Qt- C++图形用户界面库

荆鹏

2009-12

http://blog.csdn.net/tingsking18/

- 一、QT介绍
- 二、QT特征
- 三、QT开发环境搭建
- 四、QT系统构造
- 五、QT系统的类和工具
- 六、信号和槽

1.1历史

Qt(发音同cute)是一个跨平台的C++应用程序开发框架，由挪威Troll Tech (奇趣)公司出品。

2008年6月被Nokia公司收购，目前他是Nokia的Qt Development Frameworks 部门的产品。

2009年5月Nokia宣布Qt源代码管理系统面向公众开放。Qt的源代码管理采用Git。

1.2授权模式

Qt分为商业版本和开源版本。

Qt提供三种授权方式，功能和性能没有区别。

- Qt 商业授权适用于开发专属和/或商业软件。此版本适用于不希望与他人共享源代码，可以任意的修改Qt的源代码，而不需要公开。 提供了技术支持服务。
- Qt 4.5及以后的版本开始遵循LGPL。LGPL允许链接到它的软件使用任意的许可证，可以被专属软件作为类库引用、发布和销售。
- 使用GPL V3许可，必须在源代码中包含GPL的许可，不可用于商业应用。

1.3支持平台

- **Linux/X11**：用于 **X Window System(如Solaris、AIX、HP-UX、Linux、BSD)**。支持 **KDevelop IDE** 集成
- **Mac**：用于 **Apple Mac OS X**。基于 **Cocoa** 框架。**Universal Binary** 支持。支持以 **Xcode** 编辑、编译和测试。
- **Windows**：用于 **Microsoft Windows**。支持 **Visual Studio** 集成
- **Embedded Linux**：用于嵌入式**Linux**。可以通过编译移除不常使用的组件与功能。通过自己的视窗系统**QWS**，不需依赖**X Window System**。可以减少存储器消耗。方便在桌面系统上进行嵌入式测试。
- **Windows CE / Mobile** ：用于 **Windows CE**
- **Symbian**：用于 **Symbian**

1.4Qt Solutions

Qt Solutions 提供 Qt 额外的组件和工具，使Qt的开发更有效率。在 Qt 4.5 之后，Qt Solutions 加入了 LGPL 的授权。

- 平台和特定行业的组件和工具
- 集成 Qt与特定第三方产品的组件和工具
- 尖端的组件和新的工具也会被直接加入在Qt 框架中发布

例如：**Qt/MFC Migration Framework 、qtbrowserplugin、CORBA Framework、Qt State Machine Framework、Qt Windows Forms Interop Framework、SOAP**

1.5语言绑定

- PyQt：Python绑定
- QtRuby：Ruby绑定
- Qt Jambi：Java绑定
- Qyoto：C# 或其他 .NET 语言绑定
- PythonQt：LGPL授权的 Python 绑定
- QtAda：Ada绑定
- FreePascal QT4：Pascal绑定
- Perl Qt4：Perl绑定
- PHP-Qt：PHP绑定
- QtLua：Lua 绑定
- QtD：D语言绑定
- qtcl：tcl绑定

1.6使用Qt开发的程序

- KDE：著名的桌面环境。
- Google地球（Google Earth）：三维虚拟地图软件。
- VirtualBox：虚拟机软件。
- Opera：著名的网页浏览器。
- VLC多媒体播放器：一个体积小巧、功能强大的开源媒体播放器。
- Arora：一款跨平台的开源网页浏览器
- eva：Linux版QQ聊天软件。
- Skype：一个使用人数众多的基于P2P的VOIP聊天软件。

- 一、QT介绍
- 二、QT特征
- 三、QT开发环境搭建
- 四、QT系统构造
- 五、QT系统的类和工具
- 六、信号和槽

Qt作为新型的GUI开发工具，具有与一般的工具包所不同的特征，使它的应用非常广泛。

2.1.面向对象
2.2.组件间的相互通信
2.3.友好的联机帮助
2.4.用户自定义
2.5.方便性
2.6.国际化
2.7.丰富的API函数
2.8.完整的一套组件
2.9.高性能的工具
2.10.GUI竞争
2.11.可用户化的外观
2.12.优越的绘画功能
2.13.绘制2D/3D图形功能

2.1.面向对象

Qt具有模块设计和组件或元素的可重用性的特点。一个组件不需要知道它的内容和用途，通过signal和slot与外界通信、交流。而且，所有Qt的组件都可通过继承。

2.2.组件间的相互通信

Qt提供signal和slot概念，这是一种安全可靠的方法，它允许回调，并支持对象之间在彼此不知道对方信息的情况下，进行合作，这使Qt非常合适于真正的组件编程。

2.3.友好的联机帮助

Qt包括大量的联机参考文档，有超文本HTML方式、UNIX帮助页、man手册和补充的指南。对于初学者，指南将一步步地解释Qt编程。

2.4.用户自定义

其他的工具包在应用时都存在一个普遍的问题，就是经常没有真正适合需求的组件，生成的自定义组件对用户来说，也是一个黑匣子。比如，在Motif手册中就讨论了用户自定义的组件的问题。而在Qt中，能够创建组件，具有绝对的优越性，生成自定义组件非常简单，并且容易修改组件。

2.5.方便性

由于Qt是一种跨平台的GUI工具包，所以，它对编程者隐藏了在处理不同窗口系统时的潜在问题。为了将基于Qt程序更加方便，Qt包含了一系列类，该类能够使程序员避免了在文件处理、时间处理等方面存在依赖操作系统方面的细节问题。

2.6.国际化

Qt为本地化应用提供完全的支持，所有用户界面的文本或字符串都可以利用翻译工具将其译成各国语言。另外，Qt完全支持双字节16bit国际字符标准。

2.7.丰富的API函数

为了适合用户的需求，Qt的API提供了250个C++类，该类大多数用于专门的GUI。Qt还提供了基于模板的初始化、文件和通用的I/O设备、
目录管理、日期/时间类、常用表达式解析等。目的是利用这些类，建立或生成不同的功能，用它们来实现Qt的通用化。除此之外，也可以利用STL标准模块库或其他工具包。

2.8.完整的一套组件

Qt编程的基本模块(构件)称为组件，一个组件是一个用户界面的组成部分，比如按钮、滚动条。Qt包含用来创建专业外观的用户界面所需要的所有组件。

2.9.高性能的工具

对于库来讲，它的有效性远超过应用性。为了提高Qt库的有效性、快捷性，对其进行了优化，Qt能执行一些基本的任务，比如图形的色，比一般的基于平台的代码要快。Qt是基于Xlib, 而不依赖Motif工具包。

2.10.GUI竞争

大多数GUI工具包是基于分层的方法。比如，工具包为本地窗口系统组件提供了很多C++类，这种结构使组件的继承性和通用性变得很差。在层次化的工具包中，GUI功能常成为所有使用的窗口系统所必须的最普遍的基础。Qt仿效本地窗口系统的组件，这是一种非常复杂的技术。Qt还提供一些更有用的函数，类似文本的旋转，适用于多种平台。

2.11.可用户化的外观

Qt支持主题，所以基于Qt的应用软件能在Mac外观、 Windows等外观主题之间互换，甚至改变运行时间。这些应用程序不管是在X Window下，还是在Microsoft Windows下都可以独立操作、运行。

2.12.优越的绘画功能

Qt的绘画工具QPainter类，在任意一个绘图设备上都可以润色图形。

绘图设备包括组件、像素映射、图形文件和打印机，相同的代码可以用在4种不同类型的设备上。 QPainter类支持复杂的同等系统的转换，很容易在所有平台上画旋转文本和像素映射。

2.13.绘制2D/3D图形功能

Qt提供了QGLWidget类,使用该类能够绘制2D/3D图形。用QGLWidget就像用一个Qt组件一样方便。这比纯粹的用OpenGL做的3D图形更好。推荐使用Troll Tech软件公司的HooPS库。

- 一、QT介绍
- 二、QT特征
- 三、QT开发环境搭建
- 四、QT系统构造
- 五、QT系统的类和工具
- 六、信号和槽

3.1Makefile介绍

3.1.1程序的编译和链接

一般来说，无论是C、C++、还是pas，首先要把源文件编译成中间代码文件，在Windows下也就是.obj 文件，UNIX下是.o文件，即Object File，这个动作叫做编译（compile）。然后再把大量的Object File合成执行文件，这个动作叫作链接（link）。

编译时，编译器需要的是语法的正确，函数与变量的声明的正确。对于后者，通常是你需要告诉编译器头文件的所在位置（头文件中应该只是声明，而定义应该放在C/C++文件中），只要所有的语法正确，编译器就可以编译出中间目标文件。一般来说，每个源文件都应该对应于一个中间目标文件（O文件或是OBJ文件）。

链接时，主要是链接函数和全局变量，所以，我们可以使用这些中间目标文件（O文件或是OBJ文件）来链接我们的应用程序。链接器只管函数的中间目标文件（Object File）

3.1.2Makefile 介绍

一个简单的Makefile文件包含一系列的“规则”，其样式如下：

目标(target): 依赖(prerequiries)

命令(command)

目标(target)通常是要产生的文件的名称，目标的例子是可执行文件或OBJ文件。目标也可是一个执行的动作名称

依赖是用来输入从而产生目标的文件，一个目标经常有几个依赖。

命令是Make执行的动作，一个规则可以含有几个命令，每个命令占一行。

注意：每个命令行前面必须是一个Tab字符，即命令行第一个字符是Tab

3.1.3Makefile 示例

```
edit : main.o kbd.o command.o display.o \
      insert.o search.o files.o utils.o
       cc -o edit main.o kbd.o command.o display.o \
           insert.o search.o files.o utils.o
main.o : main.c defs.h
       cc -c main.c
kbd.o : kbd.c defs.h command.h
       cc -c kbd.c
command.o : command.c defs.h command.h
       cc -c command.c
display.o : display.c defs.h buffer.h
       cc -c display.c
insert.o : insert.c defs.h buffer.h
       cc -c insert.c
search.o : search.c defs.h buffer.h
       cc -c search.c
files.o : files.c defs.h buffer.h command.h
       cc -c files.c
utils.o : utils.c defs.h
       cc -c utils.c
clean :
       rm edit main.o kbd.o command.o display.o \
          insert.o search.o files.o utils.o
```

- 3.2两种方法建立QT开发环境

3.2.1.编译源码

configure

make　(make install)

设置环境变量

3.2.2.安装SDK

设置环境变量

如果需要调试QT源码，也需要对源码进行编译。

- 3.3开发工具介绍

3.3.1.Qt Creator Nokia的开发工具。

速度慢，bug比较多。

跨平台，与QT集成比较紧密。

3.3.2.Eclipse+QT插件。

速度慢。

可以跨平台，环境比较统一。

3.3.3. Visual Studio (+QT插件) 。

只能在win下使用。

大家都比较熟悉，使用方便。

- 3.4.QT程序编译步骤

  1.qmake –project

  生成QT的工程文件.pro

  2.qmake

  根据.pro文件生成平台相关的Makefile

  3.make

  真正的编译。执行Makefile

- 一、QT介绍
- 二、QT特征
- 三、QT开发环境搭建
- 四、QT系统构造
- 五、QT系统的类和工具
- 六、信号和槽

库	描述
QtCore	核心非GUI功能
QtGui	核心GUI功能
QtNetwork	网络模块
QtOpenGL	OpenGL 模块
QtSql	SQL 模块
QtSvg	SVG 透视图类
QtXml	XML 模块
Qt3Support	支持Qt3的类
QAxContainer	ActiveQt 客户端的扩充
QAxServer	ActiveQt 服务器段的扩充
QtAssistant	Qt助手的语言类
QtDesigner	Qt设计器的扩展类
QtUiTools	生成动态GUI类
QtTest	单元测试工具类

QtCore不但包含QString、QList和QFile等工具类，而且包含QObject和QTimer等内核类。因为QApplication类有refactored，所以它能使用在非GUI应用程序中。它将拆分为：QCoreApplication(在QtCore中)和QApplication(在QtGui中)。

这种拆分将使用Qt开发服务器应用程序，无需连接所有多余的与GUI相关的代码，无需要求与GUI相关的系统程序将要放到当前的目标机器中成为可能。(例如：Xlib在X11上，Carbon在Mac OS X上)。

如果你想利用qmake命令生成Makefile文件，qmake将默认链接到你的应用程序依赖的QtCore和QtGui中。如果你想删除具有依赖关系的GUI，请在您的.profile文件中加入以下内容：

```
QT -= gui
```

如果想使用其他的库文件，请加入以下内容：

```
QT += network opengl sql qt3support
```

- 一、QT介绍
- 二、QT特征
- 三、QT开发环境搭建
- 四、QT系统构造
- 五、QT系统的类和工具
- 六、信号和槽

主要的类	描述
抽象窗口部件类	抽象窗口部件类是通过子类来使用的。
高级窗口部件类	高级的GUI窗口部件，比如列表视图和进度条
基本窗口部件类	基本的GUI窗口部件，比如按钮、组合框和滚动条
数据库类	与数据库相关的类，比如与SQL数据库相关的类。
日期与时间类	处理日期与时间的类
拖放类	处理拖放和MIME类型的编码和解码类
环境类	提供了多样全面的服务，比如事件处理、系统设置访问和国际化等 服务的类
事件类	用来生成和处理事件的类
非GUI类	非GUI类是一个集合类。比如：列表、队列、堆栈和字符串，它们不需要QApplication类就可以和其他类一起使用。
多媒体、图形和打印类	该类主要提供支持图形（2D、3D和OpenGL），图像的编码、解码和处理，声音，动画，打印等等。
帮助系统类	用来给应用程序提供在线帮助的类。
布局管理类	用来处理自动调整窗口部件的大小和位置，能够构成复杂对话框的类。
共享类	为了快速复制而使用引用计数的类。

主要的类	描述
输入/输出和网络类	提供文件输入输出，目录和网络操作。
主窗口和相关类	一切你所需要的典型现代主程序窗口，包括目录、工具条、工作区等等。
杂类	各种各样其他有用的类
模块/视图类	该类主要用于设计模块/视图平台的类。
对象模型	Qt图形用户界面的工具包底层对象模型。
组织者	用户接口组织者，比如：分隔器、TAB条、按钮组等等。
插件类	插件相关类。
标准对话框	用于文件、字体、颜色选择和更多的已经做好的对话框。
模板类	Qt的模板库容器类。
文本相关类	文本处理的类。（也可以参考XML类。）
线程类	提供线程支持的类。
窗口外观和风格	可以自定义风格、字体、颜色等等外观的类。
XML类	支持XML的类，例如：DOM和SAX。
Qtopia Core类	是Qtopia Core一个特殊的类(Qt的嵌入Linux)。

- assistant QT助手，提供QT的帮助
- Designer QT界面设计器
- Linguist QT语言家，提供国际化支持
- Lrelease 生成用于translation的ts文件
- Lupdate 生成qm文件
- Moc meta-object compiler
- Qmake 生成makefile，构建QT工程
- Qt3to4 从QT3到QT4转换的工具
- Rcc 编译qrc资源的工具
- Uic 编译ui文件的工具

QT国际化步骤

1.编写源代码

2.在pro中添加TRANSLATIONS+*.ts，有多少中语言就添加多少个ts文件。

3.运行lupdate *.pro 生成ts文件。lupdate会根据源代码中的内容提取出待翻译的字段，然后生成ts文件，ts文件是xml格式的。

4.用qt linguist打开ts文件，并翻译相应字段。

5.运行lrelease *.pro生成qm文件。lrelease会根据ts文件生成二进制的qm翻译文件。

6.使用QTranslator 加载qm文件。

7.为QApplication安装translator。

- 一、QT介绍
- 二、QT特征
- 三、QT开发环境搭建
- 四、QT系统构造
- 五、QT系统的类和工具
- 六、信号和槽

Qt部件不同于其他用户交互方式的GUI工具包。用户交互方式是所有GUI(Graphical user interface)应用程序关心的问题。通过将某种用户事件(比如按下鼠标)与程序事件(比如退出程序)联系起来，使用户能够在图形界面中只使用鼠标来控制程序。

而其他工具包是利用回调函数来进行用户交互的。

所谓的回调是指：你自己定义一个函数，并告诉系统何时为何调用。你可以写一个特定数量和类型参数的函数，然后告诉系统何时使用，并传递给它所需的参数，系统就会调用你定义的函数，处理参数，并给你返回值。

所谓的回调函数是指：按照一定的形式由开发人员定义并编写实现内容。使用回调函数，实际上就是在调用某个函数(通常是API函数)时，将自己的一个函数(也就是回调函数)的地址作为参数传递给那个函数。而那个函数在需要的时候，也就是某种事情发生的时候，利用传递的函数地址调用回调函数，这时开发人员可以利用这个机会在回调函数中处理消息或完成一定的操作。

回调函数只能是全局函数，或者是静态函数，因为这个函数只是在类中使用，所以为了维护类的完整性，我们用类的静态成员函数来做回调函数。

虽然回调函数能够实现用户之间的交互。但是，回调函数非常复杂，容易混淆，又难以理解(至少大部分编写Qt的工作人员或者程序员有过这样的想法)。因此，Qt的开发者使用另一种方法来完成这一工作。这种方法依赖于Qt特有的两个功能，信号和槽。使用这种新方法是非常简单的，只需要编写一行代码就能够将用户事件和程序事件连接起来。

这种将用户事件连接到程序事件的方法要比回调函数，更加容易使用的两个重要原因是：1> 槽和信号是你调用Qt库中的函数。2> 信号和槽不使用其他工具包。

6.1 理解信号和槽

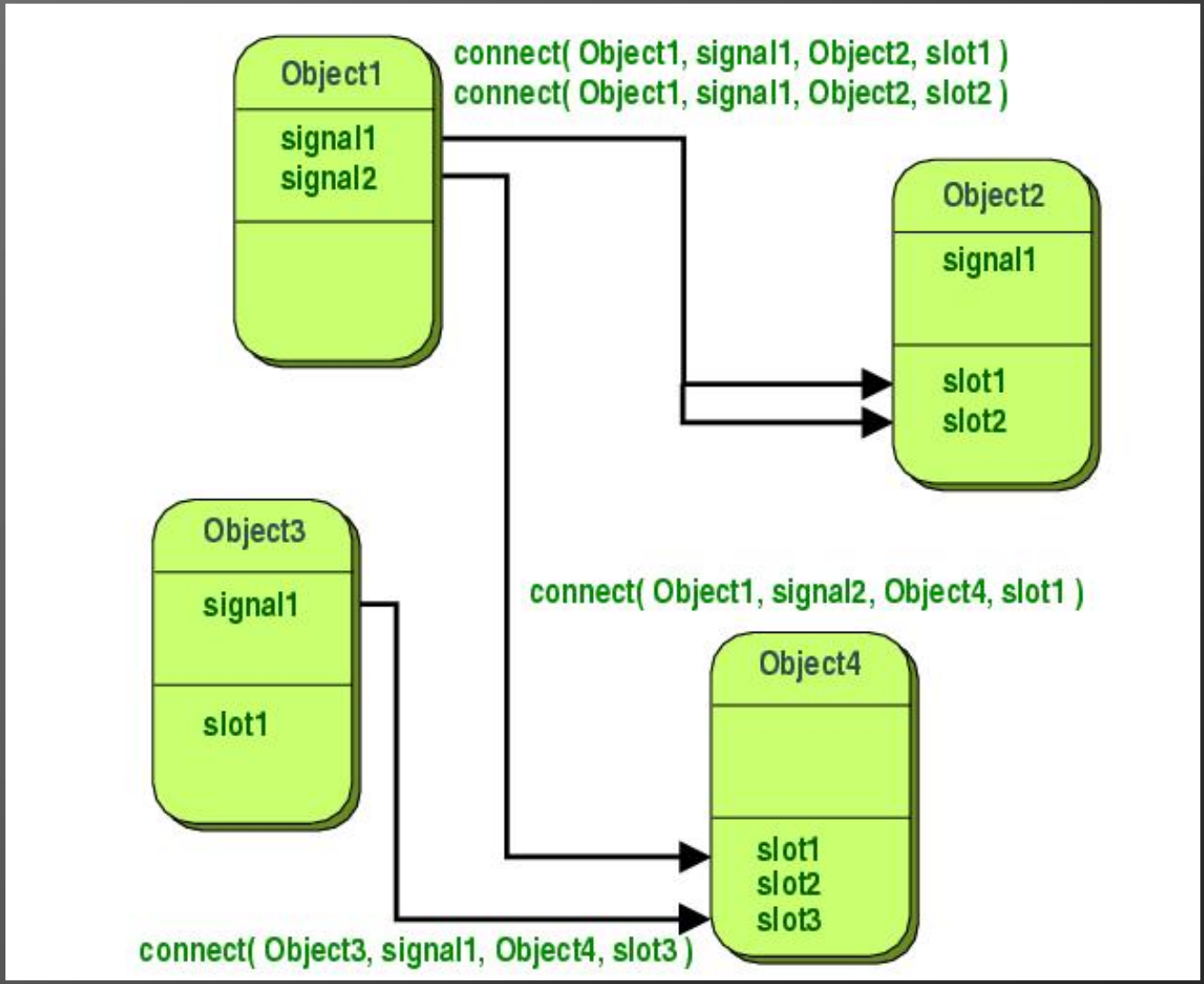
首先，我们必须理解信号(signal)和槽(slot)。这一技术有点不同于传统的回调(call back)函数。信号和槽技术是由Troll Tech公司独立开发的---而不是C++的功能。为了能够更好的对本章的理解，先让大家理解几个术语。

术 语	解 释
用户事件	指程序的用户所产生的事件。例如：点击鼠标或拖动滚动条
程序事件	指程序所产生的事件。例如：当用户点击鼠标后程序退出
发射信号	指“发出”或“发送”一个信号。例如：当你点击鼠标时，将发送一个clicked()信号。为了发送这个信号，使用emit关键字。
内部状态	当谈论信号时，你可能知道或者听说过“内部状态已经改变”。这就意味着对象内部数据已经发生改变，因此对象发送或发射一个信
MOC	元对象编辑器。用于构造用户自己的信号和槽。它处理Qt特有的关键字(例如：emit)，创建合法的C++代码。

信号和槽用于对象间的通信，该机制是Qt的一个中心特征，并且最能体现Qt与其他工具包它们之间有什么不同。在图形用户界面编程中，我们经常希望一个窗口部件的一个变化被告知给另外一个窗口部件。简单的说，我们希望任何一类的对象可以和其他对象进行通信。例如：我们正在解析一个XML文件，当我们遇到一个新的标签的时候，我们也许希望告知列表视图，我们正在用来表达XML文件的结构。

比较老的工具包使用一种被称做回调的通信方式来实现同一目的。回调是指一个函数的指针，所以如果你希望一个处理函数告知你一些事件，你可以把另一个函数(回调函数)的指针传递给处理函数。处理函数在适当的时候回调。回调有两个主要缺点：1> 它们不是类型安全的。我们从来都不能确定处理函数使用了正确的参数来调用回调。2> 回调和处理函数是非常强有力的联系在一起的，因为处理函数必须要知道调用哪个回调。

以下是一个信号与槽的连接图。



在Qt中我们有一种可以代替回调的技术，就是用信号和槽来代替。Qt的窗口部件有很多预定义的槽，当一个特定事件发生的时候，一个信号被发射，对信号感兴趣的槽就会调用对应的响应函数。

信号/槽机制是在QObject类中实现的。在QObject类或者其一个子类(比如：QWidget类)继承的所有类中，都存在了信号和槽。当对象改变其状态的时候，信号被发送，对象不关心有没有其他对象接收到这个信号。槽是类的正常成员函数，可以将信号和槽通过connect()函数任意连接。当一个信号被发射，它所连接的槽会被立即执行，如同一个普通函数调用一样。

6.1.1 槽

当信号被发出时，会调用与之相连接的槽。槽是普通的C++函数，可以用普通的方式来调用。它唯一特殊的地方在于可以与信号相连接。槽的参数不能有默认值。同样，信号的参数也不能有默认值。在槽的参数中尽量不使用自定义的数据类型，因为这样将会使通用性降低。

既然槽和普通的成员函数差不多，它们和普通成员函数一样有访问限制，

根据槽的访问限制谁可以与它们相连接，能够分为以下三种情况。

public slot: 任何信号都可以与之相连接。这在窗口部件编程中非常有用，
用于创建一些对彼此一无所知的对象，只有通过信号和槽来交换信息。public slot就像是标准的铁路一样。

protected slot: 只有该类及其子类所派生的对象的信号才可以与之相连接。这类槽的目的通常是为了类的完善，而不是类与外界的接口。

private slot: 只有该类自己的信号才可以与之相连接。

当然还可以将定义为virtual，这将非常有用。

信号和槽是相当高效的。当然，它们与“实时”的回调函数相比，在增加了灵活性的同时也损失了一些速度，正所谓有利必有弊，但是这种速度的损失相当微不足道。因此，信号/槽机制具有的简便性和灵活性的特性，使用信号和槽是用户交互的必然选择。

槽的声明也是在头文件中进行的。例如，下面声明了三个槽：

```
public slots:
void mySlot();
void mySlot(int x);
void mySignalParam(int x,int y);
```

6.1.2 信号

当某个信号对其客户或所有者发生的内部状态发生改变，信号被一个对象发射。只有定义过这个信号的类及其派生类能够发射这个信号。当一个信号被发射时，与其相关联的槽将被立刻执行，就象一个正常的函数调用一样。信号-槽机制完全独立于任何GUI事件循环。只有当所有的槽返回以后发射函数(emit)才返回。如果存在多个槽与某个信号相关联，那么，当这个信号被发射时，这些槽将会一个接一个地执行，但是它们执行的顺序将会是随机的、不确定的，我们不能人为地指定哪个先执行、哪个后执行。

例如，一个列表框可以发出highlighted()和activated()的信号，大多数对象也许只对activated()的信号感兴趣，但也许有些对象需要知道该列表框中的哪一项被选中了。如果有两个不同的对象对一个信号感兴趣，只要将该信号连接到这两个对象的槽上就可以了

当一个信号被发出的时候，与之相连接的槽就立即执行，就像通常的函数调用一样。信号/槽机制与任何图形用户界面的事件循环完全无关。当所有的槽返回之后，才返回到发出信号的地方。

如果有多个槽与一个信号相连接，则这些槽将被一个接一个地执行，而且其执行顺序是随意的。

信号是由MOC自动生成的，必须在.cpp文件中实现，而且永远没有返回值。

6.2 使用预定义信号和槽

在上一章的“Qt4的类”中的程序中实现按钮退出的功能时，使用了信号和槽。当然，Qt还包括许多其他信号和槽，大家可以以各种方法来使用它们。本节我们就介绍预定义的信号和槽，以及如果使用它们。

6.2.1 利用QSlider和QSpinBox来实现信号和槽

我们现在要用信号和槽来实现当你移动滚动条的时候SpinBox中的数字随之改变功能。该程序如何编写呢？请看下面的程序。

效果：

6.3 元对象系统

6.3.1 元对象系统的概述

Qt中的元对象系统是用来处理对象间通信的信号/槽机制、运行时的类型信息和动态属性系统，它基于QObject类、类声明中的私有段中的Q_OBJECT宏和元对象编译器(MOC)。

MOC读取C++源文件，如果发现类的声明中含有Q_OBJECT宏，则为含有Q_OBJECT宏的类生成另一个含有元对象代码的源文件，这个生成的源文件可以把类的源文件和这个类的实现一起编译连接。

元对象系统不但提供对象间通信的信号/槽机制，而且在QObject中的元对象代码能够实现以下特性：

className()函数 该函数在运行的时候，以字符串返回类的名称，不需要C++编译器中的本地运行类型信息(RTTI)的支持。

interits()函数 该函数返回本对象在QObject继承树中一个特定类的实例。

tr()和trUtf8()函数 该函数用于国际化中的字符串的翻译。

setProperty()和property()函数 该函数用来通过名称动态设置，并且获得对象属性。

metaObject()函数 该函数返回这个类所关联的元对象。

只有在类的定义中声明了Q_OBJECT宏，该类才能够使用元对象系统相关的特性。给大家提一个建议：QObject类的所有子类必须使用Q_OBJECT宏，无论它们是否使用了信号、槽。

当元对象编译器读取C++源文件的时候，如果发现其中一个或多个类的声明中含有Q_OBJECT宏，那么，使用Q_OBJECT宏的类生成另外一个包含元对象代码的C++源文件。

当使用qmake来生成Makefile文件的时候，编译规则中会包含了调用元对象编译器。因此，不需要直接使用元对象编译器。

元对象编译器生成的输出文件必须被编译和连接的，就像用户程序中的其他C++代码一样，该操作可以通过两种方法来解决。

通过类的声明放在一个头文件(.h文件)中来解决

如果在自定义的头文件中发现了类的声明，元对象编译器的输出文件将会被放在一个名为“moc_自定义文件名.cpp”中。该文件和普通文件一样也要进行编译，输出对象的结果是“moc_自定义文件名.o”(在unix系统或者Linux系统下)文件或者是“moc_自定义文件名.obj”(在Windows系统下)文件。之后，该对象会被包含到一个对象列表中，他们在程序的最后连接的时候连接在一起。

通过类的声明放在一个实现文件(.cpp文件)中来解决

如果在自定义文件customfile.cpp中发现了类的声明，元对象编译器的输出文件将会放在一个名为“customfile.moc”的文件里。该文件需要通过customfile.cpp文件包含，也就是说，customefile.cpp文件的所有代码之后添加了以下的程序：

```
#include "customfile.moc"
```

这样，元对象编译器生成的代码将会与customfile.cpp中的类定义一起编译和连接。

通过类的声明放在一个头文件(.h文件)中来解决的方法是常规的方法；通过类的声明放在一个实现文件(.cpp文件)中来解决的方法在实现文件自包含，或者是在使用Q_OBJECT宏的类的内部实现，并且在头文件中不可见的情况下使用。

6.3.2 Makefile文件中自动使用元对象编译器的方法

除了最简单的测试程序之外的任何程序，建议自动使用元对象编译器。在你的程序的Makefile文件中加入一些规则，make就会在需要的时候运行元对象编译器和处理元对象编译器的输出。

我们建议使用Trolltech的自由makefile的生成工具---qmake，来生成你的Makefile文件。这个工具可以识别上述两种方法风格的源文件，并且建立一个可以执行所有必要的元对象编译操作的Makefile文件。

如果你想自己建立一个Makefile文件，以下是如果包含元对象操作的一些方法。

对于在头文件中声明了Q_OBJECT宏的类，如果只使用GNU的make,则在Makefile文件中添加元对象编译规则如下：

```
moc_%.cpp: %.h
moc $< -o $@
```

如果你想方便的写Makefile文件，你可以按以下的格式写单独的规则：

```
moc_NAME.cpp: NAME.h
moc $< -o $@
```

你必须记住要把moc_NAME.cpp添加到你的SOURCES(也可以使用你自定义的名称代替)变量中，并且把moc_NAME.o(在UNIX操作系统或者Linux操作系统下)或者moc_NAME.obj(在Window操作系统下)添加到你的OBJECT变量中。

对于在实现文件(.cpp文件)中声明Q_OBJECT宏的类，建议你使用以下的Makefile文件规则：

```
NAME.o: NAME.moc
NAME.moc: NAME.cpp
moc –I $< -o $@
```

这将保证make程序在编译NAME.cpp文件之前运行元对象编译器，然后把下面这行程序放在NAME.cpp的文件结尾，这样在这个文件中的所有的类的声明都知道这个元对象：

```
#include "NAME.moc“
```

元对象在编译的过程中常见的错误如下所示：

```
YourClass::className() is undefined
```

或者是：

```
YourClass lacks a vtbl
```

出现这种错误的绝大多数原因是忘记了编译或者#include元对象编译器产生的C++代码，或者没有在连接命令中包含相应的对象文件。

6.3.3 调用元对象编译器moc

以下是元对象编译器moc所支持地命令行选项：

-o file 将输出写入file文件中，而不是标准输出。

-f 强制在输出文件中生成#include声明。文件的名称必须符合正则表达式，也就是说，扩展名必须以H或者h开始。这个选项只有在你的头文件没有遵循匈牙利标准命名法则的时候在有作用。

-ldbg 把大量的lex调试信息写到标准输出。

-p path 使元对象编译器生成的#include声明的文件名称中预先考虑到path/。

-q path 使元对象编译器在生成的文件中的qt #include 文件的名称中预先考虑到path/。

你可以明确地告诉元对象编译器不要解析头文件中的成分。它可以识别包含子字符串MOC_SKIP_BEGIN或者MOC_SKIP_END的任何C++的注释(//)。它们正如你所期望的那样工作，并且你可以把它们划分为若干层次。元对象编译器所看到的最终结果就好象你把一个MOC_SKIP_BEGIN和MOC_SKIP_END当中的所有行删除一样。

6.3.4 元对象编译器的限制

元对象编译器并不展开#include或者#define,而只是简单地忽略所遇到的所有预处理程序，并且无法处理所有的C++语法，主要问题是类模板不能含有信号和槽。以下是一个错误的实例：

```
class SomeTemplate<int> : public Qframe
{
        Q_OBJECT
        signals:
        void bugInMocDetected(int);
};
```

元对象编译器的限制下说明如下：

多重继承把QObject的子类作为第一个父类

如果使用多重继承，元对象编译器认为第一个继承类是QObject的子类。也就是说，必须首先要继承的类是QObject类。这是因为元对象编译器并不展开#include或者#define，所以它无法发现基类中哪个是Qobject类。比如：

```
class SomeClass : public QObject, public OtherClass
{
        ……
}
```

函数指针不能作为信号和槽的参数

在你考虑使用函数指针作为信号/槽的参数的时候，继承是一个不错的替代方法。以下是一个不合法的程序：

```
class SomeClass : public QObject
{
    public slots:
        void apply(void (*apply)(List *,void *),char *);//不合法
};
```

可以用以下的方法纠正这个错误：

```
typedef void (*ApplyFunctionType)(List *, void *);
class SomeClass : public QObject
{
        Q_OBJECT
public slots:
        void apply(ApplyFunctionType, char *);
};
```

有时用继承和虚函数、信号和槽来替换函数指针效果会更好。

不能把友元声明friend放在信号或者槽的声明部分

通常情况下，友元声明friend不能放在信号或者槽的声明的部分，把它们替换为private、protected或者public部分中。以下是一个不合法的实例：

```
class SomeClass : public QObject
{
        Q_OBJECT
        ……
        signals:
        friend class ClassTemplate<char>; //错误
};
```

信号和槽不能被升级

把继承的成员函数升级为公有状态，这个C++特征对信号和槽并不适用。以下是一个不合法的实例：

```
class Whatever : public QButtonGroup
{
    public slots:
        void QButtonGroup::buttonPressed; //错误，槽是保护的
        ……
};
```

类型宏不能用于信号和槽的参数中

因为元对象编辑器不能展开#define，所以，在信号和槽中类型宏作为一个参数不能工作。以下是一个不合法的实例：

```
#ifdef ultrix
#define SIGNEDNESS(a) unsigned a
#else
#define SIGNEDNESS(a) a
#endif
class Whatever : public QObject
{
        ……
signals:
        void someSignal(SIGNEDNESS(int));
        ……
};
```

不含有参数的#define，将会向你所想的那样工作。

嵌套类不能放在信号部分或者槽部分，也不能含有信号和槽

请看以下实例：

```
class A
{
    Q_OBJECT
    public:
        class B
        {
            public slots:
                void b();       //错误的
                ……
};
signals:
        class B
        {
            void b();           //错误的
};
};
```

构造函数不能用于信号部分和槽部分

为什么一个人会把一个构造函数放到信号部分或者槽部分，这对于大家来说都是很有意思的事情。无论如何也不能这样做。请把它们放到private、protected或者public部分中，它们本来就属于那个地方。以下是不合法的实例：

```
class SomeClass : public QObject
{
    public slots:
    someClass(QObject *parent, const char *name) : QObject(parent , name) //错误的
    {
        ……
    }
};
```

属性的声明应该放在含有相应的读写函数的公有部分之前

如果在public之后声明属性，元对象编译器将不能找到函数或者解析这个类型，以下是一个错误的实例：

```
class SomeClass : public QObject
{
    Q_OBJECT public:
    ……
    Q_PROPERTY( Priority priority READ priority WRITE setPriority ) // 错的
    Q_ENUMS( Priority ) // 错误的
    enum Priority { High, Low, VeryHigh, VeryLow };
    void setPriority( Priority );
    Priority priority() const;
    ……
};
```

该实例纠正错误以后的代码，如下所示：

```
class SomeClass : public QObject
{
    Q_OBJECT
    Q_PROPERTY( Priority priority READ priority WRITE setPriority )
    Q_ENUMS( Priority )
    public:
     ……
    enum Priority { High, Low, VeryHigh, VeryLow };
    void setPriority( Priority );
    Priority priority() const;
     ……
};
```

6.4 创建自定义的信号和槽

创建一个信号和槽的工作是相当简单的，大量的工作都自动由Q_OBJECT宏和元对象编译器来完成了。发送信号的过程和槽接受信号的过程完全分离开来。一个对象可以发送任意数目的信号而不必知道到底发了多少，不管信号的数目有多大，槽都可以接收。那么，我们如何创建一个信号呢？

以下是创建一个信号的方法：

在类定义的第一行加入Q_OBJECT宏。类中的其他项都需要一个分号终止符，而Q_OBJECT宏却不需要，但如果喜欢，也可以加上分号(是因为编译器在处理时根本不考虑分号)。我们可以按照以下方式定义一个类：

```
class SenderClass
{
        Q_OBJECT
        ……
```

值得注意的是：在一个类中可以定义任意个槽和信号，但是Q_OBJECT只需要一次。

向类定义中加入信号的原型。比如：如果信号将要发送一个字符串作为该信号的参数，那么原型大概以如下的方式去编写：

```
……
signals:
        void newName(QString &name);
        ……
```

上面代码中没有公有或者私有的说明，这是因为并没有实际的方法---这只是一个原型的定义，用来调用接收的槽。

使用发送语句来调用所有监听这个信号的方法。这一步使用的语法和用来调用一个局部方法的语法是一样的，只不过这时用emit关键字开头：

```
QString name;
emit newName(name);
```

值得注意的是：并没有关于信号方法实体的实际定义，emit命令并不寻找局部的方法；相反，它调用已经连接到此信号的槽列表中的所有的槽。

既然我们已经知道如何创建信号，那么，接着我们介绍如何创建一个槽？

以下是创建槽，并把它和信号相连接的方法：

与信号一样，槽需要在类定义的上部加入Q_OBJECT宏：

```
class ReceiverClass
{
        Q_OBJECT
        ……
```

向类定义中加入槽方法的原型。这个原型必须与它将要接收的信号一样(也就是说，具有同样的一套参数)。由于槽是方法，所以，在作为槽使用的同时，也可以被直接调用。槽的方法可以设置成为公有的属性。

```
……
public slots:
void nameChanged(QString &name);
……
```

更常见的情况是，槽的目的仅仅是接收信号，这时你可以将它设置成为私有的属性。

```
……
private slots:
void nameChanged(QString &name);
……
```

包含定义了将要发送信号的类的头文件。

编写代码创建将要发送信号的类的实例。只有这个实例的存在，才能把槽和信号联系在一起。

把槽和信号连接起来。这个工作通常在构造函数中完成，但是如果这个对象构造得比较晚，那么连接工作也可以晚点做。调用connect()方法把你的槽加入到方法列表中，每当指定的信号发出的时候，这个方法就会被调用。

可以按照以下的方式调用connect()方法：

```
connect(sender,SIGNAL(newName(QString &)),this,
SLOT(nameChanged(QString &)));
```

注 意：

前两个参数指定信号的来源，后两个参数指定目标槽。宏SIGNAL()和SLOT()都需要完整的方法原型，原型必须遵循，用来调用一种方法的参数必须和该种方法可以使用的参数保持一致。

无论任何使用emit发送信号，就好象是你编写的程序直接调用每一个槽方法一样。也就是说，直到槽方法返回，你编写的程序才能继续执行。因此，通常应当保持在槽方法内部的处理过程中尽可能的简单，这样才不会因此中止信号的发送。发送信号的可能是用户接口过程，操作过程表现得比较慢或者缓慢。

必须小心不要创建死循环。如果一个槽方法发送一个信号，此信号直接或者间接地执行了发送一个信号的方法，而这个信号又被最开始的槽所接收，那么信号将连续不断的调用槽，你编写的程序就会崩溃。比如：如果名为firstfun()的方法发送了一个A信号，A信号被second()槽所接收，而second()槽发送了信号B，最后，名为firstfun()的方法接收了信号B，这样就产生了一个死循环。这种循环将一直执行，直到该程序崩溃为止(或者用户进入长时间的等待)。

还需要小心槽和信号方法在连接语句中的参数是否匹配。当程序运行的时候，直到试着去解决一个问题时，才可能得到出错的信息。为了避免这个问题的出现，必须确定每次增加内容的时候，都要进行测试，或者改变槽和信号部分。唯一的出错信息是当connect()方法找不到匹配对象的时候，输出一个写入控制台上的字符串。此后，程序就忽略了这个信号的存在。只有从命令行运行程序的时候，才能够看到控制台输出的信息。

必须小心不要创建死循环。如果一个槽方法发送一个信号，此信号直接或者间接地执行了发送一个信号的方法，而这个信号又被最开始的槽所接收，那么信号将连续不断的调用槽，你编写的程序就会崩溃。比如：如果名为firstfun()的方法发送了一个A信号，A信号被second()槽所接收，而second()槽发送了信号B，最后，名为firstfun()的方法接收了信号B，这样就产生了一个死循环。这种循环将一直执行，直到该程序崩溃为止(或者用户进入长时间的等待)。

还需要小心槽和信号方法在连接语句中的参数是否匹配。当程序运行的时候，直到试着去解决一个问题时，才可能得到出错的信息。为了避免这个问题的出现，必须确定每次增加内容的时候，都要进行测试，或者改变槽和信号部分。唯一的出错信息是当connect()方法找不到匹配对象的时候，输出一个写入控制台上的字符串。此后，程序就忽略了这个信号的存在。只有从命令行运行程序的时候，才能够看到控制台输出的信息。

Q&A

谢谢大家！