文本编辑器项目设计说明

第十九小组 成员：杜绅 王涵 李元 曹颢玮

需求分析

- 设计需求

  本文本编辑器的开发主要包括界面窗口的菜单栏和工具栏。以菜单栏和工具栏为主体函数，下面包括诸多小的功能模块实现函数，同时还涉及大量的函数调用。在功能模块实现函数中，包含大量的信号和槽之间的关联和触发，以及实现按钮和工具操作与实际操作之间的响应。

- 功能需求

  本次设计在计算机操作系统下，以QT作为开发工具的面向对象，用C++编写一款可以实现基本的创建、保存等文本操作；剪切、复制、粘贴等编辑操作；字体颜色、大小设置的格式操作。

- 用户需求

  计算机以及网络技术的飞速发展，社会正快速向信息化社会前进，我们需要更智能、更专业的软件帮助我们完成工作，从而提高工作效率。所以本项目所设计的应用程序完美的诠释了效率的问题，而且具有使用方便，易于修改，界面简单等特点。

总体设计

- 功能目标

  文本编辑器的设计是基于Qt开发的一款小型的软件，主要是为了满足普通用户对文本文档进行一些的简单操作。通过Qt的编程环境，生成一个用户友好的界面，用户可以使用该文本编辑器针对连续的问题进行一些基本操作。

- 开发工具介绍

  Qt是一个跨平台的桌面，广泛用于开发GUI程序，又被称为部件工具箱。Qt利用C ++编写的框架，方便开发者针对已经完善的类进行继承，在已有的功能上去实现符合用户需要的功能

- 前端编写

  文本编辑器的GUI可利用Qt框架自带的设计模式，该工具能够直观、便捷地设计出美观的GUI。在本次合作中，前端的编写也同在textedit.cpp当中完成，这就要求前端设计者对于后端编写实现的函数与基本功能具有一定的熟悉，同时也要了解信号与槽机制的相关使用方法。

- 后端编写

  根据实现功能的要求，负责的后端的人使用Qt编写相对应的函数，函数要求有充分的注释，包括函数功能、参数说明和用法示例等。我们根据需求封装了edit类、TextEditor类、qdialog等。

  同时，我们安装了事件过滤器，实现对快捷键功能的重定义。我们还继承了TextEditor类的文本编辑器框架。实现了用户需要的功能，我们还通过qdialog实现了查找与替换功能。

系统模块说明

- 文本编辑器的界面

- 相关类的介绍

  我们继承了QTextEdit类，在继承了该类的基础上重写了文本编辑器的基本功能，并实现对原先默认右键菜单的修改。

  我们继承了QMainWindow类，实现在相关操作被触发时跳出弹窗。

  Qstack类的成员函数也同时被调用，实现文本编辑器当中文本的存储，进而满足撤销，重做等相关功能。

  同时，我们使用QDialog类进行查找替换等操作，并使用QpushButton实现按钮的设计。我们使用Qaction类进行信号与槽机制中槽函数的连接，并通过Qmenu实现了菜单栏等功能。

系统设计难点及解决

- 带格式的字体保存与读取

  文件的保存与读取的难点在于如何读取和保存带有格式的文本，不带文字样式的文本编辑器只需要将文本框中的文字导入到系统文件就能实现保存，读取也是直接导入系统文件，直接显示QString类型的变量即可，但是读取保存带有样式的文本就不仅仅如此了。

  因为QTextEdit类是一种富文本，可以使用ui->TextEdit->document()->toHtml()，将带样式的文本信息保存成html文本，并赋值给QString类型的变量，此后就可以使用QFile将该变量写入系统文件中去，实现保存功能，程序采用了陈奇老师上课时的建议，保存的文本采用了二进制文件；导入时也是通过QFile实现二进制文本的导入，再使用ui->TextEdit->document()->setText()，将html语言描述的文本显示在界面中。

- 查找与替换的实现

  针对如何查找文字，又如何选择将找到的文字单一或者批量的替换成所需的文字这一问题，我们定义了一个QDialog类型的变量，通过记录用户在查找框和替换框中的文字与按下的按钮实现了相应的功能。

  查找功能是先记录查找的内容，并将查找的内容与文本内容进行对比，记录总共出现的次数，按下Next后，会找到下一个段文字的下标，并将文字亮标处理，当查找光标来到文本的末尾，程序会通过判断，自动的从头开始查找，实现循环查找的功能。

替换功能分单一替换和全部替换，全部替换采用了QString类中的replace成员函数，可以直接将文本中的需要替换的内容进行一次性替换；单一替换则用到了查找功能记录下的文字下标，通过确定下标的方式剪切文本信息，将下标之前的文本、替换后的文本和之前无需改变的文本进行组合，通过先清空再显示的方式，将单一替换后的文本信息显示在界面上，实现了单一替换的功能

- 判断文档是否被保存

  为了提醒用户文档尚未保存，我们通过ui->textEdit->document()->isModified()进行判断，若文档被修改了，则在用户选择新建或者退出时会跳出Messagebox的弹窗，弹窗中有选择保存、不保存、取消的三个按钮，选择不保存则会直接退出，选择取消则会什么也不处理，直接点右上角的取消也会进行什么也不处理的操作；若没有修改进行之前选择的操作

- 文件的插入

  如何在指定的位置中选择系统文件进行导入，这里的难点在于如何确定插入的位置，如何保证插入后原本文字的样式内容不变，插入文本正常显示等；

  因为我们采用的文本格式使用了html语言的框架，因此导入之后势必会出现重复的成分，比如<head></head><body></body>等均为重复代码，体现文本样式的只有<body></body>中间包含的 文本块，所以程序对html代码使用了筛选切割，将 的信息提取了出来，并根据光标在文本编辑器中的位置，确定了需要插入信息的位置，最后实现了插入。

  关于确定光标的位置也是一个关键点，程序通过切割QString找到了光标前的一个字符，再记录文本从开始到光标之前的位置字符出现的次数n，之前，在html代码中第n次出现指定字符的位置，并向后移动，直到 之后，此时即为正确的插入文本的光标位置

- 字体与文字颜色

  字体与文字颜色的设置难点并不在这两个函数本身，而在如何让字体与文字颜色的变化只发生在选定的范围中，为完成这项判断我们对目前是否有选择区域先做判断，若有选区则让字体与文字颜色函数正常作用于选区，若当前无选区，则选择光标目前所在处的字符（WordUnderCursor），这样就能使字体与文字颜色的设置操作发生在使用者想要的区域，具体代码见于“开发需求的代码实现”部分

- 右键菜单与UI设计
  此部分主要在于将文本框的鼠标右键事件进行修改，并让自定义菜单中的各项与槽函数进行联系，UI设计则需要做好ToolBar的设计，如“工具栏”中各按键的提示文本等，在创键（.qrc）文件时也需要主义避免嵌套导致的无法正确找到图片导致显示丢失的问题

- 撤销与重做

  在执行撤销操作时，读取保存的最近一次文本编辑框中的内容并加载到文本框中，并将当下文本编辑器中显示的内容再次保存，以便于下一次执行重做功能时再次恢复之前的文本框中的内容。为此，我们构建了两个QStack类型的栈用于保存数据信息。我们建立两个存储QString类型信息的栈tempp和store，在每次打开文本的时候需要将栈初始化。

  撤销过程目的在于实现一个新的结构存储每次文本框中的内容。每次文本框的内容更新的时候，调用一个函数来存储当前文本编辑器中的内容，因此我们首先重写了这个函数并作为槽函数。我们继承了textedit类中的textchanged()函数，并且自己定义了一个槽函数，槽函数的定义如下，可以在每次文本更改时更新堆栈

  我们考虑了不是每一次textchanged函数的调用都是由用户引起的，也可能由textedit的toplaintext引起。因此，我们需要给on_textedit_textchanged()函数加锁，每一次undo和redo操作的时候加锁，函数结束时解锁。

- 段落格式

  在段落格式的设置中，首先要读取testEdit中的Cursor游标，然后针对游标定位的QTextBlockFormat去设置段落，这样可以确保设置选中的文本块可以被设置段落格式，而不仅仅是一行当中的某些段落被设置格式。同时，在改变格式之后要重写设置Cursor游标

- 文本编辑器的复制粘贴与系统自带复制粘贴冲突的问题

比如我在文本编辑器之外剪切或者粘贴一个文本，我必须让文本编辑器也能识别出剪切板对象上已经有内容，让它也能粘贴上，刚开始我试着运用了标志全局变量来实现，通过判断这个标志来分清是在文本编辑器里面剪切的还是在文本编辑器外面剪切的，之后我发现这个有一点Bug，因为你一旦在文本编辑器里面点击粘贴后，就会给那个标志造成一种假象，使得标志位无法被取反

开发需求的代码实现

文件的保存与导入

使用QFileDialog::getSaveFileName()让用户选择保存文本的路径，再将文本信息转换成html代码，倘若文本打开正常，则将代码写入并关闭文件，再对路径进行切割，提取*.bin的文件名，显示在窗口名称上面，最后将newFile设置为false，代表不是新的尚未拥有路径名的文件；倘若文本打开失败，则跳出弹窗，提示保存失败

```
void Texteditor_self::save(){
    // 获取文本信息
    QString file_name;
    file_name = QFileDialog::getSaveFileName(this,tr("Save
files"),"./",tr("Binary files(*.bin)"));
    QString strHtml = ui->textEdit->document()->toHtml();
    QFile file(file_name);
    if ( file.open(QFile::WriteOnly) ){
        QDataStream out(&file);
        out<<strHtml;
        file.close();

        Test = file_name;
        QStringList tmp = Test.split(QRegExp("/"));
        int i=tmp.count();
        QString str = tmp[i-1];
        setWindowTitle(str + tr(" - TextEditor_self"));
        newFile = false;
    }
    else{
        QMessageBox msgBox;
        msgBox.setText("fail to save file!");
        msgBox.exec();
    }
}
```

多态的save()函数，应用于newFile为false，已有保存路径名的情况

```
void Texteditor_self::save(QString file_name)
{
    QString strHtml = ui->textEdit->document()->toHtml();
    QFile file(file_name);
    if ( file.open(QFile::WriteOnly) ){
        QDataStream out(&file);
        out<<strHtml;
        file.close();

        Test = file_name;
        QStringList tmp = Test.split(QRegExp("/"));
        int i=tmp.count();
        QString str = tmp[i-1];
        setWindowTitle(str + tr(" - TextEditor_self"));
```

```
        newFile = false;
    }
    else{
        QMessageBox msgBox;
        msgBox.setText("fail to save file!");
        msgBox.exec();
    }
}
```

在导入功能中，首先初始化栈，再让用户选择导入文件的路径，若文件可读则直接导入html代码，并通过setText显示富文本，最后切割路径名，将*.bin显示在窗口上，进行压栈；若导入失败，则会跳出Messagebox提示

```
void Texteditor_self::LoadFile(){
    // ui->textEdit->append("Hello World!");
    initstack();//打开时初始化undostack

    QString file_name;
    file_name = QFileDialog::getOpenFileName(this,"open file","./","Binary
files(*.bin)");
    QFile file(file_name);

    if (file.open(QIODevice::ReadOnly)){
        QDataStream in(&file);

        QString strHtml;
        in>>strHtml;
        file.close();
        ui->textEdit->clear();
        ui->textEdit->setText(strHtml);

        Test = file_name;
        QStringList tmp = Test.split(QRegExp("/"));
        int i=tmp.count();
        QString str = tmp[i-1];
        setWindowTitle(str + tr(" - TextEditor_self"));
        newFile = false;
        temp.push(strHtml);
    }else{
        QMessageBox msgBox;
        msgBox.setText("fail to load file!");
        msgBox.exec();
    }
}
```

搜索与替换

搜索替换功能的槽函数，主要内容是QDialog界面的初始化

```
void Texteditor_self::searchText(){
    findDlg = new QDialog(this);
    findDlg->setWindowTitle("Search And Replace");
    findLineEdit = new QLineEdit(findDlg);
    replaceLineEdit = new QLineEdit(findDlg);

    QPushButton *btn = new QPushButton(tr("Next"), findDlg);
```

```
    QPushButton *btn1 = new QPushButton(tr("Replace This"), findDlg);
    QPushButton *btn2 = new QPushButton(tr("Replace All"), findDlg);
    QVBoxLayout *layout = new QVBoxLayout(findDlg);

    layout->addWidget(findLineEdit);
    layout->addWidget(replaceLineEdit);
    layout->addWidget(btn);
    layout->addWidget(btn1);
    layout->addWidget(btn2);
    connect(btn, &QPushButton::clicked, this, &Texteditor_self::showFindText);
    connect(btn1,&QPushButton::clicked, this,
&Texteditor_self::replaceThisText);
    connect(btn2,&QPushButton::clicked, this, &Texteditor_self::replaceAllText);
    SearchTextNum=0;
    findDlg->show();
}
```

搜索功能：记录查找的字符searchContent和文本框中的文字content，定义变量precount和count，用来是重新初始化搜索的文字被修改过或者搜索替换的功能框被关闭后重新打开的情况；之后进行判断，倘若光标没有走到尾，则将找到的文字进行高亮处理，并更新SearchTextNum以供单一替换功能的使用；若光标走到尾，则将cursor移动到start的位置，进行循环查找

```
void Texteditor_self::showFindText(){
    static QString pre;
    QString searchContent = findLineEdit->text();
    if (searchContent!=pre){
        SearchTextNum=0;qDebug()<<"Clear num already.";
    }
    pre = searchContent;
    QString content = ui->textEdit->toPlainText();
    static int precount;
    int count = content.count(searchContent);
    if (precount!=count){
        SearchTextNum=0;qDebug()<<"Clear num in count part.";
    }
    qDebug()<<"Count is "<<count;
    precount=count;
    if (!ui->textEdit->find(searchContent, QTextDocument::FindCaseSensitively))
    {
        ui->textEdit->moveCursor(QTextCursor::Start);
    }
    else{
        QPalette palette = ui->textEdit->palette();

 palette.setColor(QPalette::Highlight,palette.color(QPalette::Active,QPalette::H
ighlight));
        ui->textEdit->setPalette(palette);
        SearchTextNum = (++SearchTextNum)%(1+count);
        if (SearchTextNum==0)
            SearchTextNum++;
        qDebug()<<"num is "<<SearchTextNum;
    }
}
```

单一替换：记录查找和替换的内容，根据查找功能中记录的searchTextNum找到对应的下标位置，进行字符剪切整合，并重新显示

```
void Texteditor_self::replaceThisText(){
    QString searchContent = findLineEdit->text();
    // QString content = ui->textEdit->toPlainText();
    QString content = ui->textEdit->document()->toHtml();
    QString replaceContent = replaceLineEdit->text();

    int tmp=0;
    for (int i=0;i<=SearchTextNum-1;i++){
        tmp = content.indexOf(searchContent, searchContent.length()+tmp);
        qDebug()<<"Current tmp is:"<<tmp;
    }

    int bpos=tmp;
    if (bpos!=-1){
        QString conString=content.left(bpos);
        conString.append(replaceContent);
        conString.append(content.right(content.length()-bpos-
searchContent.length()));

        ui->textEdit->clear();
        ui->textEdit->setText(conString);
    }
    ui->textEdit->moveCursor(QTextCursor::Start);
}
```

全部替换：直接使用replace实现全局替换

```
void Texteditor_self::replaceAllText(){

    QString searchContent = findLineEdit->text();
    QString content = ui->textEdit->document()->toHtml();
    QString replaceContent = replaceLineEdit->text();

    content.replace(searchContent, replaceContent);
    ui->textEdit->clear();
    ui->textEdit->setText(content);
}
```

保存与另存为

保存功能通过newFile进行判断，确定是否需要选择路径；另存为功能默认为需要选择保存路径；

```
void Texteditor_self::SaveFile(){
    if ( newFile==true ){ // 新文件需要给一个路径和名字
        save();
    }
    else{// 老文件直接保存就行
        save(Test);
    }
}

void Texteditor_self::SaveasFile(){
    save();
}
```

文件保存与否的提醒

新建文本时需要判断是否保存

```
void Texteditor_self::newWindow(){
    if ( ui->textEdit->document()->isModified() ){
        // 自定义一个警告对话框
        QMessageBox box;
        box.setWindowTitle(tr("警告"));
        box.setIcon(QMessageBox::Warning);
        box.setText(tr(" 尚未保存，是否保存？"));
        QPushButton *yesBtn = box.addButton(tr("是(&Y)"),QMessageBox::YesRole);

        QPushButton *noBtn = box.addButton(tr("否(&N)"), QMessageBox::NoRole);
        QPushButton *cancelBtn = box.addButton(tr("取
消"),QMessageBox::RejectRole);
        box.exec();

        if (box.clickedButton() == yesBtn)
            save();
        else if (box.clickedButton() == cancelBtn){
            //
        }
        else if (box.clickedButton() == noBtn){
            pin = 0; //初始化锁
            Test = tr("新建");
            newFile = true;
            // 初始化窗口标题为文件名
            setWindowTitle(Test + tr(" - TextEditor_self"));
            ui->textEdit->clear();
        }
    }
}
```

退出文本编辑器时需要判断是否保存

```
void Texteditor_self::closeEvent(QCloseEvent *event)
{
    if ( ui->textEdit->document()->isModified() ){
        // 自定义一个警告对话框
        QMessageBox box;
        box.setWindowTitle(tr("警告"));
        box.setIcon(QMessageBox::Warning);
        box.setText(tr(" 尚未保存，是否保存？"));
        QPushButton *yesBtn = box.addButton(tr("是(&Y)"),QMessageBox::YesRole);

        QPushButton *exit = box.addButton(tr("否(&N)"), QMessageBox::NoRole);
        QPushButton *cancelBut = box.addButton(tr("取
消"),QMessageBox::RejectRole);
        box.exec();

        if (box.clickedButton() == yesBtn){
            save();
            event->accept();
            return ;
```

```
        }
        else if (box.clickedButton() == cancelBut){
            event->ignore();
            return ;
        }
        else if (box.clickedButton() == exit){
            event->accept();
        }
    }
    // event->ignore();
}
```

文件的插入

实现思路见“系统设计难点及解决”

```
void Texteditor_self::on_actionInsert_triggered(){
    QString file_name;
    file_name = QFileDialog::getOpenFileName(this,"open file","./","Binary
files(*.bin)");
    QFile file(file_name);

    if (file.open(QIODevice::ReadOnly)){
        QDataStream in(&file);

        QString strHtml;
        in>>strHtml;
        file.close();

        QTextCursor cursor=ui->textEdit->textCursor();//得到当前text的光标
        int index =  cursor.position();

        qDebug() << index;
        QString content = ui->textEdit->document()->toHtml();
        qDebug() << content<<endl;

        QString tmp;
        QString plaintxt = ui->textEdit->document()->toPlainText();
        QString prechar = plaintxt.left(index);
        prechar = prechar.right(1);
        qDebug()<<prechar;

        int totalnum = plaintxt.count(prechar);
        int precharnum = plaintxt.left(index).count(prechar);
        qDebug()<<totalnum<<"|"<<precharnum<<endl;

        int insertindex;
        for (int i=0;i<=precharnum-1;i++){
            insertindex  = content.indexOf(prechar,
prechar.length()+insertindex);
        }

        while ( QString::compare(content.left(insertindex).right(3),"/p>")!=0 ){
```

```
            insertindex++;
        }
        qDebug()<<"Current insertindex is:"<<insertindex;

        tmp = content.left(insertindex);
        int start = strHtml.indexOf("<p");
        int end = strHtml.lastIndexOf("/p>");

        strHtml = strHtml.mid(start,end);
        strHtml = strHtml.left(strHtml.length()-14);
        // strHtml = strHtml.left(start);

        qDebug()<<"start is"<<start<<"| end is"<<end<<endl;
        qDebug() << strHtml<<endl;

        tmp.append(strHtml);
        tmp.append(content.right(content.length()-insertindex));
        ui->textEdit->clear();
        ui->textEdit->setText(tmp);

        temp.push(strHtml);
    }else{
        QMessageBox msgBox;
        msgBox.setText("fail to insert!");
        msgBox.exec();
    }

}
```

删除和复制

首先的到当前的光标，通过if语句调用hasSelection函数来判断是否有选中，如果有调用deleteChar函数删除选定的内容，否则删除前一个字符，并且让光标移动到删除后的位置。

```
QTextCursor cursor=ui->textEdit->textCursor();
    if(cursor.hasSelection()){
          cursor. deleteChar ();
    }else{
        QTextCursor cursor=ui->textEdit->textCursor();//得到当前text的光标
             if(cursor.hasSelection())
 cursor.clearSelection();
             cursor.deletePreviousChar();              ui->textEdit-
>setTextCursor(cursor);
    }
```

因为复制的话和粘贴在不同的函数里面，并且粘贴需要复制功能函数的内容，所以在头文件设置一个全部变量实现共享。

```
QString originalText;
```

然后调用textCursor().selectedText()获得当前选中的文本。

```
 originalText = ui->textEdit->textCursor().selectedText();
```

粘贴和剪切

获得当前的光标，然后利用全局变量originalText获得文本框的内容，使用在光标后追加的方式插入文本。

```
QString text=ui->textEdit->toPlainText();
QTextCursor textCursor=ui->textEdit->textCursor();
textCursor.insertText(originalText);
```

获得系统剪贴板对象，然后获得剪切板内容，用QSting类型变量去获得当前文本框内容，+=表示在后面追加，这样就不会出现之前文本被覆盖的现象，然后调用setPlainText函数设置文本的内容。

```
QClipboard *clipboard = QApplication::clipboard();

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

text+=clipboard->text();

ui->textEdit->setPlainText(text);
```

右键菜单的重写

为了实现重写系统自带右键菜单与屏蔽系统自带快捷键的目的，我们继承了TextEdit类，其中文本编辑框的右键菜单是一个QWidget类中的函数mousePressEvent，为了重写这个函数，我们需要继承这个类，并在继承类中重载这个函数。

```
//部分texteditor_self.cpp
    m_contextMenu = new QMenu;
    m_contextMenu->addAction("Undo",this,SLOT(Undo()));
    m_contextMenu->addAction("Redo",this,SLOT(Redo()));
    m_contextMenu->addAction("Delete",this,SLOT(on_actionDel_triggered()));
    m_contextMenu->addAction("Cut",this,SLOT(on_actionCut_triggered()));
    m_contextMenu->addAction("Copy",this,SLOT(on_fu()));
    m_contextMenu->addAction("Paste",this,SLOT(on_tie()));
    m_contextMenu->addAction("Insert",this,SLOT(on_actionInsert_triggered()));

    void Texteditor_self::mousePressEvent(QMouseEvent *event)
{
    //确保右键点击，然后跳出菜单.
    if (event->button() == Qt::RightButton)
    {
        //在鼠标右击的地方弹出菜单
        m_contextMenu->exec(event->globalPos());
    }

    event->accept();
}
```

字体与颜色设置

```
void Texteditor_self:: ChangeFormat(const QTextCharFormat &mod)//让改变格式的操作作用于被选取的区域，如果当前没有被选取的区域则使操作作用于光标所在处的字符
{
    QTextCursor cursor =ui->textEdit->textCursor();
    if ( !cursor.hasSelection() )//判断是否有被选取区域，如果有返回true，否则返回false
           cursor.select( QTextCursor::WordUnderCursor ); //无被选取区域时，相当于选择光标目前所在处字符（WordUnderCursor）

    cursor.mergeCharFormat( mod );
    ui->textEdit->mergeCurrentCharFormat( mod );
}


void Texteditor_self::TextColor()
{
    QColor color = QColorDialog::getColor(Qt::black, this);//调用颜色操作框
    if ( color.isValid() )
    {
         QTextCharFormat mod;
         mod.setForeground( color );
         ChangeFormat( mod );
    }
}

void Texteditor_self::TextFont()
{
    bool ok;
    QFont font = QFontDialog::getFont(&ok);//调用字体操作框
    if(ok){
        QTextCharFormat mod;
        mod.setFont( font );
        ChangeFormat( mod );
    }
}
```

撤销与重做

在执行撤销操作时，读取保存的最近一次文本编辑框中的内容并加载到文本框中，并将当下文本编辑器中显示的内容再次保存，以便于下一次执行重做功能时再次恢复之前的文本框中的内容。为此，我们构建了两个QStack类型的栈用于保存数据信息。我们建立两个存储QString类型信息的栈tempp和store，在每次打开文本的时候需要将栈初始化。

```
public:
    QStack<QString> temp,store;
    int pin=0;//用于undo redo

    private slots:
        void Undo();
        void Redo();
        void on_textEdit_textChanged(); //每次更新文本的时候压入栈temp
        void initstack();
void Texteditor_self::initstack() //栈初始化
{
    temp.clear();
```

```
    store.clear();
}
```

撤销过程目的在于实现一个新的结构存储每次文本框中的内容。每次文本框的内容更新的时候，调用一个函数来存储当前文本编辑器中的内容，因此我们首先重写了这个函数并作为槽函数。我们继承了textedit类中的textchanged()函数，并且自己定义了一个槽函数，槽函数的定义如下，可以在每次文本更改时更新堆栈

```
void Texteditor_self::on_textEdit_textChanged(){//每次文本更改时更新堆栈
    if (pin==0){
        temp.push(ui->textEdit->toPlainText());
        QString strHtml = ui->textEdit->document()->toHtml();
        temp.push(strHtml);
    }
    if (temp.size()>3){
        QVBoxLayout *layout =new QVBoxLayout();
            ui->scrollArea->setLayout(layout);
            ui->scrollArea->setWidgetResizable(true);
            QTextEdit *edit1 = new QTextEdit("hello world");
            edit1-
>setSizePolicy(QSizePolicy::QSizePolicy::Preferred,QSizePolicy::Preferred);
            layout->addWidget(edit1);
    }
}
```

在实现了temp这个qstack栈之后，我们便可以新增一个store栈来实现撤销的操作，撤销的操作会将temp栈中的qtring给pop掉到store中，再通过textedit继承的toplaintext函数来显示栈顶的qtring，以达到撤销的目的，同时store中存储了被撤销的qtring，可以为重做功能提供方便，具体代码实现如下

```
void Texteditor_self::Undo(){
    if (!temp.isEmpty()){//新增限制条件
    pin++; //加锁
    store.push(temp.pop());
    store.push(temp.pop());//当前栈加入store中以供redo调用
    ui->textEdit->setText(temp.top());//显示上一个内容
    //store.push(ui->textEdit->toPlainText());//上一个内容压入store
    ui->textEdit->moveCursor(QTextCursor::End, QTextCursor::MoveAnchor);
    pin--; //解锁
    }
}
```

在重做时，我们使用了撤销操作时同样用到的store栈，pop出栈顶元素并重新压入temp栈，同时显示栈顶元素的qtring到文本框中，redo函数定义如下。我们考虑了不是每一次textchanged函数的调用都是由用户引起的，也可能由textedit的toplaintext引起。因此，我们需要给on_textedit_textchanged()函数加锁，每一次undo和redo操作的时候加锁，函数结束时解锁。

```
void Texteditor_self::Redo(){
    if (!store.isEmpty()){//新增限制条件
    pin++;
    temp.push(store.pop());
    temp.push(store.pop());
    ui->textEdit->setText(temp.top());
    //temp.push(ui->textEdit->toPlainText());
    ui->textEdit->moveCursor(QTextCursor::End, QTextCursor::MoveAnchor);
    pin--;
    }
}
```

段落格式

实现思路见“系统设计难点及解决”

左对齐

```
void Texteditor_self::TextLeft(){//左对齐
    QTextCursor cursor = ui->textEdit->textCursor();
    QTextBlockFormat textBlockFormat = cursor.blockFormat();
    textBlockFormat.setAlignment(Qt::AlignLeft);
    cursor.mergeBlockFormat(textBlockFormat);
    ui->textEdit->setTextCursor(cursor);
}
```

右对齐

```
void Texteditor_self::TextRight(){//右对齐
    QTextCursor cursor = ui->textEdit->textCursor();
    QTextBlockFormat textBlockFormat = cursor.blockFormat();
    textBlockFormat.setAlignment(Qt::AlignRight);
    cursor.mergeBlockFormat(textBlockFormat);
    ui->textEdit->setTextCursor(cursor);
}
```

居中对齐

```
void Texteditor_self::TextCenter(){//居中对齐
    QTextCursor cursor = ui->textEdit->textCursor();
    QTextBlockFormat textBlockFormat = cursor.blockFormat();
    textBlockFormat.setAlignment(Qt::AlignCenter);
    cursor.mergeBlockFormat(textBlockFormat);
    ui->textEdit->setTextCursor(cursor);
}
```

两端对齐

```
void Texteditor_self::TextJustify(){//两端对齐
    QTextCursor cursor = ui->textEdit->textCursor();
    QTextBlockFormat textBlockFormat = cursor.blockFormat();
    textBlockFormat.setAlignment(Qt::AlignJustify);
    cursor.mergeBlockFormat(textBlockFormat);
    ui->textEdit->setTextCursor(cursor);
}
```

UI设计

```
void Texteditor_self::createActions() {
    newAct = new QAction ( QIcon ( ":/images/rsc/new.png" ), tr ( "&New" ), this
);
    newAct->setShortcuts ( QKeySequence::New );
    newAct->setStatusTip ( tr ( "Create a new file" ) );
    connect ( newAct, SIGNAL ( triggered() ), this, SLOT ( newWindow() ) );
    open = new QAction ( QIcon ( ":/images/rsc/open.png" ), tr ( "&Open" ), this
);
    open->setStatusTip ( tr ( "Open a new file" ) );
    connect ( open, SIGNAL ( triggered() ), this, SLOT ( LoadFile() ) );
    Save = new QAction ( QIcon ( ":/images/rsc/save.png" ), tr ( "&Save" ), this
);
    Save->setStatusTip ( tr ( "Save a new file" ) );
    connect ( Save, SIGNAL ( triggered() ), this, SLOT ( SaveFile() ) );
    saveAs = new QAction ( QIcon ( ":/images/rsc/saveas.png" ),tr ( "&SaveAs" ),
this );
    saveAs->setShortcuts ( QKeySequence::SaveAs );
    saveAs->setStatusTip ( tr ( "Save as a new file" ) );
    connect ( saveAs, SIGNAL ( triggered() ), this, SLOT ( test() ) );
    cut = new QAction ( QIcon ( ":/images/rsc/cut.png" ), tr ( "&Cut" ), this );
    cut->setShortcuts ( QKeySequence::Cut );
    cut->setStatusTip ( tr ( "Cut" ) );
    connect( cut, SIGNAL ( triggered() ), this, SLOT ( on_actionCut_triggered()
) );
    copy = new QAction ( QIcon ( ":/images/rsc/copy.png" ), tr ( "&Copy" ), this
);
    copy->setShortcuts ( QKeySequence::Copy );
    copy->setStatusTip ( tr ( "Copy" ) );
    connect( copy, SIGNAL ( triggered() ), this, SLOT ( on_fu() ) );
    paste = new QAction ( QIcon ( ":/images/rsc/paste.png" ), tr ( "&Paste" ),
this );
    paste->setShortcuts ( QKeySequence::Paste );
    paste->setStatusTip ( tr ( "Paste" ) );
    connect( paste, SIGNAL ( triggered() ), this, SLOT ( on_tie() ) );
    find = new QAction ( QIcon ( ":/images/rsc/find.png" ), tr (
"&Search/Replace" ), this );
    find->setStatusTip ( tr ( "Search/Replace" ) );
    connect( find, SIGNAL ( triggered() ), this, SLOT ( searchText() ) );
    undo = new QAction ( QIcon ( ":/images/rsc/undo.png" ), tr ( "&Undo" ), this
);
    undo->setShortcuts ( QKeySequence::Undo );
    undo->setStatusTip ( tr ( "Undo" ) );
    connect( undo, SIGNAL ( triggered() ), this, SLOT ( Undo() ) );
    redo = new QAction ( QIcon ( ":/images/rsc/redo.png" ), tr ( "&Redo" ), this
);
    redo->setShortcuts ( QKeySequence::Redo );
```

```
    redo->setStatusTip ( tr ( "Redo" ) );
    connect( redo, SIGNAL ( triggered() ), this, SLOT ( Redo() ) );
    left = new QAction ( QIcon ( ":/images/rsc/left.png" ), tr ( "&Left" ), this
);
    left->setStatusTip ( tr ( "Left" ) );
    connect( left, SIGNAL ( triggered() ), this, SLOT ( TextLeft() ) );
    right = new QAction ( QIcon ( ":/images/rsc/right.png" ), tr ( "&Left" ),
this );
    right->setStatusTip ( tr ( "Left" ) );
    connect( right, SIGNAL ( triggered() ), this, SLOT ( TextRight() ) );
    middle = new QAction ( QIcon ( ":/images/rsc/middle.png" ), tr ( "&Center"
), this );
    middle->setStatusTip ( tr ( "Center" ) );
    connect( middle, SIGNAL ( triggered() ), this, SLOT ( TextCenter() ) );
    just = new QAction ( QIcon ( ":/images/rsc/just.png" ), tr ( "&Just" ), this
);
    just->setStatusTip ( tr ( "Center" ) );
    connect( just, SIGNAL ( triggered() ), this, SLOT ( TextJustify() ) );
    font = new QAction ( QIcon ( ":/images/rsc/font.png" ), tr ( "&Font" ), this
);
    font->setStatusTip ( tr ( "Font" ) );
    connect( font, SIGNAL ( triggered() ), this, SLOT ( TextFont() ) );
    color = new QAction ( QIcon ( ":/images/rsc/color.png" ), tr ( "&Color" ),
this );
    color->setStatusTip ( tr ( "Color" ) );
    connect( color, SIGNAL ( triggered() ), this, SLOT ( TextColor() ) );
    insert = new QAction ( QIcon ( ":/images/rsc/insert.png" ), tr ( "&Insert"
), this );
    insert->setStatusTip ( tr ( "Insert" ) );
    connect( insert, SIGNAL ( triggered() ), this, SLOT (
on_actionInsert_triggered() ) );
}

void Texteditor_self::createToolBars()
{
    ui->mainToolBar->addAction ( open );
    ui->mainToolBar->addAction ( newAct );
    ui->mainToolBar->addAction ( Save );
    ui->mainToolBar->addAction ( saveAs );
    editToolBar = addToolBar ( tr ( "Edit" ) );
    editToolBar->addAction ( cut );
    editToolBar->addAction ( copy );
    editToolBar->addAction ( paste );
    editToolBar->addAction ( insert );
    editToolBar->addAction ( find );
    editToolBar->addAction ( undo );
    editToolBar->addAction ( redo );
    fmtToolBar = addToolBar ( tr ( "Format" ) );
    fmtToolBar->addAction( left );
    fmtToolBar->addAction( right );
    fmtToolBar->addAction( middle );
    fmtToolBar->addAction( just );
    fmtToolBar->addAction( font );
    fmtToolBar->addAction( color );
}
```

程序使用说明

File（文件）

① New（新建）：创建一篇空白文档，从下拉菜单“File”或“工具栏”中创建

② Open（打开）:打开二进制（.bin）文件从下拉菜单“Edit”或“工具栏”中打开；或将二进制（.bin）文件拖入程序界面打开或应用程序图标上打开；或右键单击文本文件，在“打开方式”中选择本程序打开

③ Save（保存）：保存文档，从“工具栏”或“文件下拉菜单”中打开

④ SaveAs（另存为）：保存文件副本，在不同位置或以不同文件名保存文档，从“工具栏”或“文件下拉菜单”中另存

Edit（编辑）

① Undo（撤销）：撤销前一步所进行的操作，从下拉菜单“Edit”或“工具栏”中撤销

② Redo（重做）：重做前一步所撤销的操作，从下拉菜单“Edit” 或“工具栏”中重做

③ Cut（剪切）：复制并删除选定字符（串），从下拉菜单“Edit” 或“工具栏”中剪切

④ Copy（复制）：复制选定字符（串），从下拉菜单“Edit” 或“工具栏”中复制

⑤ Paste（粘贴）：对粘贴内容进行粘贴，从下拉菜单“Edit” 或“工具栏”中粘贴

⑥ Search/Replace（查找/替换）：输入查找内容（和替换内容），可从光标位置逐个查找（或替换）相应内容，也可一次性全部替换掉相应内容，从下拉菜单“Edit”或“工具栏”中执行

⑦ Insert（插入）：选择一个二进制（.bin）文件，将该文件中的内容直接插入到当前正在编辑的文本框中，从下拉菜单“Edit” 或“工具栏”中插入

⑧ Delete（删除）：删除选定字符（串），从下拉菜单“Edit” 或“工具栏”中删除

Format（格式）

① Font（字体）：设置字体、字形及字的大小，从下拉菜单“Format”或“工具栏”中设置字体设置字体

② Color（颜色）：设置字的颜色, 从下拉菜单“Format”或“工具栏”中设置颜色

③ 段落设置：设置段落格式，格式有Left（左对齐）、Right（右对齐）、Center（居中）和两端对齐（Justify），从下拉菜单“Format”或“工具栏”中设置段落格式

系统开发日志：

时间	项目进度
4.28	完成：组队，确定选题
4.29--5.10	完成：学习QT，确定分工
5.13	完成：实现了基本功能的输入和回退
5.18	完成：实现基本的redo和undo功能
5.20	完成：实现文本的读取和保存；初步的UI设计，将button的button clicked信号和保存和导入的slot绑定
5.21	完成：简单的UI设计，使用了menubar作为交互界面
5.22	问题：撤销和回退功能与导入和保存功能之间存在冲突；右键菜单功能无法使用；
5.23	完成：解决redo和undo功能与导入文本之间的问题
5.23	完成：实现删除和插入功能； 问题：删除光标所在的字符和删除选中的字段这两部分的功能没有整合
6.1	完成：实现高级功能，字体大小、颜色、种类的设置
6.2	完成：实现不带样式文本的搜索和替换功能；解决右键菜单功能无法使用的bug； 问题：文本框为空白时，undo会导致程序崩溃的问题；带格式的文本保存和读取与不带格式的文本保存和读取方式需要截然不同的处理；插入功能会导致样式的清空；搜索替换窗口会保留之前的已经关闭了的搜索替换窗口中的内容
6.3	完成：文本的排版功能；实现了带格式的文本保存和读取；解决undo引起的程序崩溃问题；解决搜索替换功能的bug
6.7	完成：实现了UI设计
6.8	完成：实现了复制粘贴功能；解决插入功能导致原有样式清空的问题； 问题：插入功能与粘贴功能有重合；退出时，选择是否保存的交互存在问题
6.9	完成：实现了文本的插入功能；实现了粘贴功能的整合；解决退出时保存窗口的交互问题
6.11	完成：代码的全部内容

总结

本次编程作业让我明白了编写大程序时成员之间相互配合的重要性，在交接时要能看懂队友的代码，理解他们的逻辑才能做到最终的成品有序不杂乱。

过程中虽然遇到许多困难，但通过与组员交流，或是网上查阅资料，都得以解决，让我感受到了Qt在带UI的程序设计中的优越之处，遗憾是本次的UI设计大多在代码中直接完成，而少用到（.ui）文件，希望之后还能有机会进一步学习如何使用Qt。

在本次的合作中，我们感受到了Qt在继承类实现所需功能的强大之处，也体会到了面向对象程序设计的基本方法和核心要点。同时，由于开发中需要继承某些类，我们也体会到了信息检索，文档阅读的重要性。

这次编程作业让我对团队开发有了新的体验和认识，要想高效合作开发好一个程序，成员间的沟通是十分重要的，遇到困难的问题可以通过成员的交流一起解决。