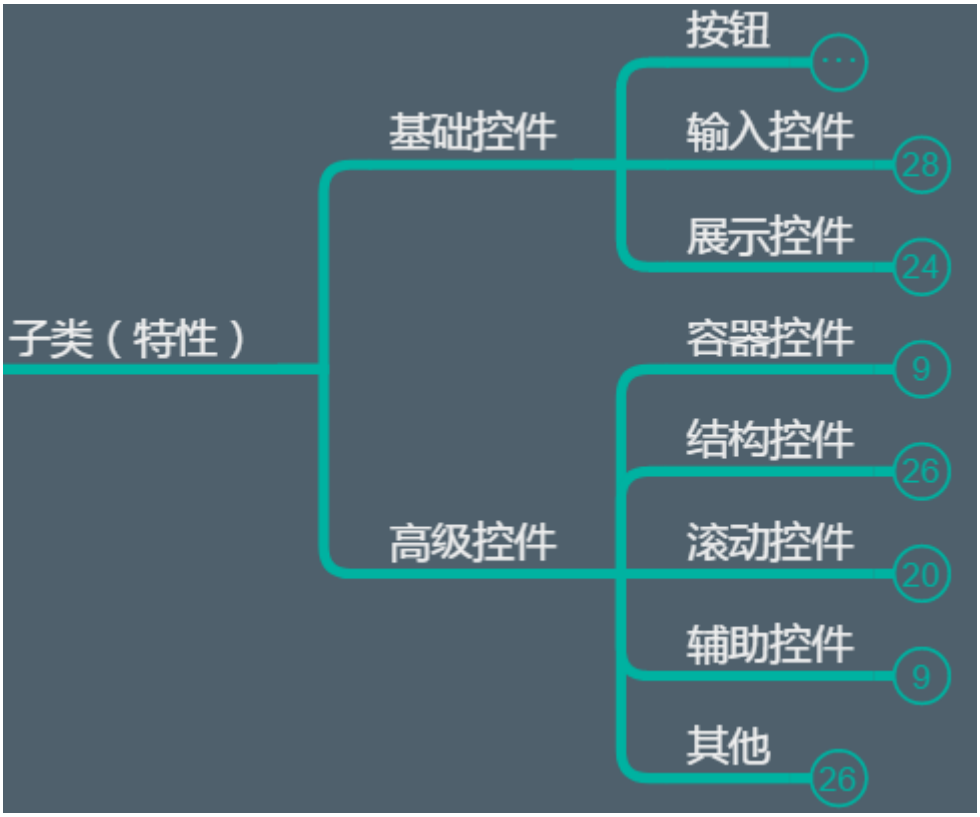
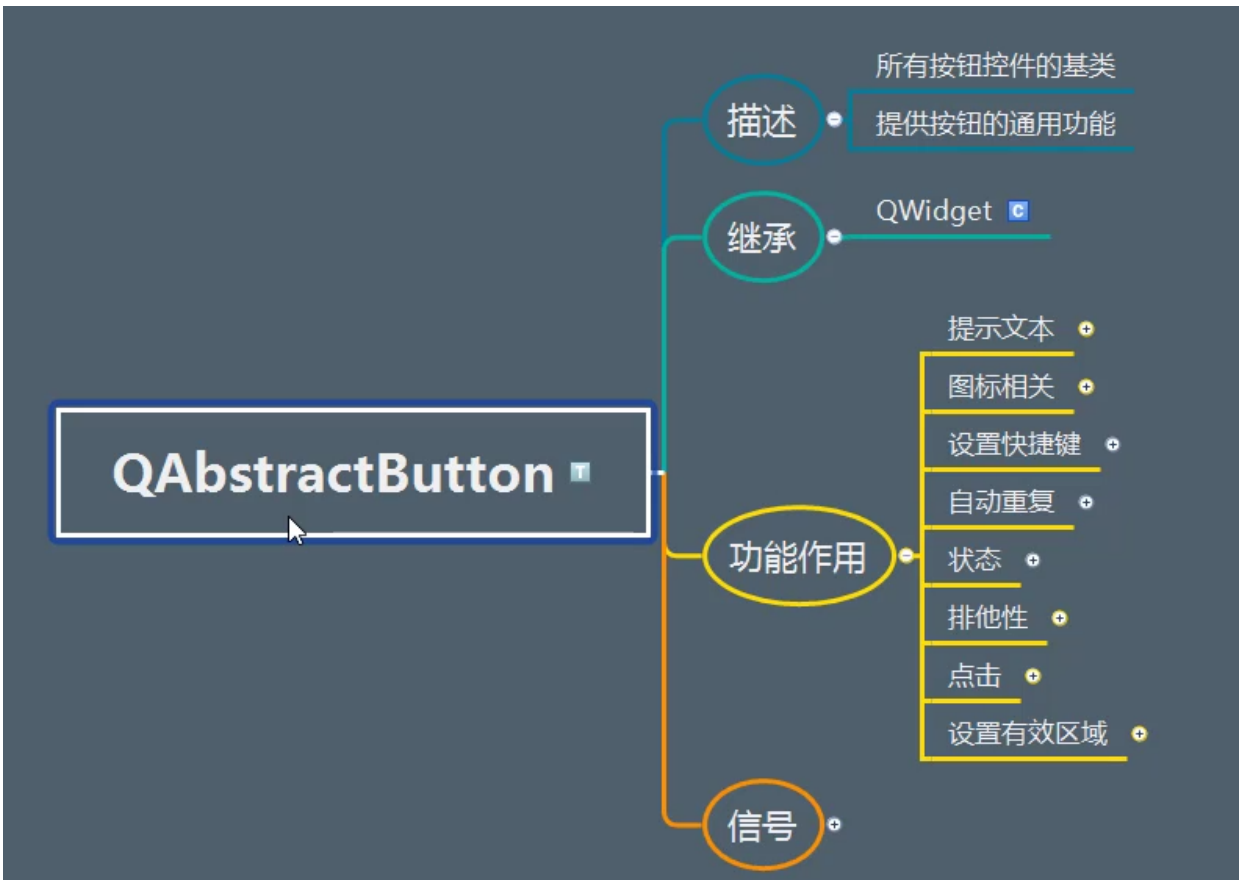
[笔记][LIKE-Python-GUI编程-PyQt5][09]

PyQt5

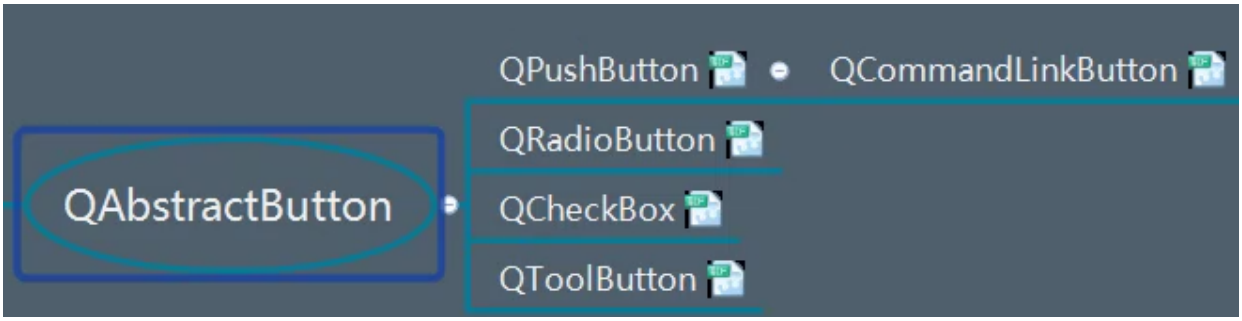
---

078. Python-GUI编程-PyQt5-QAbstractButton-简介





抽象类：把一些公共的特性放到虚拟的类里面，但是这个类本身不能被直接使用。需要其他类继承它，来实现它。

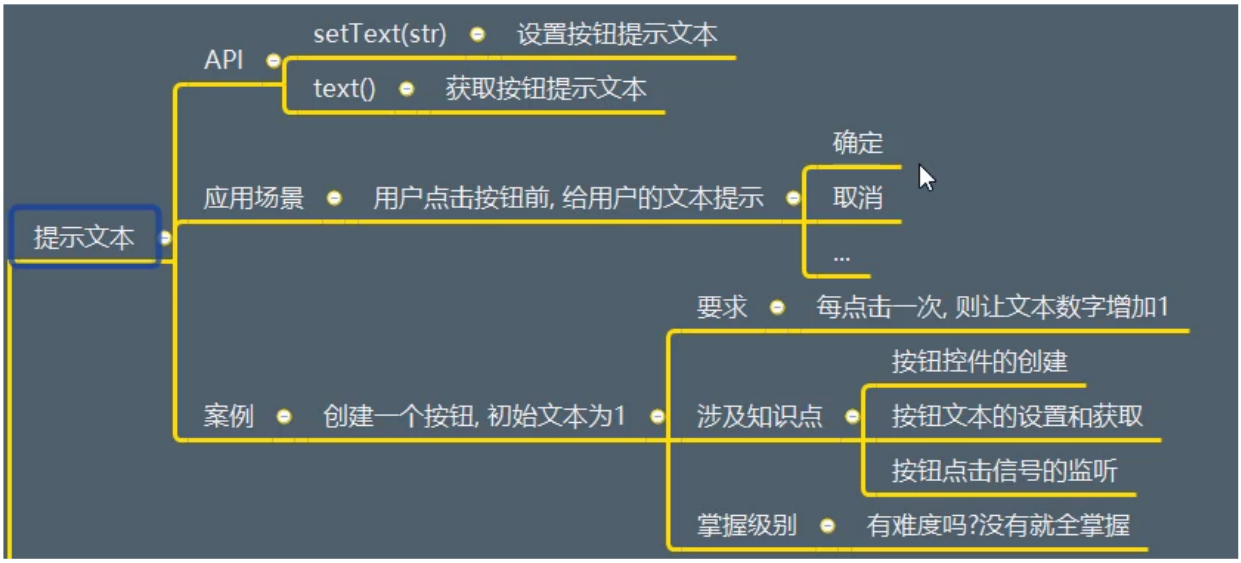


079. Python-GUI编程-PyQt5-QAbstractButton-子类化抽象类

```
# 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('QAbstractButton 简介')
window.resize(500, 500)

# 抽象类无法直接被实例化!
# 会报错:
# TypeError: PyQt5.QtWidgets.QAbstractButton represents a C++ abstract cl
ass and cannot be instantiated
# 需要进行子类化!
# btn = QAbstractButton(window)


# 子类化抽象类
class Button(QAbstractButton):
    # pass
    # 这样会报错:
    # NotImplementedError: QAbstractButton.paintEvent() is abstract and m
ust be overridden
    # 必须要实现抽象类里面所有的抽象方法
    # 其中很重要的就是 paintEvent() 这个抽象方法

    def paintEvent(self, evt):
        # print('绘制按钮')
        # 绘制按钮上要展示的一个界面内容
```

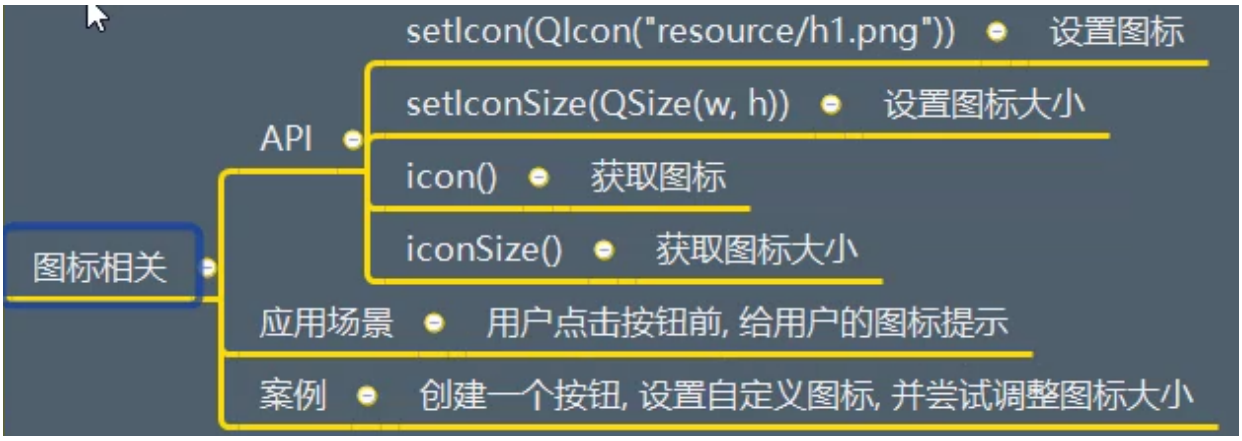
```
            # 画家、笔、纸
            # 要传入一个 QPaintDevice 给 QPainter
            # QPaintDevice 相当于那张纸，在这里就是按钮
            # 注意：所有控件都可以被当成纸
            # 因为所有可视控件都继承自 QWidget
            # 而 QWidget 是多继承的，先继承自 QObject，然后继承自 QPaintDevice
            painter = QPainter(self)

            # 创建笔
            pen = QPen(QColor(111, 200, 20), 5)  # 颜色，粗细
            # 设置笔
            painter.setPen(pen)

            # 画家画画
            # 画笔宽度不适用于文本
            # painter.drawText(20, 20, '画家画画')
            # 获取设置的文本
            painter.drawText(25, 40, self.text())
            painter.drawEllipse(0, 0, 100, 100)


btn = Button(window)
btn.setText('文本内容')
btn.resize(100, 100)

btn.pressed.connect(lambda: print('点击了按钮'))

# 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_())
```

080. Python-GUI编程-PyQt5-QAbstractButton-文本设置




```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('文本设置')
window.resize(500, 500)

btn = QPushButton(window)
# 不设置文本的话是一个空白的小方块
btn.setText('1')


def plus_one():
    print('加一')
    num = int(btn.text()) + 1
    btn.setText(str(num))


btn.pressed.connect(plus_one)

# 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_())
```

081. Python-GUI编程-PyQt5-QAbstractButton-图标设置




```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('图标设置')
window.resize(500, 500)

btn = QPushButton(window)
# 假如有文本，默认在文本左侧显示图标
icon = QIcon('img/Python.png')
btn.setIcon(icon)
size = QSize(100, 100)
btn.setIconSize(size)
# 获取图标对象
print(btn.icon())
# 获取图标大小
print(btn.iconSize())

# 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_())
```

082. Python-GUI编程-PyQt5-QAbstractButton-快捷键设置

触发按钮按下信号的几个方式：

- 鼠标点击
- 获取焦点时，空格键
- 使用快捷键触发

```
# 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('快捷键设置')
window.resize(500, 500)

btn = QPushButton(window)

# 设置快捷键
# 方式1: 有提示文本
# btn.setText('&button')  # Alt - b 触发
# 带中文可以这么写
# btn.setText('按钮(&b)')

# 方式2: setShortcut 直接设置快捷方式
# 适用于整个按钮只展示图标, 没有任何文本
btn.setShortcut('Alt+b')


def slot():
    print('按钮被点击了! ')


btn.clicked.connect(slot)

# 2.3 展示控件
window.show()
```

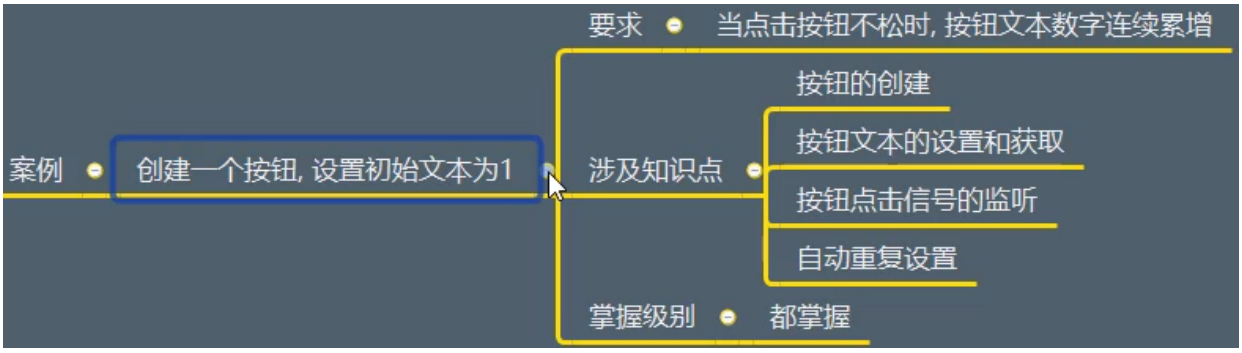
```
# 3. 应用程序的执行，进入到消息循环
sys.exit(app.exec_())
```

083. Python-GUI编程-PyQt5-QAbstractButton-自动重复

按钮假如被点击了而且没有松开，就会自动重复，不断触发被点击的信号。

```
# 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('自动重复')
window.resize(500, 500)

btn = QPushButton(window)
icon = QIcon('img/Python.png')
btn.setIcon(icon)
size = QSize(100, 100)
btn.setIconSize(size)
btn.clicked.connect(lambda: print('按钮被点击了！'))

# 获取自动重复
```

```
print('当前是否自动重复：', btn.autoRepeat())
# 设置自动重复
btn.setAutoRepeat(True)
# 获取重复延迟
print('当前重复延迟：', btn.autoRepeatDelay())  # 300 毫秒延迟
# 设置重复延迟为 2000 ms
btn.setAutoRepeatDelay(2000)
# 获取重复间隔
print('当前重复间隔：', btn.autoRepeatInterval())  # 100 毫秒间隔
# 设置重复间隔为 1000 ms
btn.setAutoRepeatInterval(1000)

# 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_())
```




```
# 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('自动重复')
window.resize(500, 500)


def plus_one():
    print('加一')
    num = int(btn.text()) + 1
    btn.setText(str(num))


btn = QPushButton(window)
```

```
    btn.setText('1')
    icon = QIcon('img/Python.png')
    btn.setIcon(icon)
    size = QSize(100, 100)
    btn.setIconSize(size)
    # btn.clicked.connect(lambda: print('按钮被点击了！'))
    btn.clicked.connect(plus_one)

    # 获取自动重复
    print('当前是否自动重复：', btn.autoRepeat())
    # 设置自动重复
    btn.setAutoRepeat(True)
    # 获取重复延迟
    print('当前重复延迟：', btn.autoRepeatDelay())  # 300 毫秒延迟
    # 设置重复延迟为 2000 ms
    btn.setAutoRepeatDelay(2000)
    # 获取重复间隔
    print('当前重复间隔：', btn.autoRepeatInterval())  # 100 毫秒间隔
    # 设置重复间隔为 1000 ms
    btn.setAutoRepeatInterval(1000)

    # 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_())
```

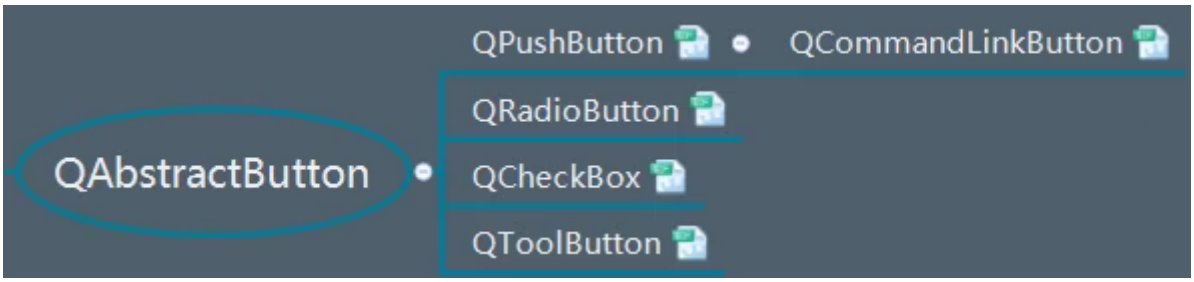
084. Python-GUI编程-PyQt5-QAbstractButton-状态设置

按钮的能用状态经典应用场景

- 有一个账号文本框和密码文本框
- 假如按钮不可用
- 用户会向上检查是否少输入了什么东西

除了能用状态（继承于 `QWidget`），还有**是否按下**和**是否选中**

按下状态：用户点击了按钮，但还没有松手的状态



> 【?】通过 `qss` 设置按下状态的样式，为什么下面这一句多了一个空格的 `qss` 字符串不能成功设置？
> `push_button.setStyleSheet('QPushButton: pressed {background-color: red;}')`
> 正确的字符串是这样的：`push_button.setStyleSheet('QPushButton:pressed {background-color: red;}')`

按下状态

```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('状态设置')
window.resize(500, 500)

push_button = QPushButton(window)
push_button.setText('QPushButton')
push_button.move(100, 100)

radio_button = QRadioButton(window)
radio_button.setText('QRadioButton')
radio_button.move(100, 150)

check_box = QCheckBox(window)
check_box.setText('QCheckBox')
check_box.move(100, 200)
```

```
# 把三个按钮，都设置为按下状态
push_button.setDown(True)
radio_button.setDown(True)
check_box.setDown(True)

# 通过 qss 设定按下状态的样式
# 注意：我之前写的这一句无法设置背景为红色，多了一个空格
# push_button.setStyleSheet('QPushButton: pressed {background-color: re
d;}')
push_button.setStyleSheet('QPushButton:pressed {background-color: red;}')

# 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_())
```

选中状态

```
# 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('状态设置')
window.resize(500, 500)

btn = QPushButton(window)
btn.setIcon(QIcon('img/Python.png'))
btn.setIconSize(QSize(80, 80))

push_button = QPushButton(window)
push_button.setText('QPushButton')
push_button.move(100, 100)

radio_button = QRadioButton(window)
radio_button.setText('QRadioButton')
radio_button.move(100, 150)

check_box = QCheckBox(window)
check_box.setText('QCheckBox')
check_box.move(100, 200)
```

```
# 查看是否可以被选中
print('QPushButton是否可以被选中：', push_button.isCheckable())
print('QRadioButton是否可以被选中：', radio_button.isCheckable())
print('QCheckBox是否可以被选中：', check_box.isCheckable())

# 设置 QPushButton 可以被选中
push_button.setCheckable(True)
print('-' * 50)
print('再次查看QPushButton是否可以被选中：', push_button.isCheckable())

# 设置按钮为选中状态
push_button.setChecked(True)
radio_button.setChecked(True)
check_box.setChecked(True)

# 查看三个按钮的选中状态
print('-' * 50)
print('QPushButton当前选中状态：', push_button.isChecked())
print('QRadioButton当前选中状态：', radio_button.isChecked())
print('QCheckBox当前选中状态：', check_box.isChecked())


# 使用 toggle() 切换选中状态
def slot():
    push_button.toggle()
    radio_button.toggle()
    check_box.toggle()

    # 另外一种写法:
    # push_button.setChecked(not push_button.isChecked())
    # radio_button.setChecked(not radio_button.isChecked())
    # check_box.setChecked(not check_box.isChecked())


btn.pressed.connect(slot)

# 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_())
```

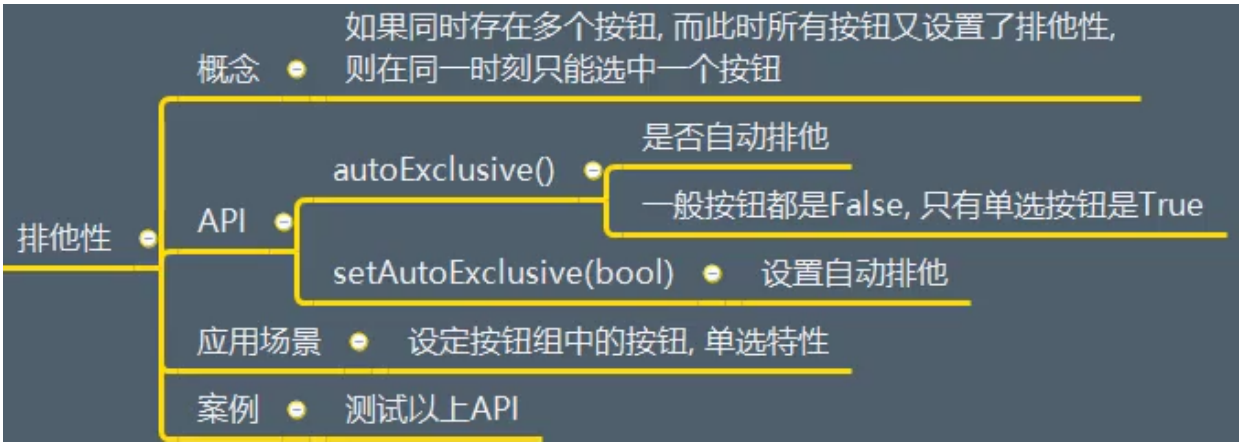
能用状态

```
# 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)

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

btn = QPushButton(window)
btn.setIcon(QIcon('img/Python.png'))
btn.setIconSize(QSize(80, 80))

push_button = QPushButton(window)
push_button.setText('QPushButton')
push_button.move(100, 100)

radio_button = QRadioButton(window)
radio_button.setText('QRadioButton')
radio_button.move(100, 150)

check_box = QCheckBox(window)
check_box.setText('QCheckBox')
check_box.move(100, 200)

# 注意 setEnabled 是从 QWidget 继承而来的方法
# 而不是 QAbstractButton 的方法
# 虽然控件不可用，但还是可以通过代码来更改它们的状态
# 注意：QPushButton可能看不出来状态的变化
# 是因为它的不可用状态有一个特定样式，造成我们看不出由于状态更改而带来的样式区别
push_button.setEnabled(False)
radio_button.setEnabled(False)
check_box.setEnabled(False)


# 使用 toggle() 切换选中状态
def slot():
    push_button.toggle()
    radio_button.toggle()
    check_box.toggle()

    # 另外一种写法:
    # push_button.setChecked(not push_button.isChecked())
    # radio_button.setChecked(not radio_button.isChecked())
    # check_box.setChecked(not check_box.isChecked())


btn.pressed.connect(slot)

# 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_())
```

085. Python-GUI编程-PyQt5-QAbstractButton-排他性




```
# 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('排他性')
window.resize(500, 500)

for i in range(3):
    btn = QPushButton(window)
    btn_name = '按钮' + str(i + 1)
    btn.setText(btn_name)
    btn.move(50 * i, 50 * i)

    # 使用 autoExclusive() 获取按钮排他性状态
    print(btn_name, '是否排他：', btn.autoExclusive())
    # 默认情况下 QPushButton 是不可选中的
    print(f'默认情况 {btn_name} 是否可选中：', btn.isCheckable())
    print('-' * 50)
```

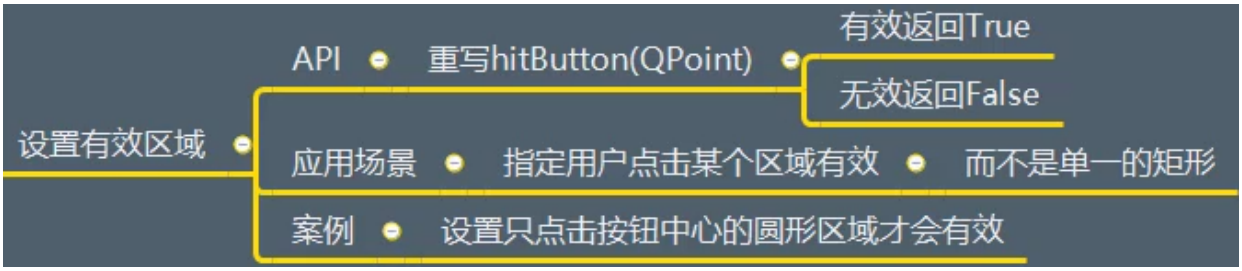
```
        # 设置 QPushButton 为可选中
        btn.setCheckable(True)

        # 设置排他性
        btn.setAutoExclusive(True)

# 单独加一个按钮，默认没有排他性
# 同一级别当中（拥有相同的父控件）：
# 只要设置了排他性为 True，就互相排斥
btn = QPushButton(window)
btn.setCheckable(True)
btn.setText('按钮4')
btn.move(250, 250)

# QRadioButton
# 默认是可以被选中的 checkable
# 默认也是排他的 auto exclusive

# QCheckBox
# 默认是可以被选中的
# 默认不排他

# 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_())
```

086. Python-GUI编程-PyQt5-QAbstractButton-模拟点击


```
# 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)

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

btn = QPushButton(window)
btn.setText('按钮1')
btn.move(200, 200)
btn.pressed.connect(lambda: print('按钮1被点击了！'))

# 模拟点击
# btn.click()

# 模拟动画点击 animateClick(ms)
# btn.animateClick(2000)

btn2 = QPushButton(window)
btn2.setText('按钮2')


def test():
    # 模拟点击
    # btn.click()
    # 动画点击
    btn.animateClick(1000)


btn2.pressed.connect(test)

# 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_())
```

087. Python-GUI编程-PyQt5-QAbstractButton-设置点击有效区域



示例程序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('设置有效区域')
window.resize(500, 500)


class Button(QPushButton):
    # 需求：点击按钮右半部分有效，左半部分无效
    # 注意：点的坐标参照按钮的坐标，是相对坐标
    # 右侧区域：x 大于整个按钮宽度的一半
    def hitButton(self, point):
        print(point)
        # point 的类型是 PyQt5.QtCore.QPoint
        # 可以 Ctrl - 单击 查看文档
        if point.x() > self.width() / 2:
            return True

        return False


button = Button(window)
button.setText('点击')
button.move(200, 200)
button.pressed.connect(lambda: print('按钮被点击了！'))

# 按钮被点击之后
# 会把点坐标传递给 hitButton 方法
# 看看 hitButton 方法的返回值是什么
# 如果是 True，说明点击是有效的，会发射信号
# 如果是 False，说明点击是无效的，就不会触发相关信号的发射

# 2.3 展示控件
window.show()
```

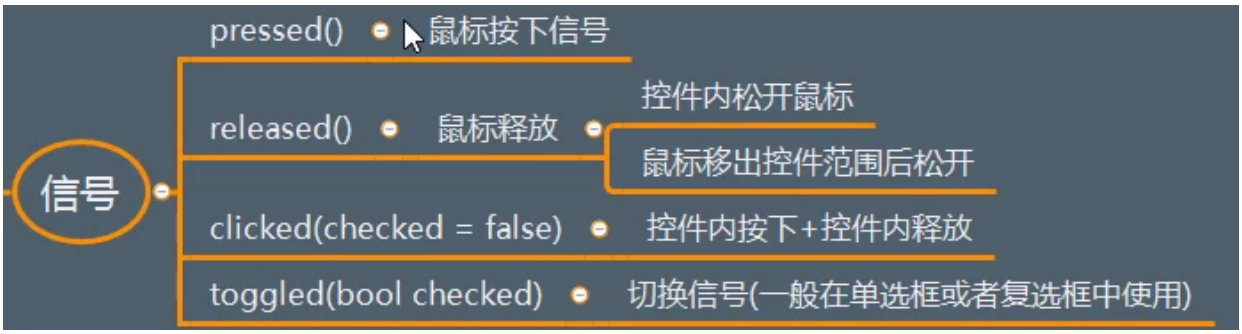
```
# 3. 应用程序的执行，进入到消息循环
sys.exit(app.exec_())
```

示例程序2

```
# 0. 导入需要的包和模块
import sys
from math import sqrt
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('设置有效区域-圆形')
window.resize(500, 500)


class Button(QPushButton):
    def hitButton(self, point):
        # 通过给定的一个点的坐标，计算与圆心的距离
        center_x = self.width() / 2
        center_y = self.height() / 2

        hit_x = point.x()
        hit_y = point.y()

        # 平方也可以用 math.pow(x, 2)
        dist = sqrt((center_x - hit_x) ** 2 + (center_y - hit_y) ** 2)
        # print(dist)

        # 注意：必须要写 return True 或者 return False
        # 否则程序会崩溃

        if dist < self.width() / 2:
            return True
        return False

    # 在整个按钮内部画一个内切圆
    def paintEvent(self, evt):
        # 通过调用父类方法，保留之前的绘制（即按钮上的文本）
        super().paintEvent(evt)
        # 创建画家，传入画布（就是按钮）
```

```
        painter = QPainter(self)
        # 给画家一根笔
        pen = QPen(QColor(100, 150, 200), 2)
        painter.setPen(pen)
        # 画画
        painter.drawEllipse(self.rect())


button = Button(window)
button.setText('按钮')
button.resize(200, 200)
button.move(100, 100)
button.clicked.connect(lambda: print('按钮被点击了！'))


# 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_())
```

088. Python-GUI编程-PyQt5-QAbstractButton-可用信号




```
# 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('信号')
```

```
window.resize(500, 500)

btn = QPushButton(window)
btn.move(100, 100)
btn.setText('按钮')

btn.pressed.connect(lambda: print('按钮被按下了'))

# released 信号何时发射：
# 1. 控件内松开鼠标
# 2. 鼠标移出控件范围后松开
btn.released.connect(lambda: print('按钮被释放了'))

# clicked 信号何时发射：
# 鼠标在按钮有效区域按下并松开，才会发射
# 如果移出来了之后再松开，是不会发射这个信号的
# btn.clicked.connect(lambda: print('按钮被点击了'))
# 接收传过来的值，并打印
btn.clicked.connect(lambda value: print('按钮被点击了', value))
# 该值为 False，代表该按钮点击之后，是否处于被选中的状态
# 现在设置按钮为可以被选中
btn.setCheckable(True)

# toggled：选中状态发生改变时发射的信号
# 得到的值是当前按钮的选中状态
btn.toggled.connect(lambda value: print('按钮选中状态发生了改变', value))
# 假如按钮不可选中，那么该信号不会被触发
# btn.setCheckable(False)

# 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_())
```

完成于 `201810231536`