# 1.窗口类的种类

|  |
| --- |
|  |

# 2.QWidget

## 没有菜单栏，工具栏，状态栏，比较自由在QT开发中比较常用

|  |
| --- |
|  |

# 3.QMainWindow

## 有菜单栏，工具栏，状态栏和内容区域

|  |
| --- |
|  |

# 4.QDialog

## 对话框窗口，一般不作为顶级窗口，可以 中文消息窗口，不过它是可以作为顶级窗口的

### 示例代码

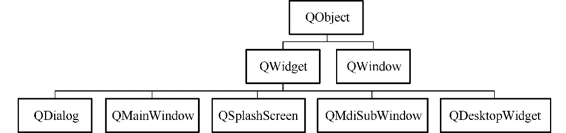
|  |
| --- |
| **import** sys **from** PyQt5.QtWidgets **import** QDialog, QApplication, QLabel,QPushButton,QMessageBox **from** PyQt5.QtGui **import** QPixmap   **class** MyWidget(QDialog):  **def** \_\_init\_\_(self):  super().\_\_init\_\_()  self.setupUI()   **def** setupUI(self):  self.lbl = QLabel(**""**, self)  self.lbl.setPixmap(QPixmap(**'cocktail.png'**))  self.setWindowTitle(**"QDialog Demo"**)  self.setFixedSize(650, 400)  *# 创建两个按钮* btnOK = QPushButton(**"确定"**,self)  btnOK.move(150,340)  btnOK.clicked.connect(self.onOk)  btnCl = QPushButton(**"取消"**,self)  btnCl.move(300,340)  btnCl.clicked.connect(self.onCancel)    **def** onOk(self):  QMessageBox.information(**None**,**"info"**,**"你点击了确定按钮"**)   **def** onCancel(self):  QMessageBox.information(**None**,**"info"**,**"你点击了取消按钮"**)   **if** \_\_name\_\_ == **'\_\_main\_\_'**:  app = QApplication(sys.argv)  w = MyWidget()  w.show()  app.exec() |

# Qt窗口的参考文档1

# [QT学习：QWidget、QDialog、QMainWindow等窗体类的区别和用途](https://www.cnblogs.com/zjacky/p/16723170.html)

## 各窗体之间继承的关系

* QWidget 直接继承于 QObject
* QDialog\QMainWindow 继承于 QWidget
* 其他继承于 QWidget 的窗体类还有 QSplashScreen、QMdiSubWindow 和 QDesktopWidget
* 还有一个窗口类 QWindow ，它同时继承于 QObject 和 QSurface

继承关系如下图所示：  


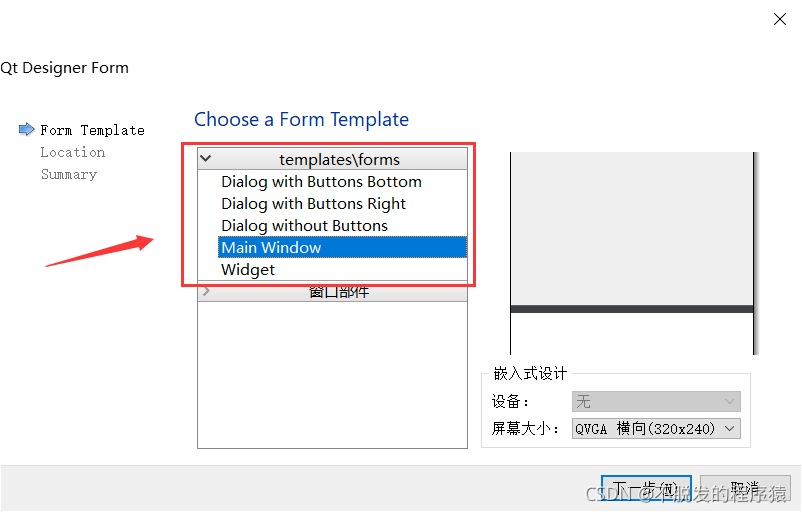
## 这些窗体类的特点和用途

* QWidget：在没有指定父容器时可作为独立的窗口，指定父容器后可以作为容器的内部组件
* QDialog：用于设计对话框，以独立窗口显示
* QMainWindow：用于设计带有菜单栏、工具栏、状态栏的主窗口，一般以独立窗口显示
* QSplashScreen：一般在大型应用程序启动时会显示一个启动画面，这个启动画面即Splash窗口
  + 特点：Splash窗口一般被设计为一个无边对话框，在Windows任务栏不显示当前窗口的图标信息。
  + 我们通常在Splash窗口中只显示一个图片，并展示软件的信息等
  + 使用场景：Splash窗口显示的时候，程序一般在后台做一些比较耗时的启动准备工作，等到软件的准备工作完成之后，Splash窗口就会关闭，然后软件的主窗口显示出来
* QMdiSubWindow：用于为QMdiArea提供一个子窗体，用于MDI（多文档）应用程序的设计。
* QDesktopWidget：具有多个显卡和多个显示器的系统具有多个桌面，这个类提供用户桌面信息，如屏幕个数、每个屏幕的大小等。
* QWindow：通过底层的窗口系统表示一个窗口的类，一般作为一个父容器的嵌入式窗体，不作为独立窗体。

# Qt窗口的参考文档2

# Python Qt GUI设计：QMainWindow、QWidget和QDialog窗口类

QMainWindow、QWidget和QDialog三个类都是用来创建窗口的，可以直接使用，也可以继承后再使用，在Qt Designer创建UI文件可以选择这三种窗口类型。



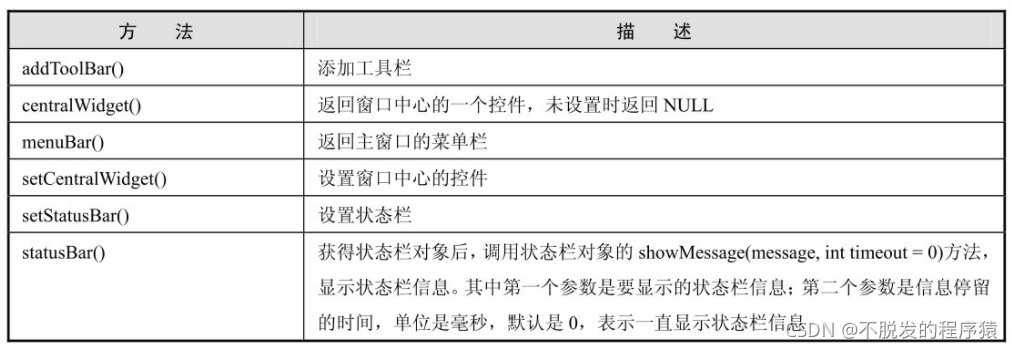
如果是主窗口，就使用QMainWindow类，如果是对话框，就使用QDialog类，如果不确定，或者有可能作为顶层窗口，也有可能嵌入到其他窗口中，那么就使用QWidget类。

让我们看看具体区别吧~

**1、 QMainWindow窗口**

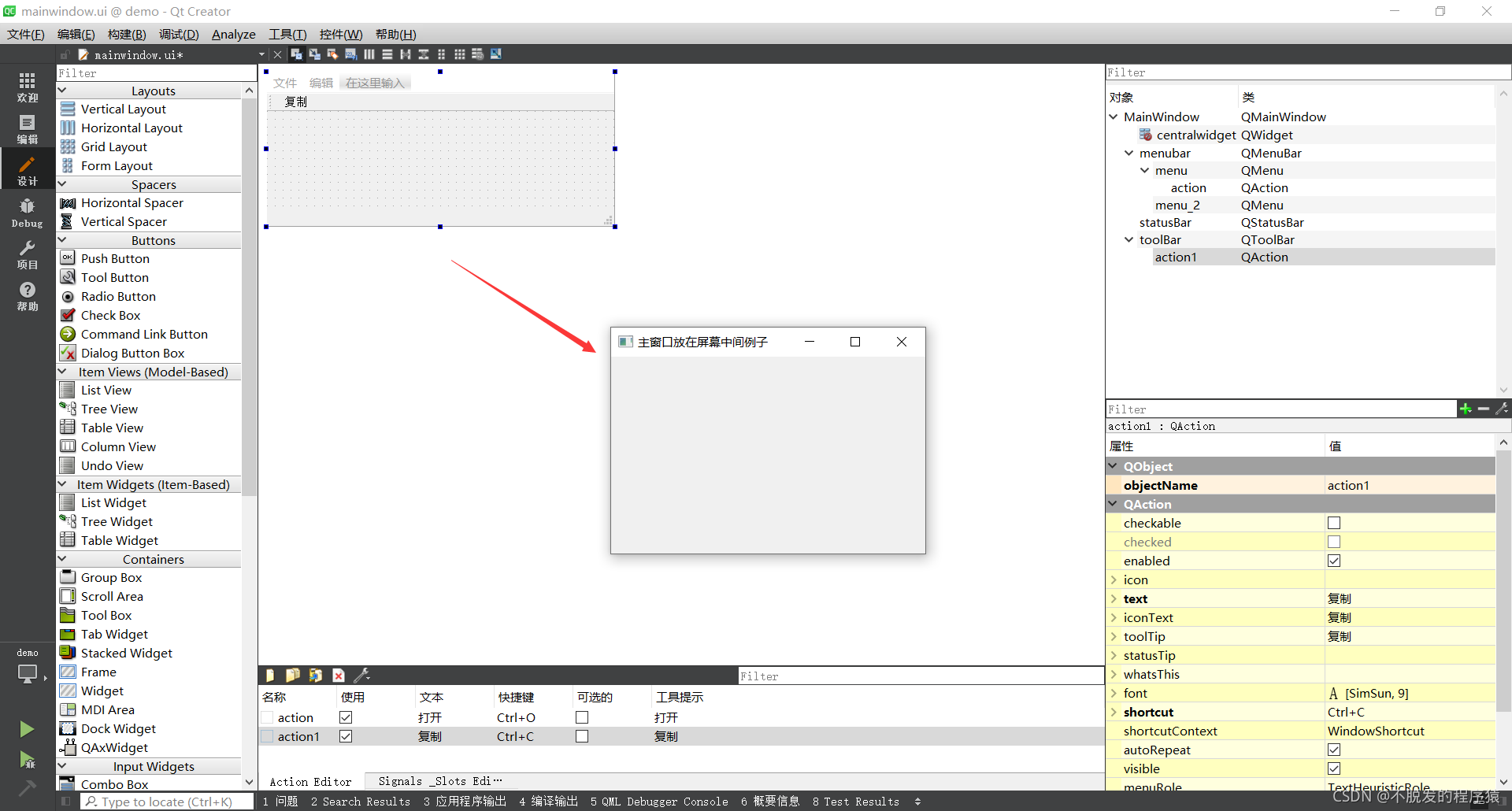
QMainWindow主窗口为用户提供一个应用程序框架，它有自己的布局，可以在布局中添加控件。在主窗口中可以添加控件，比如将工具栏、菜单栏和状态栏等添加到布局管理器中。

QMainWindow类中比较重要的方法如下表所示：



QMainWindow有自己的布局不能设置布局（使用setLayout()方法）。

聊一个实用功能，当主窗口打开运行后，实现默认电脑屏幕居中，效果如下所示：



实现代码如下所示：

python

复制代码

**from** PyQt5.QtWidgets **import** QDesktopWidget, QApplication ,QMainWindow

**import** sys

**class** **Winform**( **QMainWindow**):

**def** **\_\_init\_\_**(self, parent=None):

super( Winform, self).\_\_init\_\_(parent)

self.setWindowTitle('主窗口放在屏幕中间例子')

self.resize(370, 250)

self.center()

**def** **center**(self):

screen = QDesktopWidget().screenGeometry()

size = self.geometry()

self.move((screen.width() - size.width()) / 2, (screen.height() - size.height()) / 2)

**if** \_\_name\_\_ == "\_\_main\_\_":

app = QApplication(sys.argv)

win = Winform()

win.show()

sys.exit(app.exec\_())

以下语句用来设置QWidget窗口的大小，宽度为370像素，高度为250像素。

python

复制代码

self.resize(370, 250)

以下语句用来计算显示屏幕的大小：(screen.width()\*screen.height())，其中  
QDesktopWidget是描述显示屏幕的类，通过QDesktopWidget().screenGeometry()来获得屏幕的大小。

python

复制代码

screen = QDesktopWidget().screenGeometry()

以下语句用来获取QWidget窗口的大小：(size.width()\*size.heiget())。

python

复制代码

size = self.geometry()

以下语句将窗口移动到屏幕中间。

python

复制代码

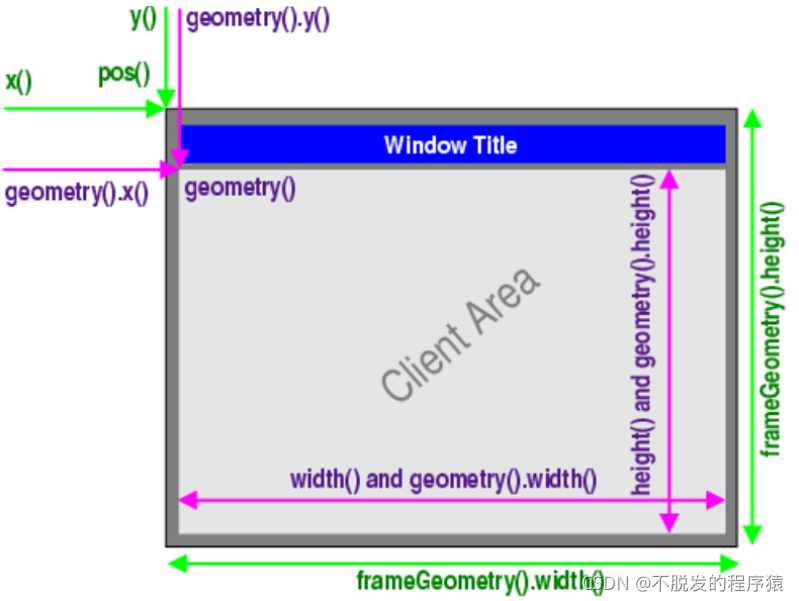
self.move((screen.width() - size.width()) / 2, (screen.height() - size.height()) / 2)

**2、QWidget窗口**

基础窗口控件QWidget类是所有用户界面对象的基类，所有的窗口和控件都直接或间接继承自QWidget类。

窗口控件(Widget，简称“控件”)是在PyQt中建立界面的主要元素。在PyQt中把没有嵌入到其他控件中的控件称为窗口，一般窗口都有边框、标题栏。窗口是指程序的整体界面，可以包含标题栏、菜单栏、工具栏、关闭按钮、最小化按钮、最大化按钮等；控件是指按钮、复选框、文本框、表格、进度条等这些组成程序的基本元素。一个程序可以有多个窗口，一个窗口也可以有多个控件。

在Qt官方提供的帮助文档中：“Qt 5.14 Qt Widgets Window and Dialog Widgets”，可见QWidget几何结构如下所示：



从上图可以看出，这些成员函数分为三类：

* QWidget直接提供的成员函数: x()、y()获得窗口左上角的坐标，width()、 height()获得客户区的宽度和高度；
* QWidget的geometry()提供的成员函数:x()、y()获得客户区左上角的坐标，width()、 height()获得客户区的宽度和高度；
* Widget的frameGeometry()提供的成员函数:x()、y()获得窗口左上角的坐标，width()、 height()获得包含客户区、标题栏和边框在内的整个窗口的宽度和高度。

而且可以知道QWidget有两种常用的几何结构：

* 不包含外边各种边框的几何结构；
* 包含外边各种边框的几何结构。

**1、QWidget不包含边框的常用函数**

一般情况下，不包含边框的部分是客户区，这里面就是我们正常操作的地方，可以添加子控件。这部分是一个长方形，会有大小和位置。大小就是指宽度（width）和高度（height）；位置就是指这个长方形在屏幕上的位置。在Qt中保存这个长方形使用的是QRect类，这个类也有自己的大小和位置。要改变其大小和位置，可以使用如下几个函数：

* **改变客户区的面积**

以下两个函数改变了长方形的大小，其中第一个参数是宽度，第二个参数是高度。设置了大小的窗口，还可以用鼠标来改变它的大小。

QWidget.resize(width,height)

QWidget.resize(QSize)

* **获得客户区的大小**

Qwidget.size()

* **获得客户区的宽度和高度**

QWidget.width()

QWidget.height()

* **设置客户区的宽度和高度**

使用这个函数，客户区的高度就是固定的，不可以改变，只可以改变宽度。

QWidget.setFixedWidth(**int** width)

这时候宽度就是固定的，不可以改变，但是可以改变高度。

QWidget.setFixedHeight(**int** height)

以下这两个函数，高度和宽度都是固定的，不可以通过鼠标来改变窗口的宽度和高度。

QWidget.setFixedSize(QSize size)

QWidget.setFixedSize(**int** width,**int** height)

如果要同时改变客户区的大小和位置，需要用到以下函数。

QWidget.setGeometry(**int** x, **int** y,**int** width,**int** height)

Widget.setGeometry(QRect rect)

x和y对应的就是x和y坐标，也可以不单独设置x和y坐标。

**2、QWidget包含边框的常用函数**

QWidget包含边框，这个边框有大小和位置，是窗口在屏幕上显示的整个区域。

这里没有设置这个边框大小的函数，因为通过上面不包含边框函数的设置，就可以设置包含边框的大小了。通过下面的函数，可以获得整个窗口的位置和大小。

* **获得窗口的大小和位置**

QWidget.frameGeometry()

* **设置窗口的位置**

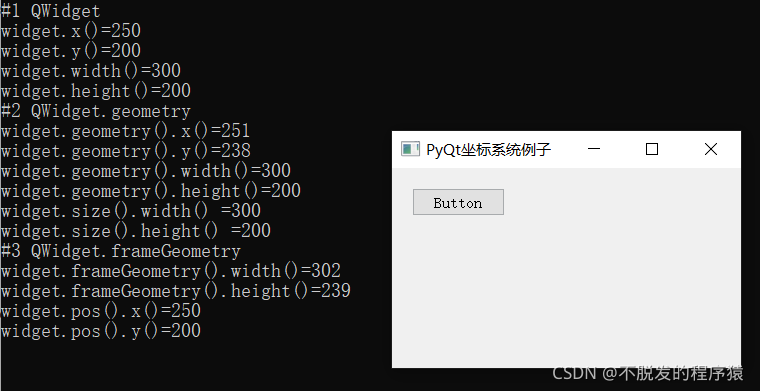
QWidget.move(**int** x,**int** y)

QWidget.move(QPoint point)

* **获得窗口左上角的坐标**

QWidget.pos()

最后将上述函数API整合成一个示例，获取QWidget控件在屏幕上的坐标，效果如下所示：



**from** PyQt5.QtWidgets **import** QApplication ,QWidget ,QPushButton

**import** sys

app = QApplication(sys.argv)

widget = QWidget()

btn = QPushButton( widget )

btn.setText("Button")

*#以QWidget左上角为(0, 0)点*

btn.move(20, 20)

*#不同操作系统可能对窗口最小宽度有规定，若设置宽度小于规定值，则会以规定值进行显示*

widget.resize(300, 200)

*#以屏幕左上角为(0, 0)点*

widget.move(250, 200)

widget.setWindowTitle('PyQt坐标系统例子')

widget.show()

print("#1 QWidget")

print("widget.x()=%d" % widget.x() )

print("widget.y()=%d" % widget.y() )

print("widget.width()=%d" % widget.width() )

print("widget.height()=%d" % widget.height() )

print("#2 QWidget.geometry")

print("widget.geometry().x()=%d" % widget.geometry().x() )

print("widget.geometry().y()=%d" % widget.geometry().y() )

print("widget.geometry().width()=%d" % widget.geometry().width() )

print("widget.geometry().height()=%d" % widget.geometry().height() )

print("widget.size().width() =%d" % widget.size().width() )

print("widget.size().height() =%d" % widget.size().height() )

print("#3 QWidget.frameGeometry")

print("widget.frameGeometry().width()=%d" % widget.frameGeometry().width() )

print("widget.frameGeometry().height()=%d" % widget.frameGeometry().height() )

print("widget.pos().x()=%d" % widget.pos().x() )

print("widget.pos().y()=%d" % widget.pos().y() )

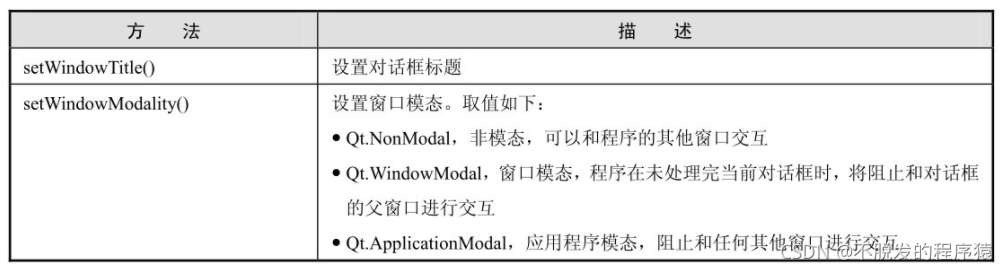
sys.exit(app.exec\_())

**3、QDialog窗口**

为了更好地实现人机交互，比如Windows及Linux等系统均会提供一系列的标准对话框来完成特定场景下的功能，如选择字号大小、字体颜色等。在PyQt 5中定义了一系列的标准对话框类，让使用者能够方便和快捷地通过各个类完成字号大小、字体颜色以及文件的选择等。

QDialog是对话框窗口的基类，对话框主要用来执行短期任务，或者与用户进行互动，它可以是模态的，也可以是非模态的。QDialog窗口没有菜单栏、工具栏、状态栏等。

QDialog类中的常用方法如下表所示：



来看一个QDialog窗口的例子，在这个例子中，Dialog窗口的WindowModality属性决定是否为模态或非模态。单击QWidget窗口中的PushButton按钮时，将生成一个对话框窗口。在对话框窗口的标题栏上没有最小化和最大化控件。代码中将给按钮的clicked信号添加槽函数showdialog()。当用户按下Esc键时，对话框窗口将会默认调用QDialog.reject()方法，然后关闭对话框窗口。效果如下所示：



实现代码如下所示：

**import** sys

**from** PyQt5.QtCore **import** \*

**from** PyQt5.QtGui **import** \*

**from** PyQt5.QtWidgets **import** \*

**class** **DialogDemo**( **QMainWindow** ):

**def** **\_\_init\_\_**(self, parent=None):

super(DialogDemo, self).\_\_init\_\_(parent)

self.setWindowTitle("Dialog 例子")

self.resize(350,300)

self.btn = QPushButton( self)

self.btn.setText("弹出对话框")

self.btn.move(50,50)

self.btn.clicked.connect(self.showdialog)

**def** **showdialog**(self ):

dialog = QDialog()

btn = QPushButton("ok", dialog )

btn.move(50,50)

dialog.setWindowTitle("Dialog")

dialog.setWindowModality(Qt.ApplicationModal)

dialog.exec\_()

**if** \_\_name\_\_ == '\_\_main\_\_':

app = QApplication(sys.argv)

demo = DialogDemo()

demo.show()

sys.exit(app.exec\_())

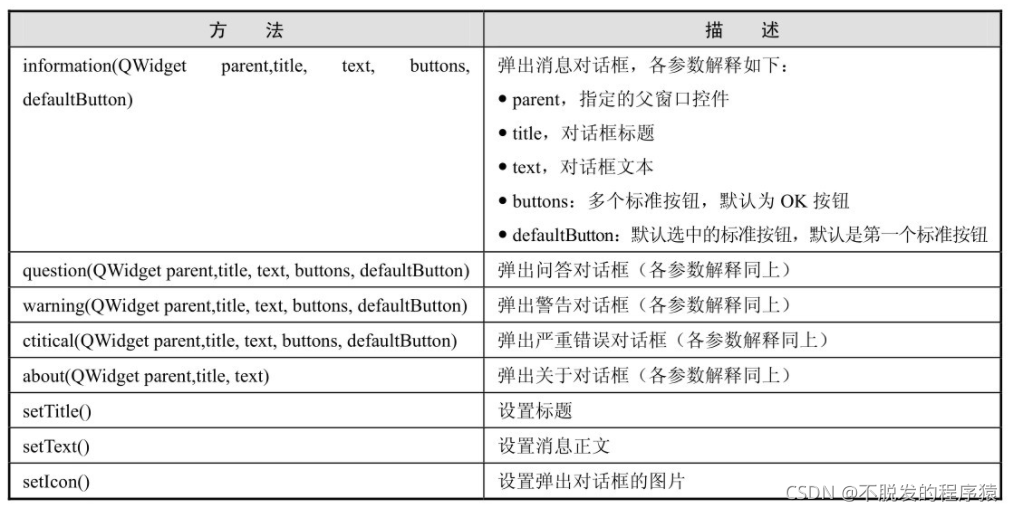
QDialog类的子类主要有QMessageBox、QlnputDialog、QFontDialog、QFileDialog等。来看看它们的特性吧~

**3.1、QMessageBox窗口**

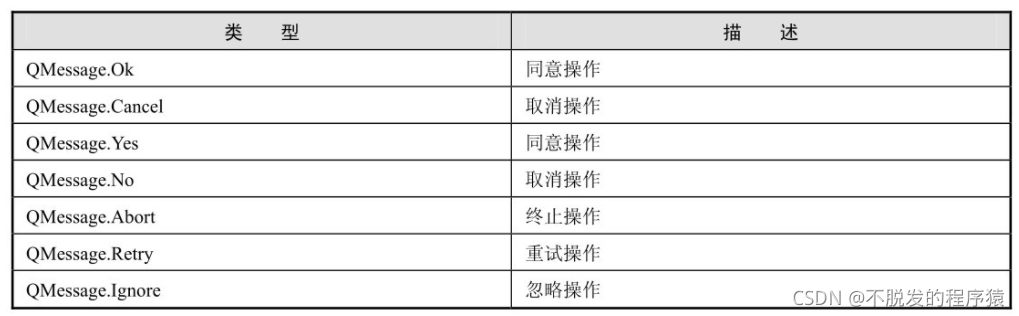
QMessageBox是一种通用的弹出式对话框，用于显示消息，允许用户通过单击不同的标准按钮对消息进行反馈。每个标准按钮都有一个预定义的文本、角色和十六进制数。

QMessageBox类提供了许多常用的弹出式对话框，如提示、警告、错误、询问、关于等对话框。这些不同类型的QMessageBox对话框只是显示时的图标不同，其他功能是—样的。

QMessageBox类中的常用方法如下表所示：



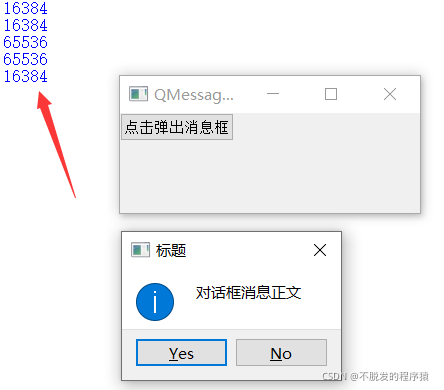
QMessageBox的标准按钮类型如下表所示：



5种常用的消息对话框及其显示效果如下表所示：



以下示例就是QMessageBox的典型应用，效果如下所示：



实现代码如下所示：

**import** sys

**from** PyQt5.QtCore **import** \*

**from** PyQt5.QtGui **import** \*

**from** PyQt5.QtWidgets **import** \*

**class** **WinForm**( **QWidget**):

**def** **\_\_init\_\_**(self):

super(WinForm,self).\_\_init\_\_()

self.setWindowTitle("QMessageBox 例子")

self.resize(300, 100)

self.myButton = QPushButton(self)

self.myButton.setText("点击弹出消息框")

self.myButton.clicked.connect(self.msg)

**def** **msg**(self):

*# 使用infomation信息框*

reply = QMessageBox.information(self, "标题", "对话框消息正文", QMessageBox.Yes | QMessageBox.No , QMessageBox.Yes )

print( reply )

**if** \_\_name\_\_ == '\_\_main\_\_':

app= QApplication(sys.argv)

demo = WinForm()

demo.show()

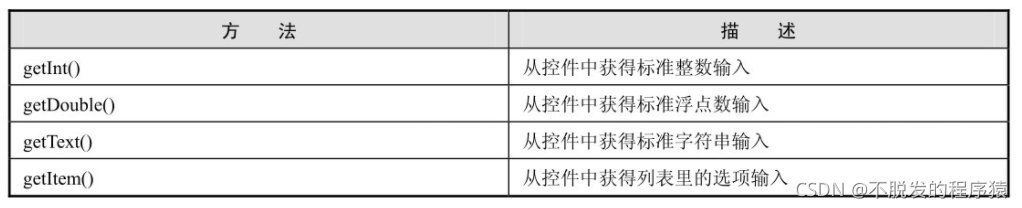
sys.exit(app.exec\_())

**3.2、QInputDialog窗口**

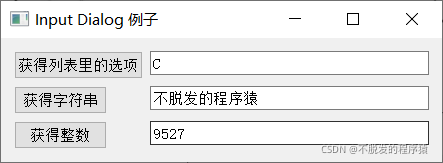
QInputDialog 控件是一个标准对话框，由一个文本框和两个按钮（OK按钮和Cancel按钮）组成。当用户单击OK按钮或按Enter 键后，在父窗口可以收集通过QInputDialog控件输入的信息。

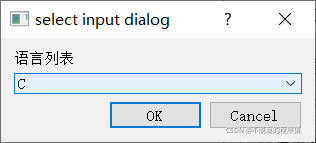
QInputDialog控件是QDialog标准对话框的一部分，在QlnputDialog控件中可以输入数字、字符串或列表中的选项，标签用于提示必要的信息。

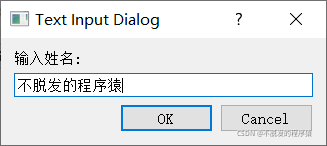
QlnputDialog类中的常用方法如下表所示：

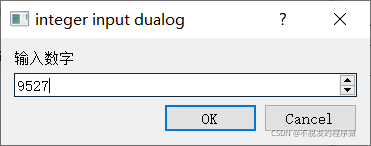


以下示例就是QFileDialog的典型应用，效果如下所示：









实现代码如下所示：

**import** sys

**from** PyQt5.QtCore **import** \*

**from** PyQt5.QtGui **import** \*

**from** PyQt5.QtWidgets **import** \*

**class** **InputdialogDemo**(**QWidget**):

**def** **\_\_init\_\_**(self, parent=None):

super(InputdialogDemo, self).\_\_init\_\_(parent)

layout = QFormLayout()

self.btn1 = QPushButton("获得列表里的选项")

self.btn1.clicked.connect(self.getItem)

self.le1 = QLineEdit()

layout.addRow(self.btn1,self.le1)

self.btn2 = QPushButton("获得字符串")

self.btn2.clicked.connect(self.getIext)

self.le2 = QLineEdit()

layout.addRow(self.btn2,self.le2)

self.btn3 = QPushButton("获得整数")

self.btn3.clicked.connect(self.getInt)

self.le3 = QLineEdit()

layout.addRow(self.btn3,self.le3)

self.setLayout(layout)

self.setWindowTitle("Input Dialog 例子")

**def** **getItem**(self):

items = ("C", "C++", "Java", "Python")

item, ok = QInputDialog.getItem(self, "select input dialog",

"语言列表", items, 0, False)

**if** ok **and** item:

self.le1.setText(item)

**def** **getIext**(self):

text, ok = QInputDialog.getText(self, 'Text Input Dialog', '输入姓名:')

**if** ok:

self.le2.setText(str(text))

**def** **getInt**(self):

num,ok = QInputDialog.getInt(self,"integer input dualog","输入数字")

**if** ok:

self.le3.setText(str(num))

**if** \_\_name\_\_ == '\_\_main\_\_':

app = QApplication(sys.argv)

demo = InputdialogDemo()

demo.show()

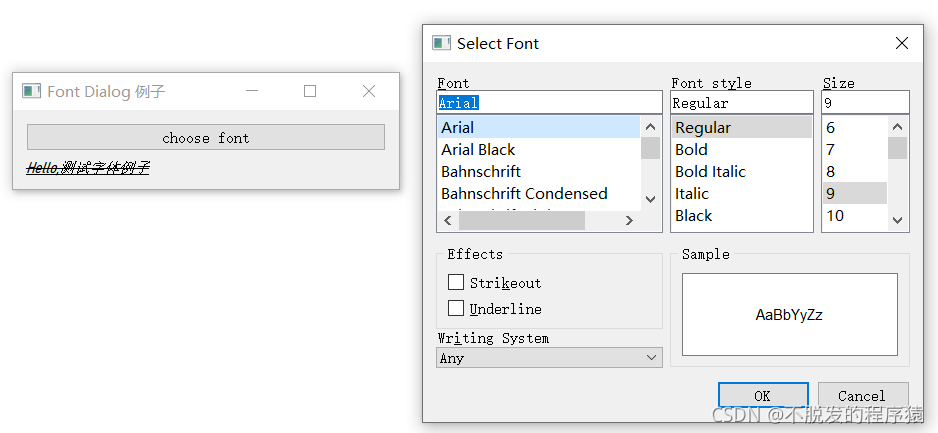
sys.exit(app.exec\_())

**3.3、QFontDialog窗口**

FontDialog控件是一个常用的字体选择对话框，可以让用户选择所显示文本的字号大小、样式和格式。

QFontDialog 是QDialog 标准对话框的一部分，使用QFontDialog类的静态方法getFont()，可以从字体选择对话框中选择文本的显示字号大小、样式和格式。

以下示例就是QFontDialog的典型应用，效果如下所示：



在这个例子中，通过字体选择对话框选择相应的字体，并且所选择字体的效果显示在QLineEdit的文本上。 实现代码如下所示：

**import** sys

**from** PyQt5.QtCore **import** \*

**from** PyQt5.QtGui **import** \*

**from** PyQt5.QtWidgets **import** \*

**class** **FontDialogDemo**(**QWidget**):

**def** **\_\_init\_\_**(self, parent=None):

super(FontDialogDemo, self).\_\_init\_\_(parent)

*#创建QVBoxLayout布局，在垂直布局管理器中添加fontButton 和fontLineEdit组件，并按照相应的位置添加到栅格布局中。*

layout = QVBoxLayout()

self.fontButton = QPushButton("choose font")

*#当单击fontButton按钮时，将clicked信号发送到槽函数getFont()中。*

self.fontButton .clicked.connect(self.getFont)

*#实例化fontButton和fontLineEdit对象，并将fontButton的clicked信号和槽函数getFont()绑定在一起。*

layout.addWidget(self.fontButton )

self.fontLineEdit = QLabel("Hello,测试字体例子")

layout.addWidget(self.fontLineEdit )

self.setLayout(layout)

self.setWindowTitle("Font Dialog 例子")

*#自定义槽函数，选择字体，并将字体效果设置到fontLineEdit中。getFont()方法返回的为元组类型，同时返回所选择的字体和函数执行的状态。*

**def** **getFont**(self):

font, ok = QFontDialog.getFont()

**if** ok:

self.fontLineEdit .setFont(font)

**if** \_\_name\_\_ == '\_\_main\_\_':

app = QApplication(sys.argv)

demo = FontDialogDemo()

demo.show()

sys.exit(app.exec\_())

**3.4、QFileDialog窗口**

QFileDialog是用于打开和保存文件的标准对话框，QFileDialog类继承自QDialog类。

QFileDialog在打开文件时使用了文件过滤器，用于显示指定扩展名的文件，也可以设置使用QFileDialog打开文件时的起始目录和指定扩展名的文件。

QFileDialog类中的常用方法如下表所示：



以下示例就是QFileDialog的典型应用，效果如下所示：



在这个例子中，通过文件对话框来选择文件，其中第一个文件对话框只允许打开图片文件，并把所加载的图片显示在标签（QLabel）中；第二个文件对话框只允许打开文本文件，并把文本内容显示在文本框（QTextEdit）内。 实现代码如下所示：

**import** sys

**from** PyQt5.QtCore **import** \*

**from** PyQt5.QtGui **import** \*

**from** PyQt5.QtWidgets **import** \*

**class** **filedialogdemo**(**QWidget**):

**def** **\_\_init\_\_**(self, parent=None):

super(filedialogdemo, self).\_\_init\_\_(parent)

layout = QVBoxLayout()

*#第一个按钮使用QFileDialog.getOpenFileNam()，调用文件对话框来显示图像，并显示在一个标签控件中。它负责打开C盘目录下的文件。*

self.btn = QPushButton("加载图片")

self.btn.clicked.connect(self.getfile)

layout.addWidget(self.btn)

self.le = QLabel("")

layout.addWidget(self.le)

*#第二个按钮使用文件对话框(QFileDialog）对象的exec\_()方法来选择文件，并把所选文件的内容显示在文本编辑控件中。*

self.btn1 = QPushButton("加载文本文件")

self.btn1.clicked.connect(self.getfiles)

layout.addWidget(self.btn1)

self.contents = QTextEdit()

layout.addWidget(self.contents)

self.setLayout(layout)

self.setWindowTitle("File Dialog 例子")

**def** **getfile**(self):

fname, \_ = QFileDialog.getOpenFileName(self, 'Open file', 'c:\\',"Image files (\*.jpg \*.gif)")

self.le.setPixmap(QPixmap(fname))

**def** **getfiles**(self):

dlg = QFileDialog()

dlg.setFileMode(QFileDialog.AnyFile)

dlg.setFilter( QDir.Files )

**if** dlg.exec\_():

filenames= dlg.selectedFiles()

f = open(filenames[0], 'r')

**with** f:

data = f.read()

self.contents.setText(data)

**if** \_\_name\_\_ == '\_\_main\_\_':

app = QApplication(sys.argv)

ex = filedialogdemo()

ex.show()

sys.exit(app.exec\_())