高级程序设计

纪念品交易平台

罗昊

2022 年 4 月 1 日

计算机科学与技术系 191220073

目录

概述

总体设计思路

此次设计的纪念品交易平台项目设计要求中已经明言需要使用多模块编程以及面向对象编程，是以在设计交易平台时，需要剖分交易平台而成各个相对独立的模块。
现划分模块如下：

- 用户交互模块
- 用户操作模块
- 数据管理模块
- 简易计算器模块

类图展示

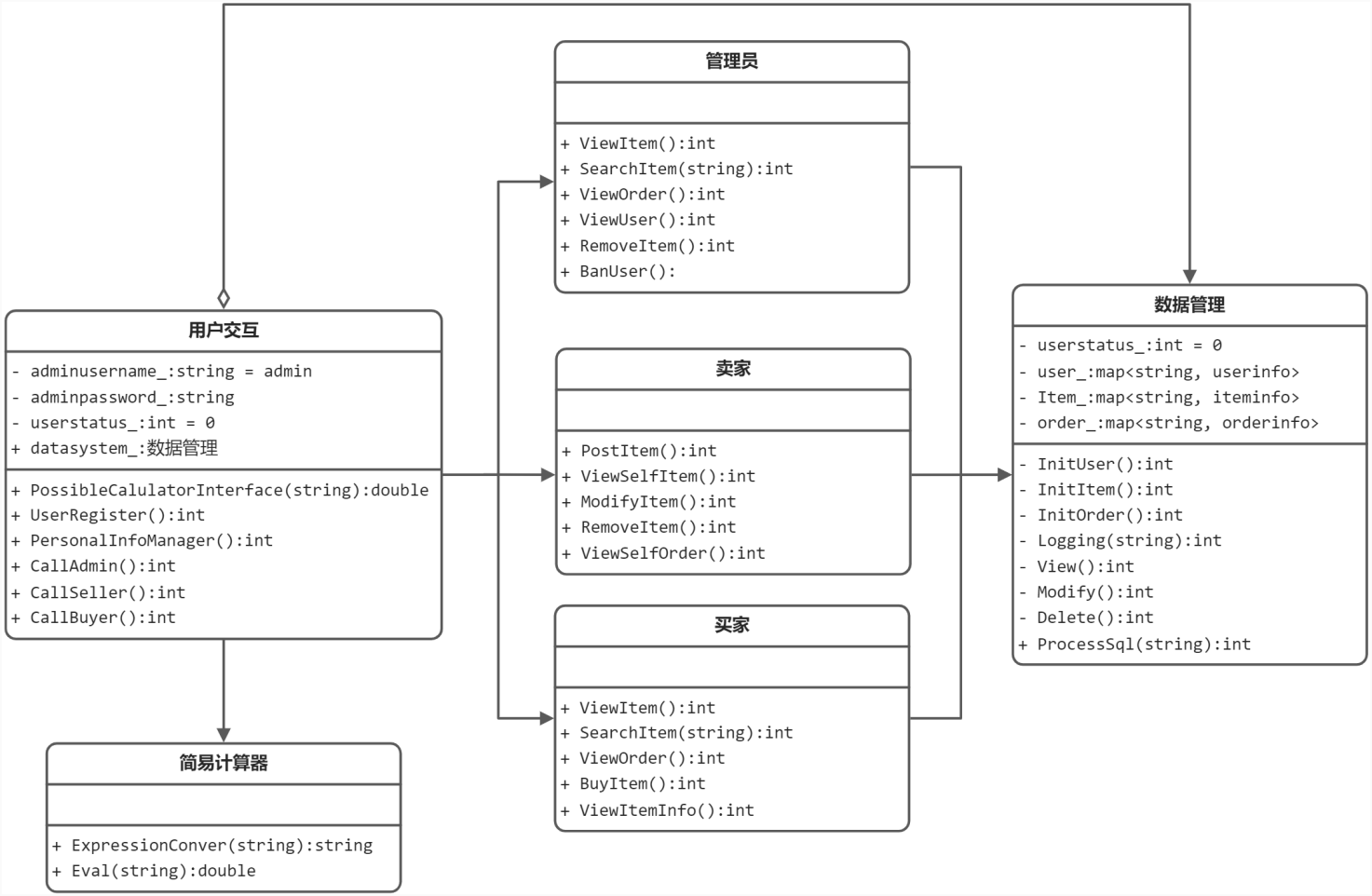
管理员
+ ViewItem():int
+ SearchItem(string):int
+ ViewOrder():int
+ ViewUser():int
+ RemoveItem():int
+ BanUser():
用户交互
- adminusername_:string = admin
- adminpassword_:string
- userstatus_:int = 0
+ datasystem_:数据管理
+ PossibleCalulatorInterface(string):double
+ UserRegister():int
+ PersonalInfoManager():int
+ CallAdmin():int
+ CallSeller():int
+ CallBuyer():int
卖家
+ PostItem():int
+ ViewSelfItem():int
+ ModifyItem():int
+ RemoveItem():int
+ ViewSelfOrder():int
买家
+ ViewItem():int
+ SearchItem(string):int
+ ViewOrder():int
+ BuyItem():int
+ ViewItemInfo():int
数据管理
- userstatus_:int = 0
- user_:map<string, userinfo>
- Item_:map<string, iteminfo>
- order_:map<string, orderinfo>
- InitUser():int
- InitItem():int
- InitOrder():int
- Logging(string):int
- View():int
- Modify():int
- Delete():int
+ ProcessSql(string):int
简易计算器
+ ExpressionConver(string):string
+ Eval(string):double

图 1: 整体类图

模块设计思考

如前文图 1 所示，该交易平台实现总共要实现六个模块。前文提及的用户操作模块包含有管理员、卖家、买家三个角色。以下将具体地讲述如何实现这些模块。

但就目前而言，这些模块的设计还不够完善，还需要进一步的设计。譬如，管理员、卖家、买家三个角色可以继续抽象为一用户以提高代码复用程度。但是否如此，还需在实践中进一步检测。

模块设计补充 3.18

在实践中，发现有如下问题。

- 管理员、卖家、买家三个角色模块可以考虑抽象自用户这一抽象基类。因为该三者有较多相似功能。
- 如上，若考虑使三者继承同一抽象基类，则需要使数据管理模块有较好地多状态处理机制。否则以上三者仍需调用不同数据模块方法，如此则使得抽象几无意义，接口复杂程度仍未减少。
- 假使不将该三者视作用户基类，可以调用数据管理模块的基础方法，但无疑会使得数据管理模块的抽象程度降低。
- 后续可能还需要慎重考虑修改。

类图设计更新

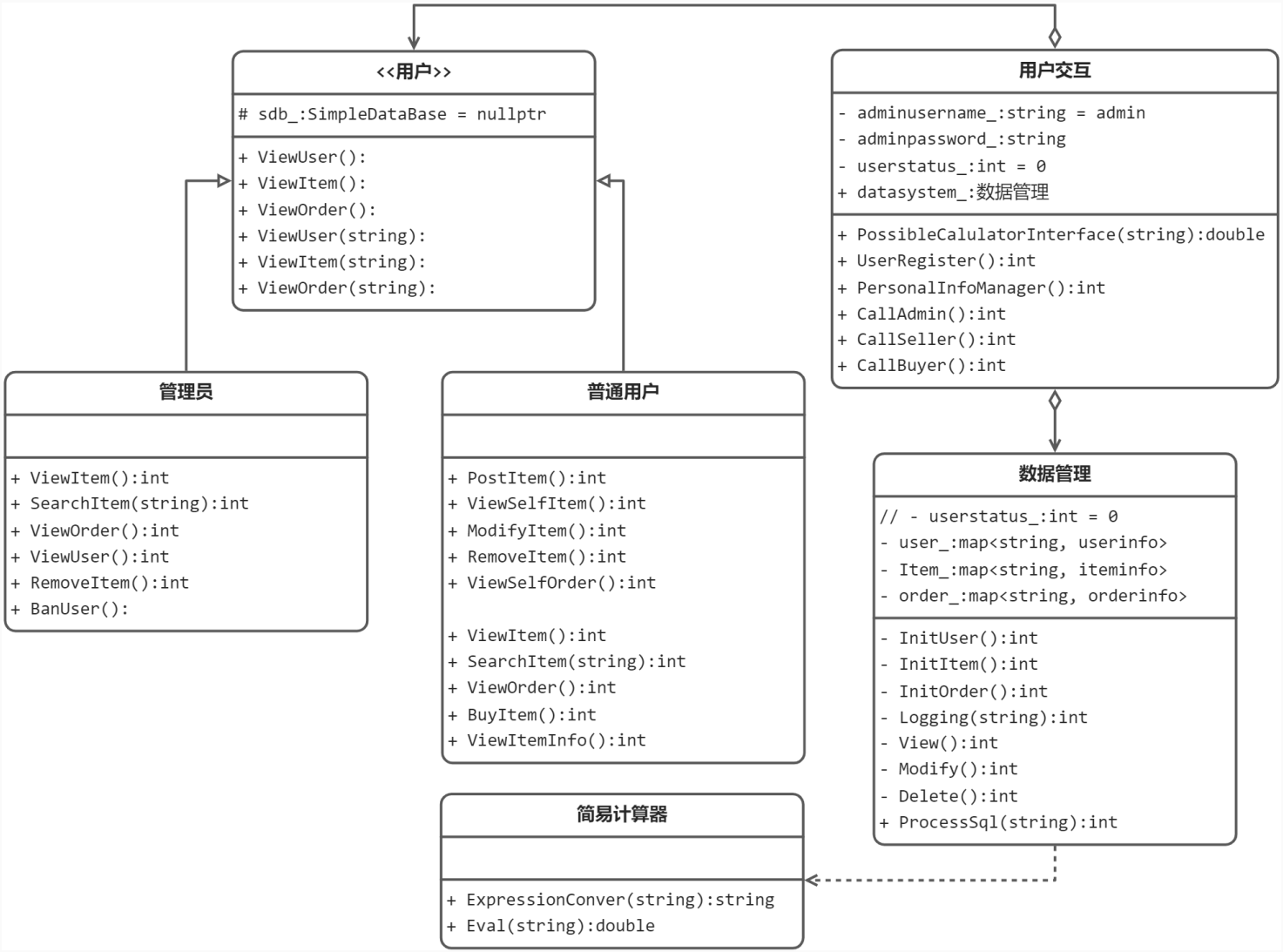

图 2: 整体类图更新

类图设计更新说明

如上页图 2 所示，本次更新最大的变化是为用户抽象基类的改变。
之所以进行如此改变缘故在于之前的设计中，想当然地将卖家、买家视作不同角色的用户。但在实际实现中，发现卖家、买家角色可以访问的数据都是本用户数据，而管理员却能够访问全用户组的数据。
实质上，买家、卖家之间本就可以互相转化。毕竟这是个人交易平台，该而角色本非泾渭分明。将买家、卖家角色进行划分，实质上需要外界来处理卖家、买家的转化。是以初步改进版本中，我将卖家角色模块以及买家角色模块分离，并为其设计 CommonUser 抽象基类，以买家卖家为具体实现。但很快发现如此为画蛇添足，二者统合作为 User 抽象基类的实现似乎更好。

用户交互模块

交互模块展示

如图 3 所示，该模块内有管理员用户名、密码数据，并设置为不可访问。此外，该模块中有用户状态字，用于区别目前用户登陆状态。初始化为 0，表示未登陆。1 为管理员，2 为卖家，3 为买家。其下为五个方法(不计首个预留接口)，将在下面展示。

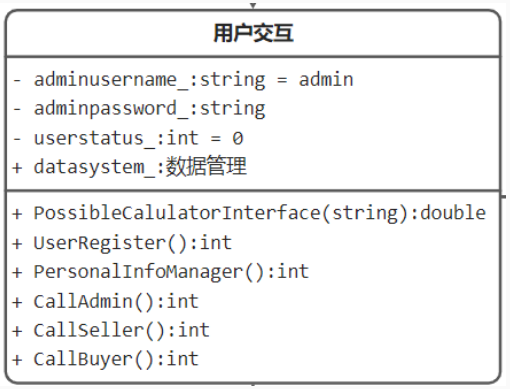

图 3: 用户交互模块

交互模块方法

UserRegister() 方法首先校验用户状态字，正确则向模块内 datasystem_ 对象调用 ProcessSql() 方法，传递特殊修改指令。
PersonalInfoManager() 方法实际包括查看、修改个人信息与充值功能，具体实现仍是调用 ProcessSql() 方法，传递查看、修改指令即可。计算金额，则需要生成计算器对象并提供基础计算表达式，调用计算器模块两个处理方法。
剩余三个 CALL 方法，则是在调用 ProcessSql() 方法传递查询指令后，若成功即调用相应的 CALL 指令。如此则可以进入对应的角色模块。

交互模块方法更新

在后续实践中，交互模块方法的职责改易。原定所有字符显示都需要由交互模块负责，现在交互模块的职责为管理非用户状态下的字符显示。即，用户状态下，包括管理员用户以及普通用户 (卖家、买家)，其字符显示现在都需要自理。
更新原因在于，如果需要交易模块全权处理输出，则模块内需要更多的信息交流。规划一个通用度高的信息交流通道较为复杂，而规划一个专用的信息交流通道则代码未免繁杂。是以实践中，直接删除了这种信息交流通道，转而实现更简单的信息暴露接口。
最终，将交互模块的职责改易，仅作为非用户状态下的字符显示与信息读入。

用户操作模块

各用户类型模块

实质工作是根据需求生成相对应的 SQL 语句，并调用 ProcessSql() 方法，传递给 datasystem_ 对象。
因为这些用户实例一般运行在交互界面实例方法之中，可以访问交互界面实例内公开的数据管理模块实例。
然而需要思考的是用户状态字信息传递。目前设计中，用户状态字存在于交互实例与数据管理实例之间，需要考虑同步问题。需要提供一个方法，将用户状态字传递给数据管理模块。但未免显得数据冗余，后续可能会对此进行修改。

用户操作备忘

各成员如何获取自身信息？需要组合成 SQL 传递给 ProcessSql() 方法
若从交互界面传入，如何保证数据独立并良好同步？
操作错误码如何分配并传递？

用户操作模块更新

如上，因为交互模块功能的减少，用户操作模块需要担负起不同用户层次下的操作输入与信息输出。
是以实际上，用户实例也需要与数据管理模块实例进行交互，通过一些信息暴露接口。
这自然具有弊端，原设计为用户-交互-信息管理，条例简单。现设计三者为三角关系，实际上更为混乱。

数据管理模块

数据管理模块核心数据成员

核心数据成员包括如下:

- userstatus_：用户状态字，用于区别目前用户登陆状态
- user_：用户信息读取
- item_：商品信息读取
- order_：订单信息读取

其中后三者全部采用 STL 库中 map 类型，其中 key 键为各自信息编号，其余信息统一组织成相应结构体进行储存。
猜想编号很可能具有一定顺序性质，是以不使用 unordered_map 的方式进行储存。但具体实现如何还需实践。

数据管理模块初步实现

```cpp
class SimpleDataBase
{
private:
    map<string, UserInfo> user_;
    map<string, ItemInfo> item_;
    map<string, OrderInfo> order_;

public:
    SimpleDataBase();
    // ~SimpleDataBase();

    bool LoadUser(const string& filename);
    bool LoadItem(const string& filename);
    bool LoadOrder(const string& filename);

    bool SaveUser(const string& filename);
    bool SaveItem(const string& filename);
    bool SaveOrder(const string& filename);

    // print is debug tool, delete it later
    bool print(string tablename);
    bool LogFile(string sql);
```

图 4: 数据管理模块初步实现

数据管理模块方法

首先为三个 Init 类型函数，其意义为读取具体文件，并将其内容储存在相应的数据成员中。
所以该三种方法会在数据管理模块初始化为实例时调用。同样地，在文件被改变时也需要调用该三种方法以更新数据成员。
Logging() 方法用于记录操作日志并记录至文件之中。目前没有读取记录文件的需求。
剩余三个不可见方法则是具体的对文件进行查询、修改、删除操作指令。目前接收参数我留空，是为了方便匹配最重要的 ProcessSql() 方法。该方法将在下节中详细介绍。

数据管理模块 SQL 处理方法

这可能是本项目中最为繁杂的一部分。该方法接收简易的 SQL 语句字符串，并将其转换为相应的操作指令，即调用上文提及的 View()、Modify()、Delete() 方法。
显然，需要解析 SQL 字符串并提供目标、表名、操作类型等信息。
目前对于这些这一模块的实现细节仍在思索。但可以肯定的是，项目实现中只允许数据管理模块访问文件，不允许其他模块访问文件。
实质上将该模块视作简易数据库即可，可以参考其余数据库实现?

数据管理模块更新

原设计中，数据管理模块对标的是一个简单的数据库——包括建表、插入、查询、修改、删除等操作。这些操作通过同一的字符串接口进行，独立程度甚至与四则运算计算器相当。
但实现中发觉实现一个通用的数据管理模块——即是是简单的数据管理，一旦对标数据库，那么总体复杂程度会提升很多。
不得已，实际实现的数据管理模块更加专用，也添加了更多的数据暴露接口，耦合度也更高。也算是妥协之处。

简易计算器模块

四则运算计算器实现

该模块是本次项目中独立性最高之部分，只需要接收运算表达式字符串，返回答案或错误码即可。
可以维护两个栈，一个用于存储数字，一个用于存储运算符。转为逆波兰表达式，并计算结果。
这部分内容在演示文稿中已有展现。
或是可以选择使用递归下降方法进行表达式求值，以下为相应文法:

- exp -> exp op1 exp | term
- op1 -> + | -
- term -> term op2 factor | factor
- op2 -> * | /
- factor -> (exp) | number

后续补充小数与负号即可，当然，这种方法可能在表达式过长时使用过多栈空间产生问题。

实践中梯度下降调用依据

expression:

- expression + term
- expression – term
- term

term:

- term / primary
- term ? primary
- primary

primary:

- number
- – primar
- (expression)

实践具体函数

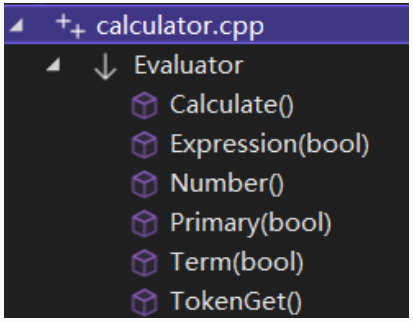

图 5: cal

其中 Calculate 函数作总体调用，TokenGet 函数遍历字符串而后返回类型。其余依照上文所示层次依次下降调用，没有循环调用。
似乎性能不是很良好，但足够应用于该项目。后续会进行优化。