Suite

Chaoua Najiba

05.02.2022

Plan

- Le State en React js?
- Le Props en React js
- Le cycle de vide d'un composant

Le State en React js

Le State est un objet React intégré qui est utilisé pour contenir des données ou des informations sur le composant. L'état d'un composant peut changer au fil du temps ; chaque fois qu'il change, le composant est restitué. Le changement d'état (state) peut se produire en réponse à une action de l'utilisateur ou à des événements générés par le système et ces changements déterminent le comportement du composant et son rendu.

Le composant Greetings ci dessous comporte le state name :

```
import React, { Component } from "react";

class Greetings extends React.Component {
  state = {
    name: "World",
  };

  updateName() {
    this.setState({ name: "Simplilearn" });
  }

  render() {
    return <div>{this.state.name}</div>;
  }
}
```

- Le state peut être modifié en fonction de l'action de l'utilisateur ou des modifications du réseau

- Chaque fois que l'état d'un objet change(le state), React restitue le composant au navigateur ce dernier ,est initialisé dans le constructeur.
- Le state peut stocker plusieurs propriétés .
- this.setState() est utilisé pour changer la valeur du State.
- La fonction setState() effectue une fusion superficielle entre le nouvel état (state)et l'état précédent(state précédente)

- Le state peut être mis à jour en réponse aux gestionnaires d'événements, aux réponses du serveur ou aux changements des props. Ceci est fait en utilisant la méthode setState().
- La méthode setState() met en file d'attente toutes les mises à jour apportées à l'état du composant et demande à React de restituer le composant et ses enfants avec l'état mis à jour.
- On utilise la méthode setState() pour modifier le state, car cela garantira que le composant sait qu'il a été mis à jour et appelle la méthode render().

Dans le exemple mentionné ce dessus ,le state name est changé dans la fonction updateName via la méthode setState .

Props

- Les props signifient "Propriétés". Ce sont des composants en lecture seule. C'est un objet qui stocke la valeur des attributs d'une balise et fonctionne de manière similaire aux attributs HTML. Il permet de transmettre des données d'un composant à d'autres composants. Il est similaire aux arguments de fonction.
- Les props sont immuables, nous ne pouvons donc pas les modifier depuis l'intérieur du composant. Les props sont disponibles dans le composant en tant que this.props et peuvent être utilisés pour restituer des données dynamiques dans notre méthode de render.

Dans l'exemple ce dessous ,"title" est un props du Composant **Header**.Le Composant Header est un composant fils du Composant parent App.

```
import Header from "./Header";      You,

function App() {
  return (
    <div className="App">
      <Header title="React js course" />
    </div>
  );
}

export default App;
```

Le Composant **Header** peut exploiter le props via this.props + nom du propriété.

```
class Header extends React.Component {
  render() {
    return (
      <div>
        <h1>{this.props.title}</h1>
      </div>
    );
  }
}
export default Header;
```

Le cycle de vie d'un composant

Le cycle de vie d'un composant regroupe les étapes de la vie d'un composant. Le moment où le composant va être inséré dans le DOM, le **mounting** , et le moment où le composant va être retiré du DOM, l'**unmounting**. Quand un composant est monté, il va alors pouvoir être mis à jour ou "re-rendered", quand son state est modifié via la méthode **this.setState**.

Les étapes du cycle de vie d'un composant :

Les méthodes liées au mounting:

Deux méthodes sont prévues pour pouvoir agir avant et immédiatement après que le composant soit monté.

On va utiliser pour ça deux méthodes appelées componentWillMount() et componentDidMount().

```
class MonComposant extends React.Component {
  componentWillMount() {
    // Ce bloc de code sera exécuté juste avant que le composant soit monté
  }

  componentDidMount() {
    // Ce bloc de code sera exécuté dès que le composant aura été monté
  }
}
```

Les méthodes liées à l'unmounting

Une méthode va être mise à notre disposition pour réaliser des actions avant l'unmounting de notre composant, la méthode componentWillUnmount(). Cette méthode peut être utile pour détruire un interval javascript, par exemple:

```
componentWillUnmount() {
  // Ce code sera exécuté juste avant que le composant soit démonté et retiré du DOM
  clearInterval(monInterval);
}
```

Les causes du “re-rendering” du composant:
Quand un composant est monté et qu'il est affiché dans le DOM, il va être mis à jour dans 3 cas de figure:

- si son **state est modifié** via ‘this.setState‘
- si ses **props sont modifiées**
- si son composant parent est mis à jour.

Les méthodes liées à la mise à jour du composant sont :

- componentDidUpdate(): nous permet de réaliser des actions **après** que le composant ait été mis à jour
- componentWillUpdate()‘: nous permet de réaliser des actions **juste avant** que le composant soit mis à jour
- shouldComponentUpdate(): cette méthode va nous permettre de définir le comportement du composant et plus précisément dans quels cas de figure il doit être mis à jour (très utile pour éviter des re-rendering inutiles)