

Django

1. 软件框架



一个公司是由公司中的各 departments 来组成的，每一个部门拥有特定的职能，部门与部门之间通过相互的配合来完成让公司运转起来。

一个软件框架是由其中各个软件模块组成的，每一个模块都有特定的功能，模块与模块之间通过相互配合来完成软件的开发。

软件框架是针对某一类软件设计问题而产生的。

2. MVC 框架

2.1 MVC 简介

MVC 最初是由施乐公司旗下的帕罗奥多研究中心中的一位研究人员给 smalltalk 语言发明的一种软件设计模式。

MVC 的产生理念：分工。让专门的人去做专门的事。

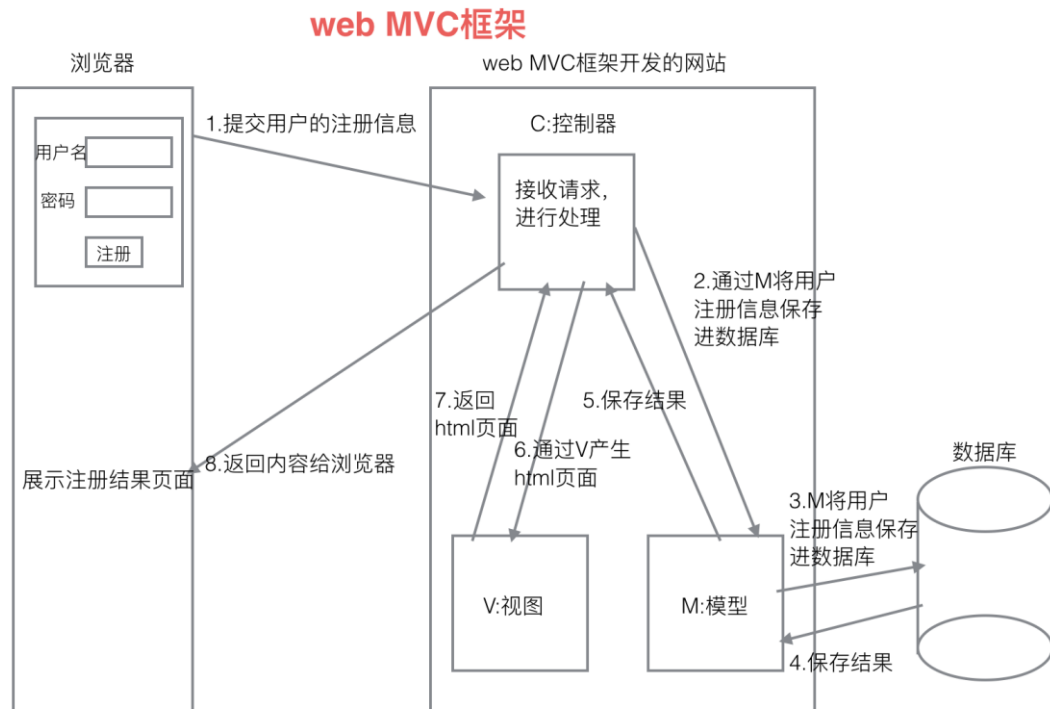
MVC 的核心思想：解耦。



MVC 的思想被应用在 web 开发的方面，产生了 web MVC 框架。

2.2 Web MVC 框架模块功能

通过浏览器注册用户信息。



M:Model, 模型， 和数据库进行交互。

V:View, 视图， 产生html 页面。

C:Controller, 控制器， 接收请求， 进行处理， 与 M 和 V 进行交互， 返回应答。

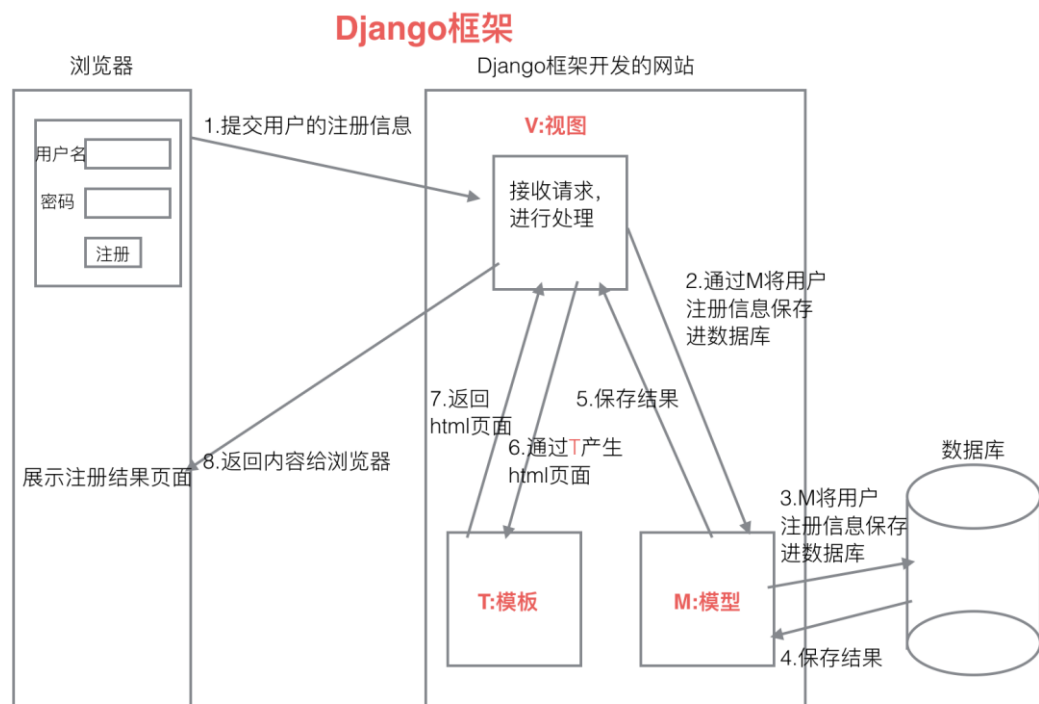
3. Django 框架

3.1 简介

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。

3.2 MVT 各部分功能



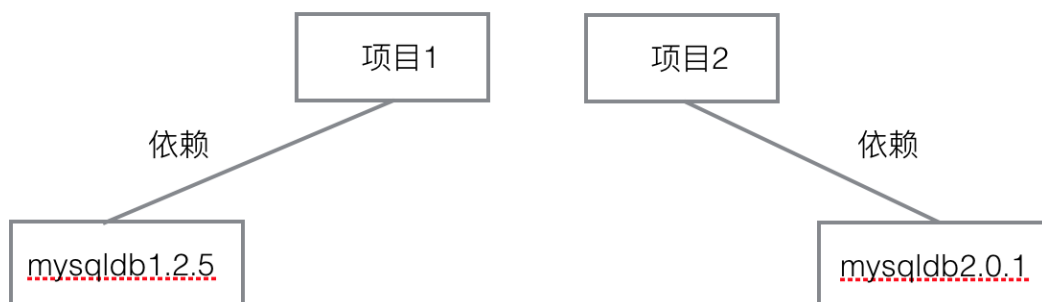
M:Model, 模型, 和 MVC 中 M 功能相同, 和数据库进行交互。

V:View, 视图, 和 MVC 中 C 功能相同, 接收请求, 进行处理, 与 M 和 T 进行交互, 返回应答。

T:Template, 模板, 和 MVC 中 V 功能相同, 产生 html 页面。

4. 虚拟环境

4.1 概念



之前安装 python 包的命令: `sudo pip3 install 包名`

包的安装路径: /usr/local/lib/python3.5/dist-packages

在同一个 python 环境中安装同一个包的不同版本, 后安装的包会把原来安装的包覆盖掉。这样, 如果同一台机器上两个项目依赖于相同包的不同版本, 则会导致一些项目运行失败。

解决的方案就是: 虚拟环境。

虚拟环境是真实 python 环境的复制版本。

在虚拟环境中使用的 python 是复制的 python, 安装 python 包也是安装在复制的 python 中。

4.2 安装和配置

安装虚拟环境的命令:

- 1) `sudo pip install virtualenv` #安装虚拟环境
- 2) `sudo pip install virtualenvwrapper` #安装虚拟环境扩展包
- 3) 编辑家目录下面的.bashrc 文件, 添加下面两行。

```
export WORKON_HOME=$HOME/.virtualenvs  
source /usr/local/bin/virtualenvwrapper.sh
```

- 4) 使用 `source .bashrc` 使其生效一下。

4.3 使用

创建虚拟环境命令:

```
mkvirtualenv 虚拟环境名
```

创建 python3 虚拟环境:

```
mkvirtualenv -p python3 bj11_py3
```

进入虚拟环境工作:

```
workon 虚拟环境名
```

查看机器上有多少个虚拟环境:

```
workon 空格 + 两个 tab 键
```

退出虚拟环境:

```
deactivate
```

删除虚拟环境：

```
rmvirtualenv 虚拟环境名
```

虚拟环境下安装包的命令：

```
pip install 包名
```

注意：不能使用 `sudo pip install 包名`，这个命令会把包安装到真实的主机环境上而不是安装到虚拟环境中。

查看虚拟环境中安装了哪些 python 包：

```
pip list
```

```
pip freeze
```

安装 django 环境：

```
pip install django==1.8.2
```

拓展：

```
apt-get install 软件
```

```
pip install python 包名
```

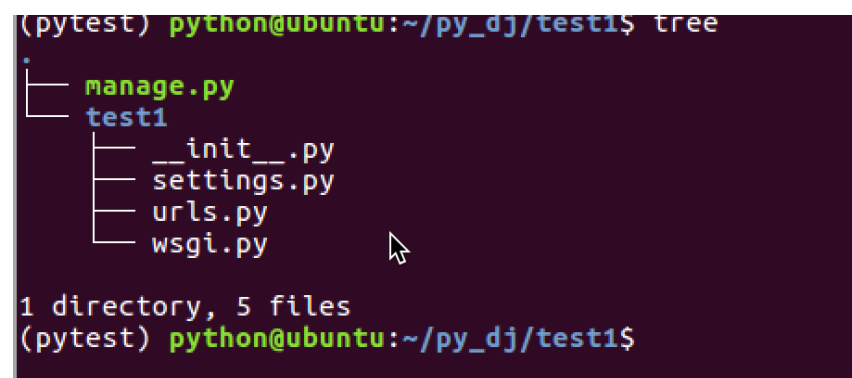
5. 项目创建

5.1 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：



```
(pytest) python@ubuntu:~/py_dj/test1$ tree
.
├── manage.py
└── test1
    ├── __init__.py
    ├── settings.py
    ├── urls.py
    └── wsgi.py

1 directory, 5 files
(pytest) python@ubuntu:~/py_dj/test1$
```

`__init__.py`：说明 `test1` 是一个 python 包。

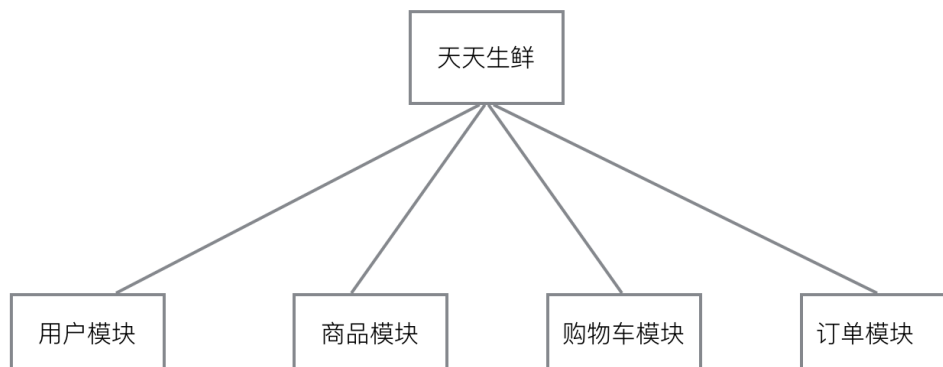
`settings.py`：项目的配置文件。

urls.py: 进行 url 路由的配置。

wsgi.py: web 服务器和 Django 交互的入口。

manage.py: 项目的管理文件。

5.2 创建 Django 应用



在Django中，一个**功能模块**使用一个**应用**来实现。

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 manage.py startapp 应用名
```

注意：创建应用时需要先进入项目目录。

应用目录如下：

```
(pytest) python@ubuntu:~/py_dj/test1/booktest$ tree
.
├── admin.py
├── __init__.py
├── migrations
│   └── __init__.py
├── models.py
├── tests.py
└── views.py
1 directory, 6 files
```

__init__.py: 说明目录是一个 Python 模块。

models.py: 写和数据库项目的内容，设计模型类。

views.py: ，接收请求，进行处理，与 M 和 T 进行交互，返回应答。

定义处理函数，**视图函数**。

tests.py: 写测试代码的文件。

admin.py: 网站后台管理相关的文件。

5.3 应用注册

建立应用和项目之间的联系，需要对应用进行注册。

修改 settings.py 中的 INSTALLED_APPS 配置项。

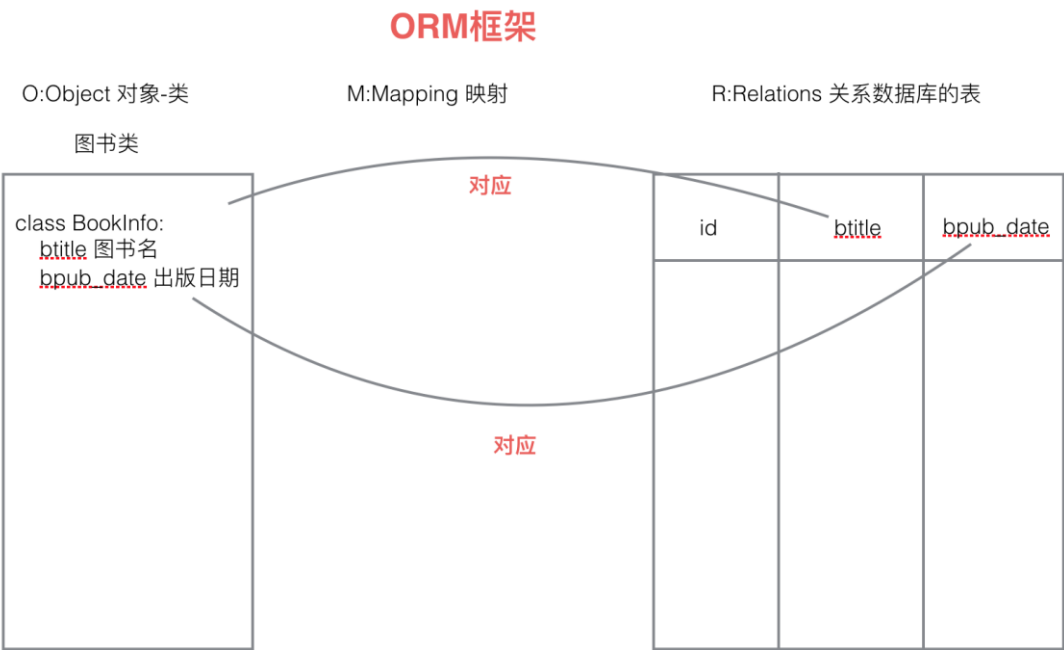
5.4 启动项目

运行开发 web 服务器命令：

```
python manage.py runserver
```

6. 模型类

6.1 ORM



django 中内嵌了 ORM 框架，ORM 框架可以将类和数据库表进行对应起来，只需要通过类和对象就可以对数据库表进行操作。

在 Django 中主要是设计类：模型类。

ORM 另外一个作用：根据设计的类生成数据库中的表。

6.2 模型类设计

在应用 `models.py` 中设计模型类。

必须继承与 `models.Model` 类。

- 1) 设计 BookInfo 类。
- 2) 设计 HeroInfo 类。

`Models.ForeignKey` 可以建立两个模型类之间一对多的关系, django 在生成表的时候, 就会在多端的表中创建一列作为外键, 建立两个表之间一对多的关系。

6.3 模型类生成表

- 1) 生成迁移文件

命令: `python manage.py makemigrations`



迁移文件是根据模型类生成的。

- 2) 执行迁移生成表

命令: `python manage.py migrate`

根据迁移文件生成表。

生成表名的默认格式:

应用名_模型类名小写

6.4 通过模型类操作数据表

进入项目 shell 的命令:

```
python manage.py shell
```

以下为在相互 shell 终端中演示的例子:

首先导入模型类:


```
from booktest.models import BookInfo, HeroInfo
```

1) 向 booktest_bookinfo 表中插入一条数据。

```
b = BookInfo() #定义一个 BookInfo 类的对象
b.btitle = '天龙八部' #定义 b 对象的属性并赋值
b.bpub_date = date(1990, 10, 11)
b.save() #才会将数据保存进数据库
```

2) 查询出 booktest_bookinfo 表中 id 为 1 的数据。

```
b = BookInfo.objects.get(id=1)
```

3) 在上一步的基础上改变 b 对应图书的出版日期。

```
b.bpub_date = date(1989, 10, 21)
b.save() #才会更新表格中的数据
```

4) 紧接上一步，删除 b 对应的图书的数据。

```
b.delete() #才会删除
```

5) 向 booktest_heroInfo 表中插入一条数据。

```
h = HeroInfo()
h.hname = '郭靖'
h.hgender = False
h.hcomment = '降龙十八掌'
b2 = BookInfo.objects.get(id=2)
h.hbook = b2 #给关系属性赋值，英雄对象所属的图书对象
h.save()
```

6) 查询图书表里面的所有内容。

```
BookInfo.objects.all()
HeroInfo.objects.all()
```

6.5 关联操作



- 1) 查询出 **id 为 2 的图书**中所有**英雄**人物的信息。

```
b = BookInfo.objects.get(id=2)
```

```
b.heroinfo_set.all() #查询出 b 图书中所有英雄人物的信息
```

7. 后台管理

- 1) 本地化

语言和**时区**的本地化。

修改 `settings.py` 文件。

- 2) 创建管理员

命令: `python manage.py createsuperuser`

- 3) 注册模型类

在应用下的 `admin.py` 中注册模型类。

告诉 django 框架根据注册的模型类来生成对应表管理页面。

```
b = BookInfo()
```

```
str(b) __str__
```

- 4) 自定义管理页面

自定义**模型管理类**。模型管理类就是告诉 django 在生成的管理页面上显示哪些内容。

8. 视图

在 Django 中，通过浏览器去请求一个页面时，使用**视图函数**来处理这个请求的，视图函数处理之后，要给浏览器返回页面内容。

8.1 视图函数的使用

1) 定义视图函数

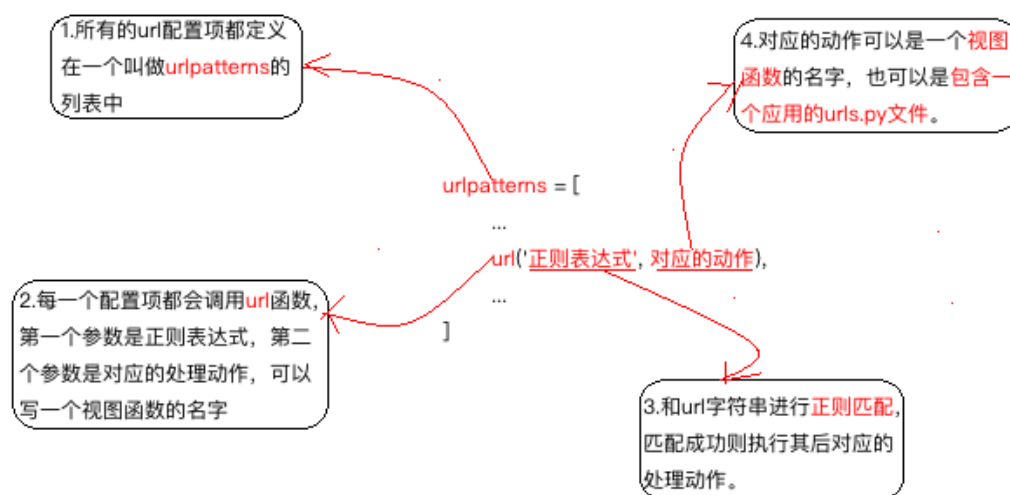
视图函数定义在 `views.py` 中。

例：

```
def index(request):  
    #进行处理...  
    return HttpResponse('hello python')
```

视图函数必须有一个参数 `request`，进行处理之后，需要返回一个 `HttpResponse` 的类对象，`hello python` 就是返回给浏览器显示的内容。

2) 进行 url 配置



url 配置的目的是让建立 url 和视图函数的对应关系。 url 配置项定义在 `urlpatterns` 的列表中，每一个配置项都调用 `url` 函数。

`url` 函数有两个参数，第一个参数是一个正则表达式，第二个是对应的处理动作。

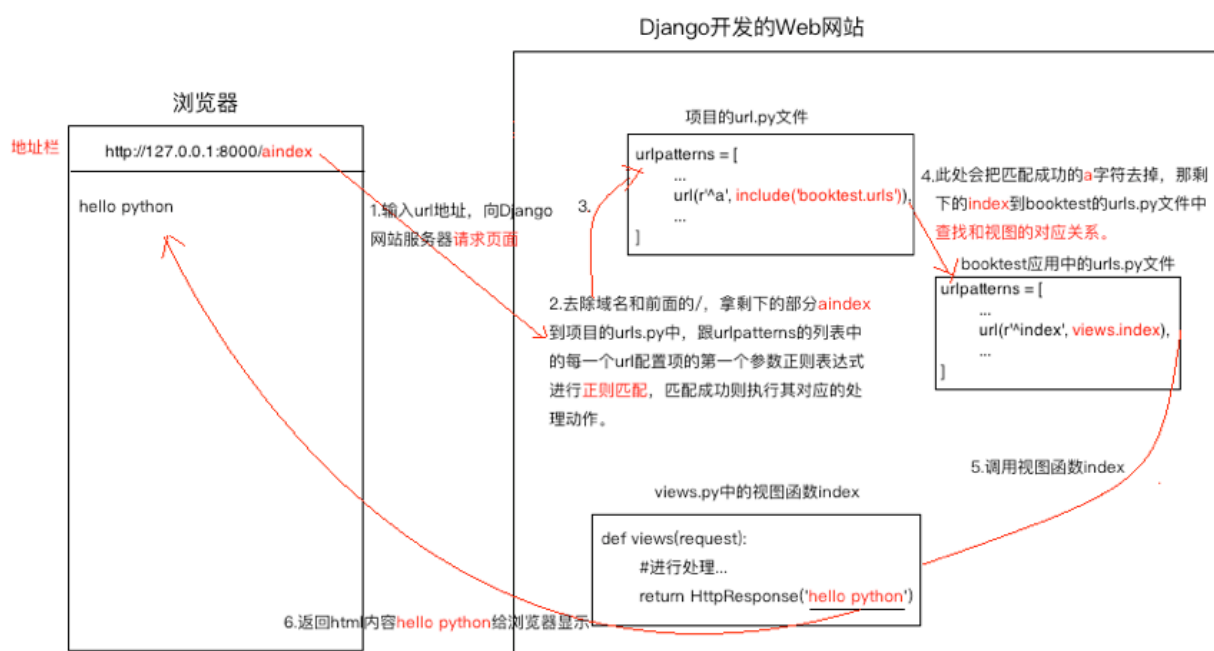
配置 `url` 时，有两种语法格式：

- a) url(正则表达式, 视图函数名)
- b) url(正则表达式, include(应用中的 urls 文件))

工作中在配置 url 时, 首先在项目的 urls.py 文件中添加配置项时, 并不写具体的 url 和视图函数之间的对应关系, 而是包含具体应用的 urls.py 文件, 在应用的 urls.py 文件中写 url 和视图函数的对应关系。

8.2 url 匹配的过程

在项目的 urls.py 文件中包含具体应用的 urls.py 文件, 应用的 urls.py 文件中写 url 和视图函数的对应关系。



当用户输入如 <http://127.0.0.1:8000/aindex> 时, 去除域名和最前面的/, 剩下 aindex, 拿 aindex 字符串到项目的 urls 文件中进行匹配, 配置成功之后, 去除匹配的 a 字符, 那剩下的 index 字符串继续到应用的 urls 文件中进行正则匹配, 匹配成功之后执行视图函数 index, index 视图函数返回内容 hello python 给浏览器来显示。

9. 模板

模板不仅仅是一个 html 文件。

9.1 模板文件的使用

- 1) 创建模板文件夹
- 2) 配置模板目录

```
TEMPLATES = [  
    {  
        'BACKEND': 'django.template.backends.django.DjangoTemplates',  
        # 'DIRS': [],  
        'DIRS': [os.path.join(BASE_DIR, 'templates')], # 设置模板文件目录  
        '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',  
            ],  
        },  
    ],  
]
```

- 3) 使用模板文件

- a) 加载模板文件

去模板目录下面获取 html 文件的内容，得到一个模板对象。

- b) 定义模板上下文

向模板文件传递数据。

- c) 模板渲染

得到一个标准的 html 内容。

9.2 给模板文件传递数据

模板变量使用：{{ 模板变量名 }}

模板代码段：{%代码段%}

for 循环：

{% for i in list %}

list 不为空时执行的逻辑

{% empty %}

list 为空时执行的逻辑

```
{% endfor %}
```

10. 案例完成

编码之前的准备工作：

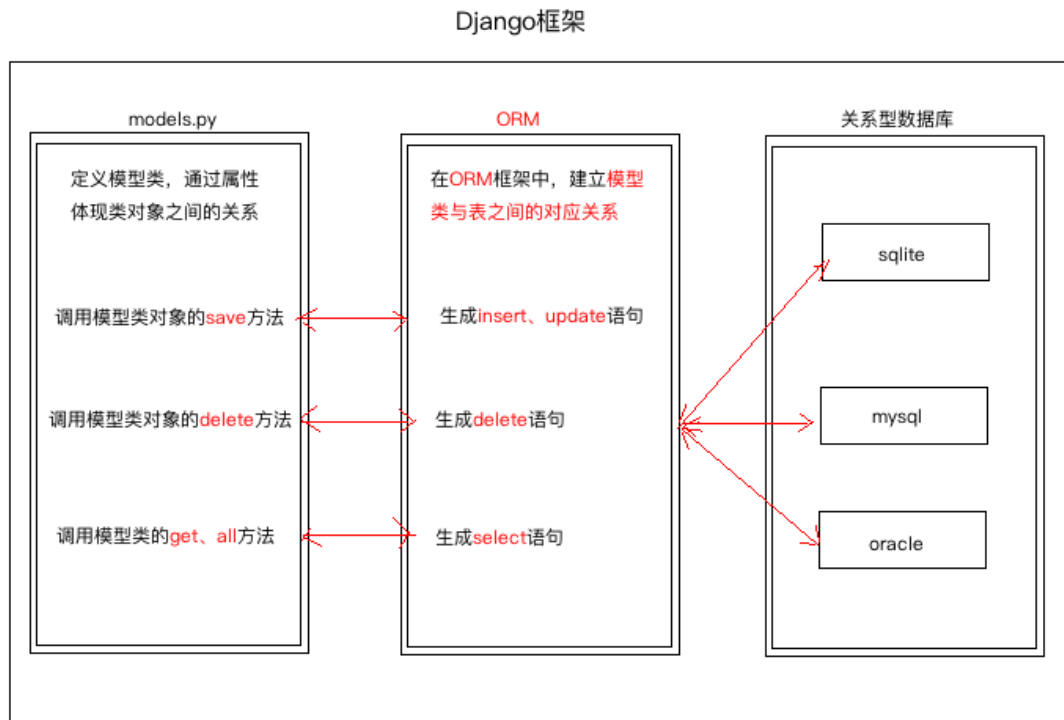
- 1) 设计出访问页面的 url 和对应的视图函数的名字，确定视图函数的功能。
- 2) 设计模板文件的名字。

以下为案例中的简单设计过程：

- 1) 完成图书信息的展示：
 - a) 设计 url，通过浏览器访问 <http://127.0.0.1:8000/books> 时显示图书信息页面。
 - b) 设计 url 对应的视图函数 show_books。
查询出所有图书的信息，将这些信息传递给模板文件。
 - c) 编写模板文件 show_books.html。
遍历显示出每一本图书的信息。
- 2) 完成点击某本图书时，显示出图书里所有英雄信息的页面。
 - a) 设计 url，通过访问 <http://127.0.0.1:8000/books/数字> 时显示对应的英雄信息页面。
这里数字指点击的图书的 id。
 - b) 设计对应的视图函数 detail。
接收图书的 id，根据 id 查询出相应的图书信息，然后查询出图书中的所有英雄信息。
 - c) 编写模板文件 detail.html。

模型

1. Django ORM



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:

- 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。
- 根据设计的模型类生成数据库中的表格。
- 通过方便的配置就可以进行数据库的切换。

2. Django 数据库配置

2.1 mysql 命令回顾

登录 mysql 数据库: `mysql -uroot -p`

查看有哪些数据库: `show databases`

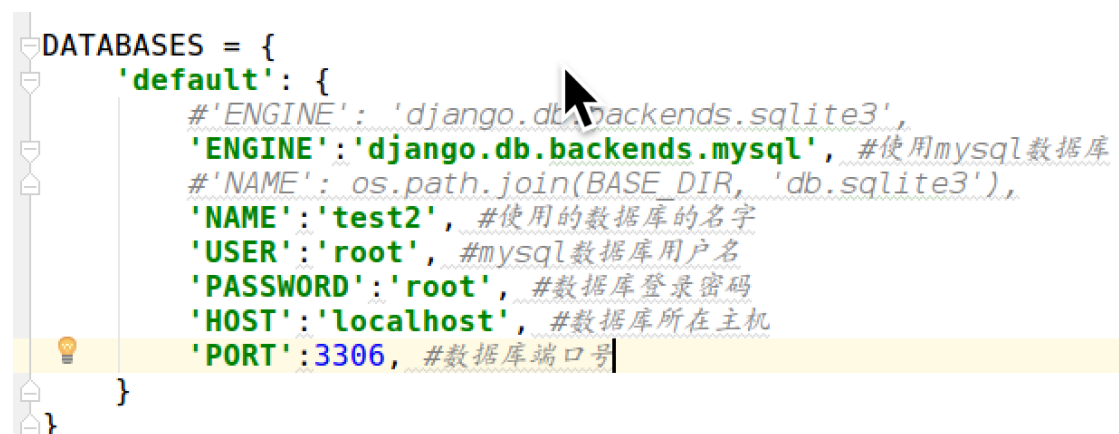
创建数据库: `create database test2 charset=utf8; #切记:指定编码`

使用数据库: `use test2;`

查看数据库中的表: `show tables;`

2.2 Django 配置使用 mysql 数据库

修改 settings.py 中的 DATABASES。



```
DATABASES = {  
    'default': {  
        # 'ENGINE': 'django.db.backends.sqlite3',  
        'ENGINE': 'django.db.backends.mysql', # 使用mysql数据库  
        # 'NAME': os.path.join(BASE_DIR, 'db.sqlite3'),  
        'NAME': 'test2', # 使用的数据库的名字  
        'USER': 'root', # mysql数据库用户名  
        'PASSWORD': 'root', # 数据库登录密码  
        'HOST': 'localhost', # 数据库所在主机  
        'PORT': 3306, # 数据库端口号  
    }  
}
```

注意: django 框架不会自动帮我们生成 mysql 数据库, 所以我们需要自己去创建。

2.3 切换 mysql 数据库之后不能启动服务器

需要安装操作 mysql 数据库的包, python2 环境和 python3 环境有以下区别。

a) python2 需要安装 mysql-python:

```
pip install mysql-python
```

b) python3 需要安装 pymysql:

```
pip install pymysql
```


python3 中安装好 pymysql, 需要在 test2/__init__.py 中加如下内容:

```
import pymysql  
  
pymysql.install_as_MySQLdb()
```

3. 复习案例

1) 设计模型类并生成表

- a) 设计 BookInfo, 增加属性 bread 和 bcomment, 另外设置软删除标记属性 isDelete。
- b) 设计 HeroInfo 类, 增加软删除标记属性 isDelete。

软删除标记: 删除数据时不做真正的删除, 而是把标记数据设置为 1 表示删除, 目的是防止重要的数据丢失。

2) 编写视图函数并配置 URL。

3) 创建模板文件。

拆解功能:

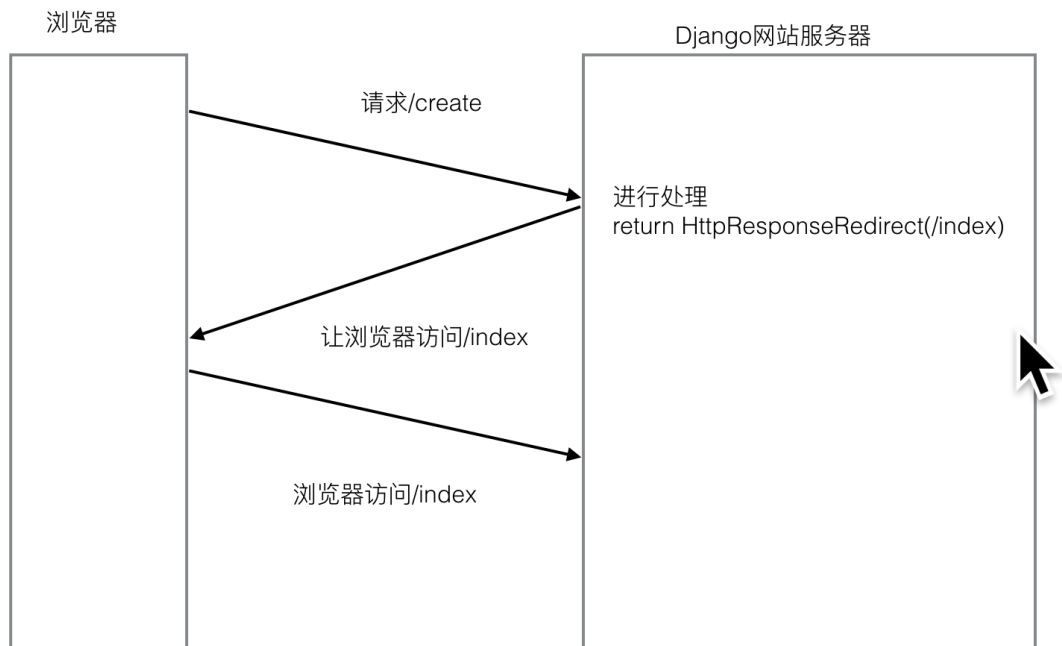
1) 图书信息展示页。

- a) 设计 url, 通过浏览器访问 <http://127.0.0.1:8000/index> 时显示图书信息页面。
- b) 设计 url 对应的视图函数 index。
查询出所有图书的信息, 将这些信息传递给模板文件。
- c) 编写模板文件 index.html。
遍历显示出每一本图书的信息并增加新建和删除超链接。

2) 图书信息新增。

- a) 设计 url, 通过浏览器访问 <http://127.0.0.1:8000/create> 时向数据库中新增一条图书信息。
- b) 设计 url 对应得视图函数 create。

重定向



页面重定向: 服务器不返回页面, 而是告诉浏览器再去请求其他的 url 地址。

3) 图书信息删除。

a) 设计 url, 通过浏览器访问 <http://127.0.0.1:8000/delete> 数字删除数据库中对应的一条图书数据。

其中数字是点击的图书的 id。

b) 设计 url 对应的视图函数 delete。

获取图书的 id, 进行删除。

4. 字段属性和选项

4.1 模型类属性命名限制

1) 不能是 python 的保留关键字。

2) 不允许使用连续的下划线, 这是由 django 的查询方式决定的。

3) 定义属性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)

4.2 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
<code>AutoField</code>	自动增长的 <code>IntegerField</code> ，通常不用指定，不指定时 Django 会自动创建属性名为 <code>id</code> 的自动增长属性。
<code>BooleanField</code>	布尔字段，值为 <code>True</code> 或 <code>False</code> 。
<code>NullBooleanField</code>	支持 <code>Null</code> 、 <code>True</code> 、 <code>False</code> 三种值。
<code>CharField(max_length=最大长度)</code>	字符串。参数 <code>max_length</code> 表示最大字符个数。
<code>TextField</code>	大文本字段，一般超过 4000 个字符时使用。
<code>IntegerField</code>	整数
<code>DecimalField(max_digits=None, decimal_places=None)</code>	十进制浮点数。参数 <code>max_digits</code> 表示总位。参数 <code>decimal_places</code> 表示小数位数。
<code>FloatField</code>	浮点数。参数同上
<code>DateField : ([auto_now=False, auto_now_add=False])</code>	<p>日期。</p> <p>1) 参数 <code>auto_now</code> 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 <code>false</code>。</p> <p>2) 参数 <code>auto_now_add</code> 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 <code>false</code>。</p> <p>3) 参数 <code>auto_now_add</code> 和 <code>auto_now</code> 是</p>

	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。

4.3 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，则该字段允许为空白，默认值是 False。

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

5. 查询

5.1 mysql 的日志文件

mysql.log 是 mysql 的日志文件，里面记录的对 MySQL 数据库的操作记录。
默认情况下 mysql 的日志文件没有产生，需要修改 mysql 的配置文件，步骤如下：

1) 使用下面的命令打开 mysql 的配置文件，去除 68,69 行的注释，然后保存。

```
sudo vi /etc/mysql/mysql.conf.d/mysqld.cnf
```

2) 重启 mysql 服务，就会产生 mysql 日志文件。

```
sudo service mysql restart
```

3) 打开 MySQL 的日志文件。

/var/log/mysql/mysql.log 是 mysql 日志文件所在的位置。

使用下面的命令可以实时查看 mysql 的日志文件：

```
sudo tail -f /var/log/mysql/mysql.log
```

5.2 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	返回值是	查询集

	对应表格中的所有数据。	QuerySet 类型	
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。

get 示例:

例: 查询图书 id 为 3 的图书信息。

all 方法示例:

例: 查询图书所有信息。

filter 方法示例:

条件格式:

模型类属性名__条件名=值

查询图书评论量为 34 的图书的信息:

a) 判等 条件名: **exact**。

例: 查询编号为 1 的图书。

BookInfo.objects.get(id=1)

b) 模糊查询

例: 查询书名包含'传'的图书。contains

BookInfo.objects.filter(btitle__contains='传')

例: 查询书名以'部'结尾的图书 endswith 开头:startswith

BookInfo.objects.filter(btitle__endswith='部')

c) 空查询 isnull

例：查询书名不为空的图书。`isnull`

```
select * from booktest_bookinfo where btitle is not null;
```

```
BookInfo.objects.filter(btitle__isnull=False)
```

d) 范围查询 `in`

例：查询 id 为 1 或 3 或 5 的图书。

```
select * from booktest_bookinfo where id in (1,3,5);
```

```
BookInfo.objects.filter(id__in = [1,3,5])
```

e) 比较查询 `gt`(greater than) `lt`(less than) `gte`(equal) 大于等于
`lte` 小于等于

例：查询 id 大于 3 的图书。

```
Select * from booktest_bookinfo where id>3;
```

```
BookInfo.objects.filter(id__gt=3)
```

f) 日期查询

例：查询 1980 年发表的图书。

```
BookInfo.objects.filter(bpub_date__year=1980)
```

例：查询 1980 年 1 月 1 日后发表的图书。

```
from datetime import date
```

```
BookInfo.objects.filter(bpub_date__gt=date(1980,1,1))
```

exclude 方法示例：

例：查询 id 不为 3 的图书信息。

```
BookInfo.objects.exclude(id=3)
```

order_by 方法示例：

作用：进行查询结果进行排序。

例：查询所有图书的信息，按照 id 从小到大进行排序。

```
BookInfo.objects.all().order_by('id')
```

例：查询所有图书的信息，按照 id 从大到小进行排序。

```
BookInfo.objects.all().order_by('-id')
```

例：把 id 大于 3 的图书信息按阅读量从大到小排序显示。

```
BookInfo.objects.filter(id__gt=3).order_by('-bread')
```

6. F 对象

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
```

例：查询图书阅读量大于评论量图书信息。

```
BookInfo.objects.filter(bread__gt=F('bcomment'))
```

例：查询图书阅读量大于 2 倍评论量图书信息。

```
BookInfo.objects.filter(bread__gt=F('bcomment')*2)
```

7. Q 对象

作用：用于查询时条件之间的逻辑关系。not and or，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
```

例：查询 id 大于 3 且阅读量大于 30 的图书的信息。

```
BookInfo.objects.filter(id__gt=3, bread__gt=30)
```

```
BookInfo.objects.filter(Q(id__gt=3)&Q(bread__gt=30))
```

例：查询 id 大于 3 或者阅读量大于 30 的图书的信息。

```
BookInfo.objects.filter(Q(id__gt=3)|Q(bread__gt=30))
```

例：查询 id 不等于 3 图书的信息。

```
BookInfo.objects.filter(~Q(id=3))
```

8. 聚合函数

作用：对查询结果进行聚合操作。

```
sum count avg max min
```

aggregate：调用这个函数来使用聚合。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
```


例：查询所有图书的数目。

```
BookInfo.objects.all().aggregate(Count('id'))  
{'id__count': 5}
```

例：查询所有图书阅读量的总和。

```
BookInfo.objects.aggregate(Sum('bread'))  
{'bread__sum': 126}
```

count 函数 返回值是一个数字

作用：统计满足条件数据的数目。

例：统计所有图书的数目。

```
BookInfo.objects.all().count()  
BookInfo.objects.count()
```

例：统计 id 大于 3 的所有图书的数目。

```
BookInfo.objects.filter(id__gt=3).count()
```

小结：

查询相关函数

get:返回一条且只能有一条数据，返回值是一个对象，参数可以写查询条件。
all:返回模型类对应表的所有数据，返回值是QuerySet。
filter:返回满足条件的数据，返回值是QuerySet，参数可以写查询条件。
exclude:返回不满足条件的数据，返回值是QuerySet，参数可以写查询条件。
order_by:对查询结果进行排序，返回值是QuerySet，参数中写排序的字段。

注意：

1. **get, filter, exclude**函数中可以写查询条件，如果传多个参数，条件之间代表且的关系。
2. **all, filter, exclude, order_by**函数的返回值是**QuerySet**类的实例对象，叫做查询集。

```
from django.db.models import F,Q,Sum,Count,Avg,Max,Min
```

F对象: 用于类属性之间的比较。

Q对象: 用于条件之间的逻辑关系。

aggregate:进行聚合操作，返回值是一个字典，进行聚合的时候需要先导入聚合类。

count:返回结果集中数据的数目，返回值是一个数字。

注意：

对一个**QuerySet**实例对象，可以继续调用上面的所有函数。

参考文档：

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

9. 查询集

`all`, `filter`, `exclude`, `order_by` 调用这些函数会产生一个查询集, `QuerySet` 类对象可以继续调用上面的所有函数。

9.1 查询集特性

- 1) 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。
- 2) 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。

9.2 限制查询集

可以对一个查询集进行取下标或者切片操作来限制查询集的结果。

对一个查询集进行切片操作会产生一个新的查询集, 下标不允许为负数。

取出查询集第一条数据的两种方式:

方式	说明
<code>b[0]</code>	如果 <code>b[0]</code> 不存在, 会抛出 <code>IndexError</code> 异常
<code>b[0:1].get()</code>	如果 <code>b[0:1].get()</code> 不存在, 会抛出 <code>DoesNotExist</code> 异常。

`exists`: 判断一个查询集中是否有数据。True False

10. 模型类关系

1) 一对多关系

例: 图书类-英雄类

`models.ForeignKey()` 定义在多的类中。

2) 多对多关系

例：新闻类-新闻类型类 体育新闻 国际新闻

`models.ManyToManyField()` 定义在哪个类中都可以。

3) 一对一关系

例：员工基本信息类-员工详细信息类. 员工工号

`models.OneToOneField` 定义在哪个类中都可以。

11. 关联查询（一对多）

11.1 查询和对象关联的数据

在一对多关系中，一对应的类我们把它叫做**一类**，多对应的那个类我们把它叫做**多类**，我们把多类中定义的建立关联的类属性叫做**关联属性**。

例：查询 **id 为 1 的图书**关联的英雄的信息。

```
b=BookInfo.objects.get(id=1)
```

```
b.heroinfo_set.all()
```

通过模型类查询：

```
HeroInfo.objects.filter(hbook__id=1)
```

例：查询 **id 为 1 的英雄**关联的图书信息。

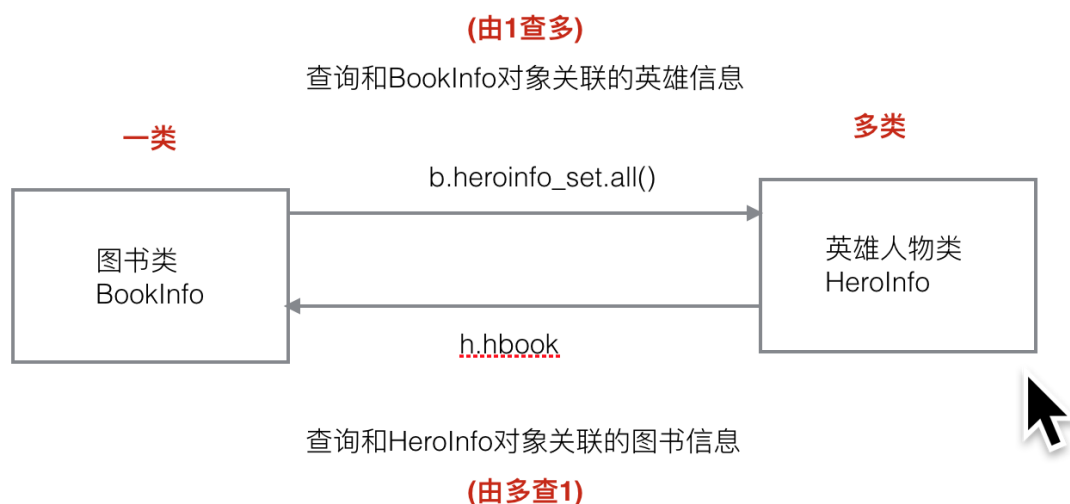
```
h = HeroInfo.objects.get(id=1)
```

```
h.hbook
```

通过模型类查询：

```
BookInfo.objects.filter(heroinfo__id=1)
```

格式：



由一类的对象查询多类的时候：

一类的对象. 多类名小写_set.all() #查询所用数据

由多类的对象查询一类的时候：

多类的对象. 关联属性 #查询多类的对象对应的一类的对象

由多类的对象查询一类对象的 id 时候：

多类的对象. 关联属性_id

11.2 通过模型类实现关联查询

关联查询

注意：

1. 通过模型类实现关联查询时，要查哪个表中的数据，就需要通过哪个类来查。
2. 写关联查询条件的时候，如果类中没有关系属性，条件需要些对应类的名，如果类中有关系属性，直接写关系属性。

例：查询图书信息，要求图书关联的英雄的描述包含‘八’。

```
BookInfo.objects.filter(heroinfo__hcomment__contains='八')
```

例：查询图书信息，要求图书中的英雄的 id 大于 3。

```
BookInfo.objects.filter(heroinfo__id__gt=3)
```

例：查询书名为“天龙八部”的所有英雄。

```
HeroInfo.objects.filter(hbook__btitle='天龙八部')
```

通过多类的条件查询一类的数据：

一类名.objects.filter(多类名小写__多类属性名__条件名)

通过一类的条件查询多类的数据：

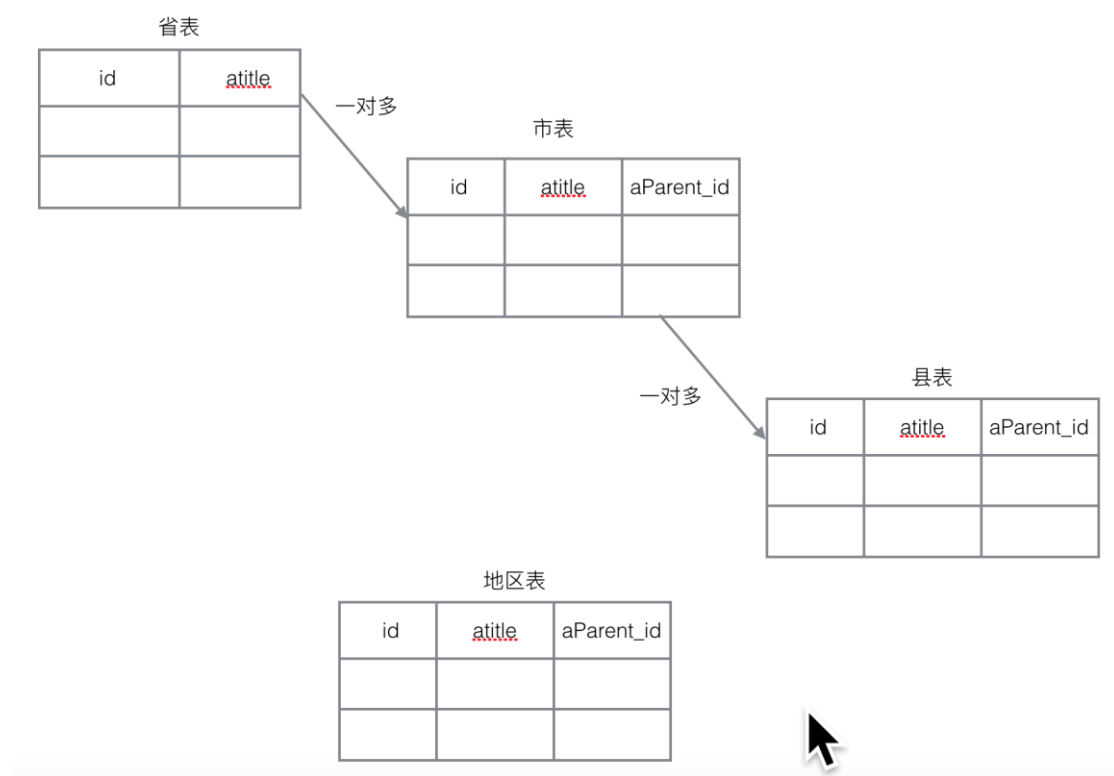
多类名.objects.filter(关联属性__一类属性名__条件名)

12. 插入、更新和删除

调用一个模型类对象的 save 方法的时候就可以实现对模型类对应数据表的插入和更新。

调用一个模型类对象的 delete 方法的时候就可以实现对模型类对应数据表数据的删除。

13. 自关联



自关联是一种特殊的一对多的关系。

案例：显示广州市的上级地区和下级地区。

地区表：id, atitle, aParent_id;

mysql 终端中批量执行 sql 语句：source areas.sql;

14. 管理器

BookInfo.objects.all()->objects 是一个什么东西呢？

答：objects 是 Django 帮我自动生成的管理器对象，通过这个管理器可以实现对数据的查询。

objects 是 models.Manager 类的一个对象。自定义管理器之后 Django 不再帮我们生成默认的 objects 管理器。

14.1 自定义模型管理器类

- 1) 自定义一个管理器类，这个类继承 models.Manager 类。
- 2) 再在具体的模型类里定义一个自定义管理器类的对象。

14.2 自定义管理器类的应用场景

- 1) 改变查询的结果集。

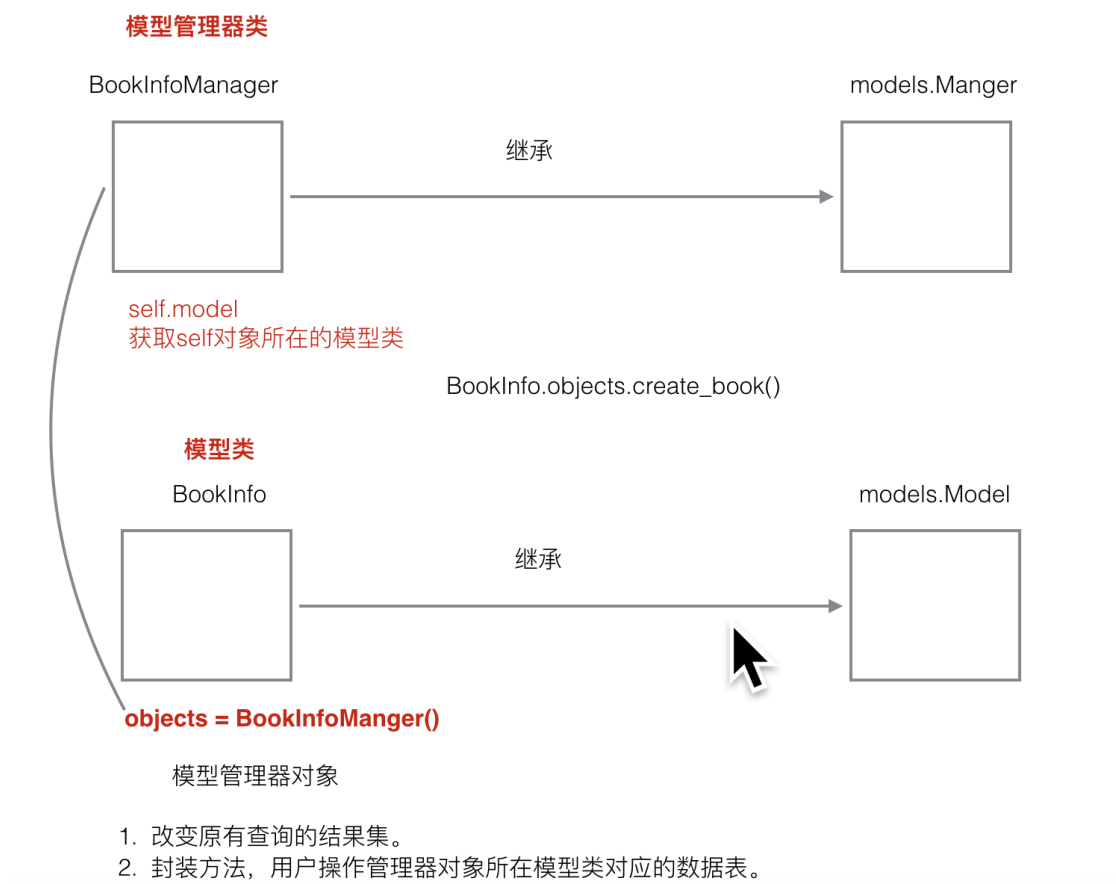
比如调用 BookInfo.books.all() 返回的是没有删除的图书的数据。

- 2) 添加额外的方法。

管理器类中定义一个方法帮我们操作模型类对应的数据表。

使用 `self.model()` 就可以创建一个跟自定义管理器对应的模型类对象。

小结：



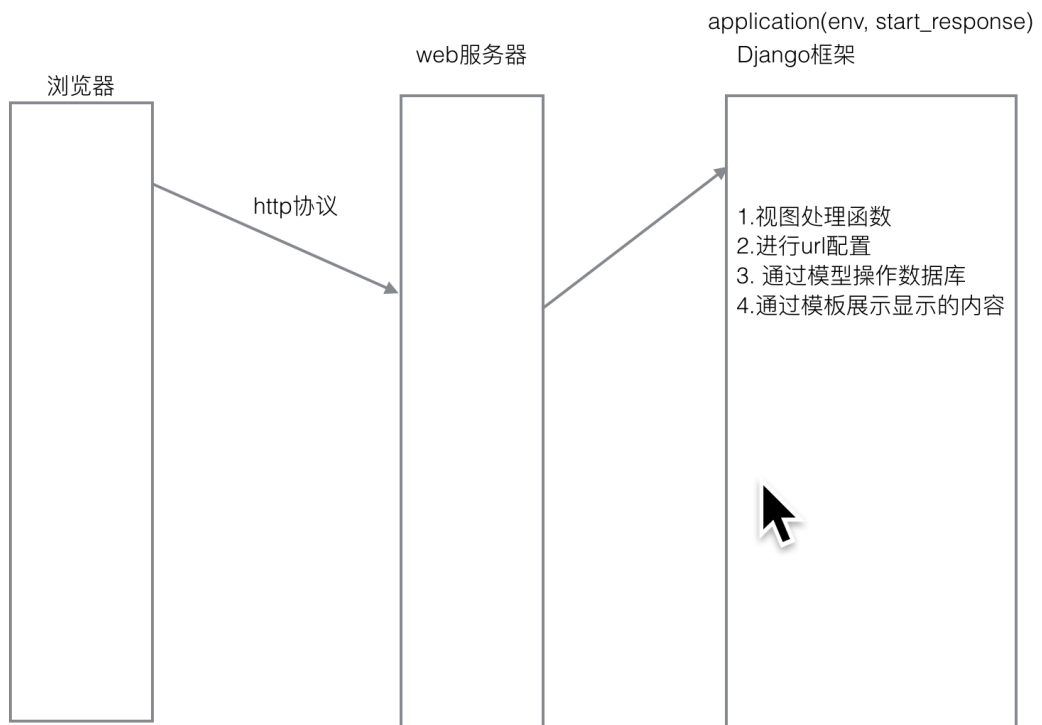
15. 元选项

Django 默认生成的表名：

应用名小写_模型类名小写。

元选项：

需要在模型类中定义一个元类 Meta, 在里面定义一个类属性 `db_table` 就可以指定表名。



视图

1. 视图的功能

接收请求，进行处理，与 M 和 T 进行交互，返回应答。

返回 html 内容 `HttpResponse`，也可能重定向 `redirect`，还可以返回 json 数据。

2. 视图函数使用

2.1 使用

1) 定义视图函数

`request` 参数必须有。是一个 `HttpRequest` 类型的对象。参数名可以变化，但不要更改。

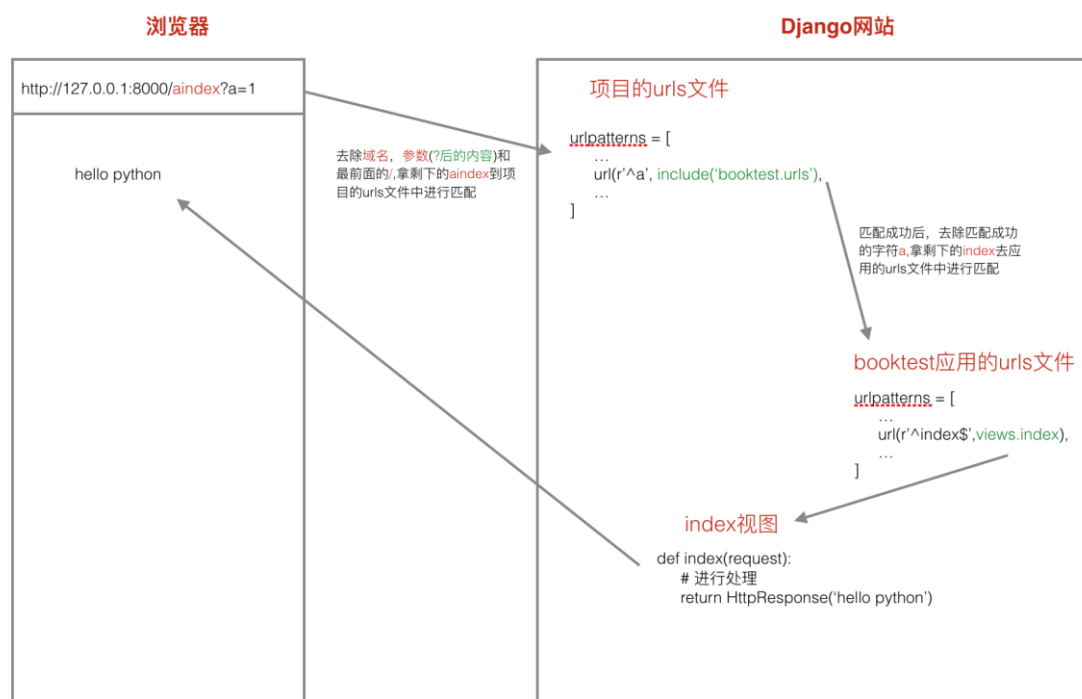
2) 配置 url

建立 url 和视图函数之间的对应关系。

2.2 url 配置的过程

- 1) 在项目的 urls 文件中包含具体应用的 urls 文件，在具体应用的 urls 文件中包含具体 url 和视图的对应关系。
- 2) url 配置项是定义在一个名叫 `urlpatterns` 的列表中，其中的每一个元素就是一个配置项，每一个配置项都调用 `url` 函数。

3. url 匹配的过程



url: `http://127.0.0.1:8000/aindex?a=1`

- 1) 去除域名和后面的参数，剩下 `/aindex`，再把前面的 `/` 去掉，剩下 `aindex`
- 2) 拿 `aindex` 先到项目的 `url.py` 文件中进行从上到下的匹配，匹配成功之后执行后面对应的处理动作，就是把匹配成功的一部分 `a` 字符去除，然后拿剩下的部分 `index` 到应用的 `urls.py` 文件中再进行从上到下的匹配。
- 3) 如果匹配成功则调用相应的视图产生内容返回给客户端。如果匹配失败则产生 404 错误。

4. 错误视图

404: 找不到页面, 关闭调试模式之后, 默认会显示一个标准的错误页面, 如果要显示自定义的页面, 则需要的 templates 目录下面自定义一个 404.html 文件。

a) url 没有配置

b) url 配置错误

500: 服务器端的错误。

a) 视图出错

网站开发完成需要关闭调试模式, 在 settings.py 文件中:

```
DEBUG=False  
ALLOWED_HOST=[ '*' ]
```

5. 捕获 url 参数

进行 url 匹配时, 把所需要的捕获的部分设置成一个正则表达式组, 这样 django 框架就会自动把匹配成功后相应组的内容作为参数传递给视图函数。

1) 位置参数

位置参数, 参数名可以随意指定()

2) 关键字参数: 在位置参数的基础上给正则表达式组命名即可。

?P<组名>

关键字参数, 视图中参数名必须和正则表达式组名一致。

6. 普通登录案例

1) 显示出登录页面

a) 设计 url, 通过浏览器访问 <http://127.0.0.1:8000/login> 时显示登录页面。

b) 设计 url 对应的视图函数 login。

c) 编写模板文件 login.html。

url	视图	模板文件
-----	----	------

/login	login	login.html
--------	-------	------------

2) 登录校验功能

a) 设计 url, 点击登录页的登录按钮发起请求
http://127.0.0.1:8000/login_check 时进行登录校验。

b) 设计 url 对应的视图函数 login_check。

接收表单提交过来的数据。

进行登录校验, 若用户名密码正确则跳转到登录成功页。若失败在跳转到登录页面。重定向

c) 登录成功后跳转到首页。

url	视图	模板文件
/login_check	login_check	无

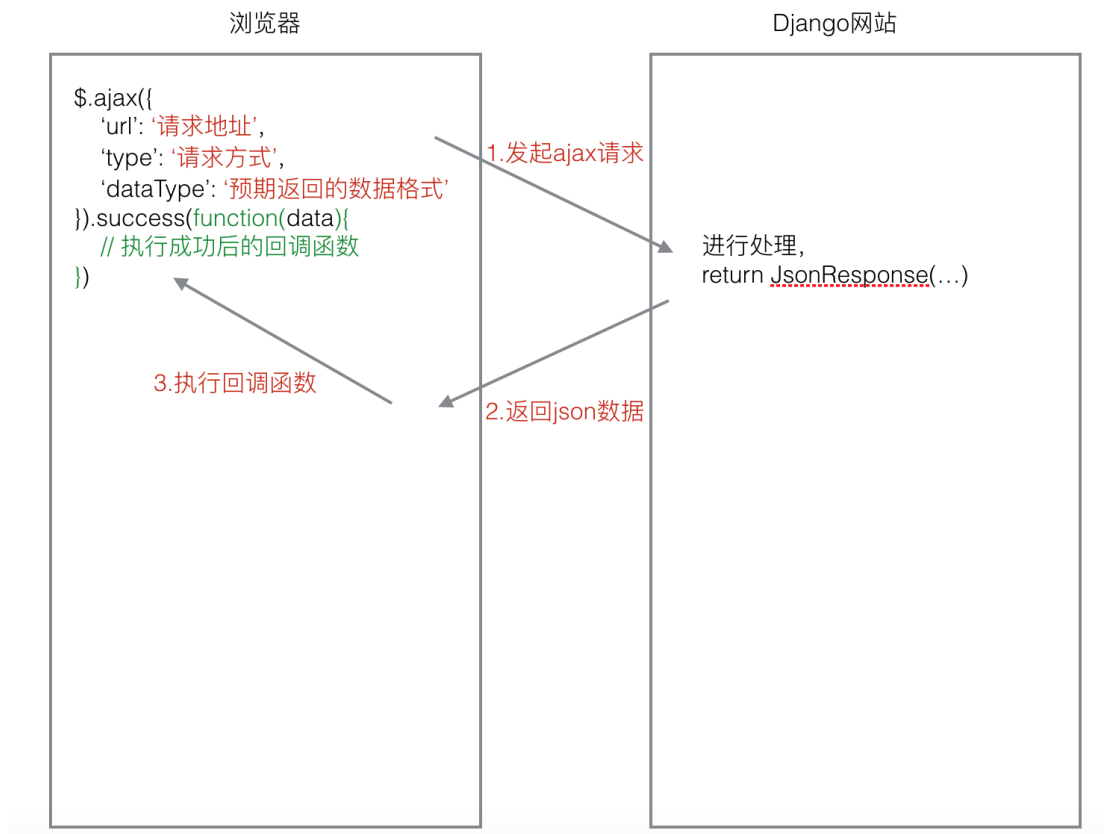
7. Ajax

7.1 基本概念

异步的 javascript。在不全部加载某一个页面部的情况下, 对页面进行局的刷新, ajax 请求都在后台。

图片, css 文件, js 文件都是静态文件。

ajax请求



- 1) 发起 ajax 请求：jquery 发起
- 2) 执行相应的视图函数，返回 json 内容
- 3) 执行相应的回调函数。通过判断 json 内容，进行相应处理。

7.2 Ajax 登录案例

- 1) 首先分析出请求地址时需要携带的参数, 请求方式。
- 2) 视图函数处理完成之后，所返回的 json 的格式。

1) 显示出登录页面

- a) 设计 url，通过浏览器访问 http://127.0.0.1:8000/login_ajax 时显示 [登录页面](#)。
- b) 设计 url 对应的视图函数 login_ajax。
- c) 编写模板文件 login_ajax.html。

在里面写 jquery 代码发起 ajax 请求。

2) 登录校验功能

a) 设计 url, 点击登录页的登录按钮发起请求
http://127.0.0.1:8000/login_ajax_check 时进行登录校验。

b) 设计 url 对应的视图函数 login_ajax_check。

接收 post 提交过来的数据。

进行登录校验, 并返回 json 内容。 **JsonRepsone**

Json 格式如下:

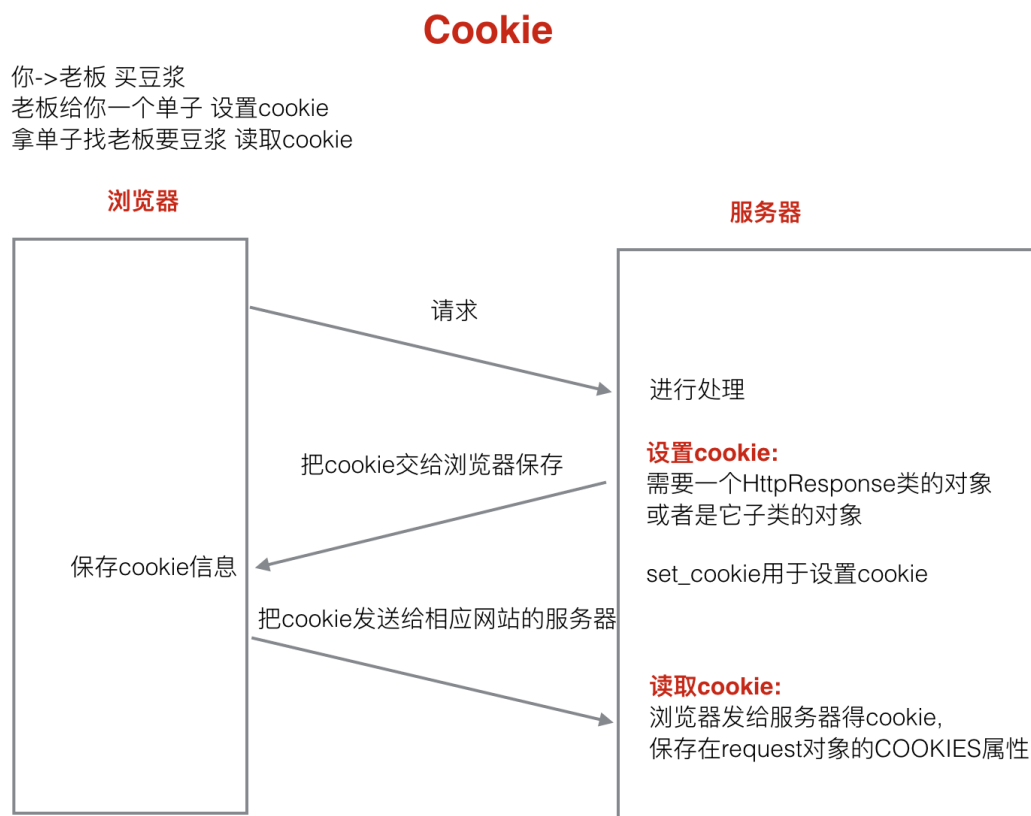
{ 'res': '1' } #表示登录成功

{ 'res': '0' } #表示登录失败

8. 状态保持

http 协议是无状态的。下一次去访问一个页面时并不知道上一次对这个页面做了什么。

8.1 Cookie



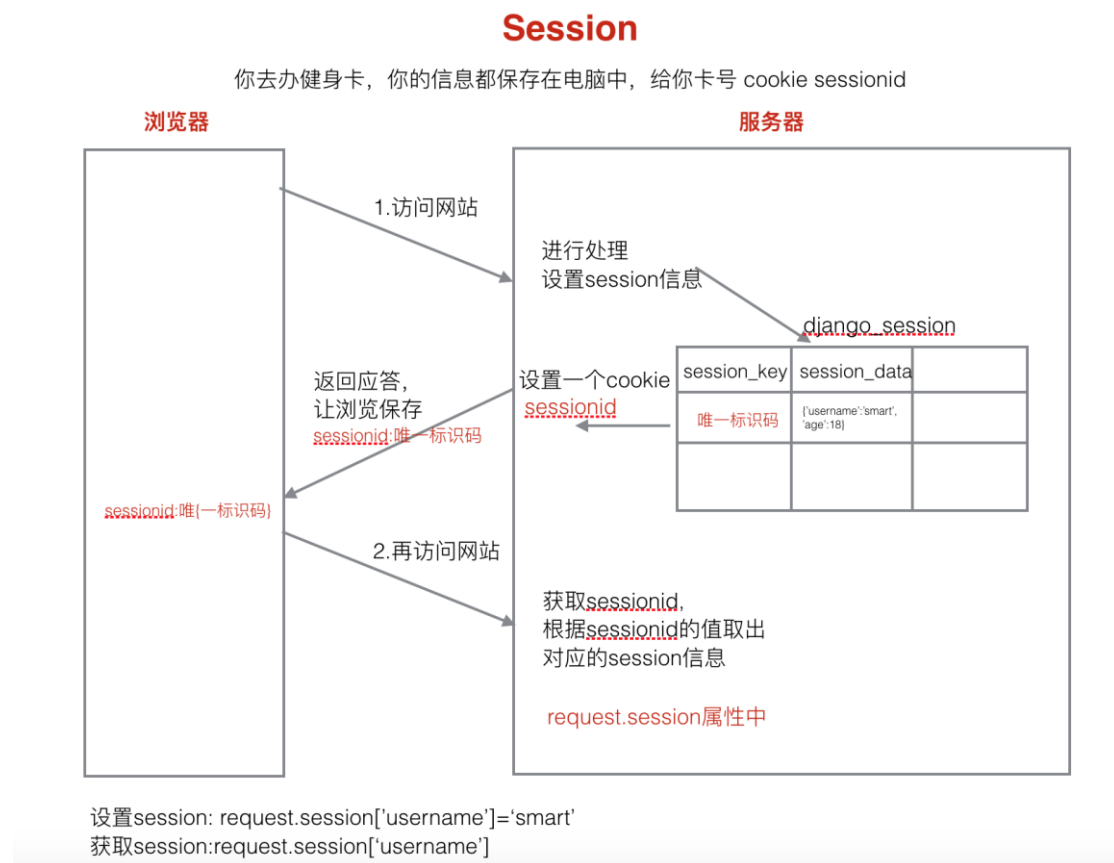
cookie 是由服务器生成, 存储在浏览器端的一小段文本信息。

cookie 的特点:

- 1) 以键值对方式进行存储。
- 2) 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
- 3) cookie 是基于域名安全的。www.baidu.com www.tudou.com
- 4) 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。

记住用户名案例。

8.2 Session



session 存储在**服务器端**。

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。

记住用户登录状态案例。

8.3 cookie 和 session 的应用场景

cookie: 记住用户名。安全性要求不高。

session: 涉及到安全性要求比较高的数据。银行卡账户, 密码

模板

1. 模板的功能

产生 html, 控制页面上展示的内容。模板文件不仅仅是一个 html 文件。

模板文件包含两部分内容:

- 1) 静态内容: css, js, html。
- 2) 动态内容: 用于动态去产生一些网页内容。通过模板语言来产生。

2. 模板文件的使用

通常是在视图函数中使用模板产生 html 内容返回给客户端。

- a) 加载模板文件 loader.get_template

获取模板文件的内容, 产生一个模板对象。

- b) 定义模板上下文 RequesContext

给模板文件传递数据。

- c) 模板渲染产生 html 页面内容 render

用传递的数据替换相应的变量, 产生一个替换后的标准的 html 内容。

3. 模板文件加载顺序

Template-loader postmortem

Django tried loading these templates, in this order:

- Using loader django.template.loaders.filesystem.Loader:
 - /home/python/bj09/test4/templates/booktest/index3.html (File does not exist)
- Using loader django.template.loaders.app_directories.Loader:
 - /home/python/.virtualenvs/bj09_py3/lib/python3.5/site-packages/django/contrib/admin/templates/booktest/index3.html (File does not exist)
 - /home/python/.virtualenvs/bj09_py3/lib/python3.5/site-packages/django/contrib/auth/templates/booktest/index3.html (File does not exist)
 - /home/python/bj09/test4/booktest/templates/booktest/index3.html (File does not exist)

- 1) 首先去配置的模板目录下面去找模板文件。
- 2) 去 INSTALLED_APPS 下面的每个应用的 templates 去找模板文件,前提是应用中必须有 templates 文件夹。

4. 模板语言

模板语言简称为 DTL。(Django Template Language)

4.1 模板变量

模板变量名是由数字, 字母, 下划线和点组成的, 不能以下划线开头。

使用模板变量: `{{模板变量名}}`

模板变量的解析顺序:

例如: `{{ book.btitle }}`

- 1) 首先把 book 当成一个字典, 把 btitle 当成键名, 进行取值 `book['btitle']`
- 2) 把 book 当成一个对象, 把 btitle 当成属性, 进行取值 `book.btitle`
- 3) 把 book 当成一个对象, 把 btitle 当成对象的方法, 进行取值 `book.btitle`

例如: `{{book.0}}`

- 1) 首先把 book 当成一个字典, 把 0 当成键名, 进行取值 `book[0]`
- 2) 把 book 当成一个列表, 把 0 当成下标, 进行取值 `book[0]`

如果解析失败, 则产生内容时用空字符串填充模板变量。

使用模板变量时, . 前面的可能是一个字典, 可能是一个对象, 还可能是一个列表。

4.2 模板标签

`{% 代码段 %}`

for 循环:

`{% for x in 列表 %}`

`# 列表不为空时执行`

`{% empty %}`

列表为空时执行

{% endfor %}

可以通过 `{{ forloop.counter }}` 得到 for 循环遍历到了第几次。

{% if 条件 %}

{% elif 条件 %}

{% else %}

{% endif %}

关系比较操作符：> < >= <= == !=

注意：进行比较操作时，比较操作符两边必须有空格。

逻辑运算：not and or

4.3 过滤器

过滤器用于对模板变量进行操作。

date: 改变日期的显示格式。

length: 求长度。字符串，列表。

default: 设置模板变量的默认值。

格式：模板变量 | 过滤器：参数

自定义过滤器。

自定义的过滤器函数，至少有一个参数，最多两个

参考资料：(模板标签和内置过滤器)

http://python.usyiyi.cn/documents/django_182/ref/templates/builtins.html

4.4 模板注释

单行注释：{# 注释内容 #}

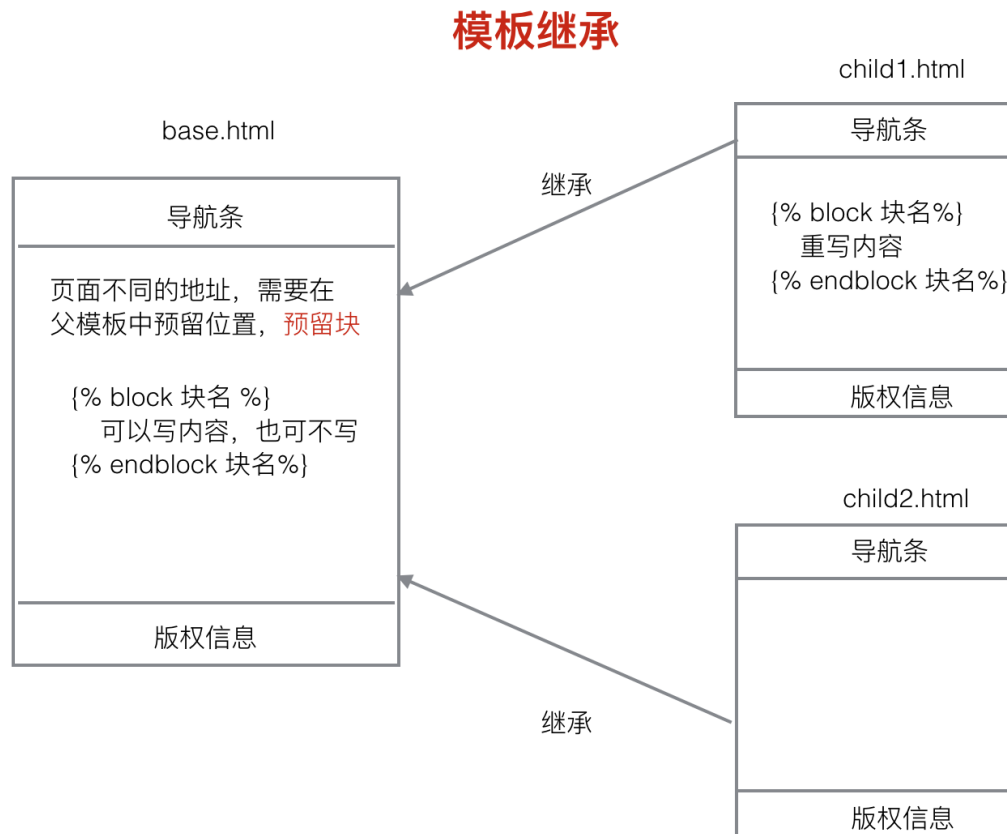
多行注释：{% comment %}

注释内容

{% endcomment %}

5. 模板继承

模板继承也是为了重用 html 页面内容。



把所有页面相同的内容放到父模板文件中，不需要放在块中。
有些位置页面内容不同，需要在父模板中预留块。

在父模板里可以定义块，使用标签：

```
{% block 块名 %}
```

块中间可以写内容，也可以不写

```
{% endblock 块名 %}
```

子模板去继承父模板之后，可以重写父模板中的某一块的内容。

继承格式：`{% extends 父模板文件路径 %}`

```
{% block 块名 %}
```

```
{{ block.super }} #获取父模板中块的默认内容
```

重写的内容

```
{% endblock 块名 %}
```

6. html 转义

编辑商品详情信息，数据表中保存的是 html 内容。

在模板上下文中的 html 标记默认是会被转义的。

```
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;
```

要关闭模板上下文字符串的转义：可以使用 `{{ 模板变量|safe }}`

也可以使用：

```
{% autoescape off %}
    模板语言代码
{% endautoescape %}
```

模板硬编码中的字符串默认不会经过转义，如果需要转义，那需要手动进行转义。

7. csrf 攻击

首先做一个登录页，让用户输入用户名和密码进行登录，登录成功之后跳转的修改密码页面。在修改密码页面输入新密码，点击确认按钮完成密码修改。

登录页需要一个模板文件 `login.html`。修改密码页面也需要一个模板文件 `change_pwd.html`。

显示登录页的视图 `login`，验证登录的视图 `login_check`，显示发帖页的视图 `change_pwd`，处理修改密码的视图 `change_pwd_action`。

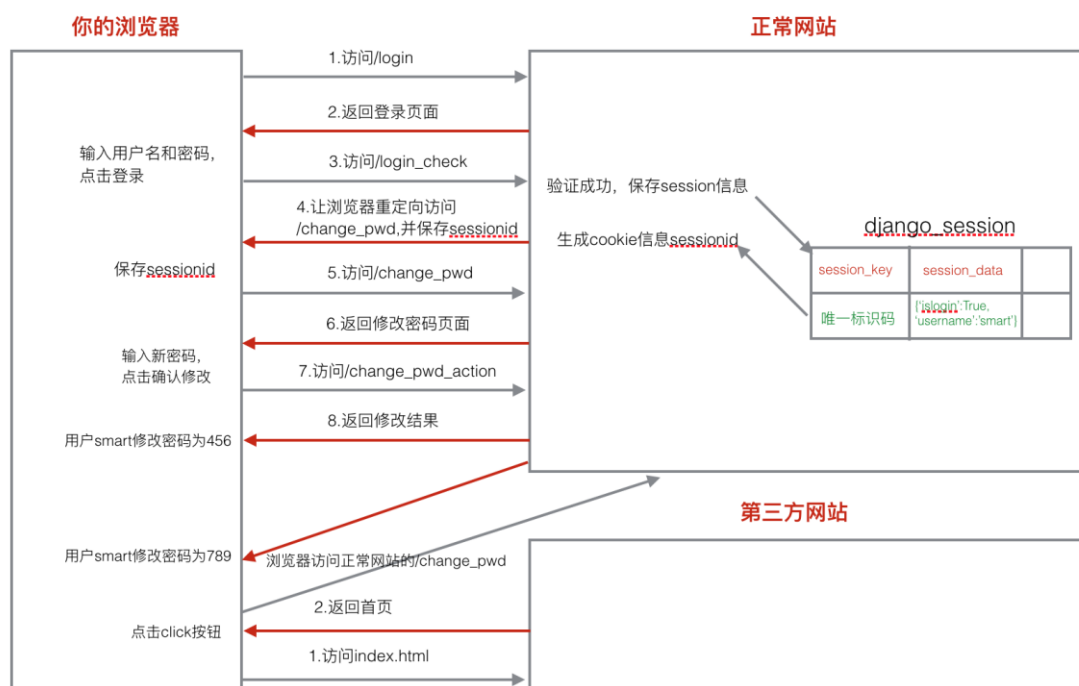
加功能：

a) 只有用户登录之后才可以进行修改密码操作。

登录装饰器函数。

```
def login_required(view_func):
    '''登录判断装饰器'''
    def wrapper(request, *view_args, **view_kwargs):
        # 判断用户是否登录
        if request.session.has_key('islogin'):
            # 用户已登录,调用对应的视图
            return view_func(request, *view_args, **view_kwargs)
        else:
            # 用户未登录,跳转到登录页
            return redirect('/login')
    return wrapper
```

案例流程图:



django 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。

8. 验证码

在用户注册、登录页面，为了防止暴力请求，可以加入验证码功能，如果验证码错误，则不需要继续处理，可以减轻业务服务器、数据库服务器的压力。

9. 反向解析

当某一个 url 配置的地址发生变化时，页面上使用反向解析生成地址的位置不需要发生变化。

根据 url 正则表达式的配置动态的生成 url。

在项目 urls 中包含具体应用的 urls 文件时指定 namespace；

```
urlpatterns = [
    url(r'^admin/', include(admin.site.urls)),
    url(r'^$', include('booktest.urls', namespace='booktest')), # 包含应用的urls文件
]
```

在应用的 urls 中配置是指定 name；

```
urlpatterns = [
    url(r'^index2/$', views.index, name='index', # 模板变量
    url(r'^temp_var/$', views.temp_var), # 模板标签
    url(r'^temp_tags/$', views.temp_tags), # 模板过滤器
    url(r'^temp_filters/$', views.temp_filters), # 模板继承
    url(r'^temp_inherit/$', views.temp_inherit), # 模板多重继承
    url(r'^grand_son/$', views.grand_son), # 模板多重继承

    url(r'^html_escape/$', views.html_escape, name='html_escape'), # html转义
    url(r'^show_reverse/$', views.show_reverse), # 显示反向解析页面

    url(r'^show_arg/(\d+)/(\d+)/', views.show_arg, name='show_arg'), # 携带位置参数
    url(r'^show_kwarg/(?P<num1>\d+)/(?P<num2>\d+)/', views.show_kwarg, name='show_kwarg'), # 携带关键字参数
    url(r'^redirect_test/$', views.redirect_test), # 视图函数中使用反向解析函数
]
```

在模板文件中使用时，格式如下：

{% url 'namespace 名字: name' %} 例如 {% url 'booktest:fan2' %}

带位置参数：

{% url 'namespace 名字: name' 参数 %} 例如 {% url 'booktest:fan2' 1 %}

带关键字参数：

{% url 'namespace 名字: name' 关键字参数 %} 例如 {% url 'booktest:fan2' id=1 %}

在重定向的时候使用反向解析：

```
from django.core.urlresolvers import reverse
```

无参数:

```
reverse('namespace 名字:name 名字')
```

如果有位置参数

```
reverse('namespace 名字:name 名字', args = 位置参数元组)
```

如果有关键字参数

```
reverse('namespace 名字:name 名字', kwargs=字典)
```

其他技术

1. 静态文件

1.1 使用

在 网页使用的 **css** 文件, **js** 文件和图片叫做静态文件。

1) 新建静态文件夹 **static**。



2) 配置静态文件所在的物理目录。Settings.py

```
#STATIC_URL = '/static/'  
STATIC_URL = '/abc/'  
STATICFILES_DIRS = [os.path.join(BASE_DIR, 'static')] # 设置静态文件存放的物理目录
```

STATIC_URL 设置访问静态文件对应的 **url**。

STATICFILES_DIRS 设置静态文件所在的物理目录。

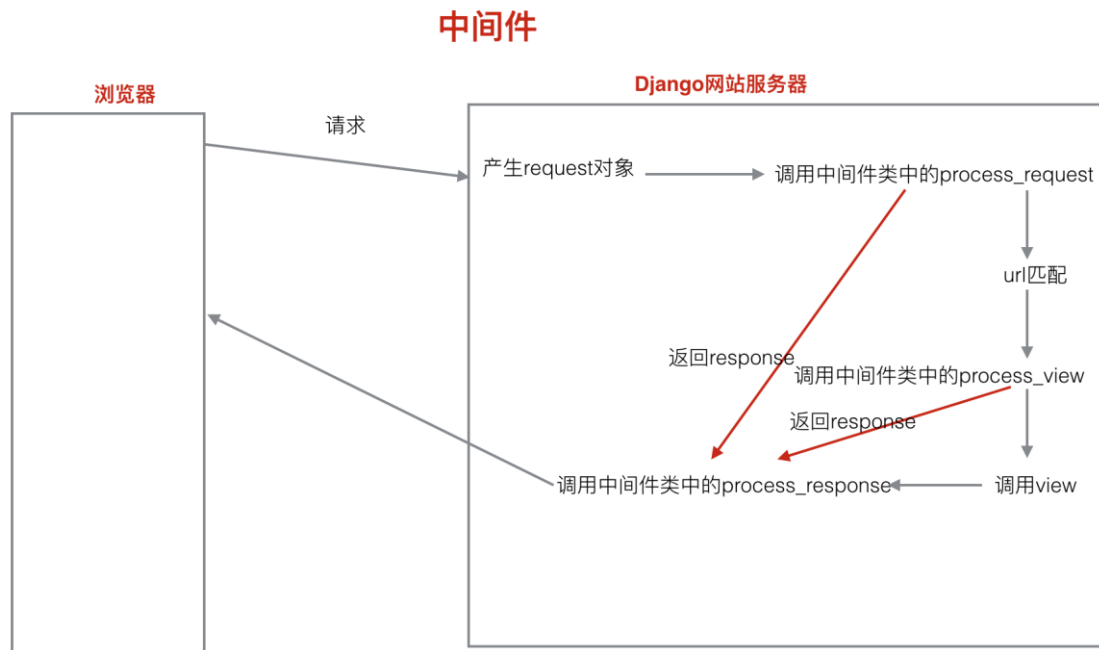
动态生成静态文件的路径。

1.2 加载目录

```
STATICFILES_FINDERS=(  
    'django.contrib.staticfiles.finders.FileSystemFinder',  
    'django.contrib.staticfiles.finders.AppDirectoriesFinder')
```

2. 中间件

中间件函数是 django 框架给我们预留的函数接口，让我们可以干预请求和应答的过程。

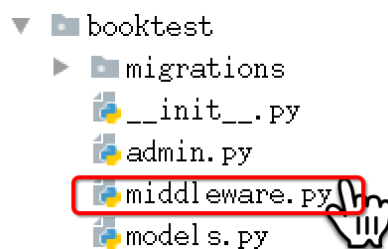


2.1 获取浏览器端的 ip 地址

使用 request 对象的 META 属性: `request.META['REMOTE_ADDR']`

2.2 使用中间件

1) 新建 middleware.py 文件。



2) 定义中间件类。

```

class BlockedIPSMiddleware(object):
    '''中间件类'''
    exclude_ips = ['172.16.179.152']

    # 在调用每一个视图函数之前调用，中间件函数
    def process_view(self, request, view_func, *view_arg,
**view kwargs):
        # 获取浏览器端的ip地址
        user_ip = request.META['REMOTE_ADDR']
        if user_ip in BlockedIPSMiddleware.exclude_ips:
            # 进行访问
            return HttpResponse('<h1>Forbidden</h1>')

```

在类中定义中间件预留函数。

`__init__`:服务器响应第一个请求的时候调用。

`process_request`:是在产生 `request` 对象，进行 url 匹配之前调用。

`process_view`: 是 url 匹配之后，调用视图函数之前。

`process_response`: 视图函数调用之后，内容返回给浏览器之前。

`process_exception`:视图函数出现异常，会调用这个函数。

如果注册的多个中间件类中包含 `process_exception` 函数的时候，调用的顺序跟注册的顺序是相反的。

3) 注册中间件类。

```

MIDDLEWARE_CLASSES = (
    'django.contrib.sessions.middleware.SessionMiddleware',
    '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',
    'booktest.middleware.BlockedIPSMiddleware',
)

```

3. Admin 后台管理

3.1 使用

1) 本地化。语言和时区本地化。

2) 创建超级管理员。

```
python manage.py createsuperuser
```

3) 注册模型类。

4) 自定义管理页面。

自定义模型管理类。

注册模型类的时候给 `register` 函数添加第二个参数，就是自定义模型管理类
的名字。

3.2 模型管理类相关属性

1) 列表页相关的选项。

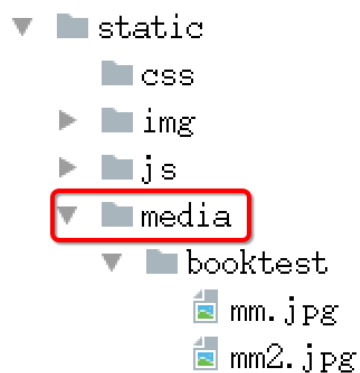
2) 编辑页相关的选项。

4. 上传图片

商品销售网站。

4.1 配置上传文件保存目录

1) 新建上传文件保存目录。



2) 配置上传文件保存目录。

```
MEDIA_ROOT = os.path.join(BASE_DIR, 'static/media') # 设置上传文件的存放目录
```

4.2 后台管理页面上上传图片

1) 设计模型类。

```
class PicTest(models.Model):  
    '''上传图片'''  
    goods_pic = models.ImageField(upload_to='booktest')
```

2) 迁移生成表格。

```
mysql> desc booktest_pictest;  
+-----+-----+-----+-----+-----+-----+  
| Field      | Type          | Null | Key | Default | Extra          |  
+-----+-----+-----+-----+-----+-----+  
| id         | int(11)       | NO   | PRI | NULL    | auto_increment |  
| goods_pic  | varchar(100)  | NO   |     | NULL    |                |  
+-----+-----+-----+-----+-----+-----+  
2 rows in set (0.03 sec)
```

3) 注册模型类。

4.3 用户自定义页面上上传图片

1) 定义用户上传图片的页面并显示，是一个自定义的表单。

```
<form method="post" enctype="multipart/form-data" action="/upload_handle">  
    {% csrf_token %}  
    <input type="file" name="pic"><br/>  
    <input type="submit" value="上传">  
</form>  
</body>  
</html>
```

2) 定义接收上传文件的视图函数。

request 对象有一个 **FILES** 的属性，类似于字典，通过 request.FILES 可以获得上传文件的处理对象。

在 django 中，上传文件不大于 2.5M,文件放在内存中。上传文件大于 2.5M,文件内容写到一个临时文件中。

Django 处理上传文件的两个类：

```
FILE_UPLOAD_HANDLERS=  
    ("django.core.files.uploadhandler.MemoryFileUploadHandler",  
     "django.core.files.uploadhandler.TemporaryFileUploadHandler")
```

上传图片参考资料:

1. http://python.usyiyi.cn/documents/django_182/topics/http/file-uploads.html
2. http://python.usyiyi.cn/documents/django_182/ref/files/uploads.html#django.core.files.uploadedfile.UploadedFile

5. 分页

查询出所有省级地区的信息，显示在页面上。

```
AerolInfo.objects.filter(aParent__isnull = True)
```

- 1) 查询出所有省级地区的信息。
- 2) 按每页显示 10 条信息进行分页，默认显示第一页的信息，下面并显示出页码。
- 3) 点击 **i** 页链接的时候，就显示第 **i** 页的省级地区信息。

```
from django.core.paginator import Paginator
```

```
paginator = Paginator(areas, 10) #按每页 10 条数据进行分页
```

Paginator 类对象的属性:

属性名	说明
num_pages	返回分页之后的总页数
page_range	返回分页后页码的列表

Paginator 类对象的方法:

方法名	说明
page(self, number)	返回第 number 页的 Page 类实例对象

Page 类对象的属性:

属性名	说明
-----	----

number	返回当前页的页码
object_list	返回包含当前页的数据的查询集
paginator	返回对应的 Paginator 类对象

Page 类对象的方法：

属性名	说明
has_previous	判断当前页是否有前一页
has_next	判断当前页是否有下一页
previous_page_number	返回前一页的页码
next_page_number	返回下一页的页码

分页参考资料：

http://python.usyiyi.cn/translate/django_182/topics/pagination.html

6. 省市县选择案例

1) 显示省地区信息。

```
$.get('/prov', function(data){
})
```

2) 省改变时在对应的下拉列表框中显示下级市的信息。

```
$.get('/city?pid='+pid, function(data){
})
request.GET.get('pid')
```

或者：

```
$.get('/city'+$(this).val(), function(data){
})
```

3) 市改变时在对应的下拉列表框中显示下级县的信息。

```
$.get('/dis?pid='+pid, function(data){
})
```

或者:

```
$.get('/dis'+$(this).val(), function(data){  
  })
```