<https://www.cnblogs.com/findumars/p/4948896.html>

Qt 是 C++ 的库，Qt 在 ansi C++ 的基础上进行了一点扩展。

但国内似乎比较浮躁，学Qt的很多连基本的C++如何编译似乎都不太清楚。**本文舍弃IDE或qmake、cmake等工具的束缚，尝试通过几个例子，一步一步从标准 C++ 的编译过渡到 Qt 的编译。**

本文涉及的都是最基本的东西，或许可以说，只要你用C++ Qt，不管是通过哪种工具(qmake、cmake、boost.build、qtcreator、vs2008、Eclipse、...)，本文的内容都是需要理解的(尽管真正写程序时，我们都不会直接用C++编译器来编译Qt程序)。

如果你对命令行比较恐惧，或许愿意先看看我原来整理的这个 [GCC新手入门](http://wiki.ubuntu.org.cn/Gcchowto)

## 例子一：简单的控制台程序

一个很简单的例子，没用到Qt扩展：(也就是说，这是一个普通的C++程序)

#include <QtCore/QCoreApplication>

#include <QtCore/QDebug>

int main(int argc, char\*\* argv)

{

QCoreApplication app(argc, argv);

qDebug()<<"hello qt!";

app.exec();

}

我们都知道，编译一个C++的程序，无非是 编译预处理，编译、链接

* 编译预处理器：头文件路径 和 必要的宏
* 编译器：一些编译参数
* 链接器：一些链接参数 和 要链接的库

### g++

简单一行命令，即可生成 main.exe (linux下，则生成可执行程序 main)

g++ main.cpp -DQT\_CORE\_LIB -Ie:\Qt\4.7.0\include -o main -Le:\Qt\4.7.0\lib -lQtCore4

单行命令，很简单：

* -I 指定头文件路径
* -L 指定库文件路径
* -l 指定需要链接的库
* -D 定义必要的宏(其实对这个小程序，这个宏也没必要用)
* -o 指定生成的可执行文件名

### cl

简单一行命令，即可生成 main.exe

cl main.cpp -ID:/Qt/4.7.0/include -DQT\_CORE\_LIB -Femain -link -LIBPATH:D:/Qt/4.7.0/lib QtCore4.lib

依然很简单

* -I 头文件路径
* -D 定义必要的宏
* -Fe 指定可执行程序文件名
* -link 后面是链接器参数
  + -LIBPATH 库文件路径

## 例子二：简单的GUI程序

这次稍微复杂一点，不是单一的控制台程序，而是一个简单的GUI程序

* main.cpp
* #include <QtGui/QApplication>
* #include "widget.h"
* int main(int argc, char\*\* argv)
* {
* QApplication app(argc, argv);
* Widget w;
* w.show();
* return app.exec();
* }
* widget.h
* #include <QtGui/QWidget>
* class Widget : public QWidget
* {
* public:
* Widget(QWidget \* parent=NULL);
* };
* widget.cpp
* #include "widget.h"
* Widget::Widget(QWidget \* parent)
* :QWidget(parent)
* {
* }

同样，这个程序未使用Qt的扩展，直接用C++的编译器编译：

### g++

g++ main.cpp widget.cpp -DQT\_CORE\_LIB -DQT\_GUI\_LIB -Ie:\Qt\4.7.0-beta2\include -o main -Le:\Qt\4.7.0-beta2\lib -lQtCore4 -lQtGui4

因为我们使用了QtGui模块，所以和前面相比：

* 增加了 -DQT\_GUI\_LIB 和 -lQtGui4
* 多了一个文件 widget.cpp

注意: Windows下

如果在非windows平台下，这条命令就可以了。但windows下，你知道的：分console和windows两个链接子系统，而且入口函数分 main 和 WinMain 。

这条命令，编译出的 main.exe 会弹出控制台。要想不要控制台，则使用下面的命令：

g++ main.cpp widget.cpp -DQT\_CORE\_LIB -DQT\_GUI\_LIB -DQT\_NEEDS\_QMAIN -Ie:\Qt\4.7.0-beta2\include -o main -Le:\Qt\4.7.0-beta2\lib -lQtCore4 -lQtGui4 -lqtmain -Wl,-subsystem,windows

多了两个选项：

* qtmain 该库中一个WinMain 函数，它会调用我们的代码的main函数。即对编译器来说：入口函数的名字变了
* -Wl,-subsystem,windows 你知道的，链接windows子系统
* 对与MinGW来说，此处多了一个宏 QT\_NEEDS\_QMAIN，这个东西很有意思。在Qt Windows下链接子系统与入口函数(终结版) 中我们详细提到了这个。(在此处，不过你可以忽略它，不会出错，而且也不会有可感觉到的差异)

### cl

同windows下的g++基本一样，带控制台：

cl main.cpp widget.cpp -ID:/Qt/4.7.0/include -DQT\_CORE\_LIB -DQT\_GUI\_LIB -Femain -link -LIBPATH:D:/Qt/4.7.0/lib QtCore4.lib QtGui4.lib

不带控制台：

cl main.cpp widget.cpp -ID:/Qt/4.7.0/include -DQT\_CORE\_LIB -DQT\_GUI\_LIB -Femain /MD -link -LIBPATH:D:/Qt/4.7.0/lib -subsystem:windows qtmain.lib QtCore4.lib QtGui4.lib

分析同上：指定链接子系统，启用WinMain入口函数

### 多文件的程序如何管理

直接调用编译器有什么坏处呢？

* 参数多啊，每次手动输入，难免出错。（例子中我们用的参数已经尽可能少了，可能都还是让你眼晕了）。
* 其次呢，很重要的一点，每次只要一个文件修改，所有东西都要重新编译。

改变这种状况的办法，传统的就是写 Makefile，然后编译时只需要输入 make 就行了，他会判断哪些文件被改动需要重新编译。

另外就是VS等一些IDE自己提供的功能。下面简单看一下本例子对应makefile文件：

mingw32-make的Makefile文件

CPPFLAGS = -DQT\_CORE\_LIB -DQT\_GUI\_LIB -Ie:\Qt\4.7.0\include

LDFLAGS = -Le:\Qt\4.7.0\lib -lQtCore4 -lQtGui4 -lqtmain -Wl,-subsystem,windows

objects = main.o widget.o

dest = main

$(dest) : $(objects)

g++ -o $@ $(objects) $(LDFLAGS)

nmake的Makefile文件

CPPFLAGS = -ID:/Qt/4.7.0/include -DQT\_CORE\_LIB -DQT\_GUI\_LIB -MD

LDFLAGS = -LIBPATH:D:/Qt/4.7.0/lib -subsystem:windows qtmain.lib QtCore4.lib QtGui4.lib

objects = main.obj widget.obj

dest = main.exe

$(dest) : $(objects)

link $(objects) $(LDFLAGS)

对此不做介绍，因为Makefile编写也是一门学问。相当难写，所有才有qmake、cmake这些工具来帮我们生成Makefile文件

## 例子三：引入moc

Qt 对 C++ 的扩展主要是3个方面：

* 元对象系统，包含Q\_OBJECT宏的文件(.h, .cpp等)需要 moc 预处理
* 资源系统，.qrc 文件 需要 rcc 进行预处理
* 界面系统，.ui 文件 需要 uic 进行预处理

这3者之中，元对象系统最复杂，也是 Qt 程序中重要的。其他两个你都可以不要，唯独这个不要就有点不像话了(没它还叫Qt程序么？像我们前面写的，只不过是普通的C++程序)

废话少说，看例子：（修改前面的widget.h，加入Q\_OBJECT）

#include <QtGui/QWidget>

class Widget : public QWidget

{

Q\_OBJECT

public:

Widget(QWidget \* parent=NULL);

};

如何编译这个程序呢？例子二中的命令还能用吗？不妨试试：

**哇，输出好丰富啊！**

来自 g++ 的问候：

main.o:main.cpp:(.text$\_ZN6WidgetD1Ev[Widget::~Widget()]+0xb): undefined reference to `vtable for Widget'

main.o:main.cpp:(.text$\_ZN6WidgetD1Ev[Widget::~Widget()]+0x15): undefined reference to `vtable for Widget'

widget.o:widget.cpp:(.text+0x39): undefined reference to `vtable for Widget'

widget.o:widget.cpp:(.text+0x43): undefined reference to `vtable for Widget'

来自 cl 的问候：

widget.obj : error LNK2001: 无法解析的外部符号 "public: virtual struct QMetaObject const \* \_\_thiscall Widget::metaObject(void)const " (?metaObject@Widget@@UBEPBUQMetaObject@@XZ)

widget.obj : error LNK2001: 无法解析的外部符号 "public: virtual void \* \_\_thiscall Widget::qt\_metacast(char const \*)" (?qt\_metacast@Widget@@UAEPAXPBD@Z)

widget.obj : error LNK2001: 无法解析的外部符号 "public: virtual int \_\_thiscall Widget::qt\_metacall(enum QMetaObject::Call,int,void \* \*)" (?qt\_metacall@Widget@@UAEHW4Call@QMetaObject@@HPAPAX@Z)发生了什么？

添加一个宏后，发生了什么？我们看看编译器将宏展开后是什么样子的：

#include <QtGui/QWidget>

class Widget : public QWidget

{

static const QMetaObject staticMetaObject;

virtual const QMetaObject \*metaObject() const;

virtual void \*qt\_metacast(const char \*);

virtual int qt\_metacall(QMetaObject::Call, int, void \*\*);

...

public:

Widget(QWidget \* parent=NULL);

};

一下子多出来这么多函数，而且还没有函数体，不出错才怪。如何生成函数体呢？这正是moc所做的：

moc widget.h -o moc\_widget.cpp

这样一来，这些函数都在 moc\_widget.cpp 被实现了，只要我们将该文件一块编译链接就行了

对g++来说，在例子二的基础上，直接添加一个 moc\_widget.cpp 文件，然后一切正常了：

g++ main.cpp widget.cpp moc\_widget.cpp -DQT\_CORE\_LIB -DQT\_GUI\_LIB -Ie:\Qt\4.7.0-beta2\include -o main -Le:\Qt\4.7.0-beta2\lib -lQtCore4 -lQtGui4

对 cl 编译器，同样只要添加一个 moc\_widget.cpp 即可。

## 例子四，rcc和uic

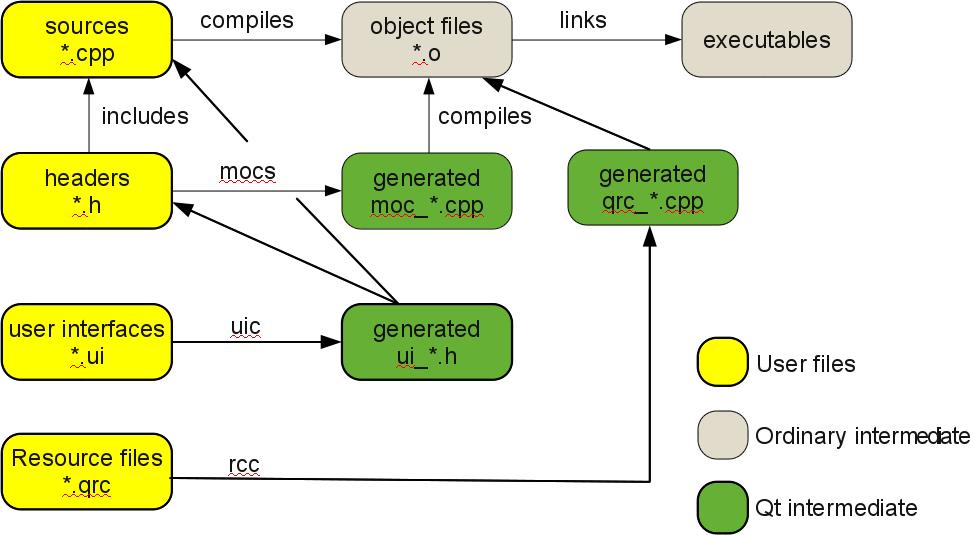
有点糟蹋这个名字了，本节中不讲例子（因为 rcc 和 uic 概念比较简单）

* 如果我们用了资源，那么需要一个 xxx.qrc 文件，这个文件呢，C++ 编译器不认识，于是
* rcc xxx.qrc -o qrc\_xxx.cpp
* 如果我们用了designer设计的界面 .ui。C++ 编译器不认识这个文件，于是
* uic xxx.ui -o ui\_xxx.h

这样一来，我们得到是就全是 .h 和 .cpp 的文件了，剩下的工作，你知道的，交给 C++ 编译器就行了。

## 其他

现在来看这个图：是不是很简单了？



## 2015更新

对Qt5用户，本文基本适用，只需注意

* Qt4中gui在Qt5中主要分成了Qt5Gui、Qt5Widget等几个库。

| **Qt4** | **Qt5** |
| --- | --- |
| QtCore4 | Qt5Core |
| QtGui4 | Qt5Gui Qt5Widget |

* 头文件 QtGui/QWidget 等需要变成 QtWidgets/QWidget

http://blog.debao.me/zh/2010/11/from-cpp-to-qt/