Qt

#主窗口方法控制：

resize(800,600);//设置窗口大小

setWindowTitle(“Title”);//设置主窗口标题

#父子窗口之间的关系管理：

class ParentWindow:public QMainWindow{

//...

public slots:

void onChildWindowOpened(){

**ChildWindow\*child=new ChildWindow(this);//关键在于子窗口用父this指针设置父窗口**

//...

}

}

class ChildWindow:public QDialog{//...}

#QString <--> std::string :

QString类型转换为std::string: <QString>.toStdString();

std::string转换为QString: QString::formStdString(<std::string>);

#按钮的创建：

包含<QPushButton>

在widget中添加一个类成员QPushButton\*button,在构造函数中写初始化：

button=new QPushButton(“Welcome!”,this);//按钮初始化

button->move(100,100);//移动按钮

button->setText(“Welcome Man!”);//调节按钮标语

#信号槽机制：

1.在MainWindow类中可以添加”public slots:”选项，里面可以写槽函数：

public slots:

void onExitButtonClicked(){//...}

2.可以通过connect的方式将操作与槽函数连接起来：

**connect**(exit\_btn,**&**QPushButton::clicked,**this**,**&**MainWindow::onExitButtonClicked);

其中第一个是发出信号者，第二个是信号的产生时机，第三个是信号接收者，第四个是槽函数。

3.槽函数可以写成lambda表达式.

4.除了用connect，还可以使用emit广播信号，可以相应多个槽函数:

在MainWindow类中添加”signals”选项，里面可以写信号的声明（**信号只需要声明，不需要实现！！！**）:

signals:

void dataChanged(const QString&data);

在其他的某处可以写:

emit MainWindow::dataChanged(“ddd”);//释放信号

#菜单栏：

包含<QMenuBar>,<QMenu>,<QAction>

菜单栏QMenubar包括很多菜单QMenu，每个菜单有一个标语，点击菜单后会出现一列按钮QAction.

QMenuBar\*menubar=this->**menuBar()**;//添加一个菜单栏

QMenu\*menu\_file=menubar**->addMenu**(“File”);//添加一个菜单

//在菜单里面添加两个按钮

QAction\*action\_open=menu\_file**->addAction**(“Open”);

QAction\*action\_exit=menu\_file->addAction(“Exit”);

//链接功能,注意QAction需要使用triggered信号

connect(action\_open,**&QAction::triggered**,this,&MainWindow::onOpenTriggered);

connect(action\_exit, &QAction::triggered, this, &MainWindow::close);

#工具栏：

包含<QToolBar><QAction>

QToolBar\*toolbar=this->**addToolBar(**“TOOL BAR”);//添加一个工具栏

toolbar->addAction(action\_open);

#状态栏:

包含<QStatusBar><QAction>

QStatusBar\*status\_bar=this->**statusBar()**;//添加一个状态栏

status\_bar->**showMessage(“READY...”,3000);**//显示3秒的信息

创建永久状态栏：包含<QLabel>

QLabel\*label=new QLabel(“PERMANENT MESSAGE”)**;**//创建永久标签

status\_bar**->addPermanentWidget**(label);//添加永久标签

#输入框：

包含<QLineEdit><QTextEdit><QPlainTextEdit>

1.单行输入框：

QLineEdit\*line\_edit=new QLineEdit(this);//添加一个单行输入框

line\_edit->**setPlaceholderText**(“Input your name”);//设置灰色提示文本

line\_edit->**setEchoMode**(<Mode>);//设置输入模式:QLineEdit::Normal默认，Password密码

QString name=line\_edit->**text()**;//获取输出内容

2.多行输入框（文本编辑器）:

QTextEdit\*text\_edit=new QTextEdit(this);//添加一个文本编辑器

this->**setCentralWidget**(text\_edit);//将编辑器放在正中间

QString message=text\_edit->**toPlainText()**;//获取输出内容，去除格式，仅返回文本

text\_edit->**setReadOnly(true)**;//设置成只读

text\_edit->**setText**(“原有文本”);//设置原有文本

text\_edit->**append**(“追加文本”);//追加文本

#复选框:

包含<QCheckBox>

QCheckBox\*cb=new QCheckBox(“Tick Me”,this);

cb->**setChecked**(true);//默认打钩

bool ischecked=cb->isChecked();//获取是否打钩信息

#标签：

包含<QLabel>

QLabel label(“LABEL”);//创建标签

label.**setText**(“LABEL2”);//修改名称

QString text=label.**text()**;//返回标签内容

#弹窗：

包含<QMessageBox>

一个包含“确定”按钮的信息弹窗。

静态方法创建：

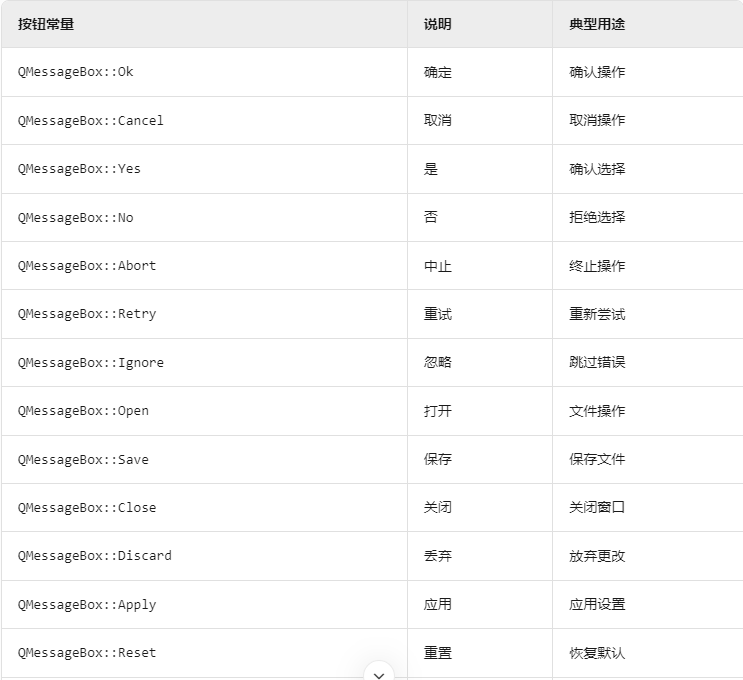
**QMessageBox::information(**nullptr,”Title”,”Content”**);提示弹窗**

**QMessageBox::warning(**nullptr,”Warning”,”Content”**)警告弹窗，**

**QMessageBox::critical(**nullptr,”Fault”,”Content”**)错误弹窗，**

**QMessageBox::question(**nullptr,”Ensure”,”Content”**,QMessageBox::Yes|QMessageBox:: No)确认弹窗,带有确认和取消**

其中标准按钮：



自定义按钮：

QMessageBox msgBox;

msgBox.**setText**("文件已修改");

msgBox.**addButton**("保存", **QMessageBox::AcceptRole**);

msgBox.addButton("不保存", **QMessageBox::RejectRole**);

msgBox.addButton("取消", **QMessageBox::DestructiveRole**);

//也可以使用标准按钮：

msgBox.setStandardButtons(QMessageBox::Save

|QMessageBox::Discard|QMessageBox::Cancel);

int ret = msgBox**.exec()**;//显示弹窗

if (ret == 0)

qDebug() << "用户选择保存";

else if (ret == 1) {

qDebug() << "用户选择不保存";

else

qDebug() << "用户取消";

按钮的属性值如下：



也可以使用标准按钮：

QMessageBox msgBox;

msgBox.setWindowTitle("警告");

msgBox.setText("文件未保存！");

msgBox.**setStandardButtons**(QMessageBox::Save | QMessageBox::Discard | QMessageBox::Cancel);

msgBox.**setDefaultButton**(QMessageBox::Save); // 设置默认按钮

int ret = msgBox.exec();

switch (ret) {

case QMessageBox::Save:

qDebug() << "用户选择保存";

break;

case QMessageBox::Discard:

qDebug() << "用户选择丢弃";

break;

case QMessageBox::Cancel:

qDebug() << "用户取消操作";

break;

}

#Qt架构的登录界面：

在主窗口类中，存在一个ui，可以在mainwindow.ui中设计窗口样式。

首先要在窗口中放一个StackedWidget：



将StackedWidget覆盖整改主窗口区域，这样就可以进行窗口的堆叠与切换。

随后可以拖拽一些控件到上面去：



主要控件：QLabel,QLineEdit,QPushButton.

注：1.QLineEdit可以编辑提示文字，在绿色的placeholderText中编辑；

2.QLabel可以编辑文字样式和大小，在右键此控件的“改变格式文本”中；

3.点击页面右上角的可以切换StackedWidget的页面；

4.编辑控件的名字，以便于在代码中书写，如按钮、编辑器名称，以及stackWidget的页面名称。

将控件与槽函数关联起来：

在主窗口类中添加public slots: ,槽函数的格式一律为**on\_<object>\_<signal>**,如:

on\_regBtn\_clicked();

on\_action\_add\_user\_triggered();

获取ui中的控件：使用**ui->**reg\_Btn之类即可获取控件指针。

切换StackedWidget的页面：如

ui->stackedWidget->**setCurrentWidget**(ui->login\_page);将页面切换到login\_page.

附：源代码（在Qt Creator中打开.pro）



#可滑动列表:  
 #include<QtableView>

#include<QstandardItemModel>

步骤:创建模型->填充数据->适应视图

*// 创建模型*

QStandardItemModel model(100, 3); *// 100行，3列*

model.**setHorizontalHeaderLabels**({"ID", "姓名", "邮箱"});//设定横坐标域

*// 填充数据*

for (int r = 0; r < 100; ++r) {

for (int c = 0; c < 3; ++c) {

QStandardItem\*it=new **QStandardItem(QString(“行%1,列%2”).arg(r).arg(c))**;

model.setItem(row, col, it);

}

}

*// 创建视图并绑定模型*

QTableView tableView;//这里可以使用自己ui上的tableview

tableView.setModel(&model);