Javascript pour QtQuick

François Delobel

Master TechMed et Setsys, Université Clermont-Auvergne

Plan

Javascript pour QtQuick - version courte!

- Usage en QtQuick: langage dynamique pour animer l'UI.
- Assez répandu, surtout dans le monde du web mais pas seulement
- Utilisé hors des navigateurs (Bureaux graphiques)
- Simple. Assez intuitif.
- Pas très rapide.
- Normalisé (enfin!) $\rightarrow$ ECMAScript.
- En gros, ça ressemble à du C dynamique, et pseudo objet...

Variables et fonctions

Variables

- Inférence de type: une variable se déclare par `var` et `nom` (minuscule en premier, chiffres, _, dollar).
- En QtQuick, obligation de déclarer toutes les variables!

Fonctions

- Mot-clef `function`, nom en minuscules (!)
- Arguments multiples
- Paramètres non prototypés
- Retour de la fonction possible (**`return`**) mais non prototypé
- Appel identique à `C`
- Variables locales possibles.

Exemple de fonction Javascript

```
function(article, prixHT, TVA)
{
  var prixTTC = prixHT * TVA;
  console.log("Achat de " + article);

  return prixTTC;
}
```

Structures de contrôle

- Comme en C: **`if`**, **`while`**, **`for`**, **`switch`**
- Mais **`switch`** possible sur des chaînes.

Comparaisons

- ==: comparaison sur les *valeurs*, avec conversions: `"1" == 1` est vrai.
- ===: comparaison sur les *types* puis *valeurs* si même type. `"1" === 1` est faux.

Objets en Javascript

- Javascript est un langage où tout est dynamique.
- Les objets se définissent avec **new**...
- Mais il n'y a pas de classe!
- On ajoute les attributs à l'objet, à la volée.

```
var vide={}

var lapin=new Object();
lapin.nbOreilles=2;

var monPinpin = {  // JSON
  nbOreilles: 1
  name: "pinpin"
}
```

Fonctions comme argument

- La fonction est un objet de premier ordre
- Il est donc possible de la passer en argument, de la stocker.

```
function forEach(array, f) {
  for(var i = 0; i < array.length; ++i)
    f(array[i]);
}
forEach([1,2,3], console.log);
```

- Attention, `console.log` n'est pas un appel de méthode mais un passage de paramètre.

Lambda

Les lambda sont des fonctions anonymes.

```
var total = 0;
forEach([1,2,3], function(elt) {
  total += elt;
});
```

Mieux: utiliser le `.map` de Array.

```
var total = 0;
var array=[1,2,3]
array.map(function(elt) {
  total += elt;
});
```

Et la méthode fût!

On vient de vous dire, la méthode c'est un objet!

```
var pitbull = {
  manger: function(lapin) {
    console.log("J'aime "+lapin.name);
  }
}
```

Utiliser ses attributs

À l'appel d'une "méthode", **this** désignera l'objet courant.

```
var raoulPitbull = {
  amisLapins: 0
  manger: function(lapin) {
    this.amisLapins += 1
    console.log("J'aime "+lapin.name);
  }
}
```

Construire un constructeur

```
function Pitbull(nbDents) {
  this.nbDents=nbDents
  this.amisLapins=0
  this.manger = function(lapin) {
    this.amisLapins += 1
    console.log("J'aime "+lapin.name);
  }
}

var raoul = new Pitbull(128);
raoul.manger(pinpin);
```

Héritage en Javascript

- Pas de classe, pas d'héritage!
- Voir `prototype`, mais sans moi.
- Si vous avez besoin de ça en QtQuick, vous faites sans doute fausse route...

Utilisation implicite de Javascript dans QML

- Rappel: QML est écrit en pseudo JSON (Javascript Object Notation).
- Utilisation implicite de Javascript dans les valeurs (après `:`).
- `Text { text: "vive le cafe".toLowerCase()}`
- Attention à la ré-évaluation en cascade:

```
Rectangle {
  width: Math.random()
  height: width + 100
}
```

- `height` recalculé quand `width` change.
- `width` pas recalculé car `Math.random()` ne dépend pas d'une propriété.

Charger explicitement un fichier Javascript

```
import "file.js" as MyLib

Item {
  Rectangle {
    width: MyLib.myFunc("120") // myFunc definie dans file.js
  }
}
```

Déclaration de méthode Javascript dans du QML

```
Item {
  Rectangle {
    width: myFunc("120")
    function myFunc(arg) {
      return parseInt(arg)
    }
  }
}
```

En QtQuick, jamais tu ne feras...

- Modifier l'*objet global*. C'est en particulier lui qui héberge les variables globales en Javascript : on se limitera donc aux variables locales (`var`).
- De référence depuis un fichier Javascript externe vers un objet QML. Par contre, on peut passer des paramètres, faire des variables locales... La portée d'un bout de code dans un fichier Javascript est limitée au fichier lui même et à l'objet global.
- De référence à à **`this`** dans le QML (utiliser les `id`), (sauf en cas de fonction liée)
- Le `with` est interdit.
- Il existe un mode *strict* pour évaluer du QML qui refuse l'évaluation de Javascript comme valeur pour des propriétés (mais le reste du code est évalué).

Scripts avec et sans état

- Un script a un état ssi il dépend d'autre chose que des paramètres de fonction
  - En particulier, il ne dépend pas du QML.
- Un script sans état peut être partagé en mémoire.
- Par défaut, un script est considéré comme *sans état*
- Ajoutez `pragma library` au *début* d'un script pour indiquer qu'il est sans état, et donc partageable.

Importer du QML ou du Javascript depuis Javascript

- Pourquoi? Pour accéder au Javascript à l'intérieur du QML

```
.import "computation.js" as Computation
.import QtTest 1.0 as JsQtTest
...
Computation.f();
JsQtTest.z();
....
```

Lier les propriétés à du javascript

Vous le faites déjà sans avoir pensé au javascript derrière:

```
height: parent.height / 2

height: Math.min(parent.width, parent.height)

height: parent.height > 100 ? parent.height : parent.height/2

height: {
    if (parent.height > 100)
        return parent.height
    else
        return parent.height / 2
}

height: someMethodThatReturnsHeight()
```

Property Binding et affectation

L'affectation (=) fait des affectations statiques et pas dynamiques

```
import QtQuick 2.0

Rectangle {
    width: 100
    height: width * 2

    focus: true
    Keys.onSpacePressed: {
        height = width * 3  // Que se passe-t-il si on retaille apres
                            // avoir presse une touche?
    }
}
```

Solution: refaire un binding explicitement

```
import QtQuick 2.0

Rectangle {
    width: 100
    height: width * 2

    focus: true
    Keys.onSpacePressed: {
        height = Qt.binding(function() { return width * 3 })
    }
}
```

Problème de portée des propriété: `this`

`this` permet de représenter l'objet qui appelle la méthode javascript, et pas celui qui la définit!

```
Item {
    width: 500
    height: 500

    Rectangle {
        id: rect
        width: 100
        color: "yellow"
    }

    Component.onCompleted: {
        rect.height = Qt.binding(function() { return this.width * 2 })
       //?? rect.height = Qt.binding(function() { return width * 2 })
    }
}
```