   在《[Qt Quick 事件处理之信号与槽](http://blog.csdn.net/foruok/article/details/30028711" \t "_blank)》中介绍了 QML 中如何使用内建类型的信号以及如何自定义信号，这次我们来看看如何处理鼠标、键盘、定时器等事件。这些时间在处理时，通常是通过信号来完成的。

**鼠标事件处理**

    桌面开发的话，难免要处理鼠标事件……

**变色矩形示例**

    看一个简单的处理鼠标事件的例子，先看代码（handle\_mouse.qml）：

**[javascript]** [view plaincopy](http://blog.csdn.net/foruok/article/details/30474299)[在CODE上查看代码片](https://code.csdn.net/snippets/390873)

1. **import** QtQuick 2.0
2. **import** QtQuick.Controls 1.1
4. Rectangle {
5. width: 320;
6. height: 240;
8. MouseArea {
9. anchors.fill: parent;
10. acceptedButtons: Qt.LeftButton | Qt.RightButton;
11. onClicked: {
12. **if**(mouse.button == Qt.RightButton){
13. Qt.quit();
14. }
15. **else** **if**(mouse.button == Qt.LeftButton){
16. color = Qt.rgba((mouse.x % 255) / 255.0 , (mouse.y % 255) / 255.0, 0.6, 1.0);
17. }
18. }
19. onDoubleClicked: {
20. color = "gray";
21. }
22. }
23. }

    使用 "qmlscene handle\_mouse.qml" 命令，可以看到运行效果。上面的代码仅仅是绘制一个矩形，当鼠标左键按下时改变矩形区域的颜色，鼠标右键按下时退出应用。图 1 是刚运行时的效果：

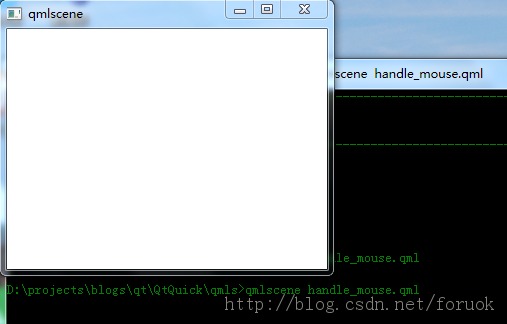


            图1 handle\_mouse.qml 初始运行效果

    图 2 是点击鼠标左键后的效果：

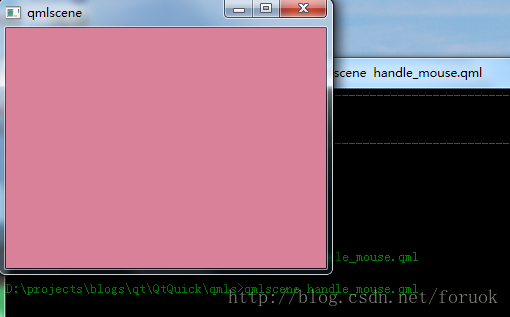


             图 2 鼠标左键点击后的效果

    如果你点一下鼠标右键，程序会退出。

    示例简陋，但足以说明如何处理鼠标事件，下面咱们慢慢来看。

**MouseArea**

    MouseArea 对象可以附加到一个 item 上供 item 处理鼠标事件，它本身是一个不可见的 item 。在其内部，可以直接引用它所附着的对象的属性和方法。你可以将 MouseArea 理解为它所附着的 item 的代理。

    MouseArea 有很多属性， enabled 用来控制是否处理鼠标事件，默认值是 true ，如果你设置为 false ，那么它所代理的 item 就会无视鼠标事件。 acceptedButtons 属性设定接收拿些个鼠标按键产生的事件（左键、右键、中键），示例代码 "acceptedButtons: Qt.LeftButton | Qt.RightButton;" 表示处理鼠标左键和右键。

    作为一个 item ， MouseArea 也拥有 anchors 属性，你可以使用它来描述有效的鼠标区域。示例代码 "anchors.fill: parent;" 表示整个矩形区域都接受鼠标事件。

    MouseArea 还有很多其他属性，如 hoverEnabled ， pressed 等等，请参考 Qt 帮助文档。

    示例代码中，在 MouseArea 对象内使用了 onClicked 和 onDoubleClicked 两个信号处理器，他们对应 MouseArea 的 onClicked 和 onDoubleClicked 信号， MouseArea 还有很多其他的信号，如 onPressed / onReleased / onEntered / onExited / onPressAndHold 等等，从名字上就可以看到这些信号的含义。

    onClicked 信号的参数是 MouseEvent 类型，名为 mouse ，所以你可以在信号处理器中直接使用 mouse 来查询鼠标事件的详情。比如哪个 button 按下，正如示例代码中看到的那样， MouseEvent 的 button 属性保存了被按下的鼠标按键标记， x ， y 属性保存鼠标指针位置。还有一个比较重要的属性 accepted ，如果你处理鼠标事件后不想这个事件再往下传递，就置其值为 true 。

    onDoubleClicked 信号代表双击事件，其参数也是 MouseEvent 类型，示例中双击鼠标，矩形颜色变为灰色。

    简单的鼠标事件处理就这些内容，根据你应用的需要，可能你还会处理 onPressed / onReleased / onEntered 等等信号，请参考 Qt 帮助。

**键盘事件处理**

    手机上你可能较少处理键盘事件（有一个例外，BACK 按键），但是电脑上你免不了要响应键盘。

**会动的文本实例**

    先看示例代码，handle\_key.qml ：

**[javascript]** [view plaincopy](http://blog.csdn.net/foruok/article/details/30474299)[在CODE上查看代码片](https://code.csdn.net/snippets/390873)

1. **import** QtQuick 2.0
2. **import** QtQuick.Controls 1.1
4. Rectangle {
5. width: 320;
6. height: 480;
7. color: "gray";
9. focus: **true**;
10. Keys.enabled: **true**;
11. Keys.onEscapePressed: {
12. Qt.quit();
13. }
14. Keys.forwardTo: [moveText, likeQt];
16. Text {
17. id: moveText;
18. x: 20;
19. y: 20;
20. width: 200;
21. height: 30;
22. text: "Moving Text";
23. color: "blue";
24. //focus: true;
25. font { bold: **true**; pixelSize: 24;}
26. Keys.enabled: **true**;
27. Keys.onPressed: {
28. **switch**(event.key){
29. **case** Qt.Key\_Left:
30. x -= 10;
31. **break**;
32. **case** Qt.Key\_Right:
33. x += 10;
34. **break**;
35. **case** Qt.Key\_Down:
36. y += 10;
37. **break**;
38. **case** Qt.Key\_Up:
39. y -= 10;
40. **break**;
41. **default**:
42. **return**;
43. }
44. event.accepted = **true**;
45. }
46. }
48. CheckBox {
49. id: likeQt;
50. text: "Like Qt Quick";
51. anchors.left: parent.left;
52. anchors.leftMargin: 10;
53. anchors.bottom: parent.bottom;
54. anchors.bottomMargin: 10;
55. z: 1;
56. }
57. }

    这个示例通过上下左右四个按键移动一个文本串，空格键选中复选框， Esc 键退出应用。图 3 是初始运行效果图：

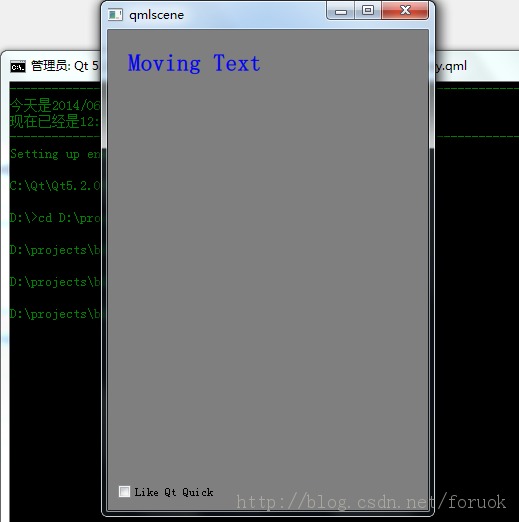


            图 3 handle\_key 初始效果

    图 4 是我按了几次方向键，按了空格键后的效果：

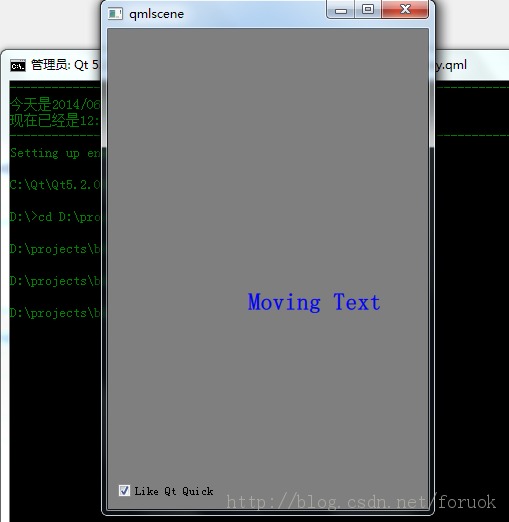


            图 4 移动文本，选中复选框

    下面解释示例代码，介绍如何使用 Keys 对象和信号处理器处理按键事件

**Keys 与 信号处理器**

    其实在 《[Qt Quick 事件处理之信号与槽](http://blog.csdn.net/foruok/article/details/30028711)》、《[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" \t "_blank)》三篇文章中我们都有提到 Keys 对象，有的示例中也用到过，这里呢，我们专门介绍一下，力求使大家对 Keys 及按键处理有个较为全面的了解。

    Keys 对象是 Qt Quick 提供的，专门供 Item 处理按键事件的对象。它定义了很多针对特定按键的信号，比如 onReturnPressed / onEscapePressed / onDownPressed / onDigit0Pressed / onBackPressed 等等；它还定义了更为普通的 onPressed 和 onReleased 信号，一般地，你可以使用这两个信号来处理大部分按键（请对照 Qt C++ 中的 keyPressEvent 和 keyReleaseEvent 来理解），它们有一个名字是 event 的 KeyEvent 参数，包含了按键的详细信息。

    KeyEvent 代表一个按键事件，如果一个按键被处理， event.accepted 应该被设置为 true 以免它被继续传递；要是你不设置它，那它可能会继续传递给其他的 item ，出现一些奇奇怪怪的问题。

    Keys 有三个属性。

    enabled 属性控制是否处理按键。

    forwardTo 属性是列表类型，它表示传递按键事件给列表内的对象，如果某个对象 accept 了某个按键，那位列其后的对象就不会收到该按键事件。示例代码 "Keys.forwardTo: [moveText, likeQt];" 表明转发按键给 id 为 moveText 的 Text 对象和 id 为 likeQt 的 CheckBox 对象。 moveText 在前面，如果它消耗掉某个键， likeQt 就收不到了。你可以修改 Text 对象的 Keys.onPressed 附加信号处理器，在 case 列表中添加 Qt.Key\_Space 看看效果。

    priority 属性允许你设置 Keys 附加属性的优先级，有两种，在 Item 之前处理按键，这是默认行为，在 Item 之后处理按键。你可以对照着 Qt C++ 的 keyPressEvent() 函数来理解，如果你在派生类中重载了 keyPressEvent() 方法，那么你可以在重载方法的一开始调用父类的 keyPressEvent() ，也可以在你处理完感兴趣的事件后再调用父类的 keyPressEvent() 。这期间的逻辑关系也很简单，假如 Keys 先处理按键，如它吃掉了某个键，它所依附的 Item 对象就收不到这个按键了；反之亦然。

    Qt Quick 提供的一些元素本身会处理按键，比如示例中的 CheckBox ，它响应空格键来选中或取消选中。而我们不需要给它附加 Keys 对象来再次处理按键事件。当然，如果你想改变它的按键响应逻辑，可以这么做，在解释 priority 属性时已经提到这点。

    最后还有一点要说明的是，如果你想某个元素处理按键，需要把焦点给它，这通过 Item 的 focus 属性来控制，置 true 即可。

    现在再来解释下示例代码。

    Rectangle 对象的附加信号处理器 Keys.onEscapePressed 调用 Qt.quit() 退出，小白很，不说了。

    Text 对象实现了 Keys.onPressed 附加信号处理器，使用 switch-case 语句，分拣 event 参数的 key 属性。如果是上下左右四个键，就变更 Text 的位置，置 accepted 为 true ，声明这几个按键已名花有主找到归宿；否则就直接返回，给别人机会处理按键。你也看到了，正是因为这样， CheckBox 才能拿到空格键来选中或取消复选框。

    示例中的 CheckBox 对象定义时，没有专门处理按键，因为 Qt Quick 提供的实现已经处理了按键了。

    嗯嗯，貌似内容很少？走着，看定时器去。

**定时器**

    定时器的作用还要说吗？好像有点儿啰嗦了呀。定时器么，就是周期性触发的一个事件，和平常用的闹钟差不多。你可以利用定时器来完成一些周期性的任务，比如检查和服务器的连接呆死了没，比如备份用户数据……

**定时器对象介绍**

    在 QML 中， Timer 代表定时器，使用起来也很简单，响应其 onTriggered() 信号即可，它也就这么一个有用的信号。另外它还有几个属性要说明一下， interval 指定定时周期，单位是毫秒，默认值是 1000 毫秒； repeat 设定定时器是周期性触发还是一次性触发，默认是一次性的（好像和 QTimer 不一样嗳）；running 属性，设置为 true 定时器就开始工作，设置为 false 就歇菜，默认是 false ； triggeredOnStart 属性，怎么说呢， Qt 总是对我们这么好都有点儿那啥不好意思了，这个属性是考虑到有些同志的特殊需求，本来定时器启动后要等待设定的间隔才触发，如果你设置这个属性为 true ，那定时器开始执行时立马先触发一次，默认值是 false 。

    Timer 还有 start() / stop() / restart() 三个方法可以调用，它们会影响 running 属性，望文生义吧您。

    现在来看一个简单的示例，倒计时。

**倒计时程序**

    世界杯倒计时按天算，山中一日世上千年，咱们这个示例用1秒顶它一天，倒数十秒，然后就开香槟庆祝下。

    看代码（count\_down.qml）：

**[javascript]** [view plaincopy](http://blog.csdn.net/foruok/article/details/30474299)[在CODE上查看代码片](https://code.csdn.net/snippets/390873)

1. **import** QtQuick 2.0
2. **import** QtQuick.Controls 1.1
4. Rectangle {
5. width: 320;
6. height: 240;
7. color: "gray";
8. QtObject{
9. id: attrs;
10. property **int** counter;
11. Component.onCompleted:{
12. attrs.counter = 10;
13. }
14. }
16. Text {
17. id: countShow;
18. anchors.centerIn: parent;
19. color: "blue";
20. font.pixelSize: 40;
21. }
23. Timer {
24. id: countDown;
25. interval: 1000;
26. repeat: **true**;
27. triggeredOnStart: **true**;
28. onTriggered:{
29. countShow.text = attrs.counter;
30. attrs.counter -= 1;
31. **if**(attrs.counter < 0)
32. {
33. countDown.stop();
34. countShow.text = "Clap Now!";
35. }
36. }
37. }
39. Button {
40. id: startButton;
41. anchors.top: countShow.bottom;
42. anchors.topMargin: 20;
43. anchors.horizontalCenter: countShow.horizontalCenter;
44. text: "Start";
45. onClicked: {
46. countDown.start();
47. }
48. }
49. }

    我在界面上放了个 Text 对象，它下面放一按钮。 Rectangle 对象内定义了一个 Timer 对象，默认不启动。当用户点击 "Start" 按钮时启动定时器。我还设置了定时器的 triggeredOnStart 属性哦，周期是 1 秒。

    计数保存在 QtObject 对象中， id 是 attrs ，在附加信号处理器 Component.onCompleted 中初始化 counter 属性的值为 10 。而在 Timer 对象的 onTriggered 信号处理器中递减 counter ，当 counter 为 0 时修改 Text 对象的文本为 "Clap Now!" 。

    喏，就这么简单。

    来看下效果。图 5 是初始效果：

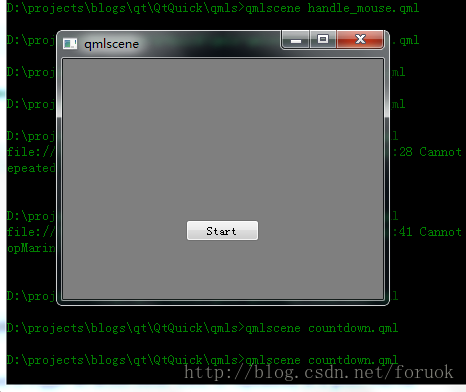


            图 5 倒计时程序初始效果

    图 6 是计时效果：

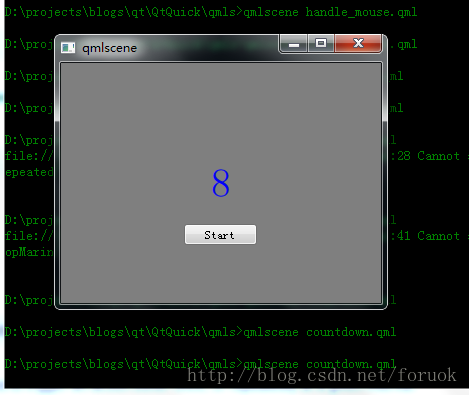


            图 6 计时

    图 7 是倒计时结束的效果：

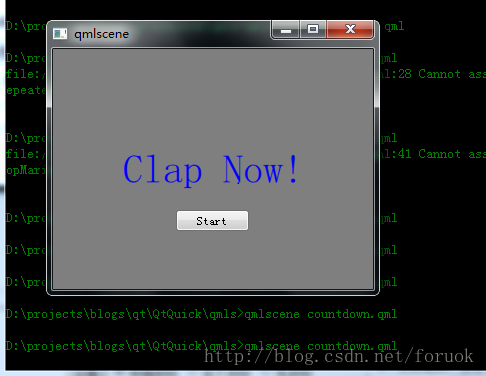


            图 7 倒计时结束

    如果你使用 qmlscene 运行 countdown.qml 文档，可能会发现它有一个 BUG 哦， "Start" 按钮第一次点击可以正常倒数计时，完了下次就不行了……我已经找到问题所在，不过还是留给你解决吧。