**面向对象程序设计课程大作业**

**——个人日记软件的设计实现**

**宋宇轩 2019K8009929042**

1. **需求分析与进度安排**

我想要实现的程序是一个日记软件，支持单篇日记的文字排版、存储和记录日期等功能，同时可以对多篇日记进行统一查看和检索。预想的界面如下图所示：

图形用户界面, 应用程序

已生成极高可信度的说明

图1 初始界面示意图

主界面主要可以分为3个部分：左侧按时间先后顺序显示目前已有的所有文档的简要信息，单击后即可打开对应文档。右上角是一个日历，右下角则为新建文档和按关键字搜索文档功能的按键。在单击日历中的某一天后，左侧的显示区会显示生成日期为当天的所有文档，如下图所示：

图形用户界面

已生成高可信度的说明

图2 按日期检索文档示意图

新建文档或打开文档后的界面与windows的记事本应用界面类似，支持中英文（UTF-8编码字符）的编辑、排版、复制粘贴和保存。同时删除文档也将在打开文档之后的子页面内实现。后续将根据时间和进度安排尝试增加实时自动保存，插入图片、视频和emoji表情图，支持云端存储，与其它设备互联同步等功能。

目前的进度安排为以三次提交课程作业的时限为节点，将整个学期划分为3个阶段。第一阶段确定主要设计目标和需求，借助网上的资料敲定基本目标的实现方法；第二阶段进行程序编写，完成基本功能的大致实现；第三阶段完善基本功能，确保基本功能正确实现，随后尝试加入更多高级功能。

1. **设计目标与思路**

整个项目可以被分为两个模块：显示总体信息的主界面和提供文本编辑的子界面。主界面的目标状态如第一节所示，提供所有文档的总体展示，支持按日期检索和按关键字搜索，可以新建文档或打开文档。除此之外，还可以在初次打开文档时以只读模式展示，在选择编辑模式之后再启动文档的编辑。子界面将会是一个类似记事本的文本编辑器，用来书写、修改文档，可以支持中英文（UTF-8）的编辑，兼容鼠标操作（通过单击改变光标位置等），支持手动保存或自动保存，在关闭窗口时提示是否需要保存。除此之外，我还希望能支持撤销功能，实现复制粘贴功能（兼容系统本身的剪切板），为需要的功能增加快捷键（按下ctrl+s进行保存等），插入图片、视频或表情，统计字数，修改字体字号，实现文本内容搜索等功能。

目前看来我需要定义三个类：文本类Text和目录类Menu，分别用于对单个文档进行编辑和保存，以及对所有文档进行管理和检索；窗口控制类Window用于显示文本，接收键盘、鼠标的操作并调用对应的处理函数。

为了实现文档的编辑，文本类Text内需要用来记录文字的缓冲区text\_buff；我计划将每一篇生成的文档都使用.txt类型文件存储在本地，因此Text内需要记录文档所在地址text\_addr；除此之外还需要记录文档的基本信息，例如生成日期creat\_date，总字数tot\_word；为了实现提醒保存，还需要变量is\_change来记录是否进行了修改。

对文档的操作包含新建，打开，编辑，保存，删除，因此需要5个公共的成员函数来实现它们。新建函数new\_text()即是Text类的构造函数，负责设置存储地址，初始化生成日期、缓冲区及光标位置等基本信息。打开函数open\_text()根据类中的text\_addr变量寻址并打开对应文件，将数据写入缓冲区text\_buff以供编辑。编辑函数edit\_text()需要接收控制部件传来的指令码，并依据指令码对缓冲区做相应的修改（插入字符、删除字符等）。保存函数save\_text()根据原有地址或新输入的地址，将缓冲区中的内容输出到对应地址的文件中。对于删除操作del\_text()，我计划单独设置一个“已删除”文件夹，删除文档时将文档从原地址删除，将其放入这个文件夹之中并修改地址变量，再次删除时才调用析构函数。为此，类中还需要一个成员变量is\_delete来判断是否已经删除过一次。同时为了支持目录类的管理和检索，还需要输出生成时间和所在地址的函数get\_date()和get\_addr()。

目录类Menu将包含一个Text类型的优先队列，按照生成时间的先后顺序排序，同时记录目前文档的总数tot\_file。成员函数将包括增加文档（新文档入队），删除文档（从队列中删除对应文档），按时间检索和按文本检索。按时间检索只需要枚举队列里的文档，找到生成时间相符的即可。对于按文本内容检索，我目前的想法是将现有的所有文本依次打开，检查内容中有无检索关键词，之后再关闭文本。但这样需要多次访问内存，对于执行速度可能会有影响，未来可以进一步寻找更优秀的实现方法。

窗口控制类Window则包含光标位置cursor\_x和cursor\_y，以及当前需要执行的指令码instruction。成员函数有用来接收键盘、鼠标输入并确定目标操作的check\_inst()，根据操作调用相应处理函数的handle\_inst()，打印当前文档缓冲区的print\_buff()，以及移动光标的move\_cursor()。

综上所述，三个类的UML图如下所示：

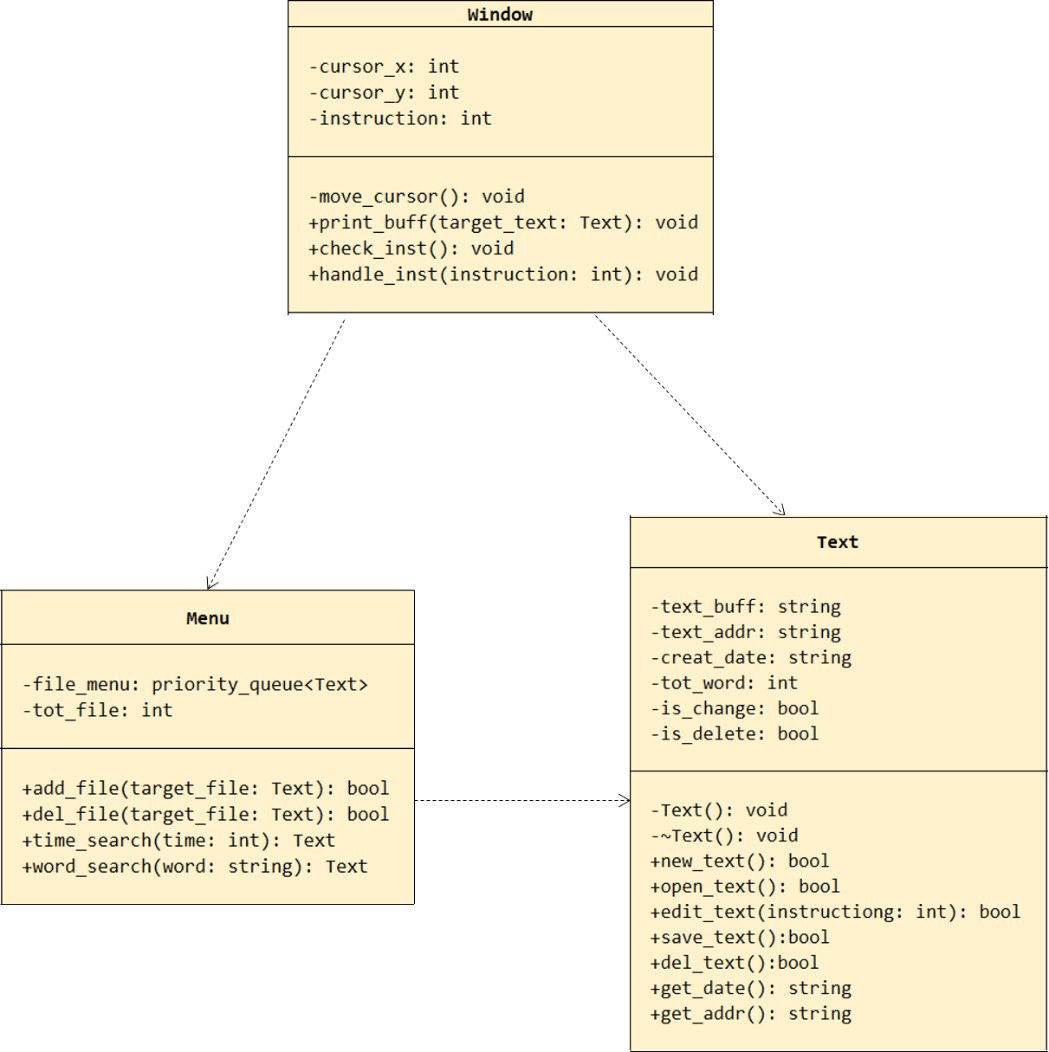


图3 程序设计UML图

1. **QT简介与窗口实现分析**

在介绍真正的程序设计之前，首先要说明的一点是我使用了C++语言和QT开发库来进行设计。使用C++语言而非JAVA语言主要是因为我对C++语言相对更为熟悉，不需要重新学习一门新的编程语言，同时设计这样一个规模稍大一些的程序也可以加深我对C++的理解。使用QT库则可以更方便的实现图形化界面，同时对鼠标、键盘操作的支持也更简单，免去了查询window API的麻烦。

QT设计中可以直接拖动功能控件来图形化地实现弹出窗口的UI设计，如下图所示：

图形用户界面

已生成高可信度的说明

图4 使用QT设计弹出窗口UI示意图

QT软件会根据设计出的界面自动创建一个UI类，包含先前加入窗口设计的各类按键、菜单、图表等控件的实现。这些控件都是QT已经封装好的类，并提供了需要的接口供设计者使用。以单击窗口中的按钮来关闭窗口为例：从零开始设计需要支持输出图形化界面，检测鼠标指针位置，检测单击动作，设计关闭窗口函数来停止运行窗口等功能，而这些功能大多需要调用windows API实现，这是我非常陌生的一块内容；而使用QT进行设计只需要在已有的on\_pushButton\_clicked()函数中调用QT封装好的close()函数，即可实现单击按钮时关闭窗口的功能，设计者不需要再担心更底层的实现，从而节省时间和精力。事实上，在正式开始编写程序代码之前，我就可以利用QT设计生成图1和图2那样的窗口了，随后只需要对各类控件的功能进行编程实现即可，这也从侧面证明了使用QT对于设计效率的巨大提升。

不过尽管利用QT进行设计如此便捷，多了解一下窗口和控件的具体实现也是有帮助的。还是以单击窗口中的按钮来关闭窗口为例，这一项目最重要的代码文件有4个： main.cpp, ui\_mainwindow.h, mainwindow.h和mainwindow.cpp。

main.cpp实现的功能非常简单：定义一个QApplication类的对象a用于运行程序，一个MainWindow类的对象w用于显示图形化界面，具体代码如下：

int main(int argc, char \*argv[])

{

QApplication a(argc, argv);

MainWindow w;

w.show();

return a.exec();

}

其中QApplication的父类为QGuiApplication，而QGuiApplication则继承自基类QCoreApplication，其主要作用为设置软件图标、接收鼠标和键盘的输入、弹出提示弹窗等较为底层的操作实现。MainWindow的父类为QMainWindow，再往上的父类为QWidget，QWidget多继承自QPaintDevice类和QObject类，其中QPaintDevice控制弹出窗口的位置、大小、颜色等特征，QObject则负责将一个信号（比如鼠标的单击）与对应的槽函数（类似处理函数）进行连接，保证对应操作的实现。

信号可以比作软件的中断，例如，按钮控件A被点击时就会广播“被单击”这一事件对应的信号。控件中可以有很多信号，每个信号代表不同的事件，而我们只需要向程序指定这一控件对哪些事件感兴趣即可。对于不感兴趣的事件的信号，程序就会忽略掉，不作响应。槽函数则是程序接收并响应某信号之后，需要执行的操作。QT提供了一些已经设计好了的槽函数，设计者也可以自定义新的槽函数。

ui\_mainwindow.h中声明了UI\_MainWindow类，包含了对于窗口界面的UI设计：

QWidget \*centralWidget;

QPushButton \*pushButton;

QMenuBar \*menuBar;

QToolBar \*mainToolBar;

QStatusBar \*statusBar;

其中中心窗口、菜单栏、工具栏和状态栏是每个窗口UI都有的，按键控件\*pushButton则是依照我的设计而自动添加的。

此外这个类中还包含两个成员函数：setupUI和retranslateUI。retranslateUI的作用是修改窗口与控件的显示文本（即窗口的标题、按键上显示的文字等）：

void retranslateUi(QMainWindow \*MainWindow)

{

MainWindow->setWindowTitle(QCoreApplication::translate("MainWindow", "MainWindow", nullptr));

pushButton->setText(QCoreApplication::translate("MainWindow", "Close", nullptr));

}

setupUI可以分为三个部分：第一部分是设置窗口与控件的大小、位置、命名等特征，第二部分是调用retranslateUI函数设置窗口与控件的显示文本，第三部分是调用QMetaObject::connectSlotsByName(MainWindow);函数来将当前窗口中的控件与其对应的槽函数连接起来：

void setupUi(QMainWindow \*MainWindow)

{

if (MainWindow->objectName().isEmpty())

MainWindow->setObjectName(QString::fromUtf8("MainWindow"));

MainWindow->resize(918, 446);

centralWidget = new QWidget(MainWindow);

centralWidget->setObjectName(QString::fromUtf8("centralWidget"));

pushButton = new QPushButton(centralWidget);

pushButton->setObjectName(QString::fromUtf8("pushButton"));

pushButton->setGeometry(QRect(280, 120, 341, 91));

MainWindow->setCentralWidget(centralWidget);

menuBar = new QMenuBar(MainWindow);

menuBar->setObjectName(QString::fromUtf8("menuBar"));

menuBar->setGeometry(QRect(0, 0, 918, 26));

MainWindow->setMenuBar(menuBar);

mainToolBar = new QToolBar(MainWindow);

mainToolBar->setObjectName(QString::fromUtf8("mainToolBar"));

MainWindow->addToolBar(Qt::TopToolBarArea, mainToolBar);

statusBar = new QStatusBar(MainWindow);

statusBar->setObjectName(QString::fromUtf8("statusBar"));

MainWindow->setStatusBar(statusBar);

retranslateUi(MainWindow);

QMetaObject::connectSlotsByName(MainWindow);

}

mainwindow.h文件中主要声明了表示图形化窗口的MainWindow类，其中除了构造函数和析构函数之外，还声明了需要用到的槽函数，并定义了对象\*ui来表示当前窗口：

class MainWindow : public QMainWindow

{

Q\_OBJECT

public:

explicit MainWindow(QWidget \*parent = nullptr);

~MainWindow();

private slots:

void on\_pushButton\_clicked();

private:

Ui::MainWindow \*ui;

};

mainwindow.cpp用于定义MainWindow类里的成员函数，这也是利用QT设计程序时主要需要编写的部分。除了自带的构造函数和析构函数以外，这个程序里增加了按键与一个槽函数on\_pushButton\_clicked()，这个函数调用QT提供的close()函数来关闭窗当前口，具体代码如下：

MainWindow::MainWindow(QWidget \*parent) :

QMainWindow(parent),

ui(new Ui::MainWindow)

{

ui->setupUi(this);

}

MainWindow::~MainWindow()

{

delete ui;

}

void MainWindow::on\_pushButton\_clicked()

{

close();

}

以上就是QT在设计好一个窗口后自动生成的主要代码了，接下来需要实现的就是根据需求安排控件和信号，并编写相应的槽函数来实现对应的功能。

1. **程序外观与功能实现**

目前实现的程序外观与先前预想的程序界面基本一致：

图形用户界面, 应用程序

已生成极高可信度的说明

图5 初始界面示意图

运行程序后立即弹出的初始界面（以下称为主界面）大小固定为870×500（像素），主体分为3个部分：左侧为文档信息的显示区，右上角是一个日历，右下角有3个按键。

在新建或打开某个文档后，文档的编辑界面（以下称为子界面）就会弹出：

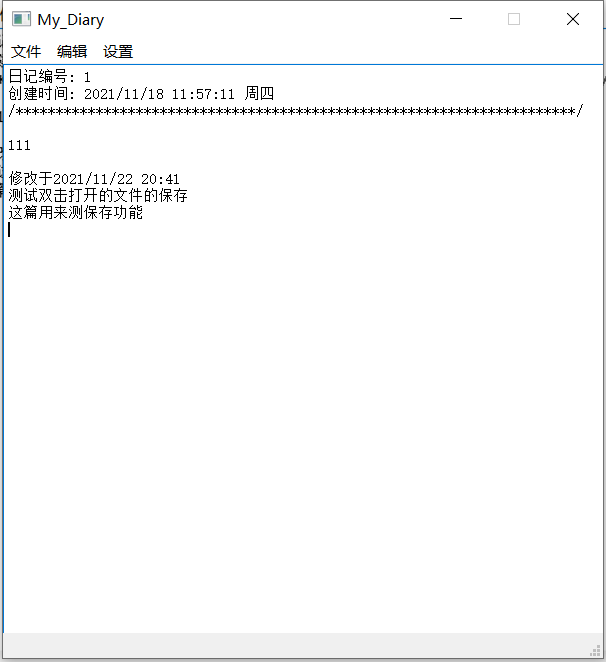


图6 文档编辑界面示意图

子界面与Windows自带的记事本程序外观类似，大小固定为600×620（像素）。子界面的主体部分为文本编辑器，上方有3个下拉菜单，其中各自包含了一些操作或选项。

就目前的完成进度而言，主界面左侧的显示区会按时间先后顺序显示目前已有的所有文档的编号和生成时间，双击某一项后即可打开其对应的文档。右上角的日历目前仅能显示日期，计划在之后添加一项新功能：在单击日历中的某一天后，左侧的显示区会显示生成日期为当天的所有文档。右下角的3个按键中目前只完成了新建文档，删除文档和搜索文档的功能会在之后加以实现。文档的默认存储位置为程序所在目录的diary\_file文件夹中，存储类型为.txt格式。

子界面的文本编辑部分支持中英文等常见字符的编辑、删改和保存，可以通过鼠标或键盘改变光标位置、选中部分内容，并支持通过快捷键或右键菜单中的选项来实现全选、剪切、复制、粘贴或撤销等功能。新建文档时会在文本编辑区的最上方自动写入该文档的编号和创建时间。

子界面上方的3个下拉菜单在单击后会展示其中包含的操作或选项，具体内容如下图所示：

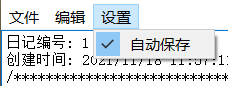
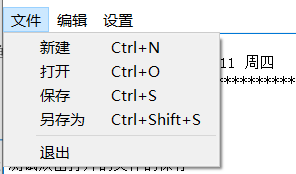


图7 “文件”菜单示意图 图8 “编辑”菜单示意图 图9 “设置”菜单示意图

“文件”菜单包含5个操作，其中新建操作和主界面的新建按键功能一致，都会新建一个空白文档并以子界面的形式呈现，文档开头会自动写入日记编号和创建时间：

图形用户界面, 应用程序

已生成极高可信度的说明

图10 新建文档界面示意图

打开操作会弹出地址检索窗口，可以选定任意文档并打开：

图形用户界面, 文本, 应用程序

已生成极高可信度的说明

图11 打开文档界面示意图

保存操作即为保存文档到当前地址；另存为操作还没有实现；退出操作与单击右上角的×按键功能一致，会在没有修改过文档或已经保存了当前文档后直接关闭子界面，在文档被修改过且没有保存时提示是否需要保存：

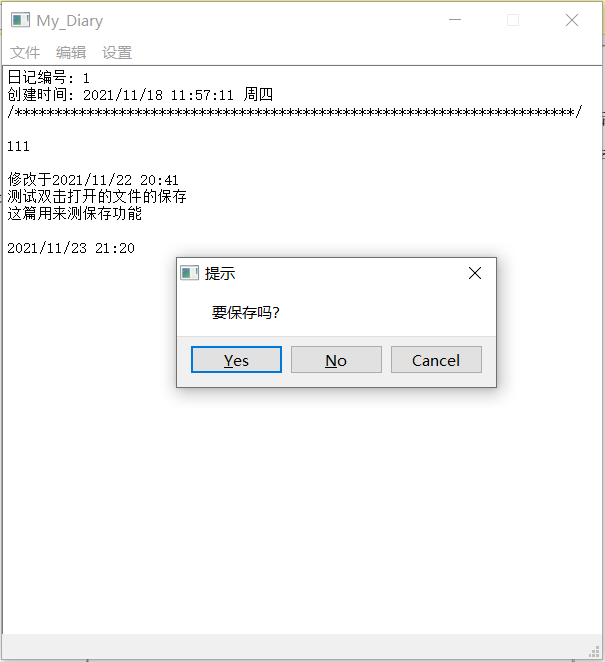


图12 退出文档时的提示界面示意图

“编辑”菜单包含9个操作，目前这些操作在被单击时都是不起作用的，但撤销、剪切、复制、粘贴、删除和全选可以通过快捷键或右键菜单来完成，恢复、查找和替换操作目前还未实现。

“设置”菜单仅包含1个选项：自动保存。在单击勾选自动保存选项时，文档会自动执行一次保存操作，并在之后的每一次修改文档后都自动进行一次保存；在再次点击自动保存选项（取消勾选）之后，文档的修改将不再自动进行保存。

1. **代码分析与功能实现**

图示, 日程表

已生成极高可信度的说明

图13 项目总体UML类图

得益于QT库中大量定义完整的类，我自己实际进行实现的类只有3个，它们之间的关系也很简单。如图9所示，上半部分展示了项目中使用的部分QT库自带的类的大致情况，下半部分则为我自己实现的3个类：MainWindow, EditWindow和FileMenu，分别用于控制主界面、控制子界面，以及管理文档与显示区。类图中的-$前缀表示这个成员函数是一个槽函数，它可以与信号连接起来，在对应信号发射时被自动调用。

MainWindow类和EditWindow类中都包含\*ui成员变量，用于表示图形化的界面；EditWindow类中的file\_name变量记录目前打开的文档的名称，is\_changed变量记录文档是否发生了修改（是否需要保存），auto\_save变量记录目前是否需要执行自动保存。FileMenu类中的file\_num变量用于记录目前的日记文档总数。file\_queue是一个链表，按照创建时间的先后顺序存储所有文档的名称。listWidget是QListWidget类的对象，用于控制主界面左侧的文档显示区。

目前该项目的程序所在目录下有一个命名为dairy\_file的文件夹，里面存有用于记录文档数目和题目的menu.txt，以及所有的日记文档。在整个项目开始运行时就会新建一个FileMenu类的全局对象menu，用于管理所有的文档。menu的构造函数会打开menu.txt文件，从中读取文档的数量和所有文档的名称并记录下来，具体代码如下所示：

FileMenu::FileMenu()

{

this->listWidget = NULL;

QFile fp("./diary\_files/menu.txt");

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)

{

this->file\_num = line.toInt(); //menu.txt的第一行写有文档总数

}

else

{

this->file\_queue += line; //此后每一行对应一个文档的名称

}

line\_num++;

}

fp.close();

}

}

项目运行时会新建MainWindow类的对象w，它的构造函数会生成图形化的主界面，并将主界面的大小设置为固定的870×500（像素）：

ui->setupUi(this);

setFixedSize(870,500);

随后会向全局对象menu中传入主界面左侧的文档显示区控件的地址，并向其中添加所有文档的信息：

menu->item\_add(ui->listWidget);

FileMenu::item\_add(QListWidget \*listWidget)函数会将menu中的成员变量listWidget指向主界面左侧的文档显示区，将其清空并输入所有已有文档的信息：

void FileMenu::item\_add(QListWidget \*listWidget)

{

if(this->listWidget == NULL)

{

this->listWidget = listWidget;

}

this->listWidget->clear();

for(int i = 0; i < this->file\_num; i++)

{

QString string = this->file\_queue.at(i);

string.replace(4, 1, "/");

string.replace(7, 1, "/");

string.replace(10, 1, " ");

string.replace(13, 1, ":");

string.replace(16, 1, ":");

QString file\_name = string.left(19); //获得目标格式的生成时间

QString target\_str = "第" + QString::number(i+1) + "篇: " + file\_name;

this->listWidget->addItem(target\_str);

}

}

这样一来，主界面的所有初始化就都完成了。

主界面的新建按键和子界面“文件”菜单下的新建操作都可以实现新建文档，不同之处在于主界面需要新建一个EditWindow类的对象New\_window，用于生成一个子界面：

void MainWindow::on\_pushButton\_new\_clicked()

{

EditWindow \*New\_window = new EditWindow;

New\_window->new\_window();

}

其中子界面的构造函数会生成图形化的界面，将其大小设置为固定的600×600（像素），将文本编辑区自带的textChanged()信号与槽函数set\_change\_status()连接起来，并初始化成员变量：

EditWindow::EditWindow(QWidget \*parent) :

QMainWindow(parent),

ui(new Ui::EditWindow)

{

ui->setupUi(this);

setFixedSize(600,620);

connect(ui->textEdit,SIGNAL(textChanged()),this,SLOT(set\_change\_status()));

is\_changed = 0;

auto\_save = 0;

}

而子界面中不会生成新的子界面，而是直接修改当前界面的文本编辑区：

void EditWindow::on\_action\_new\_triggered()

{

new\_window();

}

new\_window()函数是一个公有成员函数，会调用私有成员函数text\_init()来初始化文本，然后生成图形化的子界面：

void EditWindow::new\_window()

{

text\_init();

show();

}

void EditWindow::text\_init()

{

QDateTime \*DateTime = new QDateTime(QDateTime::currentDateTime());

QString date\_str = DateTime->toString("yyyy/MM/dd hh:mm:ss ddd");

QString num\_str = QString::number(menu->file\_num+1);

QString str = "日记编号: "

+ num\_str

+ "\n创建时间: "

+ date\_str

+ "\n/\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*/\n\n";

ui->textEdit->setText(str);

ui->textEdit->moveCursor(QTextCursor::End, QTextCursor::MoveAnchor);

file\_name = DateTime->toString("yyyy\_MM\_dd\_hh\_mm\_ss") + ".txt";

//文档名即为生成时间

is\_changed = 0;

}

文本编辑通过QT已有的控件QTextEdit实现，支持中英文字符的编辑与复制、粘贴、撤销等功能。

打开文档同样有两种方式：双击主界面左侧显示的某个文档，或是在子界面单击“文件”菜单中的打开操作。主界面打开的会是已有的日记文档；子界面则会进行地址检索，打开选定的文档，因此可以打开可找到的任意文件，但编辑器不一定兼容目标文件的格式。

两种打开方式的核心函数都是EditWindow::IO\_open(const QString &file\_name)，它会根据传入的文件名打开文件，并将文件中的所有内容写入文本编辑区：

bool EditWindow::IO\_open(const QString &file\_name)

{

QFile fp(file\_name);

if (!fp.exists())

return false;

if (!fp.open(QIODevice::ReadOnly | QIODevice::Text))

return false;

QTextStream Stream(&fp);

ui->textEdit->setPlainText(Stream.readAll());

fp.close();

is\_changed = 0;

show();

return true;

}

不同之处在于，主界面需要进行一些操作来获取目标文档的正确目录和名称，并新建一个子界面：

void MainWindow::on\_listWidget\_itemDoubleClicked(QListWidgetItem \*item)

{

QString tmp = item->text();

QString file\_name = tmp.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";

EditWindow \*New\_window = new EditWindow;

New\_window->file\_name = file\_name;

New\_window->IO\_open("./diary\_files/"+file\_name);

}

子界面则要先弹出对话框进行地址检索，再利用得到的目录尝试打开文件：

void EditWindow::on\_action\_open\_triggered()

{

open\_file();

}

void EditWindow::open\_file()

{

QString curPath=QDir::currentPath();

QString dlgTitle="打开一个文件";

QString filter="所有文件(\*.\*);;文本文件(\*.txt)";

file\_name=QFileDialog::getOpenFileName(this,dlgTitle,curPath,filter);

if (file\_name.isEmpty())

return;

if(!IO\_open(file\_name))

{

QMessageBox::warning(NULL,"警告","文件打开失败!");

}

}

保存文档主要通过EditWindow::save\_text()函数实现，大体分为修改文档和修改目录两部分。EditWindow::IO\_save(const QString &file\_name)函数通过传入的文档名新建或打开文档，将当前子界面的文本编辑器内的内容写入其中；FileMenu::menu\_add(QString file\_name)函数检查当前文件队列中是否包含传入的文档，并对menu.txt进行相应修改；最后由FileMenu::item\_add(QListWidget \*listWidget)函数对主界面的文档显示区进行修改：

void EditWindow::on\_action\_save\_triggered()

{

save\_text();

}

void EditWindow::save\_text()

{

IO\_save(file\_name);

menu->menu\_add(file\_name);

menu->item\_add(NULL);

is\_changed = 0; //保存后认为文档没有被修改，可以直接关闭

}

bool EditWindow::IO\_save(const QString &file\_name)

{

QFile fp("./diary\_files/"+file\_name);

if (!fp.open(QIODevice::WriteOnly | QIODevice::Text))

return false;

QTextStream stream\_out(&fp);

QString str=ui->textEdit->toPlainText();

stream\_out<<str<<endl;

fp.close();

return true;

}

void FileMenu::menu\_add(QString file\_name)

{

if(!this->file\_queue.contains(file\_name))

{

this->file\_queue.append(file\_name);

this->file\_num++;

} //如果文档队列中没有目标文件

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();

}

}

自动保存功能本质上就是在选项被勾选时执行一次保存操作，此后每当文档被修改时就进行一次保存，直到选项被取消勾选。textChanged()信号用于表示文本编辑器中的内容是否发生变化，EditWindow的构造函数中将其与槽函数set\_change\_status()连接了起来，每当内容被修改就会自动执行该函数，根据auto\_save变量决定是否需要保存文档：

void EditWindow::on\_action\_autosave\_toggled(bool arg1)

{

if(arg1) //arg1 == 1表示选项被勾选

{

save\_text();

}

auto\_save = arg1;

}

void EditWindow::set\_change\_status()

{

if(auto\_save)

{

save\_text();

}

else

{

is\_changed = 1;

}

} //每当文本被修改时，如果需要自动保存则执行保存，否则记录文本已改变

为了在退出子界面时检查是否需要保存文档，项目中重写了关闭事件的处理函数，利用EditWindow::window\_close()判断是否需要保存，在需要保存时弹出提示窗口，并根据选择的结果来决定下一步的动作：

void EditWindow::closeEvent(QCloseEvent \*event)

{

bool need\_close = window\_close();//拦截关闭窗口的行为

if(need\_close) event->accept();

else event->ignore();

}

bool EditWindow::window\_close()

{

if(is\_changed==0)

{

return 1; //文档没被修改则直接退出

}

else

{

QMessageBox box;

box.setWindowTitle("提示");

box.setText("要保存吗？");

box.setStandardButtons(QMessageBox::Yes

| QMessageBox::No |QMessageBox::Cancel);

int ret = box.exec();

if(ret == QMessageBox::Yes)

{

save\_text();

return 1; //保存并退出

}

else if(ret == QMessageBox::No)

{

return 1; //直接退出

}

else

{

return 0; //取消，不退出

}

}

}

1. **后续设计分析**

这个项目目前已经完成了最基本的功能，但一方面目前仍有一些我想要添加的功能没有实现，另一方面也还存在一些bug或不够完善的地方，因此在这里也对接下来需要做的工作做一个总结和分析。

对于后续计划实现的功能，主要包含完成主界面删除和搜索两个按键的功能、根据日历中选中的日期显示文档、完成子界面“文件”菜单和“编辑”菜单中的所有操作。删除可以将选中的文档移至另一个“回收站”文件夹。搜索最简单的实现方法就是依次打开所有文档逐字比对，但可能会耗费很多时间，可以再寻找更好的策略。根据选中日期显示文档也不麻烦，只需要将选中的日期传入函数，根据日期检索文档，最后刷新显示区即可。另存为与打开操作类似，结合已有的打开和保存操作的代码即可。“编辑”菜单中的操作大都是QTextEdit类自带的，添加起来也不复杂。

除此之外，要是时间足够，我还想添加插入图片和表情、字数统计、修改字体字号、云端存储等功能。QTextEdit类支持富文本编辑，应该是可以实现插入图片的。字数统计本身并不复杂，但我希望能像Windows Word一样输出在页面下方的菜单栏而不是文本之中，还不确定该怎么实现。QTextEdit类应该可以修改整个文本的字体字号，但只修改选定内容我还不确定能否实现。云端存储大概就需要用脚本连接github或百度网盘之类的网站，还要实现自动上传，目前完全没有头绪。

关于问题和缺陷，首先还有一个bug等待修复：在子界面中点击打开操作，然后关闭弹出的地址检索窗口，此时再保存原有文档会导致menu.txt中的file\_num错误+1，且没有新文档生成。此外在子界面新建文档会直接覆盖当前文档，不会检查当前文档是否需要保存，可能导致内容的丢失。所以新建也应该与退出操作类似，根据情况提示是否需要保存

另外在书写这次报告时把所有代码又重新梳理了一遍，发现有一些地方的实现存在不必要的操作，比如FileMenu::menu\_add()函数中无论是否新增了文件，都会从头重写一遍menu.txt，实际上只需要在真的新增了文件时修改第一行的file\_num，并在最后一行增加新增文件的名称即可。为了提高程序的运行效率，应该修改掉这种不必要的操作。

**参考文献**

对C++的简单介绍：https://www.runoob.com/cplusplus/cpp-inheritance.html

从0开始设计文本编辑器：https://www.catch22.net/tuts/neatpad#

UML图箭头的意义：https://blog.csdn.net/wglla/article/details/52225571

对QT的简介和初步使用：http://c.biancheng.net/view/1804.html

在线查看QT设计源码：https://code.woboq.org/qt5/

利用QT设计文本编辑器：https://www.bilibili.com/video/BV1Lo4y1Z7hz?p=7

对信号和槽函数的介绍：https://blog.csdn.net/u014453898/article/details/70242786

重载运算符实现优先队列：https://www.cnblogs.com/huashanqingzhu/p/11040390.html

对QT各个控件的简介：https://www.cnblogs.com/zach0812/category/1524140.html?page=3

QList控件的介绍：https://blog.csdn.net/qq\_30725967/article/details/98205514

QListWidget控件的介绍：https://www.pianshen.com/article/901542842/

拦截关闭事件：https://blog.csdn.net/vah101/article/details/6133728

QT类继承关系图：https://blog.csdn.net/qq\_31036127/article/details/106890462?spm=1001.2101.3001.6650.6&utm\_medium=distribute.pc\_relevant.none-task-blog-2%7Edefault%7EOPENSEARCH%7Edefault-6.no\_search\_link&depth\_1-utm\_source=distribute.pc\_relevant.none-task-blog-2%7Edefault%7EOPENSEARCH%7Edefault-6.no\_search\_link