Author: dailey.dai@openthinks.com

Animation

Category

Basic Animation

- PropertyAnimation
- NumberAnimation
- ColorAnimation
- RotationAnimation
- Vector3dAnimation
- PathAnimation
- SmoothedAnimation
- SpringAnimation

Group Animation

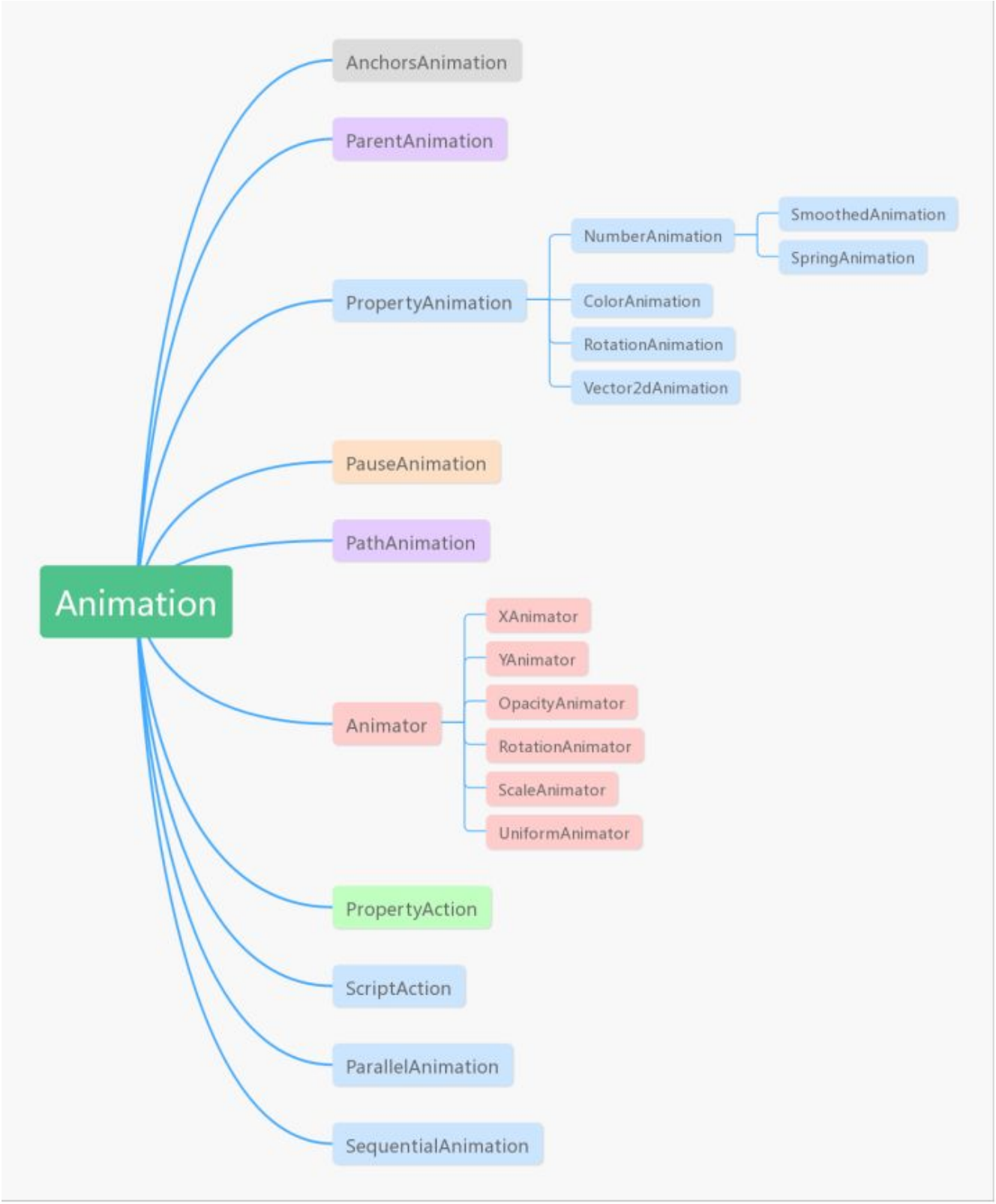
- SequentialAnimation
- ParallelAnimation

Assistant Animation

- State
- Transition

Coordination Animation

- Behavior
- ParentAnimation
- AnchorAnimation
- PauseAnimation
- PropertyAction
- ScriptAction



Basic Animation

`PropertyAnimation`

`PropertyAnimation` provides a way to animate changes to a property's value.

Standalone

```
import QtQuick 2.0
Rectangle {
    id: rootItem
    width: 360
    height: 240
    color: "#eeeeee"
    Rectangle {
        id: rect
        width: 50
        height: 150
        color: "blue"
        anchors.centerIn: parent
        PropertyAnimation{
            id:animation;
            target: rect;
            property: "width"
            to:150;
            duration: 1000;
        }
        MouseArea{
            anchors.fill: parent;
            onClicked:animation
        }
    }
}
```

Slot

```
import QtQuick 2.0
Rectangle {
    id: rootItem
    width: 360
    height: 240
    color: "#eeeeee"
    Rectangle {
        id: rect
        width: 50
        height: 150
        color: "blue"
        anchors.centerIn: parent
        MouseArea{
            anchors.fill: parent;
            onClicked:PropertyAnimation{
                target: rect;
                property: "width"
                to:150;
                duration: 1000;
            }
        }
    }
}
```

Animation on property

```
import QtQuick 2.0
Rectangle {
    id: rootItem
    width: 360
    height: 240
    color: "#eeeeee"
    Rectangle {
        id: rect
        width: 50
        height: 150
        color: "blue"
        anchors.centerIn: parent
        MouseArea{
            anchors.fill: parent;
            id:mouseArea;
        }
        PropertyAnimation on width{
            to:150;
            duration:1000;
            running:mouseArea.pressed;
        }
    }
}
```

`NumberAnimation`

NumberAnimationis a specialized PropertyAnimation that defines an animation to be applied when a numerical value changes.

```
import QtQuick 2.0

Rectangle {
    width:360;
    height:240;
    color:"#EEEEEE";
    id:rootItem;
    Rectangle{
        id:rect;
        color: "blue"
        width: 50;height: 50;x:0;y:95;
        MouseArea{
            id:mosueArea;
            anchors.fill: parent
            onClicked: {
                animationX.start()
                animationRation.running=true;
                animationRadius.start()
            }
```

```
        }
        NumberAnimation {
            id:animationX;
            target: rect
            property: "x";
            to:310;duration: 3000
            easing.type: Easing.OutCubic;
        }
        NumberAnimation {
            id:animationRation;
            target: rect
            property: "rotation"
            to:1080;duration: 3000;
            running: false;
            easing.type: Easing.OutInQuad;
        }
        NumberAnimation on radius {
            id:animationRadius;
            to:25;duration: 3000;
            running: false;
        }
    }
}
```

`ColorAnimation`

`ColorAnimation` is a specialized `PropertyAnimation` that defines an animation to be applied when a color value changes.

```
import QtQuick 2.0

  Rectangle {
      width: 100; height: 100
      color: "red"

      ColorAnimation on color { to: "yellow"; duration: 1000 }
  }
```

`RotationAnimator`

Animator types are different from normal Animation types. When using an Animator, the animation can be run in the render thread and the property value will jump to the end when the animation is complete.

```
Rectangle {
     id: rotatingBox
     width: 50
     height: 50
     color: "lightsteelblue"
     RotationAnimator {
         target: rotatingBox;
         from: 0;
         to: 360;
         duration: 1000
         running: true
     }
 }
```

`PathAnimation`

Animates an item along a path

```
  PathAnimation {
      path: Path {
          //no startX, startY
          PathCurve { x: 100; y: 100}
          PathCurve {}    //last element is empty with no end point specified
      }
  }
```

`SmoothedAnimation`

A `SmoothedAnimation` animates a property's value to a set target value using an ease in/out quad easing curve. When the target value changes, the easing curves used to animate between the old and new target values are smoothly spliced together to create a smooth movement to the new target value that maintains the current velocity.

```
import QtQuick 2.0

Rectangle {
    width: 800; height: 600
    color: "blue"
    Rectangle {
        width: 60; height: 60
        x: rect1.x - 5; y: rect1.y - 5
        color: "green"
        Behavior on x { SmoothedAnimation { velocity: 10 } }
        Behavior on y { SmoothedAnimation { velocity: 10 } }
    }

    Rectangle {
        id: rect1
        width: 50; height: 50
        color: "red"
    }
```

```
    focus: true
    Keys.onRightPressed: rect1.x = rect1.x + 100
    Keys.onLeftPressed: rect1.x = rect1.x - 100
    Keys.onUpPressed: rect1.y = rect1.y - 100
    Keys.onDownPressed: rect1.y = rect1.y + 100
}
```

`SpringAnimation`

`SpringAnimation` mimics the oscillatory behavior of a spring, with the appropriate spring constant to control the acceleration and the damping to control how quickly the effect dies away.

```
import QtQuick 2.0

Item {
    width: 300; height: 300

    Rectangle {
        id: rect
        width: 50; height: 50
        color: "red"

        Behavior on x { SpringAnimation { spring: 2; damping: 0.2 } }
        Behavior on y { SpringAnimation { spring: 2; damping: 0.2 } }
    }

    MouseArea {
        anchors.fill: parent
        onClicked: {
            rect.x = mouse.x - rect.width/2
            rect.y = mouse.y - rect.height/2
        }
    }
}
```

Group Animation

`ParallelAnimation`

Enables animations to be run in parallel

```
import QtQuick 2.0

 Rectangle {
     id: rect
     width: 100; height: 100
     color: "red"

     ParallelAnimation {
         running: true
         NumberAnimation { target: rect; property: "x"; to: 50; duration: 1000 }
         NumberAnimation { target: rect; property: "y"; to: 50; duration: 1000 }
     }
 }
```

`SequentialAnimation`

Allows animations to be run sequentially

```
 import QtQuick 2.0

 Rectangle {
     id: rect
     width: 100; height: 100
     color: "red"

     SequentialAnimation {
         running: true
         NumberAnimation { target: rect; property: "x"; to: 50; duration: 1000 }
         NumberAnimation { target: rect; property: "y"; to: 50; duration: 1000 }
     }
 }
```

Assistant Animation

State

Defines configurations of objects and properties

A *state* is a set of batched changes from the default configuration.

```
 import QtQuick 2.0

 Rectangle {
     id: myRect
     width: 100; height: 100
     color: "black"

     MouseArea {
         id: mouseArea
```

```
        anchors.fill: parent
        onClicked: myRect.state == 'clicked' ? myRect.state = "" : myRect.state =
'clicked';
    }

    states: [
        State {
            name: "clicked"
            PropertyChanges { target: myRect; color: "red" }
        }
    ]
}
```

Transition

Defines animated transitions that occur on state changes.

A Transition defines the animations to be applied when a State change occurs.

```
import QtQuick 2.0

Rectangle {
    id: rect
    width: 100; height: 100
    color: "red"

    MouseArea {
        id: mouseArea
        anchors.fill: parent
    }

    states: State {
        name: "moved"; when: mouseArea.pressed
        PropertyChanges { target: rect; x: 50; y: 50 }
    }

    transitions: Transition {
        NumberAnimation { properties: "x,y"; easing.type: Easing.InOutQuad }
    }
}
```

Coordination Animation

Behavior

A Behavior defines the default animation to be applied whenever a particular property value changes.

```
import QtQuick 2.0

Rectangle {
    id: rect
```

```
        width: 100; height: 100
        color: "red"

        Behavior on width {
            NumberAnimation { duration: 1000 }
        }

        MouseArea {
            anchors.fill: parent
            onClicked: rect.width = 50
        }
    }
```

ParentAnimation

Animates changes in parent values

```
import QtQuick 2.0
Item {
    width: 200; height: 100
    Rectangle {
        id: redRect
        width: 100; height: 100
        color: "red"
    }
    Rectangle {
        id: blueRect
        x: redRect.width
        width: 50; height: 50
        color: "blue"

        states: State {
            name: "reparented"
            ParentChange {
                target: blueRect;
                parent: redRect;
                x: 10; y: 10
            }
        }
        transitions: Transition {
            ParentAnimation {
                NumberAnimation { properties: "x,y"; duration: 1000 }
            }
        }
        MouseArea { anchors.fill: parent; onClicked: blueRect.state = "reparented" }
    }
}
```

AnchorAnimation

Animates changes in anchor values

```
import QtQuick 2.0

  Item {
      id: container
      width: 200; height: 200

      Rectangle {
          id: myRect
          width: 100; height: 100
          color: "red"
      }

      states: State {
          name: "reanchored"
          AnchorChanges { target: myRect; anchors.right: container.right }
      }

      transitions: Transition {
          // smoothly reanchor myRect and move into new position
          AnchorAnimation { duration: 1000 }
      }

      Component.onCompleted: container.state = "reanchored"
  }
```

PauseAnimation

Provides a pause for an animation

```
 SequentialAnimation {
     NumberAnimation { ... duration: 200 }
     PauseAnimation { duration: 100 }
     NumberAnimation { ... duration: 200 }
 }
```

PropertyAction

`PropertyAction` is used to specify an immediate property change during an animation. The property change is not animated.

```
SequentialAnimation {
     PropertyAction { target: img; property: "opacity"; value: .5 }
     NumberAnimation { target: img; property: "width"; to: 300; duration: 1000 }
     PropertyAction { target: img; property: "opacity"; value: 1 }
 }
```

ScriptAction

Defines scripts to be run during an animation

```
SequentialAnimation {
    NumberAnimation {
        // ...
    }
    ScriptAction { script: doSomething(); }
    NumberAnimation {
        // ...
    }
}
```