第3章

界面设计组件

通过前面两章的学习，我们已经对 Qt 有了一个初步的认识，并了解了设计 Qt 应用程序的基本原理和方法。从本章开始，正式进入 Qt 桌面应用程序的开发实践。作为 C++ GUI 组件类库，Qt 提供了大量的界面设计组件，用于数据的显示与交互。Qt 应用程序的开发应该从界面的设计开始。

本章介绍 Qt 的常用界面设计组件，包括基本窗体、常用组件以及布局管理等内容。

3.1 基本窗体

Qt 应用程序属于 GUI 程序，其用户界面一般都包含一个顶层的主窗体和多个子窗体。在 Qt 中，应用程序的主窗体默认有 3 种基本类型，即 QMainWindow、QWidget 和 QDialog。本节介绍 QWidget 类型的基本窗体，QMainWindow 主框架窗体和 QDialog 对话框窗体将在第 4 章和第 5 章详细介绍。

3.1.1 QWidget 类

QWidget 类是 Qt 中所有用户界面组件的基类，它继承于 QObject 类和 QPaintDevice 类。其中，QObject 类是所有支持 Qt 对象模型的基类，QPaintDevice 类是所有可以绘制的对象的基类。所以，Qt 的所有界面组件都具有信号与槽功能，也都能够可视化地显示自己。

QWidget 类属于 Qt 的 Widgets 模块，其部分继承关系如图 3.1 所示。

可以看出，前面提到的 Qt 的 QMainWindow 和 QDialog 类型窗体都是继承自 QWidget 类，还有后面将要介绍的按钮（QPushButton）、文本框（QLineEdit）、标签（QLabel）等组件也都与 QWidget 类或其子类相关联。

作为 Qt 界面组件的基类，QWidget 提供了大量的通用方法，用于实现组件的显示、关闭、参数传递、属性设置以及事件处理等一系列基本操作。表 3.1 列出了 QWidget 类的部分成员函数及功能描述。

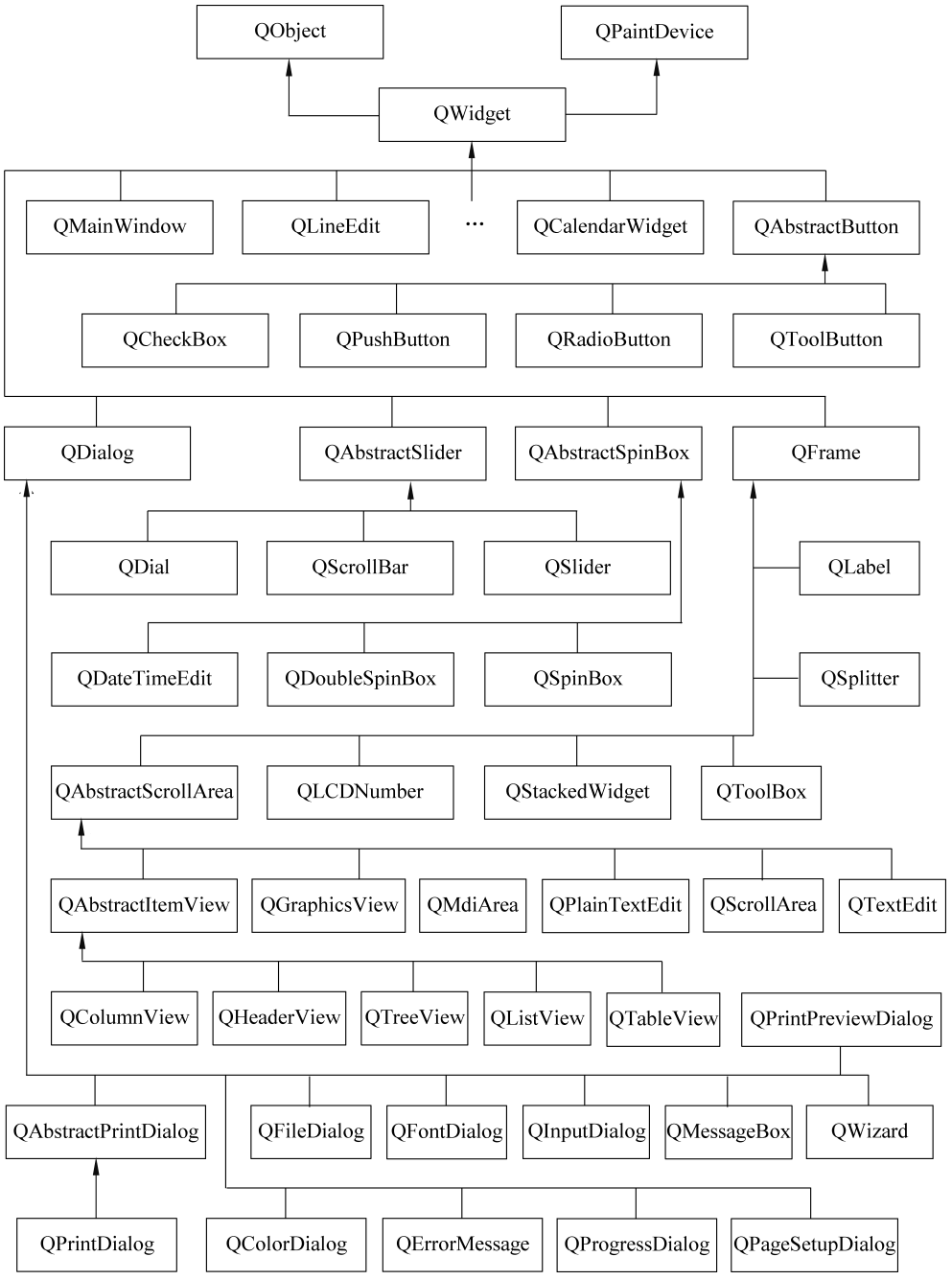

图 3.1　QWidget 类的部分继承关系

表 3.1　QWidget 类的部分成员函数及功能描述

成 员 函 数	功 能 描 述
QWidget()	构造窗体对象
actions()	返回窗体部件中的所有 Action
activateWindow()	激活窗体部件

续表

成员函数	功能描述
addAction()、addActions()	添加 Action 到窗体部件
frameGeometry()	获取窗体尺寸。返回一个 QRect 矩形区域
frameSize()	获取窗体大小。返回一个 QSize 对象
geometry()、rect()、size()	获取窗体客户区尺寸。返回一个 QRect 矩形区域
height()、width()	获取窗体客户区宽度、高度
move()	移动窗体到某个位置
pos()、x()、y()	获取窗体左上角坐标
resize()	设置窗体客户区大小
screen()	获取窗体所在的屏幕 QScreen 对象指针
setGeometry()	设置窗体客户区大小及在屏幕中的位置
setLayout()	设置窗体布局管理器
close()、hide()、show()	槽函数。关闭、隐藏、显示窗体
lower()、raise()	槽函数。将窗体缩小或放大、并放置在堆栈窗体的底部或顶部
repaint()、update()	槽函数。重绘、刷新窗体
setWindowTitle()、setXx()	槽函数。设置窗体标题(等属性)
showNormal()、showXx()	槽函数。设置窗体正常(或全屏、最大化、最小化等)显示方式
customContextMenuRequested()	信号函数。当用户请求了窗体中的上下文菜单时,会发射此信号
windowIconChanged()	信号函数。当窗体的图标发生变化时,会发射此信号
windowTitleChanged()	信号函数。当窗体的标题发生变化时,会发射此信号
find()	静态函数。返回某个 ID 的窗体对象指针
setTabOrder()	静态函数。设置窗体的 Tab 顺序

在 Qt 中,通过直接实例化 QWidget 类创建的基本窗体一般都是作为容器使用的,其中包含了很多不同功能的 QWidget 类的子类对象,如 QLabel、QPushButton 等,因此可以使用 QWidget 类的成员函数完成大部分的用户界面设计工作。实际上,使用 Qt 编写应用程序,就是要熟练掌握 Qt 类库中各种用于界面设计或其他功能的类的使用方法。

下面是使用 QWidget 类成员函数的示例代码。

```
//教材源码 code_3_1_1\main.cpp
QWidget widget;                                    //创建窗体对象
widget.setWindowTitle("QWidget 窗体");             //设置窗体标题
int w = 320;                                       //窗体宽度
int h = 240;                                       //窗体高度
QScreen * screen = widget.screen();                //获取屏幕指针
QRect screenRect = screen->geometry();             //获取屏幕大小
int x = (screenRect.width()-w)/2;                  //计算窗体左上角 x 坐标
int y = (screenRect.height()-h)/2;                 //计算窗体左上角 y 坐标
widget.setGeometry(x,y,w,h);                       //设置窗体显示位置及大小
widget.show();                                     //显示窗体
```

上述示例代码首先创建一个名为 widget 的 QWidget 基本窗体,然后设置窗体标题、大小和显示位置,最后显示窗体。

3.1.2 简单实例

QWidget 类的成员函数非常多,除了表 3.1 中列出的部分非继承的 public 访问权限函数外,还包括它自身的 static public 和 protected 权限的函数,以及从基类 QObject 和 QPaintDevice 继承下来的函数。QWidget 类的 protected 成员函数大部分是用于事件处理的虚函数,关于它们的使用将在第 6 章中进行详细介绍。下面给出一个 QWidget 基本窗体设计的简单实例。

图 3.2 例 3.1 程序初始界面

扫一扫

视频讲解

【例 3.1】 编写一个基于 QWidget 类的 Qt 应用程序。程序运行后,单击窗体中的按钮,更改应用程序主窗体标题,并给出相应的提示信息。程序初始界面如图 3.2 所示。

(1) 打开 Qt Creator 集成开发环境,创建一个基于 QWidget 类的 Qt 应用程序。项目名称为 examp3_1。

(2) 双击项目视图中的 widget.ui 界面文件,打开 Qt 界面设计器,对应用程序主窗体界面进行设计,如图 3.3 所示。

(3) 将主窗体设置为垂直布局,如图 3.4 所示。

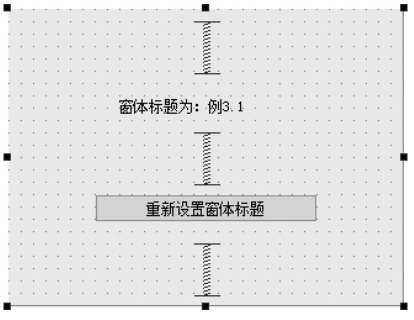

图 3.3 设置主窗体组件

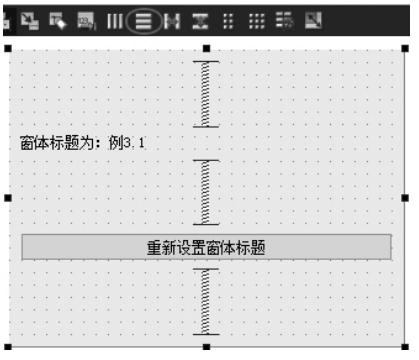

图 3.4 将主窗体设置为垂直布局

(4) 设置主窗体及按钮组件的信号与槽功能。

右击主窗体中的 QPushButton 按钮,在弹出的快捷菜单中选择 Go to slot 菜单命令,为按钮组件添加 clicked()信号的 on_pushButton_clicked()槽函数,并在其中添加如下代码。

```
void Widget::on_pushButton_clicked()
{
    setWindowTitle(tr("窗体标题%1").arg(rand()));
}
```

同理,为主窗体添加 windowTitleChanged()信号的槽函数,并添加如下代码。

```
void Widget::on_Widget_windowTitleChanged(const QString &title)
{
    ui->label->setText(tr("窗体标题更改为: %1").arg(title));
}
```

(5) 在 Widget 类构造函数中添加如下代码，将主窗体中的提示信息设置为红色。

```
ui->label->setStyleSheet(tr("color:red"));
```

(6) 构建并运行程序。单击主窗体中的按钮，更改程序主窗体标题并给出相应的提示信息，如图 3.5 所示。

图 3.5　例 3.1 程序运行结果

本实例程序采用可视化方法设计，大家可以打开 Debug 目录下的 ui_widget.h 文件，查看 QWidget 类的相关成员函数的使用情况，也可以参考第 1 章例 1.3 中的相关代码。

3.2　常用组件

从例 3.1 可以看出，Qt 应用程序的界面设计，主要是程序主窗体中组件的设计，包括组件的布局和数据传递两方面。从图 3.1 可以看出，Qt 提供了大量的窗体组件，它们按照功能不同被分为不同的类型。下面对一些常用的组件作一个简单的介绍。

3.2.1　按钮组件

按钮组件是用户使用鼠标与应用程序交互的图形界面，Qt 的按钮类组件主要有普通按钮、工具栏按钮、单选按钮、复选按钮等，其外观和英文名称如图 3.6 所示。

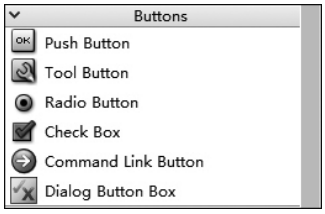

图 3.6　按钮类组件外观及英文名称

1. 普通按钮

普通按钮也称为下压按钮，它所对应的 C++ 类为 QPushButton。从图 3.1 中 QPushButton 类的继承关系可以看出，QPushButton 类是通过多次继承而得到的，因而拥有数量众多的父类成员函数。除了继承下来的成员函数之外，QPushButton 类还定义了一些特有的成员函数，用于完成一些像添加菜单、设置默认按钮等特定操作，如表 3.2 所示。

表 3.2　QPushButton 类部分成员函数及功能描述

成 员 函 数	功 能 描 述	成 员 函 数	功 能 描 述
QPushButton()	构造普通按钮对象	setAutoDefault()	将按钮设置为自动默认
autoDefault()	检测按钮是否为自动默认按钮	setDefault()	将按钮设置为默认按钮
isDefault()	判断按钮是否是默认按钮	setFlat()	设置按钮为扁平
isFlat()	判断按钮是否为扁平	setMenu()	在按钮上添加菜单
menu()	获取按钮上的菜单对象	showMenu()	槽函数。显示按钮上的菜单

QPushButton 类成员函数的使用非常简单，由于篇幅的限制，这里不再一一详述。在第 1 章的例 1.4 中使用了一些 QPushButton 的成员函数，下面再给出一段示例代码。

```
//教材源码 code_3_2_1_1\widget.cpp
QPushButton * pushBtn = new QPushButton(this);     //构造按钮对象
QIcon icon(":/images/save.png");
pushBtn->setIcon(icon);                            //设置图标
pushBtn->setGeometry(50,50,70,40);                 //设置显示位置和大小
pushBtn->setText(tr(" 保存"));                     //设置文本
//pushBtn->setFlat(true);                          //设置按钮为扁平
pushBtn->setDefault(true);                         //设置为默认按钮
//设置按钮为两种状态。单击一次按钮为下压状态,再单击一次按钮恢复
pushBtn->setCheckable(true);
```

上述代码设置了按钮组件的图标,需要先在 code_3_2_1_1 项目中添加资源文件,并加载名为 save.png 的图片。

2. 工具栏按钮

工具栏按钮就是应用程序工具栏上使用的按钮组件,它一般与某个菜单命令相关联。Qt 的工具栏按钮对应的 C++ 类为 QToolButton,它与 QPushButton 一样继承自 QAbstractButton 类,如图 3.1 所示。

由于 QToolButton 按钮一般在工具栏上使用,并且涉及 Qt 中的 Action(动作)概念,我们把它放在第 4 章中与菜单栏、工具栏一起进行详细讲解。

3. 单选按钮

单选按钮一般用于在众多选项中单独选择某一项,它对应的 C++ 类为 QRadioButton。其继承关系如图 3.1 所示。

QRadioButton 类的属性、函数、信号与槽均继承自它的父类,使用时请参考 QObject、QPaintDevice、QWidget 和 QAbstractButton 类的相关技术文档。需要注意的是,单选按钮一般放置在 QGroupBox 容器中,以便对其进行分组。下面是一段单选按钮设计的示例代码。

```
//教材源码 code_3_2_1_3\widget.cpp
QGroupBox * groupBox = new QGroupBox(this);
groupBox->setTitle(tr("性别"));
groupBox->setGeometry(20,20,120,100);
QRadioButton * radioBtn1 = new QRadioButton(groupBox);
radioBtn1->setText(tr("男"));
radioBtn1->setChecked(true);
radioBtn1->setGeometry(QRect(20,30,100,16));
QRadioButton * radioBtn2 = new QRadioButton(groupBox);
radioBtn2->setText(tr("女"));
radioBtn2->setGeometry(QRect(20,60,100,16));
```

图 3.7 单选按钮示例

该示例代码运行结果如图 3.7 所示。

4. 复选按钮

复选按钮用于在众多的选项中同时选择多项,它对应的 C++ 类为 QCheckBox,其继承关系如图 3.1 所示。

QCheckBox 类除了继承的属性、函数和信号之外,还定义了几个特有的成员函数,用于设置或检查按钮状态,如表 3.3 所示。

表 3.3　QCheckBox 类部分成员函数及功能描述

成员函数	功能描述	成员函数	功能描述
QCheckBox()	构造复选按钮对象	setCheckState()	设置复选按钮状态
checkState()	获取复选按钮状态	setTristate()	设置复选按钮为三态的
isTristate()	判断复选按钮是否为三态的		

除了上述特有成员函数之外，QCheckBox 类还定义了一个名为 stateChanged()的信号，当复选按钮状态发生变化时，会发射该信号。下面是一段复选按钮设计的示例代码。

```
//教材源码 code_3_2_1_4\widget.cpp
QGroupBox * groupBox = new QGroupBox(this);
groupBox->setTitle(tr("爱好"));
groupBox->setGeometry(20,20,120,120);
QCheckBox * checkBox1 = new QCheckBox(groupBox);
checkBox1->setText(tr("读书"));
checkBox1->setGeometry(QRect(20,30,100,16));
QCheckBox * checkBox2 = new QCheckBox(groupBox);
checkBox2->setText(tr("跑步"));
checkBox2->setGeometry(QRect(20,60,100,16));
QCheckBox * checkBox3 = new QCheckBox(groupBox);
checkBox3->setText(tr("游戏"));
checkBox3->setTristate(true);
checkBox3->setGeometry(QRect(20,90,100,16));
```

该示例代码运行结果如图 3.8 所示。

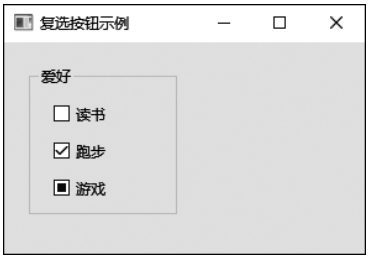

图 3.8　复选按钮示例

3.2.2　输入组件

输入组件用于应用程序与用户的交互，主要包括组合框、字体选择框、行编辑器、文本编辑器、数字选择框(自旋盒)、时间编辑器、日期编辑器、拨号器、滚动条和滑动条等，其外观和英文名称如图 3.9 所示。

1. 组合框

组合框又称为下拉列表框，用于通过选择输入数据。它对应的 C++ 类为 QComboBox，其继承关系如图 3.10 所示。

QComboBox 类是 QWidget 类的直接子类，它的部分成员函数及功能描述如表 3.4 所示。

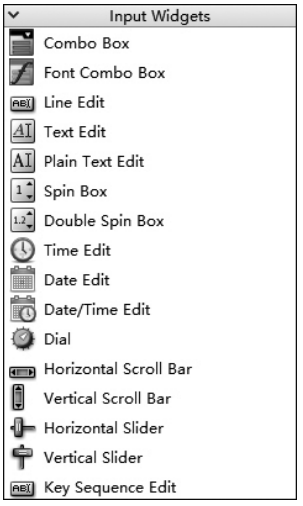

图 3.9　输入组件外观及英文名称

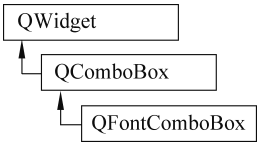

图 3.10　QComboBox 类继承关系

表 3.4　QComboBox 类的部分成员函数及功能描述

成 员 函 数	功 能 描 述
QComboBox()	构造组合框对象
addItem()、addItems()、insertItem()、insertItems()、insertSeparator()	向组合框中添加数据项或分隔线
completer()、setCompleter()	获取或设置返回组合框的自动补全器 QCompleter
count()	返回组合框数据项数量
currentData()、currentIndex()、currentText()	获取组合框中当前选择数据项、数据项索引或数据项文本
duplicatesEnabled()、setDuplicatesEnabled()	处理 duplicatesEnabled 属性,该属性表示组合框中数据项是否可以重复
findData()、findText()	按条件查找组合框中的数据项
isEditable()、setEditable()	处理数据项是否可以编辑
itemData()、itemDelegate()、itemIcon()、itemText()	获取数据项信息,包括数据、代理、图标或文本
maxCount()、setMaxCount()	处理 maxCount 属性,该属性表示组合框允许的数据项最大项数
maxVisibleItems()、setMaxVisibleItems()	处理 maxVisibleItems 属性,该属性表示组合框允许的数据项最大可视项数。默认为 10 项
model()、setModel()	获取或设置组合框使用的数据模型
removeItem()	移除组合框中的数据项
setFrame()、setXxxx()	设置组合框边框属性(Frame)或其他属性(Xxxx)
clear()	槽函数。清空组合框,移除所有数据项
clearEditText()、setEditText()	槽函数。清除或设置组合框文本编辑中的内容
setCurrentIndex()、setCurrentText()	槽函数。设置当前选择项
activated()	信号函数。当用户选择数据项时发射此信号
currentIndexChanged()	信号函数。当 currentIndex 属性变化时发射此信号

续表

成 员 函 数	功 能 描 述
currentTextChanged()	信号函数。当 currentText 属性变化时发射此信号
editTextChanged()	信号函数。当编辑器文本发生变化时发射此信号
highlighted()	信号函数。当某数据项高亮显示时发射此信号
textActivated()	信号函数。当用户选择某项文本时发射此信号
textHighlighted()	信号函数。当数据项文本高亮显示时发射此信号

当一个 QComboBox 组合框中的选择项发生变化时，它会发射以下两个信号。

```
void currentIndexChanged(int index)
void currentTextChanged(const QString &text)
```

这两个信号只是传递的参数不同，一个传递的是当前项的索引号，另一个传递的是当前项的文本。下面是一段组合框设计的示例代码。

```
//教材源码 code_3_2_2_1\widget.cpp
QComboBox * comboBox = new QComboBox(this);
comboBox->setGeometry(10,10,80,22);
comboBox->setMaxVisibleItems(5);
QStringList list;
list<<"湖北"<<"湖南"<<"广东"<<"广西"<<"河北"<<"河南";
comboBox->addItems(list);
QMap<QString,int> map;
map.insert(tr("武汉"),27);
map.insert(tr("北京"),10);
map.insert(tr("上海"),21);
foreach(const QString &str,map.keys()){
    comboBox->addItem(str,map.value(str));
}
QLabel * label = new QLabel(this);
label->setGeometry(120,10,100,22);
connect(comboBox,&QComboBox::currentTextChanged,label, &QLabel::setText);
```

该示例代码运行结果如图 3.11 所示。

图 3.11　组合框示例

2. 行编辑器

行编辑器组件是一个单行的文本编辑器，允许用户输入和编辑单行的纯文本内容，而且提供了一系列有用的功能，包括撤销、恢复、剪切和拖放等操作。行编辑器组件对应的 C++ 类为 QLineEdit，该类是 QWidget 的直接子类，如图 3.1 所示。

QLineEdit 类的部分成员函数及功能描述如表 3.5 所示。

表 3.5　QLineEdit 类的部分成员函数及功能描述

成 员 函 数	功 能 描 述
QLineEdit()	构造行编辑器对象
alignment()、setAlignment()	处理行编辑对齐方式
backspace()、del()	删除行编辑器中的字符

续表

成 员 函 数	功 能 描 述
completer()、setCompleter()	处理自动补全器(Completer)属性
cursorBackward()、cursorForward()、cursorMoveStyle()、cursorPosition()、cursorPositionAt()、cursorWordBackward()、cursorWordForward()、end()、home()	处理行编辑器中的光标
deselect()	取消行编辑中任何选定的文本
displayText()	根据设定显示不同的掩码字符
dragEnabled()、setDragEnabled()	处理行编辑器的选定文本是否可以拖动
echoMode()、setEchoMode()	处理行编辑中文本的显示模式
hasSelectedText()	判断是否有文本被选择
isReadOnly()、setReadOnly()	处理行编辑器的只读属性
maxLength()、setMaxLength()	处理行编辑中本文的最大长度属性
placeholderText()、setPlaceholderText()	处理行编辑中本文的点位符文本属性
clear()	槽函数。清空行编辑器
copy()、cut()、paste()	槽函数。复制、剪切和粘贴操作
redo()、undo()	槽函数。重做、撤销操作
selectAll()	槽函数。全选操作
setText()	槽函数。设置行编辑器文本
cursorPositionChanged()	信号函数。当光标位置属性改变时发射此信号
editingFinished()	信号函数。当编辑完成时发射此信号
inputRejected()	信号函数。用户按下不可接收键时发射此信号
returnPressed()	信号函数。按 Enter 键时发射此信号
selectionChanged()	信号函数。当选择发生变化时发射此信号
textChanged()	信号函数。当文本发生变化时发射此信号
textEdited()	信号函数。编辑文本时发射此信号

除了剪切、复制等常规功能外,Qt 中的行编辑器还具有一些特殊的功能,如显示模式、插入掩码、输入验证和自动补全等。

扫一扫

视频讲解

图 3.12 行编辑器示例

【例 3.2】 行编辑器 QLineEdit 简单示例。程序运行结果如图 3.12 所示。

(1) 打开 Qt Creator 集成开发环境,创建一个基于 QWidget 类的 Qt 应用程序,项目名称为 examp3_2。在项目创建过程中,取消自动生成界面文件选项。

(2) 双击项目视图中的 widget.h 头文件,为 Widget 类添加 4 个 QLineEdit 类型的行编辑器组件对象指针 lineEdit1～lineEdit4 和 4 个 QLabel 类型的标签组件对象指针 label1～label4;为 Widget 类添加一个

名为 labelText()的槽函数。代码如下。

```
…
#include <QWidget>
#include <QLabel>
#include <QLineEdit>
…
public slots:
    void labelText();
private:
    QLineEdit * lineEdit1, * lineEdit2, * lineEdit3, * lineEdit4;
    QLabel * label1, * label2, * label3, * label4;
…
```

(3) 双击打开项目的 widget.cpp 文件,在 Widget 类的构造函数中添加如下代码。

```
Widget::Widget(QWidget * parent): QWidget(parent)
{
    resize(320, 240);
    setWindowTitle(tr("行编辑器示例"));
    lineEdit1 = new QLineEdit(this);
    lineEdit1->setGeometry(10,10,100,20);
    label1 = new QLabel(this);
    label1->setGeometry(120,10,100,20);
    connect(lineEdit1,&QLineEdit::editingFinished,this,&Widget::labelText);
    lineEdit2 = new QLineEdit(this);
    lineEdit2->setEchoMode(QLineEdit::Password);
    lineEdit2->setGeometry(10,40,100,20);
    label2 = new QLabel(this);
    label2->setGeometry(120,40,100,20);
    connect(lineEdit2,&QLineEdit::editingFinished,this,&Widget::labelText);
    lineEdit3 = new QLineEdit(this);
    QValidator * validator = new QIntValidator(1,100,this);  //新建验证器
    lineEdit3->setValidator(validator);   //该行编辑器只能接收 1~100 的整型数据
    lineEdit3->setGeometry(10,70,100,20);
    label3 = new QLabel(this);
    label3->setGeometry(120,70,100,20);
    connect(lineEdit3,&QLineEdit::editingFinished,this,&Widget::labelText);
    lineEdit4 = new QLineEdit(this);
    QStringList list;
    list<<tr("Qt")<<tr("Qt Creator")<<tr("Qt 程序设计");
    QCompleter * completer = new QCompleter(list,this);      //构建补全功能对象
    lineEdit4->setCompleter(completer);   //该行编辑器具有自动补全功能
    lineEdit4->setGeometry(10,100,100,20);
    label4 = new QLabel(this);
    label4->setGeometry(120,100,100,20);
    connect(lineEdit4,&QLineEdit::editingFinished,this,&Widget::labelText);
}
```

(4) 在 widget.cpp 文件中添加 labelTex()槽函数的实现代码。

```
void Widget::labelText(){
    label1->setText(lineEdit1->text());
    label2->setText(lineEdit2->text());
```

```
    label3->setText(lineEdit3->text());
    label4->setText(lineEdit4->text());
}
```

(5) 构建并运行程序。程序运行结果如图 3.12 所示。

本示例程序展示了 4 种类型的行编辑器组件,在图 3.12 的主窗体中,从上至下依次为正常显示模式的行编辑器、密码显示模式的行编辑器、具有输入验证功能的行编辑器和具有自动补全功能的行编辑器。

3. 数字选择框

数字选择框又称为自旋盒,主要用于数字的输入和显示。它对应的 C++ 类有 QSpinBox、QDoubleSpinBox 和 QDateTimeEdit 等,其继承关系如图 3.1 所示。

QSpinBox 组件用于整数的显示和输入,一般显示十进制数,也可以显示二进制、十六进制的整数,而且可以在显示框中增加前缀或后缀。

QDoubleSpinBox 组件用于浮点数的显示和输入,可以设置显示小数位数,也可以设置显示的前缀和后缀。

QDateTimeEdit 组件用于编辑和显示日期时间,它对应 Qt 的 QDateTime 日期时间数据类型,如 2022-03-15 16:10:11。

另外,QDateTimeEdit 类还有两个直接子类,即 QDateEdit 类和 QTimeEdit 类。其中,QDateEdit 组件用于编辑和显示日期,它对应 Qt 的 QDate 日期数据类型,仅表示日期,如 2022-03-15;QTimeEdit 组件用于编辑和显示时间,它对应 Qt 的 QTime 时间数据类型,仅表示时间,如 16:10:11。

上述组件的使用非常简单,只需要调用成员函数对其属性进行设置,并获取当前的数字即可。这些组件类的属性、函数、信号与槽等详情请参考 Qt 的帮助文档。图 3.13 给出了这几种组件的外观示例。

4. 移动型组件

移动型组件主要用于输入数字,改变组件中的可移动部件的位置从而实现数字的输入。它对应的 C++ 类有 QDial、QScrollBar 和 QSlider 等,其继承关系如图 3.1 所示。

QDial 是一个表盘式数值输入组件,通过转动表针获得输入值,其外观如图 3.14 所示。

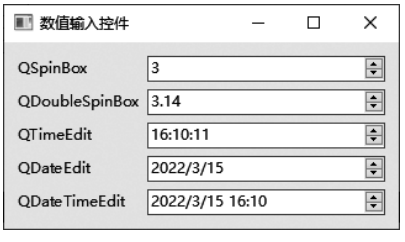

图 3.13 数字选择框组件示例

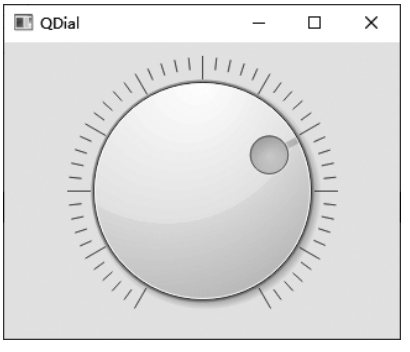

图 3.14 QDial 组件

QScrollBar 是一个滚动组件,可用于卷滚区域,其外观如图 3.15 所示。

QSlider 是一个滑动条组件,通过拖动滑块输入数值,其外观如图 3.16 所示。

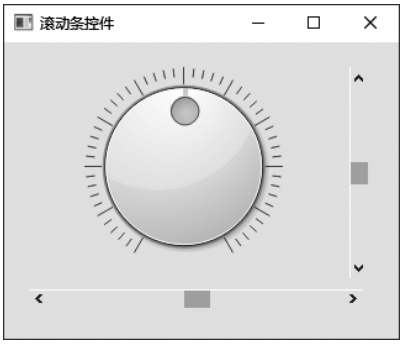

图 3.15　QScrollBar 组件

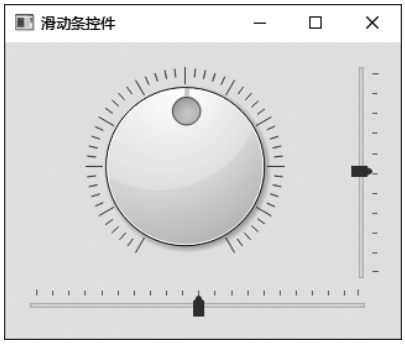

图 3.16　QSlider 组件

3.2.3　显示组件

显示组件用于应用程序的信息展示，主要包括标签、文本浏览器、图形视图、日历、液晶数字、进度条、水平线、垂直线、开放式图形库工具和嵌入 QML 工具等，其外观和英文名称如图 3.17 所示。

1. 标签

标签组件用于显示简单的文本或位图。它对应的 C++ 类为 QLabel，其继承关系如图 3.1 所示。QLabel 标签组件的使用非常简单，请大家参考前面实例中的相关代码。

2. 日历

日历组件用于选择日期数据，它对应的 C++ 类为 QCalendarWidget，该类是 QWidget 类的直接子类，没有被其他类继承。日历组件外观效果如图 3.18 所示。

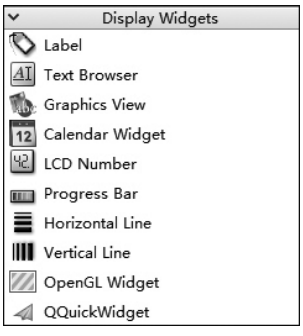

图 3.17　显示组件外观和英文名称

图 3.18　日历组件

3. LCD 数字显示框

LCD 数字显示框组件可以让数码显示为与液晶数字一样的效果。它对应的 C++ 类为 QLCDNumber，其继承关系如图 3.1 所示。外观效果如图 3.19 所示。

4. 进度条

进度条组件用于显示一件比较耗时的事情的完成情况。它对应的 C++ 类为 QProgressBar，该类是 QWidget 类的直接子类，没有被其他类继承。外观效果如图 3.20 所示。

图 3.19　LCD 数字显示框组件

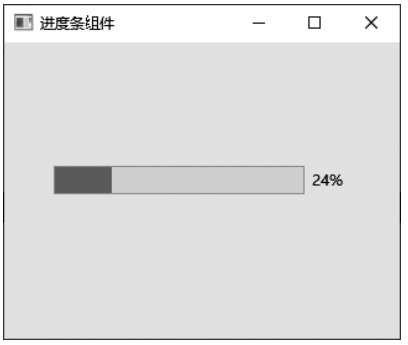

图 3.20　进度条组件

3.2.4　浏览组件

浏览组件用于应用程序的信息展示。它分为两种类型,一类是基于 Qt 的模型/视图的组件,包括列表视图、树视图、表视图和列视图等;另一类是基于数据项的组件,包括列表部件、树状部件和表部件。其实,第 2 类浏览组件就是第 1 类中相应组件的便捷组件。浏览组件的外观和英文名称如图 3.21 所示。

浏览组件所对应的 C++ 类和继承关系如图 3.1 和图 3.22 所示。

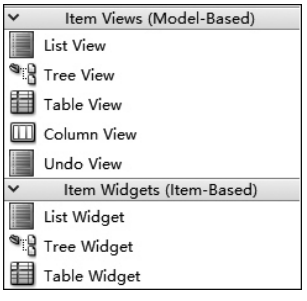

图 3.21　浏览组件的外观和英文名称

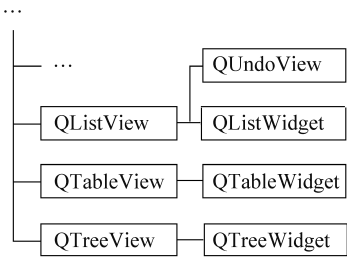

图 3.22　浏览组件类继承关系

1. 列表组件

列表组件用于以列表的形式显示信息,它对应的 C++ 类为 QListView、QListWidget 和 QUndoView,继承关系如图 3.22 所示。

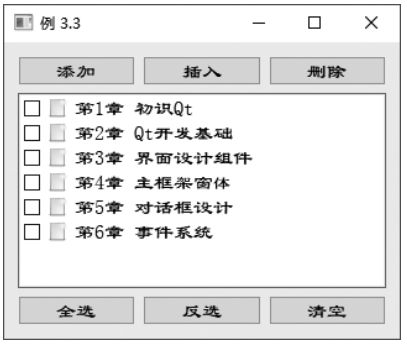

图 3.23　例 3.3 程序运行结果

扫一扫

视频讲解

QListView 和 QUndoView 类一般用于 Qt 的模型/视图结构中,将在第 8 章进行介绍。下面给出一个 QListWidget 的简单应用实例。

【例 3.3】 QListWidget 列表组件的使用。程序运行结果如图 3.23 所示。

(1) 打开 Qt Creator 集成开发环境,创建一个基于 QWidget 类的 Qt 应用程序,项目名称为 examp3_3。

(2) 双击项目视图中的 widget.ui 界面文件,打开 Qt 界面设计器,对应用程序主窗体界面进行设计,如图 3.23 所示。其中,“添加”“插入”“删除”“全选”

“反选”“清空”按钮为 QPushButton 类型的按钮组件，对象名称分别为 addButton、insertButton、deleteButton、selectAllButton、selectInvsButton 和 clearButton；主窗体中间区域是一个对象名称为 listWidget 的 QListWidget 列表框组件。

(3) 在界面设计器中分别选择 6 个按钮组件，为它们添加 clicked 信号槽函数，槽函数名称分别为 on_addButton_clicked()、on_insertButton_clicked()、on_deleteButton_clicked()、on_selectAllButton_clicked()、on_selectInvsButton_clicked()和 on_clearButton_clicked()；选择中间区域的 listWidget 列表框组件，为其添加 itemClicked 信号槽函数，槽函数名称为 on_listWidget_itemClicked(QListWidgetItem * item)。

(4) 为 Widget 类添加一个私有的 initListWidget()成员函数，用于初始化主窗体中的列表框组件。代码如下。

```
void Widget::initListWidget()
{
    QListWidgetItem * aItem;
    QIcon aIcon;
    aIcon.addFile(":/images/new.png");
    QStringList items;
    items << "第 1 章 初识 Qt" << "第 2 章 Qt 开发基础" << "第 3 章 界面设计组件"
          << "第 4 章 主框架窗体" << "第 5 章 对话框设计" << "第 6 章 事件系统";
    for(int i=0; i<items.length(); i++) {
        aItem = new QListWidgetItem();
        aItem->setText(items[i]);
        aItem->setIcon(aIcon);
        aItem->setCheckState(Qt::Unchecked);
        ui->listWidget->addItem(aItem);
    }
}
```

在 Widget 类的构造函数中添加代码，调用该初始化函数。

(5) 为步骤(3)中添加的槽函数编写代码，实现相应的操作功能。代码如下。

```
void Widget::on_addButton_clicked()
{
    QListWidgetItem * aItem = new QListWidgetItem();
    QIcon aIcon;
    aIcon.addFile(":/images/new.png");
    aItem->setText(tr("在列表中增加一项，请双击编辑新项名称"));
    aItem->setIcon(aIcon);
    aItem->setCheckState(Qt::Unchecked);
    aItem->setFlags(Qt::ItemIsSelectable | Qt::ItemIsEditable
|Qt::ItemIsUserCheckable |Qt::ItemIsEnabled);
    ui->listWidget->addItem(aItem);
}
void Widget::on_insertButton_clicked()
{
    QListWidgetItem * aItem = new QListWidgetItem();
    QIcon aIcon;
    aIcon.addFile(":/images/new.png");
    aItem->setText(tr("在这里插入一项，请双击编辑新项名称"));
    aItem->setIcon(aIcon);
    aItem->setCheckState(Qt::Unchecked);
    aItem->setFlags(Qt::ItemIsSelectable | Qt::ItemIsEditable
```

```
|Qt::ItemIsUserCheckable |Qt::ItemIsEnabled);
    ui->listWidget->insertItem(ui->listWidget->currentRow()+1,aItem);
}
void Widget::on_deleteButton_clicked()
{
    int row = ui->listWidget->currentRow();
    QListWidgetItem * aItem = ui->listWidget->takeItem(row);
    delete aItem;
}
void Widget::on_selectAllButton_clicked()
{
    int nums = ui->listWidget->count();
    for(int i = 0; i < nums; ++i) {
        QListWidgetItem * aItem = ui->listWidget->item(i);
        aItem->setCheckState(Qt::Checked);    }
}
void Widget::on_selectInvsButton_clicked()
{
    int nums = ui->listWidget->count();
    for(int i = 0; i < nums; ++i) {
        QListWidgetItem * aItem = ui->listWidget->item(i);
        if(aItem->checkState() != Qt::Checked)
            aItem->setCheckState(Qt::Checked);
        else
            aItem->setCheckState(Qt::Unchecked);
    }
}
void Widget::on_clearButton_clicked()
{
    ui->listWidget->clear();
}
void Widget::on_listWidget_itemClicked(QListWidgetItem * item)
{
    int nums = ui->listWidget->count();
    for(int i = 0; i < nums; ++i) {
        QListWidgetItem * aItem = ui->listWidget->item(i);
        aItem->setCheckState(Qt::Unchecked);
    }
    item->setCheckState(Qt::Checked);
}
```

(6) 构建并运行程序。单击主窗体中的“添加”按钮,会在列表框的末尾增加一个新项,如图 3.24 所示;双击新增加的子项,可以编辑其名称,如图 3.25 所示。

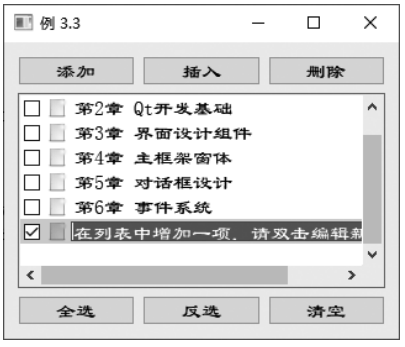

图 3.24 增加新项

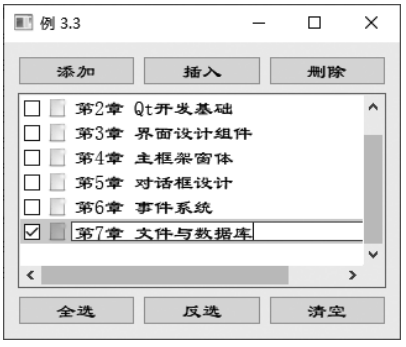

图 3.25 编辑子项名称

单击主窗体中的“插入”按钮，会在当前选择的子项后面添加一个新的子项，其操作与上述演示的“添加”功能相似。主窗体中的其他操作按钮的运行结果，请大家自行测试。

2. 树状组件

树状组件用于表现具有树状层次关系的数据，该组件的每项均可添加位图或文字，可响应单击、双击、选项改变、树状显示扩展、收缩等信号。它所对应的 C++ 类有 QTreeView 和 QTreeWidget 两个，其中 QTreeView 类一般在 Qt 的模型/视图结构中使用。

下面给出一个 QTreeWidget 的简单应用实例。

图 3.26　例 3.4 程序运行结果

扫一扫

视频讲解

【例 3.4】 QTreeWidget 树状组件的使用。程序运行结果如图 3.26 所示。

(1) 打开 Qt Creator 集成开发环境，创建一个基于 QWidget 类的 Qt 应用程序，项目名称为 examp3_4。

(2) 双击项目视图中的 widget.ui 界面文件，打开 Qt 界面设计器，对应用程序主窗体界面进行设计，如图 3.26 所示。其中，“添加章”“添加节”“删除”按钮为 QPushButton 类型的按钮组件，对象名称分别为 add1Button、add2Button 和 deleteButton；主窗体中间区域是一个对象名称为 treeWidget 的 QTreeWidget 树状组件。

(3) 在界面设计器中分别选择 3 个按钮组件，为它们添加 clicked 信号槽函数，槽函数名称分别为 on_add1Button_clicked()、on_add2Button_clicked() 和 on_deleteButton_clicked()。

(4) 为类 Widget 添加一个私有的 initTreeWidget() 成员函数，用于初始化主窗体中的树状组件。代码如下。

```
void Widget::initTreeWidget()
{
    ui->treeWidget->setColumnCount(1);
    QStringList headers;
    headers << tr("教材目录");
    ui->treeWidget->setHeaderLabels(headers);
    icon[0].addFile(":/images/new.png");
    icon[1].addFile(":/images/pencil.png");
    QStringList item1;
    item1 << "第 1 章 初识 Qt" << "第 2 章 Qt 开发基础" << "第 3 章 界面设计组件";
    QStringList item2[3];
    item2[0]<<"1.1 Qt 简介"<<"1.2 开发环境搭建";
    item2[1]<<"2.1"<<"2.2";
    item2[2]<<"3.1 基本窗体"<<"3.2 常用组件";
    QList<QTreeWidgetItem * > list1,list2;
    for(int i = 0; i < item1.count(); ++i) {
        list1.append(new QTreeWidgetItem(ui->treeWidget,
    QStringList(item1.at(i))));
        list1[i]->setIcon(0,icon[0]);
        for(int j = 0; j < item2[i].count(); ++j) {
            list2.append(new QTreeWidgetItem(list1[i],
```

```
    QStringList(item2[i].at(j))));
            list2[j]->setIcon(0,icon[1]);
        }
        list2.clear();
    }
}
```

在 Widget 类的构造函数中添加代码,调用该初始化函数。

(5) 为步骤(3)中添加的槽函数编写代码,实现相应的操作功能。代码如下。

```
void Widget::on_add1Button_clicked()
{
    QTreeWidgetItem * newItem;
    newItem = new QTreeWidgetItem(ui->treeWidget,QStringList(QString("第%1
章").arg(ui->treeWidget->topLevelItemCount()+1)));
    newItem->setIcon(0,icon[0]);
    ui->treeWidget->addTopLevelItem(newItem);
}
void Widget::on_add2Button_clicked()
{
    QTreeWidgetItem * newItem, * currentItem;
    currentItem = ui->treeWidget->currentItem();
    if(!currentItem){
        QMessageBox::information(this,tr("温馨提示"),tr("请选择教材的章目录!"));
        return;
    }
    QModelIndex modeIndex = ui->treeWidget->currentIndex();
    int row = modeIndex.row();
    int childCount = currentItem->childCount();
    if(currentItem && !currentItem->parent()){
        newItem = new QTreeWidgetItem(currentItem,QStringList(QString("%1.%2").
        arg(row+1).arg(childCount+1)));
        newItem->setIcon(0,icon[1]);
        currentItem->addChild(newItem);
    }else{
        QMessageBox::information(this,tr("温馨提示"),tr("请选择教材的章目录!"));
    }
}
void Widget::on_deleteButton_clicked()
{
    QTreeWidgetItem * currentItem, * parent;
    currentItem = ui->treeWidget->currentItem();
    parent = currentItem->parent();
    int index;
    if(parent){
        index = parent->indexOfChild(currentItem);
        delete parent->takeChild(index);
    }else{
        index = ui->treeWidget->indexOfTopLevelItem(currentItem);
        delete ui->treeWidget->takeTopLevelItem(index);
    }
}
```

(6) 构建并运行程序。单击主窗体中的“添加章”按钮,会在树状显示框的末尾增加一个新的“章”项;选择“章”项,单击“添加节”按钮,可以为该章添加“节”子项,如图 3.27 所示。

3. 表格组件

表格组件用于以标准表格的形式显示信息。它所对应的 C++ 类有 QTableView 和 QTableWidget 两个，其中 QTableView 类一般在 Qt 的模型/视图结构中使用。

下面给出一个 QTableWidget 的简单应用实例。

扫一扫 视频讲解

【例 3.5】 QTableWidget 表格组件的使用。程序运行结果如图 3.28 所示。

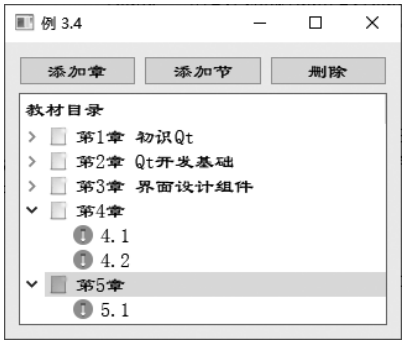

图 3.27　例 3.4 程序测试结果

图 3.28　例 3.5 程序运行结果

(1) 打开 Qt Creator 集成开发环境，创建一个基于 QWidget 类的 Qt 应用程序，项目名称为 examp3_5。

(2) 双击项目视图中的 widget.ui 界面文件，打开 Qt 界面设计器，对应用程序主窗体界面进行设计，如图 3.28 所示。其中，“添加行”“插入行”“删除行”按钮为 QPushButton 类型的按钮组件，对象名称分别为 addRowButton、insertRowButton 和 deleteRowButton；主窗体中间区域是一个对象名称为 tableWidget 的 QTableWidget 表格组件。

(3) 在界面设计器中分别选择 3 个按钮组件，为它们添加 clicked 信号槽函数，槽函数名称分别为 on_addRowButton_clicked()、on_insertRowButton_clicked() 和 on_deleteRowButton_clicked()。

(4) 为 Widget 类添加一个私有的 initTableWidget()成员函数，用于初始化主窗体中的表格组件。代码如下。

```
void Widget::initTableWidget()
{
    QStringList headerText;
    headerText<<"教材名称"<<"作者"<<"出版社";
    QStringList data[4];
    data[0]<<"面向对象程序设计"<<"马石安 魏文平"<<"清华大学出版社";
    data[1]<<"Visual C++程序设计"<<"马石安 魏文平"<<"清华大学出版社";
    data[2]<<"PHP Web 项目开发"<<"马石安 魏文平"<<"清华大学出版社";
    data[3]<<"Qt 6.2/C++程序设计"<<"马石安 魏文平"<<"清华大学出版社";
    ui->tableWidget->setColumnCount(headerText.count());
    ui->tableWidget->setHorizontalHeaderLabels(headerText);
    QTableWidgetItem * item;
    for(int i=0; i<=data->count(); i++){
        for(int j=0; j<headerText.count(); j++) {
            item = new QTableWidgetItem(data[i][j]);
            ui->tableWidget->setItem(i,j,item);
```

```
        }
    }
}
```

(5) 为步骤(3)中添加的槽函数编写代码,实现相应的操作功能。代码如下。

```
void Widget::on_addRowButton_clicked()
{
    QStringList str;
    str<<"教材名称"<<"作者"<<"出版社";
    int newRow = ui->tableWidget->rowCount();
    ui->tableWidget->insertRow(newRow);
    QTableWidgetItem * item;
    for(int j=0; j<ui->tableWidget->columnCount(); j++) {
        item = new QTableWidgetItem(str[j]);
        item->setBackground(QBrush(QColor("yellow")));
        ui->tableWidget->setItem(newRow,j,item);
    }
}
void Widget::on_insertRowButton_clicked()
{
    QStringList str;
    str<<"教材名称"<<"作者"<<"出版社";
    int newRow = ui->tableWidget->currentRow()+1;
    ui->tableWidget->insertRow(newRow);
    QTableWidgetItem * item;
    for(int j=0; j<ui->tableWidget->columnCount(); j++) {
        item = new QTableWidgetItem(str[j]);
        item->setBackground(QBrush(QColor("yellow")));
        ui->tableWidget->setItem(newRow,j,item);
    }
}
void Widget::on_deleteRowButton_clicked()
{
    int currentRow = ui->tableWidget->currentRow();
    ui->tableWidget->removeRow(currentRow);
}
```

(6) 构建并运行程序。单击主窗体中的“添加行”按钮,会在表格末尾增加一个新的行;单击“插入行”按钮,可以在表格的中间插入一个新的行,如图 3.29 所示。

图 3.29 例 3.5 程序测试结果