**将QDeclarativeItem换做QQuickPaintedItem就可以在QT5上运行**

**引言**

Qt Quick为我们提供了许多内建的[QML Elements](http://doc.qt.nokia.com/latest/qdeclarativeelements.html)，其功能可以满足我们绝大部分的需要。但有时我们想制作出自己的QML类型，从而方便自己程序的使用，这是完全可以做到的。做到这点总共有两种方法：一种方法是我们制作新的QML Component，第二种方法是我们使用C++来创建新的QML类型。这第二种方法是更加强大的。本文就详细为大家介绍一下。同时大家也可以参考：[Chapter 1: Creating a New Type](http://doc.qt.nokia.com/latest/declarative-tutorials-extending-chapter1-basics.html)。

**新QML类型**

我们要创建的QML类型叫"PieChart"，它有两个属性：name 和 color。我们将把它放入版本号为1.0的 "Charts" 模块中。  
我们希望在QML中这样使用我们的新类型PieChart

**import** Charts 1.0  
   
PieChart {  
width: 100; height: 100  
 name: "A simple pie chart"  
color: "red"  
 }

**C++类**

要完成这一工作，我们需要用一个C++的类来封装PieChart 类型以及它的两个属性。由于QML大量利用了Qt的meta object system，这个C++类必须要：

* 继承自QObject
* 使用Q\_PROPERTY 声明它的属性

下面是在piechart.h 中定义的PieChart 类：

#include <QDeclarativeItem>  
#include <QColor>  
   
 **class** PieChart : **public** QDeclarativeItem  
 {  
Q\_OBJECT  
Q\_PROPERTY(QString name READ name **WRITE** setName)  
Q\_PROPERTY(QColor color READ color **WRITE** setColor)  
   
 **public**:  
PieChart(QDeclarativeItem \*parent = 0);  
   
QString name() **const**;  
 **void** setName(**const** QString &name);  
   
QColor color() **const**;  
 **void** setColor(**const** QColor &color);  
   
 **void** paint(QPainter \*painter, **const** QStyleOptionGraphicsItem \*option, QWidget \*widget = 0);  
   
 **private**:  
QString m\_name;  
QColor m\_color;  
 };

这个类继承自QDeclarativeItem。这是因为我们想重写QDeclarativeItem::paint() 这个方法来进行绘制。如果这个类仅仅是代表一些数据类型，并不需要实际的显示，那么我们直接让它继承QObject就可以了。当然，如果我们想扩展现有的基于QObject的类的功能，那么我们也可以从那个类继承下来。

PieChart 这个类使用Q\_PROPERTY宏定义了两个属性name 和 color，并重写了QDeclarativeItem::paint()。在piechart.cpp 这个类的实现中简单的设置和返回了m\_name 和 m\_color 的值作为属性，并且实现了 paint() 方法绘制了一个简单的扇形图。同时它也关闭了QGraphicsItem::ItemHasNoContents 标志从而允许绘制：

PieChart::PieChart(QDeclarativeItem \*parent)  
 : QDeclarativeItem(parent)  
 {  
 *// need to disable this flag to draw inside a QDeclarativeItem*  
setFlag(QGraphicsItem::ItemHasNoContents, **false**);  
 }  
...  
 **void** PieChart::paint(QPainter \*painter, **const** QStyleOptionGraphicsItem \*, QWidget \*)  
 {  
QPen pen(m\_color, 2);  
painter->setPen(pen);  
painter->setRenderHints(QPainter::Antialiasing, **true**);  
painter->drawPie(boundingRect(), 90 \* 16, 290 \* 16);  
 }

**在QML中使用新类型**

现在已经定义好了PieChart 类型，那么我们就可以在QML中使用它了。app.qml 文件创建了一个PieChart item 并且使用一个标准的QML Text item来显示扇形图表的名字：

**import** Charts 1.0  
 **import** QtQuick 1.0  
   
 **Item** {  
width: 300; height: 200  
   
PieChart {  
id: aPieChart  
anchors.centerIn: parent  
width: 100; height: 100  
 name: "A simple pie chart"  
color: "red"  
 }  
   
Text {  
anchors { bottom: parent.bottom; horizontalCenter: parent.horizontalCenter; bottomMargin: 20 }  
text: aPieChart.name  
 }  
 }

需要注意的是，虽然在QML中给color 设定的是一个字符串，但是它会针对PieChart 的color 属性自动装换成 QColor对象。许多[基本类型](http://doc.qt.nokia.com/latest/qdeclarativebasictypes.html)都支持自动装换。例如，字符串"640x480" 会被自动装换为一个 QSize值。

**运行使用了新类型的qml文件**

我们将创建一个C++应用程序，它使用QDeclarativeView 来运行和显示app.qml。这个应用程序必须使用qmlRegisterType()函数来注册PieChart 类型，从而使得可以从QML中访问这个类型。如果你不注册的话，app.qml就无法使用PieChart 。这是应用程序的main.cpp：

#include "piechart.h"  
#include <qdeclarative.h>  
#include <QDeclarativeView>  
#include <QApplication>  
   
int main(int argc, char \*argv[])  
 {  
QApplication app(argc, argv);  
   
qmlRegisterType<PieChart>("Charts", 1, 0, "PieChart");  
   
QDeclarativeView view;  
view.setSource(QUrl::fromLocalFile("app.qml"));  
view.show();  
 **return** app.exec();  
 }

这里调用了qmlRegisterType() 来注册 C++ 类 PieChart 作为QML 中的PieChart 类型，并将其放入版本号为1.0的"Charts"模块中。

最后我们需要在.pro 工程文件中包含所有的文件以及 declarative 库：

QT += declarative  
   
HEADERS += piechart.h  
SOURCES += piechart.cpp \  
main.cpp

我们现在编译和运行这个程序：