Qt和MFC比较

分类： C/C++

MFC（微软基础类库）是专门为windows设计的一个用于开发图形用户界面的类库。MFC或多或少使用了面向对象的方法包装了Win32的API，正因如此，这些API有时是C++，有时是C，甚至是C和C＋＋的混合体。

Qt这个C++的图形库由Trolltech在1991年左右开发。它可以运行在Windows,Mac OS X, Unix，还有像Sharp Zaurus这类嵌入式系统中。Qt是完全面向对象的。

**Document/View model**

MFC编程需要使用Document/View模式以及模板(template),如果不使用的话，编程将变得异常困难。而且，模板(template) 设定了固定的结构，若所需结构乃模板未定义之结构，则编程难已。例如，划分一区域使显示两个视图(view)于两个文档(document)。还有一个经常的问题是：模板(template)创建了视图(view)却无法访问(access)它，文档（document）要做完所有事情，但是这经常会出现问题。

Qt不强制使用任何设计模式。如果你认为恰当，使用Document/view没有任何问题。不使用也没有任何问题。

**伪对象 vs 真对象**

归根结底，Qt和MFC的差异在于其设计的差异。

MFC的根本目的是访问包装起来的用C语言写的windows的API。这绝非好的面向对象的设计模式，在很多地方，你必须提供一个包含15个成员的C语言的struct，但是其中只有一个与你所期望的相关，或者必须用旧式的参数来调用你的函数。

Qt恰恰相反，它的架构明显是经过精心设计的面向对象的。Qt因此在命名，继承，类的组织等方面保持了优秀的一致性。你只需要提供唯一一个方法的参数，仅此一个。在不同的类中调用方式也是有很强的连贯性。返回值也很有逻辑性。所有一切达到了简单和强大的和谐统一。一旦你使用了其中一个类，其他的类也就触类旁通，因为他们是一致的。

**消息循环**

MFC是事件驱动的架构。要执行任何操作，都必须是对特定的消息作出响应。Windows对应用程序发送的

信息数以千计，遗憾的是，要分清楚这些分繁芜杂的消息是很困难的，并且关于这方面的文档并不能很好的解决这些问题。

Qt的消息机制是建立在SIGNAL()发送和SLOT()接受的基础上的。这个机制是对象间建立联系的核心机制。利用SIGNAL()可以传递任何的参数。他的功能非常的强大。可以直接大传递信号给SLOT()，因此可以清楚的理解要发生的事情。一个类所发送的信号的数量通常非常的小(4或者5)，并且文档也非常的齐全。这让你感觉到一切尽在掌握之中。SIGNAL/SLOT机制类似于Java中listener机制,不过这种机制更加轻量级，功能更齐全。

**创建界面**

MFC无法创建大小动态可变的子窗口，必须重新手动修改代码来改变窗口的位置（这恰好解释了为什么windows里的dialog是不可以改变的）这个问题在软件进行国际化翻译的时候更加严重，因为许多国家表达相同意思需要更长的词汇和句子，必须要对每个语言的版本重新修改自己的软件。

在Qt中，任何东西都可以手动的敲出来，因为它很简单：为了得到一个button，可以这样些

button = new PushButton( "buttonName", MyParentName );

如果想在按下某个按钮以后想调用某断代码的执行，可以这样写：

connect( button, SIGNAL( clicked() ), qApp, SLOT( action() ) );

Qt拥有非常简单而又不失强大的layout机制，以至于不使用它就是在浪费时间了。

Qt还提供了一个图形用户工具，QtDesigner，可以用来帮助建立用户界面。可以修改所使用的任何控件的属性。不用将他们放在严格的位置，可以通过layout完美的组织他们。这个工具所产生的代码我们是可以实际上阅读并且可以理解的。生成的代码单独放在一个文件里，在编程的同时，你可以随心所欲的多次重新生成用户界面。

Qt Designer可以让你完成许多在MFC中不可能完成的任务，比如用预先填好的生成listview，在每个tab上用不同的view来使用tab 控制。

**帮助文档**

用户选择图形开发环境的时候，帮助文档是否周全是左右其选择的重要因素。Visual的开发环境的帮助文档MSDN(这个还要单独掏钱购买)非常的庞大，有10个CDROM光盘。他包罗万象，涵盖广泛。但是难免有泥沙俱下，主题模糊，关键信息不突出的遗憾。其链接设计的也很糟糕，通过链接很难从一个类跳转到其父类或者子类以及相关的类。如果你搜索一个关键字，不管是Visual C++, Visual J++, Visual Basic，只要包含这些关键字的信息统统的返回来。

Qt的文档完备且详细的覆盖了Qt的方方面面，竟然仅有18M。每一个类和方法都被详尽描述，巨细靡遗，举例充实。通过Trolltech公司提供的链接或者是Qt Assistant工具，可以方便的从一个类或者方法跳转到其他的类。文档还包含了一个初学者教程和一些典型应用的例子。同时还提供了FAQ和邮件列表，方便通过Internet或者用户群来查阅。如果你购买了授权，在一天之内你将会得到Trolltech公司的技术支持。

**Unicode**

使用MFC，如果要显示unicode，在编译链接的时候必须用到特殊的参数(和改变可执行文件执行的入口)，必须在每个string前面加上T，将 char修改成TCHAR，每个字符串处理函数(strcpy(), strdup(),strcat()...... )都要改变成另外的函数名。更令人恼火的是支持Unicode的软件竟然不能和不支持Unicode的DLL一起工作。当使用外部DLL来开发的时候这是个很严重的问题，但是你毫无选择。

使用Qt，字符串用QString来处理，其本身是与生俱来的Unicode.不需要改变什么东西。不要在编译/链接时候增添参数，不要修改代码，只需要使用QString就可以了。

QSting类功能强大，你可以广泛的使用它，并且不要担心Unicode问题。这使得转换为Unicode非常的方便。QSting提供了转换为char ＊ 和UTF8的函数。

显然，MFC的CString的设计相比于Qt的QString设计有着巨大的不同。CString以char ＊为基础提供了很少的功能。它的优点是当需要char ＊类型的时候，可以直接使用CString类型。乍看起来这个好像是个优点，其实实质上还是有很大的缺陷的，特别是可以直接修改char \* 而不要更新类。在转变为Unicode的时候这个也碰到很大的麻烦。

相反，QString在内部以unicode存储string，需要时提供char ＊功能。实际上很少用到char ＊，因为整个Qt的API用文本的方式响应QString参数。QString还附带许多其他的功能，比如自动分享QString的内容。这是一个非常强大的类，你会喜欢在很多地方用它的。

**国际化**

使用MFC是可以国际化的，但是需要将每一个字符串放在一个字符串表中，在代码中到处使用LoadString(IDENTIFIET)。然后转化这些资源到DLL中，翻译字符串到所需要的语言，改变图形界面，然后调用程序使用这个DLL。整个过程是如此的繁琐，可谓牵一发而动全身。考虑的事情要面面俱到。

使用Qt的时候，只需要将字符串置于函数tr()中，在程序开发中这算是举手之劳。可以直接在代码中改变字符串的参考。Qt Linguist,Qt的一个工具，能够提取所有待翻译的string并按照友好的界面显示出来。这个用户界面非常适合翻译，使用字典，显示字符串内容，恰当的unicode显示，快捷方式冲突检测，检测未翻译的字符串，检测字符串修改情况，功能齐全。这个软件可以供没有任何编程经验的翻译者使用。同时该软件在GPL的版权下发布，可以按照你的需求来修改它。

翻译以后的文档保存在XML中，适合软件复用的原则。为软件增加一种新的语言版本仅仅是用QtLinguist产生一个新的文件而已。

**resources问题**

使用MFC，一部分开发过程要依靠“resources”,在很多的案例中开发者必须使用他们。这样会导致如下的后果：

出了Visual Studio，你很难使用其他的工具来完成开发。

资源编辑器仅有有限的功能，比如：通过Dialog编辑器不可能改变所有的属性，一些属性可以改变，另一些属性则不可能改变。(译者注：下面还有两条陈述MFC缺点的实例，但我感觉这些已经够说明问题了，暂时删节不译)

然而Qt并没有资源的概念，这就解决了以上所提到的问题。Qt提供了一个脚本使得能将编入你的代码。对于界面设计，Qt Designer则创建了可读的代码。

**发布**

在发布基于MFC的软件时，必须依靠存在于客户电脑上的MFC。但是这是不安全的，同样是MFC42.dll，可以基于相同的库得到3个不同的版本。通常，需要检查是否拥有正确的MFC42.dll版本，如果不是，就升级它。但是升级MFC42.dll会改变很多软件的行为。这让我感到很不舒服，如果用户在安装我的软件以后导致其机器死机该怎么办？

Qt则没有这个风险，因为Qt压根就没有“升级整个系统”这个概念。