目　录

致谢

介绍

PyQt5 简介

Hello World

菜单和工具栏

布局管理

事件和信号

对话框

控件(1)

控件(2)

拖拽

绘图

自定义组件

俄罗斯方块游戏

致谢

当前文档 《PyQt5中文教程》 由 进击的皇虫 使用 书栈(BookStack.CN) 进行构建，生成于 2018-02-27。

书栈(BookStack.CN) 仅提供文档编写、整理、归类等功能，以及对文档内容的生成和导出工具。

文档内容由网友们编写和整理，书栈(BookStack.CN) 难以确认文档内容知识点是否错漏。如果您在阅读文档获取知识的时候，发现文档内容有不恰当的地方，请向我们反馈，让我们共同携手，将知识准确、高效且有效地传递给每一个人。

同时，如果您在日常生活、工作和学习中遇到有价值有营养的知识文档，欢迎分享到 书栈(BookStack.CN) ，为知识的传承献上您的一份力量！

如果当前文档生成时间太久，请到 书栈(BookStack.CN) 获取最新的文档，以跟上知识更新换代的步伐。

文档地址: http://www.bookstack.cn/books/PyQt5-Chinese-tutoral

书栈官网: http://www.bookstack.cn

书栈开源: https://github.com/TruthHun

分享，让知识传承更久远！ 感谢知识的创造者，感谢知识的分享者，也感谢每一位阅读到此处的读者，因为我们都将成为知识的传承者。

介绍

- PyQt5-Chinese-tutorial
  - 翻译吐槽:
  - 更新:
    - 2017-8

PyQt5-Chinese-tutorial

PyQt5中文教程，翻译自 zetcode，项目地址: https://github.com/maicss/PyQt5-Chinese-tutoral

这个教程比较好的地方是，能讲解每一段代码的含义。

虽然PyQt的函数命名已经非常语义化了，但是对于新手来说，有这一步还是更好的。

所以我选择了翻译这篇教程，希望能给刚入门的你带来帮助。

翻译的水平有限(如有错误，请指出)，而且有些地方是自己的理解，也加入了自己的提示在里面（并没有标注出来），所以并不完全等于原文。

我尽量让翻译不带英语腔，做到即使一个完全不懂编程的人来看，虽然不知道说的啥，但是最起码语句通顺，不会读着别扭。也算是对老师的一点敬意吧~~

翻译吐槽:

- label这个词好难翻译，有时候就是个占位符的意思，说是文字说明吧，有专门的词caption，但是像checkbox的名称这种的，不

是文字说明又是啥…，但是居然还用label说图标这种事情，也是醉了。
- 源文档更新了，但是没有更新日志，只能一段段的比对……长了记性，创建了一个源文档的文件，下次直接copy的时候用VCS对比就可以了。
- 更新了一些图片，主要是原来没有的。因为手头没有Windows，而且源文档的图片也是不是Windows10，都不是一个风格的，凑合着看吧……

更新:

2017-8

- 菜单和工具栏 【新增】 右键菜单 子菜单 勾选菜单
- 事件和信号 【新增】 事件对象
- 绘图 【新增】 贝塞尔曲线

PyQt5 简介

- PyQt5 简介
  - 关于 PyQt5
  - PyQt4和PyQt5的区别
  - Python语言的介绍

PyQt5 简介

本教程的目的是带领你入门PyQt5。教程内所有代码都在Linux上测试通过。PyQt4 教程是PyQt4的教程，PyQt4是一个Python（同时支持2和3）版的Qt库。

关于 PyQt5

PyQt5 是Digia的一套Qt5与python绑定的应用框架，同时支持2.x和3.x。本教程使用的是3.x。Qt库由Riverbank Computing开发，是最强大的GUI库之一 ，官方网站：www.riverbankcomputing.co.uk/news。

PyQt5是由一系列Python模块组成。超过620个类，6000和函数和方法。能在诸如Unix、Windows和Mac OS等主流操作系统上运行。PyQt5有两种证书，GPL和商业证书。

PyQt5类分为很多模块，主要模块有：

- QtCore 包含了核心的非GUI的功能。主要和时间、文件与文件夹、各种数据、流、URLs、mime类文件、进程与线程一起使用。
- QtGui 包含了窗口系统、事件处理、2D图像、基本绘画、字体和文字类。

- QtWidgets
- QtMultimedia
- QtBluetooth
- QtNetwork
- QtPositioning
- Enginio
- QtWebSockets
- QtWebKit
- QtWebKitWidgets
- QtXml
- QtSvg
- QtSql
- QtTest

QtWidgets类包含了一系列创建桌面应用的UI元素。
QtMultimedia包含了处理多媒体的内容和调用摄像头API的类。
QtBluetooth模块包含了查找和连接蓝牙的类。
QtNetwork包含了网络编程的类，这些工具能让TCP/IP和UDP开发变得更加方便和可靠。
QtPositioning包含了定位的类，可以使用卫星、WiFi甚至文本。
Engine包含了通过客户端进入和管理Qt Cloud的类。
QtWebSockets包含了WebSocket协议的类。
QtWebKit包含了一个基WebKit2的web浏览器。
QtWebKitWidgets包含了基于QtWidgets的WebKit1的类。
QtXml包含了处理xml的类，提供了SAX和DOM API的工具。
QtSvg提供了显示SVG内容的类，Scalable Vector Graphics (SVG)是一种是一种基于可扩展标记语言（XML），用于描述二维矢量图形的图形格式（这句话来自于维基百科）。
QtSql提供了处理数据库的工具。

QtTest提供了测试PyQt5应用的工具。

PyQt4和PyQt5的区别

PyQt5不兼容**PyQt4**。PyQt5有一些巨大的改进。但是，迁移并不是很难，两者的区别如下:

- 重新组合模块，一些模块已经被废弃(QtScript)，有些被分为两个子模块(QtGui, QtWebKit)。
- 添加了新的模块，比如QtBluetooth, QtPositioning，和Enginio。
- 废弃了SINGAL()和SLOT()的调用方式，使用了新的信号和xx处理方式。
- 不再支持所有被标记为废弃的或不建议使用的Qt API。

Python语言的介绍

> 这个部分建议看百科，这里写的很简略。如果你还不太熟悉*Python*，建议先去官网看看文档。

Python is a general-purpose, dynamic, object-oriented programming language. The design purpose of the Python language emphasizes programmer productivity and code readability. Python was initially developed by Guido van Rossum. It was first released in 1991. Python was inspired by ABC, Haskell, Java, Lisp, Icon, and Perl programming languages. Python is a high-level, general purpose, multiplatform, interpreted language. Python is a minimalistic language. One of its most visible features is that it does not use semicolons nor

brackets. It uses indentation instead. There are two main branches of Python currently: Python 2.x and Python 3.x. Python 3.x breaks backward compatibility with previous releases of Python. It was created to correct some design flaws of the language and make the language more clean. The most recent version of Python 2.x is 2.7.9, and of Python 3.x is 3.4.2. Python is maintained by a large group of volunteers worldwide. Python is open source software. Python is an ideal start for those who want to learn programming.

Python programming language supports several programming styles. It does not force a programmer to a specific paradigm. Python supports object-oriented and procedural programming. There is also a limited support for functional programming.

Python语言的官方网站是python.org

Perl，Python，和Ruby都是使用最广泛的脚本语言，它们有很多共同的特点，也是相互的竞争对手。

Hello World

- Hello World
  - 本章学习Qt的基本功能
  - 例1，简单的窗口
  - 例2，带窗口图标
  - 例3，提示框
  - 例4，关闭窗口
  - 例5，消息盒子
  - 例6，窗口居中

Hello World

本章学习Qt的基本功能

例1，简单的窗口

这个简单的小例子展示的是一个小窗口。但是我们可以在这个小窗口上面做很多事情，改变大小，最大化，最小化等，这需要很多代码才能实现。这在很多应用中很常见，没必要每次都要重写这部分代码，Qt已经提供了这些功能。PyQt5是一个高级的工具集合，相比使用低级的工具，能省略上百行代码。

```
#!/usr/bin/python3
# -*- coding: utf-8 -*-

"""
ZetCode PyQt5 tutorial

In this example, we create a simple
window in PyQt5.
```

```
9.
10. author: Jan Bodnar
11. website: zetcode.com
12. Last edited: August 2017
13. """
14.
15. import sys
16. from PyQt5.QtWidgets import QApplication, QWidget
17.
18.
19. if __name__ == '__main__':
20.
21.     app = QApplication(sys.argv)
22.
23.     w = QWidget()
24.     w.resize(250, 150)
25.     w.move(300, 300)
26.     w.setWindowTitle('Simple')
27.     w.show()
28.
29.     sys.exit(app.exec_())
```

运行上面的代码，能展示出一个小窗口。

```
1. import sys
2. from PyQt5.QtWidgets import QApplication, QWidget
```

这里引入了PyQt5.QtWidgets模块，这个模块包含了基本的组件。

```
1. app = QApplication(sys.argv)
```

每个PyQt5应用都必须创建一个应用对象。sys.argv是一组命令行参数的列表。Python可以在shell里运行，这个参数提供对脚本控制的功能。

```
1. w = QWidget()
```

QWidge控件是一个用户界面的基本控件，它提供了基本的应用构造器。默认情况下，构造器是没有父级的，没有父级的构造器被称为窗口（window）。

```
1.  w.resize(250, 150)
```

resize()方法能改变控件的大小，这里的意思是窗口宽250px，高150px。

```
1.  w.move(300, 300)
```

move()是修改控件位置的的方法。它把控件放置到屏幕坐标的(300, 300)的位置。注：屏幕坐标系的原点是屏幕的左上角。

```
1.  w.setWindowTitle('Simple')
```

我们给这个窗口添加了一个标题，标题在标题栏展示（虽然这看起来是一句废话，但是后面还有各种栏，还是要注意一下，多了就蒙了）。

```
1.  w.show()
```

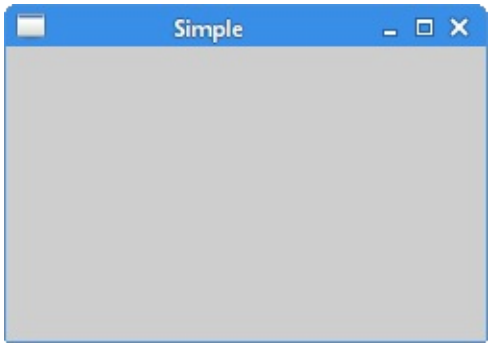
show()能让控件在桌面上显示出来。控件在内存里创建，之后才能在显示器上显示出来。

```
1.  sys.exit(app.exec_())
```

最后，我们进入了应用的主循环中，事件处理器这个时候开始工作。主循环从窗口上接收事件，并把事件传入到派发到应用控件里。当调用 `exit()` 方法或直接销毁主控件时，主循环就会结束。 `sys.exit()` 方法能确保主循环安全退出。外部环境能通知主控件怎么结束。

`exec_()` 之所以有个下划线，是因为 `exec` 是一个Python的关键字。

程序预览：



例2，带窗口图标

窗口图标通常是显示在窗口的左上角，标题栏的最左边。下面的例子就是怎么用PyQt5创建一个这样的窗口。

在某些环境下，图标显示不出来。如果你遇到了这个问题，看我在Stackoverfolw的回答

```
#!/usr/bin/python3
# -*- coding: utf-8 -*-

"""
ZetCode PyQt5 tutorial

This example shows an icon
in the titlebar of the window.

Author: Jan Bodnar
Website: zetcode.com
Last edited: August 2017
"""

import sys
from PyQt5.QtWidgets import QApplication, QWidget
```

```
from PyQt5.QtGui import QIcon


class Example(QWidget):

    def __init__(self):
        super().__init__()

        self.initUI()


    def initUI(self):

        self.setGeometry(300, 300, 300, 220)
        self.setWindowTitle('Icon')
        self.setWindowIcon(QIcon('web.png'))

        self.show()


if __name__ == '__main__':

    app = QApplication(sys.argv)
    ex = Example()
    sys.exit(app.exec_())
```

前一个例子是使用的过程式编程。Python还支持面向对象的编程：

```
class Example(QWidget):

    def __init__(self):
        super().__init__()
        ...
```

面向对象编程最重要的三个部分是类(class)、数据和方法。我们创建了一个类的调用，这个类继承自 `QWidget` 。这就意味着，我们调用了两个构造器，一个是这个类本身的，一个是这个类继承的。 `super()` 构造

器方法返回父级的对象。 `__init__()` 方法是构造器的一个方法。

```
self.initUI()
```

使用 `initUI()` 方法创建一个GUI。

```
# 自己准备一个web.png
self.setGeometry(300, 300, 300, 220)
self.setWindowTitle('Icon')
self.setWindowIcon(QIcon('web.png'))
```

上面的三个方法都继承自 `QWidget` 类。 `setGeometry()` 有两个作用：把窗口放到屏幕上并且设置窗口大小。参数分别代表屏幕坐标的x、y和窗口大小的宽、高。也就是说这个方法是 `resize()` 和 `move()` 的合体。最后一个方法是添加了图标。先创建一个QIcon对象，然后接受一个路径作为参数显示图标。

```
if __name__ == '__main__':

    app = QApplication(sys.argv)
    ex = Example()
    sys.exit(app.exec_())
```

应用和示例的对象创立，主循环开始。

程序预览：

例3，提示框

```
#!/usr/bin/python3
# -*- coding: utf-8 -*-

"""
ZetCode PyQt5 tutorial

This example shows a tooltip on
a window and a button.

Author: Jan Bodnar
Website: zetcode.com
Last edited: August 2017
"""

import sys
from PyQt5.QtWidgets import (QWidget, QToolTip,
    QPushButton, QApplication)
from PyQt5.QtGui import QFont


class Example(QWidget):

    def __init__(self):
        super().__init__()

        self.initUI()


    def initUI(self):

        QToolTip.setFont(QFont('SansSerif', 10))

        self.setToolTip('This is a <b>QWidget</b> widget')

        btn = QPushButton('Button', self)
        btn.setToolTip('This is a <b>QPushButton</b> widget')
```

```
        btn.resize(btn.sizeHint())
        btn.move(50, 50)

        self.setGeometry(300, 300, 300, 200)
        self.setWindowTitle('Tooltips')
        self.show()


if __name__ == '__main__':

    app = QApplication(sys.argv)
    ex = Example()
    sys.exit(app.exec_())
```

在这个例子中，我们为应用创建了一个提示框。

```
QToolTip.setFont(QFont('SansSerif', 10))
```

这个静态方法设置了提示框的字体，我们使用了10px的SansSerif字体。

```
self.setToolTip('This is a <b>QWidget</b> widget')
```

调用 `setTooltip()` 创建提示框可以使用富文本格式的内容。

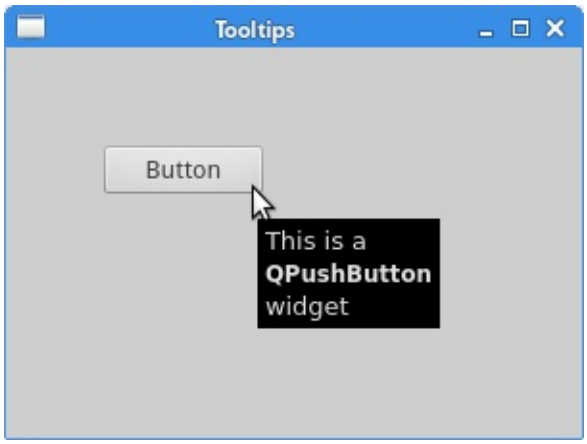
```
btn = QPushButton('Button', self)
btn.setToolTip('This is a <b>QPushButton</b> widget')
```

创建一个按钮，并且为按钮添加了一个提示框。

```
btn.resize(btn.sizeHint())
btn.move(50, 50)
```

调整按钮大小，并让按钮在屏幕上显示出来， `sizeHint()` 方法提供了一个默认的按钮大小。

程序预览：



例4，关闭窗口

关闭一个窗口最直观的方式就是点击标题栏的那个叉，这个例子里，我们展示的是如何用程序关闭一个窗口。这里我们将接触到一点single和slots的知识。

本例使用的是QPushButton组件类。

```
1. QPushButton(string text, QWidget parent = None)
```

`text` 参数是想要显示的按钮名称， `parent` 参数是放在按钮上的组件，在我们的 例子里，这个参数是 `QWidget` 。应用中的组件都是一层一层（继承而来的？）的，在这个层里，大部分的组件都有自己的父级，没有父级的组件，是顶级的窗口。

```
1. #!/usr/bin/python3
2. # -*- coding: utf-8 -*-
3.
4. """
5. ZetCode PyQt5 tutorial
6.
7. This program creates a quit
```

```
button. When we press the button,
the application terminates.

Author: Jan Bodnar
Website: zetcode.com
Last edited: August 2017
"""

import sys
from PyQt5.QtWidgets import QWidget, QPushButton, QApplication
from PyQt5.QtCore import QCoreApplication


class Example(QWidget):

    def __init__(self):
        super().__init__()

        self.initUI()


    def initUI(self):

        qbtn = QPushButton('Quit', self)
        qbtn.clicked.connect(QCoreApplication.instance().quit)
        qbtn.resize(qbtn.sizeHint())
        qbtn.move(50, 50)

        self.setGeometry(300, 300, 250, 150)
        self.setWindowTitle('Quit button')
        self.show()


if __name__ == '__main__':

    app = QApplication(sys.argv)
    ex = Example()
    sys.exit(app.exec_())
```

这里创建了一个点击之后就退出窗口的按钮。

```
1.  from PyQt5.QtCore import QCoreApplication
```

程序需要 `QtCore` 对象。

```
1.  qbtn = QPushButton('Quit', self)
```

创建一个继承自 `QPushButton` 的按钮。第一个参数是按钮的文本，第二个参数是按钮的父级组件，这个例子中，父级组件就是我们创建的继承自 `Qwidget` 的 `Example` 类。

```
1.  qbtn.clicked.connect(QCoreApplication.instance().quit)
```

事件传递系统在PyQt5内建的single和slot机制里面。点击按钮之后，信号会被捕捉并给出既定的反应。 `QCoreApplication` 包含了事件的主循环，它能添加和删除所有的事件， `instance()` 创建了一个它的实例。 `QCoreApplication` 是在 `QApplication` 里创建的。 点击事件和能终止进程并退出应用的quit函数绑定在了一起。在发送者和接受者之间建立了通讯，发送者就是按钮，接受者就是应用对象。

程序预览：

例5，消息盒子

默认情况下，我们点击标题栏的×按钮，QWidget就会关闭。但是有时候，我们修改默认行为。比如，如果我们打开的是一个文本编辑器，并且做了一些修改，我们就会想在关闭按钮的时候让用户进一步确认操作。

```
#!/usr/bin/python3
# -*- coding: utf-8 -*-

"""
ZetCode PyQt5 tutorial

This program shows a confirmation
message box when we click on the close
button of the application window.

Author: Jan Bodnar
Website: zetcode.com
Last edited: August 2017
"""

import sys
from PyQt5.QtWidgets import QWidget, QMessageBox, QApplication


class Example(QWidget):

    def __init__(self):
        super().__init__()

        self.initUI()


    def initUI(self):

        self.setGeometry(300, 300, 250, 150)
        self.setWindowTitle('Message box')
        self.show()
```

```
33.
34.
35.     def closeEvent(self, event):
36.
37.         reply = QMessageBox.question(self, 'Message',
38.             "Are you sure to quit?", QMessageBox.Yes |
39.             QMessageBox.No, QMessageBox.No)
40.
41.         if reply == QMessageBox.Yes:
42.             event.accept()
43.         else:
44.             event.ignore()
45.
46.
47. if __name__ == '__main__':
48.
49.     app = QApplication(sys.argv)
50.     ex = Example()
51.     sys.exit(app.exec_())
```

如果关闭QWidget，就会产生一个QCloseEvent。改变控件的默认行为，就是替换掉默认的事件处理。

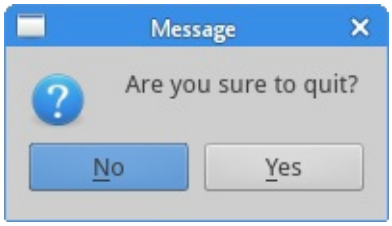
```
1. reply = QMessageBox.question(self, 'Message',
2.     "Are you sure to quit?", QMessageBox.Yes |
3.     QMessageBox.No, QMessageBox.No)
```

我们创建了一个消息框，上面有俩按钮：Yes和No.第一个字符串显示在消息框的标题栏，第二个字符串显示在对话框，第三个参数是消息框的俩按钮，最后一个参数是默认按钮，这个按钮是默认选中的。返回值在变量 `reply` 里。

```
1. if reply == QtGui.QMessageBox.Yes:
2.     event.accept()
3. else:
4.     event.ignore()
```

这里判断返回值，如果点击的是Yes按钮，我们就关闭组件和应用，否者就忽略关闭事件。

程序预览：



例6，窗口居中

```
#!/usr/bin/python3
# -*- coding: utf-8 -*-

"""
ZetCode PyQt5 tutorial

This program centers a window
on the screen.

Author: Jan Bodnar
Website: zetcode.com
Last edited: August 2017
"""

import sys
from PyQt5.QtWidgets import QWidget, QDesktopWidget, QApplication


class Example(QWidget):

    def __init__(self):
        super().__init__()

```

```
24.         self.initUI()
25.
26.
27.     def initUI(self):
28.
29.         self.resize(250, 150)
30.         self.center()
31.
32.         self.setWindowTitle('Center')
33.         self.show()
34.
35.
36.     def center(self):
37.
38.         qr = self.frameGeometry()
39.         cp = QDesktopWidget().availableGeometry().center()
40.         qr.moveCenter(cp)
41.         self.move(qr.topLeft())
42.
43.
44. if __name__ == '__main__':
45.
46.     app = QApplication(sys.argv)
47.     ex = Example()
48.     sys.exit(app.exec_())
```

`QtGui.QDesktopWidget` 提供了用户的桌面信息，包括屏幕的大小。

```
1. self.center()
```

这个方法是调用我们下面写的，实现对话框居中的方法。

```
1. qr = self.frameGeometry()
```

得到了主窗口的大小。

```
1. cp = QDesktopWidget().availableGeometry().center()
```

获取显示器的分辨率，然后得到中间点的位置。

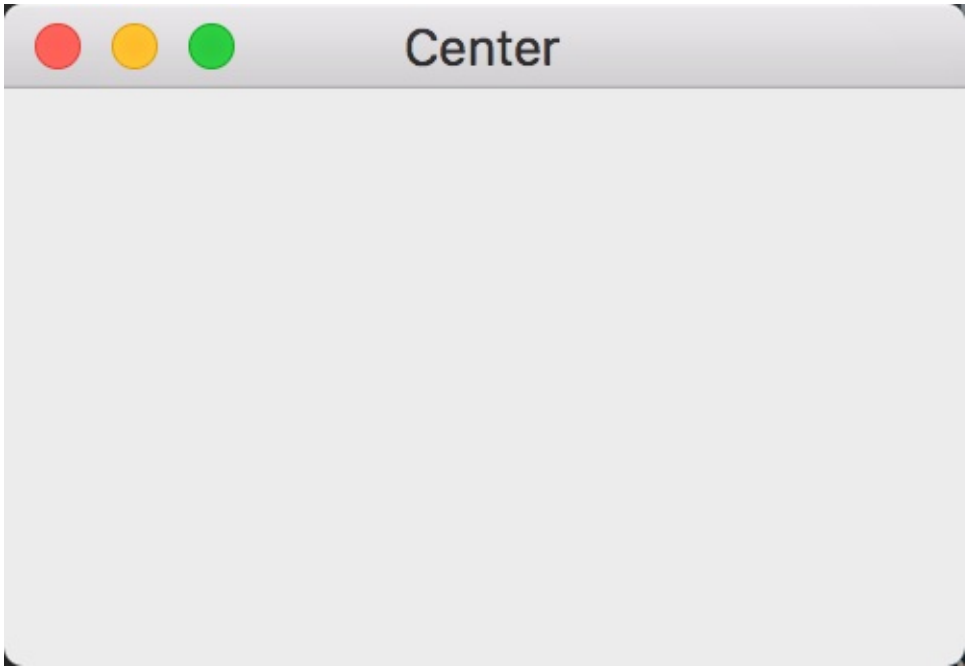
```
1.  qr.moveCenter(cp)
```

然后把自己窗口的中心点放置到qr的中心点。

```
1.  self.move(qr.topLeft())
```

然后把窗口的坐上角的坐标设置为qr的矩形左上角的坐标，这样就把窗口居中了。

程序预览：



菜单和工具栏

- 菜单和工具栏
  - 主窗口
  - 状态栏
  - 菜单栏
  - 子菜单
  - 勾选菜单
  - 右键菜单
  - 工具栏
  - 主窗口

菜单和工具栏

这个章节，我们会创建状态栏、菜单和工具栏。菜单是一组位于菜单栏的命令。工具栏是应用的一些常用工具按钮。状态栏显示一些状态信息，通常在应用的底部。

主窗口

`QMainWindow` 提供了主窗口的功能，使用它能创建一些简单的状态栏、工具栏和菜单栏。

主窗口是下面这些窗口的合称，所以教程在最下方。

状态栏

状态栏是用来显示应用的状态信息的组件。

```
1.  #!/usr/bin/python3
2.  # -*- coding: utf-8 -*-
```

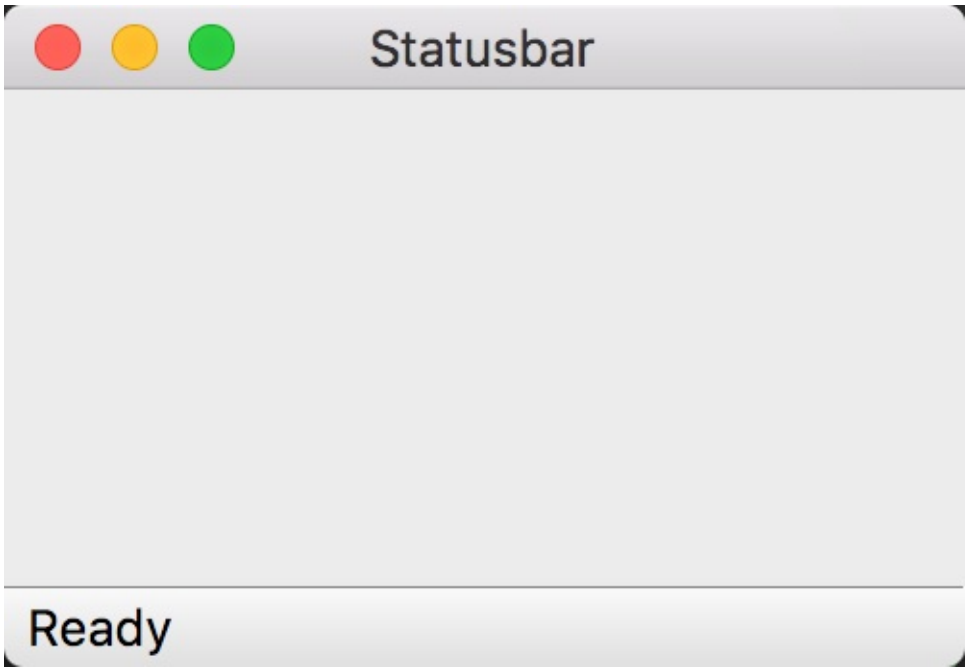
```
3.
4.  """
5.  ZetCode PyQt5 tutorial
6.
7.  This program creates a statusbar.
8.
9.  Author: Jan Bodnar
10. Website: zetcode.com
11. Last edited: August 2017
12. """
13.
14. import sys
15. from PyQt5.QtWidgets import QMainWindow, QApplication
16.
17.
18. class Example(QMainWindow):
19.
20.     def __init__(self):
21.         super().__init__()
22.
23.         self.initUI()
24.
25.
26.     def initUI(self):
27.
28.         self.statusBar().showMessage('Ready')
29.
30.         self.setGeometry(300, 300, 250, 150)
31.         self.setWindowTitle('Statusbar')
32.         self.show()
33.
34.
35. if __name__ == '__main__':
36.
37.     app = QApplication(sys.argv)
38.     ex = Example()
39.     sys.exit(app.exec_())
```

状态栏是由QMainWindow创建的。

```
1.  self.statusBar().showMessage('Ready')
```

调用 `QtGui.QMainWindow` 类的 `statusBar()` 方法，创建状态栏。第一次调用创建一个状态栏，返回一个状态栏对象。 `showMessage()` 方法在状态栏上显示一条信息。

程序预览：



菜单栏

菜单栏是非常常用的。是一组命令的集合（Mac OS下状态栏的显示不一样，为了得到最相似的外观，我们增加了一句 `menubar.setNativeMenuBar(False)` ）。

```
1.  #!/usr/bin/python3
2.  # -*- coding: utf-8 -*-
3.
4.  """
5.  ZetCode PyQt5 tutorial
```

```
6.
7.  This program creates a menubar. The
8.  menubar has one menu with an exit action.
9.
10. Author: Jan Bodnar
11. Website: zetcode.com
12. Last edited: January 2017
13. """
14.
15. import sys
16. from PyQt5.QtWidgets import QMainWindow, QAction, qApp,
    QApplication
17. from PyQt5.QtGui import QIcon
18.
19.
20. class Example(QMainWindow):
21.
22.     def __init__(self):
23.         super().__init__()
24.
25.         self.initUI()
26.
27.
28.     def initUI(self):
29.
30.         exitAct = QAction(QIcon('exit.png'), '&Exit', self)
31.         exitAct.setShortcut('Ctrl+Q')
32.         exitAct.setStatusTip('Exit application')
33.         exitAct.triggered.connect(qApp.quit)
34.
35.         self.statusBar()
36.
37.         menubar = self.menuBar()
38.         fileMenu = menubar.addMenu('&File')
39.         fileMenu.addAction(exitAct)
40.
41.         self.setGeometry(300, 300, 300, 200)
42.         self.setWindowTitle('Simple menu')
```

```
43.         self.show()
44.
45.
46. if __name__ == '__main__':
47.
48.     app = QApplication(sys.argv)
49.     ex = Example()
50.     sys.exit(app.exec_())
```

我们创建了只有一个命令的菜单栏，这个命令就是终止应用。同时也创建了一个状态栏。而且还能使用快捷键 `Ctrl+Q` 退出应用。

```
1. exitAct = QAction(QIcon('exit.png'), '&Exit', self)
2. exitAct.setShortcut('Ctrl+Q')
3. exitAct.setStatusTip('Exit application')
```

`QAction` 是菜单栏、工具栏或者快捷键的动作的组合。前面两行，我们创建了一个图标、一个exit的标签和一个快捷键组合，都执行了一个动作。第三行，创建了一个状态栏，当鼠标悬停在菜单栏的时候，能显示当前状态。

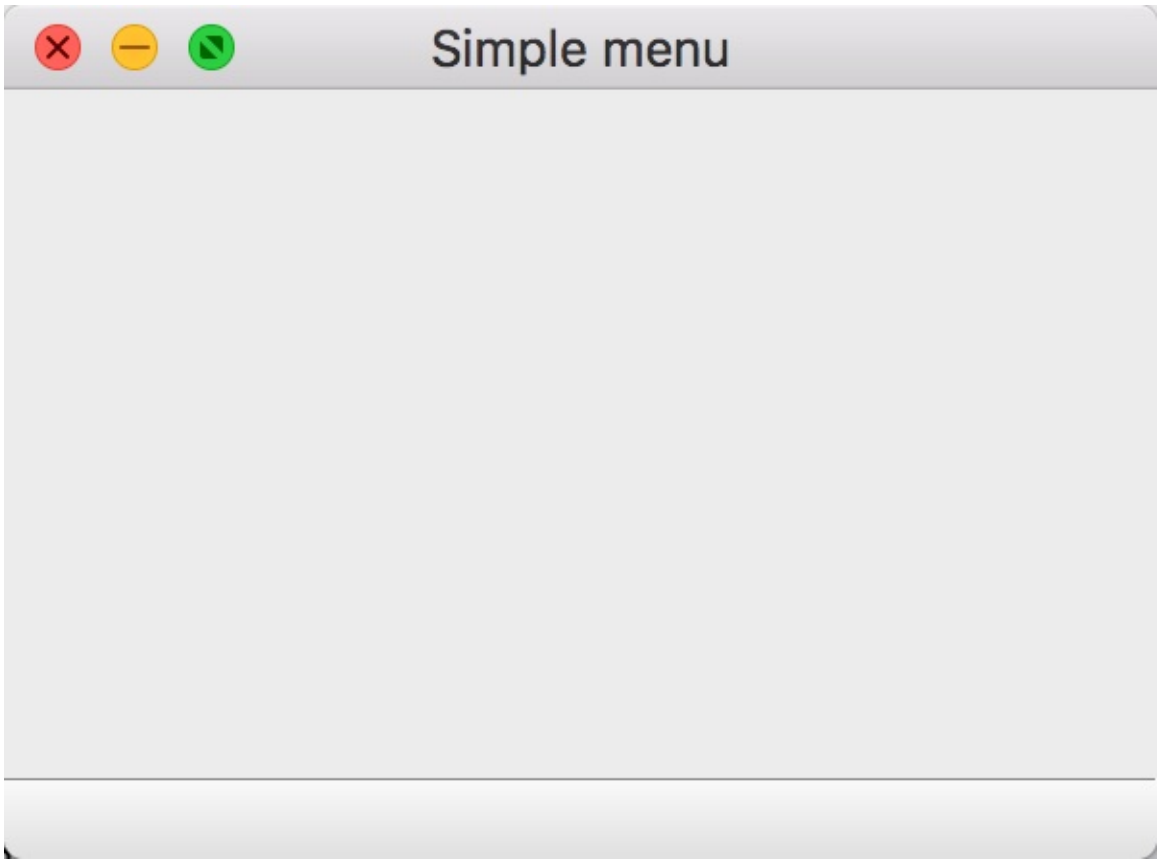
```
1. exitAct.triggered.connect(qApp.quit)
```

当执行这个指定的动作时，就触发了一个事件。这个事件跟 `QApplication的quit()` 行为相关联，所以这个动作就能终止这个应用。

```
1. menubar = self.menuBar()
2. fileMenu = menubar.addMenu('&File')
3. fileMenu.addAction(exitAct)
```

`menuBar()` 创建菜单栏。这里创建了一个菜单栏，并在上面添加了一个file菜单，并关联了点击退出应用的事件。

程序预览：



子菜单

子菜单是嵌套在菜单里面的二级或者三级等的菜单。

```
#!/usr/bin/python3
# -*- coding: utf-8 -*-

"""
ZetCode PyQt5 tutorial

This program creates a submenu.

Author: Jan Bodnar
Website: zetcode.com
Last edited: August 2017
"""

```

```
14. import sys
15. from PyQt5.QtWidgets import QMainWindow, QAction, QMenu,
    QApplication
16.
17. class Example(QMainWindow):
18.
19.     def __init__(self):
20.         super().__init__()
21.
22.         self.initUI()
23.
24.
25.     def initUI(self):
26.
27.         menubar = self.menuBar()
28.         fileMenu = menubar.addMenu('File')
29.
30.         impMenu = QMenu('Import', self)
31.         impAct = QAction('Import mail', self)
32.         impMenu.addAction(impAct)
33.
34.         newAct = QAction('New', self)
35.
36.         fileMenu.addAction(newAct)
37.         fileMenu.addMenu(impMenu)
38.
39.         self.setGeometry(300, 300, 300, 200)
40.         self.setWindowTitle('Submenu')
41.         self.show()
42.
43.
44. if __name__ == '__main__':
45.
46.     app = QApplication(sys.argv)
47.     ex = Example()
48.     sys.exit(app.exec_())
```

这个例子里，有两个子菜单，一个在file菜单下面，一个在file的

import下面。

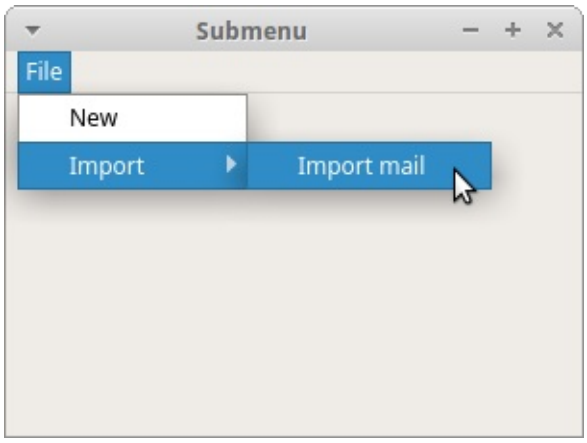
```
impMenu = QMenu('Import', self)
```

使用 `QMenu` 创建一个新菜单。

```
impAct = QAction('Import mail', self)
impMenu.addAction(impAct)
```

使用 `addAction` 添加一个动作。

程序预览：



勾选菜单

下面是一个能勾选菜单的例子

```
#!/usr/bin/python3
# -*- coding: utf-8 -*-

"""
ZetCode PyQt5 tutorial

This program creates a checkable menu.

Author: Jan Bodnar
```

```
Website: zetcode.com
Last edited: August 2017
"""

import sys
from PyQt5.QtWidgets import QMainWindow, QAction, QApplication

class Example(QMainWindow):

    def __init__(self):
        super().__init__()

        self.initUI()


    def initUI(self):

        self.statusbar = self.statusBar()
        self.statusbar.showMessage('Ready')

        menubar = self.menuBar()
        viewMenu = menubar.addMenu('View')

        viewStatAct = QAction('View statusbar', self,
checkable=True)
        viewStatAct.setStatusTip('View statusbar')
        viewStatAct.setChecked(True)
        viewStatAct.triggered.connect(self.toggleMenu)

        viewMenu.addAction(viewStatAct)

        self.setGeometry(300, 300, 300, 200)
        self.setWindowTitle('Check menu')
        self.show()

    def toggleMenu(self, state):

        if state:
```

```
            self.statusbar.show()
        else:
            self.statusbar.hide()


if __name__ == '__main__':

    app = QApplication(sys.argv)
    ex = Example()
    sys.exit(app.exec_())
```

本例创建了一个行为菜单。这个行为 / 动作能切换状态栏显示或者隐藏。

```
viewStatAct = QAction('View statusbar', self, checkable=True)
```

用 `checkable` 选项创建一个能选中的菜单。

```
viewStatAct.setChecked(True)
```

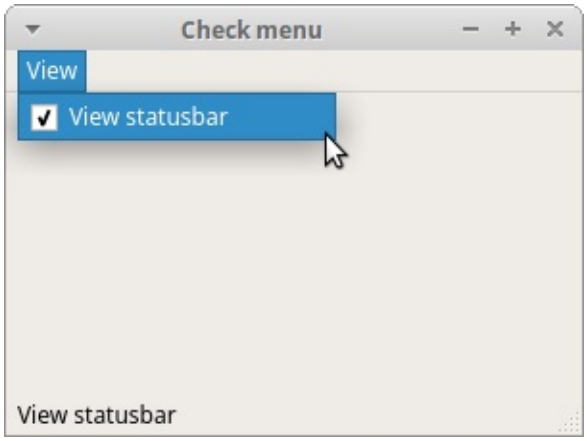
默认设置为选中状态。

```
def toggleMenu(self, state):

    if state:
        self.statusbar.show()
    else:
        self.statusbar.hide()
```

依据选中状态切换状态栏的显示与否。

程序预览：



右键菜单

右键菜单也叫弹出框（！？），是在某些场合下显示的一组命令。例如，Opera浏览器里，网页上的右键菜单里会有刷新，返回或者查看页面源代码。如果在工具栏上右键，会得到一个不同的用来管理工具栏的菜单。

```
#!/usr/bin/python3
# -*- coding: utf-8 -*-

"""
ZetCode PyQt5 tutorial

This program creates a context menu.

Author: Jan Bodnar
Website: zetcode.com
Last edited: August 2017
"""

import sys
from PyQt5.QtWidgets import QMainWindow, qApp, QMenu, QApplication

class Example(QMainWindow):

    def __init__(self):
```

```
20.         super().__init__()
21.
22.         self.initUI()
23.
24.
25.     def initUI(self):
26.
27.         self.setGeometry(300, 300, 300, 200)
28.         self.setWindowTitle('Context menu')
29.         self.show()
30.
31.
32.     def contextMenuEvent(self, event):
33.
34.             cmenu = QMenu(self)
35.
36.             newAct = cmenu.addAction("New")
37.             opnAct = cmenu.addAction("Open")
38.             quitAct = cmenu.addAction("Quit")
39.             action = cmenu.exec_(self.mapToGlobal(event.pos()))
40.
41.             if action == quitAct:
42.                 qApp.quit()
43.
44.
45. if __name__ == '__main__':
46.
47.     app = QApplication(sys.argv)
48.     ex = Example()
49.     sys.exit(app.exec_())
```

还是使用 `contextMenuEvent()` 方法实现这个菜单。

```
1.  action = cmenu.exec_(self.mapToGlobal(event.pos()))
```

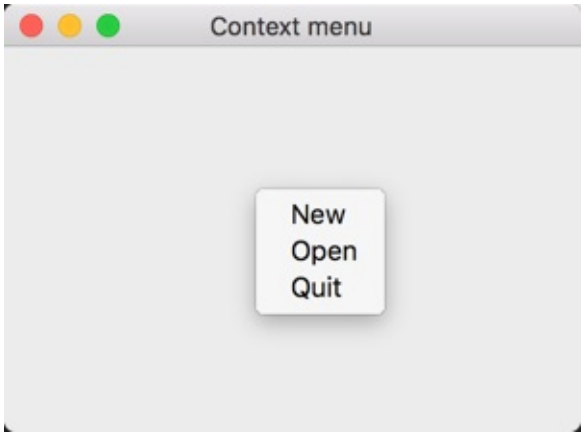
使用 `exec_()` 方法显示菜单。从鼠标右键事件对象中获得当前坐标。 `mapToGlobal()` 方法把当前组件的相对坐标转换为窗口

（window）的绝对坐标。

```
if action == quitAct:
    qApp.quit()
```

如果右键菜单里触发了事件，也就触发了推出事件，我们就关闭菜单。

程序预览：



工具栏

菜单栏包含了所有的命令，工具栏就是常用的命令的集合。

```
#!/usr/bin/python3
# -*- coding: utf-8 -*-

"""
ZetCode PyQt5 tutorial

This program creates a toolbar.
The toolbar has one action, which
terminates the application, if triggered.

Author: Jan Bodnar
Website: zetcode.com
Last edited: August 2017
```

```
"""

import sys
from PyQt5.QtWidgets import QMainWindow, QAction, qApp,
QApplication
from PyQt5.QtGui import QIcon

class Example(QMainWindow):

    def __init__(self):
        super().__init__()

        self.initUI()


    def initUI(self):

        exitAct = QAction(QIcon('exit24.png'), 'Exit', self)
        exitAct.setShortcut('Ctrl+Q')
        exitAct.triggered.connect(qApp.quit)

        self.toolbar = self.addToolBar('Exit')
        self.toolbar.addAction(exitAct)

        self.setGeometry(300, 300, 300, 200)
        self.setWindowTitle('Toolbar')
        self.show()


if __name__ == '__main__':

    app = QApplication(sys.argv)
    ex = Example()
    sys.exit(app.exec_())
```

上面的例子中，我们创建了一个工具栏。这个工具栏只有一个退出应用的动作。

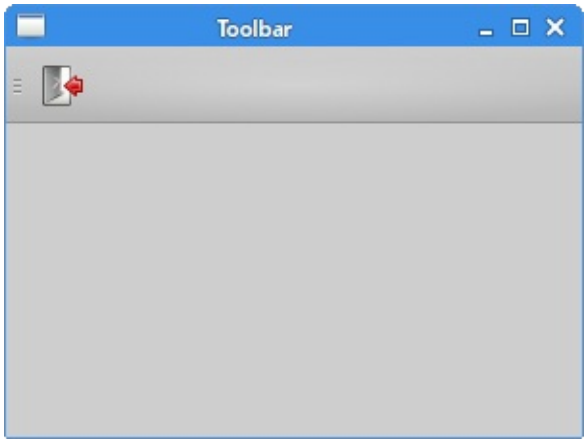
```
1.  exitAct = QAction(QIcon('exit24.png'), 'Exit', self)
2.  exitAct.setShortcut('Ctrl+Q')
3.  exitAct.triggered.connect(qApp.quit)
```

和上面的菜单栏差不多，这里使用了一个行为对象，这个对象绑定了一个标签，一个图标和一个快捷键。这些行为被触发的时候，会调用 `QtGui.QMainWindow` 的quit方法退出应用。

```
1.  self.toolbar = self.addToolBar('Exit')
2.  self.toolbar.addAction(exitAct)
```

把工具栏展示出来。

程序预览：



主窗口

主窗口就是上面三种栏目的总称，现在我们把上面的三种栏在一个应用里展示出来。

> 首先要自己弄个小图标，命名为*exit24.png*

```
1.  #!/usr/bin/python3
2.  # -*- coding: utf-8 -*-
3.
```

```
"""
ZetCode PyQt5 tutorial

This program creates a skeleton of
a classic GUI application with a menubar,
toolbar, statusbar, and a central widget.

Author: Jan Bodnar
Website: zetcode.com
Last edited: August 2017
"""

import sys
from PyQt5.QtWidgets import QMainWindow, QTextEdit, QAction,
QApplication
from PyQt5.QtGui import QIcon


class Example(QMainWindow):

    def __init__(self):
        super().__init__()

        self.initUI()


    def initUI(self):

        textEdit = QTextEdit()
        self.setCentralWidget(textEdit)

        exitAct = QAction(QIcon('exit24.png'), 'Exit', self)
        exitAct.setShortcut('Ctrl+Q')
        exitAct.setStatusTip('Exit application')
        exitAct.triggered.connect(self.close)

        self.statusBar()

```

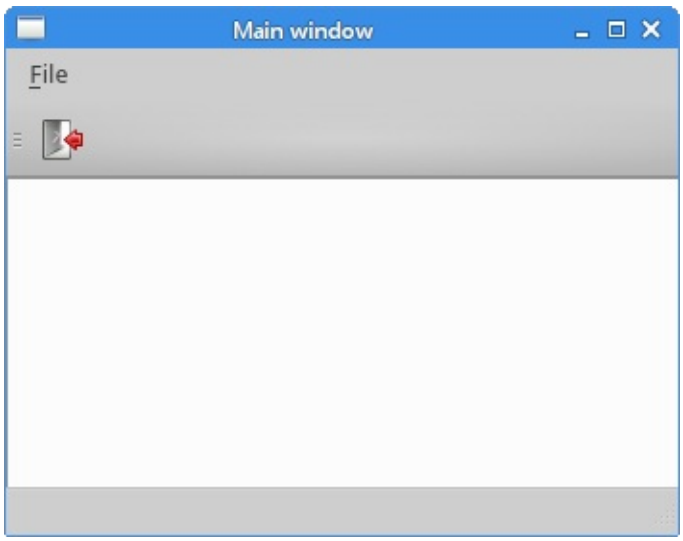
```
41.         menubar = self.menuBar()
42.         fileMenu = menubar.addMenu('&File')
43.         fileMenu.addAction(exitAct)
44.
45.         toolbar = self.addToolBar('Exit')
46.         toolbar.addAction(exitAct)
47.
48.         self.setGeometry(300, 300, 350, 250)
49.         self.setWindowTitle('Main window')
50.         self.show()
51.
52.
53. if __name__ == '__main__':
54.
55.     app = QApplication(sys.argv)
56.     ex = Example()
57.     sys.exit(app.exec_())
```

上面的代码创建了一个很经典的菜单框架，有右键菜单，工具栏和状态栏。

```
1. textEdit = QTextEdit()
2. self.setCentralWidget(textEdit)
```

这里创建了一个文本编辑区域，并把它放在 `QMainWindow` 的中间区域。这个组件或占满所有剩余的区域。

程序预览：


Main window
File

布局管理

- 布局管理
  - 绝对定位
  - 盒布局
  - 栅格布局
  - 制作提交反馈信息的布局

布局管理

在一个GUI程序里，布局是一个很重要的方面。布局就是如何管理应用中的元素和窗口。有两种方式可以搞定：绝对定位和PyQt5的layout类

绝对定位

每个程序都是以像素为单位区分元素的位置，衡量元素的大小。所以我们完全可以使用绝对定位搞定每个元素和窗口的位置。但是这也有局限性：

- 元素不会随着我们更改窗口的位置和大小而变化。
- 不能适用于不同的平台和不同分辨率的显示器
- 更改应用字体大小会破坏布局
- 如果我们决定重构这个应用，需要全部计算一下每个元素的位置和大小

下面这个就是绝对定位的应用

```
1. #!/usr/bin/python3
2. # -*- coding: utf-8 -*-
3.
```

```
"""
ZetCode PyQt5 tutorial

This example shows three labels on a window
using absolute positioning.

Author: Jan Bodnar
Website: zetcode.com
Last edited: August 2017
"""

import sys
from PyQt5.QtWidgets import QWidget, QLabel, QApplication

class Example(QWidget):

    def __init__(self):
        super().__init__()

        self.initUI()


    def initUI(self):

        lbl1 = QLabel('Zetcode', self)
        lbl1.move(15, 10)

        lbl2 = QLabel('tutorials', self)
        lbl2.move(35, 40)

        lbl3 = QLabel('for programmers', self)
        lbl3.move(55, 70)

        self.setGeometry(300, 300, 250, 150)
        self.setWindowTitle('Absolute')
        self.show()


```

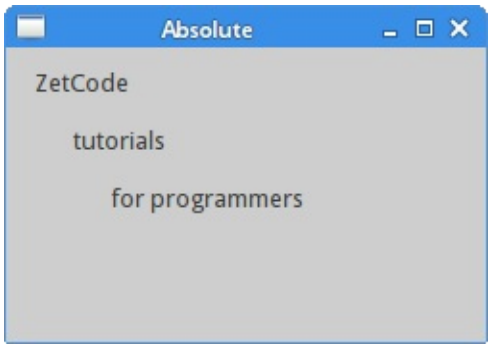
```
42. if __name__ == '__main__':
43.
44.     app = QApplication(sys.argv)
45.     ex = Example()
46.     sys.exit(app.exec_())
```

我们使用move()方法定位了每一个元素，使用x、y坐标。x、y坐标的原点是程序的左上角。

```
1. lbl1 = QLabel('Zetcode', self)
2. lbl1.move(15, 10)
```

这个元素的左上角就在这个程序的左上角开始的(15, 10)的位置。

程序展示：



盒布局

使用盒布局能让程序具有更强的适应性。这个才是布局一个应用的更合适的方式。 `QHBoxLayout` 和 `QVBoxLayout` 是基本的布局类，分别是水平布局和垂直布局。

如果我们需要把两个按钮放在程序的右下角，创建这样的布局，我们只需要一个水平布局加一个垂直布局的盒子就可以了。再用弹性布局增加一点间隙。

```
#!/usr/bin/python3
# -*- coding: utf-8 -*-

"""
ZetCode PyQt5 tutorial

In this example, we position two push
buttons in the bottom-right corner
of the window.

Author: Jan Bodnar
Website: zetcode.com
Last edited: August 2017
"""

import sys
from PyQt5.QtWidgets import (QWidget, QPushButton,
    QHBoxLayout, QVBoxLayout, QApplication)


class Example(QWidget):

    def __init__(self):
        super().__init__()

        self.initUI()


    def initUI(self):

        okButton = QPushButton("OK")
        cancelButton = QPushButton("Cancel")

        hbox = QHBoxLayout()
        hbox.addStretch(1)
        hbox.addWidget(okButton)
        hbox.addWidget(cancelButton)

```

```
        vbox = QVBoxLayout()
        vbox.addStretch(1)
        vbox.addLayout(hbox)

        self.setLayout(vbox)

        self.setGeometry(300, 300, 300, 150)
        self.setWindowTitle('Buttons')
        self.show()


if __name__ == '__main__':

    app = QApplication(sys.argv)
    ex = Example()
    sys.exit(app.exec_())
```

上面的例子完成了在应用的右下角放了两个按钮的需求。当改变窗口大小的时候，它们能依然保持在相对的位置。我们同时使用了 `QHBoxLayout` 和 `QVBoxLayout` 。

```
okButton = QPushButton("OK")
cancelButton = QPushButton("Cancel")
```

这是创建了两个按钮。

```
hbox = QHBoxLayout()
hbox.addStretch(1)
hbox.addWidget(okButton)
hbox.addWidget(cancelButton)
```

创建一个水平布局，增加两个按钮和弹性空间。stretch函数在两个按钮前面增加了一些弹性空间。下一步我们把这些元素放在应用的右下角。

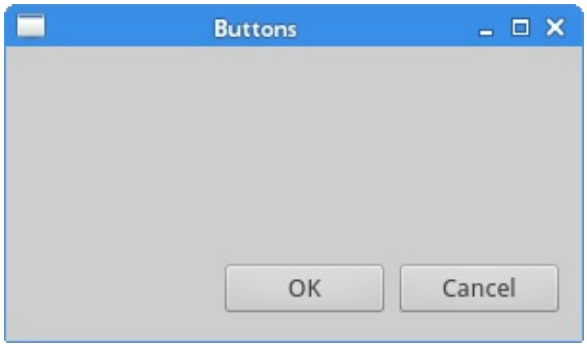
```
vbox = QVBoxLayout()
vbox.addStretch(1)
vbox.addLayout(hbox)
```

为了布局需要，我们把这个水平布局放到了一个垂直布局盒里面。弹性元素会把所有的元素一起都放置在应用的右下角。

```
self.setLayout(vbox)
```

最后，我们就得到了我们想要的布局。

程序预览：



栅格布局

最常用的还是栅格布局了。这种布局是把窗口分为行和列。创建和使用栅格布局，需要使用QGridLayout模块。

```
#!/usr/bin/python3
# -*- coding: utf-8 -*-

"""
ZetCode PyQt5 tutorial

In this example, we create a skeleton
of a calculator using a QGridLayout.

```

```
10. author: Jan Bodnar
11. website: zetcode.com
12. last edited: January 2015
13. """
14.
15. import sys
16. from PyQt5.QtWidgets import (QWidget, QGridLayout,
17.     QPushButton, QApplication)
18.
19.
20. class Example(QWidget):
21.
22.     def __init__(self):
23.         super().__init__()
24.
25.         self.initUI()
26.
27.
28.     def initUI(self):
29.
30.         grid = QGridLayout()
31.         self.setLayout(grid)
32.
33.         names = ['Cls', 'Bck', '', 'Close',
34.                  '7', '8', '9', '/',
35.                 '4', '5', '6', '*',
36.                  '1', '2', '3', '-',
37.                 '0', '.', '=', '+']
38.
39.         positions = [(i,j) for i in range(5) for j in range(4)]
40.
41.         for position, name in zip(positions, names):
42.
43.             if name == '':
44.                 continue
45.             button = QPushButton(name)
46.             grid.addWidget(button, *position)
47.
```

```
48.         self.move(300, 150)
49.         self.setWindowTitle('Calculator')
50.         self.show()
51.
52.
53. if __name__ == '__main__':
54.
55.     app = QApplication(sys.argv)
56.     ex = Example()
57.     sys.exit(app.exec_())
```

这个例子里，我们创建了栅格化的按钮。

```
1. grid = QGridLayout()
2. self.setLayout(grid)
```

创建一个QGridLayout实例，并把它放到程序窗口里。

```
1. names = ['Cls', 'Bck', '', 'Close',
2.         '7', '8', '9', '/',
3.         '4', '5', '6', '*',
4.         '1', '2', '3', '-',
5.         '0', '.', '=', '+']
```

这是我们将要使用的按钮的名称。

```
1. positions = [(i,j) for i in range(5) for j in range(4)]
```

创建按钮位置列表。

```
1. for position, name in zip(positions, names):
2.
3.     if name == '':
4.         continue
5.     button = QPushButton(name)
6.     grid.addWidget(button, *position)
```

创建按钮，并使用 `addWidget()` 方法把按钮放到布局里面。

程序预览：

制作提交反馈信息的布局

组件能跨列和跨行展示，这个例子里，我们就试试这个功能。

```
#!/usr/bin/python3
# -*- coding: utf-8 -*-

"""
ZetCode PyQt5 tutorial

In this example, we create a more
complicated window layout using
the QGridLayout manager.

Author: Jan Bodnar
Website: zetcode.com
Last edited: August 2017
"""

import sys
from PyQt5.QtWidgets import (QWidget, QLabel, QLineEdit,
    QTextEdit, QGridLayout, QApplication)

```

```
20. class Example(QWidget):
21.
22.     def __init__(self):
23.         super().__init__()
24.
25.         self.initUI()
26.
27.
28.     def initUI(self):
29.
30.         title = QLabel('Title')
31.         author = QLabel('Author')
32.         review = QLabel('Review')
33.
34.         titleEdit = QLineEdit()
35.         authorEdit = QLineEdit()
36.         reviewEdit = QTextEdit()
37.
38.         grid = QGridLayout()
39.         grid.setSpacing(10)
40.
41.         grid.addWidget(title, 1, 0)
42.         grid.addWidget(titleEdit, 1, 1)
43.
44.         grid.addWidget(author, 2, 0)
45.         grid.addWidget(authorEdit, 2, 1)
46.
47.         grid.addWidget(review, 3, 0)
48.         grid.addWidget(reviewEdit, 3, 1, 5, 1)
49.
50.         self.setLayout(grid)
51.
52.         self.setGeometry(300, 300, 350, 300)
53.         self.setWindowTitle('Review')
54.         self.show()
55.
56.
57. if __name__ == '__main__':
```

```
58.
59.      app = QApplication(sys.argv)
60.      ex = Example()
61.      sys.exit(app.exec_())
```

我们创建了一个有三个标签的窗口。两个行编辑和一个文版编辑，这是用 `QGridLayout` 模块搞定的。

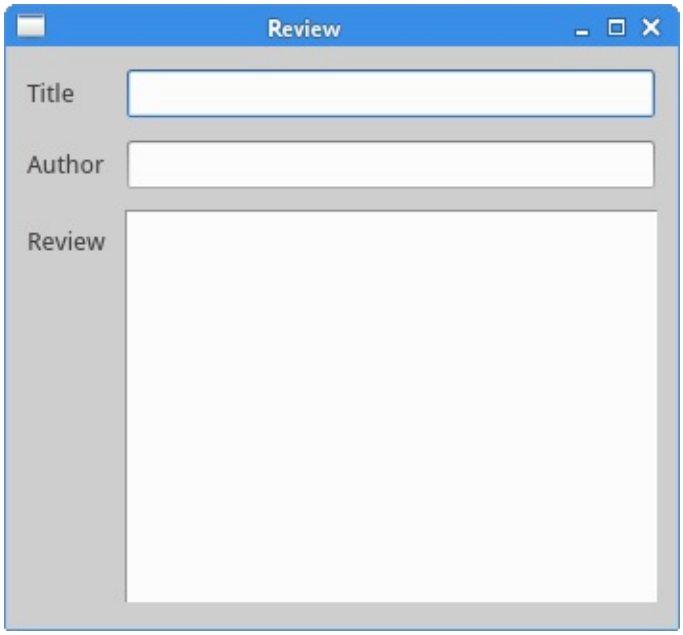
```
1.  grid = QGridLayout()
2.  grid.setSpacing(10)
```

创建标签之间的空间。

```
1.  grid.addWidget(reviewEdit, 3, 1, 5, 1)
```

我们可以指定组件的跨行和跨列的大小。这里我们指定这个元素跨5行显示。

程序预览：



事件和信号

- 事件和信号
  - 事件
  - Signals & slots
  - 重构事件处理器
  - 事件对象
  - 事件发送
  - 信号发送

事件和信号

事件

> *signals and slots* 被其他人翻译成信号和槽机制，*(⊙o⊙)*…我这里还是不翻译好了。

所有的应用都是事件驱动的。事件大部分都是由用户的行为产生的，当然也有其他的事件产生方式，比如网络的连接，窗口管理器或者定时器等。调用应用的exec_()方法时，应用会进入主循环，主循环会监听和分发事件。

在事件模型中，有三个角色：

- 事件源
- 事件
- 事件目标

事件源就是发生了状态改变的对象。事件是这个对象状态的改变撞他改变的内容。事件目标是事件想作用的目标。事件源绑定事件处理函数，然后作用于事件目标身上。

PyQt5处理事件方面有个signal and slot机制。Signals and slots用于对象间的通讯。事件触发的时候，发生一个signal，slot是用来被Python调用的（相当于一个句柄？这个词也好恶心，就是相当于事件的绑定函数）slot只有在事件触发的时候才能调用。

Signals & slots

下面是signal & slot的演示

```
#!/usr/bin/python3
# -*- coding: utf-8 -*-

"""
ZetCode PyQt5 tutorial

In this example, we connect a signal
of a QSlider to a slot of a QLCDNumber.

Author: Jan Bodnar
Website: zetcode.com
Last edited: January 2017
"""

import sys
from PyQt5.QtCore import Qt
from PyQt5.QtWidgets import (QWidget, QLCDNumber, QSlider,
    QVBoxLayout, QApplication)


class Example(QWidget):

    def __init__(self):
        super().__init__()

        self.initUI()

```

```
28.
29.     def initUI(self):
30.
31.         lcd = QLCDNumber(self)
32.         sld = QSlider(Qt.Horizontal, self)
33.
34.         vbox = QVBoxLayout()
35.         vbox.addWidget(lcd)
36.         vbox.addWidget(sld)
37.
38.         self.setLayout(vbox)
39.         sld.valueChanged.connect(lcd.display)
40.
41.         self.setGeometry(300, 300, 250, 150)
42.         self.setWindowTitle('Signal and slot')
43.         self.show()
44.
45.
46. if __name__ == '__main__':
47.
48.     app = QApplication(sys.argv)
49.     ex = Example()
50.     sys.exit(app.exec_())
```

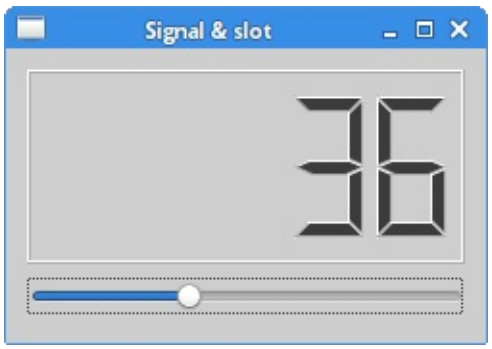
上面的例子中，显示了 `QtGui.QLCDNumber` 和 `QtGui.QSlider` 模块，我们能拖动滑块让数字跟着发生改变。

```
1. sld.valueChanged.connect(lcd.display)
```

这里是把滑块的变化和数字的变化绑定在一起。

`sender` 是信号的发送者， `receiver` 是信号的接收者， `slot` 是对这个信号应该做出的反应。

程序展示：



重构事件处理器

在PyQt5中，事件处理器经常被重写（也就是用自己的覆盖库自带的）。

```
#!/usr/bin/python3
# -*- coding: utf-8 -*-

"""
ZetCode PyQt5 tutorial

In this example, we reimplement an
event handler.

Author: Jan Bodnar
Website: zetcode.com
Last edited: August 2017
"""

import sys
from PyQt5.QtCore import Qt
from PyQt5.QtWidgets import QWidget, QApplication

class Example(QWidget):

    def __init__(self):
        super().__init__()

```

```
        self.initUI()


    def initUI(self):

        self.setGeometry(300, 300, 250, 150)
        self.setWindowTitle('Event handler')
        self.show()


    def keyPressEvent(self, e):

        if e.key() == Qt.Key_Escape:
            self.close()


if __name__ == '__main__':

    app = QApplication(sys.argv)
    ex = Example()
    sys.exit(app.exec_())
```

这个例子中，我们替换了事件处理器函数 `keyPressEvent()` 。

```
def keyPressEvent(self, e):

    if e.key() == Qt.Key_Escape:
        self.close()
```

此时如果按下ESC键程序就会退出。

程序展示：

这个就一个框，啥也没，就不展示了。

事件对象

事件对象是用python来描述一系列的事件自身属性的对象。

```
#!/usr/bin/python3
# -*- coding: utf-8 -*-

"""
ZetCode PyQt5 tutorial

In this example, we display the x and y
coordinates of a mouse pointer in a label widget.

Author: Jan Bodnar
Website: zetcode.com
Last edited: August 2017
"""

import sys
from PyQt5.QtCore import Qt
from PyQt5.QtWidgets import QWidget, QApplication, QGridLayout,
QLabel

class Example(QWidget):

    def __init__(self):
        super().__init__()

        self.initUI()


    def initUI(self):

        grid = QGridLayout()
        grid.setSpacing(10)

        x = 0
        y = 0

        self.text = "x: {0},  y: {1}".format(x, y)
```

```
36.
37.         self.label = QLabel(self.text, self)
38.         grid.addWidget(self.label, 0, 0, Qt.AlignTop)
39.
40.         self.setMouseTracking(True)
41.
42.         self.setLayout(grid)
43.
44.         self.setGeometry(300, 300, 350, 200)
45.         self.setWindowTitle('Event object')
46.         self.show()
47.
48.
49.     def mouseMoveEvent(self, e):
50.
51.         x = e.x()
52.         y = e.y()
53.
54.         text = "x: {0},  y: {1}".format(x, y)
55.         self.label.setText(text)
56.
57.
58. if __name__ == '__main__':
59.
60.     app = QApplication(sys.argv)
61.     ex = Example()
62.     sys.exit(app.exec_())
```

这个示例中，我们在一个组件里显示鼠标的X和Y坐标。

```
1. self.text = "x: {0},  y: {1}".format(x, y)
2.
3. self.label = QLabel(self.text, self)
```

X Y坐标显示在 `QLabel` 组件里

```
1. self.setMouseTracking(True)
```

鼠标追踪默认没有开启，当有鼠标点击事件发生后才会开启。

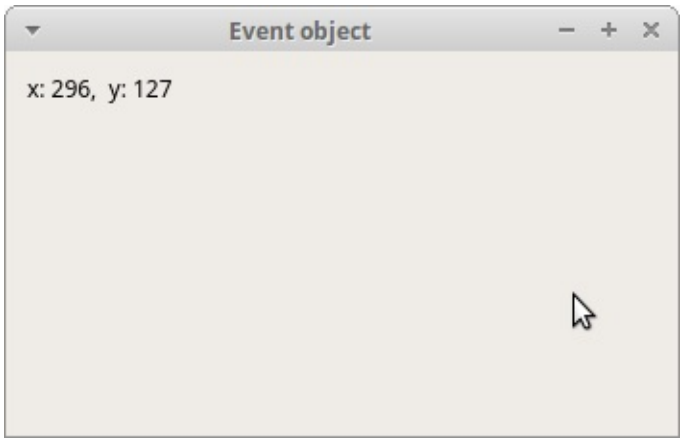
```
def mouseMoveEvent(self, e):

    x = e.x()
    y = e.y()

    text = "x: {0},  y: {1}".format(x, y)
    self.label.setText(text)
```

`e` 代表了事件对象。里面有我们触发事件（鼠标移动）的事件对象。 `x()` 和 `y()` 方法得到鼠标的x和y坐标点，然后拼成字符串输出到 `QLabel` 组件里。

程序展示：



事件发送

有时候我们会想知道是哪个组件发出了一个信号，PyQt5里的 `sender()` 方法能搞定这件事。

```
#!/usr/bin/python3
# -*- coding: utf-8 -*-

"""
```

```
ZetCode PyQt5 tutorial

In this example, we determine the event sender
object.

Author: Jan Bodnar
Website: zetcode.com
Last edited: August 2017
"""

import sys
from PyQt5.QtWidgets import QMainWindow, QPushButton, QApplication


class Example(QMainWindow):

    def __init__(self):
        super().__init__()

        self.initUI()


    def initUI(self):

        btn1 = QPushButton("Button 1", self)
        btn1.move(30, 50)

        btn2 = QPushButton("Button 2", self)
        btn2.move(150, 50)

        btn1.clicked.connect(self.buttonClicked)
        btn2.clicked.connect(self.buttonClicked)

        self.statusBar()

        self.setGeometry(300, 300, 290, 150)
        self.setWindowTitle('Event sender')
        self.show()
```

```
43.
44.
45.     def buttonClicked(self):
46.
47.         sender = self.sender()
48.         self.statusBar().showMessage(sender.text() + ' was
    pressed')
49.
50.
51. if __name__ == '__main__':
52.
53.     app = QApplication(sys.argv)
54.     ex = Example()
55.     sys.exit(app.exec_())
```

这个例子里有俩按钮，`buttonClicked()`方法决定了是哪个按钮能调用`sender()`方法。

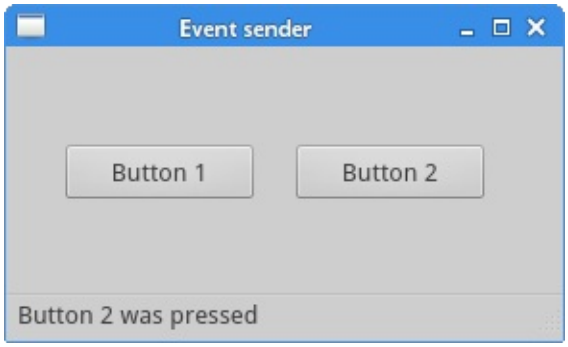
```
1. btn1.clicked.connect(self.buttonClicked)
2. btn2.clicked.connect(self.buttonClicked)
```

两个按钮都和同一个slot绑定。

```
1. def buttonClicked(self):
2.
3.     sender = self.sender()
4.     self.statusBar().showMessage(sender.text() + ' was pressed')
```

我们用调用`sender()`方法的方式决定了事件源。状态栏显示了被点击的按钮。

程序展示：



信号发送

`QObject` 实例能发送事件信号。下面的例子是发送自定义的信号。

```
#!/usr/bin/python3
# -*- coding: utf-8 -*-

"""
ZetCode PyQt5 tutorial

In this example, we show how to
emit a custom signal.

Author: Jan Bodnar
Website: zetcode.com
Last edited: August 2017
"""

import sys
from PyQt5.QtCore import pyqtSignal, QObject
from PyQt5.QtWidgets import QMainWindow, QApplication


class Communicate(QObject):

    closeApp = pyqtSignal()


class Example(QMainWindow):
```

```
26.
27.     def __init__(self):
28.         super().__init__()
29.
30.         self.initUI()
31.
32.
33.     def initUI(self):
34.
35.         self.c = Communicate()
36.         self.c.closeApp.connect(self.close)
37.
38.         self.setGeometry(300, 300, 290, 150)
39.         self.setWindowTitle('Emit signal')
40.         self.show()
41.
42.
43.     def mousePressEvent(self, event):
44.
45.         self.c.closeApp.emit()
46.
47.
48. if __name__ == '__main__':
49.
50.     app = QApplication(sys.argv)
51.     ex = Example()
52.     sys.exit(app.exec_())
```

我们创建了一个叫closeApp的信号，这个信号会在鼠标按下的时候触发，事件与 `QMainWindow` 绑定。

```
1. class Communicate(QObject):
2.
3.     closeApp = pyqtSignal()
```

`Communicate` 类创建了一个 `pyqtSignal()` 属性的信号。

```
1.  self.c = Communicate()
2.  self.c.closeApp.connect(self.close)
```

`closeApp` 信号 `QMainWindow` 的 `close()` 方法绑定。

```
1.  def mousePressEvent(self, event):
2.
3.      self.c.closeApp.emit()
```

点击窗口的时候，发送closeApp信号，程序终止。

程序展示：

这个也是啥也没。

对话框

- 对话框
  - 输入文字
  - 选取颜色
  - 选择字体
  - 选择文件

对话框

对话框是一个现代GUI应用不可或缺的一部分。对话是两个人之间的交流，对话框就是人与电脑之间的对话。对话框用来输入数据，修改数据，修改应用设置等等。

输入文字

`QInputDialog` 提供了一个简单方便的对话框，可以输入字符串，数字或列表。

```
#!/usr/bin/python3
# -*- coding: utf-8 -*-

"""
ZetCode PyQt5 tutorial

In this example, we receive data from
a QInputDialog dialog.

Aauthor: Jan Bodnar
Website: zetcode.com
Last edited: August 2017
"""

```

```
from PyQt5.QtWidgets import (QWidget, QPushButton, QLineEdit,
    QInputDialog, QApplication)
import sys

class Example(QWidget):

    def __init__(self):
        super().__init__()

        self.initUI()


    def initUI(self):

        self.btn = QPushButton('Dialog', self)
        self.btn.move(20, 20)
        self.btn.clicked.connect(self.showDialog)

        self.le = QLineEdit(self)
        self.le.move(130, 22)

        self.setGeometry(300, 300, 290, 150)
        self.setWindowTitle('Input dialog')
        self.show()


    def showDialog(self):

        text, ok = QInputDialog.getText(self, 'Input Dialog',
            'Enter your name:')

        if ok:
            self.le.setText(str(text))


if __name__ == '__main__':

    app = QApplication(sys.argv)
```

```
53.      ex = Example()
54.      sys.exit(app.exec_())
```

这个示例有一个按钮和一个输入框，点击按钮显示对话框，输入的文本会显示在输入框里。

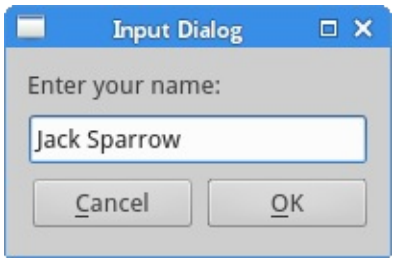
```
1.  text, ok = QInputDialog.getText(self, 'Input Dialog',
2.      'Enter your name:')
```

这是显示一个输入框的代码。第一个参数是输入框的标题，第二个参数是输入框的占位符。对话框返回输入内容和一个布尔值，如果点击的是OK按钮，布尔值就返回True。

```
1.  if ok:
2.      self.le.setText(str(text))
```

把得到的字符串放到输入框里。

程序展示：



选取颜色

QColorDialog提供颜色的选择。

```
1.  #!/usr/bin/python3
2.  # -*- coding: utf-8 -*-
3.
4.  """
```

```
ZetCode PyQt5 tutorial

In this example, we select a color value
from the QColorDialog and change the background
color of a QFrame widget.

Author: Jan Bodnar
Website: zetcode.com
Last edited: August 2017
"""

from PyQt5.QtWidgets import (QWidget, QPushButton, QFrame,
    QColorDialog, QApplication)
from PyQt5.QtGui import QColor
import sys

class Example(QWidget):

    def __init__(self):
        super().__init__()

        self.initUI()


    def initUI(self):

        col = QColor(0, 0, 0)

        self.btn = QPushButton('Dialog', self)
        self.btn.move(20, 20)

        self.btn.clicked.connect(self.showDialog)

        self.frm = QFrame(self)
        self.frm.setStyleSheet("QWidget { background-color: %s }"
            % col.name())
        self.frm.setGeometry(130, 22, 100, 100)

```

```
43.         self.setGeometry(300, 300, 250, 180)
44.         self.setWindowTitle('Color dialog')
45.         self.show()
46.
47.
48.     def showDialog(self):
49.
50.         col = QColorDialog.getColor()
51.
52.         if col.isValid():
53.             self.frm.setStyleSheet("QWidget { background-color: %s
    }"
54.                 % col.name())
55.
56.
57. if __name__ == '__main__':
58.
59.     app = QApplication(sys.argv)
60.     ex = Example()
61.     sys.exit(app.exec_())
```

例子里有一个按钮和一个 `QFrame`，默认的背景颜色为黑色，我们可以使用 `QColorDialog` 改变背景颜色。

```
1. col = QColor(0, 0, 0)
```

初始化 `QtGui.QFrame` 的背景颜色。

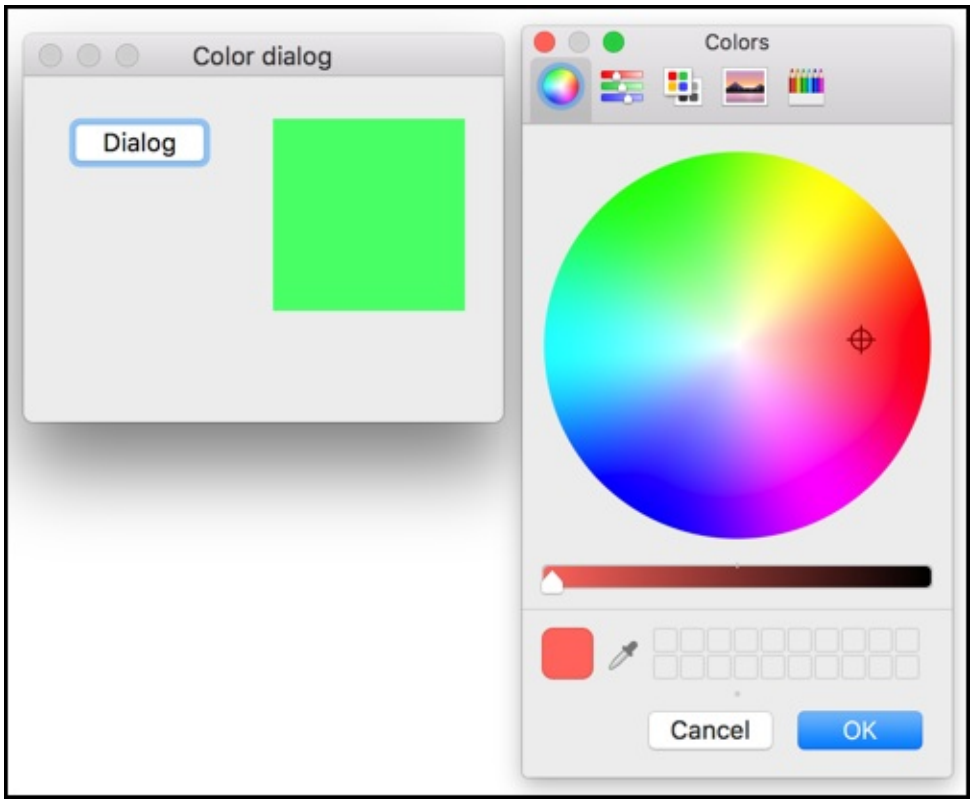
```
1. col = QColorDialog.getColor()
```

弹出一个 `QColorDialog` 对话框。

```
1. if col.isValid():
2.     self.frm.setStyleSheet("QWidget { background-color: %s }"
3.         % col.name())
```

我们可以预览颜色，如果点击取消按钮，没有颜色值返回，如果颜色是我们想要的，就从取色框里选择这个颜色。

程序展示：



选择字体

`QFontDialog` 能做字体的选择。

```
#!/usr/bin/python3
# -*- coding: utf-8 -*-

"""
ZetCode PyQt5 tutorial

In this example, we select a font name
and change the font of a label.

Author: Jan Bodnar
```

```
Website: zetcode.com
Last edited: August 2017
"""

from PyQt5.QtWidgets import (QWidget, QVBoxLayout, QPushButton,
    QSizePolicy, QLabel, QFontDialog, QApplication)
import sys

class Example(QWidget):

    def __init__(self):
        super().__init__()

        self.initUI()


    def initUI(self):

        vbox = QVBoxLayout()

        btn = QPushButton('Dialog', self)
        btn.setSizePolicy(QSizePolicy.Fixed,
            QSizePolicy.Fixed)

        btn.move(20, 20)

        vbox.addWidget(btn)

        btn.clicked.connect(self.showDialog)

        self.lbl = QLabel('Knowledge only matters', self)
        self.lbl.move(130, 20)

        vbox.addWidget(self.lbl)
        self.setLayout(vbox)

        self.setGeometry(300, 300, 250, 180)
        self.setWindowTitle('Font dialog')
```

```
        self.show()


    def showDialog(self):

        font, ok = QFontDialog.getFont()
        if ok:
            self.lbl.setFont(font)


if __name__ == '__main__':

    app = QApplication(sys.argv)
    ex = Example()
    sys.exit(app.exec_())
```

我们创建了一个有一个按钮和一个标签的 `QFontDialog` 的对话框，我们可以使用这个功能修改字体样式。

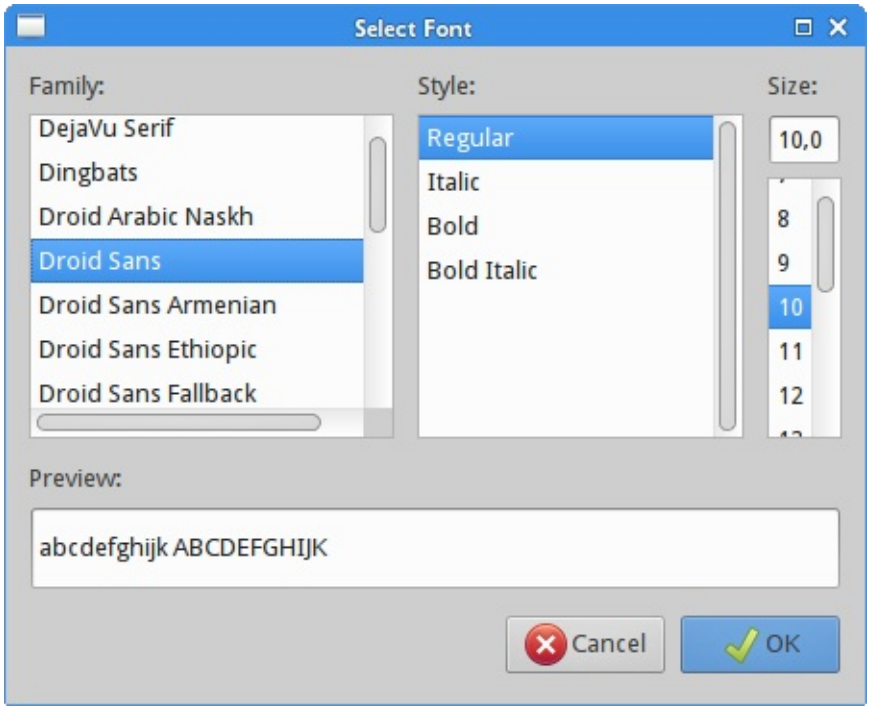
```
font, ok = QFontDialog.getFont()
```

弹出一个字体选择对话框。 `getFont()` 方法返回一个字体名称和状态信息。状态信息有OK和其他两种。

```
if ok:
    self.label.setFont(font)
```

如果点击OK，标签的字体就会随之更改。

程序展示：



选择文件

`QFileDialog` 给用户提供文件或者文件夹选择的功能。能打开和保存文件。

```
#!/usr/bin/python3
# -*- coding: utf-8 -*-

"""
ZetCode PyQt5 tutorial

In this example, we select a file with a
QFileDialog and display its contents
in a QTextEdit.

Author: Jan Bodnar
Website: zetcode.com
Last edited: August 2017
"""

from PyQt5.QtWidgets import (QMainWindow, QTextEdit,
```

```
17.     QAction, QFileDialog, QApplication)
18. from PyQt5.QtGui import QIcon
19. import sys
20. 
21. class Example(QMainWindow):
22. 
23.     def __init__(self):
24.         super().__init__()
25. 
26.         self.initUI()
27. 
28. 
29.     def initUI(self):
30. 
31.         self.textEdit = QTextEdit()
32.         self.setCentralWidget(self.textEdit)
33.         self.statusBar()
34. 
35.         openFile = QAction(QIcon('open.png'), 'Open', self)
36.         openFile.setShortcut('Ctrl+O')
37.         openFile.setStatusTip('Open new File')
38.         openFile.triggered.connect(self.showDialog)
39. 
40.         menubar = self.menuBar()
41.         fileMenu = menubar.addMenu('&File')
42.         fileMenu.addAction(openFile)
43. 
44.         self.setGeometry(300, 300, 350, 300)
45.         self.setWindowTitle('File dialog')
46.         self.show()
47. 
48. 
49.     def showDialog(self):
50. 
51.         fname = QFileDialog.getOpenFileName(self, 'Open file',
    '/home')
52. 
53.         if fname[0]:
```

```
54.             f = open(fname[0], 'r')
55.
56.             with f:
57.                 data = f.read()
58.                 self.textEdit.setText(data)
59.
60. if __name__ == '__main__':
61.
62.     app = QApplication(sys.argv)
63.     ex = Example()
64.     sys.exit(app.exec_())
```

本例中有一个菜单栏，一个置中的文本编辑框，一个状态栏。点击菜单栏选项会弹出一个 `QtGui.QFileDialog` 对话框，在这个对话框里，你能选择文件，然后文件的内容就会显示在文本编辑框里。

```
1. class Example(QMainWindow):
2.
3.     def __init__(self):
4.         super().__init__()
5.
6.         self.initUI()
```

这里设置了一个文本编辑框，文本编辑框是基于 `QMainWindow` 组件的。

```
1. fname = QFileDialog.getOpenFileName(self, 'Open file', '/home')
```

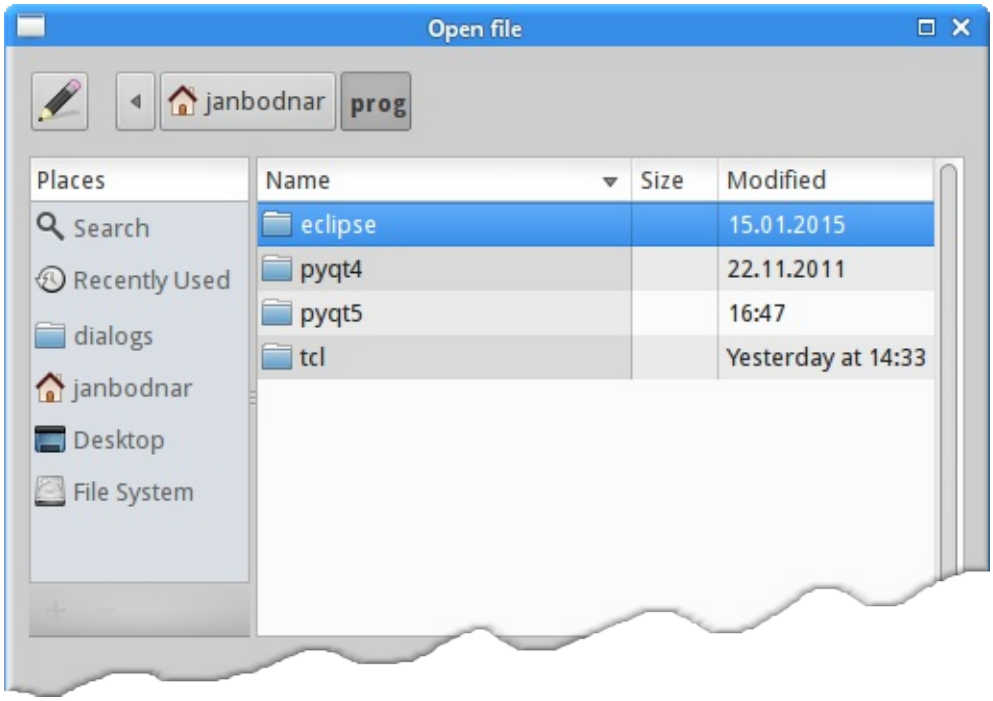
弹出 `QFileDialog` 窗口。 `getOpenFileName()` 方法的第一个参数是说明文字，第二个参数是默认打开的文件夹路径。默认情况下显示所有类型的文件。

```
1. if fname[0]:
2.     f = open(fname[0], 'r')
3.
4.     with f:
```

```
5.          data = f.read()
6.          self.textEdit.setText(data)
```

读取选中的文件，并显示在文本编辑框内（但是打开HTML文件时，是渲染后的结果，汗）。

程序展示：



控件(1)

- 控件1
  - QCheckBox
  - 切换按钮
  - 滑块
  - 进度条
  - 日历

控件1

控件就像是应用这座房子的一块块砖。PyQt5有很多的控件，比如按钮，单选框，滑动条，复选框等等。在本章，我们将介绍一些很有用的控
件：`QCheckBox`，`ToggleButton`，`QSlider`，`QProgressBar`和`QCalendarWidget`。

QCheckBox

`QCheckBox`组件有俩状态：开和关。通常跟标签一起使用，用在激活和关闭一些选项的场景。

```
1.  #!/usr/bin/python3
2.  # -*- coding: utf-8 -*-
3.
4.  """
5.  ZetCode PyQt5 tutorial
6.
7.  In this example, a QCheckBox widget
8.  is used to toggle the title of a window.
9.
10. Author: Jan Bodnar
```

```
Website: zetcode.com
Last edited: August 2017
"""

from PyQt5.QtWidgets import QWidget, QCheckBox, QApplication
from PyQt5.QtCore import Qt
import sys

class Example(QWidget):

    def __init__(self):
        super().__init__()

        self.initUI()


    def initUI(self):

        cb = QCheckBox('Show title', self)
        cb.move(20, 20)
        cb.toggle()
        cb.stateChanged.connect(self.changeTitle)

        self.setGeometry(300, 300, 250, 150)
        self.setWindowTitle('QCheckBox')
        self.show()


    def changeTitle(self, state):

        if state == Qt.Checked:
            self.setWindowTitle('QCheckBox')
        else:
            self.setWindowTitle(' ')


if __name__ == '__main__':

```

```
    app = QApplication(sys.argv)
    ex = Example()
    sys.exit(app.exec_())
```

这个例子中，有一个能切换窗口标题的单选框。

```
cb = QCheckBox('Show title', self)
```

这个是 `QCheckBox` 的构造器。

```
cb.toggle()
```

要设置窗口标题，我们就要检查单选框的状态。默认情况下，窗口没有标题，单选框未选中。

```
cb.stateChanged.connect(self.changeTitle)
```

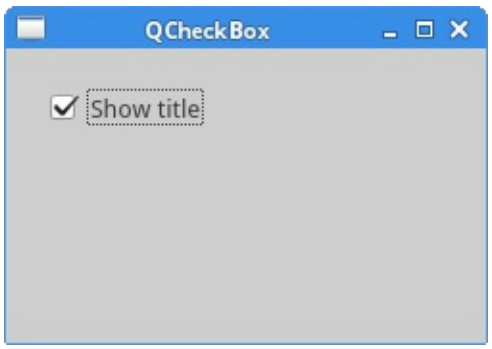
把 `changeTitle()` 方法和 `stateChanged` 信号关联起来。这样， `changeTitle()` 就能切换窗口标题了。

```
def changeTitle(self, state):

    if state == Qt.Checked:
        self.setWindowTitle('QCheckBox')
    else:
        self.setWindowTitle('')
```

控件的状态是由 `changeTitle()` 方法控制的，如果空间被选中，我们就给窗口添加一个标题，如果没被选中，就清空标题。

程序展示：



切换按钮

切换按钮就是 `QPushButton` 的一种特殊模式。 它只有两种状态：按下和未按下。我们再点击的时候切换两种状态，有很多场景会使用到这个功能。

```
#!/usr/bin/python3
# -*- coding: utf-8 -*-

"""
ZetCode PyQt5 tutorial

In this example, we create three toggle buttons.
They will control the background color of a
QFrame.

Author: Jan Bodnar
Website: zetcode.com
Last edited: August 2017
"""

from PyQt5.QtWidgets import (QWidget, QPushButton,
    QFrame, QApplication)
from PyQt5.QtGui import QColor
import sys

class Example(QWidget):

```

```
23.     def __init__(self):
24.         super().__init__()
25.
26.         self.initUI()
27.
28.
29.     def initUI(self):
30.
31.         self.col = QColor(0, 0, 0)
32.
33.         redb = QPushButton('Red', self)
34.         redb.setCheckable(True)
35.         redb.move(10, 10)
36.
37.         redb.clicked[bool].connect(self.setColor)
38.
39.         greenb = QPushButton('Green', self)
40.         greenb.setCheckable(True)
41.         greenb.move(10, 60)
42.
43.         greenb.clicked[bool].connect(self.setColor)
44.
45.         blueb = QPushButton('Blue', self)
46.         blueb.setCheckable(True)
47.         blueb.move(10, 110)
48.
49.         blueb.clicked[bool].connect(self.setColor)
50.
51.         self.square = QFrame(self)
52.         self.square.setGeometry(150, 20, 100, 100)
53.         self.square.setStyleSheet("QWidget { background-color: %s
    }" %
54.             self.col.name())
55.
56.         self.setGeometry(300, 300, 280, 170)
57.         self.setWindowTitle('Toggle button')
58.         self.show()
59.
```

```
60.
61.     def setColor(self, pressed):
62.
63.         source = self.sender()
64.
65.         if pressed:
66.             val = 255
67.         else: val = 0
68.
69.         if source.text() == "Red":
70.             self.col.setRed(val)
71.         elif source.text() == "Green":
72.             self.col.setGreen(val)
73.         else:
74.             self.col.setBlue(val)
75.
76.         self.square.setStyleSheet("QFrame { background-color: %s }"
    %
77.             self.col.name())
78.
79.
80. if __name__ == '__main__':
81.
82.     app = QApplication(sys.argv)
83.     ex = Example()
84.     sys.exit(app.exec_())
```

我们创建了一个切换按钮和一个 `QWidget` ，并把 `QWidget` 的背景设置为黑色。点击不同的切换按钮，背景色会在红、绿、蓝之间切换（而且能看到颜色合成的效果，而不是单纯的颜色覆盖）。

```
1. self.col = QColor(0, 0, 0)
```

设置颜色为黑色。

```
1. redb = QPushButton('Red', self)
2. redb.setCheckable(True)
```

```
3.  redb.move(10, 10)
```

创建一个 `QPushButton` ，然后调用它的 `setCheckable()` 的方法就把这个按钮编程了切换按钮。

```
1.  redb.clicked[bool].connect(self.setColor)
```

把点击信号和我们定义好的函数关联起来，这里是把点击事件转换成布尔值。

```
1.  source = self.sender()
```

获取被点击的按钮。

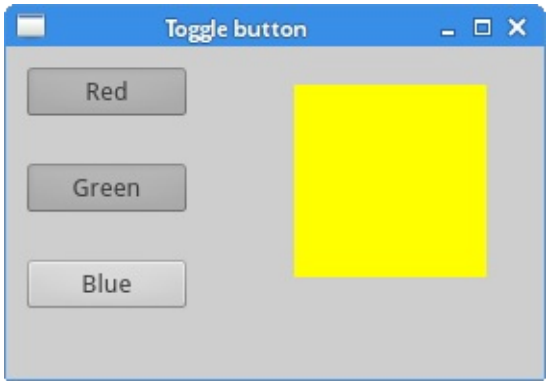
```
1.  if source.text() == "Red":
2.      self.col.setRed(val)
```

如果是标签为“red”的按钮被点击，就把颜色更改为预设好的对应颜色。

```
1.  self.square.setStyleSheet("QFrame { background-color: %s }" %
2.      self.col.name())
```

使用样式表（就是CSS的SS）改变背景色

程序展示：



滑块

`QSlider`是个有一个小滑块的组件，这个小滑块能拖着前后滑动，这个经常用于修改一些具有范围的数值，比文本框或者点击增加减少的文本框（spin box）方便多了。

本例用一个滑块和一个标签展示。标签为一个图片，滑块控制标签（的值）。

> 先弄个叫*mute.png*的静音图标准备着。

```
#!/usr/bin/python3
# -*- coding: utf-8 -*-

"""
ZetCode PyQt5 tutorial

This example shows a QSlider widget.

Author: Jan Bodnar
Website: zetcode.com
Last edited: August 2017
"""

from PyQt5.QtWidgets import (QWidget, QSlider,
    QLabel, QApplication)
from PyQt5.QtCore import Qt
from PyQt5.QtGui import QPixmap
import sys

class Example(QWidget):

    def __init__(self):
        super().__init__()

        self.initUI()

```

```
27.
28.     def initUI(self):
29.
30.         sld = QSlider(Qt.Horizontal, self)
31.         sld.setFocusPolicy(Qt.NoFocus)
32.         sld.setGeometry(30, 40, 100, 30)
33.         sld.valueChanged[int].connect(self.changeValue)
34.
35.         self.label = QLabel(self)
36.         self.label.setPixmap(QPixmap('mute.png'))
37.         self.label.setGeometry(160, 40, 80, 30)
38.
39.         self.setGeometry(300, 300, 280, 170)
40.         self.setWindowTitle('QSlider')
41.         self.show()
42.
43.
44.     def changeValue(self, value):
45.
46.         if value == 0:
47.             self.label.setPixmap(QPixmap('mute.png'))
48.         elif value > 0 and value <= 30:
49.             self.label.setPixmap(QPixmap('min.png'))
50.         elif value > 30 and value < 80:
51.             self.label.setPixmap(QPixmap('med.png'))
52.         else:
53.             self.label.setPixmap(QPixmap('max.png'))
54.
55.
56. if __name__ == '__main__':
57.
58.     app = QApplication(sys.argv)
59.     ex = Example()
60.     sys.exit(app.exec_())
```

这里是模拟的音量控制器。拖动滑块，能改变标签位置的图片。

```
1.  sld = QSlider(Qt.Horizontal, self)
```

创建一个水平的 `QSlider` 。

```
1.  self.label = QLabel(self)
2.  self.label.setPixmap(QPixmap('mute.png'))
```

创建一个 `QLabel` 组件并给它设置一个静音图标。

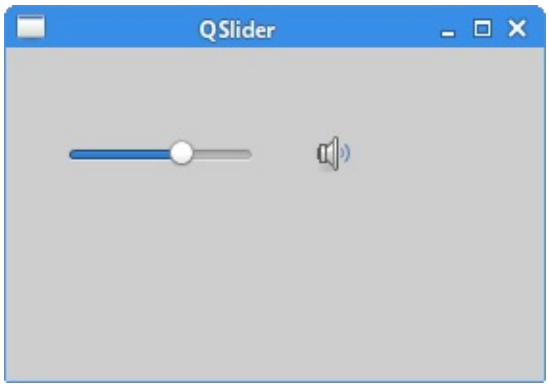
```
1.  sld.valueChanged[int].connect(self.changeValue)
```

把 `valueChanged` 信号跟 `changeValue()` 方法关联起来。

```
1.  if value == 0:
2.      self.label.setPixmap(QPixmap('mute.png'))
3.  ...
```

根据音量值的大小更换标签位置的图片。这段代码是：如果音量为0，就把图片换成 mute.png。

程序展示：



进度条

进度条是用来展示任务进度的（我也不想这样说话）。它的滚动能让用户了解到任务的进度。 `QProgressBar` 组件提供了水平和垂直两种进度条，进度条可以设置最大值和最小值，默认情况是0~99。

```
#!/usr/bin/python3
# -*- coding: utf-8 -*-

"""
ZetCode PyQt5 tutorial

This example shows a QProgressBar widget.

Author: Jan Bodnar
Website: zetcode.com
Last edited: August 2017
"""

from PyQt5.QtWidgets import (QWidget, QProgressBar,
    QPushButton, QApplication)
from PyQt5.QtCore import QBasicTimer
import sys

class Example(QWidget):

    def __init__(self):
        super().__init__()

        self.initUI()


    def initUI(self):

        self.pbar = QProgressBar(self)
        self.pbar.setGeometry(30, 40, 200, 25)

        self.btn = QPushButton('Start', self)
        self.btn.move(40, 80)
        self.btn.clicked.connect(self.doAction)

        self.timer = QBasicTimer()
        self.step = 0

```

```
        self.setGeometry(300, 300, 280, 170)
        self.setWindowTitle('QProgressBar')
        self.show()


    def timerEvent(self, e):

        if self.step >= 100:
            self.timer.stop()
            self.btn.setText('Finished')
            return

        self.step = self.step + 1
        self.pbar.setValue(self.step)


    def doAction(self):

        if self.timer.isActive():
            self.timer.stop()
            self.btn.setText('Start')
        else:
            self.timer.start(100, self)
            self.btn.setText('Stop')


if __name__ == '__main__':

    app = QApplication(sys.argv)
    ex = Example()
    sys.exit(app.exec_())
```

我们创建了一个水平的进度条和一个按钮，这个按钮控制进度条的开始和停止。

```
self.pbar = QProgressBar(self)
```

新建一个 `QProgressBar` 构造器。

```
self.timer = QtCore.QBasicTimer()
```

用时间控制进度条。

```
self.timer.start(100, self)
```

调用 `start()` 方法加载一个时间事件。这个方法有两个参数：过期时间和事件接收者。

```
def timerEvent(self, e):

    if self.step >= 100:

        self.timer.stop()
        self.btn.setText('Finished')
        return

    self.step = self.step + 1
    self.pbar.setValue(self.step)
```

每个 `QObject` 和又它继承而来的对象都有一个 `timerEvent()` 事件处理函数。为了触发事件，我们重载了这个方法。

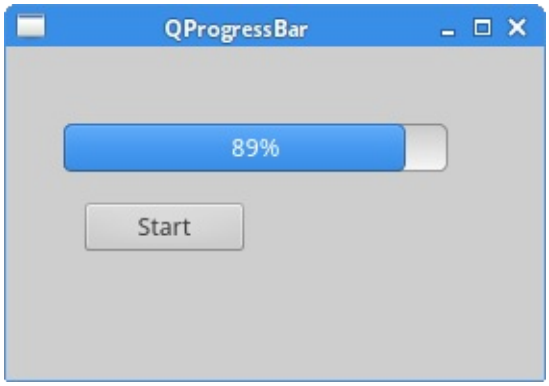
```
def doAction(self):

    if self.timer.isActive():
        self.timer.stop()
        self.btn.setText('Start')

    else:
        self.timer.start(100, self)
        self.btn.setText('Stop')
```

里面的 `doAction()` 方法是用来控制开始和停止的。

程序展示：



日历

`QCalendarWidget` 提供了基于月份的日历插件，十分简易而且直观。

```
#!/usr/bin/python3
# -*- coding: utf-8 -*-

"""
ZetCode PyQt5 tutorial

This example shows a QCalendarWidget widget.

Author: Jan Bodnar
Website: zetcode.com
Last edited: August 2017
"""

from PyQt5.QtWidgets import (QWidget, QCalendarWidget,
    QLabel, QApplication, QVBoxLayout)
from PyQt5.QtCore import QDate
import sys

class Example(QWidget):

```

```
21.     def __init__(self):
22.         super().__init__()
23.
24.         self.initUI()
25.
26.
27.     def initUI(self):
28.
29.         vbox = QVBoxLayout(self)
30.
31.         cal = QCalendarWidget(self)
32.         cal.setGridVisible(True)
33.         cal.clicked[QDate].connect(self.showDate)
34.
35.         vbox.addWidget(cal)
36.
37.         self.lbl = QLabel(self)
38.         date = cal.selectedDate()
39.         self.lbl.setText(date.toString())
40.
41.         vbox.addWidget(self.lbl)
42.
43.         self.setLayout(vbox)
44.
45.         self.setGeometry(300, 300, 350, 300)
46.         self.setWindowTitle('Calendar')
47.         self.show()
48.
49.
50.     def showDate(self, date):
51.
52.         self.lbl.setText(date.toString())
53.
54.
55. if __name__ == '__main__':
56.
57.     app = QApplication(sys.argv)
58.     ex = Example()
```

```
59.     sys.exit(app.exec_())
```

这个例子有日期组件和标签组件组成，标签显示被选中的日期。

```
1.  cal = QCalendarWidget(self)
```

创建一个 `QCalendarWidget` 。

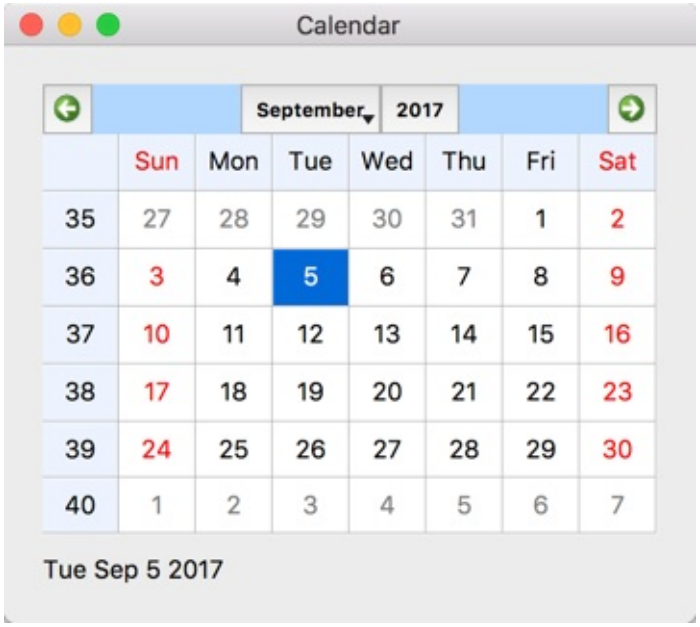
```
1.  cal.clicked[QDate].connect(self.showDate)
```

选择一个日期时， `QDate` 的点击信号就触发了，把这个信号和我们自己定义的 `showDate()` 方法关联起来。

```
1.  def showDate(self, date):
2.
3.      self.lbl.setText(date.toString())
```

使用 `selectedDate()` 方法获取选中的日期，然后把日期对象转成字符串，在标签里面显示出来。

程序展示：



控件(2)

- 控件2
  - 图片
  - 行编辑
  - QSplitter
  - 下拉选框

控件2

本章我们继续介绍PyQt5控件。这次的有 `QPixmap` ， `QLineEdit` ， `QSplitter` ，和 `QComboBox` 。

图片

`QPixmap` 是处理图片的组件。本例中，我们使用 `QPixmap` 在窗口里显示一张图片。

```
#!/usr/bin/python3
# -*- coding: utf-8 -*-

"""
ZetCode PyQt5 tutorial

In this example, we dispay an image
on the window.

Author: Jan Bodnar
Website: zetcode.com
Last edited: August 2017
"""

from PyQt5.QtWidgets import (QWidget, QHBoxLayout,
```

```
    QLabel, QApplication)
from PyQt5.QtGui import QPixmap
import sys

class Example(QWidget):

    def __init__(self):
        super().__init__()

        self.initUI()


    def initUI(self):

        hbox = QHBoxLayout(self)
        pixmap = QPixmap("redrock.png")

        lbl = QLabel(self)
        lbl.setPixmap(pixmap)

        hbox.addWidget(lbl)
        self.setLayout(hbox)

        self.move(300, 200)
        self.setWindowTitle('Red Rock')
        self.show()


if __name__ == '__main__':

    app = QApplication(sys.argv)
    ex = Example()
    sys.exit(app.exec_())
```

```
pixmap = QPixmap("redrock.png")
```

创建一个 `QPixmap` 对象，接收一个文件作为参数。

```
lbl = QLabel(self)
lbl.setPixmap(pixmap)
```

把 `QPixmap` 实例放到 `QLabel` 组件里。

程序展示：

行编辑

`QLineEdit` 组件提供了编辑文本的功能，自带了撤销、重做、剪切、粘贴、拖拽等功能。

```
#!/usr/bin/python3
# -*- coding: utf-8 -*-

"""
ZetCode PyQt5 tutorial

This example shows text which
is entered in a QLineEdit
```

```
in a QLabel widget.

Author: Jan Bodnar
Website: zetcode.com
Last edited: August 2017
"""

import sys
from PyQt5.QtWidgets import (QWidget, QLabel,
    QLineEdit, QApplication)


class Example(QWidget):

    def __init__(self):
        super().__init__()

        self.initUI()


    def initUI(self):

        self.lbl = QLabel(self)
        qle = QLineEdit(self)

        qle.move(60, 100)
        self.lbl.move(60, 40)

        qle.textChanged[str].connect(self.onChanged)

        self.setGeometry(300, 300, 280, 170)
        self.setWindowTitle('QLineEdit')
        self.show()


    def onChanged(self, text):

        self.lbl.setText(text)
```

```
47.         self.lbl.adjustSize()
48.
49.
50. if __name__ == '__main__':
51.
52.     app = QApplication(sys.argv)
53.     ex = Example()
54.     sys.exit(app.exec_())
```

例子中展示了一个编辑组件和一个标签，我们在输入框里键入的文本，会立即在标签里显示出来。

```
1. qle = QLineEdit(self)
```

创建一个 `QLineEdit` 对象。

```
1. qle.textChanged[str].connect(self.onChanged)
```

如果输入框的值有变化，就调用 `onChanged()` 方法。

```
1. def onChanged(self, text):
2.
3.     self.lbl.setText(text)
4.     self.lbl.adjustSize()
```

在 `onChanged()` 方法内部，我们把文本框里的值赋值给了标签组件，然后调用 `adjustSize()` 方法让标签自适应文本内容。

程序展示：

QSplitter

`QSplitter` 组件能让用户通过拖拽分割线的方式改变子窗口大小的组件。本例中我们展示用两个分割线隔开的三个 `QFrame` 组件。

```
#!/usr/bin/python3
# -*- coding: utf-8 -*-

"""
ZetCode PyQt5 tutorial

This example shows
how to use QSplitter widget.

Author: Jan Bodnar
Website: zetcode.com
Last edited: August 2017
"""

from PyQt5.QtWidgets import (QWidget, QHBoxLayout, QFrame,
    QSplitter, QStyleFactory, QApplication)
from PyQt5.QtCore import Qt
import sys

class Example(QWidget):

    def __init__(self):
        super().__init__()
```

```
24.
25.         self.initUI()
26.
27.
28.     def initUI(self):
29.
30.         hbox = QHBoxLayout(self)
31.
32.         topleft = QFrame(self)
33.         topleft.setFrameShape(QFrame.StyledPanel)
34.
35.         topright = QFrame(self)
36.         topright.setFrameShape(QFrame.StyledPanel)
37.
38.         bottom = QFrame(self)
39.         bottom.setFrameShape(QFrame.StyledPanel)
40.
41.         splitter1 = QSplitter(Qt.Horizontal)
42.         splitter1.addWidget(topleft)
43.         splitter1.addWidget(topright)
44.
45.         splitter2 = QSplitter(Qt.Vertical)
46.         splitter2.addWidget(splitter1)
47.         splitter2.addWidget(bottom)
48.
49.         hbox.addWidget(splitter2)
50.         self.setLayout(hbox)
51.
52.         self.setGeometry(300, 300, 300, 200)
53.         self.setWindowTitle('QSplitter')
54.         self.show()
55.
56.
57.     def onChanged(self, text):
58.
59.         self.lbl.setText(text)
60.         self.lbl.adjustSize()
61.
```

```
62.
63. if __name__ == '__main__':
64.
65.     app = QApplication(sys.argv)
66.     ex = Example()
67.     sys.exit(app.exec_())
```

三个窗口和两个分割线的布局创建完成了，但是要注意，有些主题下，分割线的显示效果不太好。

```
1. topleft = QFrame(self)
2. topleft.setFrameShape(QFrame.StyledPanel)
```

为了更清楚的看到分割线，我们使用了设置好的子窗口样式。

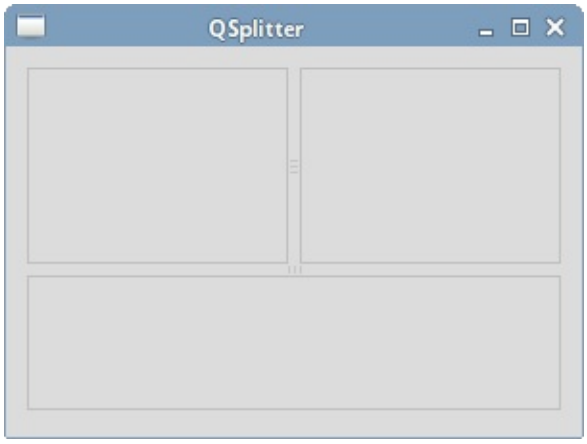
```
1. splitter1 = QSplitter(Qt.Horizontal)
2. splitter1.addWidget(topleft)
3. splitter1.addWidget(topright)
```

创建一个 `QSplitter` 组件，并在里面添加了两个框架。

```
1. splitter2 = QSplitter(Qt.Vertical)
2. splitter2.addWidget(splitter1)
```

也可以在分割线里面再进行分割。

程序展示：



下拉选框

`QComboBox` 组件能让用户在多个选择项中选择一个。

```
#!/usr/bin/python3
# -*- coding: utf-8 -*-

"""
ZetCode PyQt5 tutorial

This example shows how to use
a QComboBox widget.

Author: Jan Bodnar
Website: zetcode.com
Last edited: August 2017
"""

from PyQt5.QtWidgets import (QWidget, QLabel,
    QComboBox, QApplication)
import sys

class Example(QWidget):

    def __init__(self):
        super().__init__()

```

```
24.         self.initUI()
25.
26.
27.     def initUI(self):
28.
29.         self.lbl = QLabel("Ubuntu", self)
30.
31.         combo = QComboBox(self)
32.         combo.addItem("Ubuntu")
33.         combo.addItem("Mandriva")
34.         combo.addItem("Fedora")
35.         combo.addItem("Arch")
36.         combo.addItem("Gentoo")
37.
38.         combo.move(50, 50)
39.         self.lbl.move(50, 150)
40.
41.         combo.activated[str].connect(self.onActivated)
42.
43.         self.setGeometry(300, 300, 300, 200)
44.         self.setWindowTitle('QComboBox')
45.         self.show()
46.
47.
48.     def onActivated(self, text):
49.
50.         self.lbl.setText(text)
51.         self.lbl.adjustSize()
52.
53.
54. if __name__ == '__main__':
55.
56.     app = QApplication(sys.argv)
57.     ex = Example()
58.     sys.exit(app.exec_())
```

本例包含了一个 `QComboBox` 和一个 `QLabel` 。下拉选择框有五个选项，都是Linux的发行版名称，标签内容为选定的发行版名称。

```
combo = QComboBox(self)
combo.addItem("Ubuntu")
combo.addItem("Mandriva")
combo.addItem("Fedora")
combo.addItem("Arch")
combo.addItem("Gentoo")
```

创建一个 `QComboBox` 组件和五个选项。

```
combo.activated[str].connect(self.onActivated)
```

在选中的条目上调用 `onActivated()` 方法。

```
def onActivated(self, text):

    self.lbl.setText(text)
    self.lbl.adjustSize()
```

在方法内部，设置标签内容为选定的字符串，然后设置自适应文本大小。

程序展示：

拖拽

- 拖拽
  - 简单的拖放
  - 拖放按钮组件

拖拽

在GUI里，拖放是指用户点击一个虚拟的对象，拖动，然后放置到另外一个对象上面的动作。一般情况下，需要调用很多动作和方法，创建很多变量。

拖放能让用户很直观的操作很复杂的逻辑。

一般情况下，我们可以拖放两种东西：数据和图形界面。把一个图像从一个应用拖放到另外一个应用上的实质是操作二进制数据。把一个表格从Firefox上拖放到另外一个位置 的实质是操作一个图形组。

简单的拖放

本例使用了 `QLineEdit` 和 `QPushButton` 。把一个文本从编辑框里拖到按钮上，更新按钮上的标签（文字）。

```
#!/usr/bin/python3
# -*- coding: utf-8 -*-

"""
ZetCode PyQt5 tutorial

This is a simple drag and
drop example.

Author: Jan Bodnar
```

```
Website: zetcode.com
Last edited: August 2017
"""

from PyQt5.QtWidgets import (QPushButton, QWidget,
    QLineEdit, QApplication)
import sys

class Button(QPushButton):

    def __init__(self, title, parent):
        super().__init__(title, parent)

        self.setAcceptDrops(True)


    def dragEnterEvent(self, e):

        if e.mimeData().hasFormat('text/plain'):
            e.accept()
        else:
            e.ignore()

    def dropEvent(self, e):

        self.setText(e.mimeData().text())


class Example(QWidget):

    def __init__(self):
        super().__init__()

        self.initUI()


    def initUI(self):

```

```
        edit = QLineEdit('', self)
        edit.setDragEnabled(True)
        edit.move(30, 65)

        button = Button("Button", self)
        button.move(190, 65)

        self.setWindowTitle('Simple drag and drop')
        self.setGeometry(300, 300, 300, 150)


if __name__ == '__main__':

    app = QApplication(sys.argv)
    ex = Example()
    ex.show()
    app.exec_()
```

```
class Button(QPushButton):

    def __init__(self, title, parent):
        super().__init__(title, parent)

        self.setAcceptDrops(True)
```

为了完成预定目标，我们要重构一些方法。首先用 `QPushButton` 上构造一个按钮实例。

```
self.setAcceptDrops(True)
```

激活组件的拖拽事件。

```
def dragEnterEvent(self, e):

    if e.mimeData().hasFormat('text/plain'):
        e.accept()
```

```
5.     else:
6.         e.ignore()
```

首先，我们重构了 `dragEnterEvent()` 方法。设定好接受拖拽的数据类型（plain text）。

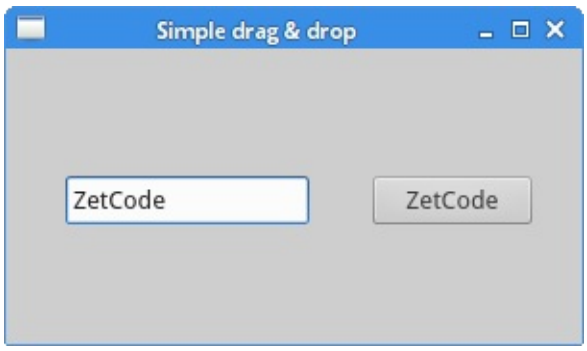
```
1. def dropEvent(self, e):
2.
3.     self.setText(e.mimeData().text())
```

然后重构 `dropEvent()` 方法，更改按钮接受鼠标的释放事件的默认行为。

```
1. edit = QLineEdit('', self)
2. edit.setDragEnabled(True)
```

`QLineEdit` 默认支持拖拽操作，所以我们只要调用 `setDragEnabled()` 方法使用就行了。

程序展示：



拖放按钮组件

这个例子展示怎么拖放一个button组件。

```
1. #!/usr/bin/python3
2. # -*- coding: utf-8 -*-
```

```
3.
4.  """
5.  ZetCode PyQt5 tutorial
6.
7.  In this program, we can press on a button with a left mouse
8.  click or drag and drop the button with  the right mouse click.
9.
10. Author: Jan Bodnar
11. Website: zetcode.com
12. Last edited: August 2017
13. """
14.
15. from PyQt5.QtWidgets import QPushButton, QWidget, QApplication
16. from PyQt5.QtCore import Qt, QMimeData
17. from PyQt5.QtGui import QDrag
18. import sys
19.
20. class Button(QPushButton):
21.
22.     def __init__(self, title, parent):
23.         super().__init__(title, parent)
24.
25.
26.     def mouseMoveEvent(self, e):
27.
28.         if e.buttons() != Qt.RightButton:
29.             return
30.
31.         mimeData = QMimeData()
32.
33.         drag = QDrag(self)
34.         drag.setMimeData(mimeData)
35.         drag.setHotSpot(e.pos() - self.rect().topLeft())
36.
37.         dropAction = drag.exec_(Qt.MoveAction)
38.
39.
40.     def mousePressEvent(self, e):
```

```
41.
42.         super().mousePressEvent(e)
43.
44.         if e.button() == Qt.LeftButton:
45.             print('press')
46.
47.
48. class Example(QWidget):
49.
50.     def __init__(self):
51.         super().__init__()
52.
53.         self.initUI()
54.
55.
56.     def initUI(self):
57.
58.         self.setAcceptDrops(True)
59.
60.         self.button = Button('Button', self)
61.         self.button.move(100, 65)
62.
63.         self.setWindowTitle('Click or Move')
64.         self.setGeometry(300, 300, 280, 150)
65.
66.
67.     def dragEnterEvent(self, e):
68.
69.         e.accept()
70.
71.
72.     def dropEvent(self, e):
73.
74.         position = e.pos()
75.         self.button.move(position)
76.
77.         e.setDropAction(Qt.MoveAction)
78.         e.accept()
```

```
79.
80.
81. if __name__ == '__main__':
82.
83.     app = QApplication(sys.argv)
84.     ex = Example()
85.     ex.show()
86.     app.exec_()
```

上面的例子中，窗口上有一个 `QPushButton` 组件。左键点击按钮，控制台就会输出 `press` 。右键可以点击然后拖动按钮。

```
1. class Button(QPushButton):
2.
3.     def __init__(self, title, parent):
4.         super().__init__(title, parent)
```

从 `QPushButton` 继承一个 `Button` 类，然后重构 `QPushButton` 的两个方法： `mouseMoveEvent()` 和 `mousePressEvent()` . `mouseMoveEvent()` 是拖拽开始的事件。

```
1. if e.buttons() != Qt.RightButton:
2.     return
```

我们只劫持按钮的右键事件，左键的操作还是默认行为。

```
1. mimeData = QMimeData()
2.
3. drag = QDrag(self)
4. drag.setMimeData(mimeData)
5. drag.setHotSpot(e.pos() - self.rect().topLeft())
```

创建一个 `QDrag` 对象，用来传输MIME-based数据。

```
1. dropAction = drag.exec_(Qt.MoveAction)
```

拖放事件开始时，用到的处理函数式 `start()` .

```
def mousePressEvent(self, e):

    QPushButton.mousePressEvent(self, e)

    if e.button() == Qt.LeftButton:
        print('press')
```

左键点击按钮，会在控制台输出“press”。注意，我们在父级上也调用了 `mousePressEvent()` 方法，不然的话，我们是看不到按钮按下的效果的。

```
position = e.pos()
self.button.move(position)
```

在 `dropEvent()` 方法里，我们定义了按钮按下后和释放后的行为，获得鼠标移动的位置，然后把按钮放到这个地方。

```
e.setDropAction(Qt.MoveAction)
e.accept()
```

指定放下的动作类型为moveAction。

程序展示：

这个就一个按钮，没啥可展示的，弄GIF太麻烦了。

绘图

- 绘图
  - 文本涂鸦
  - 点的绘画
- 颜色
  - QPen
  - QBrush
  - 贝塞尔曲线

绘图

PyQt5绘图系统能渲染矢量图像、位图图像和轮廓字体文本。一般会使用在修改或者提高现有组件的功能，或者创建自己的组件。使用PyQt5的绘图API进行操作。

绘图由 `paintEvent()` 方法完成，绘图的代码要放在 `QPainter` 对象的 `begin()` 和 `end()` 方法之间。是低级接口。

文本涂鸦

我们从画一些Unicode文本开始。

```
#!/usr/bin/python3
# -*- coding: utf-8 -*-

"""
ZetCode PyQt5 tutorial

In this example, we draw text in Russian Cylliric.

Author: Jan Bodnar
```

```
Website: zetcode.com
Last edited: August 2017
"""

import sys
from PyQt5.QtWidgets import QWidget, QApplication
from PyQt5.QtGui import QPainter, QColor, QFont
from PyQt5.QtCore import Qt

class Example(QWidget):

    def __init__(self):
        super().__init__()

        self.initUI()


    def initUI(self):

        self.text = "Лев Николаевич Толстой\nАнна Каренина"

        self.setGeometry(300, 300, 280, 170)
        self.setWindowTitle('Drawing text')
        self.show()


    def paintEvent(self, event):

        qp = QPainter()
        qp.begin(self)
        self.drawText(event, qp)
        qp.end()


    def drawText(self, event, qp):

        qp.setPen(QColor(168, 34, 3))
        qp.setFont(QFont('Decorative', 10))
```

```
48.         qp.drawText(event.rect(), Qt.AlignCenter, self.text)
49.
50.
51. if __name__ == '__main__':
52.
53.     app = QApplication(sys.argv)
54.     ex = Example()
55.     sys.exit(app.exec_())
```

写了一些文本上下居中对齐的俄罗斯Cylliric语言的文字。

```
1. def paintEvent(self, event):
2. ...
```

在绘画事件内完成绘画动作。

```
1. qp = QPainter()
2. qp.begin(self)
3. self.drawText(event, qp)
4. qp.end()
```

`QPainter` 是低级的绘画类。所有的绘画动作都在这个类的 `begin()` 和 `end()` 方法之间完成，绘画动作都封装在 `drawText()` 内部了。

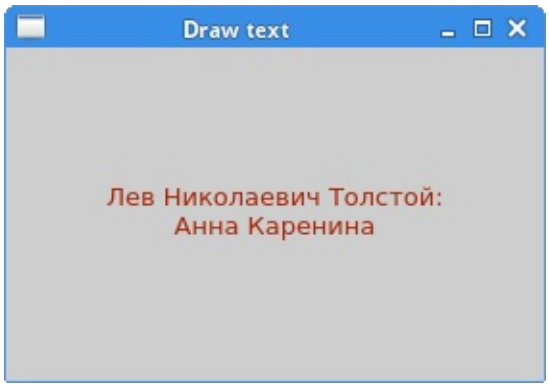
```
1. qp.setPen(QColor(168, 34, 3))
2. qp.setFont(QFont('Decorative', 10))
```

为文字绘画定义了笔和字体。

```
1. qp.drawText(event.rect(), Qt.AlignCenter, self.text)
```

`drawText()` 方法在窗口里绘制文本， `rect()` 方法返回要更新的矩形区域。

程序展示：



点的绘画

点是最简单的绘画了。

```
#!/usr/bin/python3
# -*- coding: utf-8 -*-

"""
ZetCode PyQt5 tutorial

In the example, we draw randomly 1000 red points
on the window.

Author: Jan Bodnar
Website: zetcode.com
Last edited: August 2017
"""

from PyQt5.QtWidgets import QWidget, QApplication
from PyQt5.QtGui import QPainter
from PyQt5.QtCore import Qt
import sys, random

class Example(QWidget):

    def __init__(self):
```

```
        super().__init__()

        self.initUI()


    def initUI(self):

        self.setGeometry(300, 300, 300, 190)
        self.setWindowTitle('Points')
        self.show()


    def paintEvent(self, e):

        qp = QPainter()
        qp.begin(self)
        self.drawPoints(qp)
        qp.end()


    def drawPoints(self, qp):

        qp.setPen(Qt.red)
        size = self.size()

        for i in range(1000):
            x = random.randint(1, size.width()-1)
            y = random.randint(1, size.height()-1)
            qp.drawPoint(x, y)


if __name__ == '__main__':

    app = QApplication(sys.argv)
    ex = Example()
    sys.exit(app.exec_())
```

我们在窗口里随机的画出了1000个点。

```
1.  qp.setPen(Qt.red)
```

设置笔的颜色为红色，使用的是预定义好的颜色。

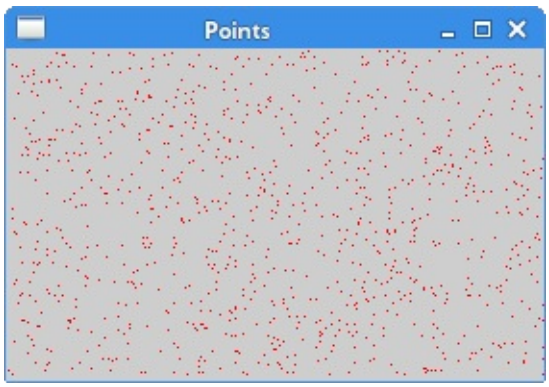
```
1.  size = self.size()
```

每次更改窗口大小，都会产生绘画事件，从 `size()` 方法里获得当前窗口的大小，然后把产生的点随机的分配到窗口的所有位置上。

```
1.  qp.drawPoint(x, y)
```

`drawPoint()` 方法绘图。

程序展示：



颜色

颜色是一个物体显示的RGB的混合色。RBG值的范围是0~255。我们有很多方式去定义一个颜色，最常见的方式就是RGB和16进制表示法，也可以使用RGBA，增加了一个透明度的选项，透明度值的范围是0~1，0代表完全透明。

```
1.  #!/usr/bin/python3
2.  # -*- coding: utf-8 -*-
3.
```

```
"""
ZetCode PyQt5 tutorial

This example draws three rectangles in three
#different colours.

Author: Jan Bodnar
Website: zetcode.com
Last edited: August 2017
"""

from PyQt5.QtWidgets import QWidget, QApplication
from PyQt5.QtGui import QPainter, QColor, QBrush
import sys

class Example(QWidget):

    def __init__(self):
        super().__init__()

        self.initUI()


    def initUI(self):

        self.setGeometry(300, 300, 350, 100)
        self.setWindowTitle('Colours')
        self.show()


    def paintEvent(self, e):

        qp = QPainter()
        qp.begin(self)
        self.drawRectangles(qp)
        qp.end()


```

```
    def drawRectangles(self, qp):

        col = QColor(0, 0, 0)
        col.setNamedColor('#d4d4d4')
        qp.setPen(col)

        qp.setBrush(QColor(200, 0, 0))
        qp.drawRect(10, 15, 90, 60)

        qp.setBrush(QColor(255, 80, 0, 160))
        qp.drawRect(130, 15, 90, 60)

        qp.setBrush(QColor(25, 0, 90, 200))
        qp.drawRect(250, 15, 90, 60)


if __name__ == '__main__':

    app = QApplication(sys.argv)
    ex = Example()
    sys.exit(app.exec_())
```

我们画出了三个颜色的矩形。

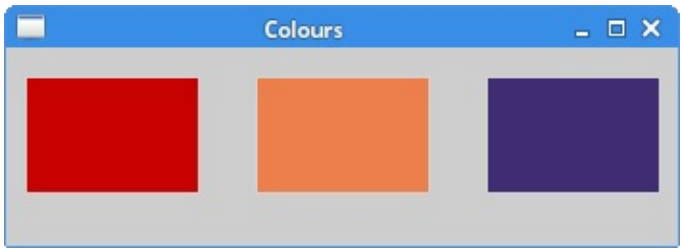
```
color = QColor(0, 0, 0)
color.setNamedColor('#d4d4d4')
```

使用16进制的方式定义一个颜色。

```
qp.setBrush(QColor(200, 0, 0))
qp.drawRect(10, 15, 90, 60)
```

定义了一个笔刷，并画出了一个矩形。笔刷是用来画一个物体的背景。 `drawRect()` 有四个参数，分别是矩形的x、y、w、h。 然后用笔刷和矩形进行绘画。

程序展示：



QPen

`QPen` 是基本的绘画对象，能用来画直线、曲线、矩形框、椭圆、多边形和其他形状。

```
1.  #!/usr/bin/python3
2.  # -*- coding: utf-8 -*-
3.
4.  """
5.  ZetCode PyQt5 tutorial
6.
7.  In this example we draw 6 lines using
8.  different pen styles.
9.
10. Author: Jan Bodnar
11. Website: zetcode.com
12. Last edited: August 2017
13. """
14.
15. from PyQt5.QtWidgets import QWidget, QApplication
16. from PyQt5.QtGui import QPainter, QPen
17. from PyQt5.QtCore import Qt
18. import sys
19.
20. class Example(QWidget):
21.
22.     def __init__(self):
23.         super().__init__()
```

```
24.
25.         self.initUI()
26.
27.
28.     def initUI(self):
29.
30.         self.setGeometry(300, 300, 280, 270)
31.         self.setWindowTitle('Pen styles')
32.         self.show()
33.
34.
35.     def paintEvent(self, e):
36.
37.         qp = QPainter()
38.         qp.begin(self)
39.         self.drawLines(qp)
40.         qp.end()
41.
42.
43.     def drawLines(self, qp):
44.
45.         pen = QPen(Qt.black, 2, Qt.SolidLine)
46.
47.         qp.setPen(pen)
48.         qp.drawLine(20, 40, 250, 40)
49.
50.         pen.setStyle(Qt.DashLine)
51.         qp.setPen(pen)
52.         qp.drawLine(20, 80, 250, 80)
53.
54.         pen.setStyle(Qt.DashDotLine)
55.         qp.setPen(pen)
56.         qp.drawLine(20, 120, 250, 120)
57.
58.         pen.setStyle(Qt.DotLine)
59.         qp.setPen(pen)
60.         qp.drawLine(20, 160, 250, 160)
61.
```

```
        pen.setStyle(Qt.DashDotDotLine)
        qp.setPen(pen)
        qp.drawLine(20, 200, 250, 200)

        pen.setStyle(Qt.CustomDashLine)
        pen.setDashPattern([1, 4, 5, 4])
        qp.setPen(pen)
        qp.drawLine(20, 240, 250, 240)


if __name__ == '__main__':

    app = QApplication(sys.argv)
    ex = Example()
    sys.exit(app.exec_())
```

在这个例子里，我们用不同的笔画了6条直线。PyQt5有五个预定义的笔，另外一个笔的样式使我们自定义的。

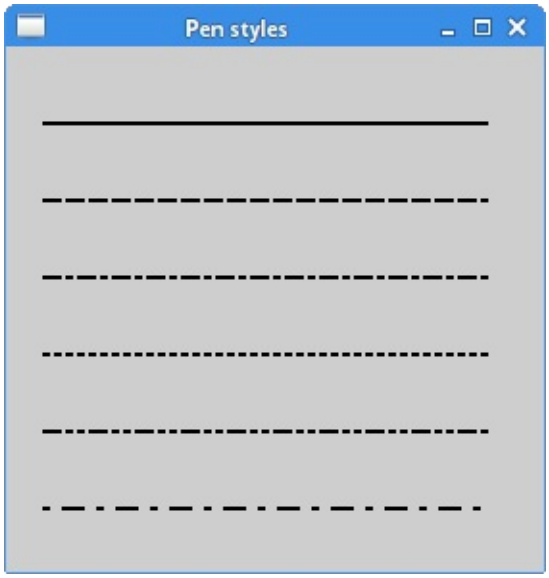
```
pen = QPen(Qt.black, 2, Qt.SolidLine)
```

新建一个 `QPen` 对象，设置颜色黑色，宽2像素，这样就能看出来各个笔样式的区别。 `Qt.SolidLine` 是预定义样式的一种。

```
pen.setStyle(Qt.CustomDashLine)
pen.setDashPattern([1, 4, 5, 4])
qp.setPen(pen)
```

这里我们自定义了一个笔的样式。定义为 `Qt.CustomDashLine` 然后调用 `setDashPattern()` 方法。数字列表是线的样式，要求必须是个数为奇数，奇数位定义的是空格，偶数位为线长，数字越大，空格或线长越大，比如本例的就是1像素线，4像素空格，5像素线，4像素空格。

程序展示：



QBrush

`QBrush` 也是图像的一个基本元素。是用来填充一些物体的背景图用的，比如矩形，椭圆，多边形等。有三种类型：预定义、渐变和纹理。

```
#!/usr/bin/python3
# -*- coding: utf-8 -*-

"""
ZetCode PyQt5 tutorial

This example draws nine rectangles in different
brush styles.

Author: Jan Bodnar
Website: zetcode.com
Last edited: August 2017
"""

from PyQt5.QtWidgets import QWidget, QApplication
from PyQt5.QtGui import QPainter, QBrush
from PyQt5.QtCore import Qt
import sys

```

```
20. class Example(QWidget):
21.
22.     def __init__(self):
23.         super().__init__()
24.
25.         self.initUI()
26.
27.
28.     def initUI(self):
29.
30.         self.setGeometry(300, 300, 355, 280)
31.         self.setWindowTitle('Brushes')
32.         self.show()
33.
34.
35.     def paintEvent(self, e):
36.
37.         qp = QPainter()
38.         qp.begin(self)
39.         self.drawBrushes(qp)
40.         qp.end()
41.
42.
43.     def drawBrushes(self, qp):
44.
45.         brush = QBrush(Qt.SolidPattern)
46.         qp.setBrush(brush)
47.         qp.drawRect(10, 15, 90, 60)
48.
49.         brush.setStyle(Qt.Dense1Pattern)
50.         qp.setBrush(brush)
51.         qp.drawRect(130, 15, 90, 60)
52.
53.         brush.setStyle(Qt.Dense2Pattern)
54.         qp.setBrush(brush)
55.         qp.drawRect(250, 15, 90, 60)
56.
57.         brush.setStyle(Qt.DiagCrossPattern)
```

```
58.         qp.setBrush(brush)
59.         qp.drawRect(10, 105, 90, 60)
60.
61.         brush.setStyle(Qt.Dense5Pattern)
62.         qp.setBrush(brush)
63.         qp.drawRect(130, 105, 90, 60)
64.
65.         brush.setStyle(Qt.Dense6Pattern)
66.         qp.setBrush(brush)
67.         qp.drawRect(250, 105, 90, 60)
68.
69.         brush.setStyle(Qt.HorPattern)
70.         qp.setBrush(brush)
71.         qp.drawRect(10, 195, 90, 60)
72.
73.         brush.setStyle(Qt.VerPattern)
74.         qp.setBrush(brush)
75.         qp.drawRect(130, 195, 90, 60)
76.
77.         brush.setStyle(Qt.BDiagPattern)
78.         qp.setBrush(brush)
79.         qp.drawRect(250, 195, 90, 60)
80.
81.
82. if __name__ == '__main__':
83.
84.     app = QApplication(sys.argv)
85.     ex = Example()
86.     sys.exit(app.exec_())
```

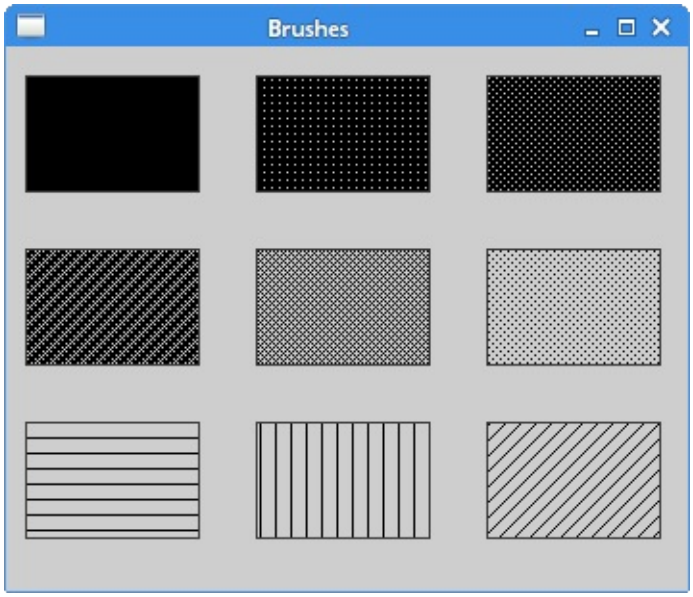
我们画了9个不同的矩形。

```
1. brush = QBrush(Qt.SolidPattern)
2. qp.setBrush(brush)
3. qp.drawRect(10, 15, 90, 60)
```

创建了一个笔刷对象，添加笔刷样式，然后调用 `drawRect()` 方法画

图。

程序展示：



贝塞尔曲线

噩梦可以使用PyQt5的 `QPainterPath` 创建贝塞尔曲线。绘画路径是由许多构建图形的对象，具体表现就是一些线的形状，比如矩形，椭圆，线和曲线。

```
1.
2.  #!/usr/bin/python3
3.  # -*- coding: utf-8 -*-
4.
5.  """
6.  ZetCode PyQt5 tutorial
7.
8.  This program draws a Bézier curve with
9.  QPainterPath.
10.
11.  Author: Jan Bodnar
12.  Website: zetcode.com
13.  Last edited: August 2017
```

```
14. """
15.
16. from PyQt5.QtWidgets import QWidget, QApplication
17. from PyQt5.QtGui import QPainter, QPainterPath
18. from PyQt5.QtCore import Qt
19. import sys
20.
21. class Example(QWidget):
22.
23.     def __init__(self):
24.         super().__init__()
25.
26.         self.initUI()
27.
28.
29.     def initUI(self):
30.
31.         self.setGeometry(300, 300, 380, 250)
32.         self.setWindowTitle('Bézier curve')
33.         self.show()
34.
35.
36.     def paintEvent(self, e):
37.
38.         qp = QPainter()
39.         qp.begin(self)
40.         qp.setRenderHint(QPainter.Antialiasing)
41.         self.drawBezierCurve(qp)
42.         qp.end()
43.
44.
45.     def drawBezierCurve(self, qp):
46.
47.         path = QPainterPath()
48.         path.moveTo(30, 30)
49.         path.cubicTo(30, 30, 200, 350, 350, 30)
50.
51.         qp.drawPath(path)
```

```
52.
53.
54. if __name__ == '__main__':
55.
56.     app = QApplication(sys.argv)
57.     ex = Example()
58.     sys.exit(app.exec_())
```

这个示例中，我们画出了一个贝塞尔曲线。

```
1. path = QPainterPath()
2. path.moveTo(30, 30)
3. path.cubicTo(30, 30, 200, 350, 350, 30)
```

用 `QPainterPath` 路径创建贝塞尔曲线。使用 `cubicTo()` 方法生成，分别需要三个点：起始点，控制点和终止点。

```
1. qp.drawPath(path)
```

`drawPath()` 绘制最后的图像。

程序展示：

自定义组件

- 自定义控件
  - Burning widget

自定义控件

PyQt5有丰富的组件，但是肯定满足不了所有开发者的所有需求，PyQt5只提供了基本的组件，像按钮，文本，滑块等。如果你还需要其他的模块，应该尝试自己去自定义一些。

自定义组件使用绘画工具创建，有两个基本方式：根据已有的创建或改进；通过自己绘图创建。

Burning widget

这个组件我们会在Nero，K3B，或者其他CD/DVD烧录软件中见到。

```
#!/usr/bin/python3
# -*- coding: utf-8 -*-

"""
ZetCode PyQt5 tutorial

In this example, we create a custom widget.

Author: Jan Bodnar
Website: zetcode.com
Last edited: August 2017
"""

from PyQt5.QtWidgets import (QWidget, QSlider, QApplication,
    QHBoxLayout, QVBoxLayout)
from PyQt5.QtCore import QObject, Qt, pyqtSignal
```

```
17. from PyQt5.QtGui import QPainter, QFont, QColor, QPen
18. import sys
19. 
20. class Communicate(QObject):
21. 
22.     updateBW = pyqtSignal(int)
23. 
24. 
25. class BurningWidget(QWidget):
26. 
27.     def __init__(self):
28.         super().__init__()
29. 
30.         self.initUI()
31. 
32. 
33.     def initUI(self):
34. 
35.         self.setMinimumSize(1, 30)
36.         self.value = 75
37.         self.num = [75, 150, 225, 300, 375, 450, 525, 600, 675]
38. 
39. 
40.     def setValue(self, value):
41. 
42.         self.value = value
43. 
44. 
45.     def paintEvent(self, e):
46. 
47.         qp = QPainter()
48.         qp.begin(self)
49.         self.drawWidget(qp)
50.         qp.end()
51. 
52. 
53.     def drawWidget(self, qp):
54. 
```

```
        MAX_CAPACITY = 700
        OVER_CAPACITY = 750

        font = QFont('Serif', 7, QFont.Light)
        qp.setFont(font)

        size = self.size()
        w = size.width()
        h = size.height()

        step = int(round(w / 10))


        till = int(((w / OVER_CAPACITY) * self.value))
        full = int(((w / OVER_CAPACITY) * MAX_CAPACITY))

        if self.value >= MAX_CAPACITY:

            qp.setPen(QColor(255, 255, 255))
            qp.setBrush(QColor(255, 255, 184))
            qp.drawRect(0, 0, full, h)
            qp.setPen(QColor(255, 175, 175))
            qp.setBrush(QColor(255, 175, 175))
            qp.drawRect(full, 0, till-full, h)

        else:

            qp.setPen(QColor(255, 255, 255))
            qp.setBrush(QColor(255, 255, 184))
            qp.drawRect(0, 0, till, h)


        pen = QPen(QColor(20, 20, 20), 1,
            Qt.SolidLine)

        qp.setPen(pen)
        qp.setBrush(Qt.NoBrush)
        qp.drawRect(0, 0, w-1, h-1)
```

```
93.
94.         j = 0
95.
96.         for i in range(step, 10*step, step):
97.
98.             qp.drawLine(i, 0, i, 5)
99.             metrics = qp.fontMetrics()
100.            fw = metrics.width(str(self.num[j]))
101.            qp.drawText(i-fw/2, h/2, str(self.num[j]))
102.            j = j + 1
103.
104.
105. class Example(QWidget):
106.
107.     def __init__(self):
108.         super().__init__()
109.
110.         self.initUI()
111.
112.
113.     def initUI(self):
114.
115.         OVER_CAPACITY = 750
116.
117.         sld = QSlider(Qt.Horizontal, self)
118.         sld.setFocusPolicy(Qt.NoFocus)
119.         sld.setRange(1, OVER_CAPACITY)
120.         sld.setValue(75)
121.         sld.setGeometry(30, 40, 150, 30)
122.
123.         self.c = Communicate()
124.         self.wid = BurningWidget()
125.         self.c.updateBW[int].connect(self.wid.setValue)
126.
127.         sld.valueChanged[int].connect(self.changeValue)
128.         hbox = QHBoxLayout()
129.         hbox.addWidget(self.wid)
130.         vbox = QVBoxLayout()
```

```
131.         vbox.addStretch(1)
132.         vbox.addLayout(hbox)
133.         self.setLayout(vbox)
134.
135.         self.setGeometry(300, 300, 390, 210)
136.         self.setWindowTitle('Burning widget')
137.         self.show()
138.
139.
140.     def changeValue(self, value):
141.
142.         self.c.updateBW.emit(value)
143.         self.wid.repaint()
144.
145.
146. if __name__ == '__main__':
147.
148.     app = QApplication(sys.argv)
149.     ex = Example()
150.     sys.exit(app.exec_())
```

本例中，我们使用了 `QSlider` 和一个自定义组件，由进度条控制。显示的有物体（也就是CD/DVD）的总容量和剩余容量。进度条的范围是1~750。如果值达到了700（OVER_CAPACITY），就显示为红色，代表了烧毁了的意思。

烧录组件在窗口的底部，这个组件是用 `QHBoxLayout` 和 `QVBoxLayout` 组成的。

```
1. class BurningWidget(QWidget):
2.
3.     def __init__(self):
4.         super().__init__()
```

基于 `QWidget` 组件。

```
self.setMinimumSize(1, 30)
```

修改组件进度条的高度，默认的有点小。

```
font = QFont('Serif', 7, QFont.Light)
qp.setFont(font)
```

使用比默认更小一点的字体，这样更配。

```
size = self.size()
w = size.width()
h = size.height()

step = int(round(w / 10.0))


till = int(((w / 750.0) * self.value))
full = int(((w / 750.0) * 700))
```

动态的渲染组件，随着窗口的大小而变化，这就是我们计算窗口大小的原因。最后一个参数决定了组件的最大范围，进度条的值是由窗口大小按比例计算出来的。最大值的地方填充的是红色。注意这里使用的是浮点数，能提高计算和渲染的精度。

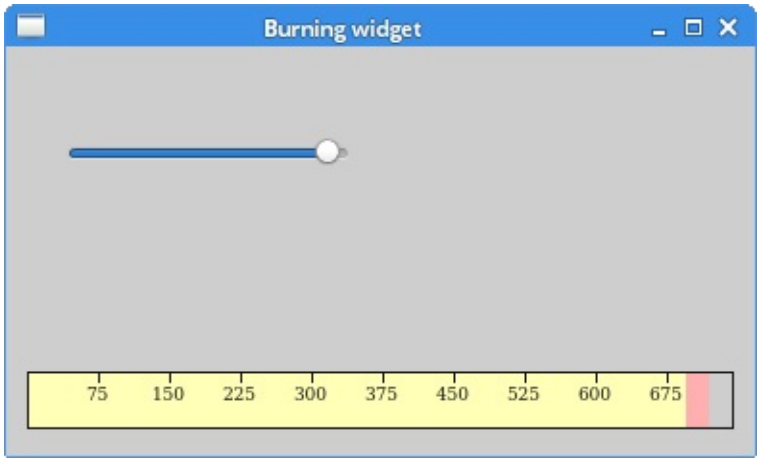
绘画由三部分组成，黄色或红色区域和黄色矩形，然后是分割线，最后是添上代表容量的数字。

```
metrics = qp.fontMetrics()
fw = metrics.width(str(self.num[j]))
qp.drawText(i-fw/2, h/2, str(self.num[j]))
```

这里使用字体去渲染文本。必须要知道文本的宽度，这样才能让文本的中间点正好落在竖线上。

```
1.  def changeValue(self, value):
2.
3.      self.c.updateBW.emit(value)
4.      self.wid.repaint()
```

拖动滑块的时候，调用了 `changeValue()` 方法。这个方法内部，我们自定义了一个可以传参的updateBW信号。参数就是滑块的当前位置。这个数值之后还用来于Burning组件，然后重新渲染Burning组件。



俄罗斯方块游戏

- 俄罗斯方块游戏
  - Tetris
  - 开发

俄罗斯方块游戏

本章我们要制作一个俄罗斯方块游戏。

Tetris

> 译注：称呼：方块是由四个小方格组成的

俄罗斯方块游戏是世界上最流行的游戏之一。是由一名叫Alexey Pajitnov的俄罗斯程序员在1985年制作的，从那时起，这个游戏就风靡了各个游戏平台。

俄罗斯方块归类为下落块迷宫游戏。游戏有7个基本形状：S、Z、T、L、反向L、直线、方块，每个形状都由4个方块组成，方块最终都会落到屏幕底部。所以玩家通过控制形状的左右位置和旋转，让每个形状都以合适的位置落下，如果有一行全部被方块填充，这行就会消失，并且得分。游戏结束的条件是有形状接触到了屏幕顶部。

方块展示：

PyQt5是专门为创建图形界面产生的，里面一些专门为制作游戏而开发的组件，所以PyQt5是能制作小游戏的。

制作电脑游戏也是提高自己编程能力的一种很好的方式。

开发

没有图片，所以就自己用绘画画出来几个图形。每个游戏里都有数学模型的，这个也是。

开工之前：

- 用`QtCore.QBasicTimer()`创建一个游戏循环
- 模型是一直下落的
- 模型的运动是以小块为基础单位的，不是按像素
- 从数学意义上来说，模型就是就是一串数字而已

代码由四个类组成：Tetris, Board, Tetrominoe和Shape。Tetris类创建游戏，Board是游戏主要逻辑。Tetrominoe包含了所有的砖块，Shape是所有砖块的代码。

```
#!/usr/bin/python3
# -*- coding: utf-8 -*-

"""
ZetCode PyQt5 tutorial

This is a Tetris game clone.

Author: Jan Bodnar
Website: zetcode.com
Last edited: August 2017
"""

from PyQt5.QtWidgets import QMainWindow, QFrame, QDesktopWidget,
```

```
    QApplication
15. from PyQt5.QtCore import Qt, QBasicTimer, pyqtSignal
16. from PyQt5.QtGui import QPainter, QColor
17. import sys, random
18.
19. class Tetris(QMainWindow):
20.
21.     def __init__(self):
22.         super().__init__()
23.
24.         self.initUI()
25.
26.
27.     def initUI(self):
28.         '''initiates application UI'''
29.
30.         self.tboard = Board(self)
31.         self.setCentralWidget(self.tboard)
32.
33.         self.statusbar = self.statusBar()
34.
    self.tboard.msg2Statusbar[str].connect(self.statusbar.showMessage)
35.
36.         self.tboard.start()
37.
38.         self.resize(180, 380)
39.         self.center()
40.         self.setWindowTitle('Tetris')
41.         self.show()
42.
43.
44.     def center(self):
45.         '''centers the window on the screen'''
46.
47.         screen = QDesktopWidget().screenGeometry()
48.         size = self.geometry()
49.         self.move((screen.width()-size.width())/2,
50.             (screen.height()-size.height())/2)
```

```
51.
52.
53. class Board(QFrame):
54.
55.     msg2Statusbar = pyqtSignal(str)
56.
57.     BoardWidth = 10
58.     BoardHeight = 22
59.     Speed = 300
60.
61.     def __init__(self, parent):
62.         super().__init__(parent)
63.
64.         self.initBoard()
65.
66.
67.     def initBoard(self):
68.         '''initiates board'''
69.
70.         self.timer = QBasicTimer()
71.         self.isWaitingAfterLine = False
72.
73.         self.curX = 0
74.         self.curY = 0
75.         self.numLinesRemoved = 0
76.         self.board = []
77.
78.         self.setFocusPolicy(Qt.StrongFocus)
79.         self.isStarted = False
80.         self.isPaused = False
81.         self.clearBoard()
82.
83.
84.     def shapeAt(self, x, y):
85.         '''determines shape at the board position'''
86.
87.         return self.board[(y * Board.BoardWidth) + x]
88.
```

```
89.
90.     def setShapeAt(self, x, y, shape):
91.         '''sets a shape at the board'''
92.
93.         self.board[(y * Board.BoardWidth) + x] = shape
94.
95.
96.     def squareWidth(self):
97.         '''returns the width of one square'''
98.
99.         return self.contentsRect().width() // Board.BoardWidth
100.
101.
102.     def squareHeight(self):
103.         '''returns the height of one square'''
104.
105.         return self.contentsRect().height() // Board.BoardHeight
106.
107.
108.     def start(self):
109.         '''starts game'''
110.
111.         if self.isPaused:
112.             return
113.
114.         self.isStarted = True
115.         self.isWaitingAfterLine = False
116.         self.numLinesRemoved = 0
117.         self.clearBoard()
118.
119.         self.msg2Statusbar.emit(str(self.numLinesRemoved))
120.
121.         self.newPiece()
122.         self.timer.start(Board.Speed, self)
123.
124.
125.     def pause(self):
126.         '''pauses game'''
```

```
127.
128.         if not self.isStarted:
129.             return
130.
131.         self.isPaused = not self.isPaused
132.
133.         if self.isPaused:
134.             self.timer.stop()
135.             self.msg2Statusbar.emit("paused")
136.
137.         else:
138.             self.timer.start(Board.Speed, self)
139.             self.msg2Statusbar.emit(str(self.numLinesRemoved))
140.
141.         self.update()
142.
143.
144.     def paintEvent(self, event):
145.         '''paints all shapes of the game'''
146.
147.         painter = QPainter(self)
148.         rect = self.contentsRect()
149.
150.         boardTop = rect.bottom() - Board.BoardHeight *
     self.squareHeight()
151.
152.         for i in range(Board.BoardHeight):
153.             for j in range(Board.BoardWidth):
154.                 shape = self.shapeAt(j, Board.BoardHeight - i - 1)
155.
156.                 if shape != Tetrominoe.NoShape:
157.                     self.drawSquare(painter,
158.                         rect.left() + j * self.squareWidth(),
159.                         boardTop + i * self.squareHeight(), shape)
160.
161.         if self.curPiece.shape() != Tetrominoe.NoShape:
162.
163.             for i in range(4):
```

```
164.
165.                 x = self.curX + self.curPiece.x(i)
166.                 y = self.curY - self.curPiece.y(i)
167.                 self.drawSquare(painter, rect.left() + x *
    self.squareWidth(),
168.                     boardTop + (Board.BoardHeight - y - 1) *
    self.squareHeight(),
169.                     self.curPiece.shape())
170.
171.
172.     def keyPressEvent(self, event):
173.         '''processes key press events'''
174.
175.         if not self.isStarted or self.curPiece.shape() ==
    Tetrominoe.NoShape:
176.             super(Board, self).keyPressEvent(event)
177.             return
178.
179.         key = event.key()
180.
181.         if key == Qt.Key_P:
182.             self.pause()
183.             return
184.
185.         if self.isPaused:
186.             return
187.
188.         elif key == Qt.Key_Left:
189.             self.tryMove(self.curPiece, self.curX - 1, self.curY)
190.
191.         elif key == Qt.Key_Right:
192.             self.tryMove(self.curPiece, self.curX + 1, self.curY)
193.
194.         elif key == Qt.Key_Down:
195.             self.tryMove(self.curPiece.rotateRight(), self.curX,
    self.curY)
196.
197.         elif key == Qt.Key_Up:
```

```
198.             self.tryMove(self.curPiece.rotateLeft(), self.curX,
    self.curY)
199.
200.         elif key == Qt.Key_Space:
201.             self.dropDown()
202.
203.         elif key == Qt.Key_D:
204.             self.oneLineDown()
205.
206.         else:
207.             super(Board, self).keyPressEvent(event)
208.
209.
210.     def timerEvent(self, event):
211.         '''handles timer event'''
212.
213.         if event.timerId() == self.timer.timerId():
214.
215.             if self.isWaitingAfterLine:
216.                 self.isWaitingAfterLine = False
217.                 self.newPiece()
218.             else:
219.                 self.oneLineDown()
220.
221.         else:
222.             super(Board, self).timerEvent(event)
223.
224.
225.     def clearBoard(self):
226.         '''clears shapes from the board'''
227.
228.         for i in range(Board.BoardHeight * Board.BoardWidth):
229.             self.board.append(Tetrominoe.NoShape)
230.
231.
232.     def dropDown(self):
233.         '''drops down a shape'''
234.
```

```
235.         newY = self.curY
236.
237.         while newY > 0:
238.
239.             if not self.tryMove(self.curPiece, self.curX, newY -
     1):
240.                 break
241.
242.             newY -= 1
243.
244.         self.pieceDropped()
245.
246.
247.     def oneLineDown(self):
248.         '''goes one line down with a shape'''
249.
250.         if not self.tryMove(self.curPiece, self.curX, self.curY -
     1):
251.             self.pieceDropped()
252.
253.
254.     def pieceDropped(self):
255.         '''after dropping shape, remove full lines and create new
     shape'''
256.
257.         for i in range(4):
258.
259.             x = self.curX + self.curPiece.x(i)
260.             y = self.curY - self.curPiece.y(i)
261.             self.setShapeAt(x, y, self.curPiece.shape())
262.
263.         self.removeFullLines()
264.
265.         if not self.isWaitingAfterLine:
266.             self.newPiece()
267.
268.
269.     def removeFullLines(self):
```

```
270.         '''removes all full lines from the board'''
271.
272.         numFullLines = 0
273.         rowsToRemove = []
274.
275.         for i in range(Board.BoardHeight):
276.
277.             n = 0
278.             for j in range(Board.BoardWidth):
279.                 if not self.shapeAt(j, i) == Tetrominoe.NoShape:
280.                     n = n + 1
281.
282.             if n == 10:
283.                 rowsToRemove.append(i)
284.
285.         rowsToRemove.reverse()
286.
287.
288.         for m in rowsToRemove:
289.
290.             for k in range(m, Board.BoardHeight):
291.                 for l in range(Board.BoardWidth):
292.                         self.setShapeAt(l, k, self.shapeAt(l, k +
    1))
293.
294.         numFullLines = numFullLines + len(rowsToRemove)
295.
296.         if numFullLines > 0:
297.
298.             self.numLinesRemoved = self.numLinesRemoved +
    numFullLines
299.             self.msg2Statusbar.emit(str(self.numLinesRemoved))
300.
301.             self.isWaitingAfterLine = True
302.             self.curPiece.setShape(Tetrominoe.NoShape)
303.             self.update()
304.
305.
```

```
306.     def newPiece(self):
307.         '''creates a new shape'''
308.
309.         self.curPiece = Shape()
310.         self.curPiece.setRandomShape()
311.         self.curX = Board.BoardWidth // 2 + 1
312.         self.curY = Board.BoardHeight - 1 + self.curPiece.minY()
313.
314.         if not self.tryMove(self.curPiece, self.curX, self.curY):
315.
316.             self.curPiece.setShape(Tetrominoe.NoShape)
317.             self.timer.stop()
318.             self.isStarted = False
319.             self.msg2Statusbar.emit("Game over")
320.
321.
322.
323.     def tryMove(self, newPiece, newX, newY):
324.         '''tries to move a shape'''
325.
326.         for i in range(4):
327.
328.             x = newX + newPiece.x(i)
329.             y = newY - newPiece.y(i)
330.
331.             if x < 0 or x >= Board.BoardWidth or y < 0 or y >=
    Board.BoardHeight:
332.                 return False
333.
334.             if self.shapeAt(x, y) != Tetrominoe.NoShape:
335.                 return False
336.
337.         self.curPiece = newPiece
338.         self.curX = newX
339.         self.curY = newY
340.         self.update()
341.
342.         return True
```

```
343.
344.
345.     def drawSquare(self, painter, x, y, shape):
346.         '''draws a square of a shape'''
347.
348.         colorTable = [0x000000, 0xCC6666, 0x66CC66, 0x6666CC,
349.                       0xCCCC66, 0xCC66CC, 0x66CCCC, 0xDAAA00]
350.
351.         color = QColor(colorTable[shape])
352.         painter.fillRect(x + 1, y + 1, self.squareWidth() - 2,
353.             self.squareHeight() - 2, color)
354.
355.         painter.setPen(color.lighter())
356.         painter.drawLine(x, y + self.squareHeight() - 1, x, y)
357.         painter.drawLine(x, y, x + self.squareWidth() - 1, y)
358.
359.         painter.setPen(color.darker())
360.         painter.drawLine(x + 1, y + self.squareHeight() - 1,
361.             x + self.squareWidth() - 1, y + self.squareHeight() -
    1)
362.         painter.drawLine(x + self.squareWidth() - 1,
363.             y + self.squareHeight() - 1, x + self.squareWidth() -
    1, y + 1)
364.
365.
366. class Tetrominoe(object):
367.
368.     NoShape = 0
369.     ZShape = 1
370.     SShape = 2
371.     LineShape = 3
372.     TShape = 4
373.     SquareShape = 5
374.     LShape = 6
375.     MirroredLShape = 7
376.
377.
378. class Shape(object):
```

```
379.
380.     coordsTable = (
381.         ((0, 0),     (0, 0),     (0, 0),     (0, 0)),
382.         ((0, -1),    (0, 0),     (-1, 0),    (-1, 1)),
383.         ((0, -1),    (0, 0),     (1, 0),     (1, 1)),
384.         ((0, -1),    (0, 0),     (0, 1),     (0, 2)),
385.         ((-1, 0),    (0, 0),     (1, 0),     (0, 1)),
386.         ((0, 0),     (1, 0),     (0, 1),     (1, 1)),
387.         ((-1, -1),   (0, -1),    (0, 0),     (0, 1)),
388.         ((1, -1),    (0, -1),    (0, 0),     (0, 1))
389.     )
390.
391.     def __init__(self):
392.
393.         self.coords = [[0,0] for i in range(4)]
394.         self.pieceShape = Tetrominoe.NoShape
395.
396.         self.setShape(Tetrominoe.NoShape)
397.
398.
399.     def shape(self):
400.         '''returns shape'''
401.
402.         return self.pieceShape
403.
404.
405.     def setShape(self, shape):
406.         '''sets a shape'''
407.
408.         table = Shape.coordsTable[shape]
409.
410.         for i in range(4):
411.             for j in range(2):
412.                 self.coords[i][j] = table[i][j]
413.
414.         self.pieceShape = shape
415.
416.
```

```
417.      def setRandomShape(self):
418.          '''chooses a random shape'''
419.
420.          self.setShape(random.randint(1, 7))
421.
422.
423.      def x(self, index):
424.          '''returns x coordinate'''
425.
426.          return self.coords[index][0]
427.
428.
429.      def y(self, index):
430.          '''returns y coordinate'''
431.
432.          return self.coords[index][1]
433.
434.
435.      def setX(self, index, x):
436.          '''sets x coordinate'''
437.
438.          self.coords[index][0] = x
439.
440.
441.      def setY(self, index, y):
442.          '''sets y coordinate'''
443.
444.          self.coords[index][1] = y
445.
446.
447.      def minX(self):
448.          '''returns min x value'''
449.
450.          m = self.coords[0][0]
451.          for i in range(4):
452.              m = min(m, self.coords[i][0])
453.
454.          return m
```

```
455.
456.
457.      def maxX(self):
458.          '''returns max x value'''
459.
460.          m = self.coords[0][0]
461.          for i in range(4):
462.              m = max(m, self.coords[i][0])
463.
464.          return m
465.
466.
467.      def minY(self):
468.          '''returns min y value'''
469.
470.          m = self.coords[0][1]
471.          for i in range(4):
472.              m = min(m, self.coords[i][1])
473.
474.          return m
475.
476.
477.      def maxY(self):
478.          '''returns max y value'''
479.
480.          m = self.coords[0][1]
481.          for i in range(4):
482.              m = max(m, self.coords[i][1])
483.
484.          return m
485.
486.
487.      def rotateLeft(self):
488.          '''rotates shape to the left'''
489.
490.          if self.pieceShape == Tetrominoe.SquareShape:
491.              return self
492.
```

```
493.          result = Shape()
494.          result.pieceShape = self.pieceShape
495.
496.          for i in range(4):
497.
498.              result.setX(i, self.y(i))
499.              result.setY(i, -self.x(i))
500.
501.          return result
502.
503.
504.      def rotateRight(self):
505.          '''rotates shape to the right'''
506.
507.          if self.pieceShape == Tetrominoe.SquareShape:
508.              return self
509.
510.          result = Shape()
511.          result.pieceShape = self.pieceShape
512.
513.          for i in range(4):
514.
515.              result.setX(i, -self.y(i))
516.              result.setY(i, self.x(i))
517.
518.          return result
519.
520.
521.  if __name__ == '__main__':
522.
523.      app = QApplication([])
524.      tetris = Tetris()
525.      sys.exit(app.exec_())
```

游戏很简单，所以也就很好理解。程序加载之后游戏也就直接开始了，可以用P键暂停游戏，空格键让方块直接落到最下面。游戏的速度是固定的，并没有实现加速的功能。分数就是游戏中消除的行数。

```
self.tboard = Board(self)
self.setCentralWidget(self.tboard)
```

创建了一个Board类的实例，并设置为应用的中心组件。

```
self.statusbar = self.statusBar()
self.tboard.msg2Statusbar[str].connect(self.statusbar.showMessage)
```

创建一个 `statusbar` 来显示三种信息：消除的行数，游戏暂停状态或者游戏结束状态。 `msg2Statusbar` 是一个自定义的信号，用在（和）Board类（交互），`showMessage()` 方法是一个内建的，用来在statusbar上显示信息的方法。

```
self.tboard.start()
```

初始化游戏：

```
class Board(QFrame):

    msg2Statusbar = pyqtSignal(str)
...
```

创建了一个自定义信号 `msg2Statusbar` ，当我们想往 `statusbar` 里显示信息的时候，发出这个信号就行了。

```
BoardWidth = 10
BoardHeight = 22
Speed = 300
```

这些是 `Board` 类的变量。 `BoardWidth` 和 `BoardHeight` 分别是board的宽度和高度。 `Speed` 是游戏的速度，每300ms出现一个新的方块。

```
...
self.curX = 0
```

```
3.  self.curY = 0
4.  self.numLinesRemoved = 0
5.  self.board = []
6.  ...
```

在 `initBoard()` 里初始化了一些重要的变量。 `self.board` 定义了方块的形状和位置，取值范围是0-7。

```
1.  def shapeAt(self, x, y):
2.      return self.board[(y * Board.BoardWidth) + x]
```

`shapeAt()` 决定了board里方块的的种类。

```
1.  def squareWidth(self):
2.      return self.contentsRect().width() // Board.BoardWidth
```

board的大小可以动态的改变。所以方格的大小也应该随之变化。 `squareWidth()` 计算并返回每个块应该占用多少像素—也即 `Board.BoardWidth` 。

```
1.  def pause(self):
2.      '''pauses game'''
3.
4.      if not self.isStarted:
5.          return
6.
7.      self.isPaused = not self.isPaused
8.
9.      if self.isPaused:
10.         self.timer.stop()
11.         self.msg2Statusbar.emit("paused")
12.
13.     else:
14.         self.timer.start(Board.Speed, self)
15.         self.msg2Statusbar.emit(str(self.numLinesRemoved))
16.
```

```
17.     self.update()
```

`pause()`方法用来暂停游戏，停止计时并在`statusbar`上显示一条信息。

```
1.  def paintEvent(self, event):
2.      '''paints all shapes of the game'''
3.
4.      painter = QPainter(self)
5.      rect = self.contentsRect()
6.  ...
```

渲染是在paintEvent()方法里发生的`QPainter`负责PyQt5里所有低级绘画操作。

```
1.  for i in range(Board.BoardHeight):
2.      for j in range(Board.BoardWidth):
3.          shape = self.shapeAt(j, Board.BoardHeight - i - 1)
4.
5.          if shape != Tetrominoe.NoShape:
6.              self.drawSquare(painter,
7.                  rect.left() + j * self.squareWidth(),
8.                  boardTop + i * self.squareHeight(), shape)
```

渲染游戏分为两步。第一步是先画出所有已经落在最下面的的图，这些保存在`self.board`里。可以使用`shapeAt()`查看这个这个变量。

```
1.  if self.curPiece.shape() != Tetrominoe.NoShape:
2.
3.      for i in range(4):
4.
5.          x = self.curX + self.curPiece.x(i)
6.          y = self.curY - self.curPiece.y(i)
7.          self.drawSquare(painter, rect.left() + x *
    self.squareWidth(),
8.              boardTop + (Board.BoardHeight - y - 1) *
```

```
    self.squareHeight(),
9.              self.curPiece.shape())
```

第二步是画出更在下落的方块。

```
1.  elif key == Qt.Key_Right:
2.      self.tryMove(self.curPiece, self.curX + 1, self.curY)
```

在 `keyPressEvent()` 方法获得用户按下的按键。如果按下的是右方向键，就尝试把方块向右移动，说尝试是因为有可能到边界不能移动了。

```
1.  elif key == Qt.Key_Up:
2.      self.tryMove(self.curPiece.rotateLeft(), self.curX, self.curY)
```

上方向键是把方块向左旋转一下

```
1.  elif key == Qt.Key_Space:
2.      self.dropDown()
```

空格键会直接把方块放到底部

```
1.  elif key == Qt.Key_D:
2.      self.oneLineDown()
```

D键是加速一次下落速度。

```
1.  def tryMove(self, newPiece, newX, newY):
2.
3.      for i in range(4):
4.
5.          x = newX + newPiece.x(i)
6.          y = newY - newPiece.y(i)
7.
8.          if x < 0 or x >= Board.BoardWidth or y < 0 or y >=
    Board.BoardHeight:
```

```
            return False

        if self.shapeAt(x, y) != Tetrominoe.NoShape:
            return False

    self.curPiece = newPiece
    self.curX = newX
    self.curY = newY
    self.update()
    return True
```

`tryMove()` 是尝试移动方块的方法。如果方块已经到达board的边缘或者遇到了其他方块，就返回False。否则就把方块下落到想要

```
def timerEvent(self, event):

    if event.timerId() == self.timer.timerId():

        if self.isWaitingAfterLine:
            self.isWaitingAfterLine = False
            self.newPiece()
        else:
            self.oneLineDown()

    else:
        super(Board, self).timerEvent(event)
```

在计时器事件里，要么是等一个方块下落完之后创建一个新的方块，要么是让一个方块直接落到底（move a falling piece one line down）。

```
def clearBoard(self):

    for i in range(Board.BoardHeight * Board.BoardWidth):
        self.board.append(Tetrominoe.NoShape)
```

`clearBoard(`)方法通过 `Tetrominoe.NoShape` 清空 `broad` 。

```
1.  def removeFullLines(self):
2.
3.      numFullLines = 0
4.      rowsToRemove = []
5.
6.      for i in range(Board.BoardHeight):
7.
8.          n = 0
9.          for j in range(Board.BoardWidth):
10.             if not self.shapeAt(j, i) == Tetrominoe.NoShape:
11.                 n = n + 1
12.
13.         if n == 10:
14.             rowsToRemove.append(i)
15.
16.     rowsToRemove.reverse()
17.
18.
19.     for m in rowsToRemove:
20.
21.         for k in range(m, Board.BoardHeight):
22.             for l in range(Board.BoardWidth):
23.                     self.setShapeAt(l, k, self.shapeAt(l, k + 1))
24.
25.     numFullLines = numFullLines + len(rowsToRemove)
26.   ...
```

如果方块碰到了底部，就调用 `removeFullLines()` 方法，找到所有能消除的行消除它们。消除的具体动作就是把符合条件的行消除掉之后，再把它上面的行下降一行。注意移除满行的动作是倒着来的，因为我们是按照重力来表现游戏的，如果不这样就有可能出现有些方块浮在空中的现象。

```
1.  def newPiece(self):
```

```
2.
3.      self.curPiece = Shape()
4.      self.curPiece.setRandomShape()
5.      self.curX = Board.BoardWidth // 2 + 1
6.      self.curY = Board.BoardHeight - 1 + self.curPiece.minY()
7.
8.      if not self.tryMove(self.curPiece, self.curX, self.curY):
9.
10.         self.curPiece.setShape(Tetrominoe.NoShape)
11.         self.timer.stop()
12.         self.isStarted = False
13.         self.msg2Statusbar.emit("Game over")
```

`newPiece()` 方法是用来创建形状随机的方块。如果随机的方块不能正确的出现在预设的位置，游戏结束。

```
1.  class Tetrominoe(object):
2.
3.      NoShape = 0
4.      ZShape = 1
5.      SShape = 2
6.      LineShape = 3
7.      TShape = 4
8.      SquareShape = 5
9.      LShape = 6
10.     MirroredLShape = 7
```

`Tetrominoe` 类保存了所有方块的形状。我们还定义了一个 `NoShape` 的空形状。

Shape类保存类方块内部的信息。

```
1.  class Shape(object):
2.
3.      coordsTable = (
4.          ((0, 0),     (0, 0),     (0, 0),     (0, 0)),
5.          ((0, -1),    (0, 0),     (-1, 0),    (-1, 1)),
```

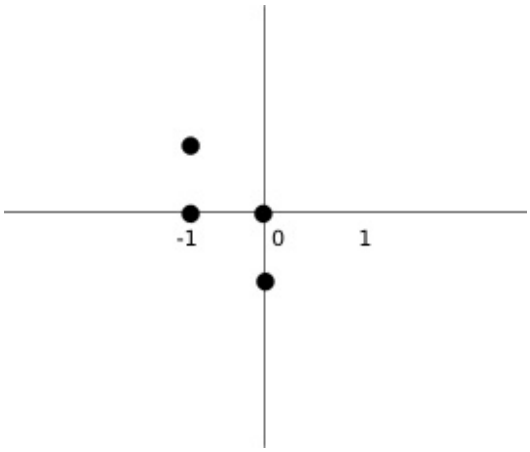
```
6.          ...
7.      )
8.  ...
```

coordsTable元组保存了所有的方块形状的组成。是一个构成方块的坐标模版。

```
1.  self.coords = [[0,0] for i in range(4)]
```

上面创建了一个新的空坐标数组，这个数组将用来保存方块的坐标。

坐标系示意图：



上面的图片可以帮助我们更好的理解坐标值的意义。比如元组 `(0, -1), (0, 0), (-1, 0), (-1, -1)` 代表了一个Z形状的方块。这个图表就描绘了这个形状。

```
1.  def rotateLeft(self):
2.
3.      if self.pieceShape == Tetrominoe.SquareShape:
4.          return self
5.
6.      result = Shape()
7.      result.pieceShape = self.pieceShape
8.
9.      for i in range(4):
```

```
10.
11.             result.setX(i, self.y(i))
12.             result.setY(i, -self.x(i))
13.
14.         return result
```

`rotateLeft()` 方法向右旋转一个方块。正方形的方块就没必要旋转，就直接返回了。其他的是返回一个新的，能表示这个形状旋转了的坐标。

程序展示：