集成 JavaScript

本文翻译来源于官方网站，未经本人允许，勿转发。持续更新中。。。。。

QML 鼓励使用属性绑定和存在的 QML 元素显示构建 UI。为了允许实现更高级的方法， QML 与重要的 JavaScript 代码紧密的集成在一起。
QML 提供的 JavaScript 的运行环境要比 web browser 提供的运行环境要严格。在 QML 中，不能添加，或者修改， Javascript 全局对象的成员变量。 如果使用没有声明的变量可能会出现异常。在 QML 中，这将会抛出一个异常，因此所有的局部变量应该被显示声明。
除了标准的 JavaScript 属性，QML 的全局对象包含了很多帮助方法，帮助构建 UI 并和 QML 的环境进行交互。

内联 JavaScript

小的 JavaScript 方法可以内联到 QML 的声明中。 这些内联的方法作为方法被添加到 QML 元素中。

```
Item {
    function factorial(a) {
        a = parseInt(a);
        if (a <= 0)
            return 1;
        else
            return a * factorial(a - 1);
    }

    MouseArea {
        anchors.fill: parent
        onClicked: console.log(factorial(10))
    }
}
```


作为 QML 的组件中的根元素方法，内联的功能，可以被外部的组件调用。如果不想要这些方法，可以把它添加到非根元素中，或者更好卸载一个外部的 Javascript 文件中。

单独的 JavaScript 文件

大的 JavaScript 模块应该卸载单独的文件中。使用 import 声明，可以将这文件导入到 QML 文件中，同样的方式，模块也可以被导入。
例如，在上述例子的被内联的方法 factorial 可以一如到一个外部文件中，叫 factorial.js，而且可以像如下方式来访问：

```
import "factorial.js" as MathFunctions
Item {
    MouseArea {
        anchors.fill: parent
        onClicked: console.log(MathFunctions.factorial(10))
    }
}
```

相对和绝对 JavaScript URI 都可以被导入。如果 script 文件是不可以访问，将会出现一个 error。如果需要从网上提取 JavaScript，QML 文档有一个"Loading"的状态，直到脚本被下载完成。导入的 JavaScript 文件总是使用“as”关键自取个别名。这个 JavaScript 的别名必须是 唯一的，因此别名和 JavaScript 文件是一一匹配的。

实现文件后的代码

大多数导入到 QML 文件的 JavaScript 文件是状态，逻辑实现。
当导入 JavaScript 文件的时候，这个默认的行为提供了一个唯一的，孤立的 QML 实例。在同样范围的代码运行 QML 组件实例，和持续的访问多个对象和声明的属性。

没有状态的 JavaScript 库

一些 JavaScript 文件表现的行为更像库-他们提供了一组没有状态的帮助方法， 例如输入和计算输出，但是永远不要直接操作 QML 组件的实例。
对于每一个 QML 组件来说，每一个都有一个唯一的库文件拷贝版，它是比较浪费的。JavaScript 编码者可以指定体术的文件是一个无状态的库，通过编码使用，例子如下：

```
// factorial.js
.pragma library

function factorial(a) {
    a = parseInt(a);
    if (a <= 0)
        return 1;
    else
```

```
        return a * factorial(a - 1);
}
```

在任何 JavaScript 代码执行注释之前，编码的声明必须有。
当但它们是被分享，无状态库文件是不能访问 QML 组件实例对象或者属性，尽管 QML 的只可以作为方法的参数可以被传递。

作为开始程序运行 JavaScript

有时候，很有必要运行一些必要的代码作为程序或者组件实例的开始程序。而且它是暂时的包含开始的脚本作为一个全局的代码在一个外部的脚本文件中，作为 QML 环境，这可能有一些限制没有。例如，一些对象可能没有被创建或者绑定的属性没有被执行。QML JavaScript 限制覆盖了全局代码的一些精确的限制。
QML 组件提供了一个附件的 onCompleted 属性，可以被用来触发执行代码开始部分的方法，在 QML 的环境完全建立之后。例如：

```
Rectangle {
    function startupFunction() {
        // ... startup code
    }

    Component.onCompleted: startupFunction();
}
```


任何在 QML 文件中的元素，包括嵌入的元素和嵌入的 QML 组件实例，可以使用附件的属性。
如果这有超过一个 onCompleted() 方法去处理 startup，它们会以一个没有的定义的顺序串行执行。
同样的，在组件销毁时 Component::onDestruction 附加的属性将会被触发。

QML JavaScript 的限制

QML 执行标准的 JavaScript 代码，将会有如下的限制：

- JavaScript 代码不可以修改全局对象。

在 QML 中，全局对象是不变的－存在的属性不可以被修改或者删除，而且没有新的属性可以被创建。

大多 JavaScript 程序不会相互的修改全局对象。然而，JavaScript 的自动创建没有定义的变量是一个全局对象的显示修改，而且在 QML 中是禁止的。
假如一个变量在范围内不存在的，下面的代码是非法的。

```
// Illegal modification of undeclared variable
a = 1;
for (var ii = 1; ii < 10; ++ii)
    a = a * ii;
console.log("Result: " + a);
```

可以很简单的把他更改成合理的代码。

```
var a = 1;
for (var ii = 1; ii < 10; ++ii)
    a = a * ii;
console.log("Result: " + a);
```

任何尝试去修改全局对象，或者显示或者隐式，都将会引起异常。如果没有被捕获，将会导致警告被输出，包含文件名和代码的行数。

- 全局代码运行在一个小的范围。

  当开始时,如果一个 QML 文件包含了一个外部的 JavaScript 文件使用“global”的代码，他将在一个范围内执行，仅在包含外部的文件和全局对象内。也就是他不会访问 QML 的的对象和属性。
  全局代码进行访问脚本的局部变量是禁止的，以下是一个例子：

```
var colors = [ "red", "blue", "green", "orange", "purple" ];
```

全局代码访问 QML 对象将不会被正确执行。

```
// Invalid global code - the "rootObject" variable is undefined
var initialPosition = { rootObject.x, rootObject.y }
```

这个限制存在与 QML 的环境中，还没有完全的建立。为了在环境建立之后运行代码，请参考 Running JavaScript at Startup.

- 在 QML 中没有定义的当前的值

  在 QML 中没有定义的值。为了引用任何元素需要提供一个 id。例如：

```
Item {
```

```
    width: 200; height: 100
    function mouseAreaClicked(area) {
        console.log("Clicked in area at: " + area.x + ", " + area.y);
    }
    // This will not work because this is undefined
    MouseArea {
        height: 50; width: 200
        onClicked: mouseAreaClicked(this)
    }
    // This will pass area2 to the function
    MouseArea {
        id: area2
        y: 50; height: 50; width: 200
        onClicked: mouseAreaClicked(area2)
    }
}
```