1 QT5.3没有QDeclarativeItem这个类了。被QQuickItem替换

2 这个示例在QT5上运行失败

**什么是QML，它与Quick的关系是什么？**

Qt Quick是Qt User Interface Creation Kit的缩写，而QML是Qt Quick最重要的组成部分，Qt Quick结合了如下技术：

1. 组件集合，其中大部分是关于图形界面的
2. 基于JavaScript陈述性语言：QML （Qt Meta-Object Language的缩写）
3. 用于管理组件并与组件交互的C++ API -  QtDeclarative模块

言归正传：通过Qt Creator，我们可以轻松生成一个Qt Quick的应用工程，从而为QML生成应用程序框架。具体操作详见：[创建qt quick (qml) 应用程序](http://blog.csdn.net/changsheng230/archive/2010/12/17/6082243.aspx)。

C++与QML的交互是通过注册C++对象给QML环境得以实现的：

在C++实现中，非可视化的型别均为QObject的子类，可视化的类型均为QDeclarativeItem的子类。注意：QDeclarativeItem等同于QML的Item类。

如果用户想要定义自己的型别，做法如下：

1. 在C++中，实现派生于QObject或QDeclarativeItem的子类，它是新定义item的实体对象；
2. 在C++中，将1中实现的新item类型注册给QML；
3. 在QML中，导入含有1中定义的新item的模块；
4. 在QML中，向使用标准的item一样使用新定义的item

现举例说明，我们现尝试使用用Qt C++实现的MyButton对象（如下qml代码），它有自己的属性、方法以及信号的handler。用法如下（它与使用其它标准的QML item一样），所需要做的是 需要导入包含MyButton的对应模块名称及其版本“MyItems 1.0 ”。

**[cpp]** [view plaincopyprint?](http://blog.csdn.net/changsheng230/article/details/6167059)

1. //main.qml
2. import Qt 4.7
3. import MyItems 1.0
4. Item {
5. width: 300; height: 200
6. MyButton {
7. //注意：x, y, width, height是继承自item的属性，无需再自定义的item中实现
8. x: 50; y: 50
9. width: 200; height: 100
10. color: "gray"   //自定义属性
11. onMySignals:  dosth  //自定义信号mySignals
12. MouseArea {
13. anchors.fill: parent
14. onClicked: parent.myColor()   // 调用C++定义的方法myColor
15. }
16. }
17. }



为了能够上述qml代码工作，需要为在Qt C++代码中注册MyButton及其所属的模块，对应的main.cpp代码如下：

**[cpp]** [view plaincopyprint?](http://blog.csdn.net/changsheng230/article/details/6167059)

1. #include <QtGui/QApplication>
2. #include "qmlapplicationviewer.h"
3. **int** main(**int** argc, **char** \*argv[])
4. {
5. QApplication app(argc, argv);
6. QmlApplicationViewer viewer;
7. // MyButtonItem是与QML中MyButton相对应的C++实现的类名称
8. // 1，0是版本信息；MyItems是MyButton所属的模块名称
9. qmlRegisterType<MyButtonItem>("MyItems", 1, 0, "MyButton ");
10. viewer.setOrientation(QmlApplicationViewer::Auto);
11. viewer.setMainQmlFile(QLatin1String("qml/untitled/main.qml"));
12. viewer.show();
13. return app.exec();
14. }



上面我们在QML中MyButton对象，有自己的属性、方法以及信号的handler，其实现均来自Qt C++。Qt C++需要作以下工作：

1. 首先要定义 QML中MyButton相对应的C++实现MyButtonItem，它必须继承自**QDeclarativeItem**
2. 为了让MyButton对象能够使用其Color属性，MyButtonItem类需要利用QT的**PROPERTY**系统，为Moc声明其属性
3. 为了让MyButton对象能够使用其myColor方法，MyButtonItem类需要声明该方法，并标记为**Q\_INVOKABLE** （另外一种解决方案是将myColor声明为槽。
4. 为了让MyButton对象能够接受到C++所emit的信号，并在**onMySignals**，MyButtonItem类需要声明mySignals信号

**[cpp]** [view plaincopyprint?](http://blog.csdn.net/changsheng230/article/details/6167059)

1. class MyButtonItem : public QDeclarativeItem
2. {
3. Q\_OBJECT
4. Q\_PROPERTY(QColor color READ color WRITE setColor NOTIFY colorChanged)
5. signals:
6. void colorChanged();
7. void mySignals();
8. public:
9. MyButtonItem(QDeclarativeItem \*parent = 0);
10. void paint(QPainter \*painter, const QStyleOptionGraphicsItem \*option,
11. QWidget \*widget = 0);
12. public:
13. const QColor &color() const;
14. void setColor(const QColor &newColor);
15. Q\_INVOKABLE QColor myColor() const;
16. // Alternatives for myColor to be called from QML
17. //public slots
18. //QColor myColor() const;
19. private:
20. QColor m\_color;
21. };