Chapitre 09 : Cycle de vie d'un composant

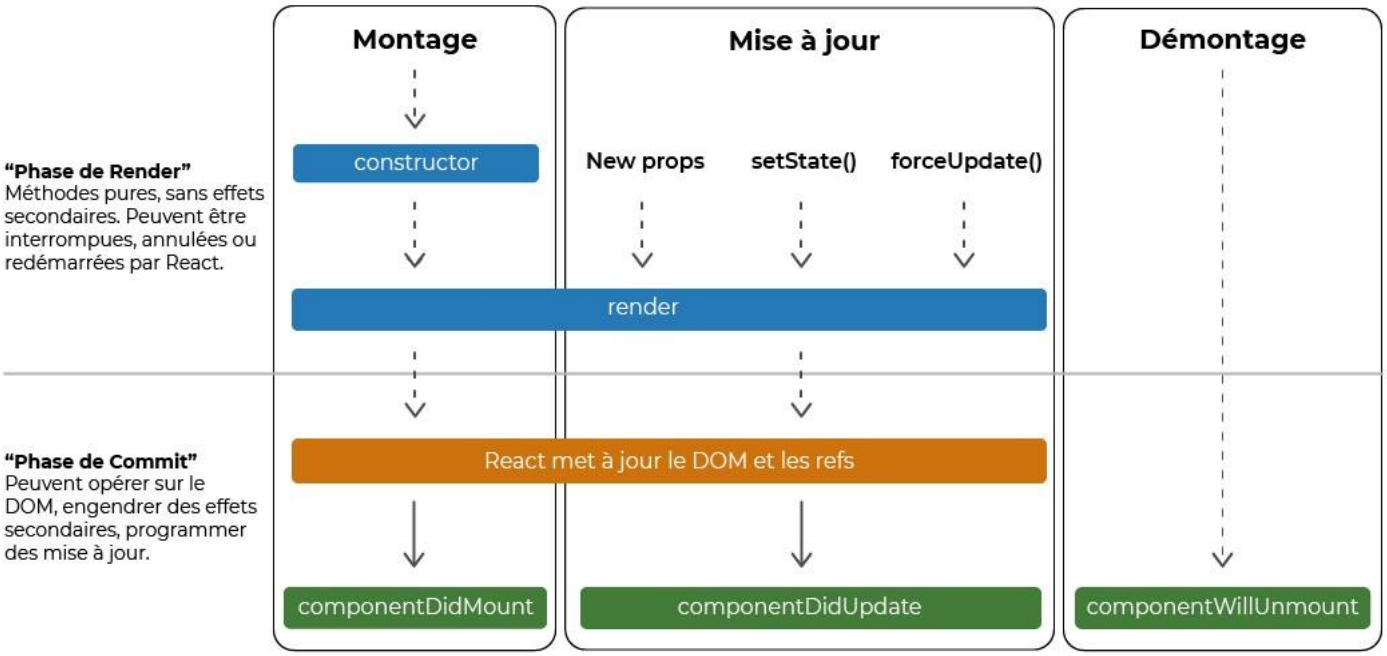
Table des matières

I. Les méthodes de cycle de vie

Le cycle de vie d'un composant regroupe les étapes de la vie d'un composant.

Le cycle de vie d'un composant sous react comprend trois phases qui sont :

- **Mount** : le montage. Il intervient quand une instance du composant est créée dans le DOM.
- **Update** : la mise à jour. Ce cycle de vie est déclenché par un changement d'état du composant.
- **Unmount** : le démontage. Cette méthode est appelée une fois qu'un composant est retiré du DOM.



II. Le montage

Ces méthodes sont appelées dans l'ordre suivant lorsqu'une instance d'un composant est créée et insérée dans le DOM :

1. Constructor
2. Render
3. componentDidMount

a. Construsteur

Comme dans la programmation oriente objet, le constructeur est appelé en premier. Il intervient dès que le composant doit apparaître dans le DOM virtuel.

Si vous l'implémentez, il est important de bien appeler le constructeur de la classe parente (Component) avec le mot-clé super afin de lui fournir les props.

C'est au niveau de constructeur que nous allons initialiser l'état du composant (le state). Et réaliser le bind pour garantir le this de certaines méthodes.

b. Render

Cette méthode est celle du rendu. Elle intervient pour rendre votre JSX dans le DOM virtuel (et donc générer le HTML). C'est à ce moment-là que vous avez l'état de votre composant à jour. Que ce soit vos props ou votre state, vos données sont disponibles et prêtes à être manipulées afin de rendre ce que vous souhaitez.

c. ComponentDidMount

La méthode componentDidMount() nous permet d'exécuter le code React lorsque le composant est déjà placé dans le DOM (Document Object Model). Cette méthode est appelée pendant la phase de montage du cycle de vie React, c'est-à-dire après le rendu du composant.

Toutes les requêtes AJAX et la mise à jour du DOM ou de l'état doivent être codées dans le bloc de méthodes componentDidMount().

d. Exemples

Exemple 1 :

```
import React, { Component } from "react";
export default class ExempleCycle extends Component {
  constructor(props) {
    super(props);
    console.log("[Appel de constructeur ...]");
  } render() {
    console.log("[Appel de render ...]");
    return <div>ExempleCycle</div>;
  } componentDidMount() {
    console.log("[Appel de componentDidMount ...]");
  }

}
```

Exemple 2 :

```
import React, { Component } from "react";
export default class ExempleCycle extends Component {
  constructor(props) {
    super(props);
    this.state = {
      color: 'red',
    }
  }
  render() {
    const style = {
      color: this.state.color, backgroundColor: "rgba(0,0,0,0.88)", textAlign:
"center", paddingTop: 20, width: 400, height: 80, margin: "auto",
    };
    return <div style={style}>Exemple des methodes de cycle de vie</div>;
  }
  componentDidMount() {
    this.setState({ color: 'blue' });
  }
}
```

Exemple 3 :

```
import React, { Component } from "react";
export default class ExempleCycle extends Component {
  constructor(props) {
    super(props);
    this.state = {
      color: "red",
    };
  } render() {
    const style = {
      color: this.state.color, backgroundColor: "rgba(0,0,0,0.88)", textAlign:
"center", paddingTop: 20, width: 400, height: 80, margin: "auto",
    }; return <div style={style}>Exemple des methodes de cycle de vie</div>;
  } componentDidMount() {
    setTimeout(() => this.setState({ color: "blue" }), 3000);
  }
}
```

III. La mise à jour

Une mise à jour peut être provoquée par des changements de props ou d'état. Ces méthodes sont appelées dans l'ordre suivant lorsqu'un composant fait l'objet d'un nouveau rendu :

1. Render
2. ComponentDidUpdate

a. ComponentDidUpdate

Un exemple d'utilisation de componentDidUpdate() est lorsque nous devons appeler une API externe à condition que l'état précédent et l'état actuel aient changé.

L'appel à l'API serait conditionné au changement d'état. S'il n'y a pas de changement d'état, aucune API n'est appelée.

```
componentDidUpdate(prevProps, prevState) {
      if (prevState.value !== this.state.value) {
            console.log('pokemons state has changed.')
            }
}
```

b. Exemples

Exemple 1 :

```
import { Component } from "react"; class ExampleComponent extends Component {
  constructor(props) {
    super(props); this.state = {
      counter: 0,
    };
    this.incrementCounter = this.incrementCounter.bind(this);
  }
  componentDidUpdate(prevProps, prevState) {
    if (this.props.isVisible != prevProps.isVisible) {
      this.incrementCounter();
    }
  }
  incrementCounter() {
    this.setState((state) => ({ counter: state.counter + 1 }));
  } render() {
    return <div>{this.state.counter}</div>;
  }
}
class App extends Component {
  constructor(props) {
    super(props); this.state = {
      visible: true,
    }
  } render() {
    return (<div>
      <ExampleComponent isVisible={this.state.visible} />
      <button onClick={() => this.setState({
        visible:
          !this.state.visible
      })}>Cliquer ici</button>
    </div>
    );
  }
}
export default App;
```

Exemple 2 :

```
import React from 'react';
class App extends React.Component {
  // Defining the state
  state = {
    filiere: 'Techniques de developpement informatique'
  };
  componentDidMount() {
    // Changing the state after 600ms
    setTimeout(() => {       this.setState({ filiere: 'Developpement web full
stack' });
  }, 6000);
}
componentDidUpdate() {
  document.getElementById('disclaimer').innerHTML =
    'ISMO TETOUAN : ' + this.state.filiere;
}
render() {
  return (
    <div>
      <h1 style={{
        margin: 'auto', width: '50%', padding: 20, marginTop: '10%', border:
'solid 1px black', textAlign: 'center', fontSize: 18,
      }}>
        Test methode componentDidUpdate :
        {this.state.filiere}
        <div id="disclaimer"></div>
      </h1>
    </div>
  );
}
}
export default App;
```

IV. Le démontage

a. ComponenWillUnmount

La méthode componentWillUnmount() nous permet d'exécuter le code React lorsque le composant est détruit ou démonté du DOM (Document Object Model). Cette méthode est appelée pendant la phase de démontage du cycle de vie de React, c'est-à-dire avant que le composant ne soit démonté.

```
import React from 'react';
class ComponentOne extends React.Component {
  componentWillUnmount() {
    alert('The component is going to be unmounted');
  } render() {
    return <h1>Hello Geeks!</h1>;
  }
}
class App extends React.Component {
  state = { display: true };
  delete = () => { this.setState({ display: false }); };
  render() {
    let comp;
    if (this.state.display) {
      comp = <ComponentOne />;
    } return (<div>
      {comp}
      <button onClick={this.delete}>
        Delete the component
      </button>
    </div>
    );
  }
} export default App;
```