基于QT Widget Application 实现的Agenda GUI

2017软件工程初级实训第三阶段扩展功能技术及测试报告

姓名：吴宇祺 学号：16340242 班级：教务3班

一、UI类设计：

以MainWindow为基础
1. 将*AgendaService*的对象作为私有数据成员，实现UI层和业务层的信息交换；
2. 用信号和槽实现和 *Welcome dialog*，*LoginDialog*，*RegisterDialog* 的信息交换；
3. 将指向 *CreateMeetingDialog*， *ADDialog*，*QTimeDialog* 的指针作为私有数据成员，用信号与槽实现部分信息交换；
4. 使用标准QT库中的 *QInputDialog*， *QMessageBox* 来实现部分简单的交互操作。

二、函数（信号和槽）设计：

将对于以上四种基于信息交换分类的设计进行举例说明。

1. 与业务层
AgendaService类作为MainWindow的私有数据成员，直接与MainWindow沟通。
举例：

```
//in MainWindow.cpp
void MainWindow::listAllMeetings(void)
{
    QString title = "Your ( " + QString::fromStdString(userName)
    printMeetings(myService.listAllMeetings(userName), title);
}
```

2. 与不直接链接的类
welcome dialog中发射代表选择“Login，Register，Quit”的信号，MainWindow中的receiveCommand()槽接受信号。由于两个类的对象都在主函数中定义，所以链接放在主函数中。

```
private slots:
    //in MainWindow
    //receive command from welcome dialog
    void receiveCommand(const int& command);
```

```
signals:
    //in WelcomeDialog
    //emit command to main window
    void command(const int& command);
```

```
//in main.cpp
QObject::connect(&myWelcome, SIGNAL(command(int)), &w, SLOT(receiveCommand(int)));
```

附上receiveConmand() 的实现

```
//in MainWindow.cpp
void MainWindow::receiveCommand(const int& command)
{
    switch(command)
    {
    case(1):
        qDebug() << "main window : Login exeeced\n";
        emit showLogin();
        break;
    case(2):
        qDebug() << "main window : register exeeced\n";
        emit showRegister();
        break;
    default:
        qDebug() << "mainwindow : emit Quit system";
        myService.quitAgenda();
        emit quitSystem();
        break;
    }
}
```

3. 以其指针作为私有成员的类

MainWindow既在成员函数内部控制指针，又有接受这些类发射的携带（键盘鼠标）输入的事件信息的信号。

举例：*CreateMeeting*

```
signals:
    //in CreateMeeting.cpp
    void meetingInfo(const string, const string, const string, const vector<string> row);
```

```
private slots:
    //in MainWindow.hpp
    //verify CM
    void verifyCreateMeeting(const string _title, const string _startDate,
                             const string _endDate, const vector<string> pa);
```

```
 //in MainWindow.cpp
 //function : createMeeting()
 QObject::connect( &(*myCM), SIGNAL(meetingInfo(string,string,string,vector<string>)),
                   &(*this), SLOT(verifyCreateMeeting(string,string,string,vector<string>)));
```

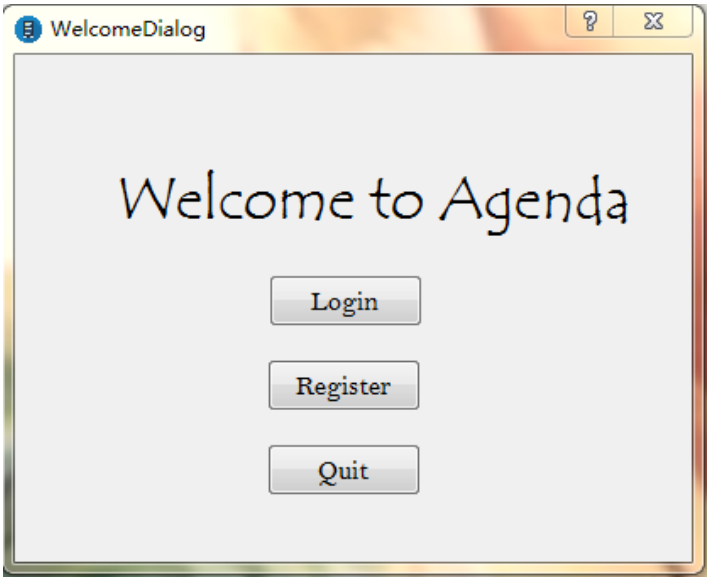
```
//in CreateMeeting.cpp
//function : on_comfirmButton_clicked()
    emit meetingInfo(_title, startDate.toStdString(), endDate.toStdString(), pa);
```

4. 使用标准QT库内QDialog的派生类实现的交互

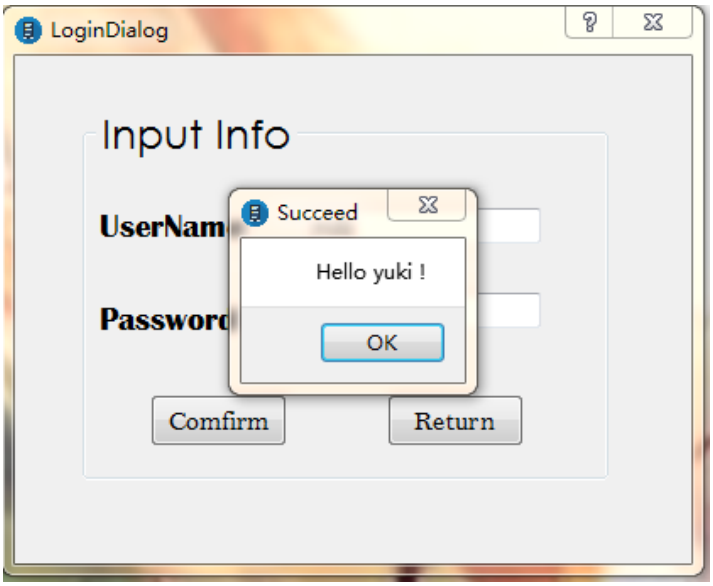
常规操作

三、测试结果

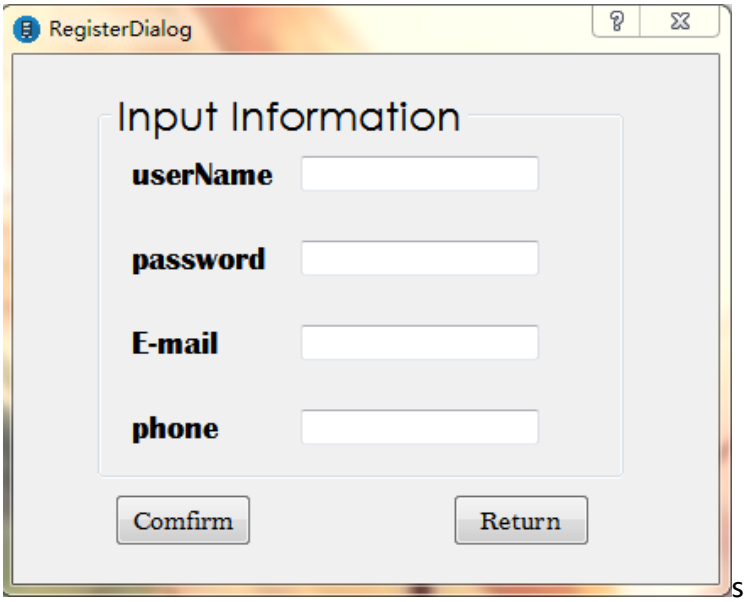
Welcome Page



Login Page



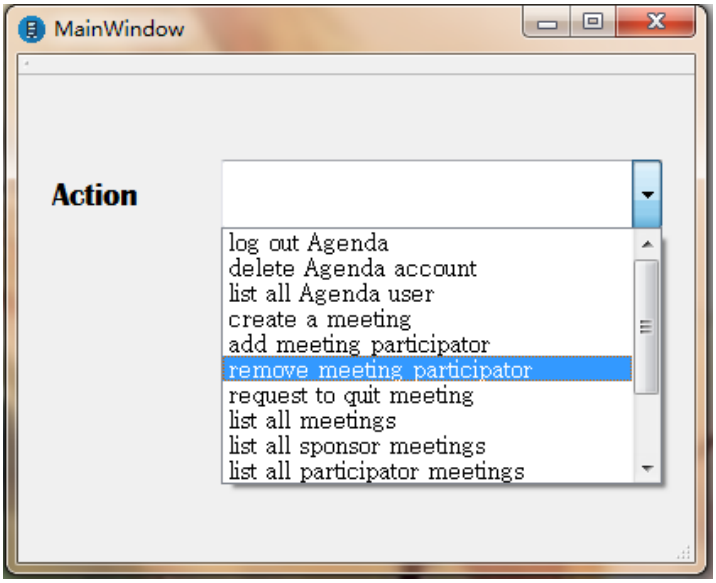
Register Page



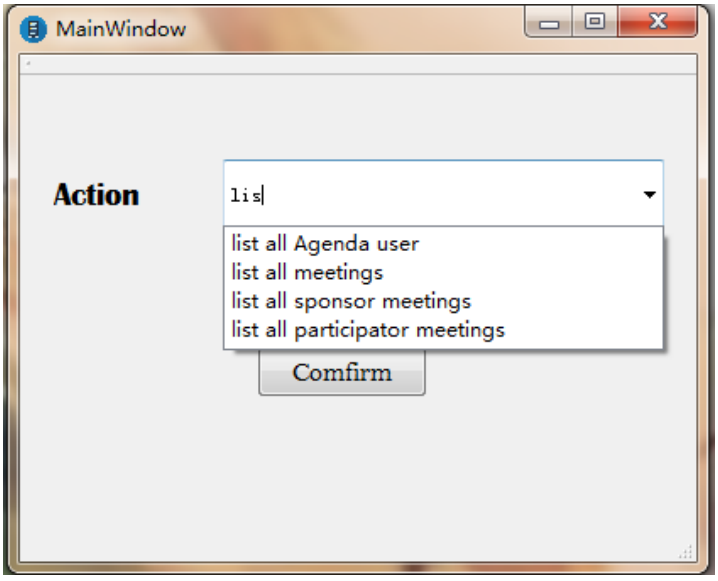
s

Menu Page

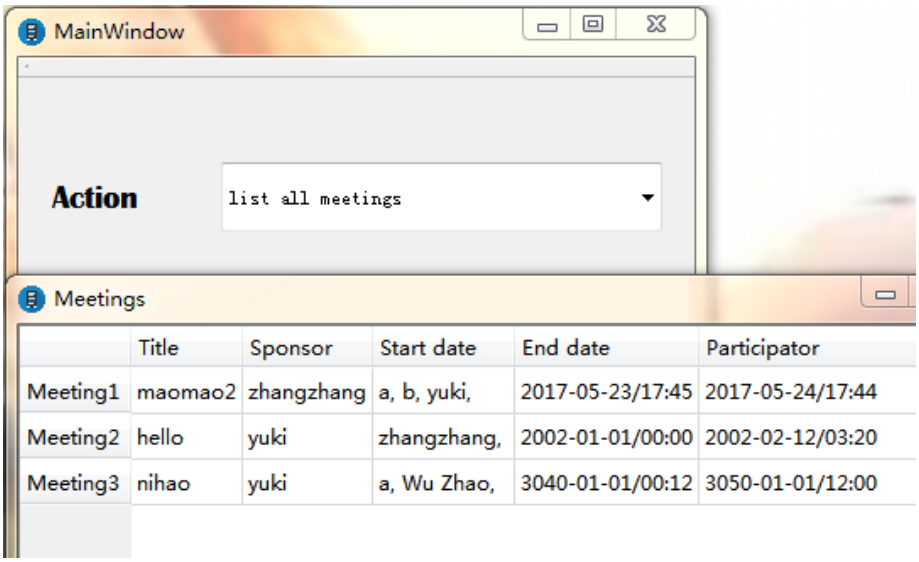
drop-down menu



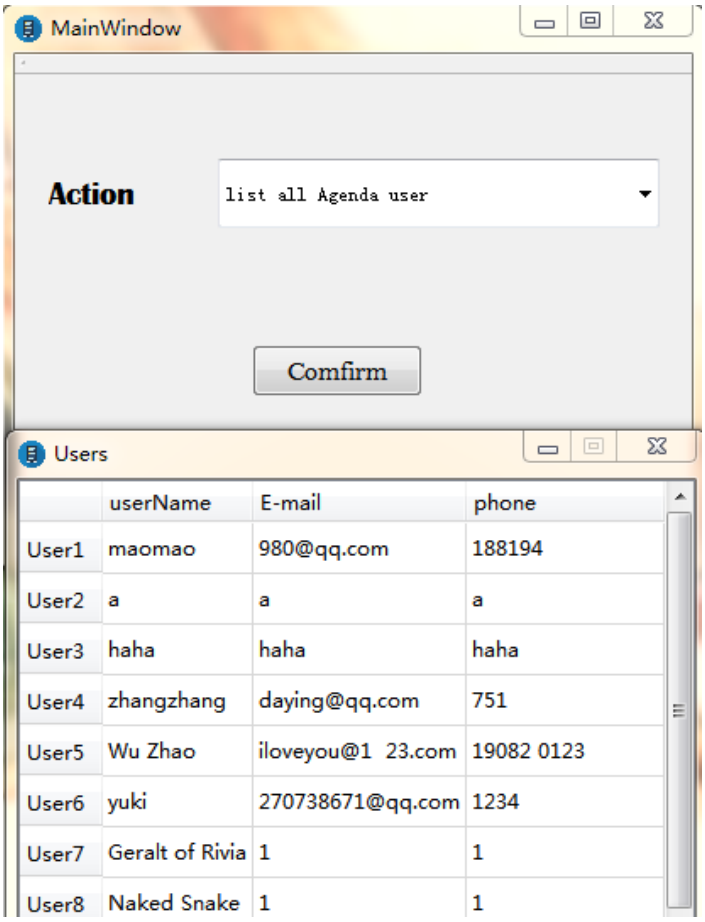
Auto-Complete



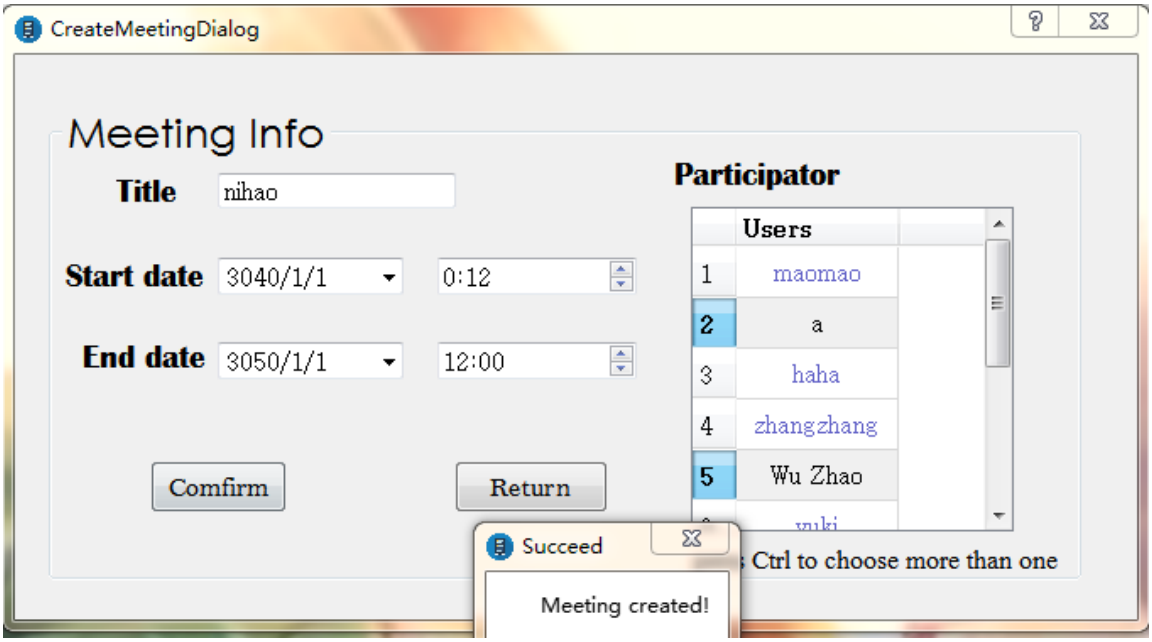
List All Meetings



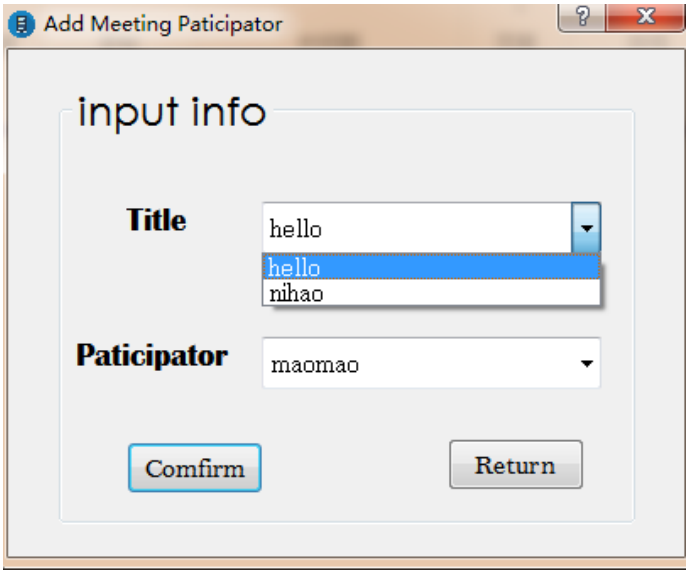
List All Users



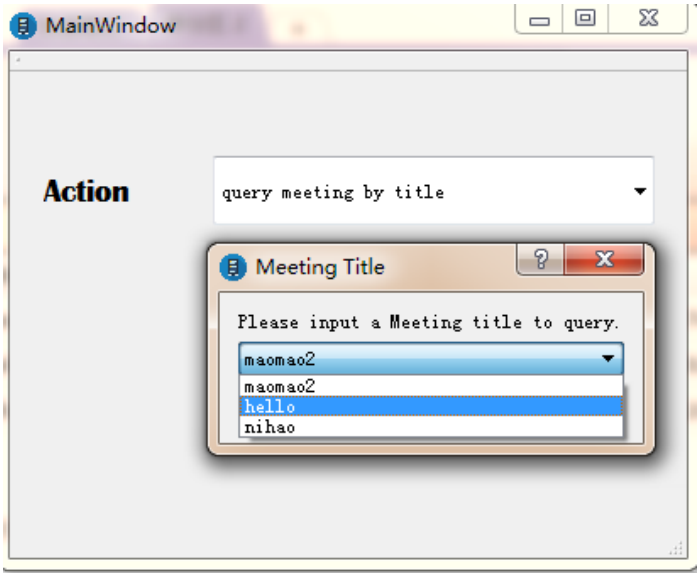
Create Meeting Page



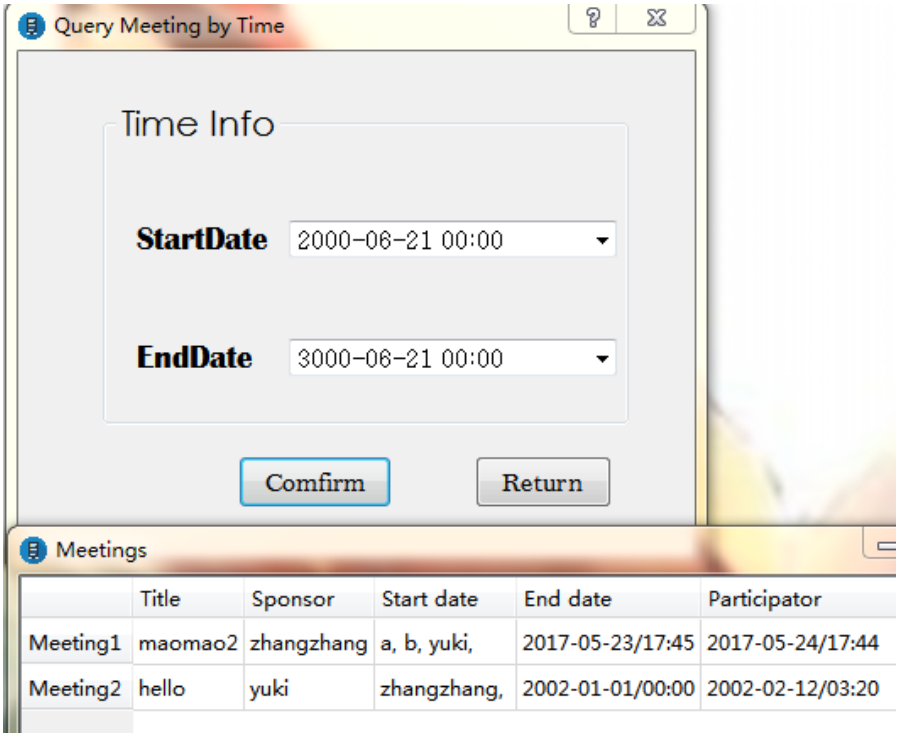
Add Meeting Participator Page



Query Meeting by Title



Query Meeting by Time



四、结语 一些杂七杂八的经验

- 多亏了三层架构的解耦，使得UI开发可以独立于逻辑和数据进行，使维护也变得非常简单。
- 虽然QT（版本5.7.0）具有特殊的QString等类来支持UI界面设计，但提供了丰富的库函数来快速转换标准库和QT库的对象或者函数，除此之外对C++11特性也有良好支持。
- 故Storage（数据层）和Service（业务层）没有做数据方面的任何修改，就当成是提供接口的库函数来运用在GUI设计的过程中。
- 这一次因为是三天（周一到周三）速成的UI开发，时间精力主要集中在思考类之间的关系和设计并实现基本交互上，并没有过多地探究UI界面设计的精妙之处。
- QT的模板库真是精彩纷呈到令人发指...希望有时间能够好好了解下。这次未能实现理由同上。

五、未结束的问题

每当使用信号的发射和槽的接受时想到，本来发射信号的函数仍在运行，信号又触发了对应槽函数的运行，这和平时“流水线”一样的程序运行完全不一样，觉得非常难以控制。
也许这就要求我需要对每一个对象的声明周期和每一个函数的调用和退出非常了解。