省略。

```
{{content|escape}}
```

过滤器safe：禁用转义，告诉模板这个变量是安全的，可以解释执行。

```
{{content|safe}}
```

1）修改`templates/app1/temp_html.html`代码如下。

```
<!DOCTYPE html>
<html lang="en">
<head>
    <meta charset="UTF-8">
    <title>HTML转义</title>
</head>
<body>
content：{{content}}
<hr>
过滤器safe关闭转义content：{{content|safe}}
</body>
</html>
```

刷新浏览器后效果如下图：

标签autoescape：设置一段代码都禁用转义，接受on、off参数。

```
{%autoescape off%}
...
{%endautoescape%}
```

1）修改templates/booktest/html_escape.html代码如下。

```
<html>
<head>
    <title>转义</title>
</head>
<body>
自动转义：{{content}}
<hr>
过滤器safe关闭转义：{{content|safe}}
<hr>
标签autoescape关闭转义：
{%autoescape off%}
{{content}}
{%endautoescape%}
</body>
</html>
```

5、CSRF

1. csrf攻击

CSRF全拼为Cross Site Request Forgery，译为跨站请求伪造。CSRF指攻击者盗用了你的身份，以你的名义发送恶意请求。CSRF能够做的事情包括：以你名义发送邮件，发消息，盗取你的账号，甚至于购买商品，虚拟货币转账……造成的问题包括：个人隐私泄露以及财产安全。

2django防止csrf的方式：

1、默认打开csrf中间件。

2、表单post提交数据时加上{% csrf_token %}标签。

防御原理:

1、 渲染模板文件时在页面生成一个名字叫做csrfmiddlewaretoken的隐藏域。

2、服务器交给浏览器保存一个名字为csrftoken的cookie信息。

3、提交表单时，两个值都会发给服务器，服务器进行比对，如果一样，则csrf验证通过，否则失败。

当启用中间件并加入标签csrf_token后，会向客户端浏览器中写入一条Cookie信息，这条信息的值与隐藏域input元素的value属性是一致的，提交到服务器后会先由csrf中间件进行验证，如果对比失败则返回403页面，而不会进行后续的处理。

五、其他功能

Django提供了这些功能后，可以帮助我们更快更好的完成开发。

1、静态文件

项目中的CSS、图片、js都是静态文件。一般会将静态文件放到一个单独的目录中，以方便管理。在html页面中调用时，也需要指定静态文件的路径，Django中提供了一种解析的方式配置静态文件路径。静态文件可以放在项目根目录下，也可以放在应用的目录下，由于有些静态文件在项目中是通用的，所以推荐放在项目的根目录下，方便管理。

案例

1、修改配置文件`settings.py`文件中定义静态文件存放的物理目录。

```
STATIC_URL = '/static/'
STATICFILES_DIRS = [
    BASE_DIR / 'static'
]
```

2、在项目根目录下创建static目录，再创建img、css、js目录。

3、定义视图

```
def static_test(request):
 return render(request,'news/static_test.html')
```

4、配置url。

```
 url(r'^static_test/$',views.static_test),
```

5、创建模板

```
<html>
<head>
 <title>静态文件</title>
</head>
<body>
<img src="/static/img/sta.png"/>
</body>
</html>
```

6、保存图片到static/img/目录下，名称为sta.png。

2、中间件

Django中的中间件是一个轻量级、底层的插件系统，可以介入Django的请求和响应处理过程，修改Django的输入或输出。中间件的设计为开发者提供了一种无侵入式的开发方式，增强了Django框架的健壮性，其它的MVC框架也有这个功能。

Django在中间件中预置了五个方法，这五个方法的区别在于不同的阶段执行，对输入或输出进行干预，方法如下：

1）初始化：

无需任何参数，服务器响应第一个请求的时候调用一次，用于确定是否启用当前中间件。

```
def __init__(self):
 pass
```

2）处理请求前：

在每个请求上，request对象产生之后，url匹配之前调用，返回None或HttpResponse对象。

```
def process_request(self, request):
 pass
```

3）处理视图前：

在每个请求上，url匹配之后，视图函数调用之前调用，返回None或HttpResponse对象。

```
def process_view(self, request, view_func, *view_args, **view_kwargs):
 pass
```

4）处理响应后：

视图函数调用之后，所有响应返回浏览器之前被调用，在每个请求上调用，返回HttpResponse对象。

```
def process_response(self, request, response):
 pass
```

5）异常处理：

当视图抛出异常时调用，在每个请求上调用，返回一个HttpResponse对象。

```
def process_exception(self, request,exception):
 pass
```

案例

中间件是一个独立的python类，，可以定义这五个方法中的一个或多个。

1、在app1/目录下创建middleware.py文件，代码如下：

```
class MyMedd:

 def __init__(self):
     print ('------init--------')

 def process_request(self,request):
     print ('------request--------')

 def process_view(self,request, view_func, *view_args, **view_kwargs):
     print ('--------view------')

 def process_response(self,request, response):
     print('-------response---')
     return response
```

2、在demo5/settings.py文件中，向MIDDLEWARE_CLASSES项中注册。

3、修改app1/views.py中视图index。

```
def index(request):
 print('-------index--------')
 return render(request,'app1/index.html')
```

注意：如果多个注册的中间件类中都有process_exception的方法，则先注册的后执行。

3、Django认证系统

Django 自带一个用户验证系统。它负责处理用户账号、组、权限和基于cookie的用户会话。

1、User对象

用户对象是认证系统的核心。它通常代表了与你的站点交互的人员，并用于允许诸如限制访问、注册用户配置文件、将内容与创建者关联等功能。Django 的认证框架中用户只有一个类，例如 “超级管理员” 或“普通管理员”只是具有特殊属性集的用户对象，而不是用户对象的不同类。

默认用户的主要属性是：

- **创建用户**

  可以使用Django自带的用户模型类来创建用户

```
from django.contrib.auth.models import User
user = User.objects.create_user('admin123', '1234@qq.com', '123456')
user.save()
```

- **创建超级用户命令**

```
python manage.py createsuperuser
```

- **更改密码**

  修改用户密码可以直接使用内置的模型类`User`来实现

```
from django.contrib.auth.models import User
# 查询用户
u = User.objects.get(username='john')
# 修改密码
u.set_password('new password')
u.save()
```

- **用户验证**

  Django中内置了一个验证用户账号密码的函数authenticate

```
from django.contrib.auth import authenticate
# 验证用户账号密码是否正确
user = authenticate(username='john', password='secret')
```

2、用户登录

通过authenticate可以验证账号密码是否正确，要把用户登录的状态保存到当前会话(session)中，可以使用Django提供的`login()`函数。下面是一个保存登录状态的登录

1、登录视图

```
from django.contrib.auth import authenticate, login
from django.http import JsonResponse
from django.views import View
class LoginView(View)

    def get(self,request):
     # 返回登录页面
     return render(request, 'login.html')

 def post(self,request):
     # 获取账号密码
     username = request.POST['username']
     password = request.POST['password']
     # 验证用户是否存在
     user = authenticate(request, username=username, password=password)
     if user is not None:
         # 保存用户的登录状态到session中
         login(request, user)
         return JsonResponse({'message': "登录成功"})
     else:
            return JsonResponse({'message': "登录失败"},status=400)
```

2、配置路由

```
re_path(r'login/$',LoginView.as_view())
```

3、退出登录

如果已经通过 `django.contrib.auth.login()` 登录的用户想退出登录，可以在视图中使用 `django.contrib.auth.logout()`，需要传入 `HttpRequest` 对象即可。下面演示一个退出登录的案例。

> **1、定义视图**

```
from django.contrib.auth import login, logout
from django.http import JsonResponse
from django.views import View
class LogoutView(View)
 def get(self,request):
     # 获取账号密码
    logout(request)
    return JsonResponse({'message': "已退出登录"},status=400)
```

> **2、配置路由**

```
re_path(r'logout/$',LogoutView.as_view())
```

4、权限校验

如果某个页面或接口，需要限制对未登录用户的访问，Django也提供了校验用户是否登录的功能。

> **1、通过 `request.user` 属性进行校验**

原始的办法就是检查 `request.user.is_authenticated` 并重定向到登录页面

```
from django.shortcuts import render,redirect
from django.views import View
from django.http import JsonResponse

class DemoView(View)
 def get(self,request):
     if not request.user.is_authenticated:
         return JsonResponse({'error':"您没有权限访问该页面"},status=403)
     else:
         res = {'message':"请求成功","data":[11,22,33,44]},
         return JsonResponse(res,status=403)
```

> **2、鉴权装饰器**

可以使用 `login_required()` 装饰器，来鉴别用户是否拥有访问权限。

```
from django.contrib.auth.decorators import login_required
from django.http import JsonResponse

# 视图函数
@login_required(login_url='/user/login/')
def my_view(request):
  return JsonResponse({'message': "登录成功"})

# 类视图
from django.contrib.auth.mixins import LoginRequiredMixin
```

```
class MyView(LoginRequiredMixin, View):
 login_url = '/login/'
 redirect_field_name = 'redirect_to'

 def get(self,request):
     pass

 def post(self,request):
     pass
```

3、`setting.py`中统一配置权限校验失败的地址重定向的路由地址

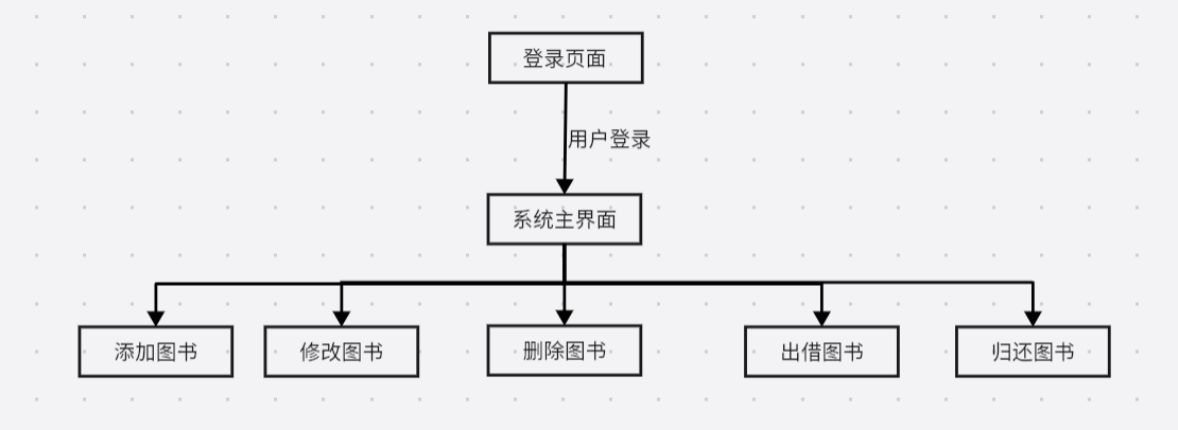
```
LOGIN_URL = 'user/login'
```

六、应用小案例

1、基本需求：

1、功能图

基于Django开发一个图书管理系统，功能要求如下：



2、数据库表结构

- **用户表(User)**

字段名	类型	说明
id	int	主键
username	str	用户名
password	str	密码

- **书籍表(Books)**

字段名	类型	说明
id	str	图书编号(唯一)
name	str	书籍名
statua	bool	是否出借()

- **借还记录表(Record)**

字段名	类型	说明
id	int	主键
book	int	书籍(ID)
s_date	datatime	借书时间
name	str	借书人
e_date	datatime	归还时间(默认为借书时间，归还后再更新为归还时间)

2、创建项目

1、创建项目`bookManage`

```
Django-admin startproject bookManage
```

2、创建应用`books`

```
python manage.py startapp books
```

3、修改配置

- **注册应用**

```
INSTALLED_APPS = [
    ...
    'books'
]
```

- **修改数据库配置**

```
DATABASES = {
    'default': {
        'ENGINE': 'django.db.backends.mysql',
        'NAME': 'books',  # 指定数据库
        'USER': 'root',  # 用户名
        'PASSWORD': 'mysql',  # 数据库所在主机的ip
        'PORT': 3306,  # 端口号
        'HOST': 'localhost'
    }
}
```

- **设置admin语言**

```
LANGUAGE_CODE = 'zh-hans'
TIME_ZONE = 'Asia/Shanghai'
```

- **注释csrftoken检验**

```
MIDDLEWARE = [
    'django.middleware.security.SecurityMiddleware',
    'django.contrib.sessions.middleware.SessionMiddleware',
    'django.middleware.common.CommonMiddleware',
    # 'django.middleware.csrf.CsrfViewMiddleware',
    'django.contrib.auth.middleware.AuthenticationMiddleware',
    'django.contrib.messages.middleware.MessageMiddleware',
    'django.middleware.clickjacking.XFrameOptionsMiddleware',
]
```

3、模型类设计

1、定义模型类

> ps:可以采用Django自带的用户系统，用户模型可以定义，只需要定义图书，和图书出借记录模型

```
class Books(models.Model):
    id = models.CharField(primary_key=True, max_length=20, verbose_name='图书编号')
    name = models.CharField(max_length=50, verbose_name='书名')
    statua = models.BooleanField(verbose_name='是否出借', default=False)

    class Meta:
        db_table = 'book'
        verbose_name = '图书表'

    def __str__(self):
        return self.name


class Record(models.Model):
    book = models.ForeignKey('Books', on_delete=models.CASCADE, verbose_name='书籍')
    name = models.CharField(max_length=20, verbose_name='借书人')
    s_time = models.DateTimeField(auto_now_add=True, verbose_name='借书时间')
    e_time = models.DateTimeField(auto_now=True, blank=True, verbose_name='还书时间')
    class Meta:
        db_table = 'record'
        verbose_name = '借还记录'

    def __str__(self):
        return self.book
```

2、激活模型

```
# 执行迁移生成迁移文件
python manage.py makemigretions
```

```
# 生成数据库表
python manage.py migrate
```

```
# 创建管理员账户
python manage.py createsuperuser
```

3、注册admin

```
from django.contrib import admin
from .models import Books, Record
# Register your models here.

@admin.register(Books)
class BookAdmin(admin.ModelAdmin):
    list_display = ['id', 'name', 'status']


@admin.register(Record)
class RecordAdmin(admin.ModelAdmin):
    list_display = ['id', 'book', 'name', 's_date', 'e_date']
```

4、接口实现

1、登录接口

1、定义视图

```
class LoginView(View):
 def post(self, request):
     # 获取账号密码
     username = request.POST.get('username')
     password = request.POST.get('password')
     # 验证用户是否存在
     user = authenticate(request, username=username, password=password)
     if user is not None:
         # 保存用户的登录状态到session中
         login(request, user)
         return JsonResponse({'code': 200, 'message': "登录成功"})
     else:
         return JsonResponse({'code': 400, 'message': "登录失败"},
status=400)
```

2、配置url

```
urlpatterns = [
 re_path(r'login/$', views.LoginView.as_view())
]
```

2、退出登录

1、定义视图

```
from django.shortcuts import render,redirect
from django.views import View
from django.http import JsonResponse
from django.contrib.auth import authenticate, login, logout
class LogoutView(View):
 def get(self, request):
     # 删除session信息
     logout(request)
     return JsonResponse({"code": 200, 'message': "已退出登录"}, status=200)
```

2、配置url

```
urlpatterns = [
 re_path(r'logout/$', views.LogoutView.as_view())
]
```

2、图书列表

1、定义视图

```
class BooksView(View):

 def get(self, request):
     # 1、查询所有的数据
     bs = Books.objects.all()
     # 2、转换为列表
     result = []
     for i in bs:
         item = dict(id=i.id, name=i.name, status=i.status, )
         result.append(item)
     # 3、返回给前端
     return JsonResponse({"code": 200, "data": result, "message": "OK"},
safe=False)
```

2、配置url

```
urlpatterns = [
 re_path(r'books/$', views.BooksView.as_view())
]
```

3、添加图书

1、定义视图

```
from django.shortcuts import render,redirect
```

```
from django.views import View
from django.http import JsonResponse

class BooksView(View):

 def post(self, request):
     # 添加图书
     id = request.POST.get('id')
     name = request.POST.get('name')
     if not (id and name):
         return JsonResponse({"code": 400, 'message': "图书编号的书名均不能为
空"})
     if Books.objects.filter(id=id).exists():
         return JsonResponse({"code": 400, 'message': "该书籍编号已存在，书籍编
号不能重复"})
     Books.objects.create(id=id, name=name)
     return JsonResponse({"code": 200, 'message': "书籍添加成功"})
```

2、配置url

```
urlpatterns = [
 re_path(r'logout/$', views.LogoutView.as_view())
]
```

4、删除图书

1、定义视图

```
from django.shortcuts import render,redirect
from django.views import View
from django.http import JsonResponse

class BooksView(View):
 def delete(self, request):
     # 删除图书
     id = request.GET.get('id')
     try:
         book = Books.objects.get(id=id)
     except Exception as e:
         return JsonResponse({"code": 400, 'message': "您输入的书籍编号有误,无
法进行删除操作"})
     else:
         book.delete()
         return JsonResponse({"code": 200, 'message': "删除成功"})
```

2、配置url

```
urlpatterns = [
 re_path(r'logout/$', views.LogoutView.as_view())
]
```

5、出借图书(练习)

1、实现步骤

- 根据书籍id找到对应书籍
- 判断书籍状态，是否处于可出借状态
- 修改书籍状态
- 添加一条借书记录

```
class LendView(View):
    """出借图书和归还退出"""

    def post(self, request):
        """
        # 1、登录权限校验
        if not request.user.is_authenticated:
            return JsonResponse({"code": 2002, "message": "认证失败，您未登
录，没有访问权限"})
        # 2、获取参数（同时支持json和表单参数）
        params = request.POST if len(request.POST) > 0 else
json.loads(request.body.decode())
        book_id = params.get('book')  # 获取要借的书籍
        name = params.get('name')  # 获取借书人的名字
        # 3、参数校验
        # 校验参数是否为空
        if not (book_id and name):
            return JsonResponse({"code": 2001, "message": "书籍编号和借书人名
字均不能为空"})
        # 校验书籍的编号是否正确
        try:
            book = Books.objects.get(id=book_id)
        except Exception as e:
            return JsonResponse({"code": 2001, "message": "书籍编有误，未找到
对应的书籍"})
        # 校验书籍的状态是否已经出借
        if book.status:
            return JsonResponse({"code": 2001, "message": "该书籍已经借出，请
确认书籍编号是否输错"})
        # 校验借书人的名字类型
        if not isinstance(name, str):
            return JsonResponse({"code": 2001, "message": "借书人的名字必须为
字符串类型"})
        # 4、借书操作(开启事务，如果下面的代码执行出现异常会自动回滚)
        with transaction.atomic():
            # 修改书籍的状态
            book.status = True
            book.save()
            # 添加一条借书记录
            Record.objects.create(book=book, name=name)
        return JsonResponse({"code": 1000, "message": "图书出借成功"})
```

6、归还图书(练习)

1、实现步骤

- 根据书籍id找到对应书籍
- 判断书籍状态，是否处于出借状态
- 修改书籍状态
- 修改借书记录