设计文档

第四组

目录

1.背景

Django 是采用 MTV 模式的框架，其目的是要实现简单快捷的网站开发，比较适合当前项目的需要。

PostgreSQL 是 Django 官方推荐的关系型数据库。

前端框架使用 Amaze UI，主要是因为找到的前端模板使用的就是这个框架。Amaze UI 是一个轻量级、Mobile first 的前端框架，具有不错的动态响应功能。

2.名词解释

- Django：python 的 web 框架
- tag/taggit：基于 django-taggit 模块，为类添加 tags = TaggableManager()语句后可以为类保存标签。
- redis：redis 是一个开源的、使用 C 语言编写的、支持网络交互的、可基于内存也可持久化的 Key-Value 数据库。
- 导入：python 的 import 语句

3.设计目标

3.1 实现的功能

- 用户的注册，登陆，登出，账号找回功能：用户通过邮箱注册账号；通过用户名和密码登陆 ,通过登出来实现会话撤销 ,通过 Django 自带的 Authentication backends，即用户认证框架实现用户的密码找回。
- 商品浏览功能：基于基础的数据库查找实现。
- 商品搜索功能：基于药品名字和品牌名字的内容以数据库的 “like”语句即模糊查询实现；适应症内容需要补全成为 tag，再基于 taggit 模块的标签搜索功能实现商品内容的返回。
- 购物车功能：基于 redis 实现数据库的存储。基于异步 Ajax 实现前端页面的局部刷新。
- 订单功能：基于表单和购物车数据的结合。

3.2 性能指标

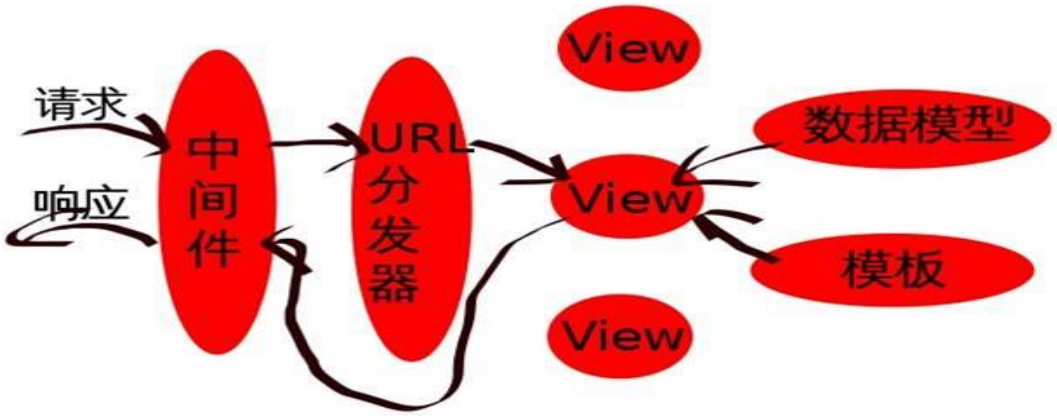
空间：redis 运行在服务器的内存中，占有一定的存储资源但是速度快于传统数据库。

时间：当商品数量和标签数量上升时搜索的时间复杂度上升很快，需要后期的改进。

4.模式设计

4.1MTV 设计模式

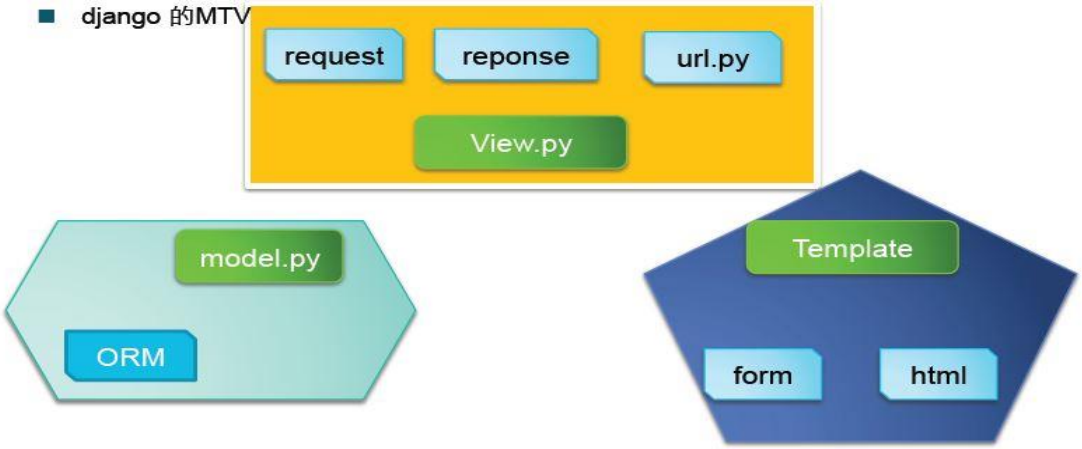
M 代表模型（Model）：负责业务对象和数据库的关系映射(ORM)。
T 代表模板 (Template)：负责如何把页面展示给用户(html)。
V 代表视图（View）：负责业务逻辑，并在适当时候调用 Model 和 Template。
除了以上三层之外，还需要一个 URL 分发器，它的作用是将一个个 URL 的页面请求分发给不同的 View 处理，View 再调用相应的 Model 和 Template，MTV 的响应模式如下所示：



1. Web 服务器（中间件）收到一个 http 请求
2. Django 在 URLconf 里查找对应的视图(View)函数来处理 http 请求
3. 视图函数调用相应的数据模型来存取数据、调用相应的模板向用户展示页面
4. 视图函数处理结束后返回一个 http 的响应给 Web 服务器
5. Web 服务器将响应发送给客户端

4.2 文件存放

Django 的 MTV 模式相对应的 app 的 python 文件如下：



而 Django 项目公用的模板文件和静态文件存放在项目根目录下。

图片等媒体文件存放在项目根目录下的 media 文件下，需要修改 setting 文件自动生成。

5.模块设计

5.1 模块详述

5.1.1account 模块

即用户认证模块：基于 Django 的认证（authentication）框架，其包含了一些开箱即用的表单（forms）和视图（views）。

使用 Django 以下的视图（views）来处理认证（authentication）：
login：操作表单（form）中的登录然后登录一个用户。
logout：登出一个用户。
logout_then_login：登出一个用户然后重定向这个用户到登录页面。

使用以下视图（views）来操作密码修改：
password_change:操作一个表单（form）来修改用户密码。
password_change_done:当用户成功修改他的密码后提供一个成功提示页面。

使用以下视图（views）允许用户重置他们的密码：
password_reset:允许用户重置他的密码。它会生成一条带有一个 token 的一次性使用链接然后发送到用户的邮箱中。
password_reset_done:告知用户已经发送了一封可以用来重置密码的邮件到他的邮箱中。
password_reset_complete:当用户重置完成他的密码后提供一个成功提示页面。

具体步骤：

1. 在 account 的目录下的 urls.py 文件以如下格式创建路由：path('logout-then-login/', logout_then_login, name='logout_then_login')。即 path('路由路径', 'Django 的默认视图', '路由名称')。
2. 在相应路径存放相应名称的 html 模板文件。

5.1.2product 模块

Model 层：

```
class Product(models.Model):
    name = models.CharField(max_length=200, db_index=True)
    slug = models.SlugField(max_length=200, db_index=True)
    image = models.ImageField(upload_to='products/%Y/%m/%d', blank=True)
    description = models.TextField(blank=True)
```

```
    price = models.DecimalField(max_digits=10, decimal_places=2)
    stock = models.PositiveIntegerField(default=100)
    available = models.BooleanField(default=True)
    created = models.DateTimeField(auto_now_add=True)
    updated = models.DateTimeField(auto_now=True)
    display = models.CharField(max_length=100, null=True, blank=True)
    brand = models.CharField(max_length=100, null=True, blank=True)
    sales = models.PositiveIntegerField(default=100)

    to_category = models.ManyToManyField(Category, related_name='c_products', blank=True, null=True)

    tags = TaggableManager()
```

View 层：

主要分为 product_list 页面和 product_detail 页面。其中 product_list 页面分为 3 种形式：

1. 基于主页的商品展示通过和 category 模块中的类形成的多对多的联系提取。
2. 基于网页链接跳转的商品展示由 taggit 模块实现。
3. 基于搜索的商品展示通过药品名字和品牌名字的内容以数据库的 “like”语句即模糊查询和适应症内容补全成为 tag，再基于 taggit 模块的标签搜索功能实现商品内容的返回。

product_detail 页面包含异步添加或删除商品的的异步 JavaScript 代码，实现添加购物车的局部刷新功能。

5.1.3category 模块

Model 层：

```
class Category(models.Model):
    title = models.CharField(max_length=15)

    floor = models.PositiveIntegerField(db_index=True)


class Kinds(models.Model):
    description = models.CharField(max_length=200)
    to_category = models.ForeignKey(Category, related_name='c_kinds', on_delete=models.CASCADE)
    tags = TaggableManager()
```

本模块只含有目录的模型层，主要实现主页的分类功能。其中 category 通过多对多对应特定商品使之绑定在主页中形成 floor 的展示区。

5.1.4cart 模块

存储形式：基于 redis 的内存高速访问。由于 redis 数据库的特殊形式，在 models 文件中不建立模型。因为每个用户都要有一条数据，所以选择 redis 的 hash 数据类型，具体形式是 cart_用户 id:{商品 1id：数量， 商品 2id：数量} 其中条目数可以用 hlen 来获取。

传输形式：在数据传输方面，需要反复在前端与后端调用的数据是商品 id（有了 id，商品的其他信息就都能从数据库里查到。）和购买的数量。前后端传输数据使用 AJAX，实现页面的局部刷新。

大致的伪代码：

前端 AJAX：

```
$('购物车').click(
Function：
  # 获取用户购买的数量
  count = $(相关的标签).val()
  # 获取商品 Id
  sku_id = $(this).attr(' sku_id ')
  # django 的 csrf 验证，在网页随便一个地方添加{% csrf_token %}
  csrf = $('input[name="csrfmiddlewaretoken"]').val()
  params = {'sku_id':sku_id, 'count':count, 'csrfmiddlewaretoken':csrf}
  $.post('/cart/add', params, function (data) {
  # 显示条目数
  $('#show_count').html(data.cart_count);
```

View 层：

添加购物车视图：后台接受数据，并返回 cart_count，条目数。

```
# 如果用户买过该商品，直接在之前的数量上添加，没有买过，那么建立相应的键值对。
cart_count = conn.hlen(cart_key)
# 返回应答
return JsonResponse({'cart_count':cart_count, 'message':'添加成功'})
```

购物车详情视图：

```
def get(self, request):
    user = request.user

    # 获取 redis 中对应用户的购物车记录 cart_用户 id: {'sku_id':商品数目}

    # 获取购物车中商品 id 对应的商品信息
    for sku_id, count in cart_dict.items():
        # 根据 sku_id 获取商品的信息
        sku = GoodsSKU.objects.get(id=sku_id)
        # 计算商品的小计
        amount = sku.price*int(count)
        # 动态给 sku 对象增加属性 amount 和 count，分别保存商品的小计和购物车中商品的数量
        sku.amount = amount
        sku.count = count
        # 添加 sku
        skus.append(sku)
        # 累加计算商品的总件数和总价格

        # 组织模板上下文

        # 使用模板
        return render(request, 'cart.html', context)
```

5.1.5order 模块

Model 层：

```
class Order(models.Model):
    first_name = models.CharField(max_length=50)
    last_name = models.CharField(max_length=50)
    email = models.EmailField()
    address = models.CharField(max_length=250)
    postal_code = models.CharField(max_length=20)
    city = models.CharField(max_length=100)
    created = models.DateTimeField(auto_now_add=True)
    updated = models.DateTimeField(auto_now=True)
    paid = models.BooleanField(default=False)


class OrderItem(models.Model):
    order = models.ForeignKey(Order, related_name='items')
    product = models.ForeignKey(Product, related_name='order_items')
```

price = models.DecimalField(max_digits=10, decimal_places=2)
quantity = models.PositiveIntegerField(default=1)

View 层：

GET 请求：实例化表单然后渲染模板。

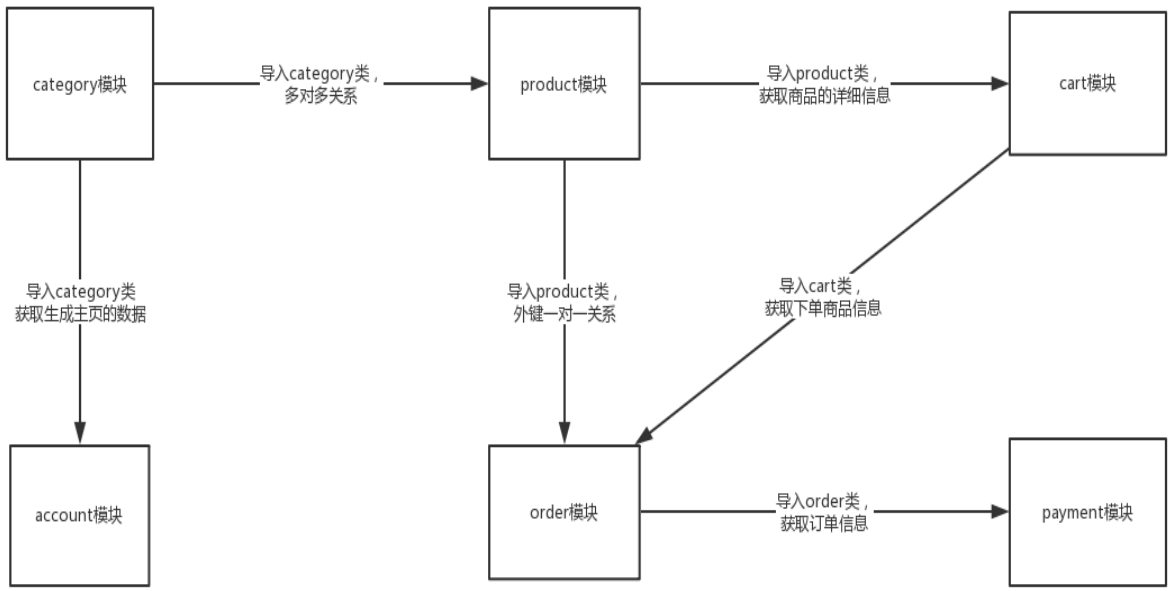
POST 请求：验证提交的数据。如果数据是合法的，创建一个新的 Order 实例。然后把它保存进数据库中，之后再把它保存进 order 变量里。在创建 order 之后，我们将迭代购物车的物品然后为每个物品创建 OrderItem。最后，清空购物车。

5.1.6payment 模块

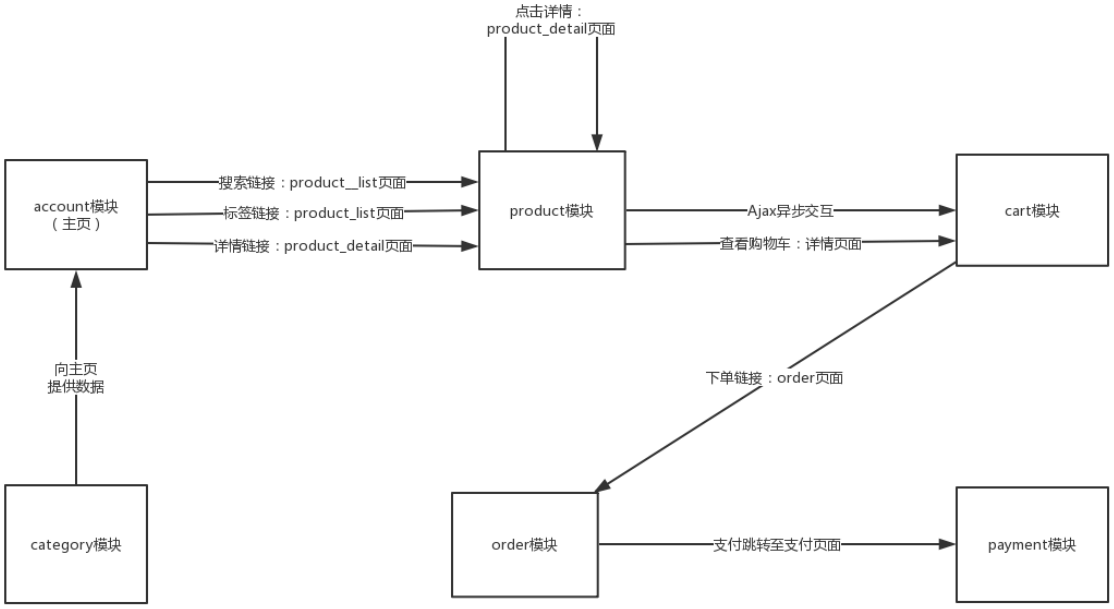
主要是视图层，调用第三方模块或接口。

5.2 模块接口分析

5.2.1 业务端耦合



5.2.2 视图端耦合（路由层）



6.系统环境

6.1 相关软件及硬件

- Windows Server 2012
- Python 3.6.0
- Redis 4.0.9
- PostgreSQL 9.5.4

6.2 python 所需模块

- Django 2.0.3
- Django-taggit 0.22.2
- Redis 2.10.6
- Pillow 4.3.0