Hooks (partie 2)

Nous avons vu précédemment quelques hooks et leur fonctionnement:

- useState
- useEffect
- useReducer
- useContext

Nous avons aussi construit un hook permettant d'ajouter facilement une fonctionnalité de fetch de données dans un composant React.

Hooks (partie 2)

Voyons maintenant quelques hooks supplémentaires, qui pourront être utiles pour optimiser le fonctionnement d'une application React

useCallback

Sert à mémoïser un callback, c'est à dire : ne pas le recréer à chaque rendu du composant le définissant mais seulement en cas de besoin.

```
const memoizedCallback = useCallback(
  () => {
    doSomething(a, b);
  },
  [a, b],
);
```

useCallback

Mémoïser est l'action de mettre en mémoire le résultat d'une opération selon des données en particulier, pour ne pas avoir à refaire les calculs si on nous demande une fois encore le résultat de cette opération.
Exemple : calculer le millionième nombre premier est dur, alors une fois le calcul fait, on notera précieusement le résultat pour pouvoir répondre facilement la prochaine fois que quelqu'un nous demande de calculer le millionième nombre premier. On aura mémoïser le résultat.

useCallback

Ici, le callback utilise 'a' et 'b', si le composant est rendu de nouveau mais que la valeur de 'a' et de 'b' ne changent pas, il n'y pas de raison de recréer le callback

```
const memoizedCallback = useCallback(
  () => {
    doSomething(a, b);
  },
  [a, b],
);
```

useCallback

Le calcul économisé sera le fait de créer le callback (avec sa closure, ie : la valeur de a et de b au moment de la création du callback)

```
const memoizedCallback = useCallback(
  () => {
    doSomething(a, b);
  },
  [a, b],
);
```

useCallback

En passant un callback inline (défini à la volée) et un array de dépendances, on obtient un callback, qui sera le même au sens de 'Object.is' au fil des rendus, sauf si une des dépendances passées dans l'array voit sa valeur changer,

C'est utile lorsqu'on passe le callback à des childs optimisés.

useMemo

On lui passe une fonction qui crée la valeur à mémoïser et un array de dépendances, et useMemo va nous calculer la valeur seulement si une des dépendances voit sa valeur changer.

Ce hook est à utiliser pour éviter de refaire des calculs complexes à chaque render.

Attention : la fonction passée sera exécutée lors de renders (opérations de rendu), ne pas y faire d'effets de bords.

useMemo

Remarque :

`useCallback(fn, deps)` is equivalent to `useMemo(() => fn, deps)`.

```
const memoizedValue = useMemo(() => computeExpensiveValue(a, b), [a, b]);
```

useMemo

Remarque :

`useCallback(fn, deps)` is equivalent to `useMemo(() => fn, deps)`.

```
const memoizedValue = useMemo(() => computeExpensiveValue(a, b), [a, b]);
```

useMemo

Remarque :

`useCallback(fn, deps)` is equivalent to `useMemo(() => fn, deps)`.

```
const memoizedValue = useMemo(() => computeExpensiveValue(a, b), [a, b]);
```