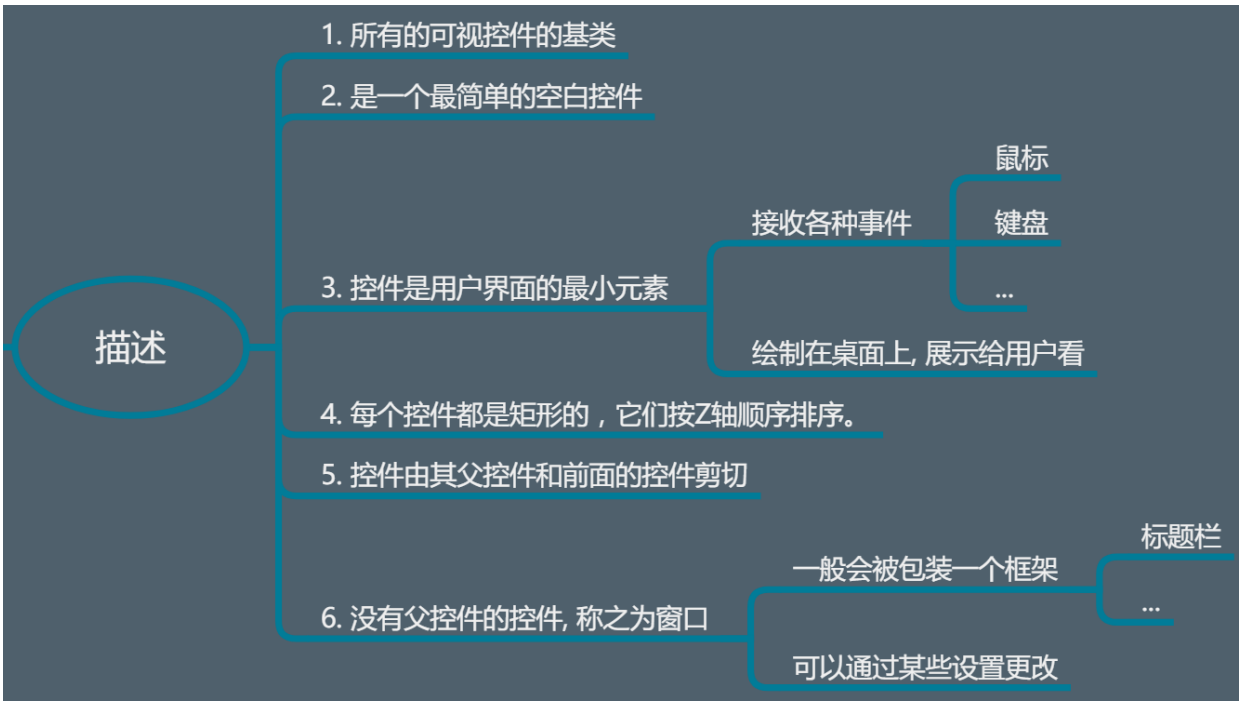
[笔记][LIKE-Python-GUI编程-PyQt5][08]

PyQt5

---

033. Python-GUI编程-PyQt5-QWidget简介



`QObject` 并不属于可视控件，它是看不见的，是所有 `Qt` 对象的基类。
而 `QWidget` 则是所有可视化控件的基类。

可视控件：用户可以看到的控件。

控件：用户界面的最小元素。

按 `z` 轴排列，靠近我们的优先被看到。

```
# 0. 导入需要的包和模块
import sys
from PyQt5.Qt import *

# 1. 创建一个应用程序对象
```

```
app = QApplication(sys.argv)

# 2. 控件的操作
window = QWidget()
window.setWindowTitle('QWidget简介')
window.resize(500, 500)

# 控件由其父控件裁剪
red = QWidget(window)
red.resize(100, 100)
red.setStyleSheet('background-color: red;')
red.move(450, 0)

# 控件被它前面的控件裁剪
green = QWidget(window)
green.resize(100, 100)
green.setStyleSheet('background-color: green;')
green.move(450, 50)

window.show()

# 3. 应用程序的执行，进入到消息循环
sys.exit(app.exec_())
```

---

034. Python-GUI编程-PyQt5-QWidget-继承

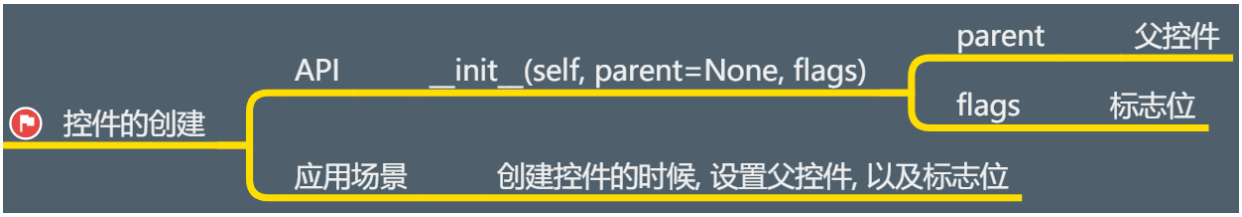
如何找到 `QWidget` 是继承自谁？

- `Ctrl + 单击` QWidget：
  `class QWidget(__PyQt5_QtCore.QObject, __PyQt5_QtGui.QPaintDevice):`
  `QPaintDevice` 是绘制设备
- `print(QWidget.__bases__)` 返回一个元组

- `print(QWidget.mro())` 方法解析顺序，返回一个继承链条，不仅包括直接父类，还包括间接父类

```
# 0. 导入需要的包和模块
import sys
from PyQt5.Qt import *

# 1. 创建一个应用程序对象
app = QApplication(sys.argv)

# 2. 控件的操作
# 2.1 创建控件
window = QWidget()
# 2.2 设置控件
window.setWindowTitle('QWidget继承')
window.resize(500, 500)

# 查看 QWidget 父类的几种方式:
# 方法1: Ctrl - 单击 QWidget
# 方法2: __bases__, 返回一个元组
print(QWidget.__bases__)
# 方法3: mro
print(QWidget.mro())

# 2.3 展示控件
window.show()

# 3. 应用程序的执行, 进入到消息循环
sys.exit(app.exec_())
```

035. Python-GUI编程-PyQt5-QWidget-控件创建

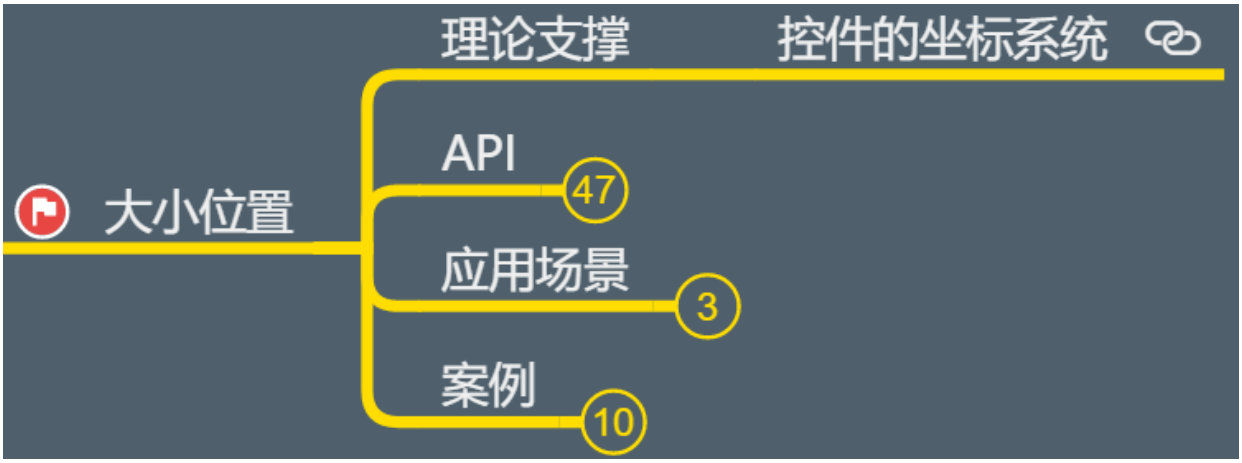



```
# 0. 导入需要的包和模块
import sys
from PyQt5.Qt import *

# 1. 创建一个应用程序对象
```

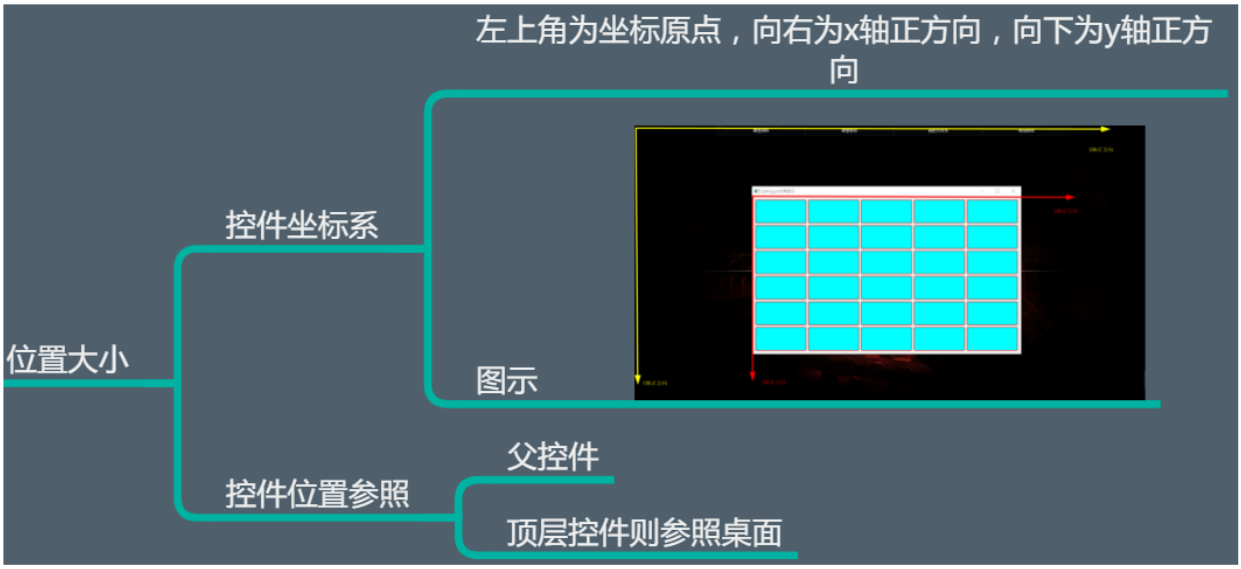
```
app = QApplication(sys.argv)

# 2. 控件的操作
window = QWidget()

# 设置父对象
red = QWidget(window)
red.resize(100, 100)
red.setStyleSheet('background-color: red;')

# 顶层窗口不会自动显示，必须调用 show 方法
# 顶层窗口会被包装一个框架
# 外部框架可以通过窗口标志来设置
window.show()

# 3. 应用程序的执行，进入到消息循环
sys.exit(app.exec_())
```

---

036. Python-GUI编程-PyQt5-QWidget-坐标系统



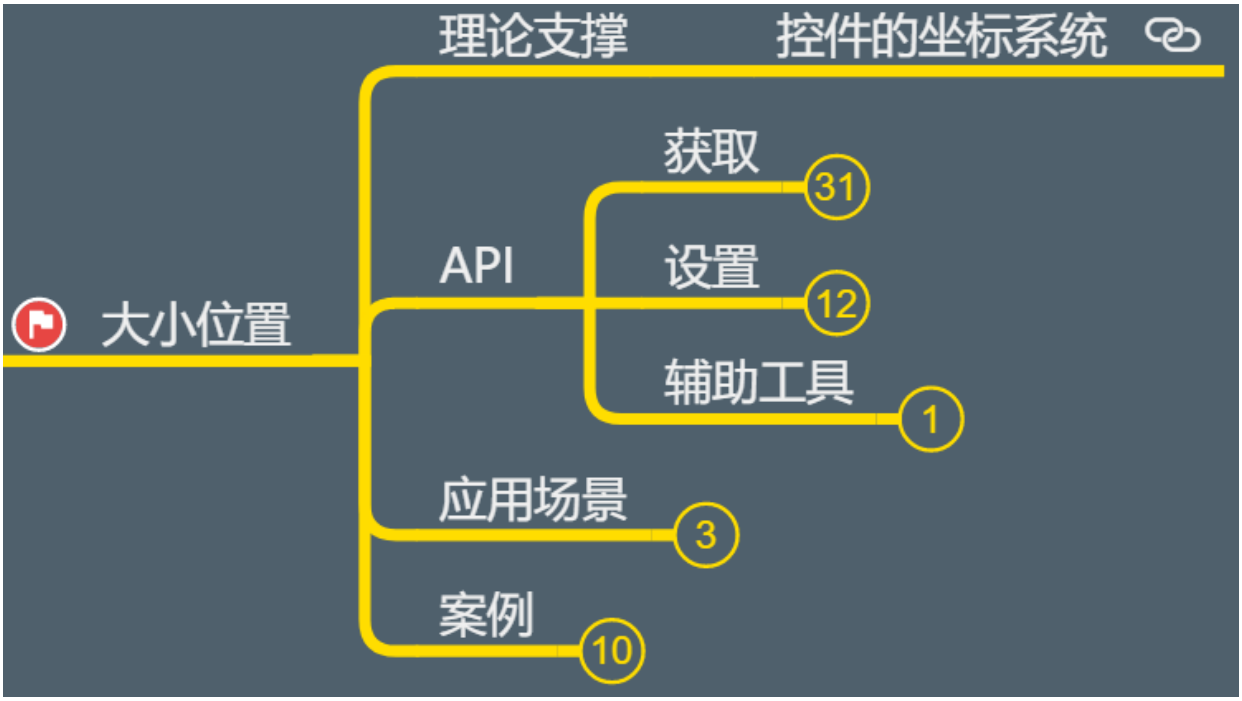
控件的坐标系统



坐标原点参照父控件，如果没有父控件，那么就是顶层窗口，顶层窗口参照整个电脑屏幕，左上角为坐标原点。

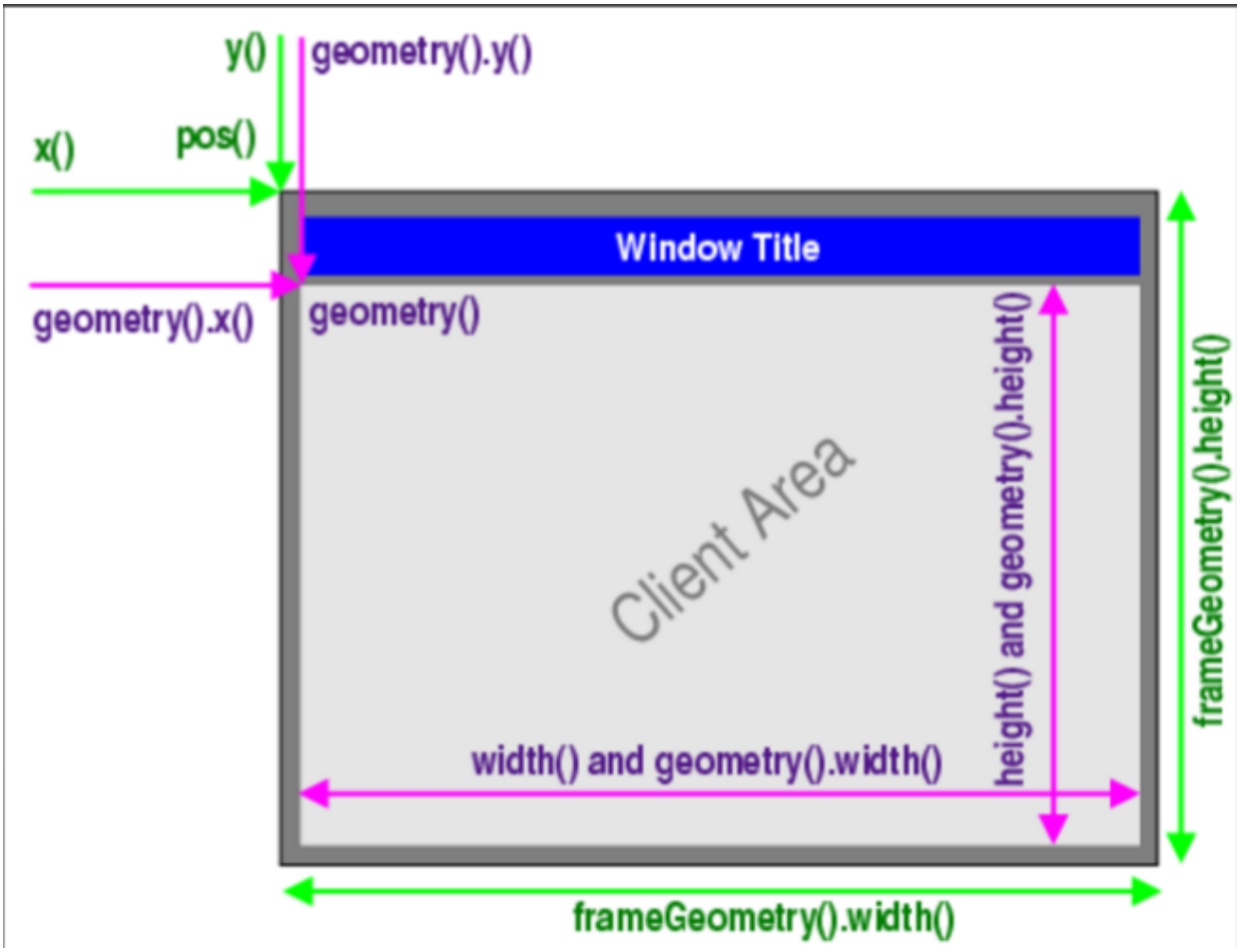
---

037. Python-GUI编程-PyQt5-QWidget-尺寸获取



`API` 分为两大类，**尺寸获取**和**尺寸设置**。
本节讲**尺寸获取**。

用户区域 `client area` ：即用户可以操作的区域。


y()
geometry().y()
x()
pos()
Window Title
geometry().x()
geometry()
Client Area
height() and geometry().height()
frameGeometry().height()
width() and geometry().width()
frameGeometry().width()

示例代码：

```
# 0. 导入需要的包和模块
import sys
from PyQt5.Qt import *

# 1. 创建一个应用程序对象
app = QApplication(sys.argv)

# 2. 控件的操作
window = QWidget()
window.move(100, 100)
```

```
window.resize(200, 200)

print('x:', window.x())
print('width:', window.width())
# 用户区域的 x, y, width, height
print('geometry:', window.geometry())

# 注意：控件显示完毕之后，获取到的尺寸数据才是正确的
window.show()

print('-' * 20 + '控件显示完毕' + '-' * 20)
print('x:', window.x())
print('width:', window.width())
# 用户区域的 x, y, width, height
print('geometry:', window.geometry())

# 3. 应用程序的执行，进入到消息循环
sys.exit(app.exec_())
```

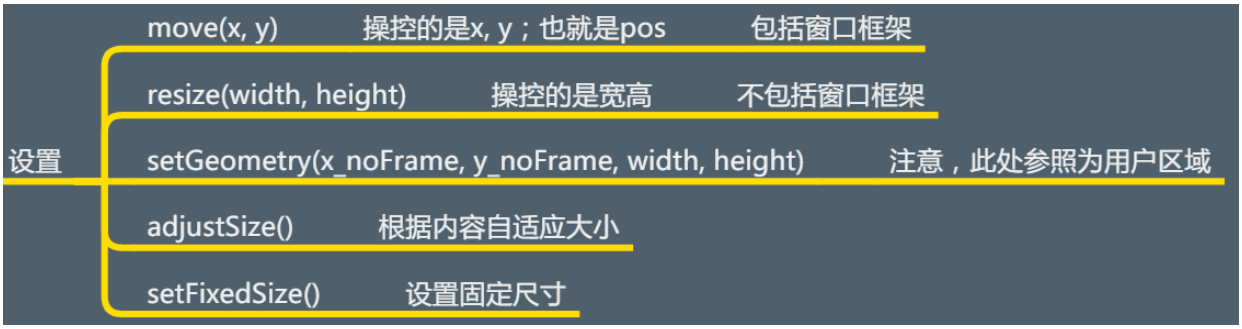
输出内容：

```
x: 100
width: 200
geometry: PyQt5.QtCore.QRect(100, 100, 200, 200)
--------------------控件显示完毕--------------------
x: 100
width: 200
geometry: PyQt5.QtCore.QRect(109, 138, 200, 200)
```

> **注意**
> `xy.exe` 程序可以辅助理解。

---

038. Python-GUI编程-PyQt5-QWidget-尺寸设置



示例代码1

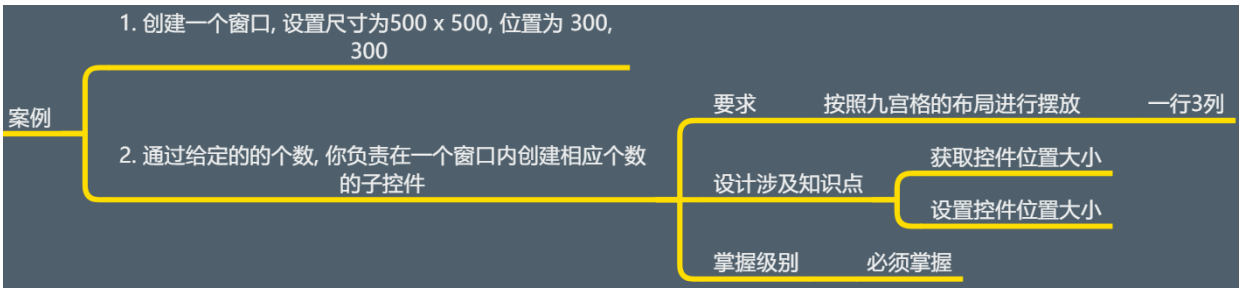
```
# 0. 导入需要的包和模块
import sys
from PyQt5.Qt import *

# 1. 创建一个应用程序对象
app = QApplication(sys.argv)

window = QWidget()
window.setWindowTitle('QWidget 尺寸设置1')

# 移动（包含窗口框架）
# window.move(100, 100)

# 更改用户区域的宽高
# 100, 100 显示出来的是一个长方形
# 因为窗口框架有一个最小值的限定
# window.resize(100, 100)
# window.resize(200, 200)

# 用户区域距离左上角的位置
# 显示之前设置会有问题
# window.setGeometry(0, 0, 200, 200)

window.show()
# 要在显示之后设置
# window.setGeometry(0, 0, 200, 200)

# 3. 应用程序的执行，进入到消息循环
sys.exit(app.exec_())
```

示例代码2

```
# 0. 导入需要的包和模块
import sys
from PyQt5.Qt import *
```

```python
# 1. 创建一个应用程序对象
app = QApplication(sys.argv)

window = QWidget()
window.setWindowTitle('QWidget 尺寸设置2')
window.move(200, 100)
# window.resize(500, 500)
# 这样就无法改变窗口大小了
window.setFixedSize(500, 500)


def slot():
    new_content = label.text() + '标签1'
    label.setText(new_content)
    # 笨方法
    # label.resize(label.width() + 100, label.height())
    # 或者使用自适应大小 label.adjustSize()
    label.adjustSize()


label = QLabel(window)
label.setText('标签1')
label.move(100, 100)
label.setStyleSheet('background-color: cyan;')

btn = QPushButton(window)
btn.setText('增加内容')
btn.move(100, 300)
btn.clicked.connect(slot)

window.show()

# 3. 应用程序的执行，进入到消息循环
sys.exit(app.exec_())
```

039. Python-GUI编程-PyQt5-QWidget-尺寸位置-案例1



案例1代码

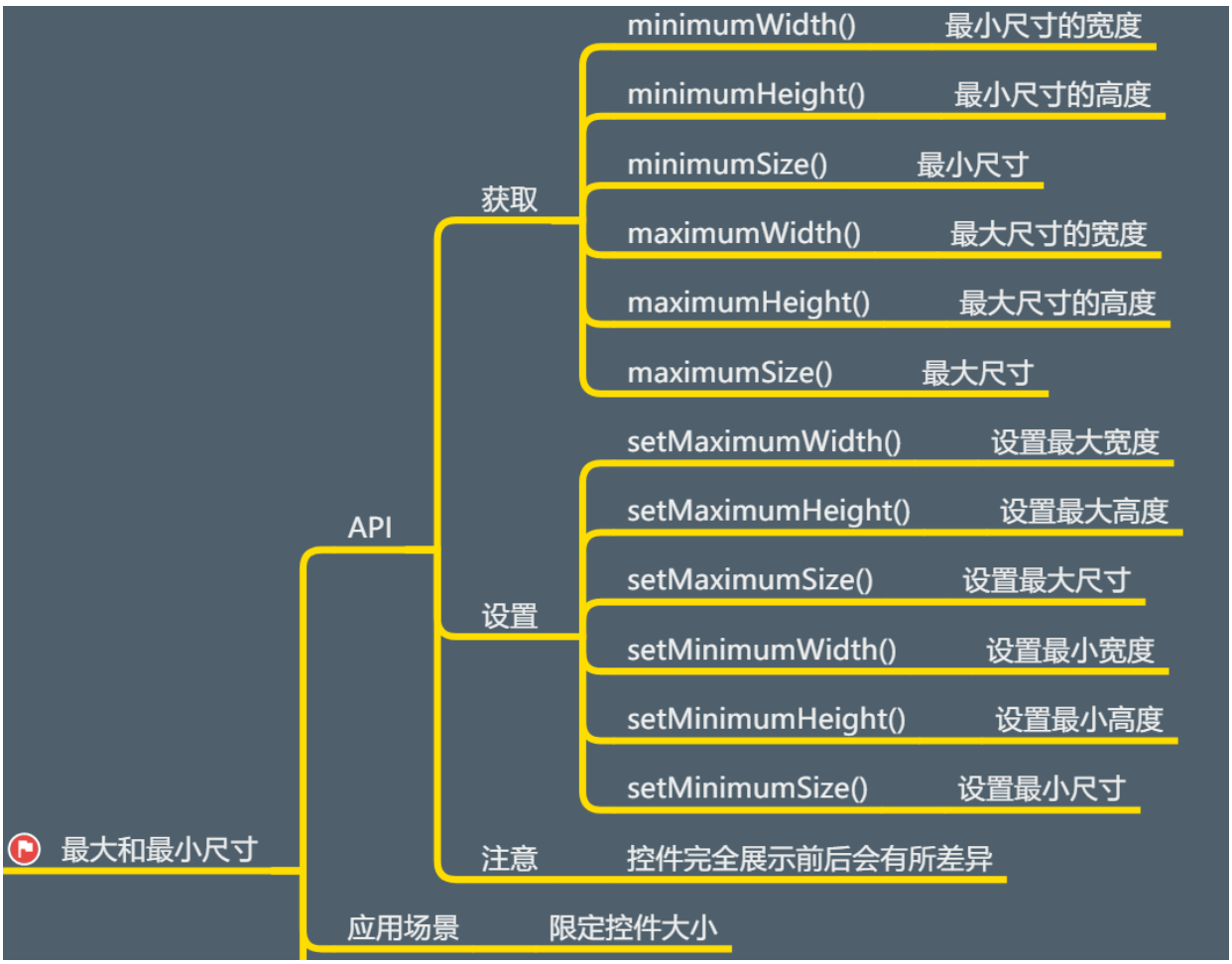
```python
import sys
from PyQt5.Qt import *

app = QApplication(sys.argv)

window = QWidget()
# window.resize
# window.setGeometry
window.resize(500, 500)  # 不加窗口框架，仅仅是用户区域
window.move(300, 300)  # 加上窗口框架
window.show()

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

040. Python-GUI编程-PyQt5-QWidget-尺寸位置-案例2

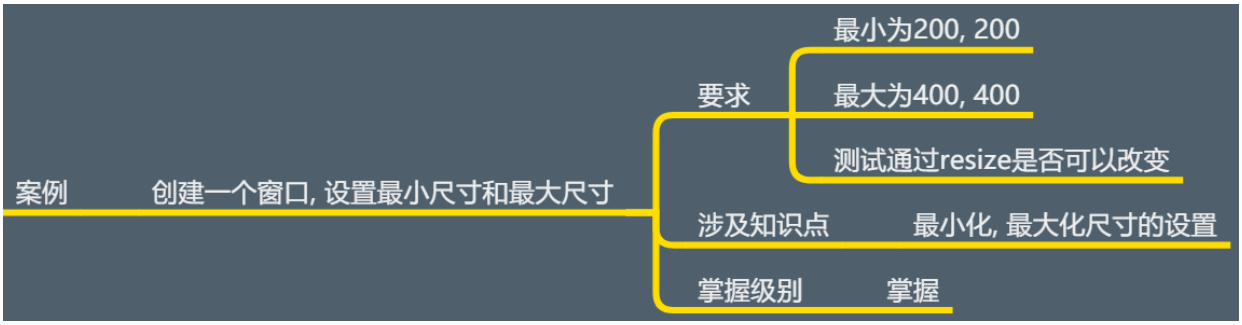
```python
import sys
from PyQt5.Qt import *

app = QApplication(sys.argv)

window = QWidget()

# 父控件展示之前先遍历其中所有子控件
# 所以在展示之后再设置子控件，子控件是无法展示的
# 这时子控件要手动展示
window.show()
window.resize(500, 500)
window.move(300, 300)

# 总的控件个数
widget_count = 50
# 一行有多少列
column_count = 4
# 计算一个控件的宽度
widget_width = window.width() / column_count
# 总共有多少行 (编号 // 一行多少列 + 1)
row_count = (widget_count - 1) // column_count + 1
# 计算一个控件的高度
widget_height = window.height() / row_count

for i in range(widget_count):
    # QWidget 是一个空白控件
```

```
        # 没有背景颜色，没有尺寸是看不到的
        w = QWidget(window)
        w.resize(widget_width, widget_height)
        widget_x = i % column_count * widget_width
        widget_y = i // column_count * widget_height
        w.move(widget_x, widget_y)
        w.setStyleSheet('background-color: red; border: 1px solid yellow;')
        w.show()

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

041. Python-GUI编程-PyQt5-QWidget-尺寸限定

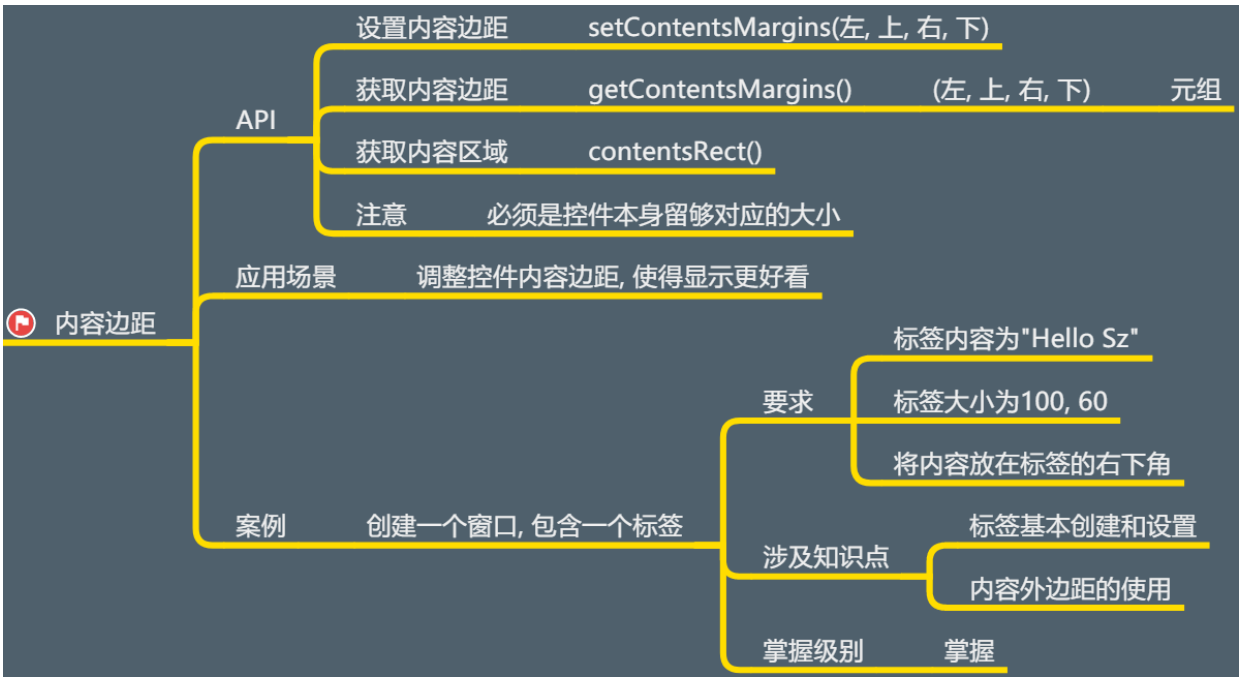


案例




```python
# 0. 导入需要的包和模块
import sys
from PyQt5.Qt import *

# 1. 创建一个应用程序对象
app = QApplication(sys.argv)

# 2. 控件的操作
# 2.1 创建控件
window = QWidget()
# 2.2 设置控件
window.setWindowTitle('尺寸限定API')
# window.resize(500, 500)
# 固定尺寸
# window.setFixedSize(500, 500)

# 整体限定最小最大尺寸
# window.setMinimumSize(300, 300)
# 如果不设置最大值，其实也有一个限定，不会大过桌面
window.setMaximumSize(600, 600)

# 单独限定尺寸
# window.setMinimumWidth(300)
# window.setMaximumWidth(600)

# 尝试能否通过 resize 来重新设置大小
# 不可以!
# 一旦通过 resize 修改的范围大于最大值
# 会被限定在最大值
window.resize(800, 800)

# 2.3 展示控件
window.show()

# 3. 应用程序的执行，进入到消息循环
sys.exit(app.exec_())
```

042. Python-GUI编程-PyQt5-QWidget-内容边距



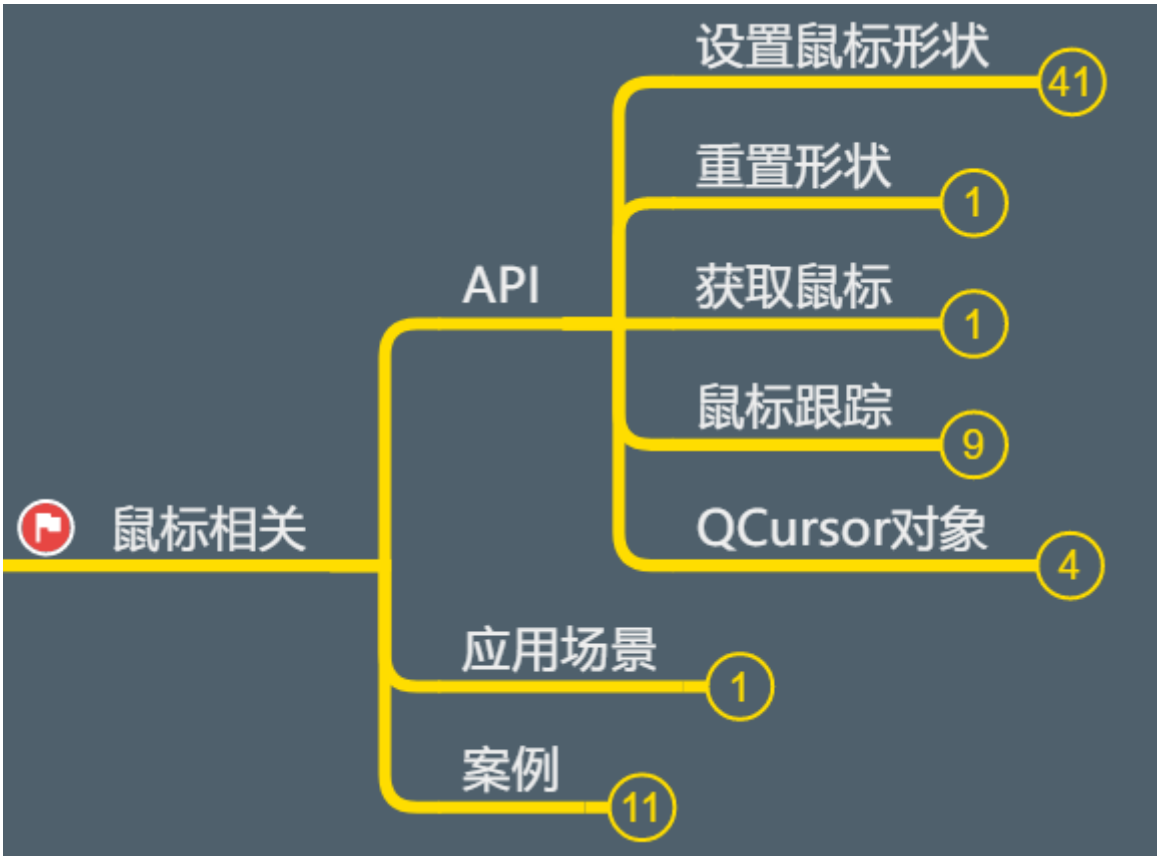

```python
# 0. 导入需要的包和模块
import sys
from PyQt5.Qt import *

# 1. 创建一个应用程序对象
app = QApplication(sys.argv)

# 2. 控件的操作
# 2.1 创建控件
window = QWidget()
# 2.2 设置控件
window.setWindowTitle('内容边距')
window.resize(500, 500)

label = QLabel(window)
# 默认是水平靠左，垂直居中
label.setText('天青色等烟雨，而我在等你')
label.resize(300, 300)
label.setStyleSheet('background-color: cyan;')
# 设置内容边距，左上右下
label.setContentsMargins(100, 200, 0, 0)
# 获取内容区域
print(label.contentsRect())
# 获取内容边距 - 注意跟上面获取内容区域是不同的!
print(label.getContentsMargins())

# 2.3 展示控件
window.show()
```

```
# 3. 应用程序的执行，进入到消息循环
sys.exit(app.exec_())
```

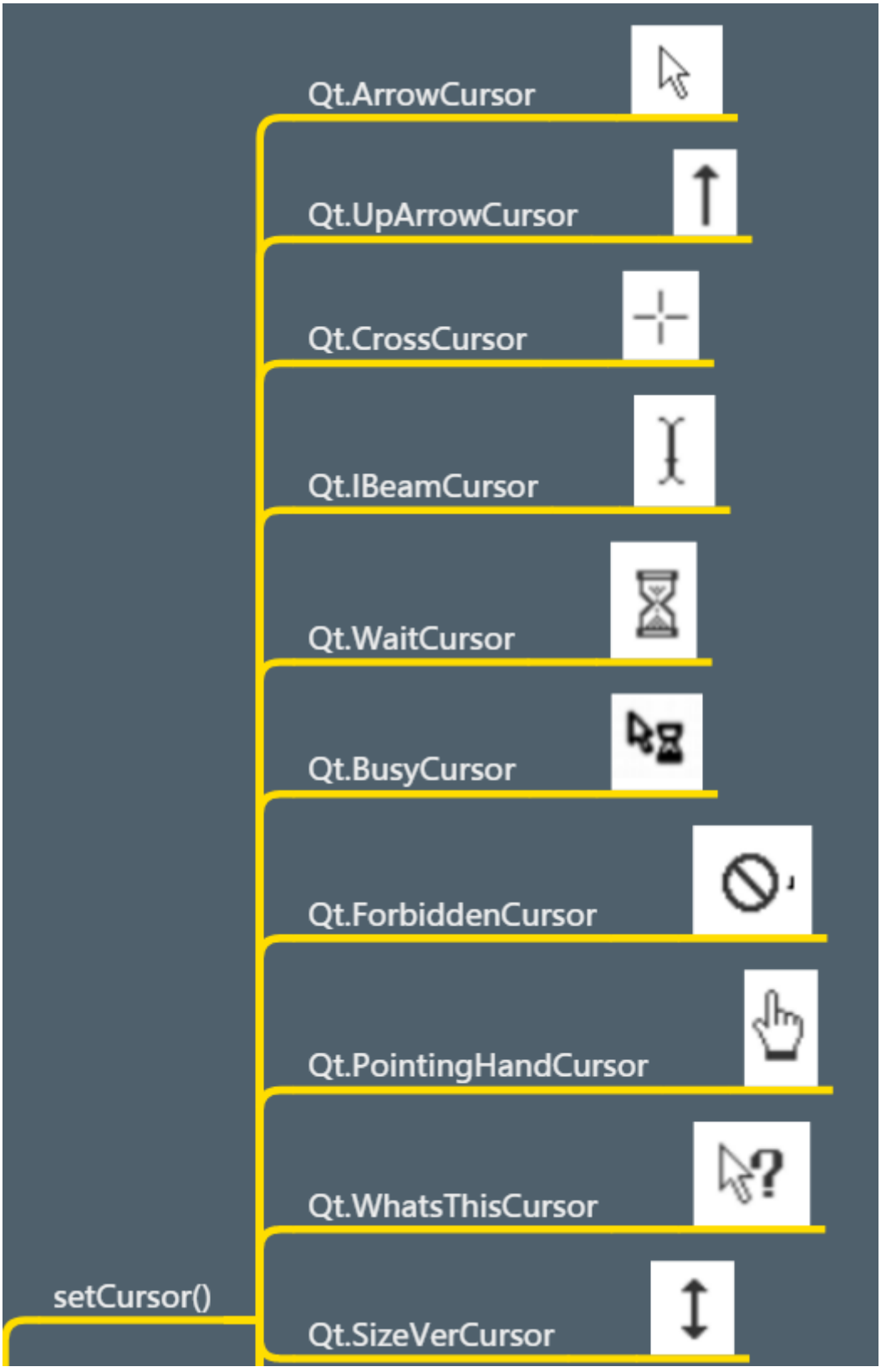
---

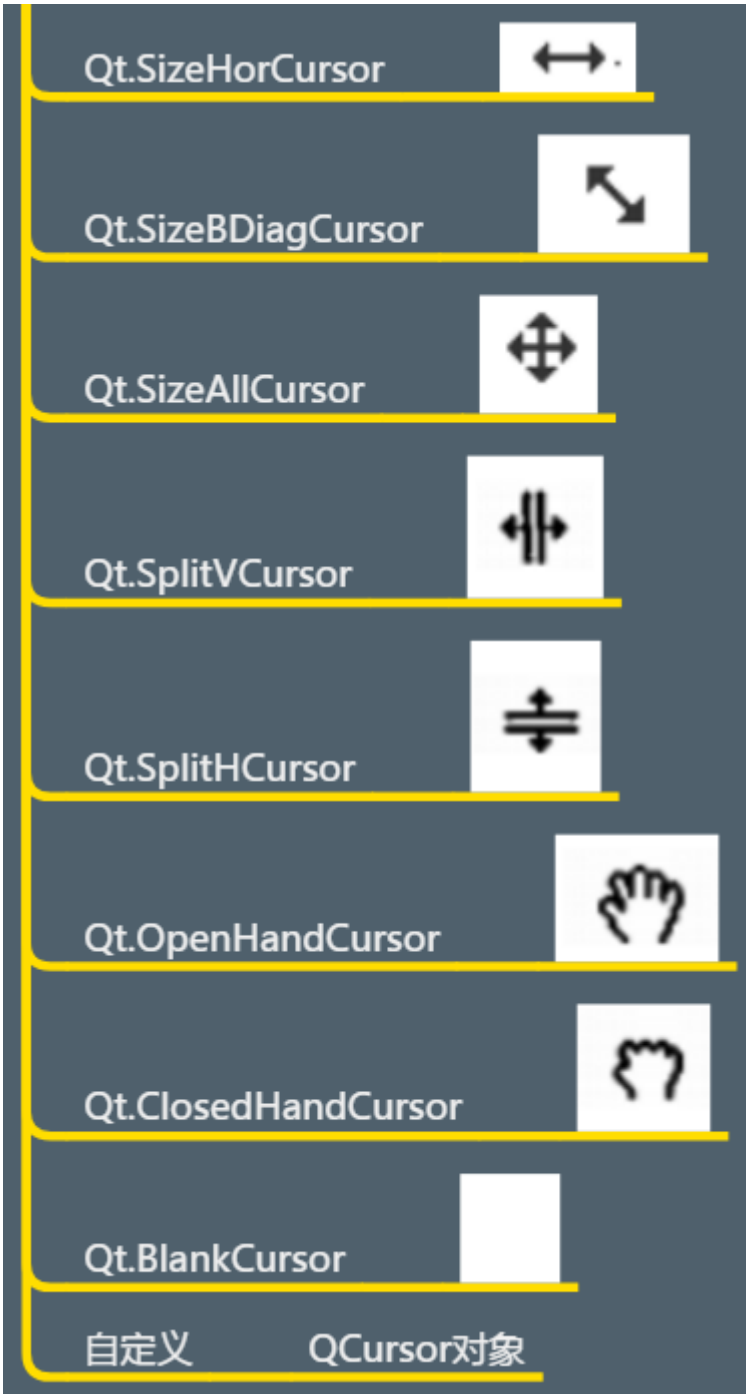
043. Python-GUI编程-PyQt5-QWidget-鼠标操作-形状设置

鼠标相关操作



设置鼠标形状


Qt.ArrowCursor
Qt.UpArrowCursor
Qt.CrossCursor
Qt.IBeamCursor
Qt.WaitCursor
Qt.BusyCursor
Qt.ForbiddenCursor
Qt.PointingHandCursor
Qt.WhatsThisCursor
Qt.SizeVerCursor
setCursor()



辅助工具：`QCursorDemo.exe`

示例代码1

```
# 0. 导入需要的包和模块
import sys
from PyQt5.Qt import *

# 1. 创建一个应用程序对象
app = QApplication(sys.argv)

# 2. 控件的操作
# 2.1 创建控件
window = QWidget()
# 2.2 设置控件
window.setWindowTitle('鼠标操作1')
```

```
window.resize(500, 500)

# 更改鼠标样式
# window.setCursor(Qt.BusyCursor)
window.setCursor(Qt.ForbiddenCursor)


# 2.3 展示控件
window.show()

# 3. 应用程序的执行，进入到消息循环
sys.exit(app.exec_())
```

示例代码2

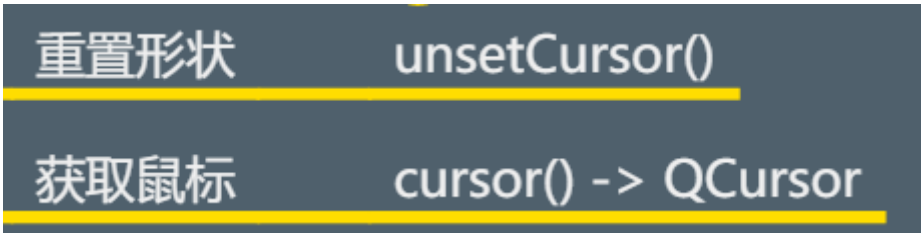
```
# 0. 导入需要的包和模块
import sys
from PyQt5.Qt import *

# 1. 创建一个应用程序对象
app = QApplication(sys.argv)

# 2. 控件的操作
# 2.1 创建控件
window = QWidget()
# 2.2 设置控件
window.setWindowTitle('鼠标操作2')
window.resize(500, 500)

label = QLabel(window)
label.setText('我是一个标签')
label.resize(100, 100)
label.setStyleSheet('background-color: cyan;')
# 更改鼠标样式，移动到标签上才会发生变化
label.setCursor(Qt.BusyCursor)

# 2.3 展示控件
window.show()

# 3. 应用程序的执行，进入到消息循环
sys.exit(app.exec_())
```

044. Python-GUI编程-PyQt5-QWidget-鼠标操作-自定义鼠标

示例代码

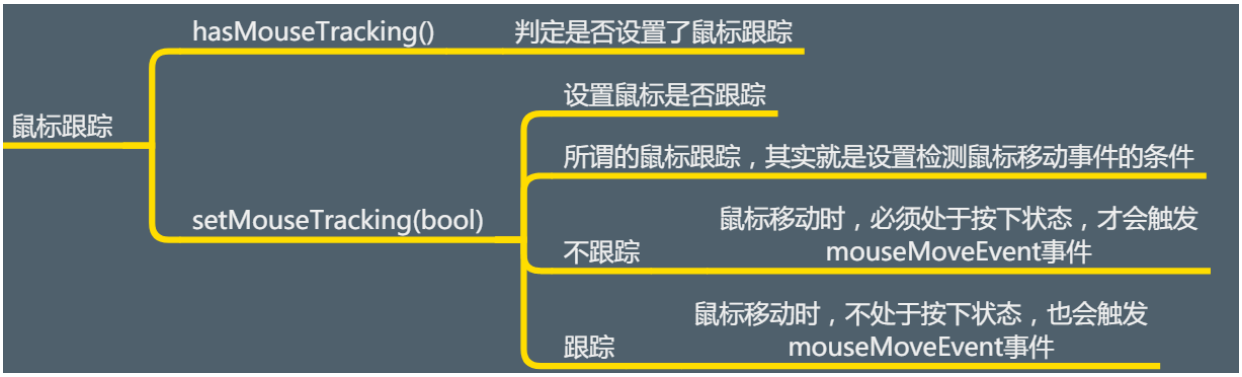
```
# 0. 导入需要的包和模块
import sys
from PyQt5.Qt import *

# 1. 创建一个应用程序对象
app = QApplication(sys.argv)

# 2. 控件的操作
# 2.1 创建控件
window = QWidget()
# 2.2 设置控件
window.setWindowTitle('自定义鼠标')
window.resize(500, 500)
# 自定义鼠标
# 1> 创建 QPixmap 对象并调整大小
pixmap = QPixmap('img/cursor.png')
# scaled方法不改变原来的 QPixmap 对象
# 而是会返回一个新的 QPixmap 对象
pixmap = pixmap.scaled(50, 50)
# 2> 创建 QCursor 对象
# QCursor(QPixmap, hotX: int = -1, hotY: int = -1)
# 热点x和热点y默认是 -1, -1: 大概是鼠标图片的中间位置
# 0, 0: 图片的左上角
# 假如是 50, 50: 图片的右下角
cursor = QCursor(pixmap, 0, 0)
# 3> 设置自定义鼠标
window.setCursor(cursor)

# 2.3 展示控件
window.show()

# 3. 应用程序的执行，进入到消息循环
sys.exit(app.exec_())
```

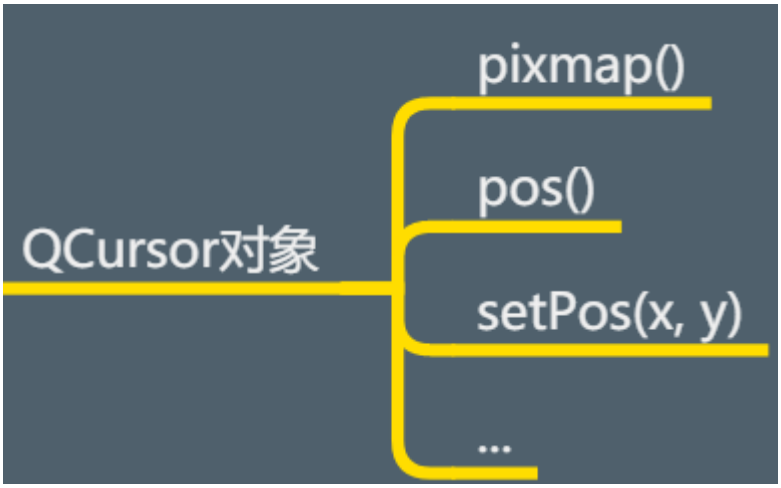
045. Python-GUI编程-PyQt5-QWidget-鼠标操作-鼠标重置获取




```python
# 0. 导入需要的包和模块
import sys
from PyQt5.Qt import *

# 1. 创建一个应用程序对象
app = QApplication(sys.argv)

# 2. 控件的操作
# 2.1 创建控件
window = QWidget()
# 2.2 设置控件
window.setWindowTitle('重置和获取鼠标')
window.resize(500, 500)

window.setCursor(Qt.BusyCursor)
# 重置鼠标 unsetCursor
window.unsetCursor()
# 获取鼠标 cursor，返回 QCursor 对象
# print(window.cursor())
current_cursor = window.cursor()
# pos() 是相对于整个电脑屏幕的位置
print(current_cursor.pos())
# setPos() 设置位置
current_cursor.setPos(0, 0)

# 2.3 展示控件
window.show()

# 3. 应用程序的执行，进入到消息循环
sys.exit(app.exec_())
```

046. Python-GUI编程-PyQt5-QWidget-鼠标操作-鼠标跟踪

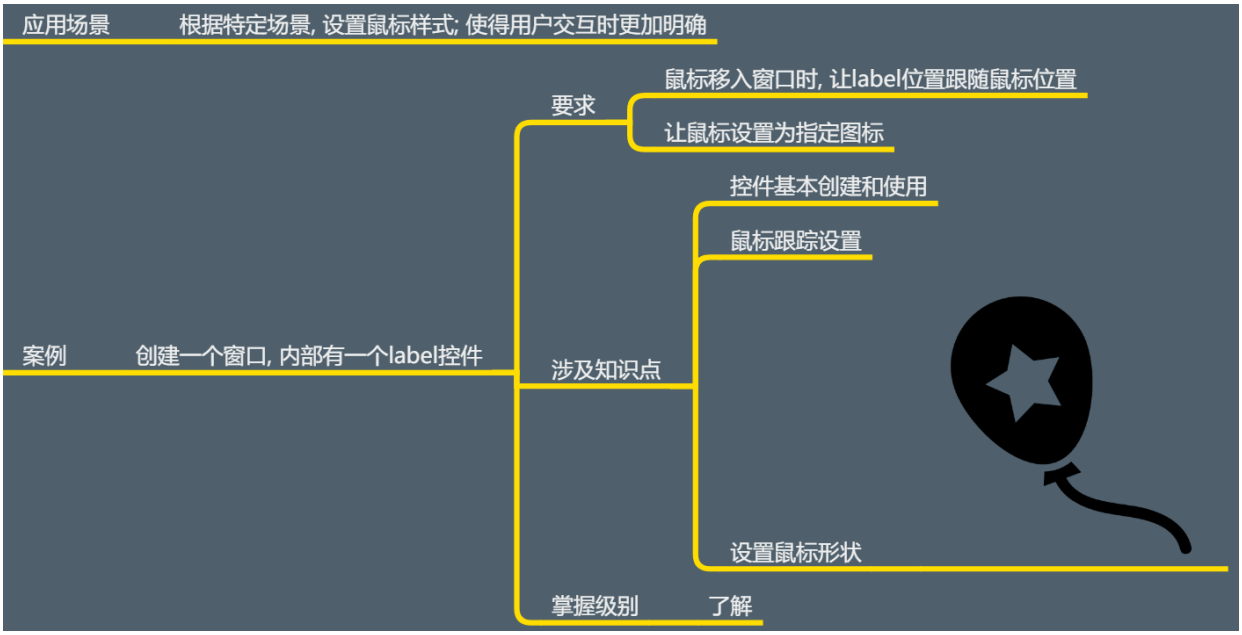





```python
# 0. 导入需要的包和模块
import sys
from PyQt5.Qt import *


class MyWindow(QWidget):
    # 参数类型是 QMouseEvent
    def mouseMoveEvent(self, me):
        # QMouseEvent 对象
        # 可以获取到是鼠标左键、右键、还是多个键被点击
        # globalPos: 鼠标相对整个电脑屏幕左上角的位置
        print('globalPos: ', me.globalPos())
        # globalX
        # globalY
        # localPos: 鼠标相对于父控件左上角的位置（不包括窗体框架）
        print('localPos: ', me.localPos())
        print('鼠标移动了')


# 1. 创建一个应用程序对象
app = QApplication(sys.argv)

# 2. 控件的操作
# 2.1 创建控件
window = MyWindow()
# 2.2 设置控件
window.setWindowTitle('鼠标跟踪')
window.resize(500, 500)
```

```
# hasMouseTracking 查看鼠标跟踪状态
print('鼠标跟踪：', window.hasMouseTracking())
# setMouseTracking(bool) 设置鼠标跟踪
window.setMouseTracking(True)
print('鼠标跟踪：', window.hasMouseTracking())

# 2.3 展示控件
window.show()

# 3. 应用程序的执行，进入到消息循环
sys.exit(app.exec_())
```

047. Python-GUI编程-PyQt5-QWidget-鼠标操作-鼠标跟踪案例




```
import sys
from PyQt5.Qt import *


class MyWindow(QWidget):
    def mouseMoveEvent(self, mv):
        print('鼠标移动：', mv.localPos())
        # 这一句也可以不用
        label = self.findChild(QLabel)
        label.move(mv.localPos().x(), mv.localPos().y())


app = QApplication(sys.argv)
```

```
window = MyWindow()
window.setWindowTitle('鼠标操作案例')
window.resize(500, 500)

# 设置鼠标跟踪
window.setMouseTracking(True)
# 自定义鼠标
pixmap = QPixmap('img/cursor.png').scaled(50, 50)
cursor = QCursor(pixmap, 0, 0)
window.setCursor(cursor)

label = QLabel(window)
label.setText('枯藤老树昏鸦')
label.move(100, 100)
label.setStyleSheet('background-color: cyan; font-size: 30px;')



window.show()

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

封装成对象

```
import sys
from PyQt5.Qt import *


class MyWindow(QWidget):
    def __init__(self):
        super().__init__()
        self.setWindowTitle('鼠标操作案例')
        self.resize(500, 500)
        # 设置鼠标跟踪
        self.setMouseTracking(True)
        # 自定义鼠标
        pixmap = QPixmap('img/cursor.png').scaled(50, 50)
        cursor = QCursor(pixmap, 0, 0)
        self.setCursor(cursor)
        self.label = QLabel(self)
        self.label.setText('枯藤老树昏鸦')
        self.label.move(100, 100)
        self.label.setStyleSheet('background-color: cyan; font-size: 30p
x;')

    def mouseMoveEvent(self, mv):
        print('鼠标移动：', mv.localPos())
        self.label.move(mv.localPos().x(), mv.localPos().y())
```

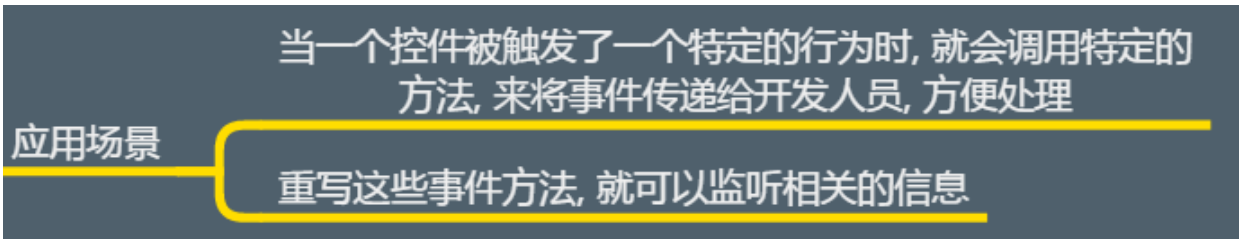
```
app = QApplication(sys.argv)

window = MyWindow()
window.show()

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

---

048. Python-GUI编程-PyQt5-QWidget-事件消息-作用

事件 `API`

自定义美观的空格，就需要实现 `paintEvent(QPaintEvent)`




```
from PyQt5.Qt import *


class Window(QWidget):
    def __init__(self):
        super().__init__()
        self.setWindowTitle('事件消息的学习')
        self.resize(500, 500)
        self.setup_ui()

    def setup_ui(self):
        pass

    def showEvent(self, QShowEvent):
        print('窗口被展示了！')

    def closeEvent(self, QCloseEvent):
        print('窗口被关闭了！')

    def moveEvent(self, QMoveEvent):
        print('窗口被移动了！')

    def resizeEvent(self, QResizeEvent):
        print('窗口大小被调整了！')
```

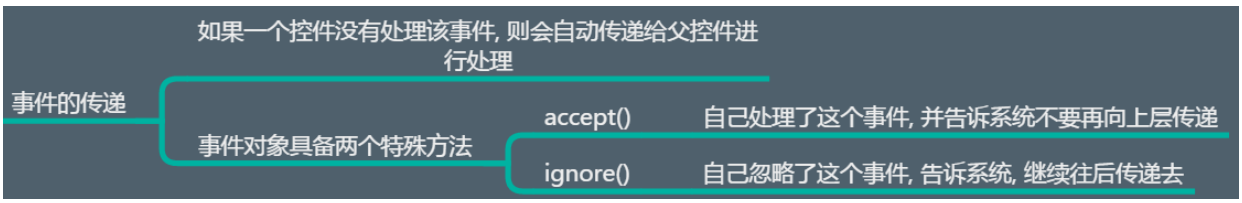
```
    def enterEvent(self, QEvent):
        print('鼠标进来了！')
        self.setStyleSheet('background-color: yellow;')

    def leaveEvent(self, QEvent):
        print('鼠标离开了！')
        self.setStyleSheet('background-color: green;')

    def mousePressEvent(self, QMouseEvent):
        # 可以通过 QMouseEvent 这个事件对象获取到底是左键还是右键被点击
        print('鼠标被按下了！')

    def mouseReleaseEvent(self, QMouseEvent):
        print('鼠标被释放了！')

    def mouseDoubleClickEvent(self, QMouseEvent):
        print('鼠标双击！')

    def mouseMoveEvent(self, QMouseEvent):
        # 必须要按下鼠标左键或者右键才会响应
        # 如果在不按键的时候追踪鼠标
        # 需要设置
        # self.setMouseTracking(True)
        print('鼠标被移动了！')

    # 单击信号：必须是在控件范围内被按下，在控件范围内被释放才算

    def keyPressEvent(self, QKeyEvent):
        # 在 QKeyEvent 这个事件对象可以找到具体是哪个键被按下了
        print('键盘被按下！')

    def keyReleaseEvent(self, QKeyEvent):
        print('键盘被释放！')


if __name__ == '__main__':
    import sys
    app = QApplication(sys.argv)

    window = Window()
    window.show()
    sys.exit(app.exec_())
```

049. Python-GUI编程-PyQt5-QWidget-事件转发机制

事件会从子控件到父控件转发。
注意是转发到父控件，而不是转发到父类。

```
# 0. 导入需要的包和模块
import sys
from PyQt5.Qt import *


class Window(QWidget):
    def mousePressEvent(self, QMouseEvent):
        print('顶层窗口：鼠标按下！')


class MidWindow(QWidget):
    # 如果中间控件不处理鼠标按下事件
    # 则会转发事件给顶层窗口
    # pass
    def mousePressEvent(self, evt):
        print('中间控件：鼠标按下！')


class Label(QLabel):
    # 如果标签不处理鼠标按下事件
    # 则会转发事件给中间控件
    # pass
    def mousePressEvent(self, evt):
        print('标签控件：鼠标按下！')
        # 每个事件里面有两个方法可以标识
        # evt.accept() 告诉系统，事件已经被处理，不要转发
        evt.accept()
        print('标签鼠标点击事件是否被处理：', evt.isAccepted())
        # evt.ignore() 将事件标识为未处理，所以会向父控件转发
        evt.ignore()
        print('标签鼠标点击事件是否被处理：', evt.isAccepted())


# 1. 创建一个应用程序对象
app = QApplication(sys.argv)

# 2. 控件的操作
# 2.1 创建控件
window = Window()
# 2.2 设置控件
window.setWindowTitle('事件转发')
window.resize(500, 500)

# 中间控件
mid_window = MidWindow(window)
mid_window.resize(300, 300)
# 让 qss 样式生效
mid_window.setAttribute(Qt.WA_StyledBackground, True)
```

```
mid_window.setStyleSheet('background-color: yellow;')

# 标签
# 标签的作用在于展示内容给用户
# 如果使用默认的 QLabel
# 这个类是没有处理 mousePressEvent 的
# 所以会向父控件转发事件
label = Label(mid_window)
label.setText('我是标签')
label.setStyleSheet('background-color: red;')
label.move(100, 100)

# 按钮
# 按钮本身的作用就是监听用户的点击
# 它会在自己类里面的 mousePressEvent 处理事件
# 所以就不会再转发给父控件
btn = QPushButton(mid_window)
btn.setText('我是按钮')
btn.move(50, 50)

# 2.3 展示控件
window.show()

# 3. 应用程序的执行，进入到消息循环
sys.exit(app.exec_())
```

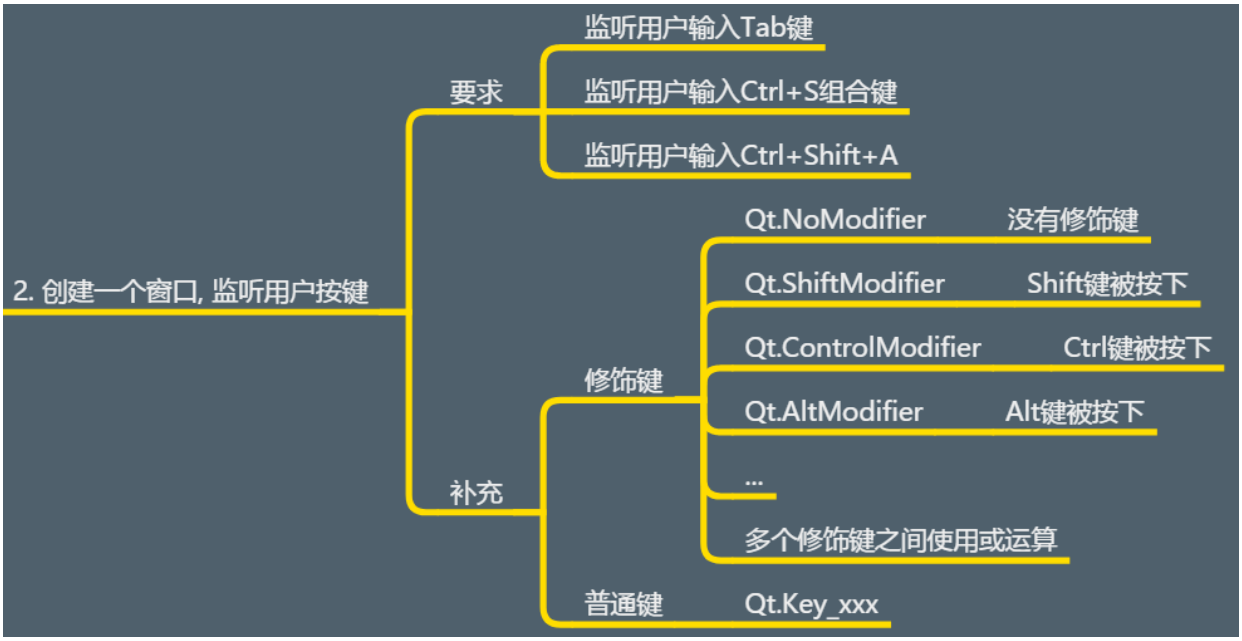


050. Python-GUI编程-PyQt5-QWidget-事件-案例1


```
# 0. 导入需要的包和模块
import sys
from PyQt5.Qt import *
```

```
class Label(QLabel):
    def enterEvent(self, *args, **kwargs):
        self.setText('欢迎光临！')
        print('鼠标进入')

    def leaveEvent(self, *args, **kwargs):
        self.setText('谢谢惠顾！')
        print('鼠标离开')


# 1. 创建一个应用程序对象
app = QApplication(sys.argv)

# 2. 控件的操作
# 2.1 创建控件
window = QWidget()
# 2.2 设置控件
window.setWindowTitle('鼠标操作案例1')
window.resize(500, 500)

label = Label(window)
label.resize(200, 200)
label.move(100, 100)
label.setStyleSheet('background-color: cyan;')


# 2.3 展示控件
window.show()

# 3. 应用程序的执行，进入到消息循环
sys.exit(app.exec_())
```

051. Python-GUI编程-PyQt5-QWidget-事件-案例2



参考：

修饰键：`Ctrl`
普通键：`Tab` 、 `abc` 等等

`Qt.Key_` 加上一个后缀，就是普通键

```
# 0. 导入需要的包和模块
import sys
from PyQt5.Qt import *


class Label(QLabel):
    def keyPressEvent(self, evt):
        if evt.key() == Qt.Key_Tab:
            print('用户点击了 Tab 键! ')
            self.setText('监听到 Tab 键')
        # 修饰键为 Ctrl，普通键为 S
        if evt.modifiers() == Qt.ControlModifier and evt.key() == Qt.Key_
S:
            print('用户点击了 Ctrl - S 组合键! ')
            self.setText('监听到 Ctrl - S 组合键! ')
        # Ctrl - Shift - A
        # 多个修饰键使用按位或!
        # 前面 modifiers() 返回了 8421 码，比如 5, 7, 3 等等
        # 后面的按位或运算的组合键可以求出 8421 码
        # 3 == 2 | 1
        # 10 ---- 2
        # 01 ---- 1
        # 11 ---- 3
```

```
        if evt.modifiers() == Qt.ControlModifier | Qt.ShiftModifier and e
vt.key() == Qt.Key_A:
            print('用户点击了 Ctrl - Shift - A 组合键! ')
            self.setText('监听到 Ctrl - Shift - A 组合键! ')


# 1. 创建一个应用程序对象
app = QApplication(sys.argv)

# 2. 控件的操作
# 2.1 创建控件
window = QWidget()
# 2.2 设置控件
window.setWindowTitle('键盘操作案例')
window.resize(500, 500)

label = Label(window)
label.resize(400, 200)
label.move(50, 100)
label.setStyleSheet('background-color: cyan; font-size: 20px;')
# 让标签捕获键盘
label.grabKeyboard()

# 2.3 展示控件
window.show()

# 3. 应用程序的执行，进入到消息循环
sys.exit(app.exec_())
```

---

052. Python-GUI编程-PyQt5-QWidget-事件-案例3-窗口移动

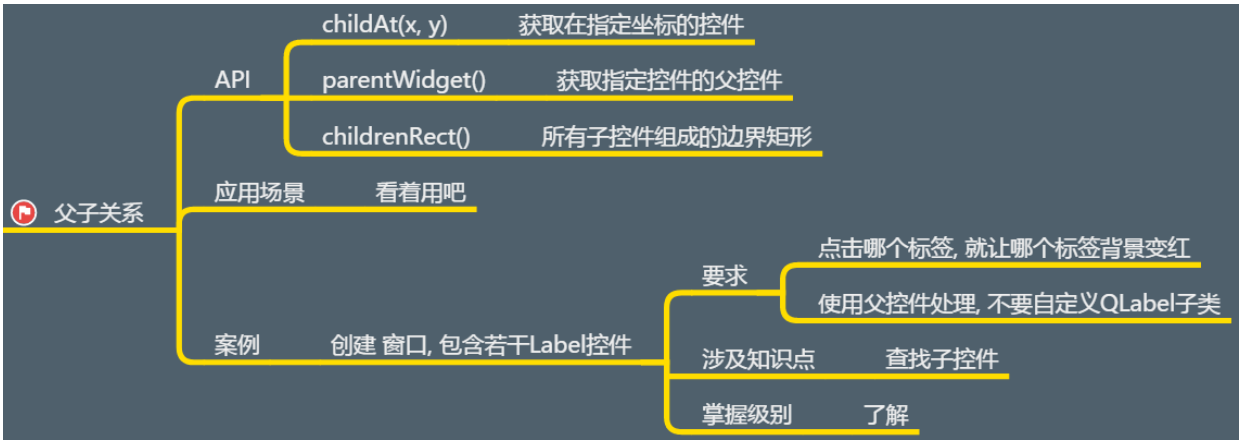
`QMouseEvent` 事件
`button` 鼠标点的是哪个键
`globalPos` 鼠标点的点相对于整个桌面左上角的位置
`globalX` 全局 `X`
`globalY` 全局 `Y`
`localPos` 相对于窗口的位置

```
from PyQt5.Qt import *
```

```
class Window(QWidget):
    def __init__(self):
        super().__init__()
        self.setWindowTitle('窗口移动案例')
        self.resize(500, 500)
        self.setup_ui()
        self.move_flag = False

    def setup_ui(self):
        pass

    def mousePressEvent(self, evt):
        # 只让鼠标左键点击的时候移动
        if evt.button() == Qt.LeftButton:
            self.move_flag = True
        # print('鼠标按下! ')
        # 确定两个点
        # 1) 鼠标第一次按下的点
        self.mouse_x = evt.globalX()
        self.mouse_y = evt.globalY()
        # 2) 窗口当前所在的原始点
        self.origin_x = self.x()
        self.origin_y = self.y()

    def mouseMoveEvent(self, evt):
        # print('鼠标移动! ')
        if self.move_flag:
            # 每次移动，都能获取最新的 globalX 和 globalY
            self.move_x = evt.globalX() - self.mouse_x
            self.move_y = evt.globalY() - self.mouse_y
            self.dest_x = self.move_x + self.origin_x
            self.dest_y = self.move_y + self.origin_y
            self.move(self.dest_x, self.dest_y)

    def mouseReleaseEvent(self, evt):
        # print('鼠标释放! ')
        # 在这里移动就不会实时移动了!
        # self.move(self.origin_x + self.move_x, self.origin_y + self.mov
e_y)

        self.move_flag = False


if __name__ == '__main__':
    import sys
    app = QApplication(sys.argv)

    window = Window()
    window.show()
    # 这样程序就崩了!
    # 因为直接就进入了 mouseMoveEvent
    # 而没有经过 mousePressEvent
```

```
    # 所以要在 mousePressEvent 里面加入 self.move_flag 标记
    window.setMouseTracking(True)
    sys.exit(app.exec_())
```

053. Python-GUI编程-PyQt5-QWidget-父子关系扩充-API

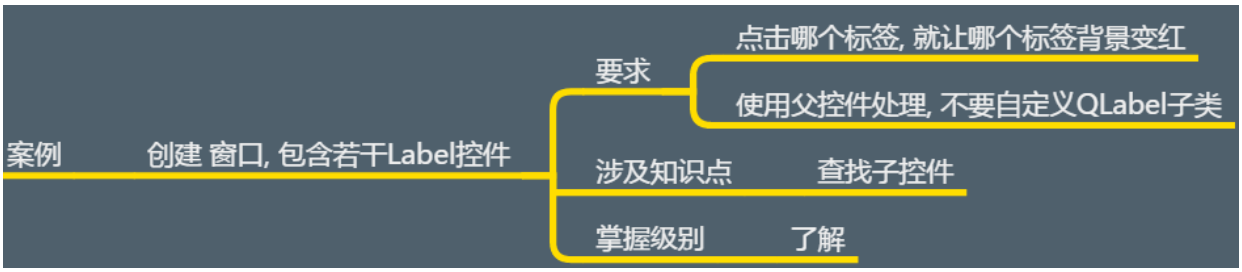



```
# 0. 导入需要的包和模块
import sys
from PyQt5.Qt import *

# 1. 创建一个应用程序对象
app = QApplication(sys.argv)

# 2. 控件的操作
# 2.1 创建控件
window = QWidget()
# 2.2 设置控件
window.setWindowTitle('QWidget父子关系学习')
window.resize(500, 500)

label1 = QLabel(window)
# 也可以这样写
# label1.setParent(window)
label1.setText('标签1')

label2 = QLabel(window)
label2.setText('标签2')
label2.move(50, 50)

label3 = QLabel(window)
label3.setText('标签3')
```

```
label3.move(100, 100)

# 需求：查看 55, 55 坐标点有没有子控件
# 注意：这个是相对于父控件左上角的位置
# 使用 childAt(x, y)
print(window.childAt(55, 55))

# parentWidget() 查看父控件
print(label2.parentWidget())

# childrenRect() 查看所有子控件组成的边界矩形
print(window.childrenRect())

# 2.3 展示控件
window.show()

# 3. 应用程序的执行，进入到消息循环
sys.exit(app.exec_())
```

---

054. Python-GUI编程-PyQt5-QWidget-父子关系扩充-案例



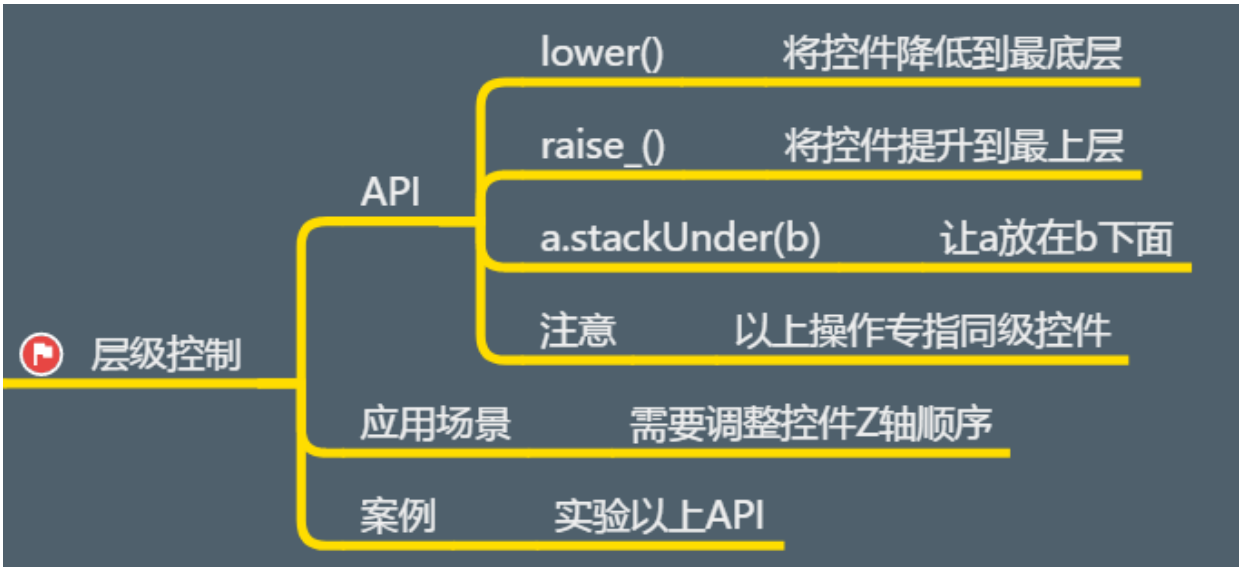
方法1
自定义标签控件，改变其鼠标点击事件，更改标签背景颜色。

方法2
自定义 `QWidget` ，用 `childAt` 查找坐标处有没有子控件

```
# 0. 导入需要的包和模块
import sys
from PyQt5.Qt import *


# 方案1：自定义 QLabel
# 这个方法的弊端：
# 一开始有很多标签并没有使用自定义的 Label 创建
# 后期要全部改过来比较复杂
# class Label(QLabel):
```

```
#     def mousePressEvent(self, evt):
#         self.setStyleSheet('background-color: red;')


# 方案2：自定义 QWidget 父控件
# 如果 QLabel 的点击事件没有被处理
# 则会传递到父控件
class Window(QWidget):
    def mousePressEvent(self, evt):
        # 这里要取局部坐标，因为 childAt 使用的就是相对于窗口的坐标
        local_x = evt.x()
        local_y = evt.y()
        sub_widget = self.childAt(local_x, local_y)
        # 如果不加判定，会崩溃
        # 因为假如没有获取到子控件（比如点击窗口空白部分）
        # 那么就是一个 None
        # 对 None 使用 setStyleSheet 显然是不行的
        # if sub_widget:
        if sub_widget is not None:
            sub_widget.setStyleSheet('background-color: red;')


# 1. 创建一个应用程序对象
app = QApplication(sys.argv)

# 2. 控件的操作
# 2.1 创建控件
window = Window()
# 2.2 设置控件
window.setWindowTitle('父子关系案例')
window.resize(500, 500)

for i in range(10):
    label = QLabel(window)
    label.setText('标签' + str(i + 1))
    label.move((i + 1) * 40, (i + 1) * 40)

# 2.3 展示控件
window.show()

# 3. 应用程序的执行，进入到消息循环
sys.exit(app.exec_())
```

055. Python-GUI编程-PyQt5-QWidget-层级关系-案例

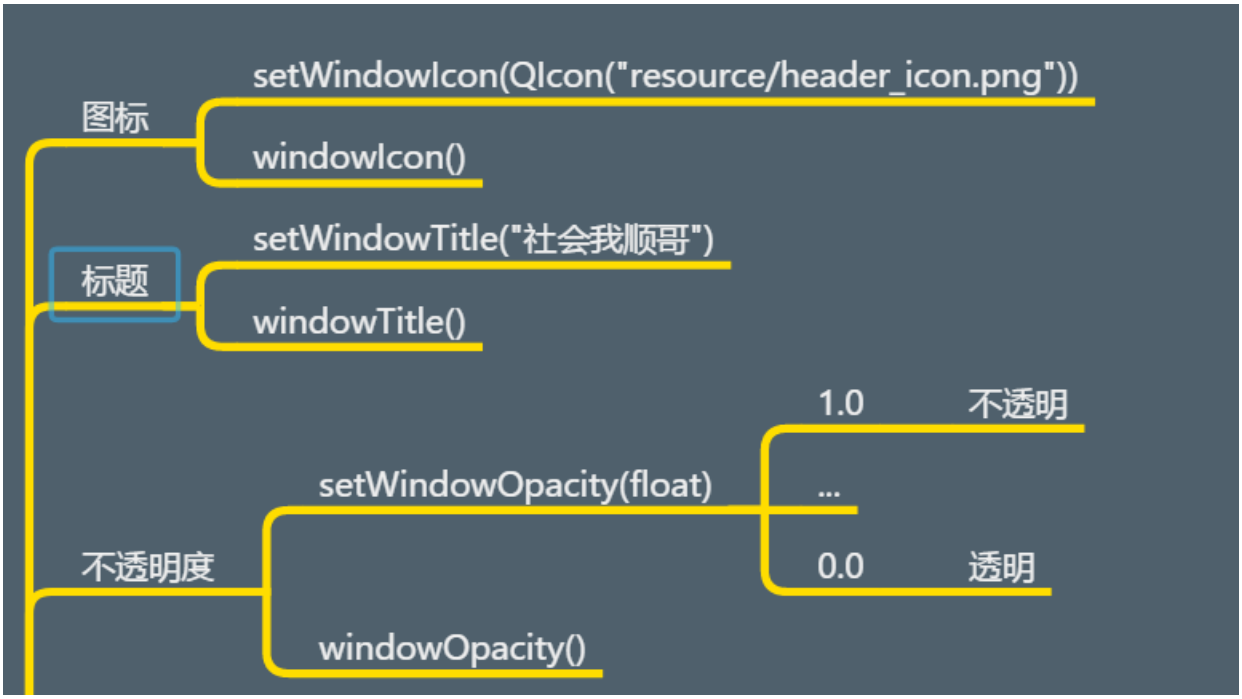


后添加的靠近我们，在上层，上层会遮挡住下层

```python
# 0. 导入需要的包和模块
import sys
from PyQt5.Qt import *


# 需求：点谁谁上来
# 方法1：自定义标签控件，检测鼠标点击事件
# class Label(QLabel):
#     def mousePressEvent(self, evt):
#         self.raise_()


# 方法2：父控件查找子控件的方式
class Window(QWidget):
    def mousePressEvent(self, evt):
        widget = self.childAt(evt.x(), evt.y())
        if widget is not None:
            widget.raise_()


# 1. 创建一个应用程序对象
app = QApplication(sys.argv)

# 2. 控件的操作
# 2.1 创建控件
window = Window()
# 2.2 设置控件
window.setWindowTitle('层级关系调整')
window.resize(500, 500)

label1 = QLabel(window)
label1.setText('标签1')
label1.resize(200, 200)
label1.setStyleSheet('background-color: red;')
```

```python
# 默认情况，后添加的在上面
label2 = QLabel(window)
label2.setText('标签2')
label2.resize(200, 200)
label2.setStyleSheet('background-color: green;')
label2.move(100, 100)

# 想让红色的 label1 在上面
# 方法1：使用 lower
# label2.lower()

# 方法2：使用 raise_
# label1.raise_()

# 方法3：使用 a.stackUnder(b)
# label2.stackUnder(label1)

# 2.3 展示控件
window.show()

# 3. 应用程序的执行，进入到消息循环
sys.exit(app.exec_())
```

056. Python-GUI编程-PyQt5-QWidget-窗口特定操作-图标标题不透明度

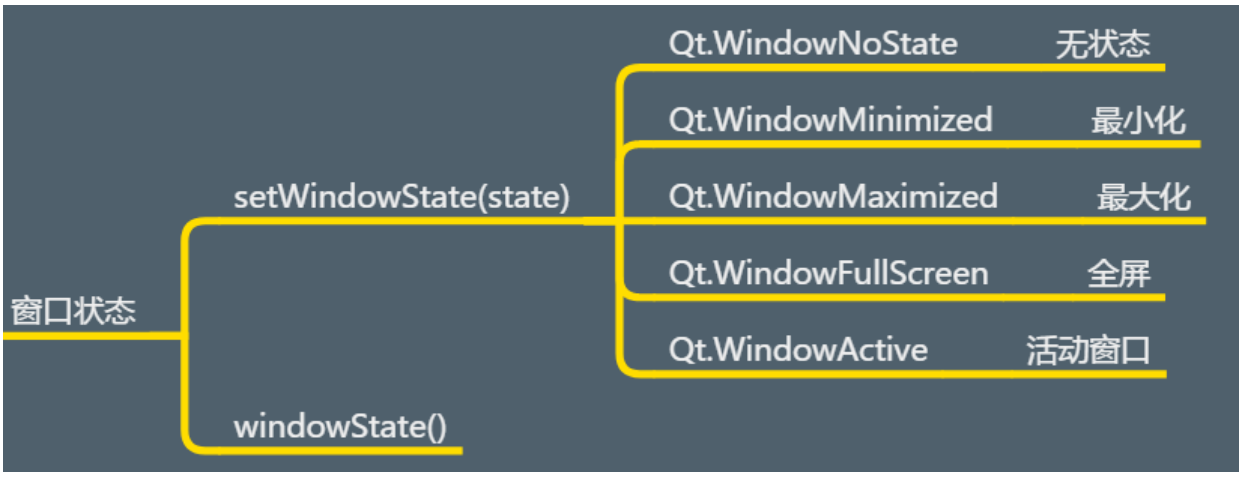


`PyCharm` 的窗口标题

learnpyqt5 [D:\code\learnpyqt5] - ...\04_QWidget_窗口相关操作.py [learnpyqt5] - PyCharm

```python
# 0. 导入需要的包和模块
import sys
from PyQt5.Qt import *

# 1. 创建一个应用程序对象
app = QApplication(sys.argv)

# 2. 控件的操作
# 2.1 创建控件
window = QWidget()
# 2.2 设置控件
window.resize(500, 500)

# 设置窗口图标
icon = QIcon('img/Python.png')
window.setWindowIcon(icon)
# 获取窗口图标
print(window.windowIcon())

# 设置窗口标题
# 如果不写，则默认为 python
# 如果写的是空字符串，也是默认 python
# 设置为空格，相当于清空了标题
window.setWindowTitle('窗口相关操作')
# 获取窗口标题
print(window.windowTitle())

# 设置窗口不透明度
# 浮点类型的数据，0.0 到 1.0
# 1.0 不透明
# 0.0 透明
window.setWindowOpacity(0.9)
# 获取不透明度
# 大概是 0.9，处理的时候有点偏差
print(window.windowOpacity())

# 2.3 展示控件
window.show()

# 3. 应用程序的执行，进入到消息循环
sys.exit(app.exec_())
```

057. Python-GUI编程-PyQt5-QWidget-窗口特定操作-窗口状态




```python
# 0. 导入需要的包和模块
import sys
from PyQt5.Qt import *

# 1. 创建一个应用程序对象
app = QApplication(sys.argv)

# 2. 控件的操作
# 2.1 创建控件
window = QWidget()
# 2.2 设置控件
window.setWindowTitle('窗口1')
window.resize(500, 500)

# windowState() 获取窗口状态
# 默认无状态 Qt.WindowNoState
print(window.windowState() == Qt.WindowNoState)

# setWindowState 设置窗口状态
# 最小化
# window.setWindowState(Qt.WindowMinimized)
# 最大化
# window.setWindowState(Qt.WindowMaximized)
# 全屏
# window.setWindowState(Qt.WindowFullScreen)

window2 = QWidget()
window2.setWindowTitle('窗口2')

# 2.3 展示控件
# 谁后显示，谁离我们更近!
window.show()
window2.show()
# 或者控制窗口活跃状态
```

```
# setWindowState(Qt.WindowActive)
window.setWindowState(Qt.WindowActive)

# 3. 应用程序的执行，进入到消息循环
sys.exit(app.exec_())
```

058. Python-GUI编程-PyQt5-QWidget-窗口特定操作-最大化最小化

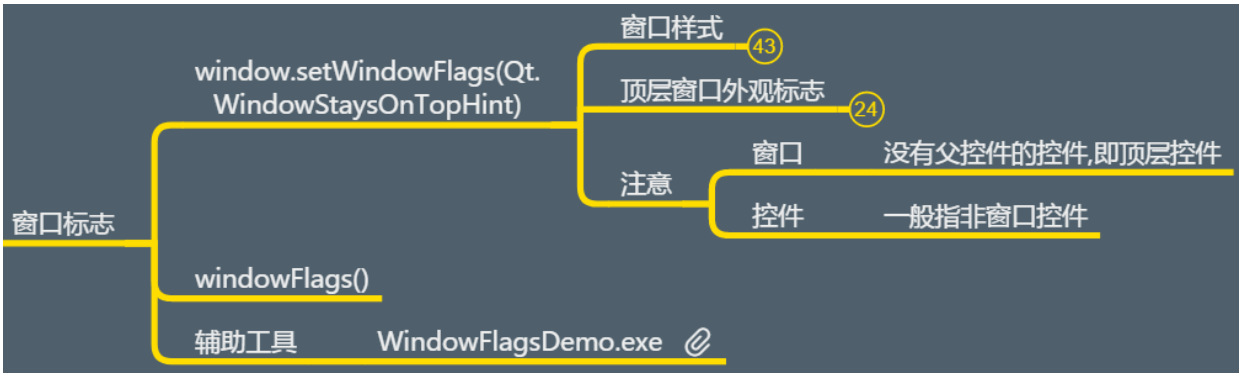
应用场景：比如自定义了一个窗口，需要定制最小化、最大化和关闭按钮

如何判定窗口状态
方法1：使用 `windowState()`
方法2：使用 `isMinimized()` 等等

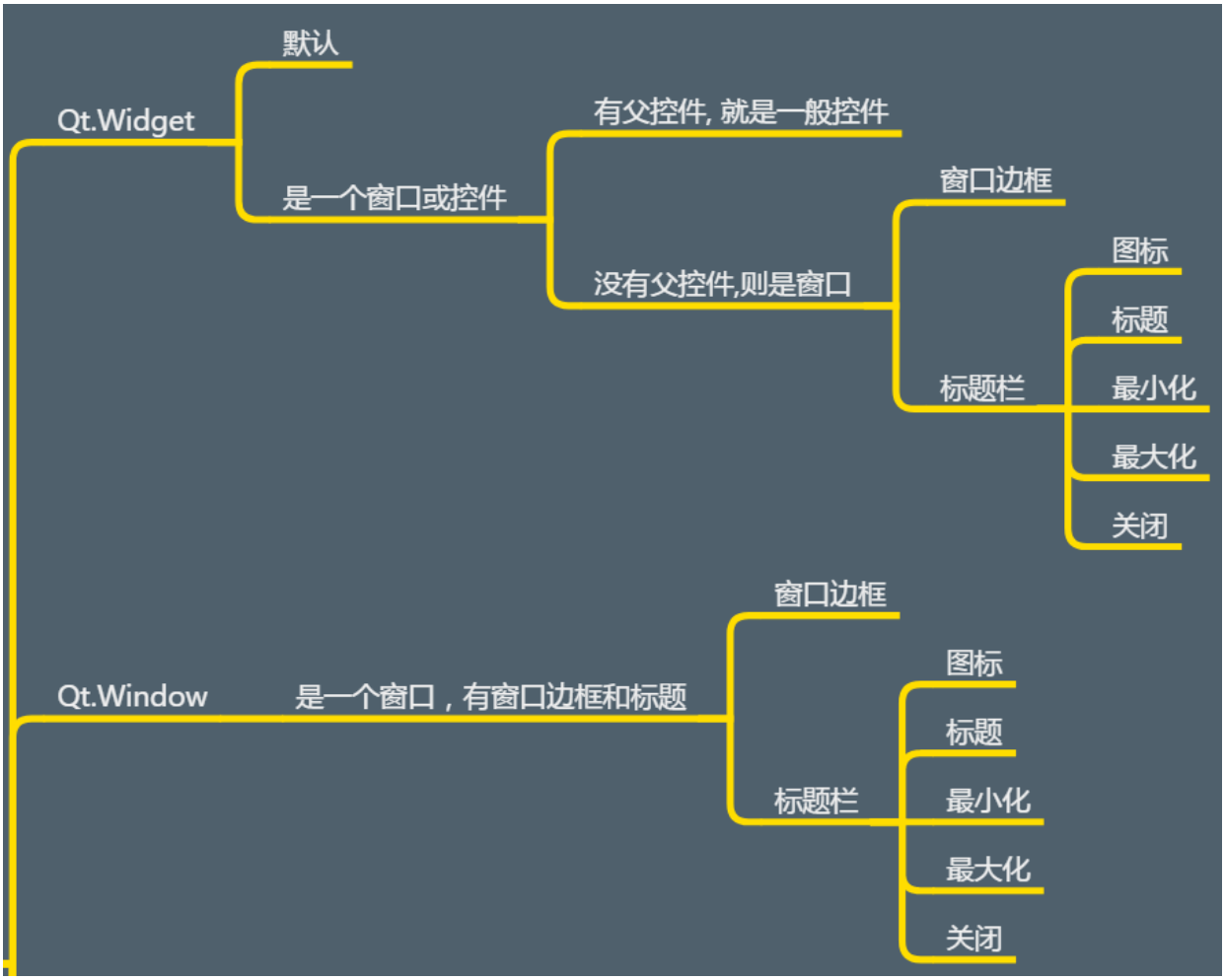
```
# 0. 导入需要的包和模块
import sys
from PyQt5.Qt import *


# 需求：一旦点击窗口，就最大化
class Window(QWidget):
    def mousePressEvent(self, evt):
        if self.isMaximized():
            self.showNormal()
        else:
            self.showMaximized()
```

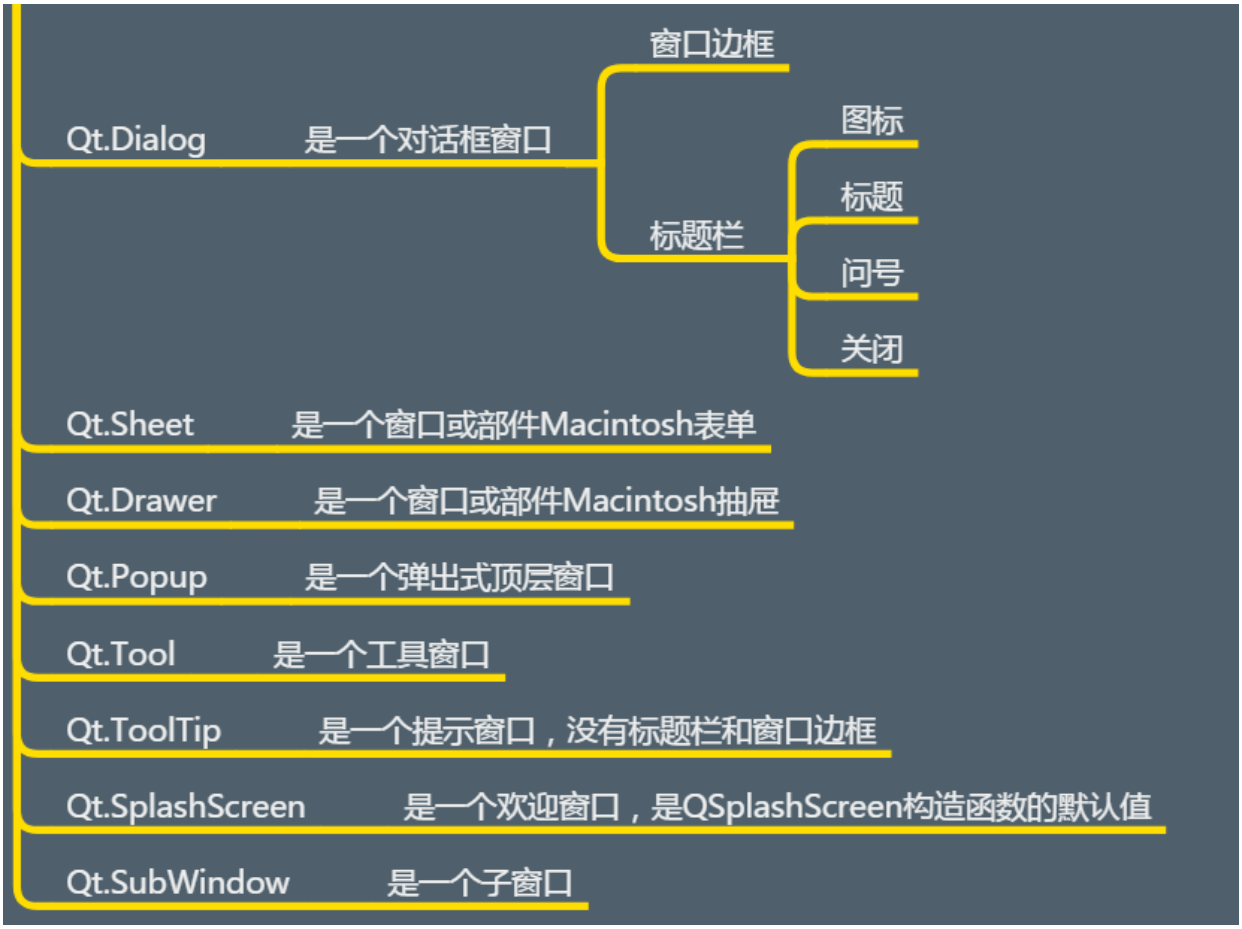
```
# 1. 创建一个应用程序对象
app = QApplication(sys.argv)

# 2. 控件的操作
# 2.1 创建控件
window = Window()
# 2.2 设置控件
window.setWindowTitle('最小化最大化')
window.resize(500, 500)

# showMaximized() 最大化
# 注意：这样就不用 show() 展示了
# window.showMaximized()

# showFullScreen() 全屏
# window.showFullScreen()

# showMinimized() 最小化
# window.showMinimized()

# 2.3 展示控件
window.show()

# 3. 应用程序的执行，进入到消息循环
sys.exit(app.exec_())
```

059. Python-GUI编程-PyQt5-QWidget-窗口特定操作-窗口标志



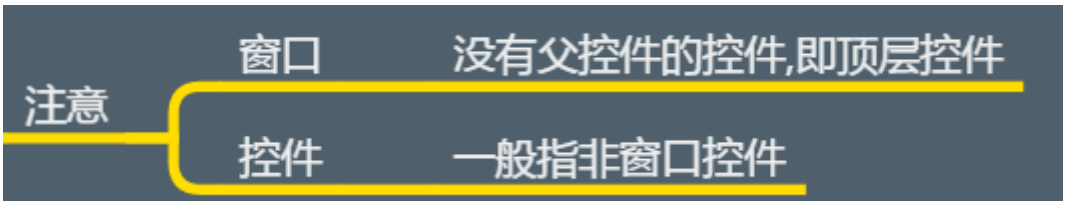
窗口样式


Qt.Widget
默认
是一个窗口或控件
有父控件, 就是一般控件
没有父控件,则是窗口
窗口边框
标题栏
图标
标题
最小化
最大化
关闭
Qt.Window
是一个窗口，有窗口边框和标题
窗口边框
标题栏
图标
标题
最小化
最大化
关闭


Qt.Dialog
是一个对话框窗口
窗口边框
标题栏
图标
标题
问号
关闭
Qt.Sheet
是一个窗口或部件Macintosh表单
Qt.Drawer
是一个窗口或部件Macintosh抽屉
Qt.Popup
是一个弹出式顶层窗口
Qt.Tool
是一个工具窗口
Qt.ToolTip
是一个提示窗口，没有标题栏和窗口边框
Qt.SplashScreen
是一个欢迎窗口，是QSplashScreen构造函数的默认值
Qt.SubWindow
是一个子窗口

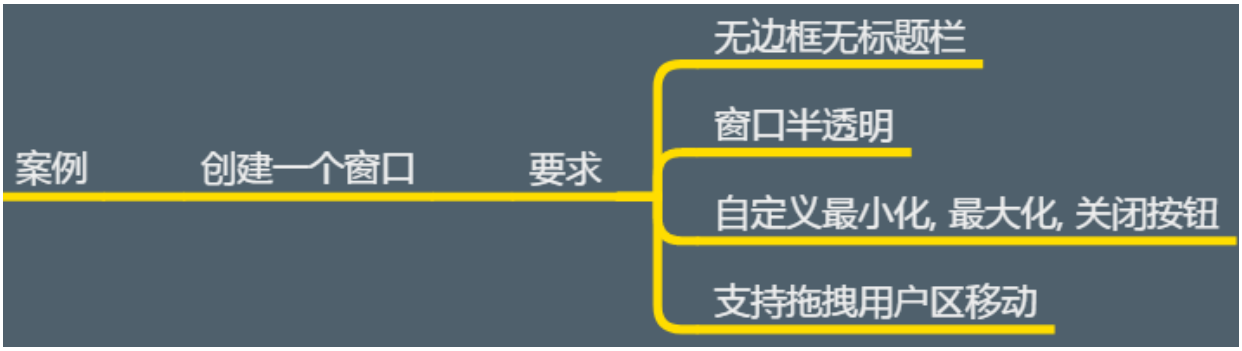
顶层窗口样式

注意



---

060. Python-GUI编程-PyQt5-QWidget-窗口特定操作-案例-步骤1




```python
# 0. 导入需要的包和模块
import sys
from PyQt5.Qt import *

# 1. 创建一个应用程序对象
app = QApplication(sys.argv)
```

```
# 2. 控件的操作
# 2.1 创建控件
# 方法1：在创建的时候设置 flags
# window = QWidget(flags=Qt.FramelessWindowHint)
# 方法2：使用 setWindowFlags()
# 注意：setWindowFlag() 也可以，设置单个标志
window = QWidget()
window.setWindowFlags(Qt.FramelessWindowHint)

# 不透明度设置
window.setWindowOpacity(0.9)

# 2.2 设置控件
window.setWindowTitle('顶层窗口操作案例')
window.resize(500, 500)

# 2.3 展示控件
window.show()

# 3. 应用程序的执行，进入到消息循环
sys.exit(app.exec_())
```

---

061. Python-GUI编程-PyQt5-QWidget-窗口特定操作-案例-步骤2

```
# 0. 导入需要的包和模块
import sys
from PyQt5.Qt import *

# 1. 创建一个应用程序对象
app = QApplication(sys.argv)

# 2. 控件的操作
# 2.1 创建控件
# 方法1：在创建的时候设置 flags
# window = QWidget(flags=Qt.FramelessWindowHint)
# 方法2：使用 setWindowFlags()
# 注意：setWindowFlag() 也可以，设置单个标志
window = QWidget()
window.setWindowFlags(Qt.FramelessWindowHint)

# 不透明度设置
window.setWindowOpacity(0.9)

# 2.2 设置控件
```

```
window.setWindowTitle('顶层窗口操作案例')
window.resize(500, 500)

# 添加 3 个子控件按钮 - 窗口的右上角
top_margin = 0
btn_w = 40
btn_h = 40

close_btn = QPushButton(window)
close_btn.setText('关闭')
close_btn.resize(btn_w, btn_h)
window_w = window.width()
close_btn_x = window_w - btn_w
close_btn_y = top_margin
close_btn.move(close_btn_x, close_btn_y)

max_btn = QPushButton(window)
max_btn.setText('最大化')
max_btn.resize(btn_w, btn_h)
max_btn_x = close_btn_x - btn_w
max_btn_y = top_margin
max_btn.move(max_btn_x, max_btn_y)

mini_btn = QPushButton(window)
mini_btn.setText('最小化')
mini_btn.resize(btn_w, btn_h)
mini_btn_x = max_btn_x - btn_w
mini_btn_y = top_margin
mini_btn.move(mini_btn_x, mini_btn_y)

# 2.3 展示控件
window.show()

# 3. 应用程序的执行，进入到消息循环
sys.exit(app.exec_())
```

---

062. Python-GUI编程-PyQt5-QWidget-窗口特定操作-案例-步骤3

监听按钮的点击

- 方法1：使用信号（推荐）
- 方法2：使用事件

```
# 0. 导入需要的包和模块
```

```python
import sys
from PyQt5.Qt import *

# 1. 创建一个应用程序对象
app = QApplication(sys.argv)

# 2. 控件的操作
# 2.1 创建控件
# 方法1：在创建的时候设置 flags
# window = QWidget(flags=Qt.FramelessWindowHint)
# 方法2：使用 setWindowFlags()
# 注意：setWindowFlag() 也可以，设置单个标志
window = QWidget()
window.setWindowFlags(Qt.FramelessWindowHint)

# 不透明度设置
window.setWindowOpacity(0.9)

# 2.2 设置控件
window.setWindowTitle('顶层窗口操作案例')
window.resize(500, 500)

# 添加 3 个子控件按钮 - 窗口的右上角
top_margin = 0
btn_w = 40
btn_h = 40

close_btn = QPushButton(window)
close_btn.setText('关闭')
close_btn.resize(btn_w, btn_h)
window_w = window.width()
close_btn_x = window_w - btn_w
close_btn_y = top_margin
close_btn.move(close_btn_x, close_btn_y)

max_btn = QPushButton(window)
max_btn.setText('最大化')
max_btn.resize(btn_w, btn_h)
max_btn_x = close_btn_x - btn_w
max_btn_y = top_margin
max_btn.move(max_btn_x, max_btn_y)

mini_btn = QPushButton(window)
mini_btn.setText('最小化')
mini_btn.resize(btn_w, btn_h)
mini_btn_x = max_btn_x - btn_w
mini_btn_y = top_margin
mini_btn.move(mini_btn_x, mini_btn_y)


def max_normal():
    if window.isMaximized():
```

```
            window.showNormal()
            max_btn.setText('最大化')
        else:
            window.showMaximized()
            max_btn.setText('还原')


    close_btn.pressed.connect(window.close)
    max_btn.pressed.connect(max_normal)
    mini_btn.pressed.connect(window.showMinimized)

    # 2.3 展示控件
    window.show()

    # 3. 应用程序的执行，进入到消息循环
    sys.exit(app.exec_())
```

063. Python-GUI编程-PyQt5-QWidget-窗口特定操作-案例-步骤4

进行代码重构
讲白了就是**调整代码结构**
一般就是把代码封装起来，叫做**封装重构**

```
# 0. 导入需要的包和模块
import sys
from PyQt5.Qt import *


class Window(QWidget):
    def __init__(self, *args, **kwargs):
        super().__init__(*args, **kwargs)
        # 方法1：在创建的时候设置 flags
        # window = QWidget(flags=Qt.FramelessWindowHint)
        # 方法2：使用 setWindowFlags()
        # 注意：setWindowFlag() 也可以，设置单个标志
        self.setWindowFlags(Qt.FramelessWindowHint)

        # 不透明度设置
        self.setWindowOpacity(0.9)

        # 设置控件
        self.setWindowTitle('顶层窗口操作案例')
        self.resize(500, 500)
```

```
        self.setup_ui()

    def setup_ui(self):
        # 公共数据
        top_margin = 0
        btn_w = 40
        btn_h = 40

        # 添加 3 个子控件按钮 - 窗口的右上角
        close_btn = QPushButton(self)
        close_btn.setText('关闭')
        close_btn.resize(btn_w, btn_h)
        window_w = self.width()
        close_btn_x = window_w - btn_w
        close_btn_y = top_margin
        close_btn.move(close_btn_x, close_btn_y)

        max_btn = QPushButton(self)
        max_btn.setText('最大化')
        max_btn.resize(btn_w, btn_h)
        max_btn_x = close_btn_x - btn_w
        max_btn_y = top_margin
        max_btn.move(max_btn_x, max_btn_y)

        mini_btn = QPushButton(self)
        mini_btn.setText('最小化')
        mini_btn.resize(btn_w, btn_h)
        mini_btn_x = max_btn_x - btn_w
        mini_btn_y = top_margin
        mini_btn.move(mini_btn_x, mini_btn_y)

        def max_normal():
            if self.isMaximized():
                self.showNormal()
                max_btn.setText('最大化')
            else:
                self.showMaximized()
                max_btn.setText('还原')

        close_btn.pressed.connect(self.close)
        max_btn.pressed.connect(max_normal)
        mini_btn.pressed.connect(self.showMinimized)


# 1. 创建一个应用程序对象
app = QApplication(sys.argv)
window = Window()

# 2 展示控件
window.show()

# 3. 应用程序的执行，进入到消息循环
```

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

064. Python-GUI编程-PyQt5-QWidget-窗口特定操作-案例-步骤5

监听窗口尺寸大小的变化，然后再调整三个按钮的位置
可以用 `resizeEvent` 事件对象实现。

注意：当程序第一次运行的时候，会进行一次 `resizeEvent`。

```
# 0. 导入需要的包和模块
import sys
from PyQt5.Qt import *


class Window(QWidget):
    def __init__(self, *args, **kwargs):
        super().__init__(*args, **kwargs)
        # 方法1：在创建的时候设置 flags
        # window = QWidget(flags=Qt.FramelessWindowHint)
        # 方法2：使用 setWindowFlags()
        # 注意：setWindowFlag() 也可以，设置单个标志
        self.setWindowFlags(Qt.FramelessWindowHint)

        # 不透明度设置
        self.setWindowOpacity(0.9)

        # 设置控件
        self.setWindowTitle('顶层窗口操作案例')
        self.resize(500, 500)

        # 公共数据
        self.top_margin = 0
        self.btn_w = 40
        self.btn_h = 40

        self.setup_ui()

    def setup_ui(self):
        # 添加 3 个子控件按钮 - 窗口的右上角
        close_btn = QPushButton(self)
        self.close_btn = close_btn
        close_btn.setText('关闭')
        close_btn.resize(self.btn_w, self.btn_h)
```

```
        max_btn = QPushButton(self)
        self.max_btn = max_btn
        max_btn.setText('最大化')
        max_btn.resize(self.btn_w, self.btn_h)

        mini_btn = QPushButton(self)
        self.mini_btn = mini_btn
        mini_btn.setText('最小化')
        mini_btn.resize(self.btn_w, self.btn_h)

        def max_normal():
            if self.isMaximized():
                self.showNormal()
                max_btn.setText('最大化')
            else:
                self.showMaximized()
                max_btn.setText('还原')

        close_btn.pressed.connect(self.close)
        max_btn.pressed.connect(max_normal)
        mini_btn.pressed.connect(self.showMinimized)

    def resizeEvent(self, evt):
        # print('窗口大小改变了')

        window_w = self.width()
        close_btn_x = window_w - self.btn_w
        close_btn_y = self.top_margin
        self.close_btn.move(close_btn_x, close_btn_y)

        max_btn_x = close_btn_x - self.btn_w
        max_btn_y = self.top_margin
        self.max_btn.move(max_btn_x, max_btn_y)

        mini_btn_x = max_btn_x - self.btn_w
        mini_btn_y = self.top_margin
        self.mini_btn.move(mini_btn_x, mini_btn_y)


# 1. 创建一个应用程序对象
app = QApplication(sys.argv)
window = Window()

# 2 展示控件
window.show()

# 3. 应用程序的执行，进入到消息循环
sys.exit(app.exec_())
```

065. Python-GUI编程-PyQt5-QWidget-窗口特定操作-案例-步骤6

```
# 0. 导入需要的包和模块
import sys
from PyQt5.Qt import *


class Window(QWidget):
    def __init__(self, *args, **kwargs):
        super().__init__(*args, **kwargs)
        # 方法1：在创建的时候设置 flags
        # window = QWidget(flags=Qt.FramelessWindowHint)
        # 方法2：使用 setWindowFlags()
        # 注意：setWindowFlag() 也可以，设置单个标志
        self.setWindowFlags(Qt.FramelessWindowHint)

        # 不透明度设置
        self.setWindowOpacity(0.9)

        # 设置控件
        self.setWindowTitle('顶层窗口操作案例')
        self.resize(500, 500)
        self.move_flag = False

        # 公共数据
        self.top_margin = 0
        self.btn_w = 40
        self.btn_h = 40

        self.setup_ui()

    def setup_ui(self):
        # 添加 3 个子控件按钮 - 窗口的右上角
        close_btn = QPushButton(self)
        self.close_btn = close_btn
        close_btn.setText('关闭')
        close_btn.resize(self.btn_w, self.btn_h)

        max_btn = QPushButton(self)
        self.max_btn = max_btn
        max_btn.setText('最大化')
        max_btn.resize(self.btn_w, self.btn_h)

        mini_btn = QPushButton(self)
        self.mini_btn = mini_btn
        mini_btn.setText('最小化')
        mini_btn.resize(self.btn_w, self.btn_h)
```

```
        def max_normal():
            if self.isMaximized():
                self.showNormal()
                max_btn.setText('最大化')
            else:
                self.showMaximized()
                max_btn.setText('还原')

        close_btn.pressed.connect(self.close)
        max_btn.pressed.connect(max_normal)
        mini_btn.pressed.connect(self.showMinimized)

    def resizeEvent(self, evt):
        # print('窗口大小改变了')

        window_w = self.width()
        close_btn_x = window_w - self.btn_w
        close_btn_y = self.top_margin
        self.close_btn.move(close_btn_x, close_btn_y)

        max_btn_x = close_btn_x - self.btn_w
        max_btn_y = self.top_margin
        self.max_btn.move(max_btn_x, max_btn_y)

        mini_btn_x = max_btn_x - self.btn_w
        mini_btn_y = self.top_margin
        self.mini_btn.move(mini_btn_x, mini_btn_y)

    def mousePressEvent(self, evt):
        # 设置标记，用于判定是否需要移动
        # 并判定是否点击了鼠标左键
        # 注意：这个判等不能用 is
        if evt.button() == Qt.LeftButton:
            self.move_flag = True
        # 窗口的原始坐标点（左上角）
        # 注意：之前这里用了 self.x 和 self.y
        # 程序报错！我怀疑是 self.x 和 self.y 已经被 PyQt 占用
        self.origin_x = self.x()
        self.origin_y = self.y()
        # 鼠标按下的点
        self.mouse_x = evt.globalX()
        self.mouse_y = evt.globalY()

    def mouseMoveEvent(self, evt):
        # 如果窗口移动标记 = True
        if self.move_flag:
            # 根据鼠标按下的点，计算移动向量
            move_x = evt.globalX() - self.mouse_x
            move_y = evt.globalY() - self.mouse_y
            # 根据移动向量和窗口原始坐标，计算最新坐标
            new_x = self.origin_x + move_x
            new_y = self.origin_y + move_y
```

```
                # 移动整个窗口的位置
                self.move(new_x, new_y)

    def mouseReleaseEvent(self, evt):
        # 重置窗口移动标记 = False
        self.move_flag = False


# 1. 创建一个应用程序对象
app = QApplication(sys.argv)
window = Window()

# 2 展示控件
window.show()

# 3. 应用程序的执行，进入到消息循环
sys.exit(app.exec_())
```

我最后又做了一个拓展：为按钮加上了图片

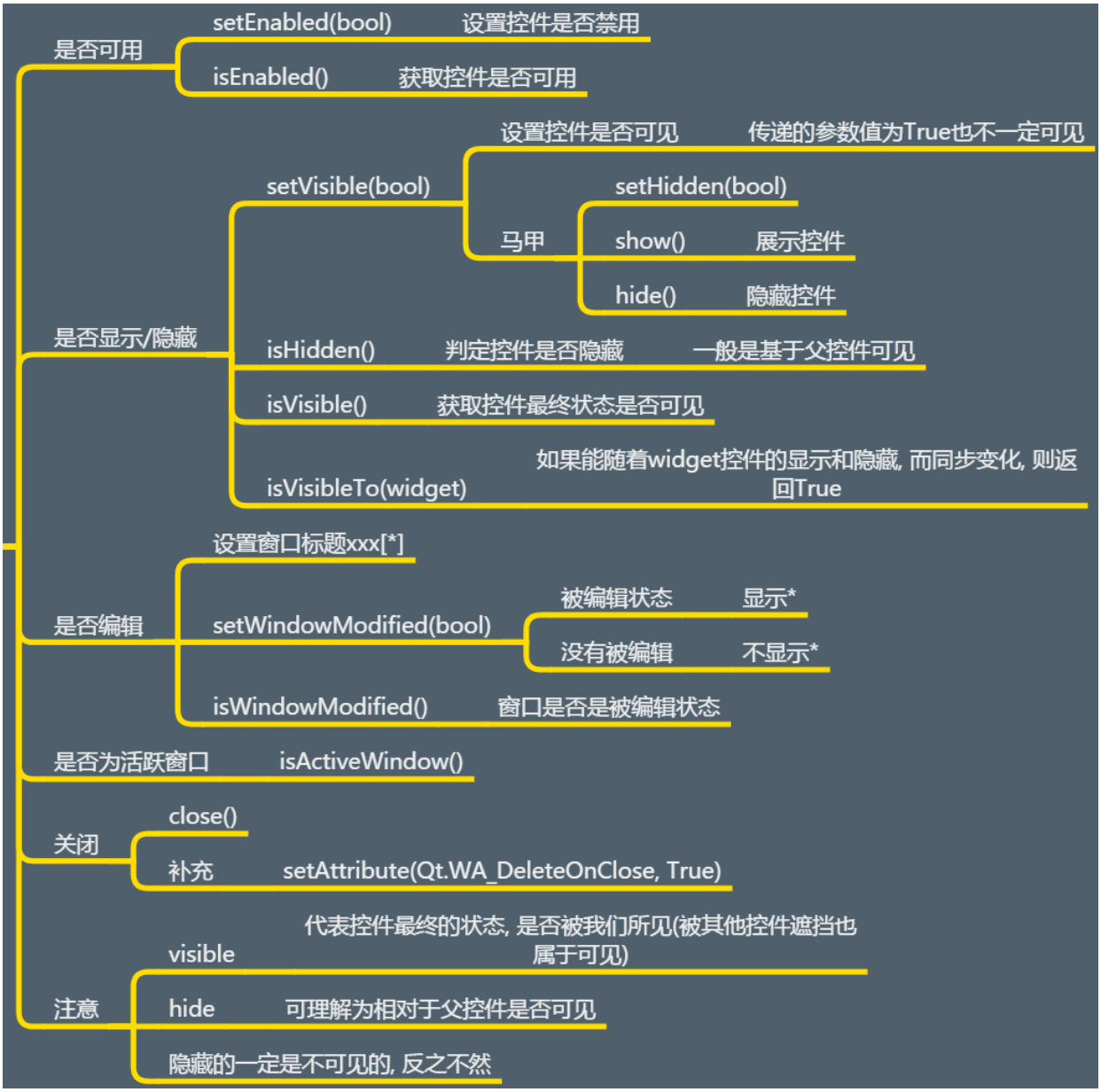
```
# 0. 导入需要的包和模块
import sys
from PyQt5.Qt import *


class Window(QWidget):
    def __init__(self, *args, **kwargs):
        super().__init__(*args, **kwargs)
        # 方法1：在创建的时候设置 flags
        # window = QWidget(flags=Qt.FramelessWindowHint)
        # 方法2：使用 setWindowFlags()
        # 注意：setWindowFlag() 也可以，设置单个标志
        self.setWindowFlags(Qt.FramelessWindowHint)

        # 不透明度设置
        self.setWindowOpacity(0.9)

        # 设置控件
        self.setWindowTitle('顶层窗口操作案例')
        self.resize(500, 500)
        self.move_flag = False

        # 公共数据
        self.top_margin = 0
        self.btn_w = 40
        self.btn_h = 40

        self.setup_ui()
```

```
def setup_ui(self):
    # 添加 3 个子控件按钮 - 窗口的右上角
    close_icon = QIcon('img/close.png')
    close_btn = QPushButton(close_icon, '', self)
    self.close_btn = close_btn
    # close_btn.setText('关闭')
    close_btn.resize(self.btn_w, self.btn_h)

    max_icon = QIcon('img/maximize.png')
    max_btn = QPushButton(max_icon, '', self)
    self.max_btn = max_btn
    # max_btn.setText('最大化')
    max_btn.resize(self.btn_w, self.btn_h)

    mini_icon = QIcon('img/minimize.png')
    mini_btn = QPushButton(mini_icon, '', self)
    self.mini_btn = mini_btn
    # mini_btn.setText('最小化')
    mini_btn.resize(self.btn_w, self.btn_h)

    def max_normal():
        if self.isMaximized():
            self.showNormal()
            # max_btn.setText('最大化')
        else:
            self.showMaximized()
            # max_btn.setText('还原')

    close_btn.pressed.connect(self.close)
    max_btn.pressed.connect(max_normal)
    mini_btn.pressed.connect(self.showMinimized)

def resizeEvent(self, evt):
    # print('窗口大小改变了')

    window_w = self.width()
    close_btn_x = window_w - self.btn_w
    close_btn_y = self.top_margin
    self.close_btn.move(close_btn_x, close_btn_y)

    max_btn_x = close_btn_x - self.btn_w
    max_btn_y = self.top_margin
    self.max_btn.move(max_btn_x, max_btn_y)

    mini_btn_x = max_btn_x - self.btn_w
    mini_btn_y = self.top_margin
    self.mini_btn.move(mini_btn_x, mini_btn_y)

def mousePressEvent(self, evt):
    # 设置标记，用于判定是否需要移动
    # 并判定是否点击了鼠标左键
```

```
        # 注意：这个判等不能用 is
        if evt.button() == Qt.LeftButton:
            self.move_flag = True
        # 窗口的原始坐标点（左上角）
        # 注意：之前这里用了 self.x 和 self.y
        # 程序报错！我怀疑是 self.x 和 self.y 已经被 PyQt 占用
        self.origin_x = self.x()
        self.origin_y = self.y()
        # 鼠标按下的点
        self.mouse_x = evt.globalX()
        self.mouse_y = evt.globalY()

    def mouseMoveEvent(self, evt):
        # 如果窗口移动标记 = True
        if self.move_flag:
            # 根据鼠标按下的点，计算移动向量
            move_x = evt.globalX() - self.mouse_x
            move_y = evt.globalY() - self.mouse_y
            # 根据移动向量和窗口原始坐标，计算最新坐标
            new_x = self.origin_x + move_x
            new_y = self.origin_y + move_y
            # 移动整个窗口的位置
            self.move(new_x, new_y)

    def mouseReleaseEvent(self, evt):
        # 重置窗口移动标记 = False
        self.move_flag = False


# 1. 创建一个应用程序对象
app = QApplication(sys.argv)
window = Window()

# 2 展示控件
window.show()

# 3. 应用程序的执行，进入到消息循环
sys.exit(app.exec_())
```

066. Python-GUI编程-PyQt5-QWidget-控件交互-是否可用

交互状态




```python
# 0. 导入需要的包和模块
import sys
from PyQt5.Qt import *

# 1. 创建一个应用程序对象
app = QApplication(sys.argv)

# 2. 控件的操作
# 2.1 创建控件
window = QWidget()
# 2.2 设置控件
window.setWindowTitle('交互状态')
window.resize(500, 500)

btn = QPushButton(window)
btn.setText('按钮')
btn.pressed.connect(lambda: print('被点击了！'))
btn.setEnabled(False)
print('按钮是否可用：', btn.isEnabled())
```

```
# 2.3 展示控件
window.show()

# 3. 应用程序的执行，进入到消息循环
sys.exit(app.exec_())
```

067. Python-GUI编程-PyQt5-QWidget-控件交互-可见隐藏

先绘制父控件，再绘制子控件

隐藏一个控件，实际上是整体重新绘制了一遍

显示和隐藏的本质：**控制它是否被绘制出来**，而不是对象是否存在（`deleteLater`）

```
# 0. 导入需要的包和模块
import sys
from PyQt5.Qt import *


class Window(QWidget):
    def paintEvent(self, evt):
        print('窗口被绘制了')
        return super().paintEvent(evt)


class Button(QPushButton):
    def paintEvent(self, evt):
        print('按钮被绘制了')
        return super().paintEvent(evt)

# 1. 创建一个应用程序对象
app = QApplication(sys.argv)

# 2. 控件的操作
# 2.1 创建控件
window = Window()
# 2.2 设置控件
window.setWindowTitle('绘制事件监听')
window.resize(500, 500)

button = Button(window)
button.setText('按钮')
# button.pressed.connect(lambda: print('点击了按钮！'))
# button.pressed.connect(lambda: button.hide())
```

```
button.pressed.connect(lambda: button.setVisible(False))
# 隐藏按钮
# button.setVisible(False)

# 2.3 展示控件
# 不写这一句，就不会调用 paintEvent
# window.show()
# 最根本的方法就是 setVisible，其他都是马甲
# window.setVisible(True)
# 不隐藏就是显示
window.setHidden(False)

# 3. 应用程序的执行，进入到消息循环
sys.exit(app.exec_())
```

---

068. Python-GUI编程-PyQt5-QWidget-控件交互-可见隐藏-注意事项

有时候 `setVisible(True)` 也不一定能看到，比如此时父控件没有显示

---

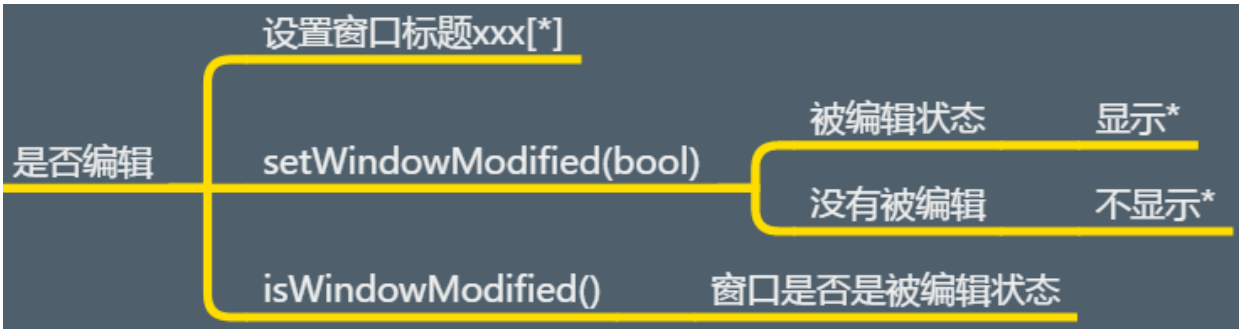
069. Python-GUI编程-PyQt5-QWidget-控件交互-可见隐藏-获取

`visible` 控件的最终状态（即使被遮挡也是可见）
`hide` 相对于父控件是否可见

```
# 0. 导入需要的包和模块
import sys
from PyQt5.Qt import *


class Window(QWidget):
    def paintEvent(self, evt):
        print('窗口被绘制了')
        return super().paintEvent(evt)


class Button(QPushButton):
    def paintEvent(self, evt):
        print('按钮被绘制了')
        return super().paintEvent(evt)
```

```python
# 1. 创建一个应用程序对象
app = QApplication(sys.argv)

# 2. 控件的操作
# 2.1 创建控件
window = Window()
# 2.2 设置控件
window.setWindowTitle('绘制事件监听')
window.resize(500, 500)

button = Button(window)
button.setText('按钮')
button.pressed.connect(lambda: button.setVisible(False))

# 2.3 展示控件
window.show()  # 注释掉这一句：按钮最终不可见，但是按钮相对于父控件没有被隐藏
print('按钮是否隐藏：', button.isHidden())
print('按钮是否可见：', button.isVisible())

# 父控件 **如果** 显示的时候，子控件是否跟着一起显示
print('按钮相对于窗口是否可见：', button.isVisibleTo(window))

# 3. 应用程序的执行，进入到消息循环
sys.exit(app.exec_())
```

070. Python-GUI编程-PyQt5-QWidget-控件交互-被编辑状态




```python
# 0. 导入需要的包和模块
import sys
from PyQt5.Qt import *

# 1. 创建一个应用程序对象
app = QApplication(sys.argv)
```

```python
# 2. 控件的操作
# 2.1 创建控件
window = QWidget()
# 2.2 设置控件
# window.setWindowTitle('是否编辑[*]')
# window.setWindowTitle('[*]是否编辑')
window.setWindowTitle('是否[*]编辑')

# 注意：换成 [&] 就不行了，只能 [*]
# window.setWindowTitle('是否[&]编辑')

window.resize(500, 500)
# 显示为已编辑
window.setWindowModified(True)
print('是否已编辑：', window.isWindowModified())

# 应用场景：关闭之前检测 isWindowModified()
# 如果已编辑，就提示保存

# 2.3 展示控件
window.show()

# 3. 应用程序的执行，进入到消息循环
sys.exit(app.exec_())
```

071. Python-GUI编程-PyQt5-QWidget-控件交互-活跃窗口

判定一个窗口是否活跃，从视觉上看**是否有光环**

```python
# 0. 导入需要的包和模块
import sys
from PyQt5.Qt import *

# 1. 创建一个应用程序对象
app = QApplication(sys.argv)

# 2. 控件的操作
# 2.1 创建控件
window = QWidget()
# 2.2 设置控件
window.setWindowTitle('窗口1')
window.resize(500, 500)
```

```
w2 = QWidget()
w2.setWindowTitle('窗口2')

# 2.3 展示控件
window.show()
w2.show()

print('窗口1是否活跃：', window.isActiveWindow())
# 后展示的是活跃窗口！
print('窗口2是否活跃：', w2.isActiveWindow())
# 窗口是否活跃跟z轴层级没有关系
window.raise_()
print('窗口1是否活跃：', window.isActiveWindow())


# 3. 应用程序的执行，进入到消息循环
sys.exit(app.exec_())
```

072. Python-GUI编程-PyQt5-QWidget-控件交互-关闭控件

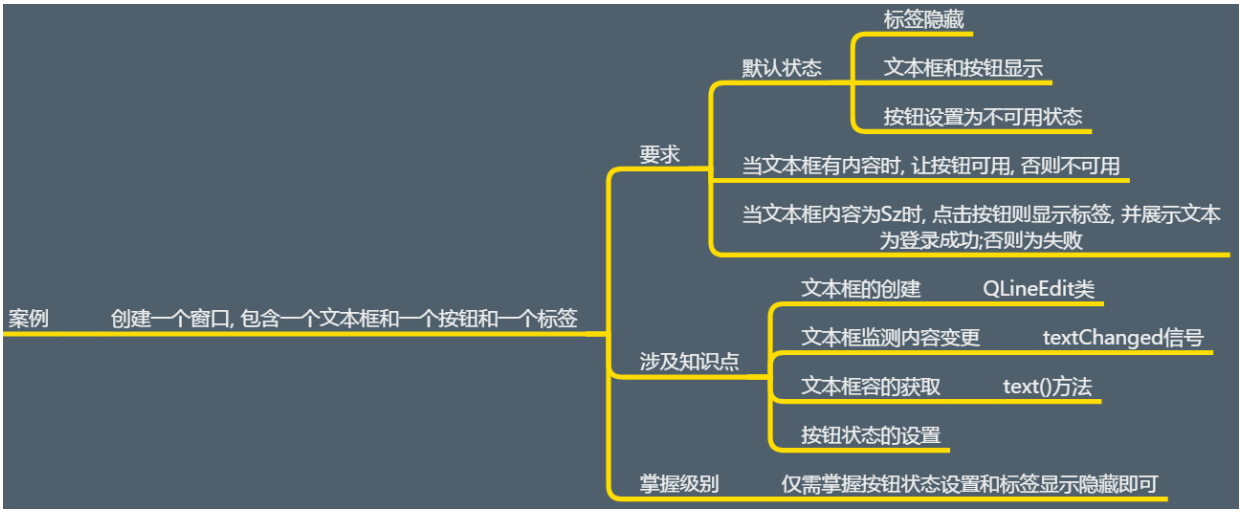
控件的关闭，一般会调用 `setVisible(False)` ，有时候还会把控件删除掉。

```
# 0. 导入需要的包和模块
import sys
from PyQt5.Qt import *


class Btn(QPushButton):
    def paintEvent(self, evt):
        print('按钮被绘制了！')
        return super().paintEvent(evt)


# 1. 创建一个应用程序对象
app = QApplication(sys.argv)

# 2. 控件的操作
# 2.1 创建控件
window = QWidget()
# 2.2 设置控件
window.setWindowTitle('关闭控件')
window.resize(500, 500)

btn = Btn(window)
```

```
    btn.setText('按钮')
    btn.destroyed.connect(lambda: print('按钮被释放了！'))

    # 隐藏按钮的三种方式
    # btn.setVisible(False)
    # btn.setHidden(True)
    # btn.hide()
    # 关闭控件 - 不会触发 destroyed 信号
    # 所以一般情况下，close() 相当于隐藏了控件
    # 但是如果设置了 setAttribute(Qt.WA_DeleteOnClose, True)
    # 则在关闭之后就会删除控件
    btn.setAttribute(Qt.WA_DeleteOnClose, True)
    btn.close()
    # 下次循环删除按钮 - 会触发 destroyed 信号
    # btn.deleteLater()

    # 2.3 展示控件
    window.show()

    # 3. 应用程序的执行，进入到消息循环
    sys.exit(app.exec_())
```

---

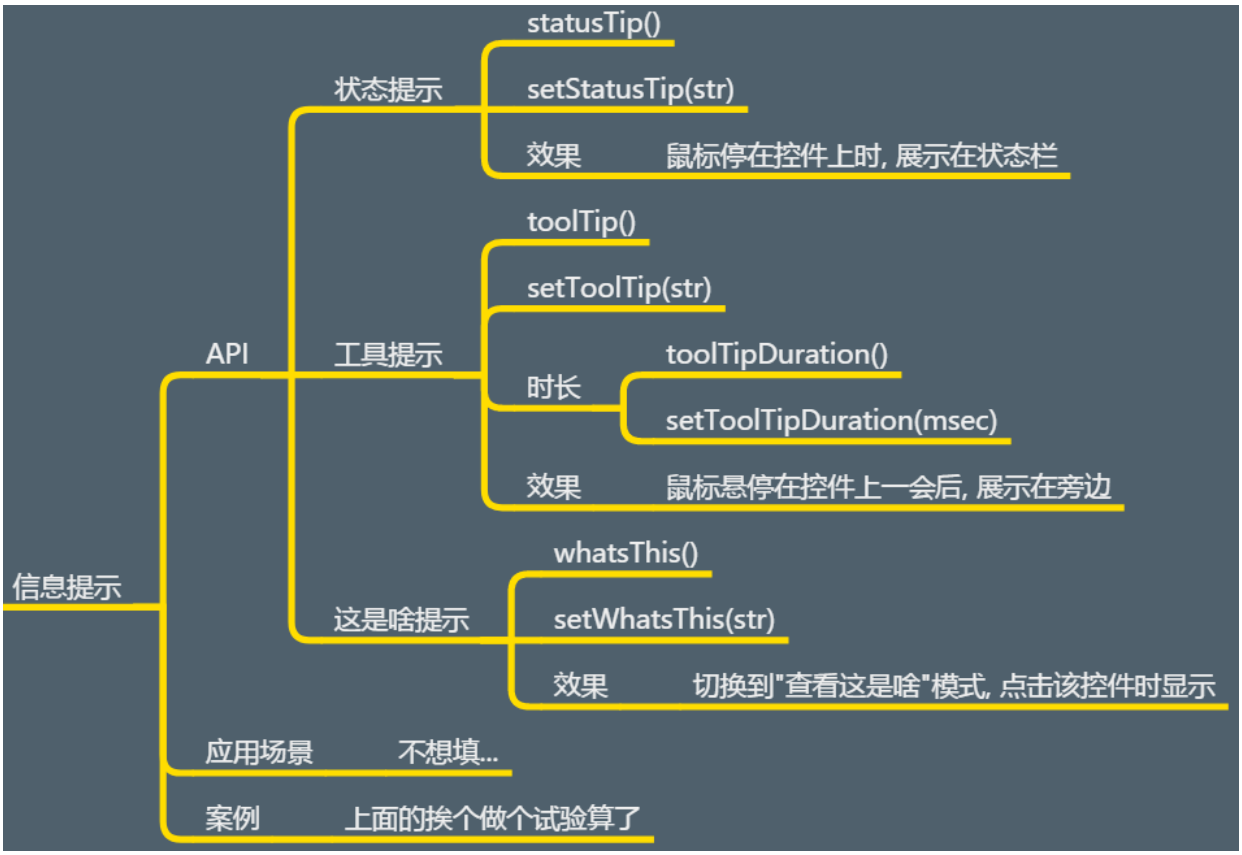
073. Python-GUI编程-PyQt5-QWidget-控件交互-综合案例




```
    from PyQt5.Qt import *


    class Window(QWidget):
        def __init__(self):
            super().__init__()
```

```
        self.setWindowTitle('交互状态案例')
        self.resize(500, 500)
        self.setup_ui()

    def setup_ui(self):
        # 添加三个子控件
        label = QLabel(self)
        label.setText('标签')
        label.move(100, 50)
        label.hide()

        line_edit = QLineEdit(self)
        # line_edit.setText('文本框')
        line_edit.move(100, 100)

        button = QPushButton(self)
        button.setText('登录')
        button.move(100, 150)
        button.setEnabled(False)

        def slot(text):
            print('文本内容发生改变：', text)
            # if len(text) > 0:
            #     button.setEnabled(True)
            # else:
            #     button.setEnabled(False)

            # 优化写法!
            button.setEnabled(len(text) > 0)

        # textChanged 的返回值是最新的文本内容!
        line_edit.textChanged.connect(slot)

        def check():
            print('按钮被点击了！')
            if line_edit.text() == '666':
                label.setText('登录成功！')
            else:
                label.setText('登录失败！')
            label.show()
            # 自适应大小
            label.adjustSize()
        button.pressed.connect(check)


if __name__ == '__main__':
    import sys
    app = QApplication(sys.argv)

    window = Window()
    window.show()
    sys.exit(app.exec_())
```

074. Python-GUI编程-PyQt5-QWidget-控件交互-信息提示

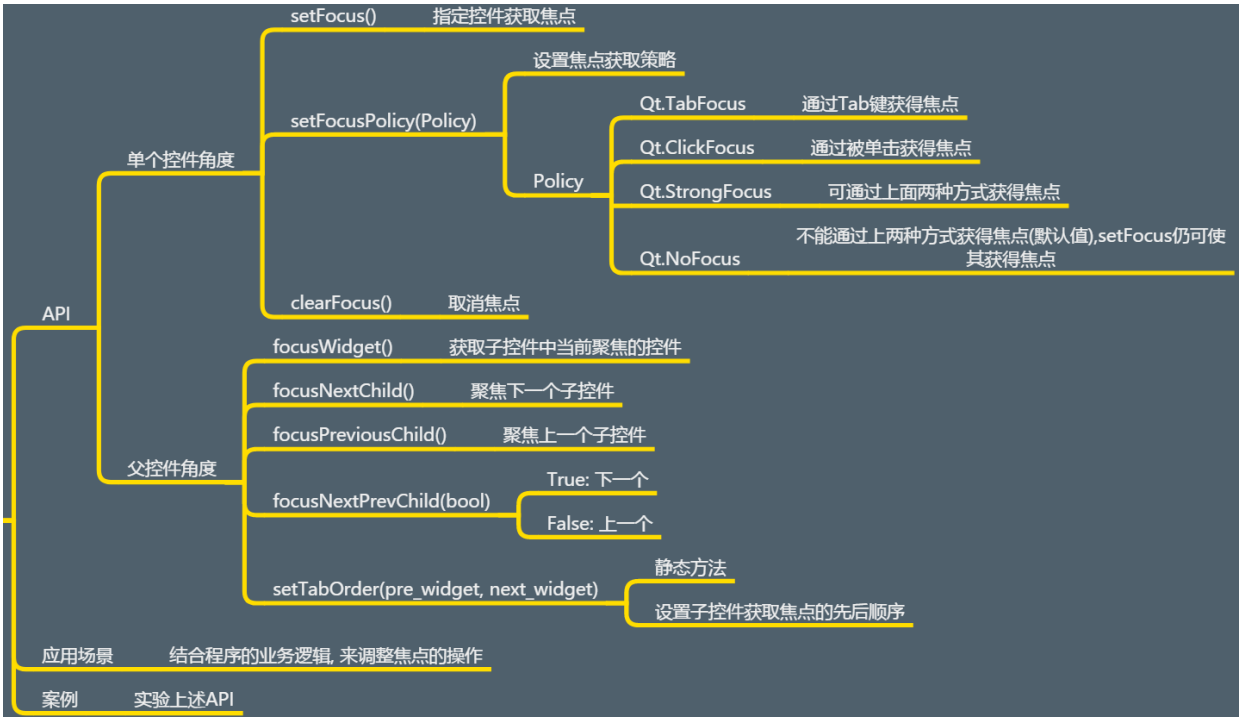



```python
# 0. 导入需要的包和模块
import sys
from PyQt5.Qt import *

# 1. 创建一个应用程序对象
app = QApplication(sys.argv)

# 2. 控件的操作
# 2.1 创建控件
window = QMainWindow()  # 组合控件: 这个控件由很多控件组合起来的
# 里面很多子控件是懒加载的 -- 用到的时候才会创建
window.statusBar()  # 加载状态栏
# 2.2 设置控件
window.setWindowTitle('信息提示案例')
window.resize(500, 500)
# Qt.WindowContextHelpButtonHint 窗口上下文帮助按钮
window.setWindowFlags(Qt.WindowContextHelpButtonHint)

# 当鼠标停留在窗口控件身上之后，在状态栏提示的一段文本
```

```
# 注意：必须先有状态栏！
window.setStatusTip('这是状态提示！')
# 获取状态提示信息
print(window.statusTip())

label = QLabel(window)
label.setText('标签控件')
# 设置状态提示
label.setStatusTip('这是标签的状态提示！')
# 设置工具提示
label.setToolTip('这是标签的工具提示！')
# 获取工具提示
print(label.toolTip())
# 设置工具提示显示时长，单位是 ms
label.setToolTipDuration(2000)
# 获取显示时长
print(label.toolTipDuration())
# 这是啥提示（黄底文本）
label.setWhatsThis('这是啥？这是标签')
# 获取这是啥提示
print(label.whatsThis())

# 2.3 展示控件
window.show()

# 3. 应用程序的执行，进入到消息循环
sys.exit(app.exec_())
```

---

075. Python-GUI编程-PyQt5-QWidget-控件交互-焦点控制-上




```python
# 0. 导入需要的包和模块
import sys
from PyQt5.Qt import *

# 1. 创建一个应用程序对象
app = QApplication(sys.argv)

# 2. 控件的操作
# 2.1 创建控件
window = QWidget()
# 2.2 设置控件
window.setWindowTitle('焦点控制')
window.resize(500, 500)

le1 = QLineEdit(window)
le1.move(50, 50)

le2 = QLineEdit(window)
le2.move(50, 100)

le3 = QLineEdit(window)
le3.move(50, 150)

# 设置焦点
le2.setFocus()
# 清除焦点
# 注意：清除焦点后，在大控件中需要找到一个控件来设置焦点
# 所以一般默认找到第一个作为焦点
le2.clearFocus()

# 设置焦点获取策略
# Qt.TabFocus 只通过 Tab 获取焦点
```

```
le3.setFocusPolicy(Qt.TabFocus)
# Qt.ClickFocus 只通过 鼠标点击 获取焦点
le3.setFocusPolicy(Qt.ClickFocus)
# Qt.StrongFoucus 可以通过鼠标或者点击获取焦点
# 默认就是这一种
le3.setFocusPolicy(Qt.StrongFocus)
# Qt.NoFocus 禁止获取焦点，但是 setFoucs() 方法可以让其获取焦点
le3.setFocusPolicy(Qt.NoFocus)

# 2.3 展示控件
window.show()

# 3. 应用程序的执行，进入到消息循环
sys.exit(app.exec_())
```

---

076. Python-GUI编程-PyQt5-QWidget-控件交互-焦点控制-下

```
# 0. 导入需要的包和模块
import sys
from PyQt5.Qt import *


class Window(QWidget):
    def mousePressEvent(self, evt):
        print(self.focusWidget())

        # 聚焦下一个子控件
        # self.focusNextChild()

        # 聚焦上一个子控件
        # self.focusPreviousChild()

        # 上面两个的函数原型
        # True -- Next -- 下一个
        # False -- Prev -- 上一个
        # self.focusNextPrevChild(True)


# 1. 创建一个应用程序对象
app = QApplication(sys.argv)

# 2. 控件的操作
# 2.1 创建控件
window = Window()
```

```
# 2.2 设置控件
window.setWindowTitle('焦点控制2')
window.resize(500, 500)

le1 = QLineEdit(window)
le1.move(50, 50)

le2 = QLineEdit(window)
le2.move(50, 100)

le3 = QLineEdit(window)
le3.move(50, 150)

# 静态方法
QWidget.setTabOrder(le1, le3)
QWidget.setTabOrder(le3, le2)

# 2.3 展示控件
window.show()

# 获取当前窗口内部，所有子控件当中获取焦点的那个控件
print(window.focusWidget())  # 结果是 None
# 因为在这一行，界面已经展示出来
# 但是焦点还没有被设置，是后来设置的
# 在这一行，所有子控件都没有获取焦点
# 除非实现用 setFocus() 设置

# 3. 应用程序的执行，进入到消息循环
sys.exit(app.exec_())
```

---

077. Python-GUI编程-PyQt5-QWidget-总结

完成于 `201810191244`