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.

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:

```
1 from PyQt4.QtCore import QT_VERSION_STR
2 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

```
1 import sys
2 from PyQt4  import QtGui
3
4 class MyQWidget(QtGui.QWidget):
5     def __init__(self,parent=None):
6         QtGui.QWidget.__init__(self,parent)
7         self.setGeometry(0, 0, 800, 600)
8         # 坐标 0 0 大小 800 600
9         self.setWindowTitle('myapp')
10
11 myapp = QtGui.QApplication(sys.argv)
12 mywidget = MyQWidget()
13 mywidget.show()
14 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

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

模块导入 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：

```
1 import sys
2 from PyQt4  import QtGui
3
4 class MyQWidget(QtGui.QWidget):
5     def __init__(self,parent=None):
6         QtGui.QWidget.__init__(self,parent)
7         self.resize(800,600)
8         self.setWindowTitle('myapp')
9         self.setWindowIcon(QtGui.QIcon\
```

```
10         ('icons/myapp.ico'))
11
12
13 myapp = QtGui.QApplication(sys.argv)
14 mywidget = MyQWidget()
15 mywidget.show()
16 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

```
1 import sys
2 from PyQt4  import QtGui
3
4 class MyQWidget(QtGui.QWidget):
5     def __init__(self,parent=None):
6         QtGui.QWidget.__init__(self,parent)
7         self.resize(800,600)
8         self.setWindowTitle('myapp')
9         self.setWindowIcon(QtGui.QIcon\
10         ('icons/myapp.ico'))
11         self.setToolTip(' 看什么看 ^_^')
```

```
12         QtGui.QToolTip.setFont(QtGui.QFont\
13         (' 微软雅黑', 12))
14
15 myapp = QtGui.QApplication(sys.argv)
16 mywidget = MyQWidget()
17 mywidget.show()
18 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。

```
1 import sys
2 from PyQt4  import QtGui
3
4 class MyQWidget(QtGui.QWidget):
5     def __init__(self,parent=None):
6         QtGui.QWidget.__init__(self,parent)
7         self.resize(800,600)
8         self.setWindowTitle('myapp')
9         self.setWindowIcon(QtGui.QIcon\
```

```
10          ('icons/myapp.ico'))
11          self.setToolTip(' 看什么看 ^_^')
12          QtGui.QToolTip.setFont(QtGui.QFont\
13          (' 微软雅黑', 12))
14
15      def closeEvent(self, event):
16          reply = QtGui.QMessageBox.question\
17          (self, ' 信息',
18              " 你确定要退出吗? ",
19               QtGui.QMessageBox.Yes,
20               QtGui.QMessageBox.No)
21
22          if reply == QtGui.QMessageBox.Yes:
23              event.accept()
24          else:
25              event.ignore()
26
27 myapp = QtGui.QApplication(sys.argv)
28 mywidget = MyQWidget()
29 mywidget.show()
30 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

```
1 import sys
2 from PyQt4  import QtGui
3
4 class MyQWidget(QtGui.QWidget):
5     def __init__(self,parent=None):
6         QtGui.QWidget.__init__(self,parent)
7         self.resize(800,600)
8         self.center()
9         self.setWindowTitle('myapp')
10        self.setWindowIcon(QtGui.QIcon\
11        ('icons/myapp.ico'))
12        self.setToolTip(' 看什么看 ^_^')
13        QtGui.QToolTip.setFont(QtGui.QFont\
14        (' 微软雅黑', 12))
15
16    def closeEvent(self, event):
17        # 重新定义 colseEvent
18        reply = QtGui.QMessageBox.question\
19        (self, ' 信息',
20            " 你确定要退出吗? ",
21             QtGui.QMessageBox.Yes,
```

```
22              QtGui.QMessageBox.No)
23
24         if reply == QtGui.QMessageBox.Yes:
25             event.accept()
26         else:
27             event.ignore()
28
29     def center(self):
30         screen = QtGui.QDesktopWidget().screenGeometry()
31         size =  self.geometry()
32         self.move((screen.width()-size.width())/2,\
33          (screen.height()-size.height())/2)
34
35 myapp = QtGui.QApplication(sys.argv)
36 mywidget = MyQWidget()
37 mywidget.show()
38 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。

```
1 import sys
2 from PyQt4  import QtGui
3
4 class MainWindow(QtGui.QMainWindow):
5     def __init__(self,parent=None):
6         QtGui.QMainWindow.__init__(self,parent)
7         self.resize(800,600)
8         self.center()
9         self.setWindowTitle('myapp')
10        self.setWindowIcon(QtGui.QIcon\
```

```
11          ('icons/myapp.ico'))
12          self.setToolTip(' 看什么看 ^_^')
13          QtGui.QToolTip.setFont(QtGui.QFont\
14          (' 微软雅黑', 12))
15
16      def closeEvent(self, event):
17          reply = QtGui.QMessageBox.question\
18          (self, ' 信息',
19              " 你确定要退出吗? ",
20               QtGui.QMessageBox.Yes,
21               QtGui.QMessageBox.No)
22
23          if reply == QtGui.QMessageBox.Yes:
24              event.accept()
25          else:
26              event.ignore()
27
28      def center(self):
29          screen = QtGui.QDesktopWidget().screenGeometry()
30          size =  self.geometry()
31          self.move((screen.width()-size.width())/2,\
32           (screen.height()-size.height())/2)
33
34  myapp = QtGui.QApplication(sys.argv)
35  mainwindow = MainWindow()
36  mainwindow.show()
37  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

在__init__ 方法下加入语句：

```
self.statusBar().showMessage('这是状态栏')
```

这样就显示了状态栏信息。

这里用 QMainWindow 类的 statusBar 方法获得状态栏，然后用状态栏的 showMessage 方法插入状态栏信息。

2.8　加上菜单栏

用 QMainWindow 类的 menuBar 方法来获得一个菜单栏。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——方法里面的内容是新建菜单显示的名字。

2.8.1　加上动作

某个菜单对象使用 addAction 方法来加上某个动作。

```
file.addAction(exit)
```

也就是所谓的 Action 对象，通过 QtGui 类的 QAction 子类创建动作对象。创建过程的三个参数是图标，文本和母体。

```
exit.setStatusTip('退出程序')
```

setStatusTip 方法设置状态栏提示信息。

2.8.2　事件和信号

下面就 pyqt4 中的事件和信号机制详细说明之。

将事件和信号联系起来用 connect 方法，该方法接受三个参数：第一个是对于谁，第二个是做了什么，第三个下面做什么。

```
1 self.connect(exit, QtCore.SIGNAL('triggered()'),
2  QtCore.SLOT('close()'))
```

这里的意思是对于 self 上面的 exit 对象，接受了信号 triggered，然后执行操作 close()。

2.9　加上工具栏