

第1部分 Qt 基础

第 1 章 Qt 概述

本章介绍什么是 Qt，如何安装 Qt 及其开发环境。通过一个计算圆面积的小实例详细介绍 Qt 的开发步骤，使读者对利用 Qt 进行 GUI 应用程序（Qt Designer）开发有一个初步的认识。同时，通过信号和槽机制（Signal & Slot）完成 Qt 应用程序的用户界面操作的响应，介绍事件关联的问题。本书提供大部分例子的源代码，如代码 CH101 就是第 1 章的第一个例子的源代码，以后以此类推。提供这些代码是为了方便读者上机模仿，书中的内容本身是系统的、完备的。



1.1 什么是 Qt

Qt 是一个跨平台的 C++ 图形用户界面应用程序框架。它为应用程序开发者提供建立艺术级图形用户界面所需的所有功能。它是完全面向对象的，很容易扩展，并且允许真正的组件编程。

1. Qt 的发展

Qt 是在 1991 年由奇趣科技公司开发的，于 1996 年进入商业领域，成为全世界范围内数千种成功的应用程序的基础。它也是目前流行的 Linux 桌面环境 KDE 的基础，KDE 是 Linux 发行版的主要一个标准组件。2008 年，奇趣科技公司被诺基亚公司收购，Qt 成为诺基亚旗下的编程语言工具。从 2009 年 5 月发布的 Qt 4.5 版起，诺基亚公司宣布 Qt 源代码库面向公众开放，Qt 开发人员可通过为 Qt 及与其相关的项目贡献代码、翻译、示例及其他内容，协助引导和塑造 Qt 的未来发展。2011 年，Digia 公司（芬兰的一家 IT 服务公司）从诺基亚公司收购了 Qt 的商业版权。2012 年 8 月 9 日，作为非核心资产剥离计划的一部分，诺基亚公司宣布将 Qt 软件业务正式出售给 Digia 公司。2013 年 7 月 3 日，Digia 公司 Qt 开发团队在其官方博客上宣布 Qt 5.1 正式版发布；同年 12 月 11 日，又发布 Qt 5.2 正式版。2014 年 4 月，跨平台集成开发环境 Qt Creator 3.1.0 正式发布；同年 5 月 20 日，配套发布了 Qt 5.3 正式版。至此，Qt 实现了对于 iOS、Android、WP 等各种平台的全面支持。The Qt Company 公司成立后，Qt 版本的升级开始加速，相继推出 Qt 5.4/5.5/5.6/5.7 版，并在官网提供免费下载。2017 年 1 月 23 日，众所期待的 Qt 5.8 正式版如期发布，它集中体现了之前各版 Qt 产品的优势，进一步扩展了 Qt 的框架，提升了速度和性能。



尤其对 Qt 的 Qt Quick Controls 2 库及 3D 支持方面进行了重大优化和升级。2018 年 5 月，Qt 发布了最新的 Qt 5.11 版。

2. Qt 5.11 版的改进

(1) 对 Qt Core 进行完善，更好地支持 Unicode。在 Qt Network 中，iOS 可支持 ALPN 和 HTTP/2。

(2) Qt GUI 基于 Microsoft UI Automation，并且更好地支持高 DPI 显示。改进 Linux 上的打印对话框，为 CUPS 的选项提供了更好的支持。在 QLineEdit 中通过鼠标快速选择文本。

(3) Qt QML 编译器管道负责解析和编译 QML，性能提高、更可维护。将 QML 编译为与平台无关的字节码。

(4) 在 Qt Quick 中，支持在 Image 元素中加载压缩纹理，支持 .ktx 和 .pkm 容器文件格式，可以通过 GPU 直接处理，减少应用程序启动时间和内存消耗。完善 Qt Quick Controls 2，新增了一些小功能。

(5) Qt Location 实现对逐向导航的支持，通过 API 来创建不绑定到 QQuickItems 的地图对象。改进 MapPolyline 对象的性能，图层支持与 Map 组件结合。Routing and Places API 增加了 WayPoint 元素 MapBox 插件对地理编码和地点的支持。

(6) 更新 Qt Webengine 的 Chromium 版本。支持嵌入式 DevTools，包含可安装的 Cookie 过滤器和配额权限。

(7) 使用 Qt for Webassembly，允许用户将 Web 和浏览器作为 Qt 应用程序的平台。

(8) 支持基于硬件的图形层和 VSP2 硬件合成平台的技术预览版，有助于提高视频性能并降低功耗。完善 Qt Serialbus 和 Bluetooth 对 CAN 总线和 BTLE 的支持。KNX 模块增加对 OPC/UA 的支持。通过 QDoc 使用 libclang 解析 C++，更好地支持现代 C++。

(9) 不再支持 MSVC 2013、QNX 6.6 和 Mac OS 10.10。

3. Qt 版本说明

Qt 按照不同的版本发行，分为商业版和开源版。Qt 商业版为商业软件提供开发环境，它们提供传统商业软件发行版，并且提供在协议有效期内的免费升级和技术支持服务。而 Qt 开源版是为了开发自由而设计的开放源代码软件，它提供了和商业版本同样的功能，在 GNU 通用公共许可证下，它是免费的。



1.2 Qt 5 的安装

1.2.1 下载 Qt 和申请免费账号

1. 下载 Qt 5.11

在 Qt 官方网站下载 Qt 5.11 安装包，Qt 5.11 官方下载地址如下：

<http://www.qtcn.org/bbs/read-hm-tid-1075.html>

Qt 5.9 之后的安装包与之前相比，不再区分 VS 版本和 MinGW 版本，而是全都整合到一

个安装包中。因此，与之前的安装包相比，体积也大了不少，以前是 1GB 多，现在是 2GB 多。

选择 Windows Host 下的 Qt 5.11.0 for Windows(2.6GB)，如图 1.1 所示。

下载完成后，得到安装包文件名为 qt-opensource-windows-x86-5.11.1.exe。

读者也可访问 QTCN 开发网 (<http://www.qtcn.org/bbs/i.php>) 等下载 Qt 各个版本的安装包。QTCN 开发网又名“Qt 中文论坛”，始建于 2005 年，面向广大初、中级 Qt 开发者，是目前最为活跃的 Qt 综合技术中文讨论区。



图 1.1 Qt 5 下载页面

2. 申请免费账号

登录 The Qt Company 公司官网 (<https://www.qt.io>)，单击 **Start free Qt trial**，选择 **Try free**，进入 Qt 申请免费账号页，如图 1.2 所示。

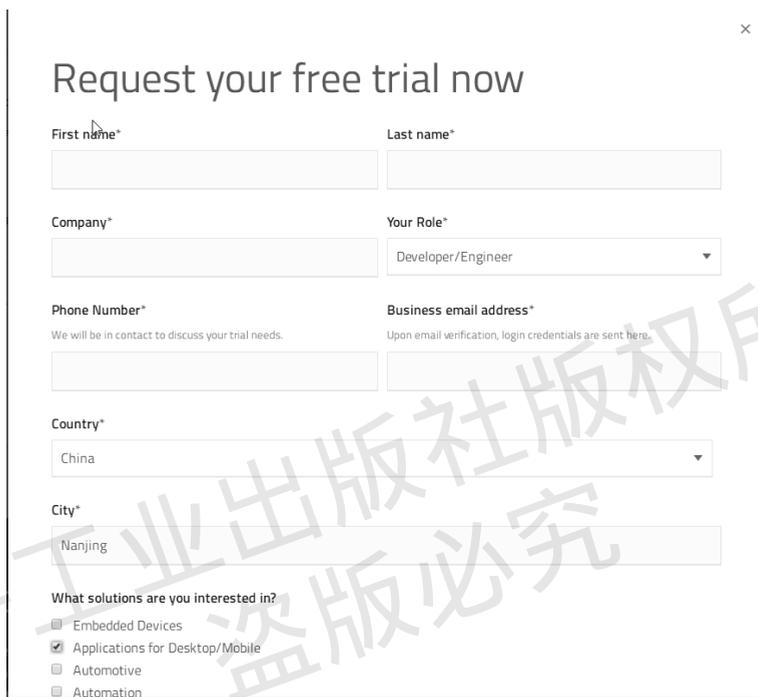


图 1.2 Qt 申请免费账号页



可以用自己的邮箱名作为 Qt 的免费账号。填写完成后，单击 **Submit my request now**，申请免费账号。

1.2.2 安装 Qt 5.11

双击启动安装向导，进入 Qt 5.11.1 安装欢迎页，如图 1.3 所示。

单击“Next”按钮，出现如图 1.4 所示的界面，输入 Qt 免费账号并设置密码。



图 1.3 Qt 5.11.1 安装欢迎页



图 1.4 输入 Qt 免费账号并设置密码

单击“Next”按钮，进入“设置-Qt5.11.1”页。单击“下一步”按钮，进入“安装文件夹”页，系统列出 Qt 的默认安装路径，用户可以修改，并勾选“Associate common file types with Qt Creator.”复选框，如图 1.5 所示。单击“下一步”按钮，进入“选择组件”页，单击“全选”按钮选择安装全部组件，如图 1.6 所示。

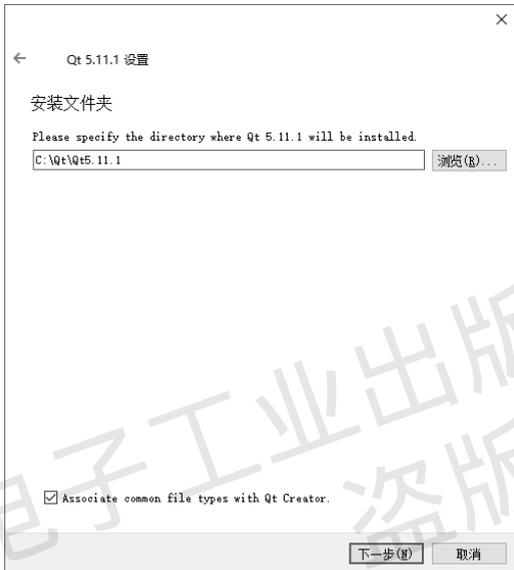


图 1.5 选择 Qt 安装路径

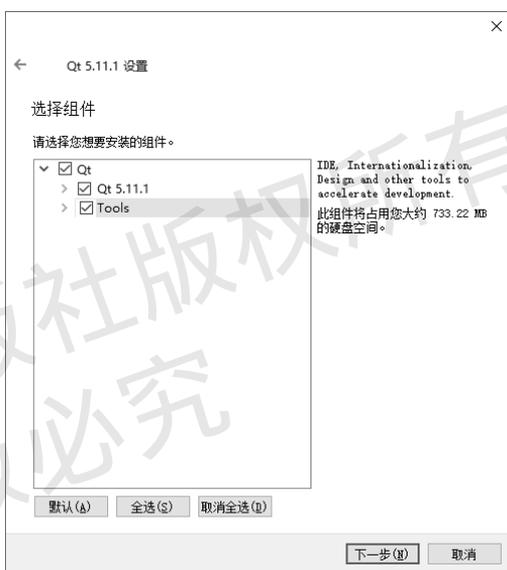


图 1.6 选择安装 Qt 5.11.1 的全部组件

单击“Next”按钮，在“许可协议”页中选中“I have read and agree to the terms contained in the license agreements.”接受 Qt 软件许可协议，如图 1.7 所示。

单击“下一步”按钮，进入“开始菜单快捷方式”页，可修改系统默认的开始菜单，如图 1.8 所示。

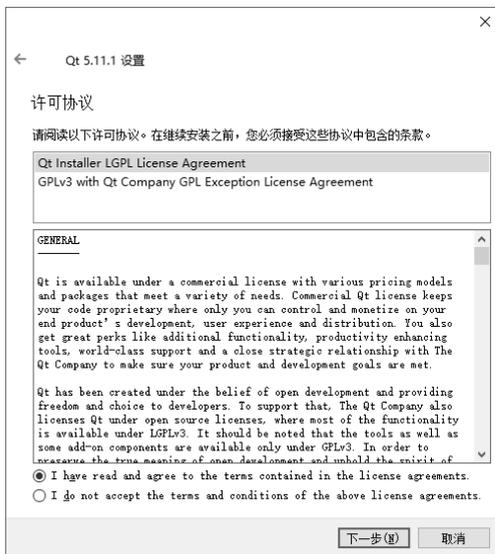


图 1.7 接受 Qt 软件许可协议



图 1.8 “开始菜单快捷方式”页

单击“下一步”按钮，进入“已做好安装准备”页，如图 1.9 所示。

单击“安装”按钮，系统进行安装，如图 1.10 所示。

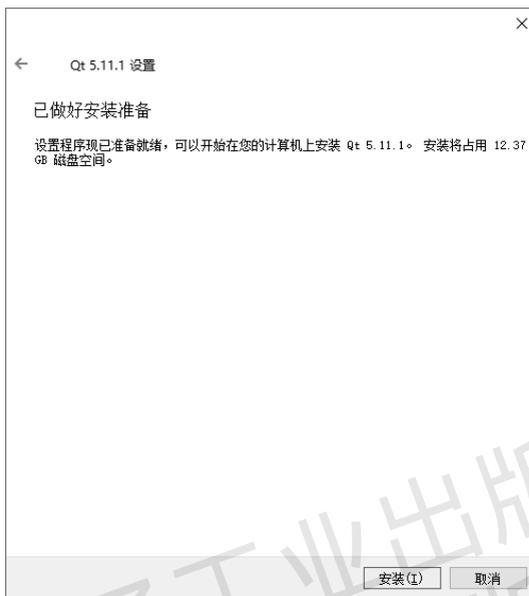


图 1.9 “已做好安装准备”页

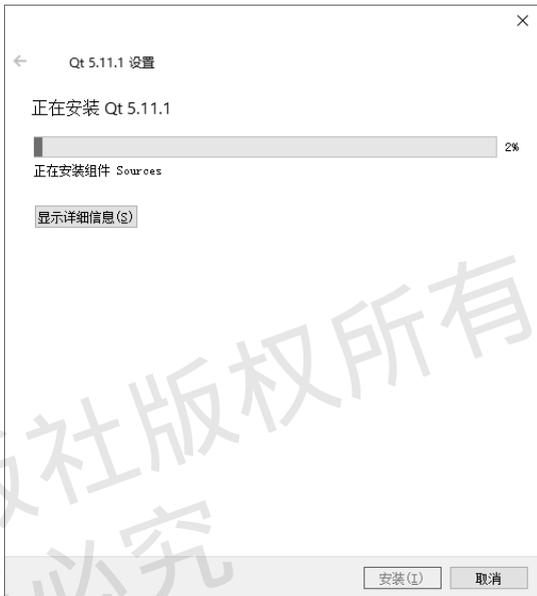


图 1.10 系统进行安装

安装完成后，勾选“Launch Qt Creator”复选框，如图 1.11 所示，单击“完成”按钮结束安装。系统会自行启动 Qt 5 Creator，进入 Qt Creator 初始界面。



图 1.11 安装完成

1.2.3 运行 Qt Creator

Qt Creator 运行后，进入 Qt Creator 初始界面，如图 1.12 所示。

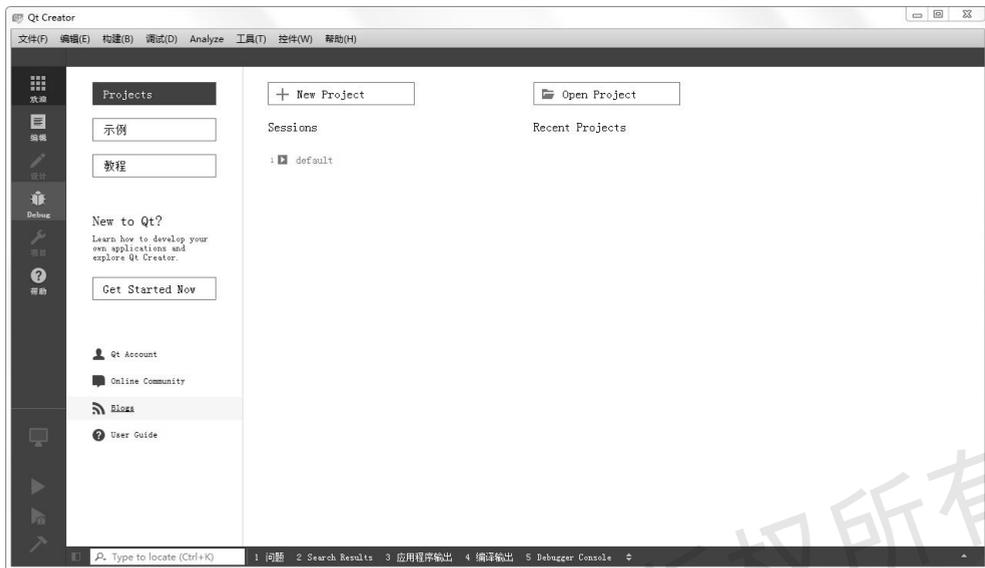


图 1.12 Qt Creator 初始界面

该界面左侧按钮的功能如下。

-  (欢迎)：可以选择自带的例子演示，在下次打开该界面时能够显示最近一次的一些项目，免除用户再去查找的麻烦。
-  (编辑)：可编写代码进行程序设计。
-  (设计)：可设计图形界面，进行部件属性设置、信号和槽设置及布局设置等操作。
-  (Debug)：可以根据需要调试程序，以便跟踪观察程序的运行情况。
-  (项目)：可以完成开发环境的相关配置。

-  (帮助): 可以输入关键字, 查找相关帮助信息。

左下角的三个按钮 、 和  分别是“运行”按钮、“开始调试”按钮和“构建项目”按钮。顾名思义, 这三个按钮相对应的功能分别为启动运行、开始调试和构建项目。

1.2.4 Qt 5 开发环境

在 Qt 程序开发过程中, 除可以通过手写代码实现软件开发功能外, 还可以通过 Qt 的 GUI 界面设计器 (Qt Designer) 进行界面的绘制和布局。该工具提供了 Qt 的基本可绘制窗口部件, 如 QWidget、QLabel、QPushButton 和 QVBoxLayout 等。在设计器中用鼠标直接拖曳这些窗口部件, 能够高效、快速地实现 GUI 界面的设计, 界面直观形象, 所见即所得。Qt 设计器主界面如图 1.13 所示。



图 1.13 Qt 设计器主界面

进入 Qt 设计器主界面后, 看到的设计区窗体部分 (如图 1.14 所示) 就是将要设计的顶层窗口部件 (顶层窗口部件是其他子窗口部件的载体)。

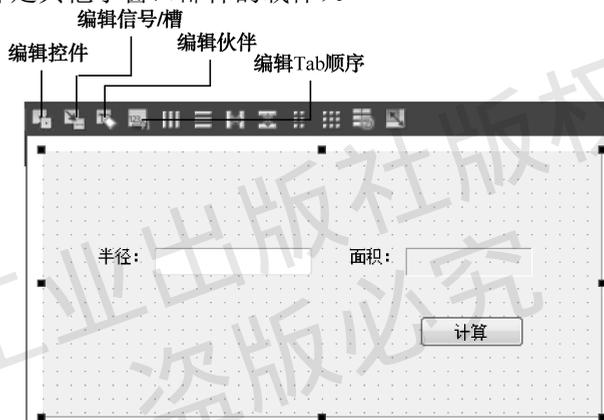


图 1.14 窗口部件编辑模式



在 Qt 设计器主界面的左侧“组件箱”栏中列出了经常使用的 Qt 标准窗口部件，可以直接拖曳相应的窗口部件图标到顶层窗口部件的界面上。同时，也可以将设计的窗口部件组合（通过布局管理器对 Qt 标准窗口部件进行布局和组合）或放置其他窗口部件的 Qt 容器类（见“组件箱”栏中的“Containers”组）直接拖曳到“组件箱”栏中，Qt 设计器会自动在“组件箱”栏中生成“Scratchpad”组，并生成新的自定义的窗口部件。此后，可以像使用 Qt 提供的标准窗口部件一样使用新创建的窗口部件。

选中 Qt 设计器主界面的“控件”→“视图”中的全部选项，在 Qt 设计器主界面上可以看到设计器提供的一些编辑工具子窗口（如图 1.13 所示）。

- **对象查看器 (Object Inspector):** 列出了主界面中所有窗口部件，以及各窗口部件的父子关系和包容关系。

- **属性编辑器 (Property Editor):** 列出了窗口部件可编辑的属性。

- **Action 编辑器 (Action Editor):** 列出了为窗口部件设计的 QAction 动作，通过“添加”或“删除”按钮可以新建一个可命名的 QAction 动作或删除指定的 QAction 动作。

- **信号和槽编辑器 (Signals & Slots Editor):** 列出了在 Qt 设计器中关联的信号和槽，通过双击列中的对象或信号/槽，可以进行对象的选择和信号/槽的选择。

此外，通过 Qt 设计器的“编辑”菜单，可以打开 Qt 设计器主界面的四种 GUI 窗口部件编辑模式。

- **控件编辑模式 (Edit Widgets):** 可以在 Qt 设计器主界面中添加 GUI 窗口部件并修改它们的属性和外观。

- **信号/槽编辑模式 (Edit Signals/Slots):** 可以在 Qt 设计器主界面中的窗口部件上关联 Qt 已经定义好的信号和槽。

- **伙伴编辑模式 (Edit Buddies):** 可以在 Qt 设计器主界面中的窗口部件上建立 QLabel 标签和其他窗口部件的伙伴关系。

- **Tab 顺序编辑模式 (Edit Tab Order):** 可以在 Qt 设计器主界面中的窗口部件上设置 Tab 键在窗口部件上的焦点顺序。



1.3 Qt 5 开发步骤及实例

大体了解开发 Qt 程序的基本流程有助于 Qt 开发快速入门。下面以完成计算圆面积功能这一简单例子来介绍 Qt 开发程序的流程，其中涉及 Qt 应用程序用户界面中的事件关联操作内容——信号和槽机制 (Signal & Slot)。

当用户输入一个圆的半径后，可以显示计算后的圆的面积值。运行效果如图 1.15 所示。

Qt 中开发应用程序既可以采用 Qt 设计器 (Qt Designer) 方式，也可以采用编写代码的方式。下面首先采用 Qt 设计器——Qt 5 Designer 进行 GUI 应用程序开发，使读者对 Qt 开发程序的流程有一个初步的认识。然后再采用编写代码的形式详细介绍 Qt 程序开发步骤。



图 1.15 计算圆面积实例

1.3.1 设计器 Qt Designer 实现

【例】（简单）（CH101）采用设计器 Qt Designer 实现计算圆面积，完成如图 1.15 所示的功能。

首先进行界面设计，然后编写相应的计算圆面积代码。

1. 界面设计

步骤如下。

（1）单击运行 Qt Creator，进入 Qt Creator 初始界面（如图 1.12 所示）。单击上端的 按钮，或者选择“文件”→“新建文件或项目...”命令，创建一个新项目，出现“新建项目”窗口，如图 1.16 所示。

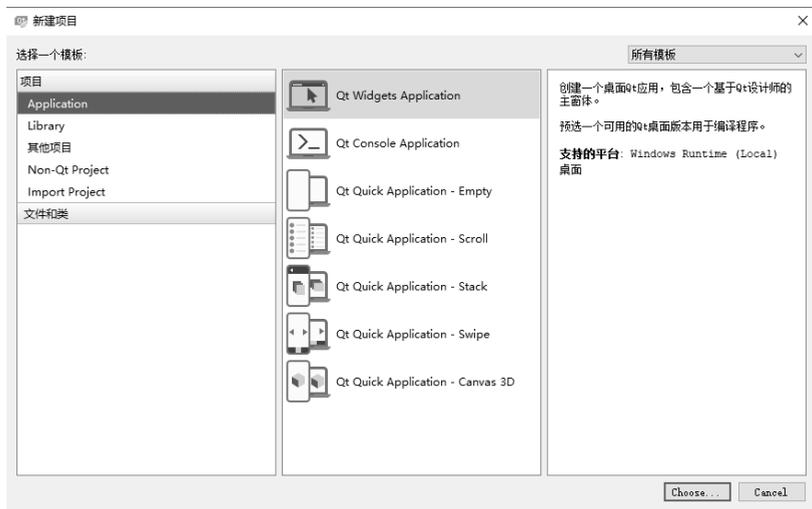


图 1.16 “新建项目”窗口

（2）单击选择项目“Application”→“Qt Widgets Application”选项，单击“Choose...”按钮。

编程者需要创建什么样的项目就选择相应的项目选项即可。例如，“Qt Console Application”选项是创建一个基于控制台的项目。这里因为需要建立一个桌面应用程序，所以选择“Qt Widgets Application”。

（3）选择保存项目的路径并定义自己项目的名字。注意，保存项目的路径中不能有中文。项目命名没有大小写要求，依据个人习惯命名即可。这里将项目命名为 Dialog，保存路径为 D:\Qt\CH1\CH101，如图 1.17 所示。单击“下一步”按钮。

（4）弹出“Kit Selection”（选择构建套件）界面，系统默认已指定 C++ 的编译器和调试器，如图 1.18 所示，直接单击“下一步”按钮。

（5）根据实际需要，选择一个“基类”。这里选择 QDialog 对话框类作为基类，这时“类名”“头文件”“源文件”“界面文件”都出现默认的文件名。注意，对这些文件名都可以根据具体需要进行相应的修改。默认选中“创建界面”复选框（如图 1.19 所示），表示需要采用界面设计器来设计界面，否则须要利用代码完成界面的设计。



图 1.17 保存项目



图 1.18 选择构建套件



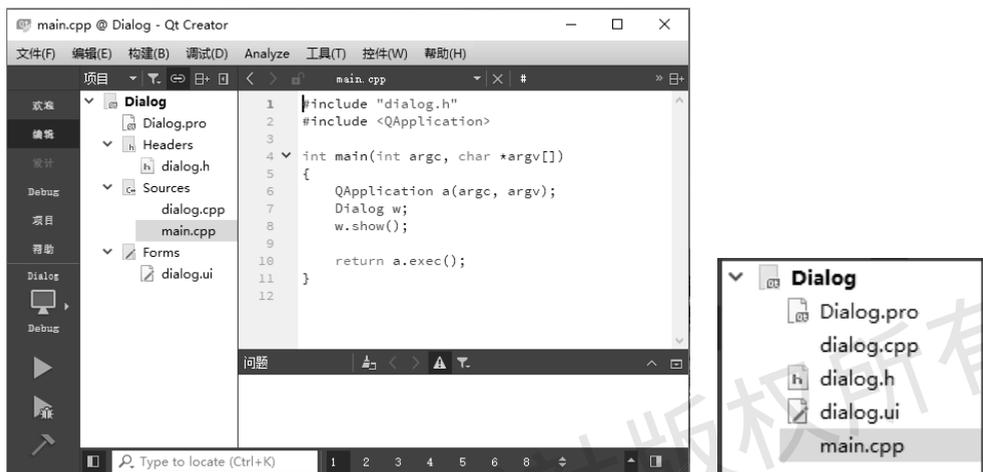
图 1.19 选择基类

(6) 选择完成后单击“下一步”按钮，相应的文件自动加载到项目文件列表中，如图 1.20 所示。



图 1.20 加载生成文件列表

单击“完成”按钮完成创建，文件列表中的文件自动分类显示，如图 1.21 (a) 所示，各个文件包含在相应的文件夹中，单击文件夹前的“▶”图标可以显示该文件夹下的文件；而单击文件夹前面的“◀”图标则可隐藏该文件夹下的文件。单击上部灰色工具栏中的过滤符号后，弹出一个下拉列表，勾选“简化树形视图”则切换到简单的文件列表，如图 1.21 (b) 所示。



(a) 文件列表中的文件自动分类显示

(b) 简单的文件列表

图 1.21 项目文件列表的显示样式

(7) 双击 dialog.ui，进入界面设计器 Qt Designer 编辑状态，开始进行设计器 (Qt Designer) 编程。

拖曳控件容器栏的滚动条，在最后的 Display Widgets 容器栏 (如图 1.22 所示) 中找到 Label 标签控件，拖曳三个这样的控件到中间的编辑框中；同样，在 Input Widgets 容器栏 (如图 1.21 所示) 中找到 Line Edit 文本控件，拖曳该控件到中间的编辑框中，用于输入半径值；在 Buttons



容器栏（如图 1.24 所示）中找到 Push Button 按钮控件，拖曳该控件到中间的编辑框中，用于提交响应单击事件。

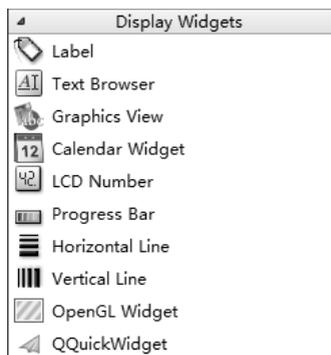


图 1.22 Display Widgets 容器栏

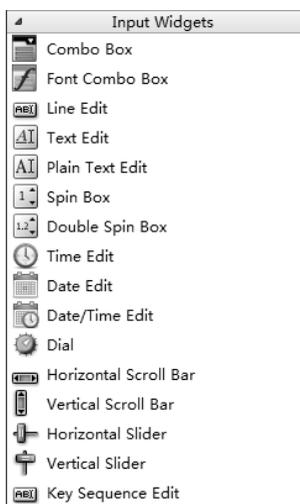


图 1.23 Input Widgets 容器栏



图 1.24 Buttons 容器栏

下面将修改拖曳到编辑框中的各控件的属性，调整后的布局如图 1.25 所示。对象监视器内容如图 1.26 所示。

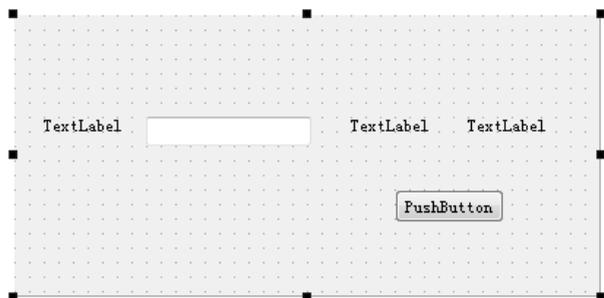


图 1.25 调整后的布局



图 1.26 对象监视器内容

然后，对各控件属性（如表 1.1 所示）进行修改。

表 1.1 各控件属性

Class	text	objectName
QLabel	半径:	radiusLabel
QLineEdit		radiusLineEdit
QLabel	面积:	areaLabel_1
QLabel		areaLabel_2
QPushButton	计算	countBtn

其中，修改控件 text 值的方法有如下两种。

- 直接双击控件本身即可修改。
- 在 Qt Designer 设计器的属性栏中修改，如修改半径标签 objectName 及 text 值，如图 1.27

所示。

radiusLabel : QLabel		radiusLabel : QLabel	
属性	值	属性	值
QObject		QLabel	
objectName	radiusLabel	text	半径:
QWidget		textFormat	AutoText

图 1.27 修改半径标签 objectName 及 text 值

将 areaLabel_2 的“frameShape”修改为 Panel;“frameShadow”为 Sunken, 如图 1.28 所示。最终效果如图 1.29 所示。

单击图 1.21 (a) 左下角的“运行”按钮 (▶) 或者使用 Ctrl+R 组合键运行程序, 最终效果如图 1.15 所示。

areaLabel_2 : QLabel	
属性	值
QFrame	
frameShape	Panel
frameShadow	Sunken

图 1.28 修改 areaLabel_2 属性



图 1.29 最终效果

至此, 程序的界面设计已经完成。

2. 编写相应的计算圆面积代码

首先简单认识一下 Qt 编程环境。找到文件列表中自动添加的 main.cpp 文件, 如图 1.21 所示。每个工程都有一个执行的入口函数, 此文件中的 main() 函数就是此工程的入口。

下面详细介绍 main() 函数的相关内容:

```
#include "dialog.h" // (a)
#include <QApplication> // (b)
int main(int argc, char *argv[]) // (c)
{
    QApplication a(argc, argv); // (d)
    Dialog w; // 创建一个对话框对象
    w.show(); // (e)
    return a.exec(); // (f)
}
```

其中,

(a) **#include "dialog.h"**: 包含了程序中要完成功能的 Dialog 类的定义, 在 Dialog 类中封装完成所需要的功能。注意, 使用哪个类就必须将包含该类的头文件引用过来。例如, 若用到一个按钮类, 则必须在此处添加一行代码 **#include <QPushButton>**, 这表明包含了按钮 (QPushButton) 类的定义。

(b) **#include <QApplication>**: Application 类的定义。在每个使用 Qt 图形化应用程序中都必须使用一个 QApplication 对象。QApplication 管理了各种各样的图形化应用程序的广泛资源、基本设置、控制流及事件处理等。



(c) `int main(int argc, char *argv[])`: 应用程序的入口, 几乎在所有使用 Qt 的情况下, `main()` 函数只需要在将控制转交给 Qt 库之前执行初始化, 然后 Qt 库通过事件向程序告知用户的行为。所有 Qt 程序中都必须有且只有一个 `main()` 函数。`main()` 函数有两个参数, 即 `argc` 和 `argv`。`argc` 是命令行变量的数量, `argv` 是命令行变量的数组。

(d) `QApplication a(argc, argv)`: `a` 是这个程序的 `QApplication` 对象。在任何 Qt 的窗口系统部件被使用之前必须创建 `QApplication` 对象。它在这里被创建并且处理这些命令行变量。所有被 Qt 识别的命令行参数都将从 `argv` 中被移去 (并且 `argc` 也因此而减少)。

(e) `w.show()`: 当创建一个窗口部件的时候, 默认它是不可见的, 必须调用 `show()` 函数使它变为可见。

(f) `return a.exec()`: 程序进入消息循环, 等待可能的输入进行响应。这里就是 `main()` 函数将控制权转交给 Qt, Qt 完成事件处理工作, 当应用程序退出的时候, `exec()` 函数的值就会返回。在 `exec()` 函数中, Qt 接收并处理用户和系统的事件并且将它们传递给适当的窗口部件。

现在, 有两种方式可以完成计算圆面积功能: 一是通过触发按钮事件完成 (方式 1); 二是通过触发输入编辑框事件完成 (方式 2)。

方式 1: 在 “Line Edit” 文本框内输入半径值, 然后单击 “计算” 按钮, 则在 `areaLabel_2` 中显示对应的圆面积。

编写代码步骤如下。

(1) 在 “计算” 按钮上按鼠标右键, 在弹出的下拉菜单中选择 “转到槽...” 命令, 在 “转到槽” 对话框中选择 `QAbstractButton` 的 “`clicked()`” 信号, 单击 “OK” 按钮, 如图 1.30 所示。

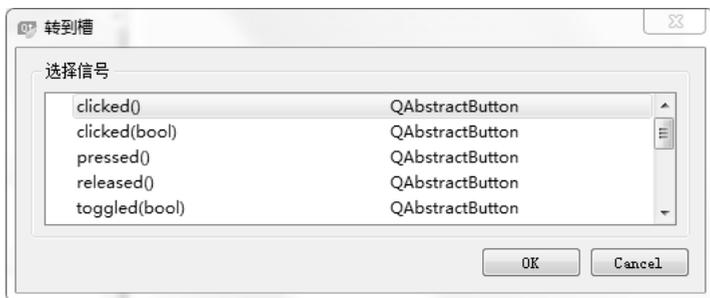


图 1.30 选择 `QAbstractButton` 的 “`clicked()`” 信号

(2) 进入 `dialog.cpp` 文件中按钮单击事件的槽函数 `on_countBtn_clicked()`。信号与槽连接的具体说明参照后面 L1.3 节的 L1 提供的知识点链接部分。在此函数中添加如下代码:

```
void Dialog::on_countBtn_clicked()
{
    bool ok;
    QString tempStr;
    QString valueStr=ui->radiusLineEdit->text();
    int valueInt=valueStr.toInt(&ok);
    double area=valueInt*valueInt*PI; //计算圆面积
    ui->areaLabel_2->setText(tempStr.setNum(area));
}
```

(3) 在 `dialog.cpp` 文件开始处添加以下语句:

```
const static double PI=3.1416;
```

定义全局变量 `PI`。

运行程序，在“Line Edit”文本框内输入半径值，单击“计算”按钮后，显示圆面积，完成计算圆面积功能。

方式 2: 在“Line Edit”文本框内输入半径值，不需要单击按钮触发单击事件，直接就在 areaLabel_2 中显示圆面积。

编写代码步骤如下。

(1) 在“Line Edit”文本框上按鼠标右键，在弹出的下拉菜单中选择“转到槽...”命令，在“转到槽”对话框中选择 QLineEdit 的“textChanged(QString)”信号，如图 1.31 所示。

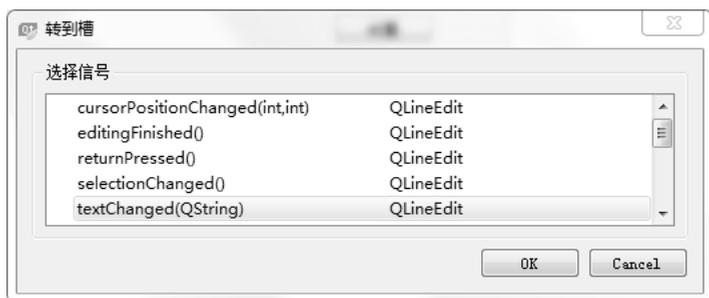


图 1.31 选择 QLineEdit 的“textChanged(QString)”信号

(2) 单击“OK”按钮，进入 dialog.cpp 文件中的文本编辑框改变值内容事件的槽函数 on_radiusLineEdit_textChanged(const QString &arg1)。在此函数中添加如下代码：

```
void Dialog::on_radiusLineEdit_textChanged(const QString &arg1)
{
    bool ok;
    QString tempStr;
    QString valueStr=ui->radiusLineEdit->text();
    int valueInt=valueStr.toInt(&ok);
    double area=valueInt*valueInt*PI;           //计算圆面积
    ui->areaLabel_2->setText(tempStr.setNum(area));
}
```

运行此程序，在“Line Edit”文本框中输入半径值后，直接在 areaLabel_2 中显示圆的面积值，完成计算圆面积的功能。

1.3.2 代码实现简单实例

【例】（简单）（CH102）采用编写代码的方式来实现计算圆面积的功能。

步骤如下。

(1) 首先创建一个新项目。创建过程和 1.3.1 节的界面设计中的第 (1) ~ (7) 步相同，只是在第 (3) 步中，项目命名为 Dialog 且保存路径为 D:\Qt\CH1\CH102，在第 (5) 步中，取消“创建界面”复选框的选中状态。

(2) 在上述项目的 dialog.h 中添加如下加黑代码：

```
#include <QLabel>           //(a)
#include <QLineEdit>        //(a)
#include <QPushButton>      //(a)
```



```
class Dialog : public QDialog
{
    Q_OBJECT
public:
    Dialog(QWidget *parent = 0);
    ~Dialog();
private:
    QLabel *label1,*label2;           //(b)
    QLineEdit *lineEdit;             //(b)
    QPushButton *button;             //(b)
};
```

其中，

(a) 加入实现 Label、LineEdit、PushButton 控件的头文件。

(b) 定义界面中的 Label、LineEdit、PushButton 控件对象。label1 标签对象提示“请输入圆的半径”，label2 标签对象显示圆面积计算结果，LineEdit 文本框对象用于输入半径，PushButton 为“计算”命令按钮对象。

Q_OBJECT 宏的作用是启动 **Qt 5 元对象系统** 的一些特性（如支持信号和槽等），它必须放置到类定义的私有区中。



注意：

在文件中用到哪个类时，需要该文件开始部分引用包含此类的头文件。

(3) 在 dialog.cpp 中添加如下代码：

```
#include <QGridLayout> // (a)
Dialog::Dialog(QWidget *parent)
    : QDialog(parent)
{
    label1=new QLabel(this);
    label1->setText(tr("请输入圆的半径: "));
    lineEdit=new QLineEdit(this);
    label2=new QLabel(this);
    button=new QPushButton(this);
    button->setText(tr("显示对应圆的面积"));
    QGridLayout *mainLayout=new QGridLayout(this); // (b)
    mainLayout->addWidget(label1,0,0);
    mainLayout->addWidget(lineEdit,0,1);
    mainLayout->addWidget(label2,1,0);
    mainLayout->addWidget(button,1,1);
}
```

其中，

(a) `#include <QGridLayout>` 为加入实现 **布局管理器** 的头文件。

(b) `QGridLayout *mainLayout=new QGridLayout(this)` 用于 **布局管理器**，将所有控件的位置固定。

界面运行效果如图 1.32 所示。

(4) 完成程序功能:

以上第 (1) ~ (3) 步代码只完成了界面设计, 下面同样通过两种触发不同控件事件的方式来完成计算圆面积的功能。

方式 1: 在 `lineEdit` 文本框内输入所需圆的半径值, 单击“显示对应圆的面积”按钮后, 在 `label2` 中显示相对应的圆的面积值。

① 打开 `dialog.h` 文件, 在类构造函数和控件成员声明后, 添加如下加黑代码:

```
class Dialog : public QDialog
{
    ...
    QPushButton *button;
private slots:
    void showArea();
};
```

② 打开 `dialog.cpp` 文件, 在构造函数中添加如下加黑代码:

```
Dialog::Dialog(QWidget *parent)
    : QDialog(parent)
{
    ...
    mainLayout->addWidget(button,1,1);
connect(button,SIGNAL(clicked()),this,SLOT(showArea()));
}
```

③ 在 `showArea()` 中实现显示圆面积功能, 代码如下:

```
const static double PI=3.1416;
void Dialog::showArea()
{
    bool ok;
    QString tempStr;
    QString valueStr=lineEdit->text();
    int valueInt=valueStr.toInt(&ok);
    double area=valueInt*valueInt*PI;
    label2->setText(tempStr.setNum(area));
}
```

④ 运行程序。

在 `lineEdit` 文本框中输入圆半径值, 单击“显示对应圆的面积”按钮后, 在 `label2` 中显示圆面积值, 最终运行结果如图 1.33 所示。

方式 2: 在 `lineEdit` 文本框中输入所需圆的半径值后, 不必单击“显示对应圆的面积”按钮, 直接在 `label2` 中显示圆的面积值。操作步骤和方式 1 相同, 只是在上述第②步中, 添加的代码修改为如下加黑代码:

```
Dialog::Dialog(QWidget *parent)
    : QDialog(parent)
```



图 1.32 界面运行效果



图 1.33 最终运行结果



```
{  
    ... ..  
    mainLayout->addWidget(button,1,1);  
    connect(lineEdit,SIGNAL(textChanged(QString)),this,SLOT(showArea()));  
}
```

重新运行程序，在 `lineEdit` 文本框中输入圆半径值后，不必单击“显示对应圆的面积”按钮，直接在 `label2` 中显示对应的圆面积值。

基本的 Qt 开发步骤已经介绍完毕，在本书以后章节的例子中就不再如此详细介绍了。

1.3.3 Qt 低版本实例迁移

《Qt 5 开发及实例》（第 3 版）以 Qt 5.8 进行介绍，所有实例在该版本下开发。本书以 Qt 5.11 作为平台开发，其基本内容与 Qt 5.8 是兼容的。如果在 Qt 5.11 平台下打开 Qt 5.8 的实例工程，系统显示如图 1.34 所示。

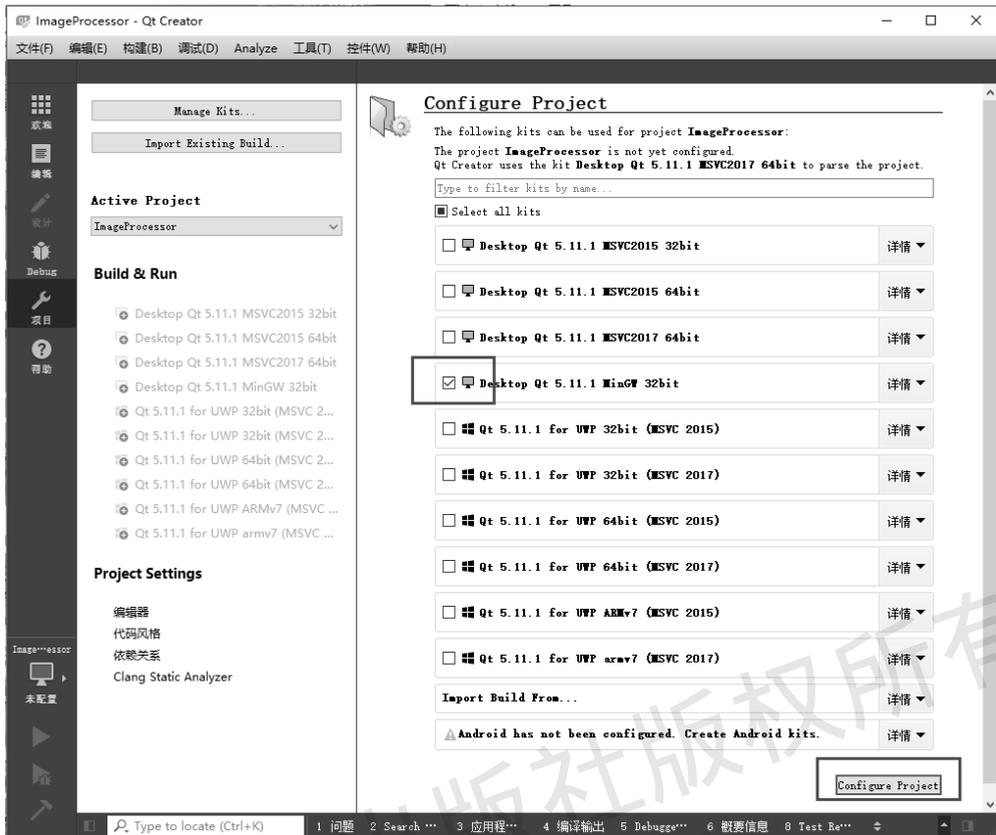


图 1.34 打开 Qt 5.8 的实例工程

勾选“Desktop Qt 5.11.1 MinGW 32bit”复选框，Qt 5.8 实例工程即可在 Qt 5.11 环境下打开，其后保存就转换为 Qt 5.11 版本的实例。



L1.2 Qt 5 的安装：概念解析

伙伴编辑模式 (Edit Buddies)

QLabel 标签和伙伴 (buddy) 窗口部件包括一个标签 (QLabel) 和一个窗口部件，它们具有伙伴关系，指当用户激活标签的快捷键时，鼠标/键盘的焦点将会转移到它的伙伴窗口部件上。只有 QLabel 标签对象才可以有伙伴窗口部件，也只有在该 QLabel 对象具有快捷键（在显示文本的某个字符前面添加一个前缀“&”，就可以定义快捷键）时，伙伴关系才有效。例如：

```
QLineEdit* ageLineEdit = new QLineEdit(this);
QLabel* ageLabel = new QLabel("&Age",this);
ageLabel->setBuddy(ageLineEdit);
```

代码定义了 ageLabel 标签的组合键为 Alt+A，并将行编辑框 ageLineEdit 设为它的伙伴窗口部件。当用户按下 Alt+A 组合键时，焦点将会跳至行编辑框 ageLineEdit 中。

Qt 设计器提供了伙伴编辑模式，可以通过鼠标拖曳操作快捷地建立标签 QLabel 和其他窗口部件的伙伴关系。



L1.3 Qt 5 开发步骤及实例：概念解析

L1 信号和槽机制 (Signal & Slot)

Qt 提供了信号和槽机制用于完成界面操作的响应，信号和槽机制是完成任意两个 Qt 对象之间的通信机制。其中，信号会在某个特定情况或动作下被触发，槽是等同于接收并处理信号的函数。例如，若要将一个窗口部件的变化情况通知给另一个窗口部件，则一个窗口部件发送信号，另一个窗口部件的槽接收此信号并进行相应的操作，即可实现两个窗口部件之间的通信。每个 Qt 对象都包含若干个预定义的信号和若干个预定义的槽。当某一个特定事件发生时，一个信号被发送，与信号相关联的槽则会响应信号并完成相应的处理。当一个类被继承时，该类的信号和槽也同时被继承，也可以根据需要自定义信号和槽。

1. 信号与槽机制的连接方式

(1) 一个信号可以与另一个信号相连，代码如下：

```
connect(Object1,SIGNAL(signal1),Object2,SIGNAL(signal1));
```

表示 Object1 的信号 1 发送可以触发 Object2 的信号 1 发送。

(2) 同一个信号可以与多个槽相连，代码如下：

```
connect(Object1,SIGNAL(signal2),Object2,SIGNAL(slot2));
connect(Object1,SIGNAL(signal2),Object3,SIGNAL(slot1));
```

(3) 同一个槽可以响应多个信号，代码如下：



```
connect(Object1,SIGNAL(signal2),Object2,SIGNAL(slot2));  
connect(Object3,SIGNAL(signal2),Object2,SIGNAL(slot2));
```

但是，常用的连接方式为：

```
connect(Object1,SIGNAL(signal),Object2,SLOT(slot));
```

其中，`signal` 为对象 `Object1` 的信号，`slot` 为对象 `Object2` 的槽。

在本书 1.3.1 节（通过设计器实现）实例中，在 Qt 应用程序的用户界面加入计算圆面积的“计算”按钮后，应用程序并没有响应计算操作。这是因为程序还没有将相应的信号和槽关联起来。因此，为了响应用户的计算面积值的操作，需要将“计算”按钮发送的单击信号 `QAbstractButton::clicked()` 和对话框 `QDialog` 的 `Dialog::on_countBtn_clicked()` 槽关联起来，可以根据需要在槽函数中进行相应的操作。类似地，改变文本编辑框内容信号 `QLineEdit::textChanged(QString)` 产生后与对话框 `QDialog` 的 `Dialog::on_radiusLineEdit_textChanged(const QString &arg1)` 槽关联起来。

在本书 1.3.2 节（通过编写代码实现）实例中，将“显示对应圆的面积”按钮发送的单击信号 `QAbstractButton::clicked()` 和对话框 `QDialog` 的 `Dialog::showArea()` 槽关联起来。类似地，改变文本编辑框内容信号 `QLineEdit::textChanged(QString)` 产生后也与对话框 `QDialog` 的 `Dialog::showArea()` 槽关联起来。

`SIGNAL()` 和 `SLOT()` 是 Qt 定义的两个宏，它们返回其参数的 C 语言风格的字符串（`const char*`）。因此，下面关联信号和槽的两个语句是等同的：

```
connect(button,SIGNAL(clicked()),this,SLOT(showArea()));  
connect(button, "clicked()",this, "showArea()");
```

2. 信号与槽机制的优点

(1) **类型安全**。需要关联的信号和槽的签名必须是等同的，即信号的类型和参数个数与接收该信号的槽的参数类型和参数个数相同。不过，一个槽的参数个数是可以少于信号的类型和参数个数的，但缺少的参数必须是信号参数的最后一个或几个参数。如果信号和槽的签名不符，编译器就会报错。

(2) **松散耦合**。信号和槽机制减弱了 Qt 对象的耦合度。激发信号的 Qt 对象无须知道是哪个对象的哪个槽需要接收它发出的信号，它只需做的是在适当的时间发送适当的信号就可以了，而不需要知道也不关心它的信号有没有被接收到，更不需要知道是哪个对象的哪个槽接收到了信号。同样，对象的槽也不知道是哪些信号关联了自己，而一旦关联信号和槽，Qt 就保证了适合的槽得到了调用。即使关联的对象在运行时被删除，应用程序也不会崩溃。

一个类若要支持信号和槽，就必须从 `QObject` 或 `QObject` 的子类继承。注意，Qt 信号和槽机制不支持对模板的使用。

3. 信号与槽机制的效率

信号和槽机制增强了对对象间通信的灵活性，然而，这也损失了一些性能。同回调函数相比，信号和槽机制运行速度有些慢。通常，通过传递一个信号来调用槽函数会比直接调用非虚函数的运行速度慢 10 倍。主要原因如下。

- (1) 需要定位接收信号的对象。
- (2) 安全地遍历所有的关联（如一个信号关联多个槽的情况）。
- (3) 编组（`marshal`）/解组（`unmarshal`）传递的参数。
- (4) 在多线程时，信号可能需要排队等待。

然而，与创建堆对象的 `new` 操作及删除堆对象的 `delete` 操作相比，信号和槽的运行代价很小。信号和槽机制导致的这点性能损失对实时应用程序是可以忽略的；同信号和槽提供的灵活性和简便性相比，这点性能损失也是值得的。

L2 Qt 5 元对象系统

Qt 5 元对象系统提供了对象间的通信机制（信号和槽）、运行时类型信息和动态属性系统的支持，是标准 C++ 的一个扩展，它使 Qt 能够更好地实现 GUI 图形用户界面编程。Qt 5 的元对象系统不支持 C++ 模板，尽管该模板扩展了标准 C++ 的功能。但是，元对象系统提供了模板无法提供的一些特性。Qt 5 元对象系统基于以下三个事实。

- (1) 基类 `QObject`：任何需要使用元对象系统功能的类必须继承自 `QObject`。
- (2) `Q_OBJECT` 宏：`Q_OBJECT` 宏必须出现在类的私有声明区中，用于启动元对象的特性。
- (3) 元对象编译器（Meta-Object Compiler, moc）：为 `QObject` 子类实现元对象特性提供必要的代码实现。

L3 布局管理器

在设计较复杂的 GUI 用户界面时，仅通过指定窗口部件的父子关系以期达到加载和排列窗口部件的方法是行不通的，最好的办法是使用 Qt 提供的布局管理器。

```
QGridLayout *mainLayout=new QGridLayout(this);           //(a)
mainLayout->addWidget(label1,0,0);                       //(b)
mainLayout->addWidget(lineEdit,0,1);
mainLayout->addWidget(label2,1,0);
mainLayout->addWidget(button,1,1);
setLayout(mainLayout);                                   //(c)
```

其中，

(a) `QGridLayout *mainLayout=new QGridLayout(this)`：创建一个网格布局管理器对象 `mainLayout`，并用 `this` 指出父窗口。

(b) `mainLayout->addWidget(...)`：分别将控件对象 `label1`、`lineEdit`、`label2` 和 `button` 放置在布局管理器中，还可以在创建布局管理器对象时不必指明父窗口。

(c) `QWidget::setLayout(...)`：将布局管理器添加到对应的窗口部件对象中。因为这里的主窗口就是父窗口，所以直接调用 `setLayout(mainLayout)` 即可。

布局管理器的具体使用方法请参照本书第 3 章有关布局管理器的内容。

电子工业出版社版权所有
盗版必究