$C++$学习论坛设计程序说明

1. 问题综述

1.1 课程设计目的

> 设计实现一个C++学习论坛系统，作为学生和老师的交流平台。

1.2 功能要求

1.2.1 用户设计

> 1. 用户包含普通用户、版主、管理员、匿名用户四类，要求以继承方式实现，所有用户均可以登入登出。
> 2. 除匿名用户外，其余四类均可以实现删帖、发帖、发评论，并根据题目要求进行权限限制
> 3. 管理员具有最高权限，可以实现把普通用户提升为版主及撤销版主资格

1.2.2 论坛设计

> 1. 论坛包括版块、帖子及下属评论。
> 2. 帖子与评论含有包括内容、作者、创建时间等信息，支持随时移除和添加相应内容。

1.2.3 其余要求

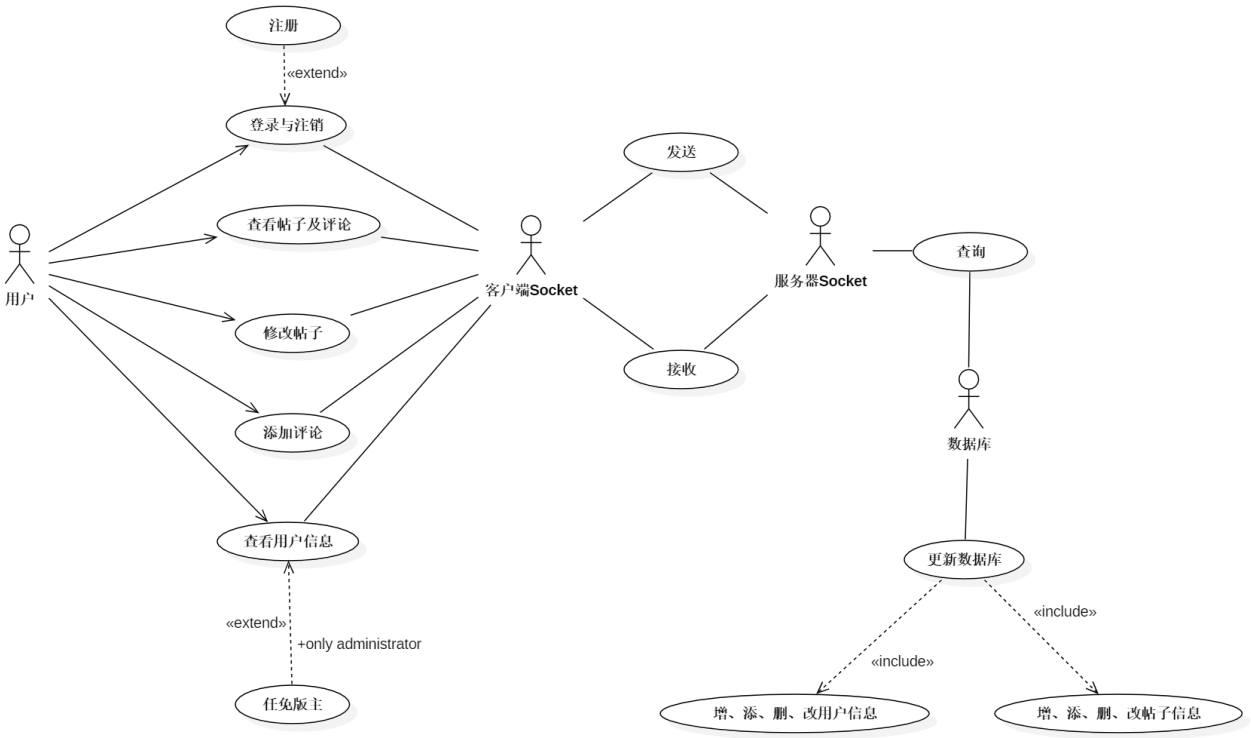
> 1. 实现所有数据的文件存取功能，即重启程序可以访问上次使用的数据
> 2. 支持网络通信，采用客户端/服务器模型

2. 开发环境与相应配置

> - IDE：Qt Creator 5.6
> - 操作系统：windows 10
> - 编程语言：C++11（结合QT封装的相应库文件）
> - 数据库：QSQLITE
> - 网络通信：基于TCP/IP协议

3. 总体结构说明

3.1 最终版用例模型示意

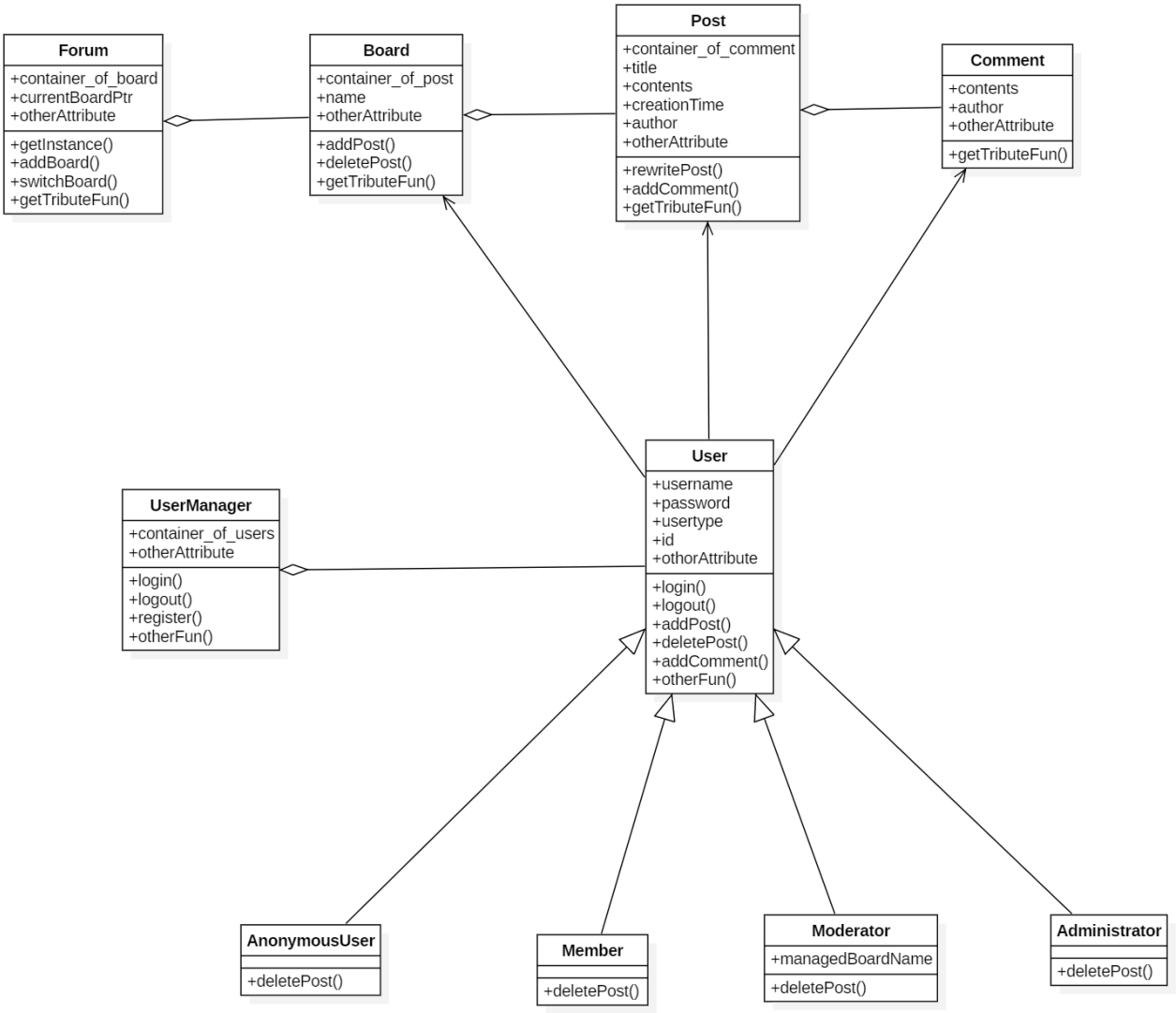


3.2 第一版

3.2.1 文字说明

- 总体上采用层层嵌套结构，所有内容均存在于内存中
- 关于用户的设计：
  - user 为基类，普通用户member、版主moderator、管理员administrator均继承自user。
  - 构造类 userManager，运用单例模式实现全局唯一，内含一个vector容器，管理指向所有用户的指针。注册时只需在new一个新的对象，再将指向该对象的指针存入vector即可
  - 在userManager类中设计一根指向当前用户的指针currentUser，注销和登录时切换指向的对象。
- 关于论坛的设计：
  - 构造论坛类Forum，运用单例模式实现全局唯一，内含一个vector，元素为指向版块类board的指针，实现对版块类的管理
  - 构造版块类Board，内含一个vector，元素为指向帖子类Post的指针，实现对板块下帖子的管理
  - 构造帖子类Post，内含一个vector，元素为指向评论类Comment的指针，实现对帖子下评论的管理

3.2.2 UML 类与类图



3.3 第二版

3.3.1 文字说明

> - 考虑到让所有用户的实例同时存在于论坛程序内不符合逻辑，且将所有帖子、评论全部存于内存可维护性较差，当数据规模增大多用户并发时不利于查找与修改且占用内存较大。基于此，并且考虑到程序的可维护性及第二、三版的要求，决定采用数据库存储所有数据。
> - 在采用数据库进行存储时，在版块类中不再设计vector存储帖子，转而将帖子所归属的版块作为一项存数据库表。帖子类中同理也不在设计vector存所有的评论。当需要用到相关操作的时候，直接从数据库中取出相应对象。进行修改与添加操作时也最终转化为对数据库相应表项的操作。此外，userManager类中也不在设计vector存所有用户实例。当用户成功登陆时创建对象，在用户退出时销毁对象。通过这样的处理，板块类、帖子类与评论类之间进一步解耦合，内存空间占用进一步优化。当然，这样的设计也更加符合实际。

- 在具体实现的时候，创建Mysql类对数据库进行操作。运用单例模式保证全局唯一，使操作更加安全。Mysql类设计多种接口，各类通过接口访问数据库，具体设计如下table：
  - user

QString primary key	QString	QString	int	bool
username	password	id	type	isOnline

  - manage

QString primary key	QString
username	board

  - comment

QString primary key	QString	QString	QString	QString
id	contents	postId	authorId	creationTime

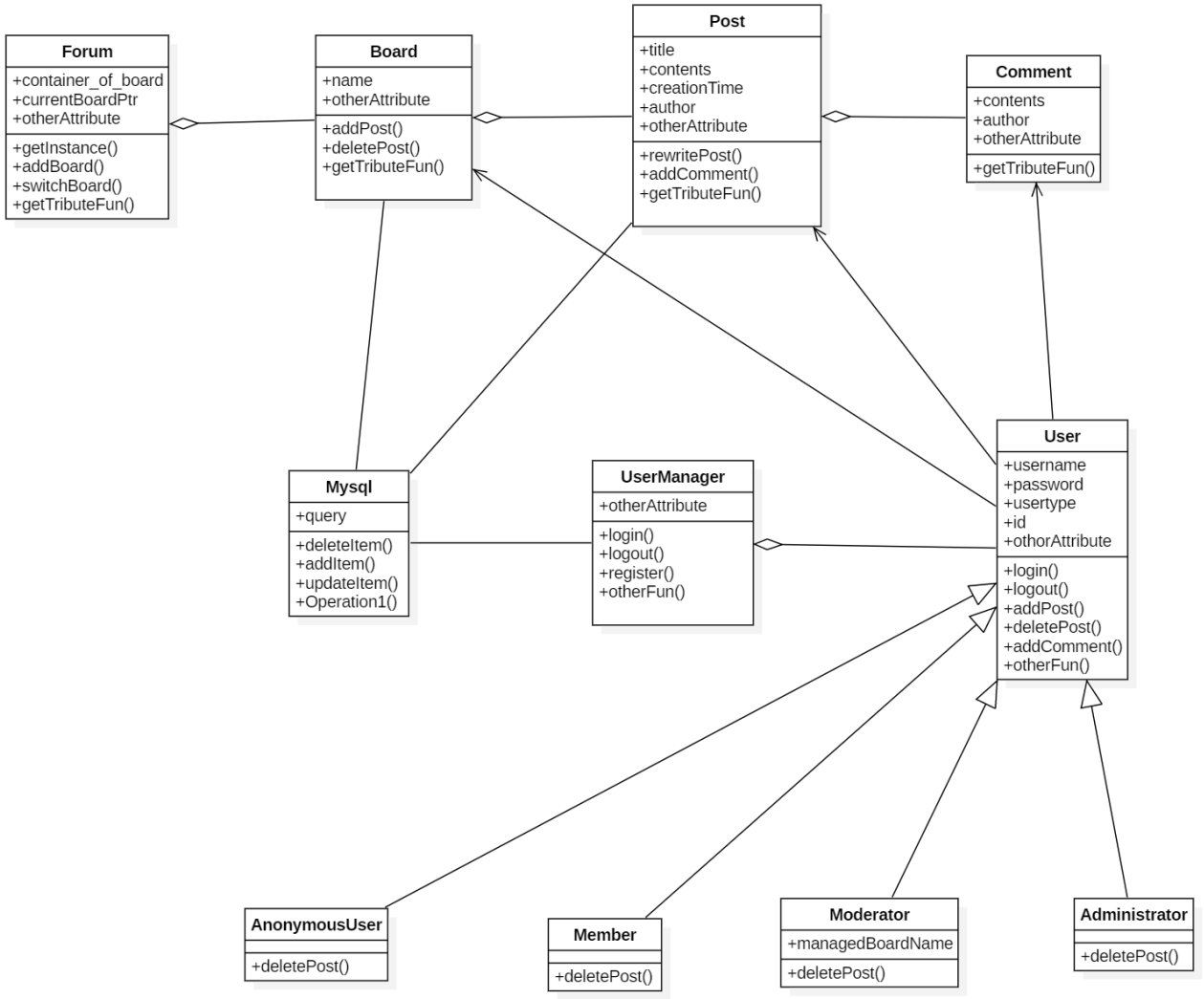
  - post

QString primary key	QString	QString	QString	QString	QString	QString
id	title	contents	authorId	boardName	isModify	creationTime

  - anonymity

QString primary key	type
username	int

3.3.2 UML 类与类图



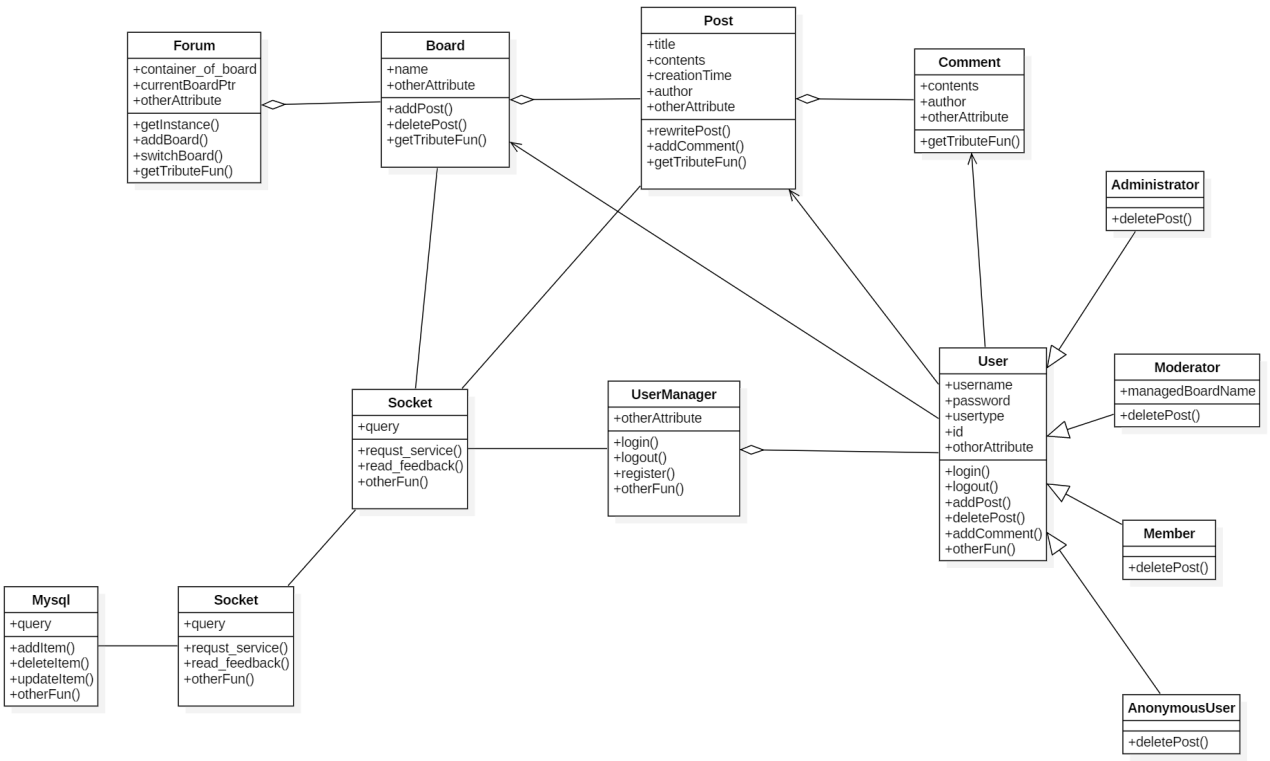
3.4 第三版

3.4.1 文字说明

> - 第三版要求实现网络通信，在此采用客户端/服务器模型。考虑到程序的可维护性，在此采用$Client$端负责展示，$Sever$端负责后台计算与操作数据库
> - 大题思路上，只需要将源程序改为$Client$端。而所有直接访问数据库的操作全部改为通过$Socket$类进行通信。当用户在客户端点击产生请求的时，调用$Socket$类，$Socket$类访问服务器。而$Socket$端接收客户端发来的请求，根据请求类型查询数据库并进行判断，最后返回结果。$Client$端收到相应，完成相应前端动作。
> - 具体实现上，采用TCP/IP协议，首先要设计包格式，便于封装与解包，具体如下：
>
> | 包长度 | 请求类型 | 参数1 | 参数2 | 参数3 |
> | --- | --- | --- | --- | --- |

- 在$Server$端，启动后开始监听。每有一个客户端尝试建立连接，则新建一个线程，在线程中新建一个$tcpSocket$，之后通信建立，双方开始通信，当客户端失去连接之后，连接断开。需要注意的是，多个线程会尝试同时访问数据库。因此，如果运用单例模式实现数据库的操作类的话，需要在初始化和操作中加入互斥锁避免异常情况的出现。

3.4.2 UML 类与类图



4. 亮点与特色

4.1 附加功能说明

除题目要求的所有功能全部实现了以外，该程序还拥有如下功能与特色，简要阐述如下：

- 提供注册功能，允许用户注册为普通用户
- 用户发布的帖子允许修改
- 允许多用户同时登陆，进行所有允许的操作，且结果可以实现实时同步
- 丰富的异常处理(具体演示见使用说明与测试)
  - 登录时异常

    > 包括：没有输入用户名/密码、用户名与密码不匹配

- 注册时异常
  > 包括：没有输入用户名/密码、两次输入的密码不一样、用户名已经被注册
- 发帖/修改帖子异常
  > 包括：没有填写标题/内容
- 查看/修改/删除帖子异常
  > 包括：帖子已经被其他用户删除、该用户不具有权限等(如被提升为版主的用户在删除帖子的时候被管理员撤销了版主身份)
- 网络异常
  > 包括：客户端登录时服务器没有打开、客户端运行过程中服务器突然关闭
- 文件读取异常
  > 包括：服务器初始化前数据库被删除

4.2 设计模式使用

- 单例模式
  - 模式说明：
    > 单例模式属于创建型模式，它提供了一种创建对象的最佳方式。这种模式涉及到一个单一的类，该类负责创建自己的对象，同时确保只有单个对象被创建。这个类提供了一种访问其唯一的对象的方式，可以直接访问，不需要实例化该类的对象。
  - $Client$端
    - $Class \quad UserManager$ => 一个客户端同时只有一个用户处于登陆状态

```
//单线程版本，未加互斥锁，获取单例指针
UserManager* UserManager::getInstance(){
    static UserManager* m_instance = nullptr;
    if(m_instance == nullptr){
        m_instance = new UserManager();
    }
    return m_instance;
}
```

    - $Class \quad Forum$ => 一个客户端只有一个论坛存在

```
Forum *Forum::getInstance()
{
    static Forum* m_instance = nullptr;
    if(m_instance == nullptr){
        m_instance = new Forum();
    }
    return m_instance;
}
```

- $Server$端
  - $Class\quad Mysql$ => 服务器端维护一个数据库（注：由于服务器端采用多线程，故相应函数要加互斥锁）

```
//获取单例，多线程共用，需要保护
Mysql* Mysql::getInstance(){
    Mysql* tmp = m_instance.load(std::memory_order_relaxed);

 std::atomic_thread_fence(std::memory_order_acquire);//获取内存fence
    if (tmp == nullptr) {
        std::lock_guard<std::mutex> lock(m_mutex);
        tmp = m_instance.load(std::memory_order_relaxed);
        if (tmp == nullptr) {
            tmp = new Mysql;

 std::atomic_thread_fence(std::memory_order_release);//释放内存fence
            m_instance.store(tmp, std::memory_order_relaxed);
        }
    }
    return tmp;
}
```

- 代理模式
  - 模式说明：已有对象不方便直接访问，通过代理为其它对象提供访问机制
    - 不变点：·接口不变，与原对象提供完全相同的接口·功能中不变的部分：最终仍然完成原对象的功能

        - 变化点：增加额外的功能（功能中变化的部分）
    - Clint端的Socket设计，在第二版中是直接访问数据库进行文件读写操作，在第三版中数据库被放到了服务器端，需要通过网络通信对数据库进行访问。该类作为中间层，在维持接口不变的前提下依旧为项目中的其他类提供了访问、修改数据的功能。
- 适配器模式
    - 模式说明：

      > 适配器模式是作为两个不兼容的接口之间的桥梁。这种类型的设计模式属于结构型模式，它结合了两个独立接口的功能。这种模式涉及到一个单一的类，该类负责加入独立的或不兼容的接口功能。

    - 具体到我的设计，在原本的$QTcpServer$类中不存在多用户同时与服务器端建立$socket$连接的接口，原本的$incomingConnection$函数只是实现了一个客户端$socket$和一个服务器$socket$进行连接，于是我用一个$Server$类继承了$QTcpServer$类，并重写了$incomingConnection$函数。(注：该设计满足适配器的替换原则：子类必须能够替换它的父类即子类的接口约定和限制必须与父类保持一致。而且也实现了符合系统规定的功能)

      ```
      // This function is called by QTcpServer when a new connection is available.
      void MyServer::incomingConnection(qintptr socketDescriptor)
      {
          // We have a new connection
          qDebug() << socketDescriptor << " Connecting...";

          // Every new connection will be run in a newly created thread
          MyThread *thread = new MyThread(socketDescriptor, this);

          // connect signal/slot
          // once a thread is not needed, it will be beleted later
          connect(thread, SIGNAL(finished()), thread, SLOT(deleteLater()));

          thread->start();
      }
      ```

- 组合模式
  - 模式说明：

    > 组合模式，又叫部分整体模式，是用于把一组相似的对象当作一个单一的对象。组合模式依据树形结构来组合对象，用来表示部分以及整体层次。这种类型的设计模式属于结构型模式，它创建了对象组的树形结构。这种模式创建了一个包含自己对象组的类。该类提供了修改相同对象组的方式。

  - 在实现中，以$User$类为基类，$deletePost$函数设计为纯虚函数，会员类$member$、版主类$moderator$、管理员类$administrator$、匿名用户类$anonymousUser$均继承自$User$类，重写了$deletePost$方法。这样做的好处是叶节点(子类)和枝节点(基类)对于外界没有区别，他们具备完全一致的接口。

4.3 多线程的同步与互斥

> 在服务器端实现多线程，每有一个Client尝试连接Server时，Server新建一个线程，在线程内新建一个$QTcpSocket$，之后跟客户端建立长连接。实现客户端与服务器之间的通信，当客户端注销断开连接的时候，服务器感知后销毁该线程。数据库运用单例模式实现了全局唯一，所有操作均需加锁确保线程安全。

```
void MyThread::run()
{
    // thread starts here
    qDebug() << " Thread started";

    socket = new QTcpSocket();

    // set the ID
    if(!socket->setSocketDescriptor(this->socketDescriptor))
    {
        // something's wrong, we just emit a signal
        emit error(socket->error());
        return;
    }
    ......
}
```

5. 类的设计

- 由于类的函数命名格式较为规范，基本可以达到见名知义的效果。且代码注释较为详细，故在此处只针对类的功能进行描述，对各个函数的功能不再赘述。

5.1 UI类

类名	功能简述
mainwindow	论坛主界面
logindlg	登录界面
infodlg	个人信息界面
postdlg	展示帖子界面
editdlg	编辑帖子(准备发布)界面
addcommentdlg	为帖子添加评论界面
modifypostdlg	修改帖子界面
chooseboarddlg	为版主分配管理版块界面
registerdlg	注册界面

5.2 功能类

- Client端

类名	功能简述
user	用户基类，定义用户各种属性，定义发帖、删帖、发评论等函数
member	普通会员类，继承自用户基类，重写基类删帖函数
moderator	版主类，继承自用户基类，重写基类删帖函数
administrator	管理员类，继承自用户基类，重写基类删帖函数，新增任免版主函数
anonymoususer	匿名用户类，继承自用户基类
usermanager	用户管理类，管理登录、注销、注册及当前用户
comment	评论类
post	帖子类，管理所有评论，包括增加评论
board	板块类，管理所有帖子，包括增删改帖子
forum	论坛类，管理所有版块，包括增删版块
Socket	通信类，负责客户端与服务器的信息交互

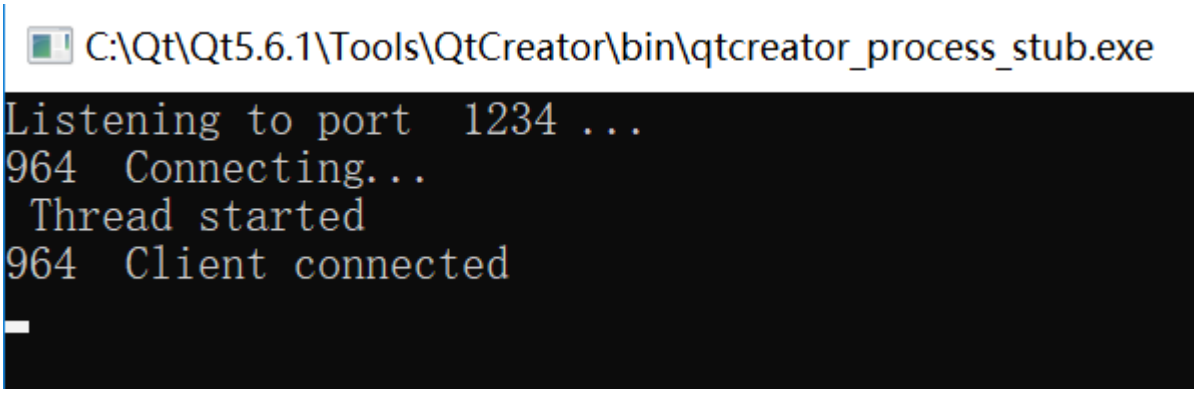
- Server端

类名	功能简述
myserver	服务器类，监听来自客户端的请求
mythread	线程类，在线程中新建socket实现与客户端的连接与通信
mysql	数据库类，负责与数据库的信息交互

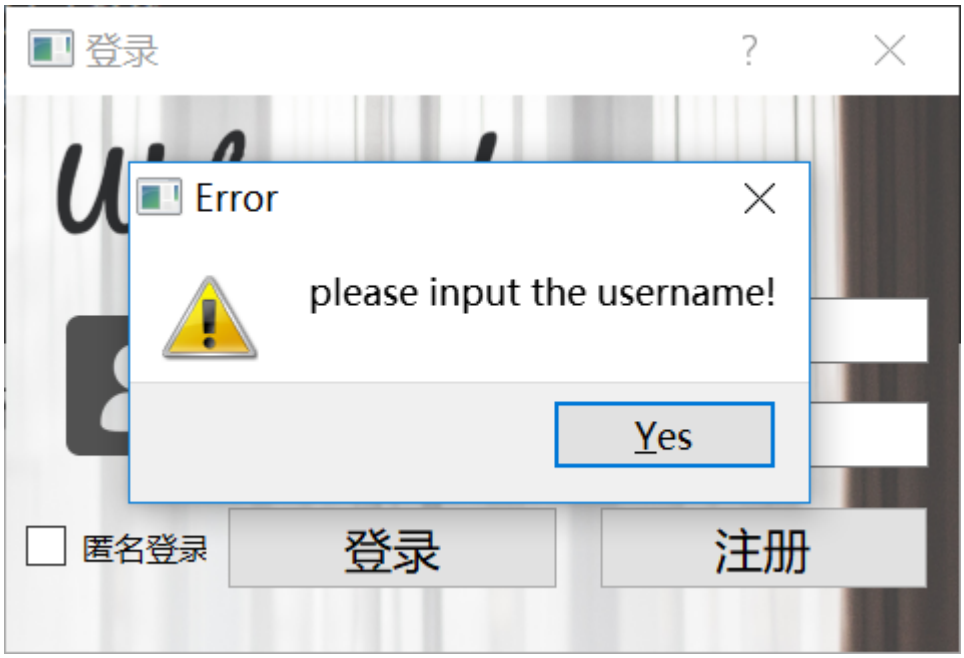
6. 使用说明与测试

6.1 登录、注销与注册

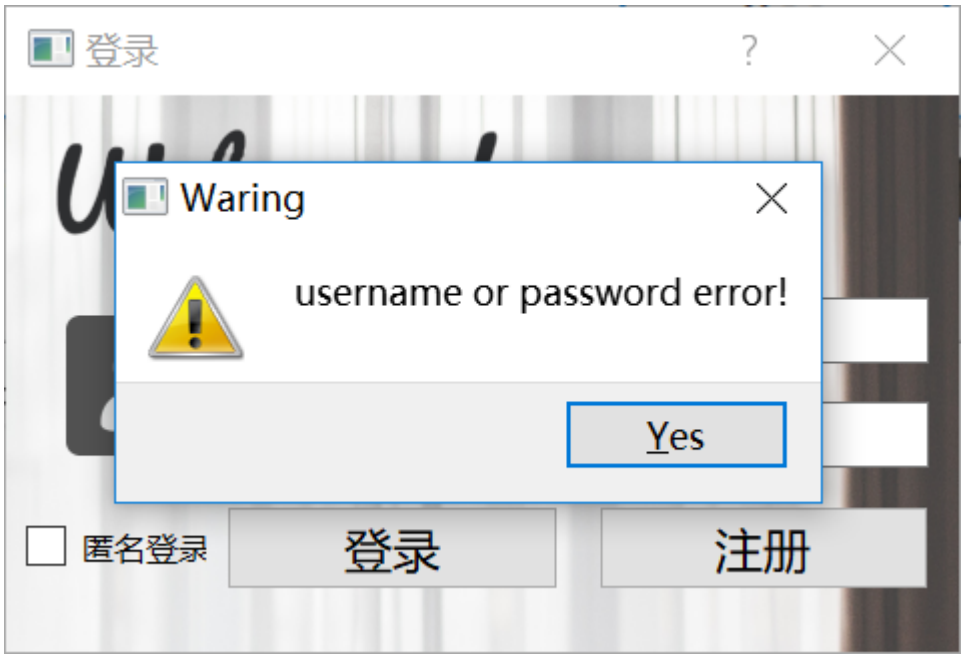
- 服务器打开，双击打开客户端准备登陆，服务器端显示Socket已建立



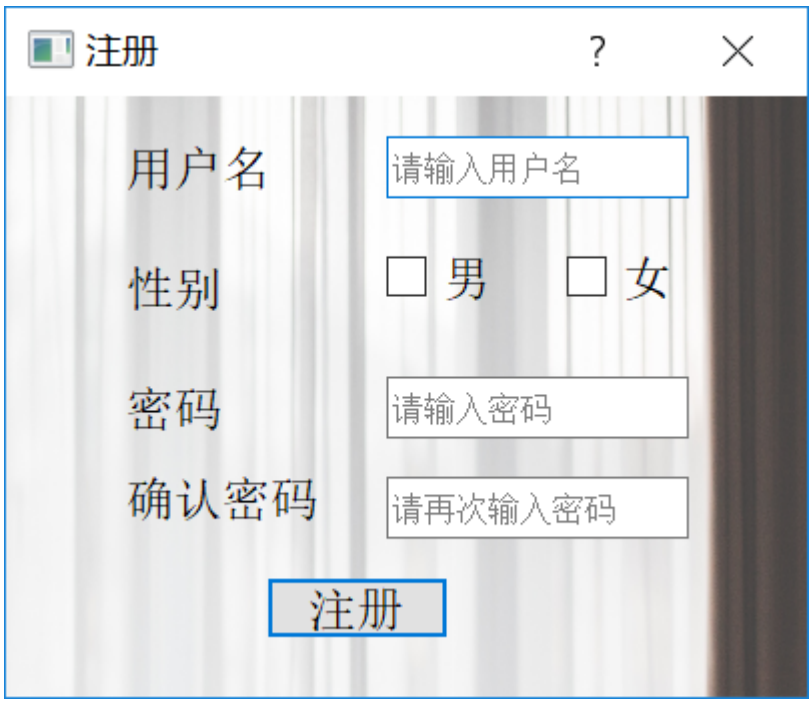
- 点击登陆，若未填写用户名/密码，则进行提示



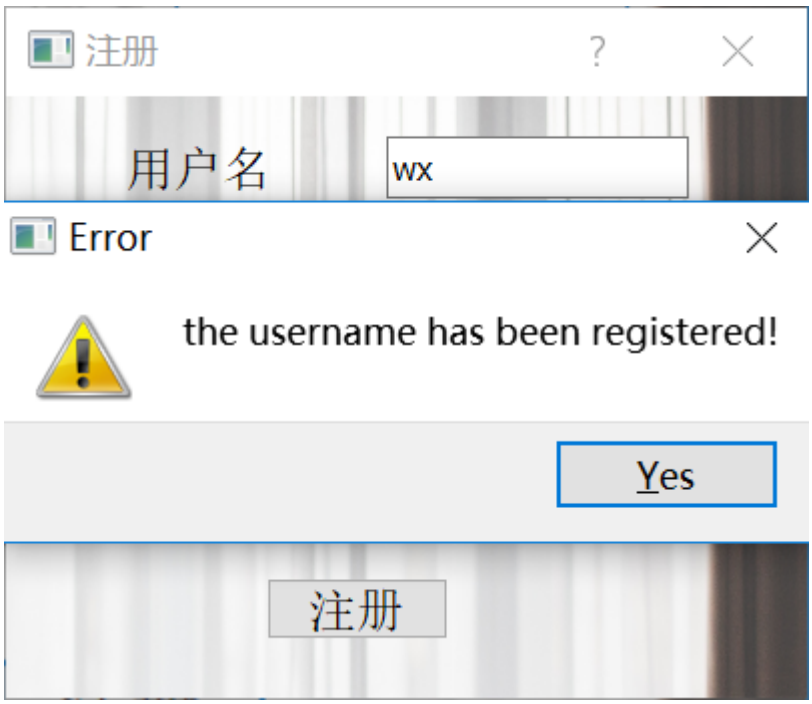
- 若用户名和密码不匹配，则进行提示



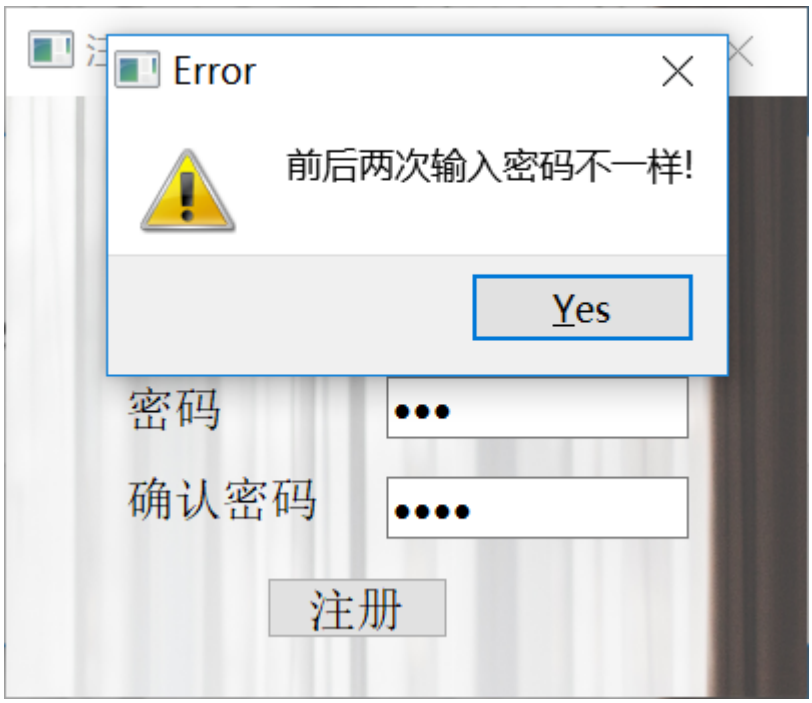
- 如需要注册则单击注册，弹出注册窗口



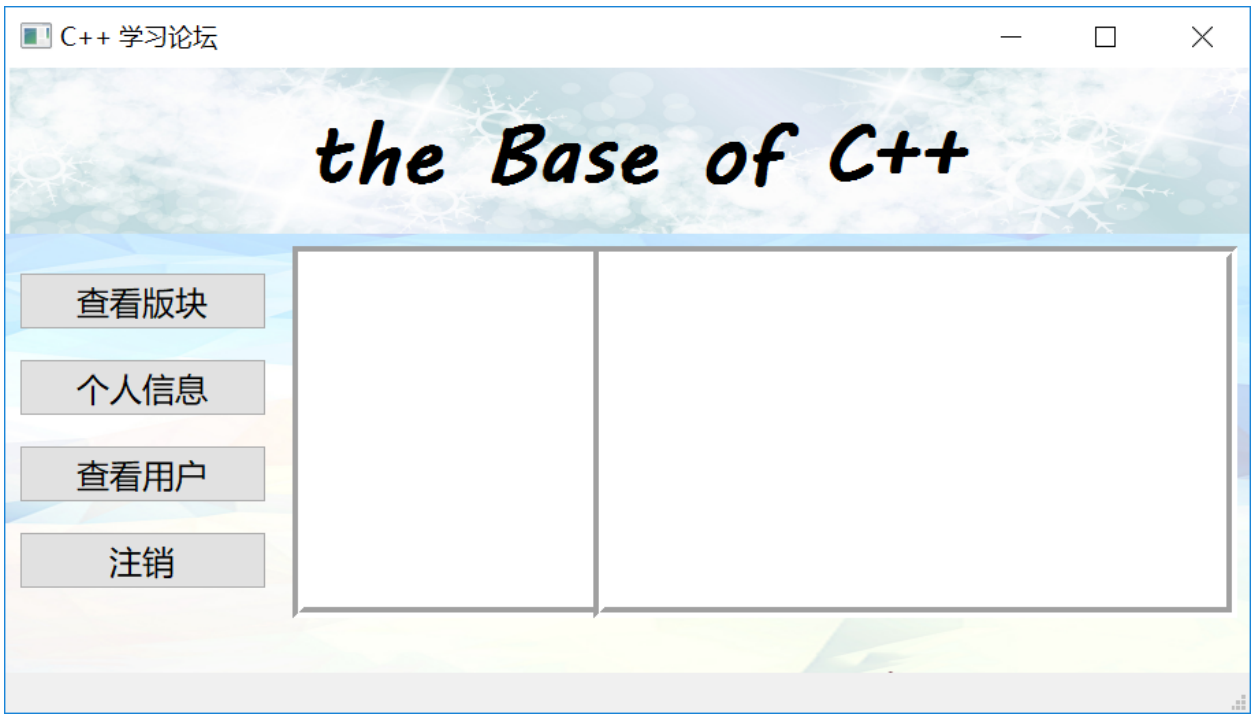
- 若用户名已被注册



- 若两次输入的密码不一样



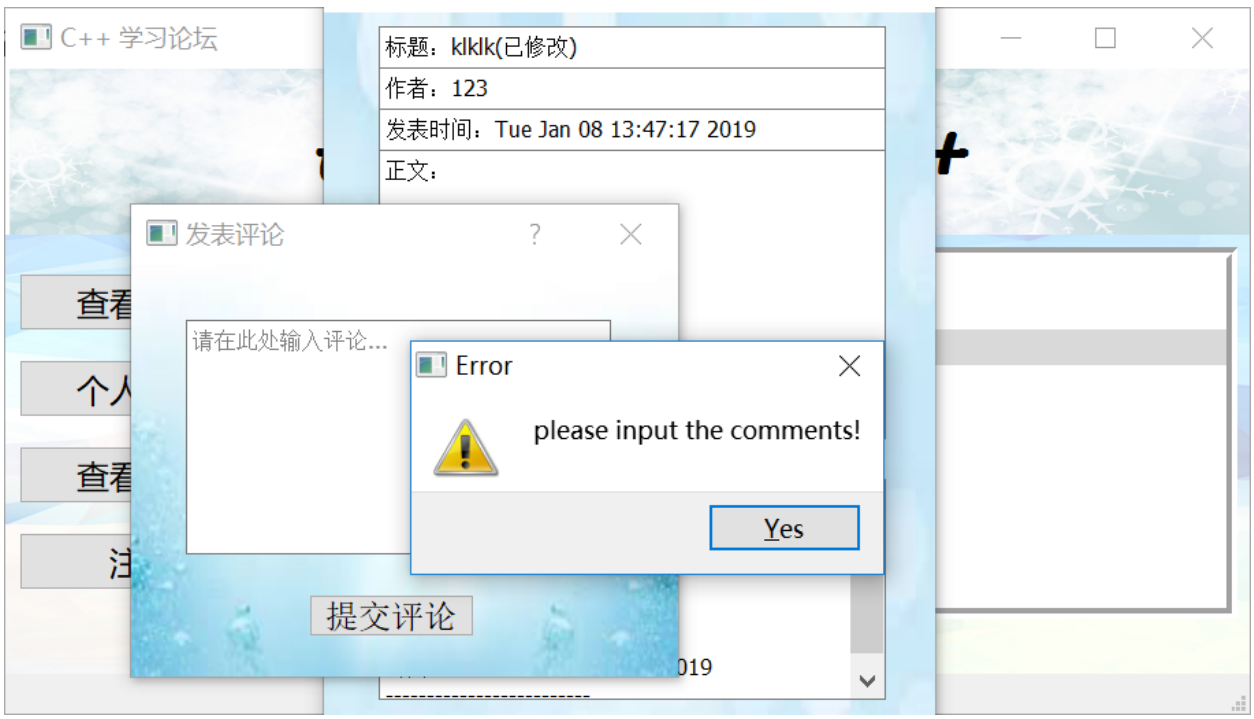
- 若成功登陆，则显示主界面



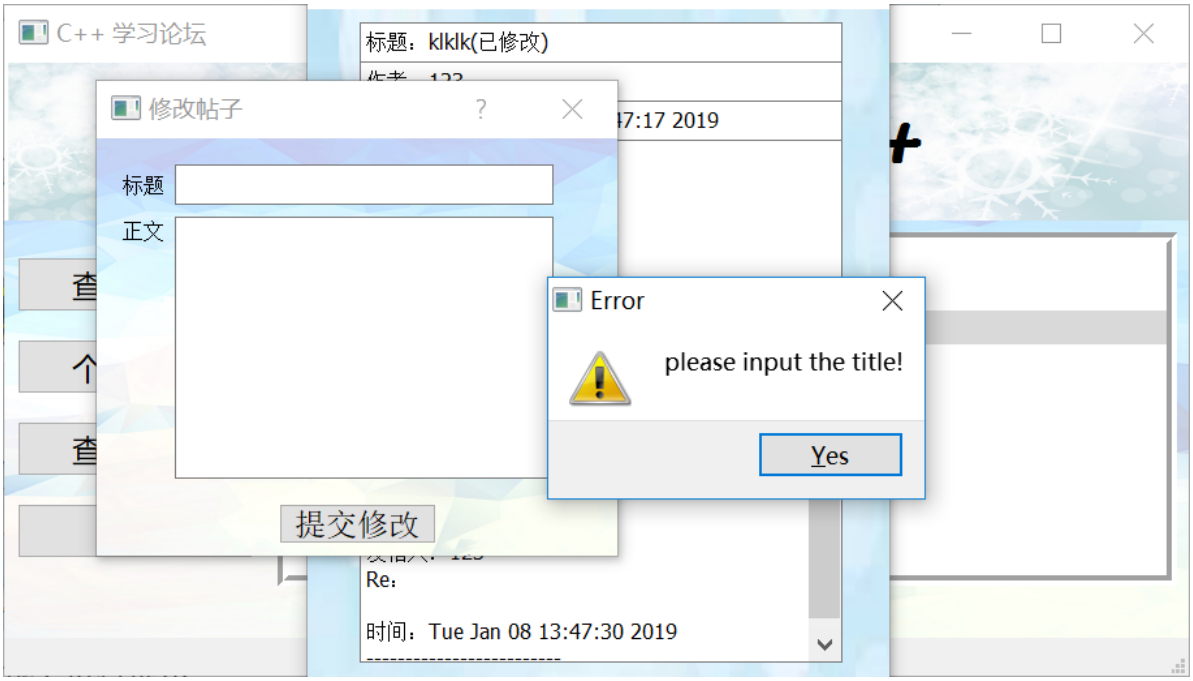
6.2 看帖、发帖、发评论

- 单击查看版块可以查看版块，双击板块名可以显示板块下所有帖子，双击帖子名可以显示帖子详细内容

- 对于自己发的帖子可以添加评论和修改帖子，若自己的帖子尚且没有评论则可以删除
  - 若评论为空会进行提示

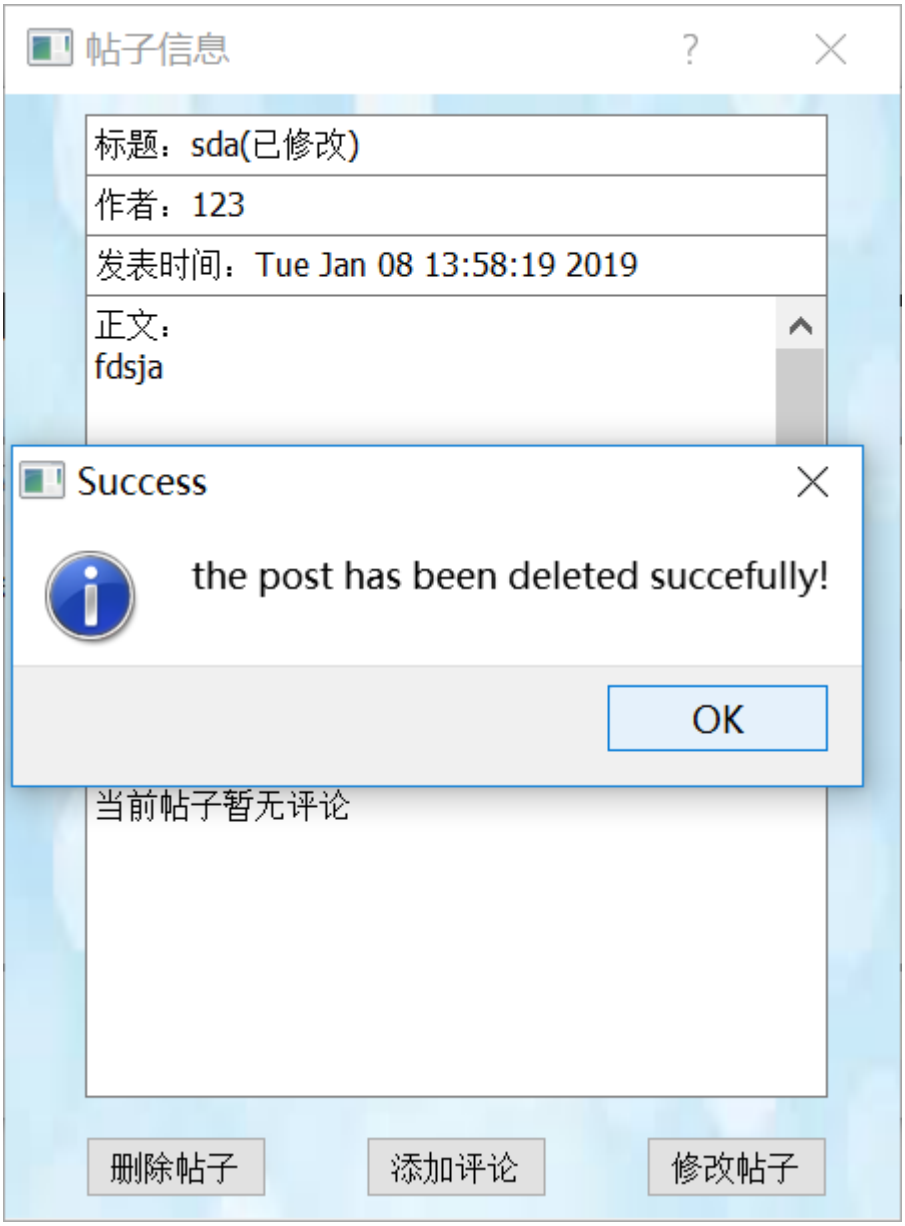


  - 若修改帖子之后的标题或者内容为空也会进行提示



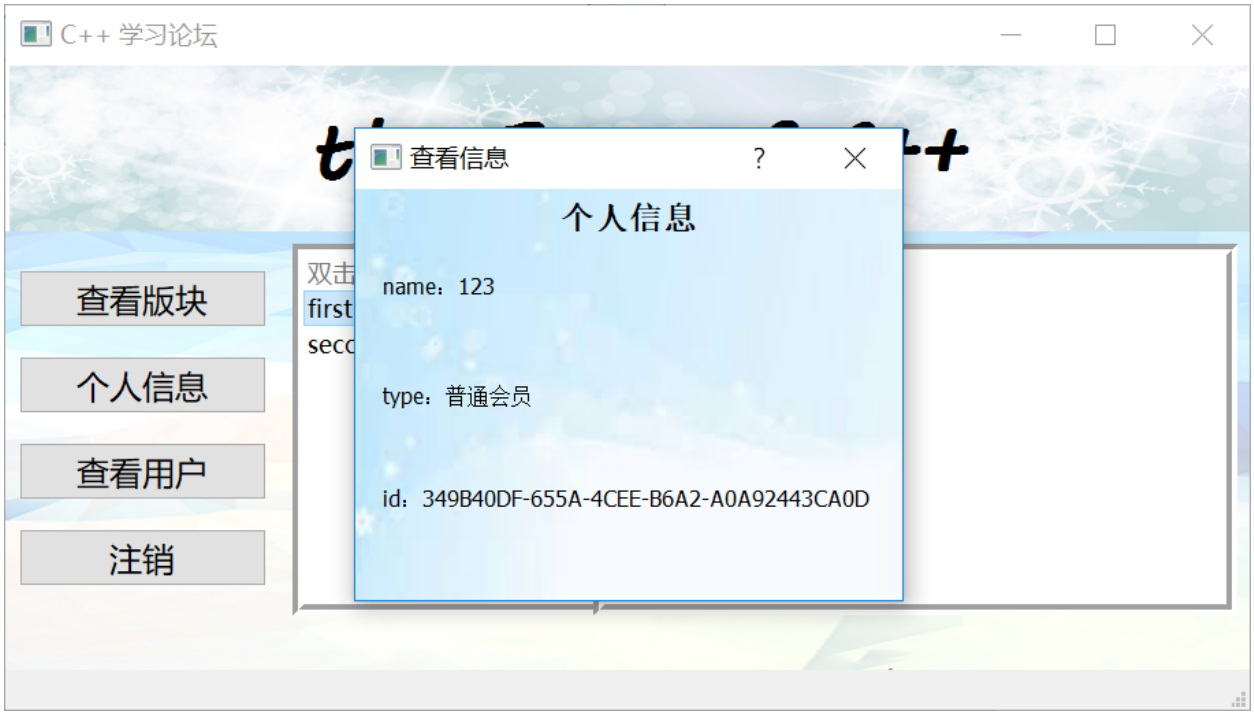
6.3 删帖

- 管理员具有最高权限，可以删除所有版块的帖子
- 版主具有删除所管理版块的所有帖子
- 普通用户可以删除自己发布的且尚未有评论的帖子
- 删帖成功后会有提示信息



6.4 查看个人信息及用户信息

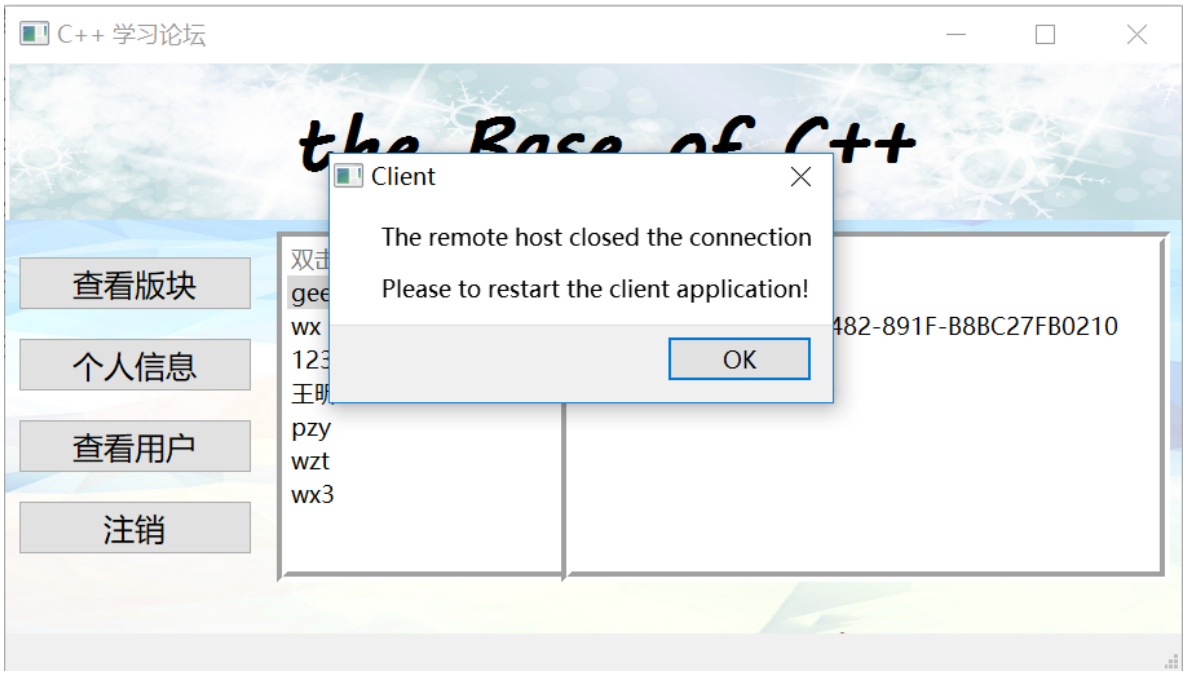
- 单击个人信息按钮可弹窗查看个人信息



- 点击查看用户，可以查看所有系统中注册的用户

6.5 多用户同步与网络通信

> - 当服务器端因为意外关闭时，客户端及时检测并退出



- 除一般意义上的同步信息，本系统针对几种极端情况进行了同步设计，分述如下
  - 当普通用户准备删除自己发布的无评论帖子时，其他用户在此时进行了评论
  - 当用户准备发布评论时对应的帖子已经被删除
  - 当版主准备删除所管辖版块下的帖子时版主身份被管理员解除
  - 当双击帖子准备查看具体内容时帖子已经被删除

7. 实验心得

这次的C++论坛设计可以说是大学三年以来个人完成的最大工程了，从第一版的简单雏形一步步完善到最终版添加网络功能后的煞有其事，断断续续也有大半个学期了，说实话确实感慨良多。一下分几个部分谈谈我的心得与体会。

- 对于程序可维护性的理解：

  本次作业老师要求按三个版本依次提交，且对一些类的实现方法做了限制，如各类用户要求以继承的方式实现。说实话，开始的时候我是不以为然的，甚至觉得这样的实现没有什么很大的意义。但是当第二版加入匿名用户的时候这样的设计就显现出优点了：只需要在新设计的匿名用户类中重写几个函数，其余的部分都不用动，极大的节省了开发的难度。

  相比较于部分同学的每一版代码都需要重构许多函数，我的开发旅程显得就顺利多了。基本上从第一版定下基本框架后，我的后两版都是在维护和拓展，极少有推翻重来的，我觉得这样归功于动手之前充分的考虑。在动手写代码之前我花了近两天的时间去构思如何设计类与实现，而不是盲目的上手去写代码。遇到不明白的地方就先去咨询了开发能力较强的同学，直到把整个论坛如何实现的大概框架弄得差不多了才开始动手。看似之前构思浪费了很多时间，但是却极大的提高了我的工作效

率，大概一个周末第一版的框架就已经出来了。这一点也体现在后两版的开发中，整个过程中我最多的时间是画在了学习各种设计模式、各种QT类的封装及TCP/IP协议，一旦学会了开始动手写就会非常迅速。

可以说，这次大作业虽然算不上什么很复杂的程序，但是提前对项目进行架构分析不盲目动手这一点我有了深刻的体会。

- 学会了许多新知识

  完成这次的C++大作业是对所学C++语法、面向对象思想的一次集中检验。虚函数、类的封装、STL这些课堂上学过的知识都有了用武之地。除此之外，还学到了许多其他知识：

  - 设计模式，这次大作业是我第一次有意识的使用设计模式。单例模式、组合模式的运用使我的开发变得简单了许多。同时也使得整个程序结构清晰很有条理。
  - TCP/IP协议，在计算机网络中曾经有过学习但是理解层面仅限于做题，如今终于上手做了。至今还记得第一次传输成功收到服务器响应时的激动。这里就必须要提到QT的强大与示例程序的重要性。在动手做第三版前我曾画了一天研究C++里如何实现TCP/IP协议，但是一直用的不是很流畅且较为复杂。偶然间知道QT封装实现了QTCP/IP Socket，但是又该怎么使用呢？以前我总是不屑于看文档，总觉得许多官方文档又是英文又讲的过于复杂，不如网上搜的博客讲的形象。但是这次许多博客上写的反而异常复杂，且由于版本不同许多函数我根本用不了。无奈之下，我只得强迫自己去看QT英文的官方文档，竟然发现上面的例子异常清晰与简单。后来经过同学提醒，我才知道QT Creator有许多示例程序，其中对阻塞式的服务器和客户端该如何设计都有demo。我如获至宝，仔细阅读后发现QT通过信号与槽机制使整个使用变得异常简单，寥寥几个函数就可以解决。再运用到我的程序上，很快就解决了网络通信的问题。不得不感叹QT封装之强悍。
  - 多线程编程

    在大一做大作业时曾经简单的接触过多线程编程，这次算是有了更深的了解。从一般的互斥访问要加锁到单例模式下getInstance函数对多线程要做多层限制，且还要防止编译器优化导致reorder以至于程序异常。我学习了许多与内存、编译器、C++11新特性的知识。对现在操作系统上所讲的互斥与并发也有了更深的理解。

- 遗憾与改进

  这次大作业也留下了一些遗憾以及值得改进的地方，分述如下：

> 第一点是第一版到第二版的迭代中，我认为将所有的用户和所有的帖子、评论都一下子存在内存不太合适，尤其是全部的用户信息都在内存中显得很奇怪，而且当数据量大了以后这样的处理太耗费资源且没有什么实际意义。所以我决定在第二版中把绝大多数放在容器中的数据移到了数据库中，这样的话显得更加合理且为第三版网络通信和多用户同步奠定了基础。
>
> 第二点也是第三版改为客户端/服务器模型之后，原则上讲客户端应该只负责前端界面，而所有的运算与判断应该都是在服务器完成的。我也是尽力这么做的，但是由于之前的设计类与类之间还是有耦合，即使迁移了一部分的判断与运算到服务器端，许多限制和判断还是在客户端完成的。所以在最后的代码结构里客户端显得比较臃肿而服务器端反而文件较少，而这种结构也导致第三版多用户同步时的许多问题判断起来较复杂，花了我许多时间。在以后的学习与开发中应该提前意识到这点，避免出现这样的瑕疵。

8. 附录(文件清单)

以下为所提交文件列表清单，具体内容见源代码

- Client端

```
//源文件
SOURCES += main.cpp\
        mainwindow.cpp \
    user.cpp \
    member.cpp \
    moderator.cpp \
    administrator.cpp \
    comment.cpp \
    post.cpp \
    usermanager.cpp \
    board.cpp \
    guid.cpp \
    forum.cpp \
    logindlg.cpp \
    infodlg.cpp \
    postdlg.cpp \
    editdlg.cpp \
    addcommentdlg.cpp \
    modifypostdlg.cpp \
    chooseboarddlg.cpp \
    registerdlg.cpp \
    anonymoususer.cpp \
```

```
    Socket.cpp
```

- ```
  //头文件
  HEADERS  += mainwindow.h \
      user.h \
      member.h \
      moderator.h \
      administrator.h \
      comment.h \
      post.h \
      usermanager.h \
      board.h \
      guid.h \
      forum.h \
      logindlg.h \
      infodlg.h \
      postdlg.h \
      editdlg.h \
      addcommentdlg.h \
      modifypostdlg.h \
      chooseboarddlg.h \
      registerdlg.h \
      anonymoususer.h \
      Socket.h
  ```

- ```
  //资源文件
  FORMS    += mainwindow.ui \
      logindlg.ui \
      infodlg.ui \
      postdlg.ui \
      editdlg.ui \
      addcommentdlg.ui \
      modifypostdlg.ui \
      chooseboarddlg.ui \
      registerdlg.ui

  RESOURCES += \
      src.qrc
  ```

- 版块名列表 => $forum.txt$

- Server端

- ```
  //源文件
  SOURCES += main.cpp \
      myserver.cpp \
      mythread.cpp \
      mysql.cpp
  ```
- ```
  //头文件
  HEADERS += \
      myserver.h \
      mythread.h \
      mysql.h
  ```
- 数据库：$mysql.db$