面向对象程序设计第三次作业

——高级设计意图分析与总结

宋宇轩 2019K8009929042

写在前面：

这是第三次作业的报告文档，主要介绍了我的项目的最终实现情况、运用了哪些面向对象的特性和方法，以及对整个项目的总结和展望。

一. 程序功能补全与完善

最终实现的程序外观与先前预想的程序界面一致，此次主要是完成了先前还未实现的功能，包含主界面的删除文档和搜索文档功能，以及子界面下拉菜单中的另存为、查找和替换功能。其中主界面的搜索功能和子界面的查找功能共用一个 ui 界面，子界面的替换功能也设计了另一个 ui 界面。两个 ui 界面的示意图如下所示：

图 1 搜索界面示意图

图 2 替换界面示意图

除此之外，我还对原先的代码做了部分修改和重构，以提高整体的性能、稳定性和可扩展性。至此，这个项目的所有基本需求都已实现完毕，后续的一些设想详见**四. 总结与展望**部分。

二. 代码分析与功能实现

项目最终的 UML 类图如下所示，相较之前主要增加了用于控制搜索/查找功能的 SrchWindow 类、MainSrchWindow 类和 EditSrchWindow 类，以及用于控制替换功能的 RepWindow 类。此外其余类中的成员函数和成员变量也有一定的增减。

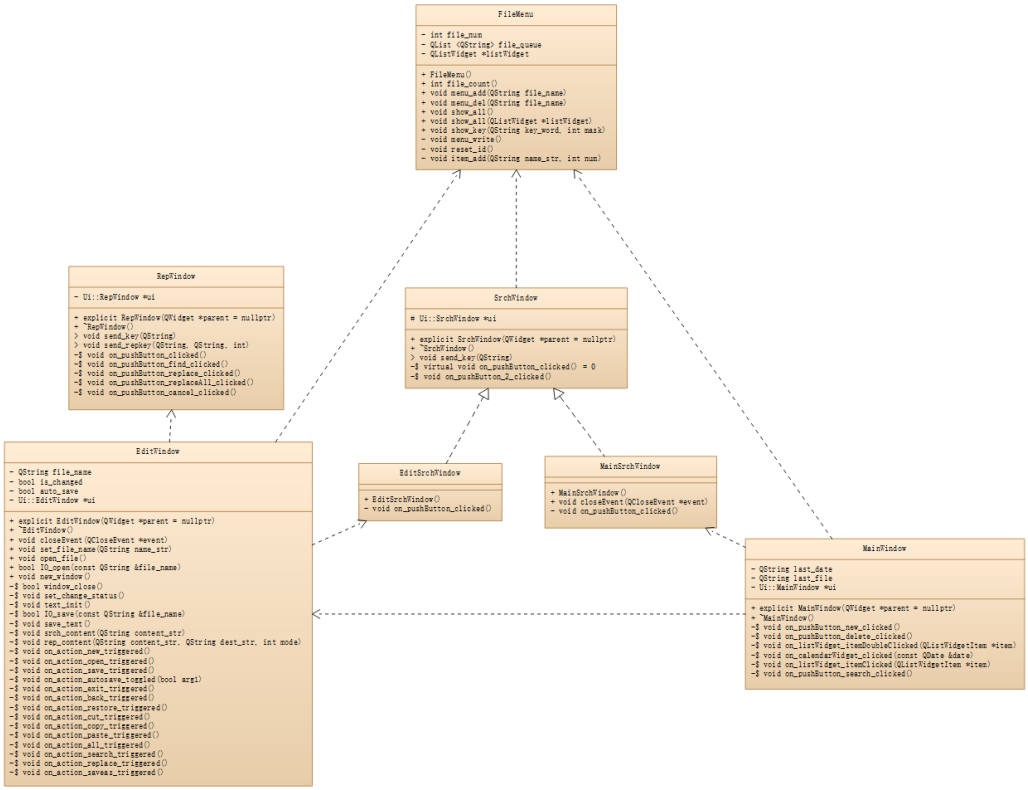

图 3 项目总体 UML 类图

主界面的删除文档功能需要先单击选中显示区中的一项文档，随后再单击删除按键即可删除选中文档。在单击“删除”按键后，主界面会弹出窗口请求确认删除行为，待用户确认后调用 FileMenu 类中的 menu_del 函数，将对应文档从文件目录中删去、重命名文档以标记已删除，并修改所有日记文档内的编号。各部分对应的具体代码如下：

```
void MainWindow::on_pushButton_delete_clicked()
{
    if(last_file == "Not a file")
    {
        QMessageBox::warning(NULL,"警告","请先选择一个文件！");
    }
    else
    {
        QString file_name = last_file.right(19);
        file_name.replace(4, 1, "_");
        file_name.replace(7, 1, "_");
        file_name.replace(10, 1, "_");
```

```
        file_name.replace(13, 1, "_");
        file_name.replace(16, 1, "_");
        file_name += ".txt";

        QMessageBox box;
        box.setWindowTitle("提示");
        box.setText("确定删除文件 \""+file_name+"\" 吗？");
        box.setStandardButtons(QMessageBox::Yes | QMessageBox::Cancel);
        int ret = box.exec();
        if(ret == QMessageBox::Yes)
        {
            //删除对应文件
            menu->menu_del(file_name);
            last_file = "Not a file";
        }
        else
        {
            //不执行操作
        }
    }
}

void FileMenu::menu_write()
{
    QFile fp("./diary_files/menu.txt");
    if(fp.open(QIODevice::WriteOnly | QIODevice::Text))
    {
        QTextStream stream_out(&fp);
        stream_out<<this->file_num<<endl;
        for(int i=0;i<this->file_num;i++)
        {
            QString str = this->file_queue[i];
            stream_out<<str<<endl;
        }
        fp.close();
    }
}

void FileMenu::reset_id()
{
    for(int i = 0; i < this->file_num; i++)
    {
       QString string = this->file_queue.at(i);
       QFile fp("./diary_files/" + string);
       QString tmp = "日记编号: " + QString::number(i+1) + "\r\n";
       if(fp.open(QIODevice::ReadOnly | QIODevice::Text))
       {
           QTextStream stream_in(&fp);
           QString line;
           int line_num = 1;
           while(stream_in.readLineInto(&line))
           {
               if(line_num != 1)
               {
                   tmp += line + "\r\n";
               }
               line_num++;
```

```
            }
            fp.close();
        }
        if(fp.open(QIODevice::WriteOnly | QIODevice::Text))
        {
            QTextStream stream_out(&fp);
            stream_out<<tmp<<endl;
            fp.close();
        }
    }
}
```

目前实现的操作并不是真正的从本地删除对应文档，只是将其从文档目录中删去，并在文档后加上已删除的标注，默认不再打开该文档而已。这样做的目的是为了便于以后添加类似“回收站”的功能，可以恢复已删除的文档。

主界面的搜索功能和子界面的查找功能使用的是同一个 ui 界面，仅在按键上的文字注释处有一些不同。不过由于二者实现的功能有所不同，该项目中设计了一个基类 SrchWindow 用于生成 ui、完成部分共有的功能；而最重要的搜索功能则实现为了一个接口，由两个子类 MainSrchWindow 和 EditSrchWindow 分别实现各自所需的操作。具体代码如下：

```
class SrchWindow : public QWidget
{
    Q_OBJECT

public:
    explicit SrchWindow(QWidget *parent = nullptr);
    ~SrchWindow();

signals:
    void send_key(QString);

private slots:
    virtual void on_pushButton_clicked() = 0;

    void on_pushButton_2_clicked();

protected:
    Ui::SrchWindow *ui;
};

class MainSrchWindow : public SrchWindow
{
public:
    MainSrchWindow();
    void closeEvent(QCloseEvent *event);
private:
    void on_pushButton_clicked();
};

class EditSrchWindow : public SrchWindow
{
public:
    EditSrchWindow();
```

```
    private:
        void on_pushButton_clicked();
    };
```

其中成员函数 on_pushButton_clicked()即为按下搜索按键后会触发的操作，在基类中是一个纯虚函数，由两个子类分别实现：

```
    void MainSrchWindow::on_pushButton_clicked()
    {
        QString srch_str = ui->lineEdit->text();
        menu->show_key(srch_str, SHOW_SRCH);
    }
    void EditSrchWindow::on_pushButton_clicked()
    {
        QString srch_str = ui->lineEdit->text();
        emit send_key(srch_str);
    }
```

主界面的搜索功能会在显示区中显示包含搜索内容的文档，子界面的查找则会先将查找内容发送给 EditWindow 类，再由 srch_content 函数将当前光标位置之后的第一个查找内容选中：

```
void EditWindow::srch_content(QString content_str)
{
    if(content_str == NULL)
    {

    }
    else if(ui->textEdit->find(content_str))//如果找到了
    {
        // 查找到后高亮显示
        QPalette palette = ui->textEdit->palette();
        palette.setColor(QPalette::Highlight,
                         palette.color(QPalette::Active,QPalette::Highlight));
        ui->textEdit->setPalette(palette);
    }
    else               //如果没找到，就把光标移到开头再找（循环），如果还找不到就是没有
    {
        ui->textEdit->moveCursor(QTextCursor::Start);
        if(ui->textEdit->find(content_str))
        {
            QPalette palette = ui->textEdit->palette();
            palette.setColor(QPalette::Highlight,
                             palette.color(QPalette::Active,QPalette::Highlight));
            ui->textEdit->setPalette(palette);
        }
        else
        {
            QMessageBox::information(this,tr("注意"),
                                     tr("没有找到内容"),QMessageBox::Ok);
        }
    }
}
```

替换功能仅在子界面中出现，实现原理与查找功能类似：在单击替换按键后，将查找内

容和目标内容都发送给 EditWindow 类，由 rep_content 函数将查找内容修改为目标内容。由于包含了“替换”和“替换全部”两种功能，还需要将一个用于标识的参数 mode 也一同传给 EditWindow 类：

```
void RepWindow::on_pushButton_replace_clicked()
{
    QString srch_str = ui->srch_text->text();
    QString dest_str = ui->rep_text->text();
    emit send_repkey(srch_str, dest_str, 0);
}
void RepWindow::on_pushButton_replaceAll_clicked()
{
    QString srch_str = ui->srch_text->text();
    QString dest_str = ui->rep_text->text();
    emit send_repkey(srch_str, dest_str, 1);
}

void EditWindow::rep_content(QString content_str, QString dest_str, int mode)
{
    if(mode == 0)
    {
        //替换
        //单击替换后先检测当前光标选定的内容是不是 find 的内容
        //如果是，就直接替换，并尝试找下一个；否则类似 find 的功能，从头开始找一个
        QTextCursor now_cursor = ui->textEdit->textCursor();
        QString select_str = now_cursor.selectedText();
        if(select_str != content_str)
        {
            srch_content(content_str);
        }
        else
        {
            now_cursor.insertText(dest_str);
            srch_content(content_str);
        }
    }
    else
    {
        //替换全部
        QString tmp_str = ui->textEdit->toPlainText();
        tmp_str.replace(content_str, dest_str);
        ui->textEdit->setText(tmp_str);
    }
}
```

另存为功能与保存功能类似，通过弹出窗口选择目标路径，然后将当前文件复制一份到目标路径即可：

```
void EditWindow::on_action_saveas_triggered()
{
    QString curPath = QDir::currentPath()+"/diary_files/"+file_name;
    QString dlgTitle = "另存为";
    QString filter = "所有文件(*.*);;文本文件(*.txt)";
    QString dirPath = QFileDialog::getSaveFileName(this, dlgTitle, curPath, filter, Q_NULLPTR,
                                QFileDialog::ShowDirsOnly | QFileDialog::DontResolveSymlinks);
```

```
    //如果选择的路径无效，则不保存
    if (!dirPath.isEmpty())
    {
       QFileInfo fileInfo(dirPath);
       if(fileInfo.exists())
           QFile::remove(dirPath);
       QMessageBox::warning(NULL,"测试",curPath);
       QFile::copy(curPath, dirPath);
    }
}
```

三. 面向对象特性分析

在实现这个项目的过程中，我也用到了不少面向对象的特性和方法，这里主要从“封装”、“继承”、“多态”、“重载”和“高内聚低耦合”几部分入手，对这一项目的面向对象特性做一简单介绍和分析。

首先是封装：数据封装是面向对象最基础也是最核心的特性之一，也常常与数据隐藏和数据抽象联系在一起。它把数据和操作数据的函数捆绑在一起，通过隐藏对象实现的具体细节来减少耦合，避免受到外界的干扰和误用。这个项目实现的所有功能都是通过创建各种类及其成员来实现的，从而保证了良好的封装性。此外，每个类中的成员变量都是私有的，仅通过公有的成员函数向外提供交互的接口，从而实现数据隐藏的需求：

```
class FileMenu
{
public:
    int file_count();       //对外提供的接口
private:
    int file_num;           //对外隐藏的数据
    ...
};
int FileMenu::file_count()
{
    return file_num;
}
```

其次是继承：继承允许设计者依据一个类来定义另一个类，使得创建和维护一个应用程序变得更加容易，也达到了复用代码功能和提高执行效率的效果。不过继承是一种强耦合关系：父类的实现细节对于子类来说都是透明的，因此继承在一定程度上破坏了封装。继承的主要作用在于让若干个功能相近的类共享一般化的父类中的结构和行为，同时也扩展出独属于自己的特殊化的属性和方法。由于这个项目是基于 Qt 框架开发的，大多数类都继承自 Qt 提供的父类，从而拥有了很多父类中设计完善的结构和功能。

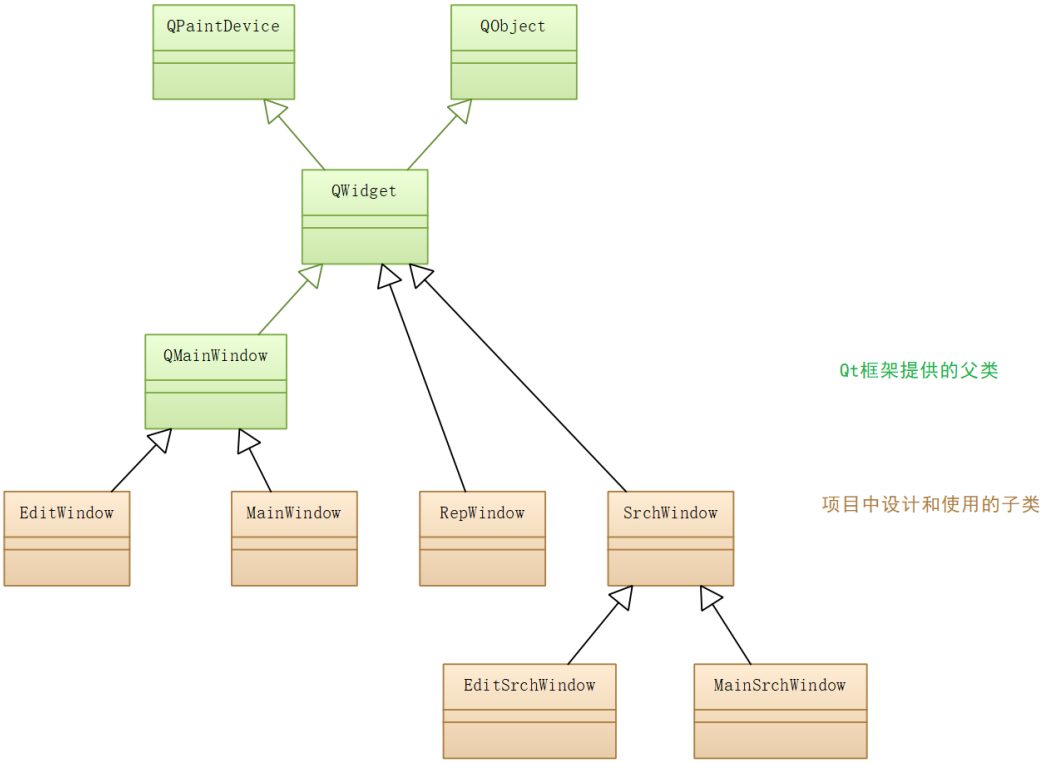

图 4 继承关系示意图

然后是多态：多态按字面的意思就是指多种形态，即一个操作可以被层次结构中的类以不同的方式实现。利用多态，子类可以扩展父类的能力，或者覆写父类的操作，根据调用函数的对象的类型来执行不同的函数。在有多种相似算法的情况下，多态可以有效避免 if-else 语句或 switch-case 语句带来的复杂和难以维护，减少设计的不稳定性。在这个项目中，主界面的搜索功能和子界面的查找功能共用同一个 ui 界面，其他很多功能也是相近的，只有核心的搜索功能根据调用来源的不同而有所区别。此时设计一个基类 SrchWindow 来控制 ui 和那些相似的功能，搜索功能则设计为接口；两个子类 EditSrchWindow 和 MainSrchWindow 分别重写搜索功能，即可根据调用函数的对象类型来选择执行相应的操作，也免去了 if-else 语句带来的种种问题。这种思想同时也属于设计模式中的策略模式：当一个系统需要动态地在几种算法中选择一种，而这些算法的区别仅在于它们的行为时，就可以将这些算法封装成单独的类并实现同一个接口，使它们可以相互替换：

```
class SrchWindow : public QWidget
{
    ...
private slots:
    virtual void on_pushButton_clicked() = 0;
};

class MainSrchWindow : public SrchWindow
{
```

```
    ...
private:
    void on_pushButton_clicked();
};

class EditSrchWindow : public SrchWindow
{
    ...
private:
    void on_pushButton_clicked();
};

void MainSrchWindow::on_pushButton_clicked()
{
    QString srch_str = ui->lineEdit->text();
    menu->show_key(srch_str, SHOW_SRCH);
}
void EditSrchWindow::on_pushButton_clicked()
{
    QString srch_str = ui->lineEdit->text();
    emit send_key(srch_str);
}
```

之后是重载：重载是 C++特有的一种概念，允许设计者用同样的名字来定义函数，只要它们的调用可以通过参数的个数或类型来进行区别。重载和多态一样，也是用来实现功能相似的函数，增加代码的可维护性和稳定性的。不过由于重载函数需要在参数个数或类型上有所区分，与多态中为同一个函数赋予不同的实现方式有所不同，分别有各自的使用场景：

```
class FileMenu
{
public:
    void show_all();
    void show_all(QListWidget *listWidget);
    ...
};

void FileMenu::show_all()
{
    this->listWidget->clear();
    for(int i = 0; i < this->file_num; i++)
    {
       QString string = this->file_queue.at(i);
       item_add(string, i+1);
    }
}
void FileMenu::show_all(QListWidget *listWidget)
{
    this->listWidget = listWidget;
    show_all();
}
```

最后是“高内聚低耦合”：“高内聚低耦合”是面向对象设计中评价一个系统设计好坏的标准，它希望一个模块中各个元素之间的联系程度尽量高，而模块之间相互联系的紧密程度尽量低。在代码设计的过程中，功能相近、联系紧密的方法被尽量归拢在同一个类中，避免

一个类为它的内部属性提供直接的外界访问，减少其向外界提供实现其职责的方法，并降低这些方法受到类内部设计变化的影响，从而提高了内聚性；类之间的依赖关系尽量减少，避免互相依赖、循环依赖的出现，并保证类之间的影响只依赖于其他类提供的公开接口，而不依赖于它的内部工作原理，从而降低了耦合性。

四. 总结与展望

这个项目的代码设计与实现过程依然存在着一定的不足：在编写代码的过程中，我依然没有充分地运用面向对象的设计方法，总是面向“需求”编程，想到什么就写什么，经常要回头返工、重构代码。也正是由于没有在一开始就想好整体框架和布局，导致一部分函数的职责、功能虽然比较相近，却都被分开来单独实现，造成了一定的重复和冗余。此外，目前的几个类的功能依然比较复杂，没有很好地满足单一职责原则，这也需要再进一步地修改和细化。

另外，这一项目仍可以在一些方面进行补充和完善。首先是文档的持久化措施：目前这个项目需要记录下来的所有数据都是通过.txt 文档存储的，这样既不方便管理，在数据查找和读写的速度上也不尽人意。下学期的数据库系统课程或许可以带来一定的帮助，为这个项目增加一种更简单高效的文档持久化措施。此外，这个项目的界面还可以进一步美化，比如为各个窗口、选项添加一些图标，修改一下 ui 界面的样式等，这些内容又可以联系上人机交互课程的相关内容了。然后是之前计划添加的插入图片或表情、提供云端存储等功能，有机会的话也可以尝试实现一下。

这次大作业是我第一次用 C++配合 Qt 框架编写代码，花费了不少时间学习、熟悉各种方法，但同样收获颇多。这也是我第一次接触面向对象的编程思想，对这种思想有了初步的了解和体验，也逐渐明白了面向对象的思想与设计方法对于整个项目的作用之大、好处之多。对我而言，这个项目不仅仅是一次课程大作业，同样也是为自己设计的一件趁手的工具。所以未来我还会继续修改、完善整个项目，相信面向对象的编程思想也会一直发挥出它的巨大价值。

参考文献

QTextEdit 成员函数的介绍：https://blog.csdn.net/AAAA202012/article/details/120622526

两个窗口进行数据传输：https://www.cnblogs.com/zyore2013/p/4662911.html

获取 QTextEdit 选定的内容：https://blog.csdn.net/luochengguo/article/details/7520022

“另存为”功能的实现：https://blog.csdn.net/caoshangpa/article/details/78711743