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：407,776
关注人气：378
获赠金笔：12
赠出金笔：0
荣誉徽章：

相关博文

- 齐亚籽的“神效”与被禁
  松鼠云无心
- 盘点结婚后坚决不生娃的10大
  我在题
- 银行板块走势短期内可能会很重
  聊四方
- 网传张伟平包养北电H女生美艳
  愚公愚乐
- 异地婚姻之伤：过年各找各妈
  吾家俩公主
- HR们不得不知的“培训五部曲
  蒋小华-
- 不学外语的美国怎么会有汉语热
  王青博士
- 12星座运势（3.2-3.8）
  百变巫女
- 降息没有改变反弹的本质
  大壮是只猫
- 周一没有V
  林毅

更多>>

正文 字体大小：大 中 小

流元素

(2014-10-13 15:27:35) [编辑][删除] 转载

标签： qml fluidelements animations states transitions 分类： QML

注

本章的源代码可以在资源文件夹中找到。

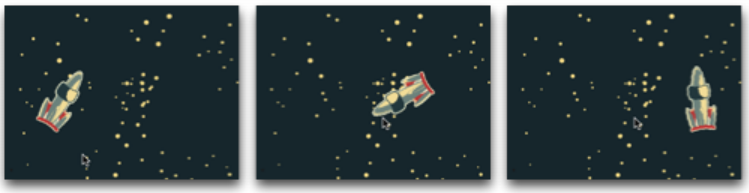
目前为止，我们大多介绍简单的图形元素以及如何操纵它们。这一章是关于如何通过动画的方式来控制属性值的变化。这种技术是现代流畅的用户界面的一个关键基础，通过使用状态和转换来扩展用户界面。每个状态定义了一组属性变更，可以结合动画状态改变，称为转换。

5.1. 动画

动画用于属性的改变。一个动画定义了差值曲线，将一个属性值从一个值平滑过渡到另一个值。动画是由一系列目标属性定义的，平滑的曲线能够定义一段时间内属性的持续变化。所有QtQuick中的动画由同一个计时器来控制，因此它们始终保持同步，这也提高了动画的性能和显示效果。

注

动画控制了属性的改变，即值插入。这是一个基本的概念，QML是基于元素、属性和脚本的。每个元素都提供了大量的属性，每个属性都在等待你使用动画。在这本书中你将会看到一个壮观的场景，你会发现自己在看一些动画，欣赏它们的美丽且肯定自己的创造性天赋。请记住：动画控制属性的改变，每个元素都有大量的属性供你使用。



```
// animation.qml

import QtQuick 2.0

Image {
    source: "assets/background.png"

    Image {
        x: 40; y: 80
        source: "assets/rocket.png"

        NumberAnimation on x {
            to: 240
            duration: 4000
            loops: Animation.Infinite
        }
        RotationAnimation on rotation {
            to: 360
            duration: 4000
            loops: Animation.Infinite
        }
    }
}
```

上面的例子在x坐标和rotation属性上应用了一个简单的动画。每一次动画有4000毫秒的持续时间并且永久循环。x轴坐标动画定义火箭的x坐标逐渐移至240px，旋转动画展示了当前角度到360度的旋转。两个动画并行运行启动UI加载。

现在可以通过to和duration属性来实现动画效果。或者可以在opacity或者scale上添加动画作为例子，集成这两个参数，可以实现火箭在太空中逐渐消失的效果，试试看!

5.1.1 动画元素

有很多类型的动画元素，每一种都有特定的效果，下面列出了一些常用的动画：

- PropertyAnimation（属性动画）- 使用属性值改变播放的动画
- NumberAnimation（数字动画）- 使用数字改变播放的动画
- ColorAnimation（颜色动画）- 使用颜色改变播放的动画
- RotationAnimation（旋转动画）- 使用旋转改变播放的动画

除了上面这些基本和广泛使用的动画元素，QtQuick还提供了一切特殊场景下使用的动画：

- PauseAnimation（停止动画）- 运行暂停一个动画
- SequentialAnimation（顺序动画）- 允许动画有序播放
- ParallelAnimation（并行动画）- 允许动画同时播放
- AnchorAnimation（锚定动画）- 使用锚定改变播放的动画
- ParentAnimation（父元素动画）- 使用父对象改变播放的动画
- SmoothedAnimation（平滑动画）- 跟踪一个平滑值播放的动画
- SpringAnimation（弹簧动画）- 跟踪一个弹簧变换的值播放的动画
- PathAnimation（路径动画）- 跟踪一个元素对象的路径的动画
- Vector3dAnimation（3D容器动画）- 使用QVector3d值改变播放的动画

我们将在后面学习怎样创建一系列动画。当使用更加复杂的动画时，我们需要在播放一个动画时改变一个属性或者运行一个脚本。对于这个问题，QtQuick提供了一些动作元素：

- PropertyAction（属性动作）- 在播放动画时改变属性
- ScriptAction（脚本动作）- 在播放动画时运行脚本

本章中我们将会使用一些小例子来讨论主要类型的动画。

5.1.2 应用动画

动画可以应用在以下几个方面：

- 动画属性 - 在元素完整加载后自动运行
- 动作属性 - 当属性值改变时自动运行
- 独立的动画 - 使用start()函数明确指定运行或者running属性被设置为true（比如通过属性绑定）

后面我们将讨论如何在状态变换时播放动画。

扩展可点击图片V2

为了演示动画的使用方法，我们重新实现了ClickableImage组件并且使用了一个文本元素（Text Element）来扩展它。

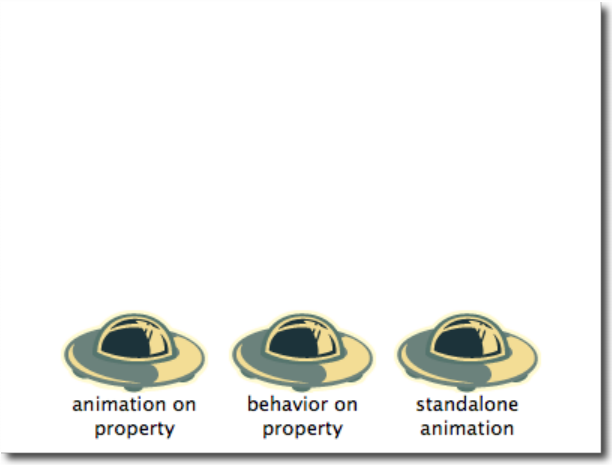
```
// ClickableImageV2.qml
// Simple image which can be clicked

import QtQuick 2.0

Item {
    id: root
    width: container.childrenRect.width
    height: container.childrenRect.height
    property alias text: label.text
    property alias source: image.source
    signal clicked

    Column {
        id: container
        Image {
            id: image
        }
        Text {
            id: label
            width: image.width
            horizontalAlignment: Text.AlignHCenter
            wrapMode: Text.WordWrap
            color: "#111111"
        }
    }

    MouseArea {
        anchors.fill: parent
        onClicked: root.clicked()
```

```
    }
}
```

为了给图片下面的元素定位，我们使用了`Column`定位器，并且使用基于列的`childrenRect`属性来计算它的宽度和高度。我们导出了两种属性：`text`和图形`source`属性，一个`clicked`信号。为了设置文本与图像一样宽并且可以自动换行，我们可以使用`Text`元素的`wrapMode`属性。

注

由于几何依赖关系的反转（父几何对象依赖于子几何对象），我们不能对ClickableImageV2设置宽度/高度，因为这样会破坏已经做好的宽度/高度属性绑定。作为一个设计组件者你需要明白这是我们内部设计的限制。通常更喜欢内部几何图像依赖于父几何对象。



三个火箭位于相同的y轴坐标（`y=200`）。它们都需要移动到`y=40`。每一个火箭都使用了一种方法来完成这个功能。

```
ClickableImageV3 {
        id: rocket1
        x: 40; y: 200
        source: "assets/rocket2.png"
        text: "animation on property"
        NumberAnimation on y {
            to: 40; duration: 4000
        }
    }
```

第一个火箭

第一个火箭使用了`Animation on`策略。动画会立即播放。当火箭被点击y轴坐标被重置到开始的位，这将应用到所有火箭。在动画播放时重置第一个火箭不会有任何影响。在动画开始前几分之一秒设置一个新的y轴坐标让人感觉挺不安全的，这样的竞争性变化属性值应当被避免。

```
    ClickableImageV3 {
        id: rocket2
        x: 152; y: 200
        source: "assets/rocket2.png"
        text: "behavior on property"
        Behavior on y {
            NumberAnimation { duration: 4000 }
        }

        onClicked: y = 40
        // random y on each click
//        onClicked: y = 40+Math.random()*(205-40)
    }
```

第二个火箭

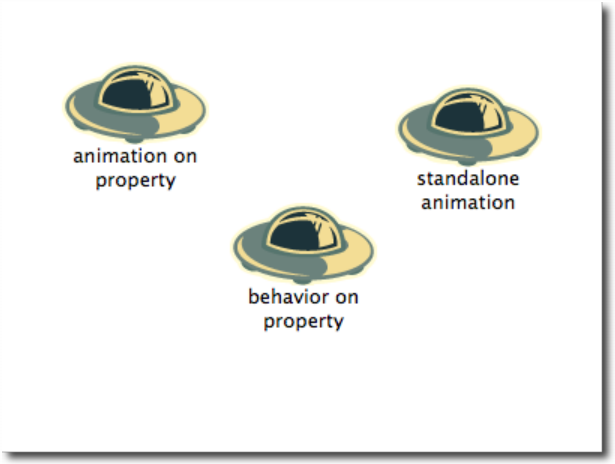
第二个火箭使用了`behavior on`策略。这个行为告诉属性值每时每刻都在变化，通过动画的方式来改变这个值。可以使用`Behavior`元素的`enabled : false`来设置行为失效。当点击火箭时它将会开始运行（y轴坐标逐渐移至40）。其它的点击对于位置的改变没有任何影响。你可以试着使用一个随机值（例如：`40+(Math.random()*(205-40))`）来设置y轴坐标。你将看到动画始终会变化到新位置的时间适应速度匹配定义的4秒到目的地动画持续时间。

```
    ClickableImageV3 {
        id: rocket3
```

```
        x: 264; y: 200
        source: "assets/rocket2.png"
        onClicked: anim.start()
//        onClicked: anim.restart()

        text: "standalone animation"

        NumberAnimation {
            id: anim
            target: rocket3
            properties: "y"
            from: 205
            to: 40
            duration: 4000
        }
    }
```

第三个火箭

第三个火箭使用`standalone animation`策略。这个动画由一个私有的元素定义并且可以写在文档的任何地方。点击火箭调用动画函数`start()`来启动动画。每一个动画都有start()、stop()、resume()、restart()函数。这个动画自身可以比其他类型的动画更早的获取到更多的相关信息。我们只需要定义目标和目标元素的属性需要怎样改变的一个动画。我们需要定义一个`to`属性的值，在这个例子中我们也定义了一个`from`属性的值允许动画可以重复运行。



点击背景能够重新设置所有火箭到它们的初始位置。第一个火箭无法被重置，只有重启程序重新加载元素才能重置它。

注

另一个启动/停止一个动画的方法是绑定一个动画的`running`属性。这种方法非常有用当用户需要输入控制属性时：

```
NumberAnimation {
    ...
    // animation runs when mouse is pressed
    running: area.pressed
}
MouseArea {
    id: area
}
```

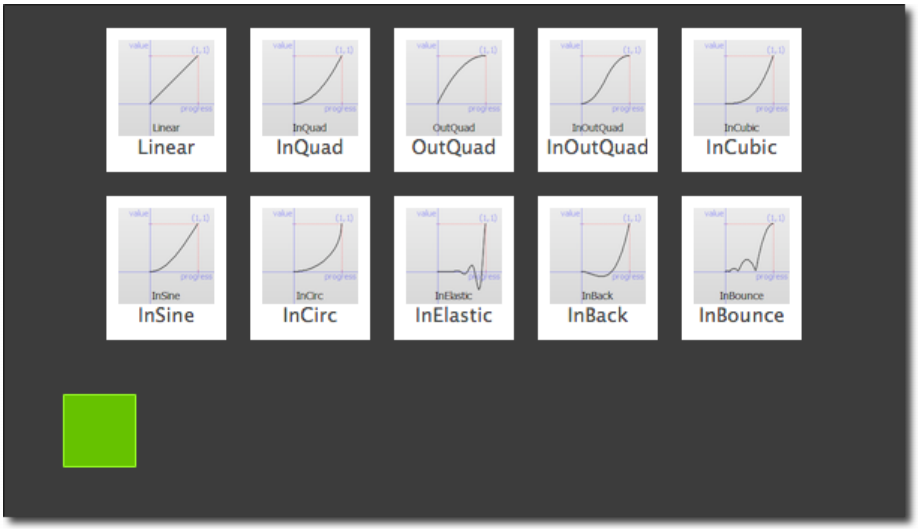
5.1.3 缓冲曲线

属性值的改变可以通过一个动画控制，缓冲曲线属性影响了一个属性值改变的插值算法。我们现在定义的动画都使用了一种线性的插值算法，因为一个动画的默认缓冲类型是`Easing.Linear`。在一个小场景下的x轴与y轴坐标改变可以得到最好的视觉效果。一个线性插值算法将会在动画开始时使用`from`值到动画结束时使用的`to`值绘制一条直线，因此缓冲类型定义了曲线的变化情况。为一个移动的对象精心挑选一个合适的缓冲类型将会使界面更加自然，例如一个页面的滑出，最初页面缓慢的滑出，最后高速滑出，类似翻书一样的效果。

注

不要过度使用动画。设计用户界面动画应尽量小心，动画是让界面更加生动而不是充满整个界面。眼睛对于移动的东西非常敏感，很容易干扰用户的使用。

下面的例子中将会使用不同的缓冲曲线，每一种缓冲曲线都使用了一个可点击图片来展示，点击将会在动画中设置一个新的缓冲类型并且使用这种曲线重新启动动画。



扩展可点击图片V3

我们给图片和文本添加了一个小的外框来增强我们的ClickableImage。添加一个property bool framed: false属性来作为我们的API，基于framed值我们能够设置这个外框是否可见，并且不破坏用户之前的使用。下面是我们做的修改。

```
// ClickableImageV2.qml
// Simple image which can be clicked

import QtQuick 2.0

Item {
    id: root
    width: container.childrenRect.width + 16
    height: container.childrenRect.height + 16
    property alias text: label.text
    property alias source: image.source
    signal clicked

    // M1>>
    // ... add a framed rectangle as container
    property bool framed : false

    Rectangle {
        anchors.fill: parent
        color: "white"
        visible: root.framed
    }
```

这个示例的代码非常简洁紧凑。我们使用了一连串的缓冲曲线名称（property variant easings）并且在一个Repeater中将它们分配给一个ClickableImage。图片源通过一个命名方案来定义，一个叫做“InQuad”的缓冲曲线在“curves/InQuad.png”中有一个对应的图片。如果你点击一个曲线图，这个点击将会分配一个缓冲类型给动画，然后重启动画。动画本身用来设置方块的x坐标在2秒内变化的独立动画。

```
// easingtypes.qml

import QtQuick 2.0

DarkSquare {
    id: root
    width: 600
    height: 340

    // A list of easing types
    property variant easings : [
        "Linear", "InQuad", "OutQuad", "InOutQuad",
        "InCubic", "InSine", "InCirc", "InElastic",
        "InBack", "InBounce" ]


    Grid {
        id: container
        anchors.top: parent.top
        anchors.horizontalCenter: parent.horizontalCenter
        anchors.margins: 16
        height: 200
```

```
        columns: 5
        spacing: 16
        // iterates over the 'easings' list
        Repeater {
            model: easings
            ClickableImageV3 {
                framed: true
                // the current data entry from 'easings' list
                text: modelData
                source: "curves/" + modelData + ".png"
                onClicked: {
                    // set the easing type on the animation
                    anim.easing.type = modelData
                    // restart the animation
                    anim.restart()
                }
            }
        }
    }

    // The square to be animated
    GreenSquare {
        id: square
        x: 40; y: 260
    }

    // The animation to test the easing types
    NumberAnimation {
        id: anim
        target: square
        from: 40; to: root.width - 40 - square.width
        properties: "x"
        duration: 2000
    }
}
```

当运行这个示例时，请注意观察在一个动画过程中速度的变化。一些动画看起来很自然，一些看起来很急躁。

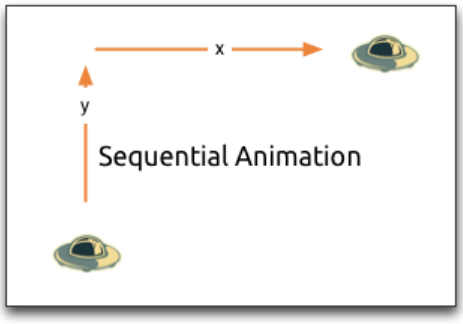
除了`duration`与`easing.type`属性，你也可以对动画进行微调。例如`PropertyAnimation`属性，大多数动画都支持附加的`easing.amplitude`（缓冲振幅），`easing.overshoot`（缓冲溢出），`easing.period`（缓冲周期），这些属性允许对个别缓冲曲线进行微调。不是所有的缓冲曲线都支持这些参数。可以参考`PropertyAnimation`文档中的缓冲列表（easing table）来查看一个缓冲曲线的相关参数。

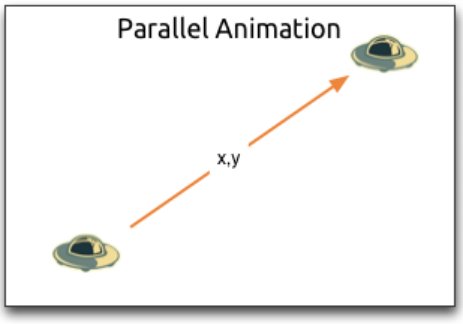
注

对于用户界面正确的动画非常重要。请记住动画是为了让用户界面更加生动而不是为了刺激用户眼睛。

5.1.4 动画分组

通常使用的动画比属性动画复杂。例如想同时运行几个动画并把他们拼接起来，或者在一个个的运行，或者在两个动画之间执行一个脚本。动画分组提供了很好的帮助，作为命名建议叫一组动画。有两种方式来分组：并行与连续。可以使用`SequentialAnimation`（连续动画）和`ParallelAnimation`（并行动画）来实现，它们作为动画的容器包含其它的动画元素。

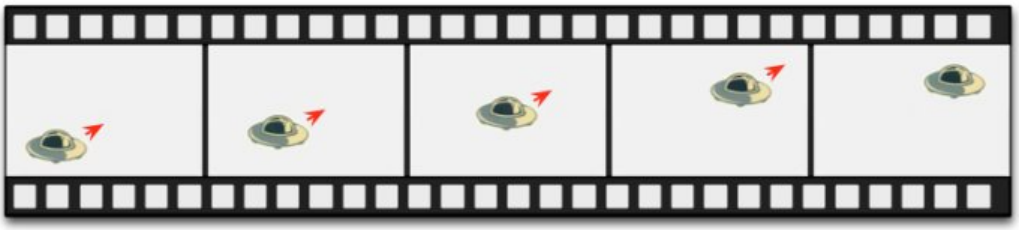




开始时，并行元素的所有子动画都会并行运行，它允许在同一时间使用不同的属性来播放动画。

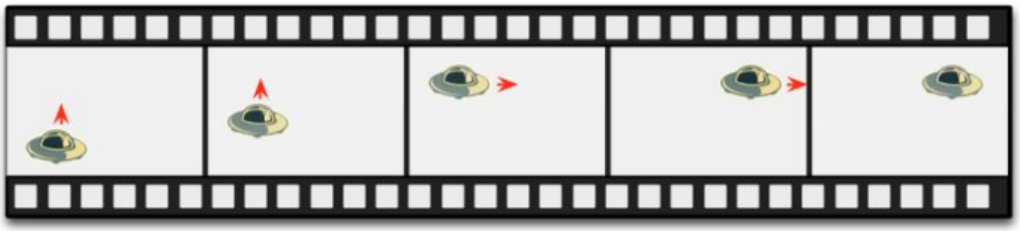
```
// parallelanimation.qml
import QtQuick 2.0

BrightSquare {
    id: root
    width: 300
    height: 200
    property int duration: 3000

    ClickableImageV3 {
```

```
        id: rocket
        x: 20; y: 120
        source: "assets/rocket2.png"
        onClicked: anim.restart()
    }

    ParallelAnimation {
        id: anim
        NumberAnimation {
            target: rocket
            properties: "y"
            to: 20
            duration: root.duration
        }
        NumberAnimation {
            target: rocket
            properties: "x"
            to: 160
            duration: root.duration
        }
    }
}
```

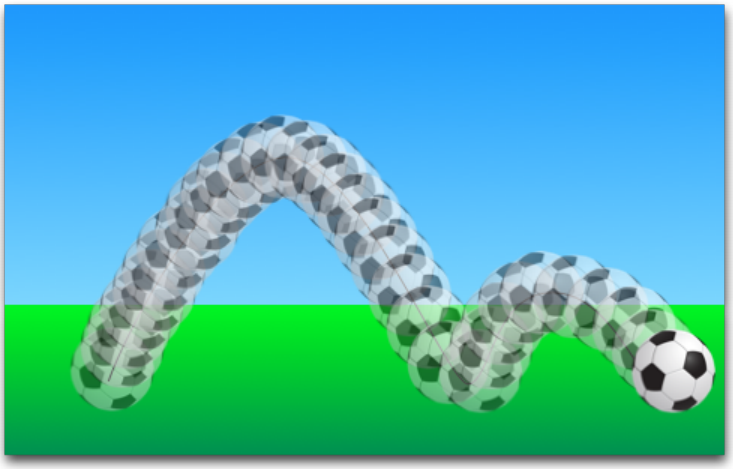


一个连续的动画将会一个一个的运行子动画。

```
// sequentialanimation.qml
import QtQuick 2.0

BrightSquare {
    id: root
    width: 300
    height: 200
    property int duration: 3000

    ClickableImageV3 {
        id: rocket
        x: 20; y: 120
        source: "assets/rocket2.png"
        onClicked: anim.restart()
    }

    SequentialAnimation {
        id: anim
        NumberAnimation {
            target: rocket
            properties: "y"
            to: 20
            // 60% of time to travel up
            duration: root.duration*0.6
        }
        NumberAnimation {
            target: rocket
            properties: "x"
            to: 160
            // 40% of time to travel sideways
            duration: root.duration*0.4
        }
    }
}
```



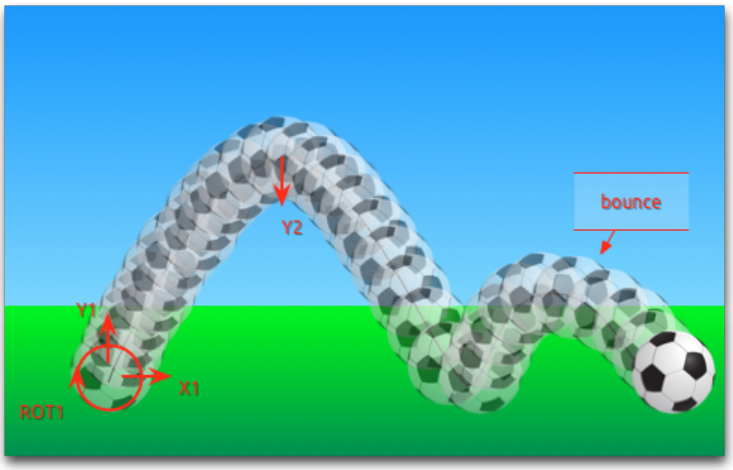
分组动画也可以嵌套，例如一个连续动画可以拥有两个并行的动画作为子动画。我们来看看这个足球的例子。这个动画描述了一个从左向右扔一个球的行为：



要弄明白这个动画我们需要剖析这个目标的运动过程。我们需要记住这个动画是通过属性变化来实现的动画，下面是不同部分的转换：

- 从左向右转换x坐标（`X1`）。
- 从下往上转换y坐标（`Y1`）然后从上往下转换y坐标（`Y2`）。
- 整个动画持续时间过程中旋转超过360度（`ROT1`）。

整个动画的持续时间为3秒。



我们使用一个空基本元素对象（Item）作为根元素，宽为480，高为300。

```
import QtQuick 1.1

Item {
    id: root
    width: 480
    height: 300
    property int duration: 3000

    ...
}
```

我们定义动画的总持续时间作为参考，以便更好的同步各部分动画。

下一步需要添加一个背景，在这个例子中有两个矩形框分别使用了绿色渐变和蓝色渐变填充。

```
Rectangle {
        id: sky
        width: parent.width
        height: 200
        gradient: Gradient {
            GradientStop { position: 0.0; color: "#0080FF" }
```

```
            GradientStop { position: 1.0; color: "#66CCFF" }
        }
    }
    Rectangle {
        id: ground
        anchors.top: sky.bottom
        anchors.bottom: root.bottom
        width: parent.width
        gradient: Gradient {
            GradientStop { position: 0.0; color: "#00FF00" }
            GradientStop { position: 1.0; color: "#00803F" }
        }
    }
```

上部分蓝色区域高度为200像素，下部分区域使用上面的蓝色区域的底作为锚顶，使用根元素的底作为底。

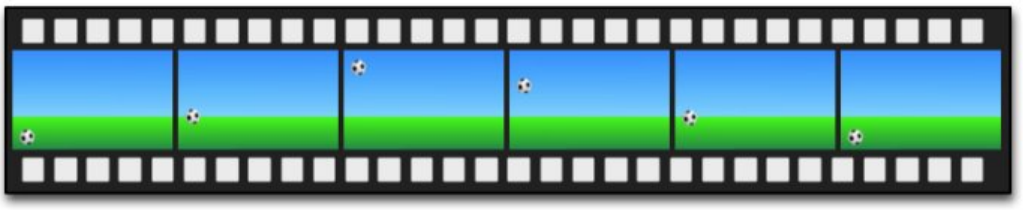
让我们将足球加入到屏幕上，足球是一个图片，位于路径“assets/soccer_ball.png”。首先我们需要将它放置在左下角接近边界处。

```
    Image {
        id: ball
        x: 20; y: 240
        source: "assets/soccer_ball.png"

        MouseArea {
            anchors.fill: parent
            onClicked: {
                ball.x = 20; ball.y = 240
                anim.restart()
            }
        }
    }
```

图片与鼠标区域连接，点击球将会重置球的状态，并且重新开始动画。

首先使用一个连续的动画来播放两次的y轴变换。

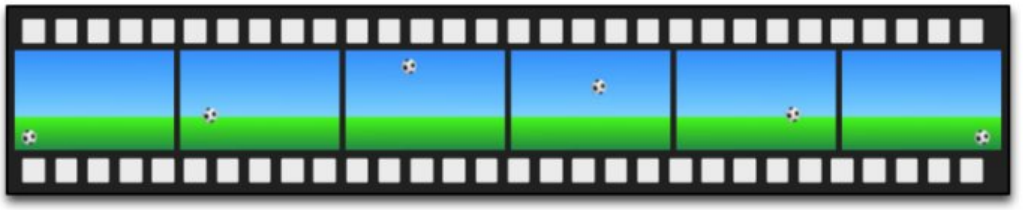
```
SequentialAnimation {
    id: anim
    NumberAnimation {
        target: ball
        properties: "y"
        to: 20
        duration: root.duration * 0.4
    }
    NumberAnimation {
        target: ball
        properties: "y"
        to: 240
        duration: root.duration * 0.6
    }
}
```



在动画总时间的40%的时间里完成上升部分，在动画总时间的60%的时间里完成下降部分，一个动画完成后播放下一个动画。目前还没有使用任何缓冲曲线。缓冲曲线将在后面使用easing curves来添加，现在我们只关心如何使用动画来完成过渡。

现在我们需要添加x轴坐标转换。x轴坐标转换需要与y轴坐标转换同时进行，所以我们需要将y轴坐标转换的连续动画和x轴坐标转换一起压缩进一个平行动画中。

```
ParallelAnimation {
    id: anim
    SequentialAnimation {
        // ... our Y1, Y2 animation
    }
    NumberAnimation { // X1 animation
        target: ball
        properties: "x"
        to: 400
        duration: root.duration
    }
}
```



最后我们想要旋转这个球，我们需要向平行动画中添加一个新动画，使用`RotationAnimation`来实现旋转。

```
ParallelAnimation {
    id: anim
    SequentialAnimation {
        // ... our Y1, Y2 animation
    }
    NumberAnimation { // X1 animation
        // X1 animation
    }
    RotationAnimation {
        target: ball
        properties: "rotation"
        to: 720
        duration: root.duration
    }
}
```

我们已经完成了整个动画，然后需要给动画提供一个正确的缓冲曲线来描述一个移动的球。对于Y1动画使用`Easing.OutCirc`缓冲曲线，它看起来更像是一个圆周运动。Y2使用了`Easing.OutBounce`缓冲曲线，因为在最后球会发生反弹。（试试使用`Easing.InBounce`，你会发现反弹将会立刻开始。）。X1和ROT1动画都使用线性曲线。

下面是这个动画最后的代码，可以作为参考：

```
ParallelAnimation {
        id: anim
        SequentialAnimation {
            NumberAnimation {
                target: ball
                properties: "y"
                to: 20
                duration: root.duration * 0.4
                easing.type: Easing.OutCirc
            }
            NumberAnimation {
                target: ball
                properties: "y"
                to: 240
                duration: root.duration * 0.6
```

```
                easing.type: Easing.OutBounce
            }
        }
        NumberAnimation {
            target: ball
            properties: "x"
            to: 400
            duration: root.duration
        }
        RotationAnimation {
            target: ball
            properties: "rotation"
            to: 720
            duration: root.duration * 1.1
        }
    }
```

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_a6fb6cc90102v1w0.html。

前一篇：Node Security
后一篇：流元素（二）

〈 前一篇
Node Security

后一篇 〉
流元素（二）