R4.01-R4.A.10

Programmation déclarative (Riot.js)

monnerat@u-pec.fr

IUT de Fontainebleau

Sommaire

Introduction

Principes et syntaxe

Expressions

Contrôle du dom

Handlers d'événements

Divers

Introduction

Riot.js ?

Framework ? Une (micro) bibliothèqe UI

comme

- Vue.js
- React.js
- solid.js
- Polymer
- etc.

⇝ Approche UI par tags personnalisés / WebComponents

⇝ Délégation de la Gestion du dom

(M)VC à l'échelle d'un composant réutilisable. Tout le reste concerne la dernière version (10) `https://riot.js.org/`

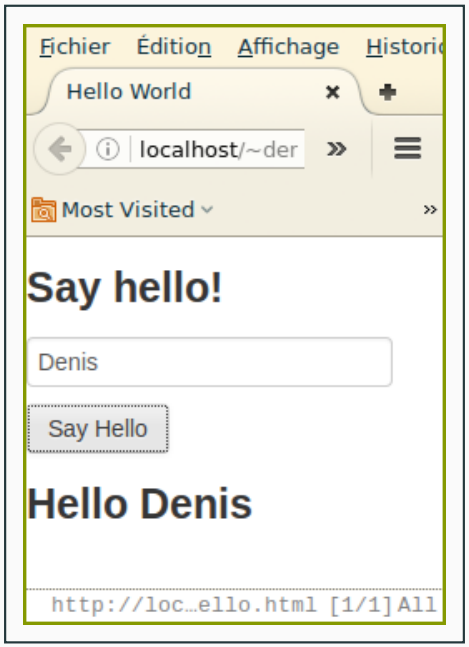
Tags personnalisés

Un nouveau Tag <hello-world>

```
<!doctype html>
<html lang="en">
  <head>
    <meta charset="utf-8">
    <title>Hello World</title>
    <script src="hello-world.riot" type="riot"></script>
    <script src="https://cdn.jsdelivr.net/npm/riot@x.x.x/riot+compiler.min.js">
    </script>
  </head>
  <body>
    <hello-world title="Say hello!"></hello-world>
  </body>
  <script>
    riot.compile().then(()=>{
      console.log(riot.mount('hello-world'));
    });
  </script>
</html>
```

Avec le tag :

```
<hello-world>
  <h3>{props.title}</h3>
  <form onsubmit={_sayhello}>
    <input type="text" name="nom" Placeholder="votre nom">
    <button type="submit">Enter</button>
  </form>
  <h4 if={state.message}> Hello  {state.message}</h4>
  <script>
    export default{
      state:{
        message:""
      },
      _sayhello(e){
        e.preventDefault();
        this.update(
          {"message":this.$('input').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, state
- ⇝ Expressions js
- ⇝ Loops
- ⇝ Conditionnals
- ⇝ LifeCycle.
- ⇝ Event Handlers.
- ⇝ Nested Tags
- ⇝ Etc.

Principes et syntaxe

Syntaxe du tag

1. La partie HTML (partie interface, vue).
2. La partie style CSS.
3. La partie logique (fonctionnelle) en JS (syntaxe ES6 `export default`)

Le composant est monté dans le dom.

```
<script type="module">
// import the component javascript output
// generated via @riotjs/compiler
import MyComponent from './my-component.js'

// register the riot component
riot.register('my-component', MyComponent)

riot.mount('my-component')
</script>
```

- Chaque fichier .riot (ou html) posséde sa logique.
- D'abord la partie HTML (vue), puis la logique (contrôleur) est imbriquée dans un tag `<script>`.
- Les expressions (templates) sont du javascript.
- `this` est optionnel.
- Les quotes sont optionnelles.
- Un tag peut avoir des attributs html
  `<my-component onclick={ click } class={ props.class }>`

Expressions

Expressions

Javascript dans l'html

```
<h3 id="{ attribute_expression }"> <!-- quotes are optionnal -->
  { nested_expression }
</h3>
```

```
{ title || 'Untitled' }
{ results ? 'ready' : 'loading' }
{ new Date() }
{ message.length > 140 && 'Message is too long' }
{ Math.round(rating) }
```

props et state

`props` (Propriétés)

- Chaque composant peut recevoir des "arguments".
- Cela peut être absolument ce qu'on veut, accessible dans le composant à travers l'objet `props`.
- `props` ne peut être changé qu'en dehors du composant lui-même.

`state`

- Chaque composant peut utilisé l'objet `this.state` pour stocker et modifier les données dynamiques représentant son état.
- Ces données peuvent être modifié directement, ou en appelant la méthode `this.update()`

Cycle de vie d'un composant

Création :

- l'objet component est créé
- exécution de la logique
- les expressions sont calculées
- le composant DOM est monté sur le dom.

Vie : les expressions sont mises à jour (et le dom avec) :

- quand `this.update()` est appelé par l'instance.
- quand `this.update()` est appelé par un composant ascendant.

Remarque : on peut enregistrer pour un composant des callbacks (avant/après) pour mount, update, unmount.

```
<timer>
  <p>Seconds Elapsed: { state.time }</p>
  <script>
    export default {
      tick() {
        this.update({ time: ++this.state.time })
      },
      onBeforeMount(props) {
        // create the component initial state
        this.state = {
          time: props.start || 0
        }
        this.timer = setInterval(this.tick, 1000)
      },
      onUnmounted() {
        clearInterval(this.timer)
      }
    }
  </script>
</timer>
```

Contrôle du dom

Contrôle du dom

Conditions

Conditions

Les conditions permettent de monter/démonter des noeuds dom ou des composants suivant une condition.

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

`if` retire le noeud du dom, ou l'ajoute suivant la valeur de la condition.

```
<template if={isReady}>
  <header></header>
  <main></main>
  <footer></footer>
</template>
```

`template` n'est pas un noeud, mais un wrapper.

Contrôle du dom

Boucle

Loops

```
<my-component>
  <ul>
     <li each={item in items}
     class={item.done?'completed':''}>
     <input type="checkbox" checked={item.done}> {item.title}
   </li>
  </ul>
  <script>
    export default {
      items: [{ title: 'First item', done: true },
             { title: 'Second item' },
             { title: 'Third item' } ]
    }
  </script>
</my-component>
```

On peut utiliser le wrapper `template` également pour `each`.

`key` attribute in loop

L'attribut `key` "aide" riot a identifié quels items ont changé, ont été ajoutés ou supprimés. L'intérêt est "minimiser" les interactions avec le dom.

```
<loop>
  <ul>
    <li each={user in users} key={user.id}>{user.name}</li>
  </ul>
  <script>
    export default {
      users: [
        { name: 'Gian', id: 0 },
        { name: 'Dan', id: 1 },
        { name: 'Teo', id: 2 }
      ]
    }
  </script>
</loop>
```

Handlers d'événements

Event Handlers

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

  <script>
    export default {
      // this method is called when above form is submitted
      submit(e) {
        e.preventDefault()
      }
    }
  </script>
</login>
```

```
<form onsubmit="{ condition ? method_a : method_b }">
```

Event Handlers

Fonction fléchée

```
<div each="{item in items}" onclick="{(e)=>console.log(item)}">
  {item.title}
</div>
```

Fonction passée en argument avec `props`

```
<button onclick="{ props.myfunc }">Reset</button>
```

Divers

Intéraction avec le DOM

Deux helpers `this.$` et `this.$$`

```
<my-component>
  <h1>My todo list</h1>
  <ul>
    <li>Learn Riot.js</li>
    <li>Build something cool</li>
  </ul>
  <script>
    export default {
      onMounted() {
        const title = this.$('h1') // single element
        const items = this.$$('li') // multiple elements
      }
    }
  </script>
</my-component>
```

Mixins

```
riot.compile().then(()=>{
  riot.install(function(component){
    if (component.name == "app")
      component.sa = makeServiceAjax();
  })
  riot.mount('app');
});
```

Ajoute un service Ajax au composant app.

Compilation

Les tags riot doivent être transformés en javascript.

- Compilés à la volée, dans le navigateur (les fichiers chargés sont au format .riot)
- 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.
- etc.

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

Riot.js (version 10)

`https://riot.js.org`