高级设计意图分析：

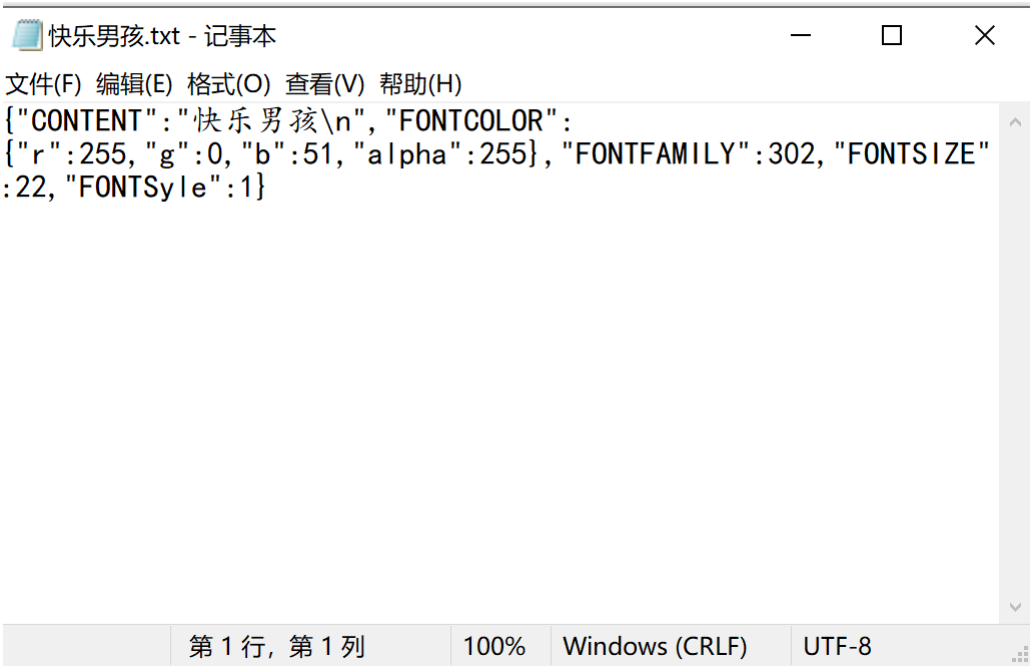
一些好的设计实现：

这里在提一提我笔记实现的一些设计吧，正好第二次代码没写完缺了不少内容。

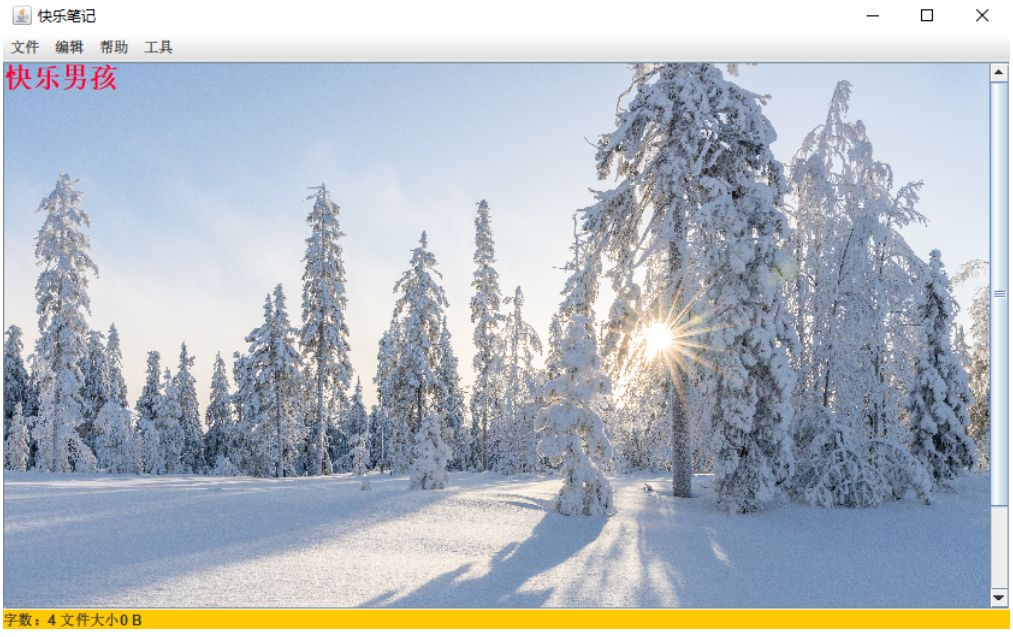
保存和读取：

在这个笔记实现中，借助fastjson很好的实现了文件的创建、读取和保存，并且这些过程中文件的排版信息是一并保存和读取的。效果如下图：

保存文件如下图所示：



笔记显示如下图所示：



这里主要的地方在于使用fastJson将笔记的内容（包括排版信息）传递给一个String 字符，之后简单的通过使用java.io提供的读写文件的方法即可完成读取和保存。

首先下面是page类的属性定义，@JSONField是fastjson提供的供给该类各个属性做注解的。

```
@JSONField(name = "PAGEID")
int his_index;
// 存放所有字符的缓存器
@JSONField(name = "CONTENT")
private String mainBuffer;
// 排版信息
@JSONField(name = "FONTFAMILY")
int fontFamily;
@JSONField(name = "FONTCOLOR")
Color fontColor;
@JSONField(name = "FONTSyle")
int fontStyle;
@JSONField(name = "FONTSIZE")
int fontSize;
```

读取文件：

```
open_page = JSON.parseObject(contend, page.class);
```

读取到文件之后再将其写入到 `notefram` 中的page中，同时由page页更新笔记（使用 `loadpage` 方法）。

保存文件：

```
String content = JSON.toJSONString(save_page);
```

类似vim模式的快捷键设计

对于模仿vim类快捷键，这里我实现的不多，是最基础的几个：在机器模式下的 `h`，`k`，`l`，`j`，`d`，`dd`，`y`，`yy`，`p`，`d`. 编辑模式则没有增加其他的快捷键。

这里的模式切换，是通过设置文本区的可编辑性为 `false` 实现，转为编辑模式再改为 `true` 即可。

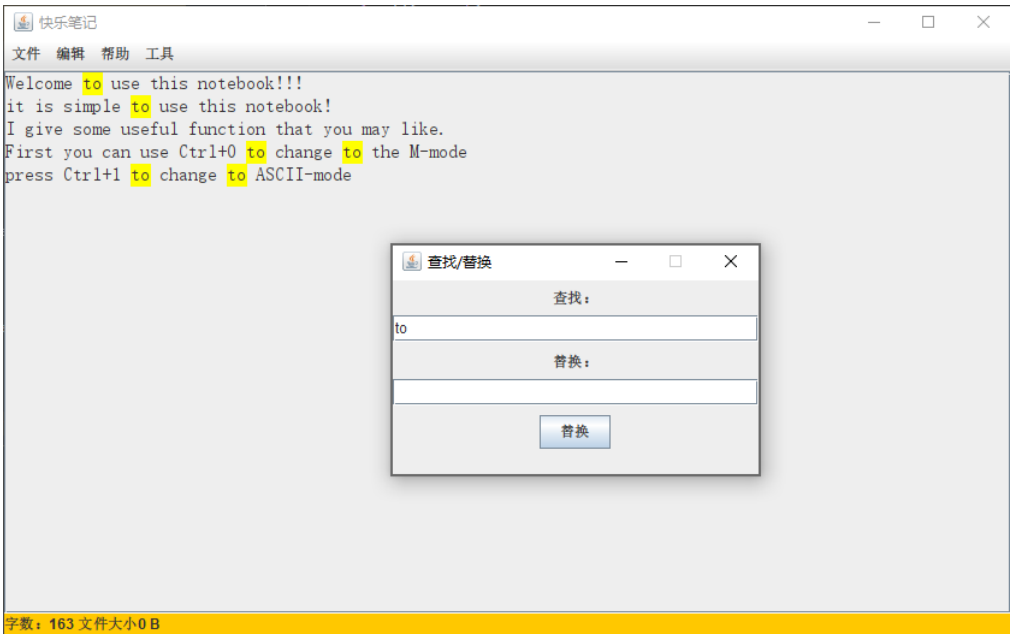
在上下左右移动的键位设置时，使用了java提供的 `Robot` 模拟上下左右键实现（之前还通过计算当前行数、列数计算偏移来实现移动，很复杂）。

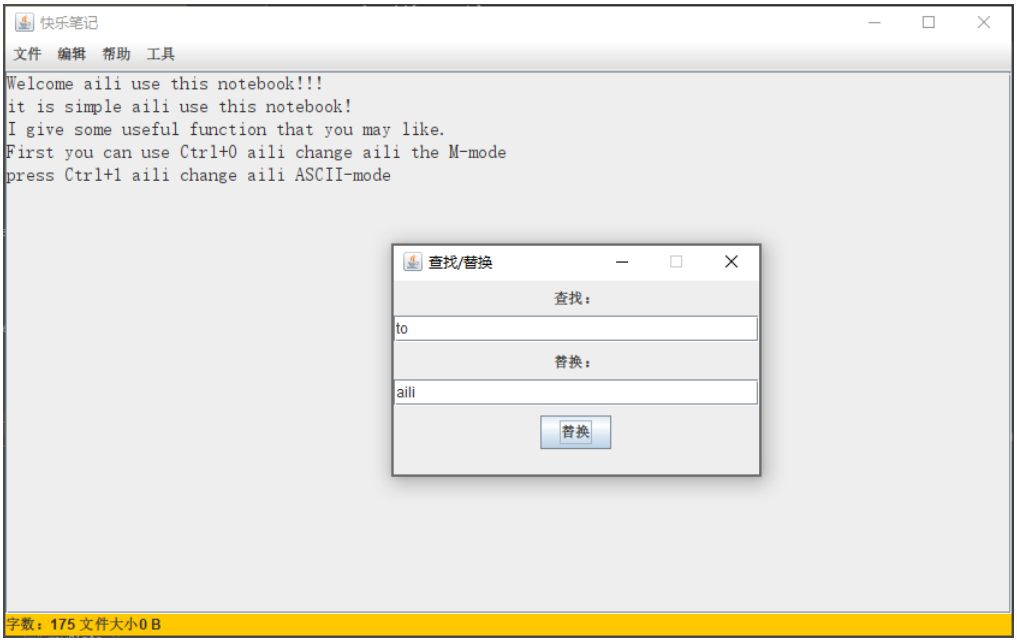
在需要写入文本区的键位实现中，需要先设置文本去可写，经过快捷键相应的操作后再设置不可写即可。(简单拿出一个展示一下代码)

```
else if(e.getKeyCode() == KeyEvent.VK_P){
                        jta.setEditable(true);
                        System.out.println(cpy_line);
                        if(cpy_line) {
                            jta.setCaretPosition(line_end);
                        }
                        jta.paste();
                        jta.setEditable(false);
                    }
```

查找和替换实现

先上效果图：





这里的难点在于搜索的时候能够实时的高亮对应的结果，当然是使用addDocumentListener监听文本域内容是否有增有减，再使用java自己提供的方法String.contain检查是否存在匹配字符串，有匹配再使用了String.indexof(String)方法找到匹配的索引。最后使用java提供的Highlighter高亮该索引匹配字符串长度的内容（循环找到所有匹配项）。

核心代码如下：

```
 while (ntf.getJta().getText().substring(sum_i).contains(source) &&
source.length() != 0) {
                    System.out.println(ntf.getJta().getText().substring(sum_i));
                    index =
ntf.getJta().getText().substring(sum_i).indexOf(source);
                    System.out.println(sum_i);
                    try {
                        highlighter.addHighlight(sum_i+index, sum_i +index+
source.length(), p);
                    } catch (BadLocationException ex) {
                        ex.printStackTrace();
                    }
                    sum_i = sum_i + index + source.length();
                }
```

替换较为简单，使用java提供的replaceall方法即可全部一起替换完成。

单例模式

在这次的笔记中，在多个类中使用了单例模式，这个模式带给我最大的高出莫过于对于每个类更准确，容易的控制和操作了。

使用的地方如下：

Class IO（即打开、保存文件的类）：

```
private static IO theIO = new IO();
...
 public static IO getInstance() {return theIO;}
```

新建文件、打开文件和保存文件都只需要一个IO对象存在即可，使用单例模式。

Class shoutcutkey(特别的块键键控制类)：

```
private static shoutcutKey my_shoutcutKey = new shoutcutKey();
...
public static shoutcutKey getInstance(){return my_shoutcutKey;}
```

同理，块键键也是全局是需要一个即可，使用单例模式。

剩下的都类似，有最大的类noteFrame(用于展示整个笔记页) 和 main类 myNote。

总结：

除此之外，我的笔记实现中没有特别使用其他模式或者高级设计意图。都以一个需要什么就加什么的路在做，并且每次想要加功能只是简单的增加需要的类，但是这样做灵活性太低，把时间花费到了功能的成功实现上而忽略了程序的可扩展性和需求变化后的修改问题，这大大增加了修改功能，扩展程序需要花费的成本。

整个代码写下来还是没有明确的规划在写，哪怕在最开始做好了需求、功能分析，画好了大概的类图和用例图，写起代码后还是陷入了我要实现什么功能，最后成为一种简单粗暴的增加代码和笔记功能了。只能说是在面向对象最边缘摸索着，在这一点上深表遗憾，现在更希望自己能够写一个具有很好的面向对象、可扩展的程序。只能说从一开始就有点误入歧途了，总想着做好什么什么功能了。

备注：

关于笔记，我在 `idea` 中读取文件的功能完全没问题，但是达成jar包之后功能无法正常运行，由于时间太少就没有再研究了，这里我准备上传整个文件的jar包和源代码，希望能有所用处。