**QT widget，MainWindow和Dialog的选择使用**

**Chapter 1 Base Class**

Qt中的每个类，都有一个对应的同名头文件，其中包含其类定义。例如要使用QApplication类，则需要在程序中添加" #include "

QApplication类用于管理应用程序范围内的资源。其构造函数需要main函数的argc和argv作为参数。

widget被创建时都是不可见的(always created hidden)。widget中可容纳其它widget。

Qt中的widget在有用户行为或状态改变时会emit signal。 signal可以和slot函数连接在一起(connect)，这样当有signal被emit时，对应的slot函数会被自动调用。

QWidget类的构造函数需要一个 QWidget \* 指针作为参数，表示其parent widget(默认值为0，即不存在parent widget)。在parent widget被删除时，Qt会自动删除其所有的child widget。

Qt中有三种Layout Manager 类： QHBoxLayout，QVBoxLayOut，QGridLayOut。基本模式是将widget添加进LayOut，由Layout自动接管widget的尺寸和位置。

启动Qt程序时可以通过 -style 参数改变程序的默认显式风格。

**Chapter 2 Creating Dialogs**

**2.1 Subclassing Dialog**

Qt中所有dialog的基类是QDialog。QDialog派生自QWidget。

Qt中所有定义了signal或slot的类，在其类定义的开始处都要使用Q\_OBJECT宏。Qt中的signal关键字实际上是宏定义。类似的，slots关键字也是宏定义。

Qt所提供的类分为若干模块：QtGui，QtNetWork，QtOpenGL，QtSql， QtSvg和QtXml等。

QObject::tr() 函数将输入的字符串转换为其他语言(国际化)。对所有用户可见的字符串都使用tr()函数是一个良好的习惯。

Buddy：两个widget A和B，若A拥有快捷键，当用户按下该快捷键时，程序的输入焦点自动转移到B上，则称B是A的buddy。

QWidget::close() 是一个slot，其默认行为是使对应的widget隐藏不可见，但并不删除该widget。

Layout 中可包含widget和其他layout。通过嵌套使用QHBoxLayOut、QVBoxLayOut、QGridLayOut，可以构造非常复杂的dialog。值得注意的是：layout manager 类并不属于widget。实际上，它派生自QLayout，而QLayout又派生自QObject。

QWidget::sizeHint() 返回一个widget()的理想大小(ideal size)。

emit关键字是Qt特有的，用于产生signal。

MOC(Meta-Object-Compiler): 对于所有使用了Q\_OBJECT宏的类，在编译时都需要通过MOC的处理，否则会出现链接错误。解决该错误的办法也很简单，重新执行qmake以更新makefile，然后重新编译。

**2.2 Signal and Slot in Depth**

Signal & Slot 机制是Qt的根基。

Slot和普通的C++类成员函数几乎完全一致；可以是virtual的，可以被重载，可以是public、protected或private的，而且也可所以被其他成员函数直接调用。

signal与slot之间的关联可以是一对一、一对多或多对一。

signal和signal之间也可以被关联，此种情况与signal-slot的区别在于，当第一个sigal被emit时，第二个signal也被emit。

可以调用disconnect()来解除signal 与slot之间的关联，通常情况下很少需要显式调用disconnect()，因为对象被删除时Qt会自动移除与其相关的关联。

sigal-slot或signal-signal这样的关联，要求二者具备相同的参数列表；若signal比slot中的参数多，多余的参数会被忽略。

思维定势：signal-slot机制只能用于widget。实际上signal-slot机制是由QObject实现的，并不仅局限于GUI编程，可以用于任何QObject子类。

**2.3 Rapid Dialog Design**

使用Qt Designer创建的form最终被转换为C++代码。

qmake工具能够检测到interface file(\*.ui files)，并调用uic，即Qt的user interface compiler。uic将.ui文件转换为C++代码，并存放在形式为ui\_xxx.h的文件中。该文件中给出了dialog对应类的完整定义，并包含一个 setupUi()成员函数，用于初始化form。

注意，由uic创建的这个类未派生自任何Qt class。

Qt的 parent-child机制是由QObject实现的。当创建一个对象时若指定了parent，则parent将该对象添加至其 childern list。当parent被删除时，Qt会遍历其childern list并删除每个child，该过程会递归进行。这一机制极大的简化了内存管理，降低了内存泄露的风险——程序员只需显式的删除通过new创建并且没有parent的对象。

对于widget，parent还有一层附加的意义：chidl widget 是显示在parent widget的范围之内的。如果删除parent widget，不仅child widget从内存中被释放，在屏幕上也会消失。

QDialog::accept() 将dialog的返回值设为QDialog::Accepted(值为1)，而QDialog::reject()将返回值设为QDIalog::Rejected(值为0)。

**2.5 Dynamic Dialogs**

Dynamic Dialog指的是程序在运行时根据.ui文件创建的dialog。这样的dialog不是通过uic将.ui转换为C++代码，而是在运行时使用QUiLoader类装载.ui文件。

可以使用QObject::findChild ()来访问form的child widget。

要使用QUiLoader，需要在Qt程序的.pro文件中添加以下内容： CONFIG += uitools

Dynamic dialog允许在不重新编译程序的前提下更改form的布局。

**Chapter 3 Creating Main Windows**

**3.1 Subclassing QMainWindow**

应用程序的主窗口是通过创建QMainWindow的派生类来完成的。QMainWindow和QDialog一样，都是派生自QWidget。

closeEvent()是由QWidget提供的一个虚函数，在用户关闭窗口时会被自动调用。

setCentralWidget()将某个Widget设置为主窗口的central widget， 而central widget意味着在显示时会占据主窗口的中央位置。

Qt下的GUI 编程支持多种图形格式。可以使用多种方式为应用程序提供图像，最常见的包括：

1). 将图像存储在文件中，运行时加载之。

2). 在源码中include XPM文件(XPM文件也是合法的C++文件)。

3). 利用Qt的资源机制。

Qt的资源机制比之运行时加载更方便，并对所有支持的图像格式都能良好工作。

为了利用Qt的资源机制，需要创建一个资源文件，并在.pro文件中对应添加一行来对资源文件进行标识。例如：

RESOURCES＝ spreadsheet.qrc

资源文件本身采用了简单的XML格式。它被编译进程序的可执行文件，因此不会被丢失。在对资源进行定位时，使用路径前缀":/"，例如“ :/images/icon.png "。资源本身可以是任何类型的文件。

**3.2 Creating Menus and Toolbars**

Qt通过引入Action这一概念简化了对menu和toolbar的编程。一个Action可以被添加到任意数量menu和toobar中。

在Qt中对menu和toolbar的编程涉及到三个步骤：

1). 创建并设置Action

2). 创建menu，并在其中添加Action

3). 创建toolbar，并在其中添加Action

Action的创建是通过QAction类来实现的，对每个Action，可以为其设置accelerator，parent，shortcut key， 可见性以及status tip等属性，并可以通过调用connect()为ACtion设置被触发要执行的操作。

QTableWidget的基类QAbstraceItemView提供了selectAll()这个slot。

QApplication类提供了aboutQt()这个slot，可以通过全局变量qApp(一个类型为QApplication \*的指针)来使用之。

在Qt中，menu由QMenu类的实例表示。而Qmenu是要被放入QMenuBar之中的。函数QMainWindow::menuBar()返回一个类型为QMenuBar \* 的指针。QMenuBar::addMenu()根据指定文本创建一个QMenu widget并将其添加进MenuBar中。QMenu::addAction() 则为Menu添加Action。

任意Qt Widget都可以具备相关的一系列QAction。通过调用QWidget::addAction() ，可以为Widget添加Action。这一特性可用来创建上下文菜单。

**3.3 Setting Up the Status Bar**

QMainWindow::statusBar() 返回一个指向status bar的指针；status bar 在statusBar()第一次被调用时被创建。

**3.4 Implementing The Menu**

QMessageBox::Defalut修饰符使得被修饰的Button成为默认Button，而QMessage::Escape修饰符则使得Esc键自动触发被修饰的Button。

QMessageBox::warning()用于弹出提示对话框。该函数属于Qt提供的static convenicence function

static convenience function

QFileDialog::getOpenFileName() 可用于从用户处获得文件名——该函数弹出一个文件选择对话框，要求用户选择一个文件，并返回文件名，或者在用户选择"Cancel"时返回空字符串。该函数的第一个参数是其parent widget。对于dialog和其他widget，parent-child关系的意味是不完全相同的。一个dialog永远是一个独立的窗口，但是如果它拥有parent，则默认在parent之上居中显示。

当用户发出关闭窗口的操作时，Qwidget::close() 这个slot会被调用，该slot向对应的widget发送close event。重新实现QWidget::closeEvent()能够拦截这个event，以便确定是否真的要关闭窗口，防止误操作。

每个QWidget都有一个windowModified属性，在窗口文档被修改时应该被设为True，否则被设为false。

QString::arg() 函数将字符串中编号最低的"%n"用参数进行替换，并返回替换后的字符串。

每个Action都可以拥有一个类型为QVariant的关联数据。

Qt中的qobject\_cast() 机制对于动态库也可以正常工作。

**3.5 Using Dialog**

modeless window——one that runs independently of any other windows in the application

对于modeless dialog ，当其被弹出时，可能处于三种情况：

1). 这是该对话框第一次被激活

2). 该对话框之前曾被激活，但用户又将其关闭

3). 该对话框之前曾被激活，而且仍可见

show() 将一个隐藏窗口变为可见，而activateWIndow()则将窗口的状态变为active。

model window——pops up when invoked and blocks the application，preventing any other processing or interactions until it is closed.

一个dialog若是用show()来激活，则是modeless dialog；若通过exec()来激活，则是model dialog。此外，还可以调用setModel()来设置dialog的显示模式。

QDialog::exec() 的返回值在dialog被确认时为true，否则为false。

在栈上创建 model dialog是一种良好的编程惯例，因为在使用完后就不再需要，而model dialog会在作用域结束后自动被销毁。

由于多数应用程序的About box 都是高度雷同的，Qt中提供了一个方便的static convenicence function QMessage::about()，该函数和QMessageBox::warning()很相似。

**3.6 Storing Setting**

Qt中是通过QSettings类来将应用程序的设置信息存储到平台相关的位置——windows下存入注册表中，unix中存在文本文件中。

QSettings的构造函数包含两个参数，分别是organization's name 和 application's name ，Qt使用这两个参数来对应用程序的设置信息进行定位。

QSettings以key-value pair的形式存储信息。

**3.7 Multiple Documents**

要想实现多文档程序，首先必须要通过new在堆上创建主窗口，而不是在栈上创建主窗口。

QAplication::closeAllWindows() 这个slot完成的操作是关闭应用程序所有的窗口，除非其中某个窗口拒绝了close event。程序员不需要担心未保存的修改，因为这会由QWidget::closeEvent()负责处理。

通过在MainWindow的构造函数中调用setAttribute()函数来设置Qt::WA\_DeleteOnClose属性，可以要求Qt在窗口被关闭时将其自动销毁

Qt在其可用所有平台上都支持SDI和MDI程序的创建。

**3.8 Splash Screnns**

在Qt中为程序添加splash screen非常简单，可通过QSplashScreen类来实现。

通常情况下，与splash screen相关的代码都放在main()中，出现在调用QApplication::exec()之前。

**Chapter 4 Implementing Application Functionality**

**4.1 The Central Widget**

QMainWindow的中央区域可以被任何类型的widget占据。

**4.2 Subclassing QTableWidget**

QTableWidget会自动创建QTableWidgetItem来存储用户的输入。

QTableWidgetItem类并不是widget，而是一个纯粹的data class。

QTabeWidget::setItermProtype()可以设置在获得用户输入的情况下自动创建哪种class。

**4.3 Loading and Saving**

QFile ＆ QDataStream

QFile的析构函数负责将打开的文件关闭。

QDataStream类具有很强的通用性，可作用于QFile，QBuffer，QProcess，QTcpSocket，QUdpSocket。

Qt还提供了一个QTextStream类用于专门读写文本文件。

**4.6 Subclassing QTableWidgetItem**

每个QTableWidgetIterm中可存储若干数据，这是通过个QVariant来实现的。每一个QVariant对象都以某个role来存储某一类数据，常用的role有Qt::EditRole和Qt::DiaplayRole。

QVarinant对象可以存放多种类型的变量值，并提供向其他类型转型的函数接口。

使用默认构造函数创建的QVariant对象被视为invalid variant。

**Chapter 5. Creating Custom Widgets**

用户自定义的控件可以通过继承现有的Qt控件实现，也可以直接从QWidget继承来实现

**5.1 Customizing Qt Widgets**

**5.2 Subclassing QWidget**

通过对QWidget进行派生，并重新编写其部分event handler来进行绘图和响应用户操作，程序员可以实现对widget的外观和行为的完全控制。

Qt的内置Widget如QLabel、QPushButton、QTabelWidget等，就是以这种方式实现的。

宏Q\_PROPERTY()用来为widget声明和添加自定义属性。

每个属性的定义都对应一个数据类型(任何被QVarinat支持的类型都可以)，一个read function以及可选的write function。

对于包含自定义属性的类，Q\_OBJECT和Q\_PROPERTY()这两个宏都是必备的。

QImage类以一种硬件无关的方式存储图像信息。

Qt中提供了两个类型用于存储色彩信息：QRgb和QColor。

QRgb其实是一个typedef，用于存放32-bit的像素信息。

QColor则是一个提供了许多接口函数的类，在Qt中广泛的用于存储色彩。

QWidget::update()函数用于对widget进行强制性的重绘。

QWidget::updateGeometry()用于告知包含该widget的layout：该widget的size hint已发生变化，layout会自动进行调整。

通过调用QWidget::update()和QWidget::repaint()，可以强制性的产生一个 paint event，两者的区别在于repaint()导致立即重绘，而update()只是将一个paint event放入event queue中。

如果对update()进行连续多次调用，Qt会将连续的paint event压缩合并为一个paint event，以防止图像抖动。

每个widget都拥有一个palette，用于设置widget中在什么情况下使用什么色彩，如背景色、文本色等。

widget的palette由三个color group组成：active ，inactive ，disabled。

QWidget::palette()以QPalette的形式返回widget的palette，而clolor group则通过枚举类型QPalette::ColorGroup指定

**5.3 Intergrating Custom Widgets with Qt Designer**

要像在Qt Designer中使用自定义widget的话，必须要让Qt Designer能够了解到它们的存在。

有两种机制：promotion approach ＆plugin approach

promotion approach 很容易也很省时，但缺点是自定义widget的自定义属性在Qt Designer中是不可见和不可访问的，而使用plugin approach时则不存在这些问题。

plugin approach要求创建一个Qt Designer 可以在运行时加载的plugin library，以用于创建widget的实例。由于Qt的MOC机制，Qt Designer可以动态获取widget的property list。

要使用plugin approach ，首先要对QDesignCustomWidgetInterface进行派生，并重写某些虚函数。

Q\_INTERFACES()宏用于告知Qt该类实现了哪个interface。

在实现plugin class的源文件尾部，必须使用Q\_EXPORT\_PLUGIN2()宏使得该plugin对Qt Designer可见、可用。该宏第一个参数是plugin的名字，第二个参数是实现该plugin的class name。

**5.4 Double Buffering**

QWidget::style()返回用于绘制该widget时所使用的style。Qt中的style都是QStyle的派生类。同一应用程序中的 widget一般都使用相同的style，然而可以调用QWidget::setStyle()来进行widget层次的特别设置