

## 前言:

本文将介绍怎样用 Qt 做一个简单的多文档编辑器, 该实验的过程中主要涉及到 Qt 窗口的设计, 菜单栏(包括右击菜单), 工具栏, 状态栏, 常见的文本文件等操作。参考资料为网址上的一个例子: <http://www.yafeilinux.com/>

本来是在 ubuntu 下做这个实验的, 可是一开始建立菜单栏等时, 里面用的是中文, 运行后中文就是不显示. 在网上找了 2 天的办法, 各种手段都试过了, 什么编码方式啊, 什么实用 QTextCodec 这个类啊都试过了, 没成功。很不爽, 暂时还是转到 windows 下吧。在 ubuntu 下只是简单的设计了该程序的界面, 然后把那些代码弄到 windows 下后, 打开 main.pp 文件后出现直接在 main.cpp 中出现 了 **could not decode "main.cpp" with "System"-encoding** 这种错误提示, 发现这个源文件中不能输入中文(比如说注释的时候想用中文注释). 但是在同一个工程的其它 cpp 文件中就可以输入中文. 看了下其它的 cpp 文件的 encoding 也是 System 的. 误解, 又是编码问题!!

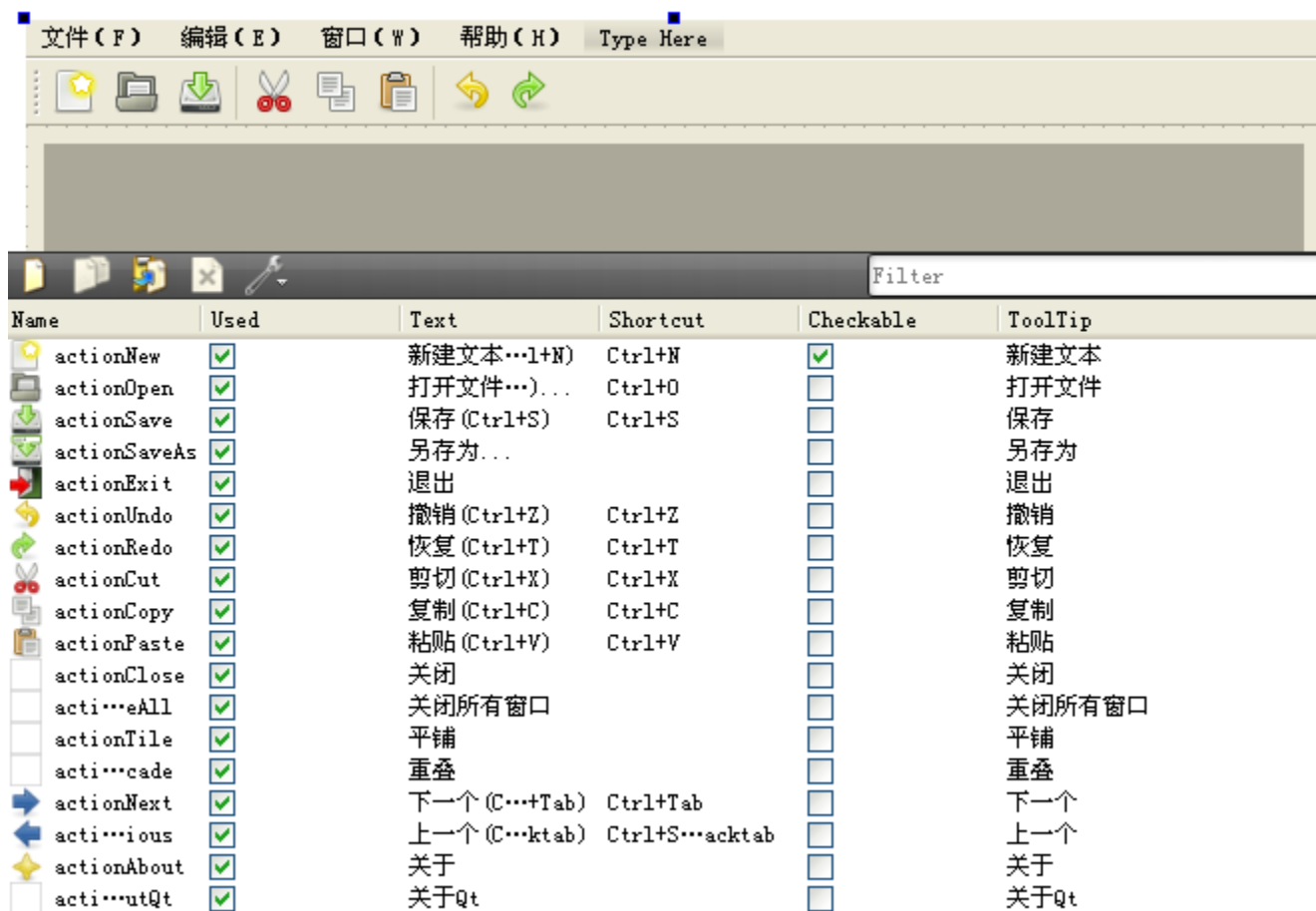
## 实验过程:

下面讲的主要是一个个简单功能的逐步实现过程的某些细节。

### 界面的设计:

在 action 编辑器中, Text 栏中输入文字内容, 如果有快捷方式, 也最好用括号将其注释起来; Object Name 中输入目标名字, 最好采用默认的 action 开头命名; Tooltip 是当鼠标靠近该菜单一会儿的时候会提示的文字, 这里一般与 Text 栏中除括号注释外的相同; ShotCut 一栏中直接用键盘按下快捷键一遍即可, 注意按下 Ctrl+Tab 时显示的为 Ctrl+Tab, 当按下 Ctrl+Shift+Tab 时显示的是 Ctrl+Shift+Backtab.;

菜单涉及完成后 如下图所示:



上面的 used 一栏不可用,当按住 action 编辑器中的每一栏,拖动到对应菜单栏下的 typehere 了就变成打勾可用了。

### MyMdi 文档类的建立:

新建一个类,名字取为 MyMdi,基类名为 QTextEdit (注意,因为下拉列框中可选的基类有限,所以这里可以自己输入),类型信息选择继承来自 QWidget。

因为我们在建立工程的时候,其主界面就是用的 MainWindow 这个类,这个类主要负责主界面的一些界面的布局(比如菜单栏,工具栏,状态栏等),显示,退出和一些人机交互等。那么我们新建的 MyMdi 这个类就不需要负责它在它的父窗口中的显示退出等,只需负责自己窗口的布局和界面显示等。这种思想是说每个界面都单独分离开来,只负责自己界面的实现和与它的子界面交互。

好像 window 和 widget 不同,window 为窗口,包括菜单栏,工具栏,状态栏等,而 widget 一般不包括这些,只包括其文本栏。

### 打开文件功能的实现:

1. 单击工具栏上的打开文件,则会弹出相应的对话框,选中所需要打开的文本文件。
2. 如果该文件已经被打开过,则设置显示该文件对应的窗口为活动窗口。

3. 如果该文件没有被打开过，则新建一个窗口，该窗口贴在其父窗口中。且此时把文件打开，打开成功则状态栏对应显示成功信息，否则输出错误信息。

4. 过程 3 中打开文件是以只读和文本方式打开文件的，打开完后将文本内容显示到窗口，并设置好文件窗口的标题信息等。

5. 如果文本内容变化后但没有保存，则窗口的标题有\*号在后面。

#### 新建文件功能的实现：

1. 单击工具栏上的新建文件，则新建立一个窗口对象，其类为 **MyMdi**，本身具备输入文字的功能，因为是继承的 **QTextEdit**。

2. 设置好标题栏等信息，且当有文字内容改变又没有保存的情况下则后面也一样显示\*号。

#### 保存文件功能的实现：

1. 如果是新建的文件，单击保存时会自动跳到另存为那边，即弹出一个另存为对话框，重新选择保存文件的目录和文件名。

2. 如果是已经保存过的文件，比如说打开的文件，单击菜单栏下的保存时，其内部执行的是用文件流将打开的文件写入到指定的文件名中。

#### 关闭窗口功能的实现：

当单击窗口右上角的关闭按钮时，程序会自动执行该窗口的 **closeEvent()** 函数，所以如果我们在关闭窗口时需要某些功能，可以重写这个函数。

#### 复制粘贴剪切撤销等功能实现：

因为 **MyMdi** 这个类是继承 **QTextEdit** 类的，所以这些方法都可以直接调用 **QTextEdit** 类里面对应的方法就可以了。

#### 更新菜单栏和工具栏功能的实现：

菜单栏中并不是所有的操作都是可用的，比如说复制，如果没有活动窗口，或者即使有活动窗口但是没有选中文本，则该操作不可以，同理，剪切也是一样。

另外，撤销和恢复都是要经过系统判断，当前是否可用执行这些操作，如果可以则这些操作对应的图标为亮色，可用，否则为灰色不可用。

状态栏的操作也是一样，当有光标移动时，状态栏显示的行列号值才会跟着变化。

#### 更新窗口子菜单栏功能实现：

当打开多个文档时，窗口子菜单下面会自动列出这些文档的名字，且作为一个组单独用分隔符与上面的子菜单隔开。我们可以在该菜单栏下选择一个文档，选完后该文档会被自动当做活动文档，且处于选中状态。前 9 个文档可以用 1~9 这些数字做为快捷键。

#### 保存窗口设置功能实现：

如果软件需要实现这一功能：当下次打开时和上次该软件关闭时的窗口大小，位置一样。那么我们就必须在每次关闭软件时，保留好窗口大小，尺寸等信息，当下次打开该软件时，重新读取这些信息并对窗口进行相应的设置。这里需要用到 **QSettings** 这个类，该类是永久保存于平台无关的应用程序的一些设置的类。在本程序中，关闭软件时写入窗口信息，打开软件在构造函数中读取该信息并设置相应的窗口。

### 自定义右键菜单栏功能实现：

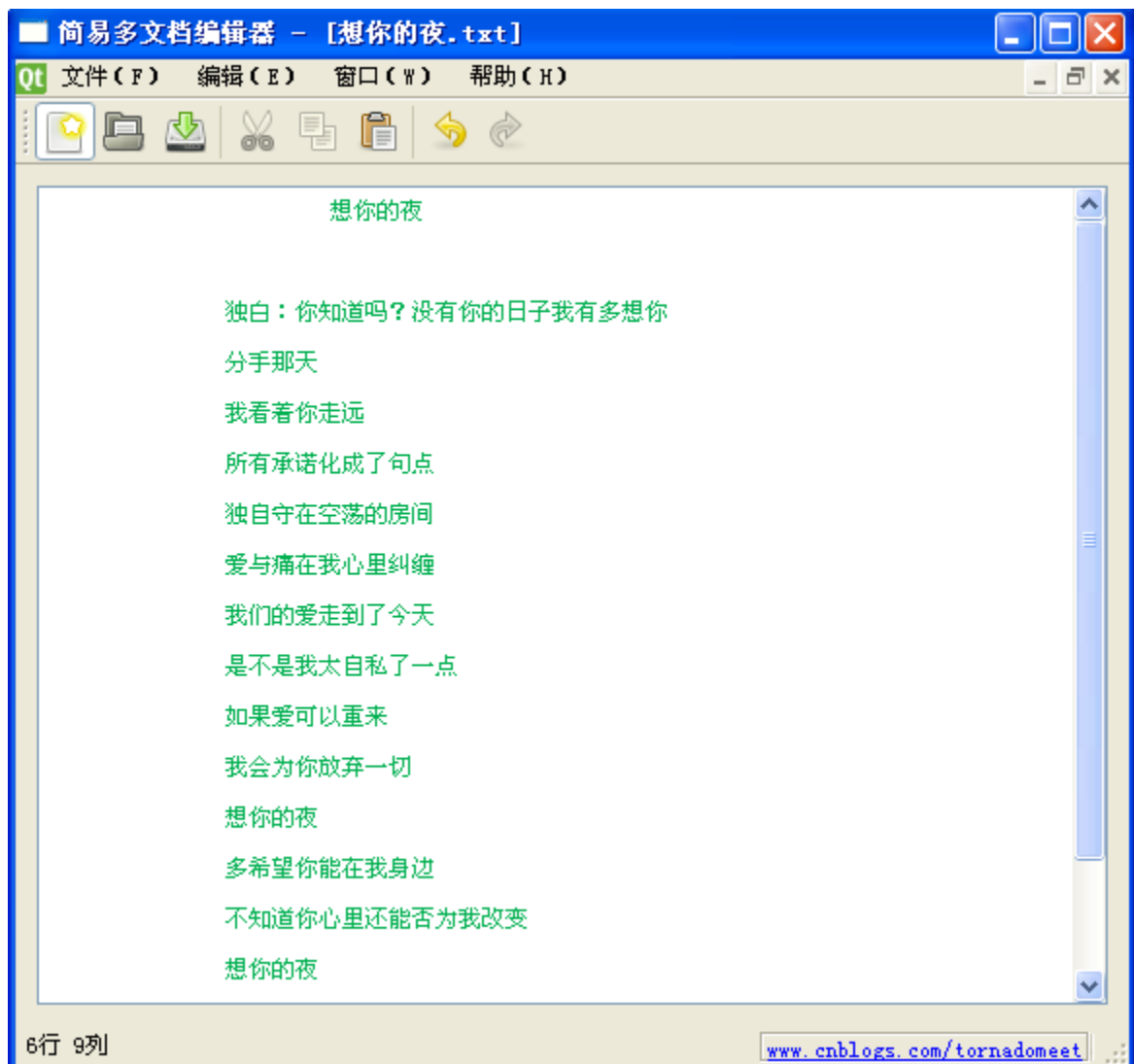
默认的右键菜单栏为英文的，我们这里需要把它弄成中文的，只需在 **MyMdi** 这个类中重写函数 **contextMenuEvent(QContextMenuEvent \*event)** 即可。在该函数中，只需新建一个菜单，然后动态为这个菜单加入 **action**，并且为每个 **action** 设置快捷键，同时也需要根据情况实现对应 **action** 是否可用。

### 初始化窗口的实现：

这一部分包括设置窗口标题，设置工具栏标题，设置水平垂直滚动条，在状态栏上加一个 **label**，状态栏上显示菜单栏上各种 **action** 的提示信息，虽然这个初始化窗口是在构造函数中调用的，但是这些设置在整个应用程序中都有效。

## 实验结果：

本实验的功能在上面几个过程中已有实现，类似于 **windows** 下的记事本一样。下面是其效果一张简单的截图：



实验主要部分代码即注释(附录有工程 **code** 下载链接):

**mymdi.h:**

```
#ifndef MYMDI_H
#define MYMDI_H

#include <QTextEdit>

class MyMdi : public QTextEdit
{
    Q_OBJECT
public:
    explicit MyMdi(QWidget *parent = 0);
    void NewFile();
};
```

```

    bool LoadFile(const QString &file_name);
    QString CurrentFilePath();
    QString get_current_file_name();
    void SetCurrentFile(const QString &file_name);
    bool Save();
    bool SaveAs();
    bool SaveFile(const QString &file_name); //因为 Save() 和 SaveAs() 有很多共同的代码, 所以最好单独写个函数供其调用。

signals:

public slots:

private:
    QString current_file_path_; //当前文件的文件名
    bool is_saved_; //文件是否保存标志
    bool has_saved();
    void contextMenuEvent(QContextMenuEvent *event);

protected:
    void closeEvent(QCloseEvent *); //重写关闭事件

private slots:
    void DocumentWasModified(); //当文档内容被改后所需执行的操作

};

#endif // MYMDI_H

```

### ***mymdi.cpp:***

```

#include "mymdi.h"
#include <QFile>
#include <QMessageBox>
#include <QTextStream>
#include <QApplication>
#include <QFileInfo>
#include <QFileDialog>
#include <QPushButton>
#include <QCloseEvent>
#include <QMenu>

MyMdi::MyMdi(QWidget *parent) :

```

```

    QTextEdit(parent) //因为 MyMdi 是继承 QTextEdit 类的，所以它本身就是一个文本编辑类，
    可以编辑文字
{
    setAttribute(Qt::WA_DeleteOnClose); //加入了这句代码后，则该窗口调用 close() 函数
    不仅仅是隐藏窗口而已，同时也被销毁
    is_saved_ = false;
}

void MyMdi::NewFile()
{
    static int sequence_number = 1;
    is_saved_ = false;
    current_file_path_ = tr("未命名文档%1.txt").arg(sequence_number++);
    setWindowTitle(current_file_path_ + "[*]"); //设置文档默认标题，“[*]”在默认情况下
    是什么都不显示的，只有当调用 setWindowModified()
    //函数的时候，会自动在由“[*]”的地方加上“*”，
    后面的文字会自动后移
    connect(document(), SIGNAL(contentsChanged()), this,
    SLOT(DocumentWasModified())); //文档内容发生改变时，
    //触发
    槽函数 DocumentWasModified().
}

QString MyMdi::CurrentFilePath()
{
    return current_file_path_; //current_file_path_ 是私有变量，对外隐藏起来了，但是
    CurrentFilePath() 是公有成员函数，显示出现
}

//设置当前文件的一些信息，比如说窗口标题，该文件的路径名等
void MyMdi::SetCurrentFile(const QString &file_name)
{
    current_file_path_ = QFileInfo(file_name).canonicalFilePath(); //得到解释过后
    的绝对路径名
    is_saved_ = true; //设置为被保存过，因为该函数是被 LoadFile() 函数调用的，所以肯定可
    以被当做是保存过的了
    document()->setModified(false); //文档没有被改过
    setWindowModified(false); //窗口不显示被更改的标志
    setWindowTitle(get_current_file_name() + "[*]"); //设置窗口标题
}

bool MyMdi::LoadFile(const QString &file_name)
{
    QFile file(file_name); //建立需打开的文件对象

```

```

        if(!file.open(QFile::ReadOnly | QFile::Text))
        {
            //打开失败时，输出错误信息
            QMessageBox::warning(this, "多文档编辑器", tr("无法读取文件 %1:
\n%2").arg(file_name).arg(file.errorString()));
            return false;
        }

        QTextStream in(&file); //文本流
        QApplication::setOverrideCursor(Qt::WaitCursor); //设置整个应用程序的光标形状为
        等待形状，因为如果文件的内容非常多时可以提醒用户
        setPlainText(in.readAll()); //读取文本流中的所有内容，并显示在其窗体中
        QApplication::restoreOverrideCursor(); //恢复开始时的光标状态
        SetCurrentFile(file_name); //设置标题什么的
        //注意这里发射信号用的是 contentsChanged(), 而不是 contentsChange().
        connect(document(), SIGNAL(contentsChanged()), this,
        SLOT(DocumentWasModified()));

        return true;
    }

    QString MyMdi::get_current_file_name()
    {
        return QFile::fileName(current_file_path_); //从当前文件路径名中提取其文件
        名
    }

    void MyMdi::DocumentWasModified()
    {
        setWindowModified(document()->isModified()); // "*" 显示出来
    }

    bool MyMdi::has_saved()
    {
        if(document()->isModified())
        {
            QMessageBox box;
            box.setWindowTitle(tr("多文档编辑器"));
            box.setText(tr("是否保存对%1 的更改? ").arg(get_current_file_name()));
            box.setIcon(QMessageBox::Warning); //警告图标
            //下面是消息 box 上添加 3 个按钮，分别为 yes, no, cancel
            QPushButton *yes_button = box.addButton(tr("是"),
            QMessageBox::YesRole);

```



```

        QPushButton *no_button = box.addButton(tr("否"),
QMessageBox::NoRole);

        QPushButton *cancel_button = box.addButton(tr("取消"),
QMessageBox::RejectRole);

        box.exec(); //在这里等待用户选择 3 个按钮中的一个
        if(box.clickedButton() == yes_button)
            return Save();
        else if(box.clickedButton() == no_button)
            return true; //不用保存, 直接关掉
        else if(box.clickedButton() == cancel_button)
            return false; //什么都不做

    }

    return true; //要么已经保存好了, 要么根本就没更改过其内容
}

bool MyMdi::Save()
{
    if(is_saved_) //已经保存过至少一次后, 则说明文件的文件名等已经弄好了, 直接保存内容即可。
        return SaveFile(current_file_path_);
    else return SaveAs(); //第一次保存时, 需要调用 SaveAs
}

bool MyMdi::SaveAs()
{
    //返回的名字 file_name 是自己手动输入的名字, 或者直接采用的是默认的名字
    QString file_name = QFileDialog::getSaveFileName(this, tr("另存为"),
current_file_path_);
    if(file_name.isEmpty())
        return false;

    return SaveFile(file_name);
}

bool MyMdi::SaveFile(const QString &file_name)
{
    QFile file(file_name);
    //即使是写入文本, 也得将文本先打开
    if(!file.open(QFile::WriteOnly | QFile::Text))
    {
        QMessageBox::warning(this, "多文档编辑器", tr("无法写入文件 %1:
\n%2").arg(file_name).arg(file.errorString()));
        return false;
    }
}

```

```

QTextStream out(&file);
QApplication::setOverrideCursor(Qt::WaitCursor);
out << toPlainText(); //以纯文本方式写入，核心函数
QApplication::restoreOverrideCursor();
//返回之前，也将该文件的标题，路径名等设置好。
setCurrentFile(file_name);
return true;
}

void MyMdi::contextMenuEvent(QContextMenuEvent *event)
{
    QMenu *menu = new QMenu;

    //QKeySequence 类是专门封装快捷键的，这里使用的是默认的快捷键操作，其快捷键位"&"号后面
    那个字母
    QAction *undo = menu->addAction(tr("撤销(&U)"), this, SLOT(undo()),
    QKeySequence::Undo); //直接调用槽函数 undo()
    undo->setEnabled(document()->isUndoAvailable()); //因为该类是一个 widget，所以
    可以直接使用 document() 函数

    QAction *redo = menu->addAction(tr("恢复(&A)"), this, SLOT(redo()),
    QKeySequence::Redo);
    redo->setEnabled(document()->isRedoAvailable());

    menu->addSeparator(); //增加分隔符

    QAction *cut = menu->addAction(tr("剪切(&T)"), this, SLOT(cut()),
    QKeySequence::Cut);
    cut->setEnabled(textCursor().hasSelection());

    QAction *copy = menu->addAction(tr("复制(&C)"), this, SLOT(copy()),
    QKeySequence::Copy);
    copy->setEnabled(textCursor().hasSelection());

    menu -> addAction(tr("粘贴&P"), this, SLOT(paste()), QKeySequence::Paste);

    QAction *clear = menu->addAction(tr("清空"), this, SLOT(clear()));
    clear->setEnabled(!document()->isEmpty()); //文本内容非空时就可以清除

    menu->addSeparator(); //增加分隔符

    QAction *select_all = menu->addAction(tr("全选"), this, SLOT(selectAll()),
    QKeySequence::SelectAll);
    select_all->setEnabled(!document()->isEmpty());

```

```

        menu->exec(event->globalPos()); //获取鼠标位置，并显示菜单

        delete menu; //销毁这个菜单
    }

    //该函数是顶层窗口被关闭时发出的事件，是关闭窗口自带的关闭符号 x
    void MyMdi::closeEvent(QCloseEvent *event) //记得加入 #include <QCloseEvent>
    {
        if(has_saved())
            event->accept(); //保存完毕后直接退出程序
        else
            event->ignore();
    }

```



### ***mainwindow.h:***



```

#ifndef MAINWINDOW_H
#define MAINWINDOW_H

#include <QMainWindow>
// #include "mymdi.h"
#include <QAction>

class MyMdi;
class QMdiSubWindow; //加入一个类相当于加入一个头文件?
class QSignalMapper; //这是个跟信号发射相关的类

namespace Ui {
class MainWindow;
}

class MainWindow : public QMainWindow
{
    Q_OBJECT

public:
    explicit MainWindow(QWidget *parent = 0);
    ~MainWindow();

private slots:
    // void set_active_sub_window(QWidget *window);

```

```
MyMdi *CreateMyMdi();
void set_active_sub_window(QWidget *window);
void UpdateMenus();
void ShowTextRowCol();
void UpdateWindowMenu();
void closeEvent(QCloseEvent *event);

void on_actionNew_triggered();

void on_actionOpen_triggered();

void on_actionExit_triggered();

void on_actionSave_triggered();

void on_actionSaveAs_triggered();

void on_actionCut_triggered();

void on_actionCopy_triggered();

void on_actionPaste_triggered();

void on_actionUndo_triggered();

void on_actionRedo_triggered();

void on_actionClose_triggered();

void on_actionCloseAll_triggered();

void on_actionTile_triggered();

void on_actionCascade_triggered();

void on_actionNext_triggered();

void on_actionPrevious_triggered();

void on_actionAbout_triggered();

void on_actionAboutQt_triggered();

private:
```

```

    Ui::MainWindow *ui;

    QAction *actionSeparator;
    QMdiSubWindow *FindMdiChild(const QString &file_name); //查找子窗口
    MyMdi *GetActiveWindow();
    QSignalMapper *window_mapper;
    void read_settings();
    void write_settings();
    void init_window();
};

#endif // MAINWINDOW_H

```

### ***mainwindow.cpp:***

```

#include "mainwindow.h"
#include "ui_mainwindow.h"
#include "mymdi.h"
#include <QFileDialog>
#include <QMdiSubWindow>
#include <QDebug>
#include <QSignalMapper>
#include <QSettings>
#include <QCloseEvent>
#include <QLabel>
#include <QMessageBox>
#include <QMenu>

MainWindow::MainWindow(QWidget *parent) :
    QMainWindow(parent),
    ui(new Ui::MainWindow)
{
    ui->setupUi(this);

    actionSeparator = new QAction(this);
    actionSeparator->setSeparator(true);
    UpdateMenus();
    //有子窗口被激活，则更新菜单栏
    connect(ui->mdiArea, SIGNAL(subWindowActivated(QMdiSubWindow*)), this,
        SLOT(UpdateMenus()));

    window_mapper = new QSignalMapper(this); //创建信号发生器

```

```

        connect(window_mapper, SIGNAL(mapped(QWidget*)), this,
SLOT(set_active_sub_window(QWidget*))); //通过信号发生器设置活动窗口

        UpdateWindowMenu(); //更新窗口子菜单
        connect(ui->menuW, SIGNAL(aboutToShow()), this,
SLOT(UpdateWindowMenu())); //当窗口子菜单将要出现时，就触发更新窗口子菜单

        read_settings(); //因为在退出窗口时，执行了 write_settings() 函数，即保存了退出窗口
        //等信息会保留
        init_window(); //初始化窗口
    }

MainWindow::~MainWindow()
{
    delete ui;
}

void MainWindow::on_actionNew_triggered()
{
    //    MyMdi *new_mdi = new MyMdi();
    //    ui->mdiArea->addSubWindow(new_mdi);

    /*为什么不能使用上面的方法呢？因为上面的方法没有涉及到文档内容改变时，比如选中了文字，有
    过撤销操作等。

    即使我们又 UpdateMenus() 函数，但是关联它的 connect 函数的信号为当有新的活动窗口出现时，
    所以一旦新

    的活动窗口出现后，后面该文档内容的改变就不会触发菜单栏和工具栏对应 action 的变化了。
    */
    MyMdi *new_mdi = CreateMyMdi();
    new_mdi->NewFile(); //新建文件
    new_mdi->show();
}

void MainWindow::on_actionOpen_triggered()
{
    QString file_name = QFileDialog::getOpenFileName(this); //手动选择需要打开的文
    件，其实返回的 file_name 是包含路径名的文件名
    if(!file_name.isEmpty())
    {
        QMdiSubWindow *existing_window = FindMdiChild(file_name);
        if(existing_window) //如果该文件对应窗口已经打开
        {

```

```

        set_active_sub_window(existing_window); //设置该窗口为活动窗口, 虽然
set_active_sub_window 是该类的成员函数, 但是不能使用
//ui->来调用, 冒失 ui->调用的都是跟
界面相关自动生成的一些量
        return ;
    }
    MyMdi *open_window = CreateMyMdi(); //否则新建子窗口, 且加入到多文档容器中
    if(open_window->LoadFile(file_name))
    {
        ui->statusBar->showMessage(tr("打开文件成功"), 2000); //状态栏显
示打开文件成功, 持续 2 秒
        open_window->show();
    }
    else
    {
        open_window->close(); //打不开该文件时, 则销毁新建的窗口
    }
}

MyMdi* MainWindow::CreateMyMdi()
{
    MyMdi *child = new MyMdi();
    ui->mdiArea->addSubWindow(child);

    //根据是否可复制来设置剪切复制动作是否可用
    connect(child, SIGNAL(copyAvailable(bool)), ui->actionCopy,
SLOT(setEnabled(bool)));
    connect(child, SIGNAL(copyAvailable(bool)), ui->actionCut,
SLOT(setEnabled(bool)));

    //根据文档时否可用撤销和恢复来设置相应的撤销恢复动作是否可用
    connect(child->document(), SIGNAL(undoAvailable(bool)), ui->actionUndo,
SLOT(setEnabled(bool)));
    connect(child->document(), SIGNAL(redoAvailable(bool)), ui->actionRedo,
SLOT(setEnabled(bool)));

    connect(child, SIGNAL(cursorPositionChanged()), this,
SLOT(ShowTextRowCol()));
    return child;
}

QMdiSubWindow* MainWindow::FindMdiChild(const QString &file_name)
{

```

```

    QString canonical_file_path = QFileInfo(file_name).canonicalFilePath();//
解释过后的绝对路径
    foreach(QMdiSubWindow *window, ui->mdiArea->subWindowList())
    {
        MyMdi *my_mdi = qobject_cast<MyMdi
*>(window->widget());//qobject_cast 为进行强制类型转换
        if(my_mdi->CurrentFilePath() == canonical_file_path)//如果已经存在该窗
口, 则返回。比较的是绝对路径名+文件名
            return window;
    }
    return 0;//没找到, 则返回 0
}

void MainWindow::set_active_sub_window(QWidget *window)
{
    if(!window)
        return;
    ui->mdiArea->setActiveSubWindow(qobject_cast<QMdiSubWindow*>(window));//
将当前窗口设置为多文档中的活动窗口
}

MyMdi* MainWindow::GetActiveWindow()
{
    //    //获得子窗口后还需要获得其 widget()
    //    MyMdi *active_window =
qobject_cast<MyMdi*>(ui->mdiArea->activeSubWindow()->widget());
    //    if(active_window)
    //        return active_window;
    //    else
    //        return 0;//虽然返回类型是类的指针, 但是这里也可以返回 0, 表示的是空指针。
    /*上面的方法在后面会报内存错误*/
    if(QMdiSubWindow *active_sub_window = ui->mdiArea->activeSubWindow())
        return qobject_cast<MyMdi*>(active_sub_window->widget());//为什么还要调用
widget() 呢?
    else
        return 0;
}

void MainWindow::on_actionExit_triggered()
{
    qApp->closeAllWindows();//qApp 为全局指针, 关闭所有窗口
}

void MainWindow::on_actionSave_triggered()

```



```
{
    if(GetActiveWindow() && GetActiveWindow()->Save())
        ui->statusBar->showMessage(tr("保存文件成功"), 2000); //状态栏显示保存成功字
样 2 秒
}

void MainWindow::on_actionSaveAs_triggered()
{
    if(GetActiveWindow() && GetActiveWindow()->SaveAs())
        ui->statusBar->showMessage(tr("保存文件成功"), 2000); //状态栏显示保存成功字
样 2 秒
}

void MainWindow::on_actionCut_triggered()
{
    if(GetActiveWindow())
        GetActiveWindow()->cut(); //直接调用 QTextEdit 这个类的 cut() 函数
}

void MainWindow::on_actionCopy_triggered()
{
    if(GetActiveWindow())
        GetActiveWindow()->copy(); //复制
}

void MainWindow::on_actionPaste_triggered()
{
    if(GetActiveWindow())
        GetActiveWindow()->paste(); //粘贴
}

void MainWindow::on_actionUndo_triggered()
{
    if(GetActiveWindow())
        GetActiveWindow()->undo(); //撤销
}

void MainWindow::on_actionRedo_triggered()
{
    if(GetActiveWindow())
        GetActiveWindow()->redo(); //恢复
}
```

```
void MainWindow::on_actionClose_triggered()
{
    ui->mdiArea->closeActiveSubWindow(); //关闭当前活动窗口
}

void MainWindow::on_actionCloseAll_triggered()
{
    ui->mdiArea->closeAllSubWindows(); //关闭所有子窗口
}

void MainWindow::on_actionTile_triggered()
{
    ui->mdiArea->tileSubWindows(); //平铺窗口
}

void MainWindow::on_actionCascade_triggered()
{
    ui->mdiArea->cascadeSubWindows(); //重叠窗口
}

void MainWindow::on_actionNext_triggered()
{
    ui->mdiArea->activateNextSubWindow(); //下一个窗口
}

void MainWindow::on_actionPrevious_triggered()
{
    ui->mdiArea->activatePreviousSubWindow(); //上一个窗口
}

void MainWindow::on_actionAbout_triggered()
{
    QMessageBox::about(this, tr("关于本软件"), tr("参考 www.yafeilinux.com 网站做的一个实验"));
}

void MainWindow::on_actionAboutQt_triggered()
{
    qApp->aboutQt(); //这里的 qApp 是 QApplication 对象的全局指针
}

void MainWindow::UpdateMenus()
{
    bool has_active_window; //如果有活动窗口，则为 1，没有则为 0
```

```

if(GetActiveWindow())
    has_active_window = true;
else has_active_window = false;

//设置间隔器是否显示, 貌似没有效果?
// actionSeparator->setVisible(has_active_window);

//下面是根据是否存在活动窗口来设置各个动作是否可用
ui->actionSave->setEnabled(has_active_window);
ui->actionSaveAs->setEnabled(has_active_window);
ui->actionPaste->setEnabled(has_active_window);
ui->actionClose->setEnabled(has_active_window);
ui->actionCloseAll->setEnabled(has_active_window);
ui->actionTile->setEnabled(has_active_window);
ui->actionCascade->setEnabled(has_active_window);
ui->actionNext->setEnabled(has_active_window);
ui->actionPrevious->setEnabled(has_active_window);

//只有当有活动窗口, 且有文字被选中时, 剪切和复制功能才可以使用
bool has_text_selection;
// QTextEdit->textCursor().hasSelection() 用来判断是否有文本被选中
has_text_selection = (GetActiveWindow() &&
GetActiveWindow()->textCursor().hasSelection());
ui->actionCut->setEnabled(has_text_selection);
ui->actionCopy->setEnabled(has_text_selection);

//有活动窗口, 且系统判断可以执行撤销操作时才显示撤销可用, 判断恢复操作可执行时恢复操作才
可用
ui->actionUndo->setEnabled(GetActiveWindow() &&
GetActiveWindow()->document()->isUndoAvailable());
ui->actionRedo->setEnabled(GetActiveWindow() &&
GetActiveWindow()->document()->isRedoAvailable());
}

//状态栏上显示光标的行号和列号
void MainWindow::ShowTextRowCol()
{
    if(GetActiveWindow())
    {
        ui->statusBar->showMessage(tr("%1 行 %2 列
").arg(GetActiveWindow()->textCursor().blockNumber()+1).
arg(GetActiveWindow()->textCursor().columnNumber()+1), 2000);
    }
}

```

```

}

void MainWindow::UpdateWindowMenu()
{
    ui->menuW->clear(); //清空所有菜单栏
    /*重新加载已有的菜单*/
    ui->menuW->addAction(ui->actionClose);
    ui->menuW->addAction(ui->actionCloseAll);
    ui->menuW->addSeparator();
    ui->menuW->addAction(ui->actionTile);
    ui->menuW->addAction(ui->actionCascade);
    ui->menuW->addSeparator();
    ui->menuW->addAction(ui->actionNext);
    ui->menuW->addAction(ui->actionPrevious);
    //加载间隔器
    ui->menuW->addAction(actionSeparator);

    QList<QMdiSubWindow *> windows = ui->mdiArea->subWindowList();
    actionSeparator->setVisible(!windows.isEmpty());

    for(int i = 0; i < windows.size(); i++)
    {
        MyMdi *child = qobject_cast<MyMdi*>(windows.at(i)->widget());
        QString text;
        if(i < 1) //这个时候变化数字就是其快捷键
            text = tr("&%
1%2").arg(i+1).arg(child->get_current_file_name()); //内容前面加了"&"表示可以使用
快捷键，为第一个字母或数字
        else
            text = tr("%1 %2").arg(i+1).arg(child->get_current_file_name());

        QAction *action = ui->menuW->addAction(text); //添加新的菜单动作
        action->setCheckable(true);
        action->setChecked(child == GetActiveWindow()); //选中当前的活动窗口
        connect(action, SIGNAL(triggered()), window_mapper, SLOT(map())); //
选中 action 会触发槽函数发送 mapped() 信号
        //该函数的作用是设置一个映射，当在运行 action 的信号函数 map() 时，该函数会自动发
送信号 mapped()，并且会以 mapped(windows.at(i)) 来发送
        //此时会触发在构造函数中设置的连接，其槽函数为设置活动窗口
        window_mapper->setMapping(action, windows.at(i));
    }
}

void MainWindow::init_window()

```

```

{
    setWindowTitle(tr("简易多文档编辑器"));
    ui->mainToolBar->setWindowTitle(tr("工具栏")); //设置工具栏的标题名称，右击时才可
    以看到

    //当需要的时候，设置水平垂直滚动条
    ui->mdiArea->setHorizontalScrollBarPolicy(Qt::ScrollBarAsNeeded);
    ui->mdiArea->setVerticalScrollBarPolicy(Qt::ScrollBarAsNeeded);

    ui->statusBar->showMessage(tr("欢迎使用多文档编辑器"));

    QLabel *label = new QLabel(this);
    label->setFrameStyle(QFrame::Box | QFrame::Sunken); //设置 label 的形状和阴影模
    式的, 这里采用的 box 形状和凹陷模式
    label->setText(tr("<a href =
    \"www.cnblogs.com/tornadomeet\">www.cnblogs.com/tornadomeet</a>")); //设置文本
    内容
    label->setTextFormat(Qt::RichText); //设置文本格式为富文本格式，又称多文本格式，用
    于跨平台使用的
    label->setOpenExternalLinks(true); //运行打开 label 上的链接

    ui->statusBar->addPermanentWidget(label); //将 label 附加到状态栏上，永久性的

    ui->actionNew->setStatusTip(tr("创建一个文件"));
    ui->actionOpen->setStatusTip(tr("打开一个已经存在的文件"));
    ui->actionSave->setStatusTip(tr("保存文档到硬盘"));
    ui->actionSaveAs->setStatusTip(tr("以新的名称保存文档"));
    ui->actionExit->setStatusTip(tr("退出应用程序"));
    ui->actionUndo->setStatusTip(tr("撤销先前的操作"));
    ui->actionRedo->setStatusTip(tr("恢复先前的操作"));
    ui->actionCut->setStatusTip(tr("剪切选中的内容到剪贴板"));
    ui->actionCopy->setStatusTip(tr("复制选中的内容到剪贴板"));
    ui->actionPaste->setStatusTip(tr("粘贴剪贴板的内容到当前位置"));
    ui->actionClose->setStatusTip(tr("关闭活动窗口"));
    ui->actionCloseAll->setStatusTip(tr("关闭所有窗口"));
    ui->actionTile->setStatusTip(tr("平铺所有窗口"));
    ui->actionCascade->setStatusTip(tr("层叠所有窗口"));
    ui->actionNext->setStatusTip(tr("将焦点移动到下一个窗口"));
    ui->actionPrevious->setStatusTip(tr("将焦点移动到前一个窗口"));
    ui->actionAbout->setStatusTip(tr("显示本软件的介绍"));
    ui->actionAboutQt->setStatusTip(tr("显示 Qt 的介绍"));
}

```

```

void MainWindow::write_settings()
{
    QSettings settings("Qt", "MyMdi");//第一个为公司的名字，第二个为软件的名字
    settings.setValue("pos", pos());//写入该窗口相对于其父窗口的位置信息
    settings.setValue("size", size());//写入窗口大小信息
}

void MainWindow::read_settings()
{
    QSettings settings("Qt", "MyMdi");
    //settings.value() 第二个参数为默认值，即如果 key: "pos"不存在，则返回默认值
    QPoint pos = settings.value("pos", QPoint(200, 200)).toPoint();
    QSize size = settings.value("size", QSize(400, 400)).toSize();
    move(pos); //在构造函数中才调用 read_settings() 函数，因此这里重新移动窗口位置和设置
窗口大小
    resize(size);
}

void MainWindow::closeEvent(QCloseEvent *event)
{
    ui->mdiArea->closeAllSubWindows();
    if(ui->mdiArea->currentSubWindow()) //如果还有窗口没关闭，则忽略该事件。应该是上面
的语句没有全部关闭成功。
        event->ignore();
    else
    {
        write_settings();//关闭前写入窗口设置
        event->accept();//关闭
    }
}

```



### **main.cpp:**




```

#include <QApplication>
#include "mainwindow.h"
#include <QTextCodec>

int main(int argc, char *argv[])
{
    QApplication a(argc, argv);
    QTextCodec::setCodecForTr(QTextCodec::codecForLocale());
    MainWindow w;
    w.show();
}

```

```
return a.exec();  
}
```



### 总结:

通过本次实验，对 Qt 中文件目录，菜单工具栏等操作有了一定的了解。

### 参考资料:

<http://www.yafeilinux.com/>

### 附录:

实验工程 code 下载。