**PyQt5学习笔记**

**布局管理器布局**

1.minimumSize 和maximumSize 属性用来设置控件在布局管理器中的最小和最大尺寸。

2.sizePolicy:通过修改窗口部件的sizePolicy，对布局进行微调

3.sizeHint：尺寸提示，是窗口控件的期望尺寸

4.minimumSize：最小尺寸，窗口控件压缩时能被压缩到的最小尺寸

**设置伙伴关系**

将label标签前加‘&’，然后点击菜单‘Edit’→‘编辑伙伴’拖到要设置伙伴关系的部件上  
**注意：只对英文开头的Display Widgets有效**

**设置tab键次序**

tab键次序作用是按tab键时光标的移动顺序。  
在edit里设置。

**Qt Designer的使用**

**一、加载资源文件**

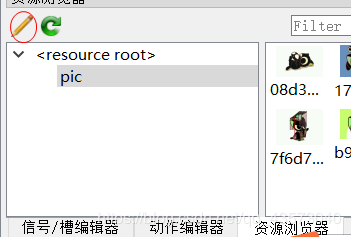
新建一个资源文件xxxxxx.qrc 内容如下：

<rcc version="1.0">

<qresource>

</qresource>

</rcc>

打开qt designer 新建类型为widget的简单窗口，打开资源浏览器  
  
然后添加图片就好了

**二、在窗体中使用资源文件**

加入label控件 → 找到pixmap属性 → 找到图片路径  
→将ui文件转换成py文件。  
py文件如下：

from PyQt5 import QtCore, QtGui, QtWidgets

class Ui\_Form(object):

def setupUi(self, Form):

Form.setObjectName("Form")

Form.resize(883, 614)

self.label = QtWidgets.QLabel(Form)

self.label.setGeometry(QtCore.QRect(140, 100, 591, 391))

self.label.setText("")

self.label.setPixmap(QtGui.QPixmap(":/pic/images/17a515a85edf8db1ae6b8e900e23dd54544e74c6.jpg"))

self.label.setObjectName("label")

self.retranslateUi(Form)

QtCore.QMetaObject.connectSlotsByName(Form)

def retranslateUi(self, Form):

\_translate = QtCore.QCoreApplication.translate

Form.setWindowTitle(\_translate("Form", "Form"))

import apprcc

注意最后一行是加载资源的文件名

转换资源文件将qrc文件转换为py文件

之后，为了将窗口显示和业务逻辑分离，再新建一个调用窗口显示的文件，代码如下：

import sys

from PyQt5.QtWidgets import QApplication,QMainWindow

from mainwin02 import Ui\_Form

class mymainwindow(QMainWindow,Ui\_Form):

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

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

self.setupUi(self)

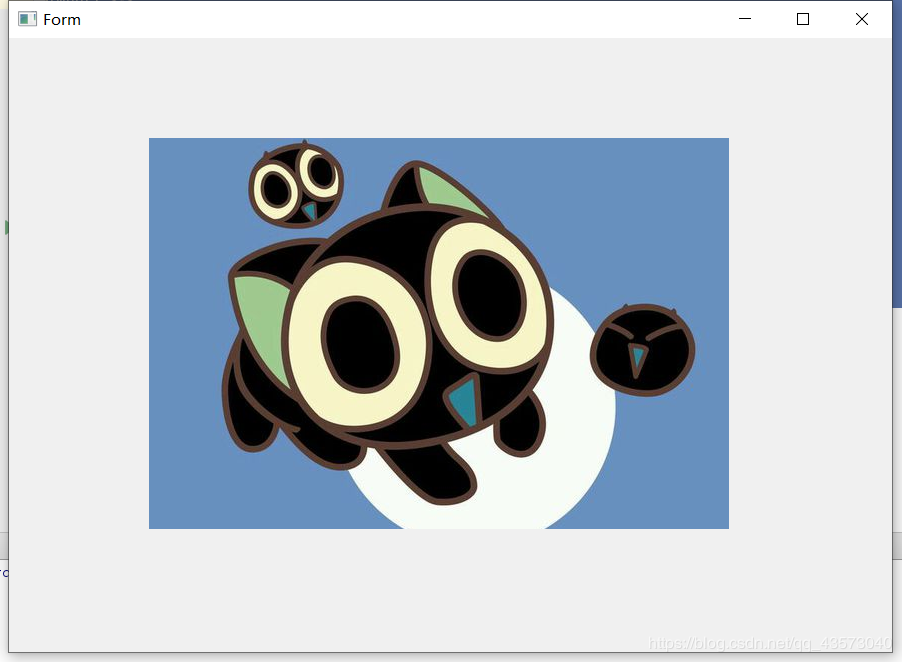
if \_\_name\_\_=="\_\_main\_\_":

app = QApplication(sys.argv)

mywin = mymainwindow()

mywin.show()

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



**信号与槽**

信号与槽的关系：  
发送者→（信号）→连接→（接收）→接受者→（调用）→槽函数

**（1）定义信号**

from PyQt5.QtCore import pyqtSignal

from PyQt5.QtWidgets import QMainWindow

class WinForm(QMainWindow):

btnClickedSignal = pyqtSignal()

**操作信号**

connect()函数可以把信号绑定到槽函数上。  
disconnect()函数可以解除信号与槽函数的绑定。  
emit()函数可以发射信号。

**入门应用**

信号与槽链接的主要步骤如下：  
（1）生成一个信号。  
（2）将信号与槽函数绑定。  
（3）连接信号与槽函数。  
（4）发射信号。  
（5）将信号绑定到槽对象中的槽函数get()上

app = QApplication([])

widget = QWidget()

def showmsg()：

QMessageBox.information(widget,"信息提示框","ok,弹出测试信息")

btn = QPushButton("测试点击按钮",widget)

btn.cilcked.connect(showMsg)//绑定

widget.show()

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

**按钮类控件**

**QAbstractButton**

**状态**

isDown():提示按钮是否被按下；  
isChecked():提示按钮是否已经标记；  
isEnable():提示按钮是否可以被用户点击；  
isCheckAble():提示按钮是否为可标记的；  
setAutoRepeat():提示按钮是否在用户长按时可以自动重复执行；

**信号**

Pressed:当鼠标指针在按钮上并按下左键时触发该信号；  
Released:当鼠标左键释放时触发该信号；  
Cllicked:当鼠标左键被按下然后释放时，或者快捷键被释放时触发该信号；  
Toggled:当按钮的标记状态发生改变时触发该信号

**方法**

setCheckable():设置按钮是否已经被选中，如果设置为true,则表示按钮将保持已点击和释放状态  
toggle()：在按钮状态之间进行切换；  
setIcon()：设置按钮上的图标  
setEnabled():设置按钮是否可以使用，当设置为False时，按钮变成不可用状态，点击它不会发射信号  
isChecked():返回按钮的状态。True or False  
setDefault():设置按钮的默认状态  
setText():设置按钮的显示文本；  
text():返回按钮的显示文本；

**QRadioButton**

**方法**

setCheckable():设置按钮是否已被选中，可以改变单选钮的选中状态  
isCheck():返回单选钮的状态。  
setText():设置单选钮的显示文本  
text():返回单选钮的显示文本

**QCheckBox**

**方法**

setChecked():设置复选框的状态。  
setText():设置复选框的显示文本。  
text():返回单选钮的显示文本  
isChecked():检查复选框是否被选中。  
setTriState():设置复选框为一个三态复选框。

**QComboBox**

**方法**

addItem():添加一个下拉选项。  
addItems():从列表中添加下拉选项。  
Clear():删除下拉选项集合中的所有选项。  
count():返回下拉选项集合中的数目。  
currentText():返回选中选项的文本  
itemText(i):获取索引为i的item的选项文本。  
currentIndex():返回选中项的索引。

**菜单栏**

**方法**

menuBar()：返回主窗口的QMenuBar  
addMenu():在菜单栏中添加一个新的Qmenu对象  
addAction():向QMenu小控件中添加一个操作按钮，其中包含文本或图标  
setEnabled():将操作按钮状态设置为启用和禁用。  
addSeperator():在菜单中添加一条分隔线  
clear():删除菜单/菜单栏的内容；  
setShortcut():将快捷键关联到操作按钮  
setText():设置菜单项的文本；  
setTitle():设置QMenu小控件的标题；  
text():返回与QAction对象关联的文本；  
title():返回QMenu()小控件的标题；

**QToolBar**

**方法**

addAction():添加具有文本或图标的工具按钮  
addSeperator():分组显示工具按钮；  
addWidget():添加工具栏中按钮以外的控件  
addToolBar():使用QMainWindow类的方法添加一个新的工具栏  
setMovable():工具栏变得可移动  
setOrientation():工具栏的方向可以设置为Qt.Horizontal或Qt.vertical