ReactJS
Maîtriser le framework

Sommaire

- Présentation de ReactJS
  - Positionnement de ReactJS
  - Virtual DOM avec ReactJS
  - Mise en place des outils de développement
  - Tour d'horizon des outils de développement et d'intégration actuelle
  - Création d'une application React avec le script "create-react-app"
- Composants ReactJS
  - Création d'un composant ReactJS
  - Amélioration des fonctionnalités du composant développé
  - Etats d'un composant et cycle de vie
  - Gestion de l'état d'un composant
  - Propriétés d'un composant
  - Présentation de JSX et ES2015, que choisir ?
  - Présentation approfondie du Virtual DOM

Présentation

ReactJS (aussi appelé React.js ou React) est une bibliothèque JavaScript libre développée par Facebook depuis 2013.

Le but principal de cette bibliothèque est de faciliter la création d'application web monopage (SPA), via la création de composants dépendant d'un état et générant une page (ou portion) HTML à chaque changement d'état.

En 2015 React Native fait son apparition. Ce framework est basé sur React et permet de créer toujours en Javascript des applications multi-plateformes Android et iOS.

Présentation

ReactJS est une bibliothèque qui ne gère que l'interface de l'application, considéré comme la vue dans le modèle MVC.

Elle peut ainsi être utilisée avec une autre bibliothèque ou un framework MVC comme AngularJS.

La bibliothèque se démarque de ses concurrents par sa flexibilité et ses performances, en travaillant avec un DOM virtuel et en ne mettant à jour le rendu dans le navigateur qu'en cas de nécessité

Présentation

La bibliothèque est utilisée par:

- Netflix (depuis le 25 octobre 2017, une migration de la partie client vers du JavaScript pur a permis d'augmenter les performances de 50%),
- Yahoo,
- Airbnb,
- Sony,
- Atlassian
- ainsi que par les équipes de Facebook, pratiquant l'auto-équipement sur le réseau social éponyme, Instagram ou encore WhatsApp.

Présentation

À la fin de 2015, WordPress.com annonce Gutenberg, une interface pour les éditeurs de sites WordPress, développée en JavaScript avec Node.js et ReactJS.

Il est d'ailleurs possible d'utiliser WordPress en mode « headless », c'est-à-dire en séparant le back-office pour la création d'articles, pages, etc. et en développant un front-office qui fait appel aux API de WP avec ReactJS.

Virtual DOM

Le DOM virtuel (VDOM) est un concept de programmation dans lequel une représentation idéale, ou « virtuelle », d'une interface utilisateur (UI) est conservée en mémoire et synchronisée avec le DOM « réel » par une bibliothèque telle que ReactDOM.

Ce processus s'appelle réconciliation.

Virtual DOM

Cette approche rend possible l'API déclarative de ReactJS : vous indiquez à ReactJS dans quel état vous souhaitez que l'UI se trouve, et il s'assure que le DOM corresponde à cet état.

Ça permet de faire abstraction de la manipulation des attributs, de la gestion des événements et de la mise à jour manuelle du DOM que vous auriez normalement dû faire vous-mêmes pour créer votre application.

Outils de développement

Pour écrire du code ReactJS, il suffit d'un éditeur de code. Le plus utilisé actuellement est Visual Studio Code (https://code.visualstudio.com/).

On peut lui ajouter quelques extensions:

- **ESLint** : ESLint analyse le code pour trouver rapidement des soucis.
- **Prettier** : Formatage du code.
- **Auto Rename Tag**: renommage automatique des paires de balises.

On peut aussi ajouter des extensions à la console du navigateur (React Developer Tools).

Premiers pas avec ReactJS

```
<!DOCTYPE html>
<html lang="fr">
<head>
        <meta charset="UTF-8">
        <meta name="viewport" content="width=device-width, initial-scale=1.0">
        <title>React Découverte</title>
        <script src="https://unpkg.com/react@16/umd/react.development.js" crossorigin></script>
        <script src="https://unpkg.com/react-dom@16/umd/react-dom.development.js" crossorigin></script>
  <!-- interprétation du code JSX dans le JS -->
        <script src="https://unpkg.com/babel-standalone@6/babel.min.js"></script>
</head>
<body>
<div id="app"></div>
</body>
</html>
```

Premiers pas avec ReactJS

```
var liste = React.createElement("ul", null ,
"Hello React !");

console.log(p);

ReactDOM.render(p,
document.getElementById("app"));
```

Premiers pas avec ReactJS

```
var liste = React.createElement("p", null ,
  React.createElement("li", null, "Element 1"),
  React.createElement("li", null, "Element 2"),
  React.createElement("li", null, "Element 3"),
  React.createElement("li", null, "Element 4"),
);
console.log(liste);
ReactDOM.render(liste,
document.getElementById("app"));
```

ES2015 VS JSX

```
const element = <h1>Hello React !</h1>;
```

Cette drôle de syntaxe n’est ni une chaîne de caractères ni du HTML.

Ça s’appelle du JSX, et c’est une extension syntaxique de JavaScript. Il est recommandé de l’utiliser avec React afin de décrire à quoi devrait ressembler l’interface utilisateur (UI).

JSX vous fait sûrement penser à un langage de balisage, mais il recèle toute la puissance de JavaScript.

create-react-app

create-react-app est le moyen officiel de créer des applications ReactJS « one page ».

Pour l'utiliser, il faut auparavant installer nodeJS (http://nodejs.org).

```
npx create-react-app my-app
cd my-app
npm start
```

Création d'un composant

Un composant se crée dans une classe JS qui hérite de React.Component.

```
import React from "react";
class HelloMessage extends React.Component {
  render() {
    return <div>Hello React !</div>;
  }
}
export default HelloMessage;
```

Amélioration d’un composant

On peut améliorer un composant avec :

- Un constructeur afin de définir des variables internes du composant.
- Un state pour gérer l’état de certaines variables du composant.
- Des propriétés.
- Des méthodes pour répondre aux actions sur le composant.

Cycle de vie d'un composant

Lors de la **création** d'un composant, les méthodes suivantes sont appelées, une seule fois (sauf render) et dans l'ordre:

- constructor(props)
- render(): appelé lors de l'affichage de la page, et lors de la mise à jour par this.setState()
- componentDidMount(): une fois le composant affiché. Le DOM a été mis à jour.

Cycle de vie d'un composant

Lors de la **mise à jour** d'un composant (setState ou deuxième render), les méthodes suivantes sont appelées dans l'ordre:

- componentWillReceiveProps(nextProps): lorsque le composant est de nouveau utilisé, avec de nouvelles propriétés ou non, cette méthode est appelée. nextProps indique les nouvelles propriétés.
- shouldComponentUpdate(nextProps, nextState): si cette méthode retourne false, les méthodes qui suivent ne seront pas exécutées. Permet de comparer les anciennes et les nouvelles propriétés pour décider de retourner false si besoin.
- componentWillUpdate(nextProps, nextState): est appelée avant render() et sert a effectuer un dernier traitement avant celui-ci.
- render(): affiche de nouveau le composant.
- componentDidUpdate(prevProps, prevState): appelée une fois le composant mis à jour. Permet de comparer les anciennes propriétés avec les nouvelles.

Cycle de vie d’un composant

La **suppression** d’un composant se fait via la méthode ReactDOM.unmountComponentAtNode(élément DOM). Élément DOM représente l’élément dans lequel le composant a été inséré avec ReactDOM.render().

La méthode componentWillUnmount() va être appelée avant la suppression pour permettre de finaliser la destruction (par exemple arrêter les timers).

Etat d’un composant

State est un objet associé à un composant React qui peut être créé dans le constructeur d’une classe dérivant React.Component.

State permet d’indiquer l’état du composant (avec des propriétés que l’on définit en interne dans la classe), sachant que si cet état est modifié, le composant est réaffiché. C’est la seule façon de provoquer l’affichage du composant (en dehors de sa création).

Pour modifier l’affichage d’un composant, il faudra modifier l’objet State. Si un composant se réaffiche, tous les composants internes à celui-ci se réaffichent aussi (this.setState()).

Etat d’un composant

```
class Alarme extends React.Component{
  constructor(props) {
    super(props);
    this.state = {min : 1, sec : 0}; //créer l'objet dans le composant
    setInterval(() => {
      this.state = this.decrTime(this.state);
      console.log(this.state);
    }, 1000);
  }
  decrTime({min, sec}){ … }
  render() {
    return <div>01:00</div>
  }
}
ReactDOM.render(<Alarme/>, document.getElementById("app"));
```

Propriétés d’un composant

Le composant parent peut envoyer des informations au composant enfant au travers de propriétés (comme les attributs HTML).

```
<ListeElems elems={elems} color="blue"/>
```

Le composant enfant devra les récupérer dans son constructeur et pourra les utiliser ensuite.

Propriétés d'un composant

```
class Element extends React.Component{
  constructor(props){
    super(props);
  }
  render(){
    return <li style={{color:this.props.color}}>
           {this.props.elem}
           </li>
  }
}
```

Virtual DOM

Puisque le « DOM virtuel » est plus un modèle qu'une technologie spécifique, on l'emploie parfois pour désigner différentes choses.

Dans le monde React, le terme « DOM virtuel » est généralement associé aux éléments React, car il s'agit des objets représentant l'interface utilisateur.

Virtual DOM

Cependant, React utilise également des objets internes appelés « fibres » (fibers, NdT) pour conserver des informations supplémentaires sur l'arbre des composants.

Ils peuvent également être considérés comme faisant partie de l'implémentation du « DOM virtuel » dans React.

Fiber est le nouveau moteur de réconciliation de React 16. Son objectif principal est de permettre le rendu incrémentiel du DOM virtuel.

Virtual DOM

Est-ce que le Shadow DOM est identique au DOM virtuel ?

Non, ils sont différents.

Le Shadow DOM est une technologie du navigateur conçue principalement pour limiter la portée des variables et du CSS dans les Web Components.

Le DOM virtuel est un concept implémenté par les bibliothèques en JavaScript en plus des API des navigateurs.