



野鸭湖秋忆



怀念圣托里尼的阳光



追寻远去的长城



民国时期的北平女警

查看更多>>

谁看过这篇博文

	右右youyou	2月24日
	八戒你变...	2月14日
	鑫语馨缘	1月30日
	用户24680...	1月27日
	Amast	1月24日
	MountainSea	1月20日
	夕下	1月6日
	羽之翼2013	12月29日
	着你追	12月9日
	乒羽cl	12月1日
	雨天的太阳	11月24日
	ants	11月7日

```
height: bg.height

Image { // nice background image
  id: bg
  source: "assets/background.png"
}

MouseArea {
  id: backgroundClicker
  // needs to be before the images as order matters
  // otherwise this mousearea would be before the other elements
  // and consume the mouse events
  anchors.fill: parent
  onClicked: {
    // reset our little scene
    rocket1.x = 20
    rocket2.rotation = 0
    rocket3.rotation = 0
    rocket3.scale = 1.0
  }
}

ClickableImage {
  id: rocket1
  x: 20; y: 100
  source: "assets/rocket.png"
  onClicked: {
    // increase the x-position on click
    x += 5
  }
}

ClickableImage {
  id: rocket2
  x: 140; y: 100
  source: "assets/rocket.png"
  smooth: true // need antialiasing
  onClicked: {
    // increase the rotation on click
    rotation += 5
  }
}

ClickableImage {
  id: rocket3
  x: 240; y: 100
  source: "assets/rocket.png"
  smooth: true // need antialiasing
  onClicked: {
    // several transformations
    rotation += 5
    scale -= 0.05
  }
}
```

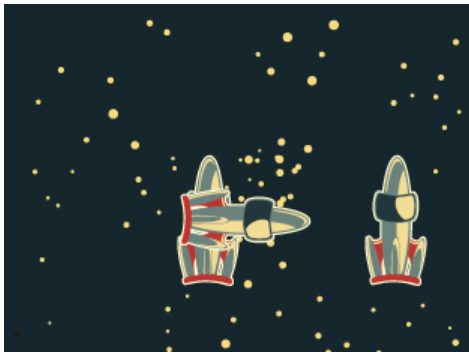


火箭1每次点击在x位置增量5像素，火箭2每次点击都将旋转，火箭3每次点击将旋转和缩小。对于缩放和旋转操作，我们设置smooth: true来抗锯齿，其被关闭（同裁剪属性clip）性能的原因。当你在自己的工作中看到一些图形光栅边缘，那么很可能你想切换smooth。

背景遥控器MouseArea覆盖整个背景和重置火箭值。

注：

元素出现在代码有较低的叠加顺序(称为z值)，如果单击火箭1足够长时间，你会看到它下面火箭2移动。Z顺序也可以通过该项目的z属性操作。



这是因为rocket2出现在之后的代码，同样也适用于鼠标区域。鼠标区域在之后的代码将重叠(因此抓住鼠标事件)早些时候鼠标区域代码。请记住：在文档中元素的顺序很重要。

4.5. 定位元素

还有一些用于定位项目的QML元素。这些被称为定位及QtQuick模块的 `Row`，`Column`，`Grid` 和 `Flow`。他们可以在下面的插图中看到显示同样的内容。

注：

在我们进入细节之前，先介绍一些辅助元素。红，蓝，绿，亮区和暗正方形。每个组件都包含一个48×48像素的彩色矩形。参考这里的RedSquare的源代码：

```
// RedSquare.qml

import QtQuick 2.0

Rectangle {
    width: 48
    height: 48
    color: "#ea7025"
    border.color: Qt.lighter(color)
}
```

请注意使用`Qt.lighter(color)`，以产生基于该填充颜色较浅边框颜色。我们将使用这些助手在下面的例子中，使源代码更紧凑，增加可读性。请记住，每个矩形最初是48×48像素。

`Column` 元素可以让子项目从上到下依次叠加。`spacing`属性可用于设置每一个子元素之间的间距。



```
// column.qml

import QtQuick 2.0

DarkSquare {
    id: root
    width: 120
    height: 240

    Column {
        id: row
        anchors.centerIn: parent
        spacing: 8
        RedSquare {}
        GreenSquare { width: 96 }
        BlueSquare {}
    }
}

// M1<<
```

`Row`元素可以让其子项彼此相邻，无论是从左侧到右侧，或者从右侧到左侧，这取决于的`layoutDirection`属

性。同样，`spacing`是用来设置子项之间的间距。



```
// row.qml

import QtQuick 2.0

BrightSquare {
    id: root
    width: 400; height: 120

    Row {
        id: row
        anchors.centerIn: parent
        spacing: 20
        BlueSquare { }
        GreenSquare { }
        RedSquare { }
    }
}
```

`Grid`元素排列其子项到网格中，通过设置`rows`和`columns`属性，可以限制行或列的数量。通过不设置其中一项，另一项是从子项的数量来计算。例如，设定3行和6个子项将会有2列。属性`flow`和`layoutDirection`被用来控制在项目被添加到网格的顺序，而`spacing`控制的子项的间距。



```
// grid.qml

import QtQuick 2.0

BrightSquare {
    id: root
    width: 160
    height: 160

    Grid {
        id: grid
        rows: 2
        columns: 2
        anchors.centerIn: parent
        spacing: 8
        RedSquare { }
        RedSquare { }
        RedSquare { }
        RedSquare { }
    }
}
```

最终的定位是`Flow`。它的子项处在流中。流的方向使用`flow`和`layoutDirection`控制。它可以横向或从顶部至底部运行。它也可以从左至右或在相反的方向。若项目被添加在流中，它们被包裹形成新的行或列。为了使流工作，它必须有一个宽度或高度。可以被直接设置地或锚布局。



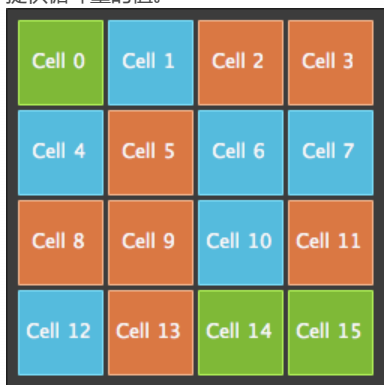
```
// flow.qml

import QtQuick 2.0

BrightSquare {
    id: root
    width: 160
    height: 160

    Flow {
        anchors.fill: parent
        anchors.margins: 20
        spacing: 20
        RedSquare { }
        BlueSquare { }
        GreenSquare { }
    }
}
```

常与定位中使用的元素是**Repeater**。它的工作原理就像一个循环遍历模型。最简单的情况下的模型仅仅是一个提供循环量的值。



```
// repeater.qml

import QtQuick 2.0

DarkSquare {
    id: root
    width: 252
    height: 252
    property variant colorArray: ["#00bde3", "#67c111", "#ea7025"]

    Grid{
        anchors.fill: parent
        anchors.margins: 8
        spacing: 4
        Repeater {
            model: 16
            Rectangle {
                width: 56; height: 56
                property int colorIndex: Math.floor(Math.random()*3)
                color: root.colorArray[colorIndex]
                border.color: Qt.lighter(color)
                Text {
                    anchors.centerIn: parent
                    color: "#f0f0f0"
                    text: "Cell " + index
                }
            }
        }
    }
}
```

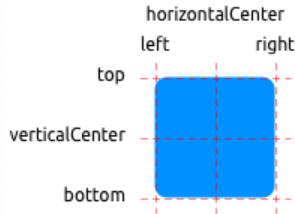
}

在这个转发器例子中，我们使用了一些新的魔法。我们定义自己的颜色属性，我们使用了一个颜色数组。转发器将创建一系列矩形（16，由模型所定义的）。对于每一个循环，他创造的定义为转发器子矩形。在矩形我们选择的颜色用JS数学函数`Math.floor(Math.random()*3)`。这给了我们一个0到2的随机数范围，我们用它来选择我们的颜色阵列的颜色。如前所述JavaScript是QtQuick的一个核心部分，因此可用于我们的标准库。

转发器注入的索引属性到转发器。它包含当前循环索引。（0,1, ...,15）。我们在这基础上利用索引可以处理一些情况，例如，以可视化的当前索引的Text元素。

4.6. 布局项目

QML提供了一种灵活的方式来布局采用锚项目。锚定的概念是项基本属性的一部分,提供给所有视觉QML元素。一个锚的作用就像一个合同，比竞争的几何形状的变化更强。锚是相对关系的表达式，你总是需要一个相关的元素锚。



一个元素有6大锚线（上，下，左，右，水平居中，垂直居中）。其他有基线锚文本中的文字内容。每个锚系配备了一个偏移量。在上，下，左，右的情况下他们被称为边缘。为水平居中，垂直居中和基线它们被称为偏移。



1.一个元素填充一个父元素

```
GreenSquare {
    BlueSquare {
        width: 12
        anchors.fill: parent
        anchors.margins: 8
        text: '(1)'
    }
}
```

2.一个元素左对齐到父元素

```
GreenSquare {
    BlueSquare {
        width: 48
        y: 8
        anchors.left: parent.left
        anchors.leftMargin: 8
        text: '(2)'
    }
}
```

3.一个元素的左侧对齐的父元素右侧

```
GreenSquare {
    BlueSquare {
        width: 48
        anchors.left: parent.right
        text: '(3)'
    }
}
```

4. 中心对齐元素。**Blue1**是水平以父元素为中心。**Blue2**也是水平居中，但**Blue2**和它的顶部对**Blue1**的底线。

```
GreenSquare {
    BlueSquare {
        id: blue1
        width: 48; height: 24
        y: 8
        anchors.horizontalCenter: parent.horizontalCenter
    }
    BlueSquare {
        id: blue2
        width: 72; height: 24
        anchors.top: blue1.bottom
        anchors.topMargin: 4
        anchors.horizontalCenter: blue1.horizontalCenter
        text: '(4)'
    }
}
```

5. 一个元素是集中在一个父元素

```
GreenSquare {
    BlueSquare {
        width: 48
        anchors.centerIn: parent
        text: '(5)'
    }
}
```

6. 一个元素居中在父元素上使用水平和垂直中心线

```
GreenSquare {
    BlueSquare {
        width: 48
        anchors.horizontalCenter: parent.horizontalCenter
        anchors.horizontalCenterOffset: -12
        anchors.verticalCenter: parent.verticalCenter
        text: '(6)'
    }
}
```

4.7. 输入元素

我们已经使用了**MouseArea**为鼠标输入元素。在这里，我们想更专注于键盘输入的可能性。我们用文本编辑元素开始：**TextInput**和**TextEdit**。

4.7.1. TextInput

TextInput允许用户输入一行文字。该元件提供了对输入的限制，例如**validator**和**inputMask**是**echoMode**。

```
// textinput.qml

import QtQuick 2.0

Rectangle {
    width: 200
    height: 80
    color: "linen"

    TextInput {
        id: input1
        x: 8; y: 8
        width: 96; height: 20
        focus: true
        text: "Text Input 1"
    }

    TextInput {
        id: input2
        x: 8; y: 36
        width: 96; height: 20
        text: "Text Input 2"
    }
}
```

}

用户可以点击`TextInput`里面更改焦点。为了支持键盘切换焦点，我们可以使用`KeyNavigation`附加属性。

```
// textinput2.qml

import QtQuick 2.0

Rectangle {
    width: 200
    height: 80
    color: "linen"

    TextInput {
        id: input1
        x: 8; y: 8
        width: 96; height: 20
        focus: true
        text: "Text Input 1"
        KeyNavigation.tab: input2
    }

    TextInput {
        id: input2
        x: 8; y: 36
        width: 96; height: 20
        text: "Text Input 2"
        KeyNavigation.tab: input1
    }
}
```

该`KeyNavigation`附加属性支持，其中一个元素的id被绑定到给定的按键切换焦点导航键的预设。

一个文本输入元件配备，除了一个闪烁的光标没有视觉呈现和输入的文本。为用户能识别的元素作为输入元件，它需要一些视觉装饰，例如一个简单的矩形。当把`TextInput`确保导出你希望别人能够访问的主要特性的元件中。

我们提出这一段代码到我们呼吁 `TLineEditV1` 再利用自己的组件。

```
// TLineEditV1.qml

import QtQuick 2.0

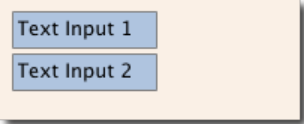
Rectangle {
    width: 96; height: input.height + 8
    color: "lightsteelblue"
    border.color: "gray"

    property alias text: input.text
    property alias input: input

    TextInput {
        id: input
        anchors.fill: parent
        anchors.margins: 4
        focus: true
    }
}
```

我们使用新`TLineEditV1`组件改写我们的`KeyNavigation`例子。

```
Rectangle {
    ...
    TLineEditV1 {
        id: input1
        ...
    }
    TLineEditV1 {
        id: input2
        ...
    }
}
```

尝试tab键进行定位。将看到焦点不会改变到focus:true是不充分的。问题出现了，那焦点被转移到

4.7.2. FocusScope

聚焦范围声明如果焦点范围接收焦点，具有焦点的最后一个子元素`focus:true`的获得焦点。因此，提出焦点最后集中请求的子元素。我们将创建TLineEdit组件的第二个版本，使用对焦范围为根元素称为TLineEditV2。

```
// TLineEditV2.qml

import QtQuick 2.0

FocusScope {
    width: 96; height: input.height + 8
    Rectangle {
        anchors.fill: parent
        color: "lightsteelblue"
        border.color: "gray"
    }

    property alias text: input.text
    property alias input: input

    TextInput {
        id: input
        anchors.fill: parent
        anchors.margins: 4
        focus: true
    }
}
```

我们的示例将现在看起来像这样:

```
Rectangle {
    ...
    TLineEditV2 {
        id: input1
        ...
    }
    TLineEditV2 {
        id: input2
        ...
    }
}
```

按tab键现在已经可以成功切换2组件以及组件中子元素之间的焦点。

4.7.3. TextEdit

`TextEdit`非常类似`TextInput`，支持多行文本编辑字段。它忽略了文本的约束属性，这也提供了查询文本（`paintHeight`，`paintWidth`）的绘制的大小。我们还创建了自己的组件叫做`TTextEdit`提供了编辑的背景，并使用对焦范围更集中转发。

```
// TTextEdit.qml

import QtQuick 2.0

FocusScope {
    width: 96; height: 96
    Rectangle {
        anchors.fill: parent
        color: "lightsteelblue"
        border.color: "gray"
    }

    property alias text: input.text
    property alias input: input
}
```

```

TextEdit {
    id: input
    anchors.fill: parent
    anchors.margins: 4
    focus: true
}

```

可以像TLineEdit组件一样使用它

```

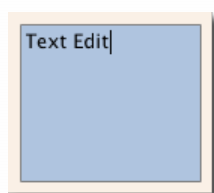
// textedit.qml

import QtQuick 2.0

Rectangle {
    width: 136
    height: 120
    color: "linen"

    TTextEdit {
        id: input
        x: 8; y: 8
        width: 120; height: 104
        focus: true
        text: "Text Edit"
    }
}

```



4.7.3. 按键元素

附加属性按键可以根据一定的按键来执行代码。例如向左、右移动的正方形我们可以连接的上，下，左，右按键来转换元件和所述加，减键来缩放元件。

```

// keys.qml

import QtQuick 2.0

DarkSquare {
    width: 400; height: 200

    GreenSquare {
        id: square
        x: 8; y: 8
    }
    focus: true
    Keys.onLeftPressed: square.x -= 8
    Keys.onRightPressed: square.x += 8
    Keys.onUpPressed: square.y -= 8
    Keys.onDownPressed: square.y += 8
    Keys.onPressed: {
        switch(event.key) {
            case Qt.Key_Plus:
                square.scale += 0.2
                break;
            case Qt.Key_Minus:
                square.scale -= 0.2
                break;
        }
    }
}

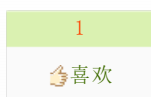
```



```
// todo
```

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_a6fb6cc90102uyua.html



分享:

阅读(275) | 评论(0) | 收藏(0) | 已有2人转载▼ | 喜欢▼ | 打印

已投稿到： 排行榜

前一篇：如何提取资源文件（QQ、360等）？

后一篇：获取当前系统的信息（CPU、物理内存、虚拟内存等）

评论

重要提示：警惕虚假中奖信息

[\[发评论\]](#)

做第一个评论者吧！  抢沙发>>

发评论

分享到微博

☐ 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
 如何提取资源文件（QQ、360等）？
 获取当前系统的信息（CPU、物理内存、虚拟内存等）
 后一篇 >

新浪BLOG意见反馈留言板
 不良信息反馈
 电话：4006900000
 提示音后按1键（按当地市话标准计费）
 欢迎批评指正
 新浪简介
 |
 About Sina
 |
 广告服务
 |
 联系我们
 |
 招聘信息
 |
 网站律师
 |
 SINA English
 |
 会员注册
 |
 产品答疑

Copyright © 1996 - 2014 SINA Corporation, All Rights Reserved
 新浪公司 版权所有