浙江大學

Mini Editor 设计文档

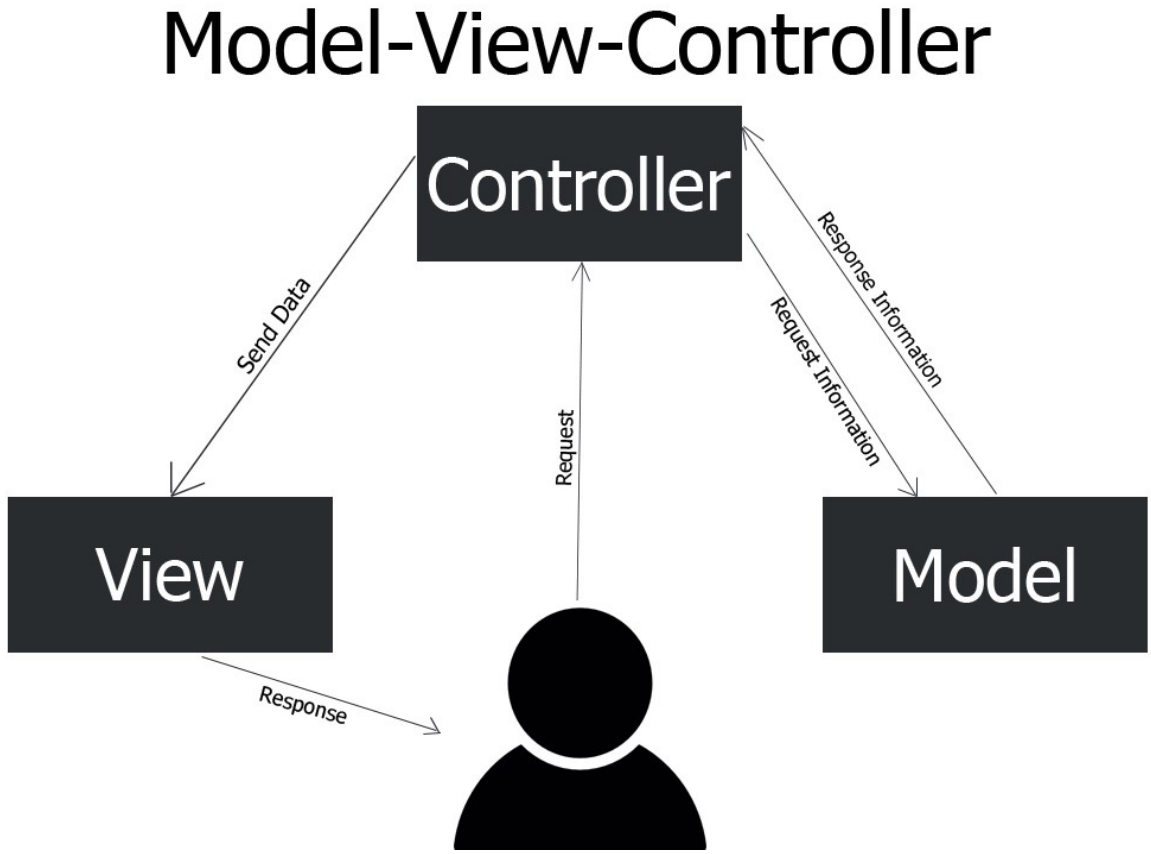
题　　目：	Mini Editor 设计文档
授课教师：	季江民
姓　　名：	邱日宏
学　　号：	3200105842
日　　期：	2022年 7月

1 摘要

命令行编辑器作为在命令行对文本进行轻松便捷地编辑的工具，为人们的生活带来了很大的便利。Mini Editor作为一款命令行编辑器，能够帮助用户在命令行模式下启动文本编辑器对文本文档进行快速而有效的编辑。在有限状态机的模型架构和模型-视图-控制器模式下，Mini Editor可以根据当前所处的状态以及用户输入进行状态转移以实现丰富的功能。与其他命令行编辑器相比，Mini Editor具有程序精简，功能齐全，高效易用等特点，能够应用于众多文本查看与编辑的场景。

2 设计思想

在命令行编辑器的设计上，我们采用了**模型-视图-控制器（MVC）模式**进行设计。在MVC的模式架构下，用户所看到的图形界面的显示与所要显示内容的控制分由不同的模块来完成。即使用户是在编辑模式下进行文本编辑时，用户所输入的文本也不是直接传输到屏幕上进行显示的，而是先经过控制器（controller）对用户输入的合法性等进行判断，确认用户的输入为合法输入且属于文本编辑范畴，然后将用户的输入送给临时存储的文本文件模型（model）并对该临时文本进行修改，最后显示模块（view）对该临时文本进行读取，并按要求渲染到屏幕之上，形成了用户所看到的文本内容。这样通过分离模型、视图、控制器的设计方法让用户的操作控制和显示都变得更为简单和容易，同时也能够让用户在各类模型之间进行自由的切换，而不必担心输入的控制字符作为文本被显示在了屏幕之上。



在我们的Mini Editor命令行编辑器中，用户所看见的屏幕显示作为视图部分，是用户感受程序运行状态以及文本编辑情况的主要窗口。用户通过键盘的输入是用户对Mini Editor控制信号产生的来源，用户始终是以键盘向程序发送信号的方式实现各种交互功能的。我们的模型集中于程序运行过程中保存的一些变量以及一份临时拷贝的保存原文件编辑修改后内容的文件，控制器可以对其中的内容进行读取并以合理的方式将这些内容显示在终端屏幕之中。

MVC的设计思想构成了整个Mini Editor中坚不可摧的基石，是整个命令行编辑器设计的出发点与落脚点，是贯穿Mini Editor设计的灵魂所在。

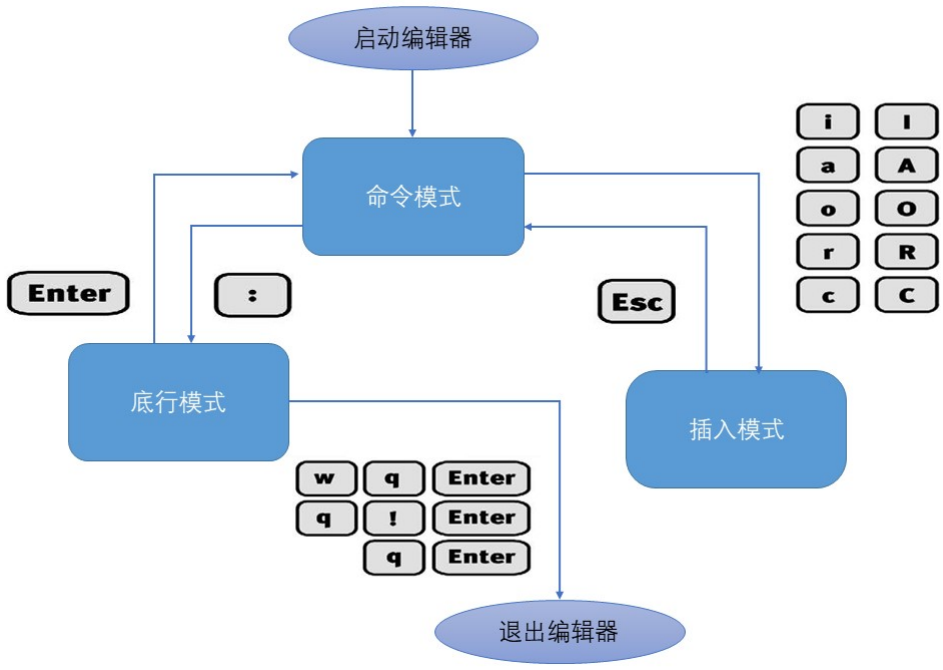
3 有限状态机控制模型

为了实现一个命令行编辑器程序，**有限状态机**的模型对于其实现将是一个不错的选择。有限状态机（Finite-State Machine，FSM）是表示有限个状态以及在这些状态之间的转移和动作等行为的数学计算模型。

我们将命令行编辑器分为三种状态，分别是**命令模式**（command mode）、**插入模式**（Insert mode）和**底行模式**（last line mode）。各模式的功能简介及区分如下：

1. 命令模式（command mode）
   控制屏幕光标的移动，字符、字或行的删除，移动复制某区段及进入插入模式、底行模式。
2. 插入模式（Insert mode）
   用户只有在插入模式下才可以对文本进行编辑，按ESC键可回到命令模式。
3. 底行模式（last line mode）
   将文件保存或退出Mini Editor，如果未退出则回到命令模式。

关于有限状态机模型中各个模式更加详细的内容会在之后进行介绍，这里仅仅列出在Mini Editor设计过程中需要用到的三个模式以及它们内容的简介。下图中列出了Mini Editor的各个模式之间的关系以及它们之间相互转换的方法。



在编辑器里，如果我们能够将不同状态下编辑器所处的状态进行记录，并且在每一种状态下，用户可以采用不同的输入并发送相应的状态转移，那么，我们就可以实现编辑器中“命令模式”、“插入模式”、“底行模式”等多种不同模式之间的转移与变化。

有限状态机在shell脚本编程中的实现并不复杂，其实现的伪代码如下：

```
case $state in
    state1)
        case $operation in
            op1)
                ...
                state=...
            op2)
                ...
                state=...
            ...
        ...
    state2)
        case $operation in
            op3)
                ...
                state=...
            op4)
                ...
                state=...
            ...
        ...
    ...
esac
```

4 独立视图显示模块

作为一款文本编辑软件，如果单纯仅仅只是将文本在用户原来的终端窗口显示所能够带来的表现力是远远不能够满足我们的需求的。如果采用不断刷新输出文档在用户原来的终端窗口显示的话，不仅会使得用户原本的终端窗口显示快速刷屏收到污染让用户难以再找到编辑文档之前的相关终端输出，而且也不利于打印文档编辑器中相关信息的控制。

因此，在我们所设计的编辑器中采用了**独立视图显示**的方法。在独立视图显示下，当用户运行Mini Editor程序之后，将进入一个新的终端屏幕显示Mini Editor的相关输出信息。在进入新窗口之前，为了保证用户在退出Mini Editor编辑器之后还能迅速恢复到原来的工作状态，继续进行下一步的工作，我们必须将用户原来屏幕的各种状态进行保存，然后再进入到新的屏幕之中。

为了能够实现屏幕上**任意位置的输入控制**，我们主要使用了Linux屏幕打印控制中的`tput`命令实现相关操作。通过定位屏幕位置打印各类文字和图标，我们可以将编辑器的各种状态信息以及用户所要编辑的文档等以一种较为好看的方式全区显示在屏幕之中，并且通过将不必要的内容清楚并重新打印信息来实现文本的更新。

5 临时文件编辑存储

在我们所设计的文本编辑器之中，采用了**临时文件存储**的方法，对用户所编辑的文档进行临时存储。当用户打开文本进行编辑的时候，我们的编辑器会在相应的目录下创建一个以_edit结尾的临时文件对用户所编辑更改的文本进行存储。

采用临时文本存储的方法，对用户所编辑的文档进行临时存储，有助于我们对于用户编辑过程存储文档、不保存文档直接退出等操作的实现。如果用户在退出编辑器的时候选择了保存并退出，那么用户所编辑的临时文件会覆盖掉原来的文件，实现保存的功能；如果用户在退出的时候选择了不保存直接退出，那么该临时文件将不会覆盖用户原本编辑的文件，用户原来的文档得以安全无恙的保存。

通过临时文件编辑存储的方法，可以在软件层面提高用户文档编辑的安全性。用户编辑过程中的任何操作都被记录在了编辑器所创建的这个临时文件之中。即使用户在编辑的过程中遇见了系统崩溃临时关机等突发情况，用户也可以通过编辑器所自动保存的临时文件找回之前所编辑的文档。

在用户退出编辑器之后，无论用户是否选择了保存文件，编辑器所创建的临时文件都会被删除。这样既可以在操作上实现对使用用户的隐藏，让用户能够无感知地体验到临时文件编辑存储所带来的功能，又能够保证用户文件夹的清洁和整齐，避免产生过多无用的临时文件。

6 插入模式文本处理

用户以插入或替换文本的方式进入到插入模式后，Mini Editor需要接受用户的输入并将其作为编辑的文本及时渲染到屏幕之上。为了实现用户文本的插入功能，我们将输入文本中的每一行内容作为一个字符数组进行处理。

如果用户是以插入的方式插入文本的话，那么我们首先会根据用户当前光标位置所处在的行号以及列号定位用户当前输入应该插入文本的位置。接着，我们将用户的输入作为一个字符，将用户将要输入行原来的文本内容作为一个字符串，并将该字符串在用户输入位置一分为二。接着，按照用户输入位置之前的字符串、用户输入字符、用户输入位置之后的字符串的方式再将该字符串重新组合，并输出到文本对应行之中，实现对特定行内容单个字符的修改。在完成插入文本后，键盘光标的位置也应该跟着一起向后移动，才符合人们通常连续文本输入的使用习惯。

替换操作的实现与插入的实现几乎是类似的，只不过是将用户输入位置之后的字符串去掉第一个字符后再拼接即可实现，其他部分的原理与插入几乎没有差异。

在实现过程中，对于一些特殊的字符需要进行特别的处理。因为这些字符导致的最终行为与其他字符有着较大的差异。

其中一个需要特殊处理的字符便是**退格键**。因为当用户输入了退格键之后，其目的并不是将退格键所对应的ASCII码127作为一个不可见字符插入到文本之中，而是要将光标之前的字符删去。

为了实现这个目的，当我们检测到用户输入了退格键之后，就将用户输入位置前一个字符串的最后一个字符去除后再和之后的字符串拼接，同时将用户屏幕上光标的位置向前移动一格。对于光标之前再无字符可以删除时的情况需要进行判断和特殊处理，毕竟我们的光标再怎么删除也不应该最后超出到屏幕输入之外的内容。

另一个需要特殊处理的字符是**回车键**。如果用户输入了回车键的话，则表明当前位置应该进行换行并跳转到下一行的开头了。所以，当检测到用户输入回车键之后无论是文本的处理还是光标位置的改变都需要特殊对待。

在遇见回车键时，用户输入光标位置之前的字符串还是照常打印在本行之内的，但输入位置之后的字符串就需要另起一行并单独打印。同时，用户光标的位置也应该移动到下一行的行首，以适应继续输入或其他操作的需要。

7　基本操作

用户界面介绍

启动Mini Editor之后，您将进入Mini Editor的主界面。在界面上方显示的是打开的等待编辑或查看的文档文本。您可以在命令模式下控制光标在文本上的移动，或是在插入模式下对这些文本的内容进行编辑。在界面下方的最后一行会显示当前Mini Editor所处在的模式，Mini Editor中光标所在的行号与列号的信息，您可以通过这些信息辅助您判断当前Mini Editor所处在的状态。如果您在使用过程中有发生了一些错误或进行了一些Mini Editor预期之外的行为，则在界面下方还会显示出相应的错误信息提示您应进行正确的操作。

a) 进入Mini Editor

在系统提示符号输入minieditor.sh及文件名称后，就进入Mini Editor全屏幕编辑画面。例如下面这个例子：

```
$ ./minieditor.sh file
```

其中，$为系统提示命令符，minieditor.sh的可执行程序位于当前目录下，file为您要编辑的文本文档的文件名称。如果minieditor.sh的可执行程序不在当前目录下的话，您需要改变该程序的引用目录，或将minieditor.sh文件所在路径添加到PATH的环境变量之中，才可以正常运行使用Mini Editor。

这里有一点需要特别注意，就是您进入Mini Editor之后，是处于命令行模式，您要切换到插入模式才能够输入文字。

b) 切换至插入模式编辑文件

在命令模式下按一下字母 i , I, a, A, o, O, r 或者 R，就可以进入插入模式，这时候您就可以以插入的方式开始输入文字了。这里的命令**不区分大小写**。其中，按 i 或者 I 切换进入插入模式后，是从光标当前位置开始输入文字；按 a 或者 A 进入插入模式后，是从光标所在位置的下一个位置开始输入文字；按 o 或者 O 进入插入模式后，是插入新的一行，从行首开始输入文字。相关内容整理如下表所示：

命令	解释
i	从光标当前位置开始以插入的方式输入文字
a	从光标所在位置的下一个位置开始输入文字
o	在文末插入新的一行并从行首开始输入文字

c) 从插入模式切换至命令模式

处于插入模式时您只能输入文字，此时您输入的任何内容都会作为文本以插入的方式直接添加到文档之中。只有当您按下`ESC`键后，才可退出插入模式并转到命令模式。在插入模式下，您可以正常编辑文本，如果需要删除当前光标所在位置的内容，您可以通过按下退格键实现。

d) 命令模式其他操作

移动光标

在Mini Editor之中，您可以在命令模式下通过键盘按下英文字母 h 、 j 、 k 、 l ，分别控制光标左、下、上、右移一格，以此逐步将光标移动到您所需要的位置。

替换文本

命令模式里有一种特殊的文本编辑方式，通过替换的方法编辑内容。您可以通过按下r或者R后进入命令模式中的替换功能，此时Mini Editor会将您的输入替换光标所到之处的字符，直到按下 ESC 键为止才退出回到正常的命令模式。

e) 退出Mini Editor及保存文件

在命令行模式下，按一下冒号键进入底行模式。在底行模式下，你可以输入相应的命令完成相关操作。目前，Mini Editor支持的命令如下：

命令	解释
:w	保存当前文件
:wq	保存并退出Mini Editor
:q	退出Mini Editor
:q!	不存盘强制退出Mini Editor
:x	相当于 :wq 的功能

这里有一点需要注意的是，当您输入`:q`命令想要退出Mini Editor时，您应该确保您所有的编辑均已经进行了保存。否则Mini Editor会提示您“文件尚未保存，无法退出”。如果您在发生编辑后想要退出Mini Editor，您应该使用`:wq`或`:x`命令保存最新的编辑成果并退出，或者使用`:q!`不保存文档强制退出编辑器。

f) 错误提示

为了让我们的提示信息等更加丰富且便于用户阅读，我为我的命令行编辑器的提示信息添加了丰富多彩的颜色。事实上，在shell中，屏幕输出文本的颜色是可以使用打印的一些不可见字符来控制的，我们可以使用 ANSI 颜色编码来指定输出的颜色。

例如，这里列出了一些常见的颜色编码：30 (黑色), 31 (红色), 32 (绿色), 33 (黄色), 34 (蓝色), 35 (洋红), 36 (青色), 37 (白色)。通过这些颜色编码与合理的控制，我们就可以让我们的输出产生丰富多彩的颜色。

8 源代码

```
# 程序：命令行编辑器
# 作者：邱日宏 3200105842
# 日期：2022年7月

HelpDoc()
{
    echo "
Mini Editor (2022 July)

Usage: minieditor.sh [arguments] [file ..]       edit specified file(s)
   or: minieditor.sh [arguments] -               read text from stdin

Arguments:
   --                   Only file names after this
   -v                   Vi mode (like "vi")
   -e                   Ex mode (like "ex")
   -E                   Improved Ex mode
   -s                   Silent (batch) mode (only for "ex")
   -d                   Diff mode (like "minieditordiff")
   -y                   Easy mode (like "eminieditor", modeless)
```

```
   -R                   Readonly mode (like "view")
   -Z                   Restricted mode (like "rminieditor")
   -m                   Modifications (writing files) not allowed
   -M                   Modifications in text not allowed
   -b                   Binary mode
   -l                   Lisp mode
   -C                   Compatible with Vi: 'compatible'
   -N                   Not fully Vi compatible: 'nocompatible'
   -V[N][fname]         Be verbose [level N] [log messages to fname]
   -D                   Debugging mode
   -n                   No swap file, use memory only
   -r                   List swap files and exit
   -r (with file name)  Recover crashed session
   -L                   Same as -r
   -A                   Start in Arabic mode
   -H                   Start in Hebrew mode
   -T <terminal>        Set terminal type to <terminal>
   --not-a-term         Skip warning for input/output not being a terminal
   --ttyfail            Exit if input or output is not a terminal
   -u <minieditorrc>    Use <minieditorrc> instead of any .minieditorrc
   --noplugin           Don't load plugin scripts
   -p[N]                Open N tab pages (default: one for each file)
   -o[N]                Open N windows (default: one for each file)
   -O[N]                Like -o but split vertically
   +                    Start at end of file
   +<lnum>              Start at line <lnum>
   --cmd <command>      Execute <command> before loading any minieditorrc file
   -c <command>         Execute <command> after loading the first file
   -S <session>         Source file <session> after loading the first file
   -s <scriptin>        Read Normal mode commands from file <scriptin>
   -w <scriptout>       Append all typed commands to file <scriptout>
   -W <scriptout>       Write all typed commands to file <scriptout>
   -x                   Edit encrypted files
   --startuptime <file> Write startup timing messages to <file>
   -i <minieditorinfo>  Use <minieditorinfo> instead of .minieditorinfo
   --clean              'nocompatible', minieditor defaults, no plugins, no
minieditorinfo
   -h  or  --help       Print Help (this message) and exit
   --version            Print version information and exit

Additional Remarks:
    Not all of the arguments are supported by Mini Editor now.
"
}

# 命令行编辑器参数初始化
InitEditor()
```

```
{
    # 状态模式
    state="Command"      # 设定当前初始状态，为command mode

    # 标记是否修改过
    modified="false"

    # 刷新文本标记
    refresh="true"

    # 行号与列号，默认从1开始
    let row=1        # 行号为1
    let col=1        # 列号为1
}

# 启动命令行编辑器
StartEditor()
{
    # echo $# $1     # 调试信息

    # 参数合法性判断
    # if [[ ($# -ge 2) ]] # 目前Mini Editor暂时仅支持一次编辑一个文件
    #     then     # 如果参数的个数超过2个
    #         echo -e "\033[31m[ERROR]\033[0m\t无效的参数，目前Mini Editor暂时仅支持一次编辑一个文件
    #         请输入'minieditor.sh --help'获取更多信息" # 提示无效参数输入
    #         exit 1   # 退出程序
    # fi

    if [[ ($# -eq 0) || ($# == 1 && $1 == "--help") ]]
        then             # 如果用户没有输入任何参数或是输入了--help
            HelpDoc      # 那么就显示帮助文档
            exit 0
    fi

    # 参数处理
    file=${!#}           # 获取需要编辑的文件
    efile=${file}_edit   # 设置临时文件
    while [ -e $efile ]    # 如果临时文件恰好重名
    do
        efile=${efile}_edit    # 则反复添加_edit后缀确保没有同名文件
    done
    # echo $efile        # 调试信息


    if [ -e $file ]          # 判断文件是否存在
        then                 # 如果原文件已存在
```

```
            if [ ! -r $file -o ! -w $file ]  # 判断用户的读写权限
                then
                    echo -e "\033[31m[ERROR]\033[0m\t 你不具有 $file 的读写权限，无法打开文档"
                    exit 1
            fi

            if [ ! -f $file ]   # 判断文件类型
                then
                    echo -e "\033[31m[ERROR]\033[0m\t $file 不是普通文件，无法打开文档"
                    exit 1
            fi

            cp $file $efile     # 拷贝临时文档准备编辑

        else                # 如果原文件不存在
            touch $efile        # 创建临时文档准备编辑
    fi

    InitEditor
}

# 退出命令行编辑器
EndEditor()
{
    # 退出时删除编辑使用的临时文件
    rm $efile
    # echo "Bye~"
}

Render()
{
    # 参数检查
    if [[ ($# != 1) || (! -f $1) ]]
        then
            echo "Render 参数错误"
            return 1
    fi

    tput civis          # 暂时隐藏光标

    # 初始化变量
    render_file=$1              # 内容暂存文件
    let LineCount=1     # 行号统计，默认从1开始
    rows=$(tput lines)  # 当前屏幕行数
    cols=$(tput cols)   # 当前屏幕列数
```

```
    # 循环显示文本
    tput cup 0 0         # 定位屏幕顶行
    if [[ $refresh == "true" ]]
        then             # 只有插入模式下发生修改了才需要刷新文本
            tput clear           # 清除屏幕
            cat $render_file     # 显示新文本

            # cat $render_file | while read line
            # do
            #     echo $line  # 为了提升打印的速度，暂时取消行号显示功能
            #     # printf "\033[35mLine%3d\033[0m: $line\n" $LineCount
            #     # (( LineCount = LineCount + 1 ))
            # done

            refresh="false" # 恢复刷新标记
    fi

    # 底行打印提示信息
    tput cup $((rows-1)) 0     # 定位屏幕最后一行
    tput el
    printf "== $state Mode ==\t      row: $row  col: $col        $ErrorMessage"
    ErrorMessage=""

    tput cup $((row-1)) $((col-1))         # 重新将光标移到原来的位置
    tput cnorm                             # 重新显示光标
    return 0
}

# 运行状态
FiniteStateMachine()
{
    # 创建新屏幕
    tput smcup           # 进入备用屏幕
    echo -e '\E7'        # 保存当前光标的位置
    echo -e '\033[?47h' # 切换到备用屏幕

    # 定义Internal Field Separator，区别回车，空格与制表符
    IFS=

    # 无限循环
    while :
    do
        Render $efile

        case "$state" in
            Command)     # 命令模式
```

```
        # echo "Command Mode"

        stty -echo      # 关闭命令回显
        read -s -n1 op  # 读入控制字符

        case $op in
            :)
                state="LastLine"
                ;;
            [iI] | [aA] | [oO])     # 插入模式
                if [[ $op == [aA] ]]     # 追加则为从光标后一个字符开始插入
                    then
                        ((col = col + 1))
                fi

                if [[ $op == [oO] ]]     # 打开则为将光标移动到末尾的插入
                    then
                        total_line=$(cat $efile | wc -l)     # 原文件总行数
                        echo >> $efile      # 追加一行新行
                        row=$((total_line + 1)) # 移动光标
                        col=1                   # 移动光标
                fi

                state="Insert"
                ;;
            [rR])           # 替换模式
                state="Replace"
                ;;
            [kK])   # 向上移动
                row=$((row>1?row-1:1))
                ;;
            [jJ])   # 向下移动
                row=$((row<(rows-2)?row+1:rows-2))
                ;;
            [hH])   # 向左移动
                col=$((col>1?col-1:1))
                ;;
            [lL])   # 向右移动
                col=$((col<(cols-1)?col+1:cols-1))
                ;;
            *)
                state="Command"
                ;;
        esac
        ;;

    Insert)     # 插入模式
```

```
                # echo "Insert Mode"

                modified="true"      # 标记发生了修改
                refresh="true"       # 标记刷新

                stty echo            # 打开命令回显
                read -n1 character
                if [[ $character == $'\e' ]]     # 如果输入的是ESC键
                    then
                        state="Command"      # 跳转到命令模式
                        continue             # 继续
                fi

                # 文本更新
                tmp=$(mktemp miniedit.XXXX)
                LineCount=0
                while read line
                do
                    (( LineCount = LineCount + 1 )) # 行号++

                    if [[ $row != $LineCount ]]
                        then
                            echo $line >> $tmp  # 正常情况下为普通复制
                        else
                            # 插入新行
                            ((col = col - 1))    # 在当前光标前插入字符
                            ((col = col > 0 ? col : 1)) # 不允许非法越界

                            if [[ $(printf "%d" \'$character) == 127 ]]     # 退格
键特殊处理
                                then     # 如果输入的是退格键
                                    new_line=${line:0:$((col-1))}${line:$col}
    # 则删除前一个字符
                                else     # 如果输入的不是退格键

                                    if [[ $character == '' ]]     # 回车键特殊处理
                                        then     # 行号和列号需要重定位
                                            new_line=${line:0:$col} # 记录光标前的
内容

                                            echo $new_line >> $tmp  # 输出
                                            new_line=${line:$col}    # 光标后的内容
另起一行

                                            # ((row = row + 1))    # 行号加1
                                            # ((col = 1))          # 列号为1
                                        else
```

```
new_line=${line:0:$col}$character${line:$col}    # 则正常输入
                                        ((col = col + 2))    # 插入字符后光标右移
                                    fi

                                fi

                                echo $new_line >> $tmp
                            fi
                        done < $efile    # 重定向

                        mv $tmp $efile  # 消除临时文件并更新文件
                        ;;

                    Replace)      # 替换模式
                        # echo "Replace Mode"

                        modified="true"      # 标记发生了修改
                        refresh="true"       # 标记刷新

                        stty echo            # 打开命令回显
                        read -n1 character
                        if [[ $character == $'\e' ]]     # 如果输入的是ESC键
                            then
                                state="Command"      # 跳转到命令模式
                                continue             # 继续
                        fi

                        # 文本更新
                        tmp=$(mktemp miniedit.XXXX)
                        LineCount=1      # 行号匹配
                        while read line
                        do
                            if [[ $row != $LineCount ]]
                                then
                                    echo $line >> $tmp  # 正常情况下为普通复制
                                else
                                    # 插入新行
                                    new_line=${line:0:$((col-1))}$character${line:$col}
                                    col=$((col + 1))    # 插入字符后光标右移
                                    echo $new_line >> $tmp  # 对于也要替换部分更新

                                    if [[ $character == '' ]]     # 回车键特殊处理
                                        then     # 行号和列号需要重定位
                                            ((row = row + 1))    # 行号加1
                                            ((col = 1))          # 列号为1
                                    fi
```

```
                fi
                (( LineCount = LineCount + 1 )) # 行号++
            done < $efile    # 重定向
            mv $tmp $efile   # 消除临时文件并更新文件
            ;;

        LastLine)    # 底行模式
            # echo "Last Line Mode"

            # 显示底行的：提示输入底行命令
            tput cup $((rows-1)) 0
            tput el
            stty echo            # 打开命令回显
            read -p ":" lastop

            # 清楚输入的底行命令
            tput cup $((rows-2)) 0
            tput el

            # 根据命令进行决策
            case $lastop in
                w)       # 保存当前文件
                    cp -f $efile $file  # 用修改文件覆盖原文件
                    state="Command"     # 回到命令模式
                    modified="false"    # 修改状态恢复
                    ;;

                wq | x) # 保存并退出Mini Editor
                    cp -f $efile $file  # 用修改文件覆盖原文件
                    break               # 退出
                    ;;

                q)       # 退出Mini Editor
                    if [[ $modified == "true" ]]
                        then
                            ErrorMessage="\033[31m[ERROR]\033[0m\t文件尚未保
存，无法退出"
                            state="Command"     # 回到命令模式
                        else
                            break       # 退出
                    fi
                    ;;

                q!)      # 不存盘强制退出Mini Editor
                    break
                    ;;
                *)
```

```
                        ErrorMessage="\033[31m[ERROR]\033[0m\t无法识别的命令"
                        state="Command"      # 回到命令模式
                        ;;
                esac
                ;;
            *)
                # 其他情况则为模式错误
                ErrorMessage="\033[31m[ERROR]\033[0m\tUnrecognized Mode"
                break
                ;;
        esac

    done

    echo -e '\E[2J'     # 清除屏幕
    echo -e '\033[?47l' # 切换回正常屏幕
    echo -e '\E8'       # 恢复光标位置
    tput rmcup          # 退出备用屏幕
}

StartEditor $@

FiniteStateMachine

EndEditor
```