第3章

电子商务网站的实现

3.1 需求描述

3.1.1 用户信息模块

用户信息模块包括“用户信息的注册”“用户登录”“显示用户信息”“用户密码的修改”。

(1) 注册信息要输入用户名、密码和邮箱。注册信息要求用户名必须唯一,如果用户名在数据库中已经存在,就会显示相应的错误提示信息。

(2) 在用户登录的时候,如果用户名和密码输入有误,就必须提示相应的错误信息。

(3) 用户登录程序后,应该允许用户查看自己的用户信息和收货信息。

(4) 允许修改密码,修改用户密码的时候,必须提供旧密码、新密码和新密码的确认信息。下列情况应该给出相应的错误提示信息。

① 旧密码不正确。

② 新密码与旧密码相同。

③ 新密码与新密码的确认信息不一致。

3.1.2 商品信息模块

商品信息模块包括“商品信息的维护”“商品概要信息的分页显示”“根据商品名称的模糊查询”“对某一条商品显示其详细信息”。

(1) “商品信息的维护”:包括增加、修改和删除操作,是利用 Django 的后台完成的。

(2) “商品概要信息的分页显示”:包括显示商品信息的 id、名称、价钱以及查看详情和放入购物车的操作链接。

(3) “根据商品名称的模糊查询”:通过商品名称的模糊查询实现,查询结果界面同概要信息,也需要实现分页功能。

(4) “对某一条商品显示其详细信息”:除了显示名称、价钱,还要显示商品的描述、图片以及放入购物车的操作。

3.1.3 购物车模块

购物车模块包括“购物车中所有商品的显示”“添加商品进入购物车”“修改购物车中某

种商品的数量”“删除购物车中某个商品”“删除购物车中所有商品”。

(1)“购物车中所有商品的显示”通过列表实现,包括显示商品 id、商品名称、单价、商品个数以及移除的操作链接。单击“商品 id”可以查看对应的商品详细信息。

(2)“添加商品进入购物车”可以在商品车列表中进行操作,也可以在商品的详细信息中进行操作。

(3)“修改购物车中某种商品的数量”和“删除购物车中某个商品”的操作在购物车列表中进行。

订单生成后,对应购物车中的商品将被删除。

3.1.4　送货地址模块

送货地址模块包括“送货地址的显示”“送货地址的添加”“送货地址的修改”“送货地址的删除”。

(1)“送货地址的显示”可以在生成订单选择送货地址的时候,也可以在查看用户信息的时候。

(2)“送货地址的添加”可以添加当前用户账号下的一个或多个送货地址。

(3)“送货地址的修改”和“送货地址的删除”可以通过送货地址的显示页面进入。

3.1.5　订单模块

订单模块包括“显示总的订单”“显示所有订单”“删除单个订单”“删除总订单”。

(1)“显示总的订单”在订单生成完毕后显示,包括生成时间、配货地址和总价钱以及订单中每个商品的订单 id、商品名称、商品价格、个数。

(2)“显示所有订单”包括该用户下的所有订单,每个订单的显示内容同单个订单。如果这个订单没有支付,系统就会提供支付的操作链接。

(3)“删除单个订单”可以在显示单个订单内容页面中进行,也可以在显示所有订单内容页面中进行。

(4)“删除总订单”在显示单个订单或显示所有订单的页面中进行。

(5) 在单个订单和所有订单中单击“商品 id”可以查看对应的商品详细信息。

3.1.6　订单支付模块

订单确认后,可以利用各种支付平台(如支付宝、微信、网银卡等)进行支付操作。

3.2　数据 Model 设计

根据 3.1 节的需求描述进行数据模型设计。电子商务系统的系统关联图(E-R 图)如图 3-1 所示。

这里建立了 5 个对象,分别是用户(User)、地址(Address)、商品(Goods)、单个订单(Order)和总订单(Orders)。

(1) 一个用户对应多个地址,一个地址对应一个用户,所以“用户,地址”是一对多的关系,需要在地址表中建立包含指向用户表的外键。

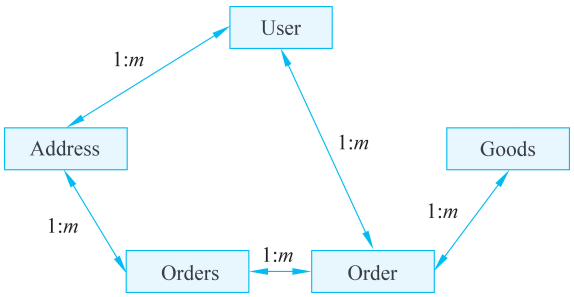

图 3-1　电子商务系统的系统关联图(E-R 图)

（2）一个地址对应多个总订单，一个总订单对应一个地址，所以“地址，总订单”是一对多的关系，需要在总订单表中建立包含指向地址表的外键。

（3）一个用户对应多个订单，一个订单对应一个用户，所以“用户，订单”是一对多的关系，需要在订单表中建立包含指向用户表的外键。

（4）一个商品对应多个单个订单，一个单个订单对应一个商品，所以“商品，单个订单”是一对多的关系，需要在单个订单表中建立包含指向商品的外键。

（5）一个总订单对应多个单个订单，一个单个订单对应一个总订单，所以“总订单，单个订单”是一对多的关系，需要在单个订单表中建立包含指向总订单的外键。

根据上述分析，建立如下 model.py 文件。

```
from django.db import models

#在这里建立模型
#用户
class User(models.Model):
    username =models.CharField(max_length=50)                    #用户名
    password =models.CharField(max_length=64)                    #密码
    email =models.EmailField()                                   #Email

    def __str__(self):
        return self.username

#商品
class Goods(models.Model):
    name =models.CharField(max_length=100)                       #商品名称
    price =models.FloatField()                                   #单价
    picture =models.FileField(upload_to ='./static/image/')      #图片
    desc =models.TextField()                                     #描述

    def __str__(self):
        return self.name
#收货地址
class Address(models.Model):
```

```
    user =models.ForeignKey(User)                                    #关联用户 id
    address =models.CharField(max_length=50)                         #地址
    phone =models.CharField(max_length=15)                           #电话

    def __str__(self):
        return self.address

#总订单
class Orders(models.Model):
    address =models.ForeignKey(Address)                              #关联送货地址 id
    create_time =models.DateTimeField(auto_now=True)                 #创建时间
    status =models.BooleanField()                                    #订单状态

    def __str__(self):
        return self.create_time

#单个订单
class Order(models.Model):
    order =models.ForeignKey(Orders)                                 #关联总订单 id
    user =models.ForeignKey(User)                                    #关联用户 id
    goods =models.ForeignKey(Goods)                                  #关联商品 id
    count =models.IntegerField()                                     #数量
```

Goods 表中 picture 使用的是 upload_to = './static/image/',也可以用 ImageField,由于 goods 是后台管理人员操作的,为了提高上传图片的性能,没有使用 ImageField 管理。upload_to ='./static/image/'表示图片上传后,放入名为./static/image/的路径。

3.3 用户信息模块

用户信息模块包括“用户信息的注册”“用户登录”“显示用户信息”“用户密码的修改”。其中,“用户信息的注册”与“用户登录”在本书第 2 章已进行了详细描述,本章将对它们进行系统的归纳与优化。数据模型如下。

```
...
#用户
class User(models.Model):
    username =models.CharField(max_length=50)      #用户名
    password =models.CharField(max_length=64)      #密码
    email =models.EmailField()                     #Email

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

3.3.1 用户注册

只有通过用户注册的用户才可以登录系统，根据需求，在这个系统中用户注册需要填写用户名、密码和 Email 地址。

1. urls.py

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

2. views.py

把所有的表单定义在一个名为 forms.py 的文件中，用户注册的表单定义如下。

```
...
from django import forms

#定义注册表单模型
class UserForm(forms.Form):
    username =forms.CharField(label='用户名',max_length=100)
    password =forms.CharField(label='密码',widget=forms.PasswordInput())
    email =forms.EmailField(label='电子邮件')
...
```

其中，

(1) username 中的 max_length=100 表示输入的字符个数最多为 100 个。

(2) password 中的 widget=forms.PasswordInput()表示文本信息为密码格式。

(3) email 中的 EmailField 表示格式为 HTML5 中的 Email 格式。

然后在 views.py 中通过 from goods.forms import UserForm 引入。下面是 views.py 中关于注册的代码。

```
...
from goods.forms import UserForm
...
#用户注册
def register(request):
    if request.method =="POST":                    #判断表单是否提交状态
        uf =UserForm(request.POST)                 #获得表单变量
        if uf.is_valid():                          #判断表单数据是否正确
            #获取表单信息
            username =(request.POST.get('username')).strip()     #获取用户名信息
            password =(request.POST.get('password')).strip()     #获取密码信息
            email =(request.POST.get('email')).strip()           #获取 Email 信息
            #查找数据库中是否存在相同的用户名
            user_list =User.objects.filter(username=username)
            if user_list:
```

```
                #如果存在,就报"用户名已经存在!"错误信息,并且回到注册页面
                 return render('register.html',{'uf':uf,"error":"用户名已经
存在!"})
            else:
                #否则将表单写入数据库
                user =User()
                user.username =username
                user.password =password
                user.email =email
                user.save()
                #返回登录页面
                uf =LoginForm()
                return render('index.html',{'uf':uf})
    else:
        #如果不是表单提交状态,就显示表单信息
        uf =UserForm()
    return render('register.html',{'uf':uf})
...
```

(1) 通过 if request.method == "POST":判断当前状态是否为表单提交状态,如果不是,就显示表单注册页面: uf = UserForm(),return render('register.html',{'uf':uf}),否则验证提交的表单信息: if uf.is_valid():。

(2) 判断表单提交是否正确,如果正确,就获取提交的信息: username = (request.POST.get('username')).strip()和 password = (request.POST.get('password')).strip()。

(3) 通过 user_list =User.objects.filter(username=username)的返回变量 user_list 是否为空判断注册的用户名是否已经被注册过,如果未注册过,就提示错误信息,否则接受提交的注册信息,通过 user.save()将其保存在数据库中。

这里特别需要注意的是,由于后面需要用到基于 requests 的接口测试,这里必须使用 request.POST.get('username')获取表单数据。

3. 模板

模板文件为 register.html,其内容如下。

```
{%load staticfiles%}
<!DOCTYPE html>
<html lang="zh-CN">
    <head>
        <meta charset="utf-8">
        <meta http-equiv="X-UA-Compatible" content="IE=edge">
<meta name="viewport" content="width=device-width, initial-scale=1">
<!--上述 3 个 meta 标签 * 必须 * 放在最前面,任何其他内容都 * 必须 * 跟随其后!-->
<meta name="description" content="">
<meta name="author" content="">
```

```
        <title>电子商务系统-注册</title>

        <!--Bootstrap core CSS -->
        <link href="{%static 'css/signin.css'%}" rel="stylesheet"-->
        <!--Custom styles for this template -->
        <link href="{%static 'css/bootstrap.min.css'%}" rel="stylesheet"-->
        <link href="{%static 'css/my.css'%}" rel="stylesheet">

    </head>

    <body>

        <div class="container">
            <form class="form-signin" method="post" enctype="multipart/form-data">
                <h2 class="form-signin-heading">电子商务系统-注册</h2>
                {{uf.as_p}}
                    <p style="color:red">{{error}}</p><br>
                <button class="btn btn-lg btn-primary btn-block" type="submit">注
册</button><br>
                <a href="/index/">登录</a>
            </form>

        </div><!--/container -->

    </body>
</html>
```

图 3-2　注册页面

其中，

（1）{{error}}：显示的是错误提示信息。

（2）{{uf.as_p}}：显示的是表单信息。

注册页面如图 3-2 所示。

4. 接口测试

这里在本书 2.11 节的基础上对测试方法进行了一些优化，主要通过利用 Python 对数据库的访问与接口测试相结合的方法进行相应的测试。

1）测试用例

表 3-1 为注册模块的测试用例，这里共设计了两个用例。

（1）注册一个数据库中已经存在的用户，系统应该提示“用户名已经存在!”。

（2）注册一个数据库中不存在的用户，注册成功，进入登录页面。

表 3-1　注册模块的测试用例

编号	描　　述	期 望 结 果
1	注册的用户名已经存在	有提示信息“用户名已经存在!”
2	注册的用户名不存在	注册成功,进入登录页面

2）XML 数据文件

loginRegConfig.xml：

```
<node>
    <case>
        <username>Johnson</username>
        <password>12345</password>
        <email>Johnson@126.com</eamil>
    </case>
    <case>
        <TestId>loginReg-testcase001</TestId>
        <Title>用户注册</Title>
        <Method>post</Method>
        <Desc>注册用户名已经存在</Desc>
        <Url>http://127.0.0.1:8000/register/</Url>
        <InptArg>{"username":"Johnson","password":"12345","email":"Johnson5@
126.com"}</InptArg>
        <Result>200</Result>
        <CheckWord>用户名已经存在!</CheckWord><!--通过“注册用户名不存在”测试完毕
删除数据库记录 -->
    </case>
    <case>
        <TestId>loginReg-testcase002</TestId>
        <Title>用户注册</Title>
        <Method>post</Method>
        <Desc>注册用户名不存在</Desc>
        <Url>http://127.0.0.1:8000/register/</Url>
        <InptArg>{"username":"smith","password":"12345","email":"smith@126.
com"}</InptArg>
        <Result>200</Result>
        <CheckWord>登录</CheckWord>
    </case>
</node>
```

第一个＜case＞…＜/case＞为测试代码用的初始化信息,将通过测试程序中的 setUp()中 Python 语言的基础类 sqlite3(注意,这里不是通过 Django 提供的数据库操作模块)向数据库中插入记录,然后运行程序进行测试,最后测试结束,需要在 tearDown()方法中将这些记录删除。在后面所有模块测试代码中都采用这种方法。

3）测试代码

为了方便，首先对一些方法进行封装。把本书2.11.5节中getXML.py中的类GetXML封装在一个名为util.py的文件中，并且把头部的两行建立在这个类的构造方法中。

```
def __init__(self,myXmlFile):
                dom =minidom.parse(myXmlFile)
                self.root =dom.documentElement
```

这样，原先的getxmldata()方法就改为：

```
def getxmldata(self):
                #从 XML 中读取数据
                TestIds =self.root.getElementsByTagName('TestId')
                Titles =self.root.getElementsByTagName('Title')
                Methods =self.root.getElementsByTagName('Method')
                Descs =self.root.getElementsByTagName('Desc')
                Urls =self.root.getElementsByTagName('Url')
                InptArgs =self.root.getElementsByTagName('InptArg')
                Results =self.root.getElementsByTagName('Result')
                CheckWords =self.root.getElementsByTagName('CheckWord')
                i =0
                mylists=[]
                for TestId in TestIds:
                        mydicts={}
                        #获取每个数据，形成字典
                        mydicts["TestId"] =(TestIds[i].firstChild.data).strip()
                       mydicts["Title"] =(Titles[i].firstChild.data).strip()
                       mydicts["Method"] =(Methods[i].firstChild.data).strip()
                        mydicts["Desc"] =(Descs[i].firstChild.data).strip()
                        mydicts["Url"] =(Urls[i].firstChild.data).strip()
                        mydicts["InptArg"] =(InptArgs[i].firstChild.data).strip()
                        mydicts["Result"] =(Results[i].firstChild.data).strip()
                        mydicts["CheckWord"]=(CheckWords[i].firstChild.data).strip()
                        mylists.append(mydicts)
                        i =i+1
                return mylists
```

然后在这个类中获得User测试初始化信息的方法getUserInitInfo()，具体实现如下。

```
def getUserInitInfo(self):
                #从 XML 中读取数据
                id =self.root.getElementsByTagName('id')
                id =(str(id[0].firstChild.data)).strip()
                username =self.root.getElementsByTagName('username')
                username ="\""+(str(username[0].firstChild.data)).strip()+"\""
                password =self.root.getElementsByTagName('password')
                password ="\""+(str(password[0].firstChild.data)).strip()+"\""
```

```
            email =self.root.getElementsByTagName('email')
            email ="\""+(str(email[0].firstChild.data)).strip()+"\""
            values =id +","+username+","+password+","+email
            return values   #返回的字符串 values 供插入数据库表 good_user 中使用
```

最后形成的字符串 values 为插入 User 表中 SQL 语句 values 后的内容。然后在这个文件中建立一个 DB 类，主要用于封装实现对数据库的操作，现在先建立以下几个方法。

```
class DB:
    #构造方法，获得 sqlite3 数据库文件的位置
    def __init__(self):
        self.url ="C:\\Python35\\Scripts\\ebusiness\\db.sqlite3"

    #连接数据库连接
    def connect(self):
        self.con =con =sqlite3.connect(self.url)
        self.cur =self.con.cursor()

    #关闭数据库连接
    def close(self):
        self.cur.close()
        self.con.close()

    #向 tablename 表中插入数据 values
    def insert(self,tablename,values):
        sql ="insert into "+tablename+" values ("+values+")"
        self.con.execute(sql)
        self.con.commit()

    #在 tablename 表中删除满足 condition 条件的记录
    def delete(self,tablename,condition):
        sql ="delete from "+tablename+" where ("+condition+")"
        self.con.execute(sql)
        self.con.commit()
```

(1) 方法 init()用于初始化数据库。

(2) 方法 connect()用于连接数据库。

(3) 方法 close()用于关闭数据库的连接。

(4) 方法 insert()用于向数据库表中插入数据。

(5) 方法 delete()用于从数据库表中删除满足条件的数据。

最后介绍用户注册模块的测试代码。

```
#!/usr/bin/env python
#coding:utf-8
import unittest,requests
```

```
from util import GetXML,DB

class myregister(unittest.TestCase):

        def setUp(self):
                print("--------测试开始--------")
                #定义数据库表名
                self.userTable ="goods_user"
                #建立 GetXML 对象变量
                xmlInfo =GetXML("registerConfig.xml")
                #获得初始化信息
                self.userValues =xmlInfo.getUserInitInfo()
                #建立 DB 对象变量
                self.dataBase=DB()
                #连接数据库
                self.dataBase.connect()
                #插入初始化数据库
                self.dataBase.insert(self.userTable,self.userValues)
                #获得所有测试数据
                self.mylists =xmlInfo.getxmldata()

        def test_register(self):
                for mylist in self.mylists:
                        #获取传输参数
                        payload =eval(mylist["InptArg"])
                        #获取测试 URL
                        url=mylist["Url"]
                        #发送请求
                        try:
                                if mylist["Method"] =="post":
                                        data =requests.post(url,data=payload)
                                elif mylist["Method"] =="get":
                                        data =requests.get(url,params=payload)
                                else:
                                        print ("Method 参数获取错误")
                        except Exception as e:
                                self.assertEqual(mylist["Result"],"404")
                        else:
                                #验证返回码
                                self.assertEqual(mylist["Result"],str(data.status_code))
                                #验证返回文本
                                self.assertIn(mylist["CheckWord"],str(data.text))

        def tearDown(self):
```

```
                #获取初始化数据库中的记录主码
                id =self.userValues.split(',')[0]
                #删除这条记录
                self.dataBase.delete(self.userTable,"id="+id)
                #关闭数据库连接
                self.dataBase.close()
                print("--------测试结束--------")


if __name__=='__main__':
        #构造测试集
        suite=unittest.TestSuite()
        suite.addTest(myregister ("test_register"))
        #运行测试集合
        runner=unittest.TextTestRunner()
        runner.run(suite)
```

(1) 在 setUp 方法中，

① 定义在这里用到的数据库表名为 goods_user。

② 通过语句 xmlInfo = GetXML("registerConfig.xml")从 XML 文件中读入测试初始化数据，并且通过 self.userValues = xmlInfo.getUserInitInfo()把它放到变量 self.userValues 中。

③ 建立数据库连接，通过语句 self.dataBase.insert(self.userTable,self.userValues)向数据库中插入设置测试需要的初始化信息。

④ 通过语句 self.mylists = xmlInfo.getxmldata()获得测试数据。

(2) 在测试程序中，

① 通过循环语句 for mylist in self.mylists 遍历所有的测试数据。

② 通过语句 payload = eval(mylist["InptArg"])获取测试参数，以及通过语句 url=mylist["Url"]获取测试执行路径。

③ 通过判断语句 mylist["Method"]是 post 还是 get，调用 data = requests.post(url,data=payload)或者 data=requests.get(url,params=payload)。

④ 通过断言语句 self.assertEqual(mylist["Result"],str(data.status_code))验证返回码是否正确。

⑤ 通过断言语句 self.assertIn(mylist["CheckWord"],str(data.text))判断验证的字符串是否在返回的文本中。

(3) 在 tearDown 方法中，

① 通过语句 id = self.userValues.split(',')[0]初始化数据库用户数据的主键。

② 通过语句 self.dataBase.delete(self.userTable,"id="+id)删除这条测试数据。

③ 断开数据库连接，结束测试。

3.3.2　用户登录

注册的用户可以通过登录页面登录系统。由于这个模块在前面讲得比较多，这里不做

过多的解释。

1. urls.py

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

2.6.3 节提到登录页面为系统首页，提供了 3 个 URL，分别对应：

(1) 127.0.0.1:8000/。

(2) 127.0.0.1:8000/index/。

(3) 127.0.0.1:8000/login_action/。

2. views.py

```
...
#首页(登录)
def index(request):
    uf =LoginForm()
    return render('index.html',{'uf':uf})
...
#用户登录
def login_action(request):
    if request.method =="POST":
        uf =LoginForm(request.POST)
        if uf.is_valid():
            #寻找名为 "username"和"password"的 POST 参数，而且如果参数没有提交，就返
            #回一个空的字符串
            username =(request.POST.get('username')).strip()
            password =(request.POST.get('password')).strip()
            #判断输入数据是否为空
            if username =='' or password =='':
                return render(request,"index.html",{'uf':uf,"error":"用户名和密
码不能为空"})
            else:
            #判断用户名和密码是否准确
                user =User.objects.filter(username =username,password =password)
                if user:
                    response =HttpResponseRedirect('/goods_view/')
                    #登录成功后跳转查看商品信息
                    request.session['username'] =username  #将 session 信息写到服务器
                    return response
                else:
                    return render(request,"index.html",{'uf':uf,"error":"用户名
或者密码错误"})
```

```
    else:
        uf =LoginForm()
    return render('index.html',{'uf':uf})
...
```

3. 模板

模板文件为 index.html，其内容如下。

```
{%load staticfiles%}
<!DOCTYPE html>
<html lang="zh-CN">
    <head>
        <meta charset="utf-8">
        <meta http-equiv="X-UA-Compatible" content="IE=edge">
        <meta name="viewport" content="width=device-width, initial-scale=1">
        <!--上述 3 个 meta 标签 * 必须 * 放在最前面，任何其他内容都 * 必须 * 跟随其后！-->
        <meta name="description" content="">
        <meta name="author" content="">
        <link rel="icon" href="../../favicon.ico">
        <title>电子商务系统-登录</title>

        <!--Bootstrap core CSS -->
        <link href="{%static 'css/signin.css'%}" rel="stylesheet"-->
        <!--Custom styles for this template -->
        <link href="{%static 'css/bootstrap.min.css'%}" rel="stylesheet"-->
        <link href="{%static 'css/my.css'%}" rel="stylesheet">

    </head>

    <body>

        <div class="container">
            <form class="form-signin" method="post" action="/login_action/"
enctype="multipart/form-data">
                <h2 class="form-signin-heading">电子商务系统-登录</h2>
                {{uf.as_p}}
                    <p style="color:red">{{error}}</p><br>
                <button class="btn btn-lg btn-primary btn-block" type="submit">
登录</button><br>
                <a href="\register\">注册</a>
            </form>

        </div><!--/container -->

    </body>
```

（1）{{error}}：显示错误提示信息。

（2）{{uf.as_p}}：显示表单信息。

用户登录界面如图 3-3 所示。

图 3-3　用户登录界面

4. 接口测试

重新构造初始化数据，代码如下。

loginConfig.xml：

```
<node>
    <case>
        <id>0</id>
        <username>Johnson</username>
        <password>000000</password>
        <email>Johnson@126.com</email>
    </case>
    <case>
        <TestId>loginReg-testcase001</TestId>
        <Title>用户登录</Title>
        <Method>post</Method>
        <Desc>正确用户名，错误密码</Desc>
        <Url>http://127.0.0.1:8000/login_action/</Url>
        <InptArg>{"username":"Johnson","password":"123456"}</InptArg>
        <Result>200</Result>
        <CheckWord>用户名或者密码错误</CheckWord>
    </case>
    <case>
        <TestId>loginReg-testcase002</TestId>
        <Title>用户登录</Title>
        <Method>post</Method>
        <Desc>错误用户名，正确密码</Desc>
        <Url>http://127.0.0.1:8000/login_action/</Url>
        <InptArg>{"username":"smith","password":"000000"}</InptArg>
        <Result>200</Result>
        <CheckWord>用户名或者密码错误</CheckWord>
```

```
    </case>
    <case>
        <TestId>loginReg-testcase003</TestId>
        <Title>用户登录</Title>
        <Method>post</Method>
        <Desc>错误用户名,错误密码</Desc>
        <Url>http://127.0.0.1:8000/login_action/</Url>
        <InptArg>{"username":"smith","password":"123456"}</InptArg>
        <Result>200</Result>
        <CheckWord>用户名和密码错误</CheckWord>
    </case>
    <case>
        <TestId>loginReg-testcase004</TestId>
        <Title>用户登录</Title>
        <Method>post</Method>
        <Desc>正确用户名,正确密码</Desc>
        <Url>http://127.0.0.1:8000/login_action/</Url>
        <InptArg>{"username":"Johnson","password":"000000"}</InptArg>
        <Result>200</Result>
        <CheckWord>查看购物车</CheckWord>
    </case>
</node>
```

测试代码如下。

loginTest.py：

```
#!/usr/bin/env python
#coding:utf-8
import unittest,requests
from util import GetXML,DB

class mylogin(unittest.TestCase):
        def setUp(self):
                print("--------测试开始--------")
                #定义数据库表名
                self.userTable ="goods_user"
                #建立 GetXML 对象变量
                xmlInfo =GetXML("loginConfig.xml")
                #获得初始化信息
                self.userValues =xmlInfo.getUserInitInfo()
                #建立 DB 对象变量
                self.dataBase=DB()
                #连接数据库
                self.dataBase.connect()
```

```
                #插入初始化数据库
                self.dataBase.insert(self.userTable,self.userValues)
                #获得所有测试数据
                self.mylists =xmlInfo.getxmldata()

        def test_login(self):
                for mylist in self.mylists:
                        #获取传输参数
                        payload =eval(mylist["InptArg"])
                        #获取测试 URL
                        url=mylist["Url"]
                        #发送请求
                        try:
                                if mylist["Method"] =="post":
                                        data =requests.post(url,data=payload)
                                elif mylist["Method"] =="get":
                                        data =requests.get(url,params=payload)
                                else:
                                        print ("Method 参数获取错误")
                        except Exception as e:
                                self.assertEqual(mylist["Result"],"404")
                        else:
                                #验证返回码
                                self.assertEqual(mylist["Result"],str(data.status_code))
                                #验证返回文本
                                self.assertIn(mylist["CheckWord"],str(data.text))

        def tearDown(self):
                #获取初始化数据库中的记录主码
                id =self.userValues.split(',')[0]
                #删除这条记录
                self.dataBase.delete(self.userTable,"id="+id)
                #关闭数据库连接
                self.dataBase.close()
                print("--------测试结束--------")


if __name__=='__main__':
        #构造测试集
        suite=unittest.TestSuite()
        suite.addTest(mylogin("test_login"))
        #运行测试集合
        runner=unittest.TextTestRunner()
        runner.run(suite)
```

3.3.3　用户信息显示

登录的用户可以单击自己的用户名，显示自己的注册信息以及自己的所有收货地址信息。

1. urls.py

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

2. views.py

```
...
#获取用户信息
def user_info(request):
    #检查用户是否登录
    util =Util()
    username =util.check_user(request)
    #如果没有登录，就跳转到首页
    if username=="":
        uf =LoginForm()
        return render(request,"index.html",{'uf':uf,"error":"请登录后再进入！"})
    else:
        #count 为当前购物车中商品的数量
        count =util.cookies_count(request)
        #获取登录用户信息
        user_list =get_object_or_404(User,username=username)
        #获取登录用户收货地址的所有信息
        address_list =Address.objects.filter(user_id=user_list.id)
        returnrender(request,"view_user.html",{"user": username,"user_info":
user_list,"address":address_list,"count":count})
...
```

关于检查用户是否通过合法途径登录，本书 2.6.2 节中已经作过详细介绍。

(1) 当检查当前用户为合法用户后，通过语句 count = util.cookies_count(request)调用 util 类中的 cookies_count()方法，显示当前用户的购物车内有多少商品。

(2) 通过语句 user_list = get_object_or_404(User,username=username)获取当前登录用户的信息。

(3) 然后通过语句 address_list = Address.objects.filter(user_id=user_list.id)获取当前用户的所有地址信息。

(4) 最后通过语句 return render(request,"view_user.html",{"user": username,"user_info":user_list,"address":address_list,"count":count})返回给模板文件 view_user.html，其中传过去的变量包括以下 4 个。

① user：当前登录用户名。

② user_info：当前登录用户名信息。

③ address：当前登录用户的所有地址信息。

④ count：当前购物车内商品的数量。

3. 模板

view_user.html：

```
{%extends "base.html" %}
{%block content %}
            <li class="active"><a href="/view_chart/">查看购物车<font color=
"#FF0000">{{ count }}</font></a></li>
            </ul>
            <ul class="nav navbar-nav navbar-right">
                <li><a href="/user_info/">{{user}}</a></li>
                <li><a href="/logout/">退出</a></li>
            </ul>
            </div><!--/.nav-collapse -->
        </div>
    </nav>

    <div class="container theme-showcase" role="main">

        <div class="page-header">
            <div id="navbar" class="navbar-collapse collapse">
            </div><!--/.navbar-collapse -->
        </div>

        <div class="row">
            <div class="col-md-6">
                用户名:{{user_info.username}}<br>
                 Email:{{user_info.email}}<br>
                <form action="/change_password/" method="get">
                    <input type="submit" value="修改密码">
                </form>
                <table class="table table-striped">
                    <thead>
                        <tr>
                            <th>编号</th>
                            <th>地址</th>
                            <th>电话</th>
                            <th>修改</th>
                            <th>删除</th>
                        </tr>
                    </thead>
                    <tbody>
```

```
                        {%for key in address %}
                        <tr>
                            <td>{{key.id}}</td>
                            <td>{{key.address}}</td>
                            <td>{{key.phone}}</td>
                            <td><a href="/update_address/{{key.id}}/1/">修改</a>
</td>
                            <td><a href="/delete_address/{{key.id}}/1/">删除</a>
</td>
                        </tr>
                    {%endfor %}
                </tbody>
            </table>
            </form><br>
            <form method="get" action="/add_address/1/">
            <input type="submit" value="添加地址">
            </form>
        </div>
        </div>

{%endblock %}
```

模板通过{% extends "base.html" %}和{% block content %}…{% endblock %}套用一个名为"base.html"的模板文件。base.html 文件如下。

```
{%load staticfiles%}
<!DOCTYPE html>
<html lang="zh-CN">
    <head>
        <meta charset="utf-8">
        <meta http-equiv="X-UA-Compatible" content="IE=edge">
        <meta name="viewport" content="width=device-width, initial-scale=1">
        <!--上述 3 个 meta 标签 * 必须 * 放在最前面,任何其他内容都 * 必须 * 跟随其后!-->
        <meta name="description" content="">
        <meta name="author" content="">

        <title>电子商务系统</title>

        <!--Bootstrap core CSS -->
        <link href="{%static 'css/signin.css'%}" rel="stylesheet">
        <!--Custom styles for this template -->
        <link href="{%static 'css/bootstrap.min.css'%}" rel="stylesheet">
    </head>

    <body role="document">
```