QtCreator快速入门

# 介绍

Ctrl + 6 帮助

# 手动编译qt工程

目录下有hellodialog.ui和main.cpp

uic –o ui\_hellodialog.h hellodialog.ui 将.ui文件生成UI文件的.h文件

qmake –project 生成.pro文件

在.pro文件中，添加QT+= widgets

qmake 生成makefile文件和目录

mingw32-make 生成.exe

# 自定义C++类添加ui界面

将界面的ui文件添加入功能.ui和配套的.h文件

## 新建HelloDialog类

### h文件

#ifndef HELLODIALOG\_H

#define HELLODIALOG\_H

#include<QDialog>

namespace **Ui** {

class **Dialog**;//界面头文件的类，怕同名加一个命名空间

}

class **HelloDialog** : public QDialog

{

Q\_OBJECT

public:

explicit **HelloDialog**(QWidget\* parent=0);

~***HelloDialog***();

private:

Ui::Dialog\* ui;

};

#endif // HELLODIALOG\_H

### cpp文件

#include "hellodialog.h"

#include "ui\_hellodialog.h"

HelloDialog::**HelloDialog**(QWidget\* parent):QDialog(parent)

{

ui = new Ui::Dialog;//新建界面

ui->setupUi(this);//启动界面

}

HelloDialog::~***HelloDialog***(){

delete ui;

}

# 窗口类型

Qt::WindowStates

Qt::Dialog|Qt::FramelessWindowHint //无边框对话框

Qt::SplashScreen|Qt::WindowStaysOnTopHint//总是最前的欢迎界面

# 窗口状态

Qt:WindowState

WindowNoState = 0x00000000,

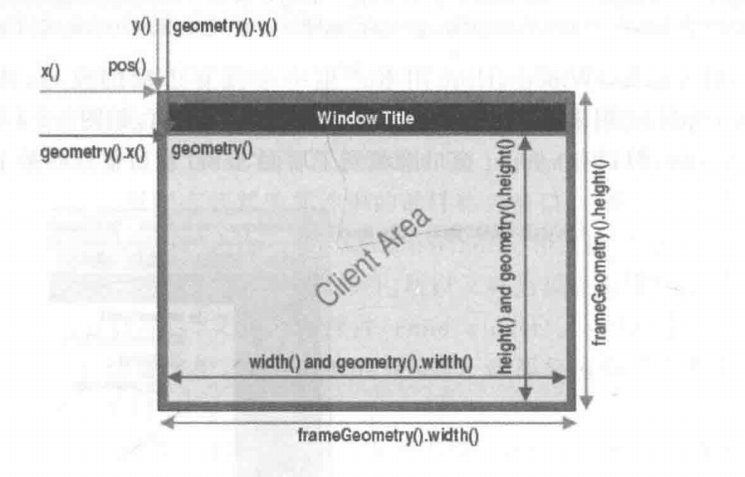
WindowMinimized = 0x00000001,

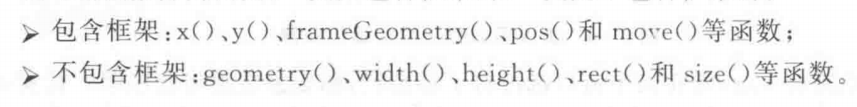
WindowMaximized = 0x00000002,

WindowFullScreen = 0x00000004,

WindowActive = 0x00000008

# 窗口几何布局





# 模态和非模态对话框

MyWidget::**MyWidget**(QWidget \*parent) :

QWidget(parent),

ui(new Ui::MyWidget)

{

ui->setupUi(this);

QDialog \*dialog = new QDialog(this);

dialog->setModal(true);//设置模态对话框

dialog->show();

}

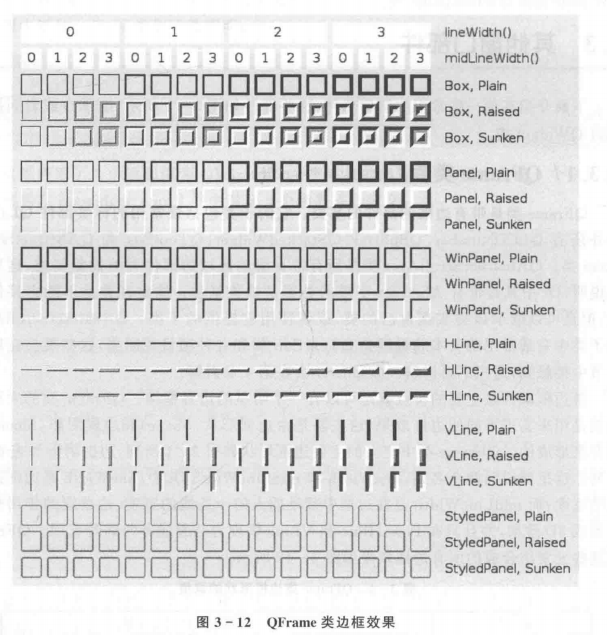
Qt::NonModal

Qt::WindowModal 阻塞它的父窗口

Qt::ApplicationModal

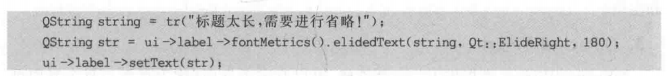
# QFrame边框效果





# QLabel





wordWrap属性：自动换行

QFontMetrics类：计算给定字体的字符或字符串大小

通过QWidget::fontMetrics()获取QFontMetrics对象。

elidedText():进行省略文本。

Qt::ElideLeft …出现在文本开头

Qt::ElideMiddle …出现在文本中间

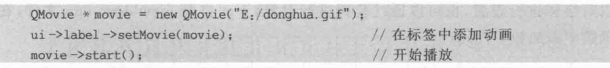
Qt::ElideRight …出现在文本最后

显示图片：



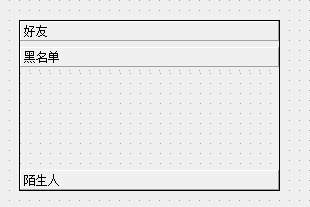


显示gif：



# QToolBox

QQ中的抽屉效果



# 按钮部件

checkable属性：拥有选中和未选中2中状态

flat属性：不显示边框

toggled（bool）：显示按钮是否处于被按下状态

## 添加加速键

ui->pushBtn1->setText(tr("&nihao")); // 这样便指定了Alt+N为加速键

显示&，使用“&&”

## 添加图标

ui->pushBtn2->setText(tr("帮助(&H)"));

ui->pushBtn2->setIcon(QIcon("../mybutton/images/help.png"));

## 添加下拉菜单

ui->pushBtn3->setText(tr("z&oom"));

QMenu \*menu = new QMenu(this);

menu->addAction(QIcon("../mybutton/images/zoom-in.png"), tr("放大"));

ui->pushBtn3->setMenu(menu);

## CheckBox三态



1. 未选中
2. 保持不变
3. 选中



# QLineEdit

## 显示模式（echoMode）

Normal：正常

NoEcho：不显示输入

password：以小黑点或星号显示

passwordEchoOnEdit：编辑时正常显示

## 输入掩码



## 获得内容

qDebug() << ui->lineEdit2->text(); // 输出lineEdit2的内容qDebug() << ui->lineEdit2->displayText(); // 输出lineEdit2显示的内容

## 输入验证

### 整数

QValidator \*validator = new QIntValidator(100, 999, this);

ui->lineEdit3->setValidator(validator);

### double

QValidator \*validator = new QDoubleValidator(100, 999, this);

### 正则表达式

RegExp rx("-?\\d{1,3}");

QValidator \*validator = new QRegExpValidator(rx, this);

// 在行编辑器中使用验证器

ui->lineEdit3->setValidator(validator);

## 自动补全

QStringList wordList;

wordList << "Qt" << "Qt Creator" << tr("你好");

QCompleter \*completer = new QCompleter(wordList, this); // 新建自动完成器

completer->setCaseSensitivity(Qt::CaseInsensitive); // 设置大小写不敏感

ui->lineEdit4->setCompleter(completer);

# QdateTimeEdit

ui->dateTimeEdit->setDateTime(QDateTime::currentDateTime());

// 设置时间的显示格式

ui->dateTimeEdit->setDisplayFormat(tr("yyyy年MM月dd日ddd HH时mm分ss秒"));

# QAbstractSlider

value与sliderPosition：当前值

tracking：是否跟踪，是否没移动一刻度触发valueChanged信号

orientation:设置方向

wrapping：是否首尾相连

# 布局

## 基本布局管理器（QBoxLayout）

使子部件在水平或垂直方向排成一列。它将所有空间分成一行盒子，然后把每个部件放入一个盒子中。

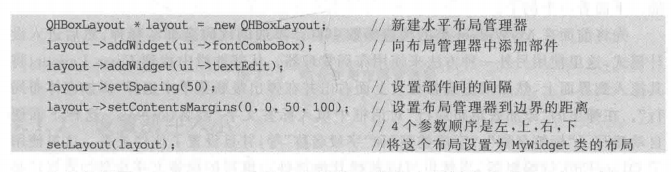
QHBoxLayout:水平布局

QVBoxLayout：垂直布局



addWidget函数在末尾添加部件

insertWidget函数实现向指定位置添加部件



## 栅格布局管理器（QGridLayout）

将空间分隔成一些行和列，形成单元格，将部件放入单元格中。

ui->fontComboBox->setParent(0);

ui->fontComboBox->setParent(0);

ui->fontComboBox->setParent(0);

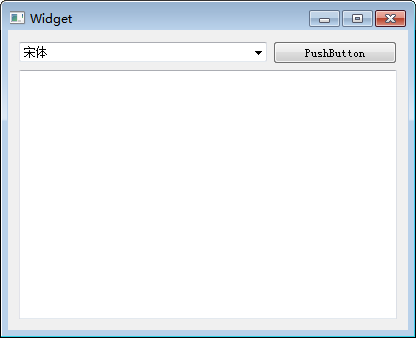
QGridLayout\* layout=new QGridLayout;

layout->addWidget(ui->fontComboBox,0,0,1,2);

layout->addWidget(ui->pushButton,0,2,1,1);

layout->addWidget(ui->textEdit,1,0,1,3);

setLayout(layout);

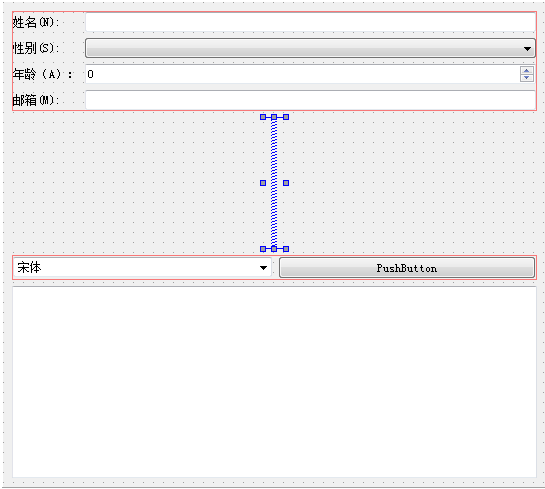


## 窗体布局管理器（QFormLayout）



## 综合布局

使用父布局管理器的addLayout()函数实现子布局放入一个父布局管理器。



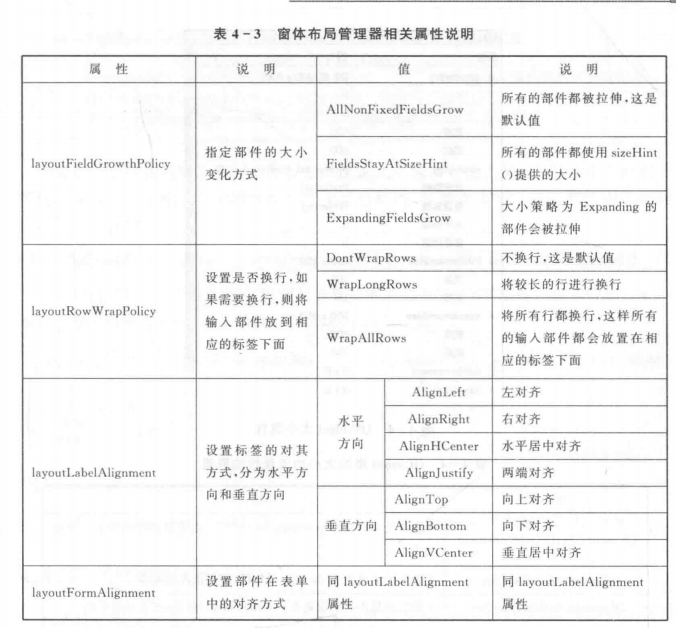
## 设置部件大小

sizeHint:大小提示。保存了部件的建议大小。

minimumSizeHint:最小大小提示。保存了一个建议的最小大小提示。

sizePolicy:大小策略属性。保存了部件的默认布局行为，在水平和垂直两个方向分别起作用。







## 可扩展窗口

void Widget::**on\_pushButton\_toggled**(bool checked)

{

ui->textEdit->*setVisible*(checked);

if(checked)

ui->pushButton->setText("隐藏可扩展窗口");

else

ui->pushButton->setText("显示可扩展窗口");

}

## 设置伙伴

QLabel与其他部件关联。提供助记符来定位键盘焦点对应的部件上。

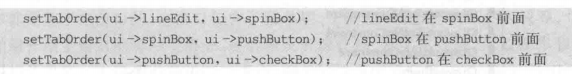
设计界面，使用伙伴设计模式可以关联。

QLabel的setBuddy()函数也可以关联。

## 设置tab键顺序

设计界面，进入Tab顺序模式，单击数字更改顺序。

代码实现：



# 应用程序主窗口

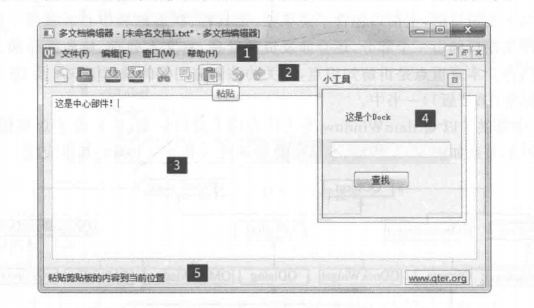
1菜单栏（QMenuBar）

2工具栏（QToolBar）

3中心部件（CentralWidget）

4Dock部件（QDockWidget）

5状态栏（QStatusBar）



## 菜单栏

设计界面可直接输入菜单项

Action编辑器中自动出现动作，双击动作可编辑动作信息

可直接将动作拖拽至工具栏

代码实现：

QMenu\* editMenu=ui->menuBar->addMenu("编辑(&E)");

QAction\* action\_open=editMenu->addAction(

QIcon(":/image/images/open.png"),

"打开文件(&O)");

action\_open->setShortcut(QKeySequence("Ctrl+O"));

ui->mainToolBar->addAction(action\_open);

ui->menuBar 获取菜单栏

ui->mainToolBar获取工具栏

## 使用资源

添加QtResourceFile

在项目文件夹下，新建文件夹images

复制资源在images文件夹下

资源管理界面->添加按钮->添加前缀->/image

添加按钮->添加文件按钮->选择要添加的资源->打开按钮

### 资源文件内容

<RCC>

<qresource prefix="/image">

<file>images/new.png</file>

<file>images/open.png</file>

</qresource>

</RCC>

### 工程文件添加资源文件



## QActionGroup类

一组动作互斥，只能有一个动作处于选中状态。

QActionGroup\* group= new QActionGroup(this);

QAction\* action\_L=group->addAction("左对齐");

QAction\* action\_R=group->addAction("右对齐");

QAction\* action\_C=group->addAction("居中");

action\_L->setCheckable(true);

action\_R->setCheckable(true);

action\_C->setCheckable(true);

action\_L->setChecked(true);

editMenu->addSeparator();

editMenu->addAction(action\_L);

editMenu->addAction(action\_R);

editMenu->addAction(action\_C);

## 工具栏

toolButtonStyle属性:设置图标和相应文本的显示

moveabel属性：是否可移动

allowedArea属性：用来设置允许停靠的位置

iconsize属性：用来设置图标大小

floatable属性：是否可悬浮

### 工具栏中添加其他部件

QToolButton\* toolBtn=new QToolButton(this);//创建QToolButton对象

toolBtn->setText("颜色");

//往工具栏添加按钮

QMenu\* colorMenu = new QMenu(this);

colorMenu->addAction("红色");

colorMenu->addAction("绿色");

toolBtn->setMenu(colorMenu);

toolBtn->setPopupMode(QToolButton::MenuButtonPopup);

ui->mainToolBar->addWidget(toolBtn);//往工具栏添加按钮

QSpinBox\* spinBox=new QSpinBox(this);

ui->mainToolBar->addWidget(spinBox);

## 中心区域

QMdiArea部件用来提供一个可以显示MDI多文档界面的区域，从而有效地管理多个窗口。

1. 界面设计中拖入一个mdiArea部件。
2. 新建“新建文件”动作，添加动作响应函数。

void myMainWindow::**on\_actionnew\_triggered**()

{

QTextEdit\* edit=new QTextEdit(this);

QMdiSubWindow\* child=ui->mdiArea->addSubWindow(edit);

child->show();

}

## dock部件

1. 界面设计拖入一个Dock Widget部件
2. 属性：标题、是否可以关闭、移动、悬浮、停靠区域设置
3. 关闭后显示停靠窗口ui->dockWidget->show();

void myMainWindow::**on\_actiondisplaydock\_triggered**()

{

ui->dockWidget->show();

}

## 状态栏

QStatusBar类提供了一个水平部件，用来显示状态信息。

3类信息：

* 临时信息：一般的提示信息showMessage（）
* 正常信息：显示页数和行号
* 永久信息：显示版本号或日期addPermanentWidget()添加如QLabel的部件

状态栏最右端有一个QSizeGrip部件，用来调整窗口的大小

使用setSizeGripEnabled（）函数禁用它

## 自定义菜单

工具栏可以添加其他部件。

QWidgetAction类提供了在菜单中使用其他部件的功能

操作：

自定义菜单项**myAction(包含一个标签和编辑器)**继承QWidgetAction

**myAction.h:**

#include<QWidgetAction>

class **QLineEdit**;

class **myAction** : public QWidgetAction

{

Q\_OBJECT

public:

explicit **myAction**(QObject\* parent=0);

protected:

QWidget\* ***createWidget***(QWidget\* parent);//将部件添加进菜单

signals:

//新建信号，将编辑器中的内容发射出去

void **getText**(const QString& string);

private slots:

//新建槽，与按下回车键信号关联

void **sendText**();

private:

QLineEdit\* lineEdit;

};

**myAction.cpp:**

myAction::**myAction**(QObject\* parent):

QWidgetAction(parent)

{

lineEdit=new QLineEdit;

//按下回车键后，触发sendText函数

connect(lineEdit,&QLineEdit::returnPressed,this,&myAction::sendText);

}

QWidget\* myAction::***createWidget***(QWidget \*parent)

{

//检查父部件是否为QMenu或者QToolBar

if(parent->inherits("QMenu")||parent->inherits("QToolBar"))

{

QSplitter\* splitter=new QSplitter(parent);

QLabel\* label=new QLabel;

label->setText("插入文本：");

splitter->addWidget(label);

splitter->addWidget(lineEdit);

return splitter;

}

}

void myAction::**sendText**()

{

emit getText(lineEdit->text());//发射编辑器内容

lineEdit->clear();//将编辑器内容清空

}

mainwindow.h:

//添加槽，项编辑器添加文本

private slots:

void **setText**(const QString& string);

mainwindow.cpp:

myMainWindow::**myMainWindow**(QWidget \*parent) :

QMainWindow(parent),

ui(new Ui::myMainWindow)

{

ui->setupUi(this);

myAction\* action=new myAction;

QMenu\* editMenu=ui->menuBar->addMenu("编辑");

//添加自定义菜单

editMenu->addAction(action);

//自定义菜单的编辑完成信号与主界面编辑器关联

connect(action,SIGNAL(getText(QString)),this,SLOT(setText(QString)));

}

myMainWindow::~***myMainWindow***()

{

delete ui;

}

void myMainWindow::**setText**(const QString &string)

{

ui->textEdit->setText(string);

}

# 富文本处理

编辑操作：使用基于光标的一些接口，不会丢失底层的文档框架。

只读操作：只读的分层次的接口函数，有利于文档的检索和输出。

文档光标:QTextCursor

文档框架:QTextDocument

富文本文档结构：

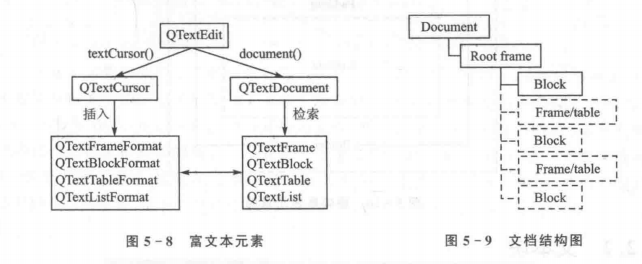
框架：QTextFrame QTextFrameFormat

文本块：QTextBlock QTextBlockFormat

表格：QTextTable QTextTableFormat

列表：QTextList QTextListFormat

QTextEdit类是一个富文本编辑器，构建它时同时构建了一个QTextDocument和QTextCursor。



## root frame

一个空文档包含一个root frame。重新设置rootFrame。

QTextDocument\* document=ui->textEdit->document();

QTextFrame\* rootFrame=document->rootFrame();

QTextFrameFormat format;

format.setBorderBrush(Qt::red);

format.setBorder(3);

rootFrame->setFrameFormat(format);

这里还可以用setHeight（）和setWidth（）来固定rootFrame的宽度和高度。

在rootFrame中添加一个子框架

QTextFrameFormat frameFromat;

frameFromat.setBackground(Qt::lightGray);

frameFromat.setMargin(10);

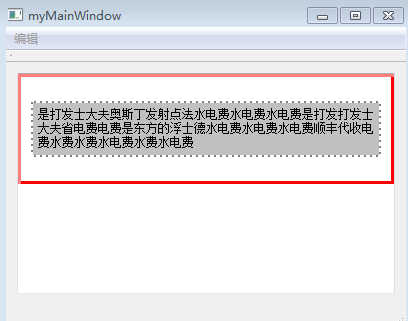
frameFromat.setPadding(5);

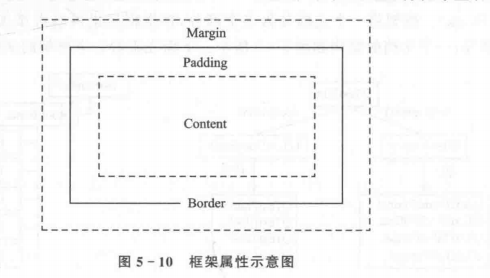
frameFromat.setBorder(2);

frameFromat.setBorderStyle(QTextFrameFormat::BorderStyle\_Dotted);

QTextCursor cursor=ui->textEdit->textCursor();

cursor.insertFrame(frameFromat);





## 文本块

QTextBlock类为文本文档QTextDocument类提供了一个文本片段（QTextFragment）的容器。

一个文本块可以看作一个段落

一个回车换行就表示创建一个新的文本块

QTextBlock提供了只读接口（文档分层次的接口的一部分）

QTextFrame看作一层，QTextBlock就是另一层。

文本块的格式由QTextBlockFormat类来处理

* + - 对齐方式
    - 四周边距
    - 缩进

文本块中的文本格式由QTextCharFormat类来设置

* 字体大小
* 加粗
* 下划线

### 遍历框架

输出根框架中的文本块内容

void myMainWindow::**showTextFrame**()

{

QTextDocument\* document=ui->textEdit->document();

QTextFrame\* frame=document->rootFrame();

//建立QTextFrame类的迭代器

QTextFrame::iterator it;

for(it=frame->begin();!(it.atEnd());++it)

{

//获取当前框架指针

QTextFrame\* childFrame =it.currentFrame();

//获取当前文本块指针

QTextBlock childBlock=it.currentBlock();

if(childFrame)

qDebug()<<"frame";

else if(childBlock.isValid())

qDebug()<<"block:"<<childBlock.text();

}

}

### 遍历文本块

void myMainWindow::**showTextBlock**()

{

QTextDocument\* document=ui->textEdit->document();

QTextBlock block=document->firstBlock();

for(int i=0;i<document->blockCount();i++)

{

qDebug()<<QObject::tr("文本块%1,文本块首行行号：%2,长度为：%3，内容为").arg(i).arg(block.firstLineNumber()).arg(block.length())<<block.text();

block=block.next();

}

}

QTextDocument::firstBlock函数获取文档的第一个文本块

QTextDocument::blockCount函数获取文档中所有文本块的个数

QTextBlock::next函数获取下一个文本块

QTextBlock::firstLineNumber第一行行号

QTextBlock::length长度（文本块的长度从1开始计算，什么都不不写是1，比实际字符多1）

QTextBlock::text文本块内容

## 字体

void myMainWindow::**setTextFont**(bool checked)

{

if(checked){

QTextCursor cursor=ui->textEdit->textCursor();

QTextBlockFormat blockFormat;

blockFormat.setAlignment(Qt::AlignCenter);

cursor.insertBlock(blockFormat);

QTextCharFormat charFormat;

charFormat.setBackground(Qt::lightGray);

charFormat.setForeground(Qt::blue);

charFormat.setFont(QFont("宋体",12,QFont::Bold,true));

charFormat.setFontUnderline(true);

cursor.setCharFormat(charFormat);

cursor.insertText("测试字体");

}

}

QTextBlockFormat:: setAlignment

QTextCharFormat:: setBackground

QTextCharFormat:: setForeground

QTextCharFormat:: setFont

QTextCharFormat:: setFontUnderline

QTextCursor:: setCharFormat

QTextCursor::insertText

## 表格

可以用QTextFrame::iterator来遍历

QTextTable::cellAt()获取制定单元格

QTextTable::insertColumns()函数用来插入列

QTextTable::insertRows()函数插入行

QTextTable::mergeCells()函数用来合并单元格

QTextTable::splitCell()函数用来拆分单元格

QTextTableCell类对应一个单元格，

QTextTableCellFormat类对应单元格的格式

void myMainWindow::**insertTable**()

{

QTextCursor cursor=ui->textEdit->textCursor();

QTextTableFormat format;

format.setCellSpacing(2);

format.setCellPadding(10);

cursor.insertTable(2,2,format);

}

## 列表

QTextList类对应列表

QTextList::count()获取列表中的项目个数

QTextList::item()获取指定项目的文本块

removeItem()删除一个项目

数字编号选项通过QTextListFormat::Style关键字查看

void myMainWindow::**insertList**()

{

QTextListFormat format;

format.setStyle(QTextListFormat::ListDecimal);

ui->textEdit->textCursor().insertList(format);

}

## 图片

QTextImageFormat::setHeight和setWidth函数设置图片的高度和宽度。

setName函数制定图片

void myMainWindow::**insertImage**()

{

QTextImageFormat format;

format.setName("D:\\chenxin\\QtProject\\MyMainWindow2\\images\\logo.png");

ui->textEdit->textCursor().insertImage(format);

}

## 查找功能

void myMainWindow::**findNext**()

{

QString string=lineEdit->text();

bool isfind = ui->textEdit->find(string);

if(isfind){

QPalette palette = ui->textEdit->palette();

palette.setColor(QPalette::Highlight,palette.color(QPalette::Active,QPalette::Highlight));

ui->textEdit->setPalette(palette);

}else{

QMessageBox::information(this,tr("注意"),tr("没有找到内容"),QMessageBox::Ok);

}

}

## 语法高亮

将编辑器的document指针作为参数，这样每当编辑器中文本改变时都会调用highlightBlock（）函数

highlighter=new MySyntaxHighlighter(ui->textEdit->document());

Qt提供了QSyntaxHighlighter类来实现语法高亮。

#include<QSyntaxHighlighter>

class **MySyntaxHighlighter** : public QSyntaxHighlighter

{

Q\_OBJECT

public:

**MySyntaxHighlighter**(QTextDocument\* parent=0);

protected:

void ***highlightBlock***(const QString &text);

};

MySyntaxHighlighter::**MySyntaxHighlighter**(QTextDocument \*parent):

QSyntaxHighlighter (parent)

{

}

void MySyntaxHighlighter::***highlightBlock***(const QString &text)

{

QTextCharFormat format;

format.setFontWeight(QFont::Bold);

format.setForeground(Qt::green);

QString pattern="\\bchar\\b";//匹配字符char

QRegExp expression(pattern);

int index=text.indexOf(*expression*);

while(index>=0){

int length=expression.matchedLength();

setFormat(index,length,format);//对匹配的字符串设置格式

index=text.indexOf(*expression*,index+length);

}

}

## HTML

支持使用HTML标签

ui->textEdit->append("<h1><font color=red>使用HTML</font></h1>");

## 富文本的帮助文档

帮助文档Advanced Rich Text Processing

查看Text Edit程序

# 拖放操作

设置主窗口接收拖拽事件

setAcceptDrops(true);

//拖拽一个数据进入主窗口时，触发***dragEnterEvent处理函数***

void MainWindow::***dragEnterEvent***(QDragEnterEvent \*event)

{

//获取MIME数据查看是否包含URL路径

if(event->mimeData()->hasUrls())

event->acceptProposedAction();

else {

event->ignore();

}

}

//

void MainWindow::***dropEvent***(QDropEvent \*event)

{

const QMimeData\* mimeData=event->mimeData();

if(mimeData->hasUrls()){

QList<QUrl>urlList=mimeData->urls();

QString fileName=urlList.at(0).toLocalFile();

if(!fileName.isEmpty()){

QFile file(fileName);

if(!file.*open*(QIODevice::ReadOnly))

return;

QTextStream in(&file);

ui->textEdit->setText(in.readAll());

}

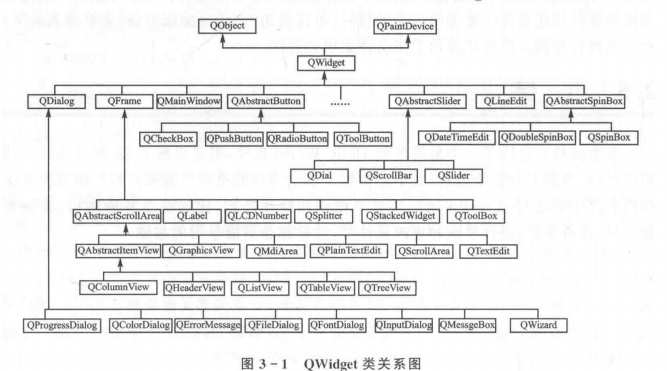
}

}

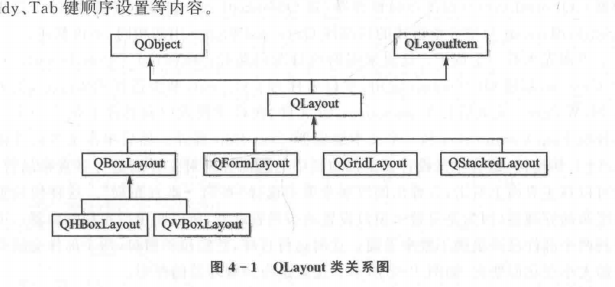
# 附录

## QT类之间的关系

### QWidget类的关系图



### QLayout类关系图



### 窗口类的关系图

