

**面向对象程序设计与实践(C++)2**

ForumSystem

程序说明书

班级： 2014211304

姓名： 史文翰

学号： 2014211218

班内序号： 04

日期： 2016.9.16

# 题目一：论坛系统初级版

## 一题题目以及需求分析

题目一要求实现一个具有基本功能的论坛系统，要求使用OOP的相关技术和C++语言实现，从整体上讲，讲论坛分为版块、系统、帖子等类，用户分为普通用户、版主、管理员等类，采用多态、封装、继承、重载等技术来实现数据的交互和控制。

## 各功能实现模块

题目一要求的功能有：

论坛方面：

1. 显示帖子列表，并提供交互式UI。
2. 实现基本的登录器和注册机。

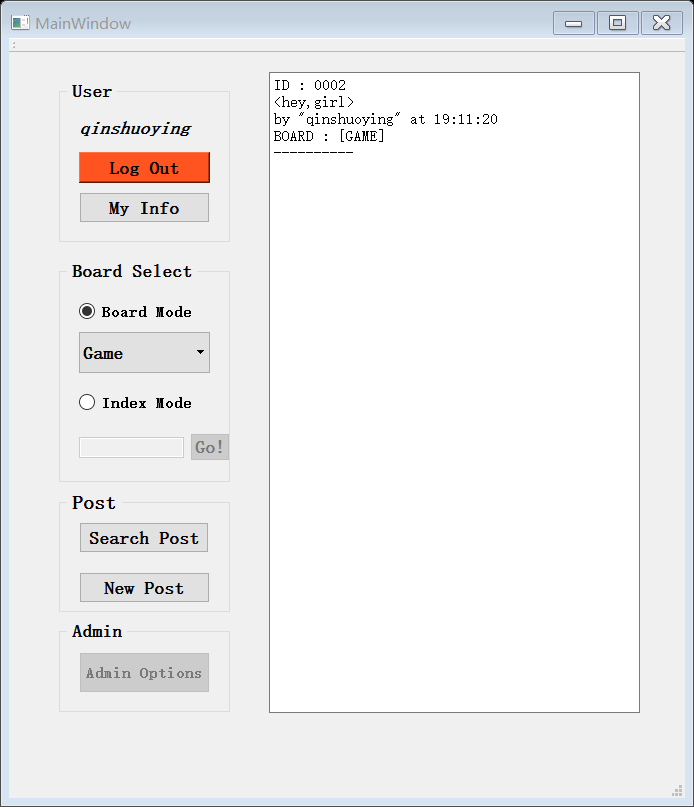
用户方面：

1. 任何用户均可以看帖，以ID查找形式，或板块形式浏览。
2. 任何用户均可以登入和登出，查看自己的相关信息。
3. 除管理员外，用户可以发帖。
4. 除管理员外，用户可以发表评论。
5. 仅管理员可以进入用户管理页面，提升版主和撤销版主。
6. 仅版主可以在自己的管理板块下删帖。

### UI界面概述

#### 实现截图

***请注意：由于第二版本的UI更为完善具体，并对第一版本完全兼容，这里和下面采用的UI截图均为第二版本的而不再赘述第一版本的UI，而代码仍是原汁原味的第一版本代码。涉及到的第二版本的功能将在《题目二》中具体分析。***

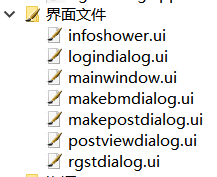


#### 实现思路和代码

主要涉及到窗体的组织和QT各类部件的运用，在此不再赘述。

这里显示了一个普通用户“qinshuoying”是他的用户名，从上往下的按钮作用依次为：登出操作，个人信息查看，查看帖子（板块模式），查看帖子（索引模式），查找帖子（并查看详细信息、可以发表评论），发帖，管理员选项（只对管理员开放）。

对话框的组织如下：



从上至下依次为：个人信息浏览框、登录器、主视窗（当前视图）、管路员窗口、发帖窗口、看帖窗口（同时也是评论窗口）、注册机。每一个窗口都具体对应了一到多个功能，将在下面一一陈述。

#### 用到的重点知识简介

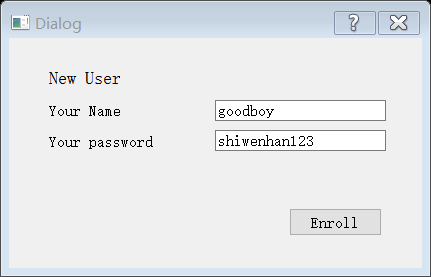
主要涉及到QT内部UI部件的运用，一些设计模式基本的元件组织常识，以及一切涉及到部件数据传输，信号&槽机制等相关知识。

#### 心得

虽然图形化是加分项，不过可以见得，一个良好的图形化界面可以极大地提高容错性和程序的健壮性，将报错信息，输入输出指示信息等以可见的形式交互显示，可以更加清晰、明确地测试程序功能，将图形化框架搭好之后，内部的数据传输、具体功能的实现会变得得心应手。

### 登录与注册

#### 实现截图



如图，图一显示了一个新用户注册的过程，输入合法的用户名（不可为空）和密码后，返回登录框，此时注册机反馈一个新的ID给新的用户，由于程序默认有三个测试用户，因此这个新用户的ID被分配了000004。用户登录需要的是ID和密码，这是因为ID唯一标识了一个用户，而非用户昵称（用户名）。

#### 实现思路和代码

User\* Forum::login(QString &id , QString &pass)

{

User\* puser = NULL;

for(auto &r : \_users)

{

if(r->getID() == id && r->getPassword() == pass)

{

puser = r;

qDebug()<<"find match!!"<<endl;

return puser;

}

}

qDebug()<<"login :no match"<<endl;

return NULL;

}

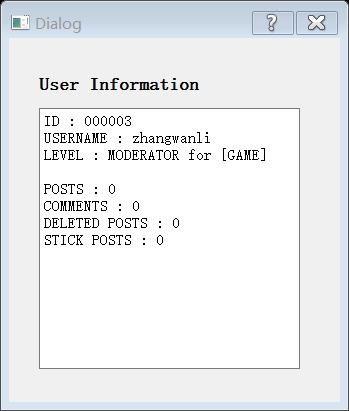
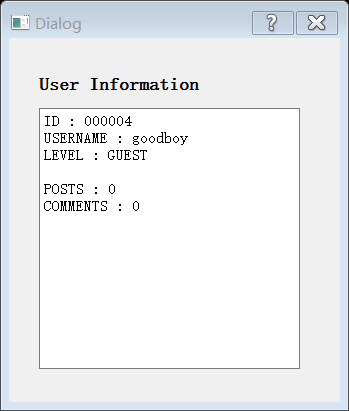
如上，可以用一个很精巧的行为来实现。本论坛在系统初始化的时候已经完成了new行为，因此login仅仅需要更改程序的单例模式指针，否则返回空指针报错即可。当然，这些都是以登录成功（即ID和密码完全匹配）为前提的。

#### 2.1.3心得

注册是额外增加的功能，本程序最大的特点是简化各个功能的代码段，使类的行为尽可能地简单而不是正确性。这样的思想在后面的功能介绍中很多见，具体表现为：多函数，多利用指针而非对象本身，函数间耦合性很小。

### 查看个人信息

#### 2.3.1 实现截图

如上两图，本功能使用多态实现：不同的用户类型接口相同，但表现出的行为（显示各自的功能性数据）不同，例如图一展示了一个普通用户，他有发帖数和看帖数；而图二展示了一个GAME版块的版主，他除了发帖数和评论数之外，还有额外的删帖数和顶帖数。

#### 实现思路和代码

利用多态实现，首先在User中把接口虚化，有如下代码：

void User::*showInfo*(QTextBrowser \*qtb)

{

qDebug()<<"ERROR : NO AUTHORITY !"<<endl;

exit(1);

}

**请注意：这里用到了这个程序最根本的思想——多态**。将不能被实体化的User类所有需要被重写的方法都预先定义了虚拟接口，这些接口本质上来说什么也不做，因为它只是在这里并等待着被重写。在这里，我们用抛错并退出的思路来处理不正确的调用User接口的后果。请注意，这样的思想贯穿本程序始终，是本程序设计的最核心元素，也是全面利用多态的重要体现，此类的代码在下面的介绍中不再赘述。**简明言之：本程序找不到任何一个子类指针，所有操作均通过继承树的根指针完成。**

在Guest和Moderator类中均有不同的重写，分别有如下代码：

void Guest::*showInfo*(QTextBrowser \*qtb)

{

QString le = levelToQstr(\_level);

qtb->setPlainText("");

QString base\_info = "ID : " + \_id + "\nUSERNAME : " + \_userName + "\nLEVEL : " + le + "\n";

QString opt\_info = "POSTS : " + QString::number ( \_numPost,10 ) + "\nCOMMENTS : " + QString::number ( \_numComment,10 ) + "\n";

qtb->append(base\_info);

qtb->append(opt\_info);

}

void Moderator::*showInfo*(QTextBrowser \*qtb)

{

QString be = boardToStr(\_board);

QString le = levelToQstr(\_level);

qtb->setPlainText("");

QString base\_info = "ID : " + \_id + "\nUSERNAME : " + \_userName + "\nLEVEL : " + le + " for [" + be + "]\n";

QString opt\_info = "POSTS : " + QString::number ( \_numPost,10 ) + "\nCOMMENTS : " + QString::number ( \_numComment,10 ) + "\n" +

"DELETED POSTS : " +QString::number( \_numDelPost , 10) + "\n" +

"STICK POSTS : " + QString::number(\_numStickPost , 10) + "\n" ;

qtb->append(base\_info);

qtb->append(opt\_info);

}

可以看出对多态思想的利用。当然，Admin类也用不同的重写，这里不再赘述。

#### 用到的重点知识简介

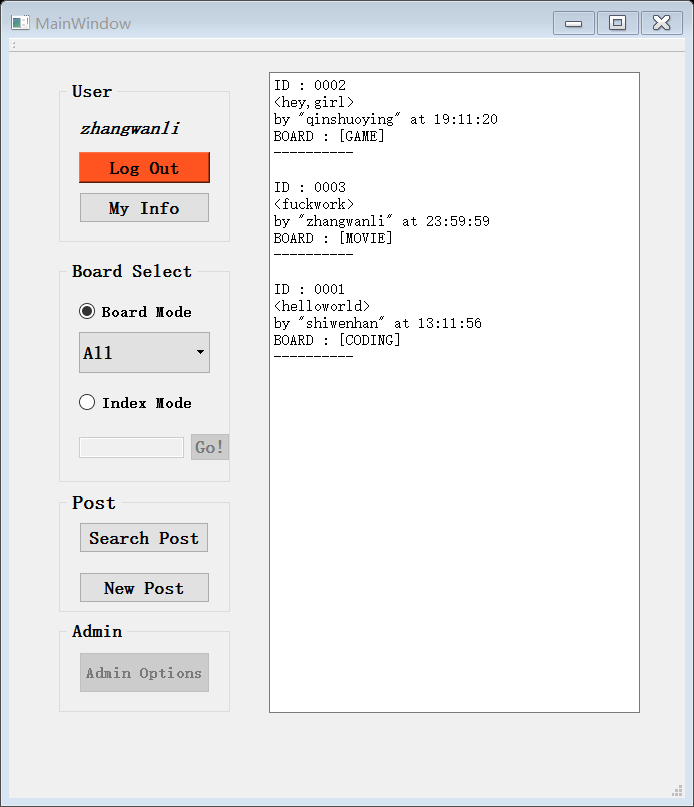
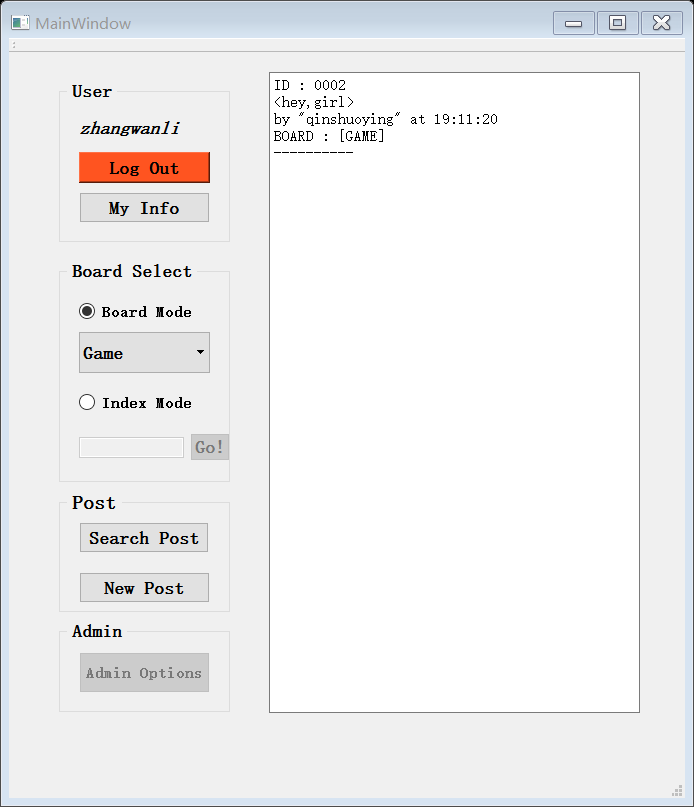
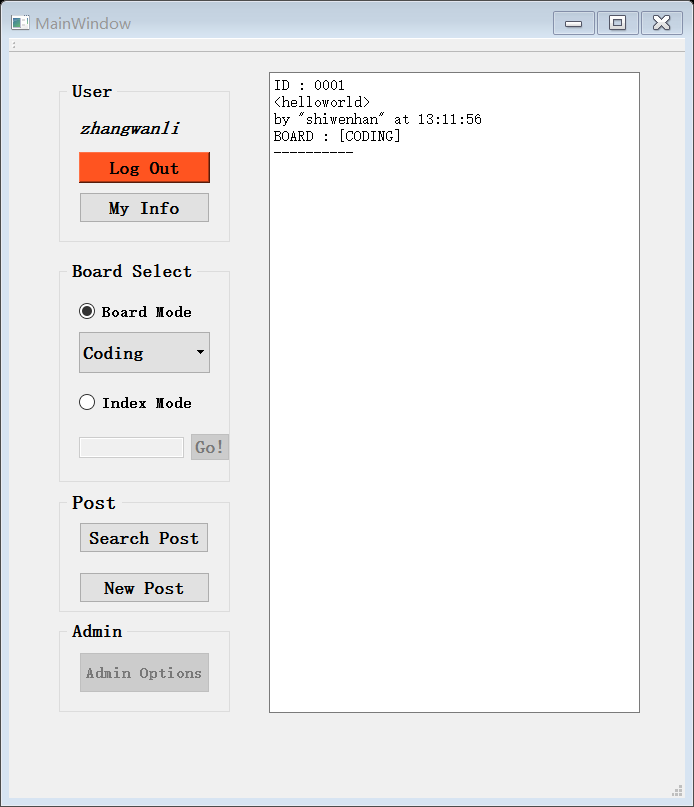
这里运用了多态，具体的阐述请见上面的代码分析。请注意，多态思想贯穿本实验始终，接下来的功能介绍中不再详述对多态的利用上。

#### 2.3.5 心得

多态的使用最大化地简化了接口的设计：一切操作均通过父接口完成，不必在乎继承树枝干部分的实现细节。

### 以版块的形式看帖

#### 2.4.1 实现截图

从左至右分别为：查看所有帖子（这里是查看帖子的缩略形式，没有内容），查看游戏版块的帖子，查看编程版块的帖子。可以明显地看出，这个功能通过一个极其简单的combobox控件实现。

#### 2.4.2 实现思路和代码

通过QT中的组合框控件完成，简单代码如下：

void MainWindow::on\_comboBox\_currentTextChanged(const QString &arg1)

{

ui->postView\_textBrowser->setPlainText("");

if(arg1 == "All") // plus : show sticky posts in dialog

{

showPostsInDialog(ui->postView\_textBrowser);

}

else

{

QString qs\_eb = arg1;

ENUM\_BOARD eb = matchBoard(qs\_eb);

qDebug()<<"QS\_EB : "<<qs\_eb<<endl;

QVector <Post\* > posts = nowUser->getPostList(eb);

for(Post\* &r : posts)

r->showPartPost(ui->postView\_textBrowser);

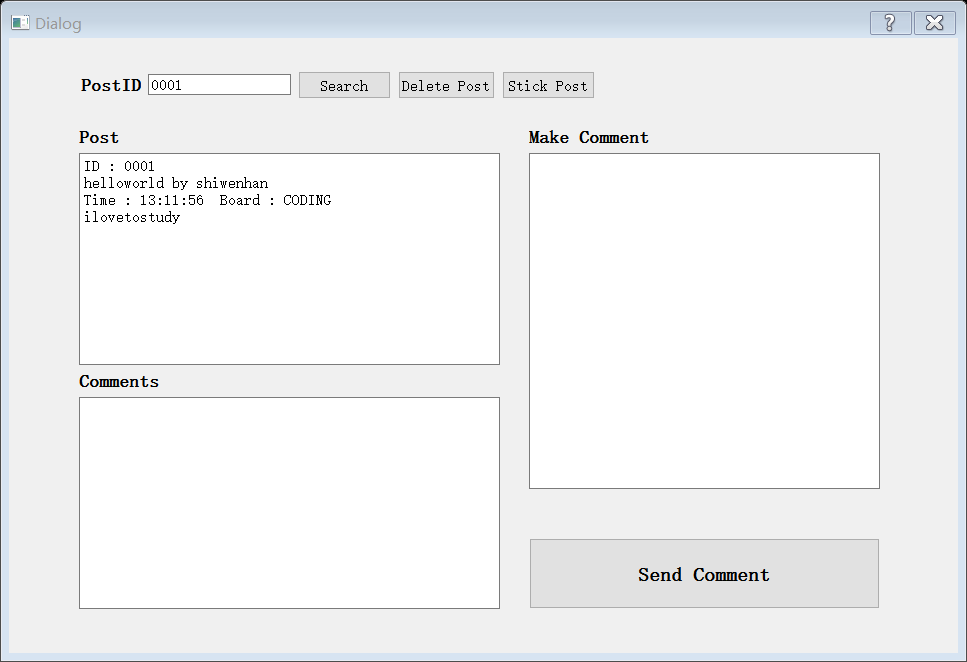
}

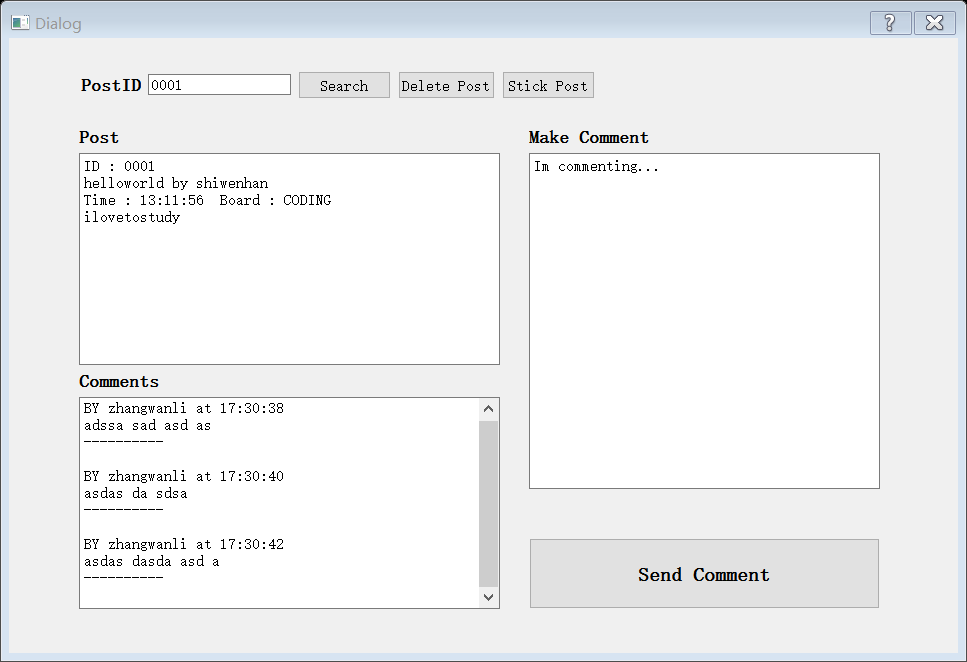
}

可以看出，这个功能相当简单，每当COMBOBOX的内容变化时，发送一个信号给显示器，它就会根据COMBOBOX当前的内容来显示帖子的缩略列表。

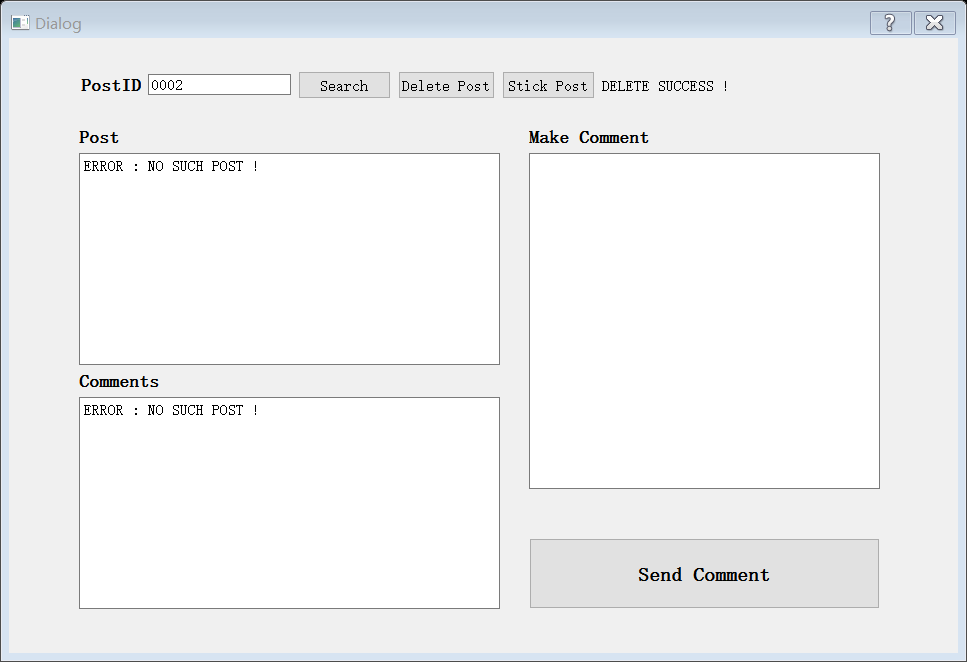
### 帖子操作：看帖、发帖、评论、删帖

#### 2.5.1 实现截图

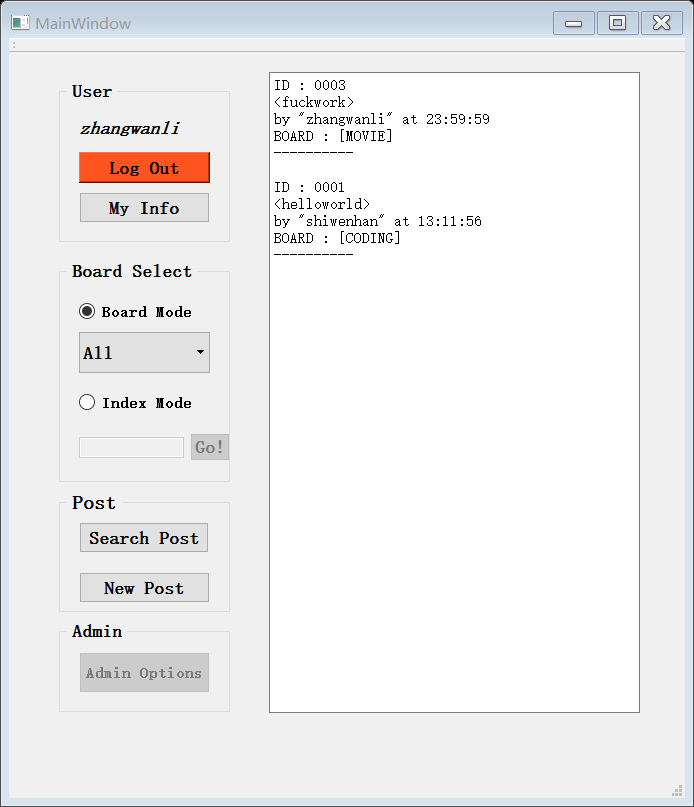




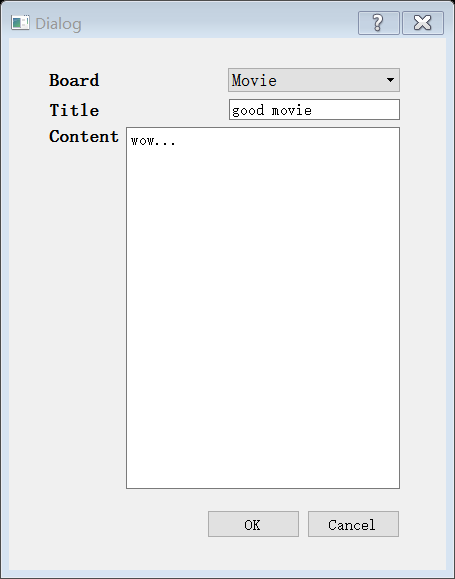
如上两图是通过某个ID来查找到指定帖子后，就可以对这个帖子进行一些操作，比如图二显示的评论功能。评论通过回显的形式来显示到显示器上，每按一次send即发送一个评论。这里的容错有：不能发送空评论，不能对一个不存在的帖子进行任何操作。

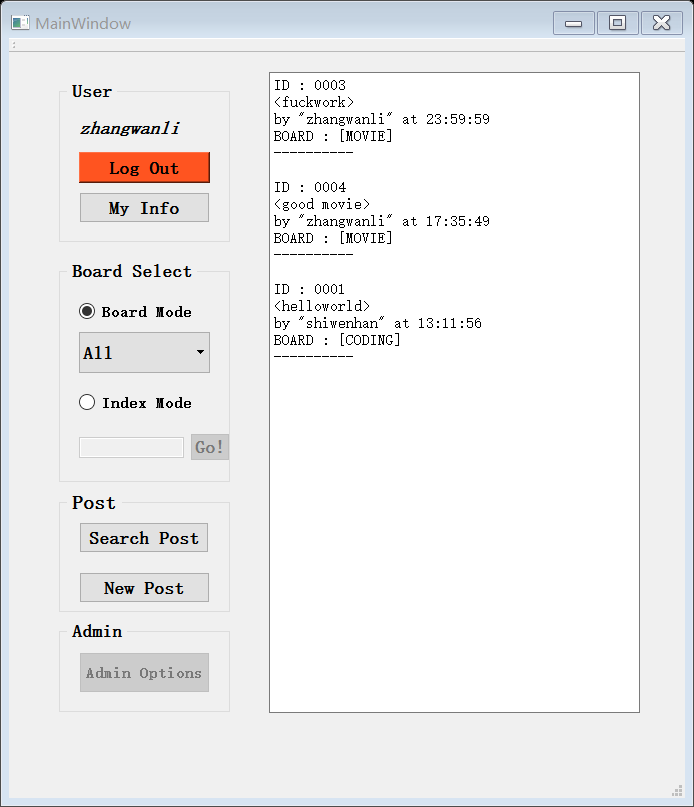


上图显示了一个GAME版主对0002号帖子进行删除的过程，删除之后，再次搜索提示帖子不存在。这里的容错有：某版块的版主只能删除自己版块下的帖子。



再次使用ALL来查看所有的帖子，可以发现0002号帖子已经被成功地删除。



上图为发帖框，可以输入一个帖子的题目，发到的版块和具体的内容。

再次用ALL查看所有帖子，可以看出这个帖子已经成功地发布。

#### 2.5.2 实现思路和代码

本论坛除了管理员之外均可以发帖、看帖和发评论，但只有版主能对帖子做删帖操作。这里前者的操作都极为简单，在这里不再赘述，只给出删帖的简单代码实现：

bool Moderator::*delPost*(QString &post\_id)

{

Post \* tempPost = nowUser->getPost(post\_id);

if(tempPost == NULL)

{

qDebug()<<"DELETE ERROR : NO SUCH POST"<<endl;

return false;

}

else

{

appendPostDelLog(f\_log,tempPost);

根据版块来删帖；

}

}

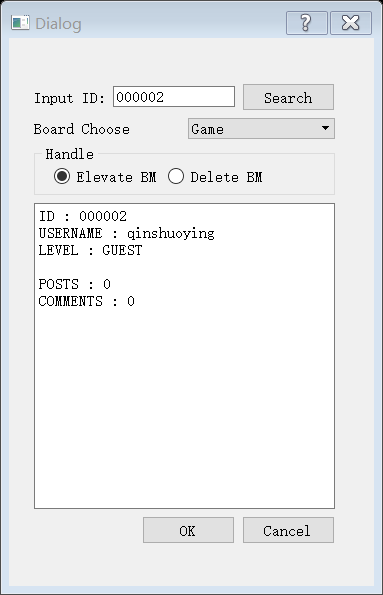
可以看出，删帖操作的本质是：讲论坛已经初始化好的Post\*容器，从中选出这个帖子并永久删除，这里需要使用QVector的索引和擦除操作，比STL封装的Vector功能更加友好和高效。当然，这些操作都要被加入到日志文件中（在第二版说明）。

#### 2.5.3 用到的重点知识简介

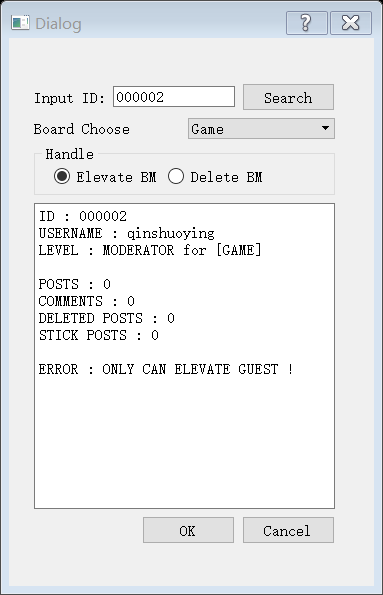
主要用到了QVector的操作，发帖的本质是new一个新对象的过程并append到容器中。发评论类似，而看帖的过程无非是给予ID索引一个容器中对象的过程。而删帖需要在目标容器（版块）中找到该帖子并删除的过程。

### 版主的提升和撤销

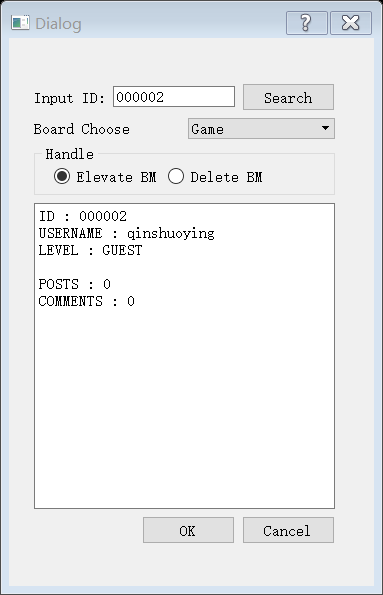
#### 2.6.1 实现截图



上图显示了ADMIN模式下的用户查询功能，通过键入任何一个注册过的用户的ID来查看他的具体私有数据，以确认他是否可以被提升为版主。如ID为000002的用户目前为普通用户，我们选择把他提升为GAME版块下的版主。



上图表示00002已经被提升为GAME版块的版主，此时我们如果有在提升他为版主的意愿，这是不被允许的，因为只有普通用户可以被提升为版主，因此他会打印提示报文。不过我们可以撤销这个用户的版主，直接将选项选为DELETE BM即可，点击确认可以完成本次操作。



上图表示，000002被撤销了版主，又回到了普通用户的行列。

#### 2.6.2 实现思路和代码

这里提升版主的本质是，将一个Guest类索引并被析构后，利用构造函数或set等手段，重新new一个Moderator类出来，将必要的数据拷贝至新类。请注意：本程序没有选择做类型提升或转换，那样做极不负责任而且容易出错，因此选择比较朴素的数据交互方法。

而撤销版主的本质基本相同，只不过是反过来操作两个类而已，在此不再赘述。

以下是两个方法的代码：

bool Admin::*makeBM*(QString &guest\_id, ENUM\_BOARD eb)

{

qDebug()<<"init fun :"<<eb<<endl;

for(User \* &r : forum.\_users)

{

QString id = r->getID();

QString pw = r->getPassword();

QString un = r->getUserName();

if(id == guest\_id && r->getLevel() == GUEST)

{

int np = r->*getNumPost*();

int nc = r->*getNumComment*();

User \* newBM = new Moderator(id,pw,un,eb);

delete r;

r = newBM;

r->*setMakeComNum*(nc);

r->*setMakePostNum*(np);

qDebug()<<"ID : "<<r->getID()<<" ELEVATE MODERTATOR SUCCESS!"<<endl;

qDebug()<<"NEW MODERATOR FOR : "<<boardToStr(eb)<<endl;

return true;

}

}

qDebug()<<"ELEVATE MODERATOR FAILDE!"<<endl;

return false;

}

bool Admin::*delBM*(QString &guest\_id)

{

for(User \* &r : forum.\_users)

{

QString id = r->getID();

QString pw = r->getPassword();

QString un = r->getUserName();

if(id == guest\_id && r->getLevel() == MODERATOR)

{

int nc = r->*getNumComment*();

int np = r->*getNumPost*();

User \* newGuest = new Guest(id,pw,un,GUEST,np,nc);

delete r;

r = newGuest;

qDebug()<<"ID : "<<r->getID()<<" DELETE MODERTATOR SUCCESS!"<<endl;

return true;

}

}

qDebug()<<"DELETE MODERATOR FAILDE!"<<endl;

return false;

}

这两个方法都只在Admin类中被重写。

#### 2.6.3 用到的重点知识简介

这里主要涉及当一个类需要向他的父类或是子类转化时，我们究竟怎么做才是安全并且可靠得？这里本程序选择较为朴素的方法进行数据交互，而不是上溯或者下溯。

# 题目二：论坛系统高级版

## 二题题目以及需求分析

本题目需要在题目一的基础上进一步完善功能，包括对置顶、索引功能的实现等，重点在于利用重载等方法在一定基础的代码上继续开发程序，而不是大范围的重构或重写。

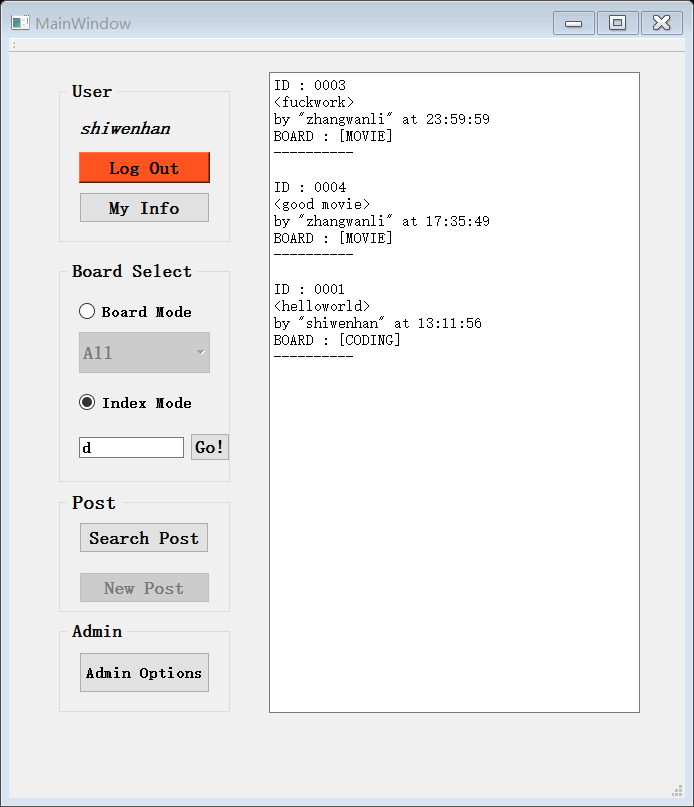
## 各功能实现模块

在题目一的基础上，新增加了如下功能：

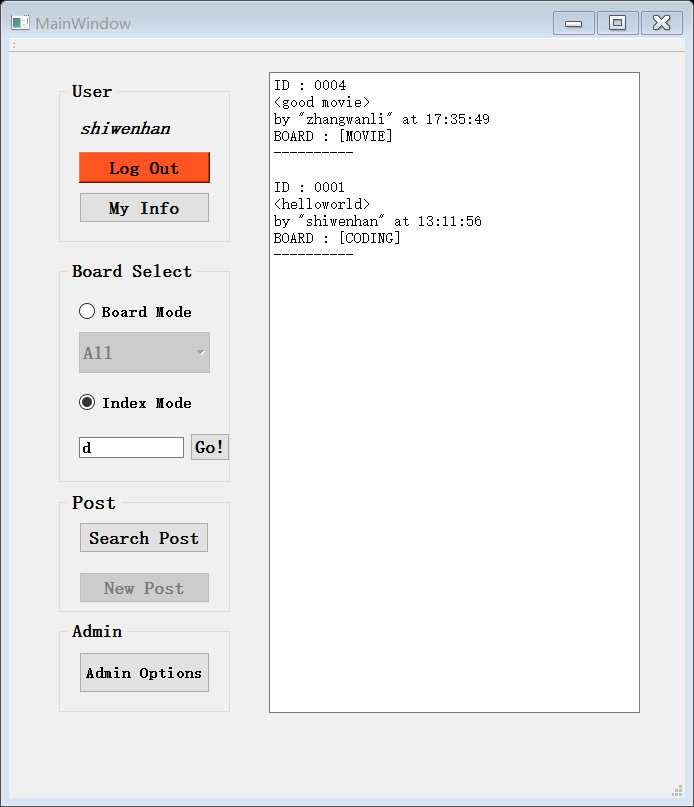
1. 重载搜索帖子函数以适应索引模式的需要。
2. 实现帖子的置顶操作和浏览效果。
3. 文件的输入输出，日志文件的保存。
4. 重载流操作符<<和>>，以简化文件输入输出。
5. 异常捕获。

### 索引查帖

#### 2.7.1 实现截图



可以直接在主页面实现这个功能，如上图，进入INDEX MODE后，输入想要索引的关键字，再点击GO即可实现对字母“d”的索引。



如上图，由于索引了“d”，只留下了两个符合要求的帖子。请注意，索引模式完全兼容置顶操作，即如果有被置顶帖子，索引后的帖子表也会置顶帖优先显示，后面我们会看到。

#### 2.7.2 实现思路和代码

索引需要利用QT模板类中封装的对于QString判断是否a是b的子串的判断函数，如果有这样的功能性函数被封装好，那我们仅需要对每一个帖子标题进行如此的判断即可，如果符合，我们就将它们单列出来，以形成索引之后的结果。

QVector<Post \* > & getPostList(ENUM\_BOARD eb);

void getPostList(QString& key, QVector<Post \*> &tgtPosts); // plus : overload

上面两个函数可以看出重载的过程，第一个函数需要一个版块枚举，返回一个帖子表，而第二个函数由于要进行索引，因此不再传入版块信息，而是一个QString对象，即索引关键字。

简略的代码如下：

void User::getPostList(QString &key,QVector <Post\*> & tgtPosts)

{

qDebug()<<"START CONSTURCT POSTS\_VEC ..."<<endl;

QVector<Post \*> topPosts;

QVector<Post \*> normPosts;

对每一个板块排序，置顶在先；

QVector<Post \*> posts = topPosts + normPosts;

for(Post\* &r : posts)

{

qDebug()<<"FIND\_A\_INDEX !"<<endl;

int index = r->getPostTitle().indexOf(key);

if(index != -1)

tgtPosts.append(r);

}

}

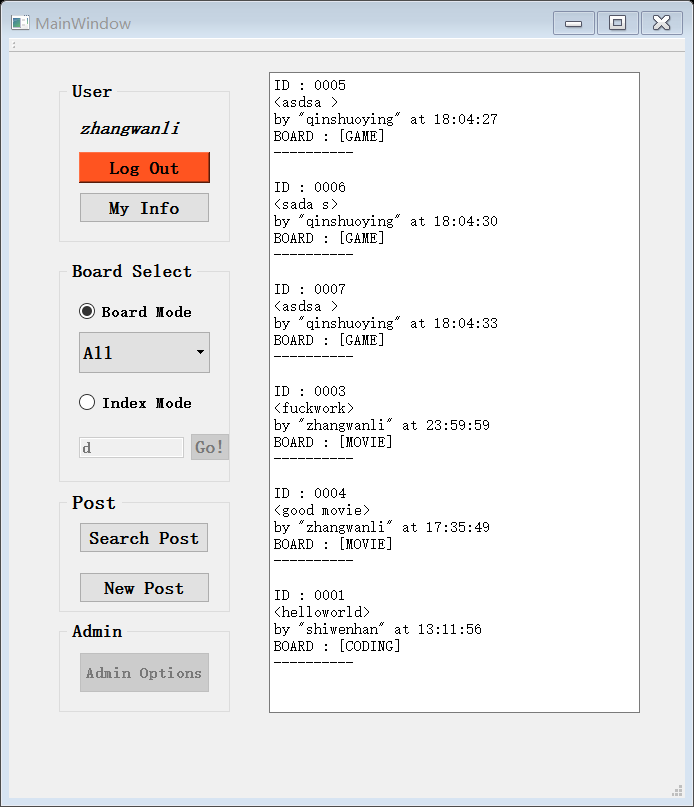
可以看出，这个操作和置顶兼容。利用indexOf方法，如果能找到这个index，则判断子串成功，索引完成。

#### 2.7.3 用到的重点知识简介

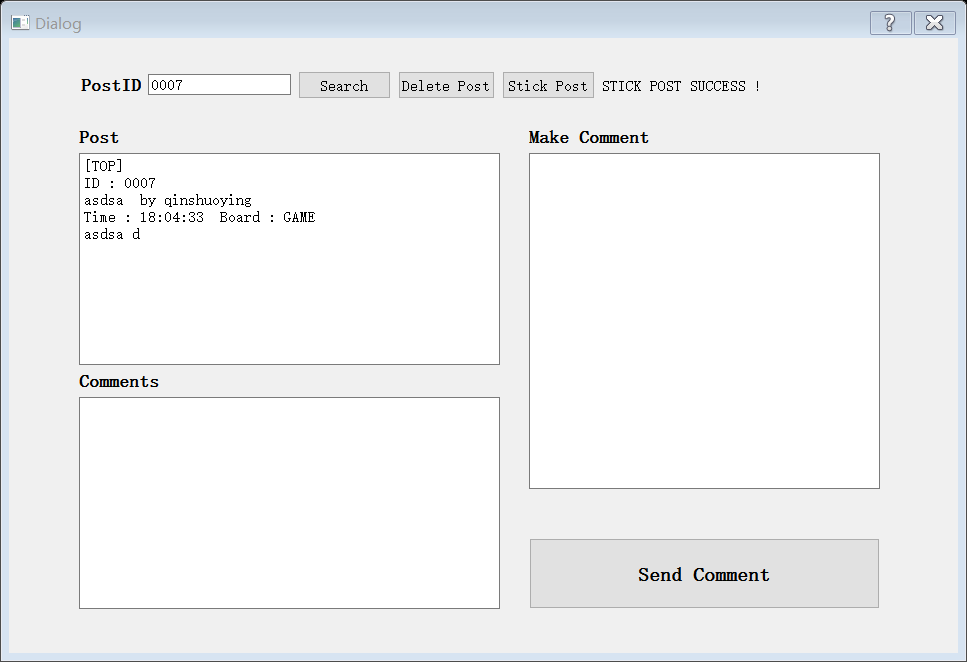
主要是对于QString类的indexOf方法的运用，设计到对置顶操作的兼容。

### 帖子的置顶和浏览效果

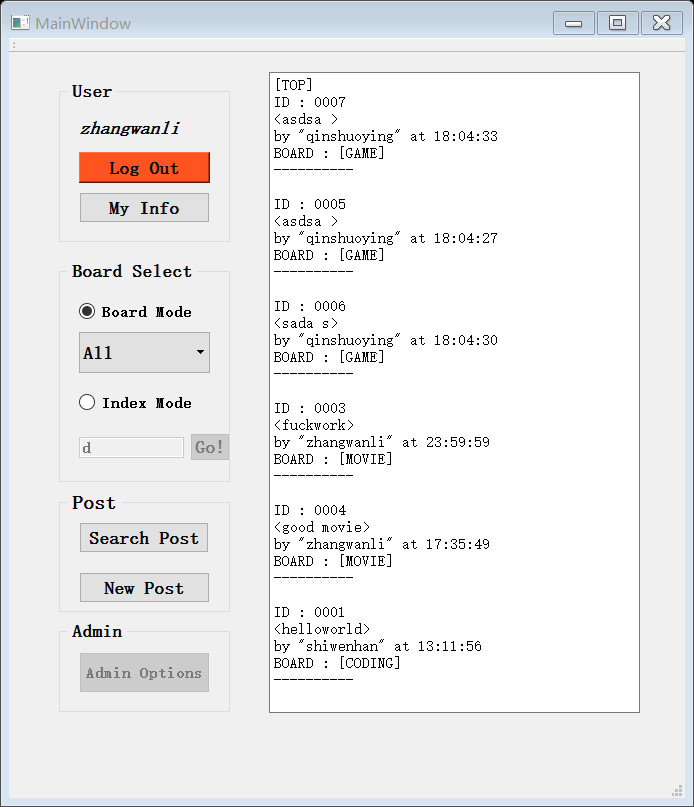
#### 2.8.1 实现截图



上图展示了若干个帖子，现在随便找一个GAME版主来对0007帖子置顶。



可以看到操作成功后，这个帖子已经显示“TOP”字样，返回主页面看看效果。



可以看出0007号帖子已经被顶到了最上层。这个操作可以多次进行，最近一次被置顶的帖子位于最上的位置。

#### 2.8.2 实现思路和代码

帖子置顶的本质是，对于某一个帖子增加一个新的属性，明显它是一个布尔量，即它是否已经被置顶。每一个帖子初始化的时候，这个选项都应该为false。而置顶操作的本质就是把某个帖子的这个属性给set成true。

至于显示效果，我们需要对帖子列表重新排序，stick属性为true的帖子被放在容器的顶部，这样在利用迭代器顺序打印容器，就可以看到置顶的效果。

bool Moderator::*stickPost*(QString& postID)

{

Post \* tgtPost = getPost(postID);

if(tgtPost == NULL)

{

qDebug()<<"CANT STICK A NONE EXIST POST ! "<<endl;

return false;

}

else if(tgtPost->getPostBoard() != \_board)

{

qDebug()<<"CANT STICK A NO\_MATCH BOARD POST ! "<<endl;

return false;

}

else

{

tgtPost->setSticky();

appendPostStickLog(f\_log,tgtPost);

Post \* temp = new Post(\*tgtPost);

QVector <Post\*> temps ;

temps.append(tgtPost);

在相应的版块中找到这个帖子并set置顶属性为true

}

}

不难看出，置顶就是查找+修改的过程。

#### 2.8.3 用到的重点知识简介

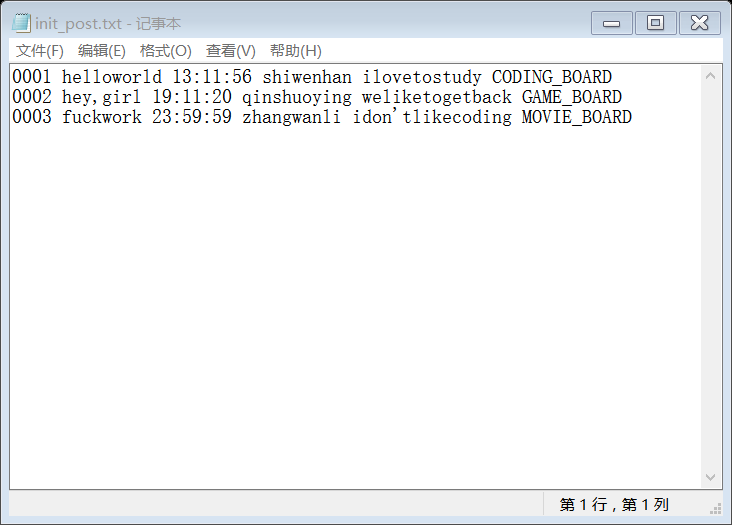
同上，均是Post的一些索引操作，需要新添加set方法来对stick属性操作。

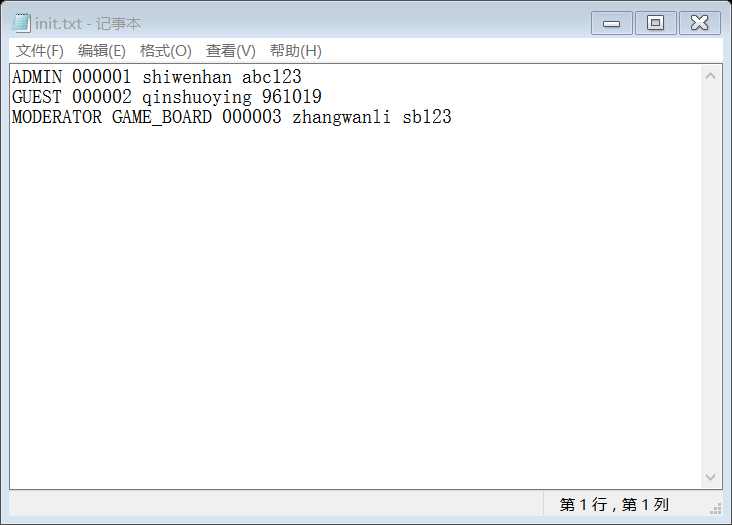
### 文件的输入输出以及<<,>>的重载

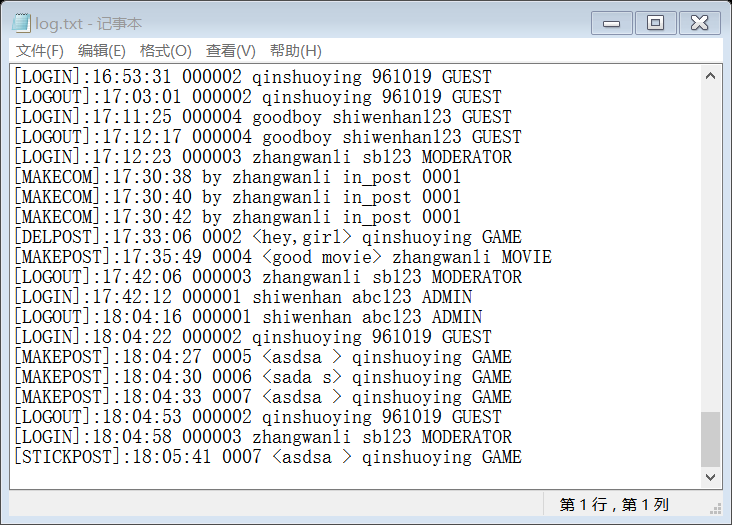
#### 2.9.1 实现截图

本程序设计三个文件，两个初始化文件，负责初始化用户和帖子。

一个日志文件，负责记录该次系统使用的一切操作信息。截图如下：







以上三个文件都以txt的形式存储，分别是：帖子初始化文件、用户信息文件、日志文件。

其中日志文件的各种操作都是刚刚我们在测试的时候进行的。

#### 2.9.2 实现思路和代码

在这里仅给出初始化用户的过程，这个过程涉及了从init文件中读取并载入数据，是一切多态的起点。并且需要重载>>来完成对类的构造，这个过程应该表现得极为简单，它才能是一个好的重载，至于<<的重载过于简单在此不再赘述。其他文件操作过于简单也不再赘述。

while(true)

{

if(in.atEnd()) break;

QString qs\_el ;

in >> qs\_el;

qDebug()<<"READ USER FIRST SEG : "<<qs\_el<<endl;

if(qs\_el == "GUEST")

{

User \* newuser = new Guest();

in >> newuser; // apply to new >>

\_users.append(newuser);

\_numUsers ++ ;

}

else if(qs\_el == "ADMIN")

{

User \* newuser = new Admin();

in >> newuser;

\_users.append(newuser);

\_numUsers ++ ;

}

else if(qs\_el == "MODERATOR")

{

User \* newuser = new Moderator();

in >> newuser;

\_users.append(newuser);

\_numUsers ++ ;

}

else

{

qDebug()<<"cant decode suchline in file!"<<endl;

}

}

file.*close*();

}

可以从如上代码段看出，以一个无限循环来控制输入流，一次仅构造一个对应的类。但是不管这个类是什么，我们相当于间接地把>>也重写了，都只需要调用in >> newuser;，实现起来非常简单实用。这里没用用标准流，而是QT中封装的QTEXTSTREAM。

#### 2.9.3 用到的重点知识简介

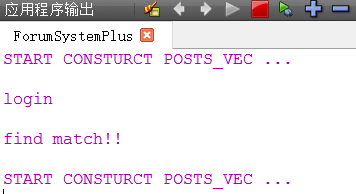
这里涉及对QFile，QTextStream，间接重写，重载操作符等知识，这些知识大多都来自《C++ PRIMER》，一些QT内置的类的知识来自于QT的官方手册。需要注意的是，我们想用最简洁的方法来实现，只能是in >> newuser;这一句话就足以说明问题，对于类的一个操作，就应该如此不拖泥带水。但是请注意，这并不代表不加区分，因为每一个用户是不确定的，如果他是版主，他需要携带额外的版块信息，也就是说，不同类的读入方法本身是不同的。但是没关系，我们仍然可以利用多态和重载的知识来统一成in >> newuser;这一句话。

### 异常捕获

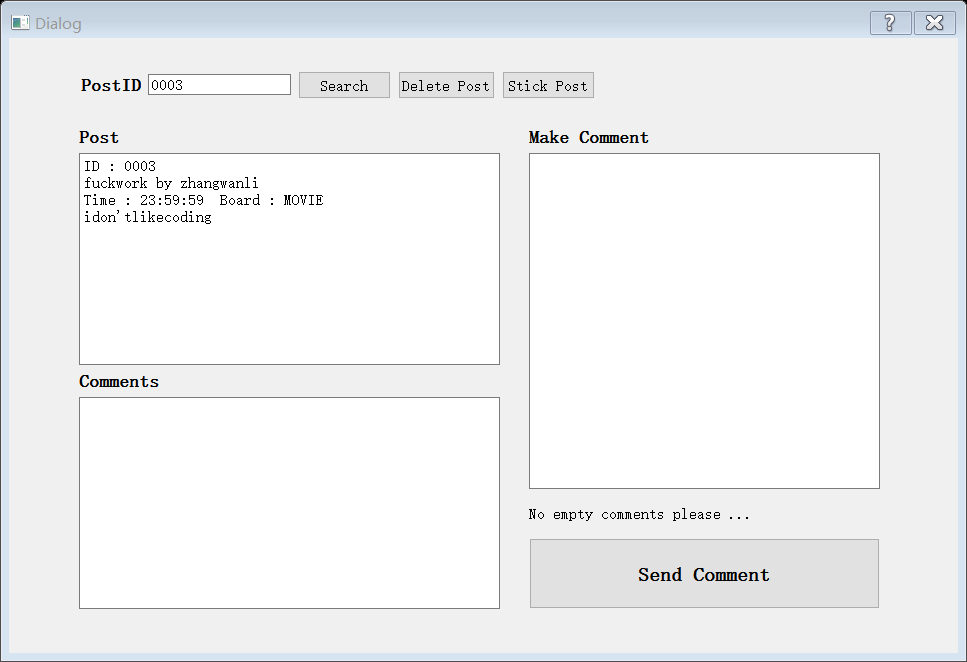
#### 2.10.1 实现截图

我们避免使用try-catch块，而是采用最简单的报文-退出机制来确保程序的可靠。我们主要对三个方面做了异常捕获：关键用户操作、数据可行性、文件读写。

我们采用了QDebug类、日志文件、QLabel等来以错误报文的形式来捕捉异常。



如上图，我们利用简单有效的Qdebug来随时随地打印信息。



再如上图，我们不准发送无意义的空评论，我们会打印相关的控制报文来提示，当然也禁止了本次的操作。

#### 2.10.2 实现思路和代码

省略，异常捕获贯穿整个程序始终，在此不再一一赘述。

#### 总体心得

本次实验利用的C++的面向对象特性，是一次真正意义上的OOP编程。全部程序在四天内，每天至少六个小时的工程量下一气呵成，有酣畅淋漓之感，期间设计对多态的讨论和运用，>>操作符重载的困难，QT内置类的探索和QT图形化之旅的钻研和发现。整体过程给我最大的感悟是，我通过这样一个小而精巧的过程收获到了C++的编程思想。全程序严格遵循多态的定义，以父接口操纵全局，避免使用任何子类指针，灵活运用虚函数的方法和抛错函数结合来实现真正意义上的多态，当一个子类需要某个功能时，再重写它，可以最大限度地保证结构的清晰化。

而在具体功能的实现上，反而在用了两天的对类的精心设计下，变得极其简单和轻松。最明显的例子就是logout操作，如果你确定了自己的访问者模式，确定操纵的是对象指针而非对象本身，那么logout方法里只需要一句话，ptr == NULL；即可（当然省略了一些细节）。如此清晰的方法是建立在精巧的设计上的，骨架确定你的程序是否有良好的性能和思路，而其中的实现是次要的。

总之，本次实验让我的工程能力和编程知识水平有了很大提高，更是初始了设计模式，运用了多态和继承等概念，结合OOP思想，编出来的第一个大作业。希望以后还能有这样的机会，因为这印证了一个良好的学习方法：实践出真知。