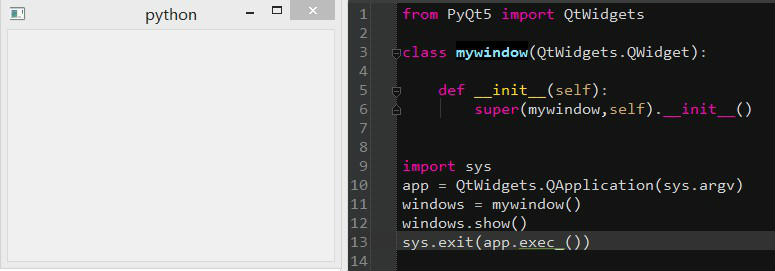
## pyqt5&python Gui入门教程（1） 第一个窗口（1）



第一个窗口和代码详细注释：

**from PyQt5 import QtWidgets**

#从PyQt库导入QtWidget通用窗口类

**class mywindow(QtWidgets.QWidget):**

#自己建一个mywindows类，以class开头，mywindows是自己的类名，

#（QtWidgets.QWidget）是继承QtWidgets.QWidget类方法，

# 定义类或函数不要忘记':'符号，判断语句也必须以':'结尾！

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

#def是定义函数（类方法）了，同样第二个\_\_init\_\_是函数名

# (self)是pyqt类方法必须要有的，代表自己，相当于java，c++中的this

#其实\_\_init\_\_是析构函数，也就是类被创建后就会预先加载的项目

**super(mywindow,self).\_\_init\_\_()**

#这里我们要重载一下mywindows同时也包含了QtWidgets.QWidget的预加载项

**import sys**

**app = QtWidgets.QApplication(sys.argv)**

#pyqt窗口必须在QApplication方法中使用，

#要不然会报错 QWidget: Must construct a QApplication before a QWidget

**windows = mywindow()**

# 生成过一个实例（对象）, windows是实例（对象）的名字，可以随便起！

# mywindows（）是我们上面自定义的类

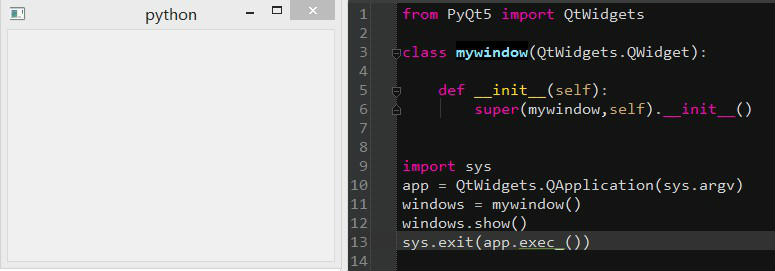
**windows.show()**

#有了实例，就得让他显示这里的show()是QWidget的方法，用来显示窗口的！

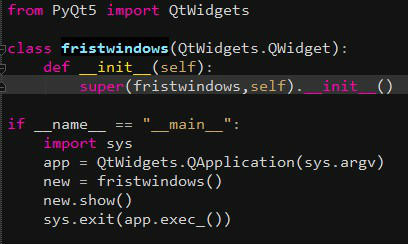
**sys.exit(app.exec\_())**

#启动事件循环

**pyqt5&python Gui入门教程（2） 第一个窗口（2）**



上图是第一篇教程，下面的显示效果都一样，我们来看看有什么不同



1、类的名字、实例的名字都换了，

2、多了一个if \_\_name\_\_ == "\_\_main\_\_": 以及下面的代码缩进了，层次改变了



1、我们把结尾的5句代码，单独建立了一个函数

2、然后直接调用函数

3、注意两个def的缩进，第一个def缩进了代表是在class里面，第二个和class平齐，则是在外面。

可以看到显示效果是一样，我们却有很多办法去实现。

知识点：

    1、if \_\_name\_\_ == "\_\_main\_\_": 是代表如果这个文件是主程序这运行下面的代码，如果是被别的程序文件调用的话，则运行下面的代码。

    2、\_\_init\_\_ 方法在类的一个对象被建立时，马上运行。这个方法可以用来对你的对象做一些你希望的 初始化 。注意，这个名称的开始和结尾都是双下划线。

    3、生成实例（对象）必须以类名()，别忘记了（）

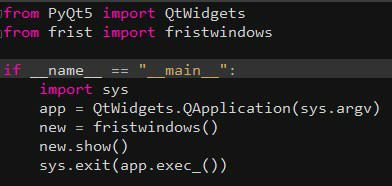
    4、类中的函数（方法）必须有self，是代表属于这个实例（对象）本身持有的，而外部定义的函数则不需要。

    整个流程；

    先导入PyQt5中的QtWidgets通用窗口库，通过继承QtWidgets.QWidget来定义自己的窗口。然后生成一个对象（实例化），再调用QWidget的show（）方法来显示这个窗口。

## pyqt5&python Gui入门教程（3） 第一个窗口（3）

frist.py 文件



second.py文件

这是两个文件，第一个文件和之前一样我们只录入的上半截，第二个文件我们保留了下半截，

也就是将一个文件分成两个文件，从second.py文件调用frist.py来显示窗口的目的。

注意second.py 文件加入了一句from frist import fristwindows 即导入first.py的fristwindows类

是不是很熟悉，将自己的文件作为库导入进来，然后调用即可！

虽然都是显示窗口，我们这里已经学习了很多办法来显示它。代码区别也是很大，关键是灵活运用。

知识点：

1、库，这个例子是将我们自己的文件作为库导入，也可以称为自定义库。

   注意不用.py结尾，pyqt会自动识别。

2、frist.py的firstwindows的代码就是显示一个窗口用的，但是没有将它实例化，并在QApplication中运行。

    所以运行frist.py是不会显示任何窗口的。

    我们将显示firstwindows的方法写在了第二个文件中，达到逻辑与界面分离的效果。

3、QApplication相当于main函数，也就是整个程序（有很多文件）的主入口函数。

    对于一个Gui程序必须至少有一个这样的一个实例来让程序运行。

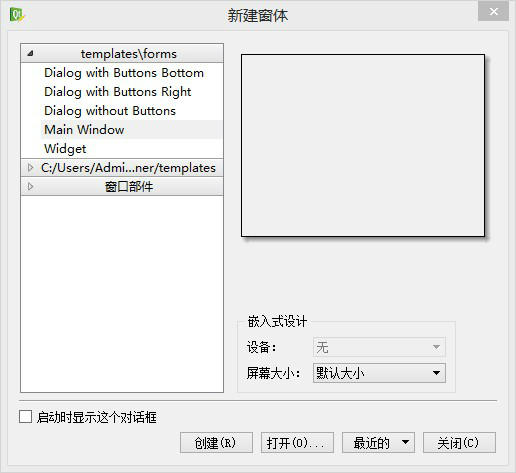
4、最后一句是调用sys库的exit退出方法，退出条件（参数）是app.exec\_()也就是整个窗口关闭。

    其实入门没什么难度，只是对一些基本概念的掌握和了解，本人也是菜鸟，写的不是很好，只是希望能和更多有兴趣爱好的新手一起交流学习。

## PyQt5&python Gui入门教程（4） 初探Qt Designer 设计师

网上很多教程都是代码，对于新手看起来就很头疼，pyqt同样为我们提供了Qt designer来设计窗口界面，

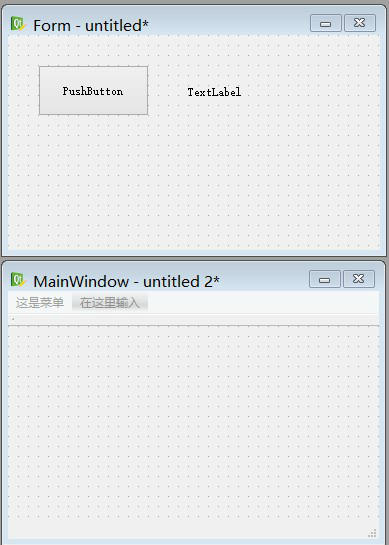
用起来也非常方便，对于新手我们应该善用它，虽然做出来的界面不那么华丽，但至少可以做出个像样的窗口来。



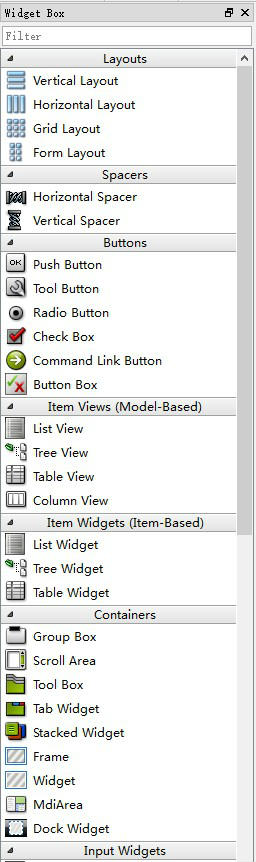
打开PyQt5的Qt Designer，会自动弹出新建窗体对话框，

对于我们最常用的就是Widget通用窗口类，还有个MainWindows顾名思义主窗口。

PyQt5的Widget被分离出来，似乎用来替代Dialog，并将Widget放入了QtWidget模块（库）中，PyQt4是QtGUI。



这是一个Widget和MainWindows，从界面上看起来没有什么，只是MainWindows默认添加了菜单栏、工具栏和状态栏等。



默认左边是控件栏，提供了很多空间类，我们可以直接拖放到widget中看到效果，点窗体--预览（Ctrl+R）

每个空间都有自己的名称，提供不同的功能，比如常用的按钮、输入框、单选、文本框等等，



右边是对窗口及控件的各种调整、设置、添加资源（列如:图片）、动作。

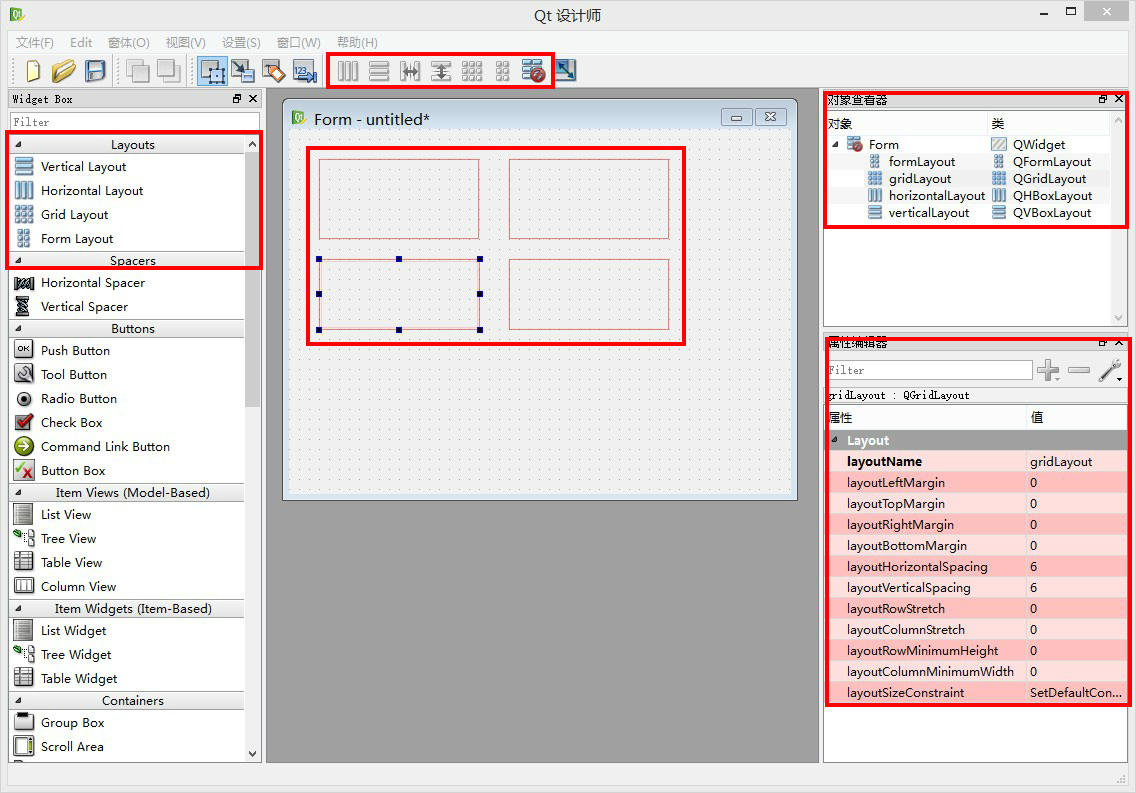
还可以直接编辑Qt引以为豪的信号槽（signal和slot）。

有了Qt Designer使得我们在程序设计中更快的能开发设计出程序界面，避免了用纯代码来写一个窗口的繁琐，

同时PyQt支持界面与逻辑分离，这对于新手来说无疑是个最大的福音，当然要做出华丽的界面还是要学代码的。

至少Qt Designer为我们提供了一些解决方法，另外我们也可以通过Qt Designer生成的代码来学习一些窗口控件的用法。

## PyQt5&python Gui入门教程（5）Qt Designer窗口布局Layouts（1）



Qt Designer窗口布局Layouts提供了四种布局方法，他们是：

Vertical Layout 纵向布局

Horizontal Layout 横向布局

Grid Layout  栅格布局

Form Layout  在窗体布局中布局

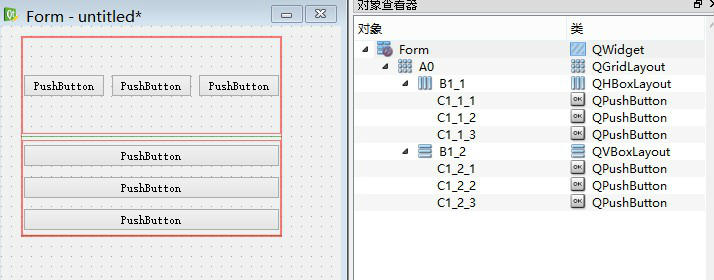
前三种是我们经常会用到的，我们将布局Layouts拖动到窗体上会有红色框来显示，

Layout的一些属性可以通过属性编辑器来控制，一般包括：

上下左右边距间隔，空间之间间隔等。

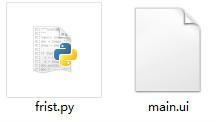
在我们使用布局之前，我们得对层次要有个了解，在程序设计中一般用父子关系来表示。

当然有过平面设计经验的童鞋对分层应该有所了解，这里我们还需要将层分成层次。

其实就像python中规定的代码缩进量代表不同层次的道理差不多。

从对象查看器中我们可以方便的看出窗体（Form）--布局（Layout）--控件（这里是PushButton按钮）之间的层次关系。

Form窗口一般作为顶层显示，然后使用Layout将控件按照我们想要的方式规划开来。



这里要注意一下，Qt Designer设计出来的文件默认为ui文件，里面包含的类css布局设计语言，

如果想要查看代码我们还需要将它转换（编译）成py文件，我们可以使用一条DOS命令来完成

D:\Python33\Lib\site-packages\PyQt5\pyuic5.bat mian.ui -o frist.py

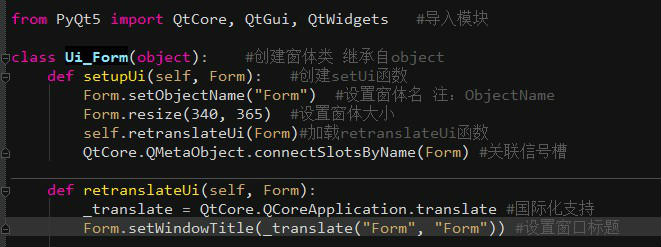
更实用的转换命令可以将当前文件夹下所有ui转换成py文件：

for /f "delims=" %%i in ('dir /b /a-d /s \*.ui') do D:\Python33\Lib\site-packages\PyQt5\pyuic5.bat %%i -o %%i.py

PyQt支持用LoadUi方法直接加载ui文件，当然我们通过转换后可以方便学习PyQt窗体控件的源代码。

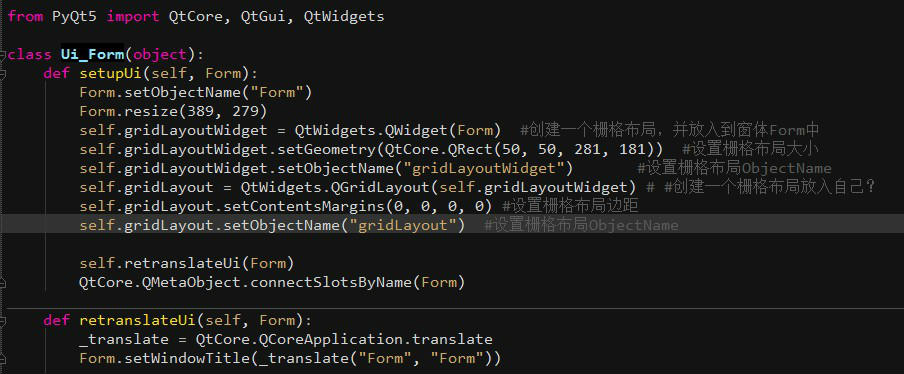
下一篇我们来分析一下Qt Designer布局的源代码

## PyQt5&python Gui入门教程（6）Qt Designer窗口布局Layouts（2）



首先我们来看一下，我们创建一个空白Widget窗体，Qt Designer都为我们做了些什么？

嗯，比我们第一个窗体，多了那么几行代码，默认转换后的py文件还不能直接显示出效果，下面我们慢慢学习。



这里我们只加入一个栅格布局，然后......Qt Designer居然给我们多了6行代码

其实Qt Designer生成的代码（编译后）还是非常给力的，至少比我们自己写的规范多了。

这些代码或许不需要我们自己写，但至少我们能看得明白。

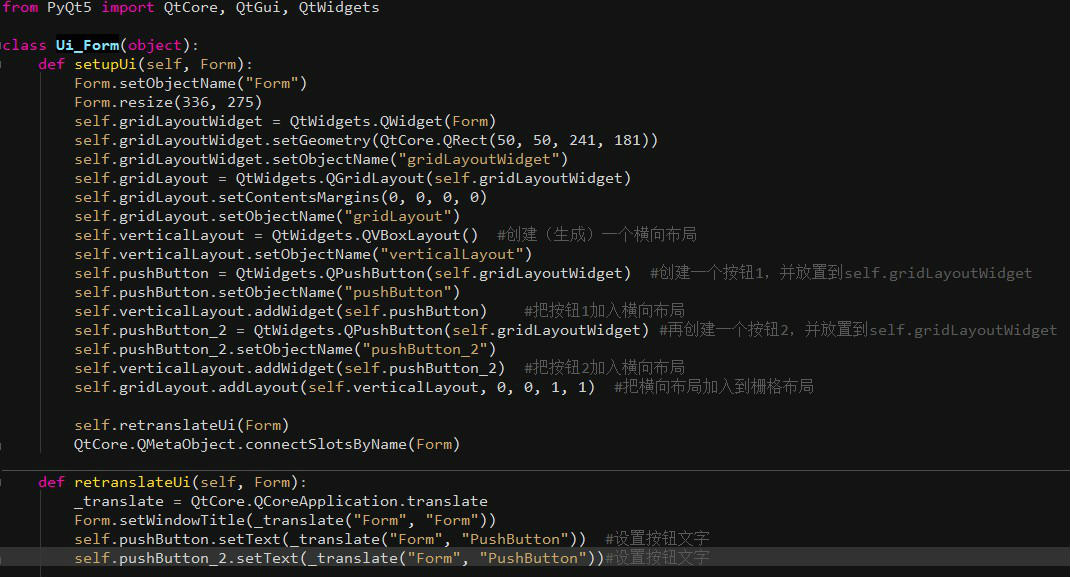
其中最主要的两句代码是：

self.gridLayoutWidget = QtWidgets.QWidget(Form)

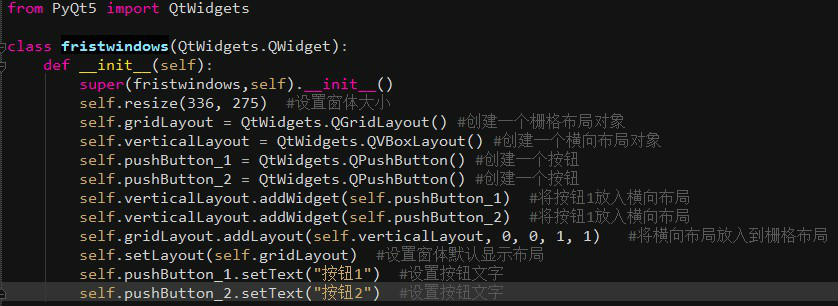
self.gridLayout = QtWidgets.QGridLayout(self.gridLayoutWidget)

从Qt Designer中可以看到栅格布局默认创建了两个对象，QtWidgets.QWidget和QtWidgets.QGridLayout。

注意(Form)和(self.gridLayoutWidget)参数，这里可以理解为放置到相应的对象里面。



Qt Designer果然做的比我们想想的多，下面我们再通过简化后的代码来对比一下，可能跟容易理解一些。

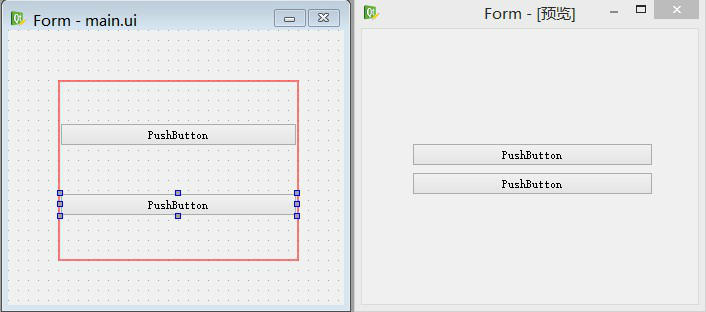


注意:Qt Designer首先会见栅格布局作为一个widget放入窗体，并且不影响窗体大小。

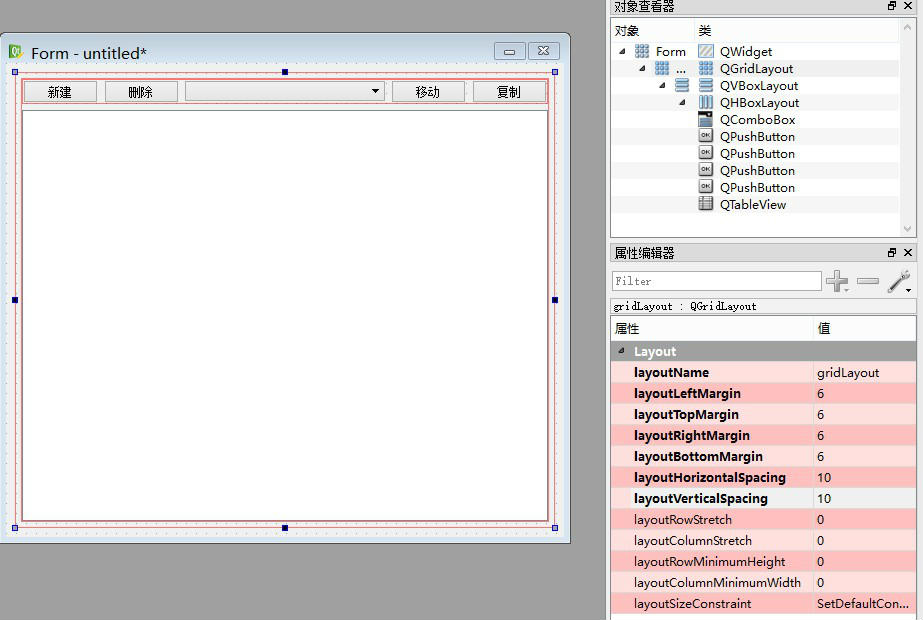
栅格布局一般作为顶层布局使用，所以简化后的代码有个self.setLayout（self.gridLayout）

同时Qt Designer会加入一些基本设置，默认信号槽声明，国际化语言支持等等，确实比我们自己写的要好的多。

当然我们需要了解其中的奥秘。



左边是Qt Designer设计效果，右边是显示效果。



再来个稍微复杂点的

知识点：

记得在Qt Designer中窗体的layout层次可以通过对象查看器来查看，layout的一些设置可以通过属性编辑器来修该。

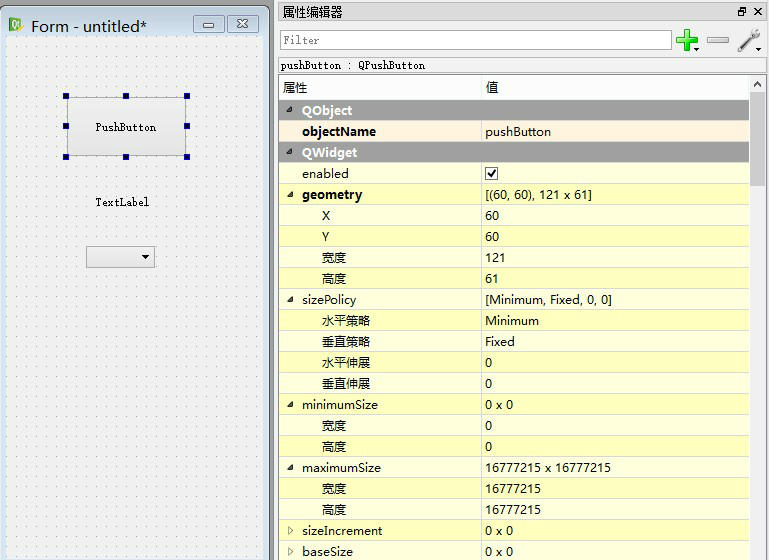
通常我们使用栅格布局作为顶层布局，将控件放置好之后可以通过右键--布局--栅格布局，将布局充满整个窗体。

我们可以先放入控件，然后ctrl选中多个控件，然后点击工具栏上快速布局工具进行布局。

在mianwindows中默认会有个centralwidget布局也是继承自QtWidgets.QWidget，表示窗口的中央部分。

在接下来的教程你会看到mianwindows的使用技巧。

## PyQt5&python Gui入门教程（7）Qt Designer控件的一些通用属性



在Qt Designer中的右边为我们提供了窗体、控件、布局的属性编辑功能。

比较常用的有:

objectName 控件对象名称 例：anniu = QtWidgets.QPushButton() 等号前面的那个名字

geometry 相对坐标系

sizePolicy 控件大小策略

minimumSize最小宽度、高度

maximumSize最大宽度、高度  如果想让窗体或控件固定大小，可以将mini和max这两个属性设置成一样的数值

font 字体

cursor 光标

windowTitle 窗体标题

windowsIcon / icon 窗体图标/控件图标

iconSize 图标大小

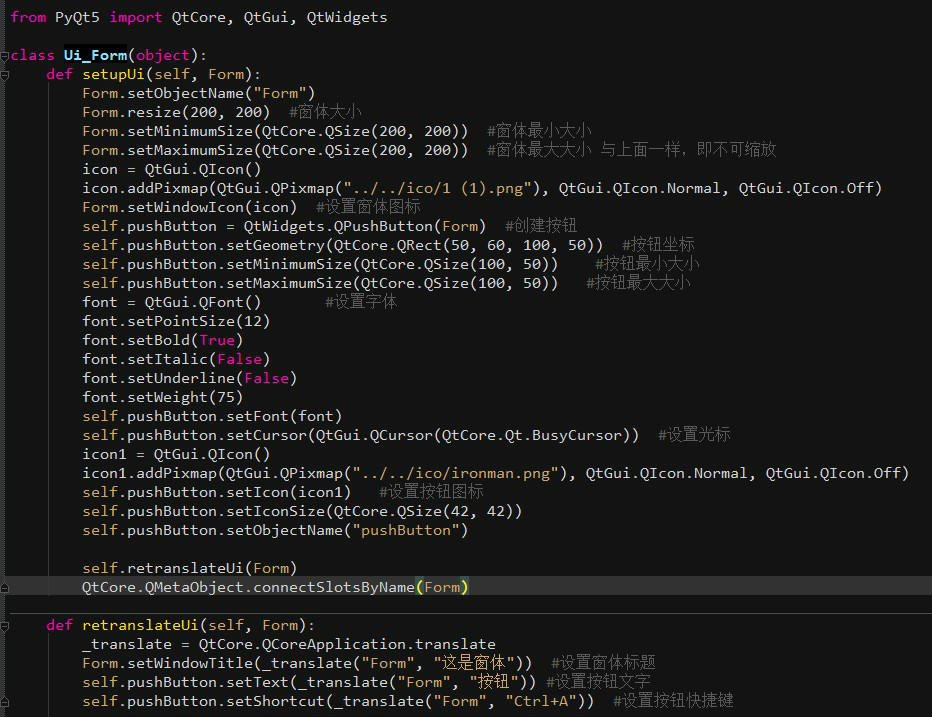
toolTip 提示信息

statusTip 任务栏提示信息

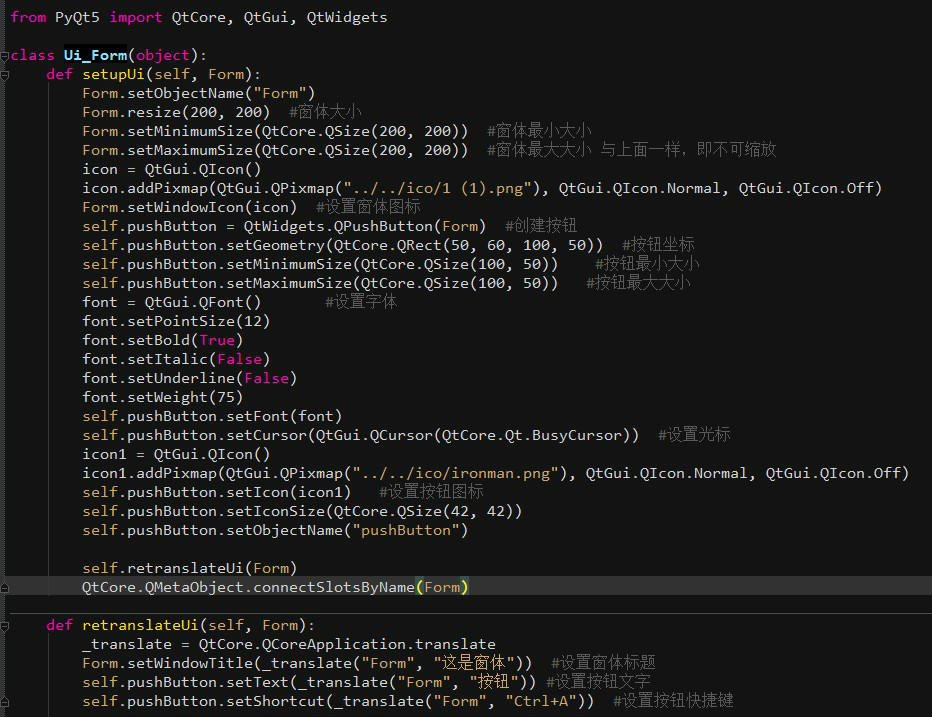
text 控件文字

shortcut 快捷键





## PyQt5&python Gui入门教程（8）Qt Designer界面与逻辑分离

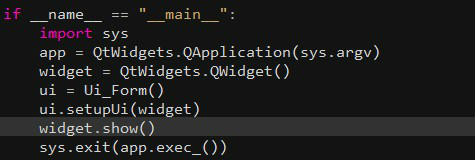


这是上一篇文章中用Qt Designer设计编译后生成的代码，但是想要让它显示出来，还是得费点功夫，

对于新手来说经常会碰壁，我也是折腾了好长时间才明白，原来还要做一下一些事情。



这是显示效果。



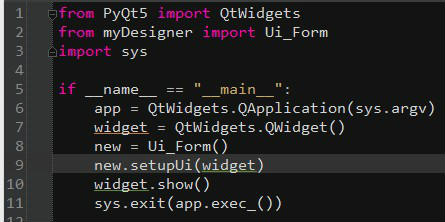
如果我们在单个文件当中显示，还需要在结尾加入以上代码。

因为Qt Designer默认继承的object类，不提供show()显示方法，

所以我们生成一个QWidget对象来重载我们设计的Ui\_Form类，达到显示效果。

在实际工作中，我们希望界面和代码分离，而防止破坏Qt Designer编译生成的代码，

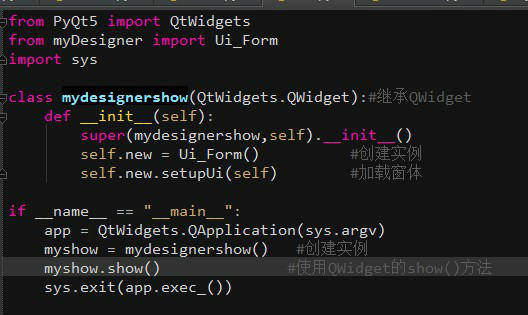
我们可以这样做：



新建一个文件，导入我们设计的myDesigner.py文件，然后下面的代码相同即可。

但是这是在mian方法中调用窗口，也不符合我们的实际工作中的要求。

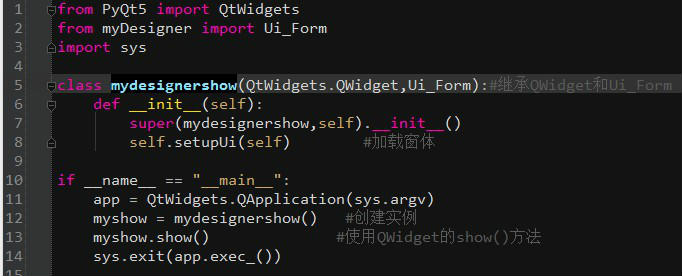
我们还可以这样搞：



默认的Qt Designer生成的代码是继承object类，所以默认编译生成的文件不能显示。

通过不同的方法，可以知道我们需要继承或重载QWidget这个类，来使用其中的show方法使窗口显示。

这也就是传说中的界面与代码分离，其实也没有想象中的那么难。



我们还可以通过同时继承QWidget和我们设计的Ui\_Form类来简化代码，这就是传说中的多态？

之前我们见过，\_\_init\_\_是析构函数，也就是类被实例化（生成对象）之后默认加载的内容。

super().\_\_init\_\_()超级加载，也就是我们需要同时加载QWidget中的内容供我们使用。

self.setupUi(self) self是指自己，setupUi也就是Ui\_Form中的方法。

还记得之前说的 ：实例.方法(参数)  self在实例是代表自己这个类

myshow 是实例，mydesignershow()是类，用=代表创建一个实例对象。

myshow.show()  然后调用对象中的方法，相当于调用类里面的函数。

## PyQt5&python Gui入门教程（9）Qt Designer 初探信号槽

信号与槽作为QT 的核心机制，往往给新手带来很多困扰，这也不是一天两天能搞懂学透的东西，

我们的宗旨是要会用，就算是简单的运用吧，学以致用才是正道，也是沧桑啊！

先入一些信号槽的基本介绍：

信号和槽是一种高级接口，应用于对象之间的通信，它是 QT 的核心特性，也是 QT 区别于其它工具包的重要地方。它为高层次的事件处理自动生成所需要的附加代码。在我们所熟知的很多 GUI 工具包中，窗口小部件 (widget) 都有一个回调函数用于响应它们能触发的每个动作，这个回调函数通常是一个指向某个函数的指针。但是，在 QT 中信号和槽取代了这些凌乱的函数指针，使得我们编写这些通信程序更为简洁明了。

所有从 QObject 或其子类 ( 例如 Qwidget) 派生的类都能够包含信号和槽。当对象改变其状态时，信号就由该对象发射 (emit) 出去，这就是对象所要做的全部事情，它不知道另一端是谁在接收这个信号。这就是真正的信息封装，它确保对象被当作一个真正的软件组件来使用。槽用于接收信号，但它们是普通的对象成员函数。一个槽并不知道是否有任何信号与自己相连接。而且，对象并不了解具体的通信机制。

你可以将很多信号与单个的槽进行连接，也可以将单个的信号与很多的槽进行连接，甚至于将一个信号与另外一个信号相连接也是可能的，这时无论第一个信号什么时候发射系统都将立刻发射第二个信号。总之，信号与槽构造了一个强大的部件编程机制。

说实话对于像我这样的新手来说看着就蛋疼，想学会它没办法，我们还是简化一下概念吧：

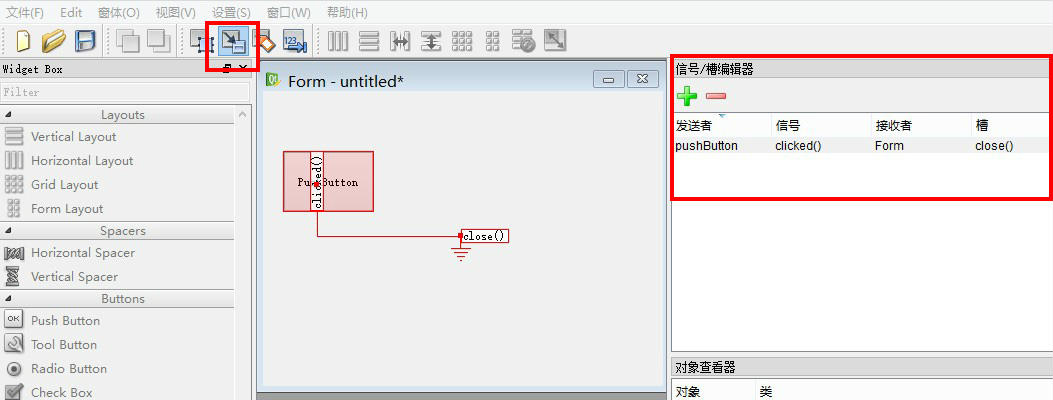
所有QObject类都可以使用信号槽，换句话来说继承自pyqt中的类基本上都可以使用信号槽机制。当然非QObject也是可以通过其他一些办法来使用信号槽的。

仅仅有了信号和槽是不行的，我们还需要了解：

信号(Signal)、槽(slot)、连接(connect)、动作事件(action)、发射(emit)、发送者、接受者等等一些列的知识。

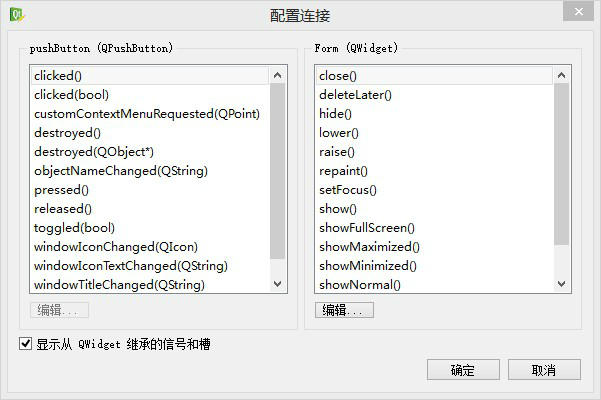
好吧，别搞的那么复杂行不行，我们还是学学该怎么用吧。

在Qt Designer中为我们提供了一些基本的信号槽方法，我们来看看：



点击工具栏上的“编辑信号/槽”，进入信号槽编辑模式，

我们可以直接在发送者（button）上按住不放拖动到接收者（Form窗体）上既可以建立连接。



然后会弹出信号槽的配置连接。

左边是发送者（按钮）的信号（动作事件），右边是接收者（窗体）的槽（动作事件）

我们看一下编译后生成的代码：

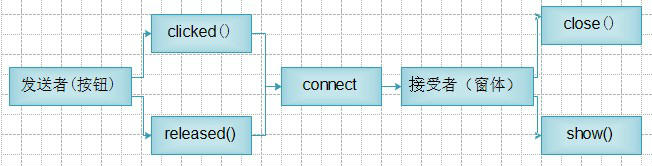
pushButton.clicked.connect(Form.close)

简单解释就是：当按钮点击之后关闭窗体。

实际却是：按钮.点击.链接(窗体.关闭)

其实应该是：按钮这个发送者，当按钮“点击”这个时间发生之后会发送一个信号出去，通过这段代码程序内部的通讯机制知道这个按钮的点击事情被连接到窗体的关闭事件上去了，然后通知接受者窗体，你该运行槽函数close了！

简单理解：比如我写了封信给你，送到邮局，邮局通过地址，转交给你。所以一个槽并不知道是否有任何信号与自己相连接。而且，对象并不了解具体的通信机制。



其实以上只是个人的浅显理解，具体的通信机制还是非常复杂的，想象一下邮局的工作就知道啦。

我们再来看看一些具体的代码：

        self.pushButton.clicked.connect(Form.close)

        按钮点击连接到窗口关闭

        self.pushButton.clicked.connect(self.pushButton\_2.hide)

        按钮点击连接到按钮\_2隐藏

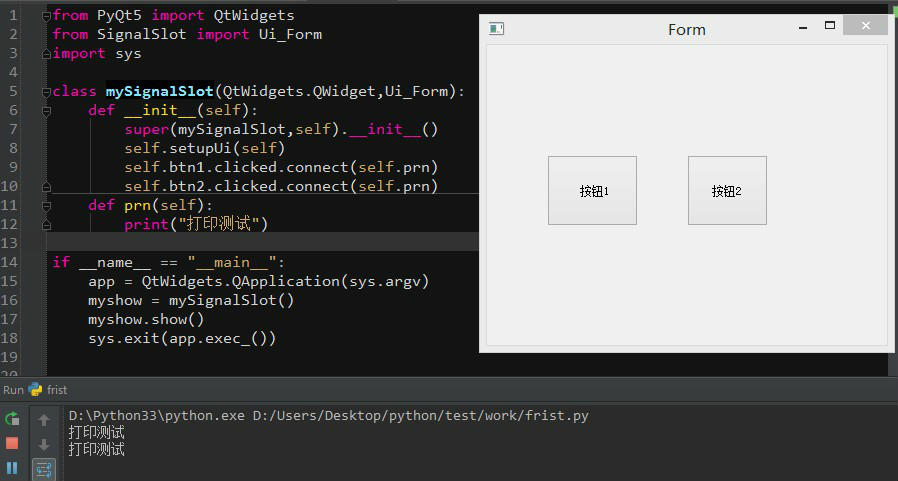
        self.pushButton.destroyed.connect(self.pushButton\_2.close)

        按钮销毁连接到按钮\_2关闭

        self.pushButton.released.connect(Form.update)

        按钮重载连接到窗口更新

那么我们怎么能执行自己的“槽”呢？



通过之前学习的方法，我们继承了Ui\_Form，然后将按钮1和按钮2都连接到prn函数上，测试OK。

对于初学者想要简单使用信号槽的机制，其实并不难，当然其中也有不少坑，还是多多测试为妙！

知识点：

        在PyQt中接受者和发送者必须是个对象（实例）！

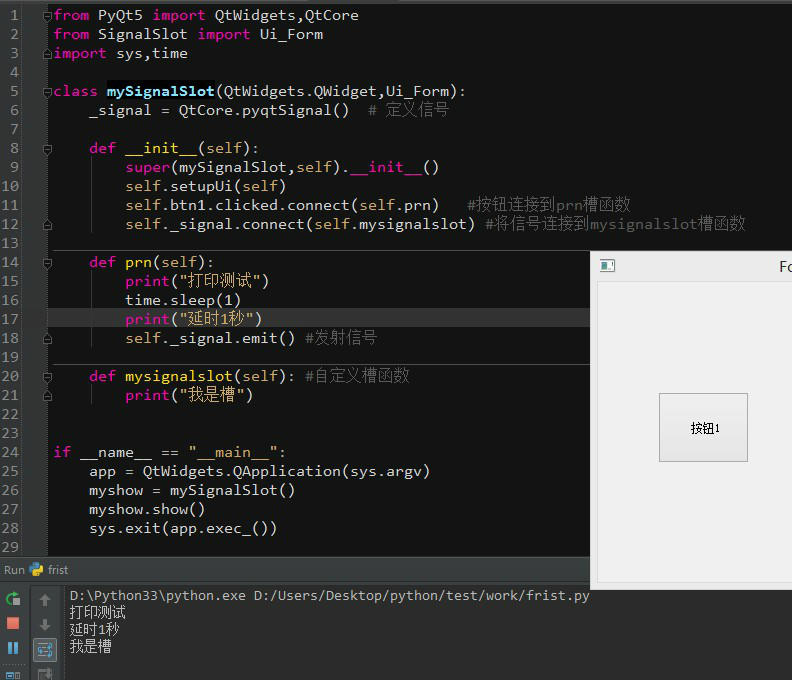
        PyQt中的控件中提供了很多信号和槽方法，大家可以多多使用Qt Designer 设计参考！

        槽其实就个函数（方法），Qt5中的槽函数不在限定必须是slot，可以是普通的函数、类的普通成员函数、lambda函数等。编译期间就会检查信号与槽是否存在！

        信号的connect连接最好放在\_\_init\_\_析构函数里面，这样只会声明一次连接，如果在类方法（函数中）使用的话，要记得disconnect，否则connect会连接多次，导致程序异常。

        信号槽函数不用加 ()，否则可能会导致连接异常。

## PyQt5&python Gui入门教程（10）Qt Designer 自定义信号emit及传参



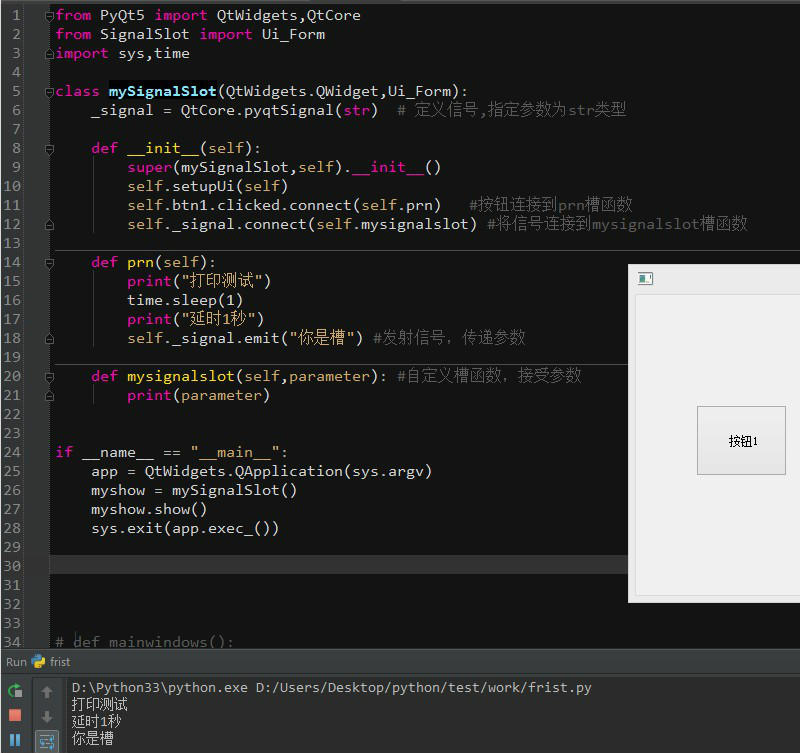
上面的代码简单的自定义了一个信号，并连接到自定义的槽函数中。

其实看起来也不是很难的样子。

注意：声明信号必须在类属性中定义，也就是所有方法之前。

使用emit可以发射信号，然后通过connect将信号与槽函数连接。

下面我们来学习一下信号与槽之间的参数传递：



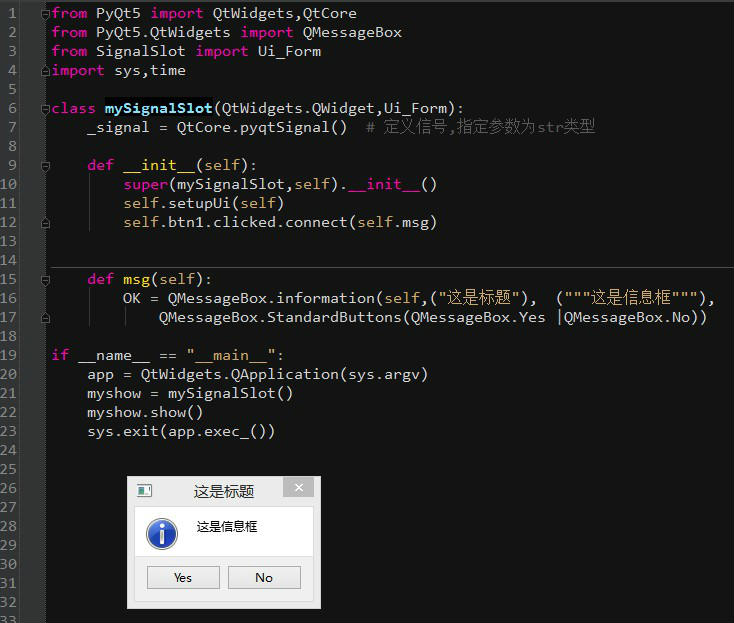
注意：当信号与槽函数的参数数量相同时，它们参数类型要完全一致。信号与槽不能有缺省参数。

          当信号的参数与槽函数的参数数量不同时，只能是信号的参数数量多于槽函数的参数数量，且前面相同数量的参数类型应一致，信号中多余的参数会被忽略。此外，在不进行参数传递时，信号槽绑定时也是要求信号的参数数量大于等于槽函数的参数数量。这种情况一般是一个带参数的信号去绑定一个无参数的槽函数。

当然可以出传递的参数类型有很多种：str、int、list、object、float、tuple、dict等等

至于信号槽使用注意事项可以百度一下

## PyQt5&python Gui入门教程（11）通用对话框QMessageBox



PyQt5中为我们提供了很多默认信息框QMessageBox，注意为方便使用需要导入模块。

QMessageBox对话框包含类型只是图标不同其他无太大差别:

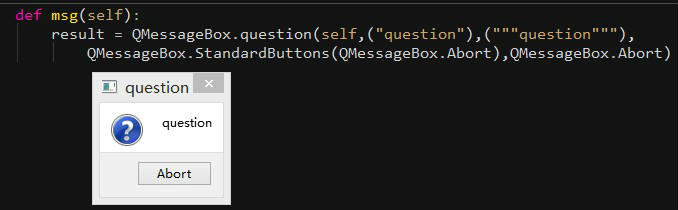
QMessageBox.information 信息框

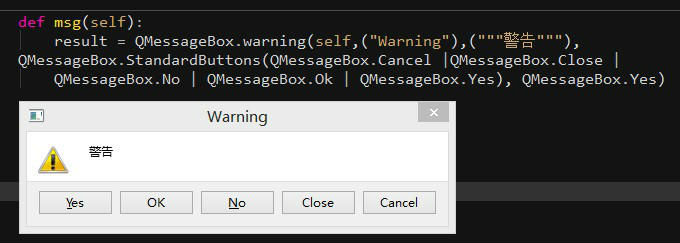
QMessageBox.question 问答框

QMessageBox.warning 警告

QMessageBox.ctitical危险

QMessageBox.about 关于





基本参数：（指定父组件，“标题”，“信息内容”，要显示的按钮，默认按钮）

Standard Buttons 默认要显示的按钮，多个按钮使用 | 隔开。其中包含，可以任意组合：

StandardButtons(QMessageBox.Abort |QMessageBox.Apply | QMessageBox.Cancel |

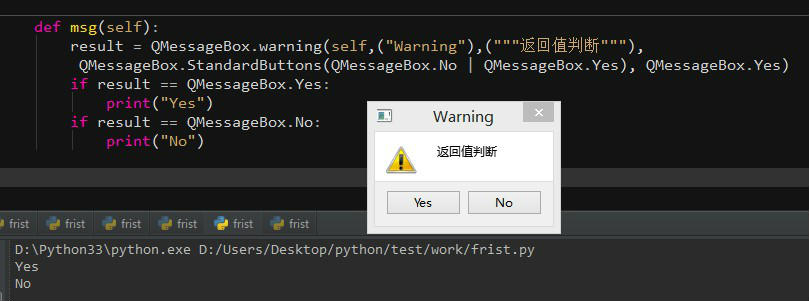
        QMessageBox.Close | QMessageBox.Discard |QMessageBox.Help |

        QMessageBox.Ignore | QMessageBox.No | QMessageBox.NoToAll |

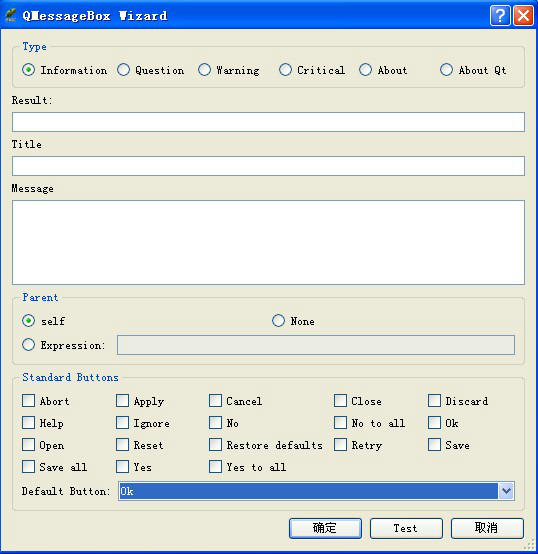
        QMessageBox.Ok |QMessageBox.Open | QMessageBox.Reset |

        QMessageBox.RestoreDefaults | QMessageBox.Retry |QMessageBox.Save |

        QMessageBox.SaveAll | QMessageBox.Yes | QMessageBox.YesToAll)

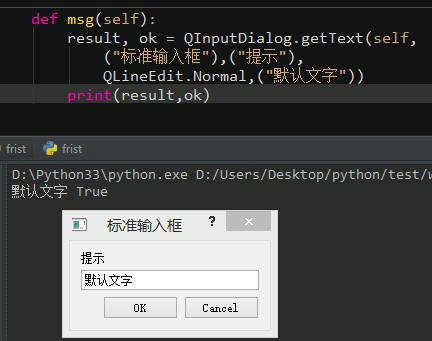


result 为返回值 默认为数值 我们可以通过上面的方法来判断用户点击了那个按钮！



之后来一张向导图，来自eric。

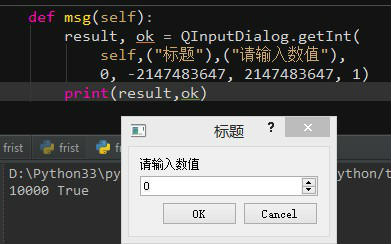
## PyQt5&python Gui入门教程（12）标准输入框QInputDialog



标准输入文字框QInputDialog.getText()

需要QInputDialog和QLineEdit模块，

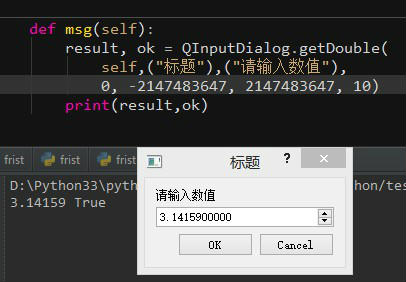
默认返回输入框文字和按钮bool值



标准整数输入框QInputDialog.getInt()

默认返回输入框文字和按钮bool值

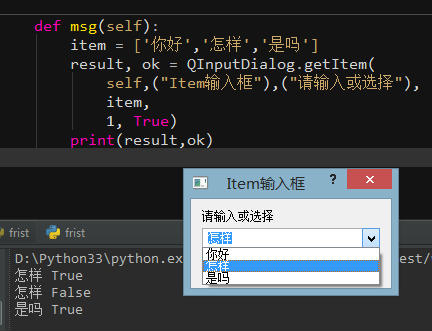
其中0, -2147483647, 2147483647, 1 为默认数值，数值范围，和步长（即按上下按钮时数值增加或减少多少）



标准浮点数输入框QInputDialog.getDouble()

默认返回输入框文字和按钮bool值

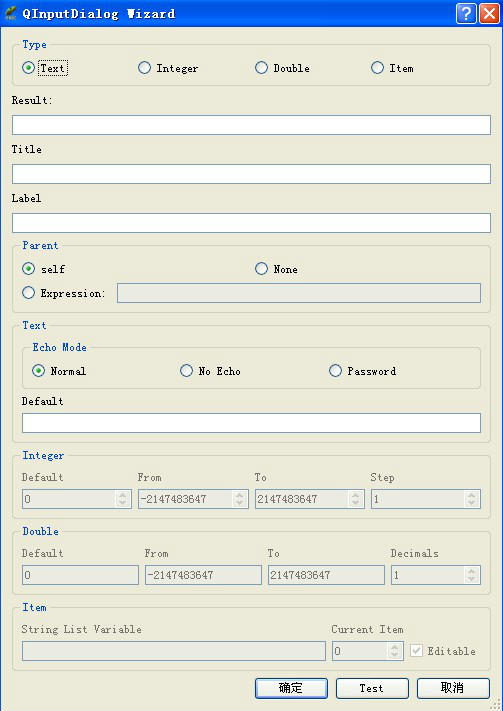
其中0, -2147483647, 2147483647, 10 为默认数值，数值范围，最后10代表小数位数。



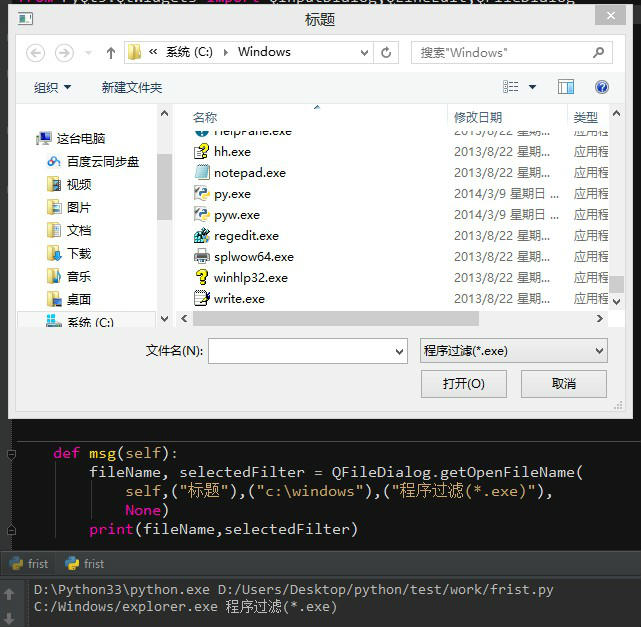
列表输入选择框QInputDialog.getItem()

默认返回输入框文字和按钮bool值

条目添加为列表类型，其中item为自定义列表，1为默认选中选项目，True/False  列表框是否可编辑。



## PyQt5&python Gui入门教程（13）标准文件打开保存框QFileDialog



单个文件打开 QFileDialog.getOpenFileName()

多个文件打开 QFileDialog.getOpenFileNames()

参数（指定父组件，“标题”，“默认打开路径”，“后缀名过滤器”）还有很多

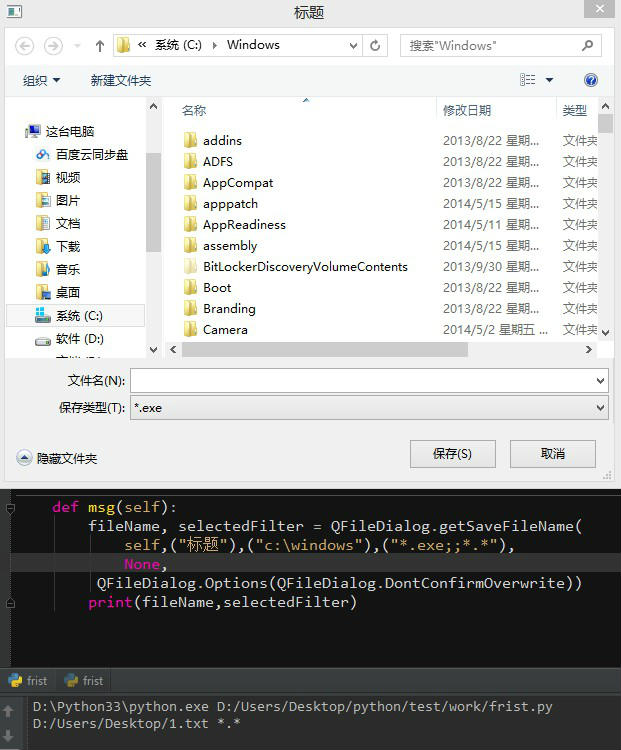
返回值 默认返回文件路径和当前过滤名

如果需要多种类型过滤，可以这样"jpg（\*.jpg）;;exe（\*.exe）;;（\*.\*）"用两个;;分割开。

getOpenFileName()函数在Windows和MacOS X平台上提供的是本地的对话框，

而QFileDialog提供的始终是Qt自己绘制的对话框，而不都是调用系统资源API。

注意多个文件打开，若要包含子目录还需要使用别的手段，自行百度吧！

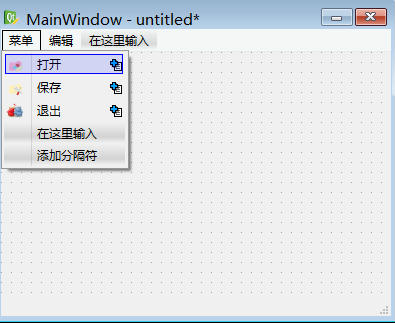


文件保存对话框 QFileDialog.getSaveFileName()

QFileDialog.DontConfirmOverwrite 覆盖提示？

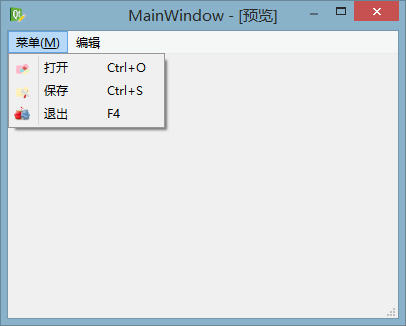
返回值同上，同时具体生成文件内容还需要自己实现。

## PyQt5&python Gui入门教程（14）Qt Designer 主窗口MainWindows



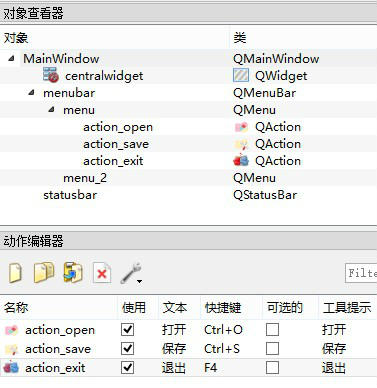
MainWindows 即主窗口，主要包含了菜单栏、工具栏、任务栏等

双击菜单栏上的“在这里输入”，然后录入文字，回车即可。注意要用回车键确认。

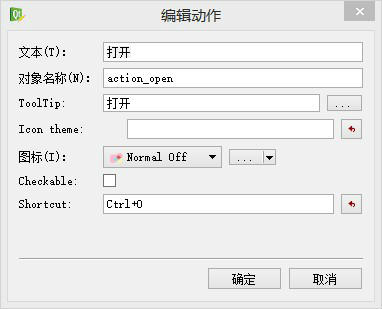


预览效果，并且加入了快捷键显示

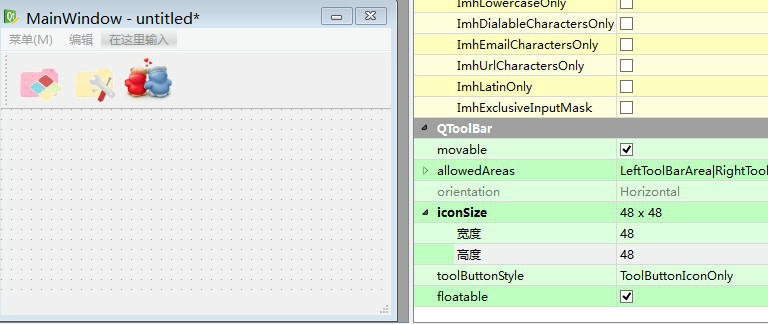
我们可以这样做  菜单(&M)  编辑(&E) 来加入菜单的快捷按钮在预览中按alt+m  alt+e 可以看到效果



下拉菜单项 是action 我们可以通过动作编辑器找到他们

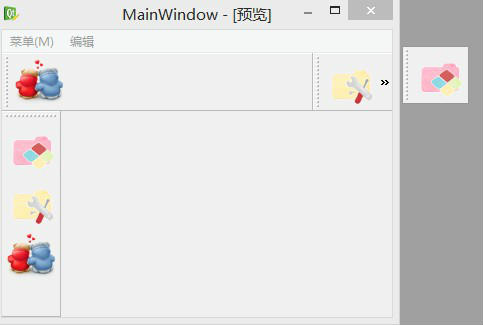


双击需要编辑的action，可以对其进行设置并添加图标、快捷键等等。

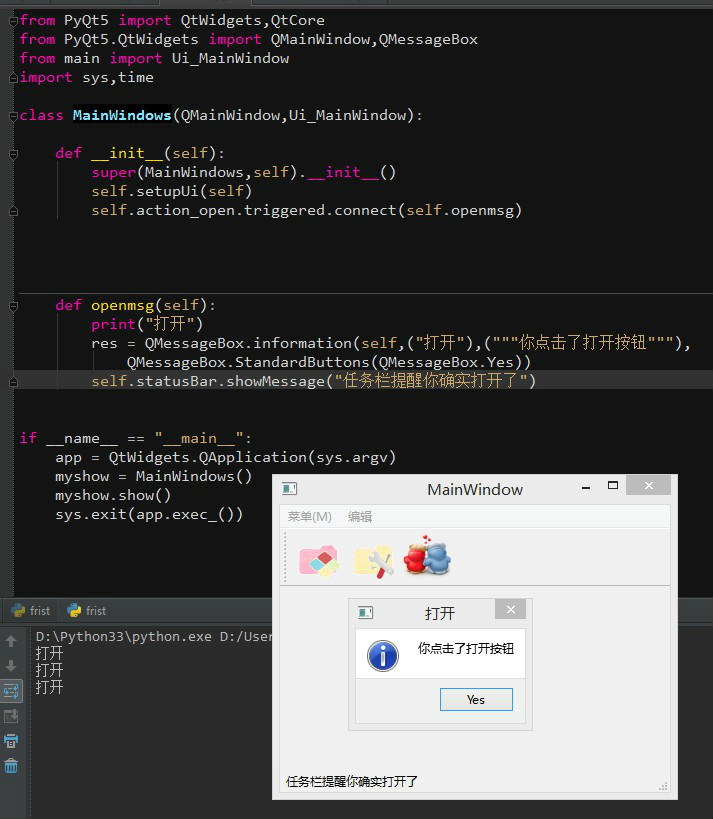


工具栏，默认在PyQt5中不显示工具栏，我们可以点击右键来添加工具栏，通过属性编辑器我们可以修改图标大小。

工具栏上也是action，在动作编辑器上选择需要的action拖入即可。



我们可以添加多个工具栏，在预览中我们可以拖放工具栏，甚至是浮动出去，有点像PS的工具栏有木有？



我们来个综合实例看看

首先需要导入QMainWindow类 并继承QMainWindow 生成自己的类。前面已经介绍很多不再复述！

然后将打开动作链接到自定义函数槽中，注意action的点击动作为triggered。

槽函数中添加了一个通用对话框的信息框提示，

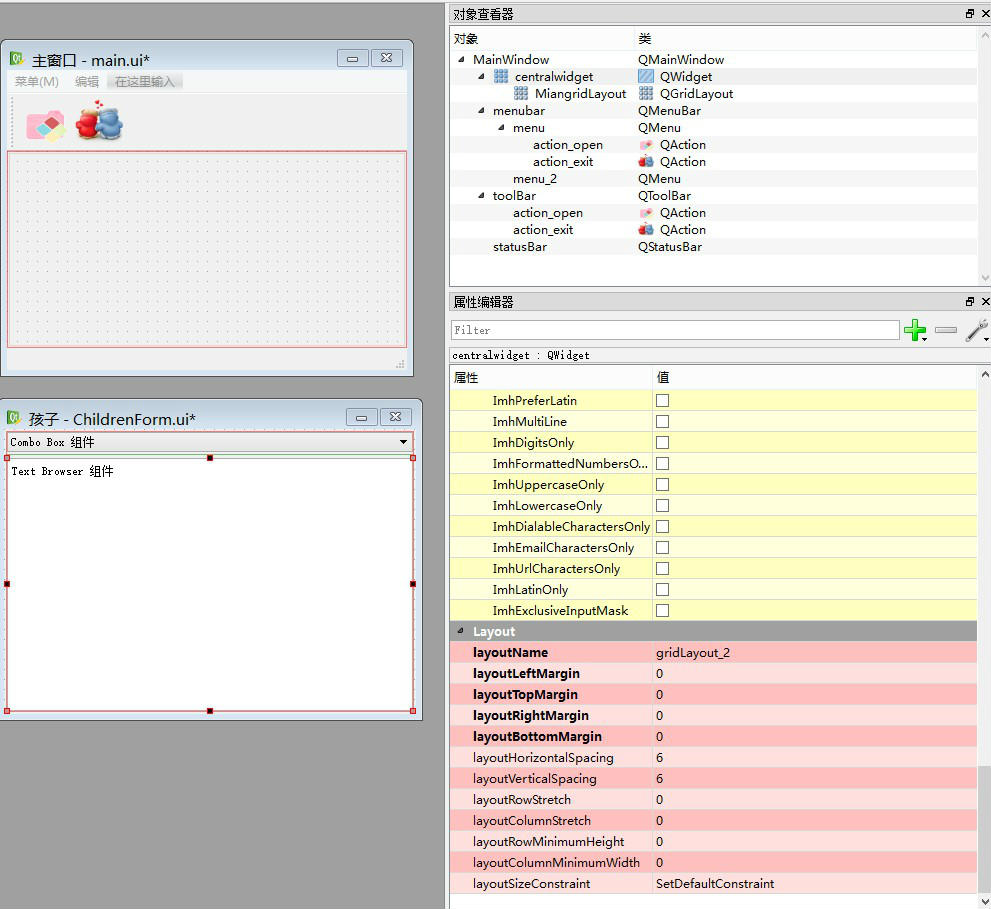
并且设置了任务栏提醒statusBar.showMessage()方法。

通过一个简单的例子可以看到PyQt5&python Gui开发结合Qt Designer非常的便捷和高效，其实入门也没那么难！

我们可以通过Qt Designer编译出来的文件去分析学习Gui构建，然后通过代码分离来实现自己的方法。

## PyQt5&python Gui入门教程（15）Qt Designer 主窗口动态加载Widget

本人的教程比较基础，所用方法不够亮骚，仅供新手入门，若大家有更好的方法不胜赐教！



我们通过Qt Designer设计两个窗口，这里暂时命名为主窗口(mian)和孩子窗口(ChildrenForm)吧

我们在主窗口的空白中央添加一个栅格布局并命名为MiangridLayout，等会需要将ChildrenForm放进去。

之前我们知道mainwindows默认有个centralwidget主层次布局，这里我们把centralwidget的边距需要设置一下见属性编辑器。

ChildrenForm也用了栅格布局同时添加了一个横向布局和两个组件。



首先导入必要的模块，并且导入我们自己设计的窗口类文件。

然后通过之前学过的方法来继承窗口类。

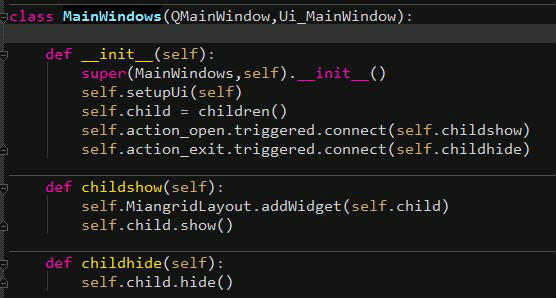
接着我们在主窗口后添加了两行代码

self.child = children()生成子窗口实例self.child

self.MaingridLayout.addWidget(self.child) 将实例加入到Layout用addWidget方法



最终显示效果，界面与逻辑完美分离，且实现了动态加载我们自己设计的子窗体。



我们在改改代码，分别定义了两个函数，用来显示和隐藏子窗口，并将菜单按钮动作分别连接上去。

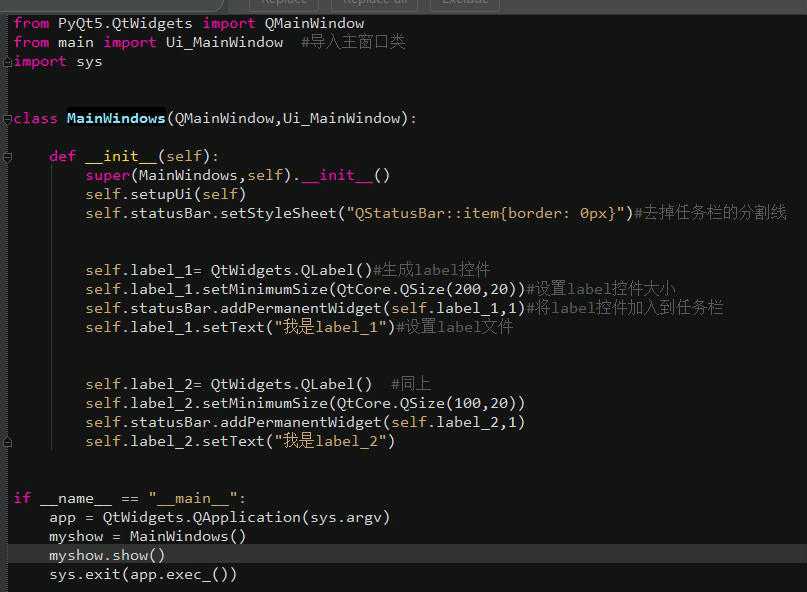
## PyQt5&python Gui入门教程（16）Qt Designer 主窗口之任务栏

上一篇我们提到了主窗口的任务栏功能，默认的任务栏只提供了一条信息框，

但是很多时候我们需要分栏来显示不同的消息，这时候我们就需要自己去搞定了。



首先我们设计一个主窗体（为了方便沿用上一篇的设计），并添加任务栏。



下面我们来看看实现的步骤吧，

首先为了美观去掉任务栏分割线，然后生成label并设置大小，

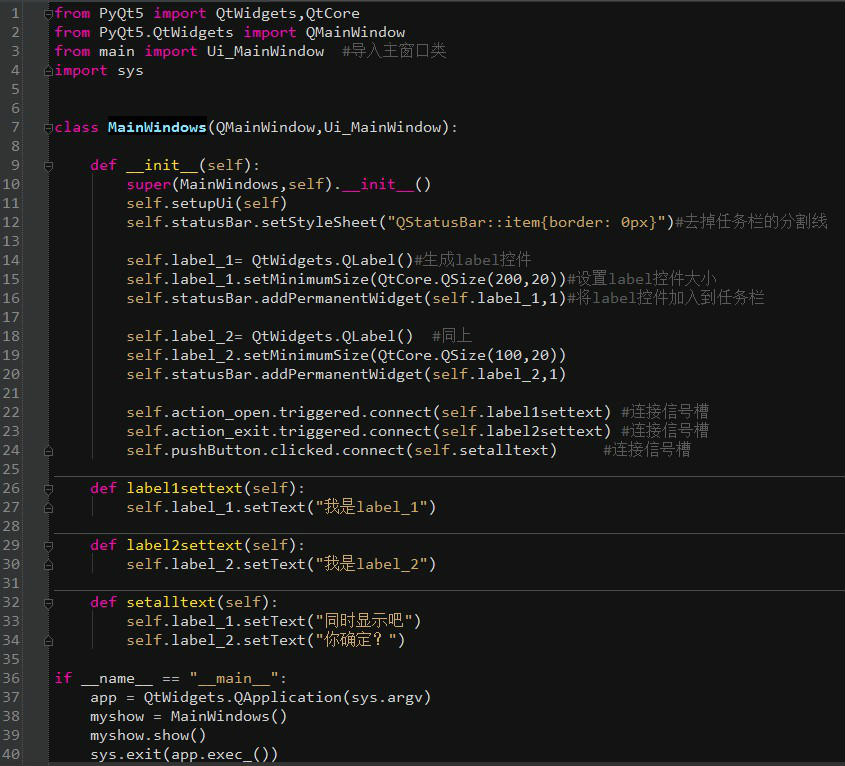
然后用addPermanentWidget方法加入到任务栏，为了防止任务栏信息遮挡label文字

然后设置label要显示的字体来测试效果。



运行效果，so easy！

我们可以知道通过之前学习的方法我们继承窗体类，也学习了动态加入窗体，现在也知道还可以动态加入控件。



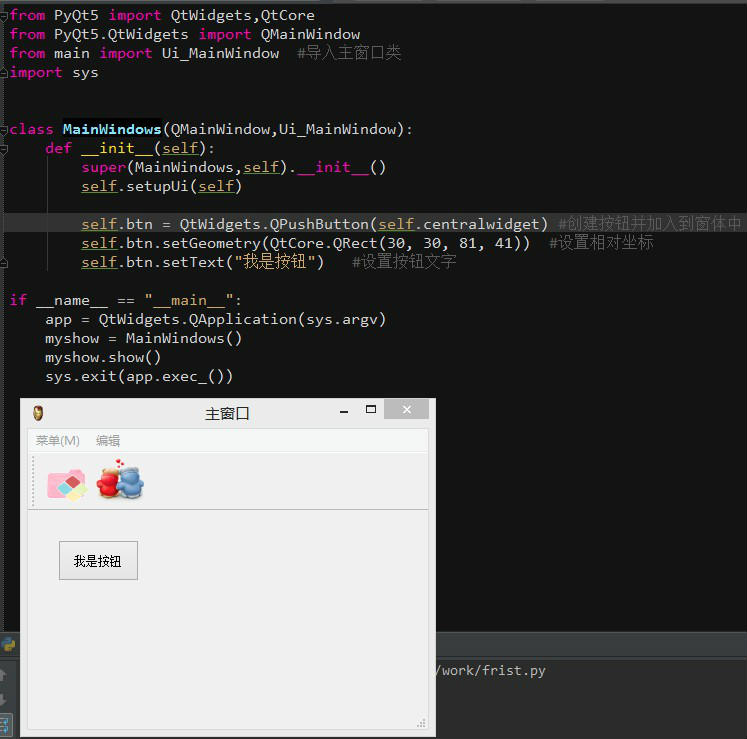
结合之前学过的知识，我们在稍微改一改代码，增加3个函数，并连接到相应的按钮上，OK我们来看看效果。



## PyQt5&python Gui入门教程（17）Qt Designer 按钮PushButton

按钮PushButton是Gui开发中最常用的控件之一，通过按钮来进行交互并完成一些事情。

下面我们来看看PushButton的一些基本用法，



简短的三行代码即可将一个按钮放置到窗体中，其中self.centralwidget是mainwindows自带的，代表中间空白区域。

然后通过setGeometry()设置按钮在窗体中的相对坐标，相对与窗体的坐标系而不是整个屏幕的坐标系。

最后设置一下文字setText(),非常简单。

通过代码体验一下按钮的创建，当然通常我们会通过Qt Designer来设计，同时Qt Designer还提供了很多常用的设置项，

比如：字体font、图标icon、大小size、快捷键shortcut、提示信息toolTip等等，看看下图。



设置一个图标，代码：

        icon1 = QtGui.QIcon()

        icon1.addPixmap(QtGui.QPixmap("../../ico/ironman.png"), QtGui.QIcon.Normal, QtGui.QIcon.Off)

        self.btn.setIcon(icon1)

        self.btn.setFlat(True)  不显示边框，点击的时候还是会有边框提示的

        self.btn.setToolTip("设置提示内容")

        self.btn.setEnabled(True) 是否可用

        self.btn..setShortcut("F4")设置快捷键

常用的事件有，一般用于连接槽函数。

clicked() 按钮点击

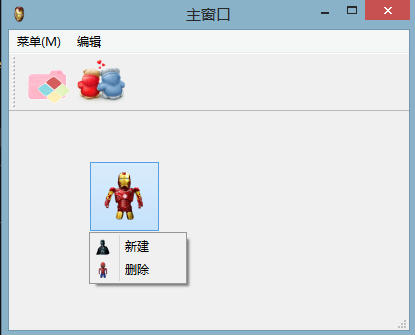
pressed()   按钮按下

released()  按钮释放

toggled()   切换按钮状态

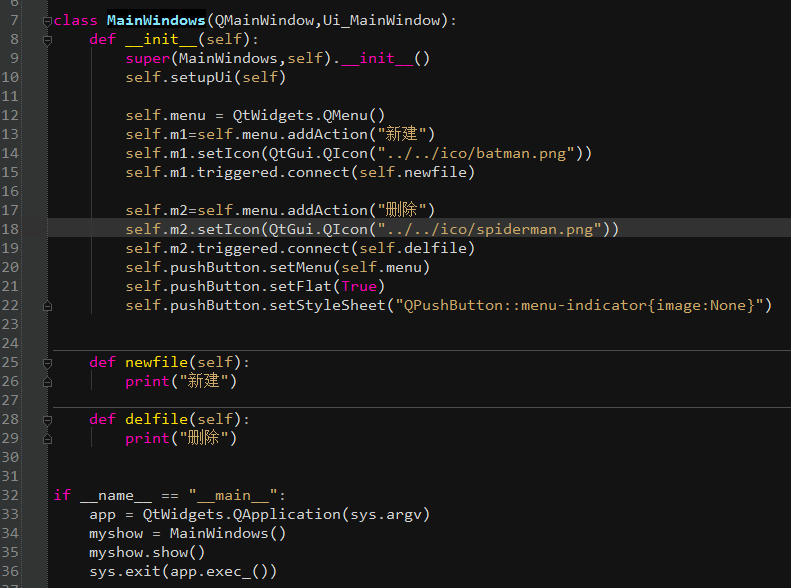
判断按钮是否被按下可以用isDown ()

## PyQt5&python Gui入门教程（18）Qt Designer 按钮PushButton显示菜单



先上效果图，有的时候我们需要让按钮弹出一个自定义菜单，来让程序显得更酷更个性化，

下面我们就来看看代码吧。



想要创建一个菜单self.menu = QtWidgets.QMenu()，

然后通过后面的self.pushButton.setMenu(self.menu)将按钮重载。

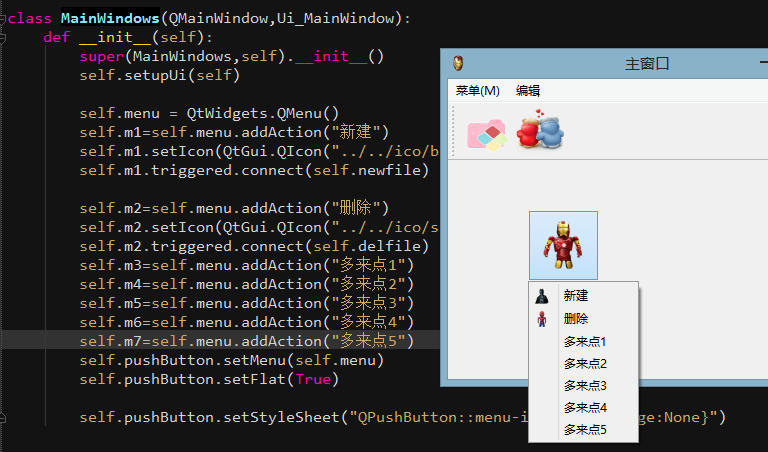
self.m1,self.m2 是两个action，

通过addAction 加入到self.menu中，

同时我们还为action设置了图标，连接了信号槽。

self.pushButton.setFlat(True) 不显示按钮边缘。

self.pushButton.setStyleSheet("QPushButton::menu-indicator{image:None}") 去掉下拉提示箭头。

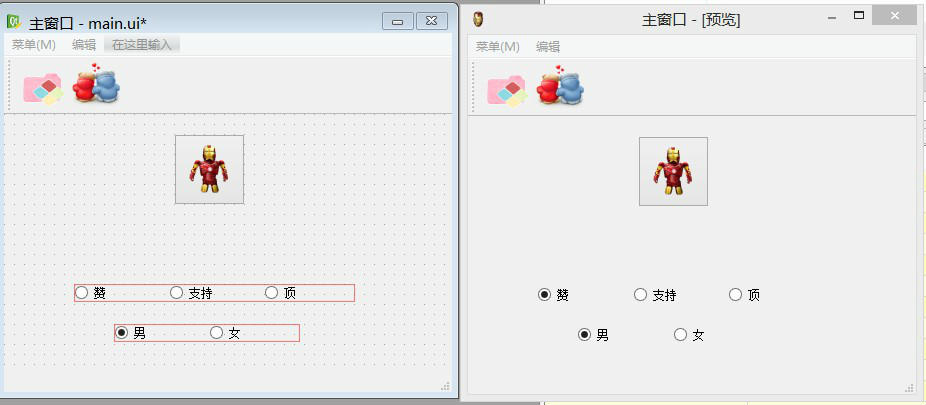


按钮菜单加长版！

之前讲过mainwindows中的菜单栏，大家可以研究一下mainwindows中的菜单代码，非常雷同。

## PyQt5&python Gui入门教程（19）Qt Designer 单选框RadioButton

单选框RadioButton作为Gui开发常用组件之一，今天咱们来简单介绍一下，

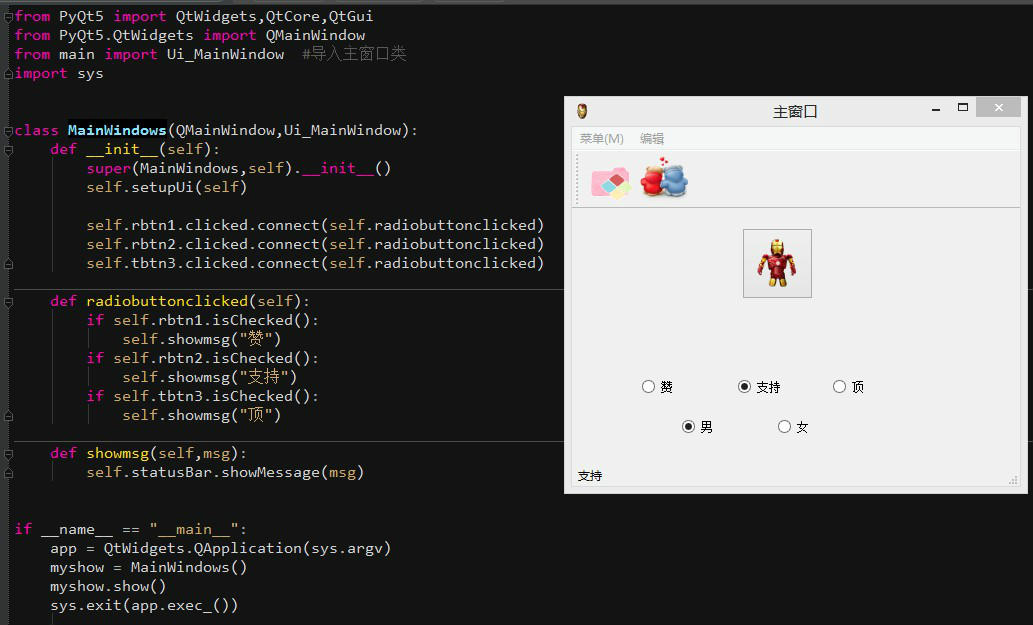


如图所示，左边是在Qt Designer完成的，右边是效果预览。

第一排horizontalLayout布局中放入了三个单选框，第二排放入了两个，并将其中一个设置为clicked(),

这样我们实现了两排单选框互不干扰，同时clicked()也实现了默认必选一项的功能。

对于单选框的判断也很简单，可以用isChecked()。



我们看下实现代码，先将单选框动作连接到槽函数，然后自定义槽函数显示任务栏信息。

为了方便显示这里也单独定义了个showmsg()函数通过传参来显示任务栏信息。

如果要确定2个单选框被按下可以用and链接，列如：

if self.tbtn3.isChecked() and self.man.isChecked():

    self.showmsg("男人你顶了一下哦！")

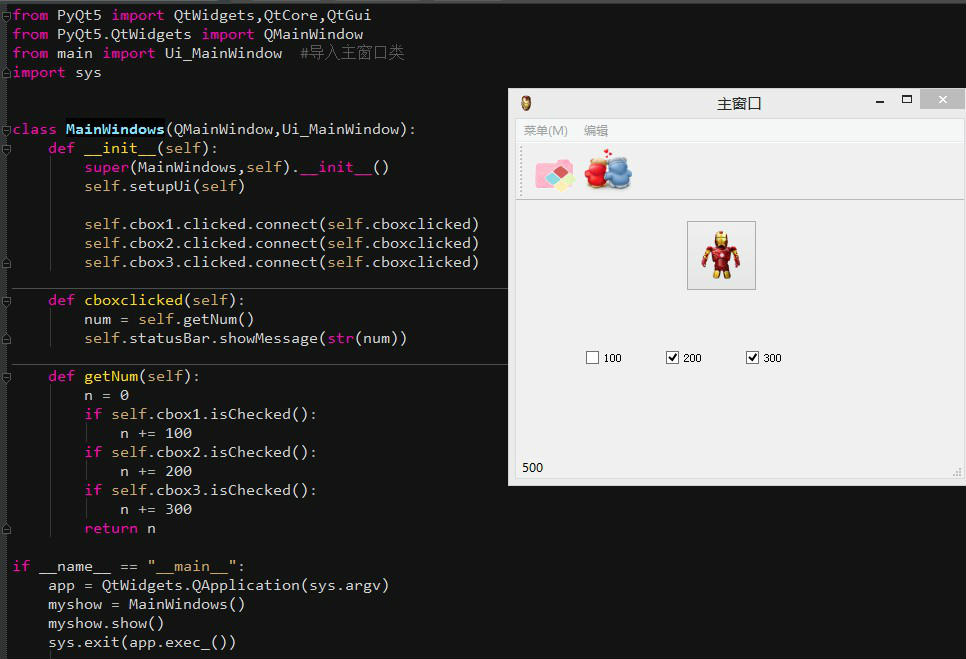
前面我们说过定义类class、函数def 需要用:结尾，记得判断语句也需要用:结尾来标识，比如：if，while等等。

## PyQt5&python Gui入门教程（20）Qt Designer 复选框CheckBox



 复选框CheckBox相比单选框来说是可以重复多选的，这也是我们常用的组件之一。

它的用法和单选框基本相似，下面我们来看看简单的数相加的实现代码：



同样将checkbox按钮动作连接到槽函数checkclicked()

槽函数通过调用getNum()来获取当前按下的值。

也就是每点击一次checkbox，getNum()会计算一次当前那些按钮被按下，并将数值累加，然后返回数值。

通过checkclicked()槽函数的第二段代码将数字显示到statusBar任务栏上。当然需要通过str类型转换将数值转换成文本。