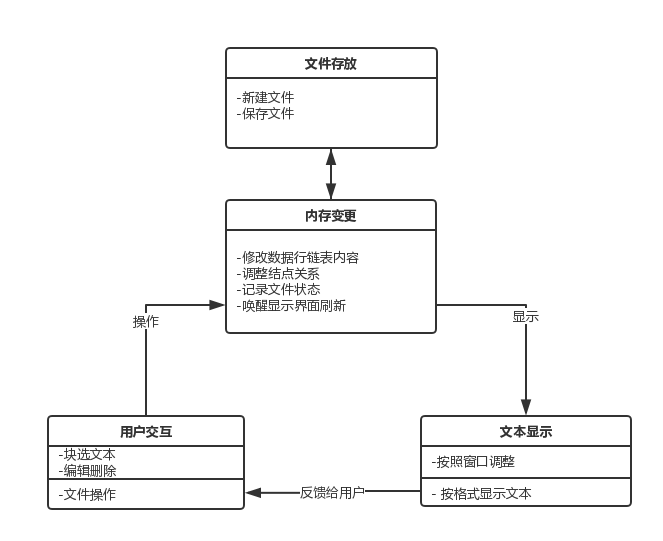
1.功能需求说明及分析

2.总体方案设计说明

2.1软件开发环境

使用QtCreator软件进行开发，主要使用Qt库函数编写用户交互部分的函数，使用C++标准库编写对内存中的文本进行存储和修改操作函数。

2.2总体结构



3.3模块划分

a. 用户交互模块：捕捉用户的编辑行为，根据行为作出相应的显示变化

b. 文件操作模块：负责外存和主存之间的数据交流

c. 内存文本模块：负责在内存中保存文本，并且对其进行增删查改操作

3.数据结构

3.1数据结构说明

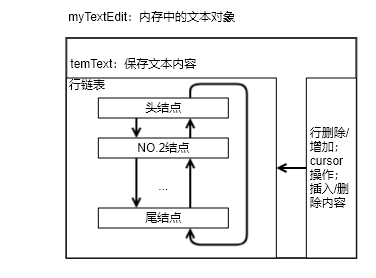
3.1.1 内存中文本的保存

a. 文本用一个特定的对象class myTextEdit保存，一部分是对文本进行修改操作的函数，另一部分保存文本内容

b. myTextEdit中提供一个接口，专门记录逻辑上的“当前光标的坐标”，几乎所有的文本操作都于该光标相关。

c. 保存文本内容class temText用于保存全体文本的内容，同时对各行链表进行链表的操作

d. 行链表对象lineheAD用于保存文本中以行为单位的内容，每行内保存一个字符串。动态申请数组对字符串进行保存，在需要更多的空间的时候对动态增加该数组的内存分配空间。行链表的整体结构为双向循环链表。特别需要指明的是：头结点的前缀链表为尾结点，尾结点的后继链表为空。如此一来可以便捷快速地移动光标，并且又可以兼容单向链表的操作。



3.2数据字典

|  |  |  |  |
| --- | --- | --- | --- |
| 对象名称 | 简要说明 | 对象成员/函数 | 功能说明 |
| myTextEdit | 文本及其编辑相关操作对象 | lineheAD\* getFirstLine(void), void setFirstLine(lineheAD\* lp) | 获取第一行行链表结点，设置第一行行链表结点 |
| lineheAD\* getNowLine()，void setNowLine(lineheAD\* lp) | 获取当前坐标所在行链表结点，设置当前行链表 |
| int getLineN() | 获取文本行数 |
| void getAxis()，void setAxis(int tR,int tC) | 输出当前光标所在坐标，设置光标位置，并且移动nowline行链表标记 |
| int getRow(),int getCol() | 返回行、列 |
| lineheAD\* axisToPtr(int row); | 用一个整形的行数获取对应的行链表结点的指针 |
| void insertStr(std::string str); | 在当前的坐标位置插入文本 |
| void delFULL(void) | 删除所有文本内容 |
| void delNL(int type); | 清空1/删除10当前行 |
| void delC(int isbackspace); | 删除当前坐标前/后的单字符 |
| std::string cutBlock(int r2,int c2); | 剪切操作，将从当前坐标到指定坐标为止的文本从内存中删除，并返回给用户 |
| void delBlock(int r2,int c2); | 将从当前坐标到指定坐标为止的文本删除， |
| std::string copyBlock(int r1,int c1,int r2,int c2); | 复制，获取指定区间的文本并返回 |
| void moveArray(lineheAD\* t,int index,int n); | 用移动数组内容的方式，删除指定的行链表中的块内容 |
| lineheAD\* firstLine; | 保存首行行链表结点 |
| lineheAD\* nowLine; | 记录当前行行链表 |
| temText\* WHOLETEXT; | 用于保存整个文本，行链表之间是一个双向单循环链表 |
| Int row, int col | 从1开始表示的行、列 |
| temText | 文本对象在内存中的存储，以lineheAD行链表结点为基本单位 | lineheAD\* newLine(void); | 在文本的最后插入一个行链表 |
| lineheAD\* insertLine(lineheAD\* preLine,CTYPE\* str,int n); | 在指定行的后面插入新的一行，同时可以指定初始化的字符串内容 |
| int linecounter; | 记录行数 |
| lineheAD \* firstLine; | 保存第一行行链表 |
| bool changeMode; | 记录模式是否自动换行 |
| bool deleteLine(lineheAD\* target); | 删除指定的行链表 |
| lineheAD | 文本中的行对象，行链表结点为双向链表，头结点的前缀链表为尾结点，尾结点的后继链表为空 | lineheAD \* getNext(void); void setNext(lineheAD\* np) | 获取后继结点；设置后继结点 |
| lineheAD\* getPre(void);  void setPre(lineheAD\* np) | 获取前缀结点；设置前缀结点 |
| int getSize(void); void setSize(int n); | 获取行链表内字符串的长度；设置当前行链表内字符串的长度 |
| CTYPE \*chs; | CTYPE为宏定义的字符类型，目前设置为char型， chs为字符类型的数组指针 |
| int arySIze; | 记录当前行的字符串数组大小（已经申请的空间大小） |
| int strSize; | 保存行内字符串的长度 |
| lineheAD \* next; | 保存行链表的后继结点指针 |
| lineheAD \* pre; | 保存行链表的前缀结点指针 |
| searchResult | 搜索对象，包含了替换功能 | searchResult(myTextEdit\* TEXT,std::string toFind,bool IC); | 重载的构造函数：指明搜索的文本对象（搜索范围），规定搜索的内容，记录是否忽略大小写 |
| int getNumber(void) | 获取搜索结果的数量 |
| int\* getSpecificOne(int INDEX) | 获取指定序号的搜索结果的坐标 |
| std::vector<int\*> allResults; | 保存所有的搜索结果，每个搜索结果用int[3]保存，内容为行序号，开始列坐标，结束列坐标 |
| bool ignoreCap; | 记录当前搜索是否忽略大小写 |
| void replace(int N,std::string newstr); | 将指定序号的搜索结果替换为新的指定字符串 |
| void search(void); | 搜索函数，获取所有的搜索结果坐标并保存 |
| nt Index\_KMP(std::string S,std::string T,int pos,const int\* const next, bool ignCap); | KMP搜索算法，使用时对某指定行链表中的字符串对指定目标字符串进行搜索 |
| void getNextVal(std::string substr,int\* next); | 计算KMP算法中需要的子串nextvalue数组 |
| myTextEdit\* TEXT; | 记录搜索的文本对象 |
| MainWindow | 主窗口，包含一系列组件和子窗口、动作和对应槽函数 | MainWindow(QWidget \*parent = 0);  ~MainWindow(); | 创建窗口界面和组件 |
| QTextEdit \* getTextEdit()  {return this->textEdit;}; | 在子窗口函数中被调用 |
| void on\_actionopen\_triggered();  void on\_actionnew\_triggered();  void on\_actionsave\_triggered();  void on\_actionfind\_triggered();  void on\_actionreplace\_triggered();  void on\_actioncut\_triggered();  void on\_actionpaste\_triggered();  void on\_actioncopy\_triggered(); | 槽函数，与对应信号连接 |
| bool eventFilter(QObject \*obj, QEvent \*event); | 事件过滤器重载，watched代表被监控的组件，event代表要转发的事件 |
| void setTwoEnd(int & r,int & c); | 修正块首/块尾，r返回块尾行数，c返回块尾列数 |
| void flush();  void correctEditCursor(int row,int col); | 刷新函数，反应内存；指针修正函数，在刷新后正确定位当前光标坐标 |
| void ShowTextRowCol(); | 在状态栏打印行列/字数信息 |
| QAction \* openAction;  QAction \* saveAction;  QAction \* newAction;  QAction \* cutAction;  QAction \* pasteAction;  QAction \* copyAction;  QAction \* findAction;  QAction \* replaceAction;  QLabel \* position; | 声明组件 |
| QTextEdit \*textEdit;  QTextCursor tcursor; | 文本编辑区；光标 |
| FindWindow \* findWin;  ReplaceWindow \* replaceWin; | 声明子窗口；用于进行搜索操作和替换操作。 |
| myTextEdit textBody; | 内存中文本及其编辑相关操作对象 |
| std::string addr;  std::string clipboard; | 打开文件的路径；临时剪贴板 |
| bool isUP=false;  bool flushFlag=1; | 大小写变量；刷新变量 |
|  |  |  |  |
|  |  |  |  |
|  |  |  |  |
|  |  |  |  |

4.各模块设计说明（算法思想、算法、特点及与其它模块的关系等）

**4.1 用户交互模块(mainwindow.h/findwindow.h/replacewindow.h)：**

显示窗口界面；捕捉用户的编辑行为，根据行为作出相应的显示变化

4.1.1 模块特点和其他模块的关系

本模块作为前端的交互部分，负责界面的显示与对用户编辑行为的处理，在总体结构中处于最表层。接收到用户的编辑行为，及时作出相应可见光标、文本选区的显示变化，并通过文本编辑函数接口与内存文本模块对接，通过刷新函数将改变后的内存打印在文本编辑框中。在此基础上，使QTextEdit控件仅起显示控件与媒介功能，通过事件过滤器接收相应事件，作自定义内存的后续编辑动作。

4.1.2 界面设计算法 MainWindow(QWidget \*parent = 0);

a. 设置标题，创建菜单栏、工具栏、状态栏、编辑区；设置字体、文本格式等。

b. 定义open/new/save/find/replace/copy/cut/paste等QAction，添加在相应控件中，并在菜单栏、工具栏、相应图标、快捷操作、输出状态信息之间连接信号和槽函数。定义搜索、替换子窗口。

c. 安装事件过滤器（见重载事件过滤器算法）。

d. 光标行列与字数信息显示算法 void ShowTextRowCol()：显示当前光标后一位的行列数，有选区时显示选中字符数，否则显示光标前方字符数。

4.1.3 文本编辑槽函数算法 void on\_actioncut\_triggered();

                                        void on\_actionpaste\_triggered();

                                        void on\_actioncopy\_triggered();

a. 剪切槽函数：若已有选区，调用块剪切算法std::string cutBlock(int r2,int c2)，将字符串存在clipboard中并刷新 。

b. 复制槽函数：若已有选区，调用块拷贝算法，将字符串存在clipboard中，并保持该选区不变。

 c. 粘贴槽函数：若已有选区，调用块删除算法与字符串插入算法void insertStr(std::string str)，将clickboard中的字符串插入在光标位置，调用刷新算法。

4.1.4 重载事件过滤器算法 bool eventFilter(QObject \*obj, QEvent \*event);

a. 通常情况下，当一个事件发生的时候，Qt 通过构造一个合适的 QEvent 子类对象来表示事件的发生，然后将该事件对象发送给特定的 QObject 对象( 或其子类实例对象)，通过调用该 QObject 的 event() 函数。event()函数并不直接处理事件，而是将这些事件对象按照它们不同的类型，分发给不同的事件处理器(event handler)，这些事件处理器是 protected virtual 的，需要继承已有类。

而事件过滤器的功能是在事件分发给不同event handler前作过滤，对象使用installEventFilter()注册后，用于目标对象的所有事件都会首先发送给这个监视对象的eventFilter()函数，在目标对象处理之前得以去处理事件。过滤器的QObject::eventFilter()实现被调用，它可以接受或丢弃过滤，也可容许或拒绝进一步去处理事件。相比于重载各事件函数/QObject::event()等方法，此处采用事件过滤器最符合任务需求且具有维护性。

b. 设置一个event filter有两个步骤：在目标对象上调用installEventFilter()，将监测对象注册到目标对象上；在监测对象的eventFilter()方法里处理目标对象的事件。

安装事件过滤器：textEdit->installEventFilter(this);  //monitoredObj->installEventFilter(filterObj);通过调用过滤器对象的 QObject::installEventFilter() 函数， 为目标对象设置一个 event filter, 就可在过滤器对象的QObject::eventFilter() 函数中处理发往目标对象的事件。重载事件过滤器：bool MainWindow::eventFilter(QObject \*obj, QEvent \*event);  //bool QObject::eventFilter (QObject \* watched, QEvent \* event); watched代表被监控的组件，event代表要转发的事件；返回true，表示该事件也被过滤掉(处理)，无需再转发了；返回false，则正常转发给watched；else返回，return QMainWindow::eventFilter(obj, event)。

c. 首先判断监视的组件是否为所选目标（textedit），然后分类event->type()，若是KeyPress事件, 把它强制转换为QKeyEvent。主要分为键盘移动光标操作、键盘块选操作、键盘删除/键入操作、键盘快捷键操作、鼠标移动光标操作。

在键盘/鼠标移动光标操作中，利用QTextedit作tcursor=textEdit->textCursor()，再将tcursor的坐标位置反馈给内存以修改内存光标。由于鼠标点击事件再QTextedit中触发了QInputMethodQueryEvent，为避免刷新函数flush()中触发该事件修改光标位置，设置flushflag变量。在键盘移动光标基础上，使用KeepAnchor模式实现键盘块选。

在作键盘删除时，判断是否有块选，分别调用与内存文本模块对接的相应文本编辑函数（单字符串删除算法void delC(int isbackspace)或块删除算法void delBlock(int r2,int c2)），修改内存；键入字符时，判断是否有块选，有则作块删除再用（单）字符串插入算法void insertStr(std::string str)，若为字母，通过isUP变量判断Key\_CapsLock当前是否有效。

4.1.5 刷新算法 void flush();

a. 每次内存做改变后进行刷新，将新的内存全部打印在文本编辑区：设置flushflag变量=0，防止触发QInputMethodQueryEvent中的setAxis，后清空编辑区再逐行由内存读出。结束后设置flushflag变量=1，并调用修正指针算法。

b. 修正指针算法 void correctEditCursor(int row,int col)：在刷新结束后将显示光标重新定位到内存光标所处位置。

4.1.6 修正块尾算法 void setTwoEnd(int & r,int & c);

a. 对已做块选做修正，将内存光标定位在块首，返回块尾坐标信息。

b. 该算法有利于与内存文本模块中的函数接口统一，以实现块操作，应用在剪切、复制、块删除等。

4.1.7查找界面 class : FindWindow

构造思想：继承于QDialog，父窗口为MainWindow，因此FindWindow可操作父窗口的显示界面，方便我们的选词高亮和进行下一个选词的操作。

功能：输入查询字符串，行内查询下一个结果，设置是否忽略大小写。

a.在主要搜索函数showFindText中,我们先检查是否已有搜索结果，如果没有的话开始第一次搜索，将待搜索字符和大小写模式传入底层函数即可得到所有搜索结果的结构体，若有，则只需要遍历结果即可。排除掉输入串为空的情况后，我们调用自定义内存底层函数进行搜索，抛出空结果情况，在非空结果中开始循环的遍历过程。输入字符串一旦变化，搜索结果就变为0，需要重新搜索。

b.下拉选框可以选择是否忽略大小写，作为参数传入底层参数。

4.1.8替换界面 class：

构造思想：继承于QDialog，父窗口为MainWindow，因此FindWindow可操作父窗口的显示界面，方便我们的选词高亮和进行下一个选词，以及替换的操作。

功能：对输入的字符串进行选择，在选中的某板块中点击替换，替换为输入的替换字符串

1. 搜索功能与查找界面几乎一致，差别在于替换的部分。替换仅限于搜索结果不为空，而且没有被全部替换完的情况。可以一直替换到所有搜索结果都被替换完的情况。输入字符串一旦变化，搜索结果就变为0，需要重新搜索。
2. 每次替换之后我们都处理搜索结果，防止坐标没变化的错误。

**4.2 文件操作模块(mainwindow.h)：**负责外存和主存之间的数据交流

4.2.1 模块特点和其他模块的关系

本模块负责处理磁盘文件与内存文本的转换，与内存文本模块对接，并作为槽函数在用户交互模块中在信号受到激励时被调用。

4.2.2 文件槽函数算法 void on\_actionopen\_triggered();

                                  void on\_actionnew\_triggered();

                                  void on\_actionsave\_triggered();

a. 文件新建槽函数：判断当前是否已有打开文件，有则弹出QMessageBox询问是否保存当前文件，是则调用文件保存槽函数并清空文件地址。调用全文删除算法void delNL(int type)清空文本编辑区。

b. 文件打开槽函数：前半部分与文件新建槽函数相似，后判断是否成功打开文件，否则弹出警告；成功得到新文件路径后，从磁盘调用到内存中，再由内存逐行显示到编辑区中。

c. 文件保存槽函数：若新建的文件无原地址，打开系统文件夹，得到路径。

**4.3 内存文本模块(back.h)**：负责在内存中保存文本，并且对其进行增删查改操作

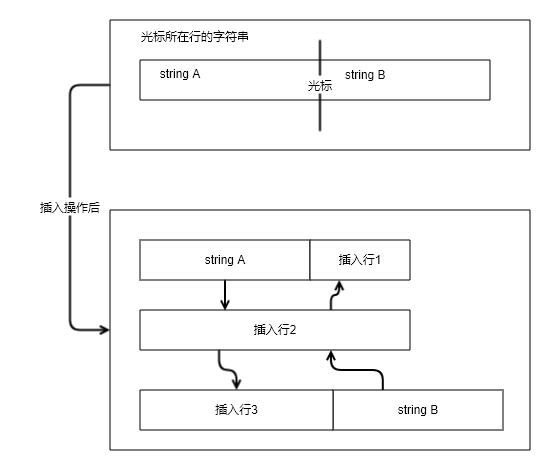
4.3.1模块特点和其他模块的关系

本模块负责在内存中的操作，是处于用户交互和文件操作模块的中间层。由于用户操作交互反馈的即时性需求，所以本模块中实现的各种修改，即逻辑上与文本显示变化有关联的部分都必须满足快速操作、快速变更的需求，使本模块中的程序运行对于用户而言是透明的，用户只需要关注前端的交互部分。同时，文件操作模块需要将内存中的文本内容以通用的形式保存到本地，所以要为文件操作模块提供方便获取文本内容的接口。

4.3.2 字符串插入算法 void insertStr(std::string str);

a. 字符串的插入算法的作用是在指定的光标位置插入指定的字符串str，本算法涵盖了各种插入字符串的情况：插入光标位置所在行的数组分配内存空间充足/不充足的情况；插入的字符串中为单字符/块选的情况；插入的字符串若为块选，是否含有跨行的内容

b.本算法的实现目标是涵盖以上所有的情况，故首先将所在行光标以后的所有字符（下图 string B）保存起来，在实现完插入操作以后，不论是否有新增的行，都在插入的块后将原来行链表中后半部分的字符串再次接上。



c.对插入的字符串（用std::string保存）判断是否有‘\n’换行符，如果有换行符就调文本存储结构中的新增行方法，增添一行。

d.本算法中的拼接直接对字符串数组进行赋值操作，如果数组内存不足，则进行动态再分配。

4.3.3 单字符串删除算法void delC(int isbackspace);

a. 单字符删除方法需要实现两种类型，一种是向前删除，另一种是向后删除，故需要接口使用者指明删除类型

b. 一般情况下的单字符删除等价于对单个行链表内的字符串数组的单个元素的删除操作，这里调用自定义的封装好的void moveArray(lineheAD\* t,int index,int n);方法，对当前行指定位置进行删除单字符。在这里，不论是向前删除还是向后删除的操作都是使用相同的删除逻辑，唯一的区别的向前删除的时候调用该函数需要指定从前一位开始。

c. 特殊的情况有两种：向前删除时光标在行的最前方，向后删除的时候光标处于当前行链表字符串的最后位置。在这两种情况需要将当前行的字符串与前一行/后一行的字符串连接起来。在本算法中，根据lineheAD中提供的成员可以判断光标位置是否在字符串尾部。如果需要进行特殊情况的操作，则将当前行的字符串保存起来，并且将当前的行链表删除，删除后再使用前文提到的void insertStr(std::string str);将临时保存起来的字符串和相邻行的字符串进行拼接。

4.3.4 行删除和全文删除算法void delNL(int type);

a.行删除的逻辑一：将本行中的内容清除，但仍占用一行，直接将当前行行链表中的字符串大小标记为0即可，不需要对内存进行修改，在逻辑上删除。

b.行删除的逻辑二：删除整个行链表，释放内存

c.当需要将所有的文本删除时，只需要利用链表遍历的方式将所有的链表结点delet即可。

4.3.5 块剪切算法std::string cutBlock(int r2,int c2);

a.块的剪切算法逻辑上将块选的内容从文本中删除，并且将刚刚选中的文本临时保存起来，故剪切算法涵盖了对文本中的块进行删除和对选中块的复制（获取）操作。

b.复制部分std::string copyBlock(int r1,int c1,int r2,int c2)：从起始坐标开始，遍历行链表结点，获取结点中的字符串内容，添加到返回结果的string中。需要注意的是，第一行只复制c1之后的字符，最后一行只复制c2之前的内容。

c.删除部分void delBlock(int r2,int c2)：将当前坐标到指定坐标的文本删除。将c2之后的字符串保存起来，如果是在同一行链表中，则直接调用moveArray将数组中起始列坐标后的内容删除，删除后将临时保存的字符串再次插入。如果需要进行跨行删除，则需要将中间包含的行释放掉，在删除行链表的时候必须注意是双向链表，要将前驱和后继的指针正确修改。

4.3.6 搜索算法 void search(void);

a.搜索算法的底层使用KMP算法，KMP算法是将目标匹配子串提前进行处理，查看是否有重复的地方，如果有重复的地方则在匹配的时候快速跳过已经比较过的部分。

b.对全文本的每一个行链表使用KMP算法，在一行的搜索中，如果获取了一个搜索结果，则先将搜索结果保存，保存后再次在该行中修改起始坐标并使用KMP算法直至搜索完毕。

c.如果有忽略大小写的需求，则将在KMP算法中添加判断，如果字符的capital character 相同则视为相等并和原问题的处理方式一致。

5. 范例执行结果及测试情况说明

文件打开、关闭、新建、上传、下载、强行关闭

文字输入：字母，数字，标点，换行

块选：选中，拷贝，复制，粘贴（快捷键，鼠标）

搜索：查询，替换（快捷键，鼠标）

排版：字体，大小，加粗，斜体，下划线

6. 评价和改进意见

7.用户使用说明