目录

[一、项目概述 3](#_Toc11339651)

[1.1项目概览 3](#_Toc11339652)

[1.1.1项目概览 3](#_Toc11339653)

[1.1.2实现的基本功能+特色功能 3](#_Toc11339654)

[1.1.3技术栈一览 4](#_Toc11339655)

[1.2项目说明 4](#_Toc11339656)

[1.2.1项目报告说明 4](#_Toc11339657)

[1.2.2代码规范说明 5](#_Toc11339658)

[1.3需求分析 6](#_Toc11339659)

[···1.3.1最初的需求 6](#_Toc11339660)

[1.3.2 项目最终需求 8](#_Toc11339661)

[1.4.项目设计 8](#_Toc11339662)

[1.4.1 设计模式 8](#_Toc11339663)

[1.4.2框架选择 9](#_Toc11339664)

[1.5开发说明 10](#_Toc11339665)

[1.5.1开发环境 10](#_Toc11339666)

[1.5.2 生产环境 10](#_Toc11339667)

[1.5.3 django项目开发流程 11](#_Toc11339668)

[二、模型层和数据库设计 13](#_Toc11339669)

[2.1 数据库设计 13](#_Toc11339670)

[2.1.1 需求分析 13](#_Toc11339671)

[2.1.2 数据存储方案 15](#_Toc11339672)

[2.1.3 数据库结构详细设计 18](#_Toc11339673)

[2.2 模型层设计 22](#_Toc11339674)

[2.3. 模型层开发过程： 23](#_Toc11339675)

[2.3.1 django项目环境搭建， 23](#_Toc11339676)

[2.3.2 app模块设计： 23](#_Toc11339677)

[2.3.3 后台管理系统搭建： 27](#_Toc11339678)

[三、模板层和视图层 30](#_Toc11339679)

[3.1用户注册和登录 30](#_Toc11339680)

[3.1.1带邮箱验证的用户注册 30](#_Toc11339681)

[3.1.1验证码登录 36](#_Toc11339682)

[3.2主页和模板继承 38](#_Toc11339683)

[3.2.1模板继承 38](#_Toc11339684)

[3.2.2主页index.html 38](#_Toc11339685)

[3.3商品列表页： 39](#_Toc11339686)

[3.4商品详情页 40](#_Toc11339687)

[3.5购物车页面 41](#_Toc11339688)

[3.6订单提交页 41](#_Toc11339689)

[3.7用户订单页 45](#_Toc11339690)

[3.8用户中心 47](#_Toc11339691)

[3.8全局搜索功能 49](#_Toc11339692)

[3.9书籍采集模块—豆瓣爬虫 51](#_Toc11339693)

[四、项目部署 52](#_Toc11339694)

[4.1 Nginx+uwsgi部署django 52](#_Toc11339695)

[4.2 部署后的性能优化 56](#_Toc11339696)

[五、web安全 56](#_Toc11339697)

[5.1 SQL注入攻击与防范 56](#_Toc11339698)

[5.2 xss攻击与防范 57](#_Toc11339699)

[5.3 csrf 跨站请求伪造 58](#_Toc11339700)

[六、项目总结 58](#_Toc11339701)

# 一、项目概述

## 1.1项目概览

### 1.1.1项目概览

项目名称：图灵书城—在线购书网站

项目演示地址：[www.37ys.top](http://www.37ys.top) (不保证链接永久有效)

开发环境：win10+pycharm

生产环境：LNMP（centos7+ngnix+mysql+python）

版本控制：分布式版本控制系统git

Github地址：https://github.com/wantaname/django

### 1.1.2实现的基本功能+特色功能

1.邮箱注册，验证码登录

2.首页按评分和销量展示书籍

3.全局搜索模块

4.更多书籍的分页展示

5.书籍详情模块,含详情和评论

6.购物车功能

7.支付宝支付功能

8.订单页面

9.用户中心

10.豆瓣爬虫系统(小心你的ip),自动采集书籍到数据库

11.xadmin杀手级后台

12.缓存购物车和最近浏览

### 1.1.3技术栈一览

前端: 三剑客 + jq + ajax + bootstrap

数据库: mysql + redis

后端:ORM + MVC(MTV) + celery

部署: Nginx + uWSGI + Django

## 1.2项目说明

### 1.2.1项目报告说明

本报告记录了整个项目设计和开发过程，集理论与实践于一体，每一个部分都有详细代码说明和运行截图。为了增强可读性，我尽量把理论与实践，设计和开发分开，力求完整，读者可以根据目录选看。

本文档的目的：争取详细、完整，以一个完整项目涵盖web框架的基本知识。

关于文档编排组织，本文档按照开发过程顺序设计，根据web架构分模块书写，看目录一目了然。

文档会提供关键性的代码和截图，所有python代码已采用planetB在线工具[[1]](#footnote-1)格式化后插入到word中。

此文档是在项目已部署上线后写的，所以所有代码都已测试通过！但限于个人水平及表达能力有限，还是难免有一些没有注意到的bug，谅解！

提醒一下，由于最初设计的各种原因，我把书籍当成商品，以下不对商品和书籍作区分。

最后一点，本文档的风格有点独特：带有强烈的主观色彩！除一些公认的原理和官网文档说明外，有大量描述是我个人的理解，望读者对这部分内容以批判的眼光看待，你也可以有自己的理解！

### 1.2.2代码规范说明

除前端外所有代码皆为python，尽量遵循最新的PE8规范[[2]](#footnote-2)，例如：

1.使用四个空格而不是tab进行缩进

2.默认使用utf-8编码

3.尽量不要使用魔术方法

4.类中使用self作为默认参数

5.命名时，尽量使用驼峰式或单词+下划线，要保证见名知意

6.操作符和逗号之后使用空格，但不要在括号内部使用

7.使用空行来间隔函数与类

8.一行不要超过79个字母

9.要写函数与类注释

另外，作为一门动态语言，python代码的可读性不如java等后端语言。所以代码的模块化设计很重要。Python支持三种编程风格：面向过程、函数式、面向对象。为了可读性后期开发和维护，有必要开发前规定使用的优先级：面向对象>函数式>面向过程

## 1.3需求分析

### ···1.3.1最初的需求

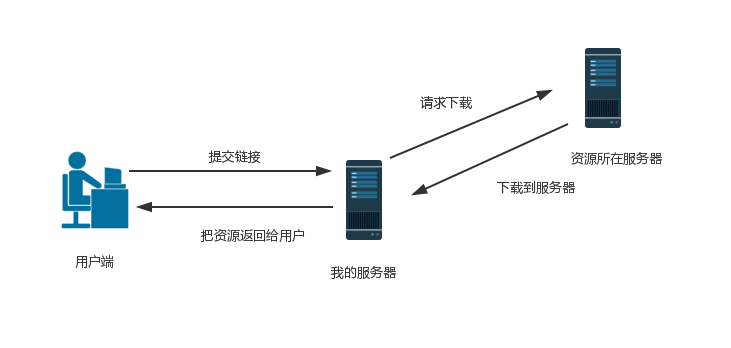
需求概述

需求源于实际生活。

个人在闲鱼等平台开了店铺，有偿提供诸如csdn、百度等下载服务，具体操作这里不做赘述。每日销量有40+，虽然能赚一点饭钱，但有时没有时间！如此重复性的工作不如交给机器处理！有两种选择，开发app和web应用。基于以下原因我选了后者：

1. 用户可能懒得下载app，而浏览器几乎每人都有
2. App的跨平台性和兼容性不好，这意味着针对不同系统的手机端和电脑端要单独开发。而web应用，早已被html+css+javascript 统一多年
3. 考虑到开发时间、效率、难度，web无疑是最好的选择

以下是我最初的设计：



说明：如图所示，用户登录我的网站，提交链接到服务器并付款，服务器向资源所在服务器请求下载，下载到本地后返回给用户

可行性分析

用户请求服务器这一模块显然是比较好实现的。项目最大的难点是，我的服务器如何从第三方服务器上下载资源到本地。以下载csdn为例，csdn服务器做了反爬虫，我尝试用简单的爬虫模拟用户登录进行下载，宣告失败。那么就只能在服务端用selenium驱动真实浏览器进行访问和下载，这是可行的，但在速度会比爬虫慢很多，不过仍然可以接受。还有另外一个问题，因为没有企业的营业执照，所以不能接入第三方支付接口，只能使用沙箱环境。但可以使用第四方支付平台接口，尽管平台会收一定手续费。

项目分析

需求虽然可行，但是对于2.2.2提到的问题我没有成熟的解决方案，而且考虑到开发时间太短，以及个人能力有限，故很难开发出这个系统。试了一段时间失败！只能放弃该项目，虽然我很想做出这个可以盈利的项目。

### 1.3.2 项目最终需求

简述

毕业季，很多大四的学长学姐藏书万卷但很多都不会带回去，弃之可惜，免费赠送又不甘。于是有人专门组织了跳蚤市场，但是，该市场开展这么久以来居然没有一个平台？上次的学代会上有人就提了这个问题。鉴于此，我开发了此项目。要实现以下基本需求：（1）卖家在后台管理系统上传书籍，包括书籍的名字，详情，价格，豆瓣评分，简述，详细信息等；（2）买家在网站上进行浏览、评论书籍，并可以添加购物车购买；（3）用户可以在网站上根据书名，描述和详情全局搜索书籍（4）用户下单后在后台生产相关订单书籍。具体分析请看数据库设计一章。

可行性分析

除了正式的支付宝接口拿不到外，其他完全可以实现。用支付吧的沙箱环境代替支付。另外，前期没有卖家，所以书籍数据从豆瓣读书爬取。

## 1.4.项目设计

基本需求确定好了，就开始设计项目了。

如前文所言，选择的是web应用。

### 1.4.1 设计模式

此项目是全栈的web项目，前后端以及数据库都要一个人完成，因此开发之前有必要规划一下整体设计。

为了解耦，设计模式毫无疑问选择前后端分离，无论是前期开发还是后期维护，都是目前来说很好的选择。尽管某些程序员提出了大前端、大后端的理念，但我认为今时不同往日，现在的开发技术更迭快、需求变更快、个人很难兼顾全局。目前对于前后端分离的一种实现方式是MVC设计模式，本项目理所当然会选择MVC模式。

另外说一点，解耦不仅仅指前后端，对于前端，前端的样式和内容也要解耦。对于后端，也可以考虑使用ORM来与数据库交互，这样可以提供开发效率以及便于后期维护。

### 1.4.2框架选择

此项目用套接字编程显然是不行的，所以前后端都要采用web框架。

首先是开发语言的选择，前端无疑采用前端三剑客：html+css+js。后端可以选java、php、python等。为web而生的php显然很适合中小型项目开发，但个人对php的语法逻辑不是很喜欢，首先排除掉。Java的开发有过些许经验，深感java是一把40米大刀，开发周期会很长。所以最终选择了有“瑞士军刀”之称的python，作为动态语言，势必会带来性能损失，但对于中小型网站来说，这点性能完全可以忽略。只考虑性能而不考虑需求就是耍流氓！

开发框架的选择。Python的web开发框架有很多，目前主流有Flask、django、Tornado。Flask是微量级的框架，django是重量级的企业框架，Tornado则是目前python中性能最高的web框架，国外用的比较多。基于以下原因我选择了django：

1. Django是企业级框架，比Flask全面，更容易理解web开发；
2. Django源于一个成功的新闻网站系统；
3. Django官方文档堪称web框架中最全的，以至于市面上很少有人出书，我所看几本django书籍基本都在django文档中能找到

## 1.5开发说明

### 1.5.1开发环境

如果读者要copy我的代码，那么请一定要注意版本问题！

开发语言python 3.7，web框架版本django2.1，数据库mysql5.7，用到了分布式版本控制系统git[[3]](#footnote-3)，所有代码托管在github，虽然是个人开发，但版本控制依然不可忽视，项目中经常会遇到需要回退的地方。Git是目前世界上最先进的分布式版本控制系统（没有之一），强烈推荐使用！

开发工具推荐使用JetBrain公司的pycharm！

### 1.5.2 生产环境

生产环境采用LNMP (python)，具体来说是: centos7 + ngnix +Mysql 5.7+Python 3.7。建议读者购买一台云服务器用以部署项目，腾讯云阿里云都还可以，不过要备案才能绑域名，为了免备案，我买的香港服务器，推荐这家ISP：<https://www.locvps.com/>

### 1.5.3 django项目开发流程

简介

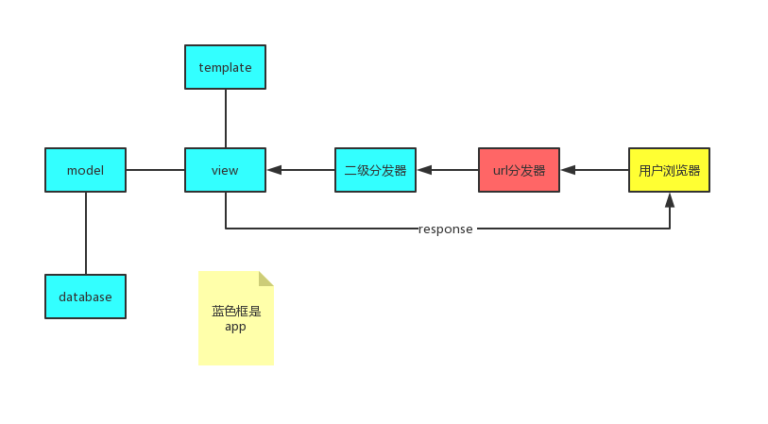
为了方便没有django基础的读者以及为了理清开发思路，我对django的开发做一个简单说明：

在django框架中，MVC模式被特别称为MTV，即模型层(Model)，模板层(Tempalte)，视图层(View)，其原理和MVC类似。

模型层：用于与数据库交互，它是整个项目唯一的可信的数据来源，本项目在模型层采用了主流的ORM技术，即把数据库中的表抽象成模型层中的一个类，表的字段对应类的属性，而表的一条记录对应类的一个实例。

模板层：为了实现动态页面，前端用模板层实现。模板层中除了静态的Html页面等，还嵌入了模板语言用于动态渲染前端页面，渲染需要使用模板引擎，如jinja2，本项目中采用了django自带的模板引擎DTL进行渲染。这一层的逻辑代码能少则少，尽量放到视图层处理，所以模板语言的语法也比较单一。

视图层：视图层是项目的核心，所有的项目逻辑在这里实现，负责接收用户请求，与模型层交互，并渲染模板返回给用户浏览器。下图是django的工作机制：



项目目录结构规范：

Django开发中的目录层级设计没有严格要求，但有必要提前设计一下增强模块的可读性和降低维护成本，下图是我的目录：

采用模块化设计：一个web项目，多个web应用

url路由转发：统一由项目url转发到应用url，再由应用处理

静态文件：由于各种原因，我把图片，css，js文件统一放到了static目录，虽然不会带来性能影响，但是static目录的本意是网站几乎不会更改的静态文件，而图片视频等用户上传的文件如果需要频繁更改则需要放到media目录

开发流程

数据库设计---模型层---模板层---视图层---模板层---视图层……

数据库设计和模型层关系紧密，所以放到一起。模板层和视图层要进行前后端交互，所以也放到一起。

# 二、模型层和数据库设计

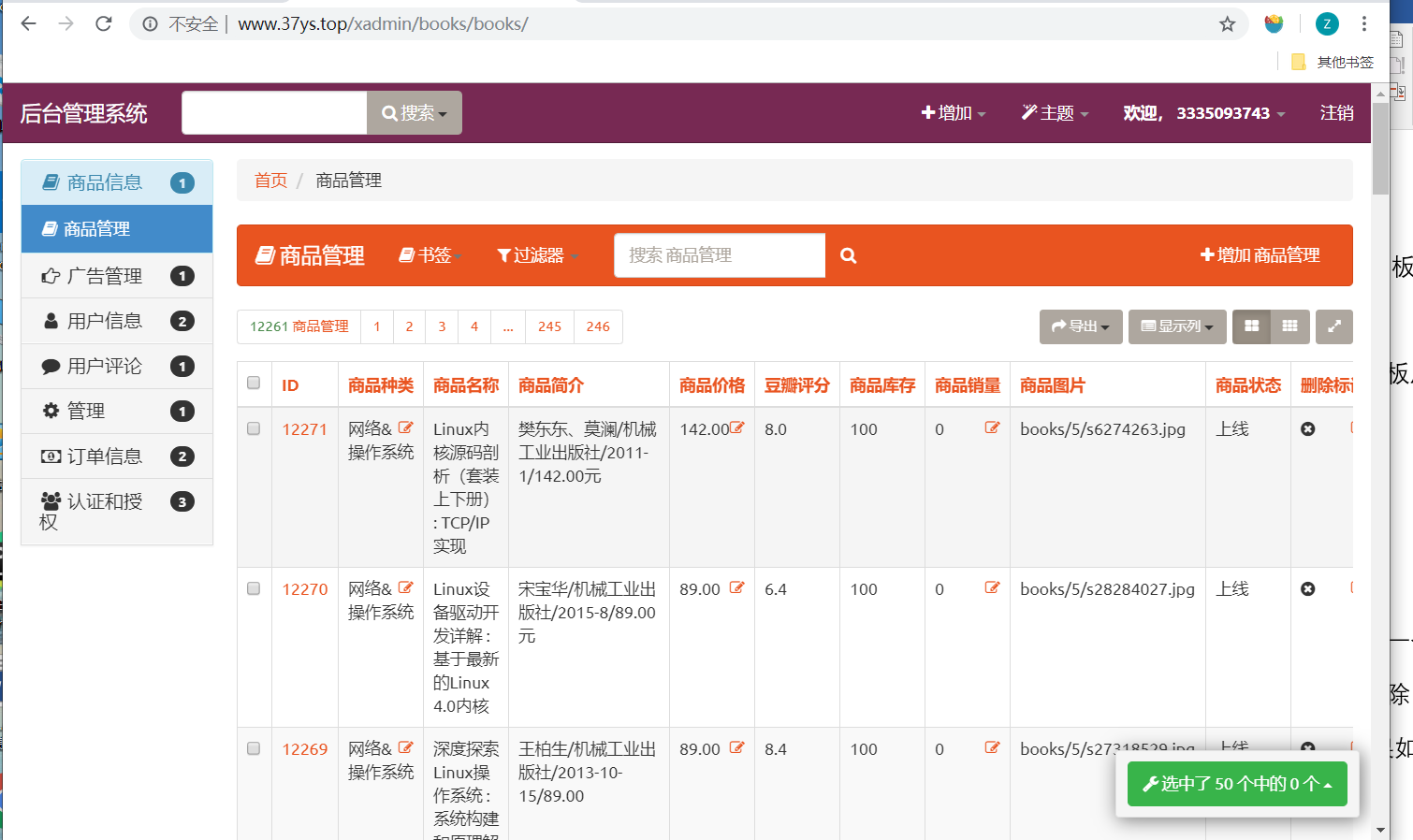
## 2.1 数据库设计

### 2.1.1 需求分析

对3.2小节的需求分析作详细说明

基本需求：

（1）首先，需要一个后台管理系统来便于管理人员对数据库进行可视化的便捷管理，除了对数据表进行管理外，还应包含权限管理系统和操作日志，最终实现的效果如图：



（2）需要对用户信息进行管理，包括用户的名称，密码，注册邮箱，激活状态，收件人，收件地址，邮政编码，联系电话，收件地址是否默认等

（3）需要对用户的评论进行管理，包括评论的用户，书籍，评论内容，禁用评论，评论时间等。因为实现楼中楼比较复杂，即任何人都可对任意再进行内容进行评论，所以在基本功能中先不做，后期有需要在进行扩展。

（4）广告管理，在网站首页放有轮播图展示广告，以及侧边栏也可以在不影响用户体验的情况下投放广告，因为现在大部分网站的盈利模式仍然是广告，有必要单独对广告进行管理。基本信息包括：广告名，广告图，广告链接等。

（5）需要对用户的操作进行单独记录，比如：用户浏览的最近商品，用户，用户登录时输入的验证码以验证是非为机器人等。

（6）购物车管理，包括用户名信息，商品信息，商品数量，加入时间等。

（7）订单信息，需要记录订单编号，下单账号，收货地址，商品数量，支付状态，运费，总价，支付方式，订单状态，支付宝回传的支付编号等。

（8）还有很重要的一个就是用户会话session的管理，session中可存储用户的账号信息，以及用户相关的数据。

（9）商品管理：包含了书籍得分类和书籍的基本信息，书籍分类有：后端开发、前端开发、数学&算法、人工智能&大数据、计算机&互联网、数据库、其他

高阶需求：

（1）评论楼中楼

（2）用户和商家的聊天系统

（3）物流系统

出于各种原因，我只实现了基本需求

初步关系模式可设计如下：

初步的E-R图设计如下：

### 2.1.2 数据存储方案

大多数人说到数据库会立马想到使用sqlserver，mysql，postgrel等，包括之前我也是，这是一种不好的习惯。为什么这样说呢？因为数据库有SQL（关系型数据库）、NoSQL[[4]](#footnote-4)（Not only SQL，非关系型数据库）。前者主要代表有SQL Server，Oracle，MySQL(开源)，PostgreSQL(开源)。后者主要代表有：MongoDB，Redis，CouchDB。

两者的区别大致如下：

（1）存储方式：SQL是基于表的，其数据是结构化的数据，数据表预先定义了列，约束等结构。Nosql数据库基于动态结构，使用非结构化数据，容易适应数据类型和结构的变化；

（2）存储扩展：SQL是纵向扩展，即想要提高处理能力，要使用速度更快的计算机，性能瓶颈受数据库的结构影响，所以扩展空间小。Nosql数据库是横向扩展的，它的存储天然就是分布式的，可以通过给资源池添加更多的普通的数据库服务来分担负载。

（3）性能方面：关系型数据库为例维护数据的一致性付出了巨大代价，读写性能较差，尤其是遇到高并发和海量数据时这种缺点尤为明显。而Nosql存储的格式都是Key-value类型的，并且一般存储在内存中，读写容易且快

（4）安全方面：Nosql数据的一致性比SQL弱很多，且Nosql存储在内存中，易失性比SQL高，且漏洞容易被利用而遭到攻击，如Nosql注入等

综上所述，两者各有优缺点，他们的关系不像python和java那样一方基本可以替代另一方，应该根据场景选择SQL+Nosql的组合方案，目前绝大多数网站都采用Mysql+Nosql的存储方案，别问为什么，大家都在用的方案就是最成熟的方案，其优点不做赘述。

本项目具体使用的是：Mysql+redis[[5]](#footnote-5)，redis是c语言编写的基于内存亦可持久化的Key-value数据库，读写性能很高，推荐使用！

以下是本项目中需要用到的redis的数据结构，详细文档请移步官网，同时推荐w3cschool上的redis基础教程[[6]](#footnote-6)：

1. String：字符串，用于键值对存储,如：set key value
2. Hash：哈希，键值对集合，常用于存储对象，如：set key field value
3. List：列表，Redis 列表是简单的字符串列表，按照插入顺序排序, 如：lpush key value

……

据此设计以下存储方案来实现需求：

（1）用户的会话数据需要频繁使用，比如网页的切换，是否登录的判断，对此，django官方文档提[[7]](#footnote-7)出了若干解决方案，包括基于数据库、缓存、缓存+数据库、文件系统、cookie的会话。我采用的是基于redis缓存的session，具体配置参见项目配置一章。可以看下django会话表的实现（尽管我们不会使用）：



Session表简单说明：session表保存用户的

session\_key: 与用户浏览器发过来的sessionid进行比较，拿到用户的session\_data ；

session\_data：对用户数据进行加密后得到的会话数据；

expire\_data: 会话过期的时间戳

注：要使用session必须要在settings中启用，django默认开启。

（2）需要时刻跟踪和展示用户的浏览记录，持久化到数据库不现实。而且浏览记录不是那么重要，所以放入缓存

（3）用户的购物车模块，购物车表面上看应该放入数据库，但是考虑到购物车变动比较频繁，比如：用户可能随时对购物车进行增删查改，生成订单后购物车对应商品要删除等等，再则，老实说我们对用户的购物车并不关心，我们只关心用户的订单！

（4）对于其他的数据，则需要设计关系表来持久化存储，如权限管理、日志管理、用户信息管理、商品管理

（5）（这是后来加的）在写邮箱验证的时候，需要用celery异步处理发邮件这一功能，celery的中间人broker要具有存储功能以存储消息队列。Celery官方文档不推荐使用mysql等数据库，推荐使用RabbitMQ，redis等缓存。

### 2.1.3 数据库结构详细设计

根据2.1.2节的分析，对mysql和redis数据库的详细设计如下：

redis数据库设计：

（1）会话模块：这个模块不需要开发者操心，一般web框架都会带这个功能。使用第三方库django-redis将会话缓存到redis中

（2）购物车模块：购物车中要存储用户id，购买的商品，购买的商品数量。所以采用Hash数据结构，key，field，value分别对应用户id，商品id和商品数量

（3）用户浏览模块：用户浏览一般只记录最近浏览的几个商品，其余记录要自动删除，本项目只记录最近浏览的前五条数据。很显然，这是一种先进先出的数据结构，故用redis的List实现

（4）用redis缓存celery异步发邮件的消息队列。

mysql数据库设计

设计表之前，考虑两点：1.表的主键问题；2.把每个表都有的共同字段抽取出来。这两点对于单独对于数据库来说可能无关紧要，但对于整个开发来很有必要！

对于问题1：就个人所见而言，现在开发中一般统一采用自增的id作为主键，有以下优势：

数据库层面：id是数据库自身维护的，不会出错，况且，我所知的大部分数据库中，自己设计的主键字段只是理论上唯一的，在实践中可能会出现重复。所以还是设置一个与业务无关的自增ID作为主键，然后把需要unique的字段添加唯一性约束。而且，在查询速度是，数据库既保存id值，同时也会通过建立索引等优化查询速度，自定义的主键如果是字符串，其检索速度是没有数字快的。

后台管理系统：方便查看和检索数据，尤其是这种表格式的数据，没有id看起来很不舒服！

视图层和模板层：写代码逻辑时，不用考虑每个表的主键字段名，直接用id标识记录，不易出错。

所以，本项目中除订单表采用订单号作为主键（前期开发没规范好的问题，影响不对）外，其余均采用自增id作为主键。

对于问题2：一般每一张表都需要有一个标识时间的字段，这里我

采用创建时间和修改时间字段。另外考虑一下删除问题：当用户或者管理员选择删除记录时，就要从数据库中物理删除该记录吗？如果是误删或者想找回删除的数据在不备份的情况下就很麻烦了。所以我们不能真正的物理删除，而采用逻辑删除。具体实现是：给每一张表添加一个删除标记字段。所以，每一个表都需要的公共字段是删除标记，创建时间和修改时间。

基本关系模式（分模块设计）：

以下用base\_db代替 (is\_delete, create\_time,update\_time)

用户模块：包含账户和地址，其中要注意默认收获地址，理论上可以将其放入账户表中，这样加载用户默认收获地址时就不用查地址表，但同样地，修改默认邮箱时会从同时需要操作两张表。所以我把默认收获地址放入了地址表中。地址和账户关系是多对一

账户信息（id, username, password, email, is\_active, base\_db）

地址信息(id, recipient\_name, recipient\_addr, zip\_code, is\_default, recipient\_phone, passport, base\_db），外键passport指向账户

商品模块：商品种类被我写死了，只有6个选择，所以放在了商品表中。商品和评论是一对多关系，和商品相关的评论放在评论表中

商品信息（id, type\_id, name, desc, price, score, stock, sales, detail, image, status, base\_db）

用户评论（id, book, user, content, disabled, base\_db）,外键user指向用户表

订单模块：除了订单信息表，还应考虑到一个订单中可能包含多个商品，订单和商品之间是一对多关系，据此，分离出一张订单商品表

订单信息（order\_id, passport, addr, total\_count, total\_price, transit\_price, pay\_method, status, trade\_id,base\_db）

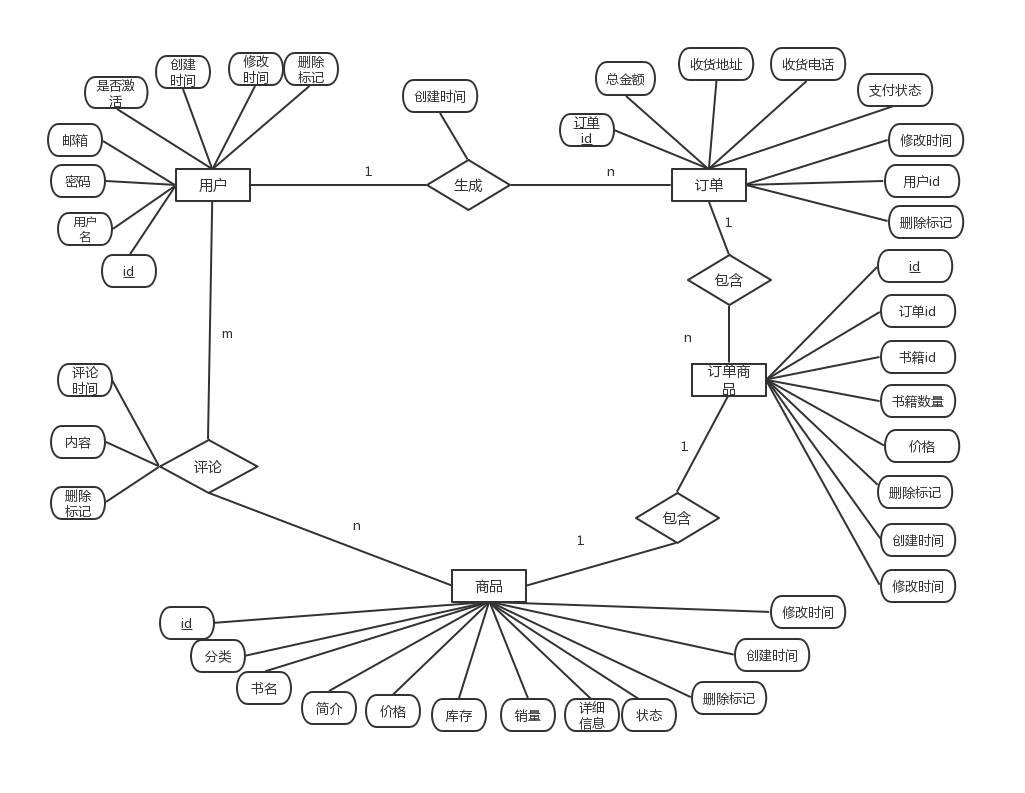
订单商品（id, order, books, count, price, base\_db）外键order指向订单表，books指向商品表

广告模块：一张广告表

广告（id, name, image, url, base\_db）

权限管理模块：django的强大之处在于它的后台管理，尤其是权限系统，默认已经有了三张表，权限，组和用户。后面我们会在此基础上进行修改，打造自己的权限管理系统！

Mysql数据库的E-R图如下：



## 2.2 模型层设计

数据库设计好之后就可以设计模型层了，这是整个项目的起点。Django模型层采用ORM技术，将数据库表映射成一个类，表的字段对于类的属性，一条记录对应类的对象。为什么要使用ORM而不是原生的SQL语句：

1. Django的orm操作本质上会根据对接的数据库引擎，成对应的sql语句，所以不需要考虑各个数据库以及版本之间带来的sql语句差异。数据库迁移的时候，基本不需要任何的维护成本，更换数据库引擎即可，简言之就是orm屏蔽了不同底层数据库的差异
2. 模型层将数据库与视图层解耦，逻辑代码的开发人员无需再关注底层数据。模型层是唯一可信的数据来源，这样可以大大提高开发效率，降低开发成本和开发周期
3. 安全性：事实证明使用orm操作数据库更加安全，因为只允许orm与数据库交互，例如：ORM，接口封装在 ORM 机制内部，不对外暴露，理论上防止了 SQL 注入。
4. 使用ORM后的性能肯定不及原生的ADO，国外某企业经测试发现会带来15%性能损失，但是依然有很多企业级开发使用orm，考虑上述原因，这个性能损失完全可以接受！况且，在一些带来明显性能损失的操作上，也可以直接用ADO代替ORM。

## 2.3. 模型层开发过程：

从模型层开始，进入正式的开发，在此之前要配置好项目环境

### 2.3.1 django项目环境搭建，

已搭建完成的可以跳过本节

Django安装---创建项目---数据库配置---新建app……

详细的配置过程可访问我的博客链接：

<http://www.fishblogs.cn/index.php/2019/04/30/django%E7%8E%AF%E5%A2%83%E6%90%AD%E5%BB%BA/>

### 2.3.2 app模块设计：

**基模块**

**一个app即是一个模块，如前文所言，先把各个app模型的共同字段is\_delete, create\_time, update\_time 抽取出来作为基模型供其他app模型继承，项目目录下新建db/base\_model:**

1. '''''
2. 所有模型抽象出来的基类
3. '''
4. **from** django.db **import** models
6. **class** BaseModel(models.Model):
7. '''''模型抽象基类'''
8. is\_delete = models.BooleanField(default=False, verbose\_name='删除标记')
9. create\_time = models.DateTimeField(auto\_now\_add=True, verbose\_name='创建时间')
10. update\_time = models.DateTimeField(auto\_now=True, verbose\_name='更新时间')
12. **class** Meta:
13. # 说明这是一个抽象类
14. abstract = True

**用户模块**

**在users/models.py中：**

1. '''''
2. 用户信息模型
3. '''
5. **from** django.db **import** models
6. **from** db.base\_model **import** BaseModel
7. **from** django.contrib.auth.hashers **import** make\_password

10. # Create your models here.
11. **class** PassportManager(models.Manager):
12. **def** add\_one\_passport(self, username, password, email):
13. '''''添加一个账户信息'''
14. passport = self.create(username=username, password=password, email=email)
15. # 3.返回passport
16. **return** passport
18. **def** get\_one\_passport(self, username, password):
19. '''''根据用户名密码查找账户的信息'''
20. **try**:
21. passport = self.get(username=username, password=password)
22. **except** self.model.DoesNotExist:
23. # 账户不存在
24. passport = None
25. **return** passport

28. **class** Passport(BaseModel):
29. '''''用户模型类'''
30. username = models.CharField(max\_length=20, unique=True, verbose\_name='用户名称')
31. password = models.CharField(max\_length=40, verbose\_name='用户密码')
32. email = models.EmailField(verbose\_name='用户邮箱')
33. is\_active = models.BooleanField(default=False, verbose\_name='激活状态')
34. # 用户表的管理器
35. objects = PassportManager()
37. # 模型元数据
38. **class** Meta:
39. # 在后台管理中的名字
40. verbose\_name\_plural = verbose\_name = '账户管理'
42. **def** \_\_str\_\_(self):
43. **return** self.username

46. **class** AddressManager(models.Manager):
47. '''''地址模型管理器类'''
49. **def** get\_default\_address(self, passport\_id):
50. '''''查询指定用户的默认收货地址'''
51. **try**:
52. addr = self.filter(passport\_id=passport\_id, is\_default=True).get(is\_delete=False)
53. **except** self.model.DoesNotExist:
54. # 没有默认收货地址
55. addr = None
56. **return** addr
58. **def** add\_one\_address(self, passport\_id, recipient\_name, recipient\_addr, zip\_code, recipient\_phone):
59. '''''添加收货地址'''
60. # 判断用户是否有默认收货地址
61. addr = self.get\_default\_address(passport\_id=passport\_id)
63. **if** addr:
64. # 存在默认地址
65. is\_default = False
66. **else**:
67. # 不存在默认地址
68. is\_default = True
70. # 添加一个地址
71. addr = self.create(passport\_id=passport\_id,
72. recipient\_name=recipient\_name,
73. recipient\_addr=recipient\_addr,
74. zip\_code=zip\_code,
75. recipient\_phone=recipient\_phone,
76. is\_default=is\_default)
77. **return** addr

80. **class** Address(BaseModel):
81. '''''地址模型类'''
82. recipient\_name = models.CharField(max\_length=20, verbose\_name='收件人')
83. recipient\_addr = models.CharField(max\_length=256, verbose\_name='收件地址')
84. zip\_code = models.CharField(max\_length=6, verbose\_name='邮政编码')
85. recipient\_phone = models.CharField(max\_length=11, verbose\_name='联系电话')
86. is\_default = models.BooleanField(default=False, verbose\_name='是否默认')
87. passport = models.ForeignKey('Passport', verbose\_name='账户', on\_delete=models.CASCADE)  # 级联删除
89. objects = AddressManager()
91. **class** Meta:
92. verbose\_name\_plural = verbose\_name = '地址管理'
94. **def** \_\_str\_\_(self):
95. **return** self.recipient\_addr

**限于篇幅，剩下的app代码请前往github查看**

### ****2.3.3 后台管理系统搭建：****

**定义好模型层就可以着手开发后台管理系统了。Django自带管理系统admin，功能比较齐全但前端界面不是很好看。我使用admin的升级版，具有“杀手锏”之称的xadmin，是一位名为“差沙”的国人为django开发的开源项目。Github链接：**

<https://github.com/sshwsfc/xadmin/tree/docs-chinese>

官方文档：

<https://xadmin.readthedocs.io/en/docs-chinese/views_api.html>

把xadmin源码拷贝到项目目录，然后在app目录下创建adminx.py文件，文件中import xadmin即可使用该管理系统。

Python中一切皆对象，管理系统也是采用模型类的方式开发，一个类对应管理系统中的一个表，以在后台中注册用户模块为例：

1. **import** xadmin
2. **from** xadmin **import** views
3. **from** .models **import** \*
5. #主题设置
6. **class** BaseSetting(object):
7. enable\_themes=True
8. use\_bootswatch=True
10. #全局设置
11. **class** GlobalSetting(object):
12. #标题
13. site\_title='后台管理系统'
14. #底部
15. site\_footer='ECNU'
16. #修改菜单栏，改成收缩形式
17. menu\_style='accordion'

20. #账户管理表
21. **class** PassportAdmin(object):
22. #显示字段,显示id便于查看
23. list\_display=['id','username','password','email','is\_active','is\_delete','create\_time','update\_time']
24. search\_fields=['id','username','password','email','is\_active','is\_delete']
25. list\_filter=['id','username','password','email','is\_active','is\_delete','create\_time','update\_time']
26. model\_icon = 'fa fa-user'
27. list\_editable = [ 'is\_delete',]
29. #地址管理
30. **class** AddressAdmin(object):
31. list\_display=['id','recipient\_name','recipient\_addr','zip\_code','recipient\_phone','is\_default','passport','is\_delete','create\_time','update\_time']
32. #passport是外键，搜索和过滤都要指定属性
33. search\_fields=['id','recipient\_name','recipient\_addr','zip\_code','recipient\_phone','is\_default','passport\_\_username','is\_delete']
34. list\_filter=['id','recipient\_name','recipient\_addr','zip\_code','recipient\_phone','is\_default','passport\_\_username','is\_delete']
35. #左侧菜单栏图标
36. model\_icon='fa fa-map-marker'
37. list\_editable = [ 'is\_delete',]

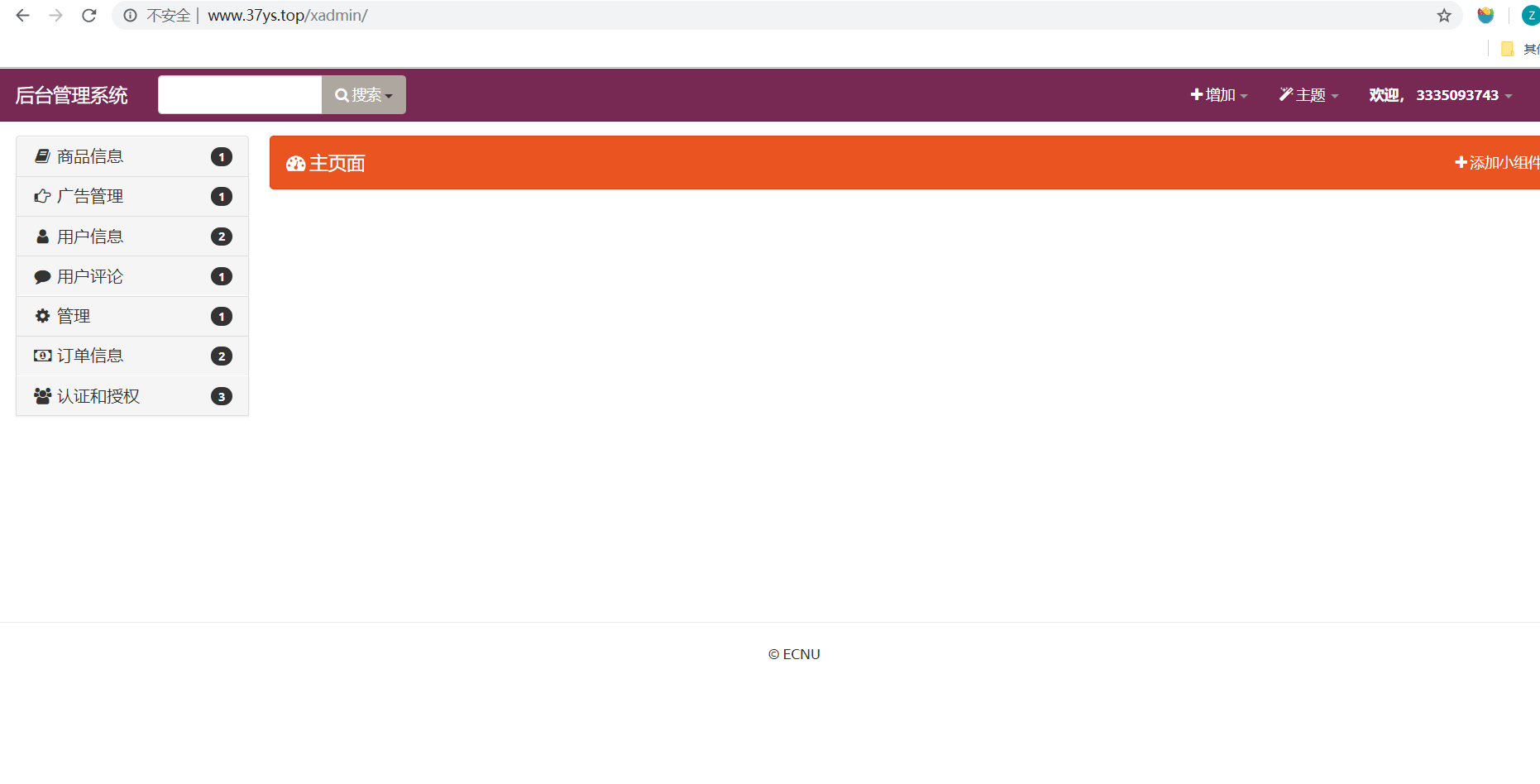
40. #注册管理表
41. xadmin.site.register(Passport,PassportAdmin)
42. xadmin.site.register(Address,AddressAdmin)
44. #注册主题
45. xadmin.site.register(views.BaseAdminView,BaseSetting)
46. #注册设置
47. xadmin.site.register(views.CommAdminView,GlobalSetting)

这里也对全局作了设置，注册好以后，后台中就可以看到账户管理和地址管理两张表，为了将这两张表放到用户信息app下，需要对users目录做一些设置：

1. #users/apps.py
3. **from** django.apps **import** AppConfig
5. **class** UsersConfig(AppConfig):
6. name = 'users'
7. #在后台中的显示
8. verbose\_name='用户信息'
9. #users/\_\_init\_\_.py
11. default\_app\_config='users.apps.UsersConfig'

其他app注册到管理系统同理。

完成之后就可以看到以下效果：



说明一点，菜单栏的图标要自己设置，我使用了font awesome网站的图标库，网址：<http://www.fontawesome.com.cn/>，下载最新版本的font awesome，并覆盖xadmin源码的font-awesome的css和js，就可以通过model\_icon='fa fa-map-marker'的方式引用图标了

# 三、模板层和视图层

模板文件全部放在templates目录，视图函数全部在各个app的views.py文件。以下小节从页面和功能两个角度详细描述各个的开发过程

## 3.1用户注册和登录

### 3.1.1带邮箱验证的用户注册

注册页面如图：



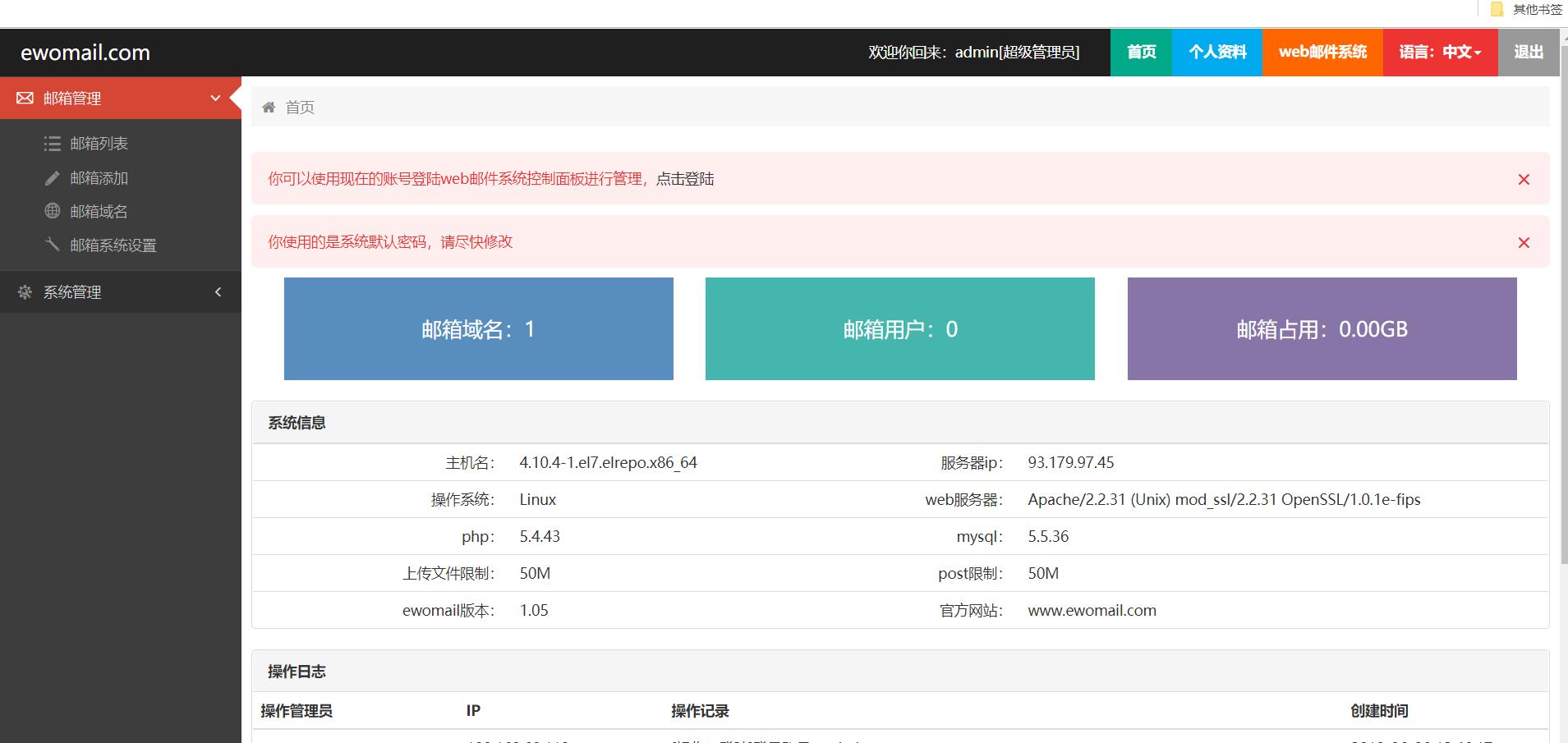
提交数据使用表单实现，前端表单如下：

1. **<form** method="post" action="/user/register/"**>**
2. {% csrf\_token %}
3. **<ul>**
4. **<li>**
5. **<label>**用户名:**</label>**
6. **<input** type="text" name="user\_name" id="user\_name"**>**
7. **<span** class="error\_tip"**>**提示信息**</span>**
8. **</li>**
9. **<li>**
10. **<label>**密码:**</label>**
11. **<input** type="password" name="pwd" id="pwd"**>**
12. **<span** class="error\_tip"**>**提示信息**</span>**
13. **</li>**
14. **<li>**
15. **<label>**确认密码:**</label>**
16. **<input** type="password" name="cpwd" id="cpwd"**>**
17. **<span** class="error\_tip"**>**提示信息**</span>**
18. **</li>**
19. **<li>**
20. **<label>**邮箱:**</label>**
21. **<input** type="text" name="email" id="email"**>**
22. **<span** class="error\_tip"**>**提示信息**</span>**
23. **</li>**
24. **<li** class="agreement"**>**
25. **<input** type="checkbox" name="allow" id="allow" checked="checked"**>**
26. **<label>**同意”图灵书城书城用户使用协议“**</label>**
27. **<span** class="error\_tip2"**>**提示信息**</span>**
28. **</li>**
29. **<li** class="reg\_sub"**>**
30. **<input** type="submit" value="注 册" name=""**>**
31. {{errmsg}}
32. **</li>**
33. **</ul>**
34. **</form>**

对应视图函数中使用基于类的view，如果用户发起的是get请求，则返回注册的html页面，如果是post请求，则作以下处理：通过name属性取到注册表单传来的数据并进行处理，先进行数据的校验判断是否有效，再保存到数据库，如果存在则返回错误信息“用户名已存在”，反之提示注册成功。视图函数如下：

1. **class** Register(View):
2. **def** get(self,request):
3. **return** render(request, 'users/register.html')
4. '''''进行用户注册处理'''
5. **def** post(self,request):
6. #接收post过来的数据
7. username=request.POST.get('user\_name')
8. password=request.POST.get('pwd')
9. email=request.POST.get('email')
11. #进行数据校验
12. **if** **not** all([username,password,email]):
13. #有数据为空
14. **return** render(request,'users/register.html',context={'errmsg':'参数不能为空'})
15. **if** **not** re.match(r'^[a-zA-Z0-9\_-]+@[a-zA-Z0-9\_-]+(\.[a-zA-Z0-9\_-]+)+$', email):
16. #邮箱不合法
17. **return** render(request, 'users/register.html', {'errmsg': '邮箱不合法!'})
18. # 注册，向数据库中添加账户
19. **try**:
20. passport=Passport.objects.add\_one\_passport(username=username,password=password,email=email)
21. **except** Exception as e:
22. **print**('error:',e)
23. **return** render(request,'users/register.html',{'errmsg':'用户名已存在！'})
25. passport.save()
27. # 注册完，还是返回注册页。
28. # return redirect(reverse('user:register'))
29. **return** HttpResponse("恭喜你！注册成功！")

现在还要实现邮箱验证的功能。先要准备一个邮件服务器，可以使用QQ邮箱，163邮箱的等，由于这些邮箱对个人限制比较多，比如频繁发邮件需要手动验证，所以我用开源的Ewomail[[8]](#footnote-8)在自己的vps上搭建了自己的邮件服务器。



但是，由于我的vps内存只有1G，启用该邮件服务导致内存占用接近90%！所以我最后还是使用163邮件服务器，嗯，真香！

项目邮件配置：

1. #发邮件
2. EMAIL\_HOST='smtp.163.com'
3. EMAIL\_PORT=25
4. EMAIL\_HOST\_USER = "nginx\_tomcat@163.com" #发送邮箱
5. #授权码
6. EMAIL\_HOST\_PASSWORD='5201314yzp'
7. EMAIL\_BACKEND='django.core.mail.backends.smtp.EmailBackend'
8. #163邮箱支持SSL
9. EMAILE\_USE\_SSL = True
10. EMAIL\_USE\_TLS = False
11. #收件人看到的名字
12. EMAIL\_FROM = "图灵书城<nginx\_tomcat@163.com>"

然后，重点来了，要实现异步发邮箱，一般使用celery[[9]](#footnote-9)。Celery 是一个简单、灵活且可靠的，处理大量消息的分布式系统。

大型项目要采用模块化设计，故celery实例应独立出来，先创建celery实例：

1. #/Book/celery.py
3. '''''
4. 创建celery的一个实例，供其他模块导入
5. '''
6. **from** \_\_future\_\_ **import** absolute\_import, unicode\_literals
7. **import** os
8. **from** celery **import** Celery
10. #设置celery默认的django设置模块
11. os.environ.setdefault('DJANGO\_SETTINGS\_MODULE','Book.settings')
12. #创建一个celery实例
13. app=Celery('Book',broker='redis://127.0.0.1:6379/2')
14. #用来加载configuration对象中的配置文件
15. app.config\_from\_object('django.conf:settings', namespace='CELERY')
16. #自动发现所有已按照应用程序中的任务task.py
17. app.autodiscover\_tasks()

设置项目启动时加载模块：

1. #/Book/\_\_init\_\_.py
2. '''''
3. \_\_init\_.py文件是一个空文件，因为Django连接MySQL时默认使用MySQLdb驱动，
4. 但MySQLdb不支持Python3，因此这里将MySQL驱动设置为pymysql。
5. '''
6. **from** \_\_future\_\_ **import** absolute\_import, unicode\_literals
7. **import** pymysql
8. pymysql.install\_as\_MySQLdb()
10. #确保django启动时app被加载
11. **from** .celery **import** app as celery\_app
12. \_\_all\_\_ = ('celery\_app',)

然后，官方文档推荐每一个app的需要异步处理的任务放在app的tasks.py文件中：

1. #/users/tasks.py
2. **from** celery **import** shared\_task
3. **from** django.conf **import** settings
4. **from** django.core.mail **import** send\_mail
6. @shared\_task
7. **def** send\_active\_email(token, email):
8. '''''发送激活邮件'''
9. subject = '图灵书城用户激活' # 标题
10. message = ''
11. sender = settings.EMAIL\_FROM # 发件人
12. receiver = [email] # 收件人列表
13. html\_message = '点击下方链接激活您的账户：\n<a href="http://127.0.0.1:8000/user/active/%s/">http://127.0.0.1:8000/user/active/</a>'%token
14. send\_mail(subject, message, sender, receiver, html\_message=html\_message)

这样就定义好了一个异步任务，在视图中通过send\_active\_email任务的delay方法就可以执行该任务：

1. serializer = Serializer(settings.SECRET\_KEY, 3600)
2. token = serializer.dumps({'confirm': passport.id})  # 返回bytes
3. token = token.decode()
4. # 发送激活邮箱
5. send\_active\_email.delay(token, email)

然后，用户点击邮箱中的链接后，激活用户账户并跳转到登录界面：

1. #邮箱验证
2. **def** register\_active(request, token):
3. '''''用户账户激活'''
4. serializer = Serializer(settings.SECRET\_KEY, 3600)
5. **try**:
6. info = serializer.loads(token)
7. passport\_id = info['confirm']
8. # 进行用户激活
9. passport = Passport.objects.get(id=passport\_id)
10. passport.is\_active = True
11. passport.save()
12. # 跳转的登录页
13. **return** redirect(reverse('user:login'))
14. **except** SignatureExpired:
15. # 链接过期
16. **return** HttpResponse('激活链接已过期')

注意要自行启动celery！最好加入守护进程！

至此，邮箱注册功能实现！

### 3.1.1验证码登录

用户的登录逻辑也很简单，为了增强用户体验，采用ajax异步提交，用户请求登录页面，图片验证码使用python的random库和PIL库生成，生成后会记录在临时存储在session种，用户登录成功后删除！

视图函数代码如下：

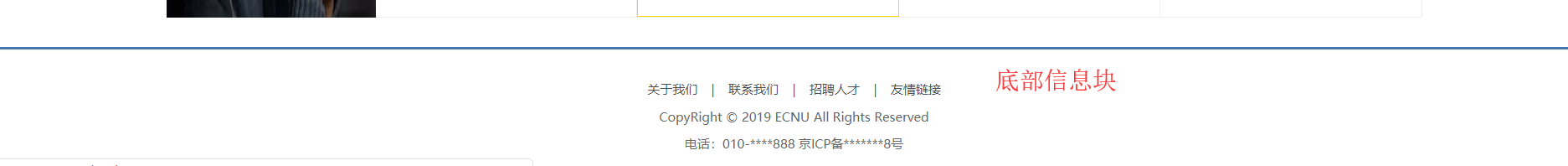
1. **class** Login(View):
2. #请求登录页面
3. **def** get(self,request):
4. #用cookie自动填用户名和密码
5. **if** request.COOKIES.get('username'):
6. username=request.COOKIES.get('username')
7. checked='checked'
8. **else**:
9. username=''
10. checked=''
11. **if** request.COOKIES.get('password'):
12. password=request.COOKIES.get('password')
13. **else**:
14. password=''
15. context={
16. 'username':username,
17. 'password':password,
18. 'checked':checked,
19. }
20. **return** render(request,'users/login.html',context=context)
22. #提交登录请求
23. **def** post(self,request):
24. username = request.POST.get('username')
25. password = request.POST.get('password')
26. remember = request.POST.get('remember')
27. verifycode = request.POST.get('verifycode')
28. # 数据校验
29. **if** **not** all([username, password]):
30. # 有数据为空
31. **return** JsonResponse({'res': 0})
32. #验证码是否正确
33. **if** verifycode.upper() != request.session['verifycode']:
34. **return** JsonResponse({'res': 1})
35. # 数据库中查找账户信息
36. passport = Passport.objects.get\_one\_passport(username=username, password=password)
37. #验证通过,跳转到首页
38. **if** passport:
39. #反转函数，即通过视图反转得到url
40. next\_url=reverse('books:index')
41. **print**(next\_url)
42. #需要返回的JsonResponse对象，是HttpResponse的子类
43. jres=JsonResponse({'res':2,'next\_url':next\_url})
45. #判断是否需要记住用户名和密码
46. #json传递过来的都是字符串，
47. **if** remember=='true':
48. #调用HttpResponse对象的方法设置cookie
49. jres.set\_cookie('username',username,max\_age=7\*24\*3600)
50. jres.set\_cookie('password',password,max\_age=7\*24\*3600)
51. **else**:
52. #不记住用户名和密码
53. jres.delete\_cookie('username')
54. jres.delete\_cookie('password')
55. #设置会话，记住用户的登录状态
56. request.session['islogin']=True
57. request.session['username']=username
58. request.session['passport\_id']=passport.id
59. #登录成功后清理图片缓存
60. **del** request.session['verifycode']
61. **return** jres
63. **else**:
64. #用户名或密码错误
65. **return** JsonResponse({'res':0})

## 3.2主页和模板继承

### 3.2.1模板继承

除注册登录外，其他前端页面有一些部分是相同的，这时就可以利用模板的继承特性，抽象出一个通用模板base.html供别的模板继承。现在的组件化思路也是这样的，松耦合，紧内聚，可复用！





可以看到，顶部信息、头部信息和底部信息块可以抽象成模板文件base.html

### 3.2.2主页index.html

点击主页图标会跳转到主页，搜索栏会根据书名、描述】详细信息搜索书籍，购物车数量从redis缓存中动态获取。

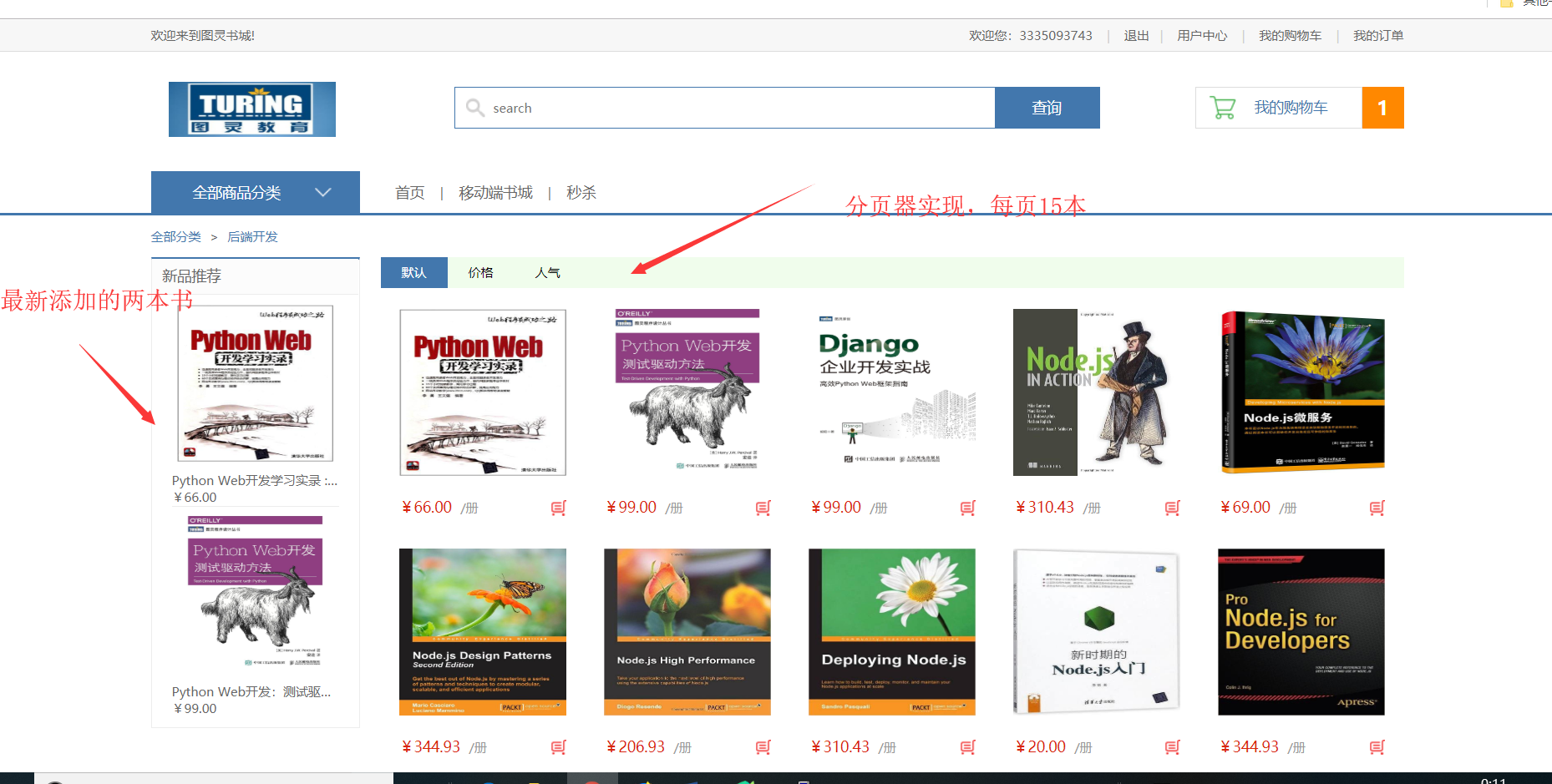
为各个目录设置锚点，点击菜单栏跳转到相应目录，

广告：从数据库获取，四张轮播图，右侧两者图片。

书籍分类条目：三本销量最高的书籍以及四本评分最高的书籍（评分相同按照销量排），为了加快检索，对销量以及评分字段设置了索引

1. price = models.DecimalField(max\_digits=10,decimal\_places=2,verbose\_name='商品价格',db\_index=True)
2. score = models.DecimalField(max\_digits=3,decimal\_places=1, verbose\_name='豆瓣评分',db\_index=True)

## 3.3商品列表页：



新品推荐：相应类目中最新添加的两本书

默认：按评分降序排列

价格：价格从低到高排列

人气：销量从高到低排列

这里使用了分页器Paginater，是从数据库层面分页，而不是使用Bootstrap进行前端分页。即每次只加载需要展示的书籍，大大减轻了数据库的负担！

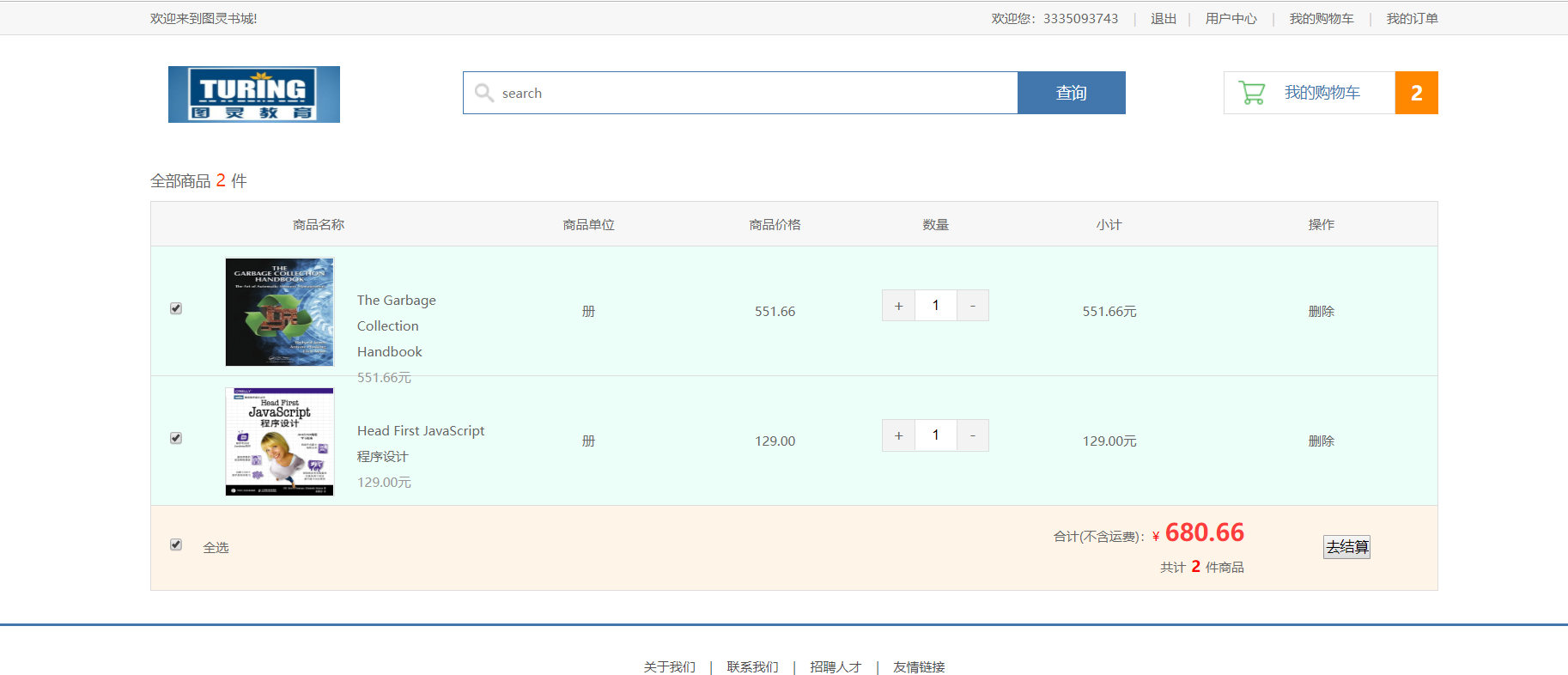
## 3.4商品详情页



把数据库中的书籍信息动态渲染到详情页，书名、简介、价格、评分、详情、评论等！

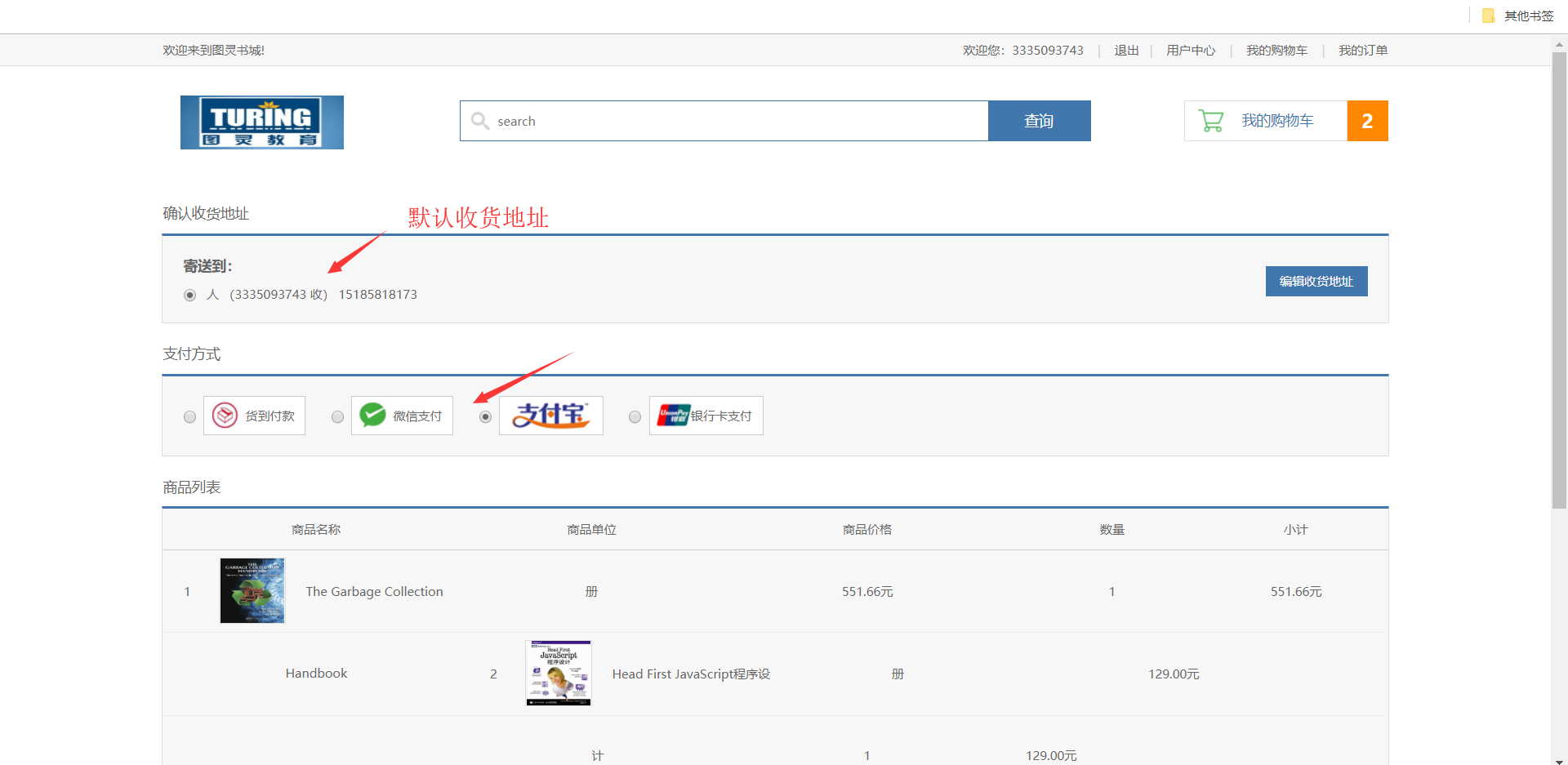
购物车数量的添加使用ajax传递数据到相应视图函数，如果成功添加到缓存redis中，则动态更新右上角购物车数量，否则alert错误信息。点击评论可切换到评论。用户登录后才能进行购买、加购物车等操作，否则弹出提示信息。

## 3.5购物车页面



用jquery计算商品总价，点击去计算，使用form表单将数据post到用户订单页面

## 3.6订单提交页



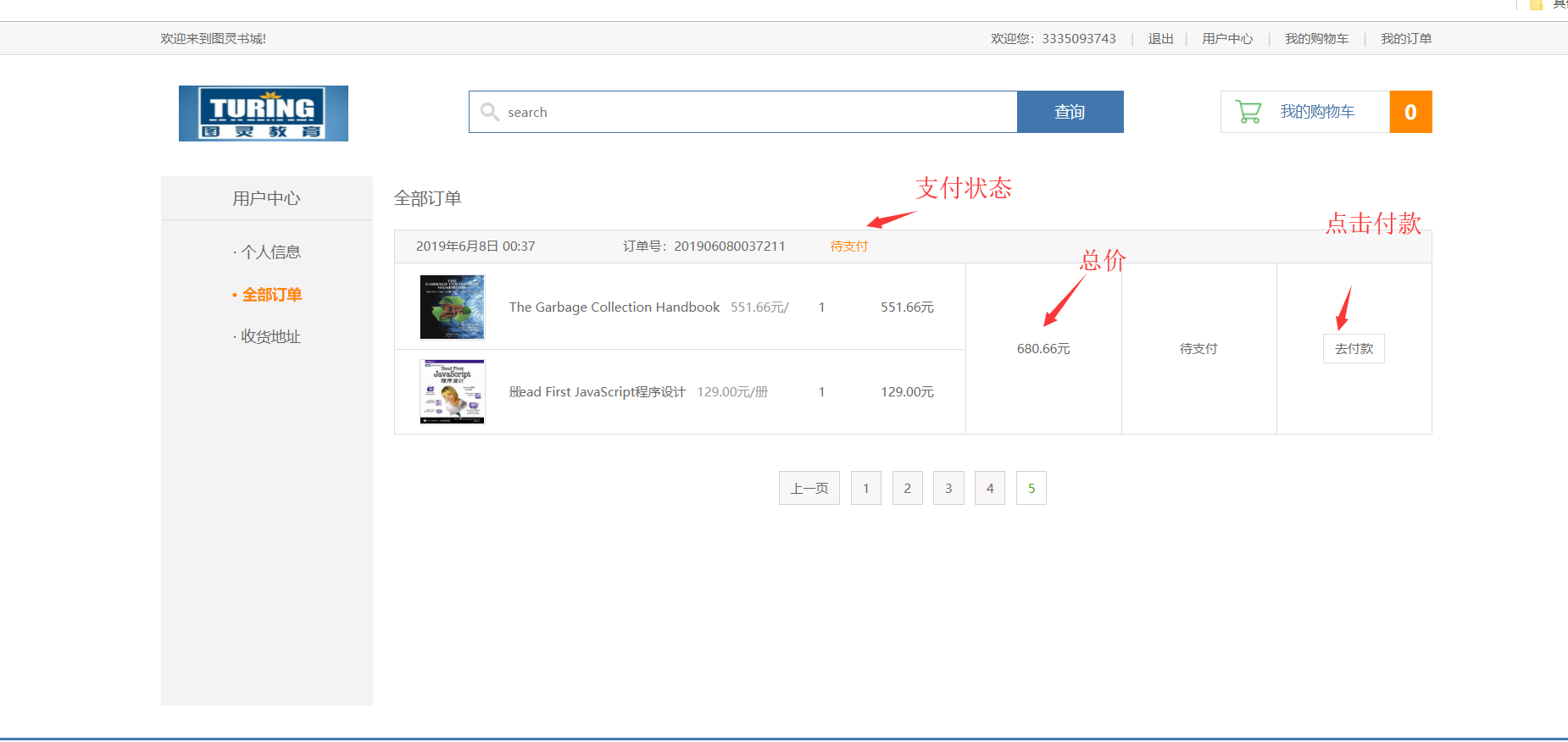
默认收获地址，没有则显示无，点击右侧按钮可编辑。

支付方式现阶段只接入了支付宝接口。点击提交订单，会发起ajax请求提交订单数据，由视图函数处理，如果无异常则会跳转到用户的订购页。

注意，提交订单要用到事务，订购提交成功则在数据库生成订购表和订单商品表，同时删除redis中的购物车中对应商品信息。反之，由于各种原因可能造成订单添加到数据库时失败，此时需要回滚到提交前的状态，详细代码和注释如下：

1. # 提交订单，需要向两张表中添加信息
2. # s\_order\_info:订单信息表 添加一条
3. # s\_order\_books: 订单商品表， 订单中买了几件商品，添加几条记录
4. # 前端需要提交过来的数据: 地址 支付方式 购买的商品id,当然还有从session中拿到的用户id
6. # 1.向订单表中添加一条信息
7. # 2.遍历向订单商品表中添加信息
8. # 2.1 添加订单商品信息之后，增加商品销量，减少库存
9. # 2.2 累计计算订单商品的总数目和总金额
10. # 3.更新订单商品的总数目和总金额
11. # 4.清除购物车对应信息
13. # 事务:原子性:一组sql操作，要么都成功，要么都失败。
14. # 开启事务: begin;
15. # 事务回滚: rollback;
16. # 事务提交: commit;
17. # 设置保存点: savepoint 保存点;
18. # 回滚到保存点: rollback to 保存点;
19. **from** django.db **import** transaction
21. # transaction.atomic装饰器可以保证该函数中所有的数据库操作都在一个事务中
22. @transaction.atomic
23. **def** order\_commit(request):
24. '''''生成订单'''
25. # 验证用户是否登录
26. **if** **not** request.session.has\_key('islogin'):
27. **return** JsonResponse({'res': 0, 'errmsg': '用户未登录'})
28. # 接收数据
29. addr\_id = request.POST.get('addr\_id')
30. pay\_method = request.POST.get('pay\_method')
31. books\_ids = request.POST.get('books\_ids')
33. # 进行数据校验
34. **if** **not** all([addr\_id, pay\_method, books\_ids]):
35. **return** JsonResponse({'res': 1, 'errmsg': '数据不完整'})
37. **try**:
38. addr = Address.objects.get(id=addr\_id)
39. **except** Exception as e:
40. # 地址信息出错
41. **return** JsonResponse({'res': 2, 'errmsg': '地址信息错误'})
43. **if** int(pay\_method) **not** **in** OrderInfo.PAY\_METHODS\_ENUM.values():
44. **return** JsonResponse({'res': 3, 'errmsg': '不支持的支付方式'})
46. # 订单创建
47. # 组织订单信息
48. passport\_id = request.session.get('passport\_id')
49. # 订单id: 日期+用户的id
50. order\_id = datetime.now().strftime('%Y%m%d%H%M%S') + str(passport\_id)
51. # 运费
52. transit\_price = 0
53. # 订单商品总数和总金额
54. total\_count = 0
55. total\_price = 0
57. # 创建一个保存点，这是一个状态，还没有向数据库中写入任何记录,遇到错误可以回滚到这个状态
58. sid = transaction.savepoint()
59. #试着添加一条记录
60. **try**:
61. # 向订单信息表中添加一条记录
62. order = OrderInfo.objects.create(order\_id=order\_id,
63. passport\_id=passport\_id,
64. addr\_id=addr\_id,
65. total\_count=total\_count,
66. total\_price=total\_price,
67. transit\_price=transit\_price,
68. pay\_method=pay\_method)
70. # 向订单商品表中添加订单商品的记录
71. #前端Ajax传过来的是以逗号分隔的字符串
72. books\_ids = books\_ids.split(',')
74. conn = get\_redis\_connection('default')
75. cart\_key = 'cart\_%d' % passport\_id
77. # 遍历获取用户购买的商品信息
78. **for** id **in** books\_ids:
79. books = Books.objects.get\_books\_by\_id(books\_id=id)
81. #商品可能被别人买了
82. **if** books **is** None:
83. transaction.savepoint\_rollback(sid)
84. **return** JsonResponse({'res': 4, 'errmsg': '商品信息错误'})
86. # 获取用户购买的商品数目
87. count = conn.hget(cart\_key, id)
89. # 判断商品的库存
90. **if** int(count) > books.stock:
91. transaction.savepoint\_rollback(sid)
92. **return** JsonResponse({'res': 5, 'errmsg': '商品库存不足'})
94. # 创建一条订单商品记录
95. OrderBooks.objects.create(order\_id=order\_id,
96. books\_id=id,
97. count=count,
98. price=books.price)
100. # 增加商品的销量，减少商品库存
101. books.sales += int(count)
102. books.stock -= int(count)
103. #提交到数据库
104. books.save()
106. # 累计计算商品的总数目和总额
107. total\_count += int(count)
108. total\_price += int(count) \* books.price
110. # 更新订单的商品总数目和总金额
111. order.total\_count = total\_count
112. order.total\_price = total\_price
113. #提交到数据库
114. order.save()
115. **except** Exception as e:
116. # 操作数据库出错，进行回滚操作，这个失败的原因有很多，可能是系统引起的
117. transaction.savepoint\_rollback(sid)
118. **return** JsonResponse({'res': 7, 'errmsg': '服务器错误'})
120. # 清除购物车对应记录，从缓存中删除
121. conn.hdel(cart\_key, \*books\_ids)
123. # 事务提交
124. transaction.savepoint\_commit(sid)
125. # 返回应答
126. **return** JsonResponse({'res': 6})

## 3.7用户订单页



仍然采用分页展示。点击付款后发起通过jquery获取前端数据发起ajax请求到视图函数order\_pay(request)。在视图函数中向支付宝发起支付请求。注意是沙箱环境，支付成功后支付宝会根据return\_url参数返回用户订单页面，此时用户订单显示状态由待支付变为待发货。右侧去付款变为查看物流，这里并没有实现物流模块。

代码及说明如下：

1. **from** alipay **import** AliPay
2. @login\_required
3. **def** order\_pay(request):
4. '''''订单支付'''
6. # 接收订单id
7. order\_id = request.POST.get('order\_id')
9. # 数据校验
10. **if** **not** order\_id:
11. **return** JsonResponse({'res': 1, 'errmsg': '订单不存在'})
13. **try**:
14. order = OrderInfo.objects.get(order\_id=order\_id,
15. status=1,
16. pay\_method=3)
17. **except** OrderInfo.DoesNotExist:
18. **return** JsonResponse({'res': 2, 'errmsg': '订单信息出错'})

21. # 用的txt文件保存密钥，不知道会不会出错
22. app\_private\_key\_path = os.path.join(settings.BASE\_DIR, 'order/app\_private\_key.txt')
23. alipay\_public\_key\_path = os.path.join(settings.BASE\_DIR, 'order/alipay\_public\_key.txt')
25. app\_private\_key\_string = open(app\_private\_key\_path).read()
26. alipay\_public\_key\_string = open(alipay\_public\_key\_path).read()
27. # 和支付宝进行交互
28. alipay = AliPay(
29. appid="2016092900620264", # 应用id
30. app\_notify\_url=None,  # 默认回调url
31. app\_private\_key\_string=app\_private\_key\_string,
32. alipay\_public\_key\_string=alipay\_public\_key\_string,  # 支付宝的公钥，验证支付宝回传消息使用，不是你自己的公钥,
33. sign\_type = "RSA2",  # RSA 或者 RSA2
34. debug = True,  # 默认False，False表示生产环境
35. )
37. # 电脑网站支付，需要跳转到https://openapi.alipaydev.com/gateway.do? + order\_string
38. #总费用
39. total\_pay = order.total\_price + order.transit\_price # decimal
40. return\_url=request.POST.get('pwd\_url')
41. **print**(return\_url)
42. order\_string = alipay.api\_alipay\_trade\_page\_pay(
43. out\_trade\_no=order\_id, # 订单id
44. total\_amount=str(total\_pay), # Json传递，需要将浮点转换为字符串
45. subject='图灵书城%s' % order\_id,
46. return\_url=return\_url,
47. notify\_url=None  # 可选, 不填则使用默认notify url
48. )
49. # 返回应答
50. pay\_url = settings.ALIPAY\_URL + '?' + order\_string
51. **return** JsonResponse({'res': 3, 'pay\_url': pay\_url, 'message': 'OK'})

注意要pip安装支付宝提供给python的接口：python-alipay-sdk,这不是支付宝官方提供的，支付宝官方只提供了java和php的sdk及demo。

支付宝开放平台开发文档：<https://docs.open.alipay.com/catalog>

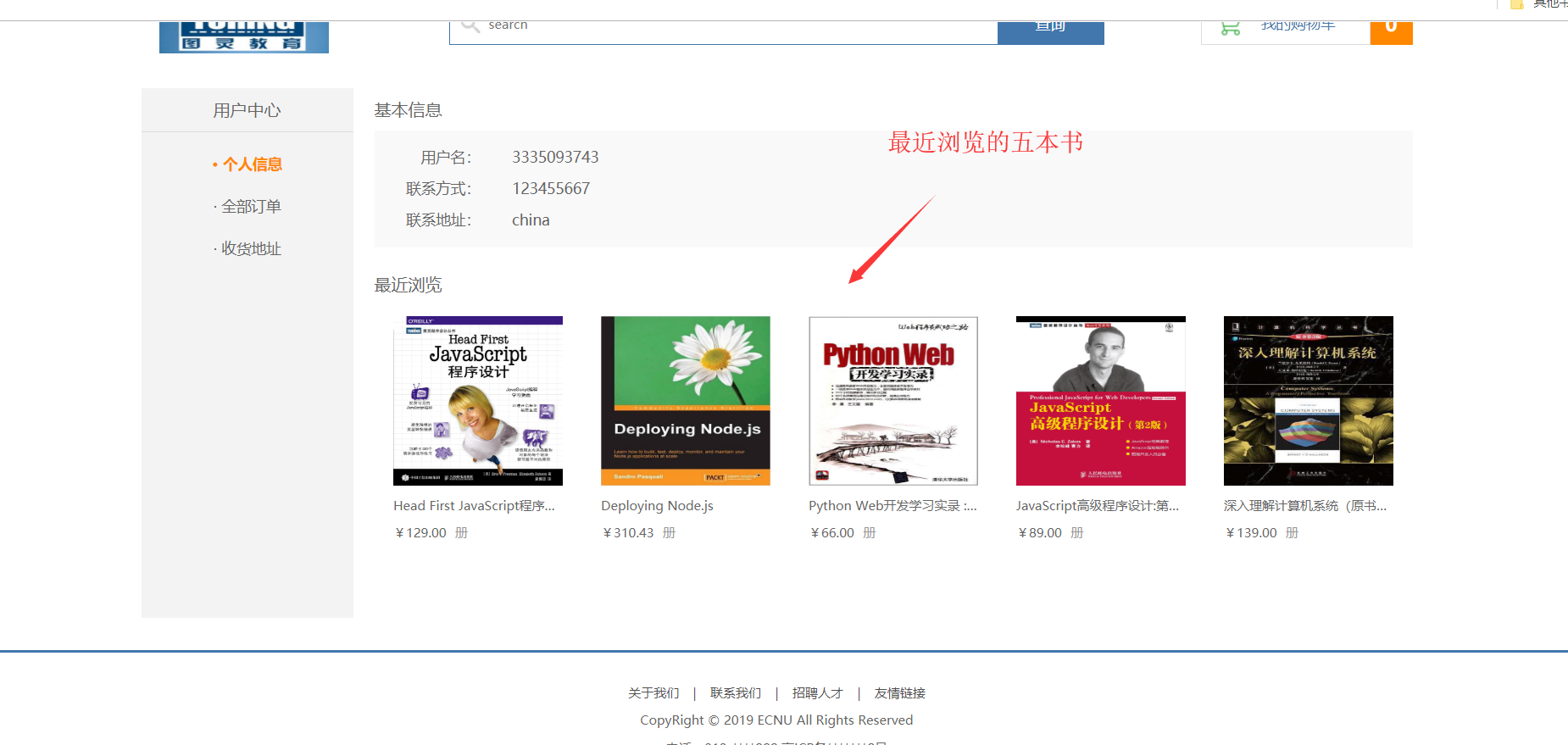
Python-alipay-sdk说明文档：

<https://github.com/fzlee/alipay/blob/master/README.zh-hans.md>

## 3.8用户中心

用户中心包含了个人信息、全部订单和收获地址。

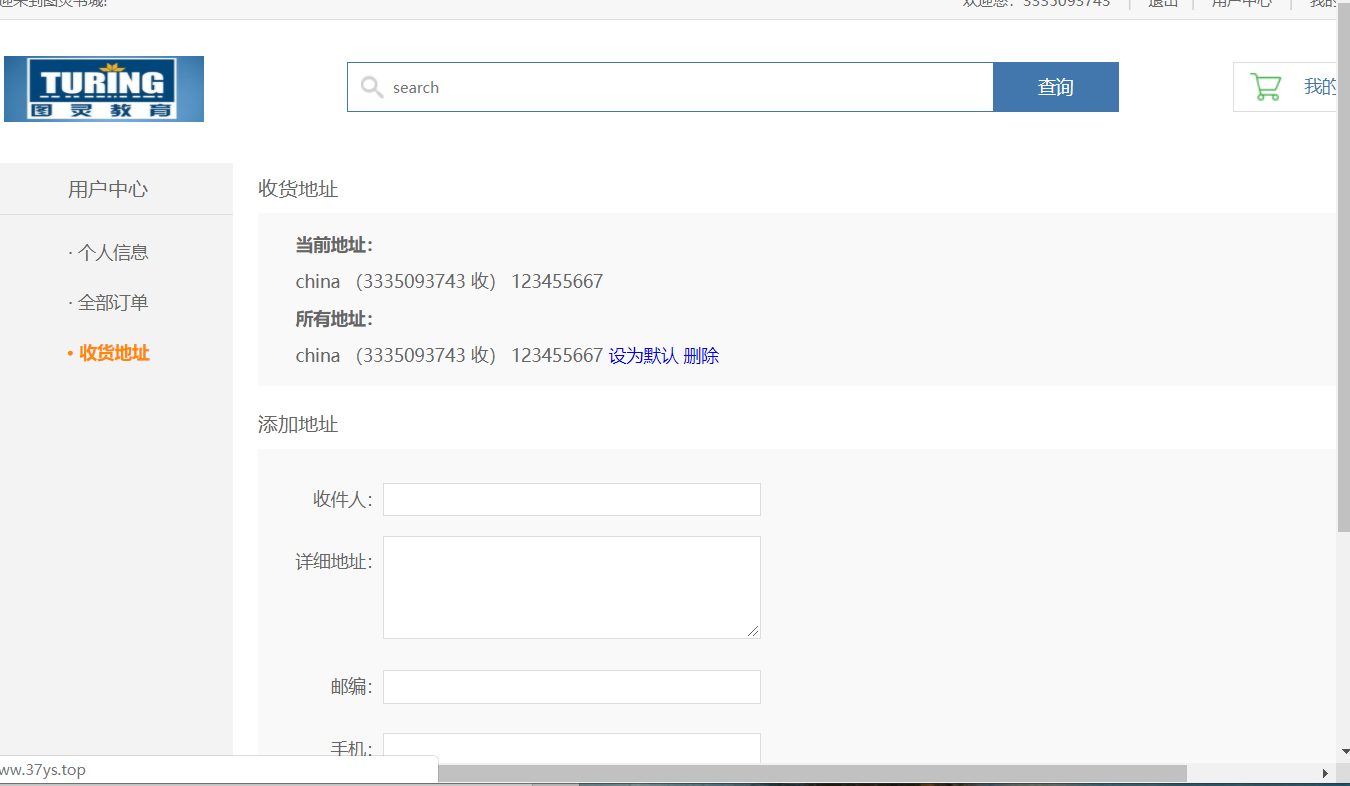
个人信息中会展示用户最近浏览的五本书：



最近浏览功能采用redis的list数据结构实现：redis缓存中记录了用户浏览的书籍id列表，通过id查询数据库商品表获取书籍信息。核心代码如下：

1. '''''用户中心-信息页'''
2. passport\_id = request.session.get('passport\_id')
3. # 获取用户的基本信息
4. addr = Address.objects.get\_default\_address(passport\_id=passport\_id)
6. # 获取用户的最近浏览信息
7. con = get\_redis\_connection('default')
8. key = 'history\_%d' % passport\_id
9. # 取出用户最近浏览的5个商品的id
10. history\_li = con.lrange(key, 0, 4)
11. books\_li = []
12. **for** id **in** history\_li:
13. books = Books.objects.get\_books\_by\_id(books\_id=id)
14. books\_li.append(books)

收货地址：



可以添加地址，点击设为默认会将当前默认地址的is\_default字段置为False，将被点击地址的is\_default置为True。点击删除会逻辑删除该地址，即将is\_delete字段设置为True。

## 3.8全局搜索功能

这里重点说一下：

要实现检索功能，首先需要一个检索引擎，django一般使用whoosh，sphinx，xapian，后两者是基于C的，效率高适合大项目，whoosh引擎基于python，适合django，所以检索引擎我就采用了whoosh，主要还是因为它的易用性和简单。

再一个是分词的问题，检索本质是把用户输入的内容对数据库进行模糊匹配，whoosh引擎默认使用正则进行分词，这在英文的世界里理所应当。但我要检索中文，那就要用到中文分词组件了，这里我选择强大的 “jieba”。

介绍一个app，haystack，它是django的一个开源搜索框架，所以也可以说django-haystack, 它是一个同时支持whoosh，solr，Xapian，Elasticsearc四种全文检索引擎的第三方app，如果你不想使用whoosh，可以在不改变代码的情况下换为其他引擎！

首先pip安装whoosh, jieba, django-haystack，然后haystack是一个app，注册到installed app中。

我们要使用haystack的whoosh引擎，所以我们可以去 /haystack/backends 目录找到 whoosh\_backend.py文件 ，它就是whoosh引擎文件，但是使用的是正则匹配，我们要改为中文分词。基于不要改库文件的原则，我们把它复制一份，文件名改为 whoosh\_cn\_backend.py ，为了方便以后迁移，把 whoosh\_cn\_backend.py 拷贝到books目录。好了，现在我们对其进行修改，使之实现中文匹配，修改也很简单：

* 引入中文分词器：from jieba.analyse import ChineseAnalyzer
* 把analyzer=StemmingAnalyzer()改为analyzer=ChineseAnalyzer()

现在中文分词的引擎我们有了，接下来就是告诉系统我们使用了这个引擎，

1. #全文检索配置
2. HAYSTACK\_CONNECTIONS={
3. 'default':{
4. #使用whoosh引擎
5. 'ENGINE':'books.whoosh\_cn\_backend.WhooshEngine',
6. # 索引文件路径
7. 'PATH': os.path.join(BASE\_DIR, 'whoosh\_index'),
8. }
9. }
10. # 当添加、修改、删除数据时，自动生成索引
11. HAYSTACK\_SIGNAL\_PROCESSOR = 'haystack.signals.RealtimeSignalProcessor'
13. HAYSTACK\_SEARCH\_RESULTS\_PER\_PAGE = 6 # 指定搜索结果每页的条数

然后配置项目路由：

1. path('search/',include('haystack.urls')),

在需要检索books的app目录下建立一个search\_indexes.py，文件名必须是这个

1. **from** haystack **import** indexes
2. **from** books.models **import** Books

5. # 指定对于某个类的某些数据建立索引, 一般类名:模型类名+Index
6. **class** BooksIndex(indexes.SearchIndex, indexes.Indexable):
7. # 指定根据表中的哪些字段建立索引:比如:商品名字 商品描述
8. text = indexes.CharField(document=True, use\_template=True)
10. **def** get\_model(self):
11. **return** Books
13. **def** index\_queryset(self, using=None):
14. **return** self.get\_model().objects.all()

在目录“templates/search/indexes/books/”下创建“books\_text.txt”文件，必须要以这种格式

1. {{ object.name }} # 根据书籍的名称建立索引
2. {{ object.desc }} # 根据书籍的描述建立索引
3. {{ object.detail }} # 根据书籍的详情建立索引

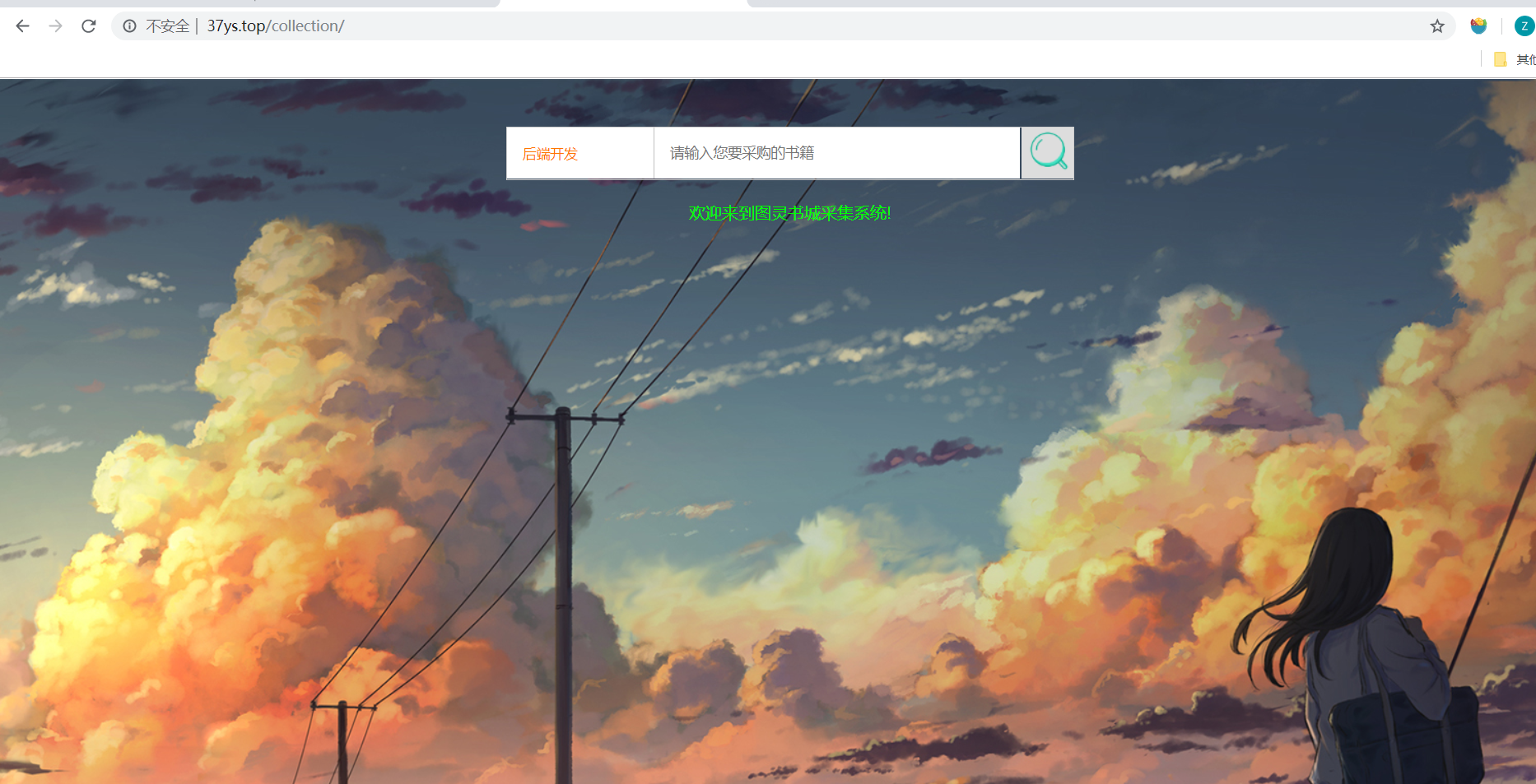
在目录“templates/search/”下建立search.html。 这是搜索结果的展示页。执行命令，生成索引：python manage.py rebuild\_index。

完结撒花！

## 3.9书籍采集模块—豆瓣爬虫

一开始我尝试通过关键词爬取豆瓣书籍，以关键词java为例，请求url：<https://book.douban.com/subject_search?search_text=java> ，但一开始根本就拿不到正确的响应。后来在chrome开发者工具中一看，好家伙！前后端分离！查看豆瓣robots协议，果然subject\_search目录下禁止爬虫。普通爬虫不行，selenium模拟浏览器成本太高！一度陷入僵局。网上爬虫教程大多是豆瓣之类的，那么他们是怎么爬的呢？原来几乎所有人都是通过tag爬取的！tag在豆瓣并没有做反爬虫限制。

基于可复用的理念，于是写了个爬虫接口：



代码在books.views.py中，对于网页文档的解析全部采用Beautiful Soup[[10]](#footnote-10)，相比于xpath正则解析简单许多。Python爬虫确实方便，但它的多线程似乎是假的，远没有java强大，鱼和熊掌不可兼得！

# 四、项目部署

## 4.1 Nginx+uwsgi部署django

本来代码已经写完了，并且能在本地利用django自带的开发服务器跑起来，但有一种神秘的力量把我推入了ngnix部署项目的大坑。虽然熬了两天夜，但见识到ngnix服务器性能的那一刻，一切都值了！

“很多做开发的同学把部署这件事情看成是运维同学的工作，这种看法是完全错误的。首先，最近流行[DevOps](http://zh.wikipedia.org/wiki/DevOps)理念，就是说，开发和运维要变成一个整体。其次，运维的难度，其实跟开发质量有很大的关系。代码写得垃圾，运维再好也架不住天天挂掉。最后，DevOps理念需要把运维、监控等功能融入到开发中。你想服务器升级时不中断用户服务？那就得在开发时考虑到这一点。”

目前主流的web服务器毫无疑问是ngnix和apache，作为后起之秀，ngnix的市场份额逐年增长，在国内越来越多人建站推荐LNMP,

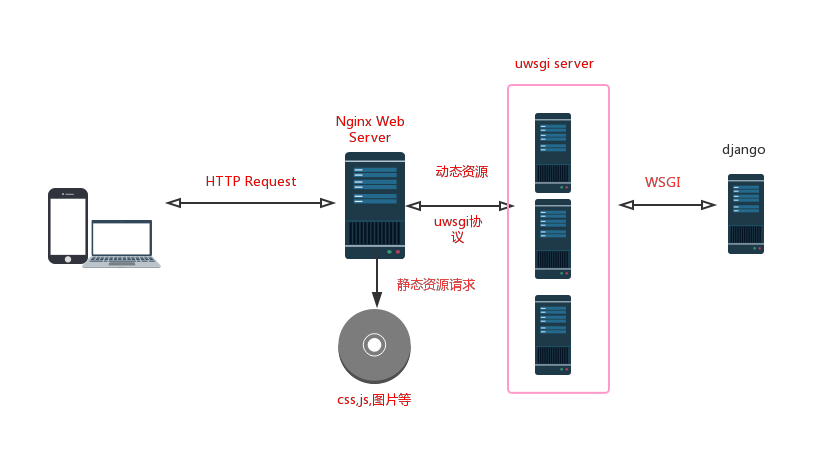
相比于apache，ngnix又以下优点：轻量级、比apache占用更少的内存及资源；能抗高并发，由于nginx处理任务是异步非阻塞的，而apache则是阻塞型的，在高并发下nginx能保持低资源消耗高性能；高度模块化的设计，编写模块相对简单。基于以上原因，选择ngnix部署项目

部署项目要有一个应用级的服务器Application Server，从严格意义上讲，Apache和Ngnix应该叫做HTTP Server。部署java项目一般用tomcat，而python目前一般采用uwsgi。

uwsgi: 首先，**WSGI**（Web Server Gateway Interface)规范[[11]](#footnote-11)，规定了Python Web应用和Python Web服务器之间的通讯方式。目前主流的Python Web框架，比如Django，Flask，Tornado等都是基于这个规范实现的。uWSGI是 实现了uwsgi协议，WSGI规范和HTTP协议的 一个C语言实现的软件。但除了搞理论的，网上好像很多人都不会对uwsgi和uWSGI作严格区分。

Nginx+uwsgi的工作流程：当一个访问进来的时候，首先到 Nginx，Nginx会把请求（HTTP协议）通过uwsgi协议传递给uWSGI，uWSGI通过WSGI和Django进行通讯取到响应结果，再通过uwsgi协议发给Nginx，最终Nginx以HTTP协议发现响应给用户。

以下是我自制的整个项目的架构图：



以下是网站的ngnix配置文件：

1. # the upstream component nginx needs to connect to
2. upstream django {
3. server unix:///www/wwwroot/www.37ys.top/Book/Book.sock; # for a file socket
4. #server 127.0.0.1:8001; # for a web port socket (we'll use this first)
5. }
7. # configuration of the server
8. server {
9. # the port your site will be served on
10. listen      80;
11. # the domain name it will serve for
12. server\_name www.37ys.top 37ys.top m.37ys.top; # substitute your machine's IP address or FQDN
13. charset     utf-8;
15. # max upload size
16. client\_max\_body\_size 75M;   # adjust to taste
18. # Django media
19. location /media  {
20. alias /www/wwwroot/www.37ys.top/Book/media;  # your Django project's media files - amend as required
21. }
23. location /static {
24. alias /www/wwwroot/www.37ys.top/Book/static; # your Django project's static files - amend as required
25. }
27. # Finally, send all non-media requests to the Django server.
28. location / {
29. uwsgi\_pass  django;
30. include     /www/wwwroot/www.37ys.top/Book/uwsgi\_params; # the uwsgi\_params file you installed
31. }
32. }

还需要写一个uwsgi的配置文件以启动django项目：

1. #Book\_uwsgi.ini file
2. [uwsgi]
4. # Django-related settings
5. # the base directory (full path)
6. chdir           = /www/wwwroot/www.37ys.top/Book
7. # Django's wsgi file
8. module          = Book.wsgi
9. # the virtualenv (full path)
10. home            = /my/project/Book
12. # process-related settings
13. # master
14. master          = true
15. # maximum number of worker processes
16. processes       = 2
17. # the socket (use the full path to be safe
18. socket          = /www/wwwroot/www.37ys.top/Book/Book.sock
19. # ... with appropriate permissions - may be needed
20. chmod-socket    = 666
21. # clear environment on exit
22. vacuum          = true
23. env = DJANGO\_SETTINGS\_MODULE=Book.settings
24. safe-pidfile = /www/wwwroot/www.37ys.top/Book.pid
25. daemonize = /www/wwwroot/www.37ys.top/Book.log

关于uwsgi的资料可查看官方文档：

<https://uwsgi-docs-zh.readthedocs.io/zh_CN/latest/>

详细配置过程可参考我的博客：

<http://www.fishblogs.cn/index.php/2019/06/04/django%E9%A1%B9%E7%9B%AE%E8%BF%81%E7%A7%BB/>

## 4.2 部署后的性能优化

部署完成后，虽然可以访问网站，但内存占用达到了90%以上，一部分原因是我的vps上还同时运行了一个博客网站，但也不至于这么吃内存！甚至有好几次宕机！

优化手段是，收集django项目的静态资源，并交给Nginx处理！这样响应速度确实快了很多。还有一点，把uwsgi进程数量设置少一点，两个就行了，毕竟只有2核2G。

另外，为了提高数据库查询速度，在一些经常查询的字段上设置索引是必要的！

# 五、web安全

常见web攻击有：SQL注入、xss攻击、csrf跨站攻击访问

## 5.1 SQL注入攻击与防范

sql注入危害：非法读取、篡改、删除数据库中的数据；注入木马等。

Sql注入的防范无非就是对用户输入的参数进行检查，可以在前端，但更多的是放在后端。Python中常用参数化处理进行防范，例如：

1. sql = "INSERT INTO USER(NAME,PASSWORD) VALUES(%s,%s)"

由于本项目采用ORM框架，ORM接口是属于内部封装机制的，从外面是无法查看的，所以ORM框架是可以有效的防止住SQL注入攻击，因为SQL根本找不到接口。

为了演示SQL注入，特做了一个登录表单，url：[www.37ys.top/user/test](http://www.37ys.top/user/test)

视图层相关代码：

1. sql="SELECT \* FROM users\_passport WHERE username='{0}' AND password='{1}' AND is\_active=1".format(username,password)

所以可以这样进行注入：用户名输入 'OR 1=1#，密码任意。

其中#是注释，可以直接绕过后面的sql语句！

## 5.2 xss攻击与防范

xss，即跨站脚本攻击（Cross Site Scripting）。

危害：可以盗取各类用户，非法转账等。

正常情况下，用户提交的姓名可以正确显示在页面上，不会构成XSS攻击，比如，当用户访问以下URL：

<http://localhost/xssTest/test.php?userName=jack>，

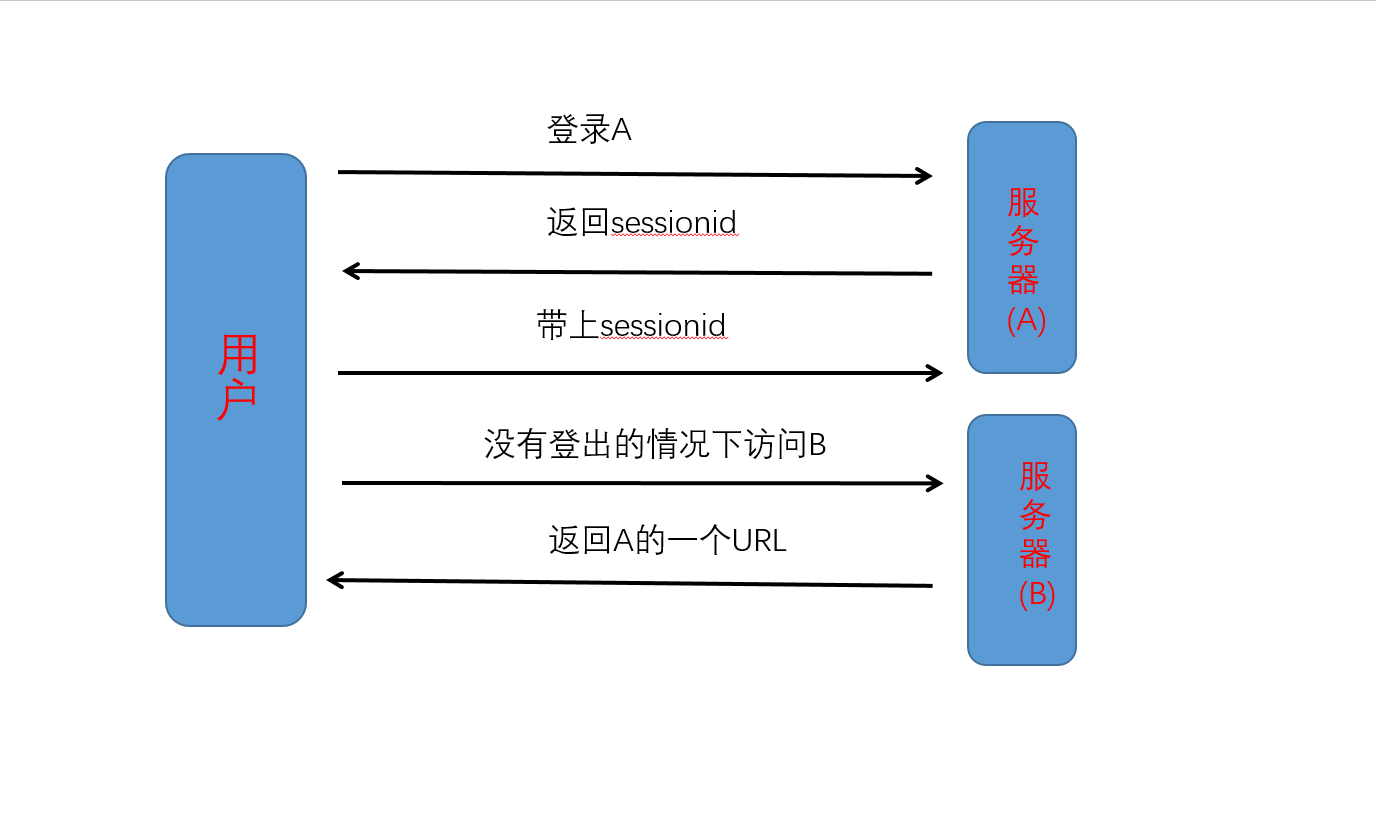
然后，我们尝试在URL中插入JavaScript代码，如：

[http://localhost/xssTest/test.php?userName=<script>window.open(http://www.baidu.com)</script](http://localhost/xssTest/test.php?userName=%3cscript%3ewindow.open(http://www.baidu.com)%3c/script)>，

页面就会跳转到百度，现在很多浏览器都内置了一些防范措施。本项目使用的whoosh检索引擎当然可以抵御这点小把戏！

## 5.3 csrf 跨站请求伪造

csrf，即Cross-site request forgery，其攻击原理如下：



防范方法：django默认禁止跨站访问，要跨站访问，必须在表单中加一个{% csrf\_token %}，这个token是一次性的，每次请求都会在前端表单中生成一个随机的token，访问把这个token带到后端进行校对。这样就可以防止其他网站跨站访问本网站！

# 六、项目总结

这个项目花了我大把的时间。从数据库设计到部署上线，白天写bug，深夜debug。之前零碎地学了一些web方面的东西：html，css，js，jq，ajax，python爬虫，python数据处理，linux，计算机网络等，这个学期接触了数据库，就尝试着做一个项目把这些都串起来，django框架满足了我的需求。详尽的官方文档，强大的社区，都给了我莫大的帮助。开发过程中遇到了上百个问题，大部分通过debug和文档解决了，还有一些问题通过google和stackflow pass掉了！现在的感觉就是：web项目信手拈来！

1. 传送门：<http://www.planetb.ca/syntax-highlight-word?tdsourcetag=s_pcqq_aiomsg> [↑](#footnote-ref-1)
2. 有关python PE8规范的详细内容请参考：

   <https://legacy.python.org/dev/peps/pep-0008/> [↑](#footnote-ref-2)
3. 官网链接：<https://git-scm.com/>，新手推荐教程：<https://www.liaoxuefeng.com/wiki/896043488029600> [↑](#footnote-ref-3)
4. [↑](#footnote-ref-4)
5. 官方网站：<https://redis.io/> [↑](#footnote-ref-5)
6. Redis基础教程：<https://www.w3cschool.cn/redis/> [↑](#footnote-ref-6)
7. Django的session：<https://docs.djangoproject.com/en/2.1/topics/http/sessions/> [↑](#footnote-ref-7)
8. 官方文档：<http://doc.ewomail.com/docs/ewomail/jianjie> [↑](#footnote-ref-8)
9. 官方文档：<http://docs.celeryproject.org/en/latest/> [↑](#footnote-ref-9)
10. 官方文档：<https://www.crummy.com/software/BeautifulSoup/bs4/doc/> [↑](#footnote-ref-10)
11. WSGI规范：<https://wsgi.readthedocs.io/en/latest/> [↑](#footnote-ref-11)