**componentDidUpdate**

N'oubliez pas les trois parties du cycle de vie d'un composant :

1. Mounting, lorsque le composant est initialisé et mis dans le DOM pour la première fois
2. Updating, lorsque le composant est mis à jour à la suite d'un changement d'état ou props modifiés
3. Unmounting, lorsque le composant est supprimé du DOM

We’ve looked at mounting (constructor(), render(), and componentDidMount()). We’ve looked at unmounting (componentWillUnmount()). Let’s finish by looking at the updating phase.

Une mise à jour est causée par des modifications apportées aux props ou à state. Vous avez déjà vu cela se produire un tas de fois. Chaque fois que vous avez appelé setState() avec de nouvelles données, vous avez déclenché une mise à jour. Chaque fois que vous modifiez les props passés à un composant, vous l'avez mis à jour.

Lorsqu'un composant se met à jour, il appelle plusieurs méthodes, mais seules deux sont couramment utilisées.

Le premier est render(), que nous avons vu dans chaque composant React. Lorsque les props ou state d'un composant changent, render() est appelé.

Le second, que nous n'avons pas encore vu, est componentDidUpdate(). Tout comme componentDidMount() est un bon endroit pour la configuration de la phase de Mounting, componentDidUpdate() est un bon endroit pour le travail de la phase de Updating.

1. Nous avons fait une petite mise à jour de l'application Horloge. Notre composant <Clock /> affichera toujours l'heure actuelle, mais il peut également être en "mode précis", où une version plus standardisée et exacte de l'heure sera affichée.

Exécutez l'application. Cliquez sur le bouton "Basculer en mode précis" et changez plusieurs fois l'horloge du mode "précis" pour voir comment cela fonctionne.

1. Tout semble correct, mais les mises à jour ne se produisent pas très fréquemment en mode précis. Même si les données changent plusieurs fois par seconde, elles ne sont mises à jour et restituées qu'une fois par seconde. Peut-on faire mieux ?

Nous aimerions que l'horloge se mette à jour plus fréquemment en mode précis. Au lieu de mettre à jour toutes les secondes, nous aimerions qu'il se mette à jour 10 fois par seconde (toutes les 100 millisecondes au lieu de toutes les 1000). Commençons là-dessus.Créez une nouvelle méthode vide, componentDidUpdate(). Il devrait prendre un argument, prevProps, qui sera les accessoires que ce composant avait la dernière fois qu'il a été rendu. Nous allons le remplir ensuite.

1. Remplissons componentDidUpdate() avec trois étapes de haut niveau :

Si this.props.isPrecise n'a pas changé, arrêtez-vous maintenant et ne faites pas les étapes suivantes.

Effacez l'intervalle existant, comme nous le faisons dans componentWillUnmount(). Commencez un nouvel intervalle. S'il est en mode "précis", il doit s'exécuter toutes les 100 millisecondes. Sinon, il devrait s'exécuter toutes les 1000 millisecondes comme avant.

Dans le code, l'étape 1 consistera à vérifier si isPrecise a changé en le comparant à prevProps. Copiez collez ce code au début :

if (this.props.isPrecise === prevProps.isPrecise) {  
  return;  
}

Pour l'étape 2, copiez-collez le corps de componentWillUnmount() - la ligne unique qui commence par clearInterval(....

Nous remplirons ensuite l'étape 3. Jouez avec la solution et remarquez que le passage au "mode précis" arrête complètement l'intervalle, gelant ainsi l'horloge. Nous allons résoudre ce problème

1. Maintenant que nous avons effacé l'intervalle, nous devons en commencer un nouveau. Au bas de componentDidUpdate(), déclarez une nouvelle variable, appelée delay. Si nous sommes en mode "précis", il doit être réglé sur 100. Sinon, il doit être réglé sur 1000.

Une fois que nous aurons cela, nous commencerons un nouvel intervalle un peu comme nous le faisons dans componentDidMount().

Voici un squelette de ce à quoi le code devrait ressembler :

let delay;  
// ...  
this.intervalID = setInterval(() => {  
  this.setState({ date: new Date() });  
}, delay);

Une fois que vous avez fait cela, essayez d'entrer en mode "precise". L'horloge devrait se mettre à jour plus fréquemment !

1. C'est l'heure du nettoyage final. Notre petite application fonctionne, mais a deux problèmes :

Si l'horloge est démarrée en mode "précis", le mauvais intervalle est démarré et l'horloge ne sera pas mise à jour assez fréquemment.

Une partie du code est dupliquée.

Refactorisons notre composant pour résoudre ces deux problèmes.

À un niveau élevé, nous allons créer une nouvelle méthode appelée startInterval(). Il sera chargé de calculer le délai d'intervalle (soit 100 ou 1000 millisecondes) et d'appeler setInterval(). Ensuite, nous l'appellerons depuis componentDidMount() et componentDidUpdate().

Créez une nouvelle méthode, startInterval(), qui inclura tout le code de configuration de l'intervalle. Ensuite, recherchez le code qui définit l'intervalle dans componentDidUpdate(). (Il doit commencer par let delay et se terminer par l'appel à setInterval().) Coupez et collez-les dans le corps de startInterval(). À sa place, mettez un appel à this.startInterval().

Enfin, supprimez l'appel à setInterval() de componentDidMount() et remplacez-le par un appel à this.startInterval(), afin que les deux méthodes utilisent le même chemin de code.

Voyez que tout fonctionne comme avant, mais avec le bug corrigé !

Question

In the context of this [exercise 11](https://www.codecademy.com/paths/web-development/tracks/front-end-applications-with-react/modules/react-102-lifecycle-methods-u/lessons/mounting-lifecycle-methods/exercises/componentdidmount), introducing the componentDidMount() method, can you set the component’s state in this method?

Answer

Yes, you can set the component’s state in this method, using the setState() method. This is not possible in some other lifecycle methods, such as render(), because calling setState() within them can result in infinite loops.

Calling setState() in componentDidMount() will not end up in an infinite loop, but will just invoke an additional rendering. This additional rendering will not be visible as both of the renders will happen be right before the browser updates the screen, but, one drawback to doing this is that the additional render can take some additional time and cause the site to display slower.

Instead of setting the state initially within componentDidMount(), you may consider setting the initial state in the constructor() method instead.

import React from 'react';

export class Clock extends React.Component {

  constructor(props) {

    super(props);

    this.state = { date: new Date() };

  }

  render() {

    return (

      <div>

        {this.props.isPrecise

          ? this.state.date.toISOString()

          : this.state.date.toLocaleTimeString()}

      </div>

    );

  }

  startInterval(){

    let delay;

    if(this.props.isPrecise){

      delay = 100;

    } else {

      delay = 1000;

    }

    this.intervalID = setInterval(() => {

      this.setState({ date: new Date() });

    }, delay)

  }

  componentDidMount() {

    this.startInterval();

  }

  componentDidUpdate(prevProps){

    if(this.props.isPrecise === prevProps.isPrecise){

     return

    }

    clearInterval(this.intervalID);

    this.startInterval();

  }

  componentWillUnmount() {

    clearInterval(this.intervalID);

  }

}