武汉大学二手书交易平台

设计模式课程报告

成员：孙汉铭，范云龙

# 1.采用mvc设计模式

## 1.1设计模式思路分析

要实现一个二手书交易平台，需要实现服务器与浏览器的交互，其中涉及到数据存储，数据在浏览器和服务器之间传送等过程，我们可以用到mvc设计模式

以注册流程来看：

第一步，浏览器将用户的注册信息提交给网站服务器，第二步控制器（C）告诉模型（M）将用户注册信息保存至数据库，第三步，模型（M）将用户的注册信息保存至数据库，第四步，数据库返回注册信息给模型（M）,模型（M）再把保存的结果发送给控制器（C），第五步控制器再告诉视图（V）产生一个html的结果页面，并将产生的页面返回给控制器(C)，控制器（C）再将内容返回给浏览器，浏览器展示生成页面。

## 1.2设计模式代码：

（1）模型

class User(models.Model):  
 user\_id = models.CharField(max\_length=200)  
 user\_password = models.CharField(max\_length=200)  
 user\_email=models.CharField(max\_length=200)  
 user\_phone\_number=models.CharField(max\_length=200)  
 user\_balance=models.IntegerField(default=100)

（2）视图

function register1\_Click() {  
 var user\_id = ***document***.getElementById("id\_textfield").value;  
 var user\_password = ***document***.getElementById("password\_textfield").value;  
 var user\_phone = ***document***.getElementById("phone\_textfield").value;  
 var user\_email = ***document***.getElementById("email\_textfield").value;  
 axios.post('/trading\_platform/register1/', {  
 'id': user\_id,  
 'password': user\_password,  
 'phone': user\_phone,  
 'email': user\_email  
 })  
 .then(function (response) {  
 let data=response.data;  
 if(!(data['data\_ok'])){  
 alert("用户名已经存在")  
 }  
 else {  
 ***window***.location.href= "http://127.0.0.1:8000/trading\_platform/login";  
 alert("注册成功");  
 }  
 })  
 .catch(function (error) {  
 ***console***.log(error);  
 });

}

（3）控制器

def register(request):  
 return render(request, "trading\_platform/register.html")  
  
  
def register\_01(request):  
 user\_information = json.loads(request.body)  
 user\_information = User(user\_id=user\_information['id'], user\_password=user\_information['password'],  
 user\_email=user\_information['email'], user\_phone\_number=user\_information['phone'])  
 if User.objects.filter(user\_id=user\_information.user\_id):  
 data\_ok = False  
 else:  
 data\_ok = True  
 user\_information.save()  
  
 return JsonResponse({  
  
 "data\_ok": data\_ok  
  
 })

## 1.3应用设计模式的优势

（1）耦合度低，视图层和业务层分离，这样就允许更改视图层代码而不用重新编译模型和控制器代码，同样，一个应用的业务流程或者业务规则的改变只需要改动MVC的模型层即可。因为模型与控制器和视图相分离，所以很容易改变应用程序的数据层和业务规则。

**（2）重用性高，**[MVC模式](http://baike.baidu.com/item/MVC%E6%A8%A1%E5%BC%8F)允许使用各种不同样式的视图来访问同一个服务器端的代码，因为多个视图能共享一个模型，它包括任何WEB（HTTP）浏览器或者无线浏览器（wap），比如，用户可以通过电脑也可通过手机来订购某样产品，虽然订购的方式不一样，但处理订购产品的方式是一样的。由于模型返回的数据没有进行格式化，所以同样的构件能被不同的界面使用。

**（3）可维护性高，**分离视图层和业务逻辑层也使得WEB应用更易于维护和修改。

# 2.采用简单工厂模式

## 2.1设计模式思路分析

在二手书交易平台代码的编写过程中发现要多次创建不同的User类，book类，order类，且这些都有一个公共的父类，创建过程中涉及了大量的修改，创建等操作，使得代码较为繁琐，为了解决代码重复，创造蔓延的问题，可以使用简单工厂模式。

## 2.2设计模式代码

User类：  
class User(models.Model):  
 user\_id = models.CharField(max\_length=200)  
 user\_password = models.CharField(max\_length=200)  
 user\_email=models.CharField(max\_length=200)  
 user\_phone\_number=models.CharField(max\_length=200)  
 user\_balance=models.IntegerField(default=100)

Book类：  
class Book(models.Model):  
 book\_id=models.IntegerField(default=0)  
 book\_name=models.CharField(max\_length=200)  
 book\_author=models.CharField(max\_length=200)  
 book\_original\_price=models.IntegerField(default=0)  
 book\_selling\_price=models.IntegerField(default=0)

Order类：  
class Order(models.Mode ):  
 user\_id=models.CharField(max\_length=200)  
 book\_name=models.CharField(max\_length=200)  
 book\_price=models.IntegerField(default=0)  
 order\_status=models.CharField(default="未完成",max\_length=200)

工厂类：

class Book\_trading\_platform\_factory:

def GetProduct(string arg):

if (arg.Equals("User")):

Product product= User();

else if (arg.Equals("Book")):

Product product= Book();

else if (arg.Equals("Order"):

Product product= Order();

return product;

}

## 2.3使用该模式的好处

（1）实现了User，Book，Order对象创建和使用的分离，

（2）客户端无须知道所创建的User，Book，Order的类名，只需要知道具体产品类所对应的参数即可

（3）通过引入配置文件，可以在不修改任何客户端代码的情况下更换和增加新的具体产品类，在一定程度上提高了系统的灵活性，解决了为了解决代码重复，创造蔓延的问题。