Gestión dinámica de componentes QML – II

Las animaciones permiten cambiar el UI de forma progresiva y dinámica. Una animación (**Animation**) cambia una propiedad de forma progresiva a lo largo del tiempo. Permite dotar de fluidez al UI. Hay distintos tipos de animaciones para situaciones diferentes. Soporta agrupamiento y anidación.

- Todas las animaciones heredan del componente base **Animation**
- Las propiedades asociadas a **Animation** son: **running, paused, loops, alwaysRunToEnd**
- También se pueden usar de forma imperativa mediante los métodos: **start, stop, pause, resume, restart, complete**

Tipos de Animaciones

Existen distintos tipos de animaciones que tendremos que conocer en detalle:

- Asociadas a propiedades (Property Value Sources)
- Según comportamiento (Behavioral)
- Manejador de señal (Signal handlers)
- Independientes (Standalone)
- Transiciones de estado (State transitions)

Asociadas a propiedades (Property Value Sources)

La animación proporciona los valores a una propiedad. Existen tres propiedades fundamentales asociadas a la animación: **from** para el valor al inicio, **to** para el valor al final de la animación y **duration** para el establecimiento de la duración en milisegundos.

```
Rectangle {
    width: 200
    height: 200
    Text {
        x: 66
        text: "Hello World"
        PropertyAnimation on y {
            from: 0
            to: 93
            duration: 2000
        }
    }
}
```

Según comportamiento (Behavioral)

Define la animación por defecto que se ejecuta cuando cambia una propiedad.

No es necesario la propiedad **from**, ya que recoge el valor actual y el antiguo para realizar la

animación.

En el siguiente ejemplo cuando el usuario pulsa clic en el MouseArea cambiamos el width a 50 (originalmente estaba a 100). Automáticamente se lanza la animación retornando el width nuevamente a 100.

```
import QtQuick 1.0

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

    Behavior on width {
        NumberAnimation { duration: 1000 }
    }

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

Manejador de señal (Signal handlers)

Una animación puede crearse asociada a un manejador de una señal.

```
import QtQuick 1.0

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

    MouseArea {
        anchors.fill: parent
        onClicked: PropertyAnimation { target: rect; properties: "x,y"; to: 50; duration: 1000 }
    }
}
```

Independientes (Standalone)

Las animaciones también se pueden declarar como cualquier otro objeto QML

```
import QtQuick 1.0

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

    PropertyAnimation {
        id: animation
        target: rect
        properties: "x,y"
        duration: 1000
    }

    MouseArea {
        anchors.fill: parent
        onClicked: {
            animation.to = 50;
            animation.running = true;
        }
    }
}
```

Se asocian al objeto destino de la animación, para ello utilizamos la propiedad **target**.

Se utiliza la propiedad **properties** para establecer que propiedades serán animadas.

La propiedad **duration** establece la duración en milisegundos

Necesita ser animada explícitamente mediante **running** o mediante las funciones **start() stop()**

Transiciones de estado (State transitions)

Las transiciones se utilizan para describir las animaciones que se aplicarán cuando se produce un cambio de estado. Para crear una transición, define un objeto de transición y añádelo a la propiedad de un elemento de transición.

```
import QtQuick 1.0

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

    MouseArea {
        anchors.fill: parent
        onClicked: rect.state = "moved"
    }

    states: State {
        name: "moved"
        PropertyChanges { target: rect; x: 50; y: 50 }
    }

    transitions: Transition {
        PropertyAnimation { properties: "x,y"; duration: 1000 }
    }
}
```

1.- El objeto **PropertyChanges** define que cuando el rectángulo se encuentra en el estado “moved”, su posición se debe cambiar a (50, 50).

2.- Cuando alguien pulsa clic sobre el MouseArea se cambia el estado mediante la propiedad state.

3.- Cuando cambia la posición del rectángulo se dispara la transición, y **PropertyAnimation** animará el rectángulo mediante las propiedades x e y.

La animación solo se aplicará en el momento de cambio de estado.

Observe que no se establece ningún valor para **from** y **to**. Para nuestra comodidad, estas propiedades se ajustan automáticamente a los valores de x e y de antes y después del cambio de estado, respectivamente. Aún siendo así, se pueden establecer explícitamente si requiere sobrescribir estos valores.

Property Animation Elements

PropertyAnimation es el elemento de animación más básico para la animación de una propiedad. Puede ser utilizado para animar int, real, color, rect, point, size y `vector3d`. Existen otros tipos de animaciones NumberAnimation, ColorAnimation, RotationAnimation y Vector3DAnimation que heredan de ella:

- **NumberAnimation** proporciona una implementación más eficiente para la animación de las propiedades real y int
- **Vector3DAnimation** hace lo mismo para las propiedades de Vector3D.
- **ColorAnimation** y **RotationAnimation** proporciona atributos específicos para la

animación de los cambios de color y rotación.

```
Rectangle {
    width: 100; height: 100

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

- **RotationAnimation** permite una rotación de la dirección que se determine.

```
Item {
    width: 300; height: 300

    Rectangle {
        width: 100; height: 100; anchors.centerIn: parent
        color: "red"

        RotationAnimation on rotation { to: 90; direction: RotationAnimation.Clockwise }
    }
}
```

Otros a estudiar más adelante:

- **SmoothedAnimation**: ofrece cambios suaves en la animación cuando cambia el valor
- **SpringAnimation**: proporciona una animación “primaveral” con atributos especializados, tales como la masa, amortiguación y epsilon
- **ParentAnimation**: se utiliza para animar un cambio de matriz
- **AnchorAnimation**: utilizado para la animación de anclajes

Agrupamiento de animaciones (Animation grouping)

Las animaciones se pueden agrupar para realizar tareas complejas de animación. Para ello disponemos distintas formas de agrupar las animaciones:

- De forma secuencial mediante **SequentialAnimation**. Es una lista de animaciones que se ejecutan una detrás de otra.
- De forma paralela mediante **ParallelAnimation**. Se ejecutan de forma simultanea.
- Utilizamos **PauseAnimation** para introducir retardos en las animaciones.

  Ejemplo de secuencial con pausas.

```
Rectangle {
    id: rect
    width: 120; height: 200

    Image {
        id: img
        source: "pics/qt.png"
        anchors.horizontalCenter: parent.horizontalCenter
        y: 0

        SequentialAnimation on y {
            loops: Animation.Infinite
            NumberAnimation { to: rect.height - img.height; easing.type: Easing.OutBounce; duration: 2000 }
            PauseAnimation { duration: 1000 }
            NumberAnimation { to: 0; easing.type: Easing.OutQuad; duration: 1000 }
        }
    }
}
```

También podemos realizar agrupaciones de secuenciales con paralelas.

```
Rectangle {
    id: redRect
    width: 100; height: 100
    color: "red"

    MouseArea { id: mouseArea; anchors.fill: parent }

    states: State {
        name: "pressed"; when: mouseArea.pressed
        PropertyChanges { target: redRect; color: "blue"; y: mouseArea.mouseY; width: mouseArea.mouseX }
    }

    transitions: Transition {

        SequentialAnimation {
            ColorAnimation { duration: 200 }
            PauseAnimation { duration: 100 }

            ParallelAnimation {
                NumberAnimation {
                    duration: 500
                    easing.type: Easing.OutBounce
                    targets: redRect
                    properties: "y"
                }

                NumberAnimation {
                    duration: 800
                    easing.type: Easing.InOutQuad
                    targets: redRect
                    properties: "width"
                }
            }
        }
    }
}
```