第 5 章

布局管理与容器

在前面的章节中，已经介绍了多种控件。如何在一个窗口中布局多个控件？如何整齐地排列多个控件？更改窗口大小后，如何自动地调整控件的位置，以及宽和高？这些问题都是开发者要考虑的问题。

针对这些问题，PySide6 提供了布局管理类和容器类。布局管理类可以自动定位控件并调整控件的宽和高。容器类不仅可以装载更多的控件，而且能美观地显示控件。

5.1 布局管理

在 PySide6 中，有一组布局管理类。布局管理类可以设置控件在窗口中的布局方式。当控件的可用空间发生变化时，这些布局类会自动定位并调整控件的宽和高，确保它们排列一致，让窗口界面作为一个整体可用。

5.1.1 布局管理的基础知识

在 PySide6 中，所有的控件类都是 QWidget 类的子类，其他窗口类（QMainWindow、QDialog）也是 QWidget 类的子类。如果要在窗口中使用布局管理类，则需要使用 QWidget.setLayout(Layout:QLayout)方法设置布局管理，Layout 表示布局管理类创建的对象。

当在窗口中设置了布局管理时，布局管理负责以下任务：子控件的定位、合理化窗口的默认尺寸、合理化窗口的最小尺寸、调整窗口的大小、窗口内容更改时自动更新。自动更新的内容包括子控件的字号及文本或其他内容、隐藏或显示小控件、删除小空间。

在 PySide6 中，常用的布局管理类见表 5-1。

表 5-1 常用的布局管理类

布局管理类	说　明
QLayoutItem	QLayout 操作的抽象项，一般不会单独使用
QLayout	所有布局管理类的基类，一般不会单独使用
QBoxLayout	垂直或水平排列控件，有两个子类：QHBoxLayout、QVBoxLayout
QHBoxLayout	水平排列控件

续表

布局管理类	说　明
QVBoxLayout	垂直排列控件
QGridLayout	在网格中排列控件
QFormLayout	在表单中排列控件,即以 2 列多行的形式排列控件,其中 1 列显示标签
QStackedLayout	堆叠排列控件,即一次只能看见一部分控件,其他控件被隐藏了

在 PySide6 中,这些布局管理类有继承关系,其继承关系如图 5-1 所示。

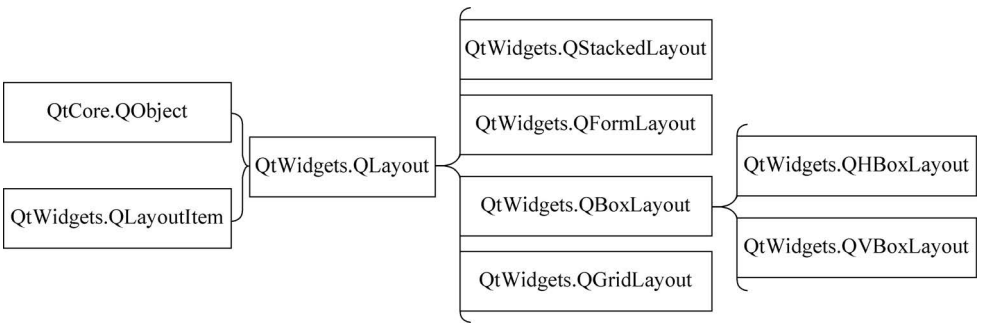

图 5-1　布局管理类的继承关系

5.1.2　水平布局与垂直布局(QHBoxLayout、QVBoxLayout)

在 PySide6 中,使用 QHBoxLayout 类表示水平布局对象,使用 QVBoxLayout 类表示垂直布局对象。这两个类的构造函数如下:

```
QHBoxLayout(parent:QWidget = None)
QVBoxLayout(parent:QWidget = None)
```

其中,parent 表示父窗口或父容器。

由于 QHBoxLayout 类和 QVBoxLayout 类都是 QBoxLayout 类的子类,所以这两个类具有相同的方法。QHBoxLayout 类和 QVBoxLayout 类的常用方法见表 5-2。

表 5-2　QHBoxLayout 类和 QVBoxLayout 类的常用方法

方法及参数类型	说　明	返回值类型
addWidget(QWidget, stretch: int = 0, Qt.Alignment)	添加控件,可设置伸缩系数和对齐方式	None
addLayout(QLayout,stretch:int=0)	添加子布局	None
addSpacing(size:int)	添加固定长度的占位空间	None
addStretch(stretch:int=0)	添加可伸缩空间	None
addStrut(int)	设置竖直方向的最小值	None
insertWidget(index:int,QWidget,stretch:int=0,Qt.Alignment)	根据索引插入控件,可设置伸缩系数和对齐方式	None

续表

方法及参数类型	说　明	返回值类型
insertLayout（index：int，QLayout，stretch:int=0）	根据索引插入子布局，可设置伸缩系数和对齐方式	None
insertSpacing(index:int,size:int)	根据索引插入固定长度的占位空间	None
insertStretch(index:int,stretch:int)	根据索引插入可伸缩空间	None
count()	获取控件、布局、占位空间的数量	int
maximumSize()	获取最大尺寸	QSize
minimumSize()	获取最小尺寸	QSize
setDirection(QBoxLayout.Direction)	设置布局的方向，其中 QBoxLayout.LeftToRight：从左到右水平布局 QBoxLayout.RightToLeft：从右到左水平布局 QBoxLayout.TopToBottom：从上到下竖直布局 QBoxLayout.BottomToTop：从下到上竖直布局	None
setGeometry(QRect)	设置左上角的位置，以及宽度、高度	None
setSpacing(spacing:int)	设置布局内部控件之间的间隙	None
spacing()	获取内部控件之间的间隙	int
setStretch(index:int,stretch:int)	根据索引设置控件或布局的伸缩系数	None
stretch(index:int)	根据索引获取某个控件的伸缩系数	int
setStretchFactor(QWidget,stretch:int)	设置控件的伸缩系数，若成功，则返回值为 True	bool
setStretchFactor(QLayout,stretch:int)	设置布局的伸缩系数，若成功，则返回值为 True	bool
setContentsMargins(int,int,int,int)	设置布局内的控件与边框的页边距	None
setContentsMargins(margins:QMargins)	同上	None
setSizeConstraint(QLayout.SizeConstraint)	设置控件随窗口宽和高改变的变化方式	None

在 PySide6 中，QLayout.SizeConstraint 的枚举值见表 5-3。

表 5-3　QLayout.SizeConstraint 的枚举值

枚　举　值	说　明
QLayout.SetDefaultConstraint	控件的最小宽和高根据 setMinimumSize()方法确定
QLayout.SetNoConstraint	控件宽和高的变化量不受限制
QLayout.SetMinimumSize	控件的宽和高为 setMinimumSize()方法设定的宽和高
QLayout.SetMaximumSize	控件的宽和高为 setMaximumSize()方法设定的宽和高
QLayout.SetMinAndMaxSize	控件的宽和高在最大值和最小值之间变化

【实例 5-1】 创建一个窗口，该窗口包含 4 个按压按钮，并设置水平布局，代码如下：

```
# === 第 5 章 代码 demo1.py === #
import sys
from PySide6.QtWidgets import QApplication,QWidget,QPushButton,QHBoxLayout

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

```
        self.setGeometry(200,200,560,220)
        self.setWindowTitle('QHBoxLayout 类')
        #创建 4 个按压按钮
        btn1 = QPushButton('东方')
        btn2 = QPushButton('西方')
        btn3 = QPushButton('南部')
        btn4 = QPushButton('北部')
        #创建水平布局对象
        hbox = QHBoxLayout()
        #添加控件
        hbox.addWidget(btn1)
        hbox.addWidget(btn2)
        hbox.addWidget(btn3)
        hbox.addWidget(btn4)
        #设置主窗口的布局方式
        self.setLayout(hbox)


if __name__ == '__main__':
    app = QApplication(sys.argv)
    win = Window()
    win.show()
    sys.exit(app.exec())
```

运行结果如图 5-2 和图 5-3 所示。

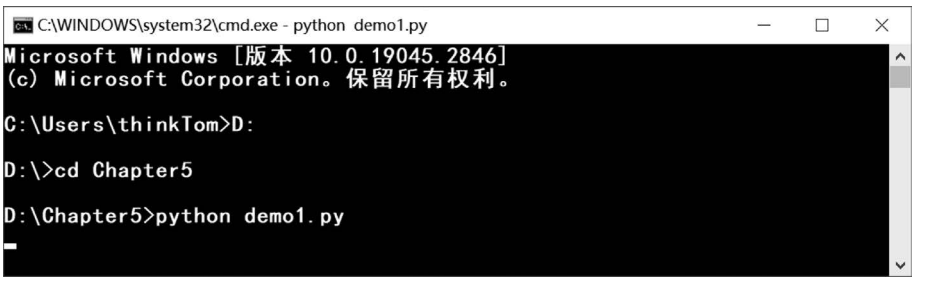

图 5-2　运行代码 demo1.py

图 5-3　代码 demo1.py 的运行结果

【实例 5-2】 创建一个窗口，该窗口包含 4 个按压按钮，并设置垂直布局，代码如下：

```
# === 第 5 章 代码 demo2.py === #
import sys
from PySide6.QtWidgets import QApplication,QWidget,QPushButton,QVBoxLayout

class Window(QWidget):
    def __init__(self):
        super().__init__()
        self.setGeometry(200,200,560,220)
        self.setWindowTitle('QVBoxLayout 类')
        #创建 4 个按压按钮
        btn1 = QPushButton('东方')
        btn2 = QPushButton('西方')
        btn3 = QPushButton('南部')
        btn4 = QPushButton('北部')
        #创建垂直布局对象
        vbox = QVBoxLayout()
        #添加控件
        vbox.addWidget(btn1)
        vbox.addWidget(btn2)
        vbox.addWidget(btn3)
        vbox.addWidget(btn4)
        #设置主窗口的布局方式
        self.setLayout(vbox)


if __name__ == '__main__':
    app = QApplication(sys.argv)
    win = Window()
    win.show()
    sys.exit(app.exec())
```

运行结果如图 5-4 所示。

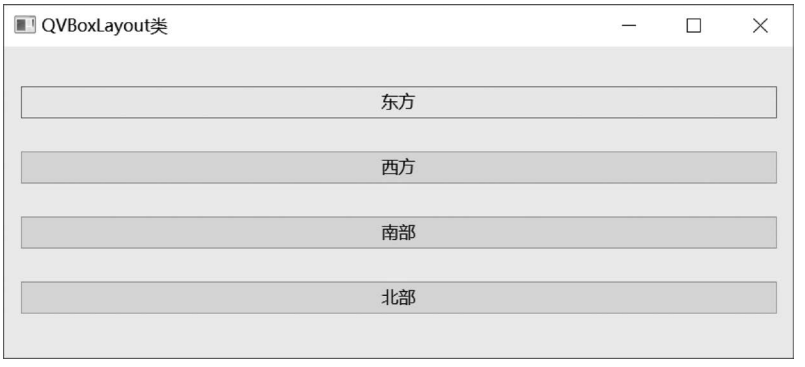

图 5-4　代码 demo2.py 的运行结果

在 PySide6 中，可以根据实际情况嵌套使用水平布局和垂直布局，即在垂直布局下使用水平布局，或在水平布局下使用垂直布局。

【实例 5-3】 创建一个窗口,该窗口包含 8 个按压按钮,前 4 个按钮使用水平布局,后 4 个按钮也使用水平布局,但总体使用垂直布局,代码如下:

```
# === 第 5 章 代码 demo3.py === #
import sys
from PySide6.QtWidgets import (QApplication,QWidget,QPushButton,
    QHBoxLayout,QVBoxLayout)

class Window(QWidget):
    def __init__(self):
        super().__init__()
        self.setGeometry(200,200,560,220)
        self.setWindowTitle('QHBoxLayout、QVBoxLayout 类')
        #创建 8 个按压按钮
        btn1 = QPushButton('东方')
        btn2 = QPushButton('西方')
        btn3 = QPushButton('南部')
        btn4 = QPushButton('北部')
        btn5 = QPushButton('甲型')
        btn6 = QPushButton('乙型')
        btn7 = QPushButton('丙型')
        btn8 = QPushButton('丁型')
        #创建水平布局对象 1
        hbox1 = QHBoxLayout()
        hbox1.addWidget(btn1)
        hbox1.addWidget(btn2)
        hbox1.addWidget(btn3)
        hbox1.addWidget(btn4)
        #创建水平布局对象 2
        hbox2 = QHBoxLayout()
        hbox2.addWidget(btn5)
        hbox2.addWidget(btn6)
        hbox2.addWidget(btn7)
        hbox2.addWidget(btn8)
        #创建竖直布局对象
        vbox = QVBoxLayout()
        #添加子布局对象
        vbox.addLayout(hbox1)
        vbox.addLayout(hbox2)
        #设置主窗口的布局方式
        self.setLayout(vbox)


if __name__ == '__main__':
    app = QApplication(sys.argv)
    win = Window()
    win.show()
    sys.exit(app.exec())
```

运行结果如图 5-5 所示。

【实例 5-4】 创建一个窗口,该窗口包含 8 个按压按钮,前 4 个按钮使用垂直布局,后 4 个按钮也使用垂直布局,但总体使用水平布局,代码如下:

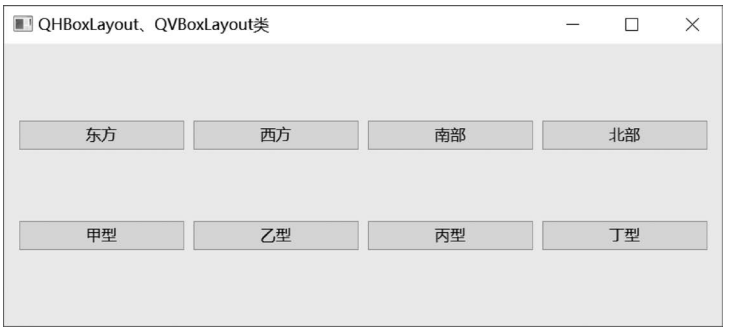

图 5-5　代码 demo3. py 的运行结果

```
# === 第 5 章 代码 demo4.py === #
import sys
from PySide6.QtWidgets import (QApplication,QWidget,QPushButton,QHBoxLayout,QVBoxLayout)

class Window(QWidget):
    def __init__(self):
        super().__init__()
        self.setGeometry(200,200,560,220)
        self.setWindowTitle('QHBoxLayout、QVBoxLayout 类')
        # 创建 8 个按压按钮
        self.btn1 = QPushButton('东方')
        self.btn2 = QPushButton('西方')
        self.btn3 = QPushButton('南部')
        self.btn4 = QPushButton('北部')
        self.btn5 = QPushButton('甲型')
        self.btn6 = QPushButton('乙型')
        self.btn7 = QPushButton('丙型')
        self.btn8 = QPushButton('丁型')
        # 创建垂直布局对象 1
        vbox1 = QVBoxLayout()
        vbox1.addWidget(self.btn1)
        vbox1.addWidget(self.btn2)
        vbox1.addWidget(self.btn3)
        vbox1.addWidget(self.btn4)
        # 创建垂直布局对象 2
        vbox2 = QVBoxLayout()
        vbox2.addWidget(self.btn5)
        vbox2.addWidget(self.btn6)
        vbox2.addWidget(self.btn7)
        vbox2.addWidget(self.btn8)
        # 创建水平布局对象
        hbox = QHBoxLayout()
        # 添加子布局对象
        hbox.addLayout(vbox1)
        hbox.addLayout(vbox2)
        # 设置主窗口的布局方式
        self.setLayout(hbox)
```

```
if __name__ == '__main__':
    app = QApplication(sys.argv)
    win = Window()
    win.show()
    sys.exit(app.exec())
```

运行结果如图 5-6 所示。

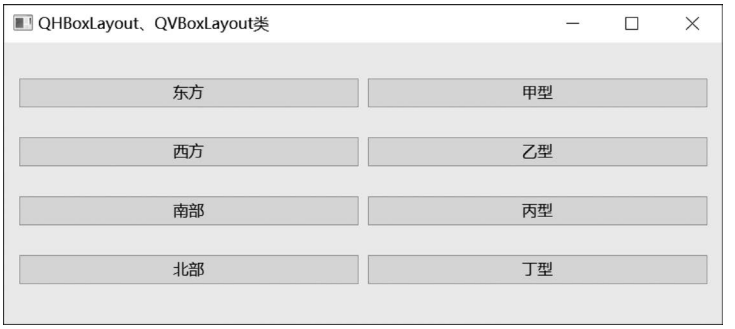

图 5-6　代码 demo4.py 的运行结果

5.1.3　栅格布局(QGridLayout)

栅格布局也称为网格布局,栅格布局可以把窗口划分为多行多列,而产生很多单元格,然后将控件或子布局放置到单元格中。在 PySide6 中,使用 QGridLayout 类创建栅格布局对象,其构造函数如下:

```
QGridLayout(parent:QWidget = None)
```

其中,parent 表示父窗口或父容器。

在 PySide6 中,QGridLayout 类的常用方法见表 5-4。

表 5-4　QGridLayout 类的常用方法

方法及参数类型	说　明	返回值类型
addWidget(QWidget)	在第 1 列的末尾添加控件	None
addWidget(QWidget, row: int, column: int, Qt.Alignment)	在指定的行列位置添加控件	None
addWidget(QWidget, row: int, column: int, rowSpan:int,columnSpan:int,Qt.Alignment)	在指定的行列位置添加控件,该控件可以跨多行多列	None
addLayout(QLayout, row: int, column: int, Qt.Alignment)	在指定的行列位置添加子布局	None
addLayout(QLayout, row: int, column: int, rowSpan:int,columnSpan:int,Qt.Alignment)	在指定的行列位置添加子布局,该子布局可以跨多行多列	None

续表

方法及参数类型	说　明	返回值类型
setRowStretch(row:int,stretch:int)	设置行的伸缩系数	None
setColumnStretch(column:int,stretch:int)	设置列的伸缩系数	None
setHorizontalSpacing(spacing:int)	设置控件的水平间距	None
setVerticalSpacing(spacing:int)	设置控件的竖直间距	None
setSpacing(spacing:int)	设置控件的水平和竖直间距	None
rowCount()	获取行数	int
columnCount()	获取列数	int
setRowMinimumHeight(row:int,minSize:int)	设置行的最小高度	None
setColumnMinimumWidth(column:int,MiniSize:int)	设置列的最小宽度	None
setGeometry(QRect)	设置栅格布局的位置,以及宽和高	None
setContentsMargins(left:int,top:int,right:int,bottom:int)	设置布局内控件与边框的页边距	None
setContentsMargins(margins:QMargins)	同上	None
setSizeConstraint(QLayout.SizeConstraint)	设置控件随窗口宽和高改变时的变化方式	None
cellRect(row:int,column:int)	设置单元格的矩形区域	QRect

【实例 5-5】 创建一个窗口,该窗口包含 12 个按压按钮,使用栅格布局将这 12 个按钮分为 3 行 4 列,代码如下:

```
# === 第 5 章 代码 demo5.py === #
import sys
from PySide6.QtWidgets import QApplication,QWidget,QPushButton,QGridLayout

class Window(QWidget):
    def __init__(self):
        super().__init__()
        self.setGeometry(200,200,560,220)
        self.setWindowTitle('QGridLayout 类')
        # 创建 12 个按压按钮
        self.btn0 = QPushButton('0')
        self.btn1 = QPushButton('1')
        self.btn2 = QPushButton('2')
        self.btn3 = QPushButton('3')
        self.btn4 = QPushButton('4')
        self.btn5 = QPushButton('5')
        self.btn6 = QPushButton('6')
        self.btn7 = QPushButton('7')
        self.btn8 = QPushButton('8')
        self.btn9 = QPushButton('9')
        self.btn11 = QPushButton(' + ')
        self.btn12 = QPushButton(' = ')
        # 创建栅格布局对象
```

```
        grid = QGridLayout()
        #添加控件
        grid.addWidget(self.btn0,0,0)
        grid.addWidget(self.btn1,0,1)
        grid.addWidget(self.btn2,0,2)
        grid.addWidget(self.btn3,0,3)
        grid.addWidget(self.btn4,1,0)
        grid.addWidget(self.btn5,1,1)
        grid.addWidget(self.btn6,1,2)
        grid.addWidget(self.btn7,1,3)
        grid.addWidget(self.btn8,2,0)
        grid.addWidget(self.btn9,2,1)
        grid.addWidget(self.btn11,2,2)
        grid.addWidget(self.btn12,2,3)
        #设置主窗口的布局方式
        self.setLayout(grid)


if __name__ == '__main__':
    app = QApplication(sys.argv)
    win = Window()
    win.show()
    sys.exit(app.exec())
```

运行结果如图 5-7 所示。

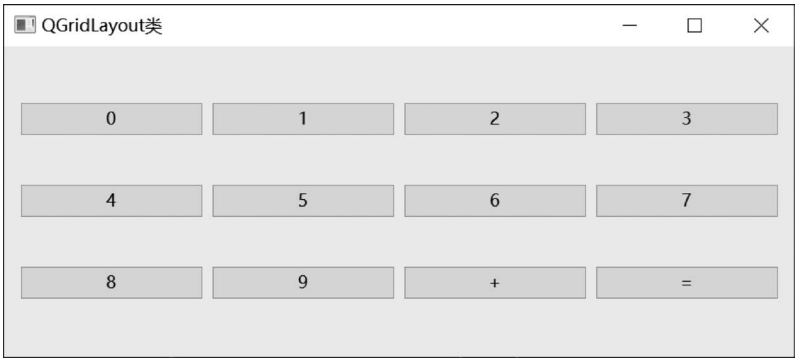

图 5-7　代码 demo5.py 的运行结果

5.1.4　表单布局(QFormLayout)

表单布局一般由两列多行构成,通常左列放置标签控件,右列放置单行文本框控件或数字输入框控件。在 PySide6 中,使用 QFormLayout 类创建表单布局对象,其构造函数如下:

```
QFormLayout(parent:QWidget = None)
```

其中,parent 表示父窗口或父容器。

在 PySide6 中,QFormLayout 类的常用方法见表 5-5。

表 5-5　QFormLayout 类的常用方法

方法及参数类型	说　明	返回值类型
addRow(label:QWidget,field:QWidget)	末尾添加一行,两个控件分别在左右	None
addRow(label:QWidget,field:QLayout)	末尾添加一行,控件在左,子布局在右	None
addRow(labelText:str,field:QWidget)	末尾添加一行,左侧创建名称为 str 的标签控件,右侧为控件	None
addRow(labelText:str,field:QLayout)	末尾添加一行,左侧创建名称为 str 的标签控件,右侧为子布局	None
addRow(widget:QWidget)	末尾添加一行,只有一个控件,占据左右两列	None
addRow(layout:QLayout)	末尾添加一行,只有一个子布局,占据左右两列	None
insertRow(row:int,QWidget,QWidget)	在第 row 行插入一行,两个控件分别在左右	None
insertRow(row:int,QWidget,QLayout)	在第 row 行插入一行,控件在左,子布局在右	None
insertRow(row:int,str,QWidget)	在第 row 行插入一行,左侧创建名称为 str 的标签控件,右侧是控件	None
insertRow(row:int,str,QLayout)	在第 row 行插入一行,左侧创建名称为 str 的标签控件,右侧是子布局	None
insertRow(row:int,QWidget)	在第 row 行插入,只有一个控件,占据左右两列	None
insertRow(row:int,QLayout)	在第 row 行插入,只有一个子布局,占据左右两列	None
removeRow(row:int)	删除第 row 行及其控件	None
removeRow(layout:QLayOut)	删除子布局	None
removeRow(widget:QWidget)	删除控件	None
setHorizontalSpacing(spacing:int)	设置水平方向的间距	None
setVerticalSpacing(spacing:int)	设置竖直方向的间距	None
setRowWrapPolicy(QFormLayout.RowWrapPolicy)	设置左列控件和右列控件的换行策略,其中 QFormLayout.DontWrapRows 表示右列的控件始终在左列控件的右侧;QFormLayout.WrapLongRows 表示若左侧的控件比较长,则会挤压右侧控件,如果左侧控件占据一行,则右侧控件会放在下一行;QFormLayout.WrapAllRows 表示左侧控件始终在右侧控件之上	None
rowCount()	获取表单布局中行的数量	int
setLabelAlignment(Qt.Alignment)	设置表单布局中左列的对齐方式	None
setFormAlignment(Qt.Alignment)	设置表单布局中右列的对齐方式	None
setContentsMargins(int,int,int,int)	设置布局内控件与边框的页边距	None
setContentsMargins(QMargins)	同上	None
setFieldGrowthPolicy(QFormLayout.FieldGrowthPolicy)	设置可伸缩控件的伸缩方式	None
setSizeConstraint(QLayout.SizeConstraint)	设置控件随窗口大小改变时的改变方式	None

在 PySide6 中,伸缩方式 QFormLayout.FieldGrowthPolicy 的枚举值见表 5-6。

表 5-6 QFormLayout.FieldGrowthPolicy 的枚举值

枚 举 值	说 明
QFormLayout.FieldStayAtSizeHint	控件的伸缩量不会超过有效的范围(由 setHint()方法设置)
QFormLayout.ExpandingFieldsGrowth	如果控件设置了最小伸缩量或使用 setSizePolicy()设置了属性,则使其扩充到可以使用的范围,否则控件在有效的范围内变化
QFormLayout.AllNonFixedGrow	如果使用 setSizePolicy()方法设置了属性,则使其扩充到可以使用的空间

【实例 5-6】 创建一个窗口,该窗口包含两个标签、两个单行文本框,使用表单布局排列控件,代码如下:

```
# === 第 5 章 代码 demo6.py === #
import sys
from PySide6.QtWidgets import (QApplication,QWidget,QLabel,QFormLayout,QLineEdit)

class Window(QWidget):
    def __init__(self):
        super().__init__()
        self.setGeometry(200,200,560,220)
        self.setWindowTitle('QFormLayout 类')
        #创建两个标签、两个单行文本框
        name = QLabel("账号(UserName):")
        code = QLabel("密码(Password):")
        self.lineEdit1 = QLineEdit()
        self.lineEdit2 = QLineEdit()
        #创建表单布局对象
        form = QFormLayout()
        #添加行
        form.addRow(name,self.lineEdit1)
        form.addRow(code,self.lineEdit2)
        #设置主窗口的布局方式
        self.setLayout(form)


if __name__ == '__main__':
    app = QApplication(sys.argv)
    win = Window()
    win.show()
    sys.exit(app.exec())
```

运行结果如图 5-8 所示。

【实例 5-7】 创建一个窗口,该窗口为一个登录界面,使用表单布局排列控件,代码如下:

```
# === 第 5 章 代码 demo7.py === #
import sys
from PySide6.QtWidgets import (QApplication,QWidget,QFormLayout,QLineEdit,QPushButton)
```

图 5-8　代码 demo6.py 的运行结果

```
class Window(QWidget):
    def __init__(self):
        super().__init__()
        self.setGeometry(200,200,560,220)
        self.setWindowTitle('QFormLayout 类')
        #创建两个单行文本框、两个按压按钮
        self.lineEdit1 = QLineEdit()
        self.lineEdit2 = QLineEdit()
        btn1 = QPushButton("确定")
        btn2 = QPushButton("取消")
        #创建表单布局对象
        form = QFormLayout()
        #添加行
        form.addRow("账号(&UserName):",self.lineEdit1)
        form.addRow("密码(&Password):",self.lineEdit2)
        form.addRow(btn1)
        form.addRow(btn2)
        #设置主窗口的布局方式
        self.setLayout(form)


if __name__ == '__main__':
    app = QApplication(sys.argv)
    win = Window()
    win.show()
    sys.exit(app.exec())
```

运行结果如图 5-9 所示。

5.1.5　堆叠布局（QStackedLayout）

在 PySide6 中，使用 QStackedLayout 类创建堆叠布局对象。使用堆叠布局可以创建多个页面，但每次只显示其中一个页面。QStackedLayout 类的构造函数如下：


```
QStackedLayout(parent:QWidget = None)
QStackedLayout(parent:QLayout = None)
```

图 5-9　代码 demo7.py 的运行结果

其中,parent 表示父窗口、父容器或父布局。

在 PySide6 中,QStackedLayout 类的常用方法见表 5-7。

表 5-7　QStackedLayout 类的常用方法

方法及参数类型	说　明	返回值的类型
[slot]setCurrentIndex(index:int)	设置当前索引	None
[slot]setCurrentWidget(QWidget)	设置当前控件	None
addWidget(QWidget)	添加控件	None
addLayout(QLayout)	添加布局	None
currentIndex()	获取当前索引	int
currentWidget()	获取当前控件	QWidget
insertWidget(index:int,QWidget)	根据索引插入控件	None

在 PySide6 中,QStackedLayout 类的信号见表 5-8。

表 5-8　QStackedLayout 类的信号

信　号	说　明
currentChanged(index:int)	当前控件发生变化时发送信号
widgetRemoved(index:int)	当布局内的控件被移除时发送信号

【实例 5-8】 创建一个窗口,使用堆叠布局使该窗口包含 3 个页面,使用下拉列表框切换页面,代码如下:

```
# === 第 5 章 代码 demo8.py === #
import sys
from PySide6.QtWidgets import (QApplication, QWidget, QVBoxLayout, QLabel, QStackedLayout,
QComboBox,QHBoxLayout)

class Window(QWidget):
    def __init__(self):
        super().__init__()
        self.setGeometry(200,200,560,220)
        self.setWindowTitle('QStackedLayout 类')
        #窗口使用垂直布局
```

```
        vbox = QVBoxLayout()
        self.setLayout(vbox)
        #创建下拉列表对象
        combo1 = QComboBox(self)
        combo1.addItem("页面 1")
        combo1.addItem("页面 2")
        combo1.addItem("页面 3")
        vbox.addWidget(combo1)
        #创建堆叠布局对象
        stacked1 = QStackedLayout()
        #创建页面 1
        page1 = QWidget()
        layout1 = QHBoxLayout()
        label1 = QLabel("这是第 1 个页面。")
        layout1.addWidget(label1)
        page1.setLayout(layout1)
        #创建页面 2
        page2 = QWidget()
        layout2 = QHBoxLayout()
        label2 = QLabel("这是第 2 个页面。")
        layout2.addWidget(label2)
        page2.setLayout(layout2)
        #创建页面 3
        page3 = QWidget()
        layout3 = QHBoxLayout()
        label3 = QLabel("这是第 3 个页面。")
        layout3.addWidget(label3)
        page3.setLayout(layout3)
        #向堆叠布局对象中添加页面
        stacked1.addWidget(page1)
        stacked1.addWidget(page2)
        stacked1.addWidget(page3)
        #向垂直布局中添加堆叠布局
        vbox.addLayout(stacked1)
        #使用信号/槽机制
        combo1.activated.connect(stacked1.setCurrentIndex)


if __name__ == '__main__':
    app = QApplication(sys.argv)
    win = Window()
    win.show()
    sys.exit(app.exec())
```

运行结果如图 5-10 所示。

【实例 5-9】 创建一个窗口，使用堆叠布局使该窗口包含 3 个页面，使用按压按钮切换页面，代码如下：

```
#=== 第 5 章 代码 demo9.py === #
import sys
from PySide6. QtWidgets import (QApplication, QWidget, QVBoxLayout, QLabel, QStackedLayout,
QPushButton,QHBoxLayout)
```

图 5-10　代码 demo8.py 的运行结果

```
class Window(QWidget):
    def __init__(self):
        super().__init__()
        self.setGeometry(200,200,560,220)
        self.setWindowTitle('QStackedLayout 类')
        #窗口使用垂直布局
        vbox = QVBoxLayout()
        self.setLayout(vbox)
        #创建堆叠布局对象
        self.stacked = QStackedLayout()
        #创建页面 1
        page1 = QWidget()
        layout1 = QHBoxLayout()
        label1 = QLabel("这是第 1 个页面。")
        layout1.addWidget(label1)
        page1.setLayout(layout1)
        #创建页面 2
        page2 = QWidget()
        layout2 = QHBoxLayout()
        label2 = QLabel("这是第 2 个页面。")
        layout2.addWidget(label2)
        page2.setLayout(layout2)
        #创建页面 3
        page3 = QWidget()
        layout3 = QHBoxLayout()
        label3 = QLabel("这是第 3 个页面。")
        layout3.addWidget(label3)
        page3.setLayout(layout3)
        #向堆叠布局对象中添加页面
        self.stacked.addWidget(page1)
        self.stacked.addWidget(page2)
        self.stacked.addWidget(page3)
        vbox.addLayout(self.stacked)
        #创建水平布局对象,并添加 3 个按压按钮
        btn_layout = QHBoxLayout()
        btn1 = QPushButton("页面 1")
```

```
        btn2 = QPushButton("页面 2")
        btn3 = QPushButton("页面 3")
        btn_layout.addWidget(btn1)
        btn_layout.addWidget(btn2)
        btn_layout.addWidget(btn3)
        vbox.addLayout(btn_layout)
        #使用信号/槽机制
        btn1.clicked.connect(lambda:self.stacked.setCurrentIndex(0))
        btn2.clicked.connect(lambda:self.stacked.setCurrentIndex(1))
        btn3.clicked.connect(lambda:self.stacked.setCurrentIndex(2))


if __name__ == '__main__':
    app = QApplication(sys.argv)
    win = Window()
    win.show()
    sys.exit(app.exec())
```

运行结果如图 5-11 所示。

图 5-11　代码 demo9.py 的运行结果

5.2　容器：装载更多的控件

在 PySide6 中，可以使用多种容器类创建多种容器控件。可以将其他控件放置到容器控件内，容器控件被作为其他控件的父容器或载体。容器控件可以对其内部控件进行管理。

PySide6 提供的容器类见表 5-9。

表 5-9　PySide6 中的容器类

容　器　类	说　　明	容　器　类	说　　明
QGroupBox	分组框控件	QFrame	框架控件
QScrollArea	滚动区控件	QTabWidget	切换卡控件

续表

容 器 类	说 明	容 器 类	说 明
QStackedWidget	堆叠控件	QToolBox	工具箱控件
QWidget	容器窗口控件	QMdiArea	多文档区
QDockWidget	停靠窗口控件	QAxWidget	插件窗口控件

表 5-8 容器类对应了 Qt Designer 中工具箱中的控件，如图 5-12 所示。

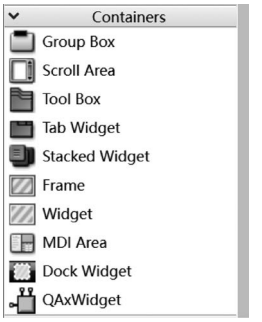

图 5-12 Qt Designer 中的容器控件

本节主要介绍容器类中的 QGroupBox、QFrame、QScrollArea、QTabWidget、QStackedWidget、QToolBox、QAxWidget。其他的容器类将在后面的章节介绍。

5.2.1 分组框控件

在 PySide6 中，可以使用 QGroupBox 类创建分组框控件。分组框控件可以容纳一组单选按钮控件或复选框控件，并带有一条边框和标题栏，而且可以为标题栏设置勾选项。

在 PySide6 中，QGroupBox 类是 QWidget 类的子类，其构造函数如下：

```
QGroupBox(parent:QWidget = None)
QGroupBox(title:str,parent:QWidget = None)
```

其中，parent 表示父窗口或父容器；title 表示分组框控件上显示的文本。

在 PySide6 中，QGroupBox 类的常用方法见表 5-10。

表 5-10 QGroupBox 类的常用方法

方法及参数类型	说 明	返回值的类型
[slot]setCheckable(bool)	设置标题栏上是否有勾选项	None
setTitle()	设置标题的名称	None
title()	获取标题的名称	str
setFlat(bool)	设置是否处于扁平状态	None
isFlat(bool)	获取是否处于扁平状态	bool
isCheckable()	获取标题栏是否有勾选项	bool
setAlignment(Qt. Alignment)	设置标题栏的对齐方式	None
alignment()	获取标题栏的对齐方式	Qt. Alignment
setGeometry(x:int,y:int,w:int,h:int)	设置分组框控件在父窗口中的位置、宽度、高度	None
setGeometry(QRect)	同上	None
resize(QSize)	设置分组框控件的宽度、高度	None
resize(w:int,h:int)	同上	None
setLayout(QLayout)	设置分组框中的布局	None

【实例 5-10】 创建一个窗口，窗口中有一个分组框控件，分组框控件中有 5 个单选按

钮，代码如下：

```
# === 第 5 章 代码 demo10.py === #
import sys
from PySide6.QtWidgets import (QApplication,QWidget,QRadioButton,QGroupBox,QHBoxLayout)
from PySide6.QtGui import QFont

class Window(QWidget):
    def __init__(self):
        super().__init__()
        self.setGeometry(200,200,560,220)
        self.setWindowTitle('QGroupBox 类')
        self.setFont(QFont("黑体",14))
        # 创建 QGroupBox 对象
        group = QGroupBox(self)
        group.setTitle("选择北宋时期的人物")
        # 创建 5 个单选按钮
        radio1 = QRadioButton("李白")
        radio2 = QRadioButton("杜甫")
        radio3 = QRadioButton("陶渊明")
        radio4 = QRadioButton("苏轼")
        radio5 = QRadioButton("司马迁")
        # 创建水平布局对象
        hbox = QHBoxLayout()
        # 添加控件
        hbox.addWidget(radio1)
        hbox.addWidget(radio2)
        hbox.addWidget(radio3)
        hbox.addWidget(radio4)
        hbox.addWidget(radio5)
        # 设置 group 对象的布局方式
        group.setLayout(hbox)


if __name__ == '__main__':
    app = QApplication(sys.argv)
    win = Window()
    win.show()
    sys.exit(app.exec())
```

运行结果如图 5-13 所示。

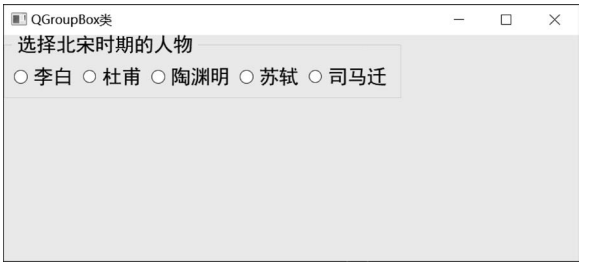

图 5-13　代码 demo10.py 的运行结果

在 PySide6 中,QGroupBox 类的信号见表 5-11。

表 5-11 QGroupBox 类的信号

信 号	说 明
clicked()	当被单击时发送信号
clicked(checked:bool)	当被单击时发送信号
toggle(bool)	当勾选状态发生变化时发送信号

【实例 5-11】 创建一个窗口,窗口中有一个设置了勾选项的分组框控件,分组框控件中有两个单选按钮。如果切换勾选项的状态,则打印提示信息,代码如下:

```
# === 第 5 章 代码 demo11.py === #
import sys
from PySide6.QtWidgets import (QApplication,QWidget,QRadioButton,QGroupBox,QHBoxLayout)
from PySide6.QtGui import QFont

class Window(QWidget):
    def __init__(self):
        super().__init__()
        self.setGeometry(200,200,500,200)
        self.setWindowTitle('QGroupBox 类')
        self.setFont(QFont("黑体",14))
        #创建 QGroupBox 对象
        self.group = QGroupBox(self)
        self.group.setTitle("性别")
        self.group.setCheckable(True)
        #创建两个单选按钮
        radio1 = QRadioButton("男")
        radio2 = QRadioButton("女")
        #创建水平布局对象
        hbox = QHBoxLayout()
        #添加控件
        hbox.addWidget(radio1)
        hbox.addWidget(radio2)
        #设置 group 对象的布局方式
        self.group.setLayout(hbox)
        #使用信号/槽机制
        self.group.toggled.connect(self.echo_text)

    #自定义槽函数
    def echo_text(self,state):
        if state == True:
            print("已经勾选")
        else:
            print("取消勾选")


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

```
win = Window()
win.show()
sys.exit(app.exec())
```

运行结果如图 5-14 所示。

图 5-14　代码 demo11.py 的运行结果

5.2.2　框架控件(QFrame)

在 PySide6 中,可以使用 QFrame 类创建框架控件。框架控件可以容纳各种窗口控件,但框架控件没有自己特有的信号或槽函数,不接受用户的输入信息。框架控件可以提供一个框架,可以设置外边框的样式、线宽。

在 PySide6 中,QFrame 类是 QWidget 类的子类,其构造函数如下:

```
QFrame(parent:QWidget = None,f:Qt.WindowFlags = Default(Qt.WindowFlags))
```

其中,parent 表示父窗口或父容器。

在 PySide6 中,QFrame 类的常用方法见表 5-12。

表 5-12　QFrame 类的常用方法

方法及参数类型	说　　明	返回值类型
setFrameShadow (QFrame.Shadow)	设置框架控件的阴影形式,参数值为 QFrame.Plain(平面)、QFrame.Raised(凸起)、QFrame.Sunken(凹陷)	None
frameShadow()	获取窗口的阴影形式	QFrame.Shadow
setFrameShape (QFrame.Shape)	设置框架控件的边框形状,其中 QFrame.NoFrame:无边框,默认值 QFrame.Box:矩形框,边框内部不填充 QFrame.Panel:面板,边框线内部填充 QFrame.WinPanel:Windows 风格的面板,边框线宽为 2 像素 QFrame.HLine:边框线只在中间有一条水平线 QFrame.VLine:边框线只在中间有一条竖直线 QFrame.StyledPanel:根据当前的 GUI 画矩形面板	None

续表

方法及参数类型	说　明	返回值类型
frameShape()	获取框架控件的边框形状	QFrame.Shape
setFrameStyle(int)	设置边框的样式	None
frameStyle()	获取边框的样式	int
setLineWidth(int)	设置边框线的宽度	None
lineWidth()	获取边框线的宽度	int
setMidLineWidth(int)	设置边框线的中间线的宽度	None
midLineWidth()	获取边框线的中间线的宽度	int
frameWidth()	获取边框内线的宽度	int
setFrameRect(QRect)	设置边框线所在的范围	None
frameRect()	获取框架控件所在的范围	QRect
drawFrame(QPainter)	绘制边框线	None
setLayout(QLayout)	设置框架控件中的布局	None
setGeometry(QRect)	设置框架控件左上角的位置,以及宽度和高度	None
resize(QSize)	设置框架控件的宽度、高度	None
resize(w:int,h:int)	设置框架控件的宽度、高度	None

【实例 5-12】 创建一个窗口,窗口中有一个显示边框的框架控件。框架控件内部是一个登录界面,代码如下:

```
# === 第 5 章 代码 demo12.py === #
import sys
from PySide6.QtWidgets import (QApplication, QWidget, QLabel, QFormLayout, QLineEdit, QFrame,
QPushButton)

class Window(QWidget):
    def __init__(self):
        super().__init__()
        self.setGeometry(200,200,560,220)
        self.setWindowTitle('QFrame 类')
        #创建 Frame 对象
        frame1 = QFrame(self)
        frame1.setFrameShape(QFrame.Box)
        #创建两个标签、两个单行文本框
        name = QLabel("账号(UserName):")
        code = QLabel("密码(Password):")
        self.lineEdit1 = QLineEdit()
        self.lineEdit2 = QLineEdit()
        btn1 = QPushButton("确定")
        btn2 = QPushButton("取消")
        #创建表单布局对象
        form = QFormLayout()
        #添加行
        form.addRow(name,self.lineEdit1)
        form.addRow(code,self.lineEdit2)
        form.addRow(btn1)
```

```
        form.addRow(btn2)
        #设置 Frame 对象的布局方式
        frame1.setLayout(form)


if __name__ == '__main__':
    app = QApplication(sys.argv)
    win = Window()
    win.show()
    sys.exit(app.exec())
```

运行结果如图 5-15 所示。

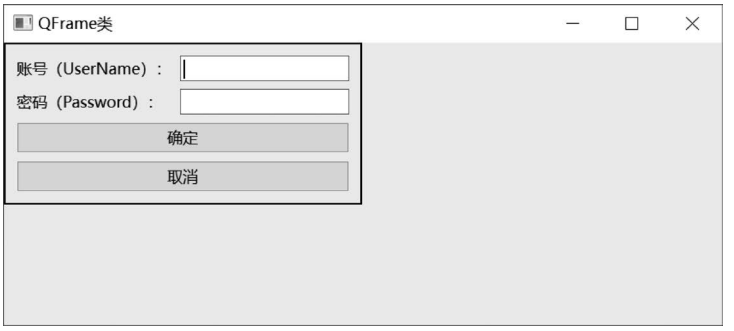

图 5-15　代码 demo12.py 的运行结果

注意：框架控件的边框线由外线、内线、中间线构成。可使用 setLineWidth()方法设置外线的宽度，使用 setMidLineWidth()方法设置中间线的宽度，可使用 frameWidth()获取边框内线的宽度。

5.2.3　滚动区控件(QScrollArea)

在 PySide6 中，可使用 QScrollArea 类创建滚动区控件。滚动区控件可以容纳其他控件，如果内部控件的宽和高超过滚动区控件的宽和高，则滚动区控件会自动提供水平滚动条、竖直滚动条。用户可通过拖动滚动条的方法查看滚动区控件内部的所有内容。QScrollArea 类的继承关系如图 5-16 所示。

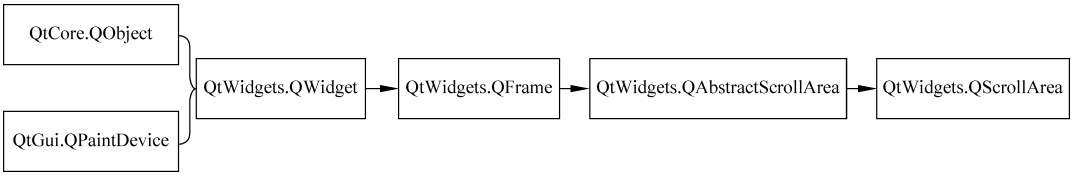

图 5-16　QScrollArea 类的继承关系

在 PySide6 中，QScrollArea 类的构造函数如下：

```
QScrollArea(parent:QWidget = None)
```

其中,parent 表示父窗口或父容器。

QScrollArea 类的常用方法见表 5-13。

表 5-13 QScrollArea 类的常用方法

方法及参数类型	说 明	返回值的类型
setWidget(QWidget)	将某个控件设置为可滚动显示的控件	None
widget()	获取可滚动显示的控件	QWidget
setWidgetResizable(bool)	设置内部控件是否可调节宽和高,尽量不显示滚动条	None
widgetResizable()	获取内部控件是否可调节宽和高	bool
setAlignment(Qt. Alignment)	设置内部控件在滚动区控件的对齐方式	None
alignment()	获取内部控件在滚动区控件的对齐方式	Qt. Alignment
ensureVisible(x:int,y:int,xmargin:int=50,ymargin:int=50)	自动移动滚动条的位置,确保坐标(x,y)的像素是可见的,并且像素到边框的距离分别为 xmargin、ymargin,其默认值为 50 像素	None
ensureVisible(childWidget:QWidget,xmargin:int=50,ymargin:int=50)	自动移动滚动条的位置,确保控件 childWidget 是可见的	None
setHorizontalScrollBarPolicy(Qt. ScrollBarPolicy)	设置竖直滚动条的显示策略	None
setVerticalScrollBarPolicy(Qt. ScrollBarPolicy)	设置水平滚动条的显示策略	None

在表 5-13 中,Qt. ScrollBarPolicy 的枚举值为 Qt. ScrollBarAdNeeded(根据情况自动调整何时出现滚动条)、Qt. ScrollBarAlwaysOff(从不出现滚动条)、Qt. ScrollBarAlwaysOn(一直出现滚动条)。

【实例 5-13】 创建一个窗口,窗口中有一个滚动区控件。在该控件中显示一张图像,代码如下:

```
# === 第 5 章 代码 demo13.py === #
import sys
from PySide6.QtWidgets import (QApplication,QWidget,QLabel,QScrollArea)
from PySide6.QtGui import QPixmap

class Window(QWidget):
    def __init__(self):
        super().__init__()
        self.setGeometry(200,200,700,400)
        self.setWindowTitle('QScrollArea 类')
        #创建滚动区控件
        area = QScrollArea(self)
        label = QLabel()
```

```
        pic = QPixmap("D:\\Chapter5\\images\\cat1.png")
        label.setPixmap(pic)
        #将标签控件设置成可滚动显示的控件
        area.setWidget(label)


if __name__ == '__main__':
    app = QApplication(sys.argv)
    win = Window()
    win.show()
    sys.exit(app.exec())
```

运行结果如图 5-17 所示。

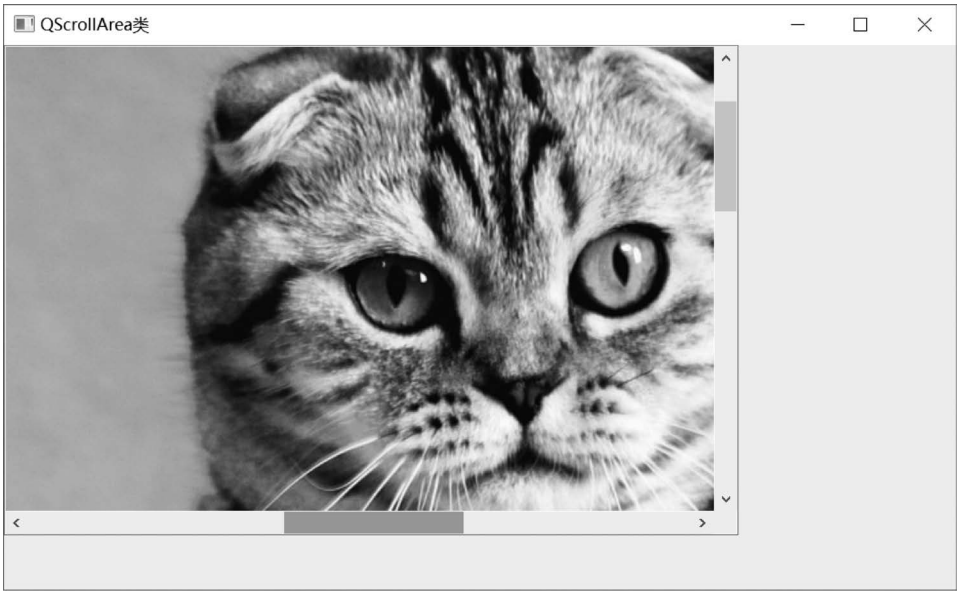

图 5-17　代码 demo13.py 的运行结果

5.2.4　切换卡控件(QTabWidget)

在 PySide6 中,可以使用 QTabWidget 类创建切换卡控件。切换卡控件由多张卡片组成,每张卡片就是一个窗口(QWidget)。可以根据实际需求,将不同的控件分别放置到不同的卡片上,这样便可以提高窗口空间的使用效率。

QTabWidget 类是 QWidget 类的子类,其构造函数如下:

```
QTabWidget(parent:QWidget = None)
```

其中,parent 表示父窗口或父容器。

在 PySide6 中,QTabWidget 类的常用方法见表 5-14。

表 5-14 QTabWidget 类的常用方法

方法及参数类型	说　明	返回值的类型
[slot]setCurrentIndex(index:int)	设置当前卡片的索引	None
[slot]setCurrentWidget(QWidget)	将窗口控件设置成当前卡片	None
addTab(QWidget,label:str)	在末尾添加一张卡片	None
addTab(QWidget,QIcon,label:str)	在末尾添加一张卡片	None
insertTab(index:int,QWidget,label,str)	在索引 int 处插入卡片	None
insertTab(index:int,QWidget,QIcon,label,str)	在索引 int 处插入卡片	None
widget(index:int)	根据索引获取卡片窗口	QWidget
clear()	清空所有卡片	None
count()	获取卡片数量	int
indexOf(QWidget)	获取某个窗口对应的卡片索引号	int
removeTab(index:int)	根据索引移除卡片	None
setCornerWidget(QWidget,Qt.Corner)	在角位置设置控件,Qt.Corner 的参数值可为 Qt.TopRightCorner、Qt.BottomRightCorner、Qt.TopLeftCorner、Qt.BottomLeftCorner	None
cornerWidget(Qt.Corner)	获取角位置的索引	QWidget
currentIndex()	获取当前卡片的索引	int
currentWidget()	获取当前卡片的窗口控件	QWidget
setDocumentMode(bool)	设置卡片是否为文档模式	None
documentMode()	获取卡片是否为文档模式	bool
setElideMode(Qt.TextElideMode)	设置卡片标题是否为省略模式,其中 Qt.ElideNone:没有省略号 Qt.ElideLeft:省略号在左侧 Qt.ElideMiddle:省略号在中间 Qt.ElideRight:省略号在右侧	None
setIconSize(QSize)	设置卡片图标的宽和高	None
iconSize()	获取卡片图标的宽和高	QSize
setMovable(bool)	设置卡片之间是否可以交换位置	None
isMovable()	获取卡片之间是否可以交换位置	bool
setTabBarAutoHide(bool)	当只有 1 张卡片时,设置卡片标题是否自动隐藏	None
tabBarAutoHide()	获取标题是否为自动隐藏	bool
setTabEnabled(index:int,bool)	设置是否激活索引为 int 的卡片	None
isTabEnabled(index:int)	获取索引为 int 的卡片是否激活	bool
setTabIcon(index:int,QIcon)	根据索引设置卡片的图标	None
tabIcon(index:int)	根据索引获取卡片的图标	QIcon
setTabPosition(QTabWidget.TabPosition)	设置标题栏的位置,参数值可为 QTabWidget.North、QTabWidget.South、QTabWidget.East、QTabWidget.West	None

续表

方法及参数类型	说　明	返回值的类型
setTabShape(QTabWidget. TabShape)	设置标题栏的形状，参数值可为 QTabWidget. Rounded、QTabWidget. Triangular	None
setTabText(index:int,str)	根据索引设置卡片的标题名称	None
tabText(index:int)	根据索引获取卡片的标题名称	str
setTabToolTip(index:int,str)	根据索引设置卡片的提示信息	None
tabToolTip(index:int)	根据索引获取卡片的提示信息	str
setVisible(bool)	设置切换卡是否可见	None
setTabsClosable(bool)	设置卡片标题上是否有关闭标识	None
tabsClosable()	获取卡片是否可以关闭	bool
setUserScrollButtons(bool)	设置是否有滚动按钮	None
userScrollButtons()	获取是否有滚动按钮	bool

【实例 5-14】 创建一个窗口，窗口中有一个切换卡控件。切换卡控件下有 3 张卡片，代码如下：

```
# === 第 5 章 代码 demo14.py === #
import sys
from PySide6.QtWidgets import (QApplication,QWidget,QTabWidget,QLabel,QHBoxLayout)

class Window(QWidget):
    def __init__(self):
        super().__init__()
        self.setGeometry(200,200,560,220)
        self.setWindowTitle('QTabWidget 类')
        # 创建切换卡控件
        tab = QTabWidget(self)
        # 创建主窗口并将其设置为水平布局
        hbox = QHBoxLayout()
        hbox.addWidget(tab)
        self.setLayout(hbox)
        # 创建页面 1
        page1 = QWidget()
        layout1 = QHBoxLayout()
        label1 = QLabel("这是第 1 个页面。")
        layout1.addWidget(label1)
        page1.setLayout(layout1)
        # 创建页面 2
        page2 = QWidget()
        layout2 = QHBoxLayout()
        label2 = QLabel("这是第 2 个页面。")
        layout2.addWidget(label2)
        page2.setLayout(layout2)
        # 创建页面 3
        page3 = QWidget()
        layout3 = QHBoxLayout()
```

```
        label3 = QLabel("这是第 3 个页面。")
        layout3.addWidget(label3)
        page3.setLayout(layout3)
        #向切换卡控件中添加页面
        tab.addTab(page1,"页面 1")
        tab.addTab(page2,"页面 2")
        tab.addTab(page3,"页面 3")


if __name__ == '__main__':
    app = QApplication(sys.argv)
    win = Window()
    win.show()
    sys.exit(app.exec())
```

运行结果如图 5-18 所示。

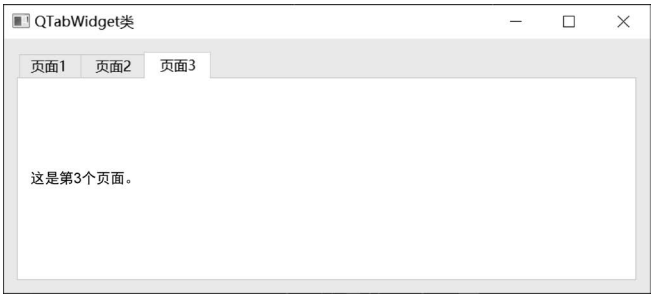

图 5-18　代码 demo14.py 的运行结果

在 PySide6 中,QTabWidget 类的信号见表 5-15。

表 5-15　QTabWidget 类的信号

信　　号	说　　明
currentChanged(index:int)	当前卡片改变时发送信号
tabBarClicked(index:int)	单击卡片的标题时发送信号
tabBarDoubleClicked(index:int)	双击卡片的标题时发送信号
tabCloseRequested(index:int)	单击卡片的关闭标识时发送信号

【实例 5-15】 创建一个窗口,窗口中有一个切换卡控件。切换卡控件下有 3 张卡片。如果单击卡片的关闭标识,则关闭卡片,代码如下:

```
#  === 第 5 章 代码 demo15.py  === #
import sys
from PySide6.QtWidgets import (QApplication,QWidget,QTabWidget,QLabel,QHBoxLayout)

class Window(QWidget):
    def __init__(self):
        super().__init__()
        self.setGeometry(200,200,560,220)
```

```
        self.setWindowTitle('QTabWidget 类')
        #创建切换卡控件
        self.tab = QTabWidget(self)
        self.tab.setTabsClosable(True)
        #创建并设置主窗口为水平布局
        hbox = QHBoxLayout()
        hbox.addWidget(self.tab)
        self.setLayout(hbox)
        #创建页面 1
        page1 = QWidget()
        layout1 = QHBoxLayout()
        label1 = QLabel("这是第 1 个页面。")
        layout1.addWidget(label1)
        page1.setLayout(layout1)
        #创建页面 2
        page2 = QWidget()
        layout2 = QHBoxLayout()
        label2 = QLabel("这是第 2 个页面。")
        layout2.addWidget(label2)
        page2.setLayout(layout2)
        #创建页面 3
        page3 = QWidget()
        layout3 = QHBoxLayout()
        label3 = QLabel("这是第 3 个页面。")
        layout3.addWidget(label3)
        page3.setLayout(layout3)
        #向切换卡控件中添加页面
        self.tab.addTab(page1,"页面 1")
        self.tab.addTab(page2,"页面 2")
        self.tab.addTab(page3,"页面 3")
        #使用信号/槽机制
        self.tab.tabCloseRequested.connect(self.close_tab)

    #自定义槽函数
    def close_tab(self,index):
        self.tab.removeTab(index)


if __name__ == '__main__':
    app = QApplication(sys.argv)
    win = Window()
    win.show()
    sys.exit(app.exec())
```

运行结果如图 5-19 所示。

5.2.5　堆叠控件(QStackedWidget)

在 PySide6 中,可以使用 QStackedWidget 类创建堆叠控件。堆叠控件在功能上与切换卡控件类似,但需要使用自定义的下拉列表或按钮切换页面,并确定当前页面为要显示的页面。QStackedWidget 类的继承关系如图 5-20 所示。

图 5-19　代码 demo15.py 的运行结果

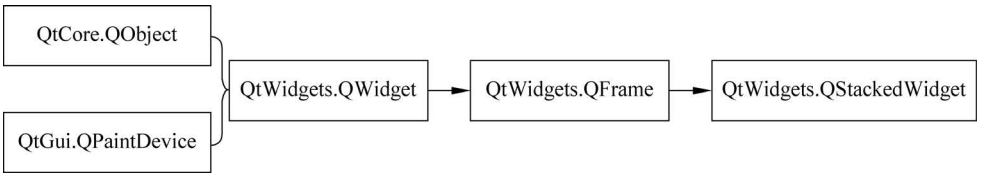

图 5-20　QStackedWidget 类的继承关系

QStackedWidget 类的构造函数如下：

```
QStackedWidget(parent:QWidget = None)
```

其中，parent 表示父窗口或父容器。

在 PySide6 中，QStackedWidget 类的常用方法见表 5-16。

表 5-16　QStackedWidget 类的常用方法

方法及参数类型	说　明	返回值的类型
[slot]setCurrentWidget(QWidget)	将指定的窗口设置为当前窗口	None
[slot]setCurrentIndex(index:int)	将索引为 index 的窗口设置为当前窗口	None
addWidget(QWidget)	在末尾添加窗口，并返回索引	int
insertWidget(index:int,QWidget)	根据索引插入新窗口	int
widget(index:int)	获取索引为 index 的窗口	QWidget
currentIndex()	获取当前窗口的索引	int
currentWidget()	获取当前的窗口	QWidget
indexOf(QWidget)	获取指定窗口的索引	int
removeWidget(QWidget)	移除指定窗口	None
count()	获取窗口的数量	int

在 PySide6 中，QStackedWidget 类的信号见表 5-17。

表 5-17　QStackedWidget 类的信号

信　号	说　明
currentChanged(index:int)	当前窗口改变时发送信号
widgetRemoved(index:int)	当移除窗口时发送信号

【实例 5-16】 创建一个窗口，窗口中有一个堆叠控件，在堆叠控件中使用下拉列表框切换 3 个页面，代码如下：

```
# === 第 5 章 代码 demo16.py === #
import sys
from PySide6. QtWidgets import (QApplication, QWidget, QVBoxLayout, QLabel, QStackedWidget,
QComboBox,QHBoxLayout)

class Window(QWidget):
    def __init__(self):
        super().__init__()
        self.setGeometry(200,200,560,220)
        self.setWindowTitle('QStackedLayout 类')
        #窗口使用垂直布局
        vbox = QVBoxLayout()
        self.setLayout(vbox)
        #创建下拉列表对象
        combo1 = QComboBox(self)
        combo1.addItem("页面 1")
        combo1.addItem("页面 2")
        combo1.addItem("页面 3")
        vbox.addWidget(combo1)
        #创建堆叠控件
        stacked1 = QStackedWidget()
        #创建页面 1
        page1 = QWidget()
        layout1 = QHBoxLayout()
        label1 = QLabel("这是第 1 个页面。")
        layout1.addWidget(label1)
        page1.setLayout(layout1)
        #创建页面 2
        page2 = QWidget()
        layout2 = QHBoxLayout()
        label2 = QLabel("这是第 2 个页面。")
        layout2.addWidget(label2)
        page2.setLayout(layout2)
        #创建页面 3
        page3 = QWidget()
        layout3 = QHBoxLayout()
        label3 = QLabel("这是第 3 个页面。")
        layout3.addWidget(label3)
        page3.setLayout(layout3)
        #向堆叠控件中添加页面
        stacked1.addWidget(page1)
        stacked1.addWidget(page2)
        stacked1.addWidget(page3)
        #向垂直布局中添加堆叠控件
        vbox.addWidget(stacked1)
        #使用信号/槽机制
        combo1.activated.connect(stacked1.setCurrentIndex)
```

```
if __name__ == '__main__':
    app = QApplication(sys.argv)
    win = Window()
    win.show()
    sys.exit(app.exec())
```

运行结果如图 5-21 所示。

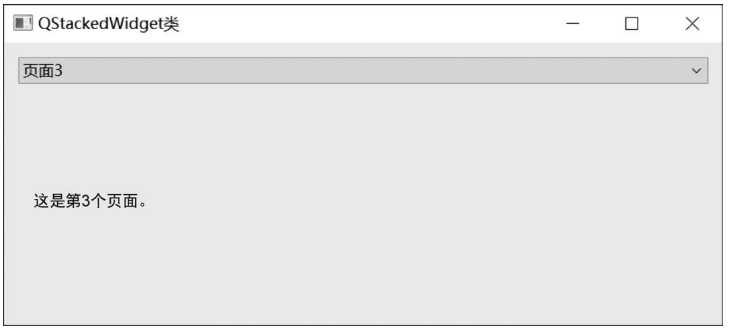

图 5-21　代码 demo16.py 的运行结果

【实例 5-17】 创建一个窗口,窗口中有一个堆叠控件,在堆叠控件中有 3 个页面。使用按钮切换页面,代码如下:

```
# === 第 5 章 代码 demo17.py === #
import sys
from PySide6.QtWidgets import (QApplication, QWidget, QVBoxLayout, QLabel, QStackedWidget,
QPushButton,QHBoxLayout)

class Window(QWidget):
    def __init__(self):
        super().__init__()
        self.setGeometry(200,200,560,220)
        self.setWindowTitle('QStackedWidget 类')
        #窗口使用垂直布局
        vbox = QVBoxLayout()
        self.setLayout(vbox)
        #创建堆叠控件
        self.stacked = QStackedWidget()
        #创建页面 1
        page1 = QWidget()
        layout1 = QHBoxLayout()
        label1 = QLabel("这是第 1 个页面。")
        layout1.addWidget(label1)
        page1.setLayout(layout1)
        #创建页面 2
        page2 = QWidget()
        layout2 = QHBoxLayout()
        label2 = QLabel("这是第 2 个页面。")
        layout2.addWidget(label2)
```

```
            page2.setLayout(layout2)
            #创建页面 3
            page3 = QWidget()
            layout3 = QHBoxLayout()
            label3 = QLabel("这是第 3 个页面。")
            layout3.addWidget(label3)
            page3.setLayout(layout3)
            #向堆叠控件中添加页面
            self.stacked.addWidget(page1)
            self.stacked.addWidget(page2)
            self.stacked.addWidget(page3)
            vbox.addWidget(self.stacked)
            #创建水平布局对象,并添加 3 个按压按钮
            btn_layout = QHBoxLayout()
            btn1 = QPushButton("页面 1")
            btn2 = QPushButton("页面 2")
            btn3 = QPushButton("页面 3")
            btn_layout.addWidget(btn1)
            btn_layout.addWidget(btn2)
            btn_layout.addWidget(btn3)
            vbox.addLayout(btn_layout)
            #使用信号/槽机制
            btn1.clicked.connect(lambda:self.stacked.setCurrentIndex(0))
            btn2.clicked.connect(lambda:self.stacked.setCurrentIndex(1))
            btn3.clicked.connect(lambda:self.stacked.setCurrentIndex(2))


if __name__ == '__main__':
    app = QApplication(sys.argv)
    win = Window()
    win.show()
    sys.exit(app.exec())
```

运行结果如图 5-22 所示。

图 5-22　代码 demo17.py 的运行结果

5.2.6 工具箱控件(QToolBox)

在 PySide6 中,可以使用 QToolBox 类创建工具箱控件。工具箱控件在功能上与切换卡控件类似,可以显示多种页面,但工具箱控件的页面是从上到下依次排列的。工具箱控件的页面标题呈按钮状,如果单击每页的标题,则会在该标题下显示每页窗口。QToolBox 类的继承关系如图 5-23 所示。

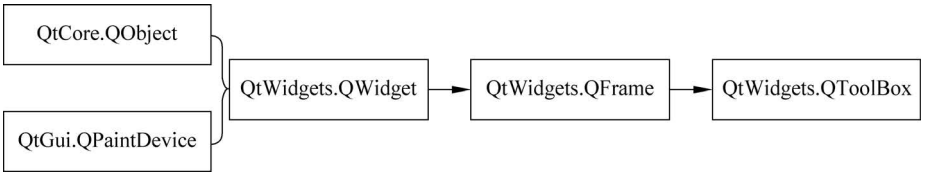

图 5-23 QToolBox 类的继承关系

QToolBox 类的构造函数如下:

```
QToolBox(parent:QWidget = None,f:Qt.WindowFlags = Default(Qt.WindowFlags))
```

其中,parent 表示父窗口或父容器。

在 PySide6 中,QToolBox 类的常用方法见表 5-18。

表 5-18 QToolBox 类的常用方法

方法及参数类型	说 明	返回值的类型
[slot]setCurrentIndex(index:int)	根据索引设置当前项	None
[slot]setCurrentWidget(QWidget)	设置当前窗口	None
addItem(QWidget,text:str)	在末尾添加项,text 表示标题	int
addItem(QWidget,QIcon,text:str)	在末尾添加项,QIcon 表示图标	int
insertItem(index:int,QWidget,text:str)	根据索引插入项	int
insertItem(index:int,QWidget,QIcon,str)	根据索引插入项	int
currentIndex()	获取当前项的索引	int
currentWidget()	获取当前项的窗口	QWidget
widget(index:int)	获取索引为 index 的窗口	int
removeItem(index:int)	根据索引移除项	None
count()	获取项的数量	int
indexOf(QWidget)	获取指定窗口的索引	int
setItemEnabled(index:int,bool)	根据索引设置项是否激活	None
isItemEnabled(index:int)	根据索引获取项是否激活	bool
setItemIcon(index:int,QIcon)	根据索引设置项的图标	None
itemIcon(index:int)	根据索引获取项的图标	bool
setItemText(index:int,str)	根据索引设置项的标题名称	None
itemText(index:int)	根据索引获取项的标题名称	str
setItemToolTip(index:int,str)	根据索引设置项的提示信息	None
itemToolTip(index:int)	根据索引获取项的提示信息	str

在 PySide6 中,QToolBox 类只有一个信号 currentChanged(index:int),表示当前项发

生变化时发送信号。

【实例 5-18】 创建一个窗口，窗口中有一个工具箱控件，在工具箱控件中有 3 个页面，代码如下：

```
# === 第 5 章 代码 demo18.py === #
import sys
from PySide6. QtWidgets import ( QApplication, QWidget, QVBoxLayout, QLabel, QToolBox,
QHBoxLayout)

class Window(QWidget):
    def __init__(self):
        super().__init__()
        self.setGeometry(200,200,560,220)
        self.setWindowTitle('QToolBox 类')
        #窗口使用垂直布局
        vbox = QVBoxLayout()
        self.setLayout(vbox)
        #创建工具箱控件
        tool = QToolBox()
        vbox.addWidget(tool)
        #创建页面 1
        page1 = QWidget()
        layout1 = QHBoxLayout()
        label1 = QLabel("这是第 1 个页面。")
        layout1.addWidget(label1)
        page1.setLayout(layout1)
        #创建页面 2
        page2 = QWidget()
        layout2 = QHBoxLayout()
        label2 = QLabel("这是第 2 个页面。")
        layout2.addWidget(label2)
        page2.setLayout(layout2)
        #创建页面 3
        page3 = QWidget()
        layout3 = QHBoxLayout()
        label3 = QLabel("这是第 3 个页面。")
        layout3.addWidget(label3)
        page3.setLayout(layout3)
        #向工具箱控件中添加页面
        tool.addItem(page1,"页面 1")
        tool.addItem(page2,"页面 2")
        tool.addItem(page3,"页面 3")
        #向垂直布局中添加工具箱控件
        vbox.addWidget(tool)


if __name__ == '__main__':
    app = QApplication(sys.argv)
    win = Window()
    win.show()
    sys.exit(app.exec())
```

运行结果如图 5-24 所示。

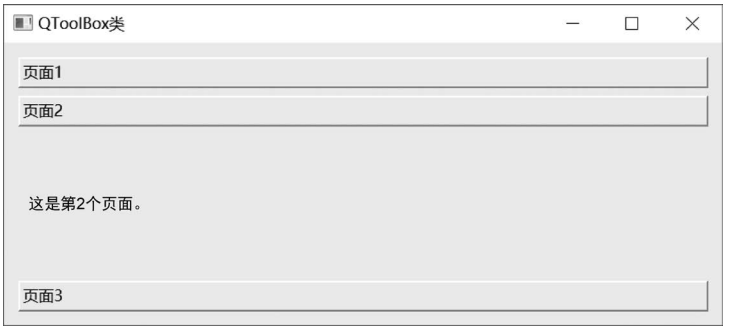

图 5-24　代码 demo18.py 的运行结果

5.2.7　单页面容器控件(QAxWidget)

在 PySide6 中,可以使用 QAxWidget 类创建单页面容器控件。可以使用单页面容器控件访问 ActiveX 控件。QAxWidget 类有一个父类 QAxBase,QAxBase 类提供了 API 初始化和访问 COM 对象的相关功能。QAxWidget 从 QAxBase 继承了大部分与 ActiveX 相关的功能。

ActiveX 控件是一种比较老的技术,只有 IE 浏览器对其提供支持。2022 年 6 月 15 日,微软宣布放弃支持 IE 浏览器,转而支持使用 Chromium 内核的 Edge 浏览器,因此,QAxWidget 类应用得比较少。如果开发浏览器,则推荐支持 Chromium 内核的 QWebEngineView 类。

5.3　分割器控件(QSplitter)

在 PySide 中,可以使用 QSplitter 类创建分割器控件。分割器控件可以将窗口分割为多个不同的部分,不同的部分之间有一条分割线,可以通过拖曳改变分割线的位置。分割器分为水平分割器和竖直分割器。可以在分割器中加入控件,也可以在分割器中加入分割器,形成多级分割,但不能在分割器中加入布局。QSplitter 类的继承关系如图 5-25 所示。

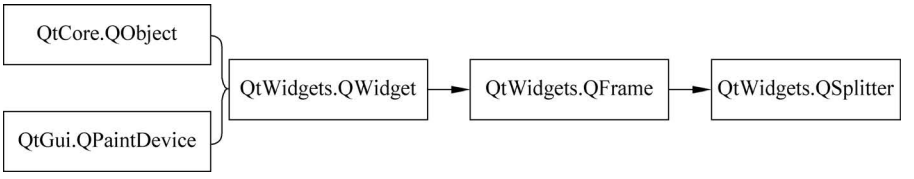

图 5-25　QSplitter 类的继承关系

QSplitter 类的构造函数如下:

```
QSplitter(parent:QWidget = None)
QSplitter(Qt.Qrientation, parent:QWidget = None)
```

其中，parent 表示父窗口或父容器；Qt. Orientation 表示分割方向，其参数值为 Qt. Vertical 或 Qt. Horizontal。

5.3.1　QSplitter 类的方法和信号

在 PySide6 中，QSplitter 类的常用方法见表 5-19。

表 5-19　QSplitter 类的常用方法

方法及参数类型	说　明	返回值类型
addWidget(QWidget)	在末尾添加控件	None
addWidget(index:int,QWidget)	根据索引插入控件	None
widget(index:int)	根据索引获取控件	QWidget
replaceWidget(index:int,QWidget)	根据索引替换控件	None
count()	获取控件的数量	int
indexOf(QWidget)	获取控件的索引	int
setOrientation(Qt. Orientation)	设置分割方向	None
orientation()	获取分割方向	Qt. Orientation
setOpaqueResize(bool)	当拖动分割条时，设置是否为动态的	None
setStretchFactor(index:int,stretch)	当窗口缩放时，设置分割区的缩放系数	None
setHandleWidth(int)	设置分割条的宽度	None
setChildrenCollapsible(bool)	设置内部控件是否可以折叠，默认值为 True	None
setCollapsible(index:int,bool)	根据索引设置控件是否可以折叠	None
setSize(list:Sequence[int])	使用序列（列表、元组）设置内部控件的宽度（水平分割）、高度（竖直分割）	None
size()	获取分割器中控件的宽度（水平分割）列表或高度列表（竖直分割）	List
setRubberBand(position:int)	将橡皮筋设置到指定位置，如果分割线不是动态的，则会看到橡皮筋	None
moveSplitter(pos:int,index:int)	将索引为 index 的分割线移动到 pos 处	None
getRange(index:int)	根据索引获取分割线的调节范围	Tuple
saveState()	保存分割器的状态	QByteArray
restoreState(QByteArray)	恢复保存的状态	bool

在 PySide6 中，QSplitter 类只有一个信号 splitterMoved(pos:int,index:int)，表示当分割线移动时发送信号，信号的参数是分割线的位置和索引。

5.3.2　QSplitter 类的应用实例

【实例 5-19】 创建一个窗口，使用分割器控件将窗口分割为左右两部分。窗口的左右两部分各显示一张图像，代码如下：

```
# === 第 5 章 代码 demo19.py === #
import sys
from PySide6.QtWidgets import (QApplication,QWidget,QLabel,QSplitter,QHBoxLayout)
```

```
from PySide6.QtGui import QPixmap
from PySide6.QtCore import Qt

class Window(QWidget):
    def __init__(self):
        super().__init__()
        self.setGeometry(200,200,560,220)
        self.setWindowTitle('QSplitter 类')
        #创建两个标签控件
        label_1 = QLabel()
        label_2 = QLabel()
        pic1 = QPixmap("D:\\Chapter5\\images\\cat1.png")
        pic2 = QPixmap("D:\\Chapter5\\images\\dog1.jpg")
        pic1 = pic1.scaled(260,220)
        pic2 = pic2.scaled(260,220)
        label_1.setPixmap(pic1)
        label_2.setPixmap(pic2)
        #创建分割器,将窗口分割为左右两部分
        hsplitter = QSplitter(Qt.Horizontal)
        hsplitter.addWidget(label_1)
        hsplitter.addWidget(label_2)
        #创建水平布局对象
        hbox = QHBoxLayout()
        hbox.addWidget(hsplitter)
        self.setLayout(hbox)


if __name__ == '__main__':
    app = QApplication(sys.argv)
    win = Window()
    win.show()
    sys.exit(app.exec())
```

运行结果如图 5-26 所示。

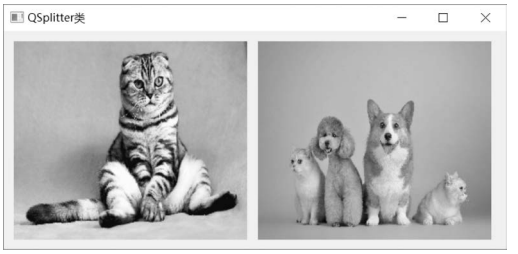

图 5-26　代码 demo19.py 的运行结果

【实例 5-20】 创建一个窗口,使用分割器控件将窗口分割为上下两部分。窗口的上下两部分各显示一张图像,代码如下:

```
# === 第 5 章 代码 demo20.py === #
import sys
from PySide6.QtWidgets import (QApplication,QWidget,QLabel,QSplitter,QVBoxLayout)
```

```
from PySide6.QtGui import QPixmap
from PySide6.QtCore import Qt

class Window(QWidget):
    def __init__(self):
        super().__init__()
        self.setGeometry(200,200,600,450)
        self.setWindowTitle('QSplitter 类')
        #创建两个标签控件
        label_1 = QLabel()
        label_2 = QLabel()
        pic1 = QPixmap("D:\\Chapter5\\images\\cat1.png")
        pic2 = QPixmap("D:\\Chapter5\\images\\dog1.jpg")
        pic1 = pic1.scaled(550,200)
        pic2 = pic2.scaled(550,200)
        label_1.setPixmap(pic1)
        label_2.setPixmap(pic2)
        #创建分割器,将窗口分割为上下两部分
        vsplitter = QSplitter(Qt.Vertical)
        vsplitter.addWidget(label_1)
        vsplitter.addWidget(label_2)
        #创建竖直布局对象
        vbox = QVBoxLayout()
        vbox.addWidget(vsplitter)
        self.setLayout(vbox)


if __name__ == '__main__':
    app = QApplication(sys.argv)
    win = Window()
    win.show()
    sys.exit(app.exec())
```

运行结果如图 5-27 所示。

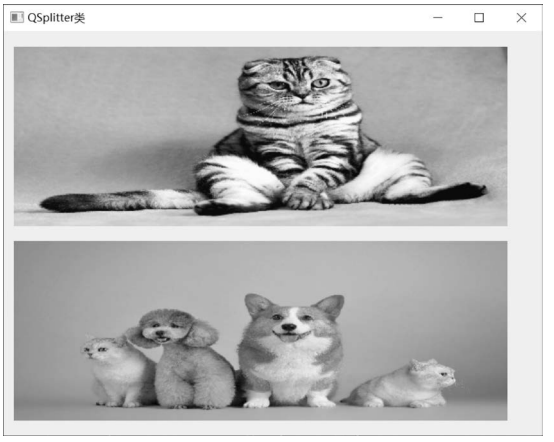

图 5-27　代码 demo20.py 的运行结果

【实例 5-21】 创建一个窗口,使用分割器控件将窗口分割为 3 部分。窗口的各部分显示一张图像,代码如下:

```
# === 第 5 章 代码 demo21.py === #
import sys
from PySide6.QtWidgets import (QApplication,QWidget,QLabel,QSplitter,QVBoxLayout)
from PySide6.QtGui import QPixmap
from PySide6.QtCore import Qt

class Window(QWidget):
    def __init__(self):
        super().__init__()
        self.setGeometry(200,200,800,500)
        self.setWindowTitle('QSplitter 类')
        #创建两个标签控件
        label_1 = QLabel()
        label_2 = QLabel()
        label_3 = QLabel()
        pic1 = QPixmap("D:\\Chapter5\\images\\cat1.png")
        pic2 = QPixmap("D:\\Chapter5\\images\\dog1.jpg")
        pic3 = QPixmap("D:\\Chapter5\\images\\hill.png")
        pic1 = pic1.scaled(220,200)
        pic2 = pic2.scaled(220,220)
        pic3 = pic3.scaled(500,300)
        label_1.setPixmap(pic1)
        label_2.setPixmap(pic2)
        label_3.setPixmap(pic3)
        #创建分割器,将窗口分割为上下两部分
        hsplitter = QSplitter(Qt.Vertical)
        hsplitter.addWidget(label_1)
        hsplitter.addWidget(label_2)
        #创建分割器,将窗口分割为左右两部分
        vsplitter = QSplitter(Qt.Horizontal)
        vsplitter.addWidget(hsplitter)
        vsplitter.addWidget(label_3)
        #创建竖直布局对象
        vbox = QVBoxLayout()
        vbox.addWidget(vsplitter)
        self.setLayout(vbox)


if __name__ == '__main__':
    app = QApplication(sys.argv)
    win = Window()
    win.show()
    sys.exit(app.exec())
```

运行结果如图 5-28 所示。

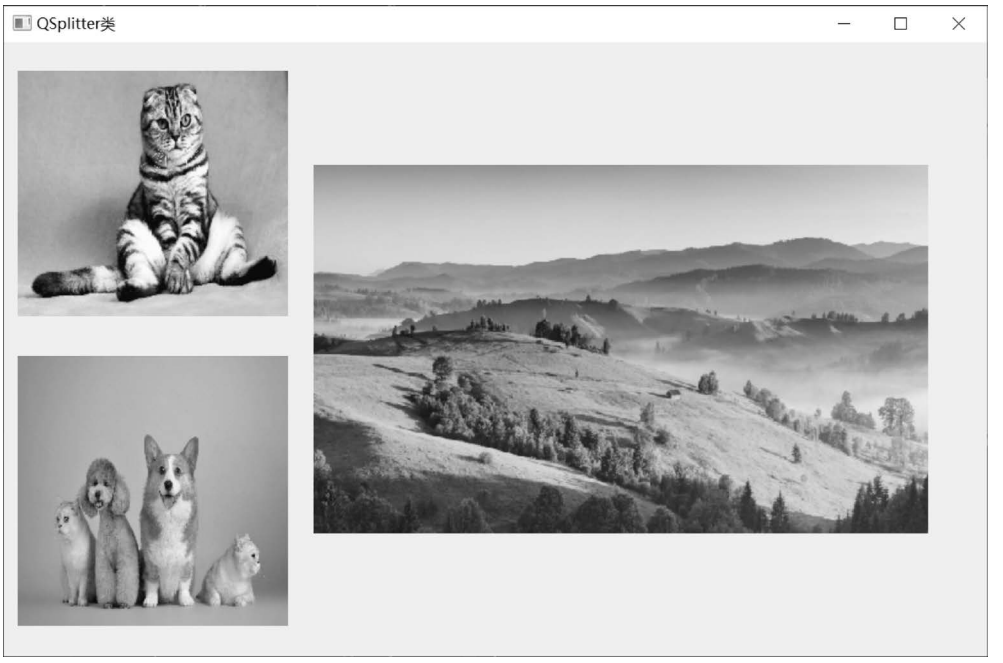

图 5-28　代码 demo21.py 的运行结果

5.4　小结

本章首先介绍了布局管理的基础知识，然后介绍了 PySide6 提供的布局管理的方法，包括水平布局、垂直布局、栅格布局、表单布局、堆叠布局。

其次介绍了 PySide6 中的容器控件，包括分组框控件、框架控件、滚动区控件、切换卡控件、堆叠控件、工具箱控件。

最后介绍了 PySide6 中的分割器控件，并介绍了分割器控件的应用实例。学习了布局管理和容器控件的相关知识后，在后面的章节中将使用本章的知识编写程序。