目录

《Linux 环境下 Qt4 图形界面与 MySQL 编程》

前言

Linux 操作系统作为源码开放的自由软件，经过近 20 年的发展与壮大，越来越受到 IT 界的认可，在信息技术领域发挥着重要作用。图形界面是 Linux 走向成熟的重要支撑技术，备受众多开发者的重视。在每个工程项目中，几乎都离不开数据库技术。因此，稳定的 OS、友好的图形界面和完善的数据库技术构成了一个完整的工程项目。

面对 Linux 操作系统图形界面与数据库编程，很多学习者不知道如何下手。本书正是在这种背景下编写的，紧紧围绕本书的写作主线“图形界面编程控件与数据库编程基础→简单易学的实例→实际工程项目开发与场景分析”，以当前最新的 Qt4.7 为依据，采用“深入分析控件+实例解析”的方式，并配合经典的实际工程项目，对 Linux 操作系统下的 Qt4.7 与 MySQL 编程技术进行了全面细致的讲解。本书主要分为以下三大部分：

Linux 图形界面编程基础。介绍 Qt4.7 的全部控件，并针对每个控件设计了一个简单易学的实例，抛砖引玉，加深读者对 Qt4.7 控件的认识和理解。

基于 Linux 操作系统的 MySQL 数据库设计基础。介绍 MySQL 基本操作，并针对每个操作设计了一个简单易学的实例，抛砖引玉，加深读者对 MySQL 操作的认识和理解。

基于 Qt4.7 与 MySQL 的经典的实际工程项目案例开发。在本书中设计了列车时刻表查询系统、酒店客房管理系统、房屋租赁系统、书店管理系统、学生上机考试系统、校园点菜系统、餐饮信息服务系统、视频音频播放器、桌面常用软件小助手、俄罗斯方块游戏和局域网聊天系统。这些案例给读者提供了实际工程项目开发参考。

在本书的编写过程中，参考和借鉴了很多资料，它们为本书的编写和实验例程的解决方案提供了重要的指导作用。本书中的范例源代码可以到华章网站（www.hzbook.com）下载。

本书主要由邱铁、周玉、张民垒等完成编著任务。在此，感谢所有参与本书构思、解决方案、编辑和出版工作的同事、同行和为本书编写提供灵感的同志们，其中马利超参加了代码调试和部分文档整理，陈方疏、易磊、刘继伟、王革正、陈坚、宋莉莉、张涛、郝若男、刘晗、张晓彤、王宇辰、高凡等在课程设计中的构思被本书采纳，并被设计成典型案例，向他们表示感谢。

开源项目还在向前飞速发展，Qt 与 MySQL 版本还在不断更新，书中难免存在错误和不妥之处，恳请读者批评指正，并将信息发送到 openlinux2011@gmail.com，我们会尽力及时答复。——编者。

第一章　绪论

《Linux 环境下 Qt4 图形界面与 MySQL 编程》本书以“图形界面编程控件与数据库编程基础→简单易学的实例→实际工程项目开发与场景分析”为写作主线，以当前最新的 Qt4.7 为依据，采用“深入分析控件+实例解析”的方式，并配合经典的实际工程项目，对 Linux 操作系统下的 Qt4.7 与 MySQL 编程技术进行了全面细致的讲解。本节为大家介绍图形界面设计的必要性。

本章目的：

了解图形界面设计的必要性

认识几种常用的图形化界面编程工具

认识几种常用的 Linux 数据库技术

图形界面与数据库综合编程

1.1　图形界面设计的必要性

在信息技术高速发展的今天，无论是 PC 应用软件还是嵌入式设备，人们对软件的友好支持性要求逐渐提高，因此出现了图形用户界面（Graphical User Interface，GUI）技术。它是一种综合了计算机技术、美学、心理学、行为学以及各商业领域需求分析的人机系统工程，强调“人—机—环境”三者作为一个系统进行总体设计。图形用户界面技术的出现，改变了传统的采用终端命令行显示与控制的方式，从可视化、交互性和友好性等方面带来了极大的优越性。

控件功能的可视化。在图形用户界面编程工具中，将不同的用户需求做成控件的形式，在编程时，可以从控件库中选择相应的控件进行设置，从而达到自己的功能需求。每个控件的功能都能直观地显示出来，具有良好的可视化效果。

图形用户界面建立了与用户的互动交流。采用终端作为控制，可操作性差。当程序执行时，与用户的交互性差。图形用户界面技术将用户与程序执行过程控制紧密结合起来，从而使用户的需求及时地在图形用户界面中得以实施并直观地显示出来。

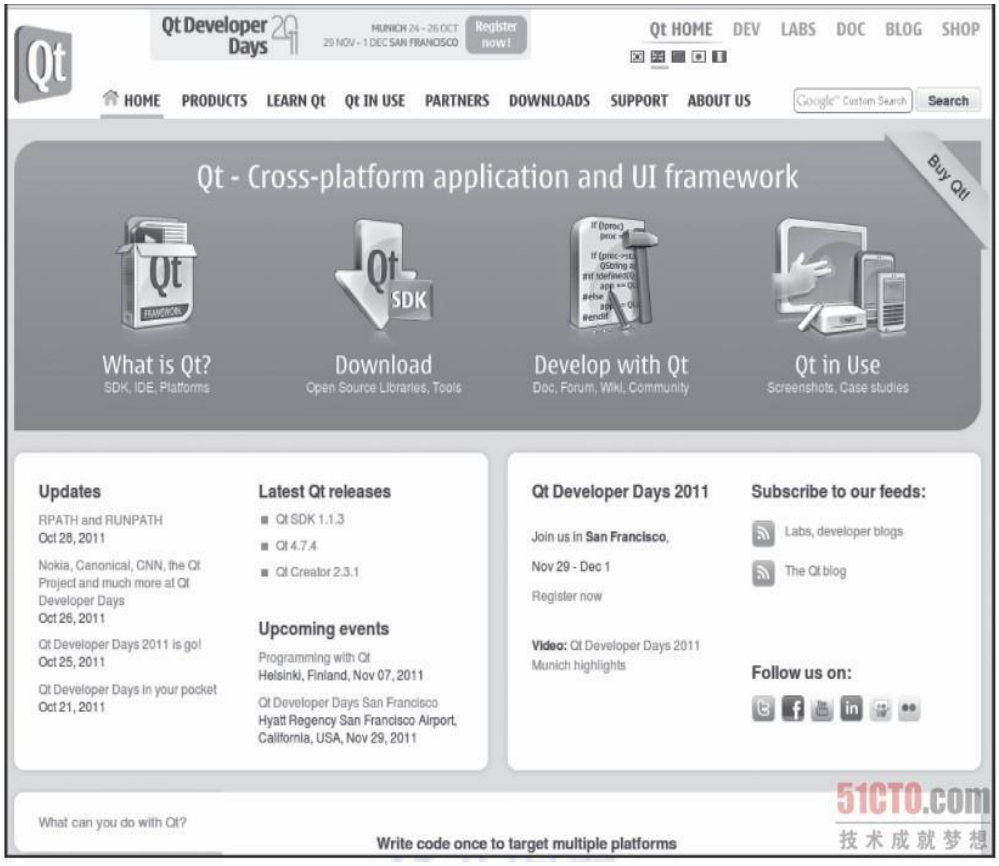
用户界面友好性。图形用户界面的设计要考虑人的行为学和心理学特点，符合用户的习惯，并在美学方面达到和谐统一。使用户操作方便，给人以舒适的感觉。

1.2 Linux 常用的图形化界面编程工具

本节介绍在 Linux 操作系统中编程时，常用的两种编程工具：Qt 和 GTK+。

1.2.1 Qt

Qt 是一个跨平台的图形界面开发平台，最早由挪威的 TrollTech 公司于 1992 年进行开发。2008 年 6 月，NOKIA 公司获得了 Qt 的开发权，继续对 Qt 平台进行开发。Qt 图形界面开发平台可进行嵌入式系统应用程序和桌面系统应用程序的开发，支持 Windows、Linux/X11、Mac OS X 等操作系统。Qt 官方网页：http://qt.nokia.com/，如图 1-1 所示。



Qt 采用 C++语言，包含了丰富的 C++类，包括窗口界面设计的接口、IO 控制接口、绘图接口、多媒体接口、数据库操作接口、网络通信接口、XML 接口、模块测试接口等丰富的开发接口。软件开发人员通过使用这些接口，可以方便、高效地完成应用设计与程序开发。由于采用 C++语言，Qt 具有较高的执行效率。此不同平台间的 Qt 开发接口是相同的，因此，可以有效地降低 Qt 应用程序跨平台开发的移植成本。

Qt 不仅仅是一个图形界面开发类库，而是一套拥有相对完整的开发环境的开发工具。Qt 提供了丰富的开发工具，用来提高应用设计开发人员的编程效率。这些工具主要包括：界面设计工具 Qt Designer、工程管理工具 qmake 等。在最新的 Qt4.5 及以上的版本中还加入了 Qt 程序开发的 IDE 环境 Qt Creator，以及与其他开发工具的扩展插件，可以支持 Visual Studio 和 Eclipse 等常用开发工具。本书的实例都是基于使用 Qt4.7 的 Creator 集成开发环境而编写的。

NOKIA 公司收购 TrollTech 公司后，推出了两个重要版本：

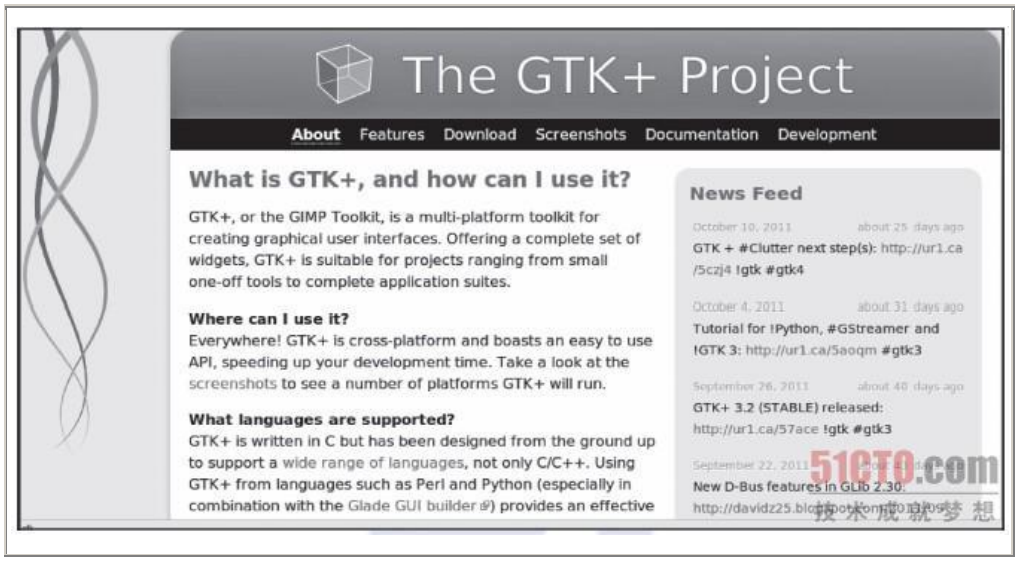
Qt 商业版：主要面向商业软件开发。这些软件用于商业用途时，需要支付一定的费用，并在有效的时间段内得到一定的技术支持。

Qt 开源版：源码开放，遵循 GNU 通用公共许可证(General Public License，GPL)，可以进行自由开发。

1.2.2 GTK+

GTK+来源于 GNU Image Manipulation Program（GIMP）（这是一个 GNU 开放源码项目），由 Peter Mattis 和 Spencer Kimball 共同开发。

确切地说，GTK+是一个图形用户界面开发工具库。利用这个工具库可以创建基于图形用户界面的应用程序，也就是说，GTK+集成程序员编程时需要用的控件库，利用这些库可以创建用户所需要的图形界面。GTK+是从 GTK 中分离出来的，“+”代表了 GTK+具有很好的扩展性和支持面向对象技术。关于 GTK+的详细内容，感兴趣的读者可以参阅 GTK+的官方网站：http://www.gtk.org/，如图 1-2 所示，本书不作过多的讨论。



1.3 两种常用的 Linux 数据库

1.3.1 MySQL

MySQL 由瑞典的 MySQL AB 公司开发，是一个中小型数据库管理系统。在 2008 年被 Sun 公司收购，成为 Sun 旗下的产品。2009 年，Sun 又被 Oracle 公司并购。目前，MySQL 是 Oracle 公司的产品之一，其 LOGO 如图 1-3 所示。MySQL 是一种关系数据库管理系统，它将数据保存在一个个数据表中，大大地提高了数据操作效率。MySQL 支持 SQL 语言访问数据库系统，操作简便。MySQL 源代码遵循 GNU 通用公共许可证。源代码是开放的，利于设计者根据不同的应用设计专用的解决方案。MySQL 广泛地应用于 Internet 上的中小型网站和嵌入式数据库的开发。

1.3.2 SQLite

SQLite 代码量特别小，大约 3 万行，是一个轻量级的小型数据库系统，占用资源少，处理速度快，最初是为嵌入式产品而设计的。其 LOGO 如图 1-4 所示。

图 1-3　MySQL 的 LOGO

图 1-4　SQLite 的 LOGO

SQLite 能够在多个平台上运行，且使用方便，无须安装和管理配置。能够支持多种编程语言直接创建数据库，灵活自由。2000 年 5 月，SQLite 第一个版本诞生，从此其每一个版本几乎都应用在嵌入式产品开发中。经过近些年嵌入式开发者和自由软件爱好者的完善与改进，SQLite 已经形成了功能相对齐全的嵌入式数据库系统，并在嵌入式产品中得以广泛应用。

1.4　图形界面与数据库综合编程技术

在当前的软件产品和嵌入式开发中，单一的图形界面开发和单一的数据库开发的案例已经很少了，一般都是图形界面配合数据库编程，以获得功能完善的产品设计方案。典型的软件产品开发分为两个部分：数据库管理员功能模块和用户应用程序模块。数据库管理员功能模块如图 1-5 所示，数据库中的数据由数据库管理工具通过数据库服务器进程来维护，数据库管理员可以利用数据库管理工具对数据库进行维护操作。

图 1-5　数据库管理员功能模块

用户应用程序功能模块如图 1-6 所示，应用程序是基于图形界面开发工具类库开发的，数据库驱动程序将数据库服务器进程与开发工具类库联系在一起，使它们建立一定的通信机制，数据库服务器进程负责对数据库中的数据进行直接操作。因此，用户可以通过应用程序对数据库进行访问操作。

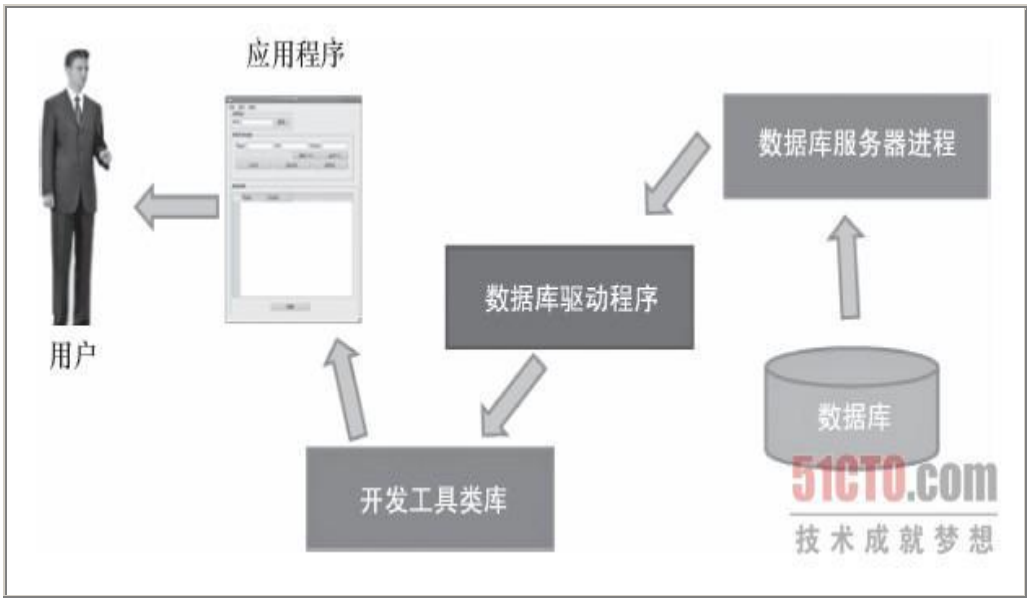


图 1-6

第 2 章　Qt4 图形界面编程基础

本章目的：

Qt Creator 的简单介绍

Qt Creator 的安装

Qt Creator 的信号槽机制

Qt Creator 的简单使用

2.1 Qt Creator

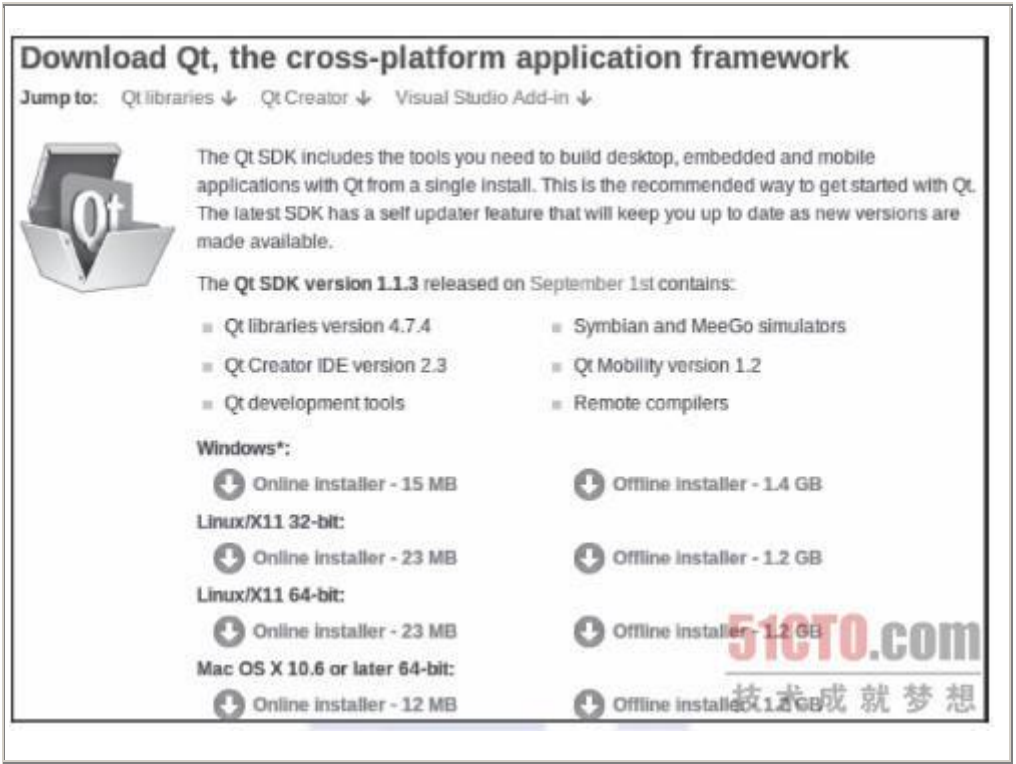
Qt Creator 是一个基于 Qt 的进行用户界面设计的可视化集成开发环境，可以跨平台运行，支持 Linux、Mac OS、Windows 等操作系统。Qt Creator 集成了 Qt Designer 的所有特性，方便用户进行界面设计。另外还集成了 Qt Assistant、Qt Linguist、图形化的 GDB 调试前端、qmake 构建工具等。Qt Creator 还包括项目生成向导、高级的 C++代码编辑器、浏览文件及类的工具。Qt Creator 使开发人员能够更加便捷地完成开发任务，Qt Creator 界面如图 2-1 所示。

（点击查看大图）图 2-1 Qt Creator 界面

2.2 安装 Qt 环境及 Qt Creator（1）

本书的所有实例都是在Ubuntu环境下进行的,这里只介绍在Ubuntu下搭建环境的具体步骤，其他 Liunx 系统环境搭建与此相类似。

可以从 Qt 官方网站上直接下载安装 Qt Creator，网址为：http://qt.nokia.com/downloads，打开后的网页如图 2-2 所示，我们选择下载 Qt SDK 安装包，其中包含用于 Windows、Linux/X11 32-bit、Linux/X11 64-bit 和 Mac OS 的安装包，笔者系统为 Linux/X11 64-bit，所以选择 Linux/X11 64-bit 的 SDK 安装包。Linux/X11 64-bit 的 SDK 安装包分为在线安装包和本地安装包，根据网速选择，网速较快可选择在线安装。



（点击查看大图）图 2-2　Qt Creator 下载选择

下载完毕后，直接在终端运行安装包，出现图 2-3 所示的界面，单击“Next”按钮，出现图 2-4 所示界面。在图 2-4 所示窗口中设置安装路径，此处使用默认的安装路径，安装类型有两种，可以是默认的安装类型，也可以是用户自定义的安装类型，默认的安装类型安装的东西比较多，占用的空间大，建议初学者选择“Default”安装类型，此处笔者选择“Customer”安装类型，单击“Next”按钮，出现图 2-5 所示的界面。图 2-5 中是一些安装选项，此处将与“Symbian”相关的选项都去掉，单击“Next”按钮，出现图 2-6 所示的界面。图 2-6 是安装协议，选择接受安装协议选项，单击“Next”按钮，出现图 2-7 所示的界面，进行 Qt Creator 的安装。安装完成之后单击“Next”按钮，出现图 2-8 所示的界面，单击“Finish”按钮成功安装 Qt Creator。此时在系统的“应用程序”选项中选择“编程”，可以看到 Qt Creator 选项，说明已成功安装 Qt Creator（见图 2-9）。下面将通过一个简单的实例说明 Qt Creator 的使用方法。

图 2-3　Qt Creator 安装第一步

图 2-4　Qt Creator 安装路径

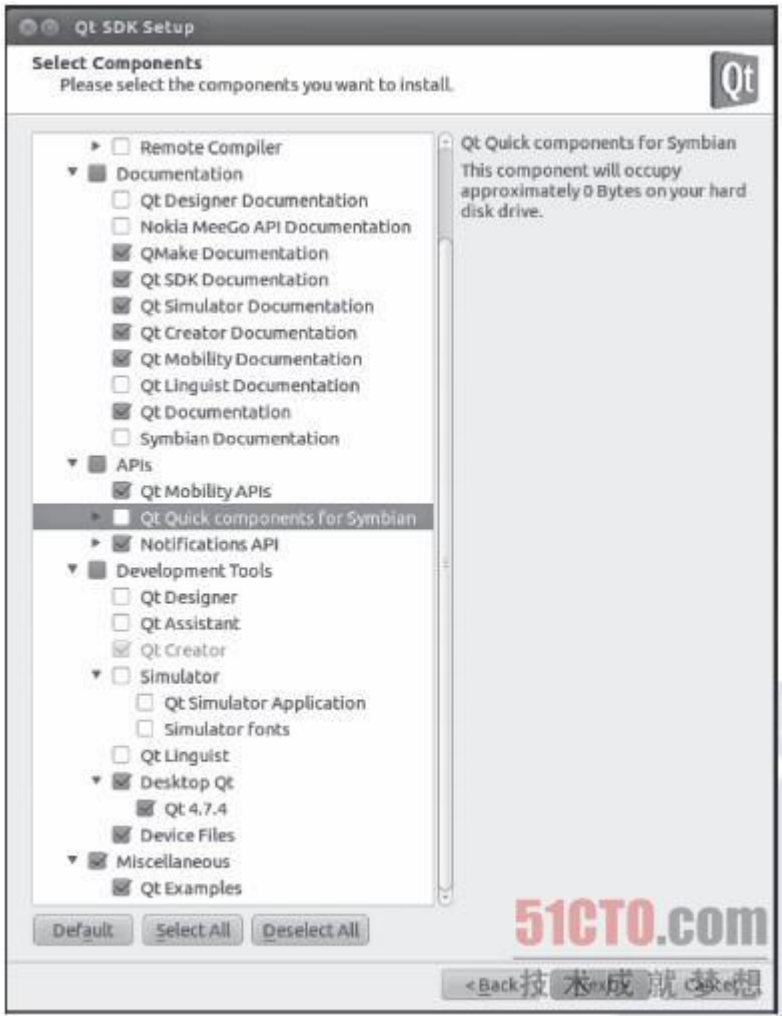

Qt SDK Setup
Select Components
Please select the components you want to install.
Remote Compiler
Documentation
Qt Designer Documentation
Nokia MeeGo API Documentation
QMake Documentation
Qt SDK Documentation
Qt Simulator Documentation
Qt Creator Documentation
Qt Mobility Documentation
Qt Linguist Documentation
Qt Documentation
Symbian Documentation
APIs
Qt Mobility APIs
Qt Quick components for Symbian
Notifications API
Development Tools
Qt Designer
Qt Assistant
Qt Creator
Simulator
Qt Simulator Application
Simulator fonts
Qt Linguist
Desktop Qt
Qt 4.7.4
Device Files
Miscellaneous
Qt Examples
Qt Quick components for Symbian
This component will occupy approximately 0 Bytes on your hard disk drive.
Default
Select All
Deselect All
51CTO.com
技术成就梦想
< Back
Cancel

图 2-5　安装组件选择

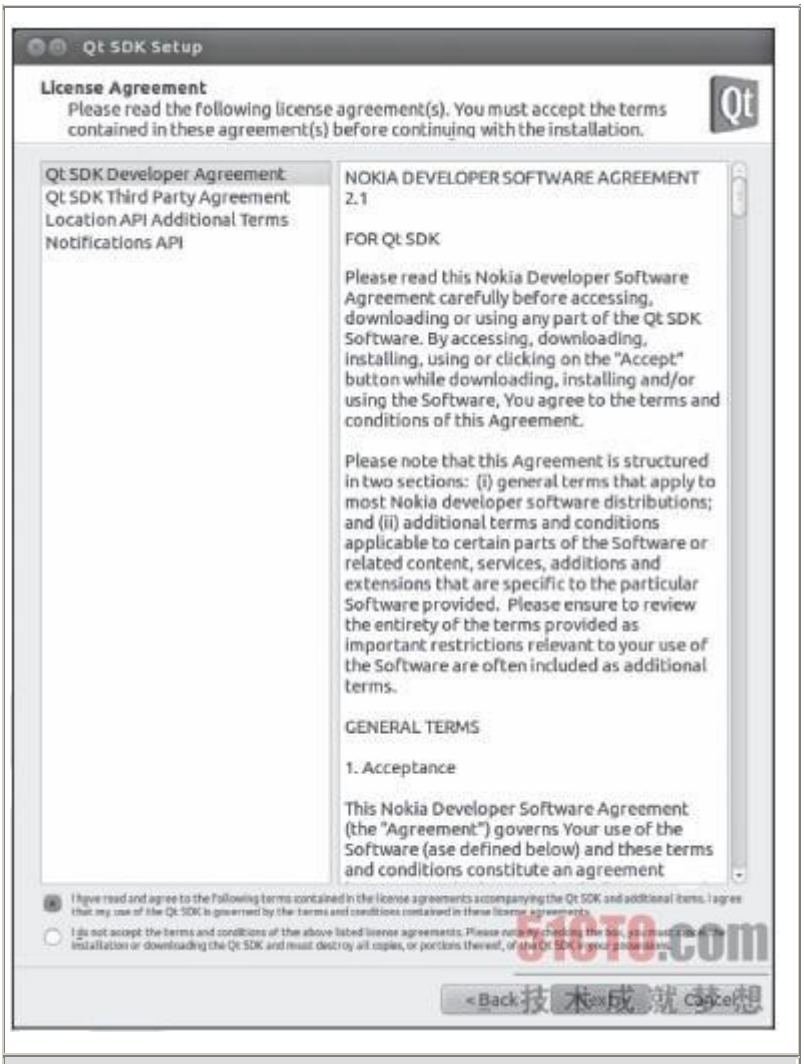


图 2-6　接受安装协议

图 2-7　安装完成

图 2-8　安装成功

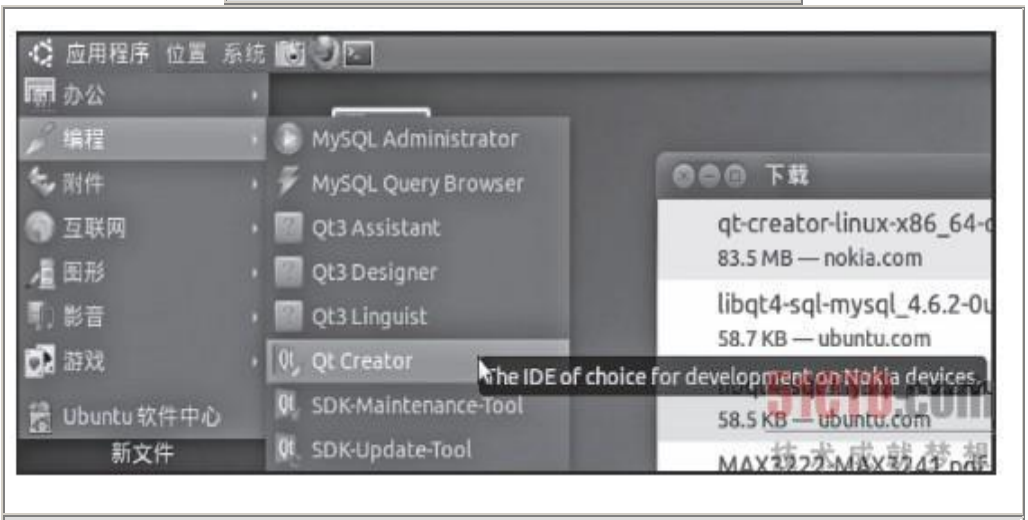


图 2-9　安装成功之后应用程序增加的选项

2.3 信号和槽机制

《Linux 环境下 Qt4 图形界面与 MySQL 编程》本书以“图形界面编程控件与数据库编程基础→简单易学的实例→实际工程项目开发与场景分析”为写作主线，以当前最新的 Qt4.7 为依据，采用“深入分析控件+实例解析”的方式，并配合经典的实际工程项目，对 Linux 操作系统下的 Qt4.7 与 MySQL 编程技术进行了全面细致的讲解。本节为大家介绍信号和槽机制。

1.概述

Qt 采用了一种全新的对象和方法的关联与通信机制，称为信号和槽机制。信号和槽机制是独立于标准 C++编译器的，在编译之前需要经过 Qt 的专门预处理工具 MOC（Meta Object Compiler，元组件编译器）对代码进行预处理后才能进一步进行 C++代码的编译。MOC 会将 Qt 应用程序中特有的代码自动转化为相应的标准 C++语法代码。

信号和槽的概念是 Qt 编程中最具代表性的特点之一。GUI 编程中通常使用回调函数进行事件处理，而回调函数通常是一个函数指针，不同的事件、不同的对象都有着各自的回调函数。当事件到来时，系统会通过调用对应的回调函数来完成相应的处理。

Qt 引入了信号和槽机制来取代回调函数。凡是继承自 QObject 的类都可以具有信号和槽成员，并可以使用它们。信号和槽的使用可以有效地减少函数指针的使用，使程序代码清晰简洁，对于事件响应管理更加容易。此外，信号和槽没有严格规定函数的类型，因此在调用过程中是安全的。

信号和槽的使用可以实现信息封装，增加程序的灵活性。信号和槽都采用函数作为存在形式。在 Qt 程序初始化或运行的过程中，可以静态或动态地将信号和槽相关联。当某一事件到来时，会发射信号，但是发射后，它并不需要关心信号的处理者是谁。当触发槽函数时，说明与其关联的信号被接收，但它不需要了解谁发出了信号，它只需要负责进行相应的处理即可。

2.信号

信号是 Qt 中对事件的一种抽象，当一个事件到来时，会发射信号。所谓的发射信号，就是通过 Qt 中特有的通信机制，调用和信号相关的各个槽函数。因此当发射信号时，将执行与其相关联的槽函数。信号采用函数的形式，因此，当所有与信号关联的函数全部返回后，信号函数才会返回。信号函数在形式上与标准 C++中的虚函数类似，信号函数只有头文件中函数的声明，而没有函数的定义（即函数体）。信号的声明形式如下：

```
signal:
void MySignal();
void MySignal(int x);
void MySignalParam(int x, int y);
```

对于信号的声明与普通的 C++函数无异，它不限制参数的个数与类型，同时它还支持重载。不过信号的返回值只要求必须为 void 型，因为事件是一种中断，对于突发性的中断，不可能期望其有返回值。

3.槽

槽是 Qt 中负责信号处理的实体，当有信号发射时，与信号关联的所有槽会依次执行。槽也采用函数的形式，不过槽需要有实际的函数定义，相当于在标准 C++中对虚函数的多态实现。槽函数的声明形式如下：

```
public slot:
void MySlot();
void MySlot(int x);
void MySlotParam(int x, int y);
```

与信号相同，槽函数的定义同普通的 C++函数无异，支持 C++函数的一些特性。同时，槽函数可以采用标准 C++函数的使用方式，在代码中直接调用。槽函数具有访问权限的标识，它们同 C++类的成员函数的标识相同，分别为：public、protected、private。public 说明该槽函数可以被其他类的信号所关联，protected 说明只能被类本身和其子类的信号所关联，private 说明该槽函数只能被类本身的信号所关联。

在 Qt 的基类 QObject 中有一个成员函数用来完成信号和槽的映射，函数的原型如下：

```
#include <QObject>
static bool QObject::connect (const QObject *sender,
const char *signal, const QObject *receiver, const char *mem
ber);
```

其中，sender 和 receiver 分别指定了被关联的信号和槽的发送者和接收者。signal 是信号，Qt 要求必须使用宏 SINGAL 将信号函数指针转化为指定的类型。member 是槽，Qt 要求必须使用宏 SLOT 转化函数指针。宏 SINGAL 和 SLOT 的参数形式如下：

```
SIGNAL(funname(param_type_1, param_type_2, …))
SLOT(funname(param_type_1, param_type_2, …))
```

其中，funname 是函数名，param_type_x 是函数中对应参数的类型。

2.4 一个抛砖引玉的实例（1）

在介绍具体控件使用方式之前，先来了解一下用 Qt Creator 进行开发的总体步骤。下面将通过 Hello 实例来简单介绍用 Qt Creator 开发程序的基本过程。

1.新建项目

运行 Qt Creator，选择菜单栏中的“文件”选项，然后选择“新建文件或项目”，出现“项目类型”选择窗口，如图 2-10 所示，单击左上角的“项目”选项框中的“Qt 控件项目”，然后选择右上角选项框中的“Qt Gui 应用”，参考图 2-10，单击“选择”按钮，出现新的窗口，如图 2-11 所示。图 2-11 所示的界面用于设置项目名和创建项目位置，设置结果如图 2-11 所示，单击“下一步”按钮，出现图 2-12 所示的界面。图 2-12 用于设置工程编译目标信息,取消有关 Desktop 编译的选项,保留 Qt 的调试和发布两部分,设置结果参考图 2-12，单击“下一步”按钮。出现图 2-13 所示的界面信息，此界面是创建主界面类的名称及类型，保持不变，单击“下一步”按钮。最后出现图 2-14 所示的信息，单击“完成”按钮完成新项目的创建，此时该项目将包含以下文件：Hello.pro、mainwindow.h、mainwindow.cpp、main.cpp、mainwindow.ui。

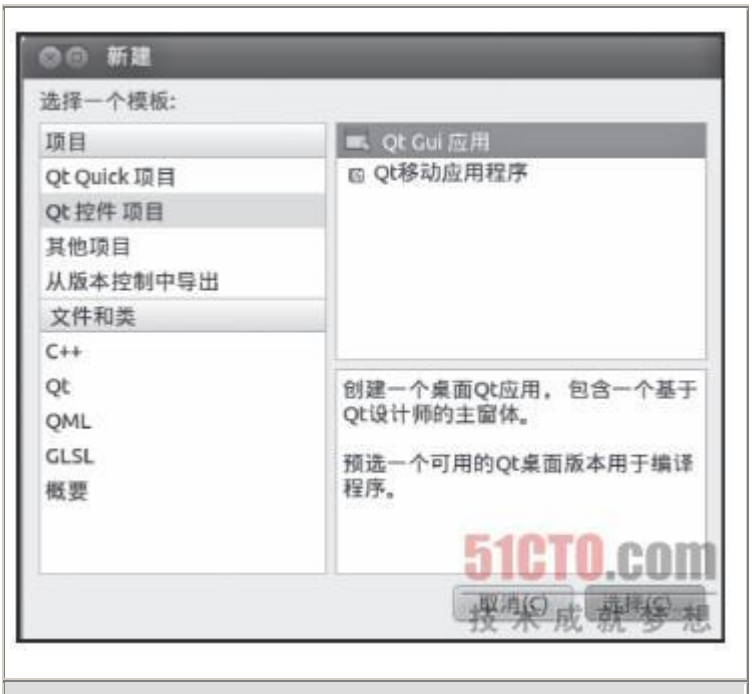


图 2-10 创建工程界面

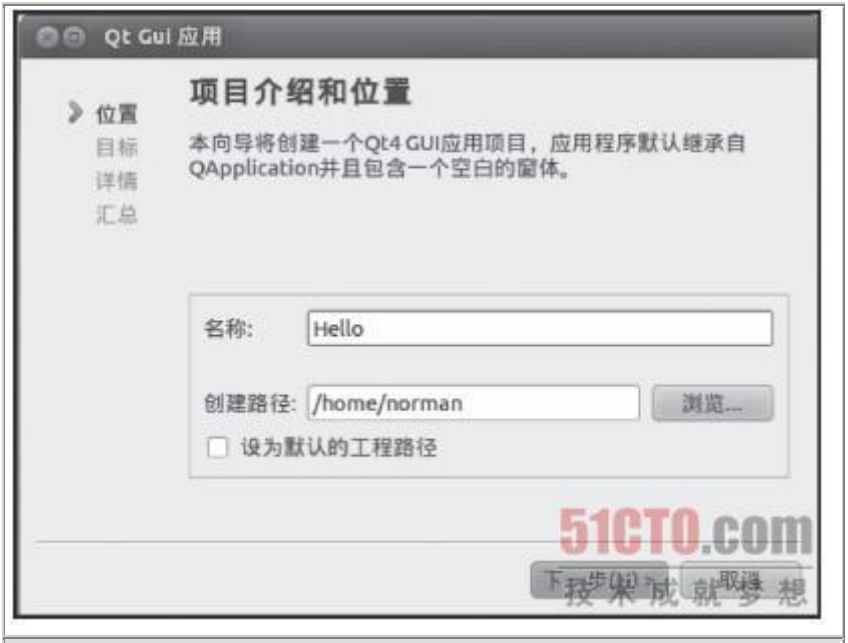


图 2-11　项目名及项目位置

（点击查看大图）图 2-12　项目编译目标设置

图 2-13　项目创建信息

图 2-14　项目编译目标文件位置

2.4　一个抛砖引玉的实例（2）

2.设计窗体

项目创建完毕后，开始设计窗体。Qt Creator 创建项目自带的 Main Window 主窗体默认是包含一个 QMenuBar、一个图 2-15　Hello 程序界面设计结果 QToolBar 和一个 QStatusBar，Hello 实例用不到这些控件，可以在窗体中删除这些自带控件。

图 2-15　Hello 程序界面设计结果

在属性设置窗口中修改主界面的 windowTitle 属性的值为“Hello”。在主界面中添加一个 QLabel 控件和一个 QPushButton 控件，QLabel 控件的属性不须更改，更改 QPushButton 控件的 text 属性为“OK”，设置结果如图 2-15 所示。

3.建立信号和槽的映射

界面设计完毕之后，需要实现其功能：当单击“OK”按钮时，“TextLabel”变为“Hello World”。此时需要将对应的信号和槽函数连接，Qt Creator 提供了三种建立信号和槽的方法。

第一种方法，自己定义槽函数和映射。

1)在文件 mainwindow.h 的类 MainWindow 的定义中声明槽函数，代码如下：

```
1. public slots:
2. void on_pushButton_clicked();
```

2)定义槽函数体在 mainwindow.cpp 文件中，代码如下：

```
1. void MainWindow:: on_pushButton_clicked ()
2. {
3. ui->label->setText("Hello World");
4. }
```

3)建立映射,在类 MainWindow 的构造函数中添加如下语句,以便将信号和槽函数进行连接:

```
1. connect(ui->pushbutton,SIGNAL(clicked()),this,SLOT(on_pushBu
   tton_clicked()));
```

第二种方法，右击“OK”按钮，选择“转到槽”选项，出现图 2-16 所示窗口，选择信号“clicked()”，单击“确定”按钮。系统自动生成槽函数的声明和定义，并建立内部映射，只须在槽函数体内编写槽函数的功能即可，添加语句参考第一种方法中的步骤 2。

第三种方法，右击界面选择“改变信号/槽”选项，单击“+”添加新的槽函数，结果如图 2-17 所示，单击“确定”按钮完成槽函数的添加。在窗体编辑区的下方有信号和槽的映射窗口，单击左上角的加号，出现一行新的映射，在这里进行映射的编辑，编辑结果如图 2-18 所示。最后重复第一种方法中的步骤 1 和步骤 2 完成槽函数的声明及定义。

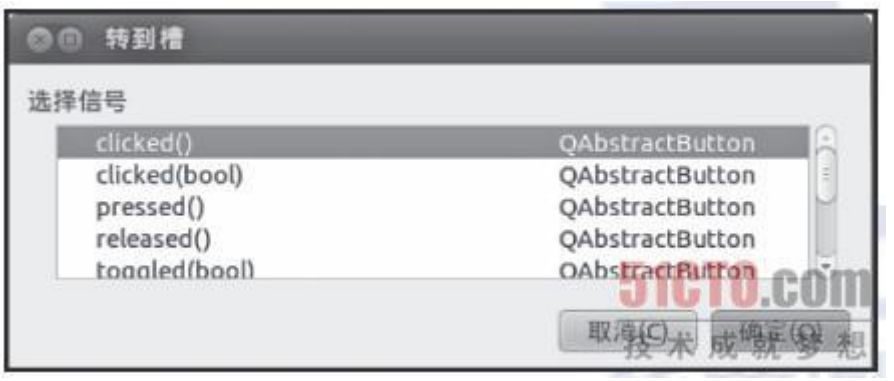


图 2-16　建立映射

图 2-17　槽函数的添加

图 2-18　信号槽函数映射 4.编译执行

首先选择菜单栏中“构建”选项中的“构建项目”选项，进行项目的构建，构建成功之后，单击窗口左下角的运行按钮即可运行程序，结果如图 2-19 所示，单击“OK”按钮，结果如图 2-20 所示。

图 2-19　Hello 程序加载界面

图 2-20　Hello 程序单击
“OK”按钮之后的界面

第 3 章　Qt4 控件与示例精讲

本章目的：

Qt4 界面编辑

Qt4 控件功能介绍

Qt4 控件简单实用举例

3.1 布局(Layouts)

窗体上的每个控件都需要有一个合适的尺寸和位置，从而设计出合理、美观的界面。Qt 中有三种方式对窗体上的控件进行布局管理，分别是：绝对布局、手工布局和 Qt 布局管理器

3.1.1 绝对布局

绝对布局控件的位置是固定的，控件不会随着窗口大小的变化而变化。

示例：

```
1. QLabel *lb=new QLabel(this);
2. lb->setText("ID:");
3. lb->setGeometry(10,10,40,20);
4. QLineEdit *leID = new QLineEdit("zml",this);
5. leID->setGeometry(50,10,50,20);  //定义 leID 的尺寸和位置
6. QLabel *lb1=new QLabel(this);
7. lb1->setText("psswd:");
8. lb1->setGeometry(10,40,40,20);//定义 lb1 的尺寸和位置
9. QLineEdit *lePwd = new QLineEdit("123456",this);
10.  lePwd->setGeometry(50,40,50,20);//定义 lePwd 的尺寸和位置
11.  QPushButton *btn = new QPushButton(QObject::tr("quit"),thi
   s);
12.  btn->setGeometry(50,70,50,20);//定义 btn 的尺寸和位置
```

图 3-1　绝对布局实例图

执行以上的布局代码，结果如图 3-1 所示。

绝对布局不足之处：

1）不能改变窗体的尺寸大小。

2）如果改变字体或者翻译成另一种语言，文本可能不能正常显示。

3）在一些样式下，控件的尺寸不容易控制。

3.1.2 手工布局

控件的位置是固定的，但是控件的尺寸会随着窗口的大小变化而变化，可以通过重写窗体控件的resizeEvent()实现对子控件的大小的设置。

示例：

```
setMinimumSize(265, 190);//窗体最小尺寸
resize(365, 240);//窗体默认大小
int w = width()- minimumWidth();


/*****************定义控件*********************************/
QLabel *nameLabel =new QLabel("Name:",this);
QLabel *pwLabel = new QLabel("Passwd:",this);
QLineEdit *nameLineEdit = new QLineEdit(this);
QLineEdit *pwLineEdit = new QLineEdit(this);
QPushButton *okButton = new QPushButton("OK",this);
QPushButton *cancelButton = new QPushButton("Cancel",this);

nameLabel->setGeometry(9, 9, 40, 25);  //定义 nameLabel 的尺寸和位置
nameLineEdit->setGeometry(55, 9, 50 + w, 25);//定义 nameLineEdit 的尺寸和位置
pwLabel->setGeometry(9, 40, 40, 25);//定义 pwLabel 的尺寸和位置
pwLineEdit->setGeometry(55, 40, 50 + w, 25);//定义 pwLineEdit 的尺寸和位置
okButton->setGeometry(150 + w, 9, 85, 25);//定义 okButton 的尺寸和位置
cancelButton->setGeometry(150 + w, 40, 85, 25);//定义 cancelButton 的尺寸和位置
```

执行以上的布局代码，结果如图 3-2 所示。

图 3-2　手工布局实例图

绝对布局和手工布局都需要开发人员编写很多代码，也需要很多的常量计算。如果设计界面改变了，所有的常量都要重新计算一遍。每个开发人员都不会愿意过多地编写这样的代码。管理窗体上控件最简单的方法就是使用 Qt 的布局管理器。下面就具体介绍 Qt 布局管理器的具体用法。

3.1.3　Qt 布局管理器

布局类能够给出所有类型控件的默认值，布局管理类能够得到控件的最大、最小尺寸，在窗口尺寸改变时自动调整控件。Qt Creator 有 4 种布局组件，如图 3-3 所示，同时存在对应的工具栏，如图 3-4 所示。

图 3-3　Qt Layouts 控件

图 3-4　Qt Layouts 的工具栏

Qt 提供的布局管理类有 QHBoxLayout、QVBoxLayout、QGridLayout、QStackLayout 4 种，具体介绍如表 3-1 所示。

表 3-1　Qt 布局类

控件类	中文名
QHBoxLayout	水平布局
QVBoxLayout	垂直布局
QGridLayout	表格布局
QStackLayout	分组布局

1.垂直布局

从左边的控件栏里选择一个 Vertical Layout（垂直布局管理器）放到中心面板，再从控件栏里选择三个 PushButton 放到垂直布局管理器上，效果如图 3-5 所示，可以看到控件按垂直方向排列，宽度随垂直布局管理器改变而改变，但是控件高度不变。

还可以先从控件栏里拖入控件,然后选中要布局的控件,单击工具栏中的垂直布局按钮即可，效果一样。如果对当前布局不是很满意,将布局管理器整体选中,单击上面工具栏上的“Break Layout”按钮，便可取消布局管理器。（也可以先将按钮移出，再按下【Delete】键将布局管理器删除。）各布局管理器之间差别不大，使用起来非常简单，其他布局管理器不做过多介绍。

2.分裂器部件（QSplitter）

先将控件同时选中，再单击工具栏上的“Lay Out Vertically in Splitter”（垂直分裂器）按钮，效果如图 3-6 所示。可以看到控件的大小可以随分裂器的大小改变而改变，这就是分裂器和布局管理器的区别。

图 3-5　垂直布局管理器

图 3-6　垂直布局分裂器

布局管理器还有一项重要的功能，使控件的大小随着窗口大小的改变而改变。例如我们使用的编辑软件 Qt Creator，它的编辑栏都是随着窗口大小的变化而变化的。如果窗口变大，而编辑栏没有变化，整个窗体就会显得不协调、不美观。

示例：

先在主窗口的中心放置一个文本编辑器 TextEdit，然后选中主窗口部件，在空白处单击鼠标右键，选择“Layout→Lay Out in a Grid”，使整个主窗口的中心区处于网格布局管理器中。任意拉伸窗口，TextEdit 的大小都会随之改变，界面如图 3-7 所示。

图 3-7　Lay Out in a Grid 示例界面

3.2　间隔器(Spacers)

Qt Creator 提供了水平间隔器和垂直间隔器，在控件栏中的位置如图 3-8 所示。

图 3-8　Spacers

1.控件位置

Spacers→Horizontal Spacer 或者 Vertical Spacer。

2.控件介绍

Spacers 控件（弹簧或间隔器）是一个用来填补空白的控件，方便布局，这里不做过多介绍。

3.3 按钮(Buttons)

Qt Creator 有 6 种 Buttons 控件，比 Qt Designer 多了 Command Link Button 和 Button Box 两种控件，如图 3-9 所示。

图 3-9 Buttons 控件

Buttons 控件的 Qt 类和名称如表 3-2 所示。

Qt Creator 的基本控件的用法和 Qt Designer 类似，因为它们使用的类基本相同，属性和成员函数没有太大变化，下面就来介绍 Qt Creator 中按钮控件的具体用法。

表 3-2 Buttons 控件介绍

控件类	控件名	中文名	控件类	控件名	中文名
QPushButton	PushButton	推动按钮	QRadioButton	RadioButton	单选按钮
QToolButton	ToolButton	工具按钮	QCommandLinkButton	Command Link Button	命令链接按钮
QCheckBox	CheckBox	复选框	QDialogButtonBox	Button Box	按钮盒

3.3.1 PushButton 控件

1. 控件位置

Buttons→PushButton

图 3-10 PushButton 控件

2. 控件介绍

PushButton 控件（推动按钮）继承自 QButton 类，样式如图 3-10 所示。通常用它来执行命令或触发事件。如果按钮具有焦点，就可以使用鼠标左键、【Enter】键或空格键触发该按钮的 Click 信号。习惯用一个标签来描述按钮的作用，标签显示内容为有下划线的字母或单词（在文本中它的前面被“&”标明），例如：

```
1. QPushButton *btn = new QPushButton("&OK",this);
```

在这个示例中加速键是【Alt+O】，并且文本标签将显示为“OK”。

3. 控件设置选项

在 PushButton 控件的 properties 选项中，一般常对以下选项进行设置。

name：该控件对应源代码中的名称；

text：该控件对应图形界面中所显示的名称；

font：设置 text 的字体；

enabled：该控件是否可用。

4. 常用成员函数

```
1. 1) PushButton::QPushButton(const QString & text, QWidget *paren
   t, const char *name = 0 )
```

构造一个名称为 name、父对象为 parent 并且文本为 text 的推动按钮。

```
1. 2) void QButton::clicked () [信号]
```

当单击该按钮时，发射这个信号。

```
1. 3) void QButton::pressed () [信号]
```

当按下该按钮时，发射这个信号。

```
1. 4) void QButton::released () [信号]
```

当释放该按钮时，发射这个信号。

```
1. 5) void QButton::setText ( const QString & )
```

设置该按钮上显示的文本。

```
1. 6) QString QButton::text () const
```

返回该按钮上显示的文本。

3.3.2 RadioButton 控件

1. 控件位置

Buttons→RadioButton

2. 控件介绍

RadioButton 控件（单选按钮）继承 QButton 类，样式如图 3-11 所示。单选按钮通常成组出现，用于提供两个或多个互斥选项，即在一组选项中只能选择一个。

图 3-11 RadioButton 控件

3. 控件设置选项

在 RadioButton 控件的 properties 选项中，一般常对以下选项进行设置。

name：该控件对应源代码中的名称；

text：该控件对应图形界面中所显示的名称；

font：设置 text 的字体；

enabled：该控件是否可用，可用时值为 true，不可用时值为 false；

checked:用来设置或返回是否选中单选按钮，选中时值为 true，没有选中时值为 false。

4. 常用成员函数

```
1. 1) QRadioButton::QRadioButton(const QString & text,QWidget *par
   ent, const char *name = 0 )
```

构造一个名称为 name、父对象为 parent 并且文本为 text 的单选按钮。

```
1. 2) bool QRadioButton::isChecked () const
```

返回是否选中单选按钮，选中时返回 true，没有选中时返回 false。

```
1. 3) void QButton::setText ( const QString & )
```

设置该按钮上显示的文本。

```
1. 4) QString QButton::text () const
```

返回该按钮上显示的文本。

```
1. 5) void QButton::stateChanged ( int state ) [signal]
```

当更改 checked 属性值时，将发射这个信号。

```
1. 6) void QRadioButton::setChecked ( bool check ) [virtual slot]
```

设置单选按钮是否被选中为 check。

3.3.3 CheckBox 控件

1. 控件位置

Buttons→CheckBox

2. 控件介绍

CheckBox 控件（复选框）继承自 QButton 类，样式如图 3-12 所示。CheckBox 和 RadioButton 都是选项按钮，它们的区别是选择模式，单选框提供的是“多选一”的选择，而复选框提供的是“多选多”的选择。

图 3-12　CheckBox 控件

3. 控件设置选项

在 CheckBox 控件的 properties 选项中，一般常对以下选项进行设置。

name：该控件对应源代码中的名称；

text：该控件对应图形界面中所显示的名称；

font：设置 text 的字体；

enabled：该控件是否可用，可用时值为 true，不可用时值为 false；

checked:设置复选框是否选中，选中时值为 true，没有选中时值为 false。

4. 常用成员函数

```
1. 1) QCheckBox::QCheckBox ( const QString & text, QWidget *parent
   , const char *name = 0 )
```

构造一个名称为 name、父对象为 parent 并且文本为 text 的复选框。

```
1. 2) bool QCheckBox::isChecked () const
```

选中该复选框，返回 true，否则返回 false。

```
1. 3) void QButton::setText ( const QString & )
```

设置该按钮上显示的文本。

```
1. 4) QString QButton::text () const
```

返回该按钮上显示的文本。

1. 5) void QButton::stateChanged (int state) [signal]

当更改 checked 属性时，将发射这个信号。

1. 6) void QCheckBox::setChecked (bool check) [槽]

设置复选框是否选中，状态是 check 的值。

3.3.4 示例 1：PushButton、RadioButton 和 CheckBox 控件的综合应用（1）

在这个示例中，我们建立标准的 Qt Gui Application 项目，它包含了菜单栏、工具栏和状态栏，但是在这个示例中不需要它们，所以把它们都删除。运用 3.1 节所学到的有关 Qt 布局的知识，设计界面如图 3-13 所示。

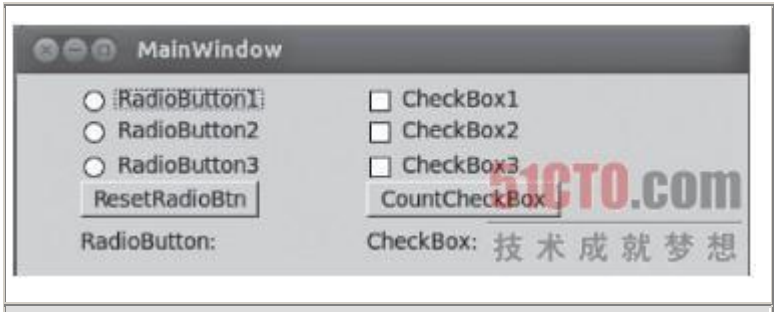


图 3-13 示例 1 界面

1. 控件说明

在属性编辑窗口中对控件的属性进行修改，修改内容如表 3-3 所示。

表 3-3 示例 1 主要控件说明

控件类型	控件名称	控件说明	文本
QRadioButton	radioButton	RadioButton1	RadioButton1
QRadioButton	radioButton_2	RadioButton2	RadioButton2
QRadioButton	radioButton_3	RadioButton3	RadioButton3
QCheckBox	checkBox	CheckBox1	CheckBox1
QCheckBox	checkBox_2	CheckBox2	CheckBox2
QCheckBox	checkBox_3	CheckBox3	CheckBox3
QPushButton	btn_RadioButton	设置 RadioButton1 为选中状态	ResetRadioBtn
QPushButton	btn_CheckBox	统计选中的 CheckBox	CountCheckBox
QLabel	label	显示选中的 RadioButton	RadioButton：
QLabel	label_2	显示选中的 CheckBox	CheckBox：

2.示例 1 说明

示例功能说明：

单击选中一个 RadioButton 时，label 显示选中的 RadioButton；

单击选中一个 CheckBox 时，label_2 显示当前所有选中的 CheckBox；

单击 btn_RadioButton 按钮时，RadioButton1 被选中，label 显示 RadioButton1 被选中；

单击 btn_CheckBox 按钮时，统计当前所有选中的 CheckBox，label_2 显示当前所有选中的 CheckBox。

3.示例实现

在这里我们使用 Qt Creator 自动生成的槽函数，不用写信号与槽函数的映射。（Qt Creator 自动生成槽函数的方法：右击控件→Go to slot，选择槽函数所要对应的信号函数，确定后就会生成槽函数的声明和定义框架。）

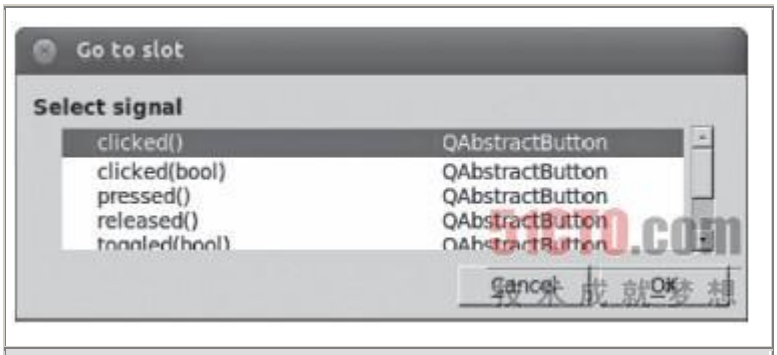


图 3-14　选择信号

右击控件 radioButton，选中“Go to slot”选项，在信号窗口中选择信号，如图 3-14 所示。在本示例中我们用到的是 clicked()信号，确定后就会跳转到槽函数的定义框架中，槽函数的声明已经自动生成，这一点和 Winform 差不多。然后只需要填充槽函数即可。

按照同样的方法为控件 radioButton_2、radioButton_3、checkBox、checkBox_2、checkBox_3、btn_RadioButton 和 btn_CheckBox 添加槽函数。

项目创建及相应的控件属性、信号、槽函数编译完成之后，接下来对相应文件进行编辑。

mainwindow.h（文中的粗体为需要添加的内容）：

```
#ifndef MAINWINDOW_H
#define MAINWINDOW_H
#include <QMainWindow>
namespace Ui {
  class MainWindow;
}
class MainWindow : public QMainWindow
{
  Q_OBJECT
 public:
   explicit MainWindow(QWidget *parent = 0);
   ~MainWindow();
   void displayCheckBox();
 private:
   Ui::MainWindow *ui;
 private slots: //槽函数，自动添加
   void on_checkBox_3_clicked();
   void on_checkBox_2_clicked();
```

```
    void on_checkBox_clicked();
    void on_radioButton_3_clicked();
    void on_radioButton_2_clicked();
    void on_radioButton_clicked();
    void on_btn_CheckBox_clicked();
    void on_btn_RadioButton_clicked();
};
#endif // MAINWINDOW_H
```

3.3.4 示例 1：PushButton、RadioButton 和 CheckBox 控件的综合应用（2）

在主窗体 mainwindow.cpp 文件中自动生成如下代码：

```
#include "mainwindow.h"
#include "ui_mainwindow.h"
MainWindow::MainWindow(QWidget *parent) :
  QMainWindow(parent),
  ui(new Ui::MainWindow)
{
  ui->setupUi(this);
}
MainWindow::~MainWindow()
{
  delete ui;
}
```

在主窗体 mainwindow.cpp 文件中添加槽函数 on_btn_RadioButton_clicked()：

```
/****槽函数：设置 RadioButton1 为选中状态****/
void MainWindow::on_btn_RadioButton_clicked()
{
ui->radioButton->setChecked(true);  //设置 radioButton 为选中状
态

/****设置 label 显示"RadioButton1 is Checked!"****/
ui->label->setText("RadioButton1 is Checked!");
}
```

在主窗体 mainwindow.cpp 文件中添加槽函数 on_btn_CheckBox_clicked()：

```
/****槽函数：显示被选中的 CheckBox****/
void MainWindow::on_btn_CheckBox_clicked()
{
QString str;
str ="";
ui->label_2->adjustSize();
if(ui->checkBox->isChecked())//checkBox 是否选中
```

```
{
  str += "checkBox1 is Checked;";
}
if(ui->checkBox_2->isChecked())//checkBox_2 是否选中
{
  str += "checkBox2 is Checked;";
}
if(ui->checkBox_3->isChecked())//checkBox_3 是否选中
{
  str +="checkBox3 is Checnked!";
}
ui->label_2->setText(str);
}
```

在主窗体 mainwindow.cpp 文件中添加槽函数 on_radioButton_clicked()：

```
/****槽函数：选中 RadioButton1****/
void MainWindow::on_radioButton_clicked()
{
  ui->radioButton->setChecked(true); //设置 radioButton 为选中状态

  /****设置 label 显示“RadioButton1 is Checked!”****/
  ui->label->setText("RadioButton1 is Checked!");
}
```

在主窗体 mainwindow.cpp 文件中添加槽函数 on_radioButton_2_clicked()：

```
/****槽函数：选中 RadioButton2****/
void MainWindow::on_radioButton_2_clicked()
{
  ui->radioButton_2->setChecked(true);   //设置 radioButton_2 为选中状态

  /****设置 label 显示“RadioButton2 is Checked!”****/
  ui->label->setText("RadioButton2 is Checked!");
}
```

在主窗体 mainwindow.cpp 文件中添加槽函数 on_radioButton_3_clicked()：

```
/****槽函数：选中 RadioButton3****/
void MainWindow::on_radioButton_3_clicked()
{
  ui->radioButton_3->setChecked(true);   //设置 radioButton_3 为选中状态

```

```
6.   /****设置 label 显示"RadioButton3 is Checked!"****/
7.   ui->label->setText("RadioButton3 is Checked!");
```

3.3.4 示例 1：PushButton、RadioButton 和 CheckBox 控件的综合应用（3）

在主窗体 mainwindow.cpp 文件中添加槽函数 on_checkBox_clicked()：

```
1. /****槽函数：选中 CheckBox1****/
2. void MainWindow::on_checkBox_clicked()
3. {
4.   if(ui->checkBox->isChecked())  //checkBox 被选中
5.   {
6. this->displayCheckBox();
7.   }
8.   else
9.   {
10.  this->displayCheckBox();
11.    }
12.  }
```

在主窗体 mainwindow.cpp 文件中添加槽函数 on_checkBox_2_clicked()：

```
1. /****槽函数：选中 CheckBox1****/
2. void MainWindow::on_checkBox_2_clicked()
3. {
4.   if(ui->checkBox_2->isChecked())//checkBox_2 被选中
5.   {
6. this->displayCheckBox();
7.   }
8.   else
9.   {
10.  this->displayCheckBox();
11.    }
12.  }
```

在主窗体 mainwindow.cpp 文件中添加槽函数 on_checkBox_3_clicked()：

```
1. /****槽函数：选中 CheckBox1****/
2. void MainWindow::on_checkBox_3_clicked()
3. {
4.   if(ui->checkBox_3->isChecked())//checkBox_3 被选中
5.   {
6. this->displayCheckBox();
7.   }
8.   else
9.   {
10.  this->displayCheckBox();
```

```
    }
}
```

在主窗体 mainwindow.cpp 文件中添加成员函数 displayCheckBox()：

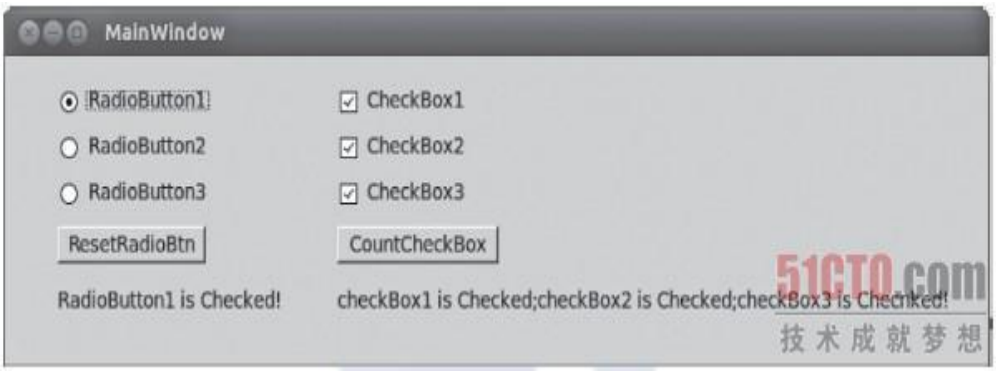
```
/****成员函数：显示当前所有 CheckBox 的状态****/
void MainWindow::displayCheckBox()
{
  QString str;
  str ="";
  if(ui->checkBox->isChecked())  //checkbox 被选中
  {
str += "checkBox1 is Checked;";
  }
   if(ui->checkBox_2->isChecked())//checkbox_2 被选中
   {
str += "checkBox2 is Checked;";
   }


   if(ui->checkBox_3->isChecked())//checkbox_3 被选中
   {
str +="checkBox3 is Checnked!";
   }
   ui->label_2->setText(str);
}
```

主文件 main.cpp 采用项目自动生成的即可。

4. 示例执行结果

示例执行结果如图 3-15 所示。



（点击查看大图）图 3-15　示例 1 执行结果

3.3.5 ToolButton 控件

1. 控件位置

Buttons→ToolButton

2. 控件介绍

ToolButton 控件（工具按钮）继承自 QButton 类，样式如图 3-16 所示。ToolButton 是一种用于命令或者选项的可以快速访问的按钮，通常用在 ToolBar 里面。工具按钮和按钮不同，工具按钮通常显示的是图标，而不是文本标签，一般用于编辑工具栏。另外，ToolButton 支持自动浮起。在自动浮起模式中，按钮只有在鼠标指向它的时候才绘制三维的框架。当按钮用在 ToolBar 里面的时候，Qt 默认启用这种模式，也可以使用 setAutoRaise()来改变它。

图 3-16 ToolButton 控件

3. 控件设置选项

在 ToolButton 控件的 properties 选项中，一般常对以下选项进行设置。

name：该控件对应源代码中的名称；

text：工具按钮标签文本；

font：设置工具按钮标签的字体；

autoRaise：自动浮起是否生效；

iconSet：提供显示在按钮上的图标的图标集；

on：工具按钮是否为开；

textLabel：工具按钮自动提示文本；

usesTextLabel：自动提示文本 textLabel 是否工作，默认为 false。

4. 常用成员函数

1. 1) QToolButton::QToolButton (QWidget *parent, const char *name = 0)

构造一个名称为 name、父对象为 parent 的 ToolButton。

1. 2) QToolButton::QToolButton(const QIconSet & iconSet, const QString & textLabel, const QString & grouptext, QObject *receiver, const char *slot, QToolBar *parent, const char *name = 0)

构造一个名称为 name,父对象为 parent(必须为 QToolBar)的工具按钮。工具按钮将显示 iconSet，工具提示为 textLabel，状态条信息为 grouptext，同时会将工具按钮连接到 receiver 对象的槽函数。

1. 3) QToolButton::QToolButton (ArrowType type, QWidget *parent, const char *name = 0)

此构造函数是把工具按钮构造成箭头按钮，type 定义了箭头的方向，可用的值有 LeftArrow、RightArrow、UpArrow 和 DownArrow。

1. 4) void QToolButton::setAutoRaise (bool enable)

根据参数 enable 值设置按钮是否可自动浮起。

1. 5) void QToolButton::setIcon (const QIconSet &)

设置显示在工具按钮上的图标。

1. 6) void QToolButton::setOn (bool enable) [虚 槽]

设置按钮是否为开，enable 等于 true 则设置为开，否则设置为关。

1. 7) void QToolButton::setTextLabel (const QString &) [槽]

设置按钮的提示标签。

1. 8) QString QToolButton::textLabel () const

返回按钮的提示标签。

3.3.6 示例 2：ToolButton 的应用（1）

ToolButton 经常与 ToolBar 一起使用，首先建立标准的 Qt Gui Application 项目。Qt Gui Application 项目会自动生成一个 ToolBar，控件名称是 mainToolBar。在这个示例中，由于用不到 menuBar 和 statusBar,因此把项目自动生成的 menuBar 和 statusBar 删除。另外，我们还用到了一个 TextEdit 控件，设计界面如图 3-17 所示。

1.控件说明

在属性编辑窗口中对控件的属性进行修改，修改内容如表 3-4 所示。

图 3-17　示例 2 界面

表 3-4　主要控件说明

控件类型	控件名称	控件说明	文本
QToolBar	mainToolBar	承载 ToolButton	无
QTextEdit	textEdit	响应单击 ToolButton 事件	无
QToolButton	buttonNew	要测试的 ToolButton	New
QToolButton	buttonOpen	要测试的 ToolButton	Open
QToolButton	buttonSave	要测试的 ToolButton	Save

2.示例说明

在 ToolBar 中添加三个 ToolButton，功能分别如下：

单击工具按钮 buttonNew，文本编辑框 textEdit 显示“New”；

单击工具按钮 buttonOpen，文本编辑框 textEdit 显示“Open”；

单击工具按钮 buttonSave，文本编辑框 textEdit 显示“Save”。

3.示例实现

由于 Qt Creator 的 ToolBar 不支持直接拖拽控件，因此我们不能通过拖曳方式在 ToolBar 中添加 ToolButton。在本示例用到的三个 ToolButton 都是自定义的。详细代码实现如下。

头文件 mainwindow.h（文中的粗体为需要添加的内容）：

```
#ifndef MAINWINDOW_H
#define MAINWINDOW_H
#include <QMainWindow>
namespace Ui {
  class MainWindow;
}
class MainWindow : public QMainWindow
{
  Q_OBJECT
  public:
    explicit MainWindow(QWidget *parent = 0);
    ~MainWindow();
  private:
    Ui::MainWindow *ui;
  private slots: //自定义槽函数
    void slotNew();
    void slotOpen();
    void slotSave();
  };
  #endif // MAINWINDOW_H
```

在主窗体 mainwindow.cpp 文件中自动生成如下代码：

```
#include "mainwindow.h"
#include "ui_mainwindow.h"
MainWindow::MainWindow(QWidget *parent) :
  QMainWindow(parent),
  ui(new Ui::MainWindow)
{
  ui->setupUi(this);
}
MainWindow::~MainWindow()
{
  delete ui;
}
```

在主窗体 mainwindow.cpp 文件中添加的头文件：

```
#include <QToolButton>
```

在主窗体 mainwindow.cpp 文件中构造函数：

```
3.
4. /*************构造函数*********************/
5. MainWindow::MainWindow(QWidget *parent) :
6.   QMainWindow(parent),
7.   ui(new Ui::MainWindow)
8. {
9.   ui->setupUi(this);
10.
11.    /**********定义 QToolButton************************/
12.    QToolButton *buttonNew = new QToolButton;//定义 buttonNew
   控件
13.
14.
15.    buttonNew->setText("New");
16.    QToolButton *buttonOpen= new QToolButton;  //定义
   buttonOpen 控件
17.    buttonOpen->setText("Open");
18.    QToolButton *buttonSave = new QToolButton;//定义
   buttonSave 控件
19.    buttonSave->setText("Save");
20.
21.    /**********在 mainToolBar 中添加 ToolButton**************/
22.    ui->mainToolBar->addWidget(buttonNew);
23.    ui->mainToolBar->addWidget(buttonOpen);
24.    ui->mainToolBar->addWidget(buttonSave);
25.
26.    /**********信号和槽的映射*******************************/
27.
   connect(buttonNew,SIGNAL(clicked()),this,SLOT(slotNew()));
28.
   connect(buttonOpen,SIGNAL(clicked()),this,SLOT(slotOpen()));

29.
   connect(buttonSave,SIGNAL(clicked()),this,SLOT(slotSave()));

30.  }
```

3.3.6 示例 2：ToolButton 的应用（2）

在主窗体 mainwindow.cpp 文件中添加槽函数 slotNew()：

```
1. /***槽函数：设置 textEdit 显示"New"***/
2. void MainWindow::slotNew()
3. {
```

```
4.    ui->textEdit->setText("New");  //设置 textEdit 显示"New"
5. }
```

在主窗体 mainwindow.cpp 文件中添加槽函数 slotOpen()：

```
1. /***槽函数：设置 textEdit 显示“Open”***/
2. void MainWindow::slotOpen()
3. {
4.    ui->textEdit->setText("Open");//设置 textEdit 显示"Open"
5. }
```

在主窗体 mainwindow.cpp 文件中添加槽函数 slotSave()：

```
1. /***槽函数：设置 textEdit 显示“Save”***/
2. void MainWindow::slotSave()
3. {
4.    ui->textEdit->setText("Save");//设置 textEdit 显示"Save"
5. }
```

主文件 main.cpp 采用项目自动生成的即可，不需任何更改。

4. 示例执行结果

示例执行结果如图 3-18 所示。

图 3-18　示例 2 执行结果

3.3.7　CommandLinkButton 控件

1. 控件位置

Buttons→CommandLinkButton

2. 控件介绍

CommandLinkButton 控件（命令链接按钮）继承自 QPushButton，样式如图 3-19 所示。CommandLinkButton 控件是一种 Windows Vista 风格的命令链接按钮，它和 RadioButton 相似，都是用于在互斥选项中选择一项。表面上同平面按钮一样，但是 CommandLinkButton 除带有正常的按钮上的文字描述文本外，默认情况下，它也将携带一个箭头图标，表明按下按钮将打开另一个窗口或页面。

图 3-19　CommandLinkButton 控件

3. 控件设置选项

在 CommandLinkButton 控件的 properties 选项中，一般常对以下选项进行设置。

name：该控件对应源代码中的名称；

text：该控件对应图形界面中所显示的标签；

font：设置 text 的字体；

enabled：该控件是否可用；

description：一个描述性的标签，以配合按钮上的文字。

4. 常用成员函数

1. 1) QCommandLinkButton::QCommandLinkButton (QWidget *parent = 0)

构造一个父对象为 parent 的命令链接按钮。

1. 2) QCommandLinkButton::QCommandLinkButton (const QString & text, QWidget *parent = 0)

构造一个父对象为 parent、文本为 text 的命令链接按钮。

```
1. 3) QCommandLinkButton::QCommandLinkButton (const
   QString & text,const QString & description,QWidget *parent = 0
   )
```

构造一个父对象为 parent、文本为 text 和描述文本为 description 的命令链接按钮。

```
1. 4) void QButton::clicked () [信号]
```

当单击该按钮时，发射这个信号。

```
1. 5) void QButton::pressed () [信号]
```

当按下该按钮时，发射这个信号。

```
1. 6) void QButton::released () [信号]
```

当释放该按钮时，发射这个信号。

```
1. 7) void QButton::setText ( const QString & )
```

设置该按钮上显示的文本。

```
1. 8) QString QButton::text () const
```

返回该按钮上显示的文本。

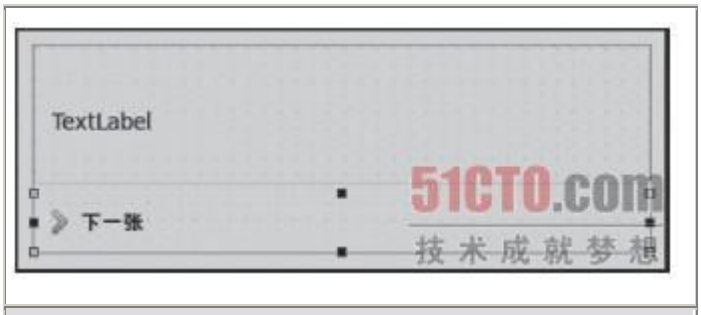


图 3-20　示例 3 界面

3.3.8　示例 3：CommandLinkButton 的应用（1）

首先建立标准的 Qt Gui Application 项目，把项目自动生成的 toolBar、menuBar 和 statusBar 删除，设计界面如图 3-20 所示。

1. 控件说明

在属性编辑窗口中对控件的属性进行修改，修改内容如表 3-5 所示。

表 3-5　主要控件说明

控件类型	控件名称	控件说明	文本
QLabel	imageLabel	显示图片	TextLabel
QScrollArea	scrollarea	承载 imageLabel	无
QCommandLinkButton	commandLinkButtonNext	触发显示下一张图片	下一张

2.示例说明

界面布局设置好之后，添加资源文件。在该示例的项目目录下新建文件夹 images，文件夹 images 中有 5 张图片，分别是：1.jpg、2.jpg、3.jpg、4.jpg 和 5.jpg。然后在 Qt Creator 中右击该项目，在弹出的窗口中选择“Qt Resource file”，如图 3-21 所示，添加的资源文件名称为 images，如图 3-22 所示，单击“Next”按钮，弹出如图 3-23 所示的窗口，保持默认设置，单击“Finish”按钮完成资源文件的添加。

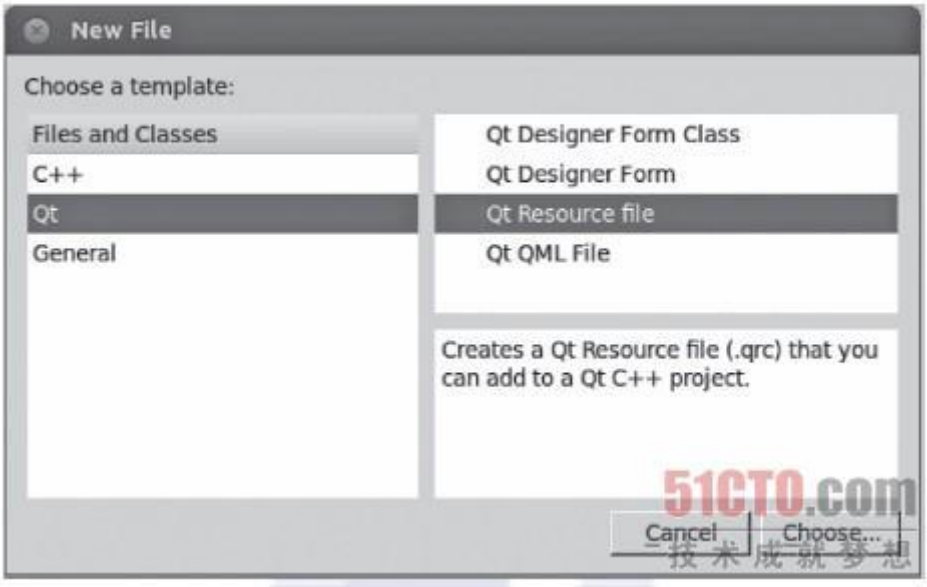


图 3-21　添加资源文件

图 3-22　资源文件设置

图 3-23　资源文件添加完成

打开新建的 images.qrc，选择“Add→Add Prefix”，如图 3-24 所示。

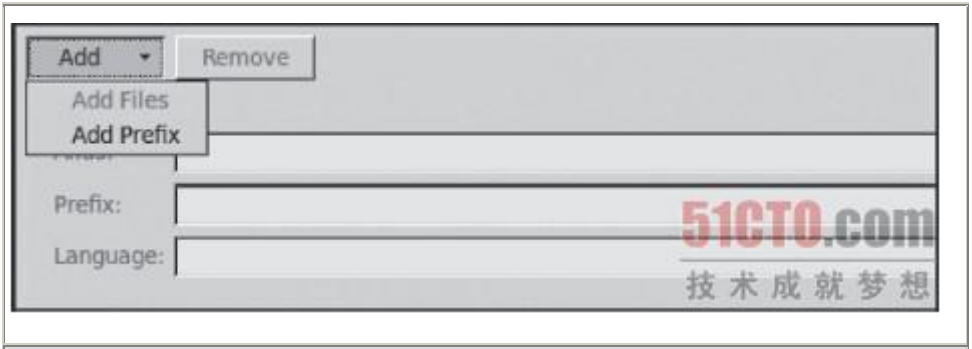


图 3-24　images.qrc 文件

3.3.8 示例 3：CommandLinkButton 的应用（2）

在 Prefix 栏内添加“/”，添加后如图 3-25 所示。

然后单击“add→Add Files”，选中要添加的 5 张图片并确定，添加成功后如图 3-26 所示。

图 3-25 添加“/”

图 3-26 添加完成

示例功能说明：这个示例很简单，程序执行后，label 显示 1.jpg，单击“Command Link Button”控件，显示下一张图片，循环显示。

3.示例实现

头文件 mainwindow.h（文中的粗体为需要添加的内容）：

```
1. #ifndef MAINWINDOW_H
2. #define MAINWINDOW_H
3. #include <QMainWindow>
4. namespace Ui {
5.   class MainWindow;
6. }
7. class MainWindow : public QMainWindow
8. {
9.   Q_OBJECT
10.  public:
11.    explicit MainWindow(QWidget *parent = 0);
12.    ~MainWindow();
13.  private:  //私有成员变量
14.    Ui::MainWindow *ui;
```

```
15.    QStringList list;//图片列表
16.    int currentImage;//当前图片编号
17.  private slots://私有槽函数
18.    void on_commandLinkButtonNext_clicked();//槽函数
19.  };
20.  #endif // MAINWINDOW_H
```

这里只在 Qt Gui Application 项目自动生成的头文件 mainwindow.h 里声明两个变量和一个槽函数。list 用于保存图片，currentImage 用于保存当前显示的图片编号。槽函数 on_commandLinkButtonNext_clicked()对应 commandLinkButtonNext 的单击信号。因此此处的槽函数是通过右击控件→Go to slots 自动生成的，所以不用在实现文件中添加映射函数。

在主窗体 mainwindow.cpp 文件中自动生成如下代码：

```
1. #include "mainwindow.h"
2. #include "ui_mainwindow.h"
3. MainWindow::MainWindow(QWidget *parent) :
4.   QMainWindow(parent),
5.   ui(new Ui::MainWindow)
6. {
7.   ui->setupUi(this);
8. }
9. MainWindow::~MainWindow()
10.  {
11.    delete ui;
12.  }
```

在主窗体 mainwindow.cpp 文件中添加的头文件：

```
1. #include <QMessageBox>
2. #include<QFileDialog>
```

在主窗体 mainwindow.cpp 文件中构造函数：

```
1. /***构造函数***/
2. MainWindow::MainWindow(QWidget *parent) :
3.   QMainWindow(parent),
4.   ui(new Ui::MainWindow)
5. {
6.   ui->setupUi(this);
7.   this->currentImage = 0;
8.
9.   /*************初始化图片列表 list*********************8/
10.
   this->list<<":/images/1.jpg"<<":/images/2.jpg"<<":/images/3.
   jpg"<<
```

```
    ":/images/4.jpg"<<":/images/5.jpg";
    ui->imageLabel->setBackgroundRole(QPalette::Base);
    ui->imageLabel->setSizePolicy(QSizePolicy::Ignored, QSizePolicy::Ignored);
    ui->imageLabel->setScaledContents(true);
    resize(500, 400);//设置窗体尺寸
    QString fileName = list.at(this->currentImage);//获取当前图片
    if(!fileName.isEmpty())//图片是否存在
    {
  QImage image(fileName);
  if(image.isNull())
  {
  QMessageBox::information(this, tr("Image Viewer"),tr("Cannot load %1.").arg(fileName));
  return;
  }
  ui->imageLabel->setPixmap(QPixmap::fromImage(image));//用图片填充 imageLabel
    }
  }
```

3.3.8 示例 3：CommandLinkButton 的应用（3）

在主窗体 mainwindow.cpp 文件中添加槽函数 on_commandLinkButtonNext_clicked()：

```
/***槽函数：实现图片循环展示***/
void MainWindow::on_commandLinkButtonNext_clicked()
{
  if(this->currentImage == 4)  //实现图片循环展示
this->currentImage =0;
  else
this->currentImage++;//更新当前图片
  QString fileName = list.at(this->currentImage);
  if(!fileName.isEmpty())//图片是否存在
    {
  QImage image(fileName);
  if(image.isNull())
  {
```

```
14.
   QMessageBox::information(this, tr("Image Viewer"),tr("Cannot
    load %1.").arg(fileName));
15.  return;
16.  }
17.  ui->imageLabel->setPixmap(QPixmap::fromImage(image)); //显
   示图片
18.    }
19.  }
```

不需要更改主文件 main.cpp，使用工程自动生成的即可。

4.示例执行结果

示例执行结果如图 3-27 所示。

图 3-27　示例执行结果

3.3.9　ButtonBox 控件

1.控件位置

Buttons→ButtonBox

2. 控件介绍

ButtonBox 控件（按钮盒）的样式如图 3-28 所示。ButtonBox 控件是由 QDialogButtonBox 类包装成的。

图 3-28　ButtonBox 控件

3. 控件设置选项

在 ButtonBox 控件的 properties 选项中，一般常对以下选项进行设置。

name：该控件对应源代码中的名称；

font：设置 text 的字体；

enabled：该控件是否可用；

centerButtons：ButtonBox 中的按钮是否居中布局，默认值为 false；

orientation：按钮布局方向，Qt 提供 QT::Horizontal 和 QT::Vertical 两种；

standardButtons：标准按钮集合。

4. 常用成员函数

1. 1) QDialogButtonBox::QDialogButtonBox (QWidget *parent = 0)

构造一个按钮盒，父对象为 parent。

1. 2) QDialogButtonBox::QDialogButtonBox (QT::Orientation orientation, QWidget *parent = 0)

构造一个按钮盒，父对象为 parent，排列方向为 orientation，并且包含 buttons。

1. 3) QDialogButtonBox::QDialogButtonBox(StandardButtons buttons, QT::Orientation orientation = QT::Horizontal, QWidget *parent = 0)

构造一个按钮盒，父对象为 parent，排列方向为 orientation。

1. 4) void QDialogButtonBox::accepted () [signal]

当单击按钮盒里的定义为 AcceptRole 和 YesRole 的按钮时，发射这个信号。

1. 5) void QDialogButtonBox::addButton (QAbstractButton *button, ButtonRole role)

向按钮盒里添加按钮 button，定义按钮 button 的角色为 role，如果 role 是无效的，则不添加按钮，如果按钮已添加，移除并再次添加为新角色。

1. 6) QPushButton *QDialogButtonBox::addButton (const QString & text, ButtonRole role)

创建一个按钮的文本为 text，以指定角色添加到按钮盒，并返回相应的按钮，如果 role 是无效的，就不创建按钮，返回 0。

1. 7) QPushButton *QDialogButtonBox::addButton (StandardButton button)

向按钮盒中添加一个标准按钮 button，并返回标准按钮。如果按钮无效，不添加，返回 0。

1. 8) void QDialogButtonBox::clear ()

清空该按钮盒里的所有按钮。

1. 9) void QDialogButtonBox::clicked (QAbstractButton *button) [signal]

当单击按钮盒里的按钮 button 时，发射这个信号。

1. 10) void QDialogButtonBox::helpRequested () [signal]

当单击按钮盒里的定义为 HelpRole 的按钮时，发射这个信号。

1. 11) void QDialogButtonBox::rejected () [signal]

当单击按钮盒里的定义为 RejectRole 和 NoRole 的按钮时，发射这个信号。

1. 12) void QDialogButtonBox::removeButton (QAbstractButton *button)

移出按钮盒里的按钮 button，但是不删除，设置它的父母为 0。

3.3.10 示例 4：ButtonBox 的应用（1）

按钮盒（ButtonBox）可以很方便地快速布置一组按钮。它有水平和垂直两种样式。可以用以下函数创建水平或垂直按钮盒。下面通过示例来看看 ButtonBox 的用法。

首先建立标准的 Qt Gui Application 项目，把项目自动生成的 toolBar、menuBar 和 statusBar 删除，设计界面如图 3-29 所示。

1.控件说明

在属性编辑窗口中对控件的属性进行修改，修改内容如表 3-6 所示。

图 3-29　示例 4 界面

表 3-6　主要控件说明

控件类型	控件名称	控件说明
QLineEdit	lineEdit	输入框
QTextEdit	textEdit	记录每次输入框中的内容并显示
QDialogButtonBox	buttonBox	Button 组 OK 和 Clear

2. 示例说明

在 buttonBox 中添加 PushButton 与“OK”按钮组成按钮组，名称是 Clear，按钮功能说明如下：

在 lineEdit 输入框中输入字符串内容，单击按钮组 buttonBox 中的“OK”按钮，lineEdit 中的内容添加到 textEdit 中；

单击按钮组 buttonBox 中的“Clear”按钮，清空 textEdit 中的所有内容。

3. 示例实现

本示例通过添加自定义的“Clear”按钮，方便大家全面地掌握 ButtonBox 的用法。

头文件 mainwindow.h（文中的粗体为需要添加的内容）：

```
1. #ifndef MAINWINDOW_H
2. #define MAINWINDOW_H
3. #include <QMainWindow>
4. #include <QAbstractButton>
5. #include <QPushButton>
6. namespace Ui {
7.   class MainWindow;
```

```
8. }
9. class MainWindow : public QMainWindow
10. {
11.   Q_OBJECT
12. public:
13.   explicit MainWindow(QWidget *parent = 0);
14.   ~MainWindow();
15. private:
16.   Ui::MainWindow *ui;
17. private:
18.   QPushButton *Clear;
19. private slots:
20.   void on_buttonBox_clicked(QAbstractButton *button);
21. };
22. #endif // MAINWINDOW_H
```

注意此处的槽函数是通过右击控件→Go to slots 自动生成的，所以不用在实现文件中添加映射函数。

在主窗体 mainwindow.cpp 文件中自动生成如下代码：

```
1. #include "mainwindow.h"
2. #include "ui_mainwindow.h"
3. MainWindow::MainWindow(QWidget *parent) :
4.   QMainWindow(parent),
5.   ui(new Ui::MainWindow)
6. {
7.   ui->setupUi(this);
8. }
9. MainWindow::~MainWindow()
10. {
11.   delete ui;
12. }
```

在主窗体 mainwindow.cpp 文件中构造函数：

```
1. /***构造函数***/
2. MainWindow::MainWindow(QWidget *parent) :
3.   QMainWindow(parent),
4.   ui(new Ui::MainWindow)
5. {
6.   ui->setupUi(this);
7.   Clear = new QPushButton;  //new QPushButton
8.   Clear->setText("Clear");//设置 Clear 的 Text
```

```
9.  ui->buttonBox->addButton(Clear,QDialogButtonBox::ActionRole)
    ;
```

3.3.10 示例 4：ButtonBox 的应用（2）

在主窗体 mainwindow.cpp 文件中添加槽函数 on_buttonBox_clicked(QAbstractButton*button)：

```
1.  /***槽函数：处理 buttonBox 中的单击按钮事件***/
2.  void MainWindow::on_buttonBox_clicked(QAbstractButton*button
    )
3.  {
4.    QString str;
5.    str = ui->textEdit->toPlainText();
6.  if(button == ui->buttonBox->button(QDialogButtonBox::Ok))
7.    {
8.  if(!ui->lineEdit->text().isEmpty())
9.  {
10.   str += ui->lineEdit->text()+"\n";
11.   ui->textEdit->setPlainText(str);  //更新 textEdit
12.   }
13.     }
14.     else if(button == this->Clear)
15.     {
16.   ui->textEdit->setPlainText("");//清空 textEdit
17.     }
18.   }
```

主文件 main.cpp 不需任何更改，使用工程自动生成的即可。

4.示例执行结果

示例执行结果如图 3-30 所示。

图 3-30　示例 4 执行结果

3.4　单元视图(Item Views)

Qt Creator 有 4 种 Item Views，如图 3-31 所示。

View Widgets 的 Qt 类和名称介绍如表 3-7 所示。

图 3-31　Item Views 控件

表 3-7　Item Views 介绍

控件类	控件名	中文名
QListView	ListView	列表视图
QTreeView	TreeView	树形视图
QTableView	TableView	表格
QColumnView	ColumnView	列视图

本节主要介绍 Qt Creator 中 View Widgets 的使用方法，视图控件可以用来很好地显示项目，在这方面它和列表框相同，只不过它的功能更加强大。下面就来具体介绍各种视图组件的用法。

3.4.1 ListView 控件

1. 控件位置

Item Views→ListView

2. 控件介绍

ListView 控件（列表视图）继承于 QAbstractItemView。ListView 是基于模型的列表/图标视图，为 Qt 的模型/视图结构提供了更灵活的方式，它是 Qt 模型/视图框架的一部分。可以看出 Qt Creator 的 ListView 和 Qt Designer 的 ListView 有很大差别。

3. 控件设置选项

在 ListView 控件的 properties 选项中，一般常对以下选项进行设置。

name：该控件对应源代码中的名称；

font：设置视图内字体；

batchSize：如果将 layoutMode 设置为 Batched，则这个属性保存批量处理的规格；

layoutMode：项目的布局模式；

modeColumn：模型中可见的列，默认情况下，值为 0，表明模型中的第一列可见；

viewMode：保存该 ListView 的视图模型。

4. 常用成员函数

1. 1) QListView::QListView (QWidget *parent = 0)

构造一个父对象为 parent 的 ListView。

1. 2) void QListView::currentChanged (const QModelIndex & current , const QModelIndex & previous) [virtual protected]

把 current 定为当前项目，previous 是以前的当前项目。

1. 3) void QListView::dataChanged (const QModelIndex & topLeft, const QModelIndex & bottomRight) [virtual protected]

更改模型中项目 topLeft 到 bottomRight。

1. 4) QModelIndex QListView::indexAt (const QPoint & p) const [virtual]

返回坐标点 p 处项目的模型索引。

1. 5) void QListView::rowsInserted (const QModelIndex & parent, int start, int end) [virtual protected]

插入新行，新行的父母是 parent，从 start 到 end 的所有项目。

1. 6) QModelIndexList QListView::selectedIndexes () const [virtual protected]

（从左到右依次是：ListView、TextEdit）返回所有选中和非隐藏的项目的模型索引。

3.4.2 示例 5：ListView 的应用

Qt 提供了 model/view 结构来管理与展示数据，model 提供数据模型，view 展示数据，delegate 对数据项进行渲染。model、view 和 delegate 通过信号/槽机制通信。下面通过这个示例来学习 ListView 的用法。

首先建立标准的 Qt Gui Application 项目，把项目自动生成的 toolBar、menuBar 和 statusBar 删除，设计界面如图 3-32 所示。

1. 控件说明

在属性编辑窗口中对控件的属性进行修改，修改内容如表 3-8 所示。

控件类	控件名	中文名
QListView	ListView	列表视图
QTreeView	TreeView	树形视图
QTableView	TableView	表格
QColumnView	ColumnView	列视图

图 3-32 示例 5 界面

2. 示例说明

程序执行后，会在 listView 中添加一些数据项。单击选中“listView”中的某一项，会在 textEdit 中显示该项。

表 3-8　主要控件说明

控件类	控件名	中文名
QListView	ListView	列表视图
QTreeView	TreeView	树形视图
QTableView	TableView	表格
QColumnView	ColumnView	列视图

3. 示例实现

头文件 mainwindow.h（文中的粗体为需要添加的内容）：

```
#ifndef MAINWINDOW_H
#define MAINWINDOW_H
#include <QMainWindow>
#include <QStandardItemModel>
#include <QModelIndex>
namespace Ui {
  class MainWindow;
}
class MainWindow : public QMainWindow
{
  Q_OBJECT
public:
  explicit MainWindow(QWidget *parent = 0);
  ~MainWindow();
private:
  Ui::MainWindow *ui;
private:
  QStandardItemModel *standardItemModel;//声明 model
private slots:
  void itemClicked(QModelIndex index);//槽函数
};
#endif // MAINWINDOW_H
```

在头文件中声明一个 QStandardItemModel 对象和一个槽函数 itemClicked(QModelIndex index)。对象 standardItemModel 是 listViewd 的 model,槽函数 itemClicked(QModelIndex index)对应 listView 中的选项被单击的事件。

在主窗体 mainwindow.cpp 文件中自动生成如下代码:

```
1. #include "mainwindow.h"
2. #include "ui_mainwindow.h"
3. MainWindow::MainWindow(QWidget *parent) :
4.   QMainWindow(parent),
5.   ui(new Ui::MainWindow)
6. {
7.   ui->setupUi(this);
8. }
9. MainWindow::~MainWindow()
10.  {
11.    delete ui;
12.  }
```

在主窗体 mainwindow.cpp 文件中添加的头文件:

```
1. #include <QBrush>
```

在主窗体 mainwindow.cpp 文件中构造函数:

```
1. /***构造函数***/
2. MainWindow::MainWindow(QWidget *parent) :
3.   QMainWindow(parent),
4.   ui(new Ui::MainWindow)
5. {
6.   ui->setupUi(this);
7.
8.   standardItemModel = new QStandardItemModel(this);
9.
10.    /*****************初始化 Item 数组
   ***********************************/
11.    QStringList strList;
12.    strList.append("a");
13.    strList.append("b");
14.    strList.append("c");
15.    strList.append("d");
16.    strList.append("e");
17.    strList.append("f");
18.    strList.append("g");
19.    strList << "h";
```

```
    strList += "i";
    int nCount = strList.size();  //获取 strList 的大小

    /****************初始化
model***********************************/
    for(int i = 0; i < nCount; i++)
    {
 QString string = static_cast<QString>(strList.at(i));
 QStandardItem *item = new QStandardItem(string);
 if(i %2 == 1)
 {
 QLinearGradient linearGrad(QPointF(0, 0), QPointF(200, 200
));
 linearGrad.setColorAt(0, QT::darkGreen);
 inearGrad.setColorAt(1, QT::yellow);
 QBrush brush(linearGrad);
 item->setBackground(brush);
 }
 standardItemModel->appendRow(item);
    }
    ui->listView->setModel(standardItemModel);//显示 model 中的
内容
    connect(ui->listView,SIGNAL(clicked(QModelIndex)),
    this,SLOT(itemClicked(QModelIndex)));
 }
```

在主窗体 mainwindow.cpp 文件中添加槽函数 itemClicked(QModelIndex index)：

```
/***槽函数：设置 textEdit 显示的内容***/
void MainWindow::itemClicked(QModelIndex index)
{

ui->textEdit->setText(ui->textEdit->toPlainText()+index.data
().toString()+"\n");
}
```

主文件 main.cpp 不需任何更改，使用项目自动生成的即可。

4. 示例执行结果

结果示例执行结果如图 3-33 所示。

图 3-33　示例 5 执行

3.4.3　TreeView 控件（1）

1.控件位置

Item Views→TreeView

2.控件介绍

TreeView 控件（树形视图）继承于 QAbstractItemView。TreeView 和 ListView 类似，是基于模型的列表/图标视图，也是 Qt 模型/视图框架的一部分。

3.控件设置选项

在 TreeView 控件的 properties 选项中，一般常对以下选项进行设置。

name：该控件对应源代码中的名称；

font：设置该控件内所有文本的字体；

sortingEnable：项目是否排序。

4.常用成员函数

1. 1) QTreeView::QTreeView (QWidget *parent = 0)

构造一个父对象为 parent 的 TreeView。

1. 2) void QTreeView::collapse (const QModelIndex & index) [slot]

折叠模型索引为 index 的项目。

1. 3) void QTreeView::collapseAll () [slot]

折叠所有项目。

1. 4) int QTreeView::columnAt (int x) const

返回 x 坐标处的列。

1. 5) void QTreeView::columnCountChanged (int oldCount, int newCount) [protected slot]

通知在树形视图中的列数，从 oldCount 改变到 newCount。

1. 6) void QTreeView::currentChanged (const QModelIndex & current, const QModelIndex & previous) [virtual protected]

把 current 定为当前项目，previous 是以前的当前项目。

1. 7) void QTreeView::dataChanged (const QModelIndex & topLeft, const QModelIndex & bottomRight) [virtual]

更改模型中项目 topLeft 到 bottomRight。

1. 8) void QTreeView::drawBranches (QPainter *painter, const QRect & rect, const QModelIndex & index) const [virtual protected]

在项目 index 的同一行，用 painter 绘制指定的 rect 矩形分支。

1. 9) void QTreeView::drawRow (QPainter *painter, const QStyleOptionViewItem & option, const QModelIndex & index) const [virtual protected]

用 painter 绘制新行，新行包含模型索引为 index 的项目，option 是如何显示项目。

1. 10) void QTreeView::drawTree (QPainter *painter, const QRegion & region) const [protected]

3.4.3 TreeView 控件（2）

用 painter 在区域 region 绘制树。

1. 11) void QTreeView::expand (const QModelIndex & index) [slot]

展开模型索引为 index 的项目。

1. 12) void QTreeView::expandAll () [slot]

展开所有的项目。

1. 13) void QTreeView::expandToDepth (int depth) [slot]

展开树形视图中的项目，深度为 depth。

1. 14) QHeaderView *QTreeView::header () const

返回该树形视图的 header。

1. 15) QModelIndex QTreeView::indexAbove (const QModelIndex & index) const

返回模型索引 index 的上一个索引。

1. 16) QModelIndex QTreeView::indexAt (const QPoint & point) const [virtual]

返回点 point 处项目的模型索引。

1. 17) QModelIndex QTreeView::indexBelow (const QModelIndex & index) const

返回模型索引 index 的下一个索引。

1. 18) bool QTreeView::isExpanded (const QModelIndex & index) const

如果模型索引 index 处的项目是展开着的，返回 true；否则返回 false。

1. 19) void QTreeView::rowsInserted (const QModelIndex & parent, int start, int end) [virtual protected]

插入新行，新行的父母是 parent，包括从 start 到 end 的所有项目。

1. 20) void QTreeView::rowsRemoved (const QModelIndex & parent, int start, int end) [protected slot]

删除行，行的父母是 parent，包括从 start 到 end 的所有项目。

1. 21) void QTreeView::selectAll () [virtual]

设置所有的项目都是选中状态。

1. 22) QModelIndexList QTreeView::selectedIndexes () const [virtual protected]

返回所有选中和非隐藏的项目的模型索引。

1. 23) void QTreeView::setHeader (QHeaderView *header)

设置该 TreeView 的标题为 header。

1. 24) void QTreeView::sortByColumn (int column, QT::SortOrder order)

对列 column 按 order 进行排序。

3.4.4 示例 6：TreeView 的应用（1）

本示例讲解了 Qt 中树控件 QTreeView 的开发应用，对于 QTreeView 没有过多的讲解，接下来介绍具体如何实现。

首先建立标准的 Qt Gui Application 项目，把项目自动生成的 toolBar、menuBar 和 statusBar 删除，设计界面如图 3-34 所示，其中从左到右依次是 TreeView、TextEdit、TextEdit 控件。

1. 控件说明

在属性编辑窗口中对控件的属性进行修改，修改内容如表 3-9 所示。

图 3-34　示例 6 界面

表 3-9　主要控件说明

控件类型	控件名称	控件说明
QTreeView	treeView	树列表
QTextEdit	textEdit	显示树列表中的所有内容
QTextEdit	textEdit_2	显示当前选中的内容

2. 示例说明

程序执行后，会通过不同方式向 treeView 中添加选项，示例功能说明如下：

textEdit 显示树列表中的所有选项内容；

textEdit_2 显示当前选中的树列表中的内容。

3. 示例实现

头文件 mainwindow.h（文中的粗体为需要添加的内容）：

```
1. #ifndef MAINWINDOW_H
2. #define MAINWINDOW_H
3. #include <QMainWindow>
4. #include <QStandardItem>
```

```
#include <QStandardItemModel>
namespace Ui {
  class MainWindow;
}
class MainWindow : public QMainWindow
{
  Q_OBJECT
public:
  explicit MainWindow(QWidget *parent = 0);
  ~MainWindow();
  void iterateOverItems();  //在 textEdit 中显示所有的 Item
  QList<QStandardItem*> returnTheItems();//返回列表中所有的 Item
private:
  Ui::MainWindow *ui;
  QStandardItemModel *model;//声明 model
private slots:
  void on_treeView_doubleClicked(QModelIndex index);//槽函数
};
#endif // MAINWINDOW_H
```

注意此处的槽函数是通过右击控件→Go to slots 自动生成的，所以不用在实现文件中添加映射函数。在主窗体 mainwindow.cpp 文件中自动生成如下代码：

```
#include "mainwindow.h"
#include "ui_mainwindow.h"
MainWindow::MainWindow(QWidget *parent) :
  QMainWindow(parent),
  ui(new Ui::MainWindow)
{
  ui->setupUi(this);
}


MainWindow::~MainWindow()
{
  delete ui;
}
```

3.4.4 示例 6：TreeView 的应用（2）

在主窗体 mainwindow.cpp 文件中添加的头文件：

```
#include <QModelIndex>
```

在主窗体 mainwindow.cpp 文件中构造函数：

```
/***构造函数***/
MainWindow::MainWindow(QWidget *parent) :
  QMainWindow(parent),
  ui(new Ui::MainWindow)
{
  ui->setupUi(this);

  ui->treeView->setEditTriggers(QAbstractItemView::NoEditTriggers);
ui->treeView->header()->setResizeMode(QHeaderView::ResizeToContents);


/************************new model*********************************/

  model = new QStandardItemModel(4,1);

  /************************设置表头
*********************************/
  model->setHeaderData(0, QT::Horizontal, tr("First"));

  /************************添加 Item 到model 中**********************/
  QStandardItem *item1 = new QStandardItem("item1");
  QStandardItem *item2 = new QStandardItem("item2");
  QStandardItem *item3 = new QStandardItem("item3");
  QStandardItem *item4 = new QStandardItem("item4");
  model->setItem(0, 0, item1);
  model->setItem(1, 0, item2);
  model->setItem(2, 0, item3);
  model->setItem(3, 0, item4);
  QStandardItem *item5 = new QStandardItem("item5");
  item4->appendRow(item5);
  QStandardItem *item6 = new QStandardItem("item6");
  item5 ->appendRow(item6);

  /***************************向 model 中插入
Item************************/
  QModelIndex modelIndex;
  for(int i = 0; i < 4; ++i)
  {
modelmodelIndex = model->index(0, 0, modelIndex);
model->insertRows(0, 1, modelIndex);
model->insertColumns(0, 1, modelIndex);
QModelIndex index = model->index(0, 0, modelIndex);
```

```
39. model->setData(index, i);
40.  }
41.  ui->treeView->setModel(model);
42.  iterateOverItems();
43. }
```

在主窗体 mainwindow.cpp 文件中添加槽函数 on_treeView_doubleClicked(QModelIndex index)：

```
1.  /***槽函数：处理双击 Item 事件***/
2.  void MainWindow::on_treeView_doubleClicked(QModelIndex index)
3.  {
4.    QModelIndex index0 = ui->treeView->currentIndex();
5.
    ui->textEdit_2->append(index0.data().toString()+" is CurrentItem!");
6.  }
```

在主窗体 mainwindow.cpp 文件中添加成员函数 returnTheItems()：

```
1.  /***成员函数：返回列表中所有的 Item***/
2.  QList<QStandardItem*> MainWindow::returnTheItems()
3.  {
4.
    return model->findItems("*", QT::MatchWildcard| QT::MatchRecursive);
5.  }
```

在主窗体 mainwindow.cpp 文件中添加成员函数 iterateOverItems()：

```
1.  /***成员函数：显示所有 Item***/
2.  void MainWindow::iterateOverItems()
3.  {
4.    QList<QStandardItem*> list = returnTheItems();
5.    foreach (QStandardItem*item, list)
6.    {
7.  ui->textEdit->setText(ui->textEdit->toPlainText()+item->text()+"\
    n");
8.    }
9.  }
```

主文件 main.cpp 不用进行任何更改，使用项目自动生成的即可。

4. 示例执行结果

示例执行结果如图 3-35 所示。

图 3-35　示例 6 执行结果

3.4.5　TableView 控件（1）

1. 控件位置

Item Views→TableView

2. 控件介绍

TableView 控件（表格视图）是一个模型/视图结构的表视图实现，用来显示模型的项目。它提供了 QTable 类提供的标准表格，但使用 Qt 的模型/视图结构具有更灵活的方式，而且也是模型/视图类之一，是 Qt 的模型/视图框架的一部分。由 QAbstractItemView 类定义的接口来实现，使其能够显示由 QAbstractItemModel 类派生的模型提供的数据。

3. 控件设置选项

在 TableView 控件的 properties 选项中，一般常对以下选项进行设置。

name：该控件对应源代码中的名称；

font：设置表格内部的字体；

cornerButtonEnabled：左上角的按钮是否有用；

gridStyle：表格的格式；

showGrid：是否显示网格，值为 true，显示，否则不显示；

sortingEnabled：是否对项目排序。

4.常用成员函数

1. 1) QTableView::QTableView (QWidget *parent = 0)

构造一个父对象为 parent 的 TableView。

1. 2) void QTableView::clearSpans ()

删除该 TableView 中的所有行和列的跨度。

1. 3) int QTableView::columnAt (int x) const

返回坐标 x 处的列，如果坐标处没有项目则返回-1。

1. 4) int QTableView::columnSpan (int row, int column) const

返回行 row、列 column 处的行跨度。

1. 5) void QTableView::currentChanged (const QModelIndex & current, const QModelIndex & previous) [virtual protected]

把 current 指定为当前项目，previous 是以前的当前项目。

1. 6) QHeaderView *QTableView::horizontalHeader () const

返回该 TableView 的水平标题。

1. 7) QModelIndex QTableView::indexAt (const QPoint & pos) const [virtual]

返回点 pos 处项目的模型索引。

1. 8) int QTableView::rowAt (int y) const

返回坐标 y 处的行，如果坐标处没有项目则返回-1。

1. 9) int QTableView::rowSpan (int row, int column) const

返回行 row、列 column 处的列跨度。

1. 10) void QTableView::selectColumn (int column) [slot]

3.4.5 TableView 控件（2）

设置列 column 为选中状态。

1. 11) void QTableView::selectRow (int row) [slot]

设置行 row 为选中状态。

1. 12) QModelIndexList QTableView::selectedIndexes () const [virtual protected]

返回所有选中和非隐藏的项目的模型索引。

1. 13) void QTableView::setHorizontalHeader (QHeaderView *header)

设置该 TableView 的水平标题为 header。

```
1. 14) void QTableView::setSpan ( int row, int column, int rowSpanCount,
   int columnSpanCount )
```

设置行 row、列 column 处的行跨度为 rowSpanCount、列跨度为 columnSpanCount。

```
1. 15) void QTableView::setVerticalHeader ( QHeaderView *header )
```

设置该 TableView 的垂直标题为 header。

```
1. 16) void QTableView::showColumn ( int column ) [slot]
```

显示列 column。

```
1. 17) void QTableView::showRow ( int row ) [slot]
```

显示行 row。

```
1. 18) QHeaderView *QTableView::verticalHeader () const
```

返回该 TableView 的垂直标题。

3.4.6 示例 7：TableView 的应用

TableView 一般和 QSqlTableModel 一起使用，来显示数据表项，这里只简单介绍 TableView 的用法，没有涉及数据库。

首先创建标准的 Qt Gui Application 项目，把项目自动生成的 toolBar、menuBar 和 statusBar 删除，设计界面如图 3-36 所示。

图 3-36 示例 7 界面

1.控件说明

在属性编辑窗口中对控件的属性进行修改，修改内容如表 3-10 所示。

表 3-10 主要控件说明

控件类型	控件名称	控件说明
QTableView	tableView	表格

2.示例说明

本示例只是简单地让 tableView 显示表格内容。

3.示例实现

头文件 mainwindow.h（文中的粗体为需要添加的内容）：

```
1. #ifndef MAINWINDOW_H
2. #define MAINWINDOW_H
3. #include <QMainWindow>
4. #include <QStandardItemModel>
5. namespace Ui {
6.   class MainWindow;
7. }
8. class MainWindow : public QMainWindow
9. {
10.  Q_OBJECT
11.public:
12.  explicit MainWindow(QWidget *parent = 0);
13.  ~MainWindow();
14.private:
15.  Ui::MainWindow *ui;
16.  QStandardItemModel *model;  //声明 model
17.  int size;//model 尺寸
18.};
19.#endif // MAINWINDOW_H
```

在主窗体 mainwindow.cpp 文件中自动生成如下代码：

```
1. #include "mainwindow.h"
2. #include "ui_mainwindow.h"
3. MainWindow::MainWindow(QWidget *parent) :
4.   QMainWindow(parent),
5.   ui(new Ui::MainWindow)
6. {
7.   ui->setupUi(this);
8. }
9. MainWindow::~MainWindow()
10.{
11.  delete ui;
```

```
12. }
```

在主窗体 mainwindow.cpp 文件中构造函数：

```
1.  /***构造函数***/
2.  MainWindow::MainWindow(QWidget *parent) :
3.    QMainWindow(parent),
4.    ui(new Ui::MainWindow)
5.  {
6.    ui->setupUi(this);
7.    size = 3;  //初始化变量
8.    model = new QStandardItemModel;
9.    model->setColumnCount(size);
10.   model->setRowCount(size);
11.
12.   /*************************设置表头标签
    ******************************/
13.   model->setHeaderData(0,QT::Horizontal,"ID");
14.   model->setHeaderData(1,QT::Horizontal,"User");
15.   model->setHeaderData(2,QT::Horizontal,"PassWd");
16.
17.   /************************填充
    model*********************************/
18.   for(int j=0;j<size;j++)
19.   {
20. QStandardItem *itemID = new QStandardItem(QString::number(j));
21. model->setItem(j,0,itemID);
22. QStandardItem *itemUser = new QStandardItem("author"+QString::number(j
    ));
23. model->setItem(j,1,itemUser);
24. QStandardItem *itemPassWd = new QStandardItem("123456");
25. model->setItem(j,2,itemPassWd);
26.   }
27.   ui->tableView->setModel(model);
28.   ui->tableView->verticalHeader()->hide();
29.   ui->tableView->setColumnWidth(0,30);
30.
    ui->tableView->setSelectionBehavior(QAbstractItemView::SelectRows);
31. }
```

主文件 main.cpp 采用该项目自动生成的即可。

4. 示例执行结果

示例执行结果如图 3-37 所示。

图 3-37　示例 7 执行结果

3.4.7　ColumnView 控件

1. 控件位置

Item Views→ColumnView

2. 控件介绍

ColumnView 控件中文称作“列视图”。

3. 控件设置选项

在 ColumnView 控件的 properties 选项中，一般常对以下选项进行设置。

name：该控件对应源代码中的名称；

font：设置表格内部的字体。

4. 常用成员函数

1. 1) QColumnView::QColumnView (QWidget *parent = 0)

构造一个父对象为 parent 的 ColumnView。

1. 2) QAbstractItemView *QColumnView::createColumn (const QModelIndex & index) [virtual protected]

index 是视图的根模型索引，返回新的视图。

1. 3) void QColumnView::currentChanged (const QModelIndex & current, const QModelIndex & previous) [virtual protected]

把 current 指定为当前项目，previous 是以前的当前项目。

1. 4) QModelIndex QColumnView::indexAt (const QPoint & point) co
   nst [virtual]

返回点 pos 处项目的模型索引。

1. 5) QWidget *QColumnView::previewWidget () const

返回预览组件，如果没有则返回 0。

1. 6) void QColumnView::rowsInserted (const QModelIndex
   & parent, int start, int end) [virtual protected]

插入新行，新行的父母是 parent，包括从 start 到 end 的所有项目。

1. 7) void QColumnView::selectAll () [virtual]

设置该 ColumnView 中的所用项目为选中状态。

1. 8) void QColumnView::setPreviewWidget (QWidget *widget)

设置 widget 为该 ColumnView 的预览组件。

ColumnView 的用法和以上三种基本相同，在此就不再举例说明。

3.5 单元组件(Item Widgets)

Qt Creator 有三种 Item Widgets，如图 3-38 所示。

图 3-38 Item Widgets 控件

Item Widgets 的 Qt 类和名称介绍如表 3-11 所示。

表 3-11 Item Widgets 介绍

控件类	控件名	中文名
QListWidget	List Widget	列表单元控件
QTreeWidget	Tree Widget	树形单元控件
QTableWidget	Table Widget	表格单元控件

本节主要介绍 Qt Creator 的 Item Widgets 的使用方法。

3.5.1 ListWidget 控件（1）

1. 控件位置

Item Widget→ListWidget

2. 控件介绍

ListWidget 控件（列表单元控件）继承于 QListview 类，样式如图 3-39 所示。与 ListView 不同的是，ListWidget 是基于 Item 的列表控件，采用的是传统的添加、删除项目。

图 3-39 ListWidget 控件

3. 控件设置选项

在 ListWidget 控件的 properties 选项中，一般常对以下选项进行设置。

name：该控件对应源代码中的名称；

font：设置表格内部的字体；

count：保存项目的数目；

currentRow：保存当前项目的行；

sortingEnabled：是否对 item 排序。

4. 常用成员函数

```
1. 1) QListWidget::QListWidget ( QWidget *parent = 0 )
```

构造一个父对象为 parent 的 ListWidget。

1. 2) void QListWidget::addItem (const QString & label)

添加一个新的项目，在新添加的项目中添加 label 标签。

1. 3) void QListWidget::addItem (QListWidgetItem *item)

添加项目 item。

1. 4) void QListWidget::addItems (const QStringList & labels)

添加一列项目。

1. 5) void QListWidget::clear () [slot]

清除该 ListWidget 中的所有项目。

1. 6) QListWidgetItem *QListWidget::currentItem () const

返回当前活动项目。

1. 7) void QListWidget::editItem (QListWidgetItem *item)

如果项目 item 是可编辑的，开始编辑项目 item。

1. 8) QList<QListWidgetItem *> QListWidget::findItems
   (const QString & text, QT::MatchFlags flags) const

查找匹配字符串 text 的项目，并返回查找结果。

1. 9) void QListWidget::insertItem (int row, QListWidgetItem *
   item)

在行 row 处插入项目 item。

1. 10) void QListWidget::insertItem (int row, const QString &
   label)

3.5.1 ListWidget 控件（3）

这是一个重载函数，函数功能同 9)，在行 row 处插入标签为 label 的新项目。

1. 11) void QListWidget::insertItems (int row, const QStringLi
   st & labels)

在行 row 处插入一列项目。

1. 12) QListWidgetItem *QListWidget::item (int row) const

返回行 row 处的项目，如果行 row 处没有项目则返回 0。

1. 13) QListWidgetItem *QListWidget::itemAt (const QPoint & p
) const

返回点 p 处的项目。

1. 14) QListWidgetItem *QListWidget::itemAt (int x, int y) co
   nst

返回坐标(x,y)处的项目。

```
1. 15) QWidget *QListWidget::itemWidget ( QListWidgetItem *item
    ) const
```

返回项目 item 处显示的控件。

```
1. 16) QListWidgetItem *QListWidget::takeItem ( int row )
```

移除行 row 处的项目，并返回项目控件。

```
1. 17) void QListWidget::removeItemWidget ( QListWidgetItem *it
   em )
```

删除项目 item 处的控件。

```
1. 18) int QListWidget::row ( const QListWidgetItem *item ) con
   st
```

返回项目 item 所在的行。

```
1. 19) QList<QListWidgetItem *> QListWidget::selectedItems () c
   onst
```

返回所有被选中的项目的控件。

```
1. 20) void QListWidget::setCurrentItem ( QListWidgetItem *item
    )
```

设置项目 item 为当前项目。

```
1. 21) void QListWidget::setItemWidget ( QListWidgetItem *item,
    QWidget *widget )
```

设置控件 widget 为项目 item 的显示控件。

```
1. 22) void QListWidget::setItemWidget ( QListWidgetItem *item,
    QWidget *widget )
```

这是个重载函数，函数功能同 23)。

```
1. 23) void QListWidget::sortItems ( QT::SortOrder order = QT::
   AscendingOrder )
```

把项目按照 order 进行排序。

3.5.2 TreeWidget 控件（1）

1. 控件位置

Item Widget→TreeWidget

2. 控件介绍

TreeWidget 控件（树形单元控件）继承于 QTreeView 类，样式如图 3-40 所示。TreeWidget 控件是树形视图使用预定义的模型，它也是基于模型/视图结构的控件。为方便开发人员使用树形视图，可以使用这个控件来创建简单的树形结构列表。

图 3-40　TreeWidget 控件

3. 控件设置选项

在 TreeWidget 控件的 properties 选项中，一般常对以下选项进行设置。

name：该控件对应源代码中的名称；

font：设置表格内部的字体；

columnCount：保存该 TreeWidget 的列数。

4. 常用成员函数

```
1. 1) QTreeWidget::QTreeWidget ( QWidget *parent = 0 )
```

构造一个父对象为 parent 的 TreeWidget。

```
2) void QTreeWidget::addTopLevelItem ( QTreeWidgetItem *item )
```

在该 TreeWidget 中追加 item 为顶级项目。

```
3) void QTreeWidget::addTopLevelItems ( const QList<QTreeWidgetItem *> & items )
```

把 items 中的项目作为顶级项目追加到该 TreeWidget 中。

```
4) void QTreeWidget::clear () [slot]
```

清除该 TreeWidget 中的所有项目。

```
5) void QTreeWidget::collapseItem ( const QTreeWidgetItem *item ) [slot]
```

折叠项目 item。

```
6) int QTreeWidget::currentColumn () const
```

返回当前活动列。

```
7) QTreeWidgetItem *QTreeWidget::currentItem () const
```

3.5.2 TreeWidget 控件（2）

返回当前活动项目。

```
8) void QTreeWidget::editItem ( QTreeWidgetItem *item, int column = 0 )
```

如果列 column 的 item 是可编辑的，开始编辑。

```
1. 9) void QTreeWidget::expandItem ( const QTreeWidgetItem *ite
   m ) [slot]
```

展开项目 item。

```
1. 10) QList<QTreeWidgetItem *> QTreeWidget::findItems ( const
   QString & text, QT::MatchFlags flags, int column = 0 ) const
```

查找匹配字符串 text 的项目，并返回查找结果。

```
1. 11) QTreeWidgetItem *QTreeWidget::headerItem () const
```

返回头项目。

```
1. 12) QModelIndex QTreeWidget::indexFromItem ( QTreeWidgetItem
    *item, int column = 0 ) const [protected]
```

返回列 column 的项目 item 的模型索引。

```
1. 13) int QTreeWidget::indexOfTopLevelItem ( QTreeWidgetItem *
   item ) const
```

返回顶级项目 item 的模型索引，如果 item 不存在则返回-1。

```
1. 14) int QTreeWidget::sortColumn () const
```

返回排序的列。

```
1. 15) void QTreeWidget::sortItems ( int column, QT::SortOrder
   order )
```

对列 column 的项目按照 order 进行排序。

```
1. 16) QTreeWidgetItem *QTreeWidget::itemAbove ( const QTreeWid
   getItem *item ) const
```

返回 item 的上一个项目。

```
1. 17) QTreeWidgetItem *QTreeWidget::itemAt ( const QPoint & p
   ) const
```

返回点 p 处的项目。

```
1. 18) QTreeWidgetItem *QTreeWidget::itemAt ( int x, int y ) co
   nst
```

返回坐标(x,y)处的项目。

```
1. 19) void QTreeWidget::setItemWidget ( QTreeWidgetItem *item,
    int column, QWidget *widget )
```

设置控件 widget 为项目 item 的显示控件，项目 item 在列 column 中。

```
1. 20) QTreeWidgetItem *QTreeWidget::itemBelow ( const QTreeWid
   getItem *item ) const
```

返回 item 的下一个项目。

```
1. 21) QWidget *QTreeWidget::itemWidget ( QTreeWidgetItem *item
   , int column ) const
```

返回列 column 中的项目 item 的显示控件。

```
1. 22) void QTreeWidget::removeItemWidget ( QTreeWidgetItem *it
   em, int column )
```

移除列 column 中的项目 item 的显示控件。

```
1. 23) QList<QTreeWidgetItem *> QTreeWidget::selectedItems () c
   onst
```

返回所有选中状态的项目。

```
1. 24) void QTreeWidget::setCurrentItem ( QTreeWidgetItem *item
    )
```

3.5.2 TreeWidget 控件（3）

设置项目 item 为当前项目。

```
1. 25) void QTreeWidget::setCurrentItem ( QTreeWidgetItem *item
   , int column )
```

设置列 column 的项目 item 为当前项目。

```
1. 26) void QTreeWidget::setHeaderItem ( QTreeWidgetItem *item
   )
```

设置 item 为该 TreeWidget 的头项目。

```
1. 27) void QTreeWidget::setHeaderLabel ( const QString & label
    )
```

设置 label 为头标题。

```
1. 28) QTreeWidgetItem *QTreeWidget::topLevelItem ( int index )
    const
```

返回所有 index 的顶级项目。

3.5.3 TableWidget 控件（1）

1.控件位置

Item Widget→TableWidget

2.控件介绍

TableWidget 控件（表格单元控件）的样式如图 3-41 所示。

图 3-41　TableWidget 控件

3.控件设置选项

在 TableWidget 控件的 properties 选项中，一般常对以下选项进行设置。

name：该控件对应源代码中的名称；

font：设置表格内部的字体；

columnCount：保存列的数目；

rowCount：保存行的数目。

4.常用成员函数

```
1. 1) QTableWidget::QTableWidget ( QWidget *parent
   = 0 )
```

构造一个父对象为 parent 的 TableWidget。

```
1. 2) QTableWidget::QTableWidget ( int rows, int co
   lumns, QWidget *parent = 0 )
```

构造一个 rows 行、colums 列、父对象为 parent 的 TableWidget 控件。

```
1. 3) QWidget *QTableWidget::cellWidget ( int row,
   int column ) const
```

返回行 row、列 column 的单元格处的控件。

```
1. 4) void QTableWidget::clear () [slot]
```

删除该 TreeWidget 中的所有项目。

```
1. 5) void QTableWidget::clearContents () [slot]
```

删除该 TreeWidget 中的除了 header 外的所有项目。

1.6) int QTableWidget::column (const QTableWidget Item *item) const

返回项目 item 所在的列。

1.7) int QTableWidget::currentColumn () const

返回当前活动的列。

1.8) QTableWidgetItem *QTableWidget::currentItem ( ) const

返回当前活动的项目。

1.9) int QTableWidget::currentRow () const

返回当前活动的行。

1.10) void QTableWidget::editItem (QTableWidgetIt em *item)

如果 item 是可编辑的，开始编辑 item。

1.11) QList<QTableWidgetItem *>QTableWidget::findI tems(const QString & text,QT::MatchFlags flags) const

查找匹配字符串 text 的项目，并返回查找结果。

1.12) void QTableWidget::insertColumn (int column) [slot]

在列 column 处插入新列。

1.13) void QTableWidget::insertRow (int row) [s lot]

在行 row 处插入新行。

1.14) QTableWidgetItem *QTableWidget::item (int r ow, int column) const

返回行 row、列 column 处的项目。

1.15) QTableWidgetItem *QTableWidget::itemAt (con st QPoint & point) const

3.5.3 TableWidget 控件（2）

返回点 point 处的项目。

1.16) QTableWidgetItem *QTableWidget::itemAt (int ax, int ay) const

返回坐标(ax,ay)处的项目。

1.17) void QTableWidget::removeCellWidget (int row, int column)

移除行 row、列 column 单元格处的显示控件。

1.18) void QTableWidget::removeColumn (int column) [slot]

移除列 column。

1.19) void QTableWidget::removeRow (int row) [slot]

移除行 row。

1.20) int QTableWidget::row (const QTableWidgetItem *item) const

返回 item 的行。

1.21) QList<QTableWidgetItem *> QTableWidget::selectedItems ()

返回所有选中状态的项目。

1.22) void QTableWidget::setCellWidget (int row, int column, QWidget *widget)

设置行 row、列 column 处的显示控件为 widget。

1.23) void QTableWidget::setCurrentCell (int row, int column)

设置行 row、列 column 处的单元格为当前活动单元格。

```
1. 24) void QTableWidget::setCurrentItem ( QTableWi
   dgetItem *item )
```

设置项目 item 为当前活动项目。25) void

QTableWidget::setHorizontalHeaderItem (int column,

QTableWidgetItem *item)设置项目 item 为列 column 的水平头项目。功能同 setVerticalHeaderItem()。

```
1. 26) void QTableWidget::setHorizontalHeaderLabels
     ( const QStringList & labels )
```

设置水平标题为 labels。功能同 seVerticalHeaderLabels()。

```
1. 27) void QTableWidget::setItem ( int row, int co
   lumn, QTableWidgetItem *item )
```

设置行 row、列 column 的单元格的项目为 item。

```
1. 28) void QTableWidget::sortItems ( int column, Q
   T::SortOrder order = QT::AscendingOrder )
```

对列 column 按照 order 进行排序。

```
1. 29) QTableWidgetItem *QTableWidget::takeHorizont
   alHeaderItem ( int column )
```

移除列 column 的水平头项目。功能同 takeVerticalHeaderItem()。

```
1. 30) QTableWidgetItem *QTableWidget::takeItem ( i
   nt row, int column )
```

移除行 row、列 column 单元格处的项目。

```
1. 31) QTableWidgetItem *QTableWidget::verticalHead
   erItem ( int row ) const
```

返回行 row 的垂直头项目。

3.5.4 示例 8：TableWidget 的示例

QTableWidget 是常用的显示数据表格的控件，类似于 VC、C#中的 DataGrid。QTableWidget 是 QTableView 的子类，两者的区别是，QTableView 可以使用自定义的数据模型来显示内容（也就是先要通过 setModel 来绑定数据源），而 QTableWidget 则只能使用标准的数据模型，并且其单元格数据是由 QTableWidgetItem 的对象来实现的（也就是不需要数据源，将逐个单元格内的信息填好即可）。

首先创建标准的 Qt Gui Application 项目，把项目在主界面中自动生成的 toolBar、menuBar 和 statusBar 删除，设计界面如图 3-42 所示。

图 3-42　示例 8 界面

1.控件说明

在属性编辑窗口中对控件的属性进行修改，修改内容如表 3-12 所示。

2.示例说明

表 3-12　主要控件说明

控件类型	控件名称	控件说明
QTableWidget	tableWidget	用来显示表格内容

本示例具体介绍怎样向 tableWidget 中添加文本和控件。

3.示例实现

头文件 mainwindow.h（文中的粗体为需要添加的内容）：

```
1.#ifndef MAINWINDOW_H
2.#define MAINWINDOW_H
3.#include <QMainWindow>
4.namespace Ui {
5.  class MainWindow;
6.}
7.class MainWindow : public QMainWindow
8.{
9.  Q_OBJECT
10.
11.
12.public:
13.  explicit MainWindow(QWidget *parent = 0);
14.  ~MainWindow();
15.private:
16.  Ui::MainWindow *ui;
17.};
18.#endif // MAINWINDOW_H
```

头文件采用 Qt Creator 自动生成的即可，不需要任何更改。

在主窗体 mainwindow.cpp 文件中自动生成如下代码：

```
#include "mainwindow.h"
#include "ui_mainwindow.h"
MainWindow::MainWindow(QWidget *parent) :
  QMainWindow(parent),
  ui(new Ui::MainWindow)
{
  ui->setupUi(this);
}
MainWindow::~MainWindow()
{
   delete ui;
}
```

在主窗体 mainwindow.cpp 文件中添加头文件：

```
#include <QLabel>
#include <QDateTimeEdit>
#include <QComboBox>
#include <QSpinBox>
```

在主窗体 mainwindow.cpp 文件中构造函数：

```
/***构造函数***/
MainWindow::MainWindow(QWidget *parent) :
  QMainWindow(parent),
  ui(new Ui::MainWindow)
{
  ui->setupUi(this);

  /***************添加
  QLabel*****************************************
  **/
  QLabel *label = new QLabel();
   label->setText("Jhon");
   ui->tableWidget->setCellWidget(1,0,label);

```

```
13.   /**************添加
    QComboBox*****************************************
    **/
14.   QComboBox *ComboBoxSex = new QComboBox();
15.   ComboBoxSex->addItem("Man");
16.   ComboBoxSex->addItem("Woman");
17.
    ui->tableWidget->setCellWidget(1,1,ComboBoxSex);

18.
19.   /**************添加
    QDateTimeEdit*************************************
    ****/
20.
    QDateTimeEdit *dateTimeEdit1 = new QDateTimeEdit
    ();
21.
    dateTimeEdit1->setDateTime(QDateTime::currentDat
    eTime());
22.
    dateTimeEdit1->setDisplayFormat("dd/MM/yyyy");
23.   dateTimeEdit1->setCalendarPopup(true);
24.
    ui->tableWidget->setCellWidget(1,2,dateTimeEdit1
    );
25.
26.
27.
28.   /**************添加
    QComboBox*****************************************
    **/
```

```
29.   QComboBox *ComboBoxWork1 = new QComboBox();
30.   ComboBoxWork1->addItem(tr("Worker"));
31.   ComboBoxWork1->addItem(tr("Farmer"));
32.   ComboBoxWork1->addItem(tr("Doctor"));
33.   ComboBoxWork1->addItem(tr("Lawyer"));
34.   ComboBoxWork1->addItem(tr("Solder"));
35.
      ui->tableWidget->setCellWidget(1,3,ComboBoxWork1
      );
36.
37.   /****************添加
      QSpinBox*****************************************
      /
38.   QSpinBox *spinboxIncome1 = new QSpinBox();
39.   spinboxIncome1->setRange(1000,10000);
40.
      ui->tableWidget->setCellWidget(1,4,spinboxIncome
      1);
41. }
```

主文件 main.cpp 不需任何更改，使用项目自动生成的即可。

4.示例执行结果

示例执行结果如图 3-43 所示。

图 3-43　示例 8 执行结果

3.6　容器(Containers)

Qt Creator 有 9 种 Containers Widgets，如图 3-44 所示。

Containers Widgets 的 Qt 类和名称介绍如表 3-13 所示。

图 3-44　Containers 控件

表 3-13　Containers Widgets 介绍

控件类	控件名	中文名
QGroupBox	GroupBox	组合框
QScrollArea	ScrollArea	滚动区
QToolBox	ToolBox	工具箱
QTabWidget	TabWidget	切换卡
QWidgetStack	StackedWidget	控件栈
QFrame	Frame	框架
QWidget	Widget	组件
QMdiArea	MdiArea	MDI 窗口显示区
QDockWidget	DockWidget	停靠窗体

本节主要介绍 Qt Creator 的 Containers Widgets 的使用方法。Qt 提供了很多 Containers 类型的控件，有十种之多。

3.6.1 GroupBox 控件

1.控件位置

Containers→GroupBox

2.控件介绍

GroupBox 控件（组合框）的样式如图 3-45 所示。使用 GroupBox 的好处就是用户可以比较清晰地了解程序的界面。我们可以把与功能相关的空间放到一个 GroupBox 中，用户可以方便地找到相应的功能控件。

图 3-45 GroupBox 控件

3.控件设置选项

在 GroupBox 控件的 properties 选项中，一般常对以下选项进行设置。

name：该控件对应源代码中的名称；

title：该控件对应图形界面中所显示的名称；

font：设置 title 的字体。

由于 GroupBox 控件非常简单，这里不再过多介绍，后面会在具体示例中向大家介绍实践用法。

3.6.2 ScrollArea 控件

1. 控件位置

Containers→ScrollArea

2. 控件介绍

ScrollArea 控件（滚动区）是一个用来显示子控件的内容的框架。如果子控件的尺寸超过了框架的大小，可以提供滚动条，方便查看整个子控件。

3. 控件设置选项

在 ScrollArea 控件的 properties 选项中，一般常对以下选项进行设置。

name：该控件对应源代码中的名称；

font：设置文本框的字体。

4. 常用成员函数

```
1. 1) QScrollArea::QScrollArea ( QWidget *parent =
   0 )
```

构造一个父对象为 parent 的 ScrollArea。例如：

```
1. QScrollArea *scrollarea = new QScrollArea(this);
```

```
2. 2) void QScrollArea::setWidget ( QWidget *widget )
```

设置控件 widget 为该 ScrollArea 的子控件。

```
1. 3) QWidget *QScrollArea::takeWidget ()
```

删除该 ScrollArea 的子控件。

```
1. 4) QWidget *QScrollArea::widget () const
```

返回该 ScrollArea 的子控件。

3.6.3 示例 9：GroupBox 和 ScrollArea 的示例（1）

由于 GroupBox 和 ScrollArea 使用起来很简单，这里用一个示例来介绍这两个控件的使用方法。

首先创建标准的 Qt Gui Application 项目，把项目在主界面中自动生成的 toolBar、menuBar 和 statusBar 删除，设计界面如图 3-46 所示。

1. 控件说明

在属性编辑窗口中对控件的属性进行修改，修改内容如表 3-14 所示。

图 3-46　示例 9 界面

表 3-14　主要控件说明

控件类型	控件名称	控件说明
QGroupBox	GroupBox	装饰说明
QScrollArea	ScrollArea	承载 label
QLabel	Label	显示图片
QPushButton	BtnOpen	打开图片文件

2.示例说明

本示例中显示图片的 label 是自定义的，示例功能说明：

单击“打开图片”按钮，程序就会按照图片原来的尺寸打开这张图片，如果窗体不够大，就会出现滚动条。

3.示例实现

头文件 mainwindow.h（文中的粗体为需要添加的内容）：

```
1.#ifndef MAINWINDOW_H
2.#define MAINWINDOW_H
3.#include <QMainWindow>
4.#include <QLabel>
5.namespace Ui {
6.  class MainWindow;
7.}
8.class MainWindow : public QMainWindow
9.{
10.  Q_OBJECT
```

```
11. public:
12.   explicit MainWindow(QWidget *parent = 0);
13.   ~MainWindow();
14. private:
15.   Ui::MainWindow *ui;
16.   QLabel *label ;
17. private slots:
18. void on_btnOpen_clicked();
19. };
20. #endif // MAINWINDOW_H
```

在头文件中声明一个 Label 和一个槽函数 on_btnOpen_clicked()，Label 的名称是 label，用来添加到 ScrollArea 中显示的图片。此处的槽函数是通过右击控件→Go to slots 自动生成的，所以不用在实现文件中添加映射函数。

在主窗体 mainwindow.cpp 文件中自动生成如下代码：

```
1. #include "mainwindow.h"
2. #include "ui_mainwindow.h"
3. MainWindow::MainWindow(QWidget *parent) :
4.   QMainWindow(parent),
5.   ui(new Ui::MainWindow)
6. {
7.   ui->setupUi(this);
8. }
9. MainWindow::~MainWindow()
10. {
11.   delete ui;
12. }
```

在主窗体 mainwindow.cpp 文件中添加的头文件：

```
#include <QFileDialog>
#include <QMessageBox>
```

在主窗体 mainwindow.cpp 文件中构造函数：

```
/***构造函数***/
MainWindow::MainWindow(QWidget *parent) :
  QMainWindow(parent),
  ui(new Ui::MainWindow)
{
  ui->setupUi(this);
  label = new QLabel;//new label
  ui->scrollArea->setWidget(label);//设置 label 处于滚动区内
}
```

3.6.3 示例 9：GroupBox 和 ScrollArea 的示例（2）

在主窗体 mainwindow.cpp 文件中添加槽函数 on_btnOpen_clicked()：

```
/***槽函数：打开图片***/
void MainWindow::on_btnOpen_clicked()
{

  QString fileName = QFileDialog::getOpenFileName(this,
tr("Open File"), QDir::currentPath());//获取文件
  if(!fileName.isEmpty())//文件不为空
  {
QImage image(fileName);
if(image.isNull())
{
```

```
11. QMessageBox::information(this, tr("Image Viewer
    "),
12. tr("Cannot load %1.").arg(fileName));
13. return;
14. }
15. label->setPixmap(QPixmap::fromImage(image));//
    显示图片
16.   }
17. }
```

主文件 main.cpp 不用任何更改，使用项目自动生成的即可。

4. 示例执行结果

示例执行结果如图 3-47 所示。

图 3-47　示例 9 执行结果

3.6.4　ToolBox 控件

1. 控件位置

Containers→ToolBox

2.控件介绍

ToolBox 控件（工具箱）的样式如图 3-48 所示。Toolbox 提供了一系列的页和隔间，就像 Qt Designer 和 Qt Creator 中的工具箱一样。

图 3-48　ToolBox 控件

3.控件设置选项

在 ToolBox 控件的 properties 选项中，一般常对以下选项进行设置。

name：该控件对应源代码中的名称；

font：设置该控件内部文本的字体；

currentIndex：当前活动页的索引；

itemLabel：当前活动页的标签；

itemName：当前活动页的名称；

itemBackgroundMode：当前活动页的背景模式。

4.常用成员函数

1.1) QToolBox::QToolBox (QWidget *parent = 0, const char *name = 0,WFlags f = 0)

构造一个名称为 name、父对象为 parent 和标志为 f 的 ToolBox。

1.2) int QToolBox(QWidget *item,const QIconSet & iconSet,const QString & label)

增加一个 item 到 ToolBox 的底部,新增加的 item 的标签的文本是 label，标签的图标是 iconSet。

1.3) int QToolBox(QWidget *item, const QString & label)

增加一个 item 到 ToolBox 的底部,新增加的 item 的标签的文本是 label。

1.4) int QToolBox::count() const

返回该工具箱中 item 的数目。

1.5) int QToolBox::currentIndex() const

返回当前活动 item 的索引。

1.6) QWidget *QToolBox::currentItem() const

返回当前活动 item，如果该 ToolBox 为空，返回 0。

1.7) int QToolBox::indexof (QWidget *item) const

返回 item 的索引。

1.8) int QToolBox::insertItem(int index,QWidget *item,const QIconSet & iconSet,const QString & label)

在索引 index 处插入一个新的项目，项目是 item，标签图标是 iconSet，标签文本是 label，返回插入 item 的索引。

1.9) int QToolBox::insertItem(int index,QWidget *item, const QString & label)

在索引 index 处插入一个新的项目，项目是 item，标签文本是 label，返回插入 item 的索引。

1.10) QWidget *QToolBox::item (int index) const

返回索引 index 位置的 item。

1.11) QString QToolBox::itemLabel(int index) const

返回索引 index 位置的标签。

1.12) int QToolBox::RemoveItem(QWidget *item)

删除该 ToolBox 中的 item 项目，删除成功后返回 item 的索引，否则返回-1。

1.13) void QToolBox::setCurrentIndex(int index)

设置索引 index 位置的项目为当前活动项目。

1.14) void QToolBox::setCurrentIndex(QWidget *item)

设置索引 item 为当前活动项目。

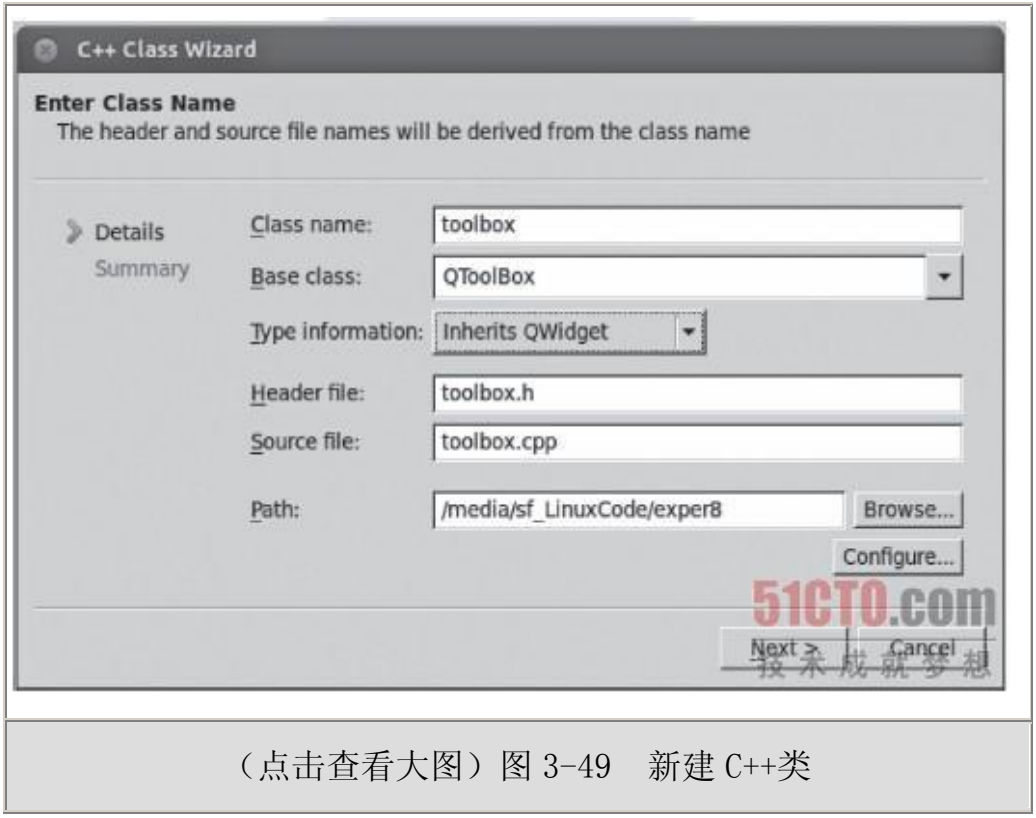
1.15) void QToolBox::setItemLabel (int index,const QString & label)

设置 label 为索引 index 位置的项目的标签文本。

3.6.5 示例 10：ToolBox 的应用

在这个示例中，我们只是简单地设计了 Qt Creator 中的工具箱的模型，没有实际操作功能。

首先建立一个空项目，在空项目中添加新建 C++Class，新添加的 C++Class 配置如图 3-49 所示。



（点击查看大图）图 3-49　新建 C++类

1.示例说明

本示例只是实现了类似于 Qt Creator 控件栏的外形，没有实际操作功能，目的是让大家学会怎样向 ToolBox 中添加控件和在 ToolBox 中对控件进行布局。

2.示例实现

头文件 toolbox.h（文中的粗体为需要添加的内容）：

```
1. #ifndef TOOLBOX_H
2. #define TOOLBOX_H
3. #include <QToolBox>
4. #include <QToolButton>
5. class toolbox : public QToolBox
6. {
7.   Q_OBJECT
8. public:
9.   explicit toolbox(QWidget *parent = 0);
10.
11.
12.   QToolButton *toolButton1_1; //声明多个
    QToolButton 类对象
13.   QToolButton *toolButton1_2;
14.   QToolButton *toolButton1_3;
15.   QToolButton *toolButton1_4;
16.   QToolButton *toolButton2_1;
17.   QToolButton *toolButton2_2;
18.   QToolButton *toolButton3_1;
19.   QToolButton *toolButton3_2;
20.   QToolButton *toolButton3_3;
21.   QToolButton *toolButton3_4;
22. signals:
23. public slots:
24. };
25. #endif // TOOLBOX_H
```

在头文件中声明多个 ToolButton 控件。

在类实现文件 toolbox.cpp 中自动生成如下代码：

```
1. #include "toolbox.h"
2. toolbox::toolbox(QWidget *parent) :
3.   QToolBox(parent)
```

```
4. {
5. }
```

在主窗体 toolbox.cpp 文件中添加的头文件：

```
1. #include <QGroupBox>
2. #include <QVBoxLayout>
3. #include <QPushButton>
```

在主窗体 toolbox.cpp 文件中构造函数：

```
1. /***构造函数***/
2. toolbox::toolbox(QWidget *parent) :
3.   QToolBox(parent)
4. {
5. /************定义 QGroupBox***********/
6. QGroupBox *groupBox1 = new QGroupBox;
7. QGroupBox*groupBox2 = new QGroupBox();
8. QGroupBox*groupBox3 = new QGroupBox();
9.
10. /************Layout 的 QToolButton***********/
11. this->toolButton1_1 = new QToolButton;
12. this->toolButton1_1->setText(tr("Vertical Layou
    t"));
13. this->toolButton1_1->setAutoRaise(true);
14. this->toolButton1_2 = new QToolButton;
15. this->toolButton1_2->setText(tr("Horizontal Lay
    out"));
16. this->toolButton1_2->setAutoRaise(true);
17. this->toolButton1_3 = new QToolButton;
18. this->toolButton1_3->setText(tr("Grid Layout"))
    ;
19. this->toolButton1_3->setAutoRaise(true);
```

```
20. this->toolButton1_4 = new QToolButton;
21. this->toolButton1_4->setText(tr("Form Layout"))
    ;
22. this->toolButton1_4->setAutoRaise(true);
23.
24. /*************Spacers 的
    ToolButton***************/
25. this->toolButton2_1 = new QToolButton;
26.
27.
28. this->toolButton2_1->setText(tr("Horizontal Spa
    cer"));
29. this->toolButton2_1->setAutoRaise(true);
30. this->toolButton2_2 = new QToolButton;
31. this->toolButton2_2->setText(tr("Vertical Space
    r"));
32. this->toolButton2_2->setAutoRaise(true);
33.
34. /*************Buttons*************************/

35. this->toolButton3_1 = new QToolButton;
36. this->toolButton3_1->setText(tr("Push Button"))
    ;
37. this->toolButton3_1->setAutoRaise(true);
38. this->toolButton3_2 = new QToolButton;
39. this->toolButton3_2->setText(tr("Tool Button"))
    ;
40. this->toolButton3_2->setAutoRaise(true);
41. this->toolButton3_3 = new QToolButton;
42. this->toolButton3_3->setText(tr("Radio Button")
    );
43. this->toolButton3_3->setAutoRaise(true);
44. this->toolButton3_4 = new QToolButton;
45. this->toolButton3_4->setText(tr("Check Box"));
```

```
46. this->toolButton3_4->setAutoRaise(true);
47.
48. /************给 Layout 布局
    ************************/
49. QVBoxLayout*layout1 = new QVBoxLayout(groupBox1
    );
50. layout1->setMargin(10);
51. layout1->setAlignment(QT::AlignLeft);
52. layout1->addWidget(toolButton1_1);
53. layout1->addWidget(toolButton1_2);
54. layout1->addWidget(toolButton1_3);
55. layout1->addWidget(toolButton1_4);
56. layout1->addStretch();
57.
58. /************给 Spacers 布局
    ************************/
59. QVBoxLayout*layout2 = new QVBoxLayout(groupBox2
    );
60. layout2->setMargin(10);
61. layout2->setAlignment(QT::AlignLeft);
62. layout2->addWidget(toolButton2_1);
63. layout2->addWidget(toolButton2_2);
64. layout2->addStretch();
65.
66. /************给 Buttons 布局
    ************************/
67. QVBoxLayout*layout3 = new QVBoxLayout(groupBox3
    );
68. layout3->setMargin(10);
69. layout3->setAlignment(QT::AlignLeft);
70. layout3->addWidget(toolButton3_1);
71. layout3->addWidget(toolButton3_2);
72. layout3->addWidget(toolButton3_3);
73. layout3->addWidget(toolButton3_4);
```

```
74. addItem(groupBox1, tr("Layouts"));
75. addItem(groupBox2, tr("Spacers"));
76. addItem(groupBox3, tr("Buttons"));
77. }
```

主文件 main.cpp：

```
1. #include <QTGui/QApplication>
2. #include "toolbox.h"
3. int main(int argc, char *argv [] )
4.
5.
6. {
7.   QApplication a(argc, argv);
8.   toolbox w;
9.   w.show();
10.   return a.exec();
11. }
```

3.示例执行结果

示例执行结果如图 3-50 所示。

图 3-50　示例 10 执行结果

3.6.6　TabWidget 控件（1）

1.控件位置

Containers→TabWidget

2.控件介绍

TabWidget 控件（切换卡）的样式如图 3-51 所示。切换卡控件顶部或底部有一个标签选项栏，每个标签选项都有一个页面。选择哪个页面，只须单击对应的标签选项即可，或按指定的【ALT+字母】快捷键组合即可。

TabWidget 的工作方式与前面的控件有一些区别。简单地说，TabWidget 控件只是用来显示页面的标签页的容器。

图 3-51　TabWidget 控件

3.控件设置选项

在 TabWidget 控件的 properties 选项中，一般常对以下选项进行设置。

name：该控件对应源代码中的名称；

currentPage：当前活动的页面；

margin：页面边框的空白宽度，默认是 0；

tabShap：标签选项的模式；

pageName：当前活动页的名称；

pageTitle：当前活动页的标签文本。

4.常用成员函数

1. 1) QTabWidget::QTabWidget (QWidget *parent = 0, const char *name = 0, WFlags f = 0)

构造一个名称为 name、父对象为 parent 和标记为 f 的 TabWidget。

1. 2) void QTabWidget::addTab (QWidget *child, const QString & label) [virtual]

增加子页到该 TabWidget，子页控件是 child，子页标签文本是 label。

1. 3) void QTabWidget::addTab(QWidget *child, const QIconSet & iconset, const QString & label) [virtual]

这是一个重载成员函数,功能同函数 2),只是增加了一个 iconset,iconset 是图标集。

1. 4) QString QTabWidget::tabLabel (QWidget *w) const

返回索引 index 处子页的选项标签文本。

1.5) void QTabWidget::changeTab (QWidget *w, const QString & label)

更改子页 w 的标签文本为 label。

1.6) void QTabWidget::changeTab (QWidget *w, const QIconSet & iconset, const QString & label)

更改子页 w 的图标为 iconset 和更改标签文本为 label。

1.7) int QTabWidget::count () const

3.6.6 TabWidget 控件（2）

返回该 TabWidget 中子页的数目。

1.8) QWidget *QTabWidget::currentPage () const

返回当前活动子页。

1.9) int QTabWidget::currentPageIndex () const

返回当前活动子页的索引。

1.10) int QTabWidget::indexOf (QWidget *w) const

返回子页 w 的索引。

1.11) void QTabWidget::insertTab (QWidget *child, const QString & label, int index = -1) [virtual]

在索引 index 处插入新的子页，子页控件是 child，子页标签文本是 label。

注意：在插入新的子页时要确保插入的子页名与标签文本与 TabWidget 中

的所有子页不同。如果指定 index，就是在指定的位置插入，否则就和简单的添加一样。

```
1. 12) void QTabWidget::insertTab ( QWidget *child,
    const QIconSet & iconset, const QString & label
   , int index = -1 ) [virtual]
```

在索引 index 处插入新的子页,子页控件是 child,子页标签文本是 label,子页图标为 iconset。

```
1. 13) QString QTabWidget::label ( int index ) cons
   t
```

返回索引 index 处子页的选项标签。

```
1. 14) QWidget *QTabWidget::page ( int index ) cons
   t
```

返回索引 index 处子页。

```
1. 15) void QTabWidget::removePage ( QWidget *w )
    [virtual slot]
```

删除子页 w。

```
1. 16) void QTabWidget::setCurrentPage ( int index
   ) [slot]
```

设置索引 index 处子页为当前活动页。

```
1. 17) void QTabWidget::setTabLabel ( QWidget *w, c
   onst QString & label)
```

设置子页 w 的标签文本为 label。

3.6.7 示例 11：TabWidget 的应用（1）

TabWidget 控件是一个经常使用的控件，它可以把很多内容集中在一个窗体中，以减少很多窗体，使软件界面美观大方，操作简便。下面我们就通过这个示例来了解 TabWidget 的具体用法。

首先创建标准的 Qt Gui Application 项目，把项目自动生成的主界面中的 toolBar、menuBar 和 statusBar 删除，界面设计结果如图 3-52 所示。

图 3-52　示例 11 界面

1.控件说明

在属性编辑窗口中对控件的属性进行修改，修改内容如表 3-15 所示。

表 3-15　主要控件说明

控件类型	控件名称	控件说明
QTabWidget	tabWidget	主体控件
QTextedit	textEdit	tabWidget 内部控件
QPushButton	btnAddTab	在 tabWidget 控件中添加新的选项卡

2.示例说明

在设计界面时，TabWidget 控件中有一个 Tab1 选项卡；

自定义 3 个选项卡 QLabel、QPushButton 和 QTextEdit，分别添加到 tabWidget 控件中；

单击“addTab”按钮，在 TabWidget 控件中添加一个新的选项卡；

程序执行后，可以关闭 TabWidget 中的选项卡。

3.示例实现

头文件 mainwindow.h（文中的粗体为需要添加的内容）：

```
1.#ifndef MAINWINDOW_H
2.#define MAINWINDOW_H
3.#include <QMainWindow>
4.namespace Ui {
5.  class MainWindow;
6.}
7.class MainWindow : public QMainWindow
8.{
9.  Q_OBJECT
10.public:
11.  explicit MainWindow(QWidget *parent = 0);
12.  ~MainWindow();
13.private:
14.  Ui::MainWindow *ui;
15.private slots:  //声明槽函数
16.  void on_btnAddTab_clicked();
17.  void removeSubTab(int index);
18.};
```

```
19. #endif // MAINWINDOW_H
```

声明两个槽函数，槽函数 on_btnAddTab_clicked()对应单击按钮“addTab”，是通过右击控件→Go to slots 自动生成的，所以不用在实现文件中添加映射函数。槽函数 removeSubTab(int index)对应关闭选项卡信号。

在主窗体 mainwindow.cpp 文件中自动生成如下代码：

```
1.#include "mainwindow.h"
2.#include "ui_mainwindow.h"
3.MainWindow::MainWindow(QWidget *parent) :
4.  QMainWindow(parent),
5.  ui(new Ui::MainWindow)
6.{
7.  ui->setupUi(this);
8.}
9.MainWindow::~MainWindow()
10.{
11.  delete ui;
12.}
```

在主窗体 mainwindow.cpp 文件中添加的头文件：

```
1.#include <QLabel>
2.#include <QPushButton>
3.#include <QTextEdit>
```

3.6.7 示例 11：TabWidget 的应用（2）

在主窗体 mainwindow.cpp 文件中构造函数：

```
1./***构造函数***/
2.MainWindow::MainWindow(QWidget *parent) :
```

```
3.  QMainWindow(parent),
4.  ui(new Ui::MainWindow)
5. {
6.  ui->setupUi(this);
7.
8.  /************界面初始化
   ************************/
9.  ui->tabWidget->setWindowTitle("TabWidget");
10.  ui->tabWidget->addTab(new QLabel("<h1><font
11.
   color=blue>Hello!World!</font></h1>"),"QLabel");


12.
   ui->tabWidget->addTab(new QPushButton("Push XD")
   , "QPushButton");
13.
   ui->tabWidget->addTab(new QTextEdit, "QTextEdit"
   );
14.  ui->tabWidget->setTabsClosable(true);
15.
16.  /************信号和槽的映射
   ************************/
17.
   connect(ui->tabWidget,SIGNAL(tabCloseRequested(i
   nt)),this,SLOT(removeSubTab(int)));
18. }
```

在主窗体 mainwindow.cpp 文件中添加槽函数 removeSubTab(int index)：

```
1. /***槽函数：删除 tab***/
2. void MainWindow::removeSubTab(int index)
```

```
3. {
4. ui->tabWidget->removeTab(index);
5. }
```

在主窗体 mainwindow.cpp 文件中添加槽函数 on_btnAddTab_clicked()：

```
1. /***槽函数：添加 tab***/
2. void MainWindow::on_btnAddTab_clicked()
3. {
4.   QTextEdit *edit=new QTextEdit;
5.
6.   /************设置 edit 显示
   Hello World!*************************/
7.
   edit->setHtml("<h1><font color=red>Hello World!<
   /font></h1>");
8.   ui->tabWidget->addTab(edit,"New Tab");
9.
   ui->tabWidget->setCurrentIndex(ui->tabWidget->co
   unt()-1);
10. }
```

主文件 main.cpp 不需任何更改，使用项目自动生成的即可。

4.示例执行结果

示例执行结果如图 3-53 所示。

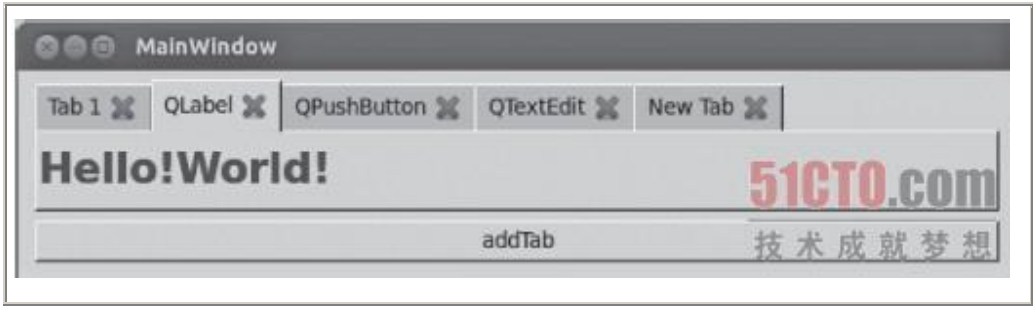


图 3-53　示例 11 执行结果

3.6.8　StackedWidget 控件

1.控件位置

Containers→StackedWidget

2.控件介绍

StackedWidget 控件中文称作“控件栈”。Qt 提供了这样一个控件栈，可以使开发人员用栈管理控件像用栈管理其他数据类型一样简单。控件栈只显示栈顶的控件。开发人员可以使用 raiseWidget()函数把栈中任何其他控件移到栈顶，从而实现控件之间的切换。

3.控件设置选项

在 Stacked Widget 控件的 properties 选项中，一般常对以下选项进行设置。

name：该控件对应源代码中的名称；

currentPage：当前活动的页面；

pageName：当前活动页的名称；

font:设置该控件内部文本的字体。

4.常用成员函数

1.1) QWidgetStack::QWidgetStack (QWidget *parent = 0, const char *name = 0)

构造一个名称为 name、父对象为 parent 的 WidgetStack。

1.2) int QWidgetStack::addWidget (QWidget *w, int id = -1)

把控件 w 添加到该控件栈中，标识是 id。

1.3) int QWidgetStack::id (QWidget *w) const

返回控件 w 的标识。

1.4) void QWidgetStack::raiseWidget (int id) [slot]

把标识为 id 的控件升到该控件栈的栈顶。

1.5) void QWidgetStack::raiseWidget (QWidget *w) [slot]

把控件 w 升到该控件栈的栈顶。

1.6) void QWidgetStack::removeWidget (QWidget *w)

把控件 w 从该控件栈中删除。

1.7) QWidget *QWidgetStack::widget (int id) const

返回标识是 id 的控件。

3.6.9 示例 12：StackedWidget 的应用

在使用 StackedWidget 创建界面时，一般和 QListWidget 一起使用，可以实现类似 TabWidget 的功能，但是它的界面布局和外形相对于 TabWidget 就灵活很多。

首先创建标准的 Qt Gui Application 项目，把项目在主界面中自动生成的 toolBar、menuBar 和 statusBar 删除，界面设计结果如图 3-54 所示。

图 3-54　示例 12 界面

1.控件说明

在属性编辑窗口中对控件的属性进行修改，修改内容如表 3-16 所示。

表 3-16　主要控件说明

控件类型	控件名称	控件说明	控件类型	控件名称	控件说明
QListWidget	listWidget	窗口列表框	QTextEdit	textEdit_2	显示在窗口 2 中
QStackedWidget	stackedWidget	承载所有窗口	QTextEdit	textEdit_3	显示在窗口 3 中
QTextEdit	textEdit	显示在窗口 1 中			

2.示例说明

这个示例很简单，单击 listWidget 中的窗口选项，右侧 stackedWidget 把对应窗口调到栈顶，显示对应窗口。

3.示例实现

所有控件都是采用拖拽的方式布局的，所以没有在头文件中声明任何控件，示例很简单，也没有声明任何槽函数和成员函数，所以没有对头文件 mainwindow.h 进行任何更改。

在主窗体 mainwindow.cpp 文件中自动生成如下代码：

```
1.#include "mainwindow.h"
2.#include "ui_mainwindow.h"
3.MainWindow::MainWindow(QWidget *parent) :
4.  QMainWindow(parent),
5.  ui(new Ui::MainWindow)
6.{
7.  ui->setupUi(this);
8.}
9.MainWindow::~MainWindow()
10.{
11.  delete ui;
12.}
```

在主窗体 mainwindow.cpp 文件中构造函数（文中的粗体为需要添加的内容）：

```
1./***构造函数***/
2.MainWindow::MainWindow(QWidget *parent) :
3.  QMainWindow(parent),
4.  ui(new Ui::MainWindow)
5.{
6.  ui->setupUi(this);
7.  setWindowTitle(tr("StackedWidget"));  //设置窗口
    标题
```

```
8.
    connect(ui->listWidget,SIGNAL(currentRowChanged(
    int)),ui->stackedWidget, SLOT(setCurrentIndex(in
    t) ) );//设置信号与槽的映射
9. }
```

在这个示例中，可以不用手写任何代码，通过在 Property Editor 中对窗口的属性进行修改，修改 windowTitle 项为 StackedWidget，如图 3-55 所示。

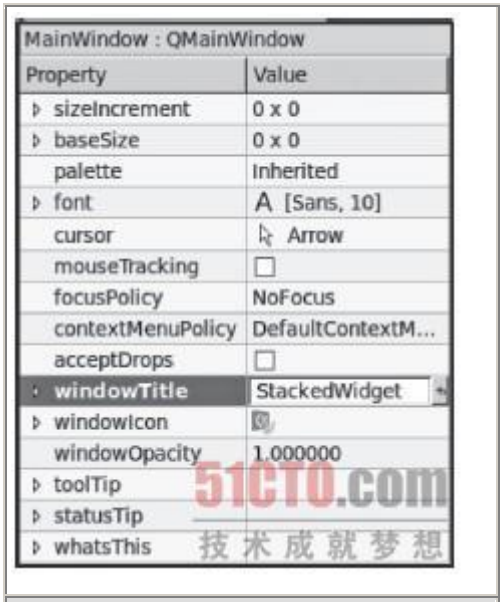


图 3-55　Property Editor 窗口

在 Signals & Slots Editor 窗口中添加新编辑项，选项如图 3-56 所示。

图 3-56　Signals & Slots Editor 窗口

这样就不用在实现函数的构造函数中添加如下代码了。

```
1. setWindowTitle(tr("StackedWidget"));
2. connect(ui->listWidget,SIGNAL(currentRowChanged(
   int)),ui->stackedWidget, SLOT(setCurrentIndex(in
   t)));
```

主文件 main.cpp 没有进行任何更改，使用项目自动生成的即可。

4.示例执行结果

示例执行结果如图 3-57 所示。

图 3-57　示例 12 执行结果

3.6.10　Frame 控件

1.控件位置

Containers→Frame

2.控件介绍

Frame 控件（框架）的样式如图 3-58 所示。Frame 用来存放其他控件，也可用于装饰。它一般用来作为更加复杂容器的基础，也可以用在 form 中作为占位控件。

图 3-58　Frame 控件

3.控件设置选项

在 Frame 控件的 properties 选项中，一般常对以下选项进行设置。

name：该控件对应源代码中的名称；

framesShape：框架外形格式；

framesShadow：框架阴影格式；

frameWidth：框架的宽度（只读）；

LineWidth：线宽。

4.常用成员函数

```
1. QFrame::QFrame ( QWidget *parent = 0, const char
    *name = 0, WFlags f = 0 )
```

构造一个框架风格为 NoFrame 并且 1 像素框架宽度的框架窗口部件,例如:

```
1. QFrame *f = new QFrame ();
```

3.6.11 Widget 控件

1. 控件位置

Containers→Widget

2. 控件介绍

Widget 控件（组件）在创建时是不可见的，它可以包含子控件，在删除该 Widget 时，子控件也将一起删除。

3. 控件设置选项

在 Widget 控件的 properties 选项中，一般常对以下选项进行设置。

name：该控件对应源代码中的名称；

font：设置表盘上的字体；

cursor：设置鼠标光标样式。

4. 常用成员函数

```
1. 1) QWidget::QWidget ( QWidget *parent = 0, const
    char *name = 0, WFlags f =0 )
```

构造一个名称为 name、父对象为 parent 的 Widget。

```
1.2) QWidget *QWidget::childAt ( int x, int y, boo
     l includeThis = FALSE ) const
```

返回该 Widget 坐标系统中像素位置(x,y)处的可视的子窗口部件。

```
1.3) QWidget *QWidget::childAt ( const QPoint & p,
      bool includeThis = FALSE ) const
```

返回该 Widget 坐标系统位置 p 处的可视的子窗口部件。

```
1.4) void QWidget::drawText ( int x, int y, const
     QString &str )
```

在该 Widget 坐标系统中像素位置(x,y)处绘制字符串 str。

```
1.5) void QWidget::drawText ( const QPoint & p,con
     st QString &str )
```

在该 Widget 坐标系统中位置 p 处绘制字符串 str。

3.6.12 MdiArea 控件

1.控件位置

Containers→MdiArea

2.控件介绍

MdiArea 控件中文称作“MDI 窗口显示区”。先来解释一下什么是 MDI，MDI 是 Multiple Document Interface 的简称，中文意思是多文档界面，多文档界面主要适用于当完成一项工作时需要用到多个文件，例如影像处理软件。QMainWindow 是 SDI（Single Document Interface，单文档界面）。每个开启的文件占据一个视窗，主要适用于所有工作没有太多文件参与的情况。

3.控件设置选项

在 MdiArea 控件的 properties 选项中，一般常对以下选项进行设置。

name：该控件对应源代码中的名称；

font：设置字体；

viewMode:设置视图模式 Qt 提供 TabbedView 和 SubWindowView 两种选择；

documentMode：保存的标签栏在选项卡式视图模式是否设置为文件的模式，默认为 false；

tabShape：（当 viewMode 是 TabbedView 时）设置该 MdiArea 的标签形式 Qt 提供两种选择：Rounded 和 Triangular；

tabPosition：（当 viewMode 是 TabbedView 时）设置标签所在方向；

activeSubWindowName：子窗口的名称；

activeSubWindowTitle：子窗口的标签。

4.常用成员函数

1.1) QMdiArea::QMdiArea (QWidget *parent = 0)
构造一个父对象为 parent 的 MdiArea。

1.2) void QMdiArea::activateNextSubWindow () [slot]
激活下一个窗口。

1.3) void QMdiArea::activatePreviousSubWindow () [slot]

激活上一个窗口。

1.4) QMdiSubWindow *QMdiArea::activeSubWindow () const

返回当前活动子窗口，如果当前没有活动子窗口，则返回 0。

1.5) QMdiSubWindow *QMdiArea::addSubWindow(QWidget *widget, QT::WindowFlags windowFlags = 0)

添加一个新的子窗口部件。

1.6) void QMdiArea::closeActiveSubWindow () [slot]

关闭当前活动子窗口。

1.7) void QMdiArea::closeAllSubWindows () [slot]

关闭所有的子窗口。

1.8) QMdiSubWindow *QMdiArea::currentSubWindow () const

函数功能同 activeSubWindow()。

1.9) void QMdiArea::removeSubWindow (QWidget *widget)

删除 widget，widget 必须是该 MdiArea 的子部件。

1.10) void QMdiArea::setActiveSubWindow (QMdiSubWindow *window) [slot]

设置子窗口 window 为当前活动子窗口。

3.6.13　示例 13：MdiArea 的应用（1）

MdiArea 一般用来设计多文档界面，下面就通过一个示例来具体学习 MdiArea 的用法。

首先创建标准的 Qt Gui Application 项目，把项目在主界面中自动生成的 toolBar、menuBar 和 statusBar 删除，界面设计结果如图 3-59 所示。

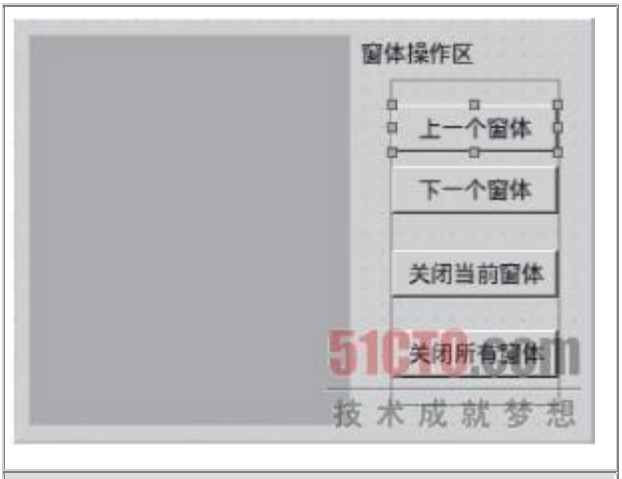


图 3-59　示例 13 界面

1.控件说明

在属性编辑窗口中对控件的属性进行修改，修改内容如表 3-17 所示。

表 3-17　主要控件说明

控件类型	控件名称	控件说明
QMdiArea	mdiArea	多文档窗体显示区
QGroupBox	groupBox	装饰界面
QPushButton	btnClose	关闭当前窗体
QPushButton	btnCloseAll	关闭所有窗体
QPushButton	btnNext	使下一个窗体为当前活动界面
QPushButton	btnPrevious	使上一个窗体为当前活动界面

2.示例说明

程序执行后，构造函数会在 mdiArea 中添加 5 个窗体；

单击“上一个窗体”按钮，mdiArea 中的上一个窗体变成当前活动窗体；

单击“下一个窗体”按钮，mdiArea 中的下一个窗体变成当前活动窗体；

单击“关闭当前窗体”按钮，程序会关闭 mdiArea 中当前活动窗体；

单击“关闭所有窗体”按钮，程序会关闭 mdiArea 中的所有窗体。

3.示例实现

头文件 mainwindow.h（文中的粗体为需要添加的内容）：

```
1.#ifndef MAINWINDOW_H
2.#define MAINWINDOW_H
3.#include <QMainWindow>
4.namespace Ui {
5.  class MainWindow;
6.}
7.class MainWindow : public QMainWindow
```

```
{
  Q_OBJECT
public:
  explicit MainWindow(QWidget *parent = 0);
  ~MainWindow();
private:
  Ui::MainWindow *ui;
private slots:  //声明槽函数
  void on_btnCloseAll_clicked();
  void on_btnClose_clicked();
  void on_btnNext_clicked();
  void on_btnPrevious_clicked();
};
#endif // MAINWINDOW_H
```

声明 4 个槽函数，对应 4 个按钮的单击信号。此处的槽函数都是通过右击控件→Go to slots 自动生成的，所以不用在实现文件中添加映射函数。

在主窗体 mainwindow.cpp 文件中自动生成如下代码：

```
#include "mainwindow.h"
#include "ui_mainwindow.h"
MainWindow::MainWindow(QWidget *parent) :
  QMainWindow(parent),
  ui(new Ui::MainWindow)
{
  ui->setupUi(this);
}
MainWindow::~MainWindow()
{
  delete ui;
}
```

3.6.13 示例 13：MdiArea 的应用（2）

在主窗体 mainwindow.cpp 文件中添加的头文件：

```
1.#include <QTextEdit>
```

在主窗体 mainwindow.cpp 文件中构造函数：

```
1./***构造函数***/
2.MainWindow::MainWindow(QWidget *parent) :
3.  QMainWindow(parent),
4.  ui(new Ui::MainWindow)
5.{
6.  ui->setupUi(this);
7.
8.
9.
10.  /****添加 window1****/
11.  QTextEdit *windows1 = new QTextEdit;
12.  windows1->setHtml("C");
13.  ui->mdiArea->addSubWindow(windows1);
14.
15.  /****添加 window2****/
16.  QTextEdit *windows2 = new QTextEdit;
17.  windows2->setHtml("C++");
18.  ui->mdiArea->addSubWindow(windows2);
19.
20.  /****添加 window3****/
21.  QTextEdit *windows3 = new QTextEdit;
22.  windows3->setHtml("Java");
23.  ui->mdiArea->addSubWindow(windows3);
24.
25.  /****添加新 window ****/
```

```
26.   QTextEdit *newnewWindows = new QTextEdit;
27.   newWindows->setHtml("New Windows!");
28.   ui->mdiArea->addSubWindow(newWindows);
29.   ui->mdiArea->cascadeSubWindows();
30. }
```

在主窗体 mainwindow.cpp 文件中添加 QPushButton btnPrevious 的槽函数：

```
1. /********设置前一个窗口为活动窗口***********/
2. void MainWindow::on_btnPrevious_clicked()
3. {
4.   ui->mdiArea->activatePreviousSubWindow();
5. }
```

在主窗体 mainwindow.cpp 文件中添加 QPushButton btnNext 的槽函数：

```
1. /********设置后一个为活动窗口***********/
2. void MainWindow::on_btnNext_clicked()
3. {
4.   ui->mdiArea->activateNextSubWindow();
5. }
```

在主窗体 mainwindow.cpp 文件中添加 QPushButton btnClose 的槽函数：

```
1. /********关闭当前窗口***********/
2. void MainWindow::on_btnClose_clicked()
3. {
4.   ui->mdiArea->closeActiveSubWindow();
5. }
```

在主窗体 mainwindow.cpp 文件中添加 QPushButton btnCloseAll 的槽函数：

```
1. /********关闭所有窗口***********/
2. void MainWindow::on_btnCloseAll_clicked()
```

```
3. {
4.   ui->mdiArea->closeAllSubWindows();
5. }
```

主文件 main.cpp 没有进行任何更改，使用项目自动生成的即可。

4.示例执行结果

示例执行结果如图 3-60 所示。

图 3-60　示例 13 执行结果

3.6.14　DockWidget 控件

1.控件位置

Containers→DockWidget

2.控件介绍

DockWidget 控件（停靠窗体）可以作为一个顶层窗口漂浮在桌面，主要作为辅助窗体出现在界面中，可以在很多 IDE 中看到停靠窗体。

3.控件设置选项

在 DockWidget 控件的 properties 选项中，一般常对以下选项进行设置。

name：该控件对应源代码中的名称；

font：设置表盘上的字体；

floating：设置该 DockWidget 是否为可漂浮；

feature：保存的停靠窗体的一些功能，是否为可移动、可关闭或浮动等，默认是可移动、可关闭和浮动；

allowedAreas：该 DockWidget 可以停靠的地方；

windowTitle：该停靠窗体的标题；

dockWidgetArea：设置该 DockWidget 的停靠地方；

docked：设置该 DockWidget 是否是停靠着的。

4.常用成员函数

```
1. 1) QDockWidget::QDockWidget ( const QString & ti
   tle, QWidget *parent = 0, QT::WindowFlags flags
   = 0 )
```

构造一个标题为 title、父对象为 parent 的 DockWidget。

1.2) QDockWidget::QDockWidget (QWidget *parent = 0, QT::WindowFlags flags = 0)

构造一个标题为 title、父对象为 parent 和标志为 flags 的 DockWidget。

1.3) void QDockWidget::setTitleBarWidget (QWidget *widget)

设置 widget 为该 DockWidget 的标题栏；如果 widget 为 0，将用默认标题栏代替。

1.4) QWidget *QDockWidget::titleBarWidget () const

返回该 DockWidget 定义的标题栏，如果没有定义标题栏，则返回 0。

1.5) void QDockWidget::setWidget (QWidget *widget)

设置 widget 为该 DockWidget 的部件，在调用这个函数之前，必须添加布局，否则 widget 就是不可见的。

1.6) QWidget *QDockWidget::widget () const

返回该 DockWidget 的部件，如果还没有设置部件，则返回 0。

3.6.15 示例 14：DockWidget 的示例

首先创建标准的 Qt Gui Application 项目，把项目在主界面中自动生成的 toolBar、menuBar 和 statusBar 删除。

1.示例说明

程序执行后，构造函数会在主窗体中添加 4 个停靠窗口，分别是 Main Window、Window1、Window2 和 Window3；

MainWindow 是不可关闭、不可移动和不可浮动的；

Window1 只可以在主窗体的左边或右边停靠，默认在左边停靠，是可移动的；

Window2 只可以在主窗体的左边或右边停靠，默认在左边停靠，是可浮动、可关闭和可移动的；

Window3 可以在主窗体任何边上停靠，默认在底部停靠，是可浮动、可关闭和可移动的。

2.示例实现

在主窗体 mainwindow.cpp 文件中自动生成如下代码：

```
1.#include "mainwindow.h"
2.#include "ui_mainwindow.h"
3.MainWindow::MainWindow(QWidget *parent) :
4.  QMainWindow(parent),
5.  ui(new Ui::MainWindow)
6.{
7.  ui->setupUi(this);
8.}
9.MainWindow::~MainWindow()
10.{
11.  delete ui;
12.}
```

在主窗体 mainwindow.cpp 文件中添加的头文件：

```
1.#include <QTextEdit>
2.#include <QDockWidget>
```

在主窗体 mainwindow.cpp 文件中构造函数（文中的粗体为需要添加的内容）：

```
1. /***构造函数***/
2. MainWindow::MainWindow(QWidget *parent) :
3.   QMainWindow(parent),
4.   ui(new Ui::MainWindow)
5. {
6.   ui->setupUi(this);
7.
   QTextEdit *title =new QTextEdit(this);//new QTex
   tEdit
8.   title->setText("Main Window");
9.   title->setAlignment(QT::AlignCenter);//设置
   title 居中布局
10.   this->setCentralWidget(title);//设置 title 为主
   体控件
11.
12.
   /*************DockWindow 1*********************
   */
13.
   QDockWidget *dock = new QDockWidget(tr("DockWind
   ow 1"),this);
14.
   dock->setFeatures(QDockWidget::DockWidgetMovable
   );
```

```
15.
    dock->setAllowedAreas(QT::LeftDockWidgetArea|QT:
    :RightDockWidgetArea);
16.   QTextEdit *title1 = new QTextEdit();
17.   title1->setText("Window 1");
18.   dock->setWidget(title1);
19.
    this->addDockWidget(QT::LeftDockWidgetArea,dock)
    ;
20.
    dock = new QDockWidget(tr("DockWindow 2"),this);

21.
    dock->setFeatures(QDockWidget::DockWidgetFloatab
    le|QDockWidget::DockWidgetClosable);
22.
23.
24.
25.
    /*************DockWindow 2**********************
    */
26.   QTextEdit *title2 = new QTextEdit();
27.   title2->setText("Window 2");
28.   dock->setWidget(title2);
29.
    this->addDockWidget(QT::LeftDockWidgetArea,dock)
    ;
30.
31.
    /*************DockWindow 3**********************
    */
```

```
32.
    dock = new QDockWidget(tr("DockWindow 3"),this);
33.
    dock->setFeatures(QDockWidget::AllDockWidgetFeat
    ures);
34.   QTextEdit *title3 = new QTextEdit();
35.   title3->setText("Window 3");
36.   dock->setWidget(title3);
37.
    this->addDockWidget(QT::BottomDockWidgetArea,doc
    k);
38. }
```

主文件 main.cpp 没有进行任何更改，使用该项目自动生成的即可。

3.示例执行结果示例执行结果如图 3-61 所示。

图 3-61　示例 14 执行结果

3.7 输入组件(Input Widgets)

Qt Creator 有 15 种 Input Widgets，如图 3-62 所示。

图 3-62 Input Widgets 控件

Input Widgets 的 Qt 类和名称介绍如表 3-18 所示。

表 3-18 Input Widgets 介绍

控件类	控件名	中文名	控件类	控件名	中文名
QComboBox	ComboBox	不可编辑组合框	QDateEdit	DateEdit	日期编辑框
QFontComboBox	Font Combo Box	可编辑组合框	QDateTimeEdit	Date/TimeEdit	日期时间编辑框
QLineEdit	LineEdit	行编辑	QDial	Dial	表盘
QTextEdit	TextEdit	文本编辑	QScrollBar	Horizontal ScrollBar	水平滚动条
QPlainTextEdit	Plain Text Edit	无格式的文本编辑	QScrollBar	Vertical ScrollBar	垂直滚动条
QSpinBox	SpinBox	整数旋转框	QSlider	Horizontal Slider	水平滑动条
QDoubleSpinBox	Double Spin Box	小数旋转框	QSlider	Vertical Slider	垂直滑动条
QTimeEdit	TimeEdit	时间编辑框			

本节主要介绍Qt Creator的Input Widgets的使用方法。Qt具有多种Input类型控件，分别有ComboBox、LineEdit、TextEdit等显示控件。Qt Creator比Qt Designer新增添了很多输入控件，使开发人员更加方便设计界面。

下面就来具体介绍各种Input Widgets的用法。

3.7.1 ComboBox控件

1.控件位置

Input Widgets→ComboBox

2.控件介绍

ComboBox控件（组合框）的样式如图3-63所示。Qt Creator的ComboBox和Qt Designer的ComboBox样式有些不同。

图 3-63　ComboBox 控件

3.控件设置选项

在 ComboBox 控件的 properties 选项中，一般常对以下选项进行设置。

name：该控件对应源代码中的名称；

font：设置显示的字体；

editable：用来获取或设置一个值，以确定 ComboBox 的编辑框是否可编辑。值为 true 时为可编辑，值为 false 时为只读；

currentIndex：当前选项的索引；

maxVisibleItems：ComboBox 可见的最大项目数；

maxCount：ComboBox 的最大项目数。

4.常用成员函数

1. 1) QComboBox::QComboBox (QWidget *parent = 0, const char *name = 0)

构造一个名称为 name、父对象为 parent 的 ComboBox。

1. 2) QComboBox::QComboBox (bool rw, QWidget *parent = 0, const char *name = 0)

构造一个名称为 name、父对象为 parent 的 ComboBox。如果 rw 是 true，则编辑栏可编辑；否则只能选择 ComboBox 项目之一。

1. 3) void QComboBox::clear () [slot]

删除 ComboBox 中的所有项目。

1.4) int QComboBox::count () const

返回 ComboBox 中的项目数。

1.5) int QComboBox::currentItem () const

返回 ComboBox 中当前项目的索引。

1.6) QString QComboBox::currentText () const

返回组合框的当前项目文本。

1.7) void QComboBox::insertItem (const QString & t, int index = -1)

在索引 index 处，插入一个文本为 t 的项目。如果 index 是负数，该项目将被追加到末尾。

1.8) void QComboBox::insertItem (const QPixmap & pixmap, int index = -1)

在索引 index 处，插入一个图标为 pixmap 的项目。如果 index 是负数，该项目将被追加到末尾。

1.9) void QComboBox::insertItem (const QPixmap & pixmap, const QString & text, int index = -1)

在索引 index 处，插入一个文本为 t 和图标为 pixmap 的项目。如果 index 是负数，该项目将被追加到末尾。

1.10) QString QComboBox::currentText () const

返回组合框的当前项目文本。

1.11) void QComboBox::removeItem (int index)

删除索引 index 处项目。

1.12) void QComboBox::setCurrentItem (int index) [virtual]

把索引 index 处的项目设为当前项目。

3.7.2 Font ComboBox 控件

1.控件位置

Input Widgets→Font ComboBox

2.控件介绍

Font ComboBox 控件（字体组合框）继承于 QComboBox 类，样式如图 3-64 所示。Font ComboBox 组合框分两个部分显示：顶部是一个允许输入文本的文本框，下面的列表框则显示列表项。可以认为 Font ComboBox 就是文本框与列表框的组合。与列表框相比，组合框不能选择多项，只能选择其中的一项，专门用于字体选择。

图 3-64 Font ComboBox 控件

3.控件设置选项

在 Font ComboBox 控件的 properties 选项中，一般常对以下选项进行设置。

name：该控件对应源代码中的名称；

font：设置显示的字体；

editable：用来获取或设置一个值，以确定 ComboBox 的编辑框是否可编辑。值为 true 时为可编辑，值为 false 时为只读；

currentIndex：当前选项的索引；

currentFont：当前字体；

maxVisibleItems：ComboBox 可见的最大项目数；

maxCount：ComboBox 的最大项目数。

4.常用成员函数

```
1. QFontComboBox::QFontComboBox ( QWidget *parent =
   0 )
```

构造一个父对象为 parent 的 Font ComboBox。

3.7.3 LineEdit 控件

1.控件位置

Input Widgets→LineEdit

2.控件介绍

LineEdit 控件（行编辑）是一种常用且比较容易掌握的控件。应用程序主要使用它来接收输入文字信息，样式如图 3-65 所示。

图 3-65　LineEdit 控件

3. 控件设置选项

在 LineEdit 控件的 properties 选项中，一般常对以下选项进行设置。

name：该控件对应源代码中的名称；

text：该控件对应图形界面中显示的名称；

font：设置 text 的字体；

ReadOnly：用来获取或设置一个值，该值指示文本框中的文本是否为只读，值为 true 时为只读，值为 false 时为可读写。

4. 常用成员函数

1.1) QLineEdit::QLineEdit (QWidget *parent, const char *name = 0)

构造一个名称为 name、父对象为 parent 的 LineEdit。

1.2) QLineEdit::QLineEdit (const QString & contents, QWidget *parent, const char *name = 0)

构造一个名称为 name、父对象为 parent 和内容为 contents 的 LineEdit。

1.3) void QLineEdit::clear () [virtual slot]

清除行编辑的内容。

1.4) bool QLineEdit::isReadOnly () const

如果行编辑是只读则返回 true，否则返回 false。

1.5) void QLineEdit::setReadOnly (bool)
[virtual slot]

设置行编辑的读写属性。

1.6) void QLineEdit::setText (const QString &)
[virtual slot]

设置行编辑的文本。

1.7) QString QLineEdit::text () const

返回行编辑的文本。

3.7.4 TextEdit 控件

1.控件位置

Input Widgets→TextEdit

2.控件介绍

TextEdit 控件（文本编辑）的样式如图 3-66 所示。应用程序主要使用它输入文本信息或显示文本信息。TextEdit 的属性和成员函数与 LineEdit 基本相同，在此不做介绍。

图 3-66　TextEdit 控件

3.7.5　PlainTextEdit 控件

1. 控件位置

Input Widgets→PlainTextEdit

2. 控件介绍

PlainTextEdit 控件中文称作“文本编辑”。PlainTextEdit 控件和 TextEdit 控件只是样式有些不同，可以通过设置属性来改变样式。

3. 常用成员函数

构造函数：

```
1. 1) QPlainTextEdit::QPlainTextEdit ( QWidget *par
   ent = 0 )
2. 2) QPlainTextEdit::QPlainTextEdit ( const QStrin
   g & text, QWidget *parent = 0 )
```

例如：

```
1. QPlainTextEdit *plaintextedit = new QPlainTextEd
   it("Hello World!",this);
```

3.7.6　示例 15：ComboBox、LineEdit 和 TextEdit 的应用（1）

ComboBox、LineEdit 和 TextEdit 都是常用控件，并且 Font ComboBox 和 ComboBox 用法相似，PlainTextEdit 和 TextEdit 用法相似，所以 Font ComboBox 和 PlainTextEdit 的示例就不再单独列举，这里用一个综合示例来介绍这几个控件的详细用法。

首先创建标准的 Qt Gui Application 项目，把项目在主界面中自动生成的 toolBar、menuBar 和 statusBar 删除，界面设计结果如图 3-67 所示。

1.控件说明

在属性编辑窗口中对控件的属性进行修改，修改内容如表 3-19 所示。

图 3-67　示例 15 界面

表 3-19　主要控件说明

控件类型	控件名称	控件说明
QLineEdit	leX	四则运算的第一个参数
QComboBox	cbxRule	四则运算的规则
QLineEdit	leY	四则运算的第二个参数
QPushButton	btnEqual	计算结果
QTextEdit	textEditResult	显示计算结果

控件类型控件名称控件说明 QLineEditleX 四则运算的第一个参数 QComboBoxcbxRule 四则运算的规则 QLineEditleY 四则运算的第二个参数 QPushButtonbtnEqual 计算结果 QTextEdittextEditResult 显示计算结果

2.示例说明

本示例实现的是简单的四则运算；

在两个输入框中输入两个参数，选择运算法则，然后单击“=”按钮，运算结果显示在右边的 textEdit 框中。

3.示例实现

头文件 mainwindow.h（文中的粗体为需要添加的内容）：

```
1.#ifndef MAINWINDOW_H
2.#define MAINWINDOW_H
3.#include <QMainWindow>
4.namespace Ui {
5.  class MainWindow;
6.}
7.class MainWindow : public QMainWindow
8.{
```

```
9.  Q_OBJECT
10. public:
11.   explicit MainWindow(QWidget *parent = 0);
12.   ~MainWindow();
13. private:
14.   Ui::MainWindow *ui;
15. private slots:  //声明槽函数
16.   void on_btnEqual_clicked();
17. };
18. #endif // MAINWINDOW_H
```

在头文件中声明一个槽函数 on_btnEqual_clicked()，对应按钮“=”的单击信号。此处的槽函数是通过右击控件→Go to slots 自动生成的，所以不用在实现文件中添加映射函数。

在主窗体 mainwindow.cpp 文件中自动生成如下代码：

```
1. #include "mainwindow.h"
2. #include "ui_mainwindow.h"
3. MainWindow::MainWindow(QWidget *parent) :
4.   QMainWindow(parent),
5.   ui(new Ui::MainWindow)
6. {
7.   ui->setupUi(this);
8. }
9. MainWindow::~MainWindow()
10. {
11.   delete ui;
12. }
```

3.7.6 示例 15：ComboBox、LineEdit 和 TextEdit 的应用（2）

在主窗体 mainwindow.cpp 文件中添加的头文件：

```
1. #include <QMessageBox>
```

在主窗体 mainwindow.cpp 文件中槽函数 on_btnEqual_clicked()：

```
1. /***槽函数：计算结果***/
2. void MainWindow::on_btnEqual_clicked()
3. {
4.   if(ui->leX->text().isEmpty())
5.   {
6. QMessageBox::warning(NULL, "warning", "Please in
   put X ", QMessageBox::Yes | QMessageBox::No, QMe
   ssageBox::Yes);
7.   }
8.   if(ui->leY->text().isEmpty())
9.   {
10. QMessageBox::warning(NULL, "warning", "Please i
   nput Y ", QMessageBox::Yes | QMessageBox::No, QM
   essageBox::Yes);
11.   }
12.   int x,y;
13.
   str += ui->leX->text()+ui->cbxRule->currentText(
   )+ui->leY->text()+" = ";
14.
15.   /**************获取 x 和
   y*********************************/
16.   x=ui->leX->text().toInt();
17.   y=ui->leY->text().toInt();
18.
19.   /**************判断法则
   *********************************/
20.   if(ui->cbxRule->currentText()=="+")
21. str +=QString::number(x+y)+"\n";
22.   else if(ui->cbxRule->currentText()=="-")
```

```
23. str +=QString::number(x-y)+"\n";
24.   else if(ui->cbxRule->currentText()=="*")
25. str +=QString::number(x*y)+"\n";
26.   else if(ui->cbxRule->currentText()=="/")
27. str +=QString::number(x/y)+"\n";
28.
29. /*****************显示结果
    ***********************************/
30.   ui->textEditResult->setText(str);
31. }
```

主文件 main.cpp 没有进行任何更改，使用该项目自动生成的即可。

4.示例执行结果

示例执行结果如图 3-68 所示。

图 3-68　示例 15 执行结果

3.7.7 SpinBox 控件

1.控件位置

Input Widgets→SpinBox

2.控件介绍

SpinBox 控件（整数旋转框）的样式如图 3-69 所示。SpinBox 允许用户通过单击向上/向下按钮来增加/减少当前显示的值，也可以直接输入旋转框的值。如果该值是直接输入旋转框，一般需要按【Enter】键确认新值（有的版本不需要），该值通常是一个整数。

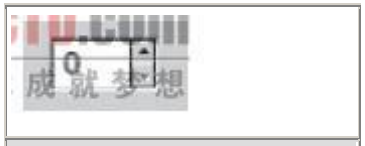


图 3-69　SpinBox 控件

3.控件设置选项

在 SpinBox 控件的 properties 选项中，一般常对以下选项进行设置。

name：该控件对应源代码中的名称；

font：设置编辑框的字体；

value：SpinBox 默认值；

lineStep：使用箭头来改变旋转框的值的递增/递减量；

minValue：SpinBox 的最小值；

maxValue：SpinBox 的最大值；

prefix:SpinBox 的前缀字符串；

Suffix:SpinBox 的后缀字符串。

4.常用成员函数

```
1. 1) QSpinBox::QSpinBox ( QWidget *parent = 0, con
   st char *name = 0 )
```

构造一个名称为 name、父对象为 parent 的 SpinBox。

```
1. 2) QSpinBox::QSpinBox ( int minValue, int maxVal
   ue, int step = 1, QWidget *parent = 0, const cha
   r *name = 0 )
```

构造一个名称为 name、父对象为 parent、最小值为 minValue、最大值为 maxValue 和增/减量为 step 的 SpinBox。例如：

```
1. QSpinBox *sb = new QSpinBox(0,99,1,this);
2. 3) void QSpinBox::stepDown () [virtual slot]
```

把旋转框的值减小一个 lineStep。等同于单击向下按钮。

```
1. 4) void QSpinBox::stepUp () [virtual slot]
```

把旋转框的值增加一个 lineStep。等同于单击向上按钮。

```
1. 5) QString QSpinBox::text () const
```

返回旋转框的文本，包括前缀和后缀。

```
1. 6) int QSpinBox::value () const
```

返回旋转框的值。

1.7) void QSpinBox::setValue (int value) [virtual slot]

设置旋转框的值。

1.8) void QSpinBox::setLineStep (int)

设置旋转框的增/减量。

1.9) void QSpinBox::setMaxValue (int)

设置旋转框的最大值。

1.10) void QSpinBox::setMinValue (int)

设置旋转框的最小值。

1.11) void QSpinBox::setPrefix (const QString & text) [virtual slot]

设置旋转框的前缀字符串。

1.12) void QSpinBox::setSuffix (const QString & text) [virtual slot]

设置旋转框的后缀字符串。

3.7.8 Double SpinBox 控件

1.控件位置

Input Widgets→Double SpinBox

2.控件介绍

Double SpinBox 控件（小数旋转框）继承于 QSpinBox，样式如图 3-70 所示。Double SpinBox 与 SpinBox 的不同是其可以表示小数，而其他的功能都一样。

图 3-70　Double SpinBox 控件

3.控件设置选项

在 Double SpinBox 控件的 properties 选项中，一般常对以下选项进行设置。

name：该控件对应源代码中的名称；

font：设置编辑框的字体；

value：SpinBox 默认值；

lineStep：使用箭头来改变旋转框的值的递增/递减量；

minValue：SpinBox 的最小值；

maxValue：SpinBox 的最大值；

prefix:SpinBox 的前缀字符串；

Suffix:SpinBox 的后缀字符串；

decimals:SpinBox 的小数位数。

4.常用成员函数

```
1. 1) QDoubleSpinBox::QDoubleSpinBox ( QWidget *par
   ent = 0 )
```

构造一个父对象为 parent 的 Double SpinBox，例如：

```
1. QDoubleSpinBox *dsb = new QDoubleSpinBox(this);
```

3.7.9 Slider 控件

1.控件位置

Input Widgets→Horizontal/Vertical Slider

2.控件介绍

Slider 控件（滑动条）的样式如图 3-71 所示。与 Qt Designer 不同的是，Qt Creator 只是把 Slider 分成 Horizontal/Vertical Slider 两个控件，但是功能相同，两种滑动条之间可以相互转换，只须改变 orientation 属性即可。

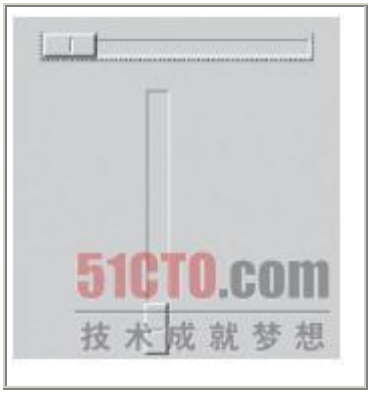


图 3-71　Slider 控件

3.控件设置选项

在 Slider 控件的 properties 选项中，一般常对以下选项进行设置。

name：该控件对应源代码中的名称；

font：设置滑动条上的字体；

lineStep：滑动条值的最小跨度；

value：滑动条的值；

minValue：滑动条的最小值；

maxValue：滑动条的最大值；

orientation：滑动条的布局方向，Qt 提供的选择有 Horizontal 和 Vertical。

4.常用成员函数

```
1. 1) QSlider::QSlider ( QWidget *parent, const cha
   r *name = 0 )
```

构造一个名称为 name、父对象为 parent 的 Slider。

```
1. 2) QSlider::QSlider ( Qrientation orientation,
   QWidget *parent , const char *name = 0 )
```

构造一个名称为 name、父对象为 parent 和布局方向为 orientation 的 Slider。

1. 3) QSlider::QSlider (int minValue, int maxValue, int pageStep, int value, Orientation orientation, QWidget *parent , const char *name = 0)

构造一个名称为 name、父对象为 parent、布局方向为 orientation、最大值为 maxValue、最小值为 minValue、页步长为 pageStep 和值为 value 的 Slider。

1. 4) void QSlider::setValue (int v) [virtual slot]

设置该滑动条的值为 v。

1. 5) int QSlider::value () const

返回该滑动条的值。

3.7.10 示例 16：SpinBox、Double SpinBox 和 Slider 的应用（1）

首先创建标准的 Qt Gui Application 项目，把项目在主界面中自动生成的 toolBar、menuBar 和 statusBar 删除。设计界面如图 3-72 所示。

图 3-72　示例 16 界面

1.控件说明

在属性编辑窗口中对控件的属性进行修改，修改内容如表 3-20 所示。

表 3-20　主要控件说明

控件类型	控件名称	控件说明
QSpinBox	spinBox	与 horizontalSlider 相互控制，minimum = 0；maxmum = 99；singleStep = 1
QSlider	horizontalSlider	与 spinBox 相互控制，minimum = 0；maxmum = 99；singleStep = 1
QDoubleSpinBox	doubleSpinBox	与 horizontalSlider_2 相互控制，minimum = 0.00；maxmum = 99.99；singleStep = 1.00
QSlider	horizontalSlider_2	与 doubleSpinBox 相互控制，minimum = 0；maxmum = 9999；singleStep = 1

2.示例说明

改变 spinBox 的值，horizontalSlider 的值随着改变；改变 horizontalSlider 的值，spinBox 的值也会随着改变；

改变 doubleSpinBox 的值，horizontalSlider_2 的值随着改变；改变 horizontalSlider_2 的值，doubleSpinBox 的值也会随着改变。

3.示例实现

头文件 mainwindow.h（文中的粗体为需要添加的内容）：

```
1.#ifndef MAINWINDOW_H
2.#define MAINWINDOW_H
3.#include <QMainWindow>
4.namespace Ui {
```

```
5.  class MainWindow;
6. }
7. class MainWindow : public QMainWindow
8. {
9.  Q_OBJECT
10. public:
11.  explicit MainWindow(QWidget *parent = 0);
12.  ~MainWindow();
13. private:
14.  Ui::MainWindow *ui;
15. private slots://声明槽函数
16.  void slotDoubleSpinbox_Slider();
17.  void slotSlider_DoubleSpinbox();
18. };
19. #endif // MAINWINDOW_H
```

在头文件中声明两个槽函数，槽函数 slotDoubleSpinbox_Slider()使 horizontalSlider_2 的值随着 doubleSpinBox 的值发生变化而变化；槽函数 slotSlider_DoubleSpinbox()使 doubleSpinBox 的值随着 horizontalSlider_2 的值改变而改变。另外，在 Signals & Slots Editor 窗口中添加了两个信号和槽的映射，如图 3-73 所示。

图 3-73　信号与槽的映射

在主窗体 mainwindow.cpp 文件中自动生成如下代码：

```
#include "mainwindow.h"
#include "ui_mainwindow.h"
MainWindow::MainWindow(QWidget *parent) :


  QMainWindow(parent),
  ui(new Ui::MainWindow)
{
  ui->setupUi(this);
}
MainWindow::~MainWindow()
{
  delete ui;
}
```

3.7.10 示例 16：SpinBox、Double SpinBox 和 Slider 的应用（2）

在主窗体 mainwindow.cpp 文件中构造函数：

```
/***构造函数***/
MainWindow::MainWindow(QWidget *parent) :
  QMainWindow(parent),
  ui(new Ui::MainWindow)
{
  ui->setupUi(this);

  /*********信号和槽的映射*****************************/
connect(ui->doubleSpinBox, SIGNAL(valueChanged(double)), this, SLOT(sl
otDoubleSpinbox_Slider()));
connect(ui->horizontalSlider_2, SIGNAL(valueChanged(int)), this, SLOT(
slotSlider_DoubleSpinbox()));
}
```

在主窗体 mainwindow.cpp 文件中添加槽函数 slotDoubleSpinbox_Slider()：

```
/***槽函数：设置 horizontalSlider_2 的值***/
void MainWindow::slotDoubleSpinbox_Slider()
{
ui->horizontalSlider_2->setValue((int)(ui->doubleSpinBox->value()*100)
);
}
```

在主窗体 mainwindow.cpp 文件中添加 QSlider horizontalSlider_2 的槽函数：

```
/***槽函数：设置 doubleSpinBox 的值***/
void MainWindow::slotSlider_DoubleSpinbox()
{
ui->doubleSpinBox->setValue((double)(ui->horizontalSlider_2->value())/
100);
}
```

主文件 main.cpp 没有进行任何更改，使用项目自动生成的即可。

4.示例执行结果

示例执行结果如图 3-74 所示。

图 3-74　示例 16 执行结果

3.7.11　Dial 控件

1. 控件位置

Input Widgets→Dial

2.控件介绍

Dial 控件（表盘）的样式如图 3-75 所示。Dial 可以用来描述各种各样的仪表盘样式。

图 3-75　Dial 控件

3.控件设置选项

在 Dial 控件的 properties 选项中，一般常对以下选项进行设置。

name：该控件对应源代码中的名称；

font：设置表盘上的字体；

lineStep：表盘值的最小跨度；

value：表盘的值；

minValue：表盘的最小值；

maxValue：表盘的最大值；

notchTarget：表盘的刻度；

notchesVisible：是否显示表盘刻度，值为 true 时显示，否则不显示。

4. 常用成员函数

```
1. 1) QDial::QDial ( QWidget *parent = 0, const cha
   r *name = 0, WFlags f = 0 )
```

构造一个名称为 name、父对象为 parent 的 Dial。

```
1. 2) QDial::QDial ( int minValue, int maxValue, in
   t pageStep, int value, QWidget *parent = 0, cons
   t char *name = 0 )
```

构造一个名称为 name、父对象为 parent、最大值为 maxValue、最小值为 minValue、步长为 pageStep 和值为 value 的 Dial。

```
1. 3) void QDial::setValue ( int v ) [virtual slot
   ]
```

设置该表盘的值为 v。

```
1. 4) int QDial::value () const
```

返回该表盘的值。

3.7.12 示例 17：Dial 的应用

Dial 是绘制一些表盘时的最好选择，界面美观大方，使用起来还很简单。下面就通过示例来了解 Dail 的具体用法。

首先创建标准的 Qt Gui Application 项目，把项目在主界面中自动生成的 toolBar、menuBar 和 statusBar 删除，界面设计结果如图 3-76 所示。

图 3-76　示例 17 界面

1.控件说明

在属性编辑窗口中对控件的属性进行修改，修改内容如表 3-21 所示。

2.示例说明

改变 spinBox 的值，dial 的值随着改变；改变 dial 的值，spinBox 的值也会随着改变。

3.示例实现

在这个示例中，不需要手动添加任何代码，只须在 Signals & Slots Editor 窗口中添加了两个信号和槽的映射即可，如图 3-77 所示。

图 3-77　信号与槽的映射

4.示例执行结果

示例执行结果如图 3-78 所示。

图 3-78　示例 17 执行结果

表 3-21　主要控件说明

控件类型	控件名称	控件说明
QDail	dial	表盘控件
QSpinBox	spinBox	与 dial 的值相互影响

3.7.13　ScrollBar 控件

1.控件位置

Input Widgets→Horizontal/Vertical ScrollBar

2.控件介绍

ScrollBar 控件（滚动条）的样式如图 3-79 所示。ScrollBar 和 Slider 相同，Qt Creator 只是把 Slider 分成 Horizontal Slider 和 Vertical

Slider 两个控件。两种滚动条之间同样可以相互转换，只须改变 orientation 属性即可。

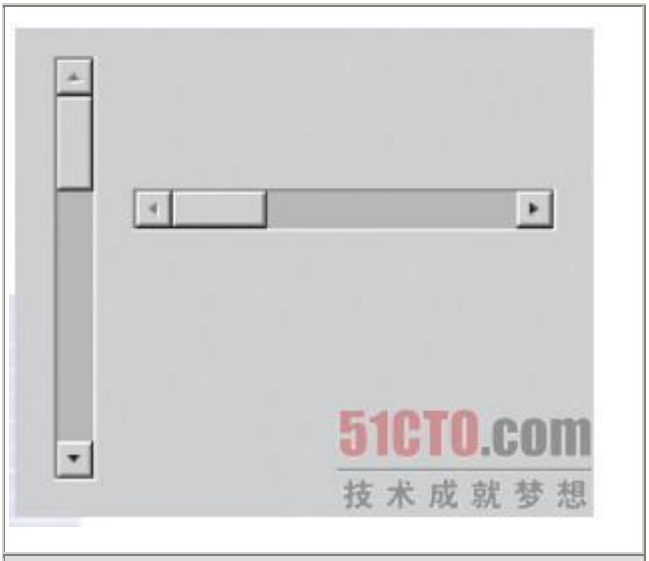


图 3-79　ScrollBar 控件 3.控件设置选项

在 ScrollBar 控件的 properties 选项中，一般常对以下选项进行设置。

name：该控件对应源代码中的名称；

font：设置滑动条上的字体；

lineStep：滑动条值的最小跨度；

value：滑动条的值；

minValue：滑动条的最小值；

maxValue：滑动条的最大值；

orientation：滑动条的布局方向，Qt 提供的选择有 Horizontal 和 Vertical。

4. 常用成员函数

1. 1) QScrollBar::QScrollBar (QWidget *parent, const char *name = 0)

构造一个名称为 name、父对象为 parent 的 ScrollBar。

1. 2) QScrollBar::QScrollBar (Orientation orientation, QWidget *parent, const char *name = 0)

构造一个名称为 name、父对象为 parent、布局方向为 orientation 的 ScrollBar。

1. 3) void QScrollBar::setValue (int v) [slot]

设置该滚动条的值为 v。

1. 4) int QScrollBar::value () const

返回该滚动条的值。

3.7.14 DateEdit 控件

1. 控件位置

Input Widgets→DateEdit

2. 控件介绍

DateEdit 控件（日期编辑框）的样式如图 3-80 所示，是用来编辑和显示日期的控件。

图 3-80　DateEdit 控件

3. 控件设置选项

在 DateEdit 控件的 properties 选项中，一般常对以下选项进行设置。

name：该控件对应源代码中的名称；

font：设置文本框的字体；

date：显示的日期；

minValue：日期的最小值；

maxValue：日期的最大值；

order：设置日期显示格式，Qt 提供的格式有 YMD、YDM、DMY。

4. 常用成员函数

```
1. 1) QDateEdit::QDateEdit ( QWidget *parent = 0, const char *name = 0 )
```

构造一个名称为 name、父对象为 parent 的 DateEdit。

```
1. 2) QDateEdit::QDateEdit ( const QDate & date, QWidget *parent = 0, con
   st char *name = 0 )
```

构造一个名称为 name、父对象为 parent、当前显示日期为 date 的 DateEdit。下面是以当前日期构造的一个对象示例：

```
1. QDateEdit *Date = new QDateEdit(QDate::currentDate(), this );
2. 3) QDate QDateEdit::date () const
```

返回 DateEdit 控件 Date 的值。

```
1. 4) void QDateEdit::setDate ( const QDate & date ) [virtual]
```

设置 DateEdit 控件 Date 的值为 date。

```
1. 5) void QDateEdit::setDay ( int day ) [virtual protected]
```

设置 DateEdit 控件 Date 的 Day 为 day，必须确保 day 为有效值。

```
1. 6) void QDateEdit::setMonth ( int month ) [virtual protected]
```

设置 DateEdit 控件 Date 的 Month 为 month，必须确保 month 为有效值。

1. 7) void QDateEdit::setYear (int year) [virtual protected]

设置 DateEdit 控件 Date 的 Year 为 year，必须确保 year 为有效值。

3.7.15 TimeEdit 控件

1.控件位置

Input Widgets→TimeEdit

2.控件介绍

TimeEdit 控件（时间编辑框）的样式如图 3-81 所示，是用来编辑和显示时间的控件。

图 3-81 TimeEdit 控件

3.控件设置选项

在 TimeEdit 控件的 properties 选项中，一般常对以下选项进行设置：

name：该控件对应源代码中的名称；

font：设置文本框的字体；

time：显示的日期；

minValue：时间的最小值，默认是 00:00:00；

maxValue：时间的最大值，默认是 23:59:59；

display：设置时间显示格式。

4.常用成员函数

1.1) QTimeEdit::QTimeEdit (QWidget *parent = 0, const char *name = 0)

构造一个名称为 name、父对象为 parent 的 TimeEdit。

1.2) QTimeEdit::QTimeEdit (const QTime & time, QWidget *parent = 0, const char *name = 0)

构造一个名称为 name、父对象为 parent、当前显示时间为 time 的 TimeEdit。

1.3) QTime QTimeEdit::time () const

返回 TimeEdit 控件 Time 的值。

1.4) void QTimeEdit::setTime (const QTime & time) [virtual]

设置 TimeEdit 控件 Time 的值为 time。

1.5) void QTimeEdit::setHour (int h) [virtual protected]

设置 TimeEdit 控件 Time 的 Hour 为 h，必须确保 h 为有效值。

1.6) void QTimeEdit::setMinute (int m) [virtual protected]

设置 TimeEdit 控件 Time 的 Minute 为 m，必须确保 m 为有效值。

1.7) void QTimeEdit::setSecond (int s) [virtual protected]

设置 TimeEdit 控件 Time 的 Second 为 s，必须确保 s 为有效值。

3.7.16 DateTimeEdit 控件

1.控件位置

Input Widgets→DateTimeEdit

2.控件介绍

DateTimeEdit 控件(日期时间编辑框)的样式如图 3-82 所示。DateTimeEdit 是一个用来编辑和显示日期和时间的控件,相当于把 DateEdit 和 TimeEdit 联合起来一起使用。这里不做过多介绍，会在接下来的示例中具体介绍 DateTimeEdit 控件的用法。

图 3-82 DateTimeEdit 控件

3.控件设置选项

在 DateTimeEdit 控件的 properties 选项中,一般常对以下选项进行设置：

name：该控件对应源代码中的名称；

font：设置文本框的字体；

datetime：显示的日期和时间。

3.7.17 示例 18：DateEdit、TimeEdit 和 DateTimeEdit 的应用（1）

首先创建标准的 Qt Gui Application 项目，把项目在主界面中自动生成的 toolBar、menuBar 和 statusBar 删除，界面设计结果如图 3-83 所示。

图 3-83 示例 18 界面

1.控件说明

在属性编辑窗口中对控件的属性进行修改，修改内容如表 3-22 所示。

表 3-22 主要控件说明

控件类型	控件名称	控件说明	控件类型	控件名称	控件说明
QDateTimeEdit	dateTimeEdit	显示日期和时间	QPushButton	btnEditDate	使 dateEdit 可编辑
QDateEdit	dateEdit	编辑日期	QPushButton	btnEditTime	使 timeEdit 可编辑
QTimeEdit	timeEdit	编辑时间	QPushButton	btnOk	编辑日期或时间结束

2.示例说明

程序执行后，dateTimeEdit 显示当前系统时间，并时刻更新；

dateTimeEdit、dateEdit 和 timeEdit 是只读的；

单击按钮“Edit Date”，可以编辑 dateEdit，dateTimeEdit 和 dateEdit 是同步的。单击按钮“OK”，结束编辑，dateEidt 变成只读的；

单击按钮“Edit Time”，可以编辑 timeEdit，dateTimeEdit 和 timeEdit 是同步的。单击按钮“OK”，结束编辑，timeEidt 变成只读的。

3.示例实现

头文件 mainwindow.h（文中的粗体为需要添加的内容）：

```
1.#ifndef MAINWINDOW_H
2.#define MAINWINDOW_H
3.#include <QMainWindow>
4.namespace Ui {
5.  class MainWindow;
6.}
7.class MainWindow : public QMainWindow
8.{
9.  Q_OBJECT
10.public:
```

```
11.   explicit MainWindow(QWidget *parent = 0);
12.   ~MainWindow();
13. private:
14.   Ui::MainWindow *ui;
15. private slots://声明槽函数
16.   void on_btnEditTime_clicked();
17.   void on_btnOk_clicked();
18.   void on_btnEditDate_clicked();
19.   void timeoutslot();
20. };
21. #endif // MAINWINDOW_H
```

声明四个槽函数,槽函数 timeoutslot()是自己声明的,对应定时器 timer;其他三个槽函数都是通过右击控件→Go to slots 自动生成的，所以不用在实现文件中添加映射函数。

在主窗体 mainwindow.cpp 文件中自动生成如下代码：

```
1. #include "mainwindow.h"
2. #include "ui_mainwindow.h"
3. MainWindow::MainWindow(QWidget *parent) :
4.   QMainWindow(parent),
5.   ui(new Ui::MainWindow)
6. {
7.   ui->setupUi(this);
8. }
9. MainWindow::~MainWindow()
10.
11.
12. {
13.   delete ui;
14. }
```

C

3.7.17 示例 18：DateEdit、TimeEdit 和 DateTimeEdit 的应用（2）

在主窗体 mainwindow.cpp 文件中添加的头文件：

```
1. #include <QTimer>
```

在主窗体 mainwindow.cpp 文件中构造函数：

```
1. /***构造函数***/
2. MainWindow::MainWindow(QWidget *parent) :
3.   QMainWindow(parent),
4.   ui(new Ui::MainWindow)
5. {
6.   ui->setupUi(this);
7.
8.   /*********初始化
   **********************************/
9.
   ui->dateTimeEdit->setDate(QDate::currentDate());
   //获取当前日期
10.
   ui->dateTimeEdit->setTime(QTime::currentTime());
   //获取当前时间
11.   QTimer *timer=new QTimer(this);
12.
   connect(timer,SIGNAL(timeout()),this,SLOT(timeou
   tslot()));//timeoutslot()为自定义槽
13.
```

```
14.   /*********启动 timer 定时器
      ****************************/
15.   timer->start(1000);
16. }
```

在主窗体 mainwindow.cpp 文件中添加 QPushButton btnEditDate 的槽函数：

```
1. /***********定义槽函数，编辑
   date**********************/
2. void MainWindow::on_btnEditDate_clicked()
3. {
4.   ui->dateEdit->setReadOnly(false);  //设置
   dateEdit 的只读属性为假
5.
   connect(ui->dateEdit,SIGNAL(dateChanged(QDate)),
   ui->dateTimeEdit, SLOT(setDate(QDate)));
6. }
```

在主窗体 mainwindow.cpp 文件中添加 QPushButton btnEditTime 的槽函数：

```
1. /**********定义槽函数，编辑时间
   ***************************/
2. void MainWindow::on_btnEditTime_clicked()
3. {
4.   ui->timeEdit->setReadOnly(false); //设置
   timeEdit 的只读属性为假
5.
   connect(ui->timeEdit,SIGNAL(timeChanged(QTime)),
   ui->dateTimeEdit, SLOT(setTime(QTime)));
6. }
```

在主窗体 mainwindow.cpp 文件中添加 QPushButton btnOk 的槽函数：

```
/***********定义槽函数，保存编辑
**************************/
void MainWindow::on_btnOk_clicked()
{
  ui->dateEdit->setReadOnly(true);
  ui->timeEdit->setReadOnly(true);
}
```

在主窗体 mainwindow.cpp 文件中添加 QTimer timer 的槽函数：

```
/***********定义 timer 的槽函数
*********************************/
void MainWindow::timeoutslot()
{
  QDate date = ui->dateTimeEdit->date();


  QTime time = ui->dateTimeEdit->time();
  QTime time1(23,59,59);
  if(time == time1)
    {
datedate = date.addDays(1);
    }
    timetime = time.addSecs(1);
    ui->dateTimeEdit->setDate(date);
    ui->dateTimeEdit->setTime(time);
}
```

主文件 main.cpp 没有进行任何更改，使用项目自动生成的即可。

4.示例执行结果

示例执行结果如图 3-84 所示。

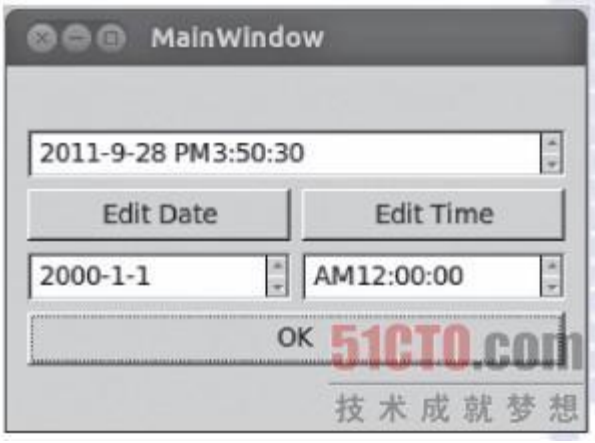


图 3-84　示例 18 执行结果

3.8　显示组件(Display Widgets)

Qt Designer 有十种 DisplayWidgets，如图 3-85 所示。

图 3-85　Display 控件

Display Widgets 的 Qt 类和名称介绍如表 3-23 所示。

表 3-23　Display Widgets 介绍

控件类	控件名	中文名	控件类	控件名	中文名
QLabel	Label	标签	QLCDNumber	LCDNumber	LCD 数字显示框
QTextBrowser	TextBrowser	文本浏览器	QProgressBar	ProgressBar	进度条
QGraphicsView	Graphics View	绘图视图	QLine	Horizontal Line	线条
QCalendarWidget	Calendar	日历控件	QLine	Vertical Line	线条

本节主要介绍 Qt Creator 的 Display Widgets 的使用方法。Qt Creator 比 Qt Designer 提供更多的 Display Widgets。

下面就来具体介绍各种 Display Widgets 的用法。

3.8.1　Label 控件

1.控件位置

Display Widgets→Label

2.控件介绍

Label 控件（标签）的样式如图 3-86 所示。同 Qt Designer 中的 TextLabel 控件。

图 3-86　Label 控件

3.控件设置选项

在 Label 控件的 properties 选项中，一般常对以下选项进行设置。

name：该控件对应源代码中的名称；

font：设置 text 的字体；

text：用来设置或返回标签控件中显示的文本信息。

4.常用成员函数

1.1) QLabel::QLabel (QWidget *parent, const char *name = 0, WFlags f = 0)

构造一个名称为 name、父对象为 parent 和标记符为 f 的 TextLabel。

1.2) QLabel::QLabel (const QString & text, QWidget *parent, const char *name = 0, WFlags f = 0)

构造一个名称为 name、父对象为 parent、标记符为 f 和内容为 text 的 TextLabel。

1.3) void QLabel::clear () [slot]

清除标签内容。

1.4) void QLabel::setText (const QString &) [virtual slot]

设置标签的文本。

1.5) QString QLabel::text () const

返回标签的文本。

3.8.2 TextBrowser 控件

1.控件位置

Display Widgets→TextBrowser

2.控件介绍

TextBrowser 控件（文本浏览器）的样式如图 3-87 所示，QTextBrowser 继承自 QTextEdit。TextBrowser 是只读的，不允许对内容进行更改，但是相对于 QTextEdit 来讲，它还具有链接文本的作用。

图 3-87 TextBrowser 控件

3.控件设置选项

在 TextBrowser 控件的 properties 选项中，一般常对以下选项进行设置。

name：该控件对应源代码中的名称；

font：设置显示框字体；

frameShape：边框样式，Qt Designer 提供了多项选择；

frameShadow:边框阴影,Qt Designer 提供的选择有 plain、raised、sunken；

lineWidth：边框线宽；

text：显示的文本；

source：显示的文件名称，如果没有文件显示或来源，则显示空字符串。

4.常用成员函数

1.1) QTextBrowser::QTextBrowser (QWidget *parent = 0, const char *name = 0)

构造一个名称为 name、父对象为 parent 的 TextBrowser。

1.2) void QTextBrowser::backward () [virtual slot]

更改内置导航链接的文件清单为显示前一个文档文件，如果没有以前的文档，就什么都不做，可以实现向前翻页的功能。

1.3) void QTextBrowser::forward () [virtual slot]

更改内置导航链接的文件清单为显示下一个文档文件，如果没有以前的文档，就什么都不做，可以实现向后翻页的功能。

1.4) void QTextBrowser::home () [virtual slot]

更改显示的文件浏览器中的链接，显示第一个文件。

1.5) void QTextBrowser::linkClicked (const QString & link) [signal]

当单击链接时，发射该信号。

1.6) void QTextBrowser::reload () [virtual slot]

重新载入当前的设置源。

1.7) void QTextBrowser::setSource (const QString & name) [virtual slot]

设置当前显示的文件名称为 name。

1.8) QString QTextBrowser::source () const

返回当前显示的文件的文件名称。

3.8.3 示例 19：TextBrower 的应用

QTextBrower 继承于 QTextEdit 类，增加了导航功能，可以用 TextBrower 打开超链接。下面通过一个示例来学习用 TextBrower 打开超链接的方法。

首先创建标准的 Qt Gui Application 项目，把项目在主界面中自动生成的 toolBar、menuBar 和 statusBar 删除，界面设计结果如图 3-88 所示。

图 3-88 示例 19 界面

1.控件说明

在属性编辑窗口中对控件的属性进行修改，修改内容如表 3-24 所示。

表 3-24　主要控件说明

控件类型	控件名称	控件说明
QTextBrower	tBwsShow	显示超链接

2.示例说明

程序执行后，会在 Text Brower 中添加三个超链接；

单击超链接，使用默认浏览器打开该超链接的网站。

3.示例实现

在主窗体 mainwindow.cpp 文件中自动生成如下代码：

```
#include "mainwindow.h"
#include "ui_mainwindow.h"
MainWindow::MainWindow(QWidget *parent) :
  QMainWindow(parent),
  ui(new Ui::MainWindow)
{
  ui->setupUi(this);
}
MainWindow::~MainWindow()
{
   delete ui;
}
```

在主窗体 mainwindow.cpp 文件中添加的头文件：

```
1. #include <QString>
```

在主窗体 mainwindow.cpp 文件中构造函数（文中的粗体为需要添加的内容）：

```
1. MainWindow::MainWindow(QWidget *parent) :
2.   QMainWindow(parent),
3.   ui(new Ui::MainWindow)
4. {
5.   ui->setupUi(this);
6.
7.   /*********打开 tBwsShow 的外部链接设置
   **************************/
8.   ui->tBwsShow->setOpenLinks(true);
9.   ui->tBwsShow->setOpenExternalLinks(true);
10.
11.   /*********添加链接
   ************************************************/


12.
   ui->tBwsShow->append(QString::fromLocal8Bit("<a
   href=\"http://www.baidu.com\">baidu</a>"));
13.
   ui->tBwsShow->append(QString::fromLocal8Bit("<a
   href = \"http://www.sina.com.cn/\">sina</a>"));


14.
   ui->tBwsShow->append(QString::fromLocal8Bit("<a
   href = \"http://www.qq.com/\">qq</a>"));
15. }
```

主文件 main.cpp 没有进行任何更改，使用项目自动生成的即可。

4.示例执行结果

示例执行结果如图 3-89 所示。

图 3-89　示例 19 执行结果

3.8.4　GraphicsView 控件

1.控件位置

Display Widgets→GraphicsView

2.控件介绍

GraphicsView（绘图视图）是用于显示 QGraphicsScene 内容的控件。学习该控件就不得不了解 GraphicsView 框架，GraphicsView 框架为 2D 绘图提供一个简单、容易使用、功能强大的解决方案。Graphics View 框架由三个主要的类组成：QGraphicsItem、QGraphicsScene 和 QGraphicsView。其中 QGraphicsItem 定义图元；QGraphicsScene 定义场景，包含所有需要绘制的图元，根据用户的操作改变图元的状态；QGraphicsView 定义观察

场景的视窗，可以充当绘图的区域，成为独立的窗体被弹出，或者嵌入其他 UI 组件中形成复合 UI 组件。

3.控件设置选项

在 GraphicsView 控件的 properties 选项中，一般常对以下选项进行设置。

name：该控件对应源代码中的名称；

font：设置显示框字体；

frameShape：边框样式；

frameShadow：边框阴影，Qt Creator 提供的选择有 plain、raised、sunken；

lineWidth：边框线宽。

4.常用成员函数

1.1) QGraphicsView::QGraphicsView (QWidget *parent = 0)

构造一个父对象为 parent 的 GraphicsView。

1.2) QGraphicsView::QGraphicsView (QGraphicsScene *scene, QWidget *parent = 0)

构造一个父对象为 parent 的 GraphicsView，填充场景 scene 到该 GraphicsView。

1.3) void QGraphicsView::centerOn (const QPointF & pos)

调整视窗的内容，确保点 pos 在视窗中居中。

1.4) void QGraphicsView::centerOn (qreal x, qreal y)

这是一个重载函数，函数功能同函数 3)，相当于调用 center(QPointF(x,y))。

1.5) void QGraphicsView::centerOn (const QGraphicsItem *item)

这是一个重载函数，函数功能同函数 3),调整视区的内容，使项目 item 为中心。

1.6) void QGraphicsView::ensureVisible (const QRectF & rect, int xmargin = 50, int ymargin = 50)

调整视窗的内容，使视窗 rect 中的内容是可见的。

1.7) void QGraphicsView::ensureVisible (qreal x, qreal y, qreal w, qreal h, int xmargin = 50, int ymargin = 50)

这是一个重载函数，函数功能同函数 6)，相当于调用 ensureVisible(QRectF(x,y,w,h),xmargin,ymargin)。

1.8) void QGraphicsView::ensureVisible (const QGraphicsItem *item, int xmargin = 50, int ymargin = 50)

这是一个重载函数，函数功能同函数 6)，调整视窗的内容，使 item 的内容可见。

1.9) QGraphicsItem *QGraphicsView::itemAt (const QPoint & pos) const

返回 pos 处的 item。

1.10) QGraphicsItem *QGraphicsView::itemAt (int x, int y) const

这是一个重载函数，函数功能同函数 9)，返回坐标(x,y)处的 item。

```
1. 11) QList<QGraphicsItem *> QGraphicsView::items
   () const
```

返回相关场景中所用的图元。

```
1. 12) QList<QGraphicsItem *> QGraphicsView::items
   ( const QPoint & pos ) const
```

返回视图中位置 pos 处的项目列表。

```
1. 13) QGraphicsScene *QGraphicsView::scene () cons
   t
```

返回当前可视化的场景；如果当前没有可视化的场景，则返回 0。

```
1. 14) void QGraphicsView::setScene ( QGraphicsScen
   e *scene )
```

设置场景 scene 为当前可视化场景。

```
1. 15) void QGraphicsView::updateScene ( const QLis
   t<QRectF> & rects ) [slot]
```

面更新场景。

《Linux 环境下 Qt4 图形界面与 MySQL 编程》本书以“图形界面编程控件与数据库编程基础→简单易学的实例→实际工程项目开发与场景分析”为写作主线，以当前最新的 Qt4.7 为依据，采用“深入分析控件+实例解析”的方式，并配合经典的实际工程项目，对 Linux 操作系统下的 Qt4.7 与 MySQL 编程技术进行了全面细致的讲解。本节为大家介绍示例 20：GraphicsView 的应用。

3.8.5 示例 20：GraphicsView 的应用（1）

本示例通过用 GraphicsView 显示图片文件，简单地介绍了 Graphics View、Graphics Scene 和 QGraphicsItem 之间的关系。

首先创建标准的 Qt Gui Application 项目，把项目在主界面中自动生成的 toolBar、menuBar 和 statusBar 删除，界面设计结果如图 3-90 所示。

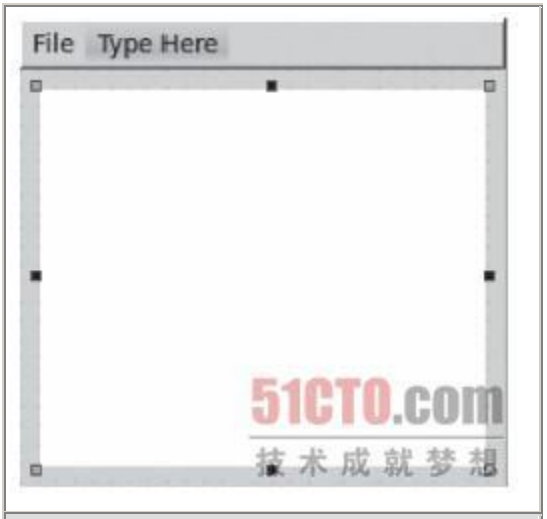


图 3-90 示例 20 界

1.控件说明

在属性编辑窗口中对控件的属性进行修改，修改内容如表 3-25 所示。

表 3-25 主要控件说明

控件类型	控件名称	控件说明	控件类型	控件名称	控件说明
QGraphicsView	view	主控件	QAction	actionOpen	打开图片
QMenuBar	menuBar	菜单栏	QAction	actionExit	退出程序

2.示例说明

界面布局设置好之后，添加资源文件，把图片文件 1.jpg 添加到资源文件中。添加资源文件的方法在示例 3 中已经详细介绍过了，这里就不再重复介绍。

程序执行后，自动加载图片 1.jpg，不要把程序最大化，使边框出现滚动条，可以通过鼠标在一定空间内拖动图片 1.jpg；

单击菜单栏选择“Open”选项，可以打开其他图片；

单击菜单栏选择“Exit”选项，退出程序。

3.示例实现

头文件 mainwindow.h（文中的粗体为需要添加的内容）：

```
1.#ifndef MAINWINDOW_H
2.#define MAINWINDOW_H
3.#include <QMainWindow>
4.#include <QGraphicsScene>
5.#include <QGraphicsView>
6.namespace Ui {
7.  class MainWindow;
8.}
9.class MainWindow : public QMainWindow
10.{
11.  Q_OBJECT
12.public:
13.  explicit MainWindow(QWidget *parent = 0);
14.  ~MainWindow();
```

```
15.   QString pictureName;//当前显示的图片
16.   QGraphicsScene *scene;//声明 scene
17.   QGraphicsPixmapItem *item;//声明 item
18. private:
19.   Ui::MainWindow *ui;
20. private slots://声明槽函数
21.   void slotOpen();
22.   void slotExit();
23. };
24. #endif // MAINWINDOW_H
```

在主窗体 mainwindow.cpp 文件中自动生成如下代码：

```
1. #include "mainwindow.h"
2. #include "ui_mainwindow.h"
3. MainWindow::MainWindow(QWidget *parent) :
4.   QMainWindow(parent),
5.   ui(new Ui::MainWindow)
6. {
7.   ui->setupUi(this);
8. }
9. MainWindow::~MainWindow()
10. {
11.   delete ui;
12. }
```

3.8.5 示例 20：GraphicsView 的应用（2）

在主窗体 mainwindow.cpp 文件中添加的头文件：

```
1. #include <QFileDialog>
```

在主窗体 mainwindow.cpp 文件中构造函数：

```
1. MainWindow::MainWindow(QWidget *parent) :
2.   QMainWindow(parent),
3.   ui(new Ui::MainWindow)
4. {
5.   ui->setupUi(this);
6.
7.   /****************界面初始化
   *******************************/
8.   this->pictureName = ":/1.jpg";
9.   scene = new QGraphicsScene;
10.    scene->setSceneRect(-300,-300,600,600);
11.
   scene->setItemIndexMethod(QGraphicsScene::NoInde
   x);
12.    QPixmap pixmap(this->pictureName);
13.    pixmappixmap = pixmap.scaledToWidth(200);
14.    item = scene->addPixmap(pixmap);
15.
16.    /***************设置 view 的场景
   ****************************/
17.    ui->view->setScene(scene);
18.
   ui->view->setRenderHint(QPainter::Antialiasing);


19.
   ui->view->setCacheMode(QGraphicsView::CacheBackg
   round);
20.
   ui->view->setViewportUpdateMode(QGraphicsView::B
   oundingRectViewportUpdate);
```

```
21.
    ui->view->setDragMode(QGraphicsView::ScrollHandDrag);
22.   ui->view->resize(400,300);
23.
24.   /***************信号和槽函数的映射**************************/
25.
    connect(ui->actionOpen,SIGNAL(triggered()),this,SLOT(slotOpen()));
26.
    connect(ui->actionExit,SIGNAL(triggered()),this,SLOT(slotExit()));
27. }
28.
29. 在主窗体 mainwindow.cpp 文件中添加 QAction actionOpen 的槽函数：
30.
31. /*****************定义槽函数，打开图片**************************/
32. void MainWindow::slotOpen()
33. {
34.
    this->pictureName = QFileDialog::getOpenFileName(this);//打开文件对话框
35.   if(!this->pictureName.isEmpty())
36.   {
37. QPixmap pixmap(this->pictureName);
38. pixmappixmap = pixmap.scaledToWidth(200);
39. item = scene->addPixmap(pixmap);
```

```
40. ui->view->setScene(scene);//设置 view 的场景
41.   }
42. }
```

在主窗体 mainwindow.cpp 文件中添加 QAction actionExit 的槽函数：

```
1. /*******************定义槽函数，退出程序
   ************************/
2. void MainWindow::slotExit()
3. {
4.   this->close();
5. }
```

4. 示例执行结果

示例执行结果如图 3-91 所示。

图 3-91　示例 20 执行结果

3.8.6　Calendar 控件

1. 控件位置

Display Widgets→Calendar

2.控件介绍

Calendar 控件（日历）的样式如图 3-92 所示。顾名思义，这是一个和日历有关的控件，使用起来很简单，也很方便。

图 3-92　Calendar 控件 3.控件设置选项

在 Calendar 控件的 properties 选项中，一般常对以下选项进行设置。

name：该控件对应源代码中的名称；

font：设置 text 的字体；

selectedDate：当前日期；

minimumDate：该日历控件能够显示的最小日期；

maxmumDate：该日历控件能够显示的最大日期；

firstDayOfWeek：每星期的第一天；

gridVisible：是否显示网格；

dateEditEnable：是否允许编辑日历。

4.常用成员函数

1.1) QCalendarWidget::QCalendarWidget (QWidget *parent = 0)

构造一个父对象为 parent 的 Calendar。

1.2) int QCalendarWidget::monthShown () const

返回当前显示的月份。

1.3) void QCalendarWidget::setCurrentPage (int year, int month) [slot]

显示给定的年份和月份。

1.4) void QCalendarWidget::setDateRange (const QDate & min, const QDate & max) [slot]

设置该 Calendar 的显示范围，最小日期为 min，最大为 max。

1.5) void QCalendarWidget::showSelectedDate () [slot]

显示当前选中的日期。

1.6) void QCalendarWidget::showToday () [slot]

显示系统当前日期。

1.7) int QCalendarWidget::yearShown () const

返回当前显示的年份。

3.8.7 示例 21：Calendar 的应用

首先创建标准的 Qt Gui Application 项目，把项目在主界面中自动生成的 menuBar、toolBar 和 statusBar 删除，界面设计结果如图 3-93 所示。

图 3-93　示例 21 界面

1.控件说明

在属性编辑窗口中对控件的属性进行修改，修改内容如表 3-26 所示。

图 3-94　信号与槽的映射

2.示例说明

在 Signals & Slots Editor 中添加槽函数如图 3-94 所示；

程序执行后，只显示 dateEdit 和 pushButton 两个控件；

单击按钮“Edit Date”，显示 calendarWidget 控件，可以通过 calendarWidget 控件来编辑 dateEdit 的日期。在 calendarWidget 控件上选定日期并单击之后，dateEdit 的日期随之改变。同时 calendarWidget 控件隐藏。

表 3-26　主要控件说明

控件类型	控件名称	控件说明
QCalendarWidget	calendarWidget	日历控件，可以编辑日期
QDateEdit	dateEdit	显示日期
QPushButton	pushButton	开始编辑日期

3.示例实现

头文件 mainwindow.h（文中的粗体为需要添加的内容）：

```
1.#ifndef MAINWINDOW_H
2.#define MAINWINDOW_H
3.#include <QMainWindow>
4.namespace Ui {
5.  class MainWindow;
6.}
7.class MainWindow : public QMainWindow
8.{
9.  Q_OBJECT
10.public:
11.  explicit MainWindow(QWidget *parent = 0);
```

```
12.   ~MainWindow();
13. private:
14.   Ui::MainWindow *ui;
15. private slots: //声明槽函数
16.   void on_calendarWidget_clicked(QDate date);
17. };
18. #endif // MAINWINDOW_H
```

声明一个槽函数 on_calendarWidget_clicked(QDate date)，这个槽函数是通过右击 calendarWidget→Go to slot 自动生成的，对应 calendarWidget 的 clicked(QDate date)信号。

在主窗体 mainwindow.cpp 文件中自动生成如下代码：

```
1. #include "mainwindow.h"
2. #include "ui_mainwindow.h"
3. MainWindow::MainWindow(QWidget *parent) :
4.   QMainWindow(parent),
5.   ui(new Ui::MainWindow)
6. {
7.   ui->setupUi(this);
8. }
9. MainWindow::~MainWindow()
10. {
11.   delete ui;
12. }
```

在主窗体 mainwindow.cpp 文件中构造函数：

```
1. MainWindow::MainWindow(QWidget *parent) :
2.   QMainWindow(parent),
3.   ui(new Ui::MainWindow)
4. {
5.   ui->setupUi(this);
```

```
6.
7.   /*****************隐藏日历控件
     *********************/
8.   ui->calendarWidget->hide();
9. }
```

在主窗体 mainwindow.cpp 文件中添加 QCalendarWidget calendarWidget 的槽函数：

```
1. /*********************定义槽函数，编辑日期
   ******************/
2. void MainWindow::on_calendarWidget_clicked(QDate
    date)
3. {
4.
   ui->dateEdit->setDate(ui->calendarWidget->select
   edDate());
5.   ui->calendarWidget->hide();
6. }
```

主文件 main.cpp 没有进行任何更改，使用项目自动生成的即可。

4.示例执行结果

示例执行结果如图 3-95 所示。

图 3-95　示例 21 执行结果

3.8.8　LCDNumber 控件

1.控件位置

Display Widgets→LCDNumber

2.控件介绍

图 3-96　LCDNumber 控件

LCDNumber 控件（LCD 数字显示框）的样式如图 3-96 所示，可以显示十六进制、十进制、八进制或二进制数。

3.控件设置选项

在 LCDNumber 控件的 properties 选项中，一般常对以下选项进行设置。

name：该控件对应源代码中的名称；

font：设置显示框字体；

frameShape：边框样式，Qt Designer 提供了多项选择。

frameShadow:边框阴影,Qt Designer 提供的选择有 plain、raised、sunken。

lineWidth：边框线宽；

mode：设置显示格式，Qt 提供的格式有十六进制、十进制、八进制、二进制。

value：该 LCDNumber 的值。

intValue：该 LCDNumber 的整数值；

numDigits：显示框最大可以显示的数字位数。

4.常用成员函数

```
1.1) QLCDNumber::QLCDNumber ( QWidget *parent = 0,
    const char *name = 0 )
```

构造一个名称为 name、父对象为 parent 的 LCD Number。

```
1.2) QLCDNumber::QLCDNumber ( uint numDigits, QWid
  get *parent = 0, const char *name = 0 )
```

构造一个名称为 name、父对象为 parent 和显示框最大可以显示的数字位数为 numDigits 的 LCDNumber。如下构造了一个最多能显示 4 位数的 LCD 显示器：

```
1. QLCDNumber *lcd= new QLCDNumber (4,this);
2. 3) void QLCDNumber::display ( int num ) [slot]
```

设置显示的值为 num。

```
1. 4) void QLCDNumber::display ( const QString & s
   ) [slot]
```

这是一个重载函数，函数功能同函数 4)，显示 s。

```
1. 5) void QLCDNumber::display ( double num ) [slo
   t]
```

这是一个重载函数，函数功能同函数 5)，显示 num。

```
1. 6) int QLCDNumber::intValue () const
```

返回显示值的整数值，对应 intValue 属性。

```
1. 7) int QLCDNumber::numDigits () const
```

返回显示框最大可以显示的数字位数，对应 numDigits 属性。

```
1. 8) double QLCDNumber::value () const
```

返回该 LCDNumber 显示的值。

3.8.9 示例 22：LCDNumber 的应用

首先创建标准的 Qt Gui Application 项目，把项目在主界面中自动生成的 menuBar、toolBar 和 statusBar 删除，界面设计结果如图 3-97 所示。

图 3-97　示例 22 界面

1.控件说明

在属性编辑窗口中对控件的属性进行修改，修改内容如表 3-27 所示。

表 3-27　主要控件说明

控件类型	控件名称	控件说明
QLCDNumber	lcdNumber	LCD 显示数字
QSlider	horizontalSlider	水平滑动条
QSpinBox	spinBox	旋转框

2.示例说明

在 Signals & Slots Editor 中添加槽函数如图 3-98 所示；

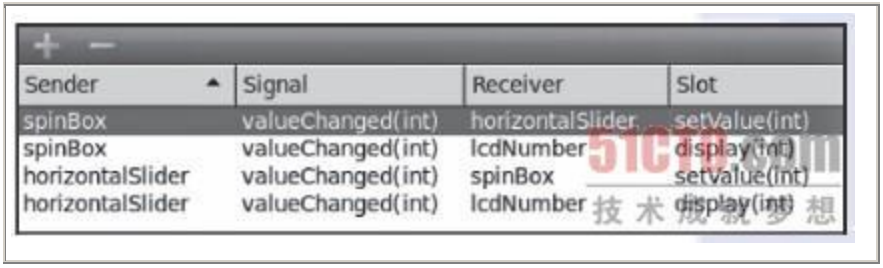


图 3-98　信号和槽的映射

程序执行后，改变 horizontalSlider 的值，lcdNumber 和 spinBox 的值随着改变；

改变 spinBox 的值，lcdNumber 和 horizontalSlider 的值随着改变。

3.示例实现

本示例不需要添加任何代码，只需新建一个 Qt Gui Application 项目，把项目在主界面中自动生成的 menuBar、toolBar 和 statusBar 删除，按照以上步骤添加完成即可。

4.示例执行结果

示例执行结果如图 3-99 所示。

图 3-99　示例 22 执行结果

3.8.10 ProgressBar 控件

1.控件位置

Display Widgets→ProgressBar

2.控件介绍

ProgressBar 控件（进度条）的样式如图 3-100 所示，显示为一个水平进度条。一个进度条是用来给用户显示操作进度的，证明他们的应用程序仍在运行，进度条控件非常常见，例如我们下载资源时经常会显示进度条来显示下载进度。

图 3-100 ProgressBar 控件面

3.控件设置选项

在 ProgressBar 控件的 properties 选项中，一般常对以下选项进行设置。

name：该控件对应源代码中的名称；

font：设置显示框字体；

frameShape：边框样式，Qt Designer 提供了多项选择；

frameShadow:边框阴影,Qt Designer 提供的选择有 plain、raised、sunken。

lineWidth：边框线宽；

progress：进度条当前的进度值。

4.常用成员函数

```
1. 1) QProgressBar::QProgressBar ( QWidget *parent
   = 0, const char *name = 0, WFlags f = 0 )
```

构造一个名称为 name、父对象为 parent 的 ProgressBar。

```
1. 2) QProgressBar::QProgressBar ( int totalSteps,
   QWidget *parent = 0, const char *name = 0, WFlag
   s f = 0 )
```

构造一个名称为 name、父对象为 parent 和总的进度值为 totalSteps 的 ProgressBar。如下构造了一个进度值为 100 的进度条：

```
1. QProgressBar *pgb= new QProgressBar (100,this);
```

```
2. 3) int QProgressBar::progress () const
```

返回该进度条的当前进度值。

```
1. 4) void QProgressBar::setProgress ( int progress
   ) [virtual slot]
```

设置该进度条的当前进度值为 progress。

```
1. 5) void QProgressBar::reset () [slot]
```

把进度条恢复到初始状态。

3.8.11　示例 23：ProgressBar 的应用（1）

首先创建标准的 Qt Gui Application 项目，把项目在主界面中自动生成的 menuBar、toolBar 和 statusBar 删除，界面设计结果如图 3-101 所示。

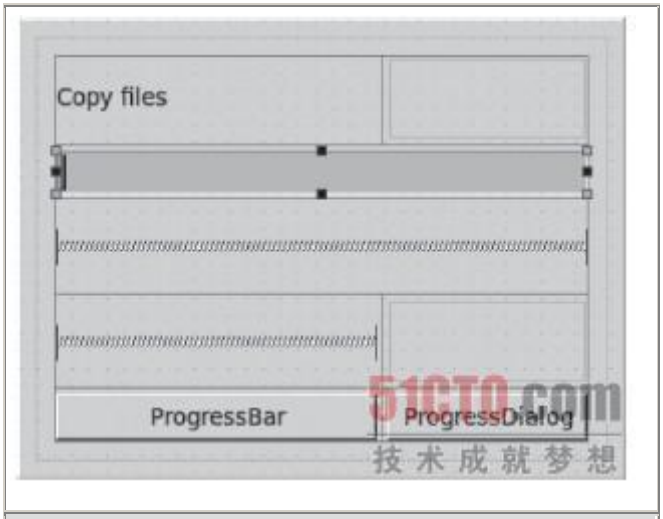


图 3-101　示例 23 界面

1.控件说明

在属性编辑窗口中对控件的属性进行修改，修改内容如表 3-28 所示。

2.示例说明

单击按钮“ProgressBar”，progressBar 的值在 10 秒内从 0 变为 10；

单击按钮“ProgressDialog”，弹出一个有进度条的窗口，进度条在 10 秒内从 0 变为 10。如图 3-102 所示，单击按钮“Cancel”，强制结束窗口。

图 3-102　ProgressDialog

表 3-28　主要控件说明

控件类型	控件名称	控件说明
QLabel	label	标签显示“Copy files”
QProgressBar	progressBar	显示进度，最小值 0，最大值 10
QPushButton	btnProgressBar	使进度条 progressBar 的值开始变化
QPushButton	btnProgressDialog	开启 ProgressDialog

3. 示例实现

头文件 mainwindow.h（文中的粗体为需要添加的内容）：

```
1. #ifndef MAINWINDOW_H
2. #define MAINWINDOW_H
3. #include <QMainWindow>
4. namespace Ui {
5.   class MainWindow;
6. }
7. class MainWindow : public QMainWindow
8. {
9.   Q_OBJECT
```

```
10. public:
11.   explicit MainWindow(QWidget *parent = 0);
12.   ~MainWindow();
13.   int num;//文件数目
14. private:
15.   Ui::MainWindow *ui;
16. private slots://声明槽函数
17.   void on_btnProgressDialog_clicked();
18.   void on_btnProgressBar_clicked();
19. };
20. #endif // MAINWINDOW_H
```

声明一个变量 num 和两个槽函数。num 是控件 progressBar 当前的值，两个槽函数分别对应两个按钮的 clicked()信号。这两个槽函数都是通过右击控件→Go to slot 生成的，所以不用在实现函数中写信号与槽的映射代码。

在主窗体 mainwindow.cpp 文件中自动生成如下代码：

```
1. #include "mainwindow.h"
2. #include "ui_mainwindow.h"
3. MainWindow::MainWindow(QWidget *parent) :
4.   QMainWindow(parent),
5.   ui(new Ui::MainWindow)
6. {
7.   ui->setupUi(this);
8. }
9. MainWindow::~MainWindow()
10. {
11.   delete ui;
12. }
```

3.8.11 示例 23：ProgressBar 的应用（2）

在主窗体 mainwindow.cpp 文件中添加的头文件：

```
#include <QProgressDialog>
#include<QThread>
```

在主窗体 mainwindow.cpp 文件中构造函数：

```
/***构造函数***/
MainWindow::MainWindow(QWidget *parent) :
  QMainWindow(parent),
  ui(new Ui::MainWindow)
{
  ui->setupUi(this);
  num=10;//初始化
}
```

在主窗体 mainwindow.cpp 文件中添加 QPushButton btnProgressBar 的槽函数：

```
/***槽函数：进度条***/
void MainWindow::on_btnProgressBar_clicked()
{
  ui->progressBar->setRange(0,num);//设置
  progressBar 的取值范围
  for(int i=1;i<=num;i++)
  {
ui->progressBar->setValue(i);//给进度条赋值
sleep(1);
```

```
9.   }
10. }
```

在主窗体 mainwindow.cpp 文件中添加 QPushButton btnProgressBarDialog 的槽函数：

```
1. /***槽函数：进度条对话框***/
2. void MainWindow::on_btnProgressDialog_clicked()
3. {
4.   QProgressDialog *progressDialog = new QProgressDialog(this);
5.   QFont font("ZYSong18030",12);  //定义字体
6.   progressDialog->setFont(font);//设置字体
7.   progressDialog->setWindowModality(QT::WindowModal);
8.   progressDialog->setMinimumDuration(5);
9.   progressDialog->setWindowTitle("Please Waiting");
10.
11.
12.  progressDialog->setLabelText(tr("Copying..."));
13.  progressDialog->setCancelButtonText(tr("Cancel"));
```

```
14.   progressDialog->setRange(0,num);//设置进度对话框
      的进度条的取值范围
15.   for(int i=1;i<=num;i++)
16.   {
17. progressDialog->setValue(i);//对进度对话框的进度条
      赋值
18. qApp->processEvents();
19. sleep(1);
20. if(progressDialog->wasCanceled())
21.   return;
22.   }
23. }
```

主文件 main.cpp 没有进行任何更改，使用项目自动生成的即可。

4.示例执行结果

示例执行结果如图 3-103 和图 3-104 所示。

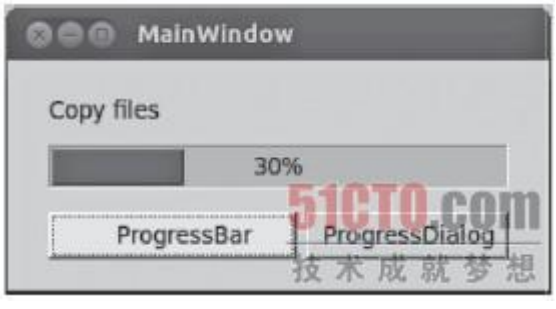


图 3-103　单击按钮“ProgressBar”的结果

图 3-104　单击按钮“ProgressDialog”的结果

3.8.12　Line 控件

1.控件位置

Display Widgets→Horizontal/Vertical Line

2.控件介绍

Line 控件（线条）的样式如图 3-105 所示，它是用来装饰界面的控件，在界面中适当添加 Line 控件可以使界面更加美观、清晰。

图 3-105　Line 控件

3.控件设置选项

在 Line 控件的 properties 选项中，一般常对以下选项进行设置。

name：该控件对应源代码中的名称；

frameShadow:线条阴影,Qt Designer 提供的选择有 plain、raised、sunken;

lineWidth：线条线宽;

orientation：线条布局方向。

Line 控件就是一个装饰控件，使用简便，在此不再进行详细介绍。