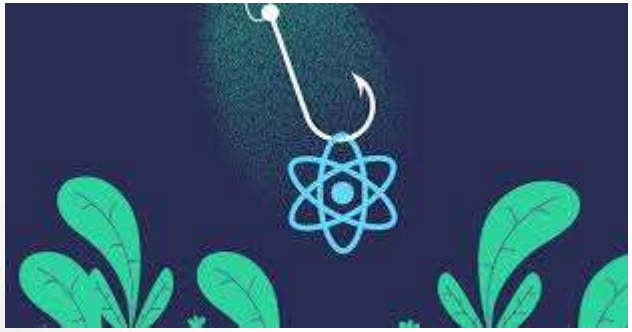
Chapitre 8 : Les Hooks



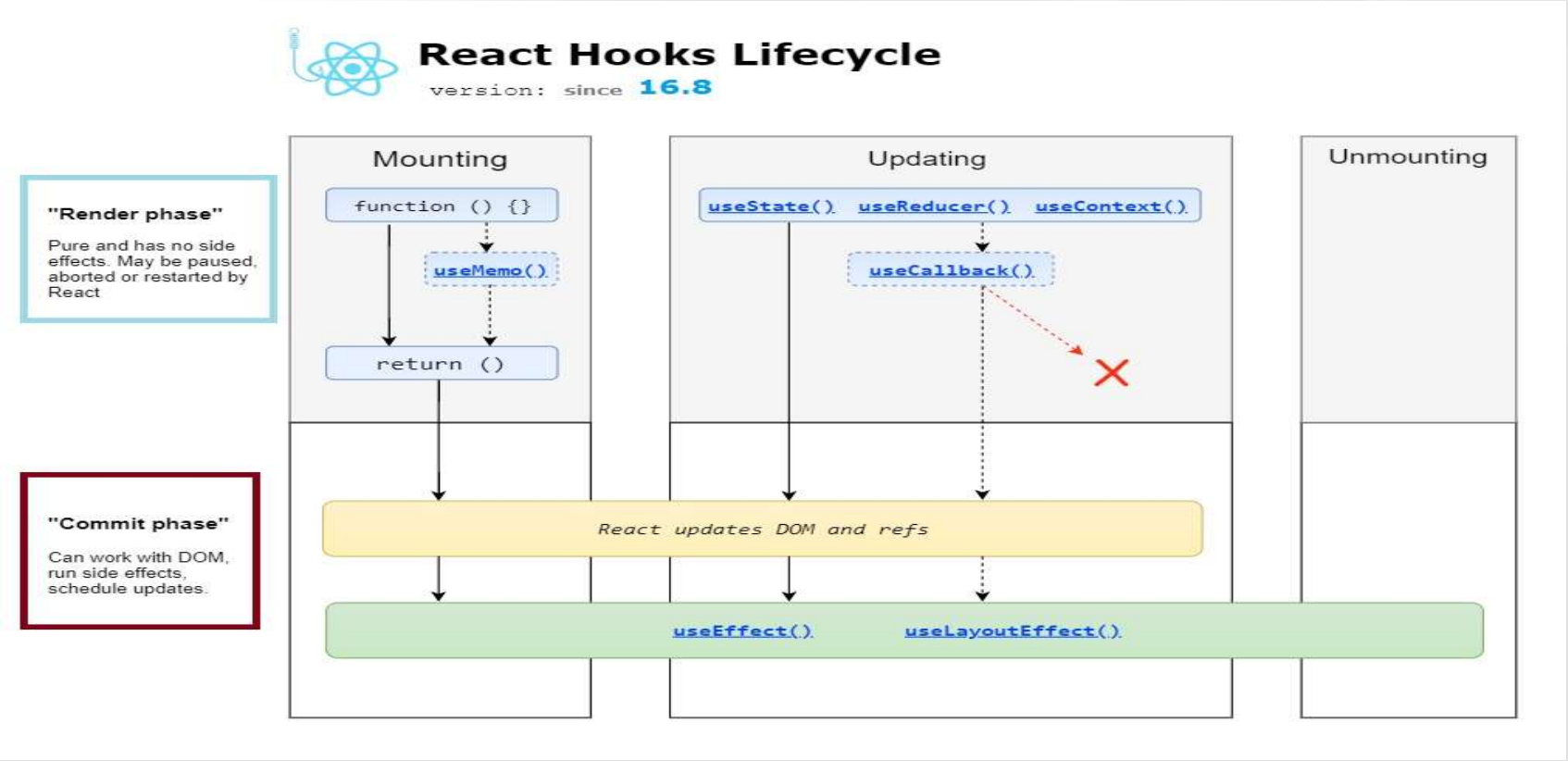
Présentation

- Fin 2018 à l'occasion de la React Conf, Facebook a annoncé l'ajout d'une fonction à React qui a séduit la plupart des développeurs : les Hooks.
- Ces nouvelles fonctionnalités sont arrivés avec React 16.8.
- Ils coïncident avec le fait d'utiliser les functional components.
- La philosophie est la suivante : tout composant devrait pouvoir s'écrire sous une forme fonctionnelle pure.
- Un Hook n'est ni plus, ni moins, qu'une fonction en JavaScript.
- On appele un Hook comme une fonction avec la possibilité de lui passer un ou plusieurs paramètres et cette fonction va renvoyer un résultat.
- Les Hooks ne fonctionnent pas dans des classes. Ils permettent d'utiliser davantage de fonctionnalités de React sans recourir aux classes

Règles d'utilisation des Hooks

- Les hooks sont d'une grande utilité, mais leur utilisation implique de respecter les règles suivantes :
  - Ne pas les utiliser autre part que dans des fonctions React.
  - Ne pas les utiliser dans des boucles.
  - Ne pas les utiliser dans les conditions.
  - Ne pas les utiliser dans les fonctions imbriquées.

Le cycle de vie des Hooks



Le Hook d'état : useState

- Le Hook d'état fait parti des hooks de React que l'on va utiliser régulièrement puisqu'il permet de générer un état local (state) de son composant de fonction.
- Là où l'on a besoin de this.state dans un composant de classe, nous allons avoir besoin du Hook d'état : useState pour un composant de fonction.

```
import React, {useState} from 'react';
const CrochetState = (props) => {
   let [name, setName] = useState('');
   return (
   <div>
   <input type="text" onChange={(event) => setName(event.target.value)}></input>
   <span>Bonjour, je m'appelle {name}</span>
   </div>
   )
  }
export default CrochetState;
```

Le Hook d'effet : useEffect 1/7

- Le deuxième Hook important est useEffect.
- Ce Hook permet d'ajouter notre logique comme nous le faisions avec les méthodes componentDidMount, componentDidUpdate et componentWillUnmount.
- Il est exécuté après chaque render du composant (premier compris) et permet donc de charger des données et interagir avec le DOM.
- useEffect permet de définir une action à effectuer dès que le composant est affiché ou mis à jour. C'est-à-dire chaque mise à jour de la valeur du champ.
- useEffect est une fonction qui accepte 2 paramètres :
  - Le 1er est une fonction de rappel (un callback) qui sera exécutée pour l'effet qu'on souhaite « écouter ».
  - Le 2ème est optionnel et permet de définir l'état (state ou props) que l'on souhaite observer.

Le Hook d'effet : useEffect 2/7

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

const CrochetEffect = (props) => {
   let [name, setName] = useState('');
   useEffect(() => {
      document.title = `Hello ${name}!`
    })
   return (
   <div>
   <input type="text" onChange={(event) => setName(event.target.value)}></input>
   <span>Bonjour, je m'appelle {name}</span>
   </div>
   )
  }
export default CrochetEffect;
```

Le Hook d'effet : useEffect 3/7

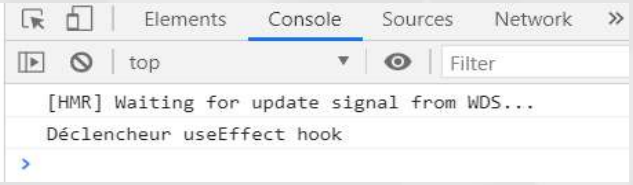
- Si on a plusieurs effets ou si de nouvelles valeurs de props sont lancées à partir d'un composant parent, il peut déclencher l'effet plusieurs fois. Cela peut entraîner des incohérences.

```
import React, {useState,useEffect} from 'react';
const WindowWidthSize  = (props) => {
   const [windowWidthSize, setWindowWidthSize] = useState(0);

   useEffect(() => {
      console.log('Déclencheur useEffect hook');
            function handleResize(e) {
                const { width } = document.body.getBoundingClientRect();
               setWindowWidthSize(Math.ceil(width));
            }
     window.addEventListener('resize', handleResize);
});

return (
     <h1>        La taille de la fenêtre est : {windowWidthSize} pixels       </h1>
   )
} export default WindowWidthSize ;
```

Le Hook d'effet : useEffect 4/7

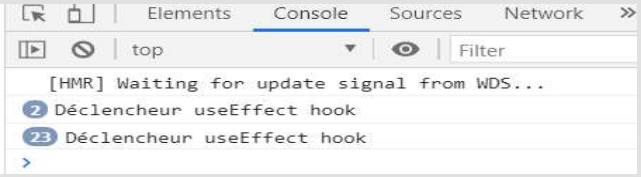
Initialement le message dans la méthode useEffect est affiché ans la console une fois :

La taille de la fenêtre est : 0 pixels



A chaque opération de redimensionnement de la fenêtre, le message est réaffiché car useEffect est invoqué :

La taille de la fenêtre est : 679 pixels



Le Hook d'effet : useEffect 5/7

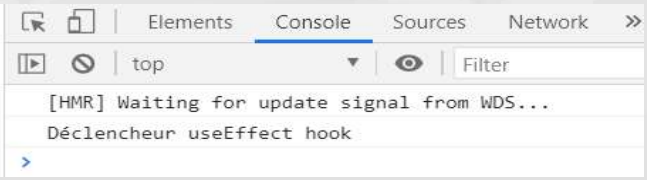
- Pour optimiser le résultat, on peut passer un tableau comme deuxième argument dans useEffetct.
- Un tableau de valeurs, indique que la fonction useEffect sera exécutée uniquement si une des valeurs du tableau a été modifiée depuis l'appel précédent.
- En exécutant un tableau vide [] comme deuxième argument, on fait savoir à React que la fonction useEffect ne dépend d'aucune valeur de props ou de l'état.

```
useEffect(() => {
      console.log('Déclencheur useEffect hook');
    function handleResize(e) {
        const { width } = document.body.getBoundingClientRect();
      setWindowWidthSize(Math.ceil(width));
    }
     window.addEventListener('resize', handleResize);
},[]);
```

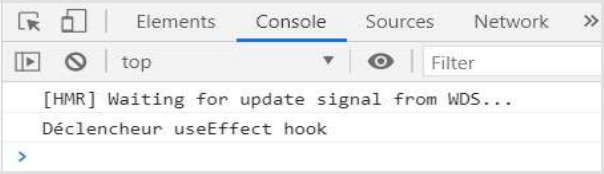
Le Hook d'effet : useEffect 6/7

Même avec plusieurs opérations de redimensionnement de la fenêtre, le message n'est affiché qu'une seule fois :

La taille de la fenêtre est : 755 pixels



La taille de la fenêtre est : 1019 pixels



Le Hook d'effet : useEffect 7/7

- Le Hook useEffect permet de retourner une fonction de clean up qui nous permet de nettoyer après que notre effet de bord ait été exécuté.
- Le return du Hook est considéré comme un mécanisme de cleanup.
- En renvoyant une fonction anonyme avec useEffect, elle s'exécutera avant le démontage du composant.

```
useEffect(() => {
    console.log('Déclencheur useEffect hook');
   function handleResize(e) {
      const { width } = document.body.getBoundingClientRect();
      setWindowWidthSize(Math.ceil(width));
   }
   window.addEventListener('resize', handleResize);

  return () => window.removeEventListener('resize', handleResize);
 },[]);
```

useRef 1/4

- useRef renvoie un objet ref modifiable dont la propriété current est initialisée avec l'argument fourni (initialValue).
- L'objet renvoyé persistera pendant toute la durée de vie composant.
- La syntaxe est la suivante.

const refContainer = useRef(initialValue);

  - refContainer : Objet avec une propriété current
  - initialValue : Optionnel, c'est la valeur donnée à la propriété current du retour de useRef

- L'usage le plus commun de cet Hook est la référence d'un composant.
- Mais c'est surtout intéressant pour garder une variable modifiable qui peut être utilisée sans influencer le reste du comportement.
- Par exemple, on va pouvoir utiliser l'id d'un intervalle de temps pour pouvoir l'arrêter n'importe où dans le composant.

useRef 2/4

```
import React, {useState, useEffect, useRef } from "react";
const Minuteur = (props) => {
  // Définition de la référence
  const intervalRef = useRef();
  const [timer, setTimer] = useState(30);

  useEffect(() => {
    const id = setInterval(() => {
      // On décrémente le minuteur géré par le state
      setTimer((oldTimer) => oldTimer - 1);
    }, 1000);
```

useRef 3/4

```
// Mise à jour de la référence
    intervalRef.current = id;
    return () => {
      // Arrêt du ‘timer’ en cas de suppression ou réaffichage du composant
      clearInterval(intervalRef.current);
    };
  }, []);
  // Fonction permettant d'arrêter le ‘timer’
  const stopTimer = () => {
    clearInterval(intervalRef.current);
  };
```

useRef 4/4

```
  return (
    <div>
      {/* Le timer est affiché ici */}
      <p>Il reste : {timer} secondes</p>
      {/* En cliquant sur ce bouton, le minuteur (et donc l'intervalle) sera arrêté */}
      <button onClick={stopTimer}> STOP! </button>
    </div>
  );
}
export default Minuteur;
```

useReducer 1/4

- Alternative à useState.
- Accepte un réducteur de type (state, action) => newState, et renvoie l'état local actuel accompagné d'une méthode dispatch.
- useReducer est souvent préférable à useState quand on a une logique d'état local complexe qui comprend plusieurs sous-valeurs, ou quand l'état suivant dépend de l'état précédent.
- useReducer permet aussi d'optimiser les performances pour des composants qui déclenchent des mises à jours profondes puisqu'on peut fournir dispatch à la place de fonctions de rappel.

useReducer 2/4

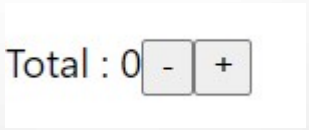
```
import React, {useReducer } from "react";
const CrochetReducer = (props) => {
const initialState = {count: 0};
const reducer=(state, action)=> {
  switch (action.type) {
    case 'increment':
      return {count: state.count + 1};
    case 'decrement':
      return {count: state.count - 1};
    default:
      throw new Error();
  } }
```

useReducer 3/4

```
  const [state, dispatch] = useReducer(reducer, initialState);
  return (
    <>
      Total : {state.count}
      <button onClick={() => dispatch({type: 'decrement'})}>-</button>
      <button onClick={() => dispatch({type: 'increment'})}>+</button>
    </>
  );

}
export default CrochetReducer;
```

useReducer 4/4

Initialement on a :



En incrémentant ou décrémentant on obtient la valeur correspondante :