






Hook useEffect

Thèmes	 React
Description	Gérer les mises à jours d'un composant durant son cycle de vie.
Date de création	@16 mars 2025 17:21
Créée par	 Kévin Wolff

Chapitres


 [Introduction](#)


 [Montage du composant](#)


 [Mise à jour du composant](#)


 [Destruction du composant](#)

Sur le même sujet

 [Premier composant React](#)

 [Cycle de vie d'un composant](#)


 [Hook useState](#)

 [Hook custom](#)

Introduction

 [Documentation officielle - useEffect](#)

Le hook `useEffect` permet de gérer les mises à jour d'un composant et de définir son comportement à différentes étapes de son cycle de vie.

Il remplace les méthodes `componentDidMount`, `componentDidUpdate` et `componentWillUnmount` utilisées dans les composants de classe. Pour en savoir plus sur le cycle de vie d'un composant, consultez le document  [Cycle de vie d'un composant](#).

Importez `useEffect` depuis React avant de l'utiliser :

```
import { useEffect } from 'react';
```

Une fois importé, il peut être utilisé au sein d'un composant fonctionnel pour exécuter des effets :

```
import { useEffect } from 'react';

function Counter() {
  useEffect(() => {
    // Effet à exécuter
  }, []);

  return (
    <div>
      <p>Hello world</p>
    </div>
  );
}
```

```
};  
}
```

La fonction `useEffect` attend deux paramètres :

1. Une fonction callback qui contient l'effet à exécuter. Ici, une fonction fléchée anonyme est utilisée.
2. Un tableau de dépendances qui peut être vide ou contenir des variables dont les changements déclencheront l'effet.

Montage du composant

Phase `componentDidMount` .

Pour reproduire l'effet de `componentDidMount` , qui s'exécute au montage du composant, il faut utiliser `useEffect` avec un tableau de dépendances vide.

```
import { useEffect } from 'react';  
  
function Counter() {  
  useEffect(() => {  
    console.log("Hello world !");  
  }, []);  
  
  return (  
    <div>  
      <p>Hello world</p>  
    </div>  
  );  
}
```

Dans cet exemple, le second paramètre de `useEffect` est un tableau vide, ce qui signifie que l'effet ne sera exécuté qu'une seule fois, immédiatement après le montage du composant.

Ce comportement est particulièrement utile pour :

- Effectuer un appel API pour récupérer des données au chargement de la page.
- Initialiser certaines variables ou paramètres globaux.
- Démarrer un timer ou un abonnement à un événement.

Mise à jour du composant

Phase `componentDidUpdate` .

Pour reproduire `componentDidUpdate` , qui s'exécute à chaque mise à jour du composant, `useEffect` doit contenir un tableau de dépendances avec une ou plusieurs variables surveillées.

```
import { useEffect, useState } from 'react';

function Counter() {
  const [count, setCount] = useState(0);

  useEffect(() => {
    console.log("La valeur de count a été mise à jour : " + count);
  }, [count]);

  return (
    <div>
      <p>Hello world</p>
    </div>
  );
}
```

Ici, `useEffect` contient `[count]` comme dépendance. Chaque fois que `count` est modifié par `setCount`, l'effet est déclenché.

Ce mécanisme permet :

- De réagir aux changements d'une variable d'état et d'exécuter du code en conséquence.
- D'effectuer un nouvel appel API si un utilisateur change un filtre dans une interface.
- De mettre à jour dynamiquement des données en fonction des entrées utilisateur.

Si plusieurs variables doivent être suivies, elles peuvent être ajoutées au tableau de dépendances :

```
useEffect(() => {
  console.log("count ou otherVar ont changé !");
}, [count, otherVar]);
```

Destruction du composant

Phase `componentWillUnmount`.

Pour reproduire `componentWillUnmount`, qui s'exécute juste avant la destruction d'un composant, `useEffect` doit retourner une fonction de nettoyage.

```
import { useEffect } from 'react';

function Counter() {

  useEffect(() => {
    return () => {
      console.log("Le composant est détruit");
    }
  }, []);
}
```

```
return (  
  <div>  
    <p>Hello world</p>  
  </div>  
);  
}
```

Dans cet exemple, la fonction retournée par `useEffect` est exécutée juste avant que le composant soit supprimé du DOM.

Ce comportement est essentiel pour :

- Nettoyer un abonnement à un événement.
- Arrêter un intervalle ou un timeout (`setInterval` , `setTimeout`).
- Annuler une requête en cours pour éviter les erreurs de mémoire.