Rendu conditionnel

Sommaire

I. Présentation

Dans React, vous pouvez créer des composants distincts qui encapsulent le comportement dont vous avez besoin. Ensuite, vous pouvez n'en rendre que certains, en fonction de l'état de votre application.

Le rendu conditionnel dans React fonctionne de la même manière que les conditions fonctionnent en JavaScript. Utilisez des opérateurs JavaScript tels que if : https://developer.mozilla.org/en-US/docs/Web/JavaScript/Reference/Statements/if...else ou l'opérateur conditionnel : https://developer.mozilla.org/en/docs/Web/JavaScript/Reference/Operators/Conditional_Operator pour créer des éléments représentant l'état actuel et laissez React mettre à jour l'interface utilisateur pour les faire correspondre.

Considérez ces deux composants :

```
function UserGreeting(props) {
  return <h1>Welcome back!</h1>;
}

function GuestGreeting(props) {
  return <h1>Please sign up.</h1>;
}
```

Nous allons créer un composant Greeting qui affiche l'un ou l'autre de ces composants selon qu'un utilisateur est connecté ou non :

```
function Greeting(props) {
  const isLoggedIn = props.isLoggedIn;
  if (isLoggedIn) {
    return <UserGreeting />;
  }
  return <GuestGreeting />;
}

ReactDOM.render(
  // Try changing to isLoggedIn={true}:
  <Greeting isLoggedIn={false} />,
  document.getElementById('root')
);
```

Cet exemple affiche un message d'accueil différent en fonction de la valeur de la «prop»: isLoggedIn.

II. Variables d'élément

Vous pouvez utiliser des variables pour stocker des éléments. Cela peut vous aider à rendre conditionnellement le rendu d'une partie du composant alors que le reste de la sortie ne change pas.

Considérez ces deux nouveaux composants représentant les boutons Déconnexion et Connexion :

```
function LoginButton(props) {
  return (
    <button onClick={props.onClick}>
      Login
    </button>
  );
}

function LogoutButton(props) {
  return (
    <button onClick={props.onClick}>
      Logout
    </button>
  );
}
```

Dans l'exemple ci-dessous, nous allons créer un composant avec état appelé LoginControl.

Il rendra soit <LoginButton />ou <LogoutButton /> en fonction de son état actuel. Il va également rendre un <Greeting /> de l'exemple précédent :

```
class LoginControl extends React.Component {
  constructor(props) {
    super(props);
    this.handleLoginClick =
this.handleLoginClick.bind(this);
    this.handleLogoutClick =
this.handleLogoutClick.bind(this);
    this.state = {isLoggedIn: false};
  }

  handleLoginClick() {
    this.setState({isLoggedIn: true});
  }

  handleLogoutClick() {
    this.setState({isLoggedIn: false});
  }

  render() {
    const isLoggedIn = this.state.isLoggedIn;
    let button;

    if (isLoggedIn) {
      button = <LogoutButton
onClick={this.handleLogoutClick} />;
    } else {
      button = <LoginButton
onClick={this.handleLoginClick} />;
    }

    return (
      <div>
        <Greeting isLoggedIn={isLoggedIn} />
        {button}
      </div>
    );
  }
}
```

```
ReactDOM.render(
  <LoginControl />,
  document.getElementById('root')
);
```

Bien que la déclaration d'une variable et l'utilisation d'une instruction if constituent un bon moyen de restituer un composant de manière conditionnelle, vous pouvez parfois utiliser une syntaxe plus courte. Il existe plusieurs façons de créer des conditions en ligne dans JSX, décrites ci-dessous.

III. En ligne si avec l'opérateur logique &&

Vous pouvez intégrer toutes les expressions dans JSX en les entourant d'accolades. Cela inclut l'opérateur logique JavaScript **&&**. Cela peut être pratique pour inclure conditionnellement un élément :

```
function Mailbox(props) {
  const unreadMessages = props.unreadMessages;
  return (
    <div>
      <h1>Hello!</h1>
      {unreadMessages.length > 0 &&
        <h2>
          You have {unreadMessages.length} unread
messages.
        </h2>
      }
    </div>
  );
}

const messages = ['React', 'Re: React', 'Re:Re: React'];
ReactDOM.render(
  <Mailbox unreadMessages={messages} />,
  document.getElementById('root')
);
```

Cela fonctionne parce qu'en JavaScript, **true && expression** évalue toujours **expression** et **false && expression** évalue toujours **false**.

Par conséquent, si la condition est **true**, l'élément juste après && apparaîtra dans la sortie. Si c'est le cas false, React l'ignorera.

IV. Inline si-sinon avec opérateur conditionnel

Une autre méthode de rendu conditionnel des éléments en ligne consiste à utiliser l'opérateur conditionnel JavaScript : **condition ? true : false**.

Dans l'exemple ci-dessous, nous l'utilisons pour restituer sous condition un petit bloc de texte.

```
render() {
  const isLoggedIn = this.state.isLoggedIn;
  return (
    <div>
      The user is <b>{isLoggedIn ? 'currently' :
'not'}</b> logged in.
    </div>
  );
}
```

Il peut également être utilisé pour les expressions plus grandes, bien qu'il soit moins évident de savoir ce qui se passe :

```
render() {
  const isLoggedIn = this.state.isLoggedIn;
  return (
    <div>
      {isLoggedIn ? (
        <LogoutButton onClick={this.handleLogoutClick} />
      ) : (
        <LoginButton onClick={this.handleLoginClick} />
      )}
    </div>
  );
}
```

Tout comme en JavaScript, il vous appartient de choisir un style approprié en fonction de ce que vous et votre équipe considérez plus lisible. N'oubliez pas non plus que lorsque les conditions deviennent trop complexes, il peut s'avérer utile d'extraire un composant : https://reactjs.org/docs/components-and-props.html#extracting-components .

V. Empêcher le rendu des composants

Dans de rares cas, vous voudrez peut-être qu'un composant se cache même s'il a été rendu par un autre composant. Pour ce faire, retourner à null place de la sortie de rendu.

Dans l'exemple ci-dessous, le rendu <WarningBanner /> dépend de la valeur de l'accessoire appelé warn. Si la valeur de l'accessoire est false, le composant ne rend pas :

```
function WarningBanner(props) {
  if (!props.warn) {
    return null;
  }

  return (
    <div className="warning">
      Warning!
    </div>
  );
}
```

```
class Page extends React.Component {
  constructor(props) {
    super(props);
    this.state = {showWarning: true};
    this.handleToggleClick =
this.handleToggleClick.bind(this);
  }

  handleToggleClick() {
    this.setState(state => ({
      showWarning: !state.showWarning
    }));
  }

  render() {
    return (
      <div>
        <WarningBanner warn={this.state.showWarning} />
        <button onClick={this.handleToggleClick}>
          {this.state.showWarning ? 'Hide' : 'Show'}
        </button>
      </div>
    );
  }
}

ReactDOM.render(
  <Page />,
  document.getElementById('root')
);
```

Le retour null d'une méthode render de composant n'affecte pas l'activation des méthodes de cycle de vie du composant. Par exemple, componentDidUpdate sera toujours appelé.