实验：学生信息管理系统

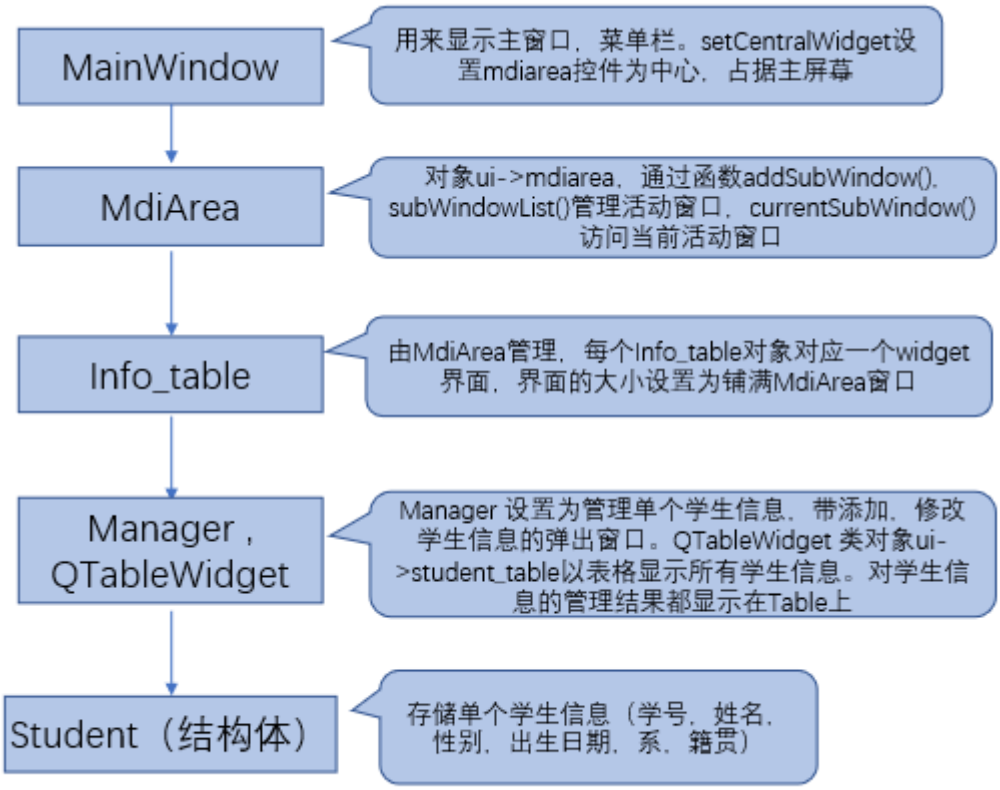
161220135-吴德亚

一：实验介绍

本次实验使用Qt5 完成。 支持文件的新建，保存，打开，关闭，多窗口界面，表格内容的增添，删除，修改（双击），排序。支持多选删除，另外增加功能： 查询（可以按照输入字串查询）。

二：类视图

在本次实验中，使用类 **MainWindow** , **Info_Table** , **Manager**, 结构体 **student** ,它们之间的关系及主要功能如下:



三：类的成员函数介绍:

***1. MainWindow:* 类**

```
    void set_current_table();
    // 通过ui->mdiarea的成员函数currentSubWindow()得到当前活动窗口
    void closeEvent(QCloseEvent *event);
    //重写closeEvent 函数

private slots:
    // 菜单栏点击触发槽函数
    void on_actionNew_triggered();   // 新建
    void on_actionSave_triggered();  // 保存
```

```
    void on_actionOpen_triggered();  // 打开
    void on_actionAdd_triggered();   // 添加
    void on_actionModify_triggered();  // 修改
    void on_actionDelete_triggered();  // 删除
    void on_actionID_triggered();      // 按id 排序
    void on_actionName_triggered();   // 按姓名 排序
    void on_actionclose_triggered();   // 关闭
    void on_actionDate_triggered();    // 按日期排序
    void on_action_triggered();         // 查找

private:
    Ui::MainWindow *ui;
    int label;                      //窗口ID
    Info_Table *current_table;       // 指向当前窗口对应Info_Table对象指针
```

（1） 由于所有的窗口都通过ui->mdiarea的addSubWindow()函数添加，故同样可以通过SubWindowList()函数得到所有窗口的QList，保持current_table指向的是当前窗口所对应的Info_Table对象

```
QMdiSubWindow *cur = ui->mdiArea->currentSubWindow();
current_table = (Info_Table *)cur->widget();
```

（2） 重写closeEvent函数可以让关闭窗口时 “让用户判断是否保存文件“，在Info_Table类中也重写了该函数。

```
if(!ui->mdiArea->subWindowList().empty())   // 是否还有子窗口
    {
        QMessageBox::StandardButton button;
        button=QMessageBox::question(this,"Warning","这不会保存你的文件，是否关
闭? ",QMessageBox::Yes|QMessageBox::No);
        if(button == QMessageBox::Yes)
            event->accept();
        else
            event->ignore();     // 忽略关闭事件
    }
    else
        this->close();
  }
```

2. *Info_Table* 类

```
    void closeEvent(QCloseEvent *event);  // 重写关闭事件，判断是否保存文件
    student *get_current_stu();          // 得到当前使用的学生信息
    void show_table();                  // 绘制表格
    void update_table();                // 更新表格
    void new_table();
    void save_table();
    void open_table();
    void add_table();
    void modify_table();
    void delete_table();
    void sort_by_id();
```

```
    void sort_by_date();
    void sort_by_name();
    void find_table(){emit_find_table();}   // 调用查找接
private slots:
// 槽函数
    void emit_delete_table(){delete_table();}
    void emit_find_table();
    void get_find_table();        // 文本框enter时触发
    void receive_add_emit(student *stu);
    void receive_modify_finish();
    void set_doubleclick_table(QTableWidgetItem *item);   // 双击修改
    void set_click_table(QTableWidgetItem *item);
    void click_menu(QPoint pos);// 添加菜单栏

signals:
     void emit_modify_start();
private:
    Manager *manager;               // 修改时指向需要修改的对象
    vector<student *>info_stu;      // 指向结构体指针的vector结构
    student *current_stu;           // 当前指向学生
    student sample;
    QList<QTableWidgetItem *> item_list,find_list; // 选中QTableWidget对象的items
    QLineEdit *get_find;         // 文本框
```

（1） 初始化QTableWidget对象时需要设定属性 **NoEditTriger(阻止直接修改)，SelectRows（选取整行）**。在保存和打开文件函数中调用了 **QFileDiaglog** 类，之后再路径的基础上实现文件的读取，存放操作。

```
QString str = filelog.getSaveFileName(this,tr("保存文件"),"student.txt"
,tr("File(*.txt"));   // 从弹出对话框选择相对路径
```

（2） 所有针对学生信息管理的操作都是先在 info_stu 上进行操作，然后再通过update_table()函数刷新table。在排序（id, name, birth_day) 时使用 sort 函数，自定义camp。例如：

```
 sort(info_stu.begin(),info_stu.end(),camp_date);
```

（3） 自定义右键生成菜单“ 删除，查找 ”，在删除时支持多选删除，查找时支持文本框输入查找包含该字串的Cell。

```
    connect(m_delete,SIGNAL(triggered()),this,SLOT(emit_delete_table())); // 由菜单触发
    connect(m_find,SIGNAL(triggered()),this,SLOT(emit_find_table()));
```

查找时对 QLineEdit 对象输入字串, 通过text()函数获取

```
find_list = ui->student_table->findItems(text,Qt::MatchContains);
// 从表格中查找包含text字串的items, 返回赋值给 QList 类型的find_list
```

3. Manager 类

```
private slots:
    void on_add_btn_clicked();
    void on_cancle_btn_clicked();
    void receive_modify_start();
    void get_value();           // 从界面的文本框中获取值
 signals:
    void add_info_emit(student *stu);     // 函数传值
    void emit_modify_finish();
    void cancle_info_emit(student *stu);
public:
    Ui::Manager *ui;
    student *stu_once;
    bool add_flag,modify_flag; // 标志当前处于增添，修改状态
```

（1） 由于增添和修改是使用同一个Manager 类的ui文件，所以需要标志当前是处于哪个状态。对于增添来说，只需要创建新对象然后添加指针到 info_stu 数组中即可，对于修改则先显示原始信息，然后交给用户，让用户选择修改哪部分信息，最后再将该结果更新到 info_stu中。

```
  ui->comboBox_sex->setCurrentText(QString::fromStdString(stu_once->s_sex));
  // 先显示原始信息
  ……
  QString sex = ui->comboBox_sex->currentText();  // 再读取用户修改信息
  stu_once->s_sex =sex.toStdString();
```

（2） 由于点击 ”录入“按钮后，需要提交信息，故通过槽函数，信号函数来传递参数。

```
  emit add_info_emit(stu_once);     // 返回新建对象的指针
```

（3）为了存取文件方便性，需要给student 每个成员赋值

```
   if(stu_once->s_id=="" || stu_once->s_name=="" || stu_once->s_dept=="" || stu_once-
>s_address=="")
    {
        QMessageBox::warning(this,tr("Warning"),tr("部分表单为空!"));
    } // 增加判断，只有当所有文本框都填写完毕才能录入
```

（4） 双击修改每行的值：将自定义SLOT函数set_doubleclick_table 和QTableWidget 成员SIGNAL 函数 itemDoubleClicked 连接；

4. 健壮性

本次实验多次使用QMessageBox，条件判断等，增强用户体验。

```
QMessageBox::warning(this,tr("Warning"),tr("文件夹为空! "));
QMessageBox::warning(this,"Warning","该同学已经存在!");
QMessageBox::warning(this,"warning","请选择对应行!");
QMessageBox::warning(this,tr("Warning"),"选中区域为空! ");
……
```

四：实验过程中遇到的问题

1. 在实现添加，修改函数的过程中，发现无法在Info_table类中直接获取返回的student *的值。后来的解决办法是添加槽函数，通过SIGNAL, SLOT函数传参。
2. 之前并没有实现多窗口管理的功能，只是单个窗口。使用了MdiArea类，通过查询Qt手册，调用相应的函数最后实现了多窗口切换，管理的功能。
3. QInputDialog 类的使用，之前使用的是标准文本输入框QInputDialog::getText(), 但是总是报错 “ qinputdialog setgeometry: unable to set geometry .....” ,而且一直没有找到解决办法，所以就修改成了QLineEdit 类，通过文本框获取输入的值。

五：参考文献及致谢

https://doc.qt.io/qt-5/qtablewidget.html // QTableWidget Class 官方手册

https://doc.qt.io/qt-5/qmdiarea.html // QMdiArea Class 官方手册

https://blog.csdn.net/u010780613/article/details/50442639 // Qt 之QTableWidget 详细使用

https://www.cnblogs.com/Jace-Lee/p/6055078.html // Qt - - QMdiArea 和 QMdiSubWindow使用

https://blog.csdn.net/swartz_lubel/article/details/54985130 // QMenu 及QAction的使用

.....