pyqt4 指南

万泽[①] | 德山书生[②]

版本: 0.1

---

① 作者:

② 编者：德山书生，湖南常德人氏。邮箱：a358003542@gmail.com。

前言

前置知识：python 语言基础知识。从变量各个操作对象到程序结构直到类。

本文主要参考资料：

1.pyqt4 教程，http://blog.cx125.com/books/PyQt4_Tutorial/

2.Prentice Hall，Rapid GUI Programming with Python and Qt，2007 年 10 月

3.http://pyqt.sourceforge.net/Docs/PyQt4/classes.html

4.http://straightedgelinux.com/blog/python/html/pyqtxt.html

5.

目 录

1 刚开始

1.1 安装 pyqt4

ubuntu 下安装 pyqt4 即安装 python3-pyqt4 即可:

```
sudo apt-get install python3-pyqt4
```

检查 pyqt4 安装情况执行以下脚本即可，显示的是当前安装的 pyqt4 的版本号:

```
from PyQt4.QtCore import QT_VERSION_STR

print(QT_VERSION_STR)
```

1.2 pyqt4 模块简介

QtCore 模块包括了核心的非 GUI 功能，该模块用来对时间、文件、目录、各种数据类型、流、网址、媒体类型、线程或进程进行处理。

QtGui 模块包括图形化窗口部件和及相关类。包括如按钮、窗体、状态栏、滑块、位图、颜色、字体等等。

QtHelp 模块包含了用于创建和查看可查找的文档的类。

QtNetwork 模块包括网络编程的类。这些类可以用来编写 TCP/IP 和 UDP 的客户端和服务器。它们使得网络编程更容易和便捷。

QtOpenGL 模块使用 OpenGL 库来渲染 3D 和 2D 图形。该模块使得 Qt GUI 库和 OpenGL 库无缝集成。

QtScript 模块包含了使 PyQt 应用程序使用 JavaScript 解释器编写脚本的类。

QtSql 模块提供操作数据库的类。

QtSvg 模块提供了显示 SVG 文件内容的类。可缩放矢量图形 (SVG) 是一种用 XML 描述二维图形和图形应用的语言。

QtTest 模块包含了对 PyQt 应用程序进行单元测试的功能。（PyQt 没有实现完全的 Qt 单元测试框架，相反，它假设使用标准的 Python 单元测试框架来实现模拟用户和 GUI 进行交互。）

QtWebKit 模块实现了基于开源浏览器引擎 WebKit 的浏览器引擎。

QtXml 包括处理 XML 文件的类，该模块提供了 SAX 和 DOM API 的接口。

QtXmlPatterns 模块包含的类实现了对 XML 和自定义数据模型的 XQuery 和 XPath 的支持。

phonon 模块包含的类实现了跨平台的多媒体框架，可以在 PyQt 应用程序中使用音频和视频内容。

QtMultimedia 模块提供了低级的多媒体功能，开发人员通常使用 **phonon** 模块。

QtAssistant 模块包含的类允许集成 **Qt Assistant** 到 PyQt 应用程序中，提供在线帮助。

QtDesigner 模块包含的类允许使用 PyQt 扩展 **Qt Designer**。

Qt 模块综合了上面描述的模块中的类到一个单一的模块中。这样做的好处是你不用担心哪个模块包含哪个特定的类，坏处是加载进了整个 Qt 框架，从而增加了应用程序的内存占用。

uic 模块包含的类用来处理.ui 文件，该文件由 Qt Designer 创建，用于描述整个或者部分用户界面。它包含的加载.ui 文件和直接渲染以及从.ui 文件生成 Python 代码为以后执行的类。

2　第一个例子

2.1　窗口

```
import sys
from PyQt4  import QtGui

class MyQWidget(QtGui.QWidget):
    def __init__(self,parent=None):
        QtGui.QWidget.__init__(self,parent)
        self.setGeometry(0, 0, 800, 600)
        # 坐标 0 0 大小 800 600
        self.setWindowTitle('myapp')

myapp = QtGui.QApplication(sys.argv)
mywidget = MyQWidget()
mywidget.show()
sys.exit(myapp.exec_())
```

首先导入 sys 宏包，是为了后面接受 sys.argv 参数[①]。从 PyQt4 模块导入

① 这里 sys.argv 就是这个 py 文件的文件名组成的列表：['窗口.py']

QtGui 宏包，是为了后面创建 QWidget 类的实例。

接下来我们定义了 MyQWidget 类，它继承自 QtGui 的 QWidget 类。然后重定义了构造函数，首先继承了 QtGui 的 QWidget 类的构造函数，这里将 parent 的默认参数传递进去了。

然后通过 QWidget 类定义好的 **setGeometry** 方法来调整窗口的左顶点的坐标位置和窗口的大小。

然后通过 **setWindowTitle** 方法来设置这个窗口程序的标题，这里就简单设置为 myapp 了。

任何窗口程序都需要创建一个 QApplication 类的实例，这里是 myapp。然后接下来创建 QWidget 类的实例 mywidget，然后通过调用 mywidget 的方法 **show** 来显示窗体。

最后我们看到系统要退出是调用的 myapp 实例的 **exec_** 方法。

2.2　加上图标

现在我们在前面第一个程序的基础上稍作修改，来给这个程序加上图标。程序代码如下:

```
import sys
from PyQt4  import QtGui


class MyQWidget(QtGui.QWidget):
    def __init__(self,parent=None):
        QtGui.QWidget.__init__(self,parent)
        self.resize(800,600)
        self.setWindowTitle('myapp')
        self.setWindowIcon(QtGui.QIcon\
```

```
        ('icons/myapp.ico'))



myapp = QtGui.QApplication(sys.argv)
mywidget = MyQWidget()
mywidget.show()
sys.exit(myapp.exec_())
```

这个程序相对上面的程序就增加了一个 **setWindowIcon** 方法，这个方法调用了 QtGui.QIcon 方法，然后后面跟的就是图标的存放路径，使用相对路径。在运行这个例子的时候，请随便弄个图标文件过来。

这个程序为了简单起见就使用了 QWidget 类的 **resize** 方法来设置窗体的大小。

2.3　弹出提示信息

```
import sys
from PyQt4  import QtGui


class MyQWidget(QtGui.QWidget):
    def __init__(self,parent=None):
        QtGui.QWidget.__init__(self,parent)
        self.resize(800,600)
        self.setWindowTitle('myapp')
        self.setWindowIcon(QtGui.QIcon\
        ('icons/myapp.ico'))
        self.setToolTip(' 看什么看 ^_^')
```

```
        QtGui.QToolTip.setFont(QtGui.QFont\
        (' 微软雅黑', 12))


myapp = QtGui.QApplication(sys.argv)
mywidget = MyQWidget()
mywidget.show()
sys.exit(myapp.exec_())
```

上面这段代码和前面的代码的不同就在于 MyQWidget 类的初始函数新加入了两条命令。其中 **setToolTip** 方法设置具体显示的文本内容，然后后面调用 QToolTip 类的 **setFont** 方法来设置字体和字号，我不太清楚这里随便设置系统的字体微软雅黑是不是有效。

这样你的鼠标停放在窗口上一会儿会弹出一小段提示文字。

2.4　关闭窗体时询问

目前程序点击那个叉叉图标关闭程序的时候将会直接退出，这里新加入一个询问机制。

```
import sys
from PyQt4  import QtGui


class MyQWidget(QtGui.QWidget):
    def __init__(self,parent=None):
        QtGui.QWidget.__init__(self,parent)
        self.resize(800,600)
        self.setWindowTitle('myapp')
        self.setWindowIcon(QtGui.QIcon\
```

```
        ('icons/myapp.ico'))
        self.setToolTip(' 看什么看 ^_^')
        QtGui.QToolTip.setFont(QtGui.QFont\
        (' 微软雅黑', 12))

    def closeEvent(self, event):
        reply = QtGui.QMessageBox.question\
        (self, ' 信息',
            " 你确定要退出吗? ",
             QtGui.QMessageBox.Yes,
             QtGui.QMessageBox.No)

        if reply == QtGui.QMessageBox.Yes:
            event.accept()
        else:
            event.ignore()

myapp = QtGui.QApplication(sys.argv)
mywidget = MyQWidget()
mywidget.show()
sys.exit(myapp.exec_())
```

这段代码和前面代码的不同就是重新定义了 **colseEvent** 事件。这段代码的核心就是 QtGui 类的 QMessageBox 类的 question 方法，这个方法将会弹出一个询问窗体。这个方法接受四个参数：第一个参数是这个窗体所属的母体，这里就是 self 也就是实例 mywidget；第二个参数是弹出窗体的标题；第三个参数是一个标准 button；第四个参数也是一个标准 button，是默认（也就是按 enter 直接选定的）的 button。然后这个方法返回的是那个被点击了的标准 button 的标识符，所以后面和标准 buttonYes 比较了，然后执行 event 的

accept 方法。

这样这个程序在关闭的时候会弹出一个对话框，询问你是否真的要关闭，具体请读者自己实验一下。

2.5　屏幕居中显示窗体

```
import sys
from PyQt4  import QtGui


class MyQWidget(QtGui.QWidget):
    def __init__(self,parent=None):
        QtGui.QWidget.__init__(self,parent)
        self.resize(800,600)
        self.center()
        self.setWindowTitle('myapp')
        self.setWindowIcon(QtGui.QIcon\
        ('icons/myapp.ico'))
        self.setToolTip(' 看什么看 ^_^')
        QtGui.QToolTip.setFont(QtGui.QFont\
        (' 微软雅黑', 12))


    def closeEvent(self, event):
        # 重新定义 colseEvent
        reply = QtGui.QMessageBox.question\
        (self, ' 信息',
            " 你确定要退出吗？",
             QtGui.QMessageBox.Yes,
```

```
                QtGui.QMessageBox.No)

        if reply == QtGui.QMessageBox.Yes:
            event.accept()
        else:
            event.ignore()

    def center(self):
        screen = QtGui.QDesktopWidget().screenGeometry()
        size =  self.geometry()
        self.move((screen.width()-size.width())/2,\
         (screen.height()-size.height())/2)

myapp = QtGui.QApplication(sys.argv)
mywidget = MyQWidget()
mywidget.show()
sys.exit(myapp.exec_())
```

这个例子和前面相比改动是新建了一个 center 方法，接受一个实例，这里是 mywidget。然后对这个实例也就是窗口的具体位置做一些调整。

QDesktopWidget 类的 **screenGeometry** 方法返回一个量，这个量的 width 属性就是屏幕的宽度（按照 pt 像素计，比如 1366×768，宽度就是 1366），这个量的 height 属性就是屏幕的高度。

然后 QWidget 类的 **geometry** 方法同样返回一个量，这个量的 width 是这个窗体的宽度，这个量的 height 属性是这个窗体的高度。

然后调用 QWidget 类的 move 方法，这里是对 mywidget 这个实例作用。我们可以看到 move 方法的 X，Y 是从屏幕的坐标原点 (0,0) 开始计算的。第一个参数 X 表示向右移动了多少宽度，Y 表示向下移动了多少高度。

整个函数的作用效果就是将这个窗体居中显示。

2.6 QMainWindow 类

QtGui.QMainWindow 类提供应用程序主窗口，可以创建一个经典的拥有状态栏、工具栏和菜单栏的应用程序骨架。（之前使用的是 QWidget 类，现在换成 QMainWindow 类。）

前面第一个例子都是用的 QtGui.QWidget 类创建的一个窗体。关于 QWidget 和 QMainWindow 这两个类的区别参考这个网站得出的结论是：QWdget 类在 Qt 中是所有可画类的基础（这里的意思可能是窗体的基础吧。）任何基于 QWidget 的类都可以作为独立窗体而显示出来而不需要母体（parent）。

QMainWindow 类是针对主窗体一般需求而设计的，它预定义了菜单栏状态栏和其他 widget（窗口小部件）。因为它继承自 QWidget，所以前面谈及的一些属性修改都适用于它。那么首先我们将之前的代码中的 QWidget 类换成 QMainWindow 类。

```
import sys
from PyQt4  import QtGui


class MainWindow(QtGui.QMainWindow):
    def __init__(self,parent=None):
        QtGui.QMainWindow.__init__(self,parent)
        self.resize(800,600)
        self.center()
        self.setWindowTitle('myapp')
        self.setWindowIcon(QtGui.QIcon\
        ('icons/myapp.ico'))
        self.setToolTip(' 看什么看 ^_^')
```

```
        QtGui.QToolTip.setFont(QtGui.QFont\
        (' 微软雅黑', 12))

    def closeEvent(self, event):
        reply = QtGui.QMessageBox.question\
        (self, ' 信息',
            " 你确定要退出吗? ",
             QtGui.QMessageBox.Yes,
             QtGui.QMessageBox.No)

        if reply == QtGui.QMessageBox.Yes:
            event.accept()
        else:
            event.ignore()

    def center(self):
        screen = QtGui.QDesktopWidget().screenGeometry()
        size =  self.geometry()
        self.move((screen.width()-size.width())/2,\
         (screen.height()-size.height())/2)

myapp = QtGui.QApplication(sys.argv)
mainwindow = MainWindow()
mainwindow.show()
sys.exit(myapp.exec_())
```

现在程序运行情况良好，我们继续加点东西进去。

2.7　加上状态栏

```
#!/usr/bin/env python3
#-*- coding: utf-8 -*-
import sys
from PyQt4  import QtGui

class MainWindow(QtGui.QMainWindow):
    def __init__(self,parent=None):
        QtGui.QMainWindow.__init__(self,parent)
        self.resize(800,600)
        self.center()
        self.setWindowTitle('myapp')
        self.setWindowIcon(QtGui.QIcon\
        ('icons/myapp.ico'))
        self.setToolTip(' 看什么看 ^_^')
        QtGui.QToolTip.setFont(QtGui.QFont\
        (' 微软雅黑', 12))



    def closeEvent(self, event):
        reply = QtGui.QMessageBox.question\
        (self, ' 信息',
            " 你确定要退出吗? ",
             QtGui.QMessageBox.Yes,
             QtGui.QMessageBox.No)
```

```
        if reply == QtGui.QMessageBox.Yes:
            event.accept()
        else:
            event.ignore()

    def center(self):
        screen = QtGui.QDesktopWidget().screenGeometry()
        size =  self.geometry()
        self.move((screen.width()-size.width())/2,\
         (screen.height()-size.height())/2)

myapp = QtGui.QApplication(sys.argv)
mainwindow = MainWindow()
mainwindow.show()
mainwindow.statusBar().showMessage('程序已就绪...')
sys.exit(myapp.exec_())
```

这个程序和前面的区别在于最后倒数第二行，调用 mainwindow 这个 QMainWindow 类生成的实例的 **statusBar** 方法生成一个 QStatusBar 对象，然后调用 QStatusBar 类的 **showMessage** 方法来显示一段文字。

如果你希望这段代码在`__init__`方法里面，那么具体实现过程也与上面描述的类似。考虑到状态栏有些程序不一定需要，这里暂时不加到构造函数里面了。

2.8　加上菜单栏

```
#!/usr/bin/env python3
#-*- coding: utf-8 -*-
import sys
from PyQt4  import QtGui

class MainWindow(QtGui.QMainWindow):
    def __init__(self,parent=None):
        QtGui.QMainWindow.__init__(self,parent)
        self.resize(800,600)
        self.center()
        self.setWindowTitle('myapp')
        self.setWindowIcon(QtGui.QIcon\
        ('icons/myapp.ico'))
        self.setToolTip(' 看什么看 ^_^')
        QtGui.QToolTip.setFont(QtGui.QFont\
        (' 微软雅黑', 12))

    # 菜单栏
        self.menubar = self.menuBar()
        menu_file=self.menubar.addMenu(' 文件')
        menu_edit=self.menubar.addMenu(' 编辑')
        menu_help=self.menubar.addMenu(' 帮助')

    def closeEvent(self, event):
        reply = QtGui.QMessageBox.question\
        (self, ' 信息',
            " 你确定要退出吗？",
             QtGui.QMessageBox.Yes,
```

```
            QtGui.QMessageBox.No)

        if reply == QtGui.QMessageBox.Yes:
            event.accept()
        else:
            event.ignore()

    def center(self):
        screen = QtGui.QDesktopWidget().screenGeometry()
        size =  self.geometry()
        self.move((screen.width()-size.width())/2,\
         (screen.height()-size.height())/2)

myapp = QtGui.QApplication(sys.argv)
mainwindow = MainWindow()
mainwindow.show()
mainwindow.statusBar().showMessage('程序已就绪...')
sys.exit(myapp.exec_())
```

和上面讨论加上状态栏类似，这里用 QMainWindow 类的 menuBar 方法来获得一个菜单栏对象。然后用这个菜单栏对象的 **addMenu** 方法来创建一个新的菜单对象（QMenu 类）——方法里面的内容是新建菜单显示的名字。

建议给菜单对象取个名字，后面方便引用。

2.9　加上动作

```
#!/usr/bin/env python3
#-*- coding: utf-8 -*-
import sys
from PyQt4  import QtGui

class MainWindow(QtGui.QMainWindow):
    def __init__(self,parent=None):
        QtGui.QMainWindow.__init__(self,parent)
        self.resize(800,600)
        self.center()
        self.setWindowTitle('myapp')
        self.setWindowIcon(QtGui.QIcon\
        ('icons/myapp.ico'))
        self.setToolTip(' 看什么看 ^_^')
        QtGui.QToolTip.setFont(QtGui.QFont\
        (' 微软雅黑', 12))

    # 动作和连接
        act_exit = QtGui.QAction(' 退出', self)
        act_exit.setStatusTip(' 退出程序')
        act_exit.triggered.connect(self.close)

        act_about = QtGui.QAction(' 关于本程序', self)
        act_about.triggered.connect(self.about)

        act_aboutqt = QtGui.QAction(' 关于 Qt', self)
        act_aboutqt.triggered.connect(self.aboutqt)
```

```
    # 菜单栏
        self.menubar = self.menuBar()
        menu_file=self.menubar.addMenu(’ 文件’)
        menu_file.addAction(act_exit)
        menu_edit=self.menubar.addMenu(’ 编辑’)
        menu_help=self.menubar.addMenu(’ 帮助’)
        menu_help.addAction(act_about)
        menu_help.addAction(act_aboutqt)

# 函数
    def about(self):
        QtGui.QMessageBox.about(self,” 关于本程序”,” 本程序是一个用于教学的程序。\n\nFell f
    def aboutqt(self):
        QtGui.QMessageBox.aboutQt(self)

    def closeEvent(self, event):
        reply = QtGui.QMessageBox.question\
        (self, ’ 信息’,
            ” 你确定要退出吗？”,
             QtGui.QMessageBox.Yes,
             QtGui.QMessageBox.No)

        if reply == QtGui.QMessageBox.Yes:
            event.accept()
        else:
            event.ignore()

    def center(self):
```

```
        screen = QtGui.QDesktopWidget().screenGeometry()
        size =  self.geometry()
        self.move((screen.width()-size.width())/2,\
         (screen.height()-size.height())/2)

myapp = QtGui.QApplication(sys.argv)
mainwindow = MainWindow()
mainwindow.show()
mainwindow.statusBar().showMessage(' 程序已就绪...')
sys.exit(myapp.exec_())
```

现在在前面例子的基础上给之前的菜单对象加上动作。比如 menu_file 就是之前创建的一个菜单对象，现在调用这个对象的 **addAction** 方法，将 act_exit 动作对象加进去。

act_exit 动作对象需要在前面定义，通过 QtGui 类的 QAction 类的构造函数来创建一个动作对象。构造函数最少需要两个参量：第一个是显示的文本，第二个是这个动作依附的母体（也就是常见的 parent 变量），通常这里填上 self 即可。

在这里这个动作对象，就是菜单的下拉选项，如果我们用鼠标点击一下的话，将会触发 **triggered** 信号，如果我们 connect 方法连接到某个槽上（或者某个你定义的函数），那么将会触发这个函数的执行。下面就信号－槽机制详细说明之。

2.9.1　信号－槽机制

GUI 程序一般都引入一种事件和信号机制，well，简单来说就是一个循环程序，这个循环程序等到某个时刻程序会自动做某些事情比如刷新程序界面啊，或者扫描键盘鼠标之类的，等用户点击鼠标或者按了键盘之后，它会接受这个信号然后做出相应的反应。

所以你一定猜到了，close 函数可能就是要退出这个循环程序。我们调用主程序的exec_ 方法，就是开启这个循环程序。

```
myapp.exec_()
```

pyqt4 的旧信号－槽连接语句我在这里忽略了，网上到处都是。下面就新的语句说明之。

```
act_exit.triggered.connect(self.close)
```

我们看到新的信号－槽机制语句变得更精简更易懂了。整个过程就是如我前面所述，某个对象发出了某个信号，然后用 connect 将这个信号和某个槽（或者你定义的某个函数）连接起来即形成了一个反射弧了。

这里的槽就是 self 主窗口实例的 close 方法，这个是主窗口自带的函数。

下面我们看到 aboutqt 和 about 函数，是重新定义的。具体读者如果不懂请翻阅 QMessageBox 类的静态方法 about 和 aboutqt。

3　简单的编辑器

4 布局设计

先就常用布局类进行介绍，然后引入用 qt 设计器来手工绘制 UI 的过程。

4.1 QHBoxLayout 类

在 QtGui 模块中，继承自 QBoxLayout，QBoxLayout 继承自 QLayout，QLayout 继承自 QObject 和 QLayoutItem。

简单布局，让元素横向排列。

使用方法如下:

```
mainlayout=QtGui.QHBoxLayout()
mainlayout.addWidget(button1)
mainlayout.addWidget(button2)
self.setLayout(mainlayout)
```

首先建立一个布局对象，有 QHBoxLayout 类生成，然后在这个布局实例里面用 **addWidget** 方法来加入元素，然后将这个布局用主窗口的 **setLayout** 方法设置进去。

4.2　QVBoxLayout 类

在 QtGui 模块中，继承自 QBoxLayout，QBoxLayout 继承自 QLayout，QLayout 继承自 QObject 和 QLayoutItem。

和 QHBoxLayout 类似，简单布局，竖向排列。

5 pyqt4 类详解

5.1 QWidget 类

在 QtGui 模块中，继承自 QObject 和 QPaintDevice。

5.1.1 构造函数

```
QWidget.__init__(self, QWidget parent = None, Qt.WindowFlags flags = 0)
```

默认 parent 是 None，如果是其他 widget，那么这个 widget 就变成那个（参量）widget 的子窗口了，如果那个（参量）widget 删除了，这个 widget 也就是删除了。

5.1.2 激活窗口

```
QWidget.activateWindow(self)
```

activateWindow 方法用于激活这个窗口，这个窗口是可见的并且可以键盘输入。点击窗口的标题栏一个窗口就上前显现了用的就是这个函数。

5.1.3 调整窗口大小

```
QWidget.setGeometry(self, QRect)
QWidget.setGeometry(self, int ax, int ay, int aw, int ah)
QWidget.resize(self, QSize)
QWidget.resize(self, int w, int h)
```

5.1.4 设置图标

```
QWidget.setWindowIcon(self, QIcon icon)
QWidget.setWindowIconText(self, QString)
```

就是设置程序下面显示的图标和图标文字。（QString 对象我测试了直接用 python3 的字符串对象也是可以的。）

5.1.5 设置窗口标题

```
QWidget.setWindowTitle(self, QString)
```

5.1.6 show 方法

```
QWidget.show(self)
```

显示某个窗口和它的子窗口，相当于让这个窗口可见，和 **setVisible(True)** 等价。

5.1.7 设置提示词

```
QWidget.setToolTip(self, QString)
```

5.1.8 closeEvent 方法

```
QWidget.closeEvent (self, QCloseEvent)
```

一般在窗口关闭时调用，你可以重定义这个方法来改变窗口在面对关闭事件时的反应。

5.2 QIcon 类

在 QtGui 模块中。

这个类提供了可缩放图标的解决方案。

5.2.1 构造函数

```
__init__(self)
__init__(self, QPixmap pixmap)
__init__(self, QIcon other)
__init__(self, QString fileName)
__init__(self, QIconEngine engine)
__init__(self, QIconEngineV2 engine)
__init__(self, QVariant variant)
```

最常用的是第四种形式，也就是用相对路径（要注意操作系统的不同可能带来的问题）引用某个图标文件。

5.3 QPushButton 类

来自 QtGui 模块，继承自 QAbstractButton 类，QAbstractButton 类继承自 QWidget 类。

QPushButton 也就是 GUI 设计中最常见的按钮。

5.3.1 构造函数

```
__init__(self, QWidget parent = None)
__init__(self, QString text, QWidget parent = None)
__init__(self, QIcon icon, QString text, QWidget parent = None)
```

我们看到按钮显示的文字和图标在这里是可以设置的。

5.4 QFileDialog

来自 QtGui 模块，继承自 QDialog，QDialog 继承自 QWidget。

弹出一个文件或目录选择对话框。

简单的使用常通过静态方法来实现：

5.4.1 静态方法

更多信息请参见官方类文档。

getExistingDirectory 方法

返回用户选定的已存在的目录。这里 parent 是对话框依附的父窗口，caption 是弹出窗口的标题，directory 是窗口弹出时默认打开的位置，options 是?

getOpenFileName 方法

返回用户选定的已存在文件。比如:

```
filename=QtGui.QFileDialog.getOpenFileName(self,” 打开文件...”,”.”,””)
```

这里 filename 是字符串类型，就是你选择的文件的名字（包含绝对引用地址），后面”.” 是当前目录的意思，在后面过滤器空字符串表示所有的文件都显示。

5.5 QColorDialog 类

来自 QtGui 模块，继承自 QDialog，QDialog 继承自 QWidget。

弹出一个选择颜色的窗口。

5.5.1 静态方法

getColor 方法

简单的设置不给参数也是可以的。

5.6 QFontDialog 类

来自 QtGui 模块，继承自 QDialog，QDialog 继承自 QWidget。

弹出一个选择字体的窗口。

5.6.1　静态方法

getFont 方法

简单的设置不给参数也是可以的。

5.7　QMessageBox 类

来自 QtGui 模块，继承自 QDialog，QDialog 继承自 QWidget。

弹出一个对话框用于用户做出一些选择或者给出一些提示信息。

5.7.1　静态方法

about 方法

需要三个参数，第一个参数是窗口的依附父窗口，第二个参数是窗口的标题，第三个参数是窗口的文字信息。

aboutQt 方法

最少只需要一个参数，即弹出窗口的依附父窗口。

5.8　QMainWindow 类

5.9　QDialog 类

6 pyqt4 实例

6.1 按钮打印你好

```
import sys
from PyQt4  import QtGui

class Mybutton(QtGui.QPushButton):
    def __init__(self,parent=None):
        QtGui.QPushButton.__init__(self,parent)
        self.resize(150, 90)
        self.center()
        self.setWindowTitle(’ 你好’)
        self.clicked.connect(self.hello)

    def center(self):
        screen = QtGui.QDesktopWidget().screenGeometry()
        size =  self.geometry()
        self.move((screen.width()-size.width())/2,\
         (screen.height()-size.height())/2)

    def hello(self):
```

```
        print(' 你好')


myapp = QtGui.QApplication(sys.argv)
mywidget = Mybutton()
mywidget.show()


myapp.exec_()
sys.exit()
```

这个代码最值得一提的就是 pyqt4 最新的信号－槽机制。就是某个对象的某个信号调用 connect 方法连接到某个槽上，所谓槽实际上就是定义的函数，只是这个函数没加上 () 圆括号，只是一个函数符号，所以称之为槽。

6.2　文件对话框

```
import sys
from PyQt4  import QtGui


class Mybutton(QtGui.QPushButton):
    def __init__(self,parent=None):
        QtGui.QPushButton.__init__(self," 打开文件",parent)
        self.resize(150, 90)
        self.center()
        self.setWindowTitle(' 打开文件')
        self.clicked.connect(self.openfile)


    def center(self):
```

```
        screen = QtGui.QDesktopWidget().screenGeometry()
        size =  self.geometry()
        self.move((screen.width()-size.width())/2,\
         (screen.height()-size.height())/2)


    def openfile(self):
        filename=QtGui.QFileDialog.getOpenFileName(self," 打开文件...","."","")
        print(' 文件'+str(filename)+' 已选择')

myapp = QtGui.QApplication(sys.argv)
mywidget = Mybutton()
mywidget.show()

myapp.exec_()
sys.exit()
```

这个例子和上面例子的区别就在于将按钮的动作改成了 openfile，然后用 QFileDialog 的静态方法 getOpenFileName 来获取打开文件的对话框。

6.3　更多的按钮更多的对话框

这个例子我们在前面两个例子的基础上，加上更多的按钮并对应更多的对话框（文件选择对话框，颜色选择对话框，字体选择对话框，一般信息对话框，一般选择对话框（关闭机制））。同时引入一种简单的布局。

```
import sys
from PyQt4  import QtGui
```

```
class Mywidget(QtGui.QWidget):
    def __init__(self,parent=None):
        QtGui.QWidget.__init__(self,parent)
        self.center()
        self.setWindowTitle('myapp')

        button1=QtGui.QPushButton(" 文件")
        button2=QtGui.QPushButton(" 颜色")
        button3=QtGui.QPushButton(" 字体")
        button4=QtGui.QPushButton(" 关于")
        button5=QtGui.QPushButton(" 关于 Qt")
        mainlayout=QtGui.QHBoxLayout()
        mainlayout.addWidget(button1)
        mainlayout.addWidget(button2)
        mainlayout.addWidget(button3)
        mainlayout.addWidget(button4)
        mainlayout.addWidget(button5)
        self.setLayout(mainlayout)
        button1.clicked.connect(self.openfile)
        button2.clicked.connect(self.choosecolor)
        button3.clicked.connect(self.choosefont)
        button4.clicked.connect(self.about)
        button5.clicked.connect(self.aboutqt)

    def center(self):
        screen = QtGui.QDesktopWidget().screenGeometry()
        size =  self.geometry()
```

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

    def openfile(self):
        filename=QtGui.QFileDialog.getOpenFileName(self,” 打开文件...”,”.”,””)
        print(’ 文件’+str(filename)+’ 已选择’)

    def choosecolor(self):
        colorname=QtGui.QColorDialog.getColor()
        print(’ 颜色’+str(colorname)+’ 已选择’)

    def choosefont(self):
        fontname=QtGui.QFontDialog.getFont()
        print(’ 字体’+str(fontname)+’ 已选择’)

    def about(self):
        QtGui.QMessageBox.about(self,” 关于本程序”,” 本程序用于测试按钮和对话框。”)
    def aboutqt(self):
        QtGui.QMessageBox.aboutQt(self)

    def closeEvent(self, event):
        # 重新定义 colseEvent
        reply = QtGui.QMessageBox.question\
        (self, ’ 信息’,
            ” 你确定要退出吗？”,
             QtGui.QMessageBox.Yes,
             QtGui.QMessageBox.No)
```

```
        if reply == QtGui.QMessageBox.Yes:
            event.accept()
        else:
            event.ignore()

myapp = QtGui.QApplication(sys.argv)
mywidget = Mywidget()
mywidget.show()
myapp.exec_()
sys.exit()
```