面向对象程序设计第二次作业

——程序功能与设计分析

宋宇轩 2019K8009929042

写在前面：

这是第二次作业的报告文档，主要介绍了截止 11 月 23 日我的项目的实现情况，包括目前程序的外观，实现的功能及其代码分析，以及对后续设计的分析和思考。目前这个项目已经可以实现最基本的新建、打开、编辑和保存日记，界面外观也已经基本确定下来了。但现在仍有一部分我想要实现的功能没有完成（例如部分按键已经存在但还不起作用），也有一些更复杂的功能不知道有没有机会实现。

一. 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：

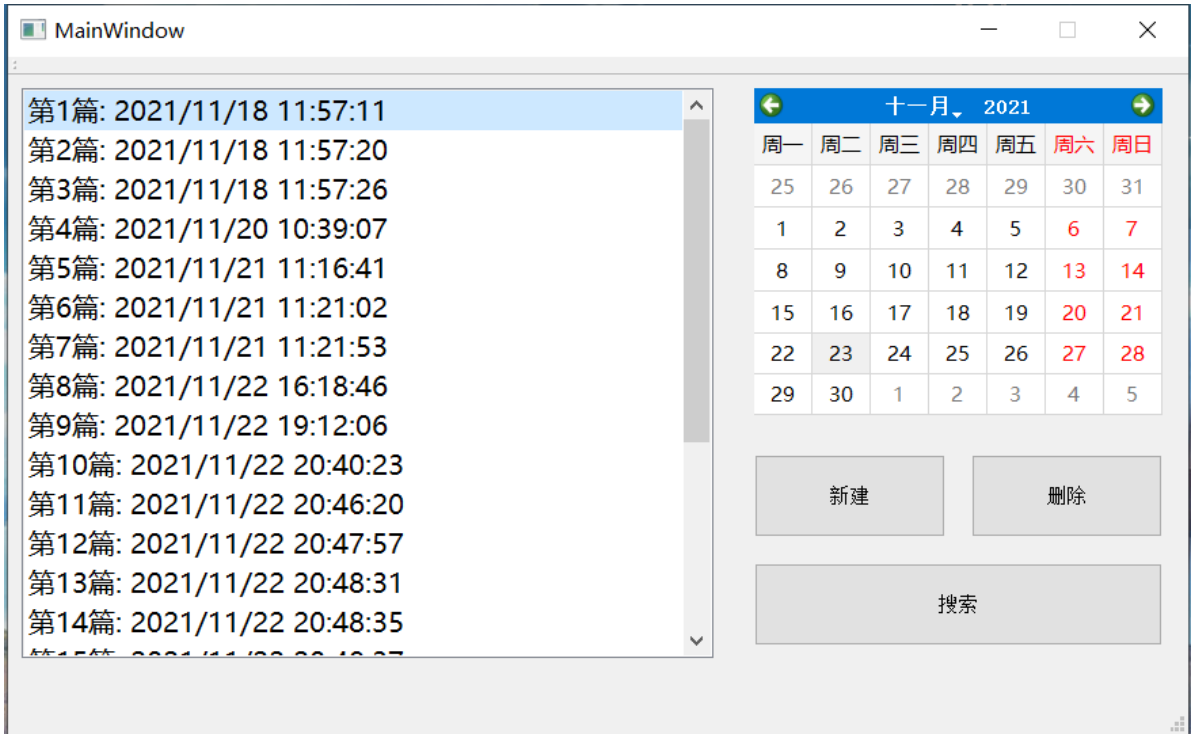

图 1 初始界面示意图

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 2 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 3 “文件”菜单示意图

图 4 “编辑”菜单示意图

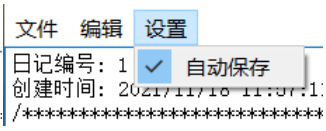

图 5 “设置”菜单示意图

“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：

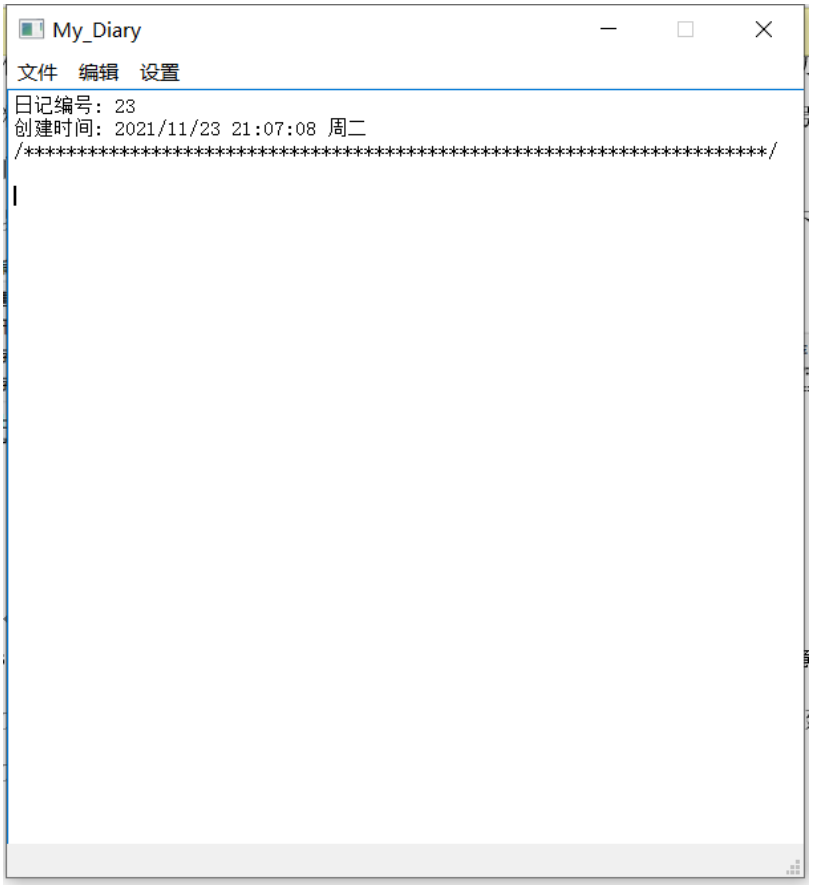

图 6 新建文档界面示意图

打开操作会弹出地址检索窗口，可以选定任意文档并打开：

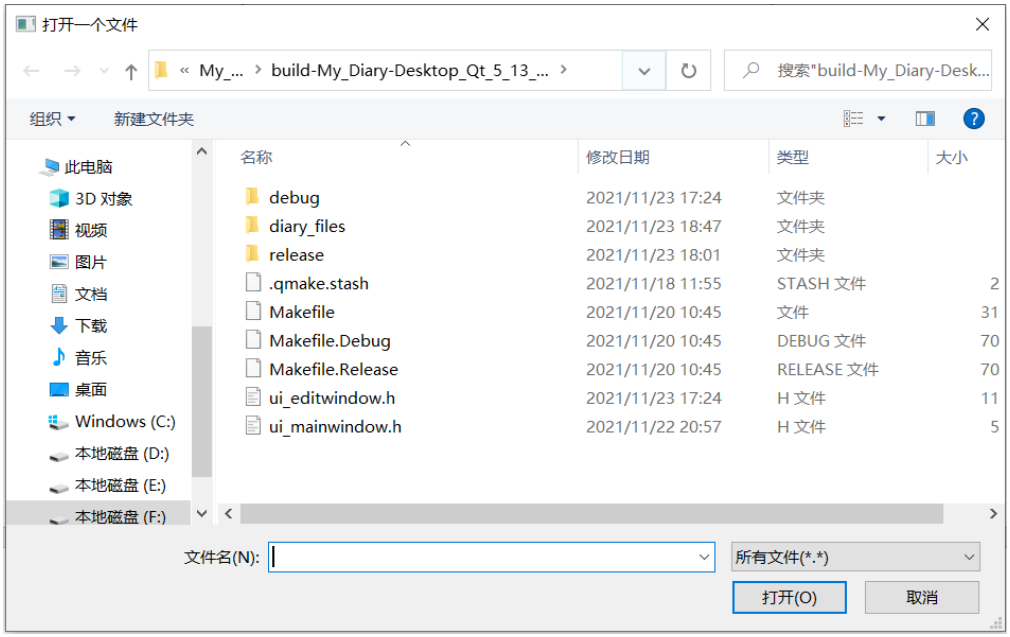

图 7 打开文档界面示意图

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：

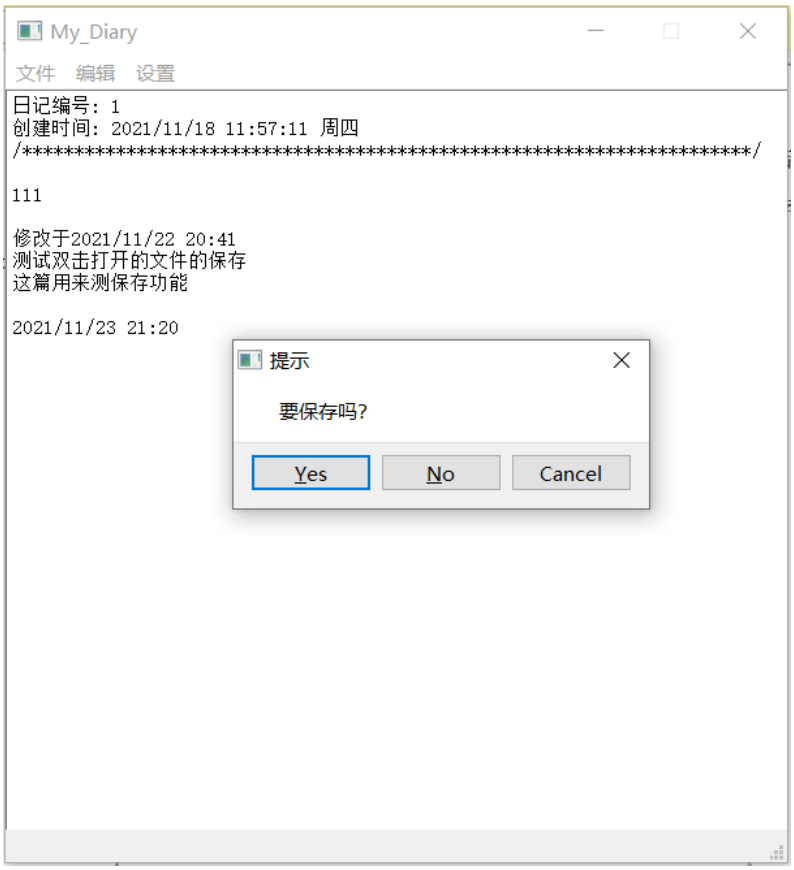

图 8 退出文档时的提示界面示意图

“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。

二. 代码分析与功能实现

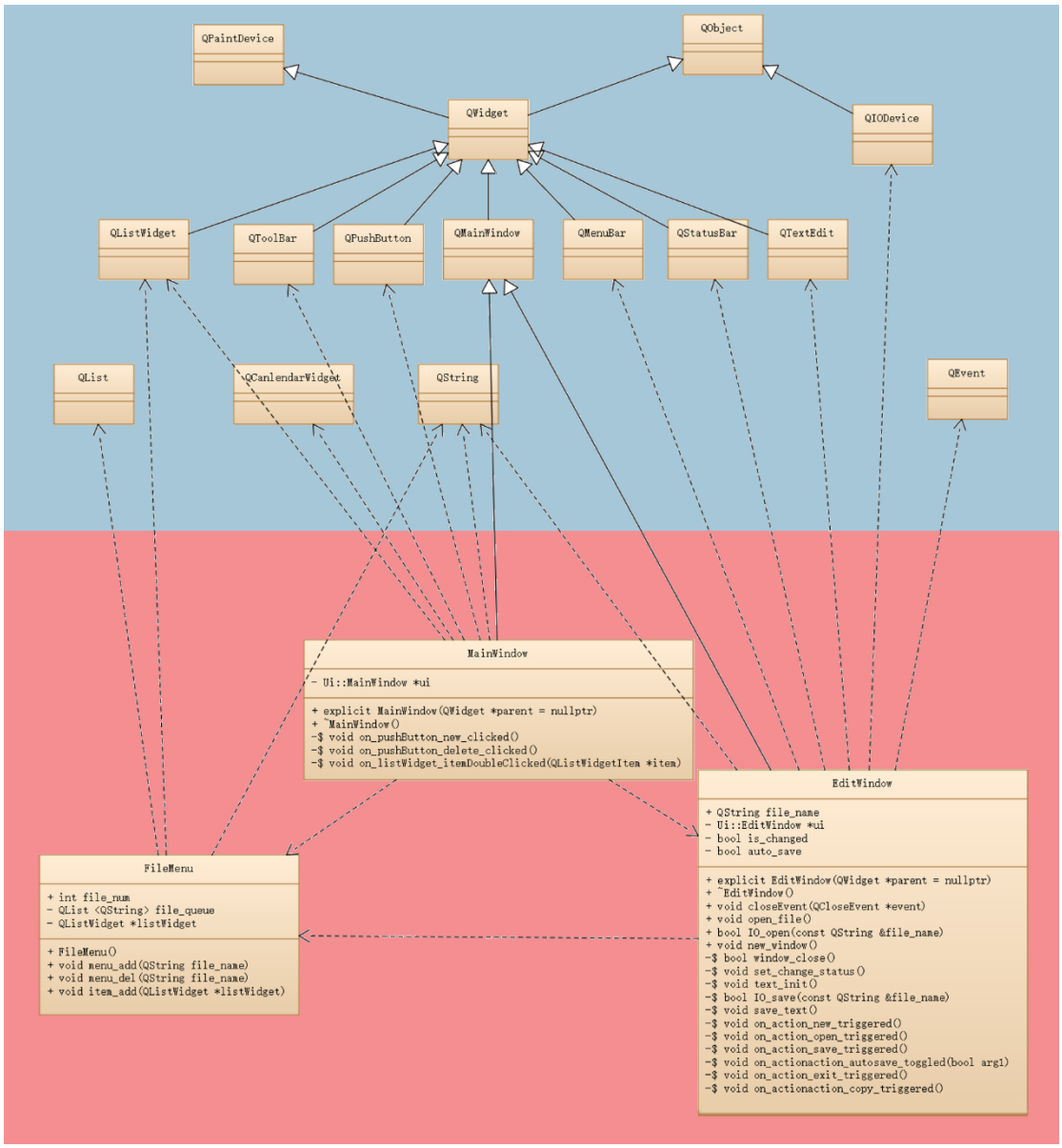

图 9 项目总体 UML 类图

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
        }
    }
}
```

三. 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 QT 的简介和初步使用：http://c.biancheng.net/view/9418.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 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