Qt 嵌入式图形开发（入门篇）

作者：深圳市优龙科技有限公司　　时间：2004/6/7

一、Qt/Embedded 开发环境的安装

一般来说，居于Qt/Embedded开发的应用程序最终会发布到安装有嵌入式Linux操作系统的小型设备上，所以使用装有Linux操作系统的PC机或者工作站来完成Qt/Embedded开发当然是最理想的环境，尽管Qt/Embedded也可以安装在Unix和Windows系统上。

下面我们将介绍如何在一台装有Linux操作系统的机器上建立Qt/Embedded开发环境。

首先，您需要拥有三个软件安装包：tmake工具安装包，Qt/Embedded 安装包，Qt的X11版的安装包。

由于上述这些软件安装包有许多不同的版本，您要注意由于版本的不同导致这些软件在使用时可能造成的冲突，为此我们将告诉您一些基本的安装原则：当您选择或下载了Qt/Embedded 的某个版本的安装包之后，您下一步要选择安装的Qt for X11的安装包的版本必须比您最先下载的Qt/Embedded 的版本要旧，这是因为Qt for X11的安装包的两个工具uic和designer产生的源文件会和Qt/Embedded的库一起被编译链接，本着“向前兼容”的原则，Qt for X11的版本应比Qt/Embedded的版本旧。

我们将以下面所列版本的安装包，一步一步介绍Qt/Embedded开发环境建立的过程（这些软件可以免费从trolltech的WEB或FTP服务器上下载），

- tmake 1.11 或更高版本；　（生成Qt/Embedded应用工程的Makefile文件）
- Qt/Embedded 2.3.7 （Qt/Embedded 安装包）
- Qt 2.3.2 for X11；　（Qt的X11版的安装包，它将产生x11开发环境所需要的两个工具）

1、安装 tmake

在 Linux 命令模式下运行以下命令：

```
tar xfz tmake-1.11.tar.gz
export TMAKEDIR=$PWD/tmake-1.11
export TMAKEPATH=$TMAKEDIR/lib/qws/linux-x86-g++
export PATH=$TMAKEDIR/bin:$PATH
```

2. 安装 Qt/Embedded 2.3.7

在 Linux 命令模式下运行以下命令：

```
tar xfz qt-embedded-2.3.7.tar.gz
cd qt-2.3.7
export QTDIR=$PWD
export QTEDIR=$QTDIR
export PATH=$QTDIR/bin:$PATH
export LD_LIBRARY_PATH=$QTDIR/lib:$LD_LIBRARY_PATH
./configure -qconfig -qvfb -depths 4,8,16,32
make sub-src
cd ..
```

上述命令 ./configure -qconfig -qvfb -depths 4,8,16,32 指定 Qt 嵌入式开发包生成虚拟缓冲帧工具 qvfb，并支持 4，8，16，32 位的显示颜色深度。另外我们也可以在 configure 的参数中添加－system－jpeg 和 gif，使 Qt/Embedded 平台能支持 jpeg、gif 格式的图形。

上述命令 make sub-src 指定按精简方式编译开发包，也就是说有些 Qt 类未被编译。Qt 嵌入式开发包有 5 种编译范围的选项，使用这些选项，可控制 Qt 生成的库文件的大小，但是您的应用所使用到的一些 Qt 类将可能因此在 Qt 的库中找不到链接。编译选项的具体用法可运行./configure －help 命令查看。

3. 安装 Qt/X11 2.3.2

在 Linux 命令模式下运行以下命令：

```
tar xfz qt-x11-2.3.2.tar.gz
cd qt-2.3.2
export QTDIR=$PWD
export PATH=$QTDIR/bin:$PATH
export LD_LIBRARY_PATH=$QTDIR/lib:$LD_LIBRARY_PATH
./configure -no-opengl
make
make -C tools/qvfb
mv tools/qvfb/qvfb bin
cp bin/uic $QTEDIR/bin
cd ..
```

根据开发者本身的开发环境，也可以在 configure 的参数中添加别的参数，比如

-no-opengl 或-no-xfs，可以键入./configure -help 来获得一些帮助信息。

二、认识 Qt/Embedded 开发环境

Qt/Embedded 的开发环境可以取代那些我们熟知的 UNIX 和 WINDOWS 开发工具。它提供了几个跨平台的工具使得开发变得迅速和方便，尤其是它的图形设计器。Unix 下的开发者可以在 PC 机或者工作站使用虚拟缓冲帧，从而可以仿真一个和嵌入式设备的显示终端大小，象素相同的显示环境。

嵌入式设备的应用可以在安装了一个跨平台开发工具链的不同的平台上编译。最通常的做法是在一个UNIX系统上安装跨平台的带有libc库的GNU c++编译器和二进制工具。在开发的许多阶段，一个可替代的做法是使用Qt的桌面版本，例如Qt/X11或是Qt/Windows来进行开发。这样开发人员就可以使用他们熟悉的开发环境，例如微软的 Visual C++ 或者 Borland C++；在UNIX操作系统下，许多环境也是可用的，例如 Kdevelop，它也支持交互式开发。

如果 Qt/Embedded 的应用是在 UNIX 平台下开发的话，那么它就可以在开发的机器上以一个独立的控制台或者虚拟缓冲帧的方式来运行,对于后者来说,其实是有一个 X11 的应用程序虚拟了一个缓冲帧。通过指定显示设备的宽度，高度和颜色深度，虚拟出来的缓冲帧将和物理的显示设备在每个像素上保持一致。这样每次调试应用时开发人员就不用总是刷新嵌入式设备的 FLASH 存储空间,从而加速了应用的编译、链接和运行周期。

运行 Qt 的虚拟缓冲帧工具的方法是：在 Linux 的图形模式下运行命令：

```
qvfb (回车)
```

当 Qt 嵌入式的应用程序要把显示结果输出到虚拟缓冲帧时，我们在命令行运行这个程序时，在程序名后加上-qws 的选项。例如： $> hello -qws

2．1 QT 的支撑工具

Qt 包含了许多支持嵌入式系统开发的工具，其中一些工具我们会在别的地方介绍。有两个最实用的工具（除了上面我们提到的虚拟缓冲帧）是 qmake 和 Qt designer(图形设计器)。

qmake 是一个为编译 Qt/Embedded 库和应用而提供的 Makefile 生成器。它能够根据一个工程文件（.pro）产生不同平台下的 Makefile 文件。qmake 支持跨平台开发和影子生成（shadow builds），影子生成是指当工程的源代码共享给网络上的多台机器时，每台机器编译链接这个工程的代码将在不同的子路径下完成，这样就不会覆盖别人的编译链接生成的文件。qmake 还易于在不同的配置之间切换。

开发者可以使用 Qt 图形设计器可视化地设计对话框而不需编写一行代码。使用 Qt 图形设计器的布局管理可以生成具有平滑改变尺寸的对话框,qmake 和 Qt 图形设计器是完全集成在一起的。

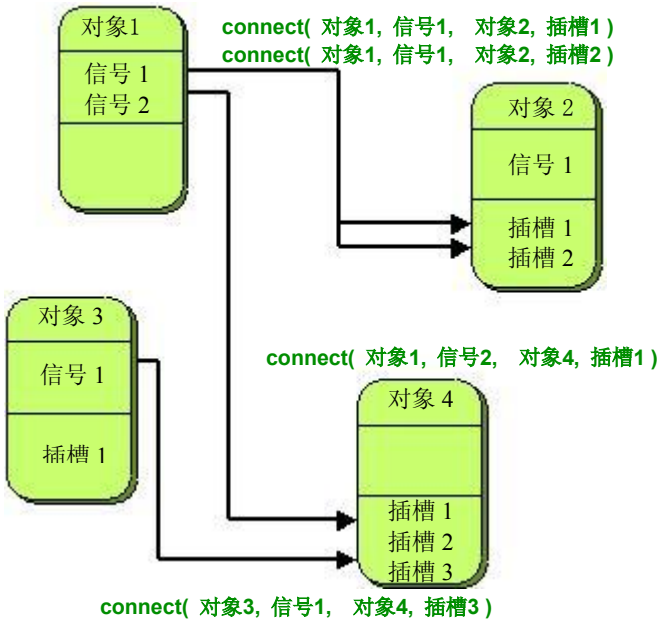
2．2 信号与插槽

信号与插槽机制提供了对象间的通信机制，它易于理解和使用，并完全被 Qt 图形设计器所支持。

图形用户接口的应用需要对用户的动作做出响应。例如，当用户点击了一个菜单项或是工具栏的按钮时，应用程序会执行某些代码。大部分情况下，我们希望不同类型的对象之间能够进行通信。程序员必须把事件和相关代码联系起来，这样才能对事件做出响应。以前的工具开发包使用的事件响应机制是易崩溃的，不够健壮的，同时也不是面向对象的。Trolltech 已经创立了一种新的机制，叫做“信号与插槽”。信号与插槽是一种强有力的对象间通信机制，它完全可以取代原始的回调和消息映射机制；信号与插槽是迅速的，类型安全的，健壮的，完全面向对象并用 C++来实现的一种机制。

在以前，当我们使用回调函数机制来把某段响应代码和一个按钮的动作相关联时，我们通常把那段响应代码写成一个函数，然后把这个函数的地址指针传给按钮，当那个按钮被按下时，这个函数就会被执行。对于这种方式，以前的开发包不能够确保回调函数被执行时所传递进来的函数参数就是正确的类型，因此容易造成进程崩溃，另外一个问题是，回调这种方式紧紧的绑定了图形用户接口的功能元素，因而很难把开发进行独立的分类。

Qt的信号与插槽机制是不同的。Qt的窗口在事件发生后会激发信号。例如一个按钮被点击时会激发一个“clicked”信号。程序员通过建立一个函数（称作一个插槽），



图一　一些信号与插槽连接的抽象图

然后调用 connect()函数把这个插槽和一个信号连接起来，这样就完成了一个事件和响应代码的连接。信号与插槽机制并不要求类之间互相知道细节，这样就可以相对容易的开发出代码可高重用的类。信号与插槽机制是类型安全的，它以警告的方式报告类型错误，而不会使系统产生崩溃。

例如，如果一个退出按钮的clicked() 信号被连接到了一个应用的退出函数－

quit() 插槽。那么一个用户点击退出键将使应用程序终止运行。上述的连接过程用代码写出来就是这样

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

我们可以在Qt应用程序的执行过程中增加或是减少信号与插槽的连接。

信号与插槽的实现扩展了C++的语法，同时也完全利用了C++面向对象的特征。信号与插槽可以被重载或者重新实现，它们可以定义为类的公有，私有或是保护成员。

2．2．1 信号与插槽的例子

如果一个类要使用信号与插槽机制，它就必须是从QObject或者QObject的子类继承，而且在类的定义中必须加上Q_OBJECT宏。信号被定义在类的信号部分，而插槽则定义在public slots, protected slots 或者 private slots 部分。

下面定义一个使用到信号与插槽机制的类。

```
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 curBalance;
};
```

和大部分的C++的类一样，BankAccount类有一个构造函数，还有一个取值的函数balance()，一个设置值的函数setBalance(int newBalance)。

这个类有一个信号balanceChanged(),这个信号声明了它在BankAccount类的成员curBalance的值被改变时产生。信号不需要被实现，当信号被激发时，和该信号连接的插槽将被执行。

上面用来设置值的函数 setBalance(int newBalance)定义在类的“public slots”部分，因此它是一个插槽。插槽是一个需要实现的标准的成员函数，它可以像其它函数一样被调用，也可以和信号相连接。

下面就是该插槽函数 setBalance(int newBalance)的实现代码：

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

其中的一段代码

```
emit balanceChanged( curBalance );
```

它的作用是当 curBalance 的值被改变后，将新的 curBalance 的值作为参数去激活 balanceChanged（）信号。对于关键词“emit”，它和信号、插槽一样是由 Qt 提供的，这些关键词都会被 c＋＋的预处理机制转换为 c＋＋代码。

一个对象的信号可以被多个不同的插槽连接，而多个信号也可以被连接到相同的插槽。当信号和插槽被连接起来时，应当确保它们的参数类型是相同的，如果插槽的参数个数小于和它连接在一起的信号的参数个数，那么从信号传递插槽的多余的参数将被忽略。

2．2．2 元对象编译器

信号与插槽机制是以纯 C++代码来实现的，实现的过程使用到了 Qt 开发工具包提供的预处理器和元对象编译器（moc）。

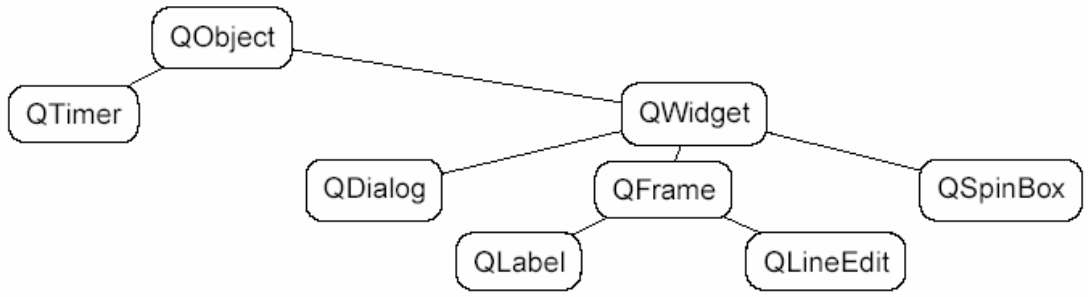
moc 读取应用程序的头文件，并产生支持信号与插槽的必要的代码。开发者没必要编辑或是浏览这些自动产生的代码，当有需要时，qmake 生成的 Makefile 文件里会显式的包含了运行 moc 的规则。

除了可以处理信号与插槽机制之外，moc 还支持翻译机制，属性系统和运行时的信息。

2．3 窗体

Qt 拥有丰富的满足不同需求的窗体（按钮，滚动条等等），Qt 的窗体使用起来很灵活，为了满足特别的要求，它很容易就可以被子类化。

窗体是**Qwidget**类或它子类的实例，客户自己的窗体类需要从**Qwidget**它的子类继承。



图二 摘录的 Qwidget 类的继承图

一个窗体可以包含任意数量的子窗体，子窗体可以显示在父窗体的客户区，一个没父窗体的窗体我们称之为顶级窗体（一个“窗口”），一个窗体通常有一个边框和标题栏作为装饰。Qt 并未对一个窗体有什么限制，任何类型的窗体可以是顶级窗体，任何类型的窗体可以是别的窗体的子窗体。在父窗体显示区域的子窗体的位置可以通过布局管理自动的进行设置，也可以人为的指定。当父窗体无效，隐藏或被删除后，它的子窗体都会进行同样的动作。

标签，消息框，工具栏等等等，并未被限制使用什么颜色，字体和语言。Qt的文本呈现窗体可以使用HTML子集显示一个多语言的宽文本。

2．3．1 一个 Hello 的例子

下面是一个显示“Hello Qt/Embedded!”的程序的完整的源代码：

图三 Hello Qt/Embedded

```
#include <qapplication.h>
#include <qlabel.h>
int main( int argc, char **argv )
{
    QApplication app( argc, argv );
    QLabel *hello = new QLabel( "<font color=blue>Hello"
    " <i>Qt/Embedded!</i></font>", 0 );
    app.setMainWidget( hello );
    hello->show();
    return app.exec();
}
```

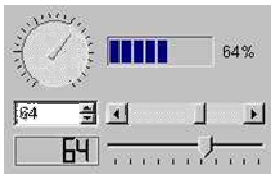
2．3．2 通用窗体

下面是一些主要的 Qt 窗体的截屏图，这些窗体使用了窗口样式。

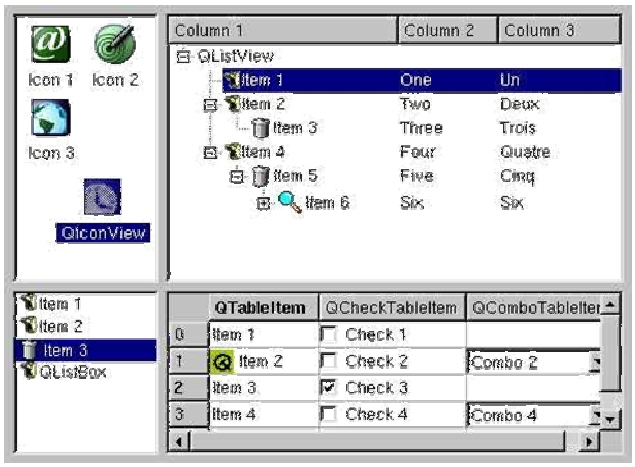
图四 使用了 QHBox 进行排列一个标签和一个按钮

图五 使用了QbuttonGroup的两个单选框和两个复选框

图六 使用了QgroupBox进行排列的的日期类QDateTimeEdit, 一个行编辑框类QLineEdit, 一个文本编辑类QTextEdit 和一个组合框类QComboBox



图七 以QGrid排列的一个 QDial, 一个QProgressBar, 一个QSpinBox, 一个QScrollBar, 一个QLCDNumber和一个QSlider



图八 以QGrid排列的一个QIconView, 一个 QListView, 一个 QListBox 和一个 QTable

有些时候在进行字符输入时，我们希望输入的字符满足了某种规则才能使输入被确认。Qt提供了解决的办法，例如QComboBox, QLineEdit 和 QspinBox 的字符输入可以通过Qvalidator的子类来进行约束和有效性检查。

通过继承QScrollView ，QTable, QListView, QTextEdit 和其它窗体就能够显示大量的数据，并且自动的拥有了一个滚动条。

许多Qt创建的窗体能够显示图像，例如按钮，标签，菜单项等等。Qimage类支持几种图形格式的输入、输出和操作，它目前支持的图形格式有BMP, GIF*, JPEG, MNG, PNG, PNM, XBM 和 XPM。

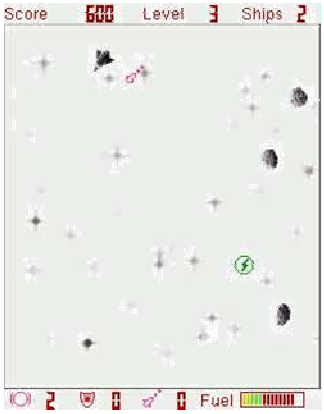
2．3．3 画布

QCanvas 类提供了一个高级的平面图形编程接口，它可以处理大量的像线条、矩形、椭圆、文本、位图、动画等这些画布项，画布项可以较容易的做成交互式的（例如做成支持用户移动的）。

画布项是QcanvasItem子类的实例，它们比窗体类Qwidget更显得轻量级，它们能够被快速的移动，隐藏和显示。Qcanvas可以更有效的支持冲突检测，它能够列出一个指定区域里面的所有的画布项。QcanvasItem可以被子类化，从而可以提供更多的客户画布项类型，或者扩展已有的画布项的功能。

Qcanvas对象是由QcanvasView进行绘制的，QcanvasView对象可以以不同的译文、比例、旋转角度，剪切方式去显示同一个画布。

Qcanvas 对象是理想的数据表现方式，它已经被消费者用于绘制地图和显示网络拓扑结构。它也可用于制作快节奏的且有大量角色的平面游戏。



图九 在 Qtopia 中用 QCanvas 实现的小行星游戏

2．3．4 客户窗体

通过对Qwidget或者它的子类进行子类化，我们可以建立自己的客户窗体或者对话框。下面是一个完整的源代码例子，它示例了如何通过子类化窗体，绘制一个模拟的时钟。
AnalogClock 窗体类是Qwidget的子类，它显示当前时间，并且可以自动地更新时间。

图十 模拟钟窗体

在 analogclock.h头文件中，**AnalogClock** 以这样地形式定义：：

```
#include <qwidget.h>
class AnalogClock : public QWidget
{
    public:
    AnalogClock( QWidget *parent = 0, const char *name = 0 );
    protected:
    virtual void timerEvent( QTimerEvent *event );
    virtual void paintEvent( QPaintEvent *event );
};
```

AnalogClock 类继承了Qwidget, 它有一个典型的窗体类构造函数，这个函数有父窗口对象指针和名字指针两个参数。(如果设置了名字的话，测试和调试起来就会容易些)
timerEvent() 函数是从QObject （Qwidget的父类）对象继承而来的，这个函数会被系统定期调用。paintEvent() 函数是从QWidget 继承而来的并且当窗体需要重画时这个函数就会被调用。

timerEvent() 和 paintEvent() 函数是“事件句柄”的两个例子。应用对象以重载父类对象的虚拟函数events (QEvent objects) 的形式接收系统的事件。大约有超过50个的系统事件是较常用的，例如 MouseButtonPress, MouseButtonRelease, KeyPress, KeyRelease, Paint, Resize 和Close. 对象可以对发给它们的事件做出响应或者筛选一些事件后再发送给别的对象。

analogclock.cpp 文件是定义在analogclock.h中的函数的实现源文件

```
#include <qdatetime.h>
#include <qpainter.h>
#include "analogclock.h"
AnalogClock::AnalogClock( QWidget *parent, const char *name )
: QWidget( parent, name )
{
    startTimer( 12000 );
    resize( 100, 100 );
}
void AnalogClock::timerEvent( QTimerEvent * )
{
    update();
}
void AnalogClock::paintEvent( QPaintEvent * )
{
    QCOORD hourHand[8] = { 2, 0, 0, 2, -2, 0, 0, -25 };
    QCOORD minuteHand[8] = { 1, 0, 0, 1, -1, 0, 0, -40 };
    QTime time = QTime::currentTime();
    QPainter painter( this );
    painter.setWindow( -50, -50, 100, 100 );
    painter.setBrush( black );
    for ( int i = 0; i < 12; i++ )
```

```
    {
        painter.drawLine( 44, 0, 46, 0 );
        painter.rotate( 30 );
    }
    painter.save();
    painter.rotate( 30 * (time.hour() % 12) + time.minute() / 2 );
    painter.drawConvexPolygon( QPointArray(4, hourHand) );
    painter.restore();
    painter.save();
    painter.rotate( 6 * time.minute() );
    painter.drawConvexPolygon( QPointArray(4, minuteHand) );
    painter.restore();
}
```

构造函数设置窗口的尺寸大小为100 x 100，并且告诉系统每隔12秒调用一次timerEvent()函数，从而对模拟钟的窗体进行刷新。

在 timerEvent()函数中， 通过调用QWidget的函数 update()就可以告诉Qt，窗体需要立即重画，紧接着Qt就会产生一个绘制事件并且调用paintEvent()函数。

在paintEvent()函数中，一个Qpainter对象用于在窗体上绘制12个刻度以及分针，时针。Qpainter类提供了一种统一的方式用于绘制窗体，位图，矢量图等，它提供了绘制点，线，椭圆，多边形，弧，贝塞尔曲线等功能，一个Qpainter的坐标系可以被转变，缩放，旋转，和剪切，这样对象就可以根据它在窗口或者窗体上的位置绘制出一个剪切的视图。剪切可以使窗体绘制时减少闪烁。使用QPainter 的子类QdirectPainter可以锁定和直接访问帧缓冲区域。

文件 analogclock.h 和 analogclock.cpp 完全的定义和实现了AnalogClock 客户窗体类，这个窗体是现在就可以使用的。

```
#include <qapplication.h>
#include "analogclock.h"
int main( int argc, char **argv )
{
    QApplication app( argc, argv );
    AnalogClock *clock = new AnalogClock;
    app.setMainWidget( clock );
    clock->show();
    return app.exec();
}
```

2．3．5 主窗口

QMainWindow类是为应用的主窗口提供一个摆放相关窗体的框架。

一个主窗口包含了一组标准窗体的集合。主窗口的顶部包含一个菜单栏，它的下方放置着一个工具栏，工具栏可以移动到其它的停靠区域。主窗口允许停靠的位置有顶部，左边，右边和底部。工具栏可以被拖放到一个停靠的位置，从而形成一个浮动的工具面板。主窗口

的下方，也就是在底部的停靠位置之下有一个状态栏。主窗口的中间区域可以包含其它的窗体。提示工具和“这是什么”帮助按钮以旁述的方式阐述了用户接口的使用方法。

对于小屏幕的设备，使用Qt图形设计器定义的标准的Qwidget模板比使用主窗口类更好一些。典型的模板包含有菜单栏，工具栏，可能没有状态栏（在必要的情况下，可以用任务栏，标题栏来显示状态）

2．3．6 菜单

弹出式菜单QpopupMenu类以垂直列表的方式显示菜单项，它可以是单个的（例如上下文相关菜单），可以以菜单栏的方式出现，或者是别的弹出式菜单的子菜单出现。

每个菜单项可以有一个图标，一个复选框和一个加速器（快捷键），菜单项通常对应一个动作（例如存盘），分隔器通常显示成一条竖线，它用于把一组相关联的动作菜单分立成组。

下面是一个建立包含有 New，Open 和 Exit 菜单项的文件菜单的例子。

```
QPopupMenu *fileMenu = new QPopupMenu( this );
fileMenu->insertItem( "&New", this, SLOT(newFile()), CTRL+Key_N );
fileMenu->insertItem( "&Open...", this, SLOT(open()), CTRL+Key_O );
fileMenu->insertSeparator();
fileMenu->insertItem( "E&xit", qApp, SLOT(quit()), CTRL+Key_Q );
```

当一个菜单项被选中，和它相关的插槽将被执行。加速器(快捷键)很少在一个没有键盘输入的设备上使用，Qt/Embedded 的典型配置并未包含对加速器的支持。上面出现的代码“&New”意思是在桌面机器上以“New”的方式显示出来，但是在嵌入式设备上，它只会显示为“New”。

QmenuBar类实现了一个菜单栏，它会自动的设置几何尺寸并在它的父窗体的顶部显示出来，如果父窗体的宽度不够宽以致不能显示一个完整的菜单栏，那么菜单栏将会分为多行显示出来。Qt内置的布局管理能够自动的调整菜单栏。

Qt的菜单系统是非常灵活的，菜单项可以被动态的使能，失效，添加或者删除。通过子类化QcustomMenuItem，我们可以建立客户化外观和功能的菜单项。

2．3．7 工具栏

QtoolButton类实现了一个带有图标，3维边框和可选标签的工具栏按钮。切换工具栏按钮具有开、关的特征，其它的按钮则执行一个命令。不同的图标用来表示按钮的活动，无效、使能模式，或者是开或关的状态。如果你仅为按钮指定了一个图标，那么Qt会使用可视提示来表现按钮不同的状态，例如按钮失效时显示灰色。

工具栏按钮通常以一排的形式显示在工具栏上，对于一个有几组工具栏的应用，用户可以随便的到处移动这些工具栏，工具栏差不多可以包含所有的窗体，例如 QComboBoxes 和 QspinBoxes。

2．3．8 旁述

现代的应用主要使用旁述的方式去解释用户接口的用法。Qt 提供了两种旁述的方式：

“提示栏”和“这是什么”帮助按钮。

“提示栏”是小的，通常是黄色的矩形，当鼠标在窗体的一些位置游动时它就会自动出现。它主要用于解释工具栏按钮，特别是那些缺少文字标签说明的工具栏按钮的用途。下面就是如何设置一个“存盘”按钮的提示的代码。

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

当提示字符出现之后，你还可以在状态栏显示更详细的文字说明。

对于一些没有鼠标的设备（例如那些使用触点输入的设备），就不会有鼠标的光标在窗体上进行游动，这样就不能激活提示栏。对于这些设备也许就需要使用“这是什么”帮助按钮，或者使用一种姿态来表示输入设备正在进行游动，例如用按下或者握住的姿态来表示现在正在进行游动。

“这是什么”帮助按钮和提示栏有些相似，只不过前者是要用户点击它才会显示旁述。在小屏幕设备上，要想点击“这是什么”帮助按钮，具体的方法是，在靠近应用的 X 窗口的关闭按钮“x”附近你会看到一个“？”符号的小按钮，这个按钮就是“这是什么”帮助按钮。一般来说，“这是什么”帮助按钮按下后要显示的提示信息应该比提示栏要多一些。下面是设置一个存盘按钮的“这是什么”文本提示信息的方法:

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

QToolTip 和 QWhatsThis 类提供了虚拟函数以供开发者重新实现更多的特定的用途。

Qtopia并未使用上述提及的两种帮助（旁述）机制。它在应用窗口的标题栏上放置一个“？”符号的按钮来代替上述的旁述机制，这个“？”按钮可以启动一个浏览器来显示和当前应用相关的HTML页面。Qtopia使用按下和握住的姿态来调用上下文菜单（右击）和属性对话框。

2．3．9 动作

应用程序通常提供给用户几种不同的方式去执行特别的动作。例如，大部分应用提供了一个“Save”动作给用于存盘的菜单(File|Save)以及工具栏（一个“软盘”图标的工具栏按钮）和快捷键(Ctrl+S)。Qaction类可以让上述过程变得简洁，它允许程序员在一个地方定义一个动作，然后把这个动作加入到菜单或者工具栏，这个过程与把菜单项加入到菜单的道理是一样的。

下面的代码实现了一个“Save”菜单项和一个“Save”工具按钮，旁述系统和快捷键可以很容易的添加进去，但是我们没添加，因为它们很少在嵌入式设备上使用。

```
QAction *saveAct = new QAction( this );
saveAct->setText( "Save" );
saveAct->setIconSet( QPixmap("save.png") );
connect( saveAct, SIGNAL(activated()), this, SLOT(save()) );
saveAct->addTo( fileMenu );
saveAct->addTo( toolbar );
```

除了不用复制代码，使用 Qaction 还可以确保菜单的状态与工具栏按钮的状态保持一致，必要的时候还可显示提示栏。使一个动作（Action）失效将导致和该动作相关联的菜单项以及工具按钮的失效。同样的，如果用户切换一个工具按钮的状态，那么相关的菜单项的也会跟着被选中或不选中。

2．4 对话框

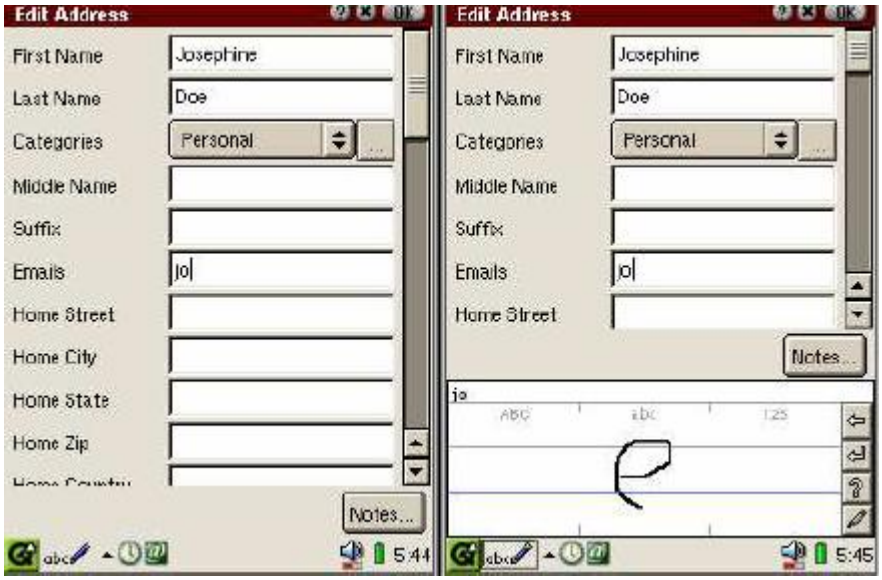
使用 Qt 图形设计器这个可视化设计工具用户可以建立自己的对话框。Qt 使用布局管理自动的设置窗体与别的窗体之间相对的尺寸和位置，这样可以确保对话框能够最好的利用屏幕上的可用空间。使用布局管理意味着按钮和标签可以根据要显示的文字自动的改变自身大小，而用户完全不用考虑文字是那一种语言。

2．4．1 布局

Qt 的布局管理用于组织管理一个父窗体区域内的子窗体。它的特点是可以自动的设置子窗体的位置和大小，并可判断出一个顶级窗体的最小和缺省的尺寸，当窗体的字体或内容变化后，它可以重置一个窗体的布局。

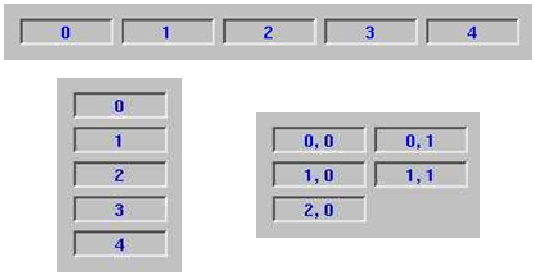
使用布局管理，开发者可以编写独立于屏幕大小和方向之外的程序，从而不需要浪费代码空间和重复编写代码。对于一些国际化的应用程序，使用布局管理，可以确保按钮和标签在不同的语言环境下有足够的空间显示文本，不会造成部分文字被剪掉。

布局管理使得提供部分用户接口组件，例如输入法和任务栏变得更容易。我们可以通过一个例子说明这一点，当Qtopia的用户正在输入文字时，输入法会占用一定的文字空间，应用程序这时也会根据可用的屏幕尺寸的变化调整自己。



图十 Qtopia 的布局管理

Qt提供了三种用于布局管理的类：QHBoxLayout, QVBoxLayout 和 QgridLayout。



图十一 QHBoxLayout, QVBoxLayout 和 QGridLayout 的布局效果

QHBoxLayout 布局管理把窗体按照水平方向从左至右排成一行
QVBoxLayout 布局管理把窗体按照垂直方向从上至下排成一列
QGridLayout 布局管理以网格的方式来排列窗体，一个窗体可以占据多个网格。

在多数情况下，布局管理在管理窗体时执行最优化的尺寸，这样窗口看起来就更好看而且可以尺寸变化会更平滑。使用以下的机制可以简化窗口布局的过程：

1、 为一些子窗口设置一个最小的尺寸，一个最大的或者固定的尺寸。

2、 增加拉伸项（stretch items）或者间隔项（spacer item)。拉伸项和间隔项可以填充一个排列的空间。

3、 改变子窗口的尺寸策略，程序员可以调整窗体尺寸改变时的一些策略。子窗体可以被设置为扩展，紧缩和保持相同尺寸等策略。

4、 *改变子窗口的尺寸提示。QWidget::sizeHint() 和 QWidget::minimumSize-Hint() 函数返回一个窗体根据自身内容计算出的首选尺寸和首选最小尺寸,我们在建立窗体时可考虑重新实现这两个函数。*

5、设置拉伸比例系数。设置拉伸比例系数是指允许开发者设置窗体之间占据空间大小的比例系数，例如我们设定可用空间的 2/3 分配给窗体 A，剩下的 1/3 则分配给窗体 B。

布局管理也可按照从右至左，从下到上的方式来进行。当一些国际化的应用需要支持从右至左阅读习惯的语言文字（例如阿拉伯和希伯来）时，使用从右至左的布局排列是更方便的。

布局是可以嵌套的和随意进行的。下面是一个对话框的例子，它以两种不同尺寸大小来显示：

图十二 小的对话框和大的对话框

这个对话框使用了三种排列方式。QVBoxLayout管理一组按钮，QHBoxLayout管理一个显示国家名称的列表框和右边那组按钮，QVBoxLayout管理窗体上剩下的组件“Now please select a country”标签。在“< Prev”和“Help”按钮之间放置了一个拉伸项(stretch items)，使得两者之间保持了一定比例的间隔。

建立这个对话框窗体和布局管理的实现代码如下：

```
QVBoxLayout *buttonBox = new QVBoxLayout( 6 );
buttonBox->addWidget( new QPushButton("Next >", this) );
buttonBox->addWidget( new QPushButton("< Prev", this) );
buttonBox->addStretch( 1 );
buttonBox->addWidget( new QPushButton("Help", this) );
QListBox *countryList = new QListBox( this );
countryList->insertItem( "Canada" );
/* … */
countryList->insertItem( "United States of America" );
QHBoxLayout *middleBox = new QHBoxLayout( 11 );
middleBox->addWidget( countryList );
middleBox->addLayout( buttonBox );
QVBoxLayout *topLevelBox = new QVBoxLayout( this, 6, 11 );
topLevelBox->addWidget( new QLabel("Now please select a country", this) );
topLevelBox->addLayout( middleBox );
```

使用 Qt 图形设计器设计的这个对话框，显示如下

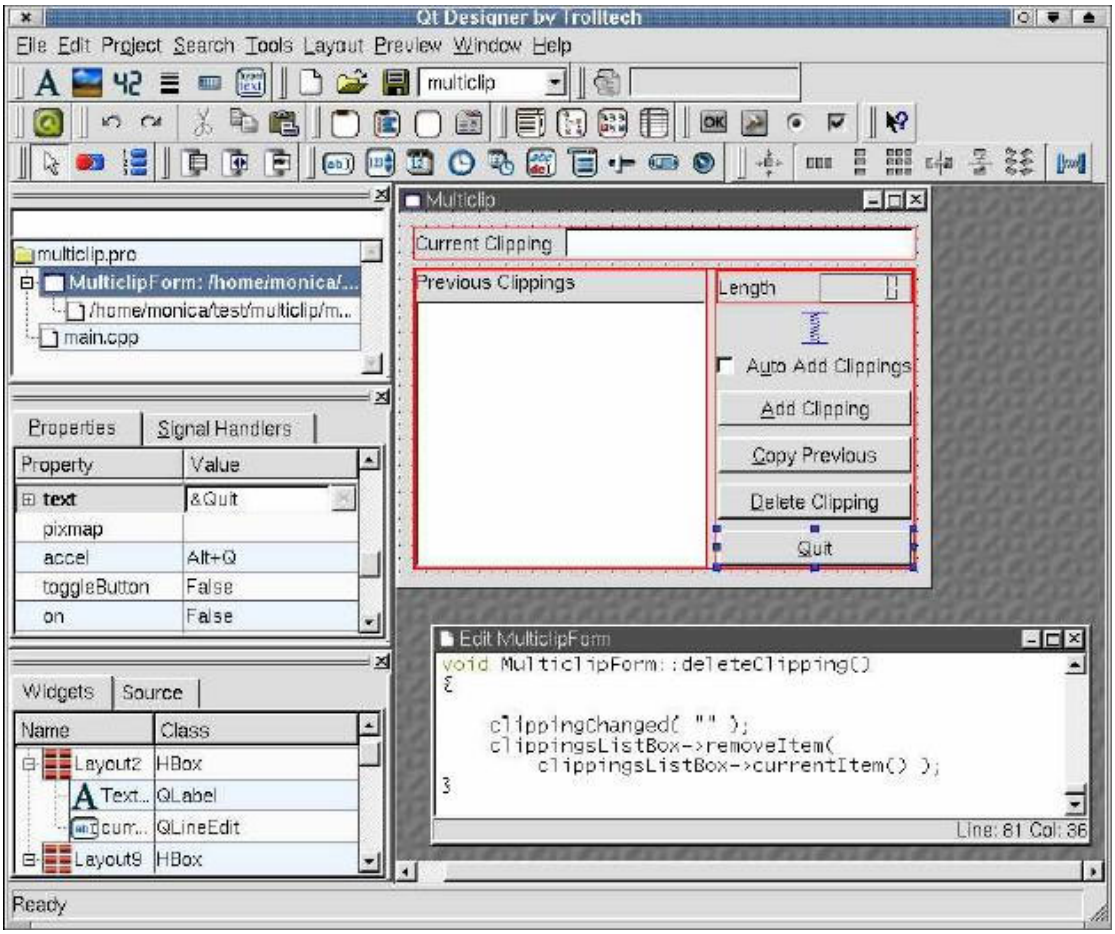
图十三 Qt 图形设计器中使用了布局的对话框

2. 4. 2 Qt 图形设计器

Qt 图形设计器是一个具有可视化用户接口的设计工具。Qt 的应用程序可以完全用源代码来编写，或者使用 Qt 图形设计器来加速开发工作。启动 Qt 图形设计器的方法是：在 Linux 命令模式下，键入以下命令(假设 Qt X11 安装在/usr/local 下)：

```
cd qt-2.3.2/bin
  ./designer
```

这样就可以启动一个与 Windows 下的 Delphi 相类似界面。 下图是使用 Qt 图形设计器设计一个表单的截屏图。



图十四 Qt 图形设计器

开发者点击工具栏上的代表不同功能的子窗体/组件的按钮，然后把它放到一个表单上面，这样就可以把一个子窗体/组件放到表单上了。开发者可以使用属性对话框来设置子窗体的属性。精确的设置子窗体的位置和尺寸大小是没必要的。开发者可以选择一组窗体，然后对他们进行排列。例如，我们选定了一些按钮窗体，然后使用“水平排列（lay out horizontally)”选项对它们进行一个接一个的水平排列。这样做使得设计工作变得更快，而且完成后的窗体将能够按照属性设置的比例填充窗口的可用尺寸范围。

使用 Qt 图形设计器进行图形用户接口的设计可以消除应用的编译，链接和运行时间，同时使得修改图形用户接口的设计变得更容易。Qt 图形设计器的预览功能可以使开发者能够在开发阶段看到各种样式的图形用户界面，包括客户样式的用户界面。通过 Qt 集成的功能强大的数据库类，Qt 图形设计器还可提供生动的数据库数据浏览和编辑操作。

开发者可以建立同时包含有对话框和主窗口的应用，其中主窗口可以放置菜单，工具栏，旁述帮助等等的子窗口部件。Qt 图形设计器提供了几种表单模板，如果窗体会被多个不同的应用反复使用，那么开发者也可建立自己的表单模板，以确保窗体的一致性。

Qt 图形设计器使用向导来帮助人们更快更方便的建立包含有工具栏、菜单和数据库等方面的应用。程序员可以建立自己的客户窗体，并把它集成到 Qt 图形设计器中。

Qt 图形设计器设计的图形界面以扩展名为“ui”的文件进行保存，这个文件有良好

的可读性，这个文件可被 uic（Qt 提供的用户接口编译工具）编译成为 C＋＋的头文件和源文件。Qmake 工具在它为工程生成的 Makefile 文件中自动的包含了 uic 生成头文件和源文件的规则。

另一种可选的做法是，在应用程序运行期间载入 ui 文件，然后把它转变为具备原先全部功能的表单。这样开发者就可以在程序运行期间动态修改应用的界面，而不需重新编译应用，另一方面，也使得应用的文件尺寸减小了。

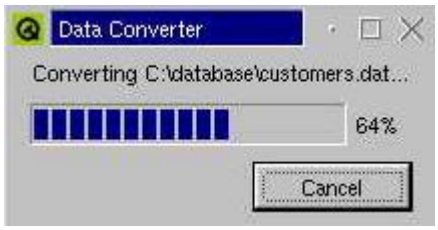
2. 4. 3 建立对话框

Qt 为许多通用的任务提供了现成的包含了实用的静态函数的对话框类，下边是一些 Qt 的标准的对话框的截屏图。

QmessageBox类是一个用于向用户提供信息或是给用户进行一些简单选择（例如“yes”或“no”）的对话框类。

图十五 一个 QMessageBox 对话框

progressDialog对话框包含了一个进度栏和一个“Cancel”按钮



图十六 一个 QprogressDialog 对话框

Qwizard类提供了一个向导对话框的框架

图十七 一个向导类

Qt提供的对话框还包括QColorDialog, QFileDialog, QFontDialog 和QprintDialog。这些类通常适用于桌面应用，一般不会在Qt/Embedded中编译使用它们。

2．5 外形与感觉

Qt桌面应用随着执行环境（例如*Windows XP, Mac OS X, Linux*）的不同而具有不同的样式，或者叫做外形与感觉。Qt/Embedded 的应用可以使用不同操作环境提供的样式，也可以以静态或插件的方式使用客户样式。开发者可以设计自己窗体的样式和窗口装饰。

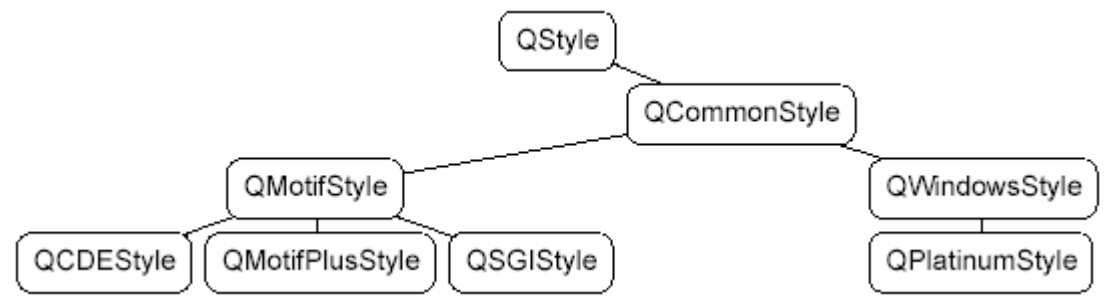
2．5．1 窗体样式

一个样式是一个实现Qt窗体外形与视觉效果的类Qstyle的子类。Qt/Embedded 程序员可以自由的使用和修改目前存在的样式，也可以使用Qt样式设计引擎实现自己的样式。目前Qt/Embedded提供的样式类型有EmbeddedareWindows, Motif, MotifPlus,CDE, Platinum 和 SGI。样式可以被动态的设置到一个应用中，甚至可以设置到特定的窗体上。

图十八 以不同样式建立的下拉框

通过给一组相关的应用编写一个客户样式，可以让这些应用的外观具有与众不同的感觉。建立客户样式可以通过子类化QStyle，QcommonStyle或者QcommonStyle的派生类来实现。通过重新实现现有样式基类的一两个虚函数可以很容易对现有样式做一些小的修改。

一个样式可以编译成为一个插件，在 Qt 图形设计器中，把样式做成插件的话，开发者就可以以设备的客户样式来预览设计出来的表单。样式插件使得开发者不需重新编译就可改变设备的外观。

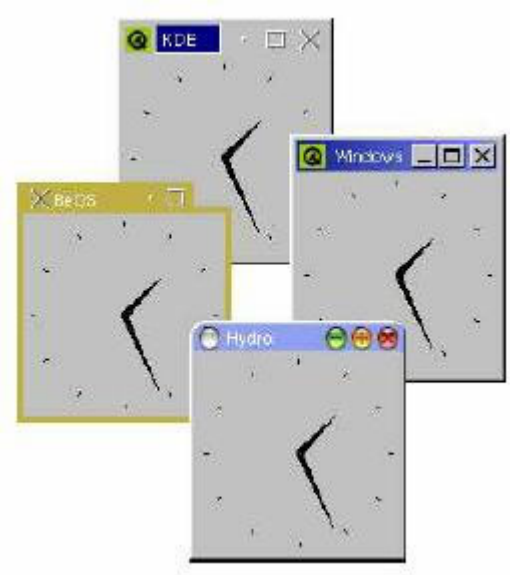


图十九 Qstyle 的继承图

Qt发布的窗体都是样式已知的，当它们的样式改变后，窗体会自动重绘。客户窗体和对话框大多数都是Qt原先发布的窗体和布局的重新整合。它们也是自动的获得原先的样式。在少数场合，当有必要写一个外观上很随意的窗体时，开发者可以使用QStyle去写一些用户接口元素，而不是直接的绘制固有的矩形。

2．5．2 窗口装饰

顶级窗口的装饰是由一个标题栏和一个框架组成的。Qt/Embedded 包括了这些窗口样式：BeOS, Hydro, KDE 和 Windows。



图二十 不同的平台上的窗口风格

如果需要的话，我们可以配置不同的窗口使用不同的窗口装饰。通过子类化 QWSDecoration,我们可以建立客户的装饰样式，并且把它们以插件的形式进行发布。为

了在窗口管理器之外进行更多的控制行为，开发者需要子类化QWSManager。

2．6 国际化

Qt/Embedded完全支持Unicode，一个国际标准的字符集。开发者在他们的应用中可以自由的混合使用被Unicode字符集支持的语言，例如阿拉伯文，英文，中文，希伯来文，日文和俄文等。为了有助于公司将产品推向国际市场，Qt还提供了将应用翻译成支持多种语言环境的工具。

2．6．1 Unicode

Qt 使用 **QString** 存储 Unicode 编码的字符串，Qstring 取代了粗糙的 const char *；它提供了用于处理 **Qstring** 和 const char *之间相互转换的构造函数和操作符。因为 Qt 使用了隐式共享(写时复制)技术来减少内存的使用,所以直接复制 Qstring 的值是不会产生问题的。为有效率的存储 ASII 码字符串，Qt 还提供了 **QCString 类。**

Qt 为所有要显示在屏幕上的文本，包括最简单的文字标签到最复杂的宽文本编辑器，提供了一个强大的 Unicode 文本呈现引擎。这个引擎支持一些先进的特征，例如特殊的间隔线、双向写和区别标记。它几乎支持世界上所有的书写系统，包括阿拉伯文，中文，古斯拉夫文，英文，希腊文，希伯来文，日文，韩文，拉丁和越南文。体现这个引擎的最优化性能的常用的例子就是：在带有加速功能的文字的下方显示一条下划线（例如 File）。

QtextCodec 的子类用于处理不同编码类型的字符集之间的转换。Qt 3.0 支持 37 种不同的编码方式，包括中文的 Big5 和 GBK,日文的 EUC-JP、JIS 和 Shift-JIS，俄罗斯的 KOI8-R 和 ISO 8859 系列。它们可以以库的一部分或者插件的形式编译，或者使用“feature”机制去除这些编译。

2．6．2 应用的翻译

Qt 提供了相应的工具和函数用于帮助开发者以他们的本地语言推出应用。

要使一个字符串可以被翻译，你需要把这个字符串作为一个参数放到 tr（）函数中调用。例如：

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

Qt 尝试寻找字符串"Save"的译文，如果找到的话，就会把它的译文显示出来，如果找不到，就用原来的字符串"Save"进行显示。例如我们把英文作为源语言，现在我们要把这个源语言翻译成中文，即中文为翻译的目标语言，以此类推。这样当我们调用 tr（）函数后，函数的参数的原先的缺省编码就会转变为 Unicode 编码。Tr()函数的通常用法为：

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

上面的 “Context” 是指一个 QObject 对象的子类的名称。如果在一个包含了 tr（）成员

函数的类的上下文环境中使用 tr（）函数时，“Context”通常可以省略掉。例如：

```
Context：：func1（）
{
    setText( tr("Save") );
}
```

“source text”是指要翻译的文本的内容
“comment”是一个可选项，它用于给手工翻译者提供一些额外的信息。

说了半天，还有一个重要的内容我们没谈，就是 tr（）函数如何寻找到译文。我们要把一个源字符串翻译为和它对应的译文（目标语言的字符串）时，我们需要让 Qt 知道这些译文放在哪里。Qt 规定了译文存储在 QTranslator 对象中，这个对象是从一个内存映射文件（扩展名为 pm）中读取译文。每个 pm 文件包含了某种语言的译文信息。所以开发者需要建立一个 pm 文件来存储应用中需要翻译的字符串的译文。

Qt 提供了 3 种工具帮助人们建立译文存储文件(.pm 文件)，这 3 个工具是 lupdate, *Qt Linguist* 和 lrelease.

1、lupdate 自动地从源代码文件（.cpp）和界面接口文件（.ui）中获取所有需要翻译的对象，即上述的（“Context”）；同时还获取要翻译的所有的字符串（源语言的字符串），即上述的“source text”；“comment”选项如果被使用的话，也会被纳入收集的范围。当这些信息收集完毕后，lupdate 最后会生成一个.ts 文件（翻译源文件），这个文件是直接可阅读的。

2、开发者使用 *Qt Linguist 工具提供的良好的人机界面打开*一个.ts 文件（翻译源文件），然后开发者根据每一个 source text 填写上相对应的译文，这样一个.ts 文件（翻译源文件）就包含了完整的“Context”，“source text”和译文信息。

3、最后通过运行 lrelease 去把一个.ts 文件（翻译源文件）压缩为一个.pm 文件（译文存储文件），生成的.pm 文件可用在嵌入式设备上。

在一个应用的生存期间里，上述的步骤有可能根据需要会被反复执行。多次运行 lupdate 是很安全的，你可以重复使用已经存在的译文（.pm）文件，或者当你不想使用某些翻译文件时你可以标记这些翻译源文件为旧文件，而不需要删除它们。