Programmation WEB

Riot.js

Denis Monnerat

monnerat@u-pec.fr

IUT de Fontainebleau

12 juin 2019

Sommaire

Introduction

Riot.js ?

Framework ?

une (micro) bibliothèqe UI comme

- Vue.js
- React.js
- Polymer

⇝ Approche UI par tags personnalisés / WebComponents

(M)VC à l'échelle d'un composant réutilisable. Tout le reste concerne la version 3 `https://v3.riotjs.now.sh/`

Tags personnalisés

Un nouveau Tag <hello-world>

```
<!doctype html>
<html lang="en">
  <head>
    <meta charset="utf-8">
    <link href="/bootstrap/css/bootstrap.css" rel="stylesheet" />
    <title>Hello World</title>
  </head>
  <body>
    <div class="container">
      <hello-world></hello-world>
    </div>
    <!-- include riot.js -->
    <script src="include/riot+compiler.min.js"></script>
    <script src="./tags/hello-world.tag" type="riot/tag"></script>
    <script> riot.mount('*', { title: 'Say hello!'}); </script>
  </body>
</html>
```

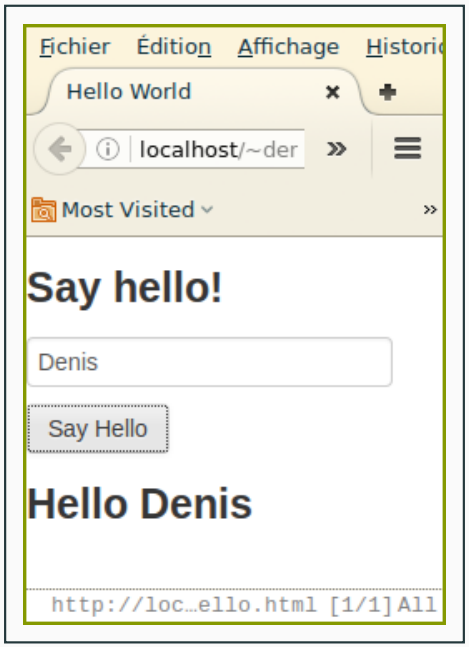
Avec le tag :

```
<hello-world>
<h3>{ opts.title }</h3>

<input type="text" name="nom" Placeholder="votre nom"><br>
<button class="btn" onclick={sayhello}>Say Hello</button>
<h3>{message}</h3>

<script>
 sayhello(){
this.message = "Hello "+this.nom.value;
}
</script>

</hello-world>
```


Fichier Édition Affichage Historic
Hello World
localhost/~der
Most Visited
Say hello!
Denis
Say Hello
Hello Denis
http://loc…ello.html [1/1] All

- ⇝ Properties Bindings.
- ⇝ Loops.
- ⇝ LifeCycle.
- ⇝ Event Handlers.
- ⇝ Injection.
- ⇝ Expressions.
- ⇝ Nested Tags
- ⇝ Conditionnals

Syntaxe

Syntaxe du tag

1. La partie HTML (partie interface).
2. La partie style CSS.
3. La partie logique (fonctionnelle) en JS.

Le composant est monté dans le dom.

```
 <script>
  riot.mount('hello-world', { title: 'Say hello!'});
</script>
```

Bindings, expressions

Bindings et expressions

```
<todo>
<ul>
  <li each={ items } class={ completed: done }>
    <input type="checkbox" checked={ done }> { title }
  </li>
</ul>
<div if={ has_items }>
  This div is inserted when 'has_items' is true.
</div>
<div show={ has_items }>
  This div is shown when 'has_items' is true.
</div>
<!-- partie logique -->
this.items = [
{ title: 'First item', done: true },
{ title: 'Second item' },
{ title: 'Third item' }
]
this.has_items = this.items.length > 0
</todo>
```

La vue (la partie HTML) et le contrôleur (la partie logique en JS) du composant partage les propriétés de l'objet.

Bindings et expressions

La vue html utilise des expressions (code javascript qui fait intervenir des variables et/ou fonctions de la partie fonctionnelle) qui conditonne son état.

Elles sont placées dans le code html entre accolades.

```
<todo>
  <h3>TODO</h3>
  <ul>
    <li each={ item, i in items }>{ item }</li>
  </ul>
  <form onsubmit={ handleSubmit }>
    <input>
    <button>Add #{ items.length + 1 }</button>
  </form>

this.items = []

handleSubmit(e) {
var input = e.target[0]
this.items.push(input.value)
input.value = ''
}
</todo>
```

Conditions

Les conditions permettent d'afficher/cacher des éléments selon une condition.

```
<div if={user.age <20 }>
  {user.nom} est jeune
</div>
```

- `if` retire le noeud du dom, ou l'ajoute.
- `show` et `hide` couvre ou découvre l'élément.

Loops

```
<todo>
  <ul>
    <li each={ items } class={ completed: done }>
      <input type="checkbox" checked={ done }> { title }
    </li>
  </ul>

  this.items = [
    { title: 'First item', done: true },
    { title: 'Second item' },
    { title: 'Third item' }
  ]
</todo>
```

Attention Chaque élément créé a son propre contexte !

Handlers d'événements

Event Handlers

```
<login>
  <form onsubmit={ submit }>
  </form>

  submit(e) {

  }
  // ou
  this.submit=((e)=>{})
</login>
```

- `e.currentTarget`
- `e.target`
- `e.which` key code (keyboard event : keypress, keyup, etc. . .).
- `e.item` élément courant dans une boucle.

Interaction avec le DOM

Les éléments avec l'attribut `ref` sont liés au contexte avec `this.refs`

```
<login>
  <form ref="login" onsubmit={ _submit }>
    <input ref="username">
    <input ref="password">
    <button ref="submit">
  </form>

  // grab above HTML elements
  _submit(e) {
    var form = this.refs.login,
        username = this.refs.username.value,
        password = this.refs.password.value,
        button = this.refs.submit
  }
</login>
```

Cycle

Cycle de vie

Chaque expression a un pointeur sur le noeud DOM correspondant. Une fois le tag monté, les expressions sont mises à jour dans les cas suivant

- callback d'événement appelé (click, submit, change, etc.).
- `this.update()` explicite de l'instance.
- `this.update()` sur tag parent, ou un parent du parent. Les mises à jour se propagent unilatéralement du parent au tag enfant.
- `riot.update()` est appelé, ce qui met à jour toutes les expressions sur la page.

Mise à jour du DOM

Si la valeur d'une expression a changée, le noeud DOM associé est mis à jour en conséquence.

this.update(data)

```
timer>
  <p>Seconds Elapsed: { time }</p>
  <script>
    this.time = opts.start || 0
    tick() {
      this.update({ time: ++this.time })
    }
    var timer = setInterval(this.tick, 1000)

    this.on('unmount', function() {
      clearInterval(timer)
    })
  </script>
</timer>
```

Evénements

Toutes instances de tag sont `observable`. Il y a des événements prédéfinis : `update`,`updated`, `before-(un)mount`, `(un)mount`.

On peut définir ses propres évenements :

```
myObs.trigger("myEvent",data);

myObs.on("myEvent",function(data){
  // le code
});
```

On peut également rendre `observable` un élément ou créer une nouvelle instance

```
riot.observable(el)
```

Utilisation classique : partager un obervable entre tag pour communiquer par événement (diminuer le couplage)

Mixins

Offre la possibilité de partager/étendre des fonctionnalités dans un tag.

```
// Mixin partage
riot.mixin(mixinName, mixinObject)
// Mixin global
riot.mixin(mixinObject)
```

Pour utiliser un mixin partagé dans un tag

```
this.mixin(mixinName)
```

Cas d'utilisation typique : pattern singleton

```
// on fabrique l'objet
riot.mixin('serviceAjax', makeServiceAjax());

// On peut l'utiliser dans n'importe quel tag
this.mixin('serviceAjax');
```

Compilation

Les tags riot doivent être transformés en javascript.

- Compilés à la volée, dans le navigateur (les fichiers chargés sont au format .tag)
- Précompilés (les fichiers chargés sont déjà du js)

Langages supportés : ES6, Coffee, Transcript, Jade, Livescript, etc.

Pour aller plus loin

Il y a beaucoup d'autres possibilités :

- Imbrication des tags.
- Routages.
- Injection de code html dans les tags.
- Mixins.
- etc.

La documentation est très clair, et la courbe d'apprentissage bien plus simple qu'Angular par exemple.

Riot.js (version 3)

`https://v3.riotjs.now.sh`

Riot.js = custom tags + events + router

- custom tags = modularité + ré-utilisation
- events = faible couplage
- router = navigation

Simplicité ! ! !