# 一去、二三里的博客

http://blog.sina.com.cn/liang19890820



个人资料 [管理]


一去、二三里

Qing 微博

博客等级：18
博客积分：78
博客访问：407,783
关注人气：378
获赠金笔：12
赠出金笔：0
荣誉徽章：

相关博文

- LINQ语法
  扯淡的烟酒生
- 女儿的幸福来自妈妈的教育
  油菜花开-
- 【奥地利】阿尔卑斯山下的世
  圣女莫尼卡
- 为何某些人总是盯着谢娜的肚子
  周易李秋雨
- 古镇月河街看做糖画
  晓薇
- 冲高回落后仍可积极参与
  财富龙哥
- 日剧经典美食【日式照烧鸡腿饭
  小豬食冊
- 不动产登记暂行条例—土地流转
  zhaocs0005
- 说过就要算数
  mj6655
- 枸杞可能导致流产！
  火爸朱剑笛

更多>>

正文 字体大小：大 中 小

## 快速入门

(2014-07-08 13:21:22) [编辑][删除] 转载

标签： qml quickstarter quick qtquick 分类： QML

**注**

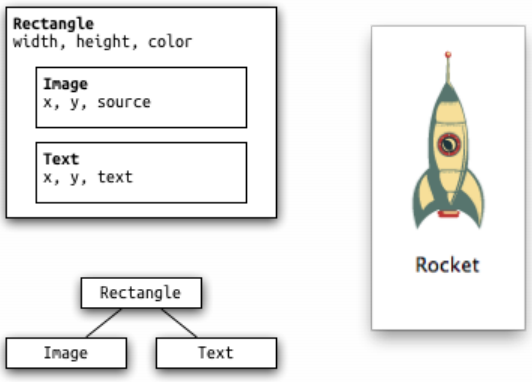
本章的源代码可以在资源文件夹中找到。

本章在Qt5中的概述中提供了QML、用户界面语言的声明。我们将讨论QML语法，这是一个树元素，然后最重要的基本元素的概述。稍后将简要介绍如何创建自己的元素，所谓的组件以及如何使用属性操作来改变元素。快要结束的时候，我们将介绍如何在布局中把元素结合到一起。最后，介绍用户可以提供输入的元素。

## 4.1.QML语法

QML是一种声明性语言，用来描述应用程序的用户界面。它将用户界面分解成更小的元素，可以与组件相结合。QML描述了这些用户界面的外观和元素行为。这种用户界面描述可以用JavaScript代码来充实，以提供简单但也更复杂的逻辑。从这个角度来看它遵循HTML JavaScript的模式，但QML是从底层向上的设计用来描述用户界面而不是文本文档。

其最简单的方式是QML元素的层次结构。子元素继承父坐标系，一个X，Y坐标总是相对于父组件。




让我们从一个简单的QML文件的例子开始，来解释不同的语法。

```
// rectangle.qml

import QtQuick 2.0

// The root element is the Rectangle
Rectangle {
    // name this element root
    id: root

    // properties: :
    width: 120; height: 240

    // color property
    color: "#D8D8D8"

    // Declare a nested element (child of root)
    Image {
```

```
        id: rocket

        // reference the parent
        x: (parent.width - width)/2; y: 40

        source: 'assets/rocket.png'
    }

    // Another child of root
    Text {
        // un-named element

        // reference element by id
        y: rocket.y + rocket.height + 20

        // reference root element
        width: root.width

        horizontalAlignment: Text.AlignHCenter
        text: 'Rocket'
    }
}
```

- import语句导入一个模块到一个特定的版本。一般来说，总是要导入QtQuick 2.0作为初始元素的集合。
- 注释可以使用//进行单行注释，或使用进行多行注释。就像C/C++和JavaScript中一样。
- 每一个QML文件需要有且只有一个根元素，如HTML。
- 元素由它的类型后跟{ }声明
- 元素可以有属性，形式为name : value
- QML文档中的任意元素可以通过使用它们的id（一个带引号的标识符）进行访问。
- 元素可以嵌套，意味着一个父元素可以有子元素，父元素可以使用parent关键字来访问。

**提示**

通常想通过id访问一个特定的元素或使用parent关键字访问父元素。所以最好的做法是使用id: root来命名根元素。然后，就不必去考虑根元素的在QML文件中如何命名的。

## 4.1.1属性

元素通过使用自己的元素名来声明，但是通过使用它们的属性或创建自定义属性被定义。属性是一个简单的键-值对，例如：width : 100，text: 'Greetings'，color: '#FF0000'。一个属性具有良好定义的类型且可以有一个初始值。

```
Text {
        // (1) identifier
        id: thisLabel

        // (2) set x- and y-position
        x: 24; y: 16

        // (3) bind height to 2 * width
        height: 2 * width

        // (4) custom property
        property int times: 24

        // (5) property alias
        property alias anotherTimes: thisLabel.times

        // (6) set text appended by value
        text: "Greetings " + times

        // (7) font is a grouped property
        font.family: "Ubuntu"
        font.pixelSize: 24

        // (8) KeyNavigation is an attached property
        KeyNavigation.tab: otherLabel

        // (9) signal handler for property changes
        onHeightChanged: console.log('height:', height)

        // focus is neeed to receive key events
        focus: true

        // change color based on focus value
        color: focus?"red":"black"
    }
```

让我们浏览一下不同特征的属性：

（1）id是一个非常特殊的属性，用来引用QML文件中的元素（QML中称为文档）。id是不是字符串类型，而是一个标识符，是QML语法的一部分。一个id需要在文件里是唯一的，它不能被重新设置为不同的值，也不能进行查询（它的行为更像C++中的指针）。
（2）一个属性可以被设置为一个值，这取决于它的类型。如果属性没有值，则会给出一个初始的默认值。这则需要咨询特定元素，来查看有关属性的初始值的详细信息。
（3）属性可以依赖于一个或多个其它属性，这就是所谓的绑定。当依赖属性发生改变，绑定属性也会被更新，它的工作原理就像一个合同，这种情况下，高度应该总是2倍的宽度。
（4）给元素添加自定义属性，使用property限定符后面跟随类型，名称，和可选的初始值( property : )。如果没有初始值，则会给出一个系统的初始值。

**注**
也可以声明一个属性为默认属性，如果从开始到最后都没有通过default关键字声明属性。这用于示例，当添加子元素，该子元素会被自动添加到类型列表的默认属性children中，如果它们是可见的。

（5）声明属性的另一个重要方式是使用alias关键字（ property alias : ）。alias关键字允许我们转发对象或对象本身的属性由内部的类型到一个外部范围。我们以后使用这种技术，当定义组件导出内部属性或元素的id到根级。属性别名并不需要一个类型，它使用了属性或对象的引用类型。
（6）text属性依赖于自定义属性times的int类型。基于int的值会自动转换为字符串类型，表达式本身是另一个绑定例子，且转化为文本每次更新times属性更改时。
（7）一些属性是分组属性。此功能用于当一个属性更有条理的和相关的属性被组合在一起时。写分组的属性的另一种方式是font { family: "Ubuntu"; pixelSize: 24 }。
（8）一些属性附加到该元素本身。这样做是为了全局相关的元素在应用程序（如键盘输入）只出现一次，可以这样写.:。
（9）对于每一个属性，可以提供一个信号处理程序，在该属性更改后处理程序被调用。例如，在这里我们希望的高度变化时收到通知，并使用内置的控制台来记录消息到系统中。

**警告**
一个元素的id应该只用于引用文档中的元素（例如当前文件）。QML提供了一个称为动态范围机制，其中后面加载的文件覆盖该元素id前面加载的文件。这使得它可以参考元素的id前面加载的文件，如果它们还没有被覆盖。这就像创建全局变量。不幸的是这通常会导致非常糟糕的代码在实践中，其程序依赖的执行顺序。不幸的是这不能被关闭。请谨慎使用，甚至最好不要使用这种机制。这更好地导出你想要提供的元素到外界使用属性来在文档的根元素。

## 4.1.2脚本处理

QML和JavaScript（也称为EcmaScript的）是最好的朋友。在JavaScript一章中，我们将进入更详细的介绍这种共生关系。目前，只是想让你知道这种关系。

```
Text {
        id: label

        x: 24; y: 24

        // custom counter property for space presses
        property int spacePresses: 0

        text: "Space pressed: " + spacePresses + " times"

        // (1) handler for text changes
        onTextChanged: console.log("text changed to:", text)

        // need focus to receive key events
        focus: true

        // (2) handler with some JS
        Keys.onSpacePressed: {
            increment()
        }

        // clear the text on escape
        Keys.onEscapePressed: {
            label.text = ''
        }

        // (3) a JS function
        function increment() {
            spacePresses = spacePresses + 1
        }
    }
```

1. 文本变化处理器onTextChanged打印当前文本，当文本每次改变且按下空格键时。

2. 当文本元素收到空格键（因为用户在键盘上按下了空格键），我们调用一个JavaScript函数`increment()`。
3. 一个JavaScript函数的定义形式`function () { ... }`，其增加spacePressed数量。每次`spacePressed`递增绑定属性也将被更新。

注

QML:（绑定）和JavaScript=（赋值）的区别是，该绑定是一个约束，绑定的生命周期里为真，而JavaScript的赋值（=）是一次性的赋值。绑定的生命周期结束，当一个新的绑定设置到该属性，甚至当一个JavaScript值分配给该属性。例如一键处理设置text属性设置为空字符串会破坏我们的增量显示：

```
Keys.onEscapePressed: {
    label.text = ''
}
```

按Escape键，按下空格键后不会再更新显示以前绑定的`text`属性（text: “Space pressed: ” + spacePresses + “ times” ）被销毁。

当你有冲突的策略，以更改属性在这种情况下（通过文字通过有约束力的文本，并通过一个JavaScript分配清除更改属性增量更新），那么你可以不使用绑定！你需要在这两个属性更改路径使用的绑定将被分配被销毁（违反了合同！）分配。

## 4.1.基本元素

元素可被分成视觉和非视觉元素。可视化元素（如矩形Rectangle）具有几何形状，通常在屏幕上呈现区域。一个非可视化元素（如定时器Timer）提供了一般的功能，通常用于操作可视元素。

目前，我们将专注于基本视觉元素，如`Item`，`Rectangle`，`Text`，`Image`和`MouseArea`.

### 4.2.1.Item元素

`Item`是所有视觉元素的基本元素，所有其他的视觉元素从`Item`继承。它本身不画什么，但定义了所有的视觉元素是共同的属性：

| 组 | 属性 |
| --- | --- |
| Geometry | x和y定义左上角的位置，`width`和`height`扩展的元素，同时在z堆叠顺序举起元素或从他们的自然顺序向下 |
| 布局处理 | anchors（左，右，上，下，垂直和水平中心）定位相对于其他元素与元素的边距 |
| 按键处理 | 连接`Key`和`KeyNavigation`属性来控制键的处理和输入焦点属性，启用密钥处理摆在首位 |
| 转型 | `scale`和`rotate`变换和通用的`transform`属性列表中X，Y，Z变换及其`transformOrigin`点 |
| 视觉 | `opacity`来控制透明度，`visible`显示/隐藏要素，`clip`抑制漆操作单元边界光滑，`smooth`提升渲染质量 |
| 状态定义 | `states`属性列表的支持列表状态和当前状态属性也转换动画状态更改属性列表。 |

为了更好地理解不同的属性，会尽量在本章中元素的上下文中介绍他们。请记住这些基本属性在每一个视觉元素都是可用的，和这些元素一样工作。

注

Item元素经常被用来作为一个容器等元素，类似于HTML中的div元素。

### 4.2.2.Rectangle元素

`Rectangle`扩展`Item`，并添加填充颜色。附加支持通过`border.color`和`border.width`定义的边界。要创建圆角矩形可以使用`radius`属性。

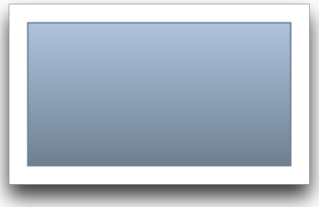
```
Rectangle {
    id: rect1
    x: 12; y: 12
    width: 76; height: 96
    color: "lightsteelblue"
}
Rectangle {
    id: rect2
    x: 112; y: 12
    width: 76; height: 96
    border.color: "lightsteelblue"
    border.width: 4
    radius: 8
}
```

注

使用命名的颜色是通过SVG颜色命名的颜色（见http://www.w3.org/TR/css3-color/#svg-color）。可以在QML以不同的方式提供颜色，最常见的方式是如RGB字符串（'＃FF4444'）或颜色名称（如“white”）。

除了填充颜色和边框的矩形还支持自定义渐变。

```
Rectangle {
        id: rect1
        x: 12; y: 12
        width: 176; height: 96
        gradient: Gradient {
            GradientStop { position: 0.0; color: "lightsteelblue" }
            GradientStop { position: 1.0; color: "slategray" }
        }
        border.color: "slategray"
    }
```



梯度是由一系列渐变梯度限定。每个限定都有一个位置和颜色。位置标记在Y轴（0 =顶部，1 =底部）的位置。`GradientStop`在该位置标记的颜色。

注

没有设置宽/高矩形将不可见。这种情况经常发生，当有几个矩形的宽度（高度）相互依赖，彼此的东西在程序中出了问题。所以要当心！

### 4.2.3.Text元素

要显示的文本可以使用`Text`元素。其最显着的属性是`string`类型的`text`属性。元素计算基于给定的文本和所使用的字体的初始宽度和高度。字体可以使用的字体属性组（如`font.family`，`font.pixelSize`，...）的影响。要更改文本的颜色只使用颜色属性。

```
    Text {
        text: "The quick brown fox"
        color: "#303030"
        font.family: "Ubuntu"
        font.pixelSize: 28
    }
```


文本可以使用`horizontalAlignment`和`verticalAlignment`属性对齐各边和中心。为了进一步提高渲染可以使用`style`和`styleColor`属性，它可以让你呈现在大纲文本的轮廓，提高凹陷模式。对于你经常要定义的长文本，可以通过使用`elide`属性来实现。`elide`属性允许您设置elid位置到左，右或文本的中间。如果你不想要的'...'elid的方式出现，但仍希望看到全文，你也可以换用了`wrapMode`属性中的文本（仅在宽度显式设置）：

```
Text {
    width: 40; height: 120
    text: 'A very long text'
    // '...' shall appear in the middle
```

```
    elide: Text.ElideMiddle
    // red sunken text styling
    style: Text.Sunken
    styleColor: '#FF4444'
    // align text to the top
    verticalAlignment: Text.AlignTop
    // only sensible when no elide mode
    // wrapMode: Text.WordWrap
}
```

Text元素只显示给定的文本。它不提供任何背景装饰。除了呈现的文本的Text元素是透明的。这是你的整体设计的一部分来提供一个合理的背景文本元素。

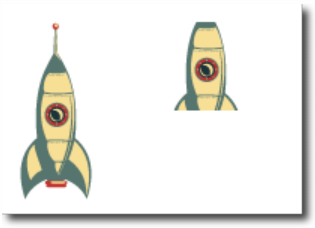
注

要知道一个Text初始宽度（高度）是根据文本字符串，并在字体集。没有设置宽度，没有文本的Text元素将不可见，作为初始宽度为0。

### 4.2.4.Image元素

一个Image元素能够以各种格式显示图像（例如，PNG，JPG，GIF，BMP）。有关支持的图像格式的完整列表，请参阅Qt文档。除了明显的source属性提供图像的URL，它包含一个fillMode控制的调整大小行为。

```
Image {
      x: 12; y: 12
      // width: 48
      // height: 118
      source: "assets/rocket.png"
  }
  Image {
      x: 112; y: 12
      width: 48
      height: 118/2
      source: "assets/rocket.png"
      fillMode: Image.PreserveAspectCrop
      clip: true
  }
```



注

一个URL可以是本地路径以正斜杠（"./images/home.png"），或者一个网络链接（如“http://example.org/home.png”）。

Image元素使用PreserveAspectCrop图像元素也应该使裁剪，以避免图像边界之外所要呈现的Image数据。默认情况下剪辑被禁用（clip : false）。您需要启用剪辑（clip : true），以抑制绘画边框的元素。可以用在任何视觉元素。

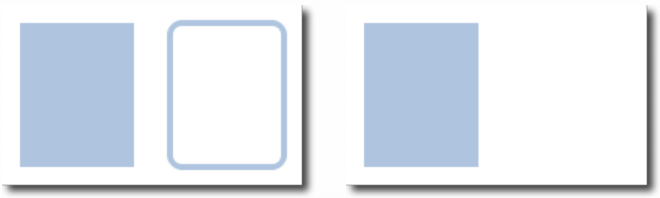
### 4.2.4.MouseArea元素

与这些元素交互你经常会用一个MouseArea。在那里你可以捕捉鼠标事件的矩形不可见项。鼠标区域通常和一个可见项一起使用，当与用户的视觉部分交互，以执行命令。

```
Rectangle {
      id: rect1
      x: 12; y: 12
      width: 76; height: 96
      color: "lightsteelblue"
      MouseArea {
          id: area
          width: parent.width
          height: parent.height
          onClicked: rect2.visible = !rect2.visible
      }
  }

  Rectangle {
      id: rect2
      x: 112; y: 12
```

```
        width: 76; height: 96
        border.color: "lightsteelblue"
        border.width: 4
        radius: 8
    }
```



注

这是Qt Quick中的一个重要方面，对输入的处理从视觉呈现分离。通过这一点，可以显示用户界面元素，但互动区域可以更大。

## 4.3.组件

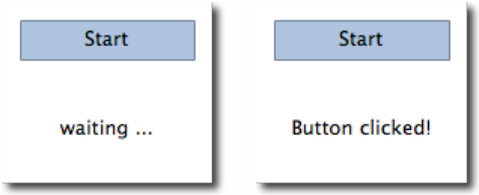
组件是一个可重用的元素，QML提供了不同的方法来创建组件。但目前我们只关心唯一途径：一种基于文件的组成部分。基于文件的组成部分是由放置在一个文件中的QML元素，给文件中的元素名称（例如`Button.qml`）创建的。您可以使用组件就像从Qt Quick模块的所有其他元素，你会在我们的例子中使用这个在你的代码中`Button { ... }`

我们来看看这个例子。我们创建一个包含文本和鼠标区域的矩形。这类似于一个简单的按钮，我们的目的并不需要更复杂了。

```
    Rectangle { // our inlined button ui
        id: button
        x: 12; y: 12
        width: 116; height: 26
        color: "lightsteelblue"
        border.color: "slategrey"
        Text {
            anchors.centerIn: parent
            text: "Start"
        }
        MouseArea {
            anchors.fill: parent
            onClicked: {
                status.text = "Button clicked!"
            }
        }
    }

    Text { // text changes when button was clicked
        id: status
        x: 12; y: 76
        width: 116; height: 26
        text: "waiting ..."
        horizontalAlignment: Text.AlignHCenter
    }
```

用户界面看起来与此类似。左边的UI是在初始状态下，右侧的按钮被点击后。




我们的任务是现在提取按钮的UI在一个可重用组件。为此，我们认为短期内对我们的按钮一个可能的API。您可以通过做到这一点想象自己别人应该如何使用您的按钮。这里是我的想象：

```
// my ideal minimal API for a button
Button {
    text: "Click Me"
    onClicked: { // do something }
}
```

我想用一个`text`属性设置文本，并实现自己的单击处理程序。此外，我会期望按钮有一个合理的初始大小，这点我可以覆盖（例如使用`width: 240`为例）。

为了实现这一目标，我们创建一个`Button.qml`文件，并在界面内复制我们的按钮。此外，我们需要导出一个用

户可能要更改根级属性。

```
// Button.qml

import QtQuick 2.0

Rectangle {
    id: root
    // export button properties
    property alias text: label.text
    signal clicked

    width: 116; height: 26
    color: "lightsteelblue"
    border.color: "slategrey"

    Text {
        id: label
        anchors.centerIn: parent
        text: "Start"
    }
    MouseArea {
        anchors.fill: parent
        onClicked: {
            root.clicked()
        }
    }
}
```

我们已经导出的文本和根级别点击信号。通常情况下我们命名根元素为 root，使引用更容易。我们在QML中使用alias特性，这是一种内部嵌套的QML元素属性导出到根级别，使这适用于外部世界的别名功能。重要的是要知道，只有根级别属性可以从该文件以外的其他组件来访问。

使用我们新的Button元素，可以在我们的简单的文件声明它。所以刚才的例子会变得有点简化。

```
    Button { // our Button component
        id: button
        x: 12; y: 12
        text: "Start"
        onClicked: {
            status.text = "Button clicked!"
        }
    }

    Text { // text changes when button was clicked
        id: status
        x: 12; y: 76
        width: 116; height: 26
        text: "waiting ..."
        horizontalAlignment: Text.AlignHCenter
    }
```

现在，你可以在用户界面通过仅仅使用 `Button { ... }` 来构建更多的按钮。一个真正的按钮可以更复杂，例如在单击时提供反馈或表现出更好的装饰。

注

个人认为，你甚至可以更进一步，用一个项目作为一个根元素。这可以防止用户更改我们的设计按钮的颜色，并为我们提供有关导出API的更多控制。目标应该是导出一个最小的API。实际上，这意味着我们将使用Item取代根Rectangle，使矩形的嵌套为根项的嵌套元素。

```
Item {
    id: root
    Rectangle {
        anchors.fill: parent
        color: "lightsteelblue"
        border.color: "slategrey"
    }
    ...
}
```

利用这种技术很容易地创建了一系列可重用的组件。

注：

技术在于交流、沟通，转载请注明出处并保持作品的完整性。

作者：`\☆奋斗ing❤孩子`  原文：http://blog.sina.com.cn/s/blog_a6fb6cc90101i7ok.html。

2

喜欢

分享：

阅读(686) | 评论 (6) | 收藏(0) | 已有3人转载▼ | 喜欢▼ | 打印 已投稿到：排行榜

前一篇：QUiLoader加载ui文件

后一篇：Qt之操作系统环境

评论 重要提示：警惕虚假中奖信息 [发评论]

EricLXing

博主终于更新了，继续支持……

2014-7-9 14:01 回复(1)

Rev01

赞，代码高亮之后舒服多了

2014-7-22 14:13 回复(1)

幻想家Corey

翻译?

2014-8-31 18:30 回复(1)

发评论

一去、二三里：

分享到微博 新

匿名评论

验证码： 请点击后输入验证码 收听验证码

发评论

以上网友发言只代表其个人观点，不代表新浪网的观点或立场。

< 前一篇
QUiLoader加载ui文件

后一篇 >
Qt之操作系统环境