第 5 章

有趣的 GUI 技术

有很多特殊的应用，在初学者看来，简直就和魔法一样。例如，在屏幕上显示一只青蛙或者一只怪兽，用鼠标还可以来回拖动。在屏幕的任何位置绘制曲线，随时可以清除这些曲线，或者在系统托盘添加图标，弹出对话气泡等。这些看似很复杂，其实用 Python 实现起来相当简单，这是因为 Python 有大量第三方模块，通过这些模块，只需要几行代码就可以实现这些操作。本章将通过大量完整的例子演示如何实现这些看似复杂，其实相当简单的功能。

5.1 特殊窗口

在很多场景中，往往需要实现各种特殊的窗口形态，例如，非矩形的窗口（称为异形窗口）、半透明窗口等。本节将介绍如何实现这些特殊窗口。

5.1.1 使用 Canvas 实现五角星窗口

窗口通常都是矩形的，但在很多应用中，尤其是游戏，窗口是不规则的，例如，圆形、椭圆形、三角形、五角形，甚至是一个怪兽的形态（如有些游戏程序），其实这些仍然是窗口，只不过通过掩模（mask）技术将某些部分变得透明，因此，用户看到的窗口就变成了不规则的形状。

在计算机科学中，掩模（mask）或位掩码（bitmask）是用于位运算的数据，特别是在位域中。使用掩模可以在一个字节的多个位上设置开或关，或者在一次位运算中将开和关反转。

在计算机图形学中，掩模是用于隐藏或显示另一图像的部分的数字图像。掩模可以用于创建特殊效果或选择图像的区域进行编辑。掩模可以从零开始在图像编辑器中创建，也可以基于现有图像生成。例如，你可以使用一张照片作为掩模，来显示或隐藏另一张照片的部分，如果对窗口使用掩模，就会让窗口变成异形窗口。

本节会使用 Canvas 实现五角星形状的异形窗口，也就是说，运行程序后，窗口会变成一个红色的五角星，其他部分是透明的，效果如图 5-1 所示。使用鼠标拖曳即可移动该窗口。

从图 5-1 所示的五角星可以看出，运行程序，只会显示一个五角星，其他部分是透明的，可以看到后面的 Python 文件目录。

这个五角星是在 Canvas 中绘制的，所以需要先了解如何绘制五角星。绘制五角星需要使用如下几组数据。

（1）五角星的中心点坐标。

（2）五角星内切圆半径和外切圆半径。

（3）五角星 10 个顶点的坐标。

（4）五角星的旋转角度。

在这些数据中，(1)、(2)和(4)是直接指定的，而(3)可以通过计算获得。本节绘制的五角星是正五角星[①]，所以这里只讨论正五角星。五角星以及内切圆和外切圆如图 5-2 所示。其中，A 到 J 一共 10 个字母，分别表示五角星的 10 个顶点。现在计算顶点 B 的坐标，其他顶点的计算方法类似。

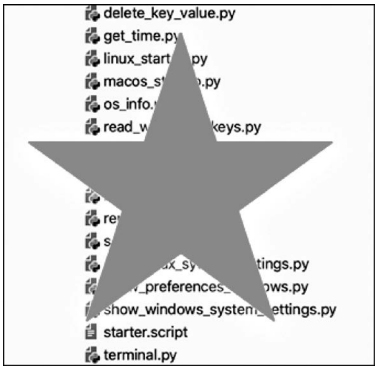

图 5-1　五角星形状的异形窗口

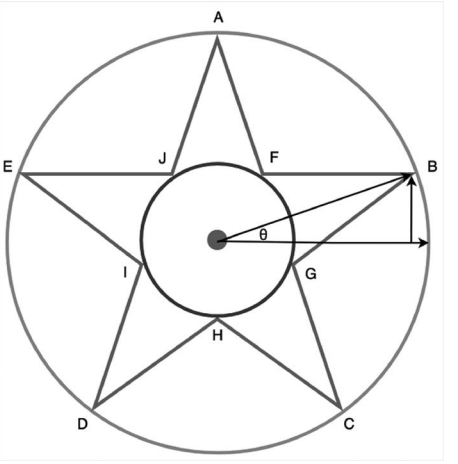

图 5-2　五角星顶点坐标的计算

假设五角星中心点的坐标是(centerX，centerY)，顶点 B 与中心点的连线与水平线的夹角是 θ，外切圆半径是 r，那么 B 点的坐标如下：

```
(centerX + r * cos(θ),centerY - r * sin(θ))
```

按类似的方法计算完 10 个顶点，就可以绘制 10 个点的多边形，最终形成一个五角星。

要想让窗口完全变成五角星，需要使用下面的代码将窗口的背景设置为透明，而且需要将窗口设置为无边框。

---

① 正五角星是一个由五条对角线构成的正五边形的内接星形。它由 5 个相等的等腰三角形组成，每个三角形的顶点是正五边形的一个顶点。正五角星的每个内角是 36°，每个外角是 72°。正五角星有很多数学和几何性质，例如，它的对角线之比是黄金比。

```
# 设置窗口的背景为透明
self.setAttribute(Qt.WidgetAttribute.WA_TranslucentBackground)
# 设置窗口的边框为无边框
self.setWindowFlag(Qt.WindowType.FramelessWindowHint)
```

五角星会将窗口的标题栏隐藏，这样将无法拖动窗口，所以需要使用下面的步骤，用鼠标可以通过拖动五角星来移动窗口。

（1）在 QWidget 的 mousePressEvent 方法中，检测鼠标左键是否按下，并记录下鼠标的位置。

（2）在 QWidget 的 mouseMoveEvent 方法中，检测鼠标是否移动，并计算移动的距离，然后用 move 方法来移动窗口。

（3）在 QWidget 的 mouseReleaseEvent 方法中，检测鼠标左键是否松开，并重置鼠标位置为 None。

下面的例子完整地演示了如何使用 PyQt6 实现一个五角星窗口，并通过鼠标拖动五角星移动窗口。

代码位置：src/interesting_gui/pyqt6/star_window_canvas.py

```
from PyQt6.QtWidgets import QApplication, QWidget
from PyQt6.QtGui import QPainter, QPen, QBrush, QColor
from PyQt6.QtCore import Qt, QPoint
import sys
import math
# 定义一个自定义的窗口类,继承自 QWidget
class StarWindow(QWidget):
    # 初始化方法
    def __init__(self):
        # 调用父类的初始化方法
        super().__init__()
        # 设置窗口的标题和尺寸
        self.setWindowTitle("红色正五角星")
        self.resize(300, 300)
        # 设置窗口的背景为透明
        self.setAttribute(Qt.WidgetAttribute.WA_TranslucentBackground)
        # 设置窗口的边框为无边框
        self.setWindowFlag(Qt.WindowType.FramelessWindowHint)
        # 初始化鼠标位置和偏移量为 None
        self.mouse_pos = None
        self.offset = None
    # 重写绘图事件方法
    def paintEvent(self, event):
        # 创建一个画笔对象,设置颜色为红色,宽度为 2 像素,样式为实线
        pen = QPen(QColor(255, 0, 0), 2, Qt.PenStyle.SolidLine)
        # 创建一个画刷对象,设置颜色为红色,样式为实心填充
        brush = QBrush(QColor(255, 0, 0), Qt.BrushStyle.SolidPattern)
        # 创建一个绘图对象,传入 self 作为参数
        painter = QPainter(self)
```

```
        # 设置画笔和画刷
        painter.setPen(pen)
        painter.setBrush(brush)
        # 获取窗口的宽度和高度
        width = self.width()
        height = self.height()
        # 计算五角星的外接圆半径,取宽度和高度中较小的一半减去 10 像素作为边距
        radius = min(width, height) / 2 - 10
        # 计算五角星的内接圆半径,根据正五角星的性质,内接圆半径是外接圆半径的 0.382 倍
        inner_radius = radius * 0.382
        # 计算五角星的中心点,即窗口的中心点
        center = QPoint(int(width / 2), int(height / 2))
        # 创建一个空列表,用于存储五角星的顶点坐标
        points = []
        # 循环 5 次,每次计算一个顶点坐标,并添加到列表中
        for i in range(5):
            # 计算当前顶点对应的角度,单位是弧度,注意要减去 90 度,使得第一个顶点在正上方
            angle = (i * 72 - 90) / 180 * math.pi
            # 计算当前顶点的横坐标和纵坐标,根据三角函数公式,横坐标等于中心点横坐标加
# 上外接圆半径乘以角度的余弦值,纵坐标等于中心点纵坐标加上外接圆半径乘以角度的正弦值
            x = center.x() + radius * math.cos(angle)
            y = center.y() + radius * math.sin(angle)
            # 创建一个 QPoint 对象,表示当前顶点坐标,并添加到列表中
            points.append(QPoint(int(x), int(y)))
            # 计算下一个顶点对应的角度,单位是弧度,注意要加上 36 度,使得每两个相邻的顶
            # 点之间有一个内角为 36 度的凹角
            next_angle = (i * 72 + 36 - 90) / 180 * math.pi
            # 计算下一个顶点的横坐标和纵坐标,根据三角函数公式,横坐标等于中心点横坐标
# 加上内接圆半径乘以角度的余弦值,纵坐标等于中心点纵坐标加上内接圆半径乘以角度的正弦值
            next_x = center.x() + inner_radius * math.cos(next_angle)
            next_y = center.y() + inner_radius * math.sin(next_angle)
            # 创建一个 QPoint 对象,表示下一个顶点坐标,并添加到列表中
            points.append(QPoint(int(next_x), int(next_y)))
            # 使用绘图对象的 drawPolygon 方法,传入列表作为参数,绘制一个多边形,即五角星
        print(len(points))
        painter.drawPolygon(*points)

        # 重写鼠标按下事件方法
    def mousePressEvent(self, event):
        # 如果鼠标左键被按下
        if event.button() == Qt.MouseButton.LeftButton:
            # 获取鼠标当前的位置,并赋值给 mouse_pos 属性
            self.mouse_pos = event.globalPosition()
            # 获取窗口当前的位置,并赋值给 offset 属性
            self.offset = self.pos()

    # 重写鼠标移动事件方法
    def mouseMoveEvent(self, event):
        # 如果鼠标左键被按下,并且 mouse_pos 和 offset 属性不为 None
        if event.buttons() == Qt.MouseButton.LeftButton and self.mouse_pos is not None and
self.offset is not None:
```

```
            # 计算鼠标移动的距离,等于鼠标当前的位置减去 mouse_pos 属性
            delta = event.globalPosition() - self.mouse_pos
            # 将 delta 转换为 QPoint 类型,使用 QPoint 的构造函数,传入 delta 的 x()和 y()作为参数
            delta = QPoint(int(delta.x()), int(delta.y()))
            # 计算窗口移动后的位置,等于 offset 属性加上 delta
            new_pos = self.offset + delta
            # 设置窗口的位置为 new_pos
            self.move(new_pos)

    # 重写鼠标松开事件方法
    def mouseReleaseEvent(self, event):
        # 如果鼠标左键被松开
        if event.button() == Qt.MouseButton.LeftButton:
            # 将 mouse_pos 和 offset 属性重置为 None
            self.mouse_pos = None
            self.offset = None

    # 创建一个应用对象,传入 sys.argv 作为参数
app = QApplication(sys.argv)
# 创建一个窗口对象
window = StarWindow()
# 显示窗口
window.show()
# 进入应用的主循环
sys.exit(app.exec())
```

运行程序,就会看到如图 5-1 所示的五角星窗口。

5.1.2　使用透明 png 图像实现美女机器人窗口

尽管使用 Canvas 可以实现异形窗口,但对于更复杂的异形窗口,使用 Canvas 就很麻烦,尤其是人物、机械装置这些几乎不可能通过 Canvas 来绘制的效果。因此,需要直接使用透明 png 图像实现异形窗口,如图 5-3 所示为美女机器人窗口。使用鼠标拖动美女机器人,就可以移动该窗口。

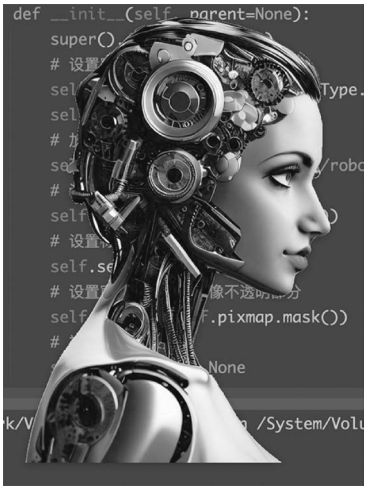

图 5-3　美女机器人窗口

本节的例子除了要显示 png 图像外,其他效果的实现方式与 5.1.1 节的例子完全相同。下面讲一下如何处理 png 图像。

可以使用 QLabel 显示 png 图像,所以创建一个从 QLabel 派生的类作为窗口类。然后使用 QPixmap 装载 png 图像,并使用 QLabel.setPixmap 方法在 QLabel 组件中显示图像。最后,使用 QLabel.setMask 设置窗口形状跟随图像不透明部分,从而让窗口的形状呈现出与 png 图像中不透明的部分完全一样的效

果。代码如下：

```
# 加载 png 图像
self.pixmap = QPixmap("images/robot.png")
# 设置标签大小和图像大小一致
self.resize(self.pixmap.size())
# 设置标签显示图像
self.setPixmap(self.pixmap)
# 设置窗口形状跟随图像不透明部分
self.setMask(self.pixmap.mask())
```

下面的例子演示了如何装载 png 图像，如何设置掩模，如何通过鼠标拖动窗口的完整实现过程。

代码位置：src/interesting_gui/pyqt6/image_transparent_window.py

```
from PyQt6.QtWidgets import QApplication, QLabel
from PyQt6.QtGui import QPixmap
from PyQt6.QtCore import Qt

class ImageTransparentWindow(QLabel):
    def __init__(self, parent = None):
        super().__init__(parent)
        # 设置窗口无边框和半透明
        self.setWindowFlags(Qt.WindowType.FramelessWindowHint)
        self.setWindowOpacity(0.99)
        # 加载 png 图像
        self.pixmap = QPixmap("images/robot.png")
        # 设置标签大小和图像大小一致
        self.resize(self.pixmap.size())
        # 设置标签显示图像
        self.setPixmap(self.pixmap)
        # 设置窗口形状跟随图像不透明部分
        self.setMask(self.pixmap.mask())
        # 初始化鼠标位置
        self.mouse_pos = None
    # 重写鼠标按下事件
    def mousePressEvent(self, event):
        # 如果是左键按下，记录当前鼠标位置
        if event.button() == Qt.MouseButton.LeftButton:
            self.mouse_pos = event.globalPosition()

    # 重写鼠标移动事件
    def mouseMoveEvent(self, event):
        # 如果鼠标位置已记录，计算鼠标移动的偏移量，并更新窗口位置
        if self.mouse_pos is not None:
            delta = event.globalPosition() - self.mouse_pos
            self.move(int(self.x() + delta.x()), int(self.y() + delta.y()))
            self.mouse_pos = event.globalPosition()
```

```
    # 重写鼠标释放事件
    def mouseReleaseEvent(self, event):
        # 如果是左键释放,清空鼠标位置记录
        if event.button() == Qt.MouseButton.LeftButton:
            self.mouse_pos = None

# 创建一个应用程序对象
app = QApplication([])
# 创建一个图像标签对象
win = ImageTransparentWindow()
# 显示透明图像
win.show()
# 运行应用程序循环
app.exec()
```

在运行程序之前,先要准备一个 png 图像文件,并将其命名为 robot.png。将这个图像文件放到 images 目录下,然后运行程序,界面上会立刻显示一个美女机器人。如果读者使用其他透明 png 图像,则会显示其他形态的异形窗口。

5.1.3　半透明窗口

通过 QWidget.SetWindowOpacity 方法可以设置窗口的透明度,通过 QWidget.setWindowFlag 方法可以隐藏窗口的标题栏和边框。将这两个方法组合起来使用,可以实现将组件放到半透明窗口上的效果。例如,图 5-4 在一个半透明窗口上放置了一个标签组件和一个按钮组件,而且隐藏了窗口的边框和标题栏。单击 close 按钮就能关闭窗口。

图 5-4　半透明窗口

下面的例子完整地演示了图 5-4 所示效果的实现方法。

代码位置：src/interesting_gui/pyqt6/translucent _window.py

```
from PyQt6.QtWidgets import QApplication, QWidget, QLabel, QPushButton
from PyQt6.QtCore import Qt
# 定义一个自定义的窗口类,继承自 QWidget
class MyWindow(QWidget):
    # 初始化方法
    def __init__(self):
        # 调用父类的初始化方法
        super().__init__()
        # 设置窗口大小为 400 * 300
```

```
        self.resize(400, 300)
        # 设置窗口透明度为 0.5
        self.setWindowOpacity(0.7)
        # 设置窗口无边框和标题栏
        self.setWindowFlag(Qt.WindowType.FramelessWindowHint)
        # 初始化鼠标位置为 None
        self.mousePos = None
        # 调用创建组件的方法
        self.createWidgets()

    # 创建组件的方法
    def createWidgets(self):
        # 创建一个标签,显示"Hello, world!"
        label = QLabel("Hello, world!", self)
        # 设置标签的字体大小为 20
        label.setStyleSheet("font-size: 20px;")
        # 设置标签的位置和大小
        label.setGeometry(150, 100, 200, 50)
        # 创建一个按钮,显示"Close"
        button = QPushButton("Close", self)
        # 设置按钮的位置和大小
        button.setGeometry(150, 200, 100, 50)
        # 绑定按钮的点击信号和关闭窗口的槽函数
        button.clicked.connect(self.close)
    # 重写鼠标按下事件的方法
    def mousePressEvent(self, event):
        # 如果鼠标左键被按下
        if event.button() == Qt.MouseButton.LeftButton:
            # 获取鼠标相对于窗口的位置
            self.mousePos = event.pos()
    # 重写鼠标移动事件的方法
    def mouseMoveEvent(self, event):
        # 如果鼠标左键被按下并且鼠标位置不为空
        if event.buttons() == Qt.MouseButton.LeftButton and self.mousePos:
            # 获取鼠标相对于屏幕的位置
            globalPos = event.globalPosition().toPoint()
            # 计算窗口应该移动到的位置
            windowPos = globalPos - self.mousePos
            # 移动窗口到新的位置
            self.move(windowPos)
# 创建一个应用对象
app = QApplication([])
# 创建一个窗口对象
window = MyWindow()
# 显示窗口
window.show()
# 进入应用的事件循环
app.exec()
```

5.2　在屏幕上绘制曲线

有很多画笔应用，可以在整个电脑屏幕上绘制曲线和各种图形，这种应用很适合在线教学或演示。例如，要讲解代码的编写过程，可以一边展示代码，一边在代码上绘制曲线、手写一些文字等，如图 5-5 所示。

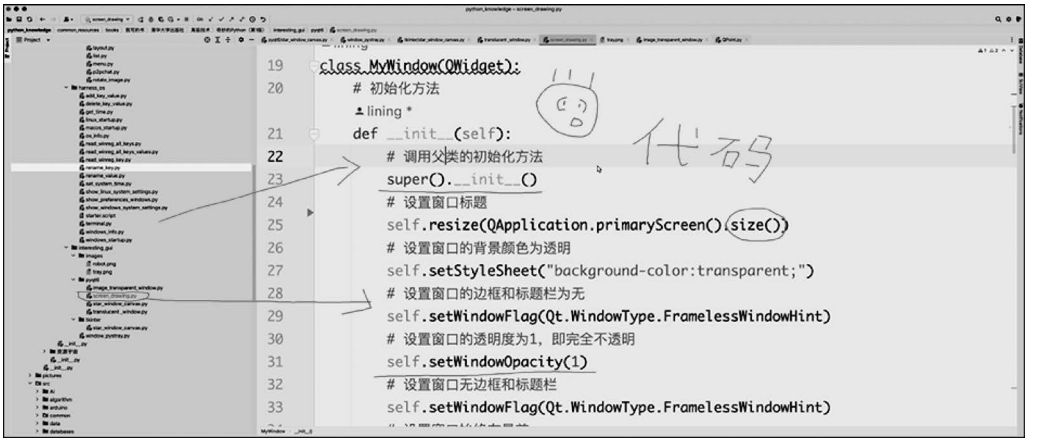

图 5-5　在屏幕上自由绘制曲线

实现这个功能的基本做法是让窗口充满整个屏幕，并且隐藏边框和标题栏，以及让窗口完全透明，这样就可以看到窗口下方的内容了。我们还可以直接在窗口上绘制曲线，放置组件，绘制各种简单和复杂的图形，放置图像等。相关内容已在前文有所涉及，下面看一个完整的例子，即如何实现在屏幕上绘制曲线。

代码位置：src/interesting_gui/pyqt6/ screen_drawing. py

```
from PyQt6.QtWidgets import QApplication, QWidget
from PyQt6.QtGui import QPainter, QPen, QColor
from PyQt6.QtCore import Qt
# 定义一个自定义的窗口类,继承自 QWidget
class MyWindow(QWidget):
    # 初始化方法
    def __init__(self):
        # 调用父类的初始化方法
        super().__init__()
        # 设置窗口标题
        self.resize(QApplication.primaryScreen().size())
        # 设置窗口的背景颜色为透明
        self.setStyleSheet("background-color:transparent;")
        # 设置窗口的边框和标题栏为无
        self.setWindowFlag(Qt.WindowType.FramelessWindowHint)
        # 设置窗口的透明度为 1,即完全不透明
        self.setWindowOpacity(1)
```

```
        # 设置窗口无边框和标题栏
        self.setWindowFlag(Qt.WindowType.FramelessWindowHint)
        # 设置窗口始终在最前
        self.setWindowFlag(Qt.WindowType.WindowStaysOnTopHint)
        # 创建一个画笔对象,设置颜色为红色,线宽为 3 像素,样式为实线
        self.pen = QPen(QColor("black"), 3, Qt.PenStyle.SolidLine)
        # 创建一个空列表,用于存储鼠标移动的点坐标
        self.points = []
        # 创建一个布尔变量,用于标记鼠标是否按下
        self.pressed = False
    # 重写绘图事件方法
    def paintEvent(self, event):
        # 调用父类的绘图事件方法
        super().paintEvent(event)
        # 创建一个画家对象,传入窗口作为参数
        painter = QPainter(self)
        # 设置画笔
        painter.setPen(self.pen)
        # 如果点列表不为空,绘制点之间的曲线
        if self.points:
            painter.drawPolyline(*self.points)
    # 重写鼠标按下事件方法
    def mousePressEvent(self, event):
        # 调用父类的鼠标按下事件方法
        super().mousePressEvent(event)
        # 如果鼠标左键被按下
        if event.button() == Qt.MouseButton.LeftButton:
            # 将布尔变量设为 True,表示鼠标按下状态
            self.pressed = True
            # 清空点列表,开始新的绘制
            self.points.clear()
            # 将鼠标当前位置添加到点列表中
            self.points.append(event.pos())
            # 更新窗口,触发绘图事件
            self.update()
    # 重写鼠标移动事件方法
    def mouseMoveEvent(self, event):
        # 调用父类的鼠标移动事件方法
        super().mouseMoveEvent(event)
        # 如果鼠标处于按下状态
        if self.pressed:
            # 将鼠标当前位置添加到点列表中
            self.points.append(event.pos())
            # 更新窗口,触发绘图事件
            self.update()

    # 重写鼠标释放事件方法
    def mouseReleaseEvent(self, event):
        # 调用父类的鼠标释放事件方法
        super().mouseReleaseEvent(event)
        # 如果鼠标左键被释放
```

```
            if event.button() == Qt.MouseButton.LeftButton:
                # 将布尔变量设为 False,表示鼠标释放状态
                self.pressed = False
        # 重写键盘按下事件方法
        def keyPressEvent(self, event):
            # 调用父类的键盘按下事件方法
            super().keyPressEvent(event)
            # 如果按下 Esc 键
            if event.key() == Qt.Key.Key_Escape:
                # 关闭窗口,退出程序
                self.close()
            elif event.key() == Qt.Key.Key_2:
                self.pen = QPen(QColor("red"), 3, Qt.PenStyle.SolidLine)
            elif event.key() == Qt.Key.Key_3:
                self.pen = QPen(QColor("blue"), 3, Qt.PenStyle.SolidLine)
            elif event.key() == Qt.Key.Key_4:
                self.pen = QPen(QColor("green"), 3, Qt.PenStyle.SolidLine)
    # 创建一个应用对象
    app = QApplication([])
    # 创建一个窗口对象
    window = MyWindow()
    # 显示窗口
    window.show()
    # 进入应用的主循环
    app.exec()
```

运行程序,默认可以在屏幕上绘制黑色曲线;按 2 键,可以绘制红色曲线;按 3 键,可以绘制蓝色曲线;按 4 键,可以绘制绿色曲线;按 Esc 键,关闭程序。

注意:本例只能在 macOS 上运行。

5.3　控制状态栏

状态栏一直是各类程序争夺的主阵地之一,有很多主流程序都会在状态栏添加 1 个或多个图标,以及添加图标弹出菜单、对话气泡等功能。本节将详细讲解如何 Python“攻占”这块主阵地。

5.3.1　在状态栏上添加图标

Windows、macOS 和 Linux 都有状态栏,而且都允许用户添加图标,只是添加图标的区域不同。**Windows** 称为**通知区域**,也就是任务栏右侧的部分,它包含了一些常用的图标和通知,如电量、Wi-Fi、音量、时钟和日历等。**macOS** 中称为**菜单栏**,在菜单栏左侧显示 macOS 菜单,右侧显示各种图标,单击这些图标后会触发不同的操作。**Linux** 右上角显示图标的区域的称呼可能因不同的桌面环境而有所不同,但一般可以称为**状态栏**或者**通知区域**。为了统一,后文统称为**状态栏**。

通常一个应用程序会对应一个图标,但也可以对应多个图标。使用 pystray 模块可以将图

标添加到状态栏上。如果没有安装 pystray 模块，可以使用下面的命令安装 pystray 模块。

```
pip3 install pystray
```

pystray 模块是跨平台的，可以用其将图标添加到 Windows 的通知区域、macOS 的菜单栏右侧以及 Linux 的通知区域。

使用 pystray 模块将图标和菜单添加到状态栏的流程大致如下。

（1）导入 pystray 模块和 PIL 模块，用于创建图标和加载图像。

（2）创建一个 pystray.Icon 对象，指定图标的名称、图像、标题和单击回调函数（这些不一定都指定，也可以指定一部分）。

（3）创建一个 pystray.Menu 对象，指定菜单的各个项，每个项可以是一个 pystray.MenuItem 对象或者一个分隔符。

（4）将菜单对象赋值给 pystray.Icon 对象的 menu 命名参数。

（5）调用图标对象的 run 方法，启动图标的主循环。

在创建图标和菜单的过程中，涉及 pystray.Icon 类、pystray.Menu 类和 pystray.MenuItem 类，下面分别给出这 3 个类的构造方法原型以及参数函数。

1. pystray.Icon 类

该类用于创建图标对象，其构造方法原型如下：

```
pystray.Icon(name, icon = None, title = None, menu = None, action = None)
```

参数含义如下。

（1）name：图标的名称，必须是唯一的字符串。

（2）icon：图标的图像，可以是一个 PIL.Image 对象或者一个返回 PIL.Image 对象的函数。

（3）title：图标的标题，可以是一个字符串或者一个返回字符串的函数。

（4）menu：图标的菜单，可以是一个 pystray.Menu 对象或者一个返回 pystray.Menu 对象的函数。

（5）action：图标被单击时执行的回调函数，可以是一个无参数的函数或者一个返回无参数函数的函数。

2. pystray.Menu 类

该类用于创建菜单对象，其构造方法原型如下：

```
pystray.Menu( * items)
```

Menu 类的构造方法只有一个 items 参数，用于表示菜单中的菜单项，可以是 pystray.MenuItem 对象或者 pystray.SEPARATOR 常量。

3. pystray.MenuItem 类

该类用于创建菜单项对象，其构造方法原型如下：

```
pystray.MenuItem(text, action, checked = None, enabled = None, visible = None)
```

参数含义如下。

（1）text：菜单项的文本，可以是一个字符串或者一个返回字符串的函数。

（2）action：菜单项被单击时执行的回调函数，可以是一个无参数的函数或者一个返回无参数函数的函数。

（3）checked：菜单项是否被选中，可以是一个布尔值或者一个返回布尔值的函数。

（4）enabled：菜单项是否可用，可以是一个布尔值或者一个返回布尔值的函数。

（5）visible：菜单项是否可见，可以是一个布尔值或者一个返回布尔值的函数。

下面的例子完整地演示了如何使用 pystray 模块在状态栏添加一个菜单，以及相应菜单项的点击动作。本例的菜单中有两个菜单项——Hello 和“退出”。单击 Hello 菜单项会在终端输出 Hello 字符串，单击“退出”菜单项，会退出应用程序。

代码位置：src/interesting_gui/pystray_demo.py

```
from pystray import Icon, MenuItem, Menu
from PIL import Image
class MyTray:
    # 定义一个无参数的函数，用于弹出消息框
    def __init__(self, image):
        image = Image.open(image)

        menu = Menu(
                    MenuItem("Hello", lambda: print("Hello")),
                    MenuItem("退出", lambda: self.icon.stop())
                )
        self.icon = Icon(name='nameTray', title='titleTray', icon=image, menu=menu)
    def stopProgram(self, icon):
        self.icon.stop()
    def runProgram(self):
        self.icon.run()
if __name__ == '__main__':
    myTray = MyTray(image="images/tray.png")
    myTray.runProgram()
```

运行程序，会看到在状态栏上显示一个绿色的图标，在图标上右击鼠标，会弹出一个菜单，图 5-6 是 Windows 下的效果，图 5-7 是 macOS 下的效果，图 5-8 是 Ubuntu Linux 下的效果。

图 5-6　Windows 中通知区域图标和菜单

图 5-7　macOS 中菜单栏图标和菜单

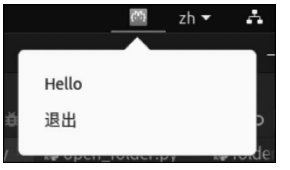

图 5-8　Ubuntu Linux 中菜单栏图标和菜单

5.3.2 添加 Windows 10 风格的 Toast 消息框

使用 win10toast 模块可以添加 Windows 10 风格的 Toast 消息框。该模块只能在 Windows 10 上运行。读者可以使用下面的命令安装 win10toast 模块：

```
pip3 install win10toast
```

与 win10toast 对应的还有一个 win10toast_click 模块，该模块的功能与 win10toast 的功能类似，只是能响应 Toast 消息框的单击事件。win10toast_click 模块可以完全取代 win10toast 模块，所以推荐使用 win10toast_click 模块。读者可以使用下面的命令安装 win10toast_click 模块：

```
pip install win10toast-click
```

注意：在使用 win10toast_click 模块时，win10toast 与 click 之间使用下画线(_)连接，而安装 win10toast-click 模块时，win10toast 与 click 之间使用连字符(-)连接。

win10toast_click 模块中的核心函数是 show_toast，该函数用于显示 Toast 消息框，其函数原型如下：

```
def show_toast(self, title, msg,
               icon_path=None, duration=None, threaded=False,
               callback_on_click=None)
```

函数参数的含义如下。

(1) title：通知的标题，必须是一个字符串。

(2) msg：通知的内容，必须是一个字符串。

(3) icon_path：通知的图标路径，必须是一个.ico 文件，如果为 None，则使用默认图标。

(4) duration：通知的持续时间，单位为 s，如果为 None，则使用默认值(5s)。

(5) threaded：是否使用多线程来显示通知。如果为 True，则不会阻塞程序的运行；如果为 False，则会等待通知消失后再继续程序的运行。

(6) callback_on_click：在用户点击通知时执行的函数，必须是一个无参数的函数。如果为 None，则不执行任何函数。

由于 icon_path 参数只支持 ico 文件，所以如果图标文件是其他图像格式(如 png 文件)，就需要使用 PIL 模块的相关 API 将其他图像格式的文件转换为 ico 图像格式的文件。

下面的例子使用 win10toast_click 模块在 Windows 通知区域添加一个图标，以及在图标上方显示一个 Toast 消息框，点击消息框，会打开浏览器，并在浏览器中显示 webbrowser.open 函数打开的页面。

代码位置：src/interesting_gui/toast_demo.py

```
from win10toast_click import ToastNotifier
from PIL import Image                                    # 导入 Image 模块
import webbrowser                                        # 导入 webbrowser 模块,用于打开网页
filename = "images\\tray.png"                            # 指定要转换的.png 文件名
img = Image.open(filename)                               # 打开图片
img.save('images\\tray.ico')                             # 保存为.ico 文件
def open_url():                                          # 定义一个函数,用于打开一个网址
    webbrowser.open("https://www.unitymarvel.com")
toaster = ToastNotifier()
toaster.show_toast("软件更新",                            # 通知的标题
                   "UnityMarvel 已经更新到 2.01 版本,点击下载",      # 通知的内容
                   icon_path="images\\tray.ico",         # 通知的图标路径,如果为 None 则使用
                                                         # 默认图标
                   duration=20,                          # 通知的持续时间,单位为 s
                   threaded=True,                        # 是否使用多线程来显示通知,如果为
                                                         # True 则不会阻塞程序的运行
                   callback_on_click=open_url)
```

运行程序后，Windows 右下角会显示如图 5-9 所示的 Toast 消息框。

图 5-9　Windows 10 风格的 Toast 消息框

win10toast_click 模块在 Windows 11 或以上版本运行可能会有问题，如果读者使用的是 Windows 11，可以尝试使用 win11toast 模块。读者可以使用下面的命令安装 win11toast 模块：

```
pip3 install win11toast
```

win11toast 模块可以在 Windows 10 和 Windows 11 上运行，例子代码如下：

```
from win11toast import toast
toast('Hello', '这是一个通知')
```

运行程序，会看到 Windows 右下角出现如图 5-10 所示的 Toast 消息框。

5.3.3　使用 PyQt6 管理系统托盘

QSystemTrayIcon 类是 PyQt6 中的一个类，它提供了一种在系统托盘[①]中显示图标的

① 系统托盘就是 Windows 中的通知区域；macOS 中的菜单栏右上角的位置；Linux 中右上角的通知区域；只是另一种称呼而已。

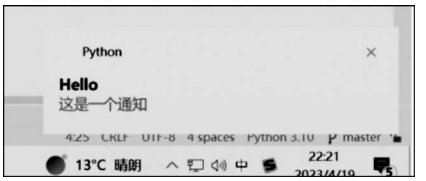

图 5-10 win11toast 模块显示的 Toast 消息框

方法。以下是 QSystemTrayIcon 类的主要功能：

（1）设置图标；

（2）设置提示信息；

（3）添加菜单；

（4）响应菜单单击事件；

（5）响应单击图标事件；

（6）显示消息(对话气泡)；

（7）响应消息单击事件。

要设置图标，可以使用 **setIcon** 方法。要设置提示信息，可以使用 **setToolTip** 方法。要添加菜单，可以使用 **setContextMenu** 方法。要响应菜单单击事件，可以使用 **connect** 方法连接 QAction 对象的 triggered 信号到槽函数。要响应单击图标事件，可以使用 activated 信号连接到槽函数。要显示消息，可以使用 **showMessage** 方法。要响应消息单击事件，可以使用 **messageClicked** 信号连接到槽函数。相关方法的描述如下。

1. setIcon 方法

方法原型如下：

```
def setIcon(self, icon: Union[QIcon, QPixmap]) -> None:
```

该方法用于设置 QSystemTrayIcon 的图标，接收一个 QIcon 或 QPixmap 对象作为参数。如果传递一个 QPixmap 对象，它将自动转换为 QIcon 对象。

2. setToolTip 方法

方法原型如下：

```
def setToolTip(self, tip: str) -> None:
```

该方法用于设置 QSystemTrayIcon 的提示信息，接收一个字符串作为参数，该字符串将显示在鼠标悬停在图标上时。

3. setContextMenu 方法

方法原型如下：

```
def setContextMenu(self, menu: QMenu) -> None:
```

该方法用于设置 QSystemTrayIcon 的上下文菜单，接收一个 QMenu 对象作为参数，该

对象包含要显示的菜单项。

4. showMessage 方法

方法原型如下：

```
def showMessage(self, title: str, message: str, icon: Union[QSystemTrayIcon.MessageIcon, int]
 = QSystemTrayIcon.Information, msecs: int = 10000) -> bool:
```

该方法用于在系统托盘中显示一条消息，接收 4 个参数——标题、消息、图标和显示时间(以 ms 为单位)。默认情况下，消息将显示 10s。

下面的例子使用 QSystemTrayIcon 类的相关 API 在系统托盘添加一个图标，以及一个菜单，并显示一个窗口。单击 Show 菜单项，会显示这个窗口；单击 Hide 菜单项，会隐藏窗口。单击托盘图标，会触发单击图标事件，在终端会输出如下信息：

```
You clicked the icon
```

单击图标的同时，在图标附近会显示一条消息，单击消息窗口，会在终端输出如下信息：

```
You clicked the message
```

代码位置：src/interesting_gui/pyqt6/bubble.py

```
import sys
from PyQt6.QtWidgets import QApplication, QWidget, QSystemTrayIcon, QMenu
from PyQt6.QtGui import QIcon,QAction
from PyQt6.QtCore import QCoreApplication
class MainWindow(QWidget):
    def __init__(self):
        super().__init__()
        self.initUI()
    def initUI(self):
        self.setWindowTitle("Tray Icon Example")
        self.resize(300, 200)
        self.tray_icon = QSystemTrayIcon()                    # 创建一个 QSystemTrayIcon 对象
        self.tray_icon.setIcon(QIcon("images/tray.png"))      # 设置图标
        self.tray_icon.setToolTip("This is a tray icon")      # 设置提示信息
        self.tray_icon.activated.connect(self.on_activated)   # 连接 activated 信号
        self.tray_icon.messageClicked.connect(self.on_message_clicked)
                                                              # 连接 messageClicked 信号
        self.tray_menu = QMenu()                              # 创建一个 QMenu 对象
        self.show_action = QAction("Show", self.tray_menu)    # 创建一个 QAction 对象,用于显
                                                              # 示窗口
        self.hide_action = QAction("Hide", self.tray_menu)    # 创建一个 QAction 对象,用于隐
                                                              # 藏窗口
        self.quit_action = QAction("Quit", self.tray_menu)    # 创建一个 QAction 对象,用于退
                                                              # 出程序
        self.show_action.triggered.connect(self.show)         # 连接 triggered 信号,显示窗口
        self.hide_action.triggered.connect(self.hide)         # 连接 triggered 信号,隐藏窗口
        self.quit_action.triggered.connect(QCoreApplication.instance().quit)
                                                              # 连接 triggered 信号,退出程序
```

```
            self.tray_menu.addAction(self.show_action)    # 将 QAction 对象添加到 QMenu 对象中
            self.tray_menu.addAction(self.hide_action)
            self.tray_menu.addSeparator()
            self.tray_menu.addAction(self.quit_action)
            self.tray_icon.setContextMenu(self.tray_menu)  # 将 QMenu 对象设置为 QSystemTrayIcon
                                                           # 对象的上下文菜单
            self.tray_icon.show()                          # 显示 QSystemTrayIcon 对象

        def on_activated(self, reason):                    # 定义一个槽函数,用于处理用户单击
                                                           # 图标的事件
            if reason == QSystemTrayIcon.ActivationReason.Trigger:      # 如果用户单击了图标
                print("You clicked the icon")
                self.tray_icon.showMessage("Hello World", "This is a message from PyQt",
                QIcon("images/tray.png"))                  # 显示对话气泡

        def on_message_clicked(self):                      # 定义一个槽函数,用于处理用户单击
                                                           # 对话气泡的事件
            print("You clicked the message")
            QCoreApplication.instance().quit()             # 退出程序
    if __name__ == '__main__':
        app = QApplication(sys.argv)
        window = MainWindow()
        window.show()
        sys.exit(app.exec())
```

运行程序,单击图标,会显示一个消息窗口(对话气泡),macOS 中的对话气泡如图 5-11 所示。Windows 中的对话气泡如图 5-12 所示。Ubuntu Linux 并不会出现对话气泡,但其他功能正常。

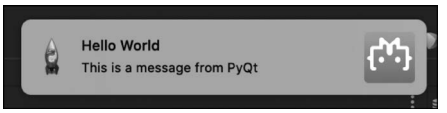

图 5-11　macOS 中的对话气泡

图 5-12　Windows 中的对话气泡

5.4　小结

本章介绍的内容相当有意思,尽管这些功能对于大多数应用程序不是必需的,但如果自己的应用程序有这些功能,会显得更酷、更专业。尤其是在状态栏中添加图标,读者可以将一些常用的功能添加到图标菜单中,这样用户就可以很方便使用这些功能了。本章的很多内容使用了第 4 章讲的 PyQt6,所以如果读者对 PyQt6 不了解,请先阅读相关内容。