工程师经验手记

Linux QtGUI开发详解

——基于Nokia Qt SDK

李彬 编著

北京航空航天大学出版社
BEIHANG UNIVERSITY PRESS

工程师经验手记

Linux Qt GUI 开发详解

——基于 Nokia Qt SDK

李 彬 编著

北京航空航天大学出版社

内 容 简 介

全书详细介绍了 Linux 下 Qt 用户界面开发的重要的核心知识。全书共分为 5 章 20 节，涉及 Qt 基础控件的使用，开发工具的使用，信号与槽机制的探秘，GUI 换肤和多国语言支持的实现，Qt 事件驱动机制，多进程和多线程编程技术，Qt 串口编程技术，Qt WebKit 模块的高级编程技术，浏览器 JavaScript 对象扩展技术，QWebPluginFactory 的使用，基于 NPAPI 标准的跨浏览器插件开发技术，MySQL 和 SQLite 数据库在 Qt 中的应用及 XML 解析技术，QNetworkAccessManager 及其相关类的使用等。本书在编写相关知识点时尽量通过例子来演示知识点的应用，尽量用通俗易懂的话来阐述知识点，每一章都会通过项目实例来强化读者对该章知识点的掌握和提高读者的实战水平及经验。

本书适合于希望尽快入门 Qt 并尽快融入开发的初学者；也适合于希望积累 Qt 项目实践经验的一线开发工程师；还可以作为嵌入式培训机构及各大中专院校嵌入式相关专业的参考用书。

图书在版编目(CIP)数据

Linux Qt GUI 开发详解：基于 Nokia Qt SDK / 李彬编著. -- 北京 ：北京航空航天大学出版社，2013.1

ISBN 978-7-5124-1034-3

Ⅰ. ①L… Ⅱ. ①李… Ⅲ. ①软件工具－程序设计 Ⅳ. ①TP311.56

中国版本图书馆 CIP 数据核字(2012)第 295368 号

版权所有，侵权必究。

Linux Qt GUI 开发详解
——基于 Nokia Qt SDK
李 彬 编著
责任编辑 苗长江 王 彤
*
北京航空航天大学出版社出版发行
北京市海淀区学院路 37 号(邮编 100191) http://www.buaapress.com.cn
发行部电话：(010)82317024 传真：(010)82328026
读者信箱：emsbook@gmail.com 邮购电话：(010)82316936
涿州市新华印刷有限公司印装 各地书店经销
*
开本：710×1 000 1/16 印张：20.5 字数：437 千字
2013 年 1 月第 1 版 2013 年 1 月第 1 次印刷 印数：4 000 册
ISBN 978-7-5124-1034-3 定价：45.00 元

若本书有倒页、脱页、缺页等印装质量问题，请与本社发行部联系调换。联系电话：(010)82317024

目 录

试读结束：需要全本请在线购买： www.ertongbook.com

第1章

Qt基础控件使用

1.1 Qt SDK环境搭建

1.1.1 g++编译器安装

当我们决定开始安装环境的时候，需要在Ubuntu系统里面安装好g++编译器，在Ubuntu11.10操作系统里面单击桌面左边的“面板主页”，在搜索栏里面输入“ter”就会找到我们要用的“终端”。打开终端之后输入“~ # apt-get install g++”回车，接着输入“y”，即可轻松地安装。计算机处于联网状态，才可以安装，这是通过包管理工具“apt-get”从网络服务器上下载、安装、卸载或者升级软件的。这些软件包的镜像站点地址我们可以在“/etc/apt/sources.list”里面找到。

apt-get小解

执行指令格式	apt-get <参数>【软件包名】，需要root权限执行。
安装软件包	apt-get install 软件包名
卸载软件包	apt-get remove 软件包名
更新软件包	apt-get upgrade
升级操作系统	apt-get dist-upgrade
软件包镜像更新	apt-get update

有了这些法宝，在Linux下安装软件也像Windows下一样方便了，要远比源码包的安装方便。

1.1.2 Qt SDK 安装

对于 Qt 的一些早期版本，在 Linux 下搭建 Qt 开发环境，需要进行源码包的安装，编译过程漫长，操作步骤繁琐。例如 qt-x11-2.3.2 版本，安装之后，当进行开发的时候，需要生成工程还需要借助 progen 工具，生成 Makefile 的时候还需要借助 tmake 工具，在需要进行工程代码管理的时候还需要借助 Kdevelop 环境，给开发人员制造毫无意义的麻烦。搭建好完整环境可能需要花费开发人员 1 周左右的时间，而开发人员使用 Qt 的积极性在经历了 1 周的折腾后所剩无几了。更让开发人员感到不方便的是，这些版本的 Qt 使用起来很不顺手，界面看起来很不美观。好在 Nokia 发布了 Qt 软件开发工具包，让一切显得那么的便捷和具有艺术性。

当我们下载好 Qt_SDK_Lin32_offline_v1_1_3_en.run 二进制安装工具包之后，打开终端输入“～$sudo - s”回车，输入密码，进入 root 权限 shell，找到 Qt_SDK_Lin32_offline_v1_1_3_en.run 存储的路径，笔者存储在“/home/libin/qtsetup”目录下面了，在终端执行“～#chmod u+x Qt_SDK_Lin32_offline_v1_1_3_en.run”。如果对 chmod 指令有所了解可以跳过下文这个 chmod 小解。

chmod 指令小解

chmod 可以改变 Linux 下文件的拥有者、同组用户和其他用户对该文件的操作权限。当然这些操作权限完全使用它们的英文首字母来表述了。比如说一个文件只有“可被读取”的权限，那么就用“r”来描述这个特性；如果该文件只有“可被写入”的权限，就用“w”来描述；用“x”可以使一个文件在 Linux 操作系统下面具有“可被执行”权限。当这个文件具有比一个权限多的时候，可以用“chmod+(r/w/x)”来增加文件的权限。用“chmod -(r/w/x)”来取消相应的权限。在多用户的操作系统上，比如 Linux，还可以限制一个文件对不同的登录用户具备不同的操作权限，如果我们想让一个文件在同组(group)用户都可以具备写入权限，就可以执行“chmod g+w filename”来实现这个功能。

回到安装过程中，把安装包的权限改成拥有者具备执行权限之后，就可以用

root@libin:～/qtsetup# ./Qt_SDK_Lin32_offline_v1_1_3_en.run 执行安装包，界面如图 1.1 所示。

单击 next 按钮进入图 1.2 所示安装界面。

可以把 SDK 安装在一个自己选择的位置，也可以按默认的路径安装。安装类型我们选择“Custom”。下一步弹出如图 1.3 所示的定制化安装界面。

点开“Documentation”把“Qt Designer Documentation”和“Qt Linguist Documentation”选中。点开“Development Tools”勾选“Qt Designer”，“Qt Assistant”和“Qt Linguist”这些工具在后面用到时再讲解它们的作用。下一步选择安装协议，如图 1.4 所示。

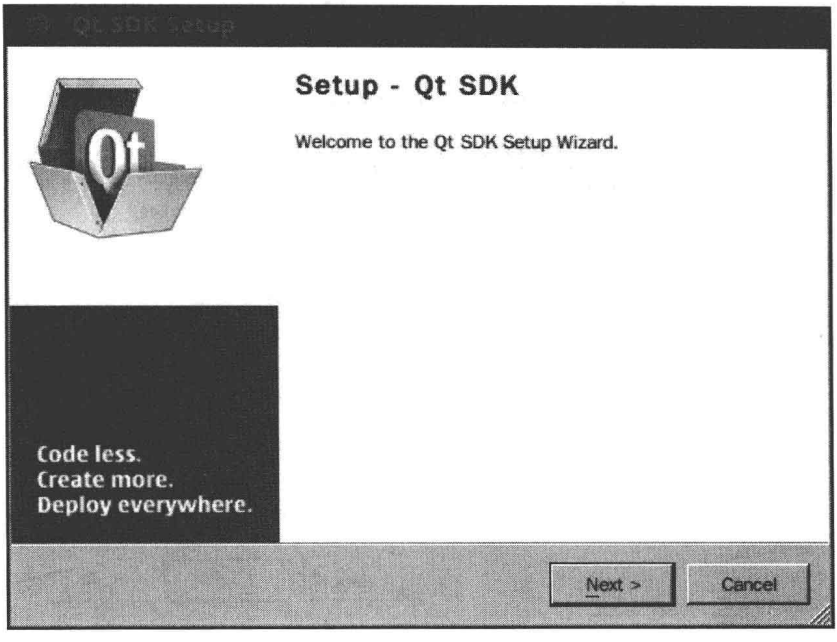

图 1.1　Qt SDK 安装包运行界面

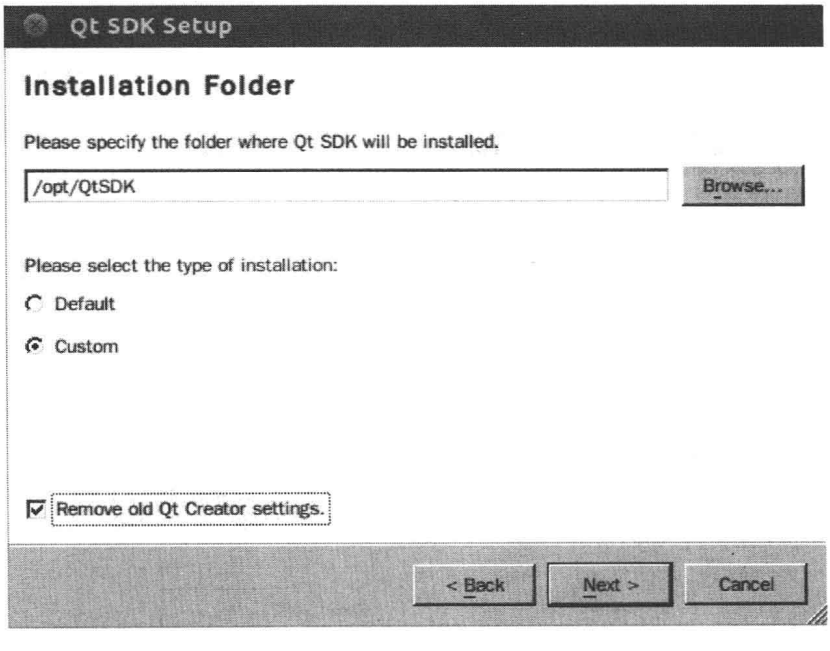

图 1.2　选择安装路径及类型

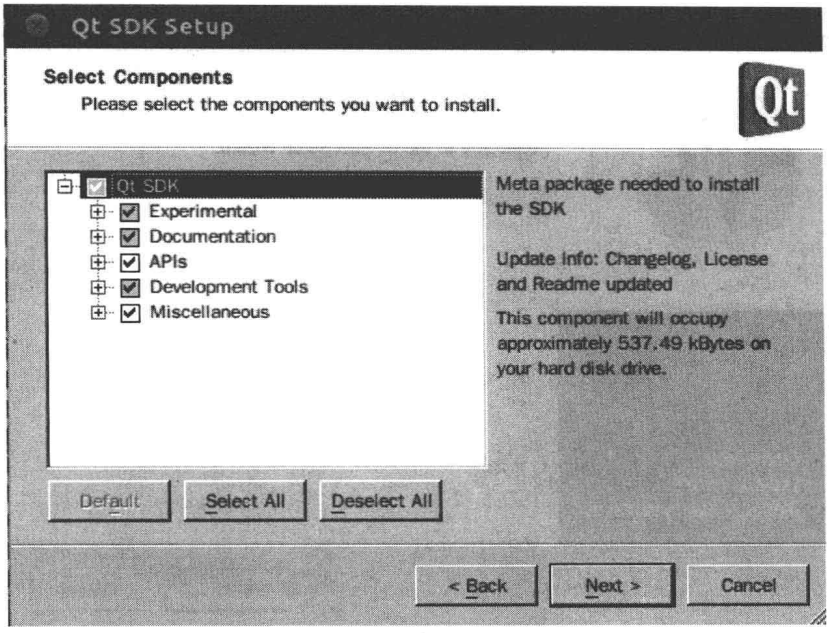

图 1.3　定制化安装组件选择

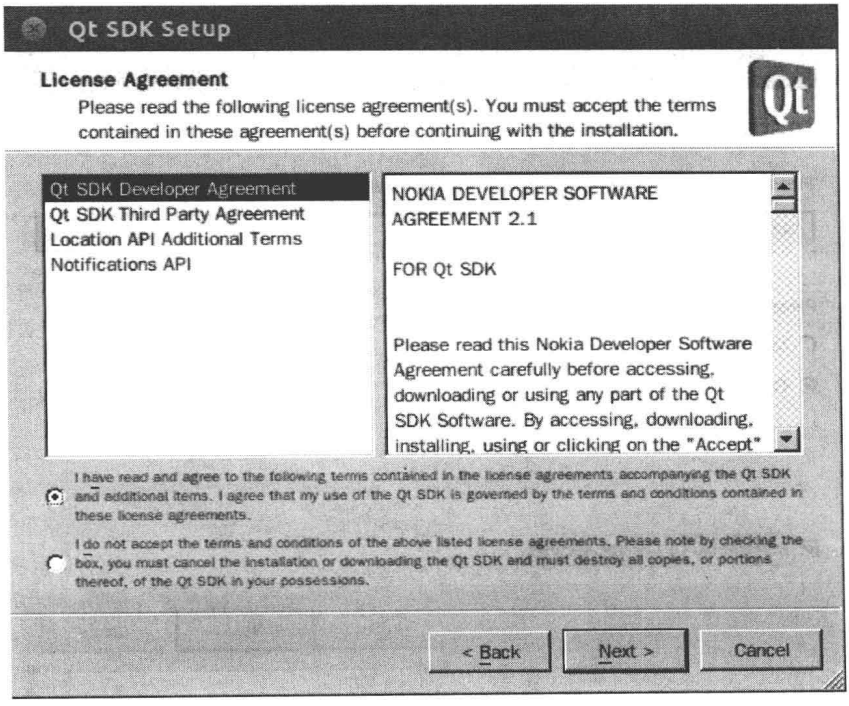

图 1.4　选择安装许可协议

在这里没有别的选择，只有把单选框“I have read and agree to…”选中。“GNU LESSER GENERAL PUBLIC LICENSE(LGPL)”是 GNU 较宽松公共许可证，为了得到更多的商用软件提供商的支持，GNU 推出了 GPL 的变种 LGPL。用户利用 LGPL 授权下的自由软件，开发出来的新软件可以是私有的，并不像 GPL 一样必须是自由软件，这样在开源软件的推广和保证商家的利益之间找到了一个平衡点。当安装的时候选择了对该协议的支持，那就意味着利用 Qt 开发出自己的软件时可以不公开源代码。从要求保密和保护商家核心代码的角度来看，的确是个不错的保护软件知识产权的方法。安装过程沿袭了 Windows 一路 next 的风格，最后单击“Finish”，即大功告成，如图 1.5 所示。

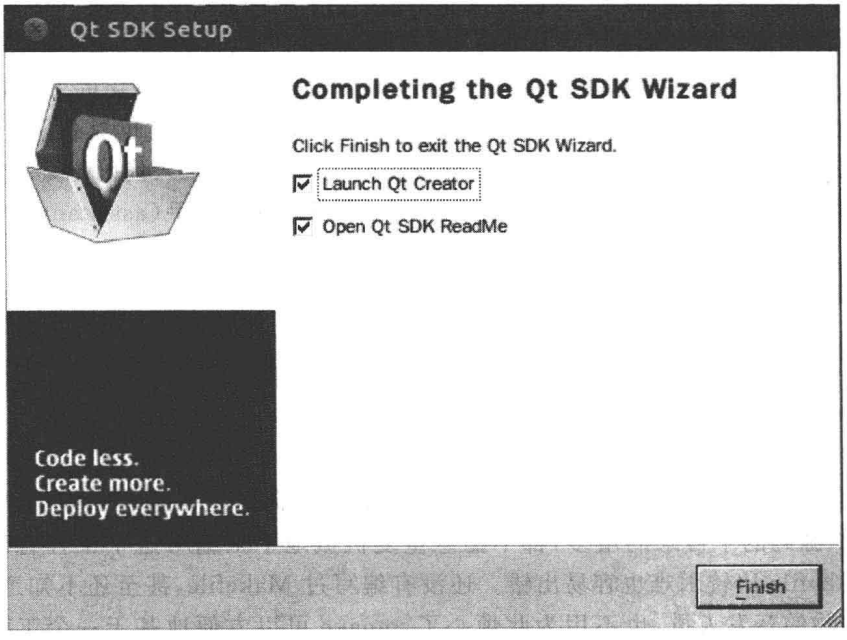

图 1.5　完成安装界面

1.2　Qt SDK 环境初体验

1.2.1　SDK 目录结构解析

去默认安装的路径“/opt/QtSDK”下面看看安装的结果吧。

```
root@libin:~# cd /opt/QtSDK
root@libin:/opt/QtSDK# ls
Changelog.txt    Documentation          Madde          SDKMaintenanceTool        Symbian
components.xml   Examples               pythongdb      SDKMaintenanceTool.dat
```

```
Demos              InstallationLog.txt  QtCreator  SDKMaintenanceTool.ini
Desktop            Licenses             readme     Simulator
root@libin:/opt/QtSDK#
```

Linux 安装工具即是复制文件到相应目录的过程，省去 Windows 注册表的那一套，着实显得很绿色很环保，通过安装后的文件结构就可以看出来。Qt 类库、Qt 工具和 QtCreator 组成了这个强大的 SDK，接下来让我们一一揭开它们的神秘面纱。Cd 到“QtCreator/bin”目录，可以看到 qtcreator 赫然在那里躺着呢。qtcreator 是会一直陪伴我们到本书最后的工具，很有必要认识一下它的历史。新官上任三把火，在 Nokia 收购 Qt 之后便迅速地推出了这款 Qt/C++集成开发环境(IDE)，我们安装的是在 Linux 下运行的版本，它还可以完美地运行在 Windows 和 Mac OS X 上。这个跨平台的 IDE 可以帮助开发人员快速而简易地建立项目、管理工程、编辑代码、浏览文件、添加文件。它还集成了最该具备的代码调试功能(图形化的 GDB 调试前端)，界面设计工具(Qt Designer)和构建工具(qmake)。当然它也具备了一般 IDE 都应有的代码补齐和参数提示功能，从而帮助开发人员快速录入代码。

功能强大的类库往往都很庞大。qtcreator 集成的 Qt 助手(assistant)工具可以方便地让开发人员在开发中快速而准确地查阅需要的技术文档，有了这个 IDE，新手得以快速上手，老手得以提高效率，实属老少咸宜，界面开发之必备工具。读者可以在“ftp://ftp.qt.nokia.com/qtcreator/”地址下载到 qtcreator 在 Mac/Linux/Windows 3 个平台上的历史和当前的安装包。退出 bin 目录，cd 到“Desktop/Qt/474/gcc/bin”目录，我们来认识几个经常用到的新朋友。

(1) No.1 qmake。

相信编写过 Makefile 的开发人员，随着工程中源码的级数递增和以类型、功能、模块组织源码的子目录的增多，都不愿意重复机械地手工编写这个工程管理文件。手写 Makefile 比较困难也容易出错。还没有编写过 Makefile，甚至还不知道 Makefile 为何物的开发人员，也不用为此烦心了，qmake 可以方便地基于一个工程文件，生成不同平台下的 Makefile。qmake 关注编译器和处理器平台的依赖性，开发人员不用再手写针对不同编译器和不同处理器平台的 Makefile，而可以花更多的精力在程序的设计上。随后我们会对它进行使用，那时你会觉得原来有它一切显得是那么的简单而又不失美好。

(2) No.2 designer。

Qt 的界面设计师，它是一个所见即所得(what-you-see-is-what-you-get)的界面绘制工具。通过这个工具我们可以在后面的操作中方便地通过鼠标的拖曳来布局和设计软件界面。

(3) No.3 assistant。

它是一个提供了 400 多种图形化用户界面的宏大的 C++类库。如果没有良好的帮助文件和在线文档阅读器对于开发人员来说是多么糟糕的一件事情呀，然而，

试读结束：需要全本请在线购买： www.ertongbook.com

Qt assistant 工具的出现，避免了这种糟糕事情发生在 Qt 开发人员身上。它做的还远远不只这些，当使用 assistan 的时候，会发现它考虑的非常周到，可以通过类似 web 浏览器导航、书签和文档文件链接，还提供关键字查询服务；当开发人员需要向最终用户提供文档支持的时候，它又是完全可定制的。

(4)No. 4 uic。

用户接口编译器，在 designer 里面绘制的软件界面可以靠这个工具生成对应的实现源码文件。它一般不需要手动执行，而是在 Makefile 中制定调用规则。

(5) No. 5 Moc(meta - object compiler)。

元对象编译器。看到编译器很容易想到 gcc、g++这些 Linux 下常用的编译器，既然 Qt 包含了 C++类库，用 g++编译 Qt 程序就成了顺理成章的事情了。然而，Qt 对标准 C++提供了扩展，这些扩展的内容包含了后面要讲到的对象间的通信机制(信号与槽)，这些是 Qt 特有的，用 g++是无法进行语法解释的。这就需要我们的 moc 站出来做个中间人，把 Qt 的这些特殊代码翻译成 C++标准语法代码，然后把翻译后的代码交给 g++进行编译连接，生成最终目标。如果你之前一直在 Windows 下面进行开发，那你已经非常熟悉和习惯图形化操作的编译器了，对你来说 gcc/g++可能的让你觉得陌生，不过这两个家伙在 Linux 的世界里面可是无人不知无人不晓。这两个命令行操作的编译器凭借它们出色的表现，已经成为 Linux 系统编程里面编译 C 代码和 C++代码不二之选的编译器。现在认识到 gcc 是常用来编译. c 后缀的 C 代码，g++常用来编译. cpp 后缀的 C++代码即可。

最后让我们来认识一下 lib 目录。Cd 到“QtCreator/lib/qtcreator”下面之后，用 ls 看到该目录的内容(内容过多就不列出来了)，呈现在我们面前的就是很多库文件，编写的界面程序要用到的动态链接库，不是“. dll”格式，而是“. so”。Linux 下动态库的命名格式是“lib * . so. * ”，第一个“ * ”是动态链接库的库名，最后一个“ * ”代表了库的版本号。Linux 表示静态链接库的后缀是“. a”也非“. lib”，如“lib * . a”，这里的“ * ”当然就是指静态库的称谓了。正如我们开发人员知道的，采用动态链接库是软件设计经常采用的技术，这跟它可以代码复用，减小执行文件占用空间，使用灵活和利于软件升级有直接的关系。通过 Qt 的动态链接库的名字，我们可以得到 Qt 在 Linux 下支持的模块(组件)，如表 1.1 所列。

表 1.1 Qt 模块说明

模块名	模块说明
QtCore	非图形化核心类，它里面涵盖的类(QThread，QChar 等)可以供其他模块使用，头文件<QtCore>
QtGui	图形化用户界面类，它涵盖了 Qt 下面所有的界面控件(对话框，按钮等)，头文件<QtGui>
phonon	多媒体框架

续表 1.1

模块名	模块说明
Qt3Support	Qt3 的兼容类，提供了由 Qt3 程序向 Qt4 程序移植的类库
QtDBus	采用 D-Bus 总线进行进程间通讯的类，该模块仅在 UNIX 中用
QtDesigner	扩展 Qt Designer 的类
QtHelp	提供在线帮助的类
QtMultimedia	提供底层多媒体编程的类(QAudioInput，QAudio)，头文件<QtMultimedia>
QtNetWork	提供让用户轻松便捷构建网络程序的类(QFtp，QTcpServer 等)，头文件<QtNetwork>
QtOpenGL	提供 2D/3D 图像处理接口，头文件<QtOpenGL>
QtSql	Qt 操作 SQL 数据库的类
QtSvg	显示和创建 SVG 文件的类
QtWebKit	提供一个 Web 浏览器引擎来和 Web 内容进行交互，头文件<QtWebKit>
QtXml	处理 XML 的类
QtXmlPatterns	用于 XML 和定制数据模型的 XQuery、XSLT 和 XPath 引擎
QtScript	Qt 的脚本引擎

接下来，在了解了 Qt SDK 是什么之后，我们开始使用它们。

1.2.2 用 SDK 编译出第一个运行在 Linux 下的软件界面

1. main.cpp 编写及 Qt 工程开发步骤

好了，是时候开始敲一下代码了，为此刻已经做了很多的准备工作。先创建一个工程目录“mkdir project1”，进入目录，创建 main.cpp 文件，编写代码如下：

```
1. #include<QtGui>
2. int main(int argc,char * argv[])
3. {
4.   QApplication app(argc,argv);
5.   QWidget * widget = new QWdiget(0);
6.   widget->show();
7.   return app.exec();
8. }
```

代码内容暂时可以先不理解，先让程序跑起来再说。在编译 main.cpp 文件之前，我们需要设置 PATH 环境变量，以便接下来可以搜索到用到的指令。关于 PATH 环境变量，其实一直在得到它的帮助。在之前我们敲入的那些 Linux 指令，操作系统都是在 PATH 环境变量指示的路径下搜索到的。比如“ls”在“/bin”目录下，但是不用输入“/bin/ls”来执行这个指令，而是直接在任何路径下输入“ls”来运行

该指令，这些就是 PATH 做的事情。它告诉系统去哪个目录找哪个指令，从而摆脱了输入指令绝对路径带来的烦恼。既然要用到 Qt 的工具，就来把这些工具的路径也加到 PATH 下面吧。执行下面的指令：

```
export PATH = /opt/QtSDK/QtCreator/bin:/opt/QtSDK/Desktop/Qt/474/gcc/bin: $ PATH
root@libin:~/project1 # echo $ PATH
/opt/QtSDK/QtCreator/bin:/opt/QtSDK/Desktop/Qt/474/gcc/bin:/usr/local/sbin:/usr/
local/bin:/usr/sbin:/usr/bin:/sbin:/bin:/usr/X11R6/bin
```

输入上面的指令后，就可以看到 PATH 内容中有刚才添加的路径了。接下来执行“qmake - project”来生成工程文件(project1. pro)，然后再执行“qmake”则会根据工程文件生成 Makefile，执行“make”就可以生成执行文件(project1)，经过这几个指令的执行，现在输入“ls”看一下工程目录 project1 的内容。

```
root@libin:~/project1 # ls
main.cpp  main.o  Makefile  project1  project1.pro
root@libin:~/project1 #
```

生成的执行文件名和文件目录名一样。执行这个 project1 程序，诞生在我们手上的第一个在 Linux 系统里面运行的软件界面就呈现在面前了，如图 1.6 所示。

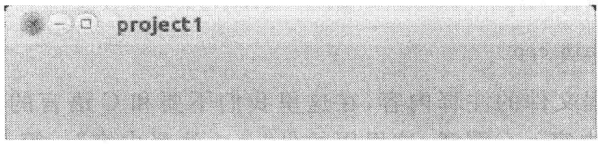

图 1.6　第一个软件界面

2. main. cpp、工程文件及工程文件模版使用和 Makefile 解析

一起来看一下这个界面到底是怎么蹦出来的。首先认识一下 main. cpp 的内容，代码的第 1 行包含了头文件 QtGui，它在“/opt/QtSDK/Desktop/Qt/474/gcc/include/QtGui”路径下面，包含了 Qt 的 GUI 类定义的头文件。在第 5 行用到的 QWidget 就在这个头文件的子文件＜qwidget. h＞中定义的。这个界面的样式已经看到了，QWidget 是 Qt GUI 一个基础窗体，这个还要在后面使用到。第 2 行是 main 函数，带了两个形参。第 4 行代码用 main 传递的命令行参数(argc，argv)构造了一个 QApplication 类的对象 app。这个对象负责管理整个用户界面的资源，是 Qt GUI 程序必须包含的一个对象。用户也可以通过全局变量 qApp(指向 QApplication 对象的一个指针)来访问这个对象，通过这个对象，来自窗口系统和其他资源的事件可以被调度和分发处理。它也可以使用用户的桌面设置来初始化应用程序、接收命令行参数、分配应用程序颜色、文本编码格式的指定、管理应用程序的鼠标光标处理、获取应用程序的窗口和提供对话管理等，这些都是 app 该做的事情。既然它这么重要，那必须在其他任何的用户界面对象创建之前创建它。接着第 5 行在堆里面 new 出来了一个基础窗体对象。new 是 C＋＋语言的一个关键字，用于动态内存创

建，和 malloc 函数作用类似，释放内存的时候和 new 配对的是 delete，而和 malloc 配对的是 free 函数，两者不能混搭了，即用 new 申请的内存不能用 free 释放，用 free 也只能释放掉用 malloc 申请的内存。第 6 行让这个窗体显示出来，最后通过 QApplication 类的 exec()函数调用，使应用程序进入主事件循环并等待，直到 exit()被调用或者主窗口部件被销毁，app－＞exec()调用之后，就可以开始事件处理，主事件循环从窗口系统接收事件并分派给应用程序窗口部件，至此用户界面程序才可以正常和用户交互。读者可以用鼠标单击软件的标题栏，移动软件位置，最大化、最小化和关闭这个软件，这些和应用界面基本交互的操作，QApplication 对象 app 功不可没。

project1. pro 工程文件内容解析如下。

打开 qmake - project 生成的工程文件 project1. pro，内容如下：

```
1 #############################################################
2 # Automatically generated by qmake (2.01a)
3 #############################################################
4
5 TEMPLATE = app
6 TARGET =
7 DEPENDPATH += .
8 INCLUDEPATH += .
9
10 # Input
11 SOURCES += main.cpp
```

1～3 行是工程文件的注释内容，在这里我们不要和 C 语言的注释风格(//＋注释内容或/＊注释内容＊/)混淆，这里用的是“＃＋注释内容”。第 5 行 TEMPLATE ＝ app 指明了依据该工程文件将建立一个应用程序的 Makefile，编译出一个可执行的应用程序。这个也是模版(TEMPLATE)的默认值，如果把 app 换成 lib，将建立一个库的 Makefile，编译生成的是动态链接库(. so 文件)，我们的库文件可以供别的工程使用。Qt 为了和 Visual Studio 兼容，TEMPLATE 还可以取值 vcapp 或 vclib，这样可以建立一个应用程序的 Visual Studio 项目文件或建立一个库的 visual studio 项目文件。在实际的开发中，经常需要向目标客户提供产品的二次开发包，而又需要保证公司产品的核心算法代码不外漏的情况，这时就不能直接把源代码毫无保留地提供给客户了，而是把那些核心代码制作成动态链接库，以库的形式提供给客户做二次开发，并向客户提供该库的详细接口说明。另外制作动态链接库也有利于代码的维护和升级。第 6 行 TARGET 可以指定生成目标的名字，不指定则默认生成和工程名一样的执行文件。第 7、8 行可以指定依赖和头文件路径。第 11 行是源码列表。我们先在 project1. pro 工程文件的基础上把 TEMPLATE＝app 改成 TEMPLATE ＝lib，来生成库文件(生成静态链接库或生成动态链接库)及练习如何使用链接库。

3. 如何生成动态链接库

修改了工程文件后，需要重新执行 qmake 来生成建立动态链接库的 Makefile。

在生成新的 Makefile 之前，把原来生成 app(应用程序)的 Makefile 先备份一份(cp Makefile Makefile.app)，便于后面对这些 Makefile 进行比较分析。

操作步骤如下：

```
root@libin:~/project1# cp Makefile Makefile.app
root@libin:~/project1# qmake
root@libin:~/project1# make
rm -f libproject1.so.1.0.0 libproject1.so libproject1.so.1 libproject1.so.1.0
g++ -Wl,-O1 -Wl,-rpath,/opt/QtSDK/Desktop/Qt/474/gcc/lib -shared -Wl,-soname,
libproject1.so.1 -o libproject1.so.1.0.0 main.o     -L/opt/QtSDK/Desktop/Qt/474/
gcc/lib -lQtGui -lQtCore -lpthread
ln -s libproject1.so.1.0.0 libproject1.so
ln -s libproject1.so.1.0.0 libproject1.so.1
ln -s libproject1.so.1.0.0 libproject1.so.1.0
root@libin:~/project1#
```

编译过程中用 ln 指令生成了动态链接库的软链接文件。我们先来认识一下 ln 指令，刚才提到了软链接，做为经常在 Windows 编程的读者来说，可能会对这个称谓感到很陌生，那换个称谓——快捷方式。没错，Linux 里面的软链接就类似于 Windows 的快捷方式，可以指向目录也可以指向文件，只不过要想建立 Linux 里面的“快捷方式”我们需要借助于 ln 指令。链接有两种：软链接(也称符号链接)和硬链接。两者的区别在于硬链接不能建立指向目录的链接，另外硬链接在建立的时候必须保证链接文件和被链接文件在同一种文件系统里面，这些限制在建立软链接时不起作用。关于 Linux 下支持的文件系统的种类和介绍，已经不在本书的讨论范围内，感兴趣的读者可以自行查阅相关资料或与笔者进行 email 交流。

ln 指令用法

格式一：ln　[选项]…[-T]目标 链接名　作用：创建指定名称且指向指定目标链接。

格式二：ln　[选项]…目标　作用：在当前目录创建指向目标位置的链接。

格式三：ln　[选项]…-t 目录 目标…　作用：在指定目录中创建指向指定目标的链接。

常用选项说明：

-f　强行删除已存在的目标文件；

-s　创建软链接；

-t　在指定的目录中创建链接。

上文在 make 编译工程时，就执行“ln - s”建立指向 libproject1. so. 1. 0. 0(编译生成的动态链接库文件)的 3 个软链接文件(libproject1. so，libproject1. so. 1 和 libproject1. so. 1. 0)。动态库的版本号是 1. 0. 0，用版本号为后缀表示库名为在同一系统里面使用同一库的不同版本提供了便捷，但是程序在链接动态库时，默认搜索“. so”后缀的文件，因而为了能使用这些库，需要建立指向它们的软链接(链接名以. so 为后缀)，这就是工程编译的时候建立软链接的意义了。

4. 测试使用动态连接库

把 main. cpp 的 main 函数更名为 maina，当第三方程序链接库 libproject1. so 时，调用 maina 即可绘制出一个软件界面。细节都在库中实现了，使用者只需调用接口，就如同使用 Qt 库一样。重新编译工程后，执行如下指令：

```
root@libin:~/project1# mkdir -p testlib/libpath testlib/libinc
root@libin:~/project1# cp -av libproject1.so* testlib/libpath/
"libproject1.so" -> "testlib/libpath/libproject1.so"
"libproject1.so.1" ->"testlib/libpath/libproject1.so.1"
"libproject1.so.1.0" ->"testlib/libpath/libproject1.so.1.0"
"libproject1.so.1.0.0" ->"testlib/libpath/libproject1.so.1.0.0"
root@libin:~/project1# cd testlib/libinc/
root@libin:~/project1/testlib/libinc# vim mylib.h
```

这些指令的主要功能是创建测试库的工作目录 testlib 及建立库的存放目录 libpath 和库的接口声明文件 mylib. h 所存储的目录 libinc，接着把生成的库文件及软链接文件一并复制到 libpath 下面，之后进入 libinc 目录下，创建 mylib. h 文件，内容如下(接口 maina 的声明)：

```
1 #ifndef MY_LIB_H
2 #define MY_LIB_H
3 /*
4   调用该接口可以在屏幕上显示一个窗体
5    argc ---- 参数个数
6    argv ---- 参数内容
7 */
8 int maina(int argc,char *argv[]);
9
10 #endif //MY_LIB_H
```

mylib. h 简要说明：

代码 1、2、10 行防止头文件重复包含。

第 3～7 行是接口功能和参数的简要说明。

第 8 行是接口的声明。

试读结束：需要全本请在线购买： www.ertongbook.com