Development File: Bookstore

李磊智 to 黄健浩

overview

本软件旨在为用户提供一个便于使用的书店管理系统。本系统的管理基于权限等级，主要需要实现用户端和管理端两方面的使用需求。用户端方面需要能够创建账户，查询书籍，购买书籍等；管理端需要能够添加书籍，管理库存，生成日志等功能。

structure

- **program：**

  main： 初始重置启动系统，并逐行读取，结束时关闭系统

  parser : 实现对指令的读取，将指令转化为操作类的一个成员进行储存，并执行

  account： 实现一个账户的类，成员变量为账户各种信息，成员函数为账户所能执行的操作，使用派生的方式来完成权限的管理。

  // [修改] 不使用派生

  book : 实现一个图书的类，成员变量为图书的信息

  operation : 实现一个操作的类，成员变量为操作的信息和执行者，成员函数为操作记录的输出

  transaction：实现一个交易的类，成员变量为交易的信息，成员函数为交易记录的输出

  // [修改] 实现较为困难，弃用 operation 以及 transaction 两个类

  block_chain : 实现一个块链的类，并进行包装

  system: 实现系统的类用于管理程序中被选中的书籍和最后登录的账号

  // [修改] 弃用，使用登录栈

- **file：**

  book_info： 储存仓库内书的信息。

  log_in_account： 为了防止系统中途down掉无法恢复原本系统内的帐号，用一个类似栈的结构来储存目前登陆在系统内的账号名单。

  // [修改] 弃用

  transaction_document : 储存所有的流水记录,并在开头储存总流水数，和总盈亏情况

  // [修改] 弃用

  operation_info : 储存所有的操作记录，并在开头储存总操作数

  account_info：储存所有的账号信息

logic

- **main**

  1. 启动系统，包括登录游客账户等
  2. 读取指令并传入到parser中处理，使用try...catch语句进行错误处理
  3. 关闭系统，包括退出所有账号等

- **parser**

  // [修改] 仅用来解析单词，无其他功能

  1. 判断语句类型，调用不同类型语句对应的parser函数，（在parser类中进行实现）
  2. 将parser处理的参数调入到account的执行函数当中
  3. 生成对应的operation对象储存到operation_document
  4. 如果有资金的流动，储存一个transaction对象储存到transaction_document

class

- **account：　　//函数描述均在README中给出**

  // [修改] 不用多态，将priority用一个int记录

  ```
  class level_0_account{
  protected:
  public:
   void register(char *User_ID , char *Password, char *User_Name){}
   void login(char* User_ID, char *Password){}
  }

  class level_1_account : level_0_account{
  protected:
   char *User_ID[30]
   char *Password[30]
  public:
   void logout(){}
   void passwd(char *User_ID, char *Old_Password, char *New_Password){}
   void show(char *identity, char *Identity){}
   //identity为show依据的标准即book类的成员变量名，Identity为具体内容，即用户定义的参量
  void buy(char* ISBN, int Quantity){}
}

level_3_account : level_1_account{
```

```
protected:
public:
  void useradd(char *User_ID, char *Password, int Priority, char *User_Name ){}
  void modify(){}
  book select(,system &System, char *ISBN){}
  import(system &System, int quantity, double total_Cost){}
  report(system &System){}     //返回的是最后登录的账号的记录
}
level_7_account : level_3_account{
protected :
public :
  show_finance(int time){}
  report_finance(...){}
  report_employee(...){}
  log(...){}
  //关于日志的接口我目前还不了解块链具体的实现，因此无法给出
  delete(char *user_ID){}
}
```

- **book**

```
struct book{
  char *ISBN[20]
  char *Book_Name[60]=""
  char *Author[60]=""
  char*[60]  Keyword=""
  int Quantity=0     //库存数量
  double price=0    //单价
}
```

- **operation**

```
// [修改] 已删
class operation{
private:
string executor=""
string Operation =""
public:
```

```
 operation(string executor, string Operation){}
 ~operation(){}
 void print_operation(){}
}
```

- **transaction**

```
// [修改] 已删
class transaction{
private:
 bool quality            //收入为true 支出为false
 double  amount          //金额数
 book item               //交易对应的书籍
public:
 transaction(bool quality, double amount){}
 ~transaction(){}
 void print_transaction(){}
}
```

[修改] 不使用 system，而用 pair<Account,string> 实例化的 vector 存储即可

- **system**

```
// [修改] 已删
class system {
private:
account last_Account;     //最后登录的账户
char* book_Selected;      //被选中的图书对应ISBN
public:
void change_Last_Account (account Account){}
void change_Book_Selected (char *Book){}
account get_Last_Account(){}
char* get_Book_Selected(){}
}
```

- **block_chain　//我也不会，自由发挥吧**

details

1. 无效操作和无效交易不计入document文档当中
2. 日志系统的实现方式（供参考）
    - 操作记录、员工工作情况报告指令

      [执行人] | [操作内容]
    - 财务记录

      ”总收入“ +[营业额] ”总支出“ -[消费额] 总利润 [利润值]

      ”单笔交易记录：“

      [买入/卖出] [书籍名称] [书籍ISBN] [单价] [数量] [总金额]（买入为负卖出为正）
    - 日志

      [执行人] | [操作内容]

      [买入/卖出] [书籍名称] [书籍ISBN] [单价] [数量] [总金额]（买入为负卖出为正）

      //操作的输出为主题，当涉及到交易时下对应操作下方输出相应交易记录


，成员变量为操作的信息和执行
，成员变量为交易的信息，成
这两个就直接用if分支了
然后parser的话
行
我想让它只有解析单词的功能
确实没什么必要封装
嗯
然后就和别的不用关联起来了
我记得那个上次给的parser就是一个单独的东西
好