

本文档系重庆大学计算机学院 Python课程的教学笔记。

其版权属于 海洋饼干叔叔 chenbo@cqu.edu.cn

仅供学习者个人研读，未经许可，不允许在互联网上提供下载。

16. 实践 - 冒泡及轻者上浮

在本书的前述章节中，我们已经见识过了Python自带的标准GUI工具包Tinkter。但Tinkter的功能相对比较简单，界面也不够漂亮，对于规模大一点的GUI应用略显不足。在当前的Python生态圈，如果读者需要学会一种功能较齐全、能满足大多数项目实践需要的GUI工具包，那么，作者认为，PyQt5是当前最好的选择。

本章的目的是向读者介绍基于PyQt5的图形应用程序的框架及开发方法，同时理解分时操作系统内消息循环机制，并了解多线程程序设计的基本概念和方法。冒泡排序不是重点。

16.1 开发环境准备

16.1.1 Qt

Qt - <https://www.qt.io/>是久负盛名的跨平台C++ GUI开发包及集成开发环境。它都过独特的信号-槽机制屏蔽了不同操作系统间的差异，使得用C++语言书写的应用程序可以在不同的操作系统(Windows, Linux如Ubuntu)下运行。由于它本身就是C++语言书写的，所以运行速度相对较快。经过多年的发展，Qt已经发展得比较成熟和系统，即便开发Android App，也可以在Qt上用C++完成。

PyQt则是对Qt的Python封装库，它是由英国的RiverBank公司- <https://riverbankcomputing.com>开发的。另外还有一款名为Eric的IDE软件，它把Python, PyQt集成得非常好，作者曾在树莓派卡片电脑上直接用Eric开发基于PyQt的应用程序。

需要注意的是，Qt执行LGPL授权协议而PyQt执行 GPL授权协议。如果读者试图在一个私有代码的商业软件中应用上述组件，可能需要付费。

16.1.2 PyQt安装

进入Windows命令行或者Linux的终端，通过pip工具安装pyqt5以及pyqt5-tools两个包。安装需要联网，并执续好几分钟，因为被安装的包是从网络软件仓库中实时下载的。

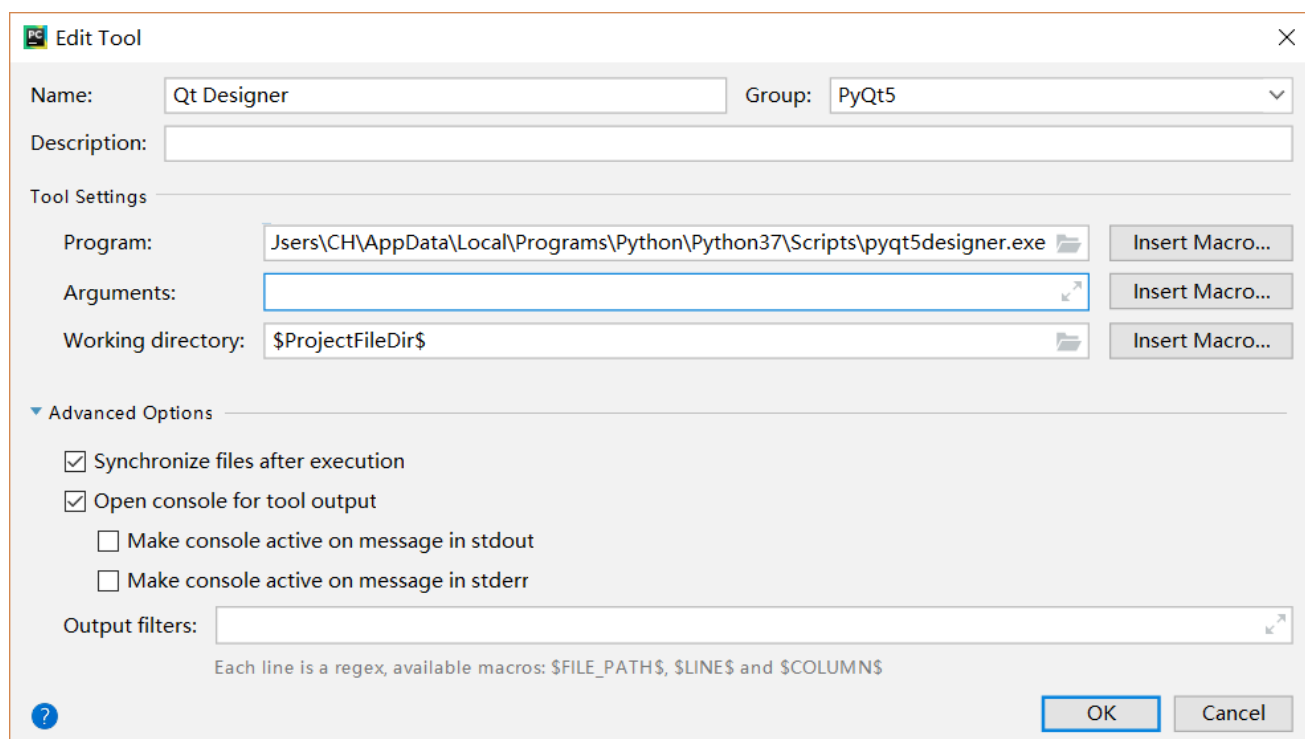
```
pip install pyqt5
pip install pyqt5-tools
```

16.1.3 PyCharm配置

Qt/PyQt中包括一系列的工具有，其中：

工具名称	用途	可执行文件名称
Qt Designer	用即见即所得的方式设计图形界面，成果表现为扩展名为ui的文件。	designer
UI Compiler	将上述ui文件“编译”成Python程序。执行该Python程序便可以得ui文件所描述的图形界面。	pyuic5
Qt Linguist	语言学家，可以便捷的实现软件的国际化，即生成软件的法语、英语、日语或者其它语种版本。工作模式大致可以描述成：先用pylupdate5扫描源代码中全部可翻译的字符串，然后用linguist翻译相应的字符串至目标语言，接下来用lrelease工具发布。软件运行时，加载法语版本的语言学家文件，软件界面就是法语，加载日语版本的语言学家文件，软件界面就是日语。	linguist, pylupdate5, lrelease
Resource Compiler	资源编译器。UI文件设计过程中可以需要使用到各种图片，这些图片以资源文件的形式组织，扩展名为qrc；资源编译器负责将qrc格式的资源文件编译成py文件，其中，图片被转换成bytes字节流。	pyrcc5

在本实践上，我们只用到其中三个。PyCharm->File->Settings里可以设置External Tools，如下。



注意，\$ProjectFileDir\$表明该工具运行时，其默认工作目录为当前PyCharm的项目文件目录。这种由两个\$符号包起来的特定标识在此处称为宏-Macro，通过编辑框后面的Insert Macro按钮可以插入。

Program编辑框中的程序位置根据Python解释器的安装位置不同可能会有差异。

PC Edit Tool

Name: UI Compiler Group: PyQt5

Description:

Tool Settings

Program: C:\Users\CH\AppData\Local\Programs\Python\Python37\Scripts\pyuic5.exe Insert Macro...

Arguments: \$FileName\$ -o Ui_\$FileNameWithoutExtension\$.py Insert Macro...

Working directory: \$ProjectFileDir\$ Insert Macro...

▼ Advanced Options

☒ Synchronize files after execution

☒ Open console for tool output

☐ Make console active on message in stdout

☐ Make console active on message in stderr

Output filters:

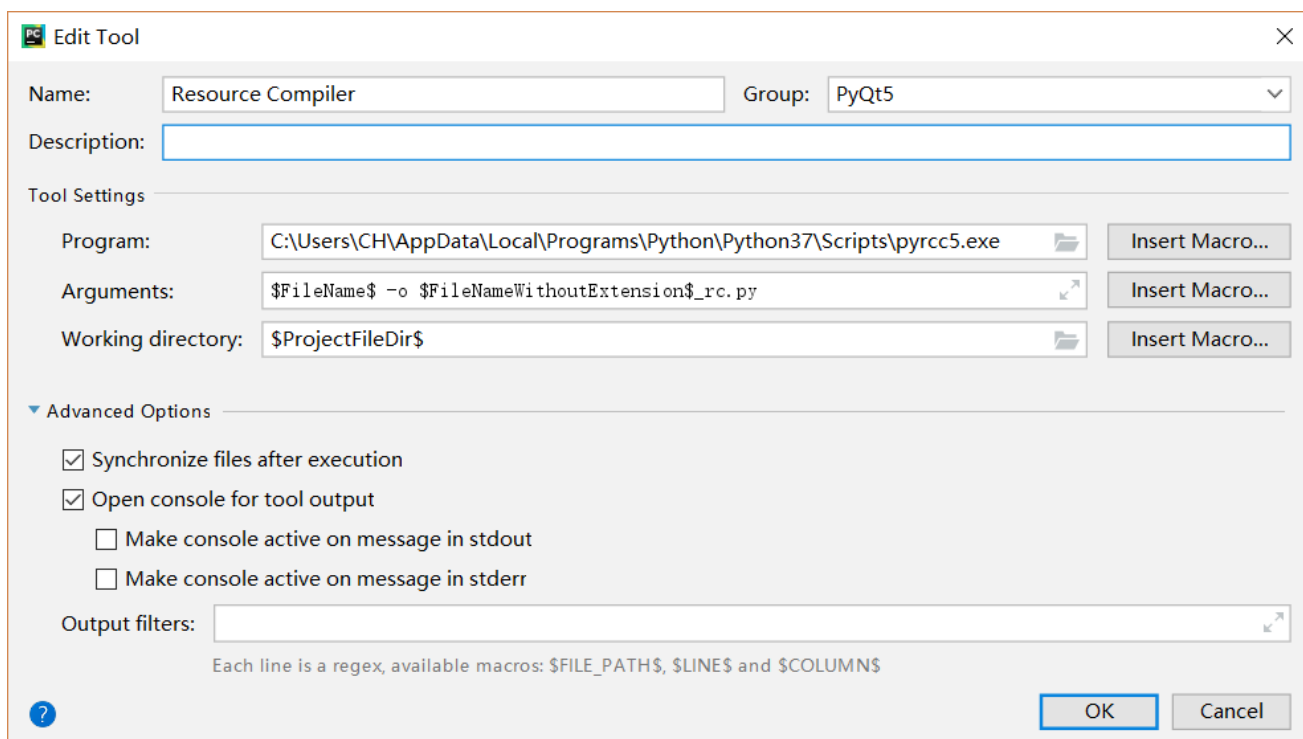
Each line is a regex, available macros: \$FILE_PATH\$, \$LINE\$ and \$COLUMN\$

? OK Cancel

注意，上述Arguments项里的宏\$FileName\$表示PyCharm里当前选中的文件的名称。这意味着，当你试图运行UI Compiler之前，应该先在PyCharm里选中那个文件，比如本例中的MainWidget.ui。以MainWidget.ui为例，\$FileNameWithoutExtension\$是指不包含扩展名的文件名，即MainWidget。综合起来，在本实践中，如果你选中MainWidget.ui，然后执行UI Compiler，最终发送给控制台的命令行即为：

```
C:\...\Python37\Scripts\pyuic5.exe MainWidget.ui -o Ui_MainWidget.py
```

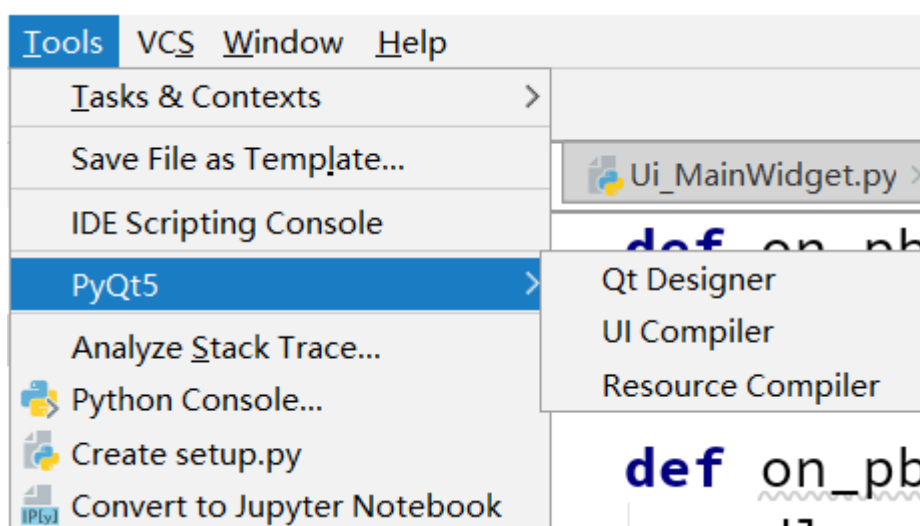
上述命令行用大白话讲，就是运行UI Compiler，编译MainWidget.ui，生成结果存入文件Ui_MainWidget.py。事实上，如果你不把上述工具集成在PyCharm中，而是直接在操作系统命令行中执行上述命令，也可以达到同样的目的。



Resource Compiler的运作方式与上述UI Compiler类似，应先在PyCharm中选中需要编译的资源文件，以Images.qrc为例，然后再执行菜单中的UI Compiler，最终生成的命令行为：

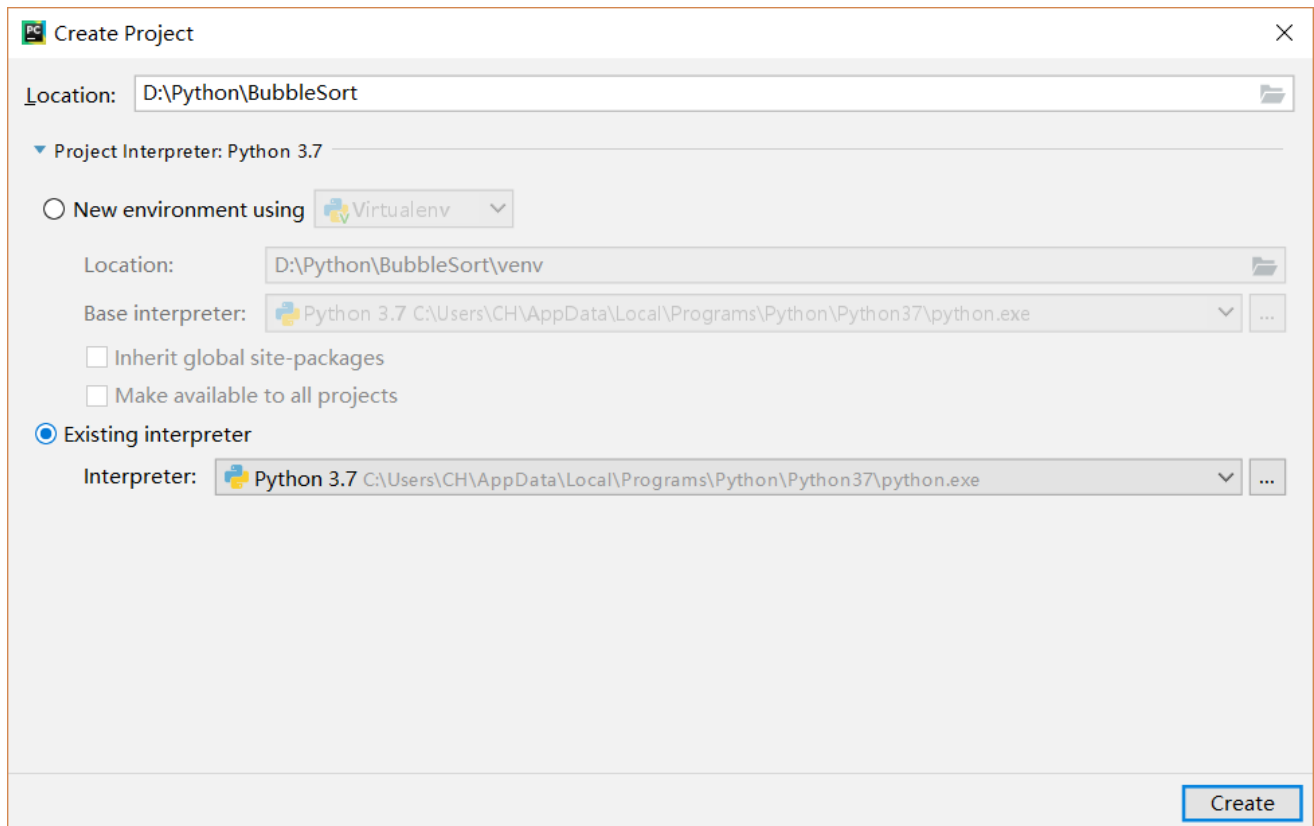
```
C:\...\Python37\Scripts\pyrcc5.exe Images.qrc -o Images_rc.py
```

上述配置完成后，可以在PyCharm的Tools菜单中看到如下内容：



16.2 简单PyQt图形应用

16.2.1 创建PyQt应用



在PyCharm里创建一个新应用，注意Project Interpreter最好使用系统全局的解释器，对于作者的电脑而言，就是C:\Users\CH\AppData\Local\Programs\Python\Python37\python.exe，如上图。

在BubbleSort项目中创建一个项目主文件，名为BubbleSort.py，内容如下：

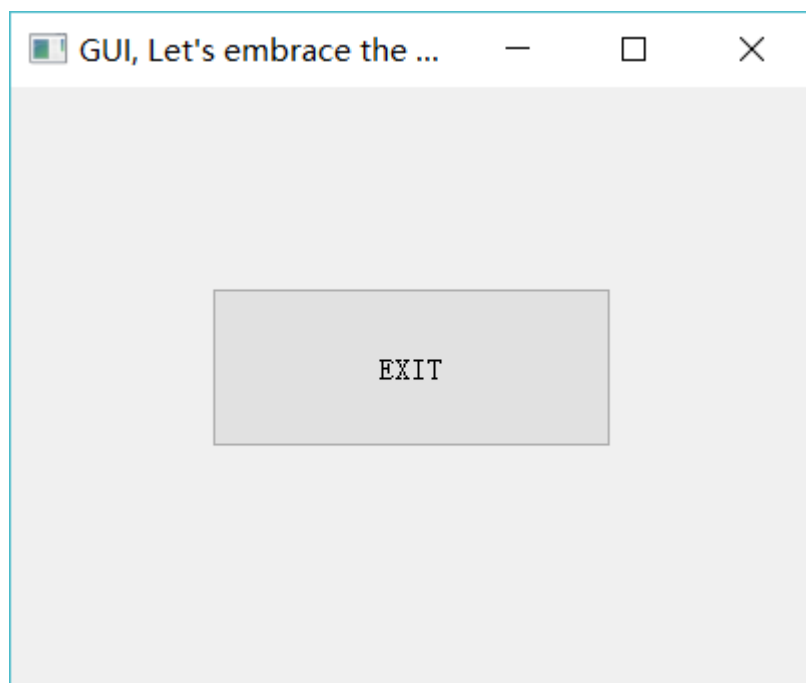
```
import sys
from PyQt5 import QtWidgets,QtCore

app = QtWidgets.QApplication(sys.argv)
wdMain = QtWidgets.QWidget()
wdMain.setGeometry(200,200,800,600)
wdMain.setWindowTitle("GUI, Let's embrace the world!")

btnExit = QtWidgets.QPushButton('EXIT',wdMain)
btnExit.resize(200,80)
btnExit.move(300,300)
btnExit.clicked.connect(QtCore.QCoreApplication.quit)

wdMain.show()
r = app.exec_()
print("message loop ended.")
exit(r)
```

执行，得到第一个Qt图形应用的运行界面，用鼠标点一下EXIT按钮，程序运行结束。



16.2.2 分时系统与消息循环

上面这个BubbleSort.py行数并不多，但要彻底理解它的工作原因却并不容易。我们得从操作系统说起。现代操作系统都是分时系统，你的计算机同时在做很多事情：浏览器、Word、音乐播放器... 读者如果打开Windows的任务管理器，可以同时看到数十至数百个进程 - process，而一个进程，可能又是由多个线程 - thread组成的。比如，当你的浏览器进程试图从网站上下一个大文件时，它可能会创建一个单独的线程来下载文件，而原有的主线程则随时待命，及时处理你的命令：输入网址，点下超链接等等。

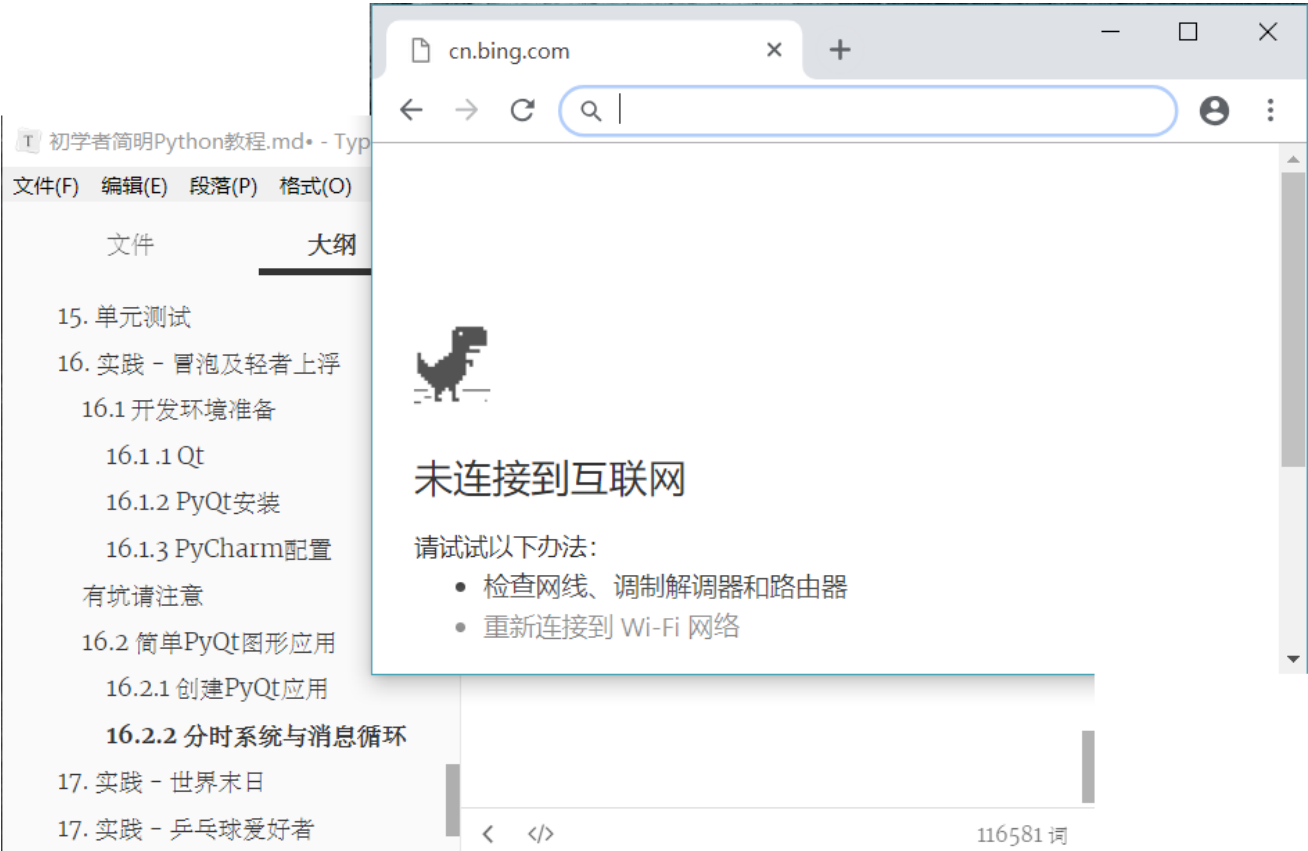
所有我们达成了第1个共识：线程竞争性地使用CPU资源。

操作系统管理着CPU，将CPU的时间切割成非常小的时间片。它按照效率与公平兼顾的原则将时间片分配线程，线程获得时间片后，将执行相应的运算或其它操作。时间片用完后，操作系统会收回CPU，将时间片分给其它线程；上述时间片的分配和回收对于应用程序而言是透明的，也就是应用程序根本不知道也无法预测或者控制时间片的获得与丧失。当一个线程被剥夺时间片时，操作系统会保存好执行现场，包括CPU内各个寄存器的值，然后线程就挂起 - suspended。当这个被挂起的线程重新获得时间片时，操作系统会先恢复执行现场，然后通过跳转指令恢复线程的执行。由于时间片的轮转速度非常快，所以，使用者一般感觉不到应用程序这种间断执行，就好像应用程序拥有一个单独的CPU一样。

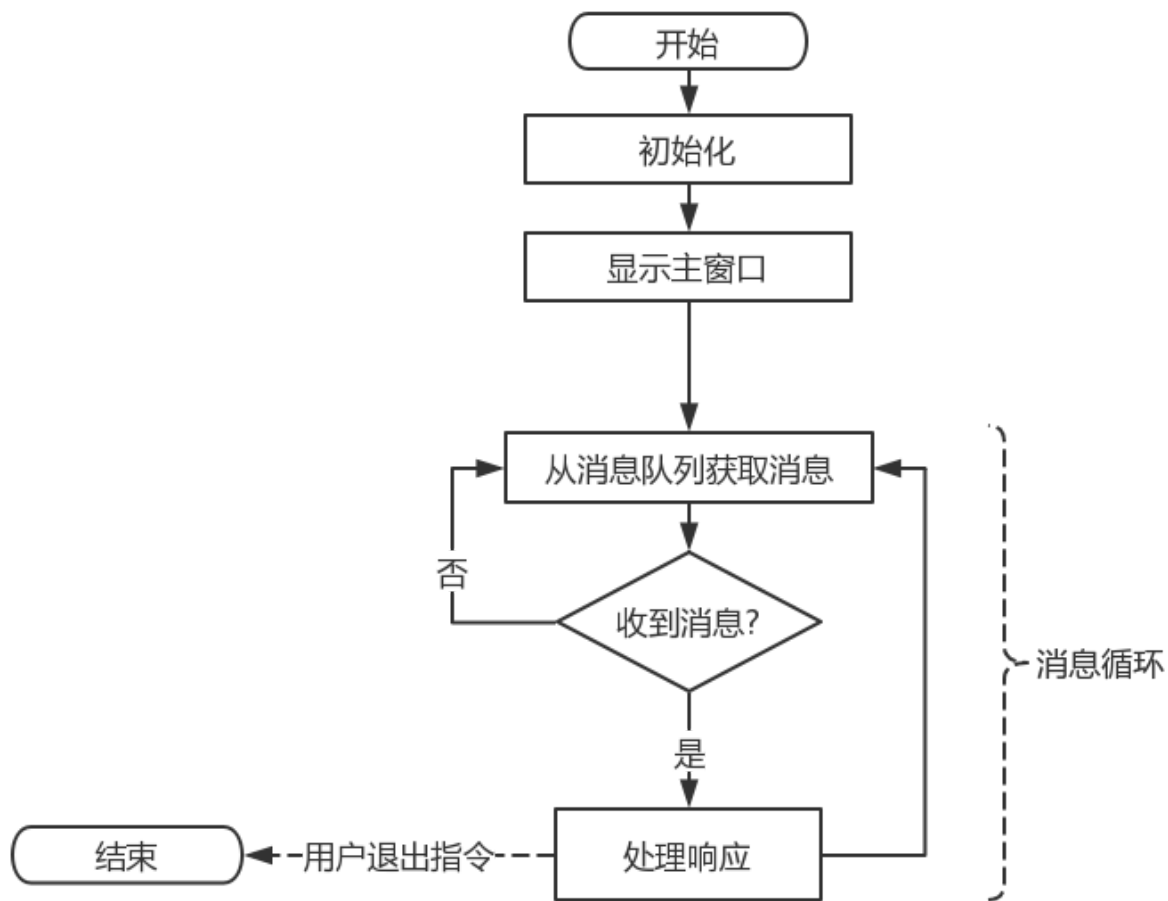
计算机通常只有一个键盘、一个鼠标。所以我们不能认为键盘和鼠标是属于WORD的，还是浏览器的。这些应用程序在共同这些外部设备。我们不能认为键盘是属于WORD的，因为浏览器也要利用键盘的输入。那么，只有一种可能：操作系统管理这些外设资源，既包括输入设备如键盘鼠标，也包括输出设备，比如显示器/显卡。

现在有了第2个共识：操作系统负责管理键盘鼠标并将键盘鼠标的输入分发给对应的进程。

如果桌面上同时有两个窗口，如下图。这里操作者如果敲下一个键，比如c，那么首先获悉这个事件的，肯定是操作系统，因为操作系统监视着键盘输入。问题是，当操作系统获悉这个事件后，将这个事件分发给哪个应用程序呢？是前面的浏览器还是后面的文字编辑器呢？显然，操作系统遵循谁有焦点-focus，就分发给谁的规则。事实上，所有的应用程序，它的窗口大小、窗口位置等信息都是向操作系统登记备案的，依据这些信息，操作系统决定信息的去向。为了更好的分发这些消息，操作系统会为每一个进程创建消息队列，凡是有发往这个进程的消息，操作系统就把这个消息放在对应的队列里。而应用程序的进程，则不断地从队列获取消息并处理，作出恰当的反应。应用程序不断读取消息队列并处理消息的机制称为消息循环。



总结，在分时操作系统下，一个图形应用程序的执行框架可以大致用下图刻画。



可以看到，图形应用程序启动后，在完成初始化，向操作系统注册，显示主窗口等任务后，即进入一个消息循环：周而复始的从操作系统的消息队列中获取分发给自己的消息并进行处理。比如，用户按了某个按钮，应用程序在收到这个消息后将执行对应的处理函数，以响应用户的要求。如果用户按下的是主窗口的关闭按钮（即窗口右上角的X），应用程序在收到这个消息后通常会退出消息循环，执行结束。

16.2.3 示例解读

现在可以尝试解释本节的PyQt图形应用的代码了。

```
from PyQt5 import QtWidgets, QtCore
```

PyQt的包，我们主要用到三个，分别是：

包/模块名	说明
QtWidgets	包括一系列GUI部件，比如QMainWindow、QDialog等。
QtGui	包括窗口集成、事件处理、2D绘图、字体和文本等GUI元素。

包/模块名	说明
QtCore	包括时间、文件及目录处理、数据类型、数据流、进程/线程等功能，属于非GUI核心模块。

除此之外，还有一些模块：**QtNetwork**用于网络通信；**QtSql**用于关系数据库访问；**QtWebKit**则是支持内置的网络浏览器；**QtMultimedia**则用于支持多媒体。

```
app = QtWidgets.QApplication(sys.argv)
...
r = app.exec_()
print("message loop ended.")
exit(r)
```

所有的Qt图形应用程序都需要创建一个QtWidgets.QApplication对象，这个对象将负责进程的消息循环和分发。**sys.argv**是程序启动时的命令行参数。**app.exec_()**函数的实质就是应用程序的消息循环，它周而复始地从操作系统消息队列中获取用户消息/指令，然后把这些消息/指令按照Qt特有的信号-槽 (**signal - slot**)机制分发给对应的处理函数进行处理，并做作适当响应。通常，当应用程序的主窗口被关闭后，**app.exec_()**函数将退出消息循环，并返回一个值表明应用的执行结果，这个值通常表明程序正确完成或者失败退出。

读者可以再试运行这个简单的只有一个按钮的图形应用，请注意，只有当你点击EXIT按钮，主窗口关闭后，上述**print("message loop ended.")**消息才会输出到控制台。这证明，**app.exec_()**函数真的在进行消息死循环，它将程序“卡”在这里。

app.exec_()执行结束后，**exit()**函数退出Python解释器，参数**r**被返回给操作系统表明程序运行结果。

```
wdMain = QtWidgets.QWidget()
wdMain.setGeometry(200,200,800,600)
wdMain.setWindowTitle("GUI, Let's embrace the world!")

btnExit = QtWidgets.QPushButton('EXIT',wdMain)
btnExit.resize(200,80)
btnExit.move(300,300)
btnExit.clicked.connect(QtCore.QCoreApplication.quit)

wdMain.show()
```

中间这段代码先是创建了一个QtWidgets.QWidget对象作为应用程序的主窗口。**Widget**这个词大致就是窗口-**Window**的另一种写法。**setGeometry()**方法显然设置了这个窗口在桌面上的呈现位置（左上角坐标）和像素单位长宽尺寸。**setWindowTitle()**则设置了窗口的标题。大多数情况下，Qt的类名、函数都具有良好自解释特性，看到名字大概就是猜出其功能。

接下来，btnExit是一个QtWidgets.QPushButton对象，就是一个按钮控件。

QtWidgets.QPushButton('EXIT',wdMain)的第一个参数将是这个按钮的标题，而参数wdMain表明了这个按钮的父控件，也就是按钮的拥有者是wdMain主窗口。resize()函数设定了按钮的尺寸，move()函数将按钮移动到窗口内部坐标(300,300)的位置。请注意，在GUI应用中，坐标系通常是top-left坐标系，以窗口的左上角为原点，向右x为正，向下y为正。

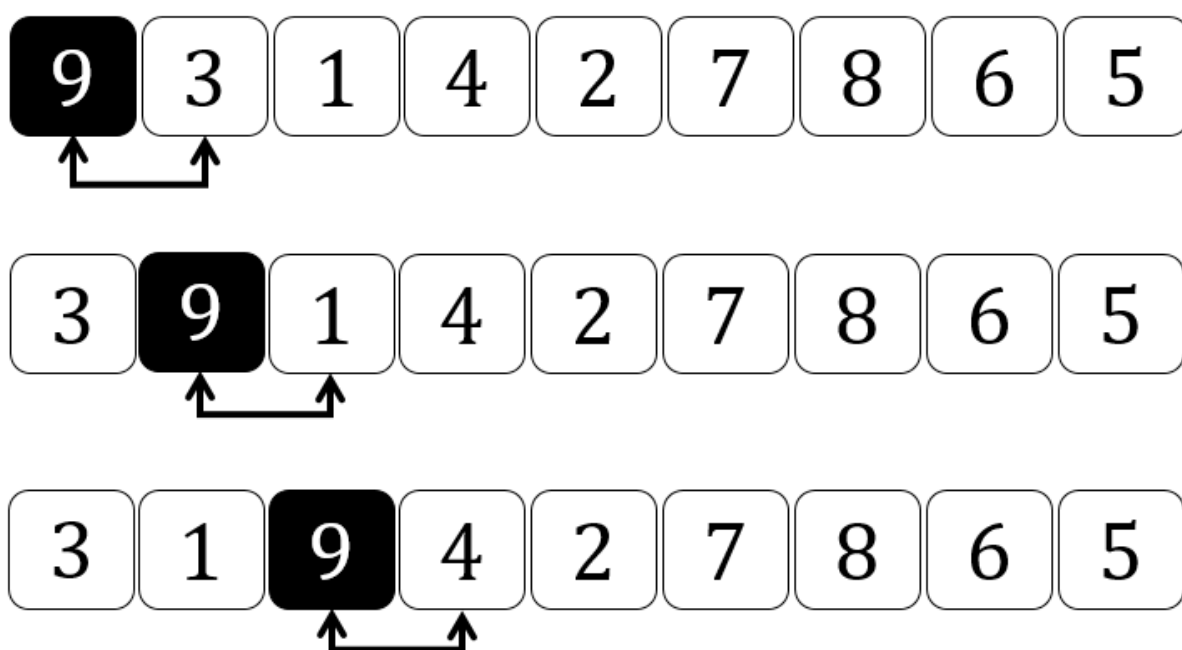
btnExit.clicked.connect(QtCore.QCoreApplication.quit)这一行最为关键。clicked为btnExit对象的属性，它是一个信号-signal，而QtCore.QCoreApplication.quit可以认为是一个特殊处理函数，称之为槽-slot，它的功能大致是导致应用程序结束运行。clicked.connect()函数则将信号clicked与槽关联起来，结果就是：当主窗口wdMain内的btnExit按钮被点击时，操作系统监控到鼠标的动作，然后将这一事件打包成一个消息，放至该应用程序的消息队列；app.exec_()内部的消息循环得到这一消息后，在内部将其处理成btnExit的clicked信号，根据信号-槽的关联，QtCore.QCoreApplication.quit槽方法被执行，跳出消息循环，程序结束。

上述代码只是"徒手"创建了wdMain窗口，而wdMain.show()函数的执行才真正将其显示出来。接下来就是app.exec_()的消息循环。

16.3 冒泡-轻者上浮

当水里有气泡时，气泡的密度比水小，流体浮力大于其重力，上浮。而水面的石头则注定下沉。冒泡排序-bubble sort即得名于此。请注意，这里的所谓浮、沉、轻和重只是形象化的描述，不要机械理解。

如下图中的9个整数所构成的序列。现在我们要进行非递减，也就是递增排序。我们从前(下标0)往后两两比较，如果左边的元素大于右边的元素，则两两交换。如下例，9与3比较， $9 > 3$ ，交换；9与1比较， $9 > 1$ ，交换；9与4比较， $9 > 4$ ，交换...最后9与5比较， $9 > 5$ ，交换。经过这一轮共8次的两两比较以及次数不确定的交换，我们发现，序列中最大的那个元素9到了序列的尾部，即下标8的位置。而这个位置正好是元素9在排序后的目标位置。



经过一轮冒泡，9来到了序列最右端的目标位置：



此时，我们可以认为9个元素中的前8个是无序的，而最后一个元素无需参与后续的排序。所以我们将前8个元素构成的子序列再来一轮从左至右的冒泡。在这一轮里，我们总共进行7次两两比较。最后，子序列中最大的元素8也到达了目标位置，即子序列的最后端。



同样，我们可以把前7个元素构成的子序列再来一轮冒泡；... 把前2个元素构成的子序列再来一轮冒泡，最后，全部元素都达到了目标位置，排序完成，如下图：



总结：对于由n个元素组成的序列，如果进行冒泡排序，总共要进行n-1轮的冒泡。第一轮冒泡执行n-1次比较，第二轮冒泡执行n-2次比较，... 最后一次冒泡执行1次比较。故，比较运算的总次数为：

$n-1 + n-2 + n-3 + \dots + 2 + 1$ ，这是一个等差数列，其和约等于 $n(n-1)/2$ 。如果以后你们学习算法或者数据结构课程，会进一步地将该式简化成仅评估n的阶，用符号表达为 $\Theta(n^2)$ 。在算法分析的术语里，我们称冒泡排序的计算机复杂性为 $\Theta(n^2)$ 。

下面冒泡排序的核心代码。其中，外层循环中的i表示进行本轮冒泡子序列的最大元素下标，本例中，取值依次为8, 7, 6, 5, 4, 3, 2, 1（不含0）。内层循环中的j表示从左到右两两比较的左元素的下标，j+1当然是右元素的下标。本例中，当i=8时，(j, j+1)依次取(0,1),(1,2),(2,3),(3,4),(4,5),(5,6),(6,7),(7,8)。

当进行非递减排序时，if子句中使用>，否则使用<。

```
def bubbleSort(seq):
    for i in range(len(seq)-1, 0, -1):
        for j in range(0, i):
            if seq[j] > seq[j+1]:
                seq[j], seq[j+1] = seq[j+1], seq[j]

seq = [3, 9, 1, 4, 7, 6, 5, 8, 2]
bubbleSort(seq)
print("Sorted:", seq)
```

执行结果：


```
Sorted: [1, 2, 3, 4, 5, 6, 7, 8, 9]
```

如果读者还是不太明白，那么拿出一支笔，一张纸，把自己当成一台计算机，来一次上述程序的模拟执行，或许有用。
















16.4 世界主要工业国GDP排名

接下来，我们要编写一个图形应用程序，使用冒泡排序来对世界主要工业国的GDP进行排名，并演示冒泡排序的执行过程。由于有一系新的新工具，新方法，新问题，所以这并不容易。

在本书配套的网站上，你可以下载到本实践的全部代码和数据。在完成本章第一节的环境准备工作后，你应该可以打开并运行该实例。请对照代码阅读本章后续内容。该实例的最后运行结果大致如下图。



GDP OF MAIN INDUSTRIAL COUNTRIES (IN BILLIONS)

	\$19,555.87	United States
	\$1,759.27	Brazil
	\$1,318.83	Spain
	\$2,586.57	France
	\$1,251.25	Mexico
	\$2,607.41	India
	\$1,309.27	Russia
	\$1,932.94	Italy
	\$1,317.16	Austrilia
	\$3,232.28	United Kingdom
	\$1,682.37	Canada
	\$13,173.58	China
	\$1,545.81	Korea
	\$3,595.41	Germany
	\$4,342.16	Japan

START

STOP

SHUFFLE

ABOUT

EXIT

16.5 数据及基础结构

16.5.1 数据文件

名为BubbleSort的PyCharm项目目录内有一个名为countries.ini的文件，其中包括了2017年世界前15位的工业国家的GDP数据，下表列出了该文件的前几行。可见，基本数据包括国家名称，GDP值（以十亿-billion美元为单位），以及这个国家的国旗的图片文件名称。

```
[Countries]
countries.size = 15
countries[0].sName = United States
countries[0].fGdp = 19555.874
countries[0].sLogoFile = us.gif
countries[1].sName = China
countries[1].fGdp = 13173.585
countries[1].sLogoFile = china.gif
```

16.5.2 数据结构

```
from enum import Enum

class CompareState(Enum):
    prev = 0    #the item in comparation as prev item
    next = 1    #the item in comparation as next item
    idle = 2    #the item is not in comparation
    fixed = 3   #the item's position have been settled by sort algorithm

class Country:
    def __init__(self, name, gdp, logfile):
        self.sName = name
        self.fGdp = gdp
        self.sLogoFile = logfile
        self.compareState = CompareState.idle
```

在Country.py文件中，我们定义了新类Country，通过其构造函数，初始化四个属性，依次是国家名称-sName，GDP值-fGdp，国旗文件-sLogoFile以及比较状态-compareState。

其中，比较状态是个枚举型，其值将用界面展示（详见后）。idle表示当前这个国家对象不参与最近的冒泡比较；prev表示对象将作为左元素参与冒泡比较；next则表示对象将作为右元素参与冒泡比较；fixed表明对象在序列中的位置已经确定，不会再移动了，将来也不会再参与冒泡比较。

16.5.3 数据读入与组织

在MainWidget.py中，MainWidget的构造函数将调用initCountries()成员函数，该函数通过configparser包从"countries.ini"文件中读出数据，形成15个Country对象并置于self.countries列表当中。这些方法我们曾在本书的前述章节中讨论过。

```
def initCountries(self):
    self.countries = []

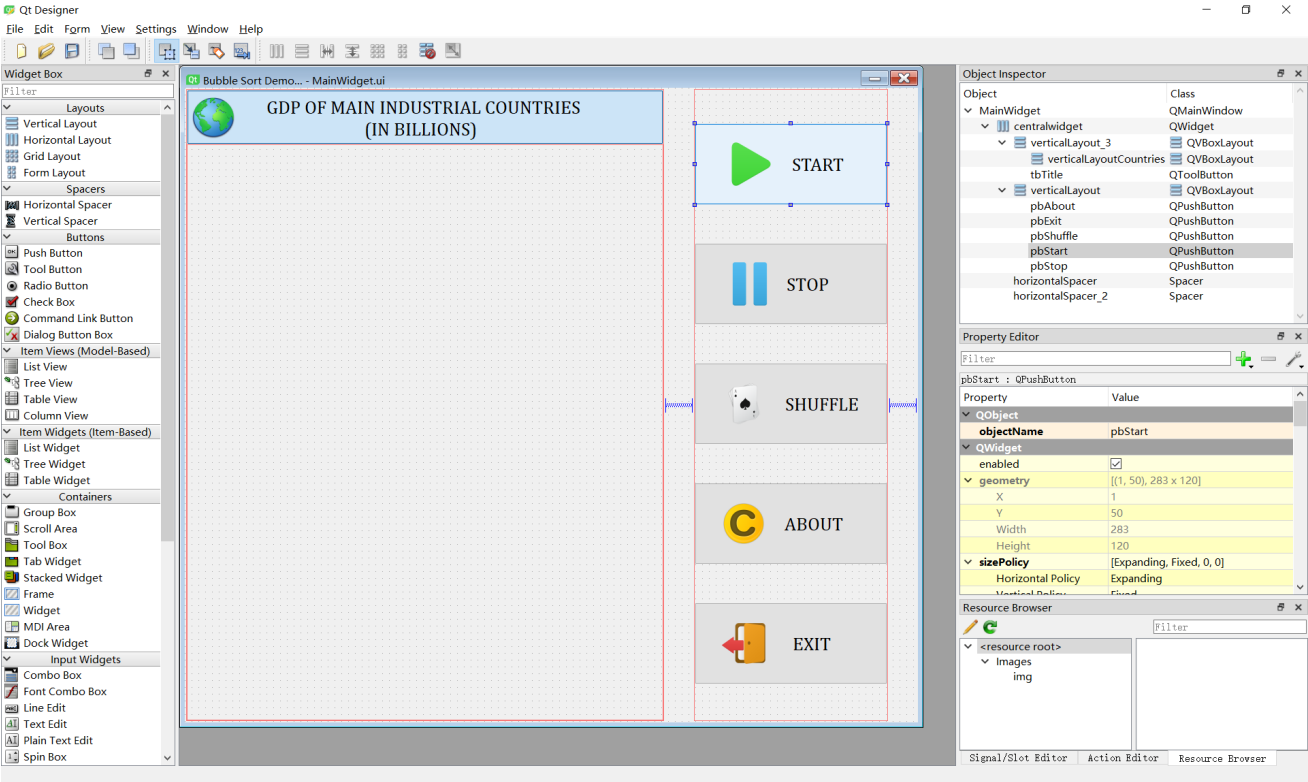
    import configparser
    data = configparser.ConfigParser()
    data.read("countries.ini")
    data = data["Countries"]
    iSize = int(data.get("countries.size", 0))
    for i in range(iSize):
        name = data.get("countries[{}].sName".format(i), "ERROR").strip()
        gdp = float(data.get("countries[{}].fGdp".format(i), "0"))
        logfile =
    data.get("countries[{}].sLogoFile".format(i), "ERROR").strip()
    self.countries.append(Country(name, gdp, logfile))
```

16.6 界面设计

在16.2节，我们“徒手”用代码创建了简单图形应用的主窗口，这样做没有问题。但更多情况下，图形界面的设计借助于那些所见即所得的设计工具将更加高效。Qt Designer就是这样一个设计工具。

16.6.1 基本操作

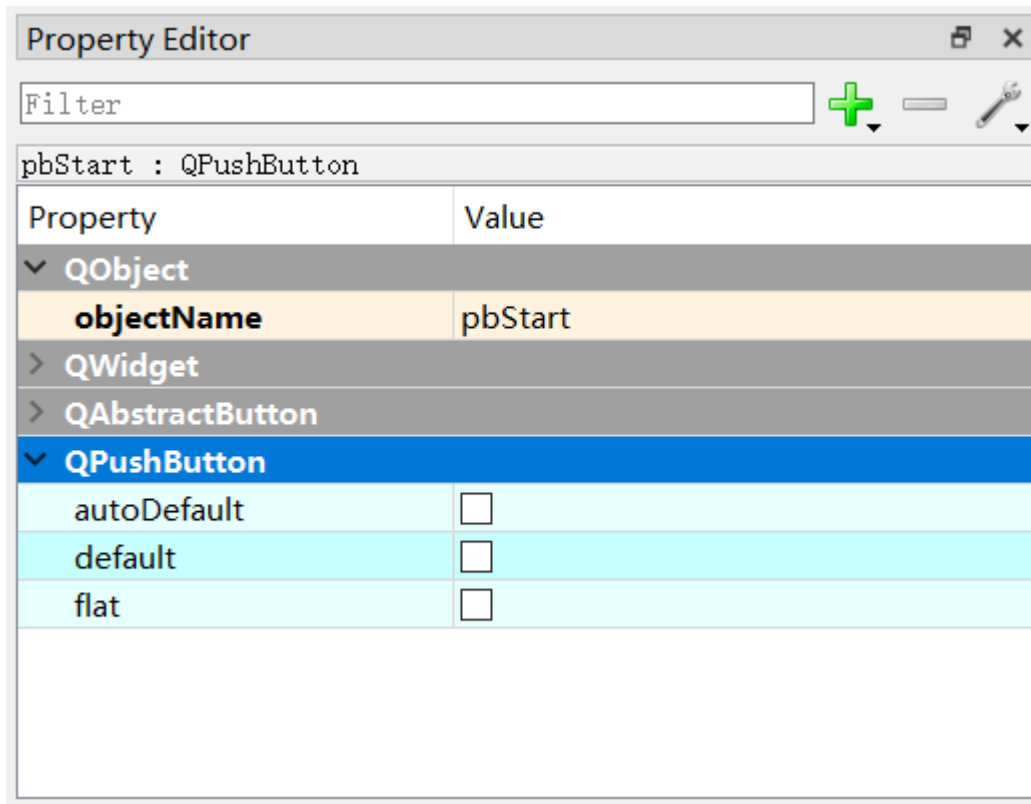
先用PyCharm打开BubbleSort项目-project，然后选择Tools->PyQt5->Qt Designer即可运行Qt Designer。在Qt Designer中打开MainWidget.ui。



这个界面的最左端，是Widget Box工具栏。这里包括了众多GUI组件，读者大致可以从名称和缩略图中猜出这些组件的功能：其中，Layouts属于布局类组件，它负责组织其中的下层GUI元素之间的相对位置关系；Buttons是各种操作按钮；Items Views以及Items Widgets是列表、树、及表格组件；Containers是容器类界面元素，它可以容纳下层组件；Input Widgets内的组件可以支持用户输入；Display Widgets内的组件负责在界面上显示信息。如果设计需要用到什么组件，左键单击该组件，按住不放，拖到中间界面内即可。

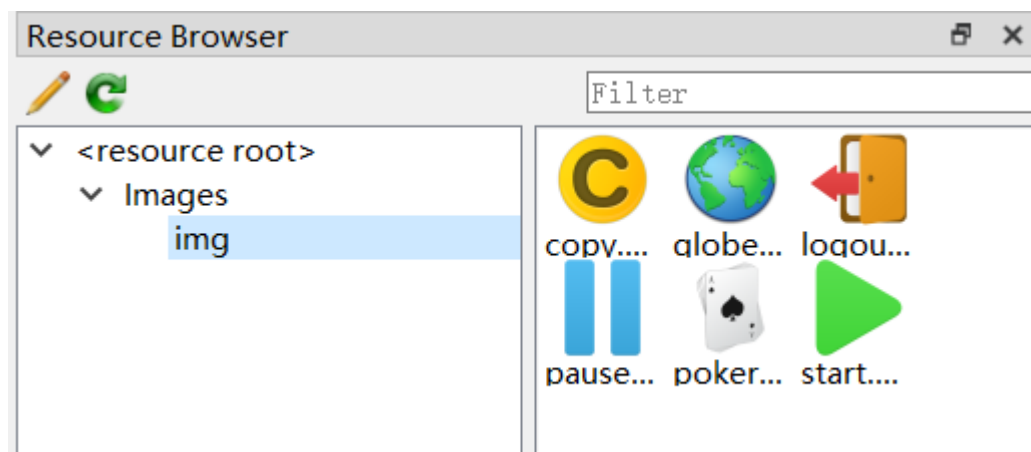
右上方的Object Inspector展示了当前窗口的树形结构。在Qt图形应用中，界面组件之间是存在父子从属关系的。比如，我们看到，名为pbStart的QPushButton按钮属于verticalLayout，而verticalLayout又从属于centralWidget，centralWidget又属于MainWindow，MainWindow的类型为QMainWindow。QMainWindow是Qt中的主窗口类，通常的Qt图形应用程序都应包括一个这样的主窗口。

当操作者选定一个界面组件时，可在右端中间的Property Editor中设置该界面组件的属性。当作者选中pbStart(START按钮)后，Property Editor如下图（部分属性被作者收起来了）。从这图中，我们可以看出QPushButton的继承关系：QObject -> QWidget -> QAbstractButton -> QPushButton。QPushButton的每一个父类型都会带来一些属性，其中QObject带来了objectName，这是这个组件的对象名称，当你的代码中操作或者引用这个组件时，就需要使用到这个名称。建议读者现在就展开pbStart的全部属性，从上到下看一遍，猜猜它们的用途。在本书中，组件的很多属性限于篇幅，无不一一介绍，读者需要自行查看和修改，通过观察变化来理解它们。



对于组件的名称取名，作者本人执行如下的铁律：当组件需要在代码中引用时，必须在UI设计阶段按照命令规则命名；如果组件不被代码引用，比如就是界面上的一个不需要修改显示内容的标签-Label，允许不改名，使用UI Designer为其取的默认名。读者也可以研究一下作者在本项目中的命名，以pbStart为例，pb为PushButton的首字母小写，Start表明其用途。

UI Designer的右下端为Resource Browser - 资源浏览器。下图中展示了本项目中使用的各按钮的配套LOGO图片。

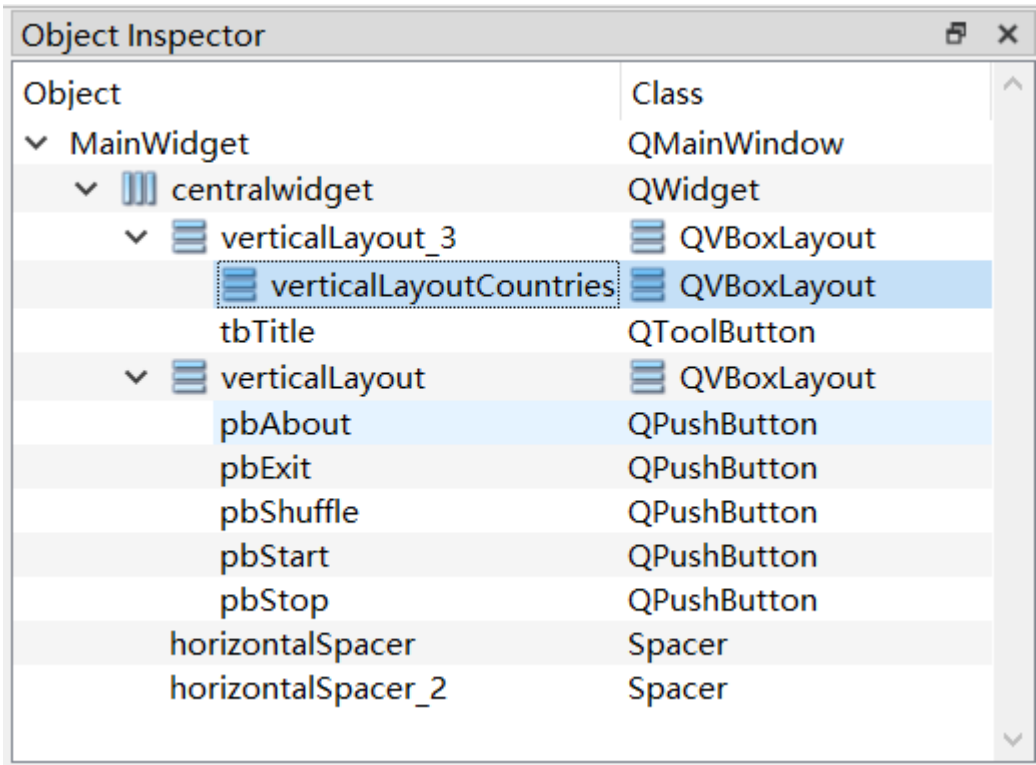


16.6.2 布局

现代计算机的形式多种多样，笔记本、台式机、手机、iPad，每种终端的屏幕比例，分辨率千差万别。所以，现代的界面设计，都要考虑应用界面在不同终端上的显示兼容。其中一个重要的工具就是布局。Qt里有4种布局组件：Vertical Layout把其下层界面组件按行布局，每行一个组件；Horizontal Layout把其下层组件按列布局，每列一个组件；Grid Layout把其下层组件按表格布局，

每个单元一个组件，当然，也同时允许组件跨多行或者多列；Form Layout称为表单布局，其表现形式跟Grid Layout有些相似。在实践中，读者可以通过不种布局之间的相互嵌套才实现复杂界面布局。

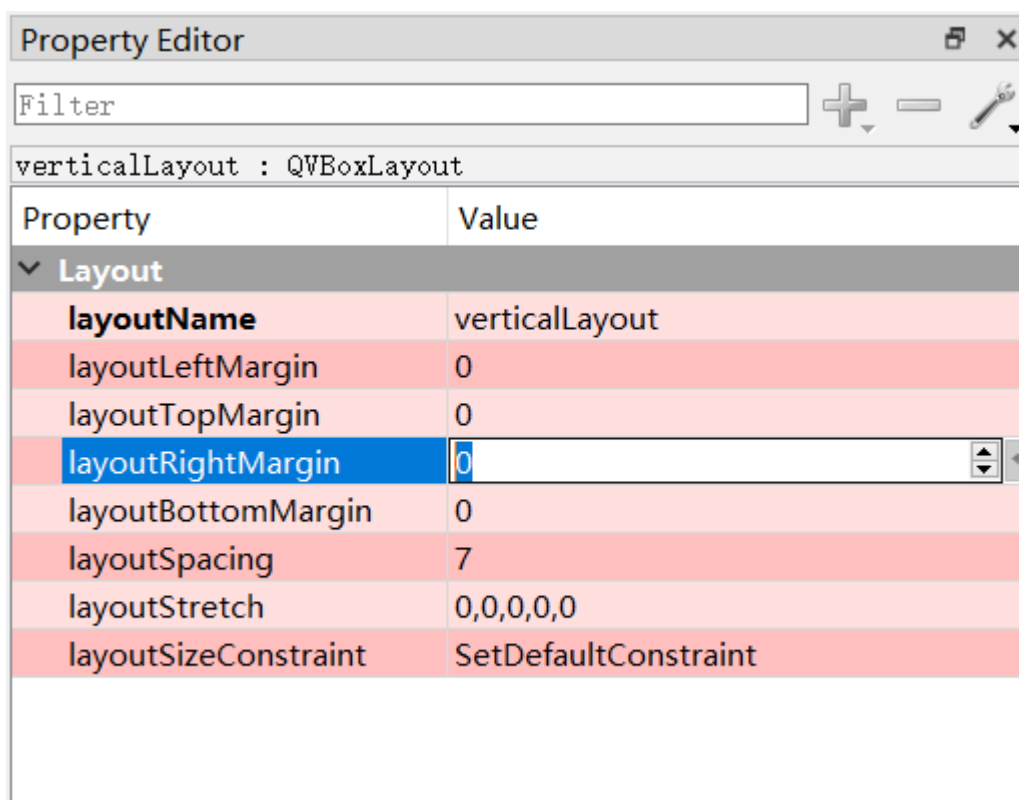
本例中的MainWidget.ui使用了下述布局结构：



可以看到，START-开始，SHUFFLE-打乱，STOP-停止，ABOUT-关于，EXIT-退出这5个QPushButton处于同一个verticalLayout(类型为QVBoxLayout)中，它们从上至下等间距显示。

还可以注意到上述结构中有一个verticalLayoutCountries，它是一个垂直布局。这是全部布局中唯一一个取了正式名称的，这是因为，作者要在代码中引用它，并把参与排序的全部国家信息放置在该布局中展示。

点取verticalLayout，可以在Property Editor中修改布局属性：



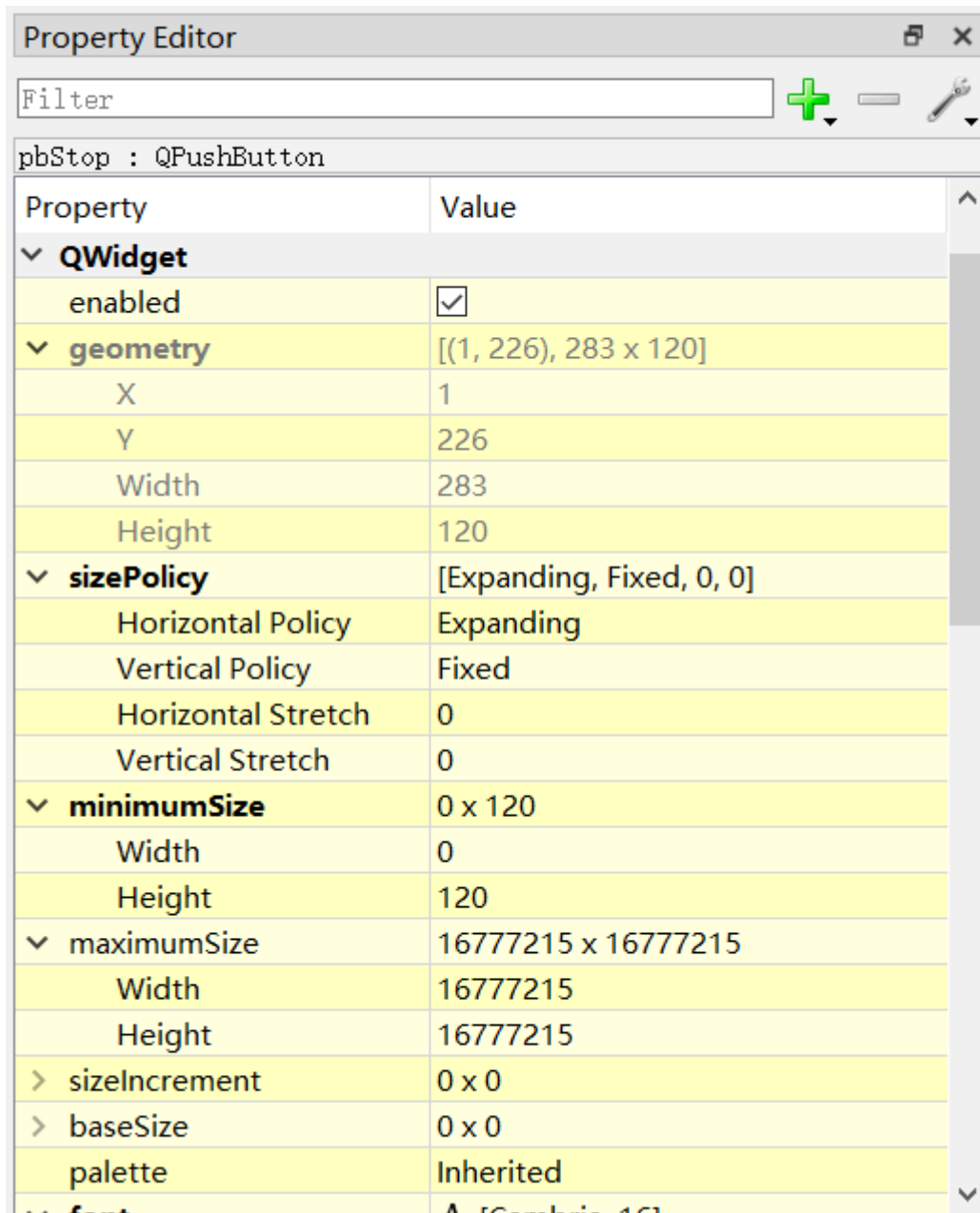
从上到下，分别是布局的名称，左右上下的边距，布局间隔-即布局内各界面组件的相对距离等信息。读者可以尝试修改这些信息，同时观察界面变化。其它的布局的属性信息大同小异。

16.6.3 组件位置及尺寸

一个组件在最终界面中的展示位置及尺寸受多方因素的制约。下图显示了STOP按钮的属性。当组件不位于任何布局容器中时，**geometry**中可以设置其**Top-Left**坐标系左上角坐标X,Y以及长宽。当组件位于布局容器中时，**geometry**不可用。

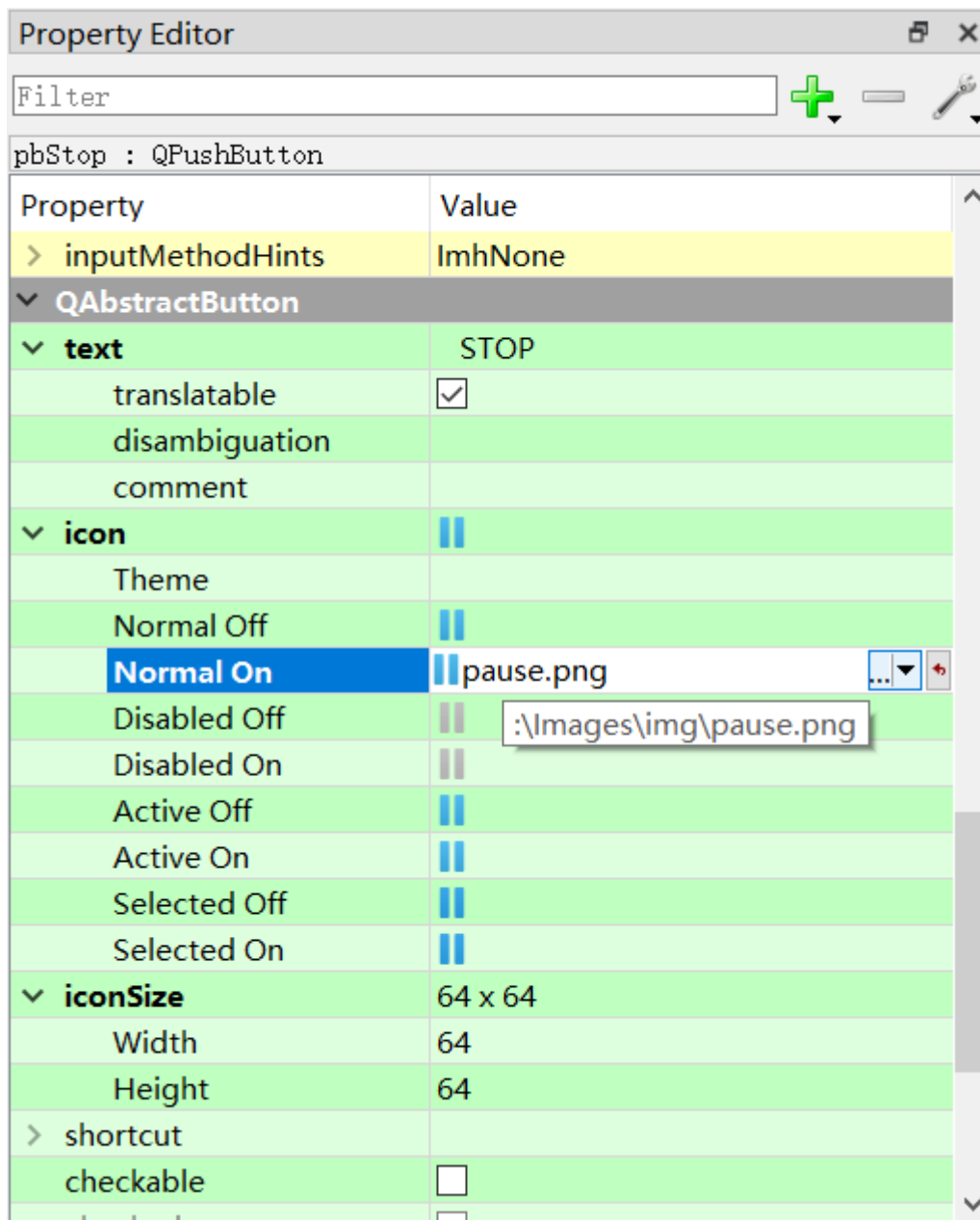
多数情况下，我们都会把组件放置在布局容器中。**sizePolicy**-尺寸策略将参与决定组件的最终尺寸，下图中，STOP按钮的**Horizontal Policy**为**Expanding**，这意味着，按钮将试图横向填满布局。STOP按钮的**Vertical Policy**则为**Fixed**，这意味着按钮的高度将为固定值，此处取了**minimumSize**中的**Height**，即120。后面的**minimum Size**和**maximum Size**则设置了组件的最大和最小尺寸。如果读者希望组件的大小是固定值，可以尝试把**minimum**和**maximum Size**设为相等的固定值。

反过来，布局中组件的**minimum Size**也可能影响布局的宽度，比如，界面中的**tbTitle**的**minimum Size**的**Width**为700，这就使得其归属容器**verticalLayout_3**被撑到700以上宽度。

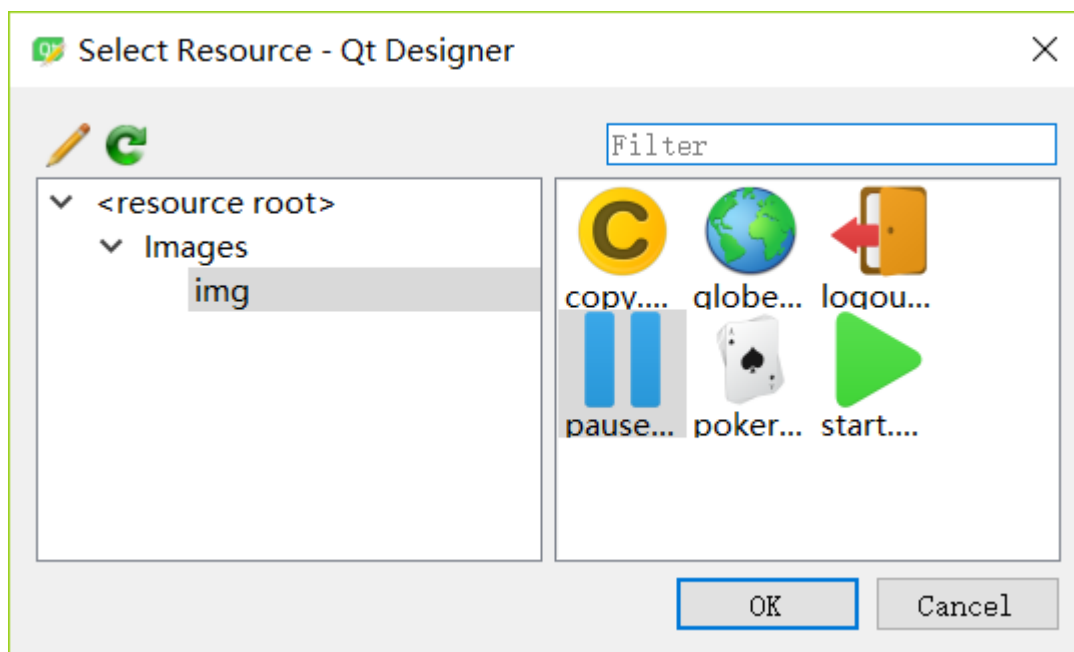


16.6.4 使用图片资源

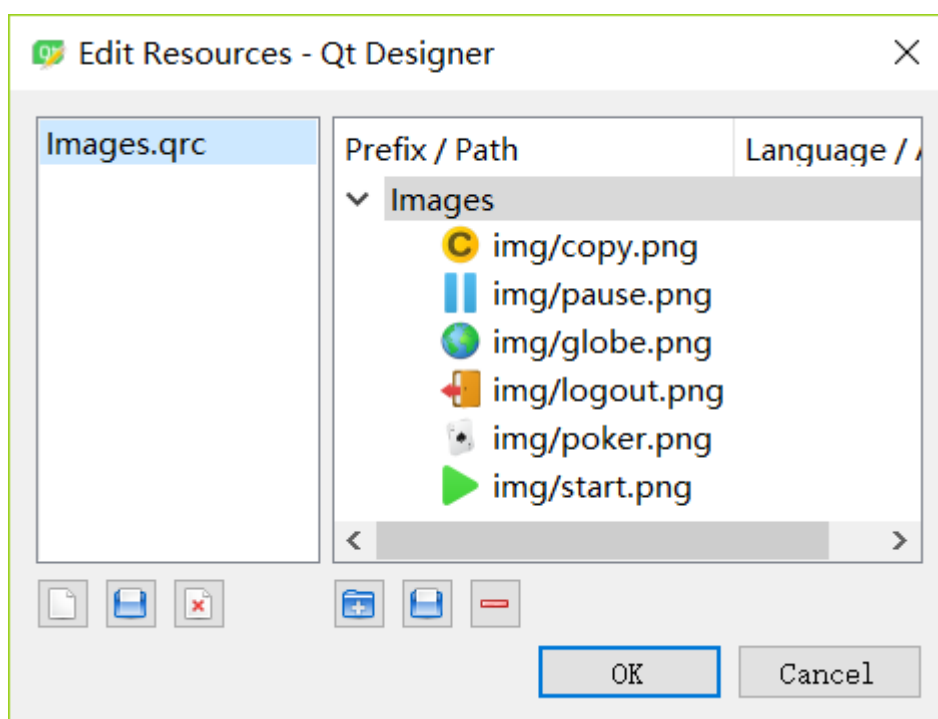
在项目目录下有个子目录名为img，其中放置了本项目中各按钮使用的LOGO-图标文件。以STOP按钮为例，其对图标文件的引用信息如下：



text属性设置了按钮上的显示文字；iconSize设置图标的长宽；在Normal On那里点有省略号的小按钮，可以打开如下界面：



通过这个资源选择器，可以选择按钮使用的图标。如前图所示，操作者甚至可以为按钮的不同状态指定不同的图标。点上图中的铅笔按钮可以进行资源编辑：



如图所示，作者创建了一个名为Images.qrc的资源文件，其中包括了6个图标文件，它们都在img子目录下。这些工具上有很多功能按钮，读者把鼠标焦点移上去即可看到提示文字。读者也可以在PyCharm的项目目录中看到文件Images.qrc。

16.7 UI转Python

Qt Designer的设计成果是扩展名为.ui的文件，比如本项目中的MainWidget.ui。作者用记事本打开了MainWidget.ui（读者注意只看不要修改），发现它事实上是一个xml格式的文本文件。

```
<?xml version="1.0" encoding="UTF-8"?>
```

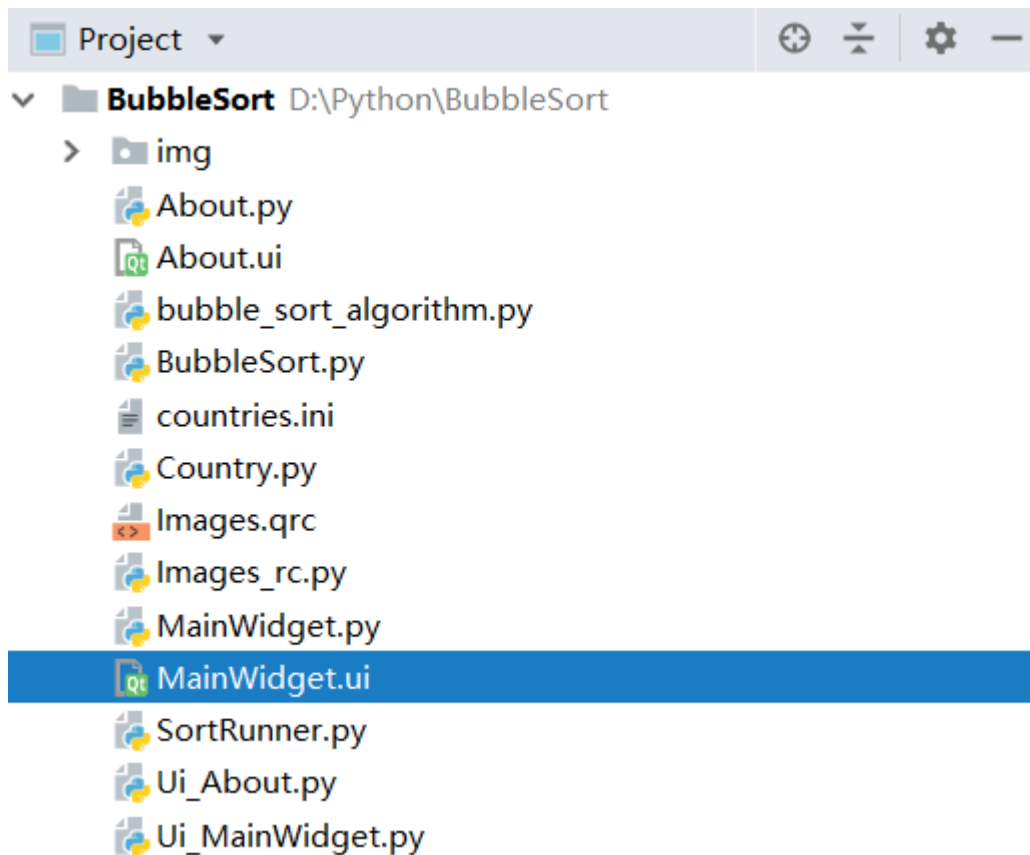
```
<ui version="4.0">
  <class>MainWidget</class>
  <widget class="QMainWindow" name="MainWidget">
    ...
    <widget class="QPushButton" name="pbStop">
      <property name="font">
        <font>
          <family>Cambria</family>
          <pointsize>16</pointsize>
        </font>
      </property>
      <property name="text">
        <string>    STOP    </string>
      </property>
      <property name="icon">
        </property>
      </property>
    </widget>
    ...
  </widget>
</ui>
```

作者随意从其中摘取了数列，可以看到，我们在Qt Designer里所作的全部设计成果，都保存在这个XML文件里了：它描述了一个窗口是如何构成的。

需要使用它，我们还需要把UI文件“编译”成Python程序。在PyCharm左边的项目管理器中先选择并高亮MainWidget.ui，然后执行Tools->PyQt5->UI Compiler，即可发现在PyCharm的下方控制台出现了一行命令：

```
C:\...\Python37\Scripts\pyuic5.exe MainWidget.ui -o Ui_MainWidget.py
```

该命令很快自动执行完成，项目目录中出现新文件Ui_MainWidget.py，如下图。事实上，读者可以选择配置UI Compiler至PyCharm，直接在操作系统终端中执行上述代码也是可以的。



接下来，看看Ui_MainWidget.py的内容：

```
from PyQt5 import QtCore, QtGui, QtWidgets

class Ui_MainWidget(object):
    def setupUi(self, MainWidget):
        MainWidget.setObjectName("MainWidget")
        MainWidget.resize(1077, 932)
        ...
        self.tbTitle = QtWidgets.QToolButton(self.centralwidget)
        self.tbTitle.setEnabled(True)
        ...
import Images_rc
```

可以看到，Ui_MainWidget.py定义了一个新类，名为Ui_MainWidget，从object类型继承。这个类型定义了一个函数setupUi()，这个函数负责用代码徒手创建一个MainWidget及其全部下层组件，然后把组件的全部属性都设置好。显然，setupUi()函数的内容是以UI文件为依据的。

由于Ui_MainWidget.py是由UI Compiler编译出来的，所以我们最好不要修改其代码。本章的后续部分，我们会定义一个MainWidget类，从Ui_MainWidget继承。我们将在MainWidget类中添加相关功能代码，避免对Ui_MainWidget.py的修改。

还应注意，上述Ui_MainWidget.py的最后一行，导入了Images_rc模块，这个模块应该是由Images.qrc资源文件编译而得。这是必要的，因为MainWidget中的按钮要使用资源文件中的图标内容。

16.8 资源编译

直接在PyCharm中双击Images.qrc文件，可以看到，它不过是一个列出了相关资源路径的XML文件。

```
<RCC>
  <qresource prefix="Images">
    <file>img/copy.png</file>
    ...
    <file>img/start.png</file>
  </qresource>
</RCC>
```

同样地，在PyCharm项目目录中选中高亮Images.qrc（最好是双击打开），然后Tools->PyQt5->Resource Compiler，PyCharm控制台执行了下述命令并生成了Images_rc.py。

```
C:\...\Python37\Scripts\pyrcc5.exe Images.qrc -o Images_rc.py
```

打开Images_rc.py，可以看到，资源文件被转换成了bytes字节流，以便后续利用。

```
from PyQt5 import QtCore

qt_resource_data = b"\
\x00\x00\x22\xb9\
\x89\
\x50\x4e\x47\x0d\x0a\x1a\x0a\x00\x00\x0d\x49\x48\x44\x52\x00\
\x00\x02\x00\x00\x00\x02\x00\x08\x06\x00\x00\x00\xf4\x78\xd4\xfa\
..."
```

16.9 BubbleSort.py

BubbleSort.py是程序的启动文件，可以看到，它首先创建了一个app对象，然后创建了MainWidget主窗口（见后节）mw，并执行mw.show()将其显示出来。最后，app.exec_()函数开始消息循环。你的主窗口将不断地查询操作系统消息队列，等待你的下一步指示。

```
import sys
from PyQt5 import QtWidgets
import MainWidget

app = QtWidgets.QApplication(sys.argv)

mw = MainWidget.MainWidget()
mw.show()

exit(app.exec_())
```


16.10 MainWindow.py

这是我们新建的程序文件，它是应用程序的主窗口。可以看到，MainWindow有两个父类，分别是QMainWindow和Ui_MainWidget。QMainWindow是Qt里表示主窗口的父类。Ui_MainWidget是我们使用UI Compiler从MainWindow.ui“编译”生成的，MainWindow从这个父类里继承了setupUi()函数，通过这个函数，MainWindow对象创建了正确的组件结构，比如，创建了一个名为pbStart的QPushButton对象，pbStart同时也是MainWindow对象的属性。在MainWindow.py中，一般地，使用self.pbStart就可以引用这个按钮对象。

```
from PyQt5 import QtWidgets, QtGui, QtCore
import Ui_MainWidget

class MainWindow(QtWidgets.QMainWindow, Ui_MainWidget.Ui_MainWidget):
    def __init__(self, parent=None):
        super(MainWidget, self).__init__(parent)
        self.setupUi(self)
```

16.10.1 国家列表

```
def createPanelCountries(self):
    for i in range(len(self.countries)):
        panel = QtWidgets.QToolButton(self.centralwidget)
        panel.setFixedHeight(52)
        panel.setSizePolicy(QSizePolicy(QSizePolicy.Expanding,
QSizePolicy.Fixed))
        panel.setFont(QtGui.QFont("Cambria", 16))
        panel.setIconSize(QtCore.QSize(64, 48))
        panel.setToolButtonStyle(QtCore.Qt.ToolButtonTextBesideIcon)
        self.verticalLayoutCountries.addWidget(panel)
        self.panelCountries.append(panel)
```

如前所述，国家及其GDP数据存储在self.countries列表中。上述createPanelCountries()成员函数则创建相同数量的QToolButton-工具按钮，并组织在self.panelCountries列表当中。这些QToolButton用来显示对应国家的国旗、国名和GDP。

self.verticalLayoutCountries.addWidget(panel)将创建好的工具按钮加入verticalLayoutCountries布局容器，国家与国家将竖向排列，一行一个。从上到下，下述代码还设置了QToolButton的固定高度、SizePolicy-尺寸策略、字体、图标大小、以及工具按钮风格-图标在左，文字在旁。请注意，QToolButton的横向尺寸策略为Expanding，这意味着，国家列表将横向扩展，尽可能填满verticalLayoutCountries的横向空间。

需要说明的是，作者在这里用QToolButton组件来显示国家信息完全是因为方便。在运行的界面中，虽然操作者可以点击界面作的国家QToolButton，但不会有任何反应。

16.10.2 国家列表显示

```

def displayCountries(self):
    if len(self.panelCountries) == 0:
        self.createPanelCountries()
    assert len(self.panelCountries) == len(self.countries)

    for x, y in zip(self.panelCountries, self.countries):
        x.setText(" " * 10 + "${:<30,.2f}{}".format(y.fGdp, y.sName))
        x.setIcon(QtGui.QIcon("img{}/{}".format(os.sep, y.sLogoFile)))
        if y.compareState == CompareState.prev:
            x.setStyleSheet("background-color: rgb(255, 255, 0);")
        elif y.compareState == CompareState.next:
            x.setStyleSheet("background-color: rgb(255, 0, 0);")
        elif y.compareState == CompareState.fixed:
            x.setStyleSheet("background-color: rgb(0, 255, 0);")
        else:
            x.setStyleSheet("")

```

在MainWidget的构造函数中，调用了displayCountries()成员函数来完成国家列表的显示。该函数首先检查国家列的组件是否已创建，如果没有，则调用self.createPanelCountries()创建国家对象列表。

接下来，zip()函数将self.countries和self.panelCountries进行了序列缝合，然后再用for循环遍历。在循环体内部，y代表一个Country对象，x则是对应的QToolButton。根据y中的国家名称、GDP，我们设置了对应QToolButton的显示文本-setText();根据y中的国旗图片文件名，我们设置了QToolButton的图标-setIcon();最后，根据y中的compareState枚举值，我们还设置了QToolButton的样式-setStyleSheet，改变其背景色，以表示对应的国家在当前排序进行程中状态：两两比较的左、右元素，已排序到位或是暂不参与比较。

16.10.3 按钮可用性

按下"START"按钮，冒泡排序将开始。而在排序开始后，"START"按钮如果还可以点，则会引起不少逻辑问题。为了避免诸如此类的问题，MainWidget.py中定义了下述函数：

```

def setToRunningState(self):
    self.pbStart.setEnabled(False)
    self.pbStop.setEnabled(True)
    self.pbShuffle.setEnabled(False)
    self.pbAbout.setEnabled(False)
    self.pbExit.setEnabled(False)

def setToIdleState(self):
    self.pbStart.setEnabled(True)
    self.pbStop.setEnabled(False)
    self.pbShuffle.setEnabled(True)
    self.pbAbout.setEnabled(True)
    self.pbExit.setEnabled(True)

```

排序开始时，将执行`setToRunningState()`函数-设置到运行态，将“STOP”按钮置为可用，而将其它按钮全部禁用-Disabled。排序结束后，`setToIdleState()`函数-设置为空闲态，将被执行，此时，除了“STOP”之外的按钮都将被设置成可用-Enabled。

16.10.4 打乱顺序

```
def on_pbShuffle_released(self):
    random.shuffle(self.countries)
    for x in self.countries:
        x.compareState = CompareState.idle
    self.displayCountries()
```

从“countries.ini”读取的原始数据是有序的，排序前需要先打乱。按下“SHUFFLE”按钮，上述成员函数将会被执行(背后的机制请见本章后续部分)。这个函数使用`random`模块把`self.countries`列表内的元素随机打乱。然后，把所有`Country`都设为“暂不参与比较的空闲状态”，再执行`displayCountries()`刷新界面上的国家列表。

16.11 信号与槽

不同的操作系统在消息循环机制方面存在差异。为了做到跨平台，Qt自己定义了一套消息分发处理机制，称为信号-signal和槽-slot。

信号大概就是消息的同义词，当我们按下某个按钮、或者在主窗口处于活动状态时按下某个键盘键，都将触发一个至多个信号。`app.exec_()`函数内部的消息循环将处理并分发这些信号至对应的信号处理程序，也就是槽。

在PyQt里，槽通常是个函数。我们可以将信号和槽连接-connect起来，即信号被触发后，槽函数将在主线程中被调用执行。一个信号可以与一个或者多个槽相连接，也可以一个都不连。

在本实践中，我们使用了两种方法将信号及槽关联起来。其中，一种就是前节中的`on_pbShuffle_released()`函数，这个成员函数的函数名有点奇怪，以`on`开头，然后是`pbShuffle`按钮对象，最后是`released`。这种成员函数的命名形式其实是PyQt的一种约定：当`pbShuffle`按钮被鼠标点击触发其`released`信号后，执行`on_pbShuffle_released()`函数。PyQt会扫描你的代码并将符合这种名字约定的信号和槽连接起来。

另一种方法我们已经在16.2节中见过了。`btnExit.clicked.connect(QtCore.QCoreApplication.quit)`将`btnExit`按钮的`clicked`信号与`QtCore.QCoreApplication.quit`槽函数连接起来。

16.12 主线程

`app._exec()`中的消息循环在程序进程的主线程中运行，主线程主要负责处理这样一些任务：

主线程的使命

-
- 执行消息循环，消息的解释和封装，信号的分发；调用执行同信号连接的槽函数；
-

- 界面内容的显示与刷新。

上述信息告诉我们两件事。第一，不要在主线程中处理耗费时间过长的任务。因为如果主线程花费时间片去处理耗时工作-比如向邮件服务器提交一个邮件发送申请，那么在这些耗时的任务未完成之前，主线程是没有时间通过消息循环去接收响应你的指令的，此时，应用程序就有“卡”住的现象：它会不理你，并且界面也不再刷新。

第二，不要在主线程的其它线程中处理与界面有关的事情，比如修改一个按钮的图标之类。因为，这些都是主线程的任务。当主线程和其它线程竞争性的“同时”使用一些资源时，可能会引起麻烦。

16.13 排序线程

实际上，15个国家的GDP冒泡排序花多了多少时间，不属于“耗时”工作的范畴。但由于我们需要演示冒泡排序的执行过程，这意味着，冒泡排序每执行一步，都需要刷新一下国家列表的显示状态，休息一会儿再继续。这种事情，线程最在行。SortRunner.py定义了本实践中的排序线程。

```
from PyQt5 import QtCore
from Country import CompareState
import time

class SortRunner(QtCore.QThread):
    updateInformer = QtCore.pyqtSignal()
    def __init__(self, countries, parent):
        super().__init__(parent)
        self.countries = countries

    def run(self):
        for x in self.countries:
            x.compareState = CompareState.idle
        self.updateInformer.emit()
        time.sleep(0.2)
        for i in range(len(self.countries)-1,0,-1):
            for j in range(0,i):
                self.countries[j].compareState = CompareState.prev
                self.countries[j+1].compareState = CompareState.next
                self.updateInformer.emit()
                time.sleep(0.5)
                if self.countries[j].fGdp < self.countries[j+1].fGdp:
                    self.countries[j],self.countries[j+1] =
self.countries[j+1],self.countries[j]
                    self.updateInformer.emit()
                    time.sleep(1.0)
                self.countries[j].compareState = CompareState.idle
                self.countries[j+1].compareState = CompareState.idle
            self.countries[i].compareState = CompareState.fixed
```

```
self.countries[0].compareState = CompareState.fixed
self.updateInformer.emit()
return
```

SoftRunner类是QtCore.QThread的子类，QThread是Qt定义的线程祖先类。构造函数接受一个countries列表并将该列表存于self.countries列表属性。当然，通过super().__init__()执行QThread的构造函数是必不可少的。

run()成员函数是这个线程的实际执行体，里面就是在对self.countries列表进行冒泡排序。注意这里排序比较的不是Country对象本身，而是比较Country的fGdp属性：self.countries[j].fGdp < self.countries[j+1].fGdp。在排序的进行过程中，还有几件事情值得注意：

- 排序算法会修改Country的compareState枚举值，以便MainWindow显示国家列表时标识排序进度。
- 排序算法会时不时执行self.updateInformer.emit()，这里的updateInformer是一个信号对象，emit()函数将触发这个信号。当这个信号被触发后，与之连接的MainWindow.py当中的槽函数handlerUpdateInformer将执行并刷新国家列表显示。updateInformer信号与handlerUpdateInformer槽函数的连接在后节中途叙。请读者注意这里的updateInformer信号对象是一个类属性。

```
# In MainWindow.py:
def handlerUpdateInformer(self):
    self.displayCountries()
```

- 排序算法在触发了updateInformer信号后会执行time.sleep()函数，参数单位为秒。time.sleep(0.5)将会主动向操作系统让出时间片，将告知操作系统，0.5秒之内我不再需要时间片，我要休息一会儿。时间到后，重新获得时间片的线程将继续排序进程。如果读者想把排序演示过程进行得慢一点，可以把time.sleep()中的时间延迟改长一点。

16.14 线程间的协调

回到主线程中运行的MainWindow.py，"START"按钮点击后，下述槽函数将被执行：

```
def on_pbStart_released(self):
    if self.sortRunner != None:
        if not self.sortRunner.isFinished():
            self.sortRunner.terminate()
            assert self.sortRunner.wait(2000)

    self.setToRunningState()

    self.sortRunner = SortRunner(self.countries, self)
    self.sortRunner.updateInformer.connect(self.handlerUpdateInformer)
    self.sortRunner.finished.connect(self.setToIdleState)
    self.sortRunner.start()
```

出于稳妥的考虑，代码首先检查了上一次排序操作的线程是否已执行完-`isFinished()`，如果没有，终止-`terminate()`，然后等待直到其事实上终止-`wait()`，如果超出2000ms，断言失败。

接下来，将界面设置至运行态，除"STOP"之外的按钮都不再可用。然后，创建了SortRunner线程对象并将`self.countries`列表数据传递给它。注意，由于“名字绑定”的关系，SortRunner内如果修改了`countries`内的元素，MainWidget的`self.countries`也会改变（两者事实上是同一个列表）。

`self.sortRunner.updateInformer.connect(self.handlerUpdateInformer)`将sortRunner的`updateInformer`信号连接至`handlerUpdateInformer`槽函数。这意味着，当sortRunner线程触发该信号后，该信号将会进入`app.exec_()`内的消息循环，主线程收到这个消息/信号后将调用对应的槽函数。对于操作系统而言，主线程和排序线程各自独立地领取时间片，排序线程只管触发信号，至于信号的槽函数何时被主线程执行，就不得而知了。但一般的，只要CPU不是过于繁忙，都会很快执行。

`self.sortRunner.finished.connect(self.setToIdleState)`这一行将sortRunner的`finished`信号连接到`setToIdleState`槽函数。`finished`信号是从QThread类继承过来的，它表明线程的`run()`函数已执行完并返回。线程结束即排序结束，`setToIdleState`将界面设置成空闲态，"START"等按钮重新变得可用。

另外，`pbStop`也有对应的槽函数：

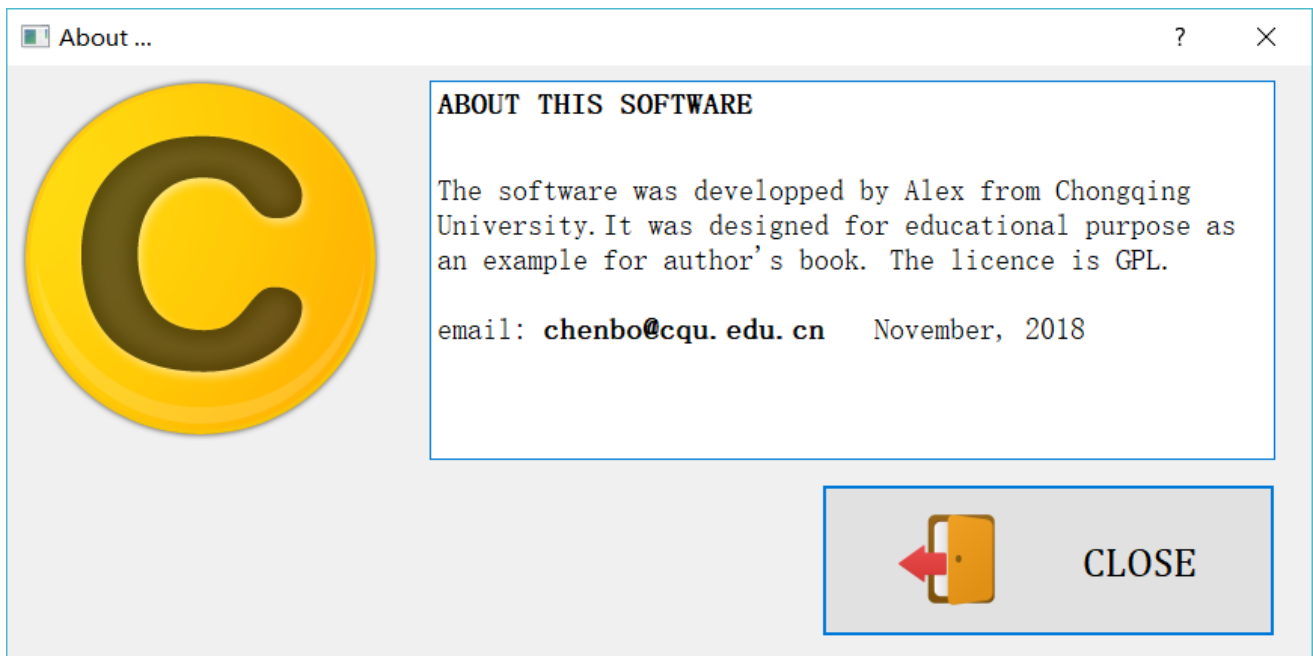
```
def on_pbStop_released(self):
    assert self.sortRunner != None
    self.sortRunner.terminate()
```

`terminate()`要求终于sortRunner线程的执行。当它真正被终止时，其`finished`信号将被触发，MainWidget.py内的`setToIdleState`槽函数将被主线程执行，界面恢复至空闲态。

到这里，我们终于说完了程序的主体部分，Enjoy it!

16.15 关于对话框

完全是为了向读者展示如何打开一个对话框-dialog，实例中还加入了一个“ABOUT"按钮功能。这很重要，因为，绝大多数应用程序都有一个主窗口及多个对话框子窗口。



About.ui是用Qt Designer设计的，编译之后有了Ui_About.py。然后继承之，得About.py中的About类，请注意，About的父类不再是QMainWindow或者QWidget，而是QDialog – Qt中的对话框祖先类。

```
from PyQt5 import QtWidgets
from Ui_About import Ui_About

class About(QtWidgets.QDialog, Ui_About):
    def __init__(self, parent):
        QtWidgets.QDialog.__init__(self, parent)
        self.setupUi(self)

    def on_pbClose_released(self):
        self.close()
```

在MainWindow.py里：

```
def on_pbAbout_released(self):
    dlg = About.About(self)
    dlg.exec()
```

dlg.exec()使得对话框以模式对话框-modal dialog形式执行，这意味着，在这个对话框结束之前，你无法再操作背后的主窗口。非模式对话框的执行函数为dlg.show()。

16.16 有坑请注意

```
def on_pbAbout_released(self):
    dlg = About.About(self)
    dlg.exec()
```


如上图，在本实践中，我们一直在响应QPushButton的released信号而不是clicked信号。在传统习惯中，clicked表示按钮被鼠标点击一次。至少在PyQt里，事实不是这样，我们尝试一下把上述代码改成响应clicked信号：

```
def on_pbAbout_clicked(self):  
    print("About button clicked.")
```

运行后，点一次About按钮，可以注意如下控制台输出，看起来，一次鼠标点击导致了两次clicked信号的发射。据推测可能是Qt内部把鼠标按下与鼠标弹起分别当成一个clicked事件处理。部分初学者不明就里，响应clicked信号进行事件处理，遇到了很多无法解释的软件BUG。作者提醒读者记住这一点，PyQt图形应用中，标准方法是响应released信号。

```
About button clicked.  
About button clicked.
```

16.17 小结

这一章的实例功能并不复杂，但信息量很大。Qt是个非常复杂，功能强大的跨平台GUI软件开发工具包。本实例只是起到把读者领进门的作用。如果读者真的希望用PyQt做点复杂的图形应用，还需要消化很多的资料，本章节远远不够。

另外，本实例中，SortRunner只管排序并向主线程发出界面刷新请求；而MainWidget.py中的代码只管展示和协调工作，不参与排序细节。MainWidget.py中的self.countries是国家列表的数据，self.panelCountries是国家列表显示的按钮列表，两者相互独立，仅在displayCountries()函数中同时使用两个列表：数据和展示是分开的。

这些细节处理试图降低程序不同部分之间的耦合度，以达到所谓松散耦合的目的。而松散耦合的程序，容易理解和维护。

练习

16-1 本实例中，SortRunner线程需要使用countries列表，MainWidget.py中的主线程也在使用同一个countries列表进行数据展示。事实上，两个线程在竞争性地使用该资源，有时，这样做会带来后果。我们希望有一种机制，可以确保同一时间，仅有一个线程使用该资源-countries列表，如果另一个线程需要用，则稍等等，等另一个线程使用完后再用。请查询资料，解决该问题。

16-2 试着修改本实例，增加一个进度条，显示排序进展到总进程的百分之多少。