 前面两篇文章《[QML 语言基础](http://blog.csdn.net/foruok/article/details/28634543" \t "_blank)》和《[Qt Quick 简单教程](http://blog.csdn.net/foruok/article/details/28859415)》中我们介绍了 QML 语言的基本语法和 Qt Quick 的常见元素，亲们，通过这两篇文章，您应该已经可以完成简单的 Qt Quick 应用了。接下来呢，哈，我们要介绍 Qt Quick 中一个灰常灰常重要的主题：事件处理。这将是比较长长长长的一篇，哦，不还有后续好几篇……废话少说，还是谈正事儿吧兄弟姐妹们。

    本文是作者 Qt Quick 系列文章中的一篇，其它文章在这里：

* [Qt Quick 简介](http://blog.csdn.net/foruok/article/details/28620525)
* [QML 语言基础](http://blog.csdn.net/foruok/article/details/28634543)
* [Qt Quick 之 Hello World 图文详解](http://blog.csdn.net/foruok/article/details/28850879)
* [Qt Quick 简单教程](http://blog.csdn.net/foruok/article/details/28859415)
* [Qt Quick 事件处理之信号与槽](http://blog.csdn.net/foruok/article/details/30028711)
* [Qt Quick事件处理之鼠标、键盘、定时器](http://blog.csdn.net/foruok/article/details/30474299)
* [Qt Quick 事件处理之捏拉缩放与旋转](http://blog.csdn.net/foruok/article/details/32078761)
* [Qt Quick 组件与对象动态创建详解](http://blog.csdn.net/foruok/article/details/32730911)
* [Qt Quick 布局介绍](http://blog.csdn.net/foruok/article/details/33738227)
* [Qt Quick 之 QML 与 C++ 混合编程详解](http://blog.csdn.net/foruok/article/details/32698603)

    GUI 应用都是基于事件的（同学，这么说对吗？），不管是用 C++ 还是用 QML ，你的应用都要处理事件，否则的话，哼哼，要你好看！当然当然，你也可以什么事儿不干，就搭个静态界面放那里欣赏……

    Qt Quick 最大的一个特点，是与 Qt 元对象系统的结合；而这里边，我们熟稔的，鼎鼎大名大名鼎鼎的，要不断打交道的，就是信号与槽了。历史是割不断的，继承与发展才是正确的价值观……，了解过 Qt 的开发者一定对信号与槽印象深刻。在 QML 中，在 Qt Quick 中，要想妥善的处理各种事件，同样离不开信号与槽。所以呢，在介绍具体的事件之前，我们先要介绍 QML 中如何使用信号与槽。

    为了演示本文的示例，我们先介绍一个工具： qmlscene 。

**qmlscene 工具**

    这是 Qt 框架提供的便利工具，使用它，你可以不用建立项目不用撰写 C++ 代码，就可以看到你编写的 qml 文件的效果。我们以 Windows 7 系统为例， qmlscene 需要在命令行窗口中使用。你可以这样打开 Qt 自带的命令行开发环境，参考图 1 找到命令行环境的快捷方式：



            图 1 Qt 命令行环境的快捷方式

    一旦你打开了命令行工具，它会帮你设置好 Qt 需要的环境变量，敲一个 qmake 命令试试吧。如和 图 2 所示：

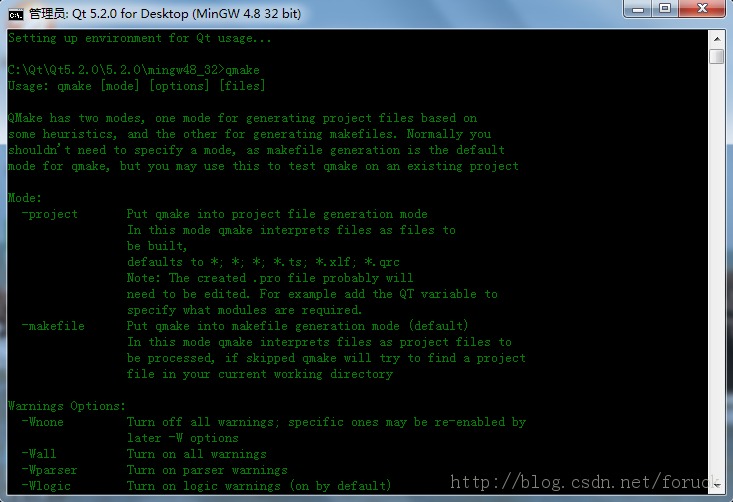


            图 2 Qt 命令行开发环境

    如果你想知道 qmlscene 工具的详细用法，请输入 qmlscene --help 命令。如图 3 ：

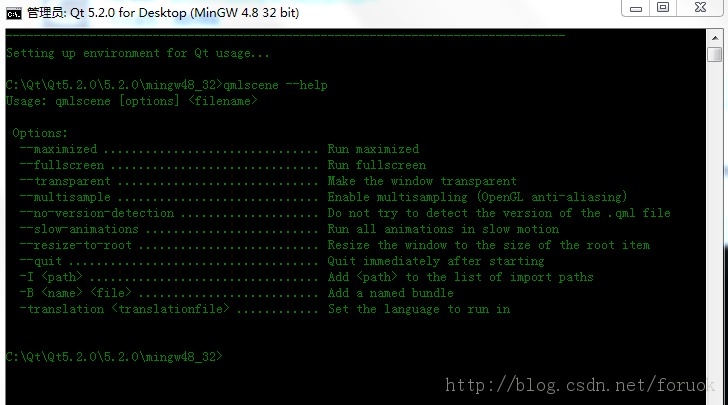


            图 3  qmlscene 的帮助

    现在，使用 cd 命令切换到你存放 qml 文档的目录下，就可以验证 qml 的效果了。只需要执行这样的命令： qmlscene yourapp.qml 。当然，yourapp.qml 要替换为你实际的 qml 文档的名字。我这里只是示例。如果你直接输入 qmlscene 命令来执行，那么它会打开一个文件选择对话框，让你选择一个 qml 文档，当你选择了之后呢，该文档对应的界面就会显示出来。我建议你使用 qmlscene yourapp.qml 这种方式，无它，唯快尔。

    来看一个简单的 Hello World ，qml 文档是 hello\_world.qml ，内容如下：

**[javascript]** [view plaincopy](http://blog.csdn.net/foruok/article/details/30028711)[在CODE上查看代码片](https://code.csdn.net/snippets/378988)

1. **import** QtQuick 2.0
2. **import** QtQuick.Controls 1.1
4. Rectangle {
5. width: 320;
6. height: 240;
7. color: "gray";
9. Text {
10. anchors.centerIn: parent;
11. text: "Hello World!";
12. color: "blue";
13. font.pixelSize: 32;
14. }
15. }

    在命令行环境执行 qmlscene hello\_world.qml ，效果如图 4 所示：

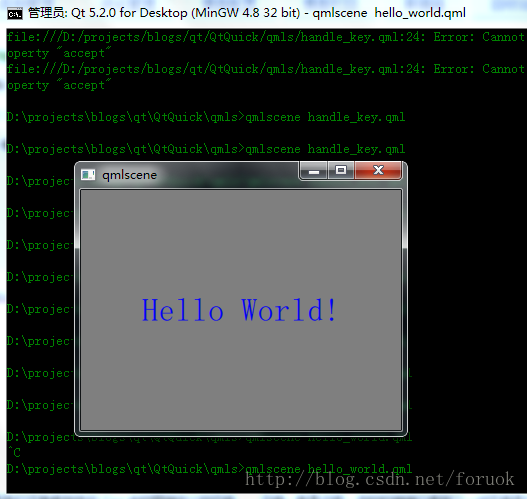


            图 4 使用 qmlscene 执行 Hello World 示例

    好啦， qmlscene 的介绍到此为止，下面我们来看信号与槽了。

**连接 QML 类型的已知信号**

    我们先看一个简单的示例， qml 中只有一个退出按钮，点击退出应用。 qml 文档为 button\_quit.qml ，内容如下：

**[javascript]** [view plaincopy](http://blog.csdn.net/foruok/article/details/30028711)[在CODE上查看代码片](https://code.csdn.net/snippets/378988)

1. **import** QtQuick 2.0
2. **import** QtQuick.Controls 1.1
4. Rectangle {
5. width: 320;
6. height: 240;
7. color: "gray";
9. Button {
10. text: "Quit";
11. anchors.centerIn: parent;
12. onClicked: {
13. Qt.quit();
14. }
15. }
16. }

    使用 qmlscene 执行 button\_quit.qml 效果如图 5 所示：



            图 5 button\_quit 示例效果

    现在看看代码有何特别之处。其实在《[Qt on Android: Qt Quick 简单教程](http://blog.csdn.net/foruok/article/details/28859415" \t "_blank)》中我们已经见过类似的代码了：

**[javascript]** [view plaincopy](http://blog.csdn.net/foruok/article/details/30028711)[在CODE上查看代码片](https://code.csdn.net/snippets/378988)

1. onClicked:{}

    对，就是这个 onClicked ，其实就包含了 QML  中使用信号与槽的一般形式：信号处理器。

**信号处理器**

    信号处理器，其实等价于 Qt 中的槽。但是我们没有看到类似 C++ 中的明确定义的函数……没错，就是这样，你的的确确只看到了一对花括号！对啦，这是 JavaScript 中的代码块。其实呢，你可以理解为它是一个匿名函数。而 JavaScript 中的函数，其实具名的代码块。函数的好处是你可以在其它地方根据名字调用它，而代码块的好处是，除了定义它的地方，没人能调用它，一句话，它是私有的。代码块就是一系列语句的组合，它的作用就是使语句序列一起执行。

    让我们回头再看信号处理器，它的名字还有点儿特别，一般是 on{Signal} 这种形式。在上节的示例中， Button 元素有一个名为 clicked() 的信号，我们提供的信号处理器是酱紫的：

**[javascript]** [view plaincopy](http://blog.csdn.net/foruok/article/details/30028711)[在CODE上查看代码片](https://code.csdn.net/snippets/378988)

1. onClicked: {
2. Qt.quit();
3. }

    非常简单吧，仅仅是调用 Qt.quit() 来退出应用而已。

    Qt 对象是 Qt Quick 导出到 QML 环境中的对象，它的 quit() 方法退出应用。还有很多其它的方法，比如 rgba() 用于构造一个颜色（color类型）， md5() 用来计算一段数据的 MD5 值……

    你看到了，当信号是 clicked() 时，信号处理器就命名为 onClicked 。就这么简单，以 on 起始后跟信号名字（第一个字母大写）。如果你点击我们的 quit 按钮，应用就真的退出了。

    上面的示例，信号处理器放在拥有信号的元素内部，当元素信号发射时处理器被调用。还有一种情况，要处理的信号不是当前元素发出来的，而是来自其它类型（对象）比如处理按键的 Keys ，这就是附加信号处理器。

**附加信号处理器**

    在 QML 语言的语法中，有一个附加属性（attached properties）和附加信号处理器（attached signal handlers）的概念，这是附加到一个对象上的额外的属性。从本质上讲，这些属性是由附加类型（attaching type）来实现和提供的，它们可能被附加到另一种类型的对象上。附加属性与普通属性的区别在于，对象的普通属性是由对象本身或其基类（或沿继承层级向上追溯的祖先们）提供的。

   举个例子，下面的 Item 对象使用了附加属性和附加信号处理器：

**[javascript]** [view plaincopy](http://blog.csdn.net/foruok/article/details/30028711)[在CODE上查看代码片](https://code.csdn.net/snippets/378988)

1. **import** QtQuick 2.0
3. Item {
4. width: 100;
5. height: 100;
7. focus: **true**;
8. Keys.enabled: **false**;
9. Keys.onReturnPressed: console.log("Return key was pressed");
10. }

    你看， Item 对象可以访问和设置 Keys.enabled 和 Keys.onReturnPressed 的值。

    enabled 是 Keys 对象的一个属性。

    onReturnPressed 其实是 Keys 对象的一个信号。

    对于附加信号处理器，和前面讲到的普通信号处理器又有所不同。普通信号处理器，你先要知道信号名字，然后按照 on{Signal} 的语法来定义信号处理器的名字；而附加信号处理器，信号名字本身已经是 onXXX 的形式，你只要通过附加类型名字引用它，把代码块赋值给它即可。下面是另外的代码片段：

**[javascript]** [view plaincopy](http://blog.csdn.net/foruok/article/details/30028711)[在CODE上查看代码片](https://code.csdn.net/snippets/378988)

1. Rectangle {
2. width: 320;
3. height: 480;
4. color: "gray";
6. focus: **true**;
7. Keys.enabled: **true**;
8. Keys.onEscapePressed: {
9. Qt.quit();
10. }
11. }

    这个代码片段实现的功能非常简单：用户按下 Esc 键时退出应用。

    Component 对象也有一些附加信号，如 Component.onCompleted() 、 Component.onDestruction() 。可以用来在 Component 创建完成或销毁时执行一些 JavaScript 代码来做与初始化或反初始化相关的工作。比如下面的代码：

**[javascript]** [view plaincopy](http://blog.csdn.net/foruok/article/details/30028711)[在CODE上查看代码片](https://code.csdn.net/snippets/378988)

1. Rectangle {
2. Component.onCompleted: console.log("Completed Running!");
3. Component.onDestruction: console.log("Destruction Beginning!");
4. }

    信号处理器与附加信号处理器有一个共性：响应信号的代码都放在元素内部，通过 JavaScript 代码块就地实现。而其实呢， Qt Quick 中还有另外一种方式来处理信号与槽，那就是：专业的 Connections 。

**Connections**

    一个 Connections 对象创建一个到 QML 信号的连接。

    前面两节在处理 QML 信号时，都是用 on{Signal} 这种就地代码块的方式。而在有些情况下，这样的处理并不方便。比如：

* 你需要将多个对象连接到同一个 QML 信号上
* 你需要在发出信号的对象的作用域之外来建立连接
* 发射信号的对象没有在 QML 中定义（可能是通过 C++ 导出的，这很常见）

    Connections 有一个属性名为 target ，它呢，指向发出信号的对象。

    下面就看看 Connections 怎么使用。一般的用法：

**[javascript]** [view plaincopy](http://blog.csdn.net/foruok/article/details/30028711)[在CODE上查看代码片](https://code.csdn.net/snippets/378988)

1. Connections {
2. target: area;
3. on{Signal}: **function** or code block;
4. }

   来看一个实际的示例，是酱紫的：界面上放置两个文本，一个按钮，每点按钮一次，两个文本对象都变颜色，而它们的颜色随机的。下面是示例代码：

**[javascript]** [view plaincopy](http://blog.csdn.net/foruok/article/details/30028711)[在CODE上查看代码片](https://code.csdn.net/snippets/378988)

1. **import** QtQuick 2.0
2. **import** QtQuick.Controls 1.1
4. Rectangle {
5. width: 320;
6. height: 240;
7. color: "gray";
9. Text {
10. id: text1;
11. anchors.horizontalCenter: parent.horizontalCenter;
12. anchors.top: parent.top;
13. anchors.topMargin: 20;
14. text: "Text One";
15. color: "blue";
16. font.pixelSize: 28;
17. }
19. Text {
20. id: text2;
21. anchors.horizontalCenter: parent.horizontalCenter;
22. anchors.top: text1.bottom;
23. anchors.topMargin: 8;
24. text: "Text Two";
25. color: "blue";
26. font.pixelSize: 28;
27. }
29. Button {
30. id: changeButton;
31. anchors.top: text2.bottom;
32. anchors.topMargin: 8;
33. anchors.horizontalCenter: parent.horizontalCenter;
34. text: "Change";
35. }
37. Connections {
38. target: changeButton;
39. onClicked: {
40. text1.color = Qt.rgba(Math.random(), Math.random(), Math.random(), 1);
41. text2.color = Qt.rgba(Math.random(), Math.random(), Math.random(), 1);
42. }
43. }
44. }

    代码比较简单，除了 Connections 和 Math ，没有其它的新内容，都是 《[Qt Quick 简单教程](http://blog.csdn.net/foruok/article/details/28859415)》和《[QML 语言基础](http://blog.csdn.net/foruok/article/details/28634543)》中讲过的。我在 Connections 对象中指定 target 为 changeButton （Change按钮的 id），然后定义了 onClicked 信号处理器，在信号处理器中使用 Math 对象的 random() 方法构造颜色值来改变两个文本的颜色。

    Math 是 JavaScript 语言内置的对象，有 random() / sin() / max() / min() / abs() 等等方法，参见[w3c 的文档](http://www.w3school.com.cn/jsref/jsref_obj_math.asp" \t "_blank)。

    图 6 是运行后的效果图：

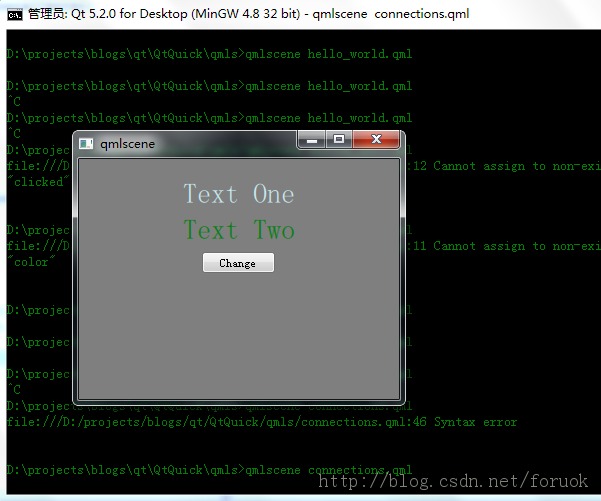


            图 6 Connections 的使用

    到现在为止，我们说的都是如何使用 QML 中已有类型定义的信号。这些信号呢，其实又分类两类。一类是由用户输入产生的，比如按键、鼠标、触摸屏、传感器等，另一类呢是由对象状态或属性变化产生的，比如 Image 对象的 status 属性（在《[Qt Quick 简单教程](http://blog.csdn.net/foruok/article/details/28859415" \t "_blank)》有用到）。那么有一个问题，就是，怎样知道一个对象有哪些信号？

**如何寻找感兴趣的信号**

    怎样找到你感兴趣的信号呢？

**Qt 帮助**

    首先是查阅 Qt 帮助，你可以使用 Qt 帮助的索引模式，以你关心的对象名字为关键字检索，比如 Button ，检索结果如图 7 所示：

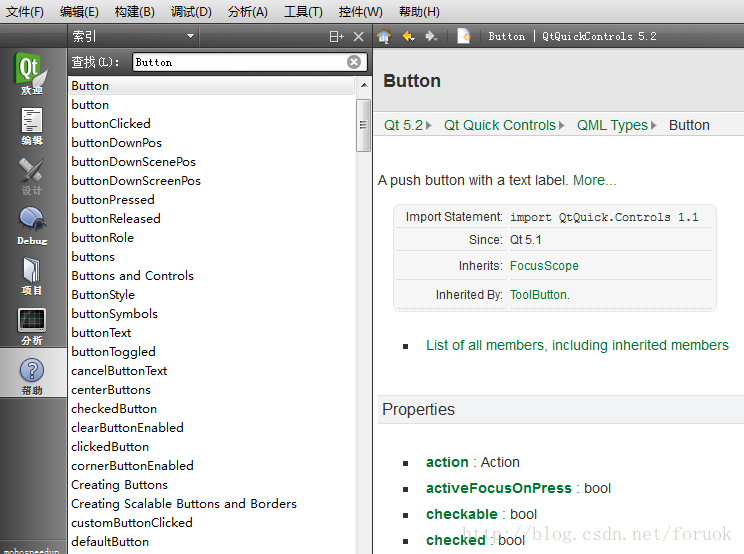


            图7 使用 Qt 帮助索引模式检索 Button 对象

    有时你会在查找结果中看到多个连接，点进去看看是否是 QML 类型即可。

    还有另外一种方式，使用 Qt 帮助的目录模式。如图 8 所示：

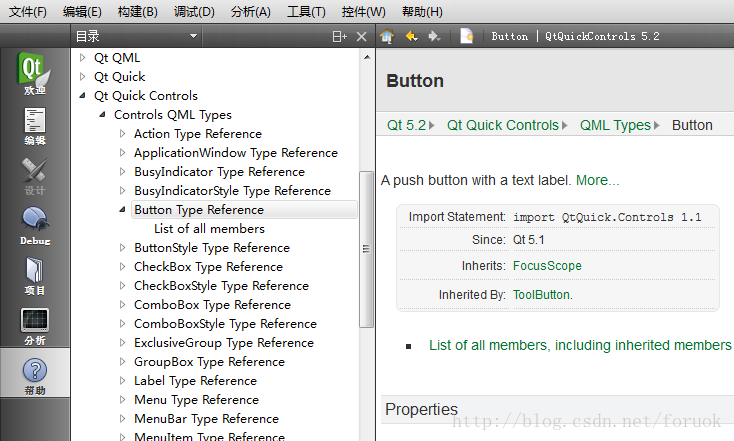


            图 8 使用 Qt 帮助目录模式

    一旦你找到一个对象的文档，你可以找到它的部分信号说明。还是以 Button 为例，看图 9：

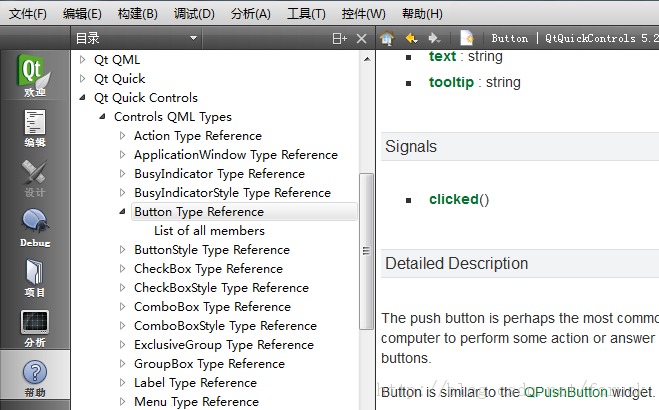


            图 9 Button 的信号 clicked()

    Qt Quick  相关类型的文档中，你可以看到对象的属性和信号。列为属性的，可以在 QML 中访问；列为信号的，可以连接它，通过信号处理器来响应用于操作。

    至于具体某个属性或信号是何含义，点击它们，跟过去看看吧。

    话说，Qt 的文档是否列出了 Qt Quick 类型的所有信号了呢？想必我这么说你用脚趾头也可以想到答案：没有！个人认为这是 Qt 5.2 文档关于 Qt Quick 和 QML 类型手册的一个缺失。前面提到 QML 中的信号，一类是输入事件触发的，一类是属性变化触发的。文档中缺失的，正是属性变化触发的那些信号。不过呢，不过我们有办法找到它。

**从 Qt Quick 头文件查看属性相关的信号**

    要说呢， Qt Quick 中你看到的很多对象，都是 Qt C++ 中实现，然后导入到 QML 环境中的。所以呢，如果你关心那些被文档隐藏了的信号，可以这么做：

1. 找到 QML 类型对应的 C++ 类型
2. 找到 C++ 类型的头文件，查看属性声明来确认是否有信号与属性关联

    怎么找 QML 类型对应的 C++ 类型呢？很简单，只需要使用 Component.onCompleted 附加信号，在附加信号处理器中输出类型信息即可。示例代码：

**[javascript]** [view plaincopy](http://blog.csdn.net/foruok/article/details/30028711)[在CODE上查看代码片](https://code.csdn.net/snippets/378988)

1. **import** QtQuick 2.0
2. **import** QtQuick.Controls 1.1
4. Rectangle {
5. width: 320;
6. height: 240;
7. color: "gray";
9. Text {
10. id: text1;
11. anchors.centerIn: parent;
12. text: "Hello World!";
13. color: "blue";
14. font.pixelSize: 32;
15. }
17. Button {
18. id: button1;
19. text: "A Button";
20. anchors.top: text1.bottom;
21. anchors.topMargin: 4;
22. }
24. Image {
25. id: image1;
26. }
28. Component.onCompleted: {
29. console.log("QML Text\'s C++ type - ", text1);
30. console.log("QML Button\'s C++ type - ", button1);
31. console.log("QML Image\'s C++ type - ", image1);
32. }
33. }

    如代码所示，我们使用 console 对象来输出 QML 对象，它会打印出 QML 对象的实际类型。图 10 是使用 qmlscene 运行 cplusplus\_types.qml 的效果图：

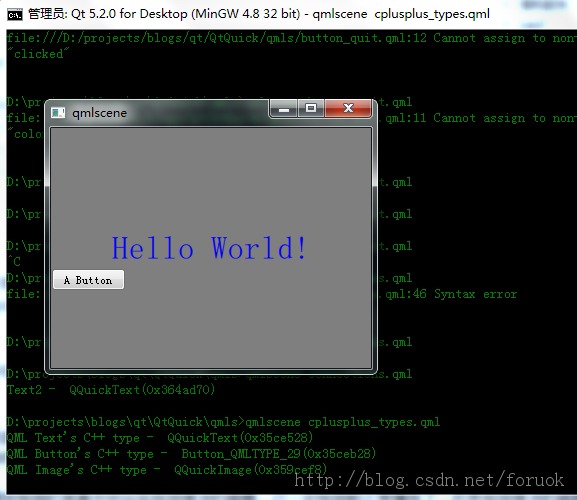


             图 10 打印 QML 类型对应的 C++ 类型

    别看界面效果哦，注意看命令行窗口的输出。对，QML Text 对应的 C++ 类型是 QQuickText ， QML Image 对应的 C++ 类型是 QQuickImage ，而 Button ，其实是 QML 中定义的对象（含有 QMLTYPE 字样）。我这里使用的 Qt 5.2.0 ，如果是其他的 Qt 版本，比如 Qt 4.7 / Qt 4.8 ，能不能看到我就不知道了。

    下面我们就以 QQuickText 为例，找到它的头文件，路径是 C:\Qt\Qt5.2.0\5.2.0\mingw48\_32\include\QtQuick\5.2.0\QtQuick\private\qquicktext\_p.h 。你的环境中根据 Qt SDK 安装目录，路径可能有所不同。

    看看 QQuickText 类的声明吧（我截取了属性部分的几行代码）：

**[cpp]** [view plaincopy](http://blog.csdn.net/foruok/article/details/30028711)[在CODE上查看代码片](https://code.csdn.net/snippets/378988)

1. // qquicktext\_p.h
2. //
3. **class** Q\_QUICK\_PRIVATE\_EXPORT QQuickText : **public** QQuickImplicitSizeItem
4. {
5. Q\_OBJECT
6. Q\_ENUMS(HAlignment)
7. Q\_ENUMS(VAlignment)
8. Q\_ENUMS(TextStyle)
9. Q\_ENUMS(TextFormat)
10. Q\_ENUMS(TextElideMode)
11. Q\_ENUMS(WrapMode)
12. Q\_ENUMS(LineHeightMode)
13. Q\_ENUMS(FontSizeMode)
14. Q\_ENUMS(RenderType)
16. Q\_PROPERTY(QString text READ text WRITE setText NOTIFY textChanged)
17. Q\_PROPERTY(QFont font READ font WRITE setFont NOTIFY fontChanged)
18. Q\_PROPERTY(QColor color READ color WRITE setColor NOTIFY colorChanged)
19. Q\_PROPERTY(QColor linkColor READ linkColor WRITE setLinkColor NOTIFY linkColorChanged)
20. Q\_PROPERTY(TextStyle style READ style WRITE setStyle NOTIFY styleChanged)
21. ...
22. }

    亲爱的，看到了吗，那么多的 Q\_PROPERTY 宏啊。  Q\_PROPERTY 宏就是用来定义 QML 中可访问属性的，当你看到 NOTIFY 字样，它后面的字段就是与属性绑定的信号的名字。 Qt 实现了动态属性绑定，当你为 QML Text 的属性 color 赋值时，实际上会调用到 QQuickText 的 setColor() 函数，也会触发 colorChanged() 信号。

    再来看看 text 和 color 对应的信号原型：

**[cpp]** [view plaincopy](http://blog.csdn.net/foruok/article/details/30028711)[在CODE上查看代码片](https://code.csdn.net/snippets/378988)

1. **class** Q\_QUICK\_PRIVATE\_EXPORT QQuickText : **public** QQuickImplicitSizeItem
2. {
3. Q\_OBJECT
4. ...
5. Q\_SIGNALS:
6. **void** textChanged(**const** QString &text);
7. **void** colorChanged();
8. }

    看到了吧， textChanged 信号有个 text 参数。我们来看看 textChanged 信号怎样在 QML 中使用。示例代码（property\_signal.qml）：

**[javascript]** [view plaincopy](http://blog.csdn.net/foruok/article/details/30028711)[在CODE上查看代码片](https://code.csdn.net/snippets/378988)

1. **import** QtQuick 2.0
2. **import** QtQuick.Controls 1.1
4. Rectangle {
5. width: 320;
6. height: 240;
7. color: "gray";
9. Text {
10. id: hello;
11. anchors.centerIn: parent;
12. text: "Hello World!";
13. color: "blue";
14. font.pixelSize: 32;
15. onTextChanged: {
16. console.log(text);
17. }
18. }
20. Button {
21. anchors.top: hello.bottom;
22. anchors.topMargin: 8;
23. anchors.horizontalCenter: parent.horizontalCenter;
24. text: "Change";
25. onClicked: {
26. hello.text = "Hello Qt Quick";
27. }
28. }
29. }

    当用户点击按钮时，改变 Text 对象的文本为 "Hello Qt Quick" 。而我在 Text 对象中实现了 onTextChanged 信号处理器，使用 console.log 输出新的文本。

    注意啦，QML 信号的参数名字，可以直接在信号处理器中访问。之前只用没说，这里特意说一下这点。如果你通过头文件找属性绑定的信号，就可以观察信号的参数，在 QML 中使用。

    图 11 是运行效果：



            图 11 属性绑定的信号

    你看，图片上的文字变了哈，命令行也输出了新的文本："Hello Qt Quick" 。喏，没骗你。

    行文至此，如何在 QML 中使用已知类型的信号，已经介绍差不多了。

**定义自己的信号**

    当自定义类型不可避免，当你需要通知别的对象你的状态发生了变化，当你对象的粉丝想了解你对象的近况……此时此刻，彼时彼刻，最好的方法， Qt 给我们的，QML 给我们的，还是信号。

    现在我们就来看怎么定义自己的信号。

**自定义信号与使用**

    如果你自己定义新的 QML 类型，可以使用 signal 关键字给你的类型添加信号。其语法如下：

**[cpp]** [view plaincopy](http://blog.csdn.net/foruok/article/details/30028711)[在CODE上查看代码片](https://code.csdn.net/snippets/378988)

1. signal <name>[([<type> <parameter name>[, ...]])]

    这是你的类型通告自己状态的最好的方式，符合 Qt 的风格，作为使用 Qt 的开发人员，还是崇德向善吧。

    信号其实是个方法（函数），所以呢，它的发射，实际是通过调用以信号名为名的方法达成的。

    举个实例的例子：我们在界面上放一个字符串，两个代表颜色的小方块，点小方块，字符串的颜色就变成小方块的颜色。先看代码（my\_signal.qml）：

**[javascript]** [view plaincopy](http://blog.csdn.net/foruok/article/details/30028711)[在CODE上查看代码片](https://code.csdn.net/snippets/378988)

1. **import** QtQuick 2.0
2. **import** QtQuick.Controls 1.1
4. Rectangle {
5. width: 320;
6. height: 240;
7. color: "#C0C0C0";
9. Text {
10. id: coloredText;
11. anchors.horizontalCenter: parent.horizontalCenter;
12. anchors.top: parent.top;
13. anchors.topMargin: 4;
14. text: "Hello World!";
15. font.pixelSize: 32;
16. }
18. Component {
19. id: colorComponent;
20. Rectangle {
21. id: colorPicker;
22. width: 50;
23. height: 30;
24. signal colorPicked(color clr);
25. MouseArea {
26. anchors.fill: parent
27. onPressed: colorPicker.colorPicked(colorPicker.color);
28. }
29. }
30. }
32. Loader{
33. id: redLoader;
34. anchors.left: parent.left;
35. anchors.leftMargin: 4;
36. anchors.bottom: parent.bottom;
37. anchors.bottomMargin: 4;
38. sourceComponent: colorComponent;
39. onLoaded:{
40. item.color = "red";
41. }
42. }
44. Loader{
45. id: blueLoader;
46. anchors.left: redLoader.right;
47. anchors.leftMargin: 4;
48. anchors.bottom: parent.bottom;
49. anchors.bottomMargin: 4;
50. sourceComponent: colorComponent;
51. onLoaded:{
52. item.color = "blue";
53. }
54. }
56. Connections {
57. target: redLoader.item;
58. onColorPicked:{
59. coloredText.color = clr;
60. }
61. }
63. Connections {
64. target: blueLoader.item;
65. onColorPicked:{
66. coloredText.color = clr;
67. }
68. }
69. }

    这次代码稍微有点长了。现在我来充当导游，请跟紧我哦，别担心，不会强制你们买东西的。

    首先定义了一个 Text 对象，id 为 coloredText ，后面会根据这个 id 来改变它的颜色。

    然后我定义了一个组件，组件内有一个 Rectangle 对象，这里是我们定义信号的地方。我设置 Rectangle 的尺寸，然后定义了信号 colorPicked ，语句如下：

**[javascript]** [view plaincopy](http://blog.csdn.net/foruok/article/details/30028711)[在CODE上查看代码片](https://code.csdn.net/snippets/378988)

1. signal colorPicked(color clr);

    为了触发信号，我给 Rectangle 引入了 MouseArea 。MouseArea 是专门处理鼠标操作的 item ，这里我们先知道它有一个 onClicked() 信号就行了。我们给这个信号指定信号处理器，在信号处理器中调用我们刚定义的信号：colorPicker.colorPicked(colorPicker.color); ……如你所见，信号的触发就是一个函数调用。  
    组件是可以重复利用的，一个组件可以在一个单独的 qml 文件中定义，也可以嵌入到其它 qml 文档中来定义以方便简单组件的使用。这里我选择在主 qml 文档内嵌入组件的定义。定义好了组件，就可以使用 Loader 来加载组件了。

    Loader 是专门用来动态创建组件的，它可以从 qml 文件中创建组件，也可以指定 sourceComponent 来创建，这里的示例，因为组件是嵌入在主 qml 文件中定义的，所以使用了 sourceComponent 方式。我给每个 Loader 一个 id ，以便后面连接时使用。我还使用 anchors 为 Loader 布局。最后呢，在 Loader 的 onLoaded 信号处理器内给 Rectangle 对象配置颜色。

    创建完 Loader ，就是建立连接了。这里使用 Connections 来建立信号的连接，target 指向刚才说的 Loader 对象的 item 属性， item 属性实际指向 Loader 创建的对象。在 Connections 对象中，通过 onColorPicked 信号处理器响应用户点击操作。你看到了，我们定义 colorPicked 信号时命名的参数是 clr ，所以 Connections 的信号处理器中可以直接使用它给 coloredText 对象复制。

    好了，看看初始运行的效果，图 12 ：



            图 12 自定义信号之变色文本初始效果图

    图 13 是我点击了红色的颜色选择组件时的效果：

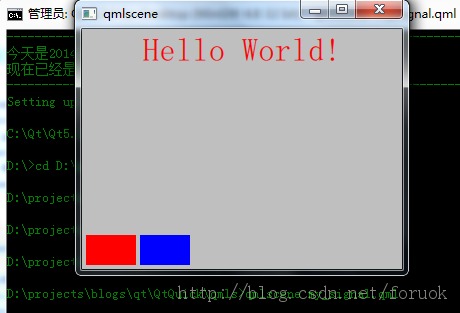


            图 13 选择红色后的效果

    嗯，不知道你是否已经明白如何使用自定义信号？多练练吧。这个示例用到的 Component 和 Loader ，还请您先自行查阅 Qt 帮助来理解，后续我们有专门的文章来讲述它们。

**连接信号与槽**

    前面我们使用信号时，要么通过信号处理器，要么使用 Connections 对象。其实在 QML 中还有一种更一般的方式。先回想下 Qt C++ 中我们如何使用信号与槽…… QObject::connection() ，诺，木错，就是它了。对应的，在 QML 中，其实 signal 是个对象，它也有一个 connect() 方法，你可以使用它连接到任意的方法上哦。有 connect() 就有 disconnect() ，正确， signal 对象的的确确有这两个方法，允许我们使用它维护连接。

    signal 对象的 connect() 方法允许你连接一个信号到另外一个信号或者方法。其实没差别，信号本身也是个方法（函数）。当信号发射时，连接到信号上的其它信号或方法就会被调用。

    signal 对象的这种连接方式，使用起来比信号处理器更加灵活。前面介绍信号处理器时，已经提到，信号处理器和信号是一对一的关系。而 signal 对象的这种连接方式，使得一个信号能够连接多个方法。

    举个简单的例子来看看如何使用吧。下面的代码（来自 Qt 帮助）， messageReceived 信号通过 connect() 方法连接到了三个方法上。

**[javascript]** [view plaincopy](http://blog.csdn.net/foruok/article/details/30028711)[在CODE上查看代码片](https://code.csdn.net/snippets/378988)

1. Rectangle {
2. id: relay;
4. signal messageReceived(string person, string notice);
6. Component.onCompleted: {
7. relay.messageReceived.connect(sendToPost);
8. relay.messageReceived.connect(sendToTelegraph);
9. relay.messageReceived.connect(sendToEmail);
10. relay.messageReceived("Tom", "Happy Birthday");
11. }
13. **function** sendToPost(person, notice) {
14. console.log("Sending to post: " + person + ", " + notice);
15. }
16. **function** sendToTelegraph(person, notice) {
17. console.log("Sending to telegraph: " + person + ", " + notice);
18. }
19. **function** sendToEmail(person, notice) {
20. console.log("Sending to email: " + person + ", " + notice);
21. }
22. }

    上面是信号连接方法的示例，实际上信号也可以连接信号。我们说了，信号本质是一个方法（函数），所以，信号连接信号等同于信号连接方法。下面是一个简单的示例：

**[javascript]** [view plaincopy](http://blog.csdn.net/foruok/article/details/30028711)[在CODE上查看代码片](https://code.csdn.net/snippets/378988)

1. Rectangle {
2. id: forwarder;
3. width: 100;
4. height: 100;
6. signal send();
7. onSend: console.log("Send clicked");
9. MouseArea {
10. id: mousearea;
11. anchors.fill: parent;
12. onClicked: console.log("MouseArea clicked");
13. }
15. Component.onCompleted: {
16. mousearea.clicked.connect(send);
17. }
18. }

    这里我们给 Rectangle 定义了一个信号 send() ，在 Component.onCompleted 附加信号处理器中，把 MouseArea 对象的 clicked 信号连接到 Rectangle 的 send() 信号上。

    K.O. ！终于把 QML 中的信号与槽介绍完了。