

WIM - M4103C

Riot.js : framework js

monnerat@u-pec.fr 

IUT de Fontainebleau

Introduction

Principes et syntaxe

Expressions

Contrôle du dom

Handlers d'événements

Divers

Introduction



Framework ? Une (micro) bibliothèque UI comme

- Vue.js
- React.js
- Polymer

⇒ Approche UI par tags personnalisés / WebComponents

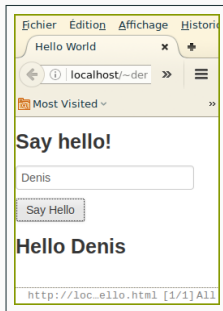
(M)VC à l'échelle d'un composant réutilisable. Tout le reste concerne la dernière version (5) <https://riot.js.org/> 

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@5.3.0/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 onclick={_sayhello}>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>
```



- ~> 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/
import MyComponent from './my-component.js'

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

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

- chaque fichier .riot possède sa logique.
- d'abord la partie HTML, puis la logique 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

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 (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 utiliser l'objet `this.state` pour stocker et modifier les données dynamiques représentant son état.
- Ces données peuvent être modifiées 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' : null }">
      <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`.

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 }">
```

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>
```

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

Ajoute un service Ajax au composant app.

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.

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 5)

<https://riot.js.org> 