Qt 4.2 白皮书

奇趣科技
www.trolltech.com

摘 要

此白皮书介绍了 Qt C++ 框架。Qt 采用“一写永逸”的方法支持开发跨平台的 GUI 应用程序。使用单一源码树和简单的重编译方式，可以为 Windows 98 至 Windows XP、Mac OS X、Linux、Solaris、HP-UX 以及其他使用 X11 的 Unix 版本编写各种程序。此外，还可以编译在嵌入式 Linux 平台中运行的 Qt 应用程序。Qt 引入了一种名为“信号和槽”的独特对象间通信机制。Qt 还为 2D 和 3D 图形、国际化、SQL、XML、单元测试提供了卓越的跨平台支持，并为特定应用程序提供了针对具体平台的扩展。您可以使用 *Qt Designer（Qt 设计者）*（一种支持 IDE 集成的灵活用户界面构建器）构建器来可视化的建立 Qt 应用程序。

目录

1. 简介

Qt是事实上的标准 C++ 框架，用于高性能的跨平台软件开发。除了拥有扩展的 C++ 类库以外，Qt 还提供了许多可用来直接快速编写应用程序的工具。此外，Qt 还具有跨平台能力并能提供国际化支持，这一切确保了 Qt 应用程序的市场应用范围极为广泛。

自 1995 年以来，Qt C++ 框架一直是商业应用程序的核心。无论是跨国公司和大型组织（例如：Adobe®、Boeing®、IBM®、Motorola®、NASA、Skype®）、还是无数小型公司和组织都在使用 Qt。Qt 4 在新增更多强大功能的同时，旨在比先前的 Qt 版本更易于使用。Qt 的类功能全面，提供一致性接口，更易于学习使用，可减轻开发人员的工作负担、提高编程人员的效率。另外，Qt 一直都是完全面向对象的。

此白皮书概述了 Qt 的工具和功能。每节开头都会附有一段非技术性的简介，然后再开始详细描述相关功能。对于每个主题范围，我们还会提供指向在线资源的链接。

若要试用 Qt 30 天，请访问 http://www.trolltech.com/。

1.1. 内容概要

Qt 拥有一系列窗体（在 Windows 术语中称为“控件”），这些窗体可提供标准的 GUI 功能（请参见第 5 页）。Qt 引入了一种名为“信号和槽”（请参见第 13 页）的新型替代技术，供对象之间通信使用，它取代了在过时的框架中使用的老旧的、缺乏安全性的传统回调技术。另外，Qt 还提供了一种传统事件模型（请参见第 58 页），用来处理鼠标单击、按键以及其他用户输入操作。Qt 的跨平台 GUI 应用程序（请参见第 16 页）可以支持现代应用程序所需的所有用户界面功能，例如：菜单、上下文菜单、拖放以及可停靠工具栏。使用 Qt 提供的桌面集成功能（请参见第 23 页），通过利用每个平台提供的服务，可以将应用程序扩展至所在的桌面环境中。

Qt 还提供了一款专门用于用户界面图形设计的工具 —— *Qt Designer*（请参见第 24 页）。Qt Designer 除了提供绝对定位功能以外，还支持强大的布局功能（请参见第 45 页）。使用 Qt Designer，既可专门设计 GUI，又可以利用它提供的与流行的集成开发环境集成的功能，用其开发整个应用程序。

Qt 对 2D 和 3D 图形有着卓越的支持（请参见第 31 页）。Qt 实际上是针对平台独立的 OpenGL® 编程而开发的标准 GUI 框架。Qt 4 的绘图系统为所有支持的平台提供了高质量的渲染功能。使用 Qt 4 的高级画布框架（请参见第 35 页），开发人员可以创建各种交互式图形应用程序，从而充分利用 Qt 的先进绘图功能。

有了 Qt，您可以使用标准数据库（请参见第 43 页）创建与平台无关的数据库应用

程序。针对 Oracle®、Microsoft® SQL Server、Sybase® Adaptive Server、IBM DB2®、PostgreSQL.、MySQL®、Borland® Interbase、SQLite 和 ODBC 兼容的数据库，Qt 提供了本地驱动。另外，Qt 还提供了专用于数据库的控件，使任何内建或自定义控件均可感知数据。

使用 Qt 的样式支持和主题支持功能（请参见第 55 页），Qt 编程可获得所有支持的平台的本地化观感。从单一源码树只需采用重新编译方式即可为 Windows 98 至 Windows XP、Mac OS X、Linux、Solaris、HP-UX 以及其他使用 X11 的 Unix 版本生成应用程序。另外，Qt 应用程序经过编译也能在 Qtopia 中运行。使用 Qt 的 qmake 构建工具可为目标平台生成相应的makefile或 .dsp 文件。

由于Qt的架构充分利用了底层平台的优点，许多用户在Windows, Mac OS X和Unix平台上做单一平台开发时也使用Qt， 因为他们更愿意使用Qt的方法。Qt 包含了对具体平台的特有功能的支持，例如：Windows 中的 ActiveX® 以及 Unix 中的 Motif. 。有关详细信息，请参阅《Qt 的架构》一节（请参见第 72 页）。

Qt 全面使用 Unicode™，并且对国际化支持十分成熟（请参见第 47 页）。Qt 还为翻译人员提供了 *Qt Linguist（语言家）*以及其他工具。应用程序可以轻松地混合使用阿拉伯语、汉语、英语、以色列语、日语、俄罗斯语以及 Unicode 支持的其他语言。

Qt 提供了一系列与特定域相关的类。例如，Qt 的 XML 模块（请参见第 61 页）提供了 SAX 和 DOM 类，可以读取并操作以 XML 格式存储的数据；使用 Qt 的 STL 兼容集合类（请参见第 65 页），可以将对象存储在内存中；使用 与Java® 和 C++ 标准模板库 (STL) 中相同样式的迭代器来操作对象；使用 Qt 的输入/输出和网络连接类（请参见第 60 页），可以使用标准协议处理本地文件和远程文件。

插件和动态库（请参见第 67 页）可以进一步扩展 Qt 应用程序的功能。插件提供了附加编解码器、数据库驱动、图像格式、样式和控件。插件和库可以作为自有知识产权的产品出售。

Qt 是一种成熟的 C++ 框架，在全球各地广泛使用。Qt 除了具有众多商业用途以外，其开源版本还为 KDE（即：Linux 桌面环境）奠定了基础。Qt跨平台的构建系统、可视化窗体设计以及一流的 API，使应用程序开发成为一种乐趣。

在线参考
http://www.trolltech.com/company/customers
http://partners.trolltech.com/

2. 窗体

Qt 提供了一系列标准窗体，使用这些部件，可以为应用程序创建图形用户界面。Qt 的窗体灵活易用，可以进一步派生子类，以满足特殊需求。

窗体是指组合在一起创建用户界面的可视元素。按钮、菜单、滚动条、消息框以及应用程序窗口都是窗体的例子。Qt 的窗体并没有在“控件”或“容器”之间加以截然区分。所有窗体既可以作为控件使用、又可以作为容器使用。通过从现有 Qt 窗体派生子类或者从头开始新建窗体（如有必要），则可轻松创建自定义的窗体。

标准窗体是由 **QWidget** 类及其子类提供的，而自定义窗体则可通过从标准窗体派生子类并重写虚函数的方式来创建。

窗体可以包含任意数量的子窗体。子窗体位于父窗体区域内。不含父窗体的则称为顶层窗体（“窗口”），这类窗体通常在桌面环境的任务栏中有自己的启动入口。Qt 对窗体没有任何硬性限制。任何一个窗体都可以成为顶层窗体；同时任何一个窗体也可以成为其他窗体的子部件。使用布局管理器（请参见第 52 页）可以自动设置子窗体在父区域内的位置，如有必要，也可以手动设置。禁用、隐藏或删除父窗体时，也同样会递归地禁用、隐藏或删除其所有子窗体。

标签、消息框、工具提示以及其他文本窗体并不限定于某一种色彩、字体和语言。Qt 的文本着色窗体使用 HTML 的子集，可以显示丰富的多语言文本（请参见第 48 页中的“文本输入和显示”）。

2.1. 内建窗体

下一页的屏幕快照展示了不同用户界面组件中所选择使用的 Qt 窗体。这些窗体使用 *Qt Designer* 来布局，使用 Plastique 样式来着色，展示了 Qt 4 在 Linux 中的标准外观。

屏幕快照中所示窗体包括标准输入控件，例如：**QLineEdit** 用来输入一行文本；**QCheckBox** 用来启用/禁用简单的独立设置；**QSpinBox** 和 **QSlider** 用来指定数量；**QRadioButton** 用来启用/禁用互斥的设置；**QComboBox** 表示单击时，将打开显示选择菜单。可点击的按钮则由 **QPushButton** 提供。

另外，屏幕快照中还显示了容器窗体，例如 **QTabWidget** 和 **QGroupBox**。这些窗体是专门由 *Qt Designer* 管理的，可以帮助设计人员快速创建并维护新用户界面。与用户界面设计人员相比，开发人员更倾向于经常使用比较复杂的窗体（例如，“Create Poster”对话框（图 1）中显示的 **QScrollArea**），其原因是这些复杂窗体可以用来显示专业化内容或动态内容。

Qt 提供的窗体远远不止此处所列出的这些。Qt 的在线 Widget Gallery 中还提供了许多可用的窗体以及指向其类文档的链接。

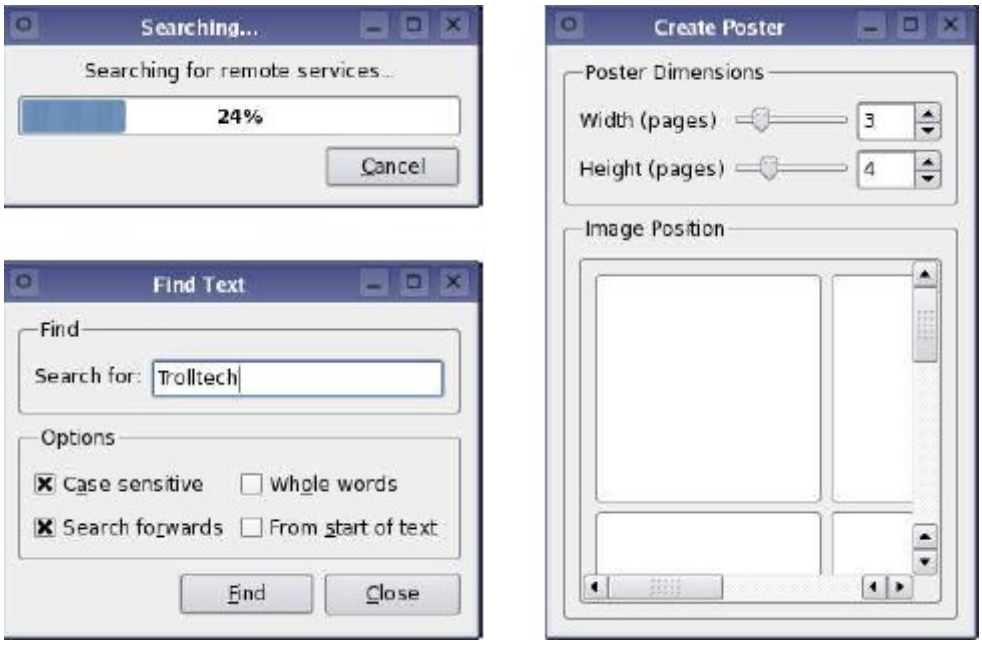

图 1：使用不同控件创建的对话框。

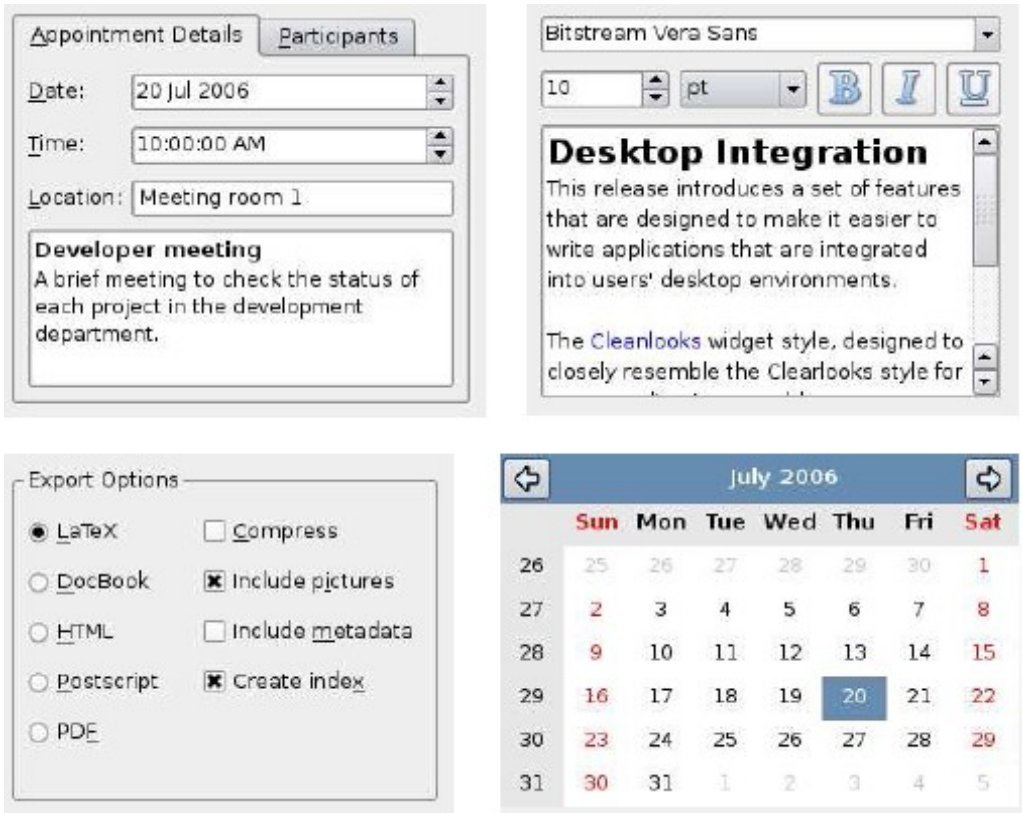

图 2：Qt 提供了一系列标准窗体。

您可轻松使用手动方式编写用户界面。可以使用以下代码来创建“Find Text”对话框（图 1）的选项组框。

```
QGroupBox *optionsGroupBox = new QGroupBox(tr("Options"));
QCheckBox *caseCheckBox = new QCheckBox(tr("C&ase sensitive"));
QCheckBox *directCheckBox = new QCheckBox(tr("Search fo&rwards"));
QCheckBox *wordsCheckBox = new QCheckBox(tr("Whole &words"));
QCheckBox *startCheckBox = new QCheckBox(tr("From &start of text"));
QGridLayout *optionsLayout = new QGridLayout;
optionsLayout->addWidget(caseCheckBox, 0, 0);
optionsLayout->addWidget(wordsCheckBox, 0, 1);
optionsLayout->addWidget(directCheckBox, 1, 0);
optionsLayout->addWidget(startCheckBox, 1, 1);
optionsGroupBox->setLayout(optionsLayout);
```

2.2. 定制窗体

通过从 **QWidget** 及其派生类派生子类，开发人员可以创建自己的窗体和对话框。为了举例说明派生子类的步骤，我们从 Qt 4 示例目录中列出了模拟时钟窗体的完整代码，该部件可以显示当前时间并自动更新。

AnalogClock 窗体在 analogclock.h 文件中定义：

```
#include <QWidget>
class AnalogClock : public QWidget
{
Q_OBJECT
public:
        AnalogClock(QWidget *parent = 0);
protected:
        void paintEvent(QPaintEvent *event);
};
```

该窗体继承通用的 **QWidget** 类，其构造函数是典型的窗体类的构造函数形式，带有一个可选的 *parent* 参数。paintEvent() 函数从 **QWidget** 继承而来，此函数在任何窗体需要更新时被调用。

AnalogClock 类的实现在文件analogclock.cpp中：

```
#include <QtGui>
#include "analogclock.h"

AnalogClock::AnalogClock(QWidget *parent)
: QWidget(parent)
{
        QTimer *timer = new QTimer(this);
        connect(timer, SIGNAL(timeout()), this, SLOT(update()));
        timer->start(1000);
        setWindowTitle(tr("Analog Clock"));
        resize(200, 200);
```

```
}
```

构造函数设置了一个定时器，给窗口设置了一个标题，并确保窗口默认大小正确合理。计时器的配置设置为每 1000 毫秒发送一次信号。启动计时器之前，系统使用 Qt 的信号和槽机制（请参见第 13 页）将此计时器连接至窗体的 update() 功能，从而确保时钟显示最新时间。

每次调用时，paintEvent() 函数都会简单地将整个窗体重新绘制一次。它使用 Qt 的绘图系统（请参见第 31 页）绘制钟表盘和时针分针。

```
void AnalogClock::paintEvent(QPaintEvent *)
{
        static const QPoint hourHand[3] = {
        QPoint(7, 8),
        QPoint(-7, 8),
        QPoint(0, -40)
};

static const QPoint minuteHand[3] = {
        QPoint(7, 8),
        QPoint(-7, 8),
        QPoint(0, -70)
};

QColor hourColor(127, 0, 127);
QColor minuteColor(0, 127, 127);
int side = qMin(width(), height());
QTime time = QTime::currentTime();
```

该函数首先以窗体的最短边作为时钟尺寸，设置简单主体信息及色彩信息。

然后在窗体的中央绘制钟表盘，并使用反锯齿（如果可用）在正确位置上绘制时针和分针。

```
QPainter painter(this);
painter.setRenderHint(QPainter::Antialiasing);
painter.translate(width() / 2, height() / 2);
painter.scale(side / 200.0, side / 200.0);

painter.setPen(Qt::NoPen);
painter.setBrush(hourColor);

painter.save();
painter.rotate(30.0 * ((time.hour() + time.minute() / 60.0)));
painter.drawConvexPolygon(hourHand, 3);
painter.restore();

painter.setPen(hourColor);

for (int i = 0; i < 12; ++i) {
        painter.drawLine(88, 0, 96, 0);
        painter.rotate(30.0);
}

painter.setPen(Qt::NoPen);
painter.setBrush(minuteColor);

painter.save();
painter.rotate(6.0 * (time.minute() + time.second() / 60.0));
```

```
painter.drawConvexPolygon(minuteHand, 3);
painter.restore();

painter.setPen(minuteColor);

for (int j = 0; j < 60; ++j) {
if ((j % 5) != 0)
        painter.drawLine(92, 0, 96, 0);
        painter.rotate(6.0);
}
```

图 3：QT 4 模拟时钟示例表明了如何创建简单的自定义窗体。

在此示例中，main() 函数是最基本的功能。它仅仅设置了一个应用程序对象，构建并显示了时钟窗体。最后，需要启动应用程序的事件循环，以便 Qt 可以开始处理事件：

```
#include <QApplication>
#include "analogclock.h"

int main(int argc, char *argv[])
{
        QApplication app(argc, argv);
        AnalogClock clock;
        clock.show();
        return app.exec();
}
```

此示例程序只包含一个顶层时钟窗体，没有任何子窗体。若要构建复杂的窗体，则需将多个窗体组合在不同布局中。

在线参考
http://doc.trolltech.com/4.2/qwidget.html
http://doc.trolltech.com/4.2/examples.html#widget-examples
http://doc.trolltech.com/4.2/tutorial.html
http://doc.trolltech.com/4.2/gallery.html

3. 信号和槽

信号和槽为对象之间的通信提供了便利条件。它们易于理解和使用，并受到 Qt Designer 的全面支持。

GUI 应用程序要对用户操作做出响应。例如，当用户单击菜单项目或者工具栏按钮时，GUI 应用程序便会执行某段代码。实际上，我们更希望任何一类对象均可彼此互相通信。编程人员必须将事件与相关代码相关联。老的开发工具套件(Toolkit)使用的机制不是类型安全(type-safe)的（例如，容易引起崩溃），缺乏灵活性而且不是面向对象的。

奇趣科技公司创造了一种名为“信号和槽”的解决方案。信号和槽机制是一种功能强大的对象间通信机制，完全可以取代老旧的开发套件所使用的粗糙的回调和消息映射。信号和槽机制极为灵活，完全面向对象，并且使用 C++ 来实现。

使用原有回调机制，若要将某一代码与按钮关联在一起，必须将函数指针传输给该按钮。单击此按钮时，系统将调用此函数。而对于老的工具套件而言，调用此函数时，它不确保将正确类型的参数传递给该函数，这样很有可能导致崩溃。回调方法的另一问题是：它将 GUI 元素与功能紧紧地捆绑在一起，这样导致很难独立开发类。

而 Qt 的信号和槽机制则不同。发生事件时，Qt 窗体将会发出信号。例如，单击某一按钮时，该按钮将发出“clicked”信号。编程人员要想连接一个信号可以创建一个函数（即“槽”）、并调用 connect() 函数将信号与槽关联起来。Qt 的信号和槽机制不要求各类彼此感知，这样可以更轻松地开发极易重新使用的类。由于信号和槽都属于类型安全的，因此，类型错误都将报告为警告，因此不会发生崩溃。

例如，如果“退出”按钮的 clicked() 信号与应用程序的 quit() 槽相连，那么如果用户单击“退出”，则会终止该应用程序。如果以代码形式表示，则应将上述过程编写为：

```
connect(button, SIGNAL(clicked()), qApp, SLOT(quit()));
```

在执行 Qt 应用程序的过程中，可以随时添加或移除连接。可将连接设置为在发出信号时执行，或者排队稍后执行，可以在不同的线程的对象之间建立连接。

信号和槽通过平滑的扩展C++语法并充分利用C++的面向对象特性实现。信号和槽是类型安全的，可以重载，也可以重新实现，可以出现在类的公有区、保护区或私有区。若要使用信号和槽，必须继承 **QObject** 或其子类，并在类的定义中包括 Q_OBJECT 宏。信号在类的“信号区”声明，而槽则是在“公有槽区”、“保护槽区”或“私有槽区”中声明的。

3.1. 信号和槽示例

以下是 **QObject** 子类的示例：

```
class BankAccount : public QObject
{
Q_OBJECT
public:
        BankAccount() { curBalance = 0; }
        int balance() const { return curBalance; }
        public slots:
        void setBalance(int newBalance);
signals:
        void balanceChanged(int newBalance);
private:
        int currentBalance;
};
```

与多数 C++ 类的风格类似，**BankAccount** 类拥有构造函数、balance() "读取" 函数和 setBalance() "设置" 函数。它还拥有 balanceChanged() 信号，帐户余额更改时将发出此信号。发出信号时，与它相连的槽将被执行。

Set 函数是在*公共槽区*中声明的，因此它是一个槽。槽既可以作为成员函数，与其他任何函数一样调用，也可以与信号相连。以下是 setBalance() 槽的实现过程：

```
void BankAccount::setBalance(int newBalance)
{
        if (newBalance != currentBalance) {
                currentBalance = newBalance;
                emit balanceChanged(currentBalance);
        }
}
```

语句
emit balanceChanged(currentBalance);
将发出 balanceChanged() 信号，并使用当前新余额作为其参数。

关键字 emit 类似于 "signals"和 "slots"，由 Qt 提供，并由 C++ 预处理器转换成标准 C++ 语句。

以下示例说明如何连接两个 **BankAccount** 对象：

```
BankAccount x, y;
connect(&x, SIGNAL(balanceChanged(int)), &y, SLOT(setBalance(int)));
x.setBalance(2450);
```

当 x 中的余额设置为 2450 时，系统将发出 balanceChanged() 信号。y 中的 setBalance() 槽收到此信号后，将 y 中的余额设置为 2450。一个对象的信号可以与多个不同槽相连，多个信号也可以与特定对象中的某一个槽相连。参数类型相同的信号和槽可以互相连接。槽的参数个数可以少于信号的参数个数，这时多余的参数将被忽略。

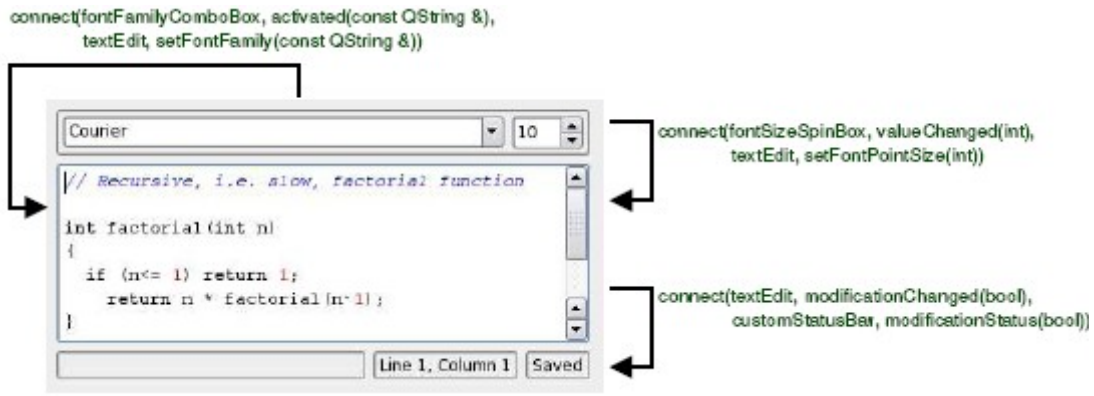

图 4：信号和槽连接的示例。

3.2. 元对象编译器

信号和槽机制是采用标准 C++ 来实现的。该实现使用C++ 预处理器和Qt 所包括的 moc（即：元对象编译器）。元对象编译器读取应用程序的头文件，并生成必要的代码，以支持信号和槽机制。qmake（请参见第 73 页中的“Qt 的构建系统”）生成的 Makefiles 将自动调用元对象编译器。开发人员无需编辑、甚至无需查看生成的代码。

除了处理信号和槽以外，元对象编译器还支持 Qt 的翻译机制、属性系统及其扩展的运行时类型信息。它还使 C++ 程序进行运行时自检成为可能，并可在所有支持的平台上工作。

在线参考
http://doc.trolltech.com/4.2/object.html
http://doc.trolltech.com/4.2/signalsandslots.html
http://doc.trolltech.com/4.2/moc.html

4. GUI 应用程序

使用 Qt 可以轻松快速地构建流行的 GUI 应用程序，通过手工编码或使用 Qt 的可视设计工具 Qt Designer 即可完成。

Qt 提供了创建现代 GUI 应用程序所需的所有类和函数。使用 Qt，可以创建“主窗口”样式的应用程序（其中心区域周围含有菜单栏、工具栏以及状态栏）和“对话框”样式的应用程序（这些应用程序使用按钮或选项卡来显示选项和信息）。Qt 既支持 SDI（单一文档界面）、又支持 MDI（多文档界面）。Qt 还持拖放操作以及剪贴板。

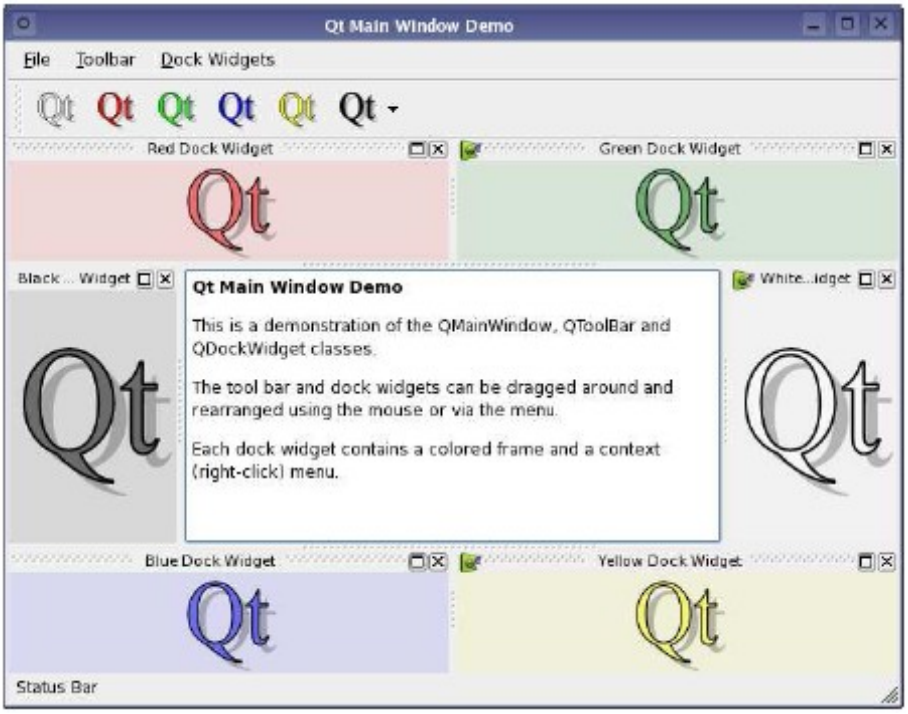

图 5：“Qt 4 主窗口演示”显示了一个应用程序主窗口，该窗口中包含主菜单、工具栏、停靠窗口和一个中心窗体。

工具栏可以在工具栏区域内移动，也可以拖到其他区域，还可以作为工具板浮动。此功能为内建功能，无需任何附加代码。当然，如有需要，编程人员可以对工具栏的行为加以限制。

Qt 使编程变得更简易。例如，如果某一菜单选项、工具栏按钮以及键盘快捷键都执行同一操作，那么只需为该操作编写一次代码。

Qt 还支持消息框和所有标准对话框，这样方便应用程序向用户提出问题，并让用户选择文件、文件夹、字体和色彩。实际上，若要显示消息框或标准对话框，则只需编写一行语句，该语句使用 Qt 某一静态函数即可。

使用系统注册表或文本文件，Qt 可以采用与平台无关的方式存储应用程序的设置，

从而记录项目（例如，用户首选项、最近使用的文件、窗口以及工具栏位置和大小），供稍后使用。

4.1. 主窗口类

4.1.1. 主窗口

QMainWindow 类为典型应用程序的主窗口提供了一个框架。主窗口包含一系列标准控件。主窗口的顶部是一个菜单栏，菜单栏下面是工具栏，整个工具栏区域位于窗口顶部、左侧、右侧以及底部。状态栏则位于底部工具栏区域下的主窗口区域中。工具提示和“这是什么?”帮助为用户界面元素提供了气球帮助。

对于 SDI 应用程序，**QMainWindow** 的中心区域可以包含任何窗体。例如，文本编辑器可以使用 **QTextEdit** 作为中心窗体：

```
QTextEdit *editor = new QTextEdit(mainWindow);
mainWindow->setCentralWidget(editor);
```

对于 MDI 应用程序，中心区域通常放置 **QWorkspace** 窗体。

4.1.2. 菜单

QMenu 窗体以垂直列表的方式向用户展示菜单项目。菜单既可以单独显示（例如，上下文弹出菜单）、可以显示在菜单栏上、也可以成为另一弹出菜单的子菜单。菜单可以具有tear-off handle。

每个菜单项目都可拥有图标、复选框和快捷键。通常，菜单项目与动作相对应（例如，“保存”）。分隔符项显示为一条横线，可以用来将相关动作采用可视方式集中在一起。下面我们举例说明如何创建带有“新建”、“打开”和“退出”菜单项目的“文件”菜单：

```
QMenu *fileMenu = new QMenu(this);
fileMenu->addAction(tr("&New"), this, SLOT(newFile()), tr("Ctrl+N");
fileMenu->addAction(tr("&Open..."), this, SLOT(open()), tr("Ctrl+O"));
fileMenu->addSeparator();
fileMenu->addAction(tr("E&xit"), qApp, SLOT(quit()), tr("Ctrl+Q"));
```

选中菜单项目时，系统将执行对应的槽。请注意：在此情况下，tr() 函数将采用用户的本地语言来获取菜单文本（请参见第 49 页中的“国际化”）。

QMenuBar 类实现了菜单栏。菜单栏会自动位于父窗体（通常是指 **QMainWindow**）的顶部，如果父窗口不够宽，则其内容将分为几行排列。Qt 的布局管理器可以管理任何菜单栏。在 Macintosh 系统中，菜单栏出现在屏幕顶部。下面我们举例说

明如何创建带有“文件”、“编辑”和“帮助”菜单的菜单栏：

```
QMenuBar *menuBar = new QMenuBar(this);
menuBar->addMenu(tr("&File"), fileMenu);
menuBar->addMenu(tr("&Edit"), editMenu);
menuBar->addMenu(tr("&Help"), helpMenu);
```

Qt 的菜单十分灵活，是整个集成“动作”系统(action system)的一个组成部分（请参见下一页中的“动作”）。您可启用或禁用任何动作，也可将操作动态添加至菜单中，稍后再移除该动作。

4.1.3. 工具栏

工具栏包含各种按钮以及用户执行操作相关的其他控件。可以随意在主窗口中央区域的顶部、左侧、右侧以及底部移动工具栏。任何工具栏均可从工具栏区域中拖出，并可作为独立的工具板浮动。

QToolButton 类实现了带有图标、不同风格的边框以及可选文字标签的工具栏按钮。切换（toggle）工具栏按钮负责开关功能， 其他工具栏按钮则负责执行命令。活动模式、禁用模式、启用模式以及开/关状态均使用不同图标表示。如果只有一个图标，那么 Qt 将使用可视标记（例如，以灰色显示禁用的按钮）自动区分上述状态。工具栏按钮还可触发弹出菜单。

通常，工具栏按钮并列排在工具栏区域内。一个应用程序可以拥有任意数量的工具栏，用户可以在四周自由移动工具栏。工具栏几乎可以包含任意控件类型；例如，它经常包含 **QComboBox** 和 **QSpinBox** 控件。

4.1.4. 动作

若要执行某一特定动作，应用程序通常向用户提供了几种不同的执行方式。例如，大多数应用程序经常会提供以下几种方式：从“文件”菜单中调用“保存”动作、从工具栏（工具栏中的软盘形状按钮）以及快捷键 (Ctrl+S)。**QAction** 类负责封装此概念。它支持编程人员在单一位置定义动作。

以下代码使用菜单项目、工具栏按钮以及键盘快捷键来实现“保存”动作，所有这些方式都可以使用工具提示和“这是什么？”信息提供交互式帮助。

```
QAction *saveAct = new QAction(tr("Save"), saveIcon, tr("&Save"), tr("Ctrl+S"),this);
connect(saveAct, SIGNAL(activated()), this, SLOT(save()));
saveAct->setWhatsThis(tr("Saves the current file."));
saveAct->addTo(fileMenu);
saveAct->addTo(toolbar);
```

为了避免重复劳动，可以使用 **QAction** 确保菜单项目的状态与相关工具栏按钮的状态保持同步，还可确保在必要时显示交互式帮助。禁用某一动作将意味着禁用任何对应的菜单项目和工具栏按钮。同样，如果用户单击工具栏中的切换（toggle）按钮，那么也会切换对应的菜单项目。

4.1.5. 停靠窗口

停靠窗口是指用户可以在工具栏区域内或区域间随意移动的窗口。用户可以对停靠窗口解锁，使该窗口浮在应用程序顶部，也可以使窗口最小化。停靠窗口是由 **QDockWidget** 类提供的。通过 **QDockWidget** 实例化并添加窗体，可以创建自定义停靠窗口。如果停靠窗口占据水平区域（例如，在主窗口的顶部），那么窗体将会横向排列；如果占据垂直区域（例如，在主窗口的左侧），那么窗体将会纵向排列。

有些应用程序（包括 *Qt Designer*（请参见第 24 页）和 *Qt Linguist*（请参见第 50 页））经常使用停靠窗口。**QMainWindow** 为操作者提供保存并恢复停靠窗口和工具栏的位置的功能，这样，应用程序可以轻松恢复用户首选工作环境。

图 6：以 Plastique 样式显示的 **QMessageBox** 和 **QProgressDialog**。

4.1.6. 对话框

对于特定操作，大多数 GUI 应用程序使用对话框来与用户交互。Qt 为大多数常见任务提供了现成的对话框类和便利功能。我们稍后将提供一些 Qt 标准对话框的屏幕快照。Qt 还为色彩选择和打印选项提供了标准对话框。

对话框提供以下三种操作方式：

1. *模式* 对话框。此类对话框阻止向同一应用程序的其他可视窗口进行输入。用户必须先关闭此对话框，然后才可访问该应用程序中的任何其他窗口。
2. *非模式* 对话框。此类对话框的运行与其他窗口无关。
3. *半模式* 对话框。此类对话框立即将控制权返回给调用程序。从用户角度来看，这些对话框类似于模式对话框，但它们允许应用程序继续进行处理。这对进度对话框而言十分有用。

通常，模式对话框按如下方式使用：

```
OptionsDialog dialog(&optionsData);
if (dialog.exec()) {
        do_something(optionsData);
}
```

QFileDialog 是一种高级文件选择对话框。它可以用来选择一个或多个本地文件或远程文件（例如，使用 FTP 来选择）， 还包括诸如文件重命名和创建目录等功能。与大多数 Qt 对话框类似，**QFileDialog** 的大小可以重新调整，这样易于查看较长的文件名和目录。您可以将应用程序设置成自动使用 Windows 和 Macintosh 系统中的本地文件对话框。

Qt 还提供了其他常见的对话框：**QMessageBox** 可以向用户提供信息，或者向用户显示简单的选择选项（例如：“是”或“否”）；**QProgressDialog** 则可显示进度条和“取消”按钮。

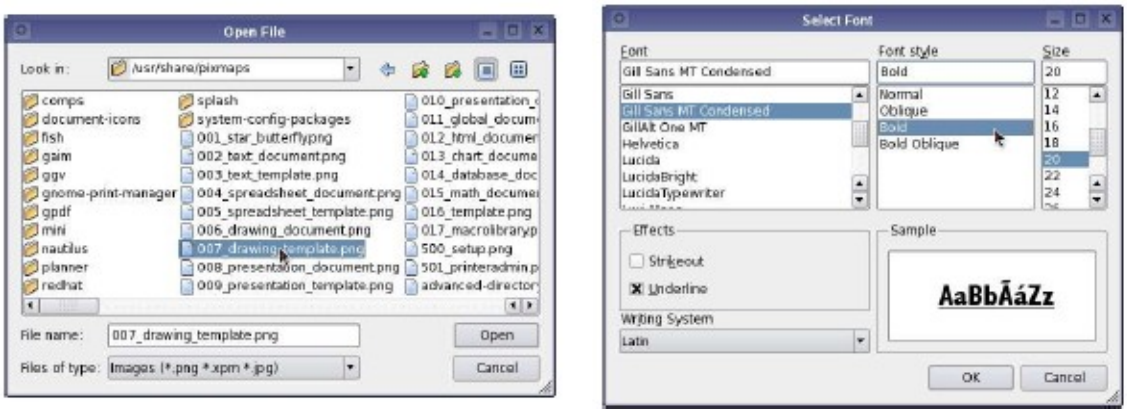

图 7：以 Plastique 样式显示的 **QFileDialog** 和 **QFontDialog**，在 Windows 和 Mac OS X系统中会用系统本地对话框代替。

编程人员可以通过从 **QWidget** 派生子类或者从 **QDialog** 派生子类，或者使用 Qt 提供的任一标准对话框创建自己的对话框。另外，*Qt Designer* 还提供了对话框模板，帮助开发人员从头开始进行设计。

4.1.7. 交互式帮助

现代应用程序经常使用不同形式的交互式帮助来阐明用户界面元素的目的。Qt 提供了两种机制来提供简单的帮助消息：工具提示和“这是什么？”帮助。

工具提示是黄色的小方框，如果鼠标指针停留在某一窗体的上方，则它会自动显示。由于工具栏按钮几乎不与文本标签一起显示，因此，工具提示经常用来阐明工具栏按钮的目的。下面，我们举例说明如何设置“保存”工具栏按钮的工具提示。

```
QToolTip::add(saveButton, tr("Save"));
```

另外，在显示每个工具提示的同时，还可以在主窗口状态栏中显示更长的文本。

“这是什么？”帮助与工具提示类似，但它必须要用户请求（例如，按 **Shift+F1** 键，然后单击某一窗体或菜单选项）才会显示。通常，“这是什么？”帮助比工具提示要更为详细。下面，我们举例说明如何为“保存”工具栏按钮设置“这是什么？”文本。

```
QWhatsThis::add(saveButton, tr("Saves the current file."));
```

QToolTip 和 **QWhatsThis** 类也可以用来实现更专业的操作，例如，提供动态工具提示，根据鼠标所在窗体的位置显示不同的文本。

在线帮助浏览器可以提供某一应用程序的更多详细信息。**QAssistantClient** 类则支持应用程序使用 *Qt Assistant*（请参见第 24 页）根据用户请求来显示用户指南的相关页面。

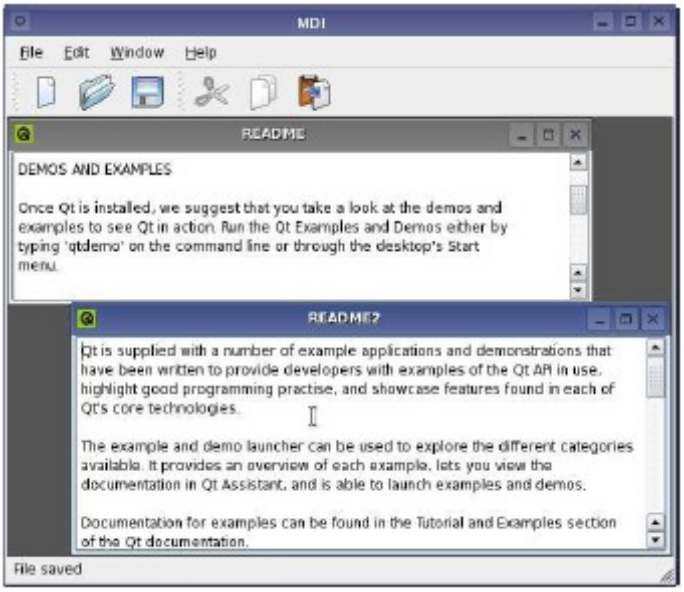

图 8：某一包含 **QWorkspace** 窗体的应用程序主窗口所提供的多文档界面。

4.1.8. 多文档界面

QWorkspace 类支持多文档界面 (MDI) 功能，通常，该类可作为 **QMainWindow** 的中心窗体使用。

任何一类窗体都可以作为**QWorkspace** 的子窗体。这些子窗体的边框与顶层窗体的边框的绘制方式类似，子 MDI 窗体的功能（例如：显示、隐藏、最大化以及设置窗口标题）与普通顶层窗体相同。

QWorkspace 提供了排列方式，例如，*层叠* 和*平铺*。如果某一子窗体超出 MDI 区域，则可将滚动条设置成自动显示。如果将子窗体最大化，则菜单栏中将显示边框按钮（例如，“最小化”按钮）。

4.2. 设置

使用 **QSettings** 类，可以将用户设置以及其他应用程序设置轻松存储在磁盘中。在 Windows 中，**QSettings** 利用系统注册表来存储；在 Mac OS 中，它使用系统的 CFPreferences 机制来存储；在其他平台中，设置则存储在文本文件中。

每个设定都用关键词存储。例如：关键词
/SoftwareInc/Zoomer/RecentFiles

可能包含最近使用过的文件列表。您也可存储布尔值、数字、Unicode 字符串以及 Unicode 字符串列表。

各种 Qt 数据类型均可与 **QSettings** 无缝使用，这些类型在存储时将被序列化，稍后由应用程序读取。有关 Qt 数据类型序列化的详细信息，请参见第 52 页中的“文件处理”。

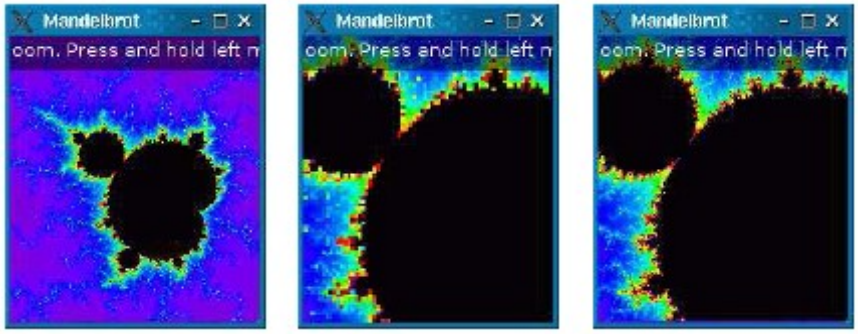

图 9：Qt 4 Mandelbrot 示例表明执行耗时冗长的任务时，如何使用线程保持用户界面的响应。

4.3. 多线程

GUI 应用程序经常使用多个线程：一个线程保持用户界面的响应，其他线程则执行耗时冗长的活动，例如：读取大型文件并执行复杂的计算。您可以配置 Qt 来支持多线程，并提供类来表示线程、互斥锁、信号灯、线程全局存储，另外还提供支持不同锁定机制的类。许多 Qt 类都可重入，且Qt 提供的许多功能是线程安全的。

Qt 4 的元对象系统支持不同线程中的对象使用信号和槽进行通信，这样开发人员可以创建单线程应用程序，这些应用程序稍后可以调整为多线程，而无需再重新设计。另外，组件之间可以通过相互发送事件的方式实现跨线程的通信。也可以在线程之间传输特定类型的对象。

4.4. 桌面集成

使用 Qt 的桌面集成类，可以扩展应用程序，使其与用户桌面环境所提供的服务交互。这些类包括 **QSystemTrayIcon**（运行时间较长的应用程序经常使用此类，以便在系统托盘中提供一致的指示器）和 **QDesktopServices**（使用此类控制诸如mailto: 和http://URLs的资源由每个平台中最适合的应用程序来处理）。

QDialogButtonBox 是 Qt 标准对话框中所使用的一个专业化的容器窗体，它可以根据每个平台的风格准则来排列按钮。自定义对话框中也可以使用此部件。

在线参考
http://doc.trolltech.com/4.2/qt4-mainwindow.html
http://doc.trolltech.com/4.2/threads.html
http://doc.trolltech.com/4.2/desktop-integration.html

5. Qt Designer

Qt Designer 是 Qt 应用程序的一个图形用户界面设计工具。应用程序可以完全使用代码编写，也可以使用 Qt Designer 加快开发过程。Qt Designer 的架构以组件为基础，这样开发人员便可以使用自定义的窗体和扩展程序来扩展 Qt Designer，甚至还可将它集成至各种开发环境中。

使用 Qt Designer 设计窗体十分简单。开发人员只需将控件从工具框拖到窗体，然后使用标准编辑工具来选择、剪切、粘贴窗体并重新调整大小即可。然后，可以使用属性编辑器来更改每个控件的属性。窗体的大小和精确位置无关紧要，开发人员只需选中窗体，并对它们运用布局即可。例如，您可以选中一些按钮控件，并通过选择“横向排放”选项将它们并列排放。采用这种方法，可以使设计更快速，设计完成后，控件会根据最终用户需要的窗体大小正确缩放。有关 Qt 自动布局的详细信息，请参见第 52 页中的“布局”。

Qt Designer 消除了用户界面设计中耗时冗长的“编译、链接和运行”周期。这样易于更正或更改设计。使用 Qt Designer 的预览选项，开发人员可以查看设计的窗体在其他样式下的显示效果；例如，Macintosh 开发人员可以预览Windows 样式的窗体。

Windows 平台的商业许可证持有人可以在 Microsoft Visual Studio® 内充分享受到 Qt Designer 用户界面设计所带来的便利。在 Mac OS X 中，开发人员则可在 Apple's Xcode® 环境内使用 Qt Designer。

5.1. 使用 Qt Designer

开发人员既可以创建“对话框”样式的应用程序、又可以创建带有菜单、工具栏、气球帮助以及其他标准功能的“主窗口”样式的应用程序。Qt Designer 提供了多种窗体模板，开发人员可以创建自己的模板，确保某一应用程序或某一系列应用程序界面的一致性。编程人员可以创建自己的自定义窗体，这些窗体可以轻松与 Qt Designer 集成。

Qt Designer 支持采用基于窗体的方式来开发应用程序。窗体是由用户界面 (.ui) 文件来表示的，这种文件既可以转换成 C++ 并编译成一个应用程序，也可以在运行时加以处理，从而生成动态用户界面。Qt 的构建系统（请参见第 69 页）能将用户界面的编译构建过程自动化，使设计过程更轻松。

5.2. Qt Assistant（Qt 助手）

Qt Designer 的在线帮助是由 *Qt Assistant* 应用程序提供的。*Qt Assistant* 可以显示

整个 Qt 的文档集，其运行方式类似于 Web 浏览器。但与 Web 浏览器不同的是，Qt Assistant 采用了一种高级索引算法，可以全文本快速搜索所有相关文档。

Qt 的参考文档包括大约 2,200 页 HTML，记录了 Qt 的所有类和工具，并概述了 Qt 编程的各个方面。

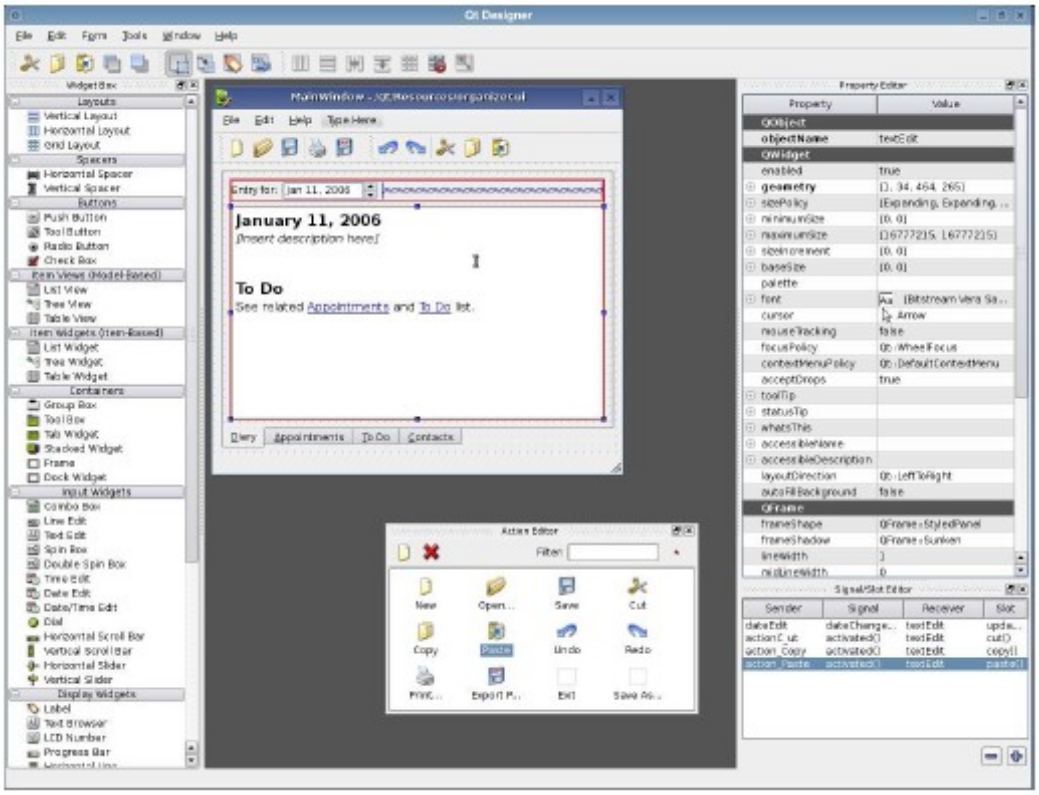

图 10：停靠窗口模式中的 *Qt Designer*。

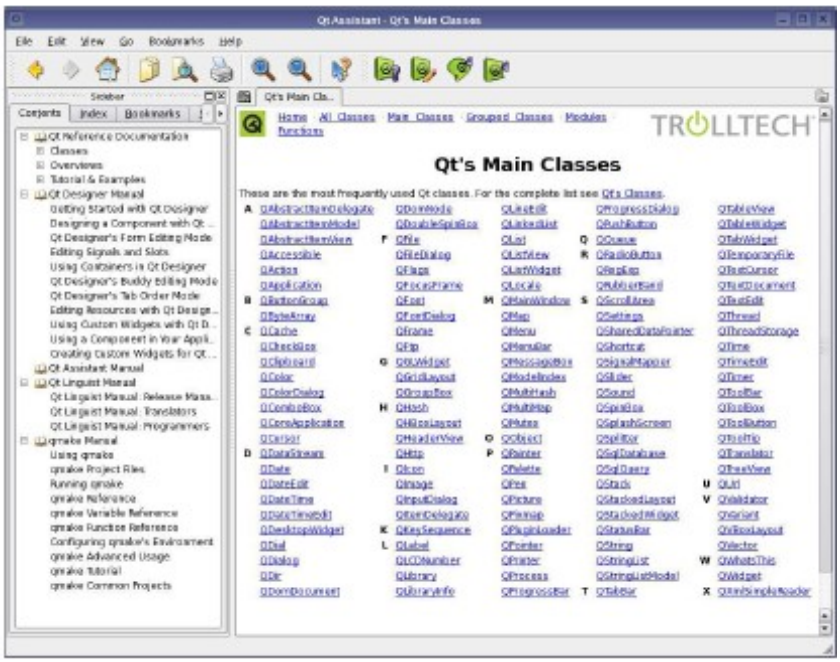

图 11：*Qt Assistant* 所显示的某一页 Qt 4 文档。

开发人员可以将 *Qt Assistant* 部署为自有应用程序和文档集的帮助浏览器。使用 **QAssistantClient** 类，可以集成 *Qt Assistant*。*Qt Assistant* 使用 **QTextEdit** 来显示 Qt 的 HTML 参考文档；如有必要，开发人员也可以使用此类来直接实现自己的帮助浏览器。**QTextEdit** 支持 HTML 4.0 的子集，它与 Qt 提供的富文本（RichText）类一起使用时，也可以用来显示其他格式的文档（请参见第 41 页中的“富文本处理”）。

5.3. GUI 应用程序示例

“类层次结构”应用程序是一个典型的对话框式的应用程序，在此程序中，用户可以选择一些选项（在此示例中是指选择路径），然后可以根据选项执行某些处理。

以下是该应用程序的完整代码。该窗体使用 *Qt Designer* 设计，并存储在 .ui 文件中。uic 将 .ui 文件转换成 C++，这样使开发人员可专心于应用程序的功能。

尽管使用 *Qt Designer* 后，设计用户界面会变得更轻松，但通常还需要一些附加的应用程序逻辑， 通常，开发人员可以通过将 uic 生成的用户界面进行派生类并实现新功能的方式来提供附加功能。

对话框的构造函数仅建立 Qt Designer 提供的用户界面，并为树状窗体创建两个标签。

```
ClassHierarchy::ClassHierarchy(QWidget *parent)
: QDialog(parent)
{
```

```
    setupUi(this);
    QStringList headings;
    headings < < tr("Class") < < tr("Source file");
    treeWidget->setHeaderLabels(headings);
}
```

on_<name>_clicked() 函数是指调用 setupUi() 时、自动从对话框中的按钮连接至信号的所有槽。以下槽仅更改列表控件中的项目：

```
void ClassHierarchy::on_addSearchPathButton_clicked()
{
    QString path = QFileDialog::getExistingDirectory(
        this, "Select a Directory", QDir::home().path());
    if (!path.isEmpty() &&
        searchPathBox->findItems(path, Qt::MatchExactly).count() == 0)
        searchPathBox->addItem(path);
}

void ClassHierarchy::on_removeSearchPathButton_clicked()
{
    delete searchPathBox->takeItem(searchPathBox->currentRow());
}
```

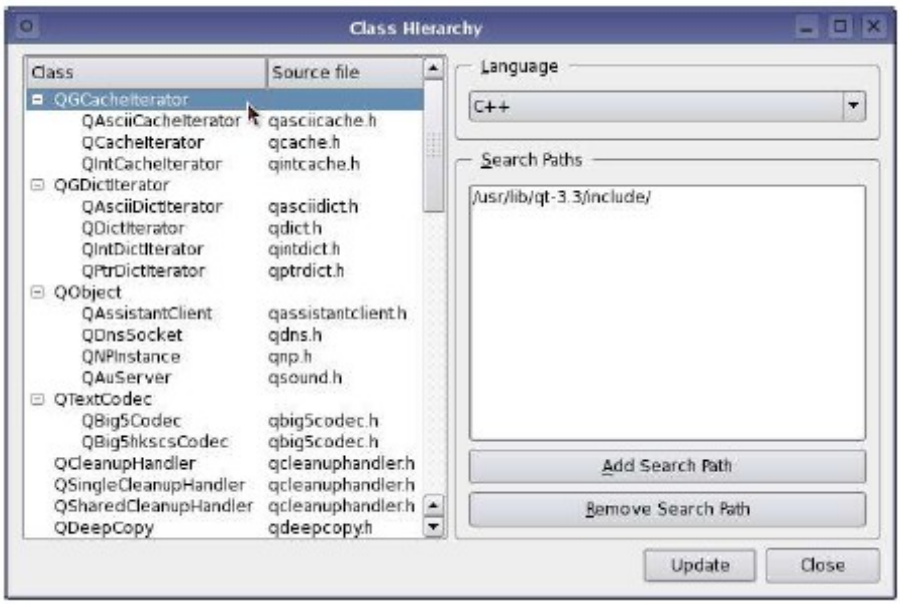

图 12：使用 *Qt Designer* 创建的简单类层次结构应用程序

on_updateButton_clicked() 槽使用在文件中找到的类（该类名与相关模式匹配）来重新填充树状窗体：

```
void ClassHierarchy::on_updateButton_clicked()
{
    QStringList fileNameFilter;
    QRegExp classDef;
    if (language->currentText() == "C++") {
        fileNameFilter < < "*.h";
        classDef.setPattern("\\bclass\\s+([A-Z_a-z0-9]+)\\s*"
```

```
                "(?:\\{|:\\s*public\\s+([A-Z_a-z0-9]+))");
        } else if (language->currentText() == "Java") {
                fileNameFilter < < "*.java";
                classDef.setPattern("\\bclass\\s+([A-Z_a-z0-9]+)\\s+extends\\s*"
                    "([A-Z_a-z0-9]+)");
        }

        classMap.clear();
        treeWidget->clear();

        for (int i = 0; i < searchPathBox->count(); i++) {
                QDir dir = searchPathBox->item(i)->text();
                QStringList names = dir.entryList(fileNameFilter);

                for (int j = 0; j < names.count(); j++) {
                        QFile file(dir.filePath(names[j]));
                        if (file.open(QIODevice::ReadOnly)) {
                                QString content = file.readAll();
                                int k = 0;
                                while ((k = classDef.indexIn(content, k)) != -1) {
                                        processClassDef(classDef.cap(1), classDef.cap(2), names[j]);
                                        k++;
                                }
                        }
                }
        }
}
```

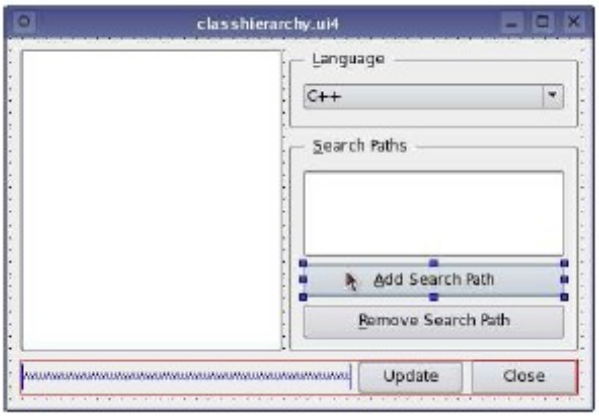

图 13：使用 *Qt Designer* 编辑类层次结构窗口

在*Qt Designer* 中，让“关闭”按钮与对话框的 accept() 槽相连。

以下列出完整的私有程序，这可以处理所有的类并将其插入至树状窗体中：

```
void ClassHierarchy::processClassDef(const QString &derived,
const QString &base, const QString &sourceFile )
{
    QTreeWidgetItem *derivedItem = insertClass(derived, sourceFile);
    if (!base.isEmpty()) {
        QTreeWidgetItem *baseItem = insertClass(base, "");
        if (derivedItem->parent() == 0) {
```

```
            treeWidget->takeTopLevelItem(
            treeWidget->indexOfTopLevelItem(derivedItem));
            baseItem->addChild(derivedItem);
            derivedItem->setText(1, sourceFile);
        }
    }
}

QTreeWidgetItem *ClassHierarchy::insertClass(const QString &name,
const QString &sourceFile)
{
    if (classMap[name] == 0) {
        QTreeWidgetItem *item = new QTreeWidgetItem(treeWidget);
        item->setText(0, name);
        item->setText(1, sourceFile);
        treeWidget->setItemExpanded(item, true);
        classMap.insert(name, item);
    }
    return classMap[name];
}
```

上述示例阐述了如何将用户界面“编译”进应用程序。也可以在运行时从 .ui 文件中动态生成用户界面，这样开发人员可以创建可单一执行的应用程序，并针对不同用途对这些应用程序定制化。

对于在使用 *Qt Designer* 时创建和编写应用程序而言，使用哪些工具来创建并编辑这些应用程序的源代码则取决于每个开发人员的个人喜好；有的人喜欢利用 *Qt Designer* 提供的集成功能在 Microsoft Visual Studio 或 Apple's Xcode 开发环境内来开发。

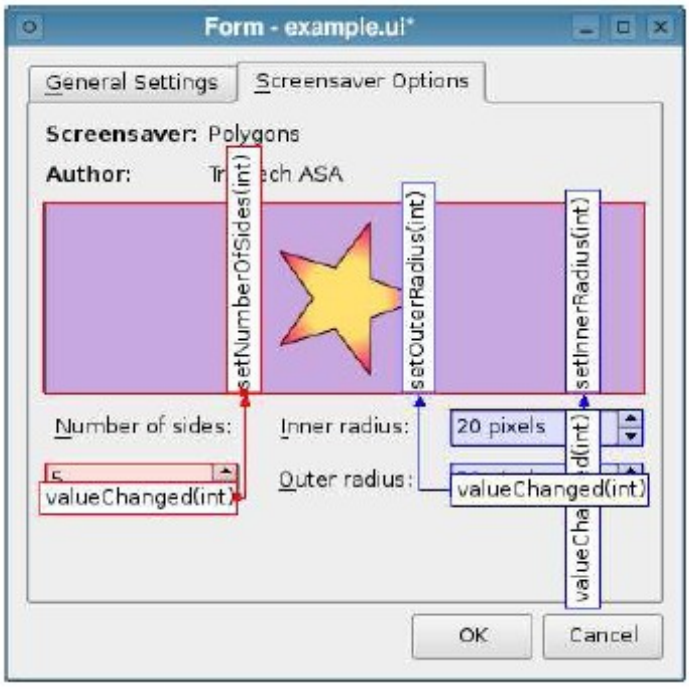

图 14：“世界时钟”自定义控件作为 *Qt Designer* 的一个小控件打包并添加至窗体中。该窗体可提供自定义的信号和槽，窗体中的其他控件也相应可以使用这些信号和槽。

5.4. 扩展 Qt Designer

Qt Designer 以基于组件的架构为基础，旨在支持开发人员使用自定义的组件来扩展其用户界面和编辑工具。此外，由于 *Qt Designer* 应用程序具有模块化的特性，因此可以在集成开发环境（例如，Microsoft Visual Studio 和 KDevelop）中使用*Qt Designer* 用户界面设计功能。

QtDesigner 模块一共提供了 20 个类来处理 .ui 文件并扩展 *Qt Designer*。其中大多数类支持第三方自定义该应用程序本身的用户界面。

用于自己开发使用的第三方控件和自定义控件可以轻松集成至 *Qt Designer* 中。若要在 *Qt Designer* 内使用现有控件，则只需实现相应的接口来提供默认控件属性并构建该控件的新实例，将该控件编译为插件即可。使用类似于第 58 页所述“插件”的宏可将插件界面导入至 *Qt Designer* 中。

在线参考
http://doc.trolltech.com/4.2/designer-manual.html
http://www.trolltech.com/products/qt/vs-integration.html
http://doc.trolltech.com/4.2/qtdesigner.html
http://doc.trolltech.com/4.2/examples.html#qt-designer-examples

6. 2D 和 3D 图形

Qt 对 2D 和 3D 图形提供了强大的支持。Qt 的 2D 图形类支持点阵图形和向量图形，可以加载并保存各种图形格式，并可将文本和图形导出为 Portable Document Format (PDF) 文件。Qt 可以绘制转换后的 Unicode 文本和 Scalable Vector Graphics (SVG) 图纸，并可为需要交互的应用程序提供功能全面的 Canvas。Qt 实际上是与平台无关的 OpenGL 编程的一个标准 GUI 框架。

图形是使用与设备无关的绘图对象来绘制的，这样开发人员可以重新使用同一代码在不同设备（Qt 中由Paint Device描述相应的不同设备，请参见第 31 页中的“绘图”）中渲染图形。使用这一方法，可以确保 Qt 所支持的每一设备都可提供一系列功能强大的绘图操作。

需要交互 *Canvas* 的图形应用程序可以利用“图形视图”框架（请参见第 35 页）来管理并渲染含有大量交互项目的场景，如有必要，还可对多个视图进行管理。

QColor 类支持设备无关的色彩。色彩是由 ARGB、AHSV、 ACMYK 值或常用名称（例如“天蓝”）指定的。**QColor** 的色彩通道为 16 位宽，还使用可选的透明度指定色彩；Qt 自动将请求的色彩分配在系统调色板中，或者在色彩受限的显示中使用类似色彩。

6.1. 绘图

QPainter 类提供了一个平台无关的 API，以便在控件和其他绘图设备中进行绘图。它提供了许多低级功能和高级功能，例如：转换和剪辑。Qt 的所有内建的控件均可使用 **QPainter** 来绘制其本身。实现自定义控件时，编程人员总是要使用 **QPainter**。

QPainter 提供了许多标准功能绘制点、线、椭圆、弧、贝塞尔（Bezier） 曲线以及其他原始图形。此外，**QPainter** 还支持更为复杂的绘图操作，比如：支持多边形和向量路径，允许预先准备详细绘制操作并使用单一函数调用进行绘图。另外，您还可以使用笔刷直接绘制文本，也可将文本合并至某一路径中，供稍后使用。

Qt 的绘图系统还提供了许多高级功能，以提高整体渲染质量。

- 使用 Alpha 混合处理和 Porter-Duff 组合模式，开发人员可以绘制高级图形，并可对整个屏幕的输出结果进行高级控制。
- Qt 针对图形原语和文本提供了反锯齿（Anti-alias）功能，使用此功能，可以模拟高分辨率显示原有的低分辨率图形。
- 使用 Qt 的线、射线以及锥型梯度填充功能，可以轻松创建更详细的图形（例如，3D 楔形按钮），而无需开发人员耗费过多的时间和精力。

图 15：Qt Affine Transformation变换图例显示了可以如何利用变换来达到想要的效果。

QPainter 使用由长方形、多边形、椭圆以及向量路径组成的区域来支持剪辑。使用标准集合操作可以将多个简单的区域组合在一起，从而创建复杂的区域。

6.2. 图像

QPixmap 和 **QImage** 类支持输入、输出并操作各种不同格式的图像，包括 BMP、GIF、JPEG、MNG、PNG、PNM、XBM 和 XPM。这两个类均可以作为绘图设备使用，并且均可以在交互图形应用程序中使用。另外，它们也可以用来预处理图像，供稍后在用户界面组件中使用。Qt 的许多内建控件（例如，按钮、标签和菜单项目）均可以显示图像。

如果应用程序需要对图像进行快速渲染，那么通常使用 **QPixmap**。而 **QImage** 则更适用于像素操作，并且处理不同色彩深度和像素格式的图像。编程人员可以操作像素和调色板数据，并实行各种变换（例如：旋转和修剪），如有必要，也可用抖动的方法降低色彩深度。Qt 支持“alpha 通道”数据以及色彩数据，这样应用程序可以针对合成图像以及其他目的使用透明度和 Alpha 值混合处理。

使用 Qt 的可扩展插件机制，可以扩展能用上述类使用的图形文件格式范围。

6.3. 绘图设备和打印

QPainter 可以在任何绘图设备上操作。无论在哪台设备上，在任何支持的设备上进行绘图所需的代码都是相同的。

Qt 支持以下绘图设备：

- 所有 **QWidget** 子类都是绘图设备。Qt 使用*backing store*机制来减少绘图过程中出现的闪烁现象。

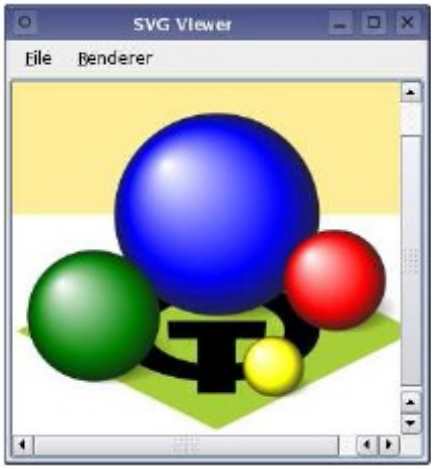

图 16: 可以在Qt 所支持的任何一个绘图设备中使用SVG绘图。

- 所有 **QGLWidget** 子类都是绘图设备，都可以将标准 **QPainter** 调用转换成 OpenGL 调用，这样，含有合适支持的硬件设备均可更快速地处理二维图形。

- **QImage** 是一个与设备无关，色彩深度和像素格式均已指定的图像。它支持使用不同级别的透明度来创建图像，并支持将图像绘制至自定义控件，从而达到特定的效果。

- 作为屏幕上的一个控件，**QPixmap** 与 **QImage** 的属性实质上是相同的。**QPixmap** 通常是一种可以快速并有效访问的系统设备。

- **QPicture** 是指一种可以平滑缩放、旋转并剪裁的向量图像。图片使用绘图命令、而不是像素数据来存储。

- **QPrinter** 表示物理打印机。在 Windows 中，绘图命令将发送至 Windows 打印引擎，该引擎将使用已安装的打印机驱动程序。在 Unix 中，系统编写出 PostScript®，并将其发送至打印后台程序。所有平台均可生成 Portable Document Format (PDF) 和 PostScript 文件，这样应用程序可以创建高质量的文档，使用适当的读取应用程序可以查看。

6.4. 可缩放的向量图形 (SVG)

SVG 是一种基于 XML 的文件格式和语言，通常在相关图形应用程序用来描述基于 Web 的二维图形。Qt 支持 SVG 是以 SVG 1.1 标准和万维网联合会 (W3C) 推荐为基础，另外还提供了许多附加功能，以支持 SVG 1.1 和 1.2 的微型简化版本。

Qt 中的 **QSvgWidget** 和 **QSvgRenderer** 类支持SVG的渲染。这两个类使用 Qt 的绘图系统来支持 SVG 显示和动画，因此可以使用任何一个 **QPaintDevice** 子类，包括 **QImage** 和 **QGLWidget**来绘图。

在 Qt 4.2 中，支持 SVG 格式这一功能已添加至 **QIcon** 类中，这样开发人员便可使用 SVG 来代替图标的位图图像。

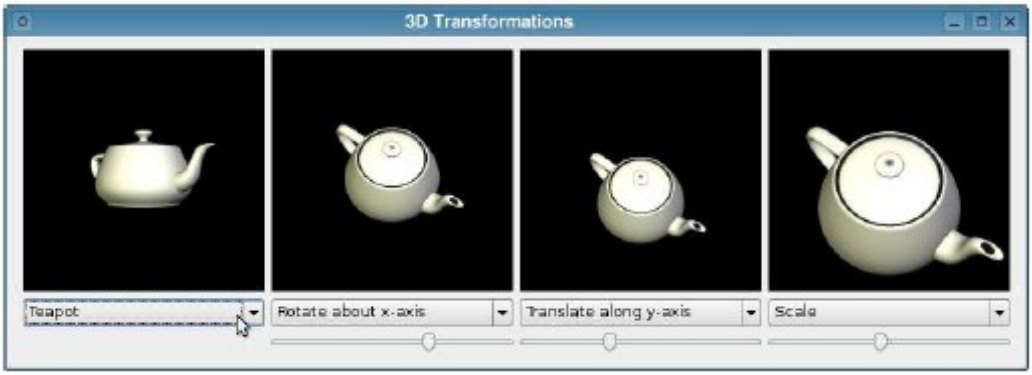

图 17：Qt 应用程序可以使用 OpenGL 将 3D 图像连接至传统的 GUI 控件中。

6.5. 3D 图形

OpenGL 是一种用于渲染 3D 图形的标准 API。Qt 开发人员可以使用 OpenGL 在 GUI 应用程序中绘制 3D 图形。Qt 的 OpenGL 模块可用于 Windows、X11 和 Mac OS X，它使用系统的 OpenGL 库。

若要在 Qt 应用程序中使用启用了 OpenGL 的控件，则开发人员只需继承 **QGLWidget** 类，然后使用标准 OpenGL 函数即可。Qt 提供了许多函数将 **QColor** 值转换成 OpenGL 的色彩格式，有助于开发人员为其应用程序提供始终一致的用户界面。

另外，Qt 还支持在 Qt 应用程序内方便快捷地使用 OpenGL 功能和扩展功能。使用 Qt 的便利函数，可以从 **QImage** 和 **QPixmap** 对象中创建纹理。**QGLPixelBuffer** 和 **QGLFramebufferObject** 类支持像素缓冲对象和帧缓冲对象。如果底层平台支持的话，可以使用 **QGLWidget** 来使用上述缓冲功能。

另外，Qt 还提供了一个 API，它支持使用 OpenGL 创建的 3D 图形集成至 Qt 应用程序中，**QGLWidget** 子类则使用专业绘图引擎将各种 **QPainter** 操作转换成 OpenGL 调用。应用程序可以利用此功能提高相应硬件中的 2D 渲染性能，也可以用OpenGL来覆盖**QPainter**在3D 场景中绘制控件和装饰。

6.6. 图形视图框架

Qt 4.2 针对互动图形应用程序引入了一种功能强大的全新框架，用来管理并显示二维场景中的大量项目。图形视图框架既提供了基于对象的 API（用于将新项目添加至场景中）、又提供了传统的 Canvas 式的 API（它包含创建预定义项目所需使用的便利函数）。

一旦创建后，项目可以按指定方向放置在所需位置，并且可根据场景加以缩放。显示功能和项目管理功能分别在 **QGraphicsView** 和 **QGraphicsScene** 类中实现，这样可以启用各种功能，例如，同一场景中含有多个视图、支持可切换的渲染器等。

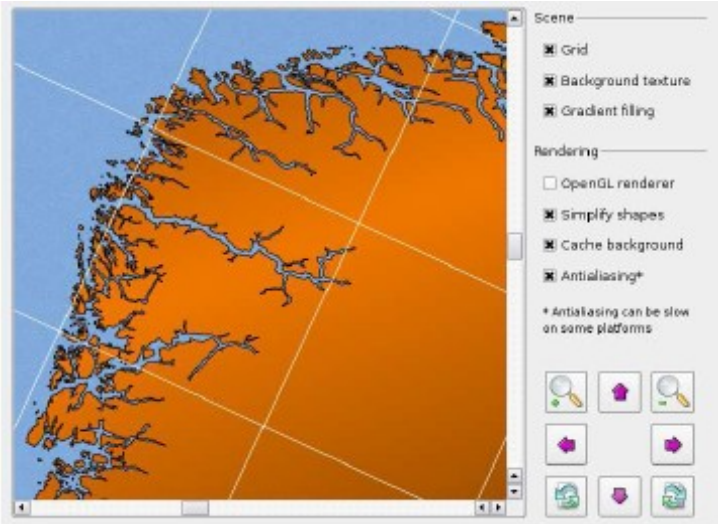

图 18：使用图形视图框架后，创建的图形应用程序既具有高质量的渲染能力、又为用户互动提供了广泛的功能。

只需编写几行代码，即可为基础场景构建一个简单的、含有一个图片项目的场景。

```
QGraphicsScene scene;
scene.addPixmap(QPixmap("konqi.png"));
QGraphicsView view(&scene);
view.show();
```

Qt 提供了许多标准项目类型供您选择，每一种类型都是 **QGraphicsItem** 基础项目类的子类。可以采用继承这些类的方式扩展这些类型，从而提供自定义的项目类型。使用 **QGraphicsItemGroup** 可以将各种项目集中到一起，从而对应用到某一场景的所有部件的变换实现高级控制。每个场景、视图以及项目本身均可提供一系列综合功能，支持不同坐标系统之间方便快捷地转换。标准项目和自定义项目都可被选择和移动，这样可以用最少的代码使项目达到基本交互能力。

图形视图是使用动画理念来设计的：您可以使用 **QGraphicsItemAnimation** 来创建动画对象，创建后，可以根据某一时间点所定义的一系列变换操作来变换这些对象。每个动画项目都是根据 **QTimeLine** 对象来运行的，该对象控制了动画的速率和持续时间。

Qt 提供了具有全面功能的事件模型，专门为图形项目而设计，这些模型仅将事件分

派给需要事件的项目，从而使事件的处理更高效。由于基础项目控制是由框架执行的，因此，如果事件需要项目环境的特定信息，那么项目只需对事件做出响应。针对这种情况，**QGraphicsItem** 提供了一系列标准事件控制器，支持开发人员根据需要实施特定的交互行为。

另外，使用 **QPainter** 对象，可以独立于附加视图，在任何一个 **QPaintDevice** 子类中执行绘图操作，**QGraphicsScene** 对象均可以渲染对象内容。这样可以自动支持直接打印至打印机或 PDF 文件。

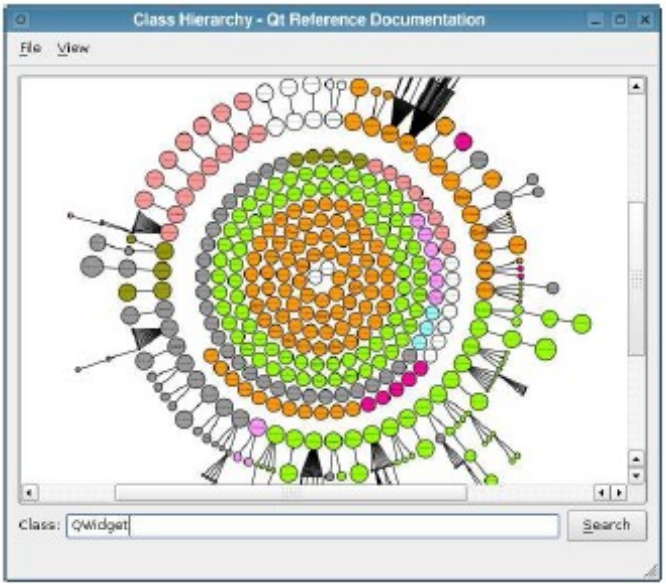

图 19：图形视图框架是图形和图表应用程序的理想之选

例如，以下函数使用用户在打印对话框中输入的选项来打印整个场景：

```
void MainWindow::print()
{
    QPrinter printer(QPrinter::HighResolution);
    QPrintDialog dialog(&printer, this);
    if (dialog.exec() == QDialog::Accepted) {
        QPainter painter;
        painter.begin(&printer);
        scene->render(&painter);
        painter.end();
    }
}
```

有些标准项目可以将 Qt 其他部件中所提供的功能带到图形视图框架中。**QGraphicsTextItem** 可以用来支持纯文本和富文本，允许用户对使用缩放、旋转以及其他操作进行过变换的文档进行编辑。**QGraphicsSvgItem** 将SVG图形显示为场景中的项目。**QGraphicsPathItem** 支持形状和路径，图像以 **QGraphicsPixmapItem** 对象的形式添加至场景中。

在线参考
http://doc.trolltech.com/4.2/qt4-arthur.html
http://doc.trolltech.com/4.2/qpainter.html
http://doc.trolltech.com/4.2/qtsvg.html
http://doc.trolltech.com/4.2/opengl.html
http://doc.trolltech.com/4.2/graphicsview.html

7. 项目视图

Qt 的项目视图窗体为显示并修改大量数据提供了标准的 GUI 控件。基础模型/视图框架将数据存储的方式与数据显示给用户的方式截然分开，这样可以针对同一数据启用各种功能，例如：数据共享、排序和过滤、多视图查看以及多种数据表示方式。

如果要编写能处理大量数据的应用程序，则开发人员通常会借助于“项目视图”窗体来快速有效地显示数据项目。现代化的 GUI 工具中提供了标准项目视图，这些视图包括列表视图（这些视图包含简单的项目列表）、树状视图（包含层次结构的项目列表）以及表视图（这些视图可以提供类似于电子数据表应用程序中的布局功能）。

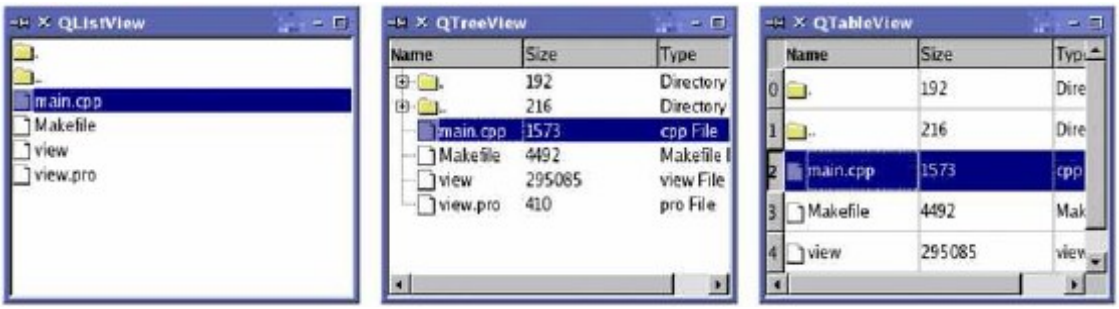

图 20：为项目提供了列表、树以及表标准视图。

Qt 4 的项目视图类可以分为两种不同格式：一种是典型的项目视图控件，另一种是指模型/视图组件。典型的项目视图控件（例如：**QListWidget**、**QTableWidget** 和 **QTreeWidget**）是自我包含的视图，它们可以管理由开发人员外部创建的项目对象。而 **QListView**、**QTableView** 和 **QTreeView** 则是等同于典型项目视图的模型/视图组件。使用这些模型/视图组件，可以采用更清楚的基于组件的方式来处理数据集。

7.1. 标准项目视图

Qt 提供了列表窗体、图标视图、树状窗体和表的标准实现。在同一视图内以及不同视图之间，Qt 完全支持各种拖放操作。Qt 的所有控件也可以与 Qt 的资源系统（请参见第 71 页）完全集成。

Qt 中不同标准对话框使用项目视图类来显示数据（请参见图 7），这些类在 *Qt Designer*、*Qt Assistant* 和 *Qt Linguist* 中使用非常广泛。

典型项目视图所使用的架构通常使用单个对象来封装一批数据，因此可以显示并管理数百个数据项目。现有 Qt 开发人员均十分熟悉这一方法，这样为快速构建丰富的用户界面提供了一条便捷的方法，可以轻松处理大量数据。

就一致性和可靠性而言，典型项目视图是以 Qt 的模型/视图框架为基础建立的，而这正提供了一条更灵活、可定制的方法来处理项目视图数据。

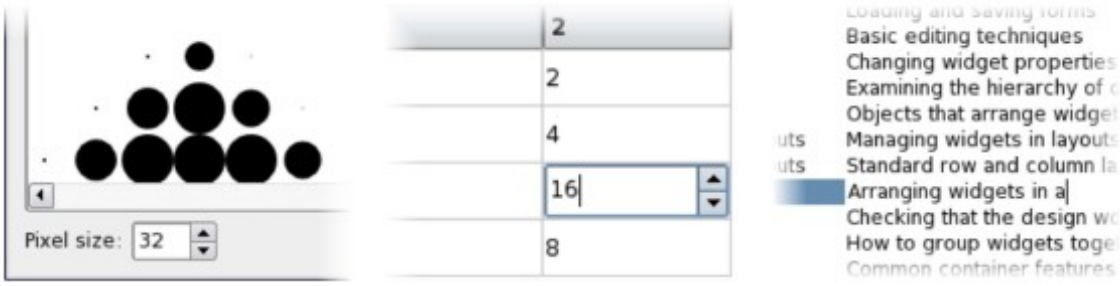

图 21： “模型/视图”框架的面向组件的架构使用户更易于自定义项目视图。

7.2. Qt 的模型/视图框架

Qt 提供的模型/视图框架实际上是著名的*模型-视图-控制器* 模式的变体，专门适用于 Qt 项目视图。在这一方式中，模型用来向其他组件提供数据，而视图则向用户显示数据项目，专门的控制程序则负责显示和编辑流程中的各个方面。

模型是指围绕数据源的包装器，编写包装器的目的是符合 **QAbstractItemModel** 提供的标准接口。使用这些接口，无论原始数据源是什么，从 **QAbstractItemView** 中派生出来的控件均可访问通过该模型提供的数据。

采用数据本身与其表示方式分离的方法，相比于传统的项目试图，有以下改进：

- 由于模型提供了访问数据的标准接口，因此，可以独立于其他组件设计并编写模型，如有必要，还可替换模型。
- 从模型中获得的数据可以在各视图之间实现共享。使用这一功能，应用程序可以基于同一数据集提供多个视图，并可用不同表示方法显示数据。
- 根据用户的需求和期望，选择的数据可以在不同视图之间共享或独立存在。
- 对于标准列表、树和表视图，大多数显示是由委派程序来执行的。这样易于自定义各种用途的视图，而无需编辑大量新代码。
- 通过使用*代理模型*，模型所提供的数据均可先转换，然后再提供给视图。这样一来，应用程序可以提供排序和过滤功能，而且这一功能可在多个视图之间实现共享。

另外，针对非数据库的开发人员，Qt 的 SQL 模型（请参见第 44 页）可以利用模型/视图系统更简便地集成数据库。

在线参考
http://doc.trolltech.com/4.2/model-view-programming.html
http://doc.trolltech.com/4.2/examples.html#item-view-examples

8. 文本处理

Qt 提供了功能强大的文本编辑器控件，使用此控件，用户可以创建并编辑富文本文档、支持导入并导出 HTML，并可准备要打印的文档。开发人员完全可以访问该编辑器所使用的基础文档结构，这样可以从应用程序内操作文档结构和文档内容。

通常，富文本文档包含了字体、颜色和大小各异的若干段落。另外，文本中也许含有列表和表格，也许会通过使用框架使文本与文档主体分离。开发人员通过富文本 API 可以设置若干属性，从而可以精确调整每个文档元素的外观。

*Qt Assistant*中，*“Qt 4* 文本编辑演示”中以及 *Qt Designer* 的文本编辑对话框中均可以看见 Qt 的富文本显示功能。

8.1. 富文本编辑

在 Qt 中，**QTextBrowser** 和 **QTextEdit** 控件负责显示和编辑交互式的富文本。这两个窗体全面支持 Unicode，它们以 **QTextDocument** 提供的结构化文档表示为基础，因此无需使用中间标记语言来创建富文本。另外，**QTextDocument** 还支持导入和导出 HTML，支持撤消/恢复功能（包括支持一组操作）以及资源处理。

另外，用来显示 **QTextEdit** 窗体中富文本的布局系统，可以根据从 **QPrintDialog** 中获得的信息、将文档格式化为一系列页面，以便使用 **QPrinter** 打印。

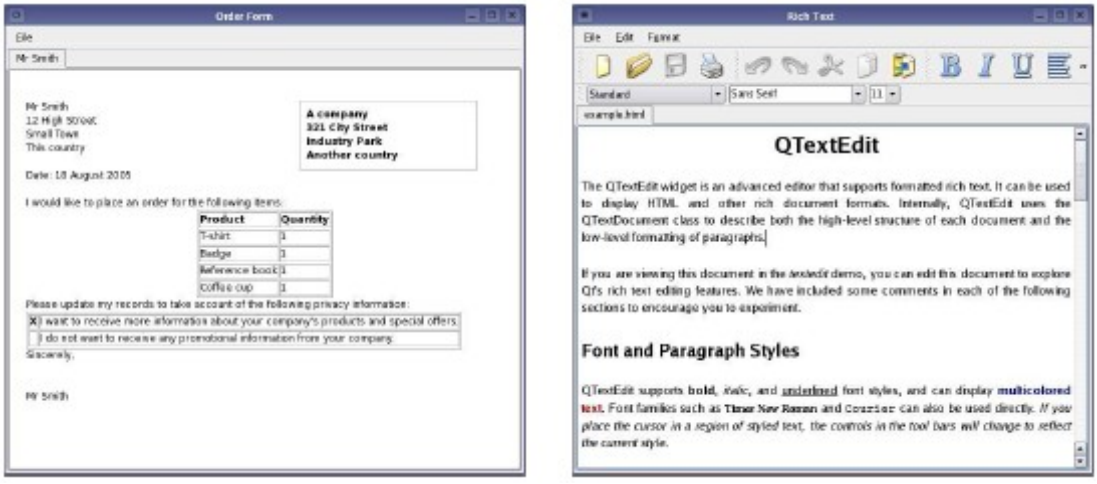

图 22：使用 Qt 的高级富文本文档功能，您可使用 **QTextEdit** 来创建并编辑复杂文档。

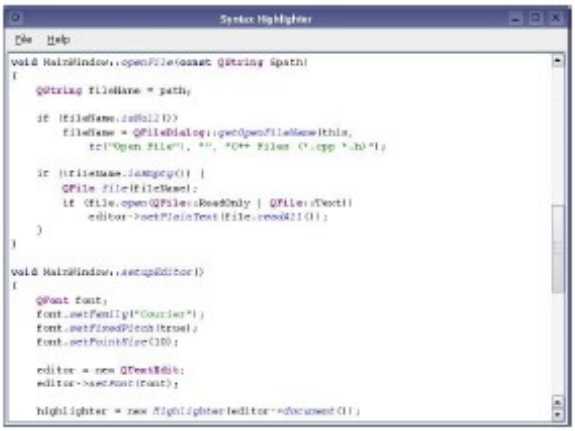

图 23：语法高亮显示可以用来更改文档的外观。

8.2. 富文本处理

将以下两种方法组合在一起，则可以以编程方式开发富文本文档：基于对象的高级方法；基于光标的方法（与使用文本编辑器类似）。使用基于对象的方法，可以轻松概览文档结构，并可在表、框架以及其他元素之间切换。而使用基于光标的方法，则可修改并转换文档内容，甚至还可以对文档结构进行大规模的调整。

以下代码首先向文档中插入一个两行两列的表格，然后再向表格的第一列单元中插入文本。

```
QTextTable *offersTable = cursor.insertTable(2, 2);
cursor = offersTable->cellAt(0, 1).firstCursorPosition();
cursor.insertText(tr("I want to receive more information about your "
"company's products and special offers."), textFormat);
cursor = offersTable->cellAt(1, 1).firstCursorPosition();
cursor.insertText(tr("I do not want to receive any promotional information "
"from your company."), textFormat);
```

在 Qt 中，除了有许多与结构和内容对应的类以外，还有许多类负责控制文本和文档元素的外观。使用这些类，可以更精确地指定文档所使用的文本样式：

```
QTextCharFormat plainFormat(cursor.charFormat());
QTextCharFormat italicFormat = plainFormat;
italicFormat.setFontItalic(true);
QTextCharFormat boldFormat = plainFormat;
boldFormat.setFontWeight(QFont::Bold);
cursor.insertText(tr("Note: "), boldFormat);
cursor.insertText(tr("We can emphasize text by making it "), plainFormat);
cursor.insertText(tr("italic"), italicFormat);
cursor.insertText(tr("."), plainFormat);
```

另外，您也可以自定义表、列表、框架以及普通段落的样式，使文档达到理想外观。

使用 **QTextEdit** 控件可以编辑采用编程方式创建的文档，并可维护完整的撤消/恢复

历史记录。开发人员可以增加可用的标准编辑功能，以便用户添加自定义结构和内容。

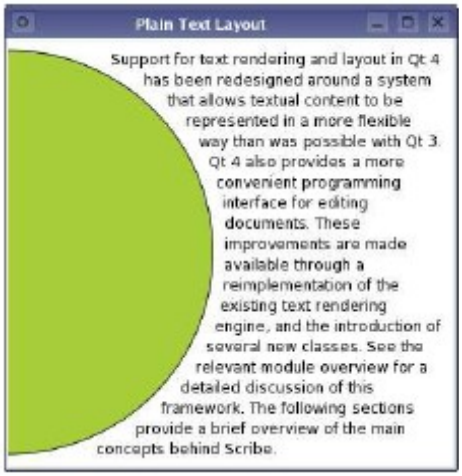

图 24：使用低级文本处理功能，可以创建专业化的自定义控件，满足专业化的文本格式需求。

8.3. 自定义

Qt 4 的文本处理功能还可以用来为自定义控件和富文本文档提供专业化的文本格式。使用低级类可以编写这类功能，例如：使用**QTextLayout**类以逐行显示文本，也可以将这类功能集成至 **QTextDocument**的扩展文本布局系统中，与**QTextEdit** 一起使用。

此外，使用 **QSyntax-Highlighter** 类可以将语法高亮显示规则应用于富文本文档中。这样允许标准的 **QTextEdit** 控件用作代码编辑器的基础，并可为文档搜索工具提供高亮显示功能。

在线参考
http://doc.trolltech.com/4.2/qt4-scribe.html
http://doc.trolltech.com/4.2/richtext.html

9. 数据库

使用 Qt SQL 模块，可以简化跨平台 GUI 数据库应用程序的创建过程。编程人员可以轻松执行 SQL 语句，可以使用数据库模型提供项目视图信息，以便使信息可视化并可实现数据输入。使用控件映射程序，编程人员可以使数据库表与基于窗体的用户界面中的特定控件相关联。

Qt SQL 模块为访问 SQL 数据库提供了多平台接口。Qt 针对 Oracle、Microsoft SQL Server 、 Sybase Adaptive Server 、 IBM DB2 、 PostgreSQL 、 MySQL 、 Borland Interbase、SQLite 和 ODBC 提供了本地驱动程序。只要 Qt 支持的平台，并可访问其客户端库，这些驱动程序均可以在其中运行。程序可以同时使用多个驱动程序来访问多个数据库。

开发人员可以轻松执行任何 SQL 语句。另外，Qt 还提供了高级 C++ 接口，使用此接口，系统可自动生成适当的 SQL 语句。

Qt 提供了一系列 SQL 模型，这些模型均可与其他模型/视图组件一起使用（请参见第 39 页）。

有了上述一系列功能，系统可以使用数据库查询结果自动填充视图控件，可以让用户和非数据库开发人员简化编辑过程。使用 Qt SQL 模块提供的便利功能，可以直接创建数据库应用程序，可以使用外部键值(Foreign Key)查找并显示主从复合结构(Master-Detail)关系。

9.1. 执行 SQL 命令

QSqlQuery 类可以直接执行任何 SQL 语句，也可以在 SELECT 语句所生成的结果集中切换。在以下示例中，我们使用 QSqlQuery::next() 来执行查询并切换查询结果集：

```
while (query.next())
    cout < < query.value(0);
```

我们已索引字段值，以便它们出现在 SELECT 语句中。另外，**QSqlQuery** 还提供了_rst()、prev()、last() 和 seek() 函数，以便帮助在结果集的记录中切换。

INSERT、UPDATE 和 DELETE 语句与 **QSqlQuery** 同样易于使用。下面，我们举例说明如何使用 UPDATE 来更改特定的记录集：

```
QSqlQuery query("UPDATE staff SET salary = salary * 1.10"
    " WHERE id > 1155 AND id < 8155");
if (query.isActive())
    cout < < "Pay rise given to " < < query.numRowsAffected() < < " staff" < < endl;
```

另外，Qt 的 SQL 模块还支持值绑定和预查询，例如：

```
QSqlQuery query;
query.prepare("INSERT INTO staff (id, surname, salary)"
" VALUES (:id, :surname, :salary)");
query.bindValue(":id", 8120);
query.bindValue(":surname", "Bean");
query.bindValue(":salary", 29960.5);
query.exec();
```

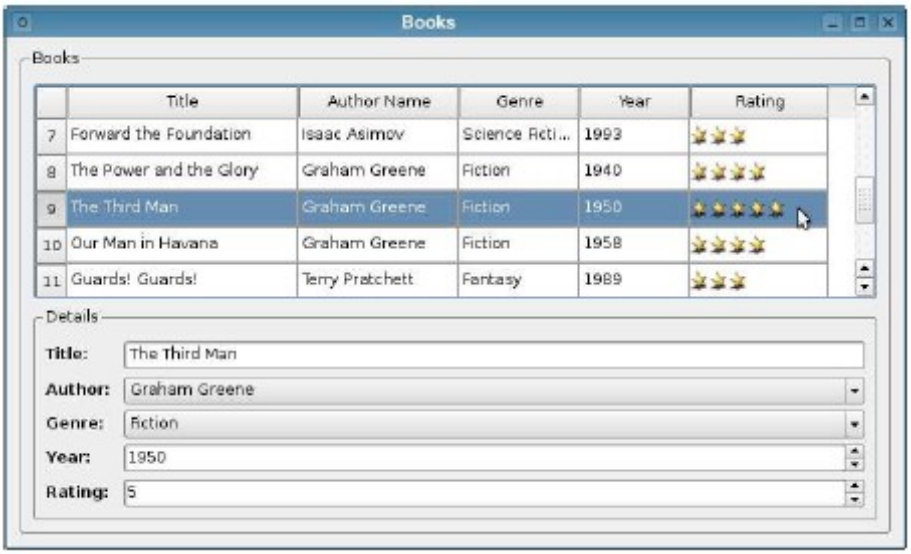

图 25：Qt 4 Books 演示显示了 Qt 的 SQL 类与模型/视图框架之间的集成。

可以使用Named Binding和Named PlaceHolder(占位符)（见上）、或者使用Positional Binding with Named或者Positional PlaceHolder，实现值绑定，例如：

```
QSqlQuery query;
query.prepare("INSERT INTO staff (id, surname, salary)"
" VALUES (?, ?, ?)");
EmployeeMap::iterator it;

for (it = employeeMap.begin(); it != employeeMap.end(); ++it) {
    query.addBindValue(it.data().id());
    query.addBindValue(it.key());
    query.addBindValue(it.data().salary());
    query.exec();
}
```

Qt 的绑定语法使用基本数据库支持或采用模拟方式，可以处理所有支持的数据库。

9.2. SQL 模型

此外，Qt 还提供了一系列模型类与模型/视图框架（请参见第 39 页）中的其他组件一起使用。使用这些模型类，开发人员可以设置 SQL 查询语句，自动向表视图提供数据库中的数据项目。使用这些数据库模型类与模型/视图框架中的其他组件后，可以最大程度

地减少开发人员的介入。

若要设置一个查询模型，则只需指定查询并选择要检查的表头即可：

```
QSqlQueryModel model;
model->setQuery("select * from person");
model->setHeaderData(0, Qt::Horizontal, QObject::tr("ID"));
model->setHeaderData(1, Qt::Horizontal, QObject::tr("First name"));
model->setHeaderData(2, Qt::Horizontal, QObject::tr("Last name"));
```

设置一个表视图来显示查询结果也同样很简单：

```
QTableView *view = new QTableView;
view->setModel(model);
view->show();
```

系统提供了以下几种模型，采用不同的方式访问 SQL 表：

- **QSqlQueryModel** 为 SQL 结果集提供了只读数据模型。
- **QSqlTableModel** 为单个数据库表提供了可编辑的数据模型。
- **QSqlRelationalTableModel** 与 **QSqlTableModel** 的功能类似，但它支持将列值作为设置为其他数据库表中的外部键值(Foreign Key)。图 25 中显示的 Qt 4 Books 演示便采用了一个关系数据库模型来查找表中每个书本的相关信息。

模型/视图框架提供了许多功能，可以充分满足数据库应用程序的要求。这些功能包括支持事务以及允许在每一行的基础上编辑表内容，从而避免出现不必要的数据库反复操作。

9.3. 数据敏感的控件

Qt 4.2 提供了许多便利功能，支持从各种模型（例如，以上所介绍的 SQL 模型）中获得数据；支持数据与窗口中的特定控件相关；支持用户从基本数据的不同存储位置查看可用数据的剖面。使用这些功能，有利于为那些习惯于数据输入和记录应用程序的用户，提供一个更熟悉的基于窗体的用户界面。

QDataWidgetMapper 类可以在模型和所选控件之间建立映射。以下代码摘自 Books 演示文档，其含义表示构建数据映射，并将此映射分配给模型，然后将要编辑数据库的字段映射至每个控件的某一列：

```
QDataWidgetMapper *mapper = new QDataWidgetMapper(this);
mapper->setModel(model);
mapper->addMapping(titleEdit, model->fieldIndex("title"));
mapper->addMapping(yearEdit, model->fieldIndex("year"));
mapper->addMapping(authorEdit, authorIdx);
mapper->addMapping(genreEdit, genreIdx);
mapper->addMapping(ratingEdit, model->fieldIndex("rating"));
```

数据映射可以直接浏览模型中的所有行，它可将每一列中的项目映射至特定的控件，然后使用此控件来显示所获得的数据。Qt 还提供了许多函数，例如 **toFirst()**、**toNext()** 和 **toPrevious()**，使得为用户提供易于使用的数据浏览控件成为可能。

由于该类还提供了一个类似于项目视图类的 API，因此，通过设置映射的根索引，可以获得层次结构模型不同剖面。另外，有了模型/视图 API 后，如果与给定模型相关联的项目视图中发生任何更改，则数据映射程序可以对此更改做出响应。在Books 演示中，只要用户在表视图中选择了另一行，演示都会更新数据；只需使用一个简单的信号 - 槽连接即可建立此更新：

```
connect(bookTable->selectionModel(),
SIGNAL(currentRowChanged(QModelIndex,QModelIndex)),
mapper,
SLOT(setCurrentModelIndex(QModelIndex)));
```

该映射程序使用模型传入的索引为新行中的每一项目获得模型索引，并自动更新所有编辑器控件。

在线参考
http://doc.trolltech.com/4.2/qtsql.html
http://doc.trolltech.com/4.2/qt4-sql.html
http://doc.trolltech.com/4.2/examples.html#sql-examples

10. 国际化

Qt 完全支持 Unicode 这一国际化标准字符集。编程人员可以随意在应用程序中混合使用阿拉伯语、英语、以色列语、日语、俄语以及其他 Unicode 所支持的语言。另外，Qt 还提供了其他工具来支持应用程序的翻译工作，帮助各大公司进军国际市场。

Qt 提供了许多工具来简化翻译过程。使用从源代码中提取文本的工具，编程人员可以轻松标记需要转换的用户可视文本。*Qt Linguist*（请参见第 50 页）是一个易于使用的 GUI 应用程序，它可以读取代码中提取出的源文本，并向该文本提供要翻译的上下文信息。完成翻译后，*Qt Linguist* 将输出一个翻译文件，供应用程序使用。

图 26：使用不同语言显示的同一用户界面。

Qt 使用 **QString** 类来存储 Unicode 字符串，并在整个 API 和系统内部都使用此类。**QString** 替换了 const char * 指针和 std::string，而 16 位 **QChar** 则替换了 char。Qt 提供了构造函数和运算符，自动在 8 位字符串之间进行转换。由于 **QString** 对象可以隐式共享（即“修改时复制”；请参见第 66 页），因此编程人员可以使用值复制来创建对象，这样可以更快捷创建对象，更好利用内存。

QString 不仅仅只是16 位字符串。它还提供了许多函数（例如：QChar::lower() 和 QChar::isPunct()）可以替换 tolower() 和 ispunct()，可以处理所有 Unicode。**QRegExp** 类负责提供 Qt 的正则表达式引擎，可以在正则表达式模式和目标字符串中使用 Unicode 字符串。

Qt 提供了区域支持，可以根据用户的地理位置和语言首选项对数字和字符串进行相应转换。例如：

```
QLocale iranian(QLocale::Persian, QLocale::Iran);

QString s1 = iranian.toString(195); // s1 == "۱۹۵"
int n = iranian.toInt(s1); // n == 195
QLocale norwegian(QLocale::Norwegian, QLocale::Norway);
QString s2 = norwegian.toString(3.14); // s2 == "3,14" (comma)
double d = norwegian.toDouble(s2); // d == 3.14
```

QTextCodec 子类负责在不同的编码以及字符集之间进行转换。Qt 使用 **QTextCodec** 来控制字体、输入/输出以及输入法；开发人员也可以根据自行需要来使用它。

QTextCodec 支持各种不同的编码，包括汉语的 Big5 和 GBK、日语的 EUC-JP、JIS 和 Shift-JIS、俄语的 KOI8-R 以及ISO-8859 系列标准编码*。通过提供字符映射表或者继承 **QTextCodec** 类的方式，开发人员可以添加自己的编码。

10.1. 文本输入和显示

远东书写系统所需字符远远超过了键盘上所能提供的字符。在窗口系统级别中，一种名叫*输入法*的软件负责将一系列按键转换成实际的字符。Qt 自动支持已安装的输入方法。

Qt 为屏幕中所显示的所有文本都提供了功能强大的文本显示引擎，包括最简单的标签以及最复杂的富文本编辑器。该引擎支持各种高级功能，例如：特殊换行、双向编写以及变音标记。它可以显示全球大部分书写系统，包括阿拉伯语、汉语、斯拉夫语、英语、希腊语、以色列语、日语、韩语、拉丁语以及越南语。Qt 会自动将系统安装的各种字体组合在一起，显示多语言文本。

10.2. 翻译程序

Qt 提供了许多工具和函数，帮助开发人员开发出本地语言的应用程序。Qt 本身大约包含 400 个用户可视的字符串，对于这些字符串，奇趣科技公司提供了法语和德语翻译文件。

若要使一个 ASCII 字符串可以被翻译，则只需将它放在 tr() 翻译函数中即可；例如：

```
saveButton->setText(tr("Save"));
```

如果有翻译字符，则 tr() 将尝试使用翻译字符来替换字符串文字（例如，"Save"）；否则它将使用原文本。例如，将英语视为源语言，汉语视为目标语言。tr() 参数将从应用程序的默认编码转换成 Unicode。作为备选方法，可以使用 trUtf8() 函数向翻译系统提供编码为 UTF-8 的文本。

tr() 的常规语法为：

```
Context::tr("source text", "comment")
```

其中“context”是指 **QObject** 子类的名称。它通常可以省略，在省略的情况下，包含 tr() 调用的类将视为上下文。“source text”是指要翻译的文本。“comment”是可选的；它与上下文一起，向各翻译人员提供附加信息。另外，您还可以使用一个可选参数来指定整数值，帮助翻译复数格式：

```
Context::tr("%n message(s) saved", "number of saved messages", messages.count())
```

其中“(s)”这一特殊符号表示将使用复数格式，这样确保系统针对用户语言使用指定整数的适当文本。

*ISO 是国际标准化机构的注册商标。

翻译内容存储在 **QTranslator** 对象中，该对象使用磁盘系统上的 .qm 文件（Qt翻译文件）。每个 .qm 文件都包含特定语言的翻译内容。可以根据实际情况或用户喜好在运行时选择特定语言。

Qt 提供了三种工具来准备 .qm 文件： lupdate、*Qt Linguist* 和 lrelease。

1. lupdate 从源代码（包括 *Qt Designer* 的 .ui 文件）中提取一系列项目，每个项目均包含一个上下文、多个源文本以及一条注释，提取后生成一个 .ts 文件（即："Translation Source"文件）。这些文件均采用可直接阅读的 XML 格式。

2. 翻译人员使用 *Qt Linguist* 来翻译 .ts 文件中的源文本。

3. .qm 文件是经过高压缩的文件，在.ts 文件上运行lrelease，即可生成此文件。

在应用程序的生命周期中，经常有必要需要重复上述步骤。如果经常运行 lupdate，也会十分安全，其原因是：lupdate 将重新使用现有翻译内容，并标记、而不是删掉那些过时不再用的源文本的翻译内容。lupdate 还会检测到源文本中出现的任何细微变化，并自动向相应的翻译内容发出建议。系统会将这些翻译内容标记为"未完成"，这样，翻译人员可以轻松对这些内容进行检查。

10.3. Qt Linguist

*Qt Linguist*应用程序是一种帮助翻译人员来翻译 Qt 应用程序的 工具。

使用 *Qt Linguist*，翻译人员可以方便快捷地编辑 .ts 文件。在 Qt Linguist 应用程序窗口的左侧，已列出 .ts 文件的上下文。在窗口右上角，则显示了当前上下文的源文本列表以及翻译内容。通过选中某一源文本，翻译人员可以开始翻译，并将此任务标记为"已完成"或"未完成"，然后继续执行下一未完成的翻译。Qt 为所有常见导航选项（例如"完成并转到下一个"和"下一个未完成"）都提供了键盘快捷键。翻译人员可依据其个人喜好随意调整用户界面的固定窗口位置。

应用程序经常在不同的源文本中多次使用同一段落。*Qt Linguist* 可以根据窗口底部显示的以前翻译过的字符串和预先定义的翻译内容自动显示合理猜测内容。猜测内容往往是一个良好的起点，有助于翻译人员始终一致地翻译类似文本。常见的翻译内容也可以存储在词汇表中，有助于更高效快速地翻译未来的应用程序。另外，*Qt Linguist* 可以有选择性地验证翻译内容，确保快捷键和句末标点翻译正确。

Qt Linguist 综合指南为版本管理员、翻译人员以及编程人员都提供了翻译过程的相关信息。

在线参考
http://doc.trolltech.com/4.2/i18n.html
http://doc.trolltech.com/4.2/unicode.html
http://doc.trolltech.com/4.2/qtextcodec.html
http://doc.trolltech.com/4.2/linguist-manual.htm

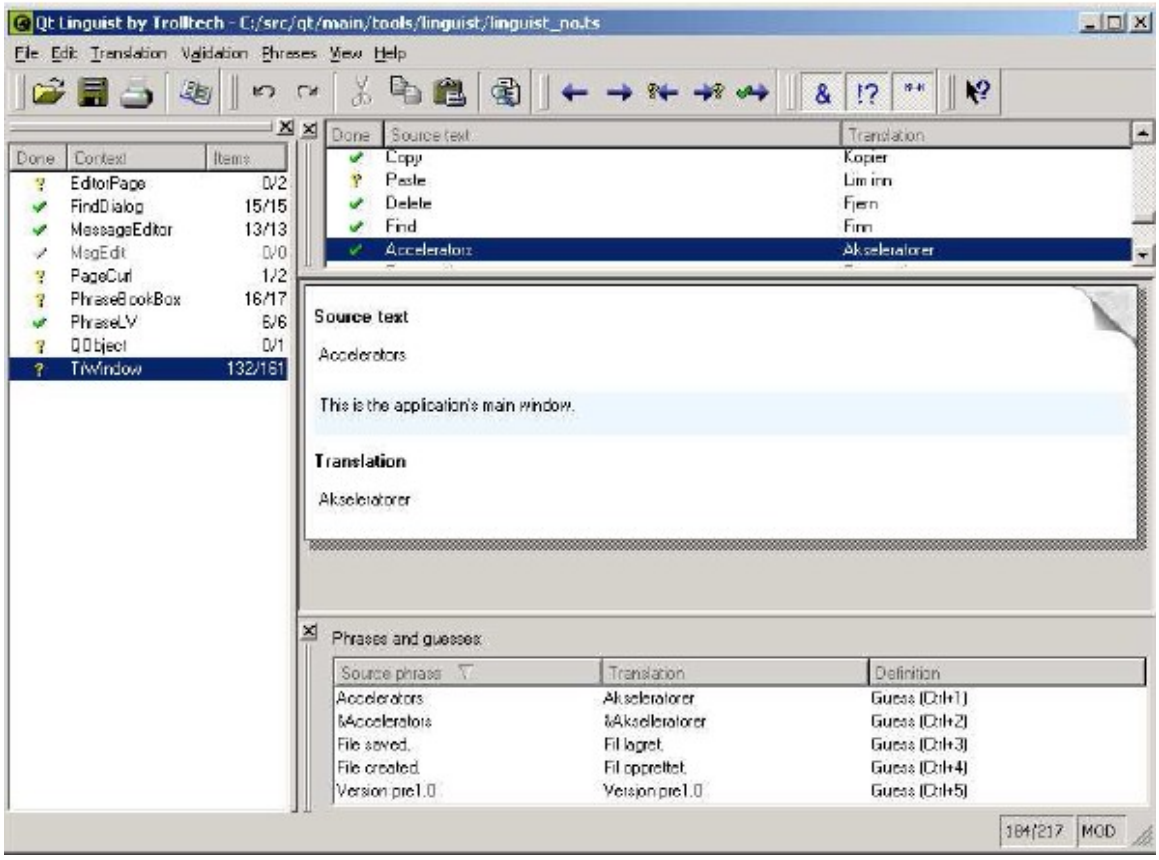

图 27：使用 *Qt Linguist* 翻译成挪威语

11. 布局

相比于使用固定尺寸和位置，布局提供了功能强大且极具灵活性的另一种方案。使用布局后，编程人员无需计算尺寸和位置，布局可以自动进行调整，符合用户屏幕、语言以及字体的要求。

Qt 的布局管理器负责在父控件区域内构建子控件。这些管理器可以使顶层控件自动定位并重新调整子控件、保持顶层控件敏感度最小化的变化和默认尺寸，并可在内容或文本字体更改时自动重新定位。在*Qt Designer*（请参见第 24 页）中，完全可以使用布局管理器来定位控件。

图 28：以不同尺寸显示的同一对话框。

布局对于国际化也非常有用。系统使用固定的尺寸和位置的情况下，经常发生文本翻译后变长而截断文本的现象（请参见图 30）；使用布局，子控件可以自动重新调整大小。另外，为了向采用从右至左书写系统的用户提供更为自然的显示效果，可以将控件的位置反转过来。

11.1. 内建布局管理器

Qt 的布局管理器可以将控件以及其他布局排列为横向，纵向或网格中。

- **QHBoxLayout** 按从左至右的顺序将管理的控件横放在一行中。
- **QVBoxLayout** 按从上至下的顺序将管理的控件竖放在一列中。
- **QGridLayout** 将管理的控件放在可扩展的单元格中，这样一来，如有必要，控件可以跨越多个单元。

每个内建布局管理器均支持控件在分配的范围内横向/纵向对齐，这样只需使用简单的布局和对齐属性，即可自定义用户界面的外观。

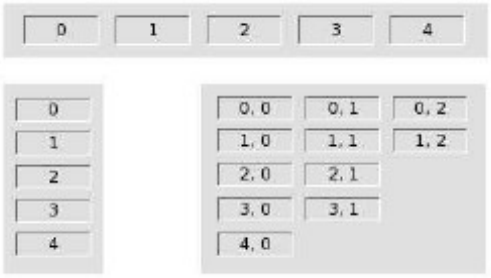

图 29：使用 **QHBoxLayout**、**QVBoxLayout** 和 **QGridLayout** 布局管理器排列的控件。

在大多数情况下，Qt 的布局管理器将为管理的控件选择最优尺寸，以便窗口可以顺利重新调整大小。如果默认值不合理，那么开发人员可以使用以下机制优化布局：

1. 为某些子控件设置最小尺寸、最大尺寸或固定尺寸。
2. 添加拉伸项目或间距项目。这些项目将填补布局中的空白区域。
3. 更改子控件的尺寸规则。通过调用 QWidget::setSizePolicy()，编程人员可以采用最优方式重新设置子控件的尺寸变化行为。可以根据布局中其他控件来扩大、缩小子控件，或者使其尺寸不变。
4. 更改子控件的尺寸提示。QWidget::sizeHint() 和 QWidget::minimum- SizeHint() 可以根据控件的内容返回其首选尺寸和首选最小尺寸。Qt内建的控件已经相应的提供了合适实现。
5. 设置拉伸因子。拉伸因子支持子控件的相对增长；例如，将 2/3 的任何多余的可用空间分配给 A 控件，将 1/3 的空间分配给 B 控件。

另外，编程人员也可以设置被布局管理的控件之间的“间距”和整个布局周围的“空白”。默认情况下，*Qt Designer* 使用与上下文相关的行业标准值。

此外，布局还可以以从右至左或从下至上的方式运行。对于支持从右至左书写系统（例如，阿拉伯语和以色列语）的国际化应用程序而言，从右至左布局十分便利。内建布局管理器与 Qt 的样式系统（请参见第 55 页）完全集成，可在反转显示中提供一致的观感。

11.2. 嵌套式布局

布局可以嵌套至任意级别中。图 28 举例以两种不同尺寸显示了一个对话框。该对话框使用了三种布局：**QVBoxLayout**，负责将按钮集中在一起、**QHBoxLayout**，负责将国家/地区列表视图与按钮集中在一起，和 **QVBoxLayout**，负责将“选择国家/地区”标签与其余控件集中在一起。拉伸项目则负责维护“取消”和“帮助”按钮之间的间距。

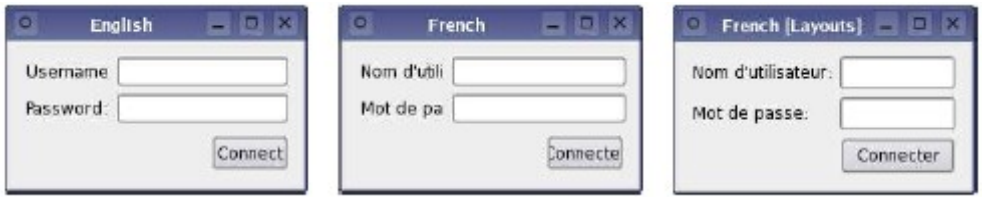

图 30：使用布局之后的效果图。在前两个图片中，显示了同一个未使用布局的原英语文本对话框和法语文本对话框；法语文本被截断了。在右边的图片中，布局中含有标签，法语文本显示完整正确。

上述对话框的控件和布局是使用以下代码创建的：

```
QVBoxLayout *buttonBox = new QVBoxLayout;
buttonBox->setSpacing(6);
buttonBox->addWidget(new QPushButton(tr("&OK")));
buttonBox->addWidget(new QPushButton(tr("&Cancel")));
buttonBox->addStretch(1);
buttonBox->addWidget(new QPushButton(tr("Help")));
QListWidget *countryList = new QListWidget(this);
countryList->addItem(tr("Canada"));
/* ... */
countryList->addItem(tr("United States of America"));
QHBoxLayout *middleBox = new QHBoxLayout;
middleBox->setSpacing(11);
middleBox->addWidget(countryList);
middleBox->addLayout(buttonBox);
QVBoxLayout *topLevelBox = new QVBoxLayout;
topLevelBox->setSpacing(11);
topLevelBox->setMargin(6);
topLevelBox->addWidget(new QLabel(tr("Select a country")));
topLevelBox->addLayout(middleBox);
setLayout(topLevelBox);
```

使用 Qt 后，布置控件变得十分容易，因此，编程人员几乎无需使用固定定位。开发人员通过继承 **QLayout** 类，可以编写自定义的布局管理器。Qt 的布局示例显示了两个自定义布局管理器：一个是边框布局，它将子控件放置在罗盘的点中；另一个是顺序布局，它象放置文字一样将控件放在页面中。编程人员可以使用并修改这些布局，为窗体创建新的布局策略。

Qt 还提供了 **QSplitter** 拆分条，最终用户可以对其进行细致操作。在某些设计情况下，**QSplitter** 比布局管理器更具优越性。

若要进行完整全面的控制，也可以重新实现 QWidget::resizeEvent() 并调用每个子控件中的 QWidget::setGeometry()，手动设置布局。

在线参考
http://doc.trolltech.com/4.2/layout.html

12. 样式和主题

Qt 针对应用程序的外观自动使用本地桌面样式。Qt 应用程序尊重用户对色彩、字体、声音以及其他桌面设置的首选项。Qt 编程人员可以自由使用提供的任一样式，并且可以覆盖任何首选项。使用 Qt 功能强大的样式引擎，编程人员可以修改现有样式或实施自己的样式。

样式负责实施用户界面在特定平台中的外观。样式是 **QStyle** 子类，负责实施基本的绘图功能（例如，绘制边框、按钮和图像等）。Qt 支持所有窗体以最快的速度灵活地绘制本身。

12.1. 内建样式

Qt 提供了以下内建样式：CDE、Cleanlooks、Motif、Mac OS X、Plastique、Windows 和 Windows XP。默认情况下，Qt 针对用户的平台和桌面环境使用适当的样式。另外，应用程序开发人员也可以通过编程方式来选择样式，或者由用户通过使用 -style 命令行选项来选择。

图 31：采用不同本地样式显示的 Comboboxes。

样式与用户桌面设置保持一致，包括：用户对色彩、字体以及声音等的首选项。Qt 自动调整至计算机的活动主题。例如，对于菜单和工具提示，Qt 支持滚动操作和淡化过渡效果。Windows XP 和 Mac OS 样式以本地样式管理器为基础建立，仅在相关平台中可用。其他样式是由 Qt 仿真的，可随处使用。

在大多数现代化的 X11 平台中，其默认样式为 Plastique，该样式由 KDE 的 Plastik 窗体样式生成，专门为大多数 Linux 和 Unix 桌面而设计。在 GNOME 桌面环境中，Qt 应用程序的默认样式为 Cleanlooks，这种样式旨在类似于 GNOME 的 Clearlooks 主题。

Qt 的内建窗体可识别样式。自定义窗体和对话框几乎总是既包括内建窗体，又包括布局，而且它们可以自动调整以适应所使用的样式。开发人员几乎无需从头开始编写自定义窗体，相反，他们可以使用 **QStyle** 来绘制基本的用户界面元素，而无需直接绘制原始图形基原。

QStyle 支持从右至左的语言。Qt 根据加载的转换文件，自动使用从右至左的窗体，

而不是使用通常使用的默认从左至右的方案。在此模式中，用户可以指定 -reverse 命令行选项来运行各个应用程序。另外，使用反转模式时，性能良好的样式会向窗体提交适合于用户桌面环境的光亮区域和阴影区域。

图 32：以上样式表举例说明用户可以如何按照自己的方式对样式表进行更改，采用互动形式体验一系列窗体的不同外观。

12.2. 窗体的样式表

Qt 4.2 开始支持窗体的样式表。样式表是使用与级联样式表 (CSS) 类似的语言编写的文本描述，它可以用来自定义窗体的外观，其方式大致类似于使用 CSS 描述来自定义 HTML（使用 WWW 浏览器来提交）的方式。通过 styleSheet 属性（可从 **QWidget** 及其子类中获得）可以访问每个窗体的样式表，使用这一方式，可以在运行应用程序时，轻松地将自定义功能运用到可识别样式的窗体中。由于此属性在 *Qt Designer* 中还可用于编辑，因此，图形设计人员可以直接控制应用程序的外观。在许多常见情况中，如果需要自定义标准窗体，则只需使用样式表，便无需编写自定义样式。

12.3. 自定义样式

使用自定义样式，可以为某一应用程序或某一系列应用程序提供独特的外观。通过将 **QStyle**、**QCommonStyle** 或其任何一个子类进行细分类，可以自定义样式。如果从适当的基础类中重新实施一或两种虚拟功能，则可对现有样式进行轻松的细微修改。

QStyle 为每个用来绘制窗体的成员组件提供信息，这样，可以创建并调整高度自定义的样式。

Qt 的本地平台样式适合于大多数应用程序。但在某些情况下，为了达到特殊的外观，有必要覆盖默认样式。设置应用程序的样式十分简单，例如：

```
QApplication::setStyle(new MyCustomStyle);
```

样式也可以编译为插件（请参见第 58 页）。使用 *Qt Designer* 中的自定义样式，插件即可预览窗体，而无需重新编译 Qt 或 *Qt Designer* 本身。使用样式插件，即可更改现有 Qt 应用程序的样式，而无需重新编译该应用程序。使用这一方法，应用程序（例如，“Qt 4 样式”示例和 qtconfig 工具）可以动态启动样式，为每个可用样式提供预览图形。

图 33： “Qt 4 样式”示例显示了内建样式，并提供了一个自定义样式进行比较。

在线参考

http://doc.trolltech.com/4.2/qt4-styles.html
http://doc.trolltech.com/4.2/widgets-styles.html
http://doc.trolltech.com/4.2/stylesheet.html
http://doc.trolltech.com/qq/qq13-styles.html

13. 事件

应用程序对象将系统消息接收为 Qt 事件。应用程序可以按照不同的粒度对事件加以监控、过滤并做出响应。

在 Qt 中，事件是指从 **QEvent继承** 的对象。QT将事件发送给每个**QObject**对象，这样对象便可对事件做出响应。编程人员可以对应用程序级别和对象级别中的事件进行监控和过滤。

13.1. 事件的创建

大多数事件是由窗口系统生成的，它们负责向应用程序通知相关的用户操作，例如：按键、鼠标单击或者重新调整窗口大小。也可以从编程角度来模拟这类事件。大约有 50 多种事件类型，最常见的事件类型是报告鼠标活动、按键、重绘请求以及窗口处理操作。编程人员可以添加自己的活动行为类似于内建事件的事件类型。

通常，接收方如果只知道按键了或者松开鼠标按钮了，这是不够的。例如，它还必须知道按的是哪个键，松开的是哪个鼠标按钮以及鼠标所在位置。每一 **QEvent** 子类均提供事件类型的相关附加信息，因此每个事件处理器均可利用此信息采取相应处理。

13.2. 事件的交付

Qt 通过调用虚函数 QObject::event() 来交付事件。出于方便起见，QObject::event() 会将大多数常见的事件类型转发给专门的处理函数，例如：QWidget::mouseReleaseEvent() 和 QWidget::keyPressEvent()。开发人员在编写自己的控件时，或者对现有控件进行定制时，可以轻松地重新实现这些处理函数。

有些事件会立即发送，而另一些事件则需要排队等候，当控制权返回至 Qt 事件循环时才会开始分发。Qt 使用排队来优化特定类型的事件。例如，Qt 会将多个 paint 事件压缩成一个事件，以便达到最大速度。

通常，一个对象需要查看另一对象的事件，以便可以对事件做出响应或阻塞事件。这可以通过调用被监控对象的 QObject::installEventFilter() 函数来实现。实施监控对象的 QObject::eventFilter() 虚函数会在受监控的对象在接收事件之前被调用。

另外，如果在应用程序的 **QApplication** 唯一实例中安装一个过滤器，则也可以过

滤应用程序的全部事件。系统先调用这类过滤器，然后再调用任何窗体特定的过滤器。您甚至还可以重新实现事件调度程序 QApplication::notify()，对整个事件交付过程进行全面控制。

在线参考
http://doc.trolltech.com/4.2/eventsand_lters.html

14. 输入/输出和网络

Qt 可以打开并保存以纯文本格式、XML 格式以及二进制格式表示的数据。Qt 使用自有类处理本地文件，使用 FTP 和 HTTP 协议处理远程文件。Qt 完全支持进程间通信以及基于套接字的 TCP 和 UDP 网络，并且可用的网络接口信息可被轻松获得。

14.1. 文件处理

Qt 提供了一些类在多个平台中执行高级输入/输出操作。**QTextStream** 类的接口与标准 <iostream> 类的接口类似，支持 **QTextCodec** 提供的编码方式。而使用 **QDataStream** 类，则可将基本的 C++ 类型以及许多以平台无关的二进制格式所表示的 Qt 类型进行序列化。例如，以下代码便向 splash.dat 文件写入了一个 Unicode 字符串、一种字体和色彩：

```
QFile outputFile("splash.dat");
if (outputFile.open(QIODevice::WriteOnly)) {
    QDataStream outputStream(&outputFile);

    outputStream < < QString("SplashWidgetStyle")
    < < QFont("Times", 18, QFont::Bold)
    < < QColor("skyblue");
}
```

使用以下代码可以轻松取得并使用上述数据：

```
QFile inputFile("splash.dat");
if (inputFile.open(QIODevice::ReadOnly)) {
    QString str;
    QFont font;
    QColor color;
    QDataStream inputStream(&inputFile);
    inputStream > > str > > font > > color;

    if (str == "SplashWidgetStyle") {
        splashWidget->setFont(font);
        splashWidget->setColor(color);
    }
}
```

QFile 类支持大文件、长文件名以及国际化的文件名。

QTextStream 和 **QDataStream** 可在任何 **QIODevice** 子类中操作。Qt 还提供了 **QBuffer**、**QTcpSocket** 和 **QUdpSocket** 子类，编程人员可以实现自己的自定义设备。另外，**QIODevice** 还提供了许多低层次函数（例如，readLine() 和 writeBlock()），使用这些函数不依赖任何数据流。

使用 **QDir**，可以读取/转换目录，也可以操作路径名称并访问基本的文件系统（例

如，创建目录或删除文件）。**QFile-Info** 提供了文件更详细的信息，例如：文件大小、权限、创建时间以及上次修改时间。

作为备选方案，应用程序可以使用 **QDirModel** 类的实例来获得与 Qt 模型/视图类兼容格式的文件和目录的相关信息（请参见第 39 页）。这使得开发者可以重用现有的视图，让用户能够图形化的查看文件。

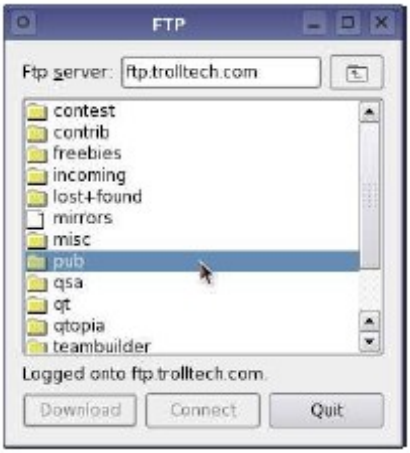

图 34：Qt 4 FTP 示例使用 **QFtp** 类来提供简单的 FTP 浏览功能。

以下示例以文件尺寸递增的顺序列出了用户主目录中的隐藏文件及文件尺寸：

```
QDir dir = QDir::home();
dir.setFilter(QDir::Files | QDir::Hidden);
dir.setSorting(QDir::Size | QDir::Reversed);
QStringList names = dir.entryList();

foreach (QString name, names) {
    QFileInfo info(dir, name);
    cout < < name.toLatin1().data() < < " " < < info.size() < < endl;
}
```

QHttp 和 **QFtp** 类支持对远程文件进行透明访问。使用 **QUrl**，可以轻松地分析并重新构建 URL。

有些文件类型可以直接读取，无需使用 **QFile** 对象。例如，通过 **QImage** 类及其可扩展的插件机制（请参见第 32 页），通常可以直接读取图像文件。而 **QPrinter** 则负责打印文本和图像（请参见第 32 页）。此外，使用 Qt 还可以对其他应用程序和服务所更改的文件和目录进行监控。**QFileSystemWatcher** 类相当于需要监控的文件路径的注册表，无论何时，只要路径中的任何一个文件或目录有所更改时，它都会发出信号。

14.2. XML

Qt 的 XML 模块提供了一个 SAX 分析器和 DOM 分析器，这两种分析器都是按非检验方式运行，并且都可以读取符合语法规范的 XML文件。SAX（Simple API for XML）实

现遵循已采纳命名惯例的SAX2 Java 实现设计。 DOM（文档对象模型）LEVEL 2实现遵循 W3C® 建议并支持命名空间。

许多 Qt 应用程序使用 XML 来存储持久数据。使用 SAX 分析器，可以读取更多数据，不但适用于只需进行简单分析的应用程序而且也包含大型文件的应用程序。DOM 分析器将整个文件读取至内存中的树形结构中，以便随时浏览。

14.3. 进程间通信

使用 **QProcess** 类，可以启动外部程序，并可采取平台无关的方式从 Qt 应用程序中与其进行通信。通信的方式是输出至外部程序的标准输入并从外部程序的标准输出和标准错误读入。由于 **QProcess** 是 **QIODevice** 的子类，因此使用 **QTextStream** 和 **QDataStream** 可以让数据在进程之间以流的方式传输。

QProcess 采用异步方式运行，如果有数据可用，它将发出信号。Qt 应用程序可以与信号相连，从而接收要处理的数据，并可通过将数据送回外部程序的方式有选择性地做出响应。另外，**QProcess** 还支持操作的阻塞模式，可以将输入和输出从外部程序重定向至文件中。

此外，使用 QtDBus 模块（请参见第 78 页中的“D-Bus 的相互操作性”），可以在特定平台中的不同应用程序、不同组件以及操作系统之间实现高层次通信。

14.4. 网络

Qt 为 编写TCP/IP 客户端程序和服务器程序提供了多平台接口，支持 IPv4和IPv6。Qt 的所有网络类均可重入，亦可从任何 **QThread** 中使用。

QTcpSocket 类提供异步缓冲 TCP 连接。**QTcpSocket** 是一个 **QIODevice**，使它很容易在套接字中使用 **QTextStream** 和 **QDataStream**。

QTcpSocket 和 **QUdpSocket** 都被设计为可与 GUI 应用程序协调运行，它们既支持阻塞模式、又支持非阻塞模式。实时货币转换应用程序便可证明这一点。该应用程序使用虚拟协议 CCP（货币转换协议）从服务器中访问最新汇率。只有与网络相关的语句写在下面。该应用程序套接字是在“Converter”构造函数中创建的：

```
...
socket = new QTcpSocket(this);
connect(socket, SIGNAL(connected()), this, SLOT(sendSourceAmount()));
connect(socket, SIGNAL(readyRead()), this, SLOT(updateTargetAmount()));
...
```

套接字通信是异步运行的，如果连接成功，则该套接字将发出 connected() 信号；如果有数据可供读取，则该套接字将发出 readyRead() 信号。用户单击“Convert”按钮时，系统将调用 convert() 槽：

```
void Converter::convert()
{
    convertButton->setEnabled(false);
    socket->connectToHost("ccp.banca-monica.nu", 1234);
}
```

发生转换时，此槽将禁用“Convert”按钮，打开连接并立即返回。当套接字与服务器相连时，它将发出 connected() 信号。

当套接字与服务器成功连接时，系统将调用 sendSourceAmount() 槽：

```
void Converter::sendSourceAmount()
{
    QString command = "CONV " + sourceAmount->text() + " " +
    sourceCurrency->currentText() + " " +
    targetCurrency->currentText() + "\r\n";
    socket->write(command.toLatin1().data());
}
```

此槽向服务器的 1234 端口发出请求（例如：CONV 100 EUR USD）。**TcpSocket** 自动将 ccp.banca-monica.nu 解析成 IP 地址。上述所有操作均未阻塞，这样用户界面可以做出响应。

```
void Converter::updateTargetAmount() {
if (socket->canReadLine()) {
targetAmount->setText(socket->readLine());
socket->close();
convertButton->setEnabled(true);
}
}
```

服务器回复 CONV 请求时将调用 updateTargetAmount() 函数。它将读取回复、更新显示内容、关闭连接并使能“Convert”按钮。

继承 **QTcpServer** 类就可以实现简单的 TCP 服务器，，与 **QTcpSocket** 类似，**QTcpServer** 采用异步方式运行。**QTcpServer** 设置了用来响应连接请求的侦听套接字，并调用虚函数为客户端提供服务。同样，**QUdpServer** 也提供了一个基于 **QUdpSocket** 的服务器。**QTcpServer** 既可在阻塞模式、也可在非阻塞模式下运行。

QAbstractSocket 类是本地套接字 API 的一个平台无关的包装器。它为 **QTcpSocket**、**QTcpServer** 和 **QUdpSocket** 提供了基础功能。**QNetworkProxy** 类支持代理服务器，既可以在应用程序范围内工作、也可以在每个套接字范围内工作。

QNetworkInterface 类负责提供有关机器网络接口的信息。它可以提供每个接口的详细信息、接口能力、分配的 IP 地址以及其他接口相关的信息。例如，对于以太网接口而言，除了获得 IP 地址以外，**QNetworkInterface** 类还可以获得基本硬件的 MAC 地址、广播地址以及子网掩码。

在线参考
http://doc.trolltech.com/4.2/qiodevice.html
http://doc.trolltech.com/4.2/xml.html
http://doc.trolltech.com/4.2/networking.html

15. 集合类

集合类用来把若干数据存储在内存中。Qt 提供了一系列与标准模板库 (STL) 相兼容的类，无论编译器是否支持 STL，这些类均可正常运行。出于安全和便利考虑，Qt 还提供了 Java风格的迭代器。

应用程序通常需要管理内存中的数据，例如：若干组图像、控件或自定义对象。许多 C++ 编译器支持 STL，可为存储项目提供准备就绪的数据结构。Qt 使用 STL 语法提供列表、堆栈以及字典。

Qt 的容器类甚至还可以与那些无法支持 STL 的编译器一起运行。在 Qt 内，各种轻量级集合类（“容器”）以及相关联的迭代器作为 Qt API 的一个组成部分，经常大量使用。使用以下两种技巧，可以优化 Qt 容器的速度和内存效率：“私有类”和“隐示共享”。编程人员也可以在支持 STL 的平台中使用 STL 容器，但这样会使 Qt 的优化性能受损。

由于编译器实际上为每个特殊类型生成相同的代码，因此模板类通常会显著增加可执行的代码数量。但 Qt 的模板收藏类是在非模板的私有类之上的薄层中实现的，因此，这些集合类已经过优化，可最大程度地减少代码的增加。

15.1. 容器

Qt 提供了五个序列容器，使用这些容器，可以保存数值或指针：**QList<T>**、**QLinkedList<T>**、**QVector<T>**、**QStack<T>** 和 **QQueue<T>**。这些容器有一个与 STL 容器特别类似的接口，可完全与 STL 算法兼容。Qt 提供了许多等同于 STL 的算法：qCopy()、qFind() 和 qSort() 等。在 STL 支持的平台中，Qt 为 STL 和 Qt 容器之间提供了自动转换的操作。

另外，Qt为熟悉 Java的开发人员 提供了 Java 式的迭代器。

Qt 提供了五个相关联的容器：**QMap<Key,T>**、**QHash<Key,T>**、**QSet<T>**、**QMultiHash<Key,T>** 和 **QMultiMap<Key,T>**。其中“Hash”容器使用 hash 函数来提高搜索性能。

使用 Qt 的序列类和相关联的集合类，既可以存储基于值的类型、又可以存储基于指针的类型，这对处理 **QWidget** 和 **QObject** 指针特别有用。保存基于指针的项目时，可以容器类删除之前先使用便利的函数来删除某一容器中的内容。

15.2. 隐示共享

收藏类中的项目与 Qt 的值类一起使用时，可以隐示共享（即“修改时复制”）。这些类的副本共享内存中的同一数据。数据共享是自动进行的；如果应用程序修改了其中某一复制对象的内容，那么系统将会进行更深入的数据复制，以便其他对象保留原样，不做任何变动。复制对象时，系统仅传输指针且仅增加一个参考计数，这样比实际复制数据的速度要快得多，而且还节约了内存。

Qt 基于值的收藏类以及其他常用类中经常使用共享，例如：**QBrush**、**QFont**、**QIcon**、**QPalette**、**QPen**、**QPixmap**、**QRegExp** 和 **String**。编程人员可以根据值来安全有效地复制这些类的对象，避免在手动优化基于指针的代码时产生相关风险。另外，Qt 还提供了低级 **QBitArray** 和 **QByteArray** 类。这些类对于处理基本数据类型十分有效。

在线参考
http://doc.trolltech.com/4.2/containers.html
http://doc.trolltech.com/4.2/shclass.html

16. 插件和动态库

使用平台无关的 API，Qt 应用程序可以访问动态库内的函数。另外，Qt 还支持插件，允许开发人员创建代码、数据库驱动器、图像格式转换器、样式以及自定义的控件，并允许将其作为单独的组件加以分发。

16.1. 插件

若要将 Qt 程序转换成插件，只需要继承相应的插件基类，并实现几个简单的函数，然后再添加一个宏。例如，采用以下方法可以将名为 **CopperStyle** 的 **QStyle** 子类转换成插件：

```
class CopperStylePlugin : public QStylePlugin
{
public:
    QStringList keys() const {
        return QStringList() < < "CopperStyle";
    }
    QStyle *create(const QString &key) {
        if (key == "CopperStyle")
            return new CopperStyle;
        return 0;
    }
};
Q_EXPORT_PLUGIN(CopperStylePlugin)
```

以下方法可以设置新样式：

```
QApplication::setStyle(QStyleFactory::create("CopperStyle"));
```

应用程序自动检测并使用作为插件提供的组件。许多第三方以源程序将 Qt 组件作为预编译的动态库和插件提供。

16.2. 动态库

QLibrary 类提供了一个跨平台的 API 来加载动态库。以下示例描述了一种最基本的动态加载库并使用库的方法。此示例实现从 mylib 库（在 Windows 中是指 mylib.dll，在 Unix 中是指 mylib.so）中获得指向 print_str 函数的指针。

```
typedef void (StrFunc)(const char *str);
QLibrary lib("mylib");
StrFunc *func = (StrFunc *) lib.resolve("print_str");
if (func) func("Hello world!");
```

这样的调用函数方法并不是类型安全的，仅支持 C 链接的符号。

在线参考
http://doc.trolltech.com/4.2/plugins-howto.html

17. 构建 Qt 应用程序

利用一套工具，Qt 开发人员可以简化在所有支持平台中构建应用程序的流程。描述应用程序、库和插件的项目文件被用来为每个平台生成适当的makefile。

17.1. Qt 的构建系统

.pro 文件描述了各个项目，该文件以文本方式概述了源文件、头文件、*Qt Designer* 窗体以及其他资源。这些资源都是由 qmake 工具来处理的，以便为每个平台中的项目生成适当的Makefile。

项目文件可描述 Qt 的所有库、工具以及示例。例如，只需以下三行即可描述 Qt 4 HTTP 示例：

```
HEADERS += httpwindow.h
SOURCES += httpwindow.cpp main.cpp
QT += network
```

前两个定义将构建此示例所需的头文件和源文件告知 qmake；而最后一个定义则确保使用 Qt 的网络连接库。使用项目文件语法，开发人员可以使用配置选项对编译流程进行精细调节，并可为不同的部署环境编写各种有条件的编译规则。

此外，使用项目文件可以描述处于目录树层次较深位置的项目。例如，Qt 示例位于顶级 examples 目录内的目录树中。examples.pro 文件要求 qmake 深入到含有下列行的各类示例的目录中：

```
TEMPLATE = subdirs
SUBDIRS = dialogs draganddrop itemviews layouts linguist \
mainwindows network painting richtext sql \
threads tools tutorial widgets xml
```

支持条件编译意味着Windows 示例程序只有在windows操作系统下的 Qt windows 版本时才会被编译。

```
win32:!contains(QT_EDITION, OpenSource|Console):SUBDIRS += activeqt
```

使用 qmake 编译项目时，编译套件中的其他工具自动会提供 Qt 的所有增强功能：moc（请参见第 12 页）将处理头文件以启用信号和槽；rcc 将编译指定的资源；而 uic 则可用来根据用户界面（即：使用 *Qt Designer* 创建的用户界面，请参见第 24 页）创建代码。

pkg-config 集成支持预编译头文件，可以生成 Visual Studio 项目文件；以及其他高级功能，可以支持开发人员在针对常见项目组件使用跨平台构建系统的同时，还支持其利

用与平台特定相关的工具。

此外，如果开发人员使用 Qt 的 Desktop 或 Desktop Light 版本，则还可从 Microsoft Visual Studio 内使用 qmake 的项目文件。在 Mac OS X 中，作为标准配置所有 Qt 版本都支持从 Apple 的 Xcode 内使用项目文件，qmake 支持创建通用的二进制文件，以便支持基于 Intel CPU和 PowerPC CPU的 Mac。

17.2.Qt 的资源系统

Qt 的资源系统支持数据文件存储在可执行文件内，这样应用程序可在运行时访问所需要的任何资源。Qt 控件支持的命名方案则允许开发人员直接访问这一整套资源。

.qrc（即：Qt Resource Collection，Qt 资源收藏）文件中列出了与应用程序一起打包的资源，包括构建目录中的文件列表以及该应用程序中所使用的资源路径。这些文件是使用 rcc 来处理的，以便创建要编译至应用程序的数据。使用这一方法，可以确保应用程序始终可以访问某些关键资源，避免出现分发问题和安装问题。

在 Qt 4.2 中，还可以使用 **QResource** 类在运行时来扩展资源系统，这样可以在额外的路径搜索资源，也可以根据需要加载没有在 .qrc 中指定的，也没有编译到应用程序中的资源。

17.3. 测试 Qt 应用程序

支持单元测试功能是作为 Qt 的标准模块加以提供的。单元测试是用C＋＋编写的一个包含测试函数的基于 **QObject** 的类，单元测试编写为可执行的测试，这样可以不依赖任何测试框架独立运行。Qt 的单元测试库还提供了扩展，支持测试图形用户界面。

在线参考
http://doc.trolltech.com/4.2/qmake-manual.html
http://doc.trolltech.com/4.2/resources.html
http://doc.trolltech.com/4.2/qtestlib-manual.html
http://www.trolltech.com/products/qt/vs-integration.html
http://www.trolltech.com/products/qt/mac.html

18. Qt 的架构

Qt 的功能是建立在它所支持平台的底层 API之上的。这使 Qt 灵活、高效，使 Qt 应用程序可与单平台的应用程序配套。

Qt 是一个跨平台的框架，它使用本地样式的 API 严格遵循每个支持平台中的用户界面指导方针。 Qt 绘制了所有控件，且编程人员可以通过重新实现虚函数的方式来扩展或自定义所有控件。正如我们在“样式和主题”（请参见第 55 页）中所述的那样，Qt 的窗体可精确模拟支持平台的观感。使用这一功能，开发人员还可以生成自己的自定义样式，为其应用程序提供具有鲜明特色的外观。

Qt 在它所支持的不同平台中使用底层 API。这与传统的“分层”跨平台工具套件不同，传统工具套件是指在单个平台工具套件中使用的简单封装（例如，在 Windows 中使用 MFC；在 X11 中使用 Motif）。通常，分层工具套件速度较慢，其原因在于：库函数的每次调用都会产生许多要经过不同 API 层的附加调用。分层工具套件往往会受到基本工具套件的功能和行为的限制，导致应用程序中出现隐性错误。

Qt 非常专业地支持各种平台，并且可以充分利用各种平台的优点：Microsoft Windows、X11、Mac OS X 和 Embedded Linux。使用单个源代码树，Qt 应用程序可以编译成每个目标平台的可执行程序。尽管 Qt 是一个跨平台的框架，但与许多平台特定的工具套件相比，客户已经体验到 Qt 更易于学习，更具有高效性。许多客户在开发单个平台时更倾向于使用 Qt，是因为 Qt 的完全面向对象的方法。

18.1. X11

Qt/X11 使用 Xlib 直接与 X 服务器通信。Qt 不使用 Xt（X Toolkit，即：X 工具套件）、Motif、Athena 或其他任何工具套件。

Qt 支持各种 Unix ：AIX®、FreeBSD®、HP-UX、Irix®、Linux、NetBSD、OpenBSD 和 Solaris。有关 Qt 所支持的编译器和操作系统版本的最新列表信息，请访问奇趣科技公司网站。

Qt 应用程序自动适应用户的窗口管理器或桌面环境，并且在 Motif、CDE、GNOME 和KDE 下具有桌面环境本身的观感。这与大多数 Unix 工具套件相反，这些套件总是把用户限制在套件自身观感下。Qt 全面支持 Unicode（请参见第 47 页）。Qt 应用程序自动支持 Unicode 和非 Unicode 字体。Qt 将多种 X 字体组合在一起，可显示多语言文本。Qt 的字体处理功能十分强大，可以在所有已安装的字体中搜索当前字体中不存在的字符。

Qt 可以充分利用 X 扩展程序。对于反锯齿字体、alpha混合字体和矢量图形，Qt 支持 RENDER 扩展程序。Qt 还为 X 输入方法提供了现场编辑功能。Qt可以 使用传统的多头显

示适配器和 Xinerama 支持多个屏幕。

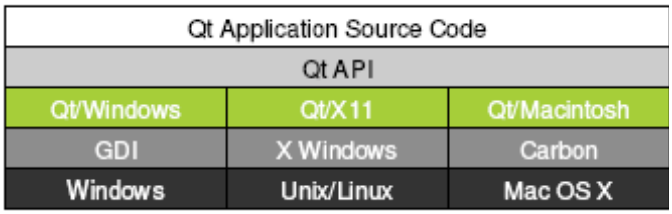

图 35：支持桌面平台中的 Qt 架构概览图。

18.2. Microsoft Windows

Qt/Windows 使用 Win32® API 和 GDI 用于事件和绘图原语。Qt 不使用 MFC 或任何其他工具套件。特别地，Qt 不使用缺乏灵活性的“常见”控件上，而是采用功能更强大的可自定义的控件（如果不是特殊应用，Qt 使用 Windows本身的 文件和打印对话框）。

使用 Windows 的客户可以在 Windows 98、NT4、ME、2000、XP 和 Vista 中使用 Microsoft Visual C++® 和 Borland C++来创建 Qt 应用程序。

Qt 为 Windows 版本执行运行检查，并使用提供的最高级功能。例如，只有 Windows NT4、2000、XP 和 Vista 支持旋转文本；Qt 则在所有 Windows 版本中都支持旋转文本，并在可能的情况下使用了操作系统本身的支持。Qt 开发人员还可以避免处理不同版本 Windows API 中的差异。

Qt 支持 Microsoft 的可访问界面。与 Windows 中的常见控件不同，您可以扩展 Qt 控件，而不会丢失基本控件的可访问信息。另外，您也可访问自定义控件。 Qt 支持 Microsoft Windows下多个屏幕显示。

18.3. Mac OS X

Qt 将 Cocoa® 和 Carbon® API 组合在一起用来支持 Mac OS X。

Qt/Mac 引入了布局并直接支持国际化，允许采用标准化方式访问 OpenGL，并使用 *Qt Designer* 提供了功能强大的可视化设计。Qt 使用事件循环处理文件和异步套接字的输入输出。Qt 提供了稳定的数据库支持。开发人员可以使用流行的面向对象的 API 来创建 Macintosh 应用程序，该 API 具有综合文档和全部的源代码。

Macintosh 开发人员可以在自己喜欢的平台上创建应用程序，在其他受支持的平台中，只需进行简单的重编译，即可显著扩大应用程序市场。Qt 支持 Mac OS X 中通用的二

进制，这意味着可以为基于 Intel CPU和 PowerPC CPU的 Mac 创建 Qt 应用程序。

在线参考
http://www.trolltech.com/products/platforms/
http://doc.trolltech.com/4.2/installation.html
http://doc.trolltech.com/4.2/deployment.html

19. 特定平台的扩展和 Qt 解决方案

除了本身十分完善以外，Qt 还提供了一些平台依赖的扩展，以便在某些情况下为开发人员提供帮助。使用 ActiveQt 扩展，开发人员可以在 Qt 应用程序内使用 ActiveX 控件，也可将 Qt 应用程序嵌入 ActiveX 服务器中。另外，通过使用 Qt Solutions解决方案（一种包含在 Qt 商业许可内的服务），也可以获得其他平台特定的扩展。

除了以下列出的 ActiveQt 扩展以外，还有许多第三方供应商提供的附加扩展。例如：*froglogic* 提供的 Tq（支持 Tcl/Tk 集成）；Klarälvdalens Datakonsult 提供的 Microsoft Windows 资源转换器。

19.1. ActiveX 的互操作性

ActiveX 建立在 Microsoft 的 COM 技术基础之上。它支持应用程序和库使用组件服务器所提供的组件，支持应用程序和库在其自有权限范围内成为组件服务器，支持使用由其他应用程序提供的 ActiveX 控件。

使用 ActiveQt，也可以与 Microsoft 的 .NET™ 技术集成。利用 ActiveQt 的 COM 支持，应用程序可以自动让. NET ™ 开发人员访问 Qt 控件和数据类型。

ActiveQt 可与 ActiveX 在 Qt 中无缝集成：ActiveX 属性、方法以及事件均可成为 Qt 的属性、槽和信号。这样，Qt 开发人员可以直接使用熟悉的编程方法来处理 ActiveX，并将其与所有不同类型的生成代码（通常，这些代码是 ActiveX 实施的一个组成部分）隔离开来。下面，我们举例说明如何注册 Internet Explorer 作为 ActiveX 组件使用：

```
#define CLSID_InternetExplorer "{8856F961-340A-11D0-A96B-00C04FD705A2}"
QAxWidget *activeX = new QAxWidget(this);
activeX->setControl(CLSID_InternetExplorer);
```

如果需要跟踪用户如何使用此组件，比如下面的方法看到它的标题如何变动：

```
connect(activeX, SIGNAL(TitleChange(const QString &)),
this, SLOT(setWindowTitle(const QString &)));
```

ActiveQt 自动处理 ActiveX 和 Qt 数据类型之间的转换操作。ActiveQt 支持 dynamicCall() 函数在控件的 **IDispatch** 界面实施中控制 ActiveX 组件：

```
activeX->dynamicCall("Navigate(const QString &)", "http://doc.trolltech.com");
```

将 Qt 应用程序转至 ActiveX 服务器十分简单。如果只需导出一个类，那么只需将 qaxfactory.h 头文件和适当的 QAXFACTORY_DEFAULT 宏包括在内即可。编译类后，该类的属性、槽和信号将成为 ActiveX 的属性、方法和指向 ActiveX 客户端的事件。此外，

ActiveQt 还提供了 QAxFactory::isServer() 函数，调用此函数，可以确定应用程序是在自有权限范围内运行、还是作为 ActiveX 控件运行。这样一来，开发人员可以控制哪些环境中可以使用哪些功能。

19.2. D-Bus 的互操作性

在支持 D-Bus 协议的 Unix 平台中，QtDBus 模块提供了进程间通信功能，使应用程序可以公开服务，供其他进程和应用程序使用。服务是使用基于 XML 的文件格式在接口文件中指定的，并通过使用 Qt 提供的 qdbusxml2cpp 工具转换成 C++ 源代码。

19.3. Qt Solutions

除了 Qt 提供的所有类以外，奇趣科技公司还提供了 Qt Solutions这一可选服务，Qt 被许可方可以在采购时购买这一服务，也可作为附加产品购买。Qt Solutions提供了一系列定期更新的组件和控件，其中许多组件和控件都可基于和 Qt 相同的双重授权方案获得。Qt 4 开发人员几乎可以使用 Qt 3 开发人员可以使用的所有解决方案，Qt 4 的许多最新解决方案也已发布。

在线参考
http://doc.trolltech.com/4.2/activeqt.html
http://doc.trolltech.com/4.2/intro-to-dbus.html
http://www.trolltech.com/products/solutions

20. Qt 开发社区

全球各大公司以及独立开发人员每天都在加入 Qt 的开发社区。他们已经认识到了 Qt 的架构本身便可加快应用程序开发进度。这些开发人员，无论是想开发单平台软件、还是想开发跨平台软件，都可从 Qt 统一而直接的 API、强大的构建系统以及各种支持工具（例如 Qt Designer）中受益无穷。

Qt具有一个极具活力并十分有益的用户社区，用户可以通过以下方式进行沟通：qt-interest 邮件列表、*Qt Centre* 网站（网址为：www.qtcentre.org）以及其他社区网站和博客。另外，许多 Qt 开发人员也是 KDE 社区的活跃成员。Qt 客户每个季度都会收到我们的开发人员新闻通讯《*Qt Quarterly*》。如今，Qt 的用户社区还提供了越来越多的第三方商业软件和开源软件；有关最新信息，请访问 www.trolltech.com。

Qt 的一系列文档可在线访问，网址为：doc.trolltech.com。另外，有关 Qt 编程 的详细介绍，市场上还有一系列英语、法语、德语、俄语、日语以及中文版本书籍。Qt 的官方书籍是《**C++ GUI Programming with Qt 4**》(ISBN 0-13-187249-4)。

奇趣科技公司及其合作伙伴为 Qt、Qtopia 和 C++ 提供了一系列培训选择，包括针对大众客户提供的开放式课程培训以及针对具有特殊需求的客户所提供的现场培训。有关详细信息，请访问 www.trolltech.com/training/。

除了为 C++ 开发人员提供综合框架以外，Qt 还可以使用其他语言编程。奇趣科技公司提供了 Qt *应用程序脚本* (QSA)，这是一种类似JavaScript的技术，使用这一技术，开发人员可以用编写脚本的方法让用户访问应用程序的某些特定区域。有关和应用程序一起部署 QSA 的详细信息，请参阅《QSA 白皮书》。

奇趣科技公司还提供了 **Qt Jambi** 这一技术，使用这种技术，Java 开发人员可以基于 Java 编程语言来使用Qt。

另外，奇趣科技公司及第三方公司针对 JavaScript、Python、Perl 和 Ruby 提供了语言绑定；其中许多解决方案都是由开源开发团队提供并加以维护的。

开发人员可在自己喜欢的平台上试用 Qt（可获得支持），时间期限为 30 天。有关详细信息，请发送电子邮件至 info@trolltech.com。

在线参考
http://partners.trolltech.com/
http://lists.trolltech.com/qt-interest/
http://doc.trolltech.com/qq/

索引