

全新打造的Qter开源社区！

短期即可入门及提高的优秀Qt系列书籍！

Qt 应用编程系列丛书

完整版请联系QQ: 859570073 (豆豆电子书)

本人因为职务便利，不管是什么类型的图书
只要不是很新，基本都能找到电子版

霍亚飞 程 梁 编著

Qt 5 编程入门

- **全新**，基于 Qt 5.3 编写，全面涉及 Qt Quick！
- **经典**，植根于 Qt 网络博客教程，可无限更新！
- **基础**，对每个知识点详尽讲解，并设计了示例程序！
- **系统**，与《Qt Creator 快速入门（第2版）》相辅相成！



北京航空航天大学出版社
BEIHANG UNIVERSITY PRESS

Qt 应用编程系列丛书

Qt 5 编程入门

霍亚飞 程 梁 编著

清华大学出版社

清华大学出版社
北京 100084

北京航空航天大学出版社

内 容 简 介

本书是基于 Qt 5 的 QML 和 Qt Quick 入门书籍,详细介绍了 QML 语言的语法和编写 Qt Quick 程序需要用到的基本知识点。本书内容主要包括 Qt 5 的介绍、QML 语法、Qt Quick 基础知识以及在图形动画、数据处理和多媒体方面的应用等。

本书侧重讲解 Qt 5 中 QML 和 Qt Quick 的内容,适合希望学习 QML 编程以及希望使用 Qt 开发移动应用的读者。对于没有任何 Qt 基础或者想学习 Qt C++ 编程的读者,可以参考《Qt Creator 快速入门(第 2 版)》一书。

本书内容源于作者的网络博客,作者会在网上及时解答读者疑问、更新修改内容、发布相关教程和配套资料,敬请读者关注 Qter 开源社区(www.qter.org)的相关内容。

图书在版编目(CIP)数据

Qt 5 编程入门 / 霍亚飞,程梁编著. —北京:北京航空航天大学出版社,2015.1
ISBN 978-7-5124-1667-3

I. ①Q… II. ①霍… ②程… 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(2015)第 009999 号

版权所有,侵权必究。

Qt 5 编程入门

霍亚飞 程 梁 编著
责任编辑 董丽娟 张耀军

*

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

北京楠海印刷厂印装 各地书店经销

*

开本:710×1 000 1/16 印张:24.25 字数:517 千字

2015 年 1 月第 1 版 2015 年 1 月第 1 次印刷 印数:4 000 册

ISBN 978-7-5124-1667-3 定价:54.00 元

前言

本书的原型是网络上的一系列 Qt 博客教程,是该系列教程衍生出的第三本。2012 年出版的《Qt Creator 快速入门》和《Qt 及 Qt Quick 开发实战精解》至今已经有两年多的时间,在这期间 Qt 的命运发生了翻天覆地的变化。2012 年 8 月,诺基亚将 Qt 出售给同在芬兰的 Digia 公司,后者致力于发展 Qt 的商业授权用户,同时提供了 Qt 社区的良好运行环境。伴随着 Qt Project(qt-project.org)的成立,Qt 本身的开发终于成为一个真正的开放源代码项目。之后的 Qt 5 发布,使得 Qt 已经开始支持包括 Windows、Linux、Mac OS X、Android、iOS、BlackBerry、Sailfish 等在内的十几个桌面和移动平台。我们看到 Qt 的身世一波三折,但是又有足够的理由相信 Qt 的未来会越来越好。

因为在博客中发表 Qt 教程,所以认识了很多 Qt 爱好者和普及者,其中包括本书的另外一名作者程梁(网名豆子、devbean)。豆子与我的经历类似:我从 2009 年开始在百度博客上编写 Qt Creator 系列教程,因为图文并茂,简单易懂,所以得到了众多网友的肯定;而豆子也从 2009 年开始,在 51 CTO 博客中推出 Qt 学习之路系列教程,因为讲解细致,拥有自己的独到见解,所以受到了广大网友的热捧,后来被 51 CTO 专题收录,现在总浏览量已有数百万。后来我从百度博客迁移到自己的 yafeilinux.com 个人博客,而豆子也从 51 CTO 博客转移到自己的个人博客网站 devbean.net。有着如此相似经历的两个人走到一起会发生什么呢?就是组建了更符合网友需求的 Qter 开源社区(qter.org)。这个社区一直在发展,我们的目标就是要为各位致力于学习 Qt 的朋友提供一个便捷之门。

随着 Qt 5 的发布,我们经常可以在论坛上看到需要 Qt 5 相关教程和书籍的朋友。Qt 5 的全新框架、Qt Quick 2 技术以及最新版本的 Qt 提供了对现在非常流行的 Android 和 iOS 等移动平台的支持,都使得现在想要学习 Qt 的读者数量剧增。现阶段国内外图书市场上极少有关于 Qt 5 的书籍,但是却拥有众多需要该类书籍的初学者和开发者。为了解决这个矛盾,我和豆子决定再做一次第一个吃螃蟹的人,从而有了本书。

Qt 的历史

Qt 是由奇趣科技公司(Trolltech)的两位创始人 Haavard Nord 和 Eirik Chambe-Eng 合作开发而成的,首个版本完成于 1991 年,而第一个商业版本则是在 1995 年才正式推出。

Qt 的名字来源于一个有趣的故事。当时这个工具之所以取名为 Qt,是因为在 Haavard 的 Emacs 字体中,字母 Q 看起来非常漂亮;而字母 t 的灵感则来源于当时的另外一个工具 Xt(X toolkit)的取名。

最初 Qt 只有两个版本:适用于类 Unix 平台的 Qt/X11 和适用于 Windows 平台的 Qt/Windows。其中,Windows 版本只有专利授权,这意味着如果需要将使用 Qt/X11 编写的开源应用移植到 Windows 平台,必须购买专利授权。

2001 年底,奇趣科技发布 Qt 3.0,从这个版本起,Qt 增加了对 Mac OS X 平台的支持。不过,直到 2003 年 6 月之前,Mac OS X 版本的 Qt 也只有专利授权。当 Qt 3.2 发布的时候,奇趣科技才增加了 Mac OS X 平台的 GPL 授权。

2005 年 6 月 28 日,奇趣科技发布了 Qt 4.0。这是一个全新的版本,与之前的 3.x 系列不仅二进制不兼容,甚至 API 也不兼容。这在一定程度上使人颇有微词。

2008 年 6 月 17 日,诺基亚公司宣布以公开竞购的方式收购奇趣科技,连同奇趣科技旗下的 Qt。同年,诺基亚将 Qt 的名字更改为 Qt Software,然后又更改为 Qt Development Framework。在诺基亚的领导下,Qt 的工作重心由桌面系统转移至诺基亚旗下的手持设备。2009 年 5 月 11 日,诺基亚宣布 Qt 源代码在著名的 git 托管平台 Gitorious 面向公众开放,标志着 Qt 正式成为面向社区的开源框架。不过,尽管诺基亚承诺 Qt 开放源代码,但事实上 Qt 的代码提交与维护仍然牢牢掌握在诺基亚手中。2009 年 12 月 1 日,Qt 4.6 发布,这是第一个移植到 Symbian S60 平台的版本。

2011 年 2 月,诺基亚宣布放弃自己的 Symbian 平台,转而投向微软公司的 Windows Phone 平台。一个月之后,诺基亚宣布将 Qt 的商业授权和专业服务出售给 Digia 公司。后者启动了 Qt Project,同时宣布将努力促使 Qt 支持 Android、iOS 和 Windows Phone 三大平台,并且继续关注于桌面和嵌入式平台的开发,这意味着 Qt 正在努力成为一个全平台的开发框架。现在看来,在一定程度上,Digia 的确做到了这一点。

Qt 4 之后的下一个大的版本 Qt 5 原计划于 2012 年 6 月发布,但是由于诺基亚的政策调整和出售 Qt 的缘故,直到 2012 年 12 月 19 日,Qt 5.0 才正式发布。这是继 Qt 4 之后另一个大的升级。表面看来,Qt 4 到 Qt 5 的改动并不比 Qt 3 到 Qt 4 来得激进,但是 Qt 5 引入了全新的硬件加速图形处理,并且将 QML 与 JavaScript 提升到同 C++ 相等的地位。传统的基于 C++ 的 Qt Widgets 仍将继续支持,但是全新的架构所带来的性能提升则更多作用于 QML 和 JavaScript。

值得一提的是,Qt 5 的开发真正由 Qt Project 社区驱动,现在 Qt 已经允许诺基亚和 Digia 之外的开发人员提交并审核代码。

Digia 接管 Qt 业务以后,Qt 的开发速度有了明显提升。2013 年 7 月 3 日,Qt 5.1 发布。Qt 5.1 除了修改 5.0 版本的 bug 之外,更带来了 Android 和 iOS 平台的实验性支持。2013 年 12 月 12 日发布的 Qt 5.2 版本,则正式引入对 Android 和 iOS 平台的支持;同时,Qt 5.2 引入了一个新的场景图像渲染器,针对 OpenGL 后端提升了矢量绘制的性能,将 GPU 的占用降至最低。2014 年 5 月 20 日,Qt 5.3 发布。Qt 5.3 改进了对 iOS 平台的支持,并且开始支持 WinRT 平台。这为 Qt 成为全平台的开发工具更进一步。

Qt 授权

Qt 的授权一直比较复杂,既包括商业授权,又包括开源授权,并且不同版本之间的

授权并不一致。下面我们来详细介绍下有关授权的问题。

首先需要说明的是,Qt 自发明以来一直都有一个商业授权。这个商业授权允许开发者开发专有程序。同时,Qt 商业授权除了包含 Qt 开源版的一切功能外,还包括一些企业应用的组件,比如 Qt Charts、Qt Data Visualization 等;还有 Digia 公司支持的 Qt 培训等内容。

真正复杂的是 Qt 开源授权。Qt 1.45 之前,Qt 的源代码一直遵循 FreeQt 协议。该协议既不符合 Open Source Initiative 定义的开放源代码原则,也不符合 Free Software Foundation 定义的自由软件。在此协议下,Qt 的源代码虽然可用,但是并不允许重新发布修改过的版本。

1998 年,基于 Qt 的 KDE 成为 Linux 最流行的桌面环境之一。不过,因为 Qt 协议的问题,很多人担心这将给 KDE 桌面环境的发展带来影响。这也正是日后 Qt 协议问题的根源之一。

Qt 2.0 的开源授权由 FreeQt 更改为 Q Public License(QPL)。这是一个自由软件协议,但是与 GPL 不兼容,而后者才是 Linux 遵循的协议。作为妥协,KDE 团队与当时的奇趣科技达成协议,即使奇趣科技濒临破产,也不能使 Qt 的协议比 QPL 更加专有。直到 2000 年,Qt/X11 2.2 正式以 GPLv2 协议发布,才终结了这一授权的法律问题。

2002 年,KDE on Cygwin 项目开始将 GPL 授权的 Qt/X11 移植到 Windows 平台。此时,Windows 平台还没有一个开源版本的 Qt。不过,该项目并没有取得成功,但却直接导致奇趣科技在 2005 年 6 月将 Qt/Windows 4 以 GPL 授权的形式发布。至此,Qt 4 在全部主流桌面平台都有了基于 GPL 协议的开源版本。随着 GPLv3 发布,Qt 又增加了具有额外条款的 GPLv3 协议。这个“额外条款”允许将 Qt 编写的最终应用程序以不兼容 GPL 的自由软件/开源协议发布。

虽然此时的 Qt 已经有了自己的开源版本,但是作为一个类库,使用 GPL 协议开源具有一定的危险性。由于 GPL 的传染性,GPL 不允许发布闭源程序。这种协议将极大地限制 Qt 在商业软件领域的应用。所以在 2009 年 1 月 14 日,Qt 4.5 终于众望所归地增加了 LGPL 协议,允许使用 Qt 开发闭源程序。

尽管历史很复杂,但是我们的结论很简单:最新的 Qt 5.3 版本使用两种协议发布——商业协议和 LGPL 2.1。

本书特色

本书是经典的 Qt 应用编程系列丛书继《Qt Creator 快速入门》和《Qt 及 Qt Quick 开发实战精解》之后的又一力作,是市面上为数不多的全面介绍 Qt 5、QML 和 Qt Quick 的入门书籍。与其他相关书籍最大的不同之处还包括,本书是基于网络教程的。综合来说,本书主要具有以下特色:

- 最新。本书基于最新的 Qt 5.3.0 和 Qt Creator 3.1.1 版本进行编写,Qt 5.3.0 是 Qt 5 的最新的稳定版本。
- 基于社区。本书以 Qt 开源社区为依托,由社区站长合作完成。读者可以通

过论坛、邮件、QQ 群等方式和作者零距离交流。

- 无限更新。本书对应的网络教程是无限更新的,本书已经是该网络教程衍生的第三本书了。
- 全新风格。本书力求以全新的视角,引领开发者进行程序代码的编写和升级,同时以初学者的角度进行叙述,每个小知识点都以一个完整的程序来讲解。尽量避免晦涩难懂的术语,使用初学者易于理解的平白的语言编写,目标是让初学者在快乐中掌握知识。
- 授之以渔。在整书的编写过程中,都是在向读者传授一种学习方法,告诉读者怎样发现问题、解决问题,怎样获取知识,而不是向读者灌输知识。本书的编写基于 Qt 参考文档,所讲解的知识点多数是 Qt 参考文档中的部分内容,大家在学习时一定要多参考 Qt 帮助文档。在本套书籍讲解的所有知识点和示例程序中,都很明显地标出了其在 Qt 帮助中对应的关键字,从而让读者对书中的内容有迹可循。

本书结构

本书着重讲解 Qt 5 中 QML 和 Qt Quick 的编程知识,首先对 Qt 5 进行了整体介绍以及如何从 Qt 4 项目移植到 Qt 5 的介绍,然后讲解了 QML 语法和 Qt Quick 的基础内容,后面的几章讲解了 Qt Quick 在图像特效、绘画、多媒体、数据显示和处理等方面的高级应用。全书的目录结构如下表所列。

目录	介绍
第 1 章 Qt 5 简介	Qt 5 整体介绍,从 Qt 4 过渡到 Qt 5 需要了解的内容
第 2 章 将 Qt 4 代码迁移到 Qt 5	
第 3 章 Qt Quick 的世界	QML 和 Qt Quick 基础内容, 编写 QML 应用程序必备的基础知识
第 4 章 QML 语法	
第 5 章 Qt Quick 基础	
第 6 章 Qt Quick 控件和对话框	
第 7 章 图形动画基础	
第 8 章 图形效果	Qt Quick 高级应用,要设计功能强大、 界面绚丽的 Quick 应用必备的专业知识
第 9 章 粒子系统	
第 10 章 Canvas 基础	
第 11 章 模型和视图	
第 12 章 多媒体应用	
第 13 章 QML 与 C++ 的集成	
第 14 章 使用 Qt Quick 设计器	

另外需要说明,使用 Qt 进行 Android 和 iOS 等移动平台的开发部分(包括环境设置、传感器应用、蓝牙通信、摄像头控制、多点触控等相关内容),鉴于相关内容还不是非常成熟,而且依赖于特定设备,所以该部分的内容没有直接编写到书中,而是放到了网

络教程中。这样可以更及时地进行更新,并且对设备等相关的问题与读者进行直接探讨。

使用本书

本书的主要内容是讲解 Qt 5 中的 QML 和 Qt Quick 开发,另外还包含了对 Qt 5 框架和移植的介绍等内容。对于 QML 初学者,建议先学习 QML 和 Qt Quick 的基础知识,等有了一定的基础后再来学习后面的内容。

本书假设读者已经具有一定的 Qt 编程基础,因而不会对 Qt 的基础面面俱到。如果大家希望系统地学习 Qt 编程,最好先自行学习基于 C++ 的 Qt Widgets 编程方面的内容。可以参考 Qt 应用编程系列丛书中的《Qt Creator 快速入门(第 2 版)》一书。《Qt Creator 快速入门(第 2 版)》的写作风格与本书一致。

对于学习过《Qt 及 Qt Quick 开发实战精解》(以下简称《精解》)一书的读者,这里做一个解释:虽然《精解》一书中也包含了 Qt Quick 的内容,但那是 Qt 4.7 时代的 Qt Quick,因为是新引进的技术,所以非常不成熟。本书中的 QML 和 Qt Quick 的内容是基于最新的 Qt 5。新版本的 Qt Quick 已经进行了巨大的更新。另外,在以后《精解》一书再版时,其中的 Qt Quick 内容会被删除。

在学习过程中,建议读者多动手,尽量自己按照步骤编写代码。只有在遇到自己无法解决的问题时,再去参考本套书籍提供的源代码。每当学习一个知识点时,书中都会给出 Qt 帮助中的关键字,建议大家详细阅读 Qt 帮助文档,看下英文原文是如何描述的。不要害怕阅读英文文档,因为很难在网上找到所有文档的中文翻译,并且即使翻译,也有可能偏离原意,所以最终还是要自己去读原始文档。学会看参考文档是入门 Qt 编程的重要一步。不要说自己英文不好,只要坚持,掌握了一些英文术语和关键词以后,阅读英文文档是不成问题的。

书中使用的 Qt 版本的说明

本书基于 Windows 平台 Qt 5.3.0 和 Qt Creator 3.1.1 版本。它们是本书完稿时的最新版本。为了避免读者使用不同的操作系统而产生不必要的问题,本书采用了常用的 Windows 7 操作系统。这里要对 Qt 版本不是很了解的读者说明一下,对于 Qt 程序开发,无论是在 Windows 系统下进行开发,还是在 Linux 系统下进行开发;无论是进行桌面程序开发,还是进行移动平台或者嵌入式平台的开发,只需要编写一次代码,然后分别进行编译就可以了,这就是 Qt 最大的特点,即所谓的“一次编写,随处编译”。也就是说,读者需要学好本书中 Qt 的基本内容,然后编写代码,使用 Qt 不同的版本进行移植、编译即可。

在学习本书时,推荐大家使用指定的 Qt 和 Qt Creator 版本,因为对于初学者来说,任何微小的差异都可能导致错误的理解。当然,大家也可以使用其他版本,Qt 的下载地址为:http://download.qt.io/official_releases/qt/。



录

第 1 章 Qt 5 简介	1	2.5 全新的插件系统	29
1.1 Qt 5 架构	1	2.6 小结	32
1.1.1 模块架构	1	第 3 章 Qt Quick 的世界	33
1.1.2 图形界面库的架构	5	3.1 全新的 QML 和 Qt Quick ...	33
1.1.3 Qt 5 架构主要特点	6	3.2 Qt Quick 项目	36
1.2 Qt 5 的特点	6	3.2.1 Qt Quick UI 项目	36
1.2.1 Qt 5 新增的主要功能	6	3.2.2 Qt Quick Application 项目	39
1.2.2 Qt 5 与 Qt 4 的兼容性	11	3.2.3 运行示例程序	42
1.2.3 C++ 还是 QML	12	3.2.4 Qt Quick 程序的发布	42
1.2.4 Qt 5 源代码文件的编码	12	3.3 将 QML 程序迁移到 Qt 5 ...	42
1.2.5 移动平台开发	13	3.3.1 Qt 5 中 QML 和 Qt Quick 的更改	43
1.3 小结	13	43
第 2 章 将 Qt 4 代码迁移到 Qt 5 ...	14	3.3.2 QML 程序移植示例	46
2.1 Qt 5 版本的 HelloWorld	14	3.4 小结	48
2.1.1 Qt 5 的下载与安装	14	第 4 章 QML 语法	49
2.1.2 创建 Hello World 程序	15	4.1 QML 语法基础	49
2.1.3 Qt 4 程序迁移到 Qt 5 的注意事项	17	4.2 import 导入语句	51
.....	17	4.2.1 模块(命名空间)导入语句	52
2.2 Qt 4 程序迁移实例	19	4.2.2 目录导入语句	53
2.2.1 修改编码	19	4.2.3 JavaScript 资源导入语句	55
2.2.2 修改代码	20	4.3 QML 类型系统	55
2.2.3 设置应用程序图标	21	4.3.1 基本类型	55
2.2.4 发布程序	22	4.3.2 JavaScript 类型	57
2.3 新的信号槽语法	22	4.3.3 对象类型	57
2.3.1 新旧语法对比	22	4.4 对象特性(Attributes)	57
2.3.2 新的语法示例	23	4.4.1 id 特性	58
2.4 对 C++11 的支持	26	4.4.2 属性特性	58
2.4.1 Lambda 表达式	26	4.4.3 信号和信号处理器特性	65
2.4.2 适用于 C++11 的宏	28	4.4.4 方法特性	69

4.4.5 附加属性和附加信号处理器	70	件(WheelEvent)	135
4.5 集成 JavaScript	72	5.3.3 拖放事件(DragEvent)	136
4.5.1 JavaScript 表达式	72	5.3.4 键盘事件(KeyEvent)和焦点作用域(FocusScope)	140
4.5.2 从 JavaScript 动态创建 QML 对象	77	5.3.5 定时器(Timer)	145
4.5.3 在 QML 中定义 JavaScript 资源	81	5.4 使用 Loader 动态加载组件	145
4.5.4 在 QML 中导入 JavaScript 资源	83	5.4.1 Loader 的大小与行为	146
4.5.5 JavaScript 宿主环境	85	5.4.2 从加载的项目中接收信号	147
4.6 QML 文档	86	5.4.3 焦点和键盘事件	147
4.6.1 通过 QML 文档定义对象类型	87	5.5 小结	148
4.6.2 QML 组件	89	第 6 章 Qt Quick 控件和对话框	149
4.6.3 作用域和命名解析	90	6.1 构建第一个示例	149
4.6.4 资源加载和网络透明性	95	6.2 ApplicationWindow 应用程序窗口	150
4.6.5 QML 的国际化	97	6.3 Window	154
4.6.6 QML 的编码约定	100	6.4 按钮类控件	155
4.7 QML 模块	103	6.4.1 ExclusiveGroup	156
4.7.1 定义一个 QML 模块	103	6.4.2 Button	157
4.7.2 支持的 QML 模块类型	104	6.4.3 CheckBox	157
4.8 小结	104	6.4.4 RadioButton	158
第 5 章 Qt Quick 基础	105	6.4.5 Switch	158
5.1 基础可视项目	105	6.5 数据选择类控件	159
5.1.1 Item	105	6.5.1 ComboBox	159
5.1.2 Rectangle	109	6.5.2 Slider	160
5.1.3 Text	110	6.5.3 SpinBox	161
5.1.4 TextInput	117	6.6 文本类控件	161
5.1.5 TextEdit	120	6.6.1 TextField	161
5.2 布局管理	121	6.6.2 TextArea	162
5.2.1 定位器(Positioners)	121	6.7 其他控件	163
5.2.2 基于锚(anchors)的布局	126	6.7.1 Label	163
5.2.3 Layouts	129	6.7.2 GroupBox	163
5.3 事件处理	133	6.7.3 BusyIndicator	164
5.3.1 MouseArea	133	6.7.4 ProgressBar	164
5.3.2 鼠标事件(MouseEvent)和滚轮事		6.8 导航类视图	165
		6.8.1 ScrollView	165
		6.8.2 SplitView	165

6.8.3 StackView	166	7.6.2 翻转效果(Flipable)	206
6.8.4 TabView	170	7.7 小 结	207
6.8.5 TableView	171	第 8 章 图形效果	208
6.9 标准对话框	172	8.1 混合效果(Blend)	209
6.9.1 ColorDialog	173	8.2 颜色效果(Color)	211
6.9.2 FileDialog	173	8.3 渐变效果(Gradient)	214
6.9.3 FontDialog	174	8.3.1 锥形渐变(ConicalGradient)	
6.9.4 MessageDialog	175	214
6.10 定义控件样式	176	8.3.2 线性渐变(LinearGradient)	
6.11 小 结	177	215
第 7 章 图形动画基础	178	8.3.3 辐射渐变(RadialGradient)	
7.1 颜色、渐变和调色板	178	215
7.1.1 颜色(color)	178	8.4 变形效果(Distortion)	216
7.1.2 渐变(Gradient)	181	8.5 阴影效果(Drop Shadow) ...	217
7.1.3 系统调色板(SystemPalette)		8.5.1 投影(DropShadow)	217
.....	181	8.5.2 内阴影(InnerShadow)	218
7.2 图片、边界图片和动态图片	182	8.6 模糊效果(Blur)	219
7.2.1 图片(Image)	182	8.6.1 快速模糊(FastBlur)	219
7.2.2 边界图片(BorderImage)	184	8.6.2 高斯模糊(GaussianBlur)	220
7.2.3 动态图片(AnimatedImage)		8.6.3 递归模糊(RecursiveBlur)	220
.....	185	8.6.4 遮罩模糊(MaskedBlur)	221
7.3 缩放、旋转和平移变换	186	8.7 动感模糊效果(Motion Blur) ...	
7.3.1 使用属性实现简单变换	186	222
7.3.2 使用 Transform 实现高级变换		8.7.1 方向模糊(DirectionalBlur) ...	222
.....	188	8.7.2 径向模糊(RadialBlur)	223
7.4 状态(State)	189	8.7.3 缩放模糊(ZoomBlur)	224
7.4.1 创建状态	190	8.8 发光效果(Glow)	225
7.4.2 默认状态和 when 属性	191	8.8.1 发光(Glow)	225
7.5 动画和过渡	192	8.8.2 矩形发光(RectangularGlow)	
7.5.1 触发动画	193	226
7.5.2 控制动画的执行	198	8.9 遮罩效果(Mask)	227
7.5.3 精灵动画(Sprite Animations)		8.9.1 不透明遮罩(OpacityMask)	
.....	199	227
7.5.4 共享动画实例	203	8.9.2 阈值遮罩(ThresholdMask)	
7.6 Flickable 和 Flipable	204	228
7.6.1 弹动效果(Flickable)	204	8.10 小 结	229
		第 9 章 粒子系统	230

9.1 ParticleSystem	230	11.2.5 Package	277
9.2 发射器(Emitter)	232	11.2.6 XmlListModel	278
9.3 渲染器(ParticlePainters) ...	233	11.2.7 LocalStorage	279
9.4 TrailEmitter	236	11.2.8 使用 C++ 扩展 QML 模型	282
9.5 粒子组	237	11.3 视图类型	285
9.6 随机参数	239	11.3.1 ListView	286
9.7 影响器(Affector)	240	11.3.2 GridView	291
9.8 小结	245	11.3.3 视图过渡	292
第 10 章 Canvas 基础	246	11.3.4 PathView	295
10.1 Canvas 的使用	246	11.3.5 WebView	298
10.2 绘制操作	248	11.4 调整性能	299
10.2.1 绘制参数设置	248	11.5 小结	300
10.2.2 绘制矩形	250	第 12 章 多媒体应用	301
10.2.3 状态的保存与恢复	251	12.1 多媒体模块介绍	301
10.2.4 绘制文本	252	12.2 播放音频	302
10.2.5 绘制路径	254	12.2.1 播放压缩音频	302
10.3 渐变填充	259	12.2.2 播放未压缩音频	304
10.3.1 线性渐变	259	12.3 播放视频	304
10.3.2 辐射渐变	260	12.3.1 使用 Video 播放视频文件	304
10.3.3 锥形渐变	261	12.3.2 对视频使用图形效果	305
10.4 阴影	261	12.4 媒体播放器(MediaPlayer)	308
10.5 使用图像	262	12.4.1 播放音视频	308
10.6 坐标变换	264	12.4.2 使用 Windows 平台附加功能	309
10.6.1 平移	264	12.5 小结	314
10.6.2 缩放	265	第 13 章 QML 与 C++ 的集成	315
10.6.3 旋转	266	13.1 QML 运行时的 C++ 类 ...	316
10.6.4 扭曲	267	13.1.1 QQmlEngine	316
10.6.5 transform() 函数总结	267	13.1.2 QQmlContext	317
10.7 小结	268	13.1.3 QQmlComponent	318
第 11 章 模型和视图	269	13.1.4 QQmlExpression	320
11.1 模型/视图架构简介	269	13.2 在 QML 中使用 C++ 特性 ...	320
11.2 数据模型	271	13.2.1 数据类型处理和所有权	320
11.2.1 ListModel	271		
11.2.2 WorkerScript	273		
11.2.3 ObjectModel	275		
11.2.4 DelegateModel	275		

13.2.2 数据类型的转换	321	13.5.3 使用 C++ 访问 QML 对象成员	340
13.2.3 使用 C++ 属性	323	13.6 小 结	343
13.2.4 使用函数和槽	327	第 14 章 使用 Qt Quick 设计器	344
13.2.5 使用信号	328	14.1 常用操作介绍	345
13.3 注册 QML 类型	329	14.1.1 管理项目层次	345
13.3.1 注册可实例化对象类型	329	14.1.2 QML 类型库	346
13.3.2 注册不可实例化对象类型	330	14.1.3 指定项目属性	347
13.3.3 注册单例类型	330	14.1.4 添加状态	349
13.3.4 类型的修订和版本	331	14.1.5 在画布上操作 QML 类型	349
13.4 定义 QML 特定类型和属性	332	14.2 使用设计器编辑 QML 文档	351
13.4.1 提供附加对象注解数据	332	14.2.1 创建项目	351
13.4.2 属性修饰符类型	335	14.2.2 创建主视图	351
13.4.3 指定 QML 对象类型的默认属性	337	14.2.3 添加视图	355
13.4.4 接收对象初始化通知	337	14.2.4 为视图添加动画	356
13.5 在 C++ 中使用 QML 对象	338	14.3 小 结	358
13.5.1 使用 C++ 加载 QML 对象	338	附录 A Qt 版本介绍	359
13.5.2 使用对象名字访问加载的 QML 对象	339	附录 B Qt 5 中 C++ API 更改	361
		附录 C QML 常用术语	371
		参考文献	373

第 1 章

Qt 5 简介

Qt 5 是跨平台应用开发框架 Qt 的最新版本,图形引擎基于 GPU,从而使应用程序能够充分利用硬件加速,同时包括了最新的应用开发套件 Qt Quick 2。作为 Qt 5 的核心之一,Qt Quick 2 提供了强大的功能,允许开发者进行适用于触摸屏的 UI 开发,并且保持与原生 C++ 程序几乎相同的运行效率。

在增加了对 C++ 11 支持的同时,Qt 5 还加强了对 JavaScript 和 QML 的支持,开发者可以更加高效和灵活地使用它们。不仅如此,QML 还可以与基于 C++ 的经典的 Qt Widgets 进行混合编程。Qt 5 是 Qt 4 渐进而平缓的升级,与 Qt 4 高度兼容,除了一些必要的情况,基本没有打破源代码级别的兼容性。

本章将从 Qt 5 的架构、功能更新等几个方面进行介绍,让读者对 Qt 5 有一个全面的认识。

1.1 Qt 5 架构

1.1.1 模块架构

Qt 5 采用了新的模块化代码库,使得平台移植变得更简单。Qt 5 将所有功能模块分为了 3 部分:Qt 基本模块(Qt Essentials)、Qt 扩展模块(Qt Add-Ons)和开发工具(Qt Tools)。可以在 Qt 帮助中通过 All Modules 关键字查看本节的内容。

Qt 基本模块定义了适用于所有平台的基础功能,是 Qt 的核心,其架构如图 1-1 所示。所有 Qt 应用程序都需要使用该模块中提供的功能。基本模块的基础是 Qt Core 模块,其他所有模块都依赖于该模块。Qt 保证,在 Qt 5 的整个生命

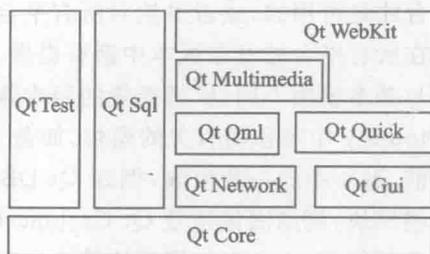


图 1-1 Qt 5 基本模块架构示意

周期内,这部分模块均会保持源代码甚至二进制级别的兼容性。另外,如果某个功能,即便是所有平台都能够支持,但是仅仅为了满足某个特定的功能或目的,那么这个功能也不会进入基本模块。典型的例子是打印,虽然绝大多数平台都有支持(甚至包括某些移动平台),但是打印模块并不包含在基本模块之中。

最新的 Qt 5.3 中基本模块部分简介如表 1-1 所列。

表 1-1 Qt 5.3 的基本模块简介

模块	简介
Qt Core	Qt 5 的核心类库,其他各个模块都建立在 Core 模块之上。相对于 Qt 4, Qt 5 增加了 JSON 支持,并且将对 XML 的支持移入到 Core 模块,不再是独立的 XML 模块
Qt GUI	图形用户界面(GUI)开发的最基础的类库,包括各种交互事件等;同时,这个模块还包括了有关 OpenGL 的内容
Qt Multimedia	提供对视频、音频、无线电以及摄像头的支持
Qt Multimedia Widgets	提供基于组件的多媒体功能的支持
Qt Network	提供网络编程类库
Qt QML	提供对 QML 和 JavaScript 语言的支持
Qt Quick	提供一个用于创建高度动画效果的应用程序的声明式框架。该框架建立在 QML 和 JavaScript 的基础之上
Qt Quick Controls	提供一组基于 Qt Quick 的 UI 控件,用于创建经典的桌面风格的用户界面
Qt Quick Dialogs	提供一组适用于 Qt Quick 应用程序的系统对话框
Qt Quick Layouts	提供一组适用于 Qt Quick 的组件布局
Qt SQL	提供对关系数据库 SQL 的支持
Qt Test	提供供 Qt 应用程序和类库使用的单元测试工具
Qt WebKit	提供基于 WebKit2 引擎的一个 Qt 移植,以及新的面向 QML 的 API
Qt WebKit Widgets	提供基于 WebKit1 的一个移植,以及来自于 Qt 4 的所有有关 WebKit 的类
Qt Widgets	提供针对 Qt GUI 的扩展 C++ 组件

Qt 扩展模块是针对某种特殊目的的额外模块。这些模块或者仅在某个或某些特定平台才是可用的,或者虽然对所有平台均可用,但却不是通用的目的。扩展模块并不一定在所有平台的发布版本中都有提供,有些仅仅是在特定平台的发布版本中才会列出。与基本模块不同,扩展模块的版本兼容性是模块自己指定的。这其中包含了以前 Qt Mobility 中与移动有关的模块,如蓝牙 Qt Bluetooth、传感器 Qt Sensors 等;还包含了以前 Qt 4 中的一些模块,例如 Qt DBus、Qt XML、Qt Script 等。除此之外,还新添了一些模块,例如图像特效 Qt GraphicalEffects、串口通信 Qt SerialPort 等。

最新的 Qt 5.3 中扩展模块简介如表 1-2 所列。

表 1-2 Qt 5.3 的扩展

模 块	开发平台	目标平台	简 介
Active Qt	Windows		提供对 ActiveX 和 COM 的支持
Qt Android Extras	全平台	Android	提供针对 Android 的专用 API
Qt Bluetooth	全平台	Qt for Linux/X11, BlackBerry	提供对蓝牙设备的访问
Qt Concurrent			提供一套不使用底层线程原语的高级多线程编程框架
Qt D-Bus	类 Unix		提供 D-Bus 协议之上的进程间通信
Qt Graphical Effects	全平台		提供 Qt Quick 使用的图像特效
Qt Image Formats	全平台		提供对 TIFF、MNG、TGA 和 WBMP 等图像文件格式的支持
Qt Mac Extras	全平台	Mac OS X	提供针对 Mac OS X 平台使用的专用 API
Qt NFC	全平台	BlackBerry	提供对近场通信 (Near-Field Communication, NFC) 设备的访问
Qt OpenGL			提供对 OpenGL 的支持。需要注意的是,这部分代码是从 Qt 4 移植过来的,仅为提供代码兼容。如果需要编写新的基于 OpenGL 的代码,则应该使用 Qt 5 的 QOpenGL 类
Qt Positioning	全平台		提供访问位置、卫星和地区监视设备的类
Qt Print Support	全平台		提供一组可移植的打印支持类
Qt Declarative	全平台		提供针对 Qt 4 的兼容性
Qt Script	全平台		提供一组脚本化 Qt 应用程序的类。需要注意的是,这部分类是从 Qt 4 移植过来的。针对新编写的代码,应该使用 Qt QML 模块提供的以 QJS 开头的类
Qt Script Tools	全平台		提供供 Qt Script 使用的组件。既然 Qt Script 都已经不再推荐使用,这个模块也仅仅为提供代码兼容而存在
Qt Sensors	全平台	Android, BlackBerry, Qt for iOS, WinRT, Mer	提供对传感器和动作手势识别设备的访问
Qt Serial Port	全平台	Windows, Linux, Mac OS X	提供对硬件和虚拟串口的访问
Qt SVG	全平台		提供显示 SVG 文件的类

模块	开发平台	目标平台	简介
Qt WebSockets	全平台	全平台	提供 WebSocket 通信标准,兼容 RFC 6455
Qt Windows Extras	全平台	Windows	提供针对 Windows 平台使用的专用 API
Qt X11 Extras	全平台	Linux/X11	提供针对 X11 平台使用的专用 API
Qt XML			提供 SAX 和 DOM 的 C++ 实现。注意,这些类已经是被废弃的,我们应该使用更新的 QDomStreamReader 和 QDomStreamWriter
Qt XML Patterns			提供对 XML 文件以及自定义数据模型的支持 XQuery 和 XPath 的支持

回顾一下 Qt 4。在 Qt 4 中主要包含 Qt Core 和 Qt Gui 两大核心模块和一些功能模块,其架构如图 1-2 所示,Qt 4 主要模块的简介如表 1-3 所列。

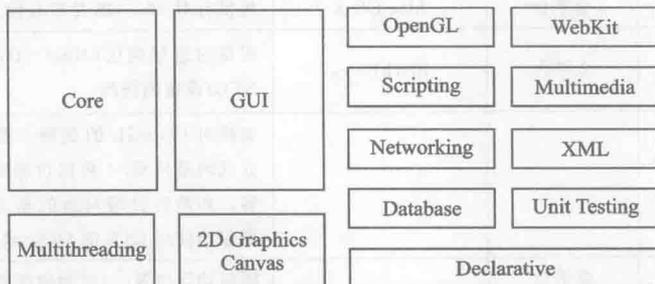


图 1-2 Qt 4 架构示意

表 1-3 Qt 4 主要模块简介

模块	简介
Qt Core	是 Qt 4 的核心类库,其他各个模块都建立在 Core 模块之上。Core 不包含任何有关 GUI 的代码
Qt GUI	提供所有与图形用户界面(Graphics User Interface,GUI)有关的组件,包括鼠标事件、键盘事件、各种图形控件等
Qt Multimedia	提供低级的多媒体支持
Qt Network	提供网络编程类库
Qt OpenGL	提供对 OpenGL 的支持
Qt Script	提供一种基于 Qt 的脚本语言的能力。这种脚本语言类似于 JavaScript 语法
Qt Script Tools	提供额外的脚本组件的支持
Qt SQL	提供对关系数据库 SQL 的支持
Qt WebKit	提供显示和编辑网页内容的支持。这是对 WebKit 引擎的一个 Qt 移植
Qt XML	提供对 XML 处理的能力

模 块	简 介
Qt Xml Patterns	提供对 XML 文件以及自定义数据模型的 XQuery 和 XPath 的支持
Qt Declarative	提供一种用于构建流畅的用户界面的声明式编程模型。该模块于 Qt 4.7 首次引入
Phonon	提供一个多媒体框架
Qt3 Support	提供对 Qt 3 的兼容类。由于 Qt 4 与 Qt 3 完全不兼容,在 Qt 4 发布时,所有 Qt 3 的类库均被改名为以 Q3 开头,并移动到一个单独的模块

Qt 5 对 Qt 4 的模块进行了重构,最明显的是对 Qt Gui 模块的修改。在 Qt 5 中,Qt Gui 不再包含有关界面的所有类。另外,相对于 Qt 4,Qt 5 将 Qt Declarative 模块分解成为与界面无关的 Qt Qml 和基于 QML 语言的一个类库 Qt Quick 2.0。正是由于 Qt Qml 和 Qt Quick 2.0 的出现,使得解释型声明式语言 QML 和 JavaScript 在 Qt 中成为与 C++ 地位平等的编程语言。Qt 5 的几乎所有上层 API 都同时提供了面向经典的 C++ 和面向 QML 的两套接口,这也成为 Qt 5 的一个很重要的特点。

1.1.2 图形界面库的架构

Qt 5 中,所有图形界面程序需要的 QApplication 以及最重要的基类 QWidget 已经不在 Qt Gui 模块,而是被重新组合到了一个新的模块 Qt Widgets 中。

就像前面提到的,Qt 5 的一个重大更改就是重新定义了 Qt Gui 模块,它不再是一个大而全的图形界面类库,而是为各种图形用户界面组件提供一般的处理,包括窗口系统集成、事件处理、OpenGL 和 OpenGL ES 的集成、2D 绘图、基本图像、字体和文本等内容。Qt 5 将以前 Qt Gui 模块中的图形部件类移动到了 Qt Widgets 模块中,将打印相关类移动到了 Qt PrintSupport 模块中。

另外,Qt 5 移除了 Qt OpenGL 模块,将 OpenGL 相关类移动到了 Qt Gui 模块中,这意味着 OpenGL 现在已经成为必选项。有的读者可能会在 Qt 扩展模块中发现 Qt OpenGL 模块,保留它只是为了兼容 Qt 4。Qt 5 图形界面库的整体架构如图 1-3 所示。

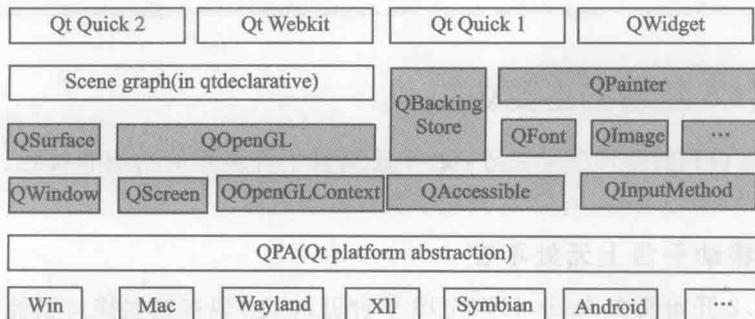


图 1-3 Qt 5 图形界面架构

在 Qt 5 支持的平台之上是平台抽象层 QPA。它曾经是 Qt 4 的一个开源项目,被称为 LightHouse(灯塔),现在被 Qt 5 吸收。在 QPA 层之上的所有深色背景组块都是 Qt Gui 模块的内容。它们被分为了两类,一类以 OpenGL 为核心,是现在最新的 Qt Quick 2 和 Qt Webkit 的基础;一类是以辅助访问和输入方式为基础的一般图形显示类,它们是经典的 QWidget 部件类和 Qt Quick 1 的基础。Qt QML 和 Qt Quick 框架将在第 3 章讲解。

1.1.3 Qt 5 架构主要特点

① 把全部的 Qt 接口迁移到 Qt 平台抽象层(Qt Platform Abstraction, QPA)之上,使 Qt 能更容易地移植到另外的系统和设备上。

QPA 的出现从根本上改变了 Qt 移植到其他环境的复杂度,由此创造一个更简洁的架构。新的平台抽象层明显简化了 Qt 的移植工作,所以 Qt 5 才能够在很短时间内移植到 QNX、Android、iOS、WinRT 等平台。

② 重新设计了图形堆栈,与 Qt 4 相比提高了性能。

Qt 5 为 Qt Quick 引入了全新的图形架构,使用了基于 OpenGL 的场景图(Scene Graph),其最低需求是 OpenGL (ES) 2.0。

新版本的 Qt Gui 包含了一组 QOpenGL 类,用来替代旧版本的 QGL 类;还引入了一个比 QApplication 更轻量级的新类 QGuiApplication 和一个处理屏幕上顶层窗口的类 QWindow。以 QWidget 为基础的组件类仍然能够像 Qt 4 一样工作,它们基于 QPainter。而 QPainter 比起以前支持更少的后端,限制使用一个光栅后端(Raster Backend)来绘制屏幕、像素与图像,一个 OpenGL 后端提供 GL 接口以及一个提供 PDF 生成与打印的后端。

③ 更加灵活的模块结构,满足桌面和移动的融合,按需添加或删除特定的模块。

模块化使 Qt 开发能够更容易、更独立地推进不同的部分,这对于 Qt 5 的稳定以及保持二进制兼容性具有很重要的意义。模块化也简化了第三方模块到 Qt 的集成。

1.2 Qt 5 的特点

1.2.1 Qt 5 新增的主要功能

除了前面讲到的架构变动之外,Qt 5 还提供了许多新的功能和特色。可以在 Qt 帮助中通过 What's New in Qt 5 关键字查看本书内容。

1. Qt 在移动平台上无处不在

从 Qt 5.2 开始便在 Android 和 iOS 平台引入 Qt,现在这些接口已经扩展到多个平台,使用一个 Qt 开发框架就可以开发出涵盖桌面、嵌入式、移动等多个平台的应用程序。

Qt 5.3 提供了对 Android、iOS、Blackberry 10 和 WinRT 等平台的全面支持,为移动市场提供了一套最佳的解决方案。将现有的桌面或者嵌入式程序移植到这些移动平台是非常简单的,只需要重新编译一次源代码。

2. 惊人的图形处理能力和性能

Qt 5 使用基于 OpenGL 的场景图来加速 Qt Quick 的图形显示,使得它可以显示出极具吸引力的用户界面,包含动画、图形效果或者粒子系统。即使在硬件资源受限的移动或者嵌入式设备上也可以流畅运行。图 1-4 是在一张图像使用 ShaderEffect 后的效果。

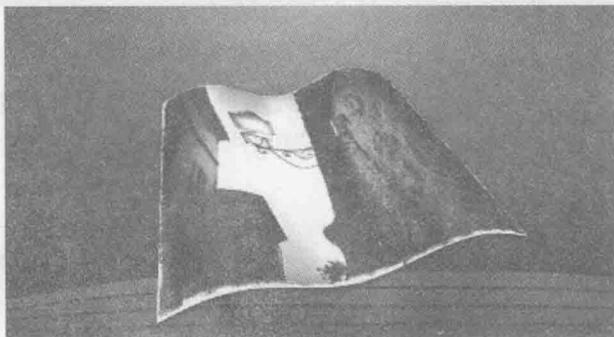


图 1-4 使用 ShaderEffect 效果图

3. Qt 5 中的 Qt Quick

Qt Quick 为开发 QML 应用提供了基础的架构。在 Qt 5 中,Qt Quick 包含了如下一些特色:

- Canvas: 在 Qt Quick 中使用 Canvas 进行绘画,提供了与 HTML5 Canvas API 相似的 API,另外还包含了一些额外的特色。
- Shader Effects: 着色器效果可以将 GLSL 着色器程序集成到 QML 代码,使项目或图像产生特效。
- Particle Effects: 粒子效果可以创建各种 2D 粒子系统,效果如图 1-5 所示。
- Sprites: 精灵可以用于 2D 图像对象的动画效果中,也可以作为粒子系统的源。
- Offline storage: 这是一个 JavaScript API,使用与 HTML5 Local Storage 类似的 API 用于在 Qt Quick 应用中存储数据。
- Window: 用来提供一个顶级窗口,其中的 Screen 类型可以用来获取屏幕分辨率等细节数据。在开发 Qt Quick 桌面应用时非常有用。
- OpenGL: Qt Quick 基于 OpenGL 渲染架构来获得最佳性能。

4. 设计用户界面变得更简单

在没有开发工具帮助的情况下,设计一个良好的 UI 是非常耗时的。Qt Quick 相对于原生的(C/C++)方式来说,已经大大降低了工作复杂度。而 Qt 5 中全新的 Qt

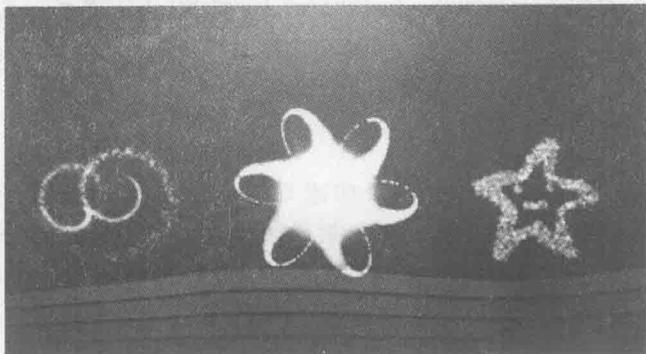


图 1-5 粒子系统效果图

Quick Controls 和 Qt Quick Layouts 使得设计用户界面变得更加简单。这些新模块提供了一些现成的 UI 控件和布局可以实现快速应用开发,其中只需要包含少量的代码,效果如图 1-6 所示。

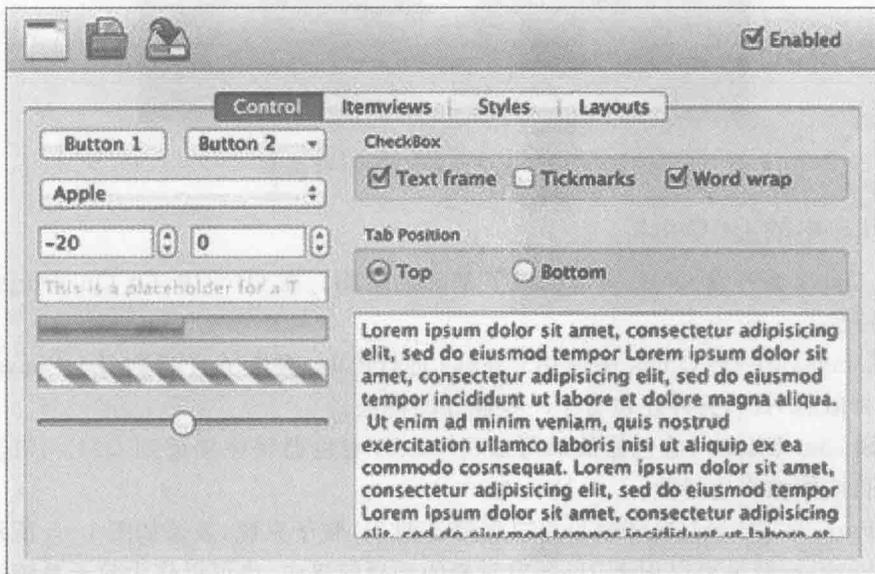


图 1-6 使用 Qt Quick 控件和布局设计的 IU 效果图

5. 传感器和位置

各种传感器和地图定位在很多便携式设备上已经很常见了。在 Qt 5 中可以使用 Sensors 和 Qt Positioning 模块在 Qt 应用中实现对传感器和定位的支持。例如基于位置的天气信息程序,效果如图 1-7 所示。

6. WebKit 和 HTML5

Qt WebKit 是一个网页内容渲染引擎,基于开源的 WebKit 项目。在 Qt Quick 和

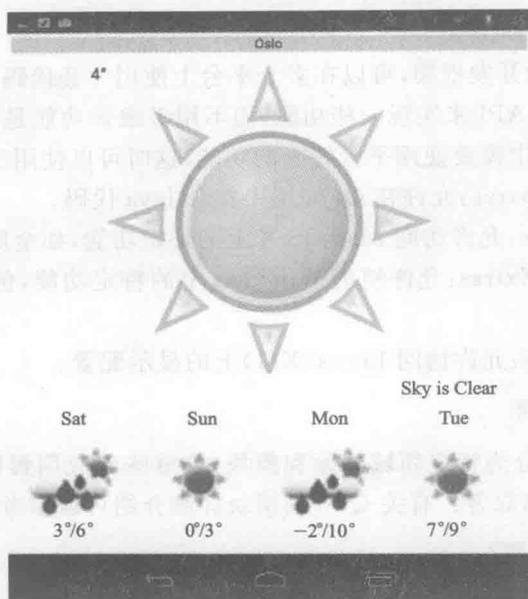


图 1-7 基于位置的天气程序运行效果图

传统的 Widget 编程中都可以使用 Qt WebKit 模块。该模块为 Qt 5 带来了最新的 HTML5 的支持,包含 CSS 滤镜和动画、视频、Canvas 和 WebGL 等。

7. 多媒体

Qt Multimedia 提供了一组丰富的 QML 类型和 C++ 类来处理多媒体内容,还提供了必要的 API 来访问摄像头和收音机功能。图 1-8 展示了嵌入到 Qt Quick 应用中的视频使用位移特效后的运行效果。

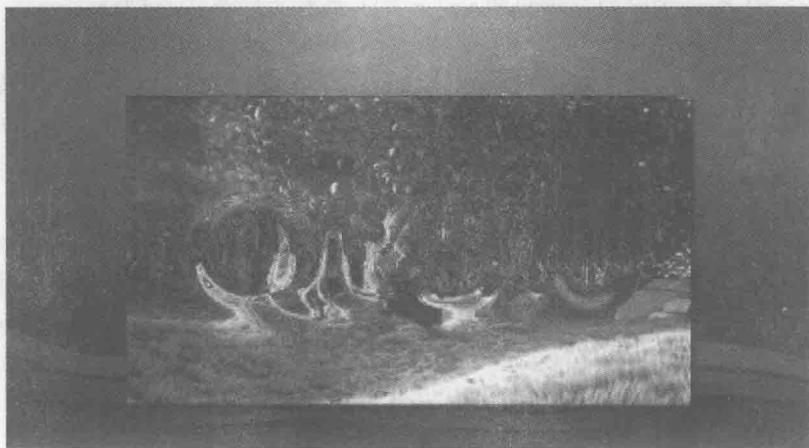


图 1-8 使用位移特效的视频效果

8. 特定平台的附加功能

Qt 是一个跨平台开发框架,可以在多个平台上使用一套代码。它为所有支持的平台提供了一套通用的 API 来实现一些功能,而不用考虑该功能是如何在一个平台上实现的。但在一些应用中需要使用平台特有的功能,这时可以使用下面的几个模块:

- Qt Android Extras: 允许在 Qt 应用中整合 Java 代码。
- Qt Mac Extras: 允许访问 Mac OS X 上的特定功能,如全屏应用等。
- Qt Windows Extras: 允许使用 Windows 上的特定功能,例如跳转列表、超级任务栏等。
- Qt X11 Extras: 允许访问 Linux(X11)上的显示配置。

9. 模块化的 Qt 库

Qt 5 将 Qt 库划分为特定领域的库和模块,这意味着应用程序可以选择自己需要的库进行编译、使用和部署。有关 Qt 5 的模块详细介绍可以参考 1.1.1 小节的内容。

10. Qt 平台抽象层

Qt 窗口系统相关的部分在 Qt 5 中全部基于 Qt 平台抽象层进行了重写。Qt 平台抽象层是一个插件架构,允许 Qt 动态加载一个窗口系统,并基于现在运行的系统进行集成。这样做有如下优点:

- 集中窗口系统集成代码库到一些类中,这些类多平台共享。
- 简化移植 Qt 到新平台所需要的工作。
- 在 Qt 中删除窗口系统依赖,使得在多个窗口系统中可以运行同一个 Qt 二进制文件成为可能。例如,Qt Linux 程序可以同时 X11 和 Wayland 上运行。

11. Qt Core 的主要更新

- 作为 Qt 最基本的信号槽机制,Qt 5 有了自诞生以来最大的改变。在 Qt 5 之前的版本中,信号槽利用宏实现,虽然语法简单,但是这种实现的缺点是没有编译期类型检查,只能在运行时发现问题。Qt 5 巧妙地利用 C++ 的函数指针,为信号槽连接提供了编译期类型检查。不仅如此,Qt 5 还减少了信号槽的限制。在 Qt 4 中,只有类的非静态成员函数才允许作为槽函数;但是在 Qt 5 中,所有函数,包括全局函数、静态函数甚至匿名函数,都可以作为槽函数。
- 包含了一个 JSON 解析器。随着 Web 2.0 的兴起,JSON 正在取代 XML 成为新一代数据交换格式。Qt 4 的年代,这种趋势尚未明显表现,直到近几年,JSON 的应用日趋增多。Qt 4 需要使用第三方库才能方便解析 JSON,而 Qt 5 则直接内置对 JSON 的支持。
- 引入了对插件形式和文件内容的 Mime 类型的识别。
- 引入了一个完整的兼容 Perl 的正则表达式引擎。
- 增加了对 C++ 11 的支持,同时保证 Qt 也能在较旧的编译器上编译运行。相比 C++ 03,C++ 11 为 C++ 语言带来了许多激动人心的新特性,比如默认

函数、常量表达式、Lambda 表达式等。Qt 5 与时俱进地增加了对 C++11 的应用。比如,在 Qt 5 中,我们可以直接将信号与一个普通的 Lambda 表达式连接起来。

12. 网络和连通性

Qt 5 对 IPv6 和双模式网络提供了更好的支持。大多数使用主机名和网络层协议的应用现在都可以添加 IPv6 的支持。这些应用可以选择同时接收 IPv4 和 IPv6 连接或者只绑定其中的一个连接类型。

Qt 5 提供了更好的方式来处理基于 TCP 套接字的连接和 SSL 证书:

- 在连接之前绑定一个 TCP 套接字到一个 IP 地址上;
- 验证 SSL 证书链;
- 访问原有系统。

许多涉及处理机密或者重要数据的应用程序往往要考虑使用严格的客户端身份验证。Qt 5 支持不透明私钥,允许应用程序从设备(例如 PKCS # 11 加密狗)中读取私钥,从而实现严格的客户端身份验证。

Qt 5 新添了 Qt Bluetooth 和 Qt NFC 两个模块,使开发者可以在应用程序中利用这些机制关联和共享信息。

13. 用户输入

Qt 4 已经在 C++ 程序中支持处理多点触控输入,而 Qt 5 将这种支持扩展到了 Qt Quick,并且涵盖了触控点的所有信息,包括触控位置、压力和速度等。

Qt 5 对鼠标事件处理进行了加强,在 Qt Quick 程序中可以对鼠标事件进行更多的控制。在 QML 文档中的鼠标区域(MouseArea)可以传递鼠标滚轮和点击事件,也可以忽略传递的事件。为了方便 Qt 开发的游戏应用,Qt 5 添加了对更多鼠标按键的支持。

1.2.2 Qt 5 与 Qt 4 的兼容性

Qt 5 的发布就是为了让这个 C++ 框架更加高效和易于使用,所以在 Qt 5 中使用了很多新的 API 取代了旧有的。Qt 努力保持各个版本中公用 API 的二进制和源代码级别的兼容性,但是为了使 Qt 成为一个更加优秀的开发框架,API 的一些改变是无法避免的。

Qt 5 作为 Qt 4 的后继,在开发 Qt 5 的时候就将目标定位成使 Qt 5 尽可能地兼容 Qt 4,让大多数应用程序通过很少的代码修改即可通过 Qt 5 重新编译。如果读者需要将 Qt 4 程序移植到 Qt 5,依然要注意 Qt 5 的一些改变。很多内容在前面已经介绍了,这里再次强调一下:

- 模块化的代码库:只需要关心自己程序中使用到的基本模块和扩展模块,可以参考 1.1.1 小节。
- 特定平台代码:Symbian 和 Meego 特定的代码已经从代码库中移除了。

- 平台定义:平台特定的代码需要使用预处理宏,Qt 5 使用 `Q_OS_*` 宏代替了以前版本中的 `Q_WS_*` 宏。
- Widgets:现在所有的部件都在一个独立的 QtWidgets 模块中,以前它们是 QtGui 模块的一部分。
- Qt Quick:旧的 Qt Quick 版本(1.0)现在成为了独立的 QtDeclarative 扩展模块的一部分,它的存在只是为了兼容目的。建议使用新的 Qt Quick(2.0)来进行开发。
- Qt3Support:这个模块从 Qt 5 中移除了。
- Qt WebKit:这个模块被分成了 Qt WebKit 和 Qt WebKit Widgets 两个模块。
- 多媒体:在 Qt 5 中多媒体通过 Qt Multimedia 模块来提供。Phonon 框架不再是 Qt 的一部分,不过它会继续由 Phonon 开发者进行维护并且支持 Qt 5。

在第 2 章将通过实际的例子详细讲述 Qt 4 程序向 Qt 5 的移植过程。

1.2.3 C++ 还是 QML

本质上,Qt 是一个 C++ 类库。在引入 QML 以前,所有的开发都是基于 C++ 的。到了 Qt 5,QML 和 Qt Quick 成为了 Qt 的核心之一,导致很多读者在犹豫是否还需要学习 C++? 其实 Qt 的本质还是 C++。

Qt 5 增加了 QML 这种应用程序开发方式。由于 QML 的简洁性,QML 和 Qt Quick 2 更适合快速开发动态界面程序。有理由相信,在未来越来越多的程序都会使用这种全新的开发模式。相比而言,QML 更能适应现代界面的要求。Windows 8 Modern UI 这样的崭新用户界面并不能简单地加入到现有的 QWidget 基础架构中。存在于智能电话和平板设备的那种富含动画效果的界面大行其道,同样也是 QWidget 这种为大量以静态图形为主要显示单元的用户界面而设计的组件所不擅长的。QML 和 Qt Quick 的出现就是为了解决这些问题。

Qt 5 的 QWidget 及其子类是开发桌面应用的核心,我们仍然能够像 Qt 4 时那样编写程序。现阶段新生的 QML 和 Qt Quick 远不如 QWidget 那样拥有丰富的开发组件,尤其缺乏复杂的企业级应用程序所必须的树等控件。这就决定了至少在现阶段,真正大型的桌面程序依旧只能选择 QWidget 为主、QML 和 Qt Quick 为辅的开发模式。

C++ 依旧是 Qt 主要的编程语言,Qt 5 也并没有忽略它。Qt 5 添加了很多新的 C++ API,而且会持续进行更新。引入 QML 和 Qt Quick,只是 Qt 5 提供的另外一种选择,并不是让它成为唯一的选择。

1.2.4 Qt 5 源代码文件的编码

C++ 标准并没有规定源代码文件的编码,但是 Qt 5 做出了自己的规定:Qt 5 的源代码文件必须使用 UTF-8 编码。

对于熟悉 Qt 4 的读者,一定知道 `QTextCodec::setCodecForCStrings` 这样的函数。这个函数最初来自 Qt 3。在 Qt 3 的时代,Unicode 尚未得到广泛使用,文本编辑

器只会使用本地编码来保存代码。那个时候,代码基本不会在不同国别、不同平台之间传递,所以这并不是一个问题。

但现在时代不同了:一份代码可能在互联网上被世界各地的开发人员编辑修改。不仅如此,代码中出现非英文字符的概率也比当时大大增加,比如欧元符号€、版权符号©等。这些代码文件如何对不同肤色的开发人员保持友好?如果说可以要求各地的开发人员必须使用规定的编码打开文件,那么又如何能避免这些特殊符号的问题?

考虑到 21 世纪前叶,文字交换的事实标准是 UTF-8。Qt 5 对此作了一个“妥协”:强制使用 UTF-8 编码源代码文件。对于 Qt 自身的代码,UTF-8 是唯一的编码。对于开发者的应用程序,允许选择不同编码,但是这其中的编码转换必须由开发者自己完成。比起编码转换的工作量,改变文件编码应该更简单一些。

1.2.5 移动平台开发

Qt 5.3 实现了对 Android、iOS、BlackBerry 10、WinRT 等平台的完整支持,使 Qt 5 成为一个支持所有主流移动平台的通用解决方案,这使得把现有的桌面或嵌入式应用移植到移动平台变得快捷、简单。Qt 真正无处不在,只使用一个框架,就可以覆盖传统桌面、嵌入式系统和移动平台。

1.3 小结

本章对 Qt 5 进行了全面的介绍。可以看到,Qt 5 是一个全新的版本,它在整体架构上进行了大量重构,特别是将 QML 和 Qt Quick 放到了更加核心的地位。

Qt 5 的改动和新增内容,在实际开发中需要如何应用,如何将现有的 Qt 4 程序升级到 Qt 5? 这些问题将在下一章通过实例详细讲解。

第2章

将 Qt 4 代码迁移到 Qt 5

自 Qt 5 发布,许多开发者都会考虑将早期 Qt 4 版本的代码迁移到 Qt 5。其实,在 Qt 5 的开发过程中,Qt 工程师已经注意将 Qt 5 与 Qt 4 保持源代码级别的兼容性。Qt 5 的类库接口没有做大的修改,更多的更新体现在内部架构方面。所以,比起 Qt 3 到 Qt 4 的激进式升级,将 Qt 4 的代码迁移到 Qt 5 还是相对简单的。本章将通过实例讲解 Qt 4 代码向 Qt 5 迁移时应该注意的一些问题,还会介绍 Qt 5 引入的全新信号槽语法、对 C++11 标准的支持等高级特性。

本章只讲解 Qt 5 与 Qt 4 的差异内容,但没有涉及 Qt Quick 程序的移植,该部分内容请参考第 4 章。本章适合于拥有一定的 Qt 4 编程基础的读者,对于 Qt 初学者,建议先学习《Qt Creator 快速入门(第 2 版)》一书。

2.1 Qt 5 版本的 HelloWorld

HelloWorld 是包含了一个程序的基本要素的最简单应用,需要输出一个“Hello World”字符串。本节使用 Qt 5 实现一个这样的程序,对比讲解 Qt 5 与 Qt 4 的基本变化。本书中的所有程序以 Windows 7 作为开发平台,使用其他操作系统的读者可以借此参考学习。

2.1.1 Qt 5 的下载与安装

为避免由于开发环境的版本差异产生不必要的问题,推荐在学习本书时下载与本书相同的开发包版本。这里使用了 Qt 5.3.0 开源版本,其中包含了 Qt Creator 3.1.1。关于 Qt 的版本授权内容,可以参考本书的前言部分。

Qt 5.3.0 的下载地址为:http://download.qt-project.org/official_releases/qt/5.3/5.3.0/。为了可以进行 Android 开发,这里选择下载文件:qt-opensource-windows-x86-android-5.3.0.exe

对于下载页面不同版本的区分,有兴趣的读者可以参考本书的附录。下载完成后,直接按照默认设置安装即可。最后打开 Qt Creator 3.1.1 的欢迎界面如图 2-1 所示。

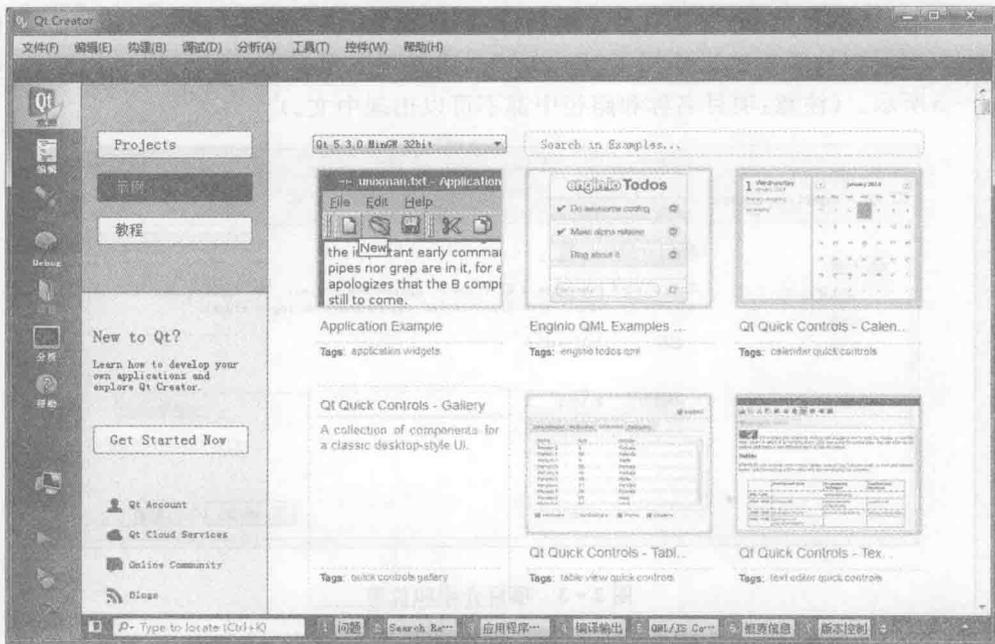


图 2-1 Qt Creator 欢迎界面

2.1.2 创建 Hello World 程序

Qt 5 安装完成后默认进行了环境设置和关联,读者无需做任何设置就可以运行一个程序。首先运行 Qt Creator,然后通过下面的步骤创建 Qt Widgets 应用。(项目源代码路径:src\02\2-1\helloworld)

选择“文件→新建文件或项目”菜单项,在选择模板页面选择应用程序中的 Qt Widgets Application 项,然后单击“Choose”按钮,如图 2-2 所示。这里将 Qt 4 中“Qt Gui 应用”改为“Qt Widgets Application”,后面再介绍这一改变的原因。

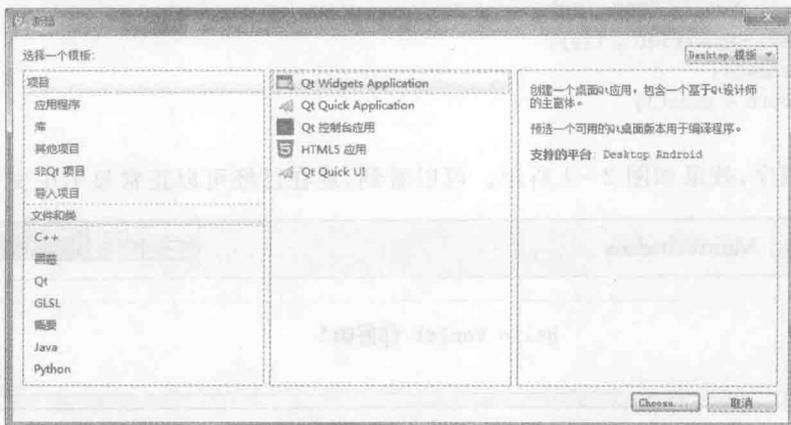


图 2-2 选择模板

然后输入项目信息。在“项目介绍和位置”页面输入项目的名称为 helloworld, 设置项目创建路径, 笔者这里使用了“F:\2-1”。读者可以根据自己的情况进行设置, 如图 2-3 所示。(注意: 项目名称和路径中都不可以出现中文。)



图 2-3 项目介绍和位置

后面的步骤与 Qt 4 中完全相同, 可以直接使用默认设置, 这里就不再赘述。项目创建完成后, 打开 main.cpp 文件, 这里要在界面上添加一个可以显示字符串的标签部件, 修改代码如下:

```
#include "mainwindow.h"
#include <QApplication>
#include <QLabel>
int main(int argc, char * argv[])
{
    QApplication a(argc, argv);
    MainWindow w;
    QLabel * label = new QLabel(&w);
    label->setText("Hello World! 你好 Qt!");
    label->resize(200, 20);
    label->move(120, 120);
    w.show();
    return a.exec();
}
```

运行程序, 效果如图 2-4 所示。可以看到, 现在已经可以正常显示中文了。

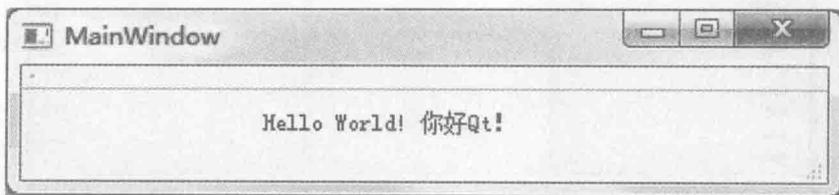


图 2-4 Hello World 运行效果

2.1.3 Qt 4 程序迁移到 Qt 5 的注意事项

通过查看 Qt Creator 自动生成的 Hello World 程序的各个文件,可以发现 Qt 5 程序最基本的几个不同之处。下面分别进行讲解。

1. .pro 项目文件增加新的语句

```
greaterThan(QT_MAJOR_VERSION, 4): QT += widgets
```

这条语句的含义是,如果 QT_MAJOR_VERSION 大于 4(也就是当前开发使用的是 Qt 5 及更高版本),则需要增加 widgets 模块。

在 Qt 4 中,Qt 提供的全部图形界面相关类都包含在 QtGui 模块。第 1 章已经提到,Qt 5 将一些图形界面类移动到了 QtWidgets 模块。所以在具有图形用户界面的应用程序中,需要增加这个模块。这也就是前面创建项目时显示 Qt Widgets Application 的原因,现在创建的程序所使用的部件都在全新的 QtWidgets 模块中,而不是 Qt 4 时的 Gui 模块。所以 Qt Creator 的项目类型名称自然由以前的“Gui 应用”修改为现在的“Widgets Application”了。

如果项目仅需要支持 Qt 5,也可以直接添加“QT += widgets”一句。不过为了保持代码兼容,最好还是按照 Qt Creator 生成的语句编写。

2. 修正代码中的 include 语句

Qt 5 添加了一些新的头文件,并且将以前 QtGui 模块的某些头文件改成 QtWidgets 模块。这个改变还是由于 Qt 5 中将图形部件从 QtGui 模块剥离开来,放到了全新的 QtWidgets 模块中的缘故。例如,Qt 4 中使用的

```
# include<QtGui/QApplication>
```

语句,需要更改为:

```
# include<QtWidgets/QApplication>
```

不过,还有另外一种更好的方式,它在 Qt 4 和 Qt 5 中都适用。那就是直接使用类名:

```
# include<QApplication>
```

除此之外,在使用一些类时发生了编译错误,则可能需要添加更多的头文件。例如在 Qt 4 中,可以直接使用下面的语句:

```
QDrag * drag = new QDrag(event->widget());
```

但是在 Qt 5 中,这个语句将会出错。因为 QDrag 位于 QtGui 模块,而语句中的 QDrag 使用了 QWidget 类。在 Qt 4 中,QWidget 同样位于 QtGui 模块,但是在 Qt 5 中,QWidget 被移动到 QtWidgets 模块。因此为了让这条语句通过编译,需要额外添加 QWidget 头文件:

```
# include<QWidget>
```

3. Qt 5 关于中文显示的问题

Qt 5 移除了 Qt 4 中极具争议的 `QTextCodec::setCodecForTr` 和 `QTextCodec::setCodecForCStrings` 等函数,也就是说,在 Qt 5 中,不能继续使用下面这种通过设置字符编码的方法来显示中文:

```
QTextCodec::setCodecForTr(QTextCodec::codecForName("GB2312"));
QLabel *label = new QLabel(this);
label->setText(tr("你好 Qt!"));
```

Qt 5 要求源代码文件统一使用 UTF-8 编码。因此,如果需要简单地显示中文,只需像前面演示程序中那样编码:

```
label->setText("Hello World! 你好 Qt!");
```

即可。但是考虑到开发后期国际化的工作,最好为需要显示到界面上的字符串添加 `tr` 函数,即

```
label->setText(tr("Hello World! 你好 Qt!"));
```

但是这里还要提醒大家,尽量不要直接在程序中使用中文字符串,并且 `tr` 函数中也不应该出现除英文之外的其他语言字符。最佳实践是,只使用英文设置程序中所有直接面向用户的字符串,并且将这些字符串全部包含在 `tr` 函数中。当需要显示成中文时,使用国际化机制完成。本书的例子大部分只是用于演示的示例程序,为讲解方便,有些文件直接在源代码中使用了中文,并不值得推荐,特此向读者说明。

除了上面提到的每个程序都可能涉及的 3 点,还有其他一些需要注意的方面。如果自己的程序中涉及以下问题,需要根据具体情况进行修改:

① Qt 5 移除了 Qt3 Support 模块,不过大多数使用 Qt3 Support 的代码都可以使用相应的 Qt 4 代码来代替,进而迁移到 Qt 5。

② 增删了另外许多模块。例如将所有打印类移动到新的 Qt Print Support 模块中;移除 Phonon 模块,使用 Qt Multimedia 模块处理多媒体;将 WebKit1 及其相关界面类移动到新的 Qt WebKit Widgets 模块中等。

③ 修改了平台相关的定义。在 Qt 5 中,所有的 `Q_WS_*` 宏统一成 `Q_OS_*` 宏。由于移除了 `Q_WS_*` 宏,那些使用了这些宏的代码都不会通过编译。

④ 移除了 QWS,Qt 5 将以前使用的 QWS 相关 API 移植到新的 QPA 架构。

以上只是列出了主要的修改,如果要了解 Qt 5 中 C++ API 更加详细的修改内容,可以参考本书的附录。

如果在 Qt 4 中使用了 QML,那么在 Qt 5 中依然可用,只不过这部分代码现在 Qt Quick1 模块中。但是,这只是为了向前兼容,Qt 的后续版本也不会对该模块进行改进。建议将这些代码直接迁移到 Qt Quick 2。由于使用的是 scene-graph 机制,而不是 Qt 4 中基于的 QPainter,因此 Qt Quick 2 在性能上有很大的提升。对于 Qt Quick 程序的迁移会在第 4 章详细讲解。

2.2 Qt 4 程序迁移实例

这一节将使用 Qt Creator 打开一个在 Windows XP 平台开发的、基于 Qt 4.7 创建的 Qt Gui 应用程序,然后将其修改为一个 Qt 5 程序。这一节的目的是为读者演示如何使用 Qt 5 重新编译 Qt 4 的程序。由于 Qt 版本的原因,一些基于 Qt 4.8 及以后版本的程序,可能不会出现下面列出的所有问题。读者可以对本节内容进行选读,如果遇到了类似的情况,可以借鉴解决;如果没有遇到,则可以直接跳过相关内容。

2.2.1 修改编码

(项目源码路径:src\02\2-2\helloworld)下载并使用 Qt Creator 打开本书提供的 Qt 4 版本 helloworld 程序,则自动跳转到项目配置页面,如图 2-5 所示。这里使用默认的构建套件即可,单击 Configure Project 按钮配置项目。

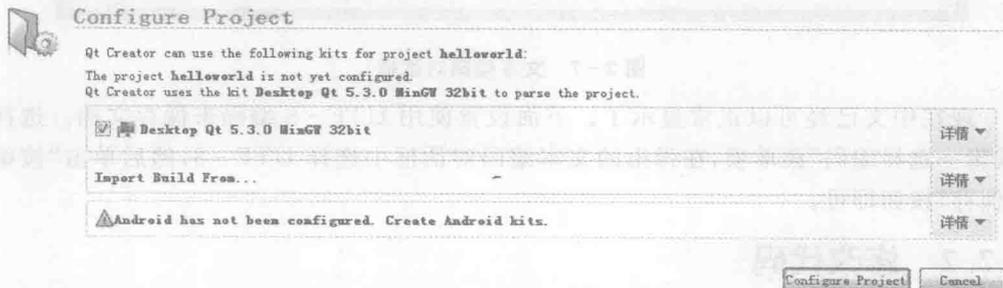


图 2-5 配置项目

下面打开 main.cpp 文件,则显示“错误:无法用‘UTF-8’-编码解码‘main.cpp’。无法编辑”的错误提示,如图 2-6 所示。这是由于程序文档中包含了中文,却没有用 UTF-8 编码存储造成的。



图 2-6 编码错误提示

为了以后再次打开该项目时不再出现这个错误提示,这里需要使用 UTF-8 编码重新保存该文件。单击“选择编码”按钮,在弹出的文本编码对话框中选择 GB-18030/gb18,030/ibm-1,392/windows-54936 项,然后单击“按编码重新载入”按钮。如图 2-7 所示。

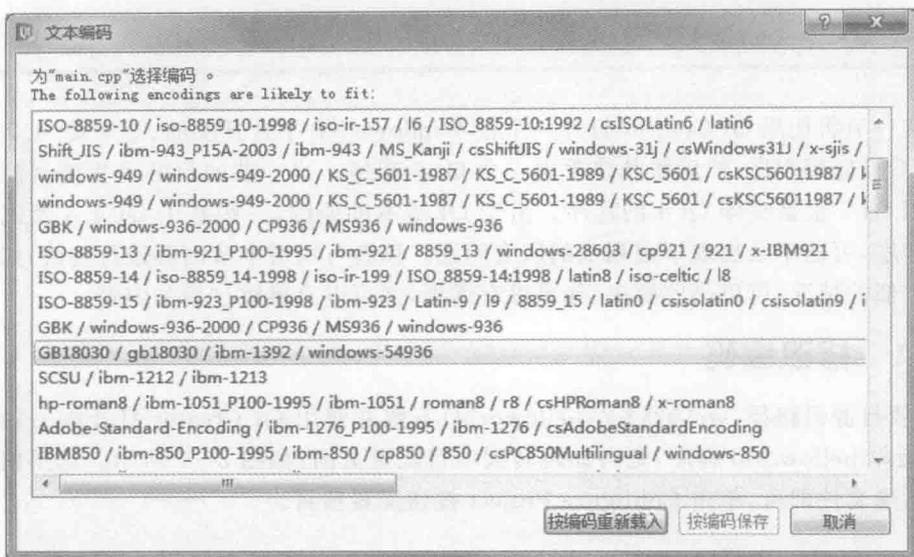


图 2-7 文本编码对话框

现在中文已经可以正常显示了。下面设置使用 UTF-8 编码来保存文档。选择“编辑→选择编码”菜单项，在弹出的文本编码对话框中选择 UTF-8，然后单击“按编码保存”按钮即可。

2.2.2 修改代码

在打开的 main.cpp 中可以看到 `#include <QtGui/QApplication>` 下面带有波浪线。将鼠标指针放到该处，可以看到“QtGui/QApplication: 没有文件或者目录”的提示，如图 2-8 所示。这就是因为 QApplication 已经不在 QtGui 模块中了，现在将该行代码改为 `#include <QApplication>`。

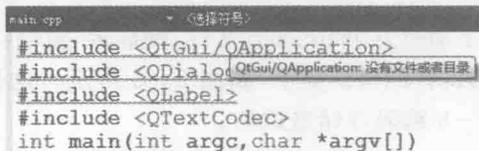


图 2-8 包含文件错误提示

这时 `#include <QApplication>` 还是有问题，依然显示“QApplication: 没有文件或者目录”。因为 QApplication 现在 QtWidgets 模块中，所以还需要修改.pro 文件。打开 helloworld.pro 文件，添加下面一行代码：

```
greaterThan(QT_MAJOR_VERSION, 4): QT += widgets
```

然后按下 Ctrl+S 保存文件。这时回到 main.cpp 页面，可以看到已经没有错误提示了。现在按下 Ctrl+B 编译程序，则会出现“setCodecForTr 不是 QTextCodec 成员”的编译错误，如图 2-9 所示。正如前面所说，Qt 5 中的 QTextCodec 类中已经移除 setCodecForTr 等函数了。

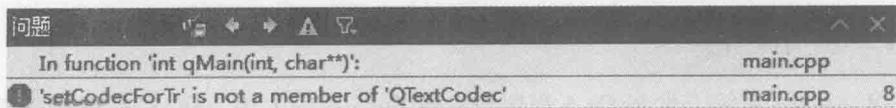


图 2-9 编译错误提示

下面去掉 QTextCodec 的相关代码,修改 main.cpp 文档如下:

```
#include<QApplication>
#include<QDialog>
#include<QLabel>
int main(int argc,char * argv[])
{
    QApplication a(argc,argv);
    QDialog w;
    w.resize(400 ,300);
    QLabel label(&w);
    label.move(120 ,120);
    label.setText(QObject::tr("Hello World! 你好 Qt!"));
    w.show();
    return a.exec();
}
```

这时按下 Ctrl+R 编译运行程序,已经可以正常运行了。

2.2.3 设置应用程序图标

在 Qt 4 的时代,为 Windows 平台的 Qt 应用程序添加图标,需要创建一个.rc 文件,然后在里面输入一行代码,还要修改.pro 文件。在 Qt 5 中,这个需求变得非常简单:只需要将.ico 图标文件放到源码目录,然后在.pro 项目文件中添加一行代码:

```
RC_ICONS = myico.ico
```

即可,这里 myico.ico 就是图标文件的名字。(项目源码路径:src\02\2-3\helloworld。)读者可以在前面的程序上进行修改,然后运行程序,效果如图 2-10 所示。

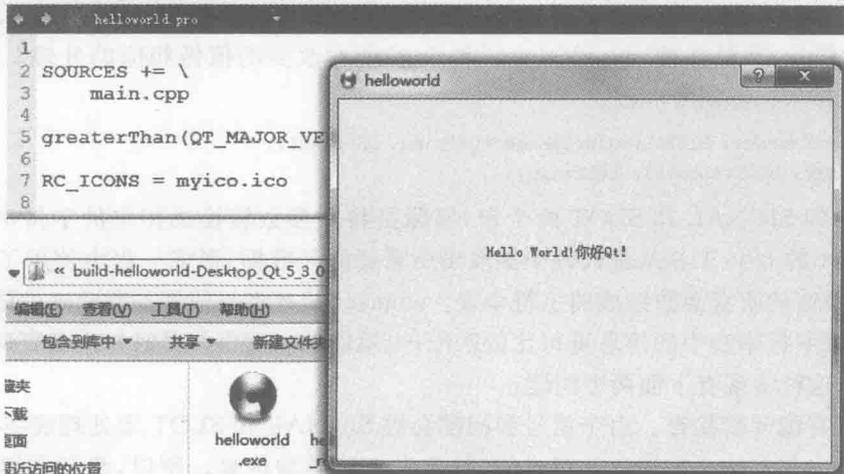


图 2-10 设置应用程序图标运行效果

2.2.4 发布程序

在 Windows 系统上发布 Qt 5 程序与发布 Qt 4 程序方法一样,只是现在需要一起发布的动态链接库文件更多。下面通过发布前面的 helloworld 程序为例进行讲解。

要发布程序,首先需要在编译时选择 Release 版本,完成后将编译生成目录中的 exe 可执行文件(笔者这里的路径是 F:\2-2\build-helloworld-Desktop_Qt_5_3_0_MinGW_32 bit-Release\release\helloworld.exe)复制到一个空白文件夹中,然后双击该 helloworld.exe 文件,根据提示到 Qt 安装目录下相应的 bin 目录(笔者这里路径是 C:\Qt\Qt5.3.0\5.3\mingw482_32\bin)中将缺少的.dll 文件复制过来。现在已经可以在本机上运行该程序了,但是在其他没有安装该版本 Qt 的机子上还是无法运行。此时,还需要将 Qt 安装目录中的 platforms 目录(笔者这里的路径是 C:\Qt\Qt5.3.0\5.3\mingw482_32\plugins\platforms)复制过来,在其中只需要保留 qminimal.dll 和 qwindows.dll 两个文件即可。

另外,如果程序中使用了其他模块,还需要将目录中对应的动态链接库文件复制过来。

2.3 新的信号槽语法

在 Qt 5 中对 Qt 的信号和槽机制进行了很大的优化,使其更加易于使用。本节只讲解信号槽在 Qt 5 中的新变化,并没有全面介绍 Qt 的信号和槽机制。如果读者对 Qt 的信号槽机制不是很了解,可以参考《Qt Creator 快速入门(第 2 版)》第 7 章的相关内容。

2.3.1 新旧语法对比

这里举一个例子来进行介绍。某个类在值改变时发送 valueChanged(QString, QString)信号,需要在槽 showValue(QString)中对改变的值做相应的处理。在 Qt 4 中一般这样来关联信号和槽:

```
connect(sender, SIGNAL(valueChanged(QString, QString)),  
receiver, SLOT(showValue(QString)));
```

Qt 4 的 SIGNAL 和 SLOT 两个宏,实际是将其参数转换成相应的字符串。在编译之前,Qt 的 moc 工具从源代码中提取出所需要的元数据,形成一张由使用了 signals 和 slots 修饰的所有函数组成的字符串表。connect 函数将与信号关联起来的槽的字符串,同这张字符串表中的信息进行比较匹配,也就能够在发出信号时知道需要调用哪个槽函数。这种实现有下面两个问题:

- 没有编译期检查。由于信号和槽都会被 SIGNAL 和 SLOT 宏处理成字符串,字符串的对比是在运行时完成的,并且失去了类型信息。所以,我们在编写 Qt 4 程序时,有时会出现编译通过但是运行时原本应该调用的槽函数却没有执行。

此时,编译器不能给出任何错误信息,只能在运行时看有没有警告。

- 无法使用相容类型的参数。由于 connect 函数使用的是字符串对比,所以槽函数参数类型的名字必须和信号的完全一致,也必须与头文件中的类型一致。这里的“一致”是严格的字符串意义上的相同,因此,那些使用了 typedef 或者 namespace 的类型,即便实际类型是相同的,依然可能由于字符串名字不一样而不能正常工作。

为了解决这两个问题,Qt 5 提供了一套全新的信号槽语法。前面在 Qt 4 中的关联可以使用下面的方式代替:

```
connect(sender, &Sender::valueChanged,receiver, &Receiver::showValue);
```

其中,Sender 是发出信号的 sender 对象的类型,Receiver 是接收信号的 receiver 对象的类型。需要说明,Qt 4 中的关联方式在 Qt 5 程序中依然可用,不过新的语法有下面几个优点:

- 支持编译期检查。Qt 5 新的关联语法可以在编译时进行检查,信号或槽的拼写错误、槽函数参数数目多于信号的参数数目等错误在编译时就能够被发现。
- 支持相容参数类型的自动转换。使用新的语法不仅支持使用 typedef 或者命名空间,还支持使用隐式类型转换。例如,当我们的信号参数类型是 QString,而槽函数对应的参数类型是 QVariant,那么,在进行信号槽的连接时,QString 将被自动转换成 QVariant。这是因为 QVariant 有一个可以使用 QString 的隐式构造函数。
- 允许连接到任意函数:在 Qt 4 中,槽函数只能是使用 slots 关键字修饰的成员函数,而新的语法通过函数指针直接调用函数,任意成员函数、静态函数或者 C++11 Lambda 表达式都可以作为槽进行关联。(以前的信号槽语法是不受 private 限制的。槽函数虽然可以被声明为 private,但仅在作为普通函数调用时起作用,作为槽连接时,SLOT 无视 private 修饰,因为仅作为字符串连接。而新语法无法取私有函数指针,在编译时就会有警报,因而更安全。)

2.3.2 新的语法示例

这里先看一下在 Qt4 中典型的信号槽的语法。(项目源码路径:src\02\2-4\signalslot。)新建 Qt Widgets Application,名称为 signalslot,其他保持默认设置。完成后,打开 mainwindow.ui 文件进入设计模式,在界面上拖入一个 Label 标签部件,然后在属性栏中修改其 objectName 为 stringLabel。下面回到编辑模式,在 mainwindow.h 中添加一个槽函数声明:

```
private slots:
void showString(QString str);
```

然后在 mainwindow.cpp 中添加该函数的定义:

```
void MainWindow::showString(QString str)
{
```

```

    ui->stringLabel->setText(str);
}

```

这里在 MainWindow 类中添加了一个 showString 槽函数,用来在界面上显示字符串。

下面向项目中添加一个新的 Qt 设计师界面类。模板选择 Dialog without Buttons,类名设置为 StringDialog。然后打开 stringdialog.ui 进入设计模式,向界面拖入一个 Line Edit 和一个 Push Button 部件;将 Line Edit 部件的 objectName 修改为 stringLineEdit,将 Push Button 部件的 objectName 修改为 emitButton。然后将 Push Button 的显示文本修改为“发送”。

现在右击“发送”按钮,选择“转到槽”菜单项,选择 clicked 信号。单击“确定”按钮跳转到 stringdialog.cpp 文件。此时打开 stringdialog.h 文件,添加一个信号的定义:

```

signals:
    void stringChanged(QString str);

```

再次打开 stringdialog.cpp 文件,将前面生成的“发送”按钮单击信号对应的槽函数修改如下:

```

void StringDialog::on_emitButton_clicked()
{
    QString str = ui->stringLineEdit->text();
    emit stringChanged(str);
}

```

这里获取了界面上行编辑器中的文本,并使用 stringChanged 信号将文本发送出去。下面在 MainWindow 类中创建 StringDialog 类对象,并关联自定义的信号和槽。打开 mainwindow.cpp 文件,首先添加头文件:

```
#include "stringdialog.h"
```

然后在构造函数中添加如下代码:

```

StringDialog * dlg = new StringDialog(this);
connect(dlg, SIGNAL(stringChanged(QString)),
        this, SLOT(showString(QString)));
dlg->show();

```

最后运行程序,在弹出的对话框中输入字符串,然后单击“发送”按钮,则会在主界面上显示输入的字符串,效果如图 2-11 所示。这是个典型的 Qt 信号槽应用:在对话框界



图 2-11 设计信号和槽

面上输入信息,按下按钮,将字符串进行处理,然后通过信号将字符串发送出去;在主界面上,通过将对话框上的信号和自己的槽函数进行关联,当对话框发射信号后,就会执行相应的槽函数,从而获取对话框界面上输入的字符串。这样便实现了界面间的数据通信。

下面使用 Qt 5 的新语法来修改这个例子。首先将 mainwin.cpp 中 connect 语句更改如下:

```
connect(dlg, &StringDialog::stringChanged, this, &MainWindow::showString);
```

使用这种语法不再需要指明信号和槽的参数。实际上,槽函数的参数类型只要可以和信号参数类型进行隐式类型转换即可。下面打开 mainwindow.h 文件,先添加头文件:

```
#include <QVariant>
```

然后将槽函数修改为:

```
private:
```

```
void showString(QVariant str);
```

这里不再使用 slots 关键字,showString() 函数作为普通函数进行声明,并且将其参数类型修改为了 QVariant。

下面打开 mainwindow.cpp 文件,修改 showString 函数的定义代码:

```
void MainWindow::showString(QVariant str)
```

```
{
```

```
    ui->stringLabel->setText(str.toString());
```

```
}
```

现在的程序可以正常运行。这表明,在 Qt 5 中信号已经可以关联到普通的函数,而不再需要使用 slots 指明槽函数,并且对参数的要求也没有那么严格了。

除了可以将普通成员函数作为槽函数,Qt 5 新语法还允许使用任意类的静态函数。例如,将 connect 语句更改如下:

```
connect(dlg, &StringDialog::stringChanged, &QApplication::quit);
```

这里我们希望在对话框的字符串发生改变时,调用 QApplication 的静态函数 quit 实现程序的退出。因为 Qt 4 的信号槽机制只能处理成员函数,所以如果要将 QApplication::quit 函数作为槽函数,必须使用一个成员函数封装这个静态函数。而在 Qt 5 中,我们使用新的语法可以将信号直接关联到静态函数。(这里接收信号的 receiver 对象是 this,允许进行省略。)

当信号有重载的情况时,使用 Qt 5 的新语法可能会有一些不方便。例如,QSpinBox 有两个重载的信号:

```
■ void valueChanged(int i)
```

```
■ void valueChanged(const QString & text)
```

当我们使用下面的语句进行连接时,

```
connect(spinBox, &QSpinBox::valueChanged,
        this, &MainWindow::onSpinBoxValueChanged);
```

编译器会发出一个错误。因为信号 valueChanged 有重载,所以使用

&QSpinBox::valueChanged 语句获取信号的指针时会有歧义:有两个相同名字的信号。为解决这个问题,一方面可以选择使用 Qt 4 类型的旧的信号槽连接语法:

```
connect(spinBox, SIGNAL(valueChanged(int)),
        this,     SLOT(onSpinBoxValueChanged(int)));
```

由于 SIGNAL 和 SLOT 两个宏都要求指明参数类型,所以不会出现歧义。但是,这样做又失去了编译期检查的优点。为了继续使用 Qt 5 的新语法,需要增加一个显式类型转换:

```
QObject::connect(spinBox,
                 static_cast<void(QSpinBox::*)(int)>(&QSpinBox::valueChanged),
                 this,
                 &MainWindow::onSpinBoxValueChanged);
```

另外,有些槽函数的参数具有默认值,比如 QPushButton 的一个槽函数:

```
void animateClick(int msec = 100)
```

如果我们有一个自定义组件 MyWidget,这个组件有一个不带参数的信号 aSignal()想要连接到 QPushButton 的这个槽函数,当试图使用新语法这样写时:

```
connect(myWidget, &MyWidget::aSignal, pushButton, &QPushButton::animateClick);
```

编译器通常会给出一个错误(错误信息可能会因为编译器版本不同而各异):

```
The slot requires more arguments than the signal provides.
```

大致是说,槽函数所要求的参数个数比信号提供的要多。这在 Qt 中一般是不允许的。但是这里的情形又有所不同:因为这个函数的参数具有默认值。但是,函数参数的默认值只在直接调用函数时才有效。在取函数地址的时候,编译器是看不到参数默认值的。所以我们取到的函数指针并不包含这个参数默认值。另一方面,由于这个槽函数的参数具有默认值,所以这个槽函数并不真正需要显式地提供一个参数。这种情况在 Qt 4 中就需要使用合适的槽函数进行封装。在 Qt 5 中,也可以这么做。但是,Qt 5 的信号槽新语法还支持使用 C++ 11 Lambda 表达式。因此,还可以选择利用 Lambda 表达式达到这一目的。下一节将详细说明这一点。

2.4 对 C++11 的支持

C++11 是 C++ 语言的最新版本。Qt 5 全面支持 C++11 标准。需要注意的是,有些编译器(比如 gcc 或者 clang)尽管支持 C++11,但是默认并不开启;另外一些编译器(比如 VC2010)则默认开启。针对这种情况,可以在 Qt 项目文件 .pro 中增加“CONFIG += c++11”一行代码。

2.4.1 Lambda 表达式

上一节讲述了 Qt 5 全新的信号槽语法,可以看到 Qt 5 提高了信号槽关联的灵活性,允许使用任意函数作为槽函数。但是如果更好地执行异步代码,连函数都不想定

义,能否在 connect 关联时直接指定信号发出时执行的代码呢?通过 Lambda 表达式可以达到这个目的。下面首先来看一个例子,然后再进行语法的讲解。

(项目源码路径:src\02\2-5\signalslot)依然在前面程序的基础上进行修改,首先打开 signalslot.pro 项目文件,在其中添加一行代码“CONFIG += c++11”,然后按下 Ctrl+S 保存该文件。下面打开 mainwindow.cpp 文件,修改其中的 connect 语句如下:

```
connect(dlg, &StringDialog::stringChanged, [=] (QString str) {
    if (str == "qt") {
        ui->stringLabel->setText("hello qt!");
    } else {
        ui->stringLabel->setText("error!");
    }
});
```

这里直接在 connect 语句中添加了信号发射后要执行的代码,使用参数值区别显示不同的字符串。这种写法就是所谓的 Lambda 表达式。Lambda 表达式是 C++11 新增加的特性。简单来说,Lambda 表达式就是匿名函数。在 C++ 中,它以一对方括号开始。这对方括号被称为 Lambda 表达式引入符。引入符后面可以添加 Lambda 表达式的返回值类型,之后是参数列表,最后是 Lambda 表达式的函数体。

不同的 Lambda 表达式引入符有不同的含义(表 2-1)。引入符描述函数体如何“获得”外部变量。这一过程被称为“捕获”。所谓“外部变量”,指的是函数体以外的变量。这些变量需要在引入符可见的作用域有定义。

表 2-1 Lambda 表达式引入符

引入符	说明
[]	不捕获任何外部变量
[=]	以传值的方式捕获所有外部变量
[&]	以传引用的方式捕获所有外部变量
[x, &y]	x 以传值的方式捕获,y 以传引用的方式捕获
[=, &x]	x 以传引用的方式捕获,其余外部变量则以传值的方式捕获
[&, x]	x 以传值的方式捕获,其余外部变量则以传引用的方式捕获

例如,一个 Lambda 表达式需要两个参数,以传值方式捕获的 a 和以传引用方式捕获的 str,其返回值是 QString 类型,那么这个 Lambda 表达式应该写成:

```
[a, &str] ->QString {}
```

如果需要了解有关 Lambda 表达式的更多信息,请参考 C++ Primer(第 5 版)或其他相关 C++11 教程。

前面提到,对于带有默认参数的槽函数,可以使用 Lambda 表达式作为一个匿名函数,达到函数封装的目的。依旧以上文提到的函数为例,可以使用下面的代码重写:

```
connect(myWidget, &MyWidget::aSignal, [] () { pushButton->animateClick() });
```

2.4.2 适用于 C++11 的宏

除了前面提到的 Lambda 表达式,Qt 5 还支持使用很多 C++11 的新特性,比如 `constexpr` 关键字等。但是,考虑到目前还有编译器不支持 C++11,所以 Qt 5 通过一系列宏兼容 C++11。利用这种技术,只需要使用 Qt 5 编写一段代码,就可以同时供支持和不支持 C++11 的两种编译器进行编译。

■ `constexpr`

C++11 增加的新关键字 `constexpr`,用于向编译器指出,某些内联函数可以在编译期运算。为此,Qt 5 引入了 `Q_DECL_CONSTEXPR` 宏。如果编译器支持 C++11 的 `constexpr` 关键字,这个宏会展开为 `constexpr`,否则是空白。

在使用时,我们可以将 `Q_DECL_CONSTEXPR` 宏放在语句的开始处,例如:

```
Q_DECL_CONSTEXPR QFlags<Val> operator|(Val1, Val2) { ... }
```

■ `static_assert`

C++11 增加了静态断言,允许我们在编译期检测一些条件是否成立。静态断言在调试模板函数的模板参数时非常有用。模板函数的模板参数在编译时实例化,我们可以使用静态断言判断实例化的模板参数是否符合预期。另外,静态断言没有普通断言的运行时开销,也不会像普通断言那样,一旦发生就会终止程序的执行(这在某些类型的程序,比如服务器程序,是相当致命的),因而更方便开发人员进行程序调试。

为支持 C++11 的静态断言,Qt 5 引入 `Q_STATIC_ASSERT` 和 `Q_STATIC_ASSERT_X` 两个宏。当编译器支持静态断言时,这两个宏将会展开为 `static_assert`;否则将使用模板技巧实现。

■ `override`

有时候,程序的错误仅仅是因为少打了一个字母。当需要在子类覆盖父类的函数时,这种错误经常发生。为了解决这一问题,C++11 引入了 `override` 关键字。当一个函数被 `override` 修饰时,这个函数必须覆盖了父类的函数,否则编译器会发出错误。

Qt 5 通过引入 `Q_DECL_OVERRIDE` 宏,支持 `override` 关键字。当编译器支持 `override` 特性时,这个宏被展开为 `override`;否则为空。这个宏用于函数声明的最后,例如:

```
void MyWidget::mouseMoveEvent(QMouseEvent * event) Q_DECL_OVERRIDE
```

■ `final`

C++ 规定,父类的函数被声明为 `virtual` 之后,所有子类的相同函数都会自动成为 `virtual` 的。不过有时候我们希望打断这种链式机制。比如,我们为接口提供了一个默认的抽象类实现,但是不希望该抽象类的子类覆盖这个实现。为此,C++11 增加了 `final` 关键字。`final` 关键字用于“打断”这种 `virtual` 的“传播”。如果一个虚函数被 `final` 修饰,那么这个函数在这个类的所有子类中都不允许被覆盖。

Qt 5 通过引入 `Q_DECL_FINAL` 宏支持 `final` 关键字。当编译器支持 `final` 时,这个宏被展开为 `final`;否则为空。同 `Q_DECL_OVERRIDE` 类似,`Q_DECL_FINAL` 也

必须用于函数末尾,例如:

```
void MyWidget2::mouseMoveEvent(QMouseEvent * event) Q_DECL_FINAL
```

■ deleted

有时 C++ 编译器会为我们自动生成一些函数。比如,当我们定义一个类却没有给出其构造函数时,编译器会自动生成一个默认的构造函数。然而,出于各种目的,有时不希望编译器为我们创建这种函数。比如,当需要创建一个单例时,就不允许用户调用构造函数,因而也不应该由编译器生成默认构造函数。一种办法是将这样生成的函数声明为私有的,这样用户就不能在类的外部调用。然而,声明成私有的函数,一旦出错,会让编译器的错误变得难以理解。

为了解决这一问题,C++11 增加了一种 deleted 函数,用于显式禁止 C++ 编译器生成代码。deleted 函数不允许被调用,一旦误用,编译器会发出错误。

Qt 5 通过引入 Q_DECL_DELETE 宏支持 deleted 函数。如果编译器支持 deleted 函数,这个宏将被展开为“= delete;”,否则为空。例如,如果不希望编译器自动生成 MyClass 类的默认构造函数,只需这样编码:

```
MyClass() Q_DECL_DELETE;
```

2.5 全新的插件系统

Qt 默认提供了一套插件系统,应用程序可以利用这个插件系统对主程序进行扩展。Qt 的插件系统严重依赖于 moc 工具。Qt 5 的 moc 工具与 Qt 4 相比,有了很大的更新。所以,插件系统在 Qt 5 与 Qt 4 中也有了显著的不同。

可以在本书附带的代码目录 2-6 中找到 ShapePaint 和 ShapePaintPlugin 两个项目。前者是可以使用插件扩展的一个简单的画板程序;后者是基于这个画板程序的插件。这两个项目精简自 Qt 自带的示例程序,可以在 Qt 安装目录下找到这个示例程序。

Qt 的插件系统有两个主要组成部分:插件接口和实现了这个接口的插件类。Qt 主程序通过接口与插件进行交互,而接口的具体实现则是通过插件类完成。这样,主程序只需要知道接口,就能够利用多态机制,通过不同插件实现完成不同的功能。

ShapePaint 中的 brushinterface.h 头文件就是演示程序的接口。首先来看 Qt 4 版本的具体定义。

```
class BrushInterface
{
public:
    virtual ~BrushInterface() {}
    virtual QStringList brushes() const = 0;
    virtual QRect mousePress(const QString &brush,
                             QPainter &painter,
                             const QPoint &pos) = 0;
    virtual QRect mouseMove(const QString &brush,
```