Introdução ao QML

Sandro S. Andrade

sandroandrade@kde.org

@andradesandro

Objetivos

- Apresentar as principais características e funcionalidades do QML, QtQuick e QtQuickControls2
- Proporcionar vivências práticas sobre como o QML/QtQuick pode ser utilizado para construções de aplicações gráficas modernas

Agenda

- Introdução
- Um primeiro exemplo
- Sintaxe básica do QML
- Atributos de objetos QML
- Property binding

Agenda

- Signals e handlers
- Sistema de tipos do QML
- Positioners e Layouts
- Multimedia
- Integrando QML com C++

Introdução

- O QML é uma linguagem declarativa para especificação e programação de interfaces gráficas de usuário
- O QtQuick é a biblioteca padrão de tipos e funcionalidades principais do QML:
  - Tipos visuais e interativos, animações, models, views, efeitos de partículas, etc

Um Primeiro Exemplo

- Importando o módulo QtQuick e definindo uma hierarquia de objetos QML:

main.qml

```
1. import QtQuick 2.3
2.
3. Rectangle {
4.     width: 200
5.     height: 100
6.     color: "red"
7.
8.     Text {
9.         anchors.centerIn: parent
10.        text: "Hello, World!"
11.    }
12.}
```

Um Primeiro Exemplo

- Usando ApplicationWindow:

main.qml

```
1. import QtQuick 2.3
2. import QtQuick.Controls 1.2
3. import QtQuick.Window 2.2
4. ApplicationWindow {
5.     title: qsTr("Hello World")
6.     width: 640; height: 480
7.     menuBar: MenuBar {
8.         Menu { title: qsTr("File")
9.                MenuItem {
10.                   text: qsTr("&Open")
11.                   onTriggered: console.log("Open action triggered");
12.               }
13.               MenuItem {
14.                   text: qsTr("Exit")
15.                   onTriggered: Qt.quit();
16.               }
17.           }
18.       }
```

Um Primeiro Exemplo

- Usando ApplicationWindow:

main.qml

```
19.     //Content Area
20.
21.     //a button in the middle of the content area
22.     Button {
23.         text: qsTr("Hello World")
24.         anchors.horizontalCenter: parent.horizontalCenter
25.         anchors.verticalCenter: parent.verticalCenter
26.     }
27.}
```

Um Primeiro Exemplo

- Capturando ações do mouse:

main.qml

```
1. Rectangle {
2.     width: 200
3.     height: 100
4.     color: "red"
5.
6.     Text {
7.         anchors.centerIn: parent
8.         text: "Hello, World!"
9.     }
10.
11.    MouseArea {
12.        anchors.fill: parent
13.        onClicked: parent.color = "blue"
14.    }
15.}
```

Um Primeiro Exemplo

- Property bindings

main.qml

```
1. Rectangle {
2.     width: 400
3.     height: 200
4.
5.     Rectangle {
6.         width: parent.width / 2
7.         height: parent.height
8.     }
9.
10.    Rectangle {
11.        width: parent.width / 2
12.        height: parent.height
13.        x: parent.width / 2
14.    }
15.}
```

Um Primeiro Exemplo

- Definindo tipos customizados para reuso

Button.qml

```
1. import QtQuick 2.3
2. Rectangle {
3.     width: 100; height: 100
4.     color: "red"
5.
6.     MouseArea {
7.         anchors.fill: parent
8.         onClicked: console.log("Button clicked!")
9.     }
10.}
```

main.qml

```
1. import QtQuick 2.3
2. Column {
3.     Button { width: 50; height: 50 }
4.     Button { x: 50; width: 100; height: 50; color: "blue" }
5.     Button { width: 50; height: 50; radius: 8 }
6. }
```

Um Primeiro Exemplo

- Usando o WebView

main.qml

```
1. import QtQuick 2.3
2. import QtWebView 1.1
3.
4. WebView {
5.         id: webView
6.         url: "http://www.kde.org"
7.         onLoadingChanged: {
8.             if (loadRequest.errorString)
9.                 console.error(loadRequest.errorString);
10.        }
11.}
```

Sintaxe Básica do QML

- Importando módulos QML e JavaScript

main.qml

```
1. import QtQuick 2.0
2. import QtQuick.LocalStorage 2.0 as Database
3. import "../privateComponents"
4. import "somefile.js" as Script
```

Sintaxe Básica do QML

- Declarando objetos

main.qml

```
1. import QtQuick 2.0
2.
3. Rectangle {
4.     width: 100
5.     height: 100
6.     color: "red"
7. }
8.
9. OU
10.
11.Rectangle { width: 100; height: 100; color: "red" }
```

Sintaxe Básica do QML

- Declarando objetos-filho

main.qml

```
1. import QtQuick 2.0
2.
3. Item {
4.     Rectangle {
5.         width: 200
6.         height: 200
7.         color: "red"
8.
9.         Text {
10.            anchors.centerIn: parent
11.            text: "Hello, QML!"
12.        }
13.    }
14.    Rectangle {
15.    }
16.}
```

Atributos de Objetos QML

- Objetos QML podem ter atributos de diferentes tipos:
  - O atributo *id*
  - Atributos do tipo *property*
  - Atributos do tipo *signal*
  - Atributos do tipo *signal handler*
  - Atributos do tipo método
  - Atributos do tipo attached properties/signal handlers

Atributos de Objetos QML

- Atributo *id*

main.qml

```
1. import QtQuick 2.0
2.
3. Column {
4.     width: 200; height: 200
5.
6.     TextInput { id: myTextInput; text: "Hello World" }
7.
8.     Text { text: myTextInput.text }
9. }
```

Atributos de Objetos QML

- Atributos do tipo *property*

main.qml

```
1. Rectangle {
2.     property color previousColor
3.     property color nextColor
4.     onNextColorChanged: console.log("Next color: " + nextColor.toString())
5. }
```

main.qml

```
6. import QtQuick 2.0
7.
8. Rectangle {
9.     color: "red"
10.    property color nextColor: "blue" // declaration + initialization
11.}
```

Atributos de Objetos QML

- *Property* alias

main.qml

```
1. // Button.qml
2. import QtQuick 2.0
3.
4. Rectangle {
5.     property alias buttonText: textItem.text
6.
7.     width: 100; height: 30; color: "yellow"
8.
9.     Text { id: textItem }
10.}
```

Atributos de Objetos QML

- Atributos do tipo *signalHandler*

main.qml

```
1. import QtQuick 2.0
2.
3. Item {
4.     width: 100; height: 100
5.
6.     MouseArea {
7.         anchors.fill: parent
8.         onClicked: {
9.             console.log("Click!")
10.        }
11.    }
12.}
```

Atributos de Objetos QML

- Definindo atributos do tipo *signal*

SquareButton.qml

```
1. Rectangle {
2.     id: root
3.
4.     signal activated(real xPos, real yPos)
5.     signal deactivated
6.
7.     property int side: 100
8.     width: side; height: side
9.
10.    MouseArea {
11.        anchors.fill: parent
12.        onPressed: root.activated(mouse.x, mouse.y)
13.        onReleased: root.deactivated()
14.    }
15.}
```

Atributos de Objetos QML

- *Signal handlers* de mudança de propriedades

main.qml

```
1. import QtQuick 2.0
2.
3. TextInput {
4.     text: "Change this!"
5.
6.     onTextChanged: console.log("Text has changed to:", text)
7. }
```

Atributos de Objetos QML

- Atributos de tipo método

main.qml

```
1. import QtQuick 2.0
2. Item {
3.     width: 200; height: 200
4.     MouseArea {
5.         anchors.fill: parent
6.         onClicked: label.moveTo(mouse.x, mouse.y)
7.     }
8.     Text {
9.         id: label
10.
11.        function moveTo(newX, newY) {
12.            label.x = newX;
13.            label.y = newY;
14.        }
15.
16.        text: "Move me!"
17.    }
18.}
```

Atributos de Objetos QML

- Definindo atributos do tipo *signal*

main.qml

```
1. SquareButton {
2.     onActivated: console.log("Activated at " + xPos + "," + yPos)
3.     onDeactivated: console.log("Deactivated!")
4. }
```

Property Binding

- Um primeiro exemplo

main.qml

```
1. Rectangle {
2.     width: 200; height: 200
3.
4.     Rectangle {
5.         width: 100
6.         height: parent.height
7.         color: "blue"
8.     }
9. }
```

Property Binding

- Qualquer expressão JavaScript pode ser utilizada

main.qml

```
1. height: parent.height / 2
2.
3. height: Math.min(parent.width, parent.height)
4.
5. height: parent.height > 100 ? parent.height : parent.height/2
6.
7. height: {
8.     if (parent.height > 100)
9.         return parent.height
10.    else
11.        return parent.height / 2
12.}
13.
14.height: someMethodThatReturnsHeight()
```

Property Binding

- Criando *bindings* via JavaScript

main.qml

```
1. import QtQuick 2.0
2.
3. Rectangle {
4.     width: 100
5.     height: width * 2
6.
7.     focus: true
8.     Keys.onSpacePressed: {
9.         height = width * 3  // O binding é quebrado
10.    }
11.}
```

Property Binding

- Criando *bindings* via JavaScript

main.qml

```
1. import QtQuick 2.0
2.
3. Rectangle {
4.     width: 100
5.     height: width * 2
6.
7.     focus: true
8.     Keys.onSpacePressed: {
9.         height = Qt.binding(function() { return width * 3 })
10.    }
11.}
```

Signal e Handlers

- Recebendo *signals* com *signal handlers*

main.qml

```
1. import QtQuick 2.0
2.
3. Rectangle {
4.   id: r
5.   width: 100; height: 100
6.
7.   MouseArea {
8.     anchors.fill: parent
9.       onClicked: {
10.        r.color = Qt.rgba(Math.random(), Math.random(), Math.random(), 1);
11.      }
12.  }
13.}
```

Signal e Handlers

- *Signal handlers* de mudança de propriedades

main.qml

```
1. import QtQuick 2.0
2.
3. Rectangle {
4.     id: rect
5.     width: 100; height: 100
6.
7.     MouseArea {
8.         anchors.fill: parent
9.         onPressedChanged: {
10.            console.log("Mouse area is pressed?", pressed)
11.        }
12.    }
13.}
```

Signal e Handlers

- O tipo Connections

main.qml

```
1. import QtQuick 2.0
2.
3. Rectangle {
4.   id: r
5.   width: 100; height: 100
6.
7.   MouseArea {
8.     id: mouseArea
9.     anchors.fill: parent
10.  }
11.
12.  Connections {
13.    target: mouseArea
14.    onClicked: {
15.      r.color = Qt.rgba(Math.random(), Math.random(), Math.random(), 1);
16.    }
17.  }
18.}
```

Signal e Handlers

- Conectando *signals* a métodos

main.qml

```
1. Rectangle {
2.     id: relay
3.
4.     signal messageReceived(string person, string notice)
5.
6.     Component.onCompleted: {
7.         relay.messageReceived.connect(sendToPost)
8.         relay.messageReceived.connect(sendToTelegraph)
9.         relay.messageReceived("Tom", "Happy Birthday")
10.    }
11.
12.    function sendToPost(person, notice) {
13.        console.log("Sending to post: " + person + ", " + notice)
14.    }
15.    function sendToTelegraph(person, notice) {
16.        console.log("Sending to telegraph: " + person + ", " + notice)
17.    }
18.}
```

Sistema de Tipos do QML

- Os tipos usados na definição de hierarquias de objetos QML podem ser:
  - Disponibilizados nativamente pela linguagem QML
  - Registrados via C++
  - Disponibilizados como documentos QML

Sistema de Tipos do QML

- Tipos disponibilizados nativamente:

bool	Binary true/false value
double	Number with a decimal point, stored in double precision
enumeration	Named enumeration value
int	Whole number, e.g. 0, 10, or -20
list	List of QML objects
real	Number with a decimal point
string	Free form text string
url	Resource locator
var	Generic property type

Sistema de Tipos do QML

- Tipos registrados pelo módulo QtQuick:

date	Date value
point	Value with x and y attributes
rect	Value with x, y, width and height attributes
size	Value with width and height attributes
color	ARGB color value. The type refers to an ARGB color value. It can be specified in a number of ways:
font	Font value with the properties of QFont. The type refers to a font value with the properties of QFont
matrix4x4	A matrix4x4 type is a 4-row and 4-column matrix
quaternion	A quaternion type has scalar, x, y, and z attributes
vector2d	A vector2d type has x and y attributes
vector3d	Value with x, y, and z attributes
vector4d	A vector4d type has x, y, z and w attributes

Sistema de Tipos do QML

- Tipos disponibilizados pelo JavaScript (var):

main.qml

```
1. import QtQuick 2.0
2.
3. Item {
4.     property var theArray: new Array()
5.     property var theDate: new Date()
6.
7.     Component.onCompleted: {
8.         for (var i = 0; i < 10; i++)
9.             theArray.push("Item " + i)
10.        console.log("There are", theArray.length, "items in the array")
11.        console.log("The time is", theDate.toUTCString())
12.    }
13.}
```

Positioners e Layouts

- O posicionamento de objetos no QML pode ser feito de diversas maneiras:
  - Manual (ajustando x e y)
  - Utilizando anchors
  - Utilizando positioners
  - Utilizando layouts

Positioners e Layouts

- Posicionamento manual:

main.qml

```
1. import QtQuick 2.3
2.
3. Item {
4.     width: 100; height: 100
5.
6.     Rectangle {
7.         // Manually positioned at 20,20
8.         x: 20
9.         y: 20
10.        width: 80
11.        height: 80
12.        color: "red"
13.    }
14.}
```

Positioners e Layouts

- Posicionamento com *anchors*:

main.qml

```
1. import QtQuick 2.3
2. Item {
3.     width: 200; height: 200
4.     Rectangle {
5.         Anchors { right: parent.right; top: parent.top; margins: 20 }
6.         width: 80; height: 80
7.         color: "orange"
8.     }
9.     Rectangle {
10.        Anchors {
11.            horizontalCenter: parent.horizontalCenter;
12.            top: parent.top;
13.            topMargin: 20 }
14.        width: 80; height: 80
15.        color: "green"
16.    }
17.}
18.
```

Positioners e Layouts

- Posicionamento com *positioners*:

main.qml

```
1. import QtQuick 2.3
2.
3. Item {
4.     width: 300; height: 100
5.
6.     Row { // The "Row" type lays out its child items in a horizontal line
7.         spacing: 20 // Places 20px of space between items
8.
9.         Rectangle { width: 80; height: 80; color: "red" }
10.        Rectangle { width: 80; height: 80; color: "green" }
11.        Rectangle { width: 80; height: 80; color: "blue" }
12.    }
13.}
```

Positioners e Layouts

- Posicionamento com *layouts*:

main.qml

```
1. GridLayout {
2.         id: gridLayout
3.         rows: 3
4.         flow: GridLayout.TopToBottom
5.         anchors.fill: parent
6.         Label { text: "Line 1" }
7.         Label { text: "Line 2" }
8.         Label { text: "Line 3" }
9.
10.        TextField { }  TextField { }  TextField { }
11.
12.        TextArea {
13.            text: "This widget spans over three rows in the GridLayout.\n"
14.            Layout.rowSpan: 3
15.            Layout.fillHeight: true
16.            Layout.fillWidth: true
17.        }
18.    }
```

Positioners e Layouts

- Posicionamento com *layouts*:
  - Ver Qt Quick Layouts - Basic Example

Multimedia

- Reproduzindo um video

main.qml

```
1. Video {
2.     id: video
3.     width : 800
4.     height : 600
5.     source: "video.avi"
6.
7.     MouseArea {
8.         anchors.fill: parent
9.         onClicked: {
10.            video.play()
11.        }
12.    }
13.
14.    focus: true
15.    Keys.onSpacePressed: video.paused = !video.paused
16.    Keys.onLeftPressed: video.position -= 5000
17.    Keys.onRightPressed: video.position += 5000
18.}
```

Integrando QML com C++

- Porque integrar QML com C++?
  - Para separar código de interface (QML+JS) de código da lógica da aplicação (C++)
  - Para usar funcionalidade C++ a partir de código QML
  - Para acessar objetos QML a partir do código C++
  - Para criar novos tipos de objetos QML a partir do C++

Lab 1

- Implementação acompanhada de um navegador web simples