QT Programming

* QT official site

* Getting started programming with QT

使用qmake後，必須在xxx.pro檔，加入以下內容:

```
QT       += core gui
greaterThan(QT_MAJOR_VERSION, 4): QT += widgets
```

source

I just add a script to /usr/local/bin and name it jmake

```
#!/bin/bash
qmake -project
echo "QT       += core gui" >> $1.pro
echo "greaterThan(QT_MAJOR_VERSION, 4): QT += widgets" >> $1.pro
qmake
make
```

QWidgets

雖然在QT 4.7推出了QtQuick並包含了QML，不過目前仍支持QWidgets，且QWidgets的架構與傳統的桌面應用程式的設計框架較為一致，所以我們將以QWidgets Programming做為開端…

* QApplication文件

QApplication類別是圖形使用者介面的應用程式的抽象對應，負責主程式的流程控制與相關的設定。事實上， QApplication類別是QGuiApplication類別的特殊化(子類別)，用以配合QWidget的相關類別一起運作，包含widget的初始化與終結。每個使用QT所設計的GUI應用程式都必須有一個「QApplication」類別的物件。若是不打算使用QWidgets，則請改用QGuiApplication代替。若是沒有圖形使用者需求的程式，non-GUI application，則可以使用「QCoreApplication」類別來免去支援圖形使用者介面所必須付出的成本。

```
QCoreApplication* createApplication(int &argc, char *argv[])
{
    for (int i = 1; i < argc; ++i)
        if (!qstrcmp(argv[i], "-no-gui"))
            return new QCoreApplication(argc, argv);
    return new QApplication(argc, argv);
}


int main(int argc, char* argv[])
```

```
{
    QScopedPointer<QCoreApplication> app(createApplication(argc, argv));

    if (qobject_cast<QApplication *>(app.data())) {
       // start GUI version...
    } else {
       // start non-GUI version...
    }

    return app->exec();
}
```

QApplication's main areas of responsibility are:

It initializes the application with the user's desktop settings such as palette(), font() and doubleClickInterval(). It keeps track of these properties in case the user changes the desktop globally, for example through some kind of control panel. It performs event handling, meaning that it receives events from the underlying window system and dispatches them to the relevant widgets. By using sendEvent() and postEvent() you can send your own events to widgets. It parses common command line arguments and sets its internal state accordingly. See the constructor documentation below for more details. It defines the application's look and feel, which is encapsulated in a QStyle object. This can be changed at runtime with setStyle(). It specifies how the application is to allocate colors. See setColorSpec() for details. It provides localization of strings that are visible to the user via translate(). It provides some magical objects like the desktop() and the clipboard(). It knows about the application's windows. You can ask which widget is at a certain position using widgetAt(), get a list of topLevelWidgets() and closeAllWindows(), etc. It manages the application's mouse cursor handling, see setOverrideCursor() Since the QApplication object does so much initialization, it must be created before any other objects related to the user interface are created. QApplication also deals with common command line arguments. Hence, it is usually a good idea to create it before any interpretation or modification of argv is done in the application itself.

QT_OBJECT Macro

QT_OBJECT Macro

Background of QWidget

background

QMenu

QMenu Example 討論

要確認statustip是否在macos下無法使用

tr() 多國語言包

tr()

QtGui and QML

Qt 5.13 Documentation

RasterWindow Example連結

???

QDoc中的*! [n]註解的作用為何？*