React Render

1. **Rendering et le Re-rendring**

Une image contenant texte, capture d’écran, Police, nombre

Description générée automatiquement

En React, le terme "render" fait référence à un processus crucial qui consiste à transformer les composants React en éléments DOM (Document Object Model) pour les afficher sur la page web. Plus précisément, le rendu en React se produit lorsque les composants définis dans l'application sont convertis en une représentation visuelle sur l'écran.

Voici comment fonctionne le rendu en React :

1. **Composants React**: Les applications React sont construites à partir de composants. Un composant est une unité indépendante et réutilisable de l'interface utilisateur, qui peut être une partie d'une page web ou même une page entière. Ces composants sont généralement écrits en utilisant du code JSX (JavaScript XML), qui ressemble beaucoup à du HTML.
2. **Éléments React**: Une fois que les composants sont créés, React les traite pour créer des éléments React. Les éléments React sont des objets simples qui représentent la structure du DOM souhaitée. Ils sont créés à partir des composants React et des propriétés qui leur sont passées.
3. **Réconciliation et rendu**: Lorsque le rendu est déclenché (par exemple, lorsque l'état d'un composant change), React compare les éléments React nouvellement créés avec les éléments existants du DOM. Ce processus est appelé "réconciliation". React identifie les différences entre les anciens et les nouveaux éléments et met à jour uniquement les parties du DOM qui ont besoin de changement.
4. **Mise à jour du DOM**: Une fois que React a effectué la réconciliation, il met à jour le DOM en ajoutant, supprimant ou modifiant les éléments nécessaires pour refléter l'état mis à jour de l'application. La mise à jour du DOM est optimisée pour minimiser les modifications directes et rendre le processus plus efficace.

Grâce à ce processus de rendu efficace, React parvient à fournir des interfaces utilisateur réactives et performantes. Les rendus fréquents sont gérés de manière optimisée, ce qui permet aux développeurs de se concentrer sur la logique métier de leur application sans se soucier de la manipulation manuelle du DOM.

* **Rendering en React**

Une image contenant texte, capture d’écran, conception

Description générée automatiquement

Pendant la phase de rendu, React commence à la racine de l'arborescence des composants et descend vers les composants feuilles. Pendant ce parcours, pour chaque composant, React invoque la méthode "createElement" et convertit le JSX du composant en un élément React, stockant ainsi la sortie de rendu. Un élément React est essentiellement un objet JavaScript qui décrit la structure de l'interface utilisateur.

Une fois que la conversion est terminée, tous les éléments React sont transmis à la phase de validation (commit phase). Dans cette phase, les éléments React sont appliqués au DOM en utilisant le package React DOM.

* **Re-rendering en React**

Une image contenant texte, capture d’écran, diagramme, Rectangle

Description générée automatiquement

Une image contenant texte

Description générée automatiquement

Pendant la phase de rendu, React commence à partir de la racine de l'arbre des composants et descend vers les composants feuilles, en recherchant tous les composants qui ont été marqués comme nécessitant des mises à jour. Un composant peut se marquer lui-même pour une mise à jour en appelant la fonction de modification de l'état (**useState**) ou la fonction de dispatch du réducteur (**useReducer**). Ensuite, pour chaque composant, React invoque la méthode **createElement** et convertit le JSX du composant en éléments React, puis stocke cette sortie de rendu.

Une fois que la conversion est effectuée pour tous les composants marqués, React compare le nouvel ensemble d'éléments React avec ceux qui ont été produits lors du dernier rendu. Une liste est créée avec toutes les modifications qui doivent être apportées au DOM et est transmise à la phase de validation. Durant cette phase, les changements sont effectivement appliqués au DOM.

1. **UseState & useReducer**
2. **useState et le render**

**Une image contenant texte, Police, capture d’écran, blanc

Description générée automatiquement**

Une image contenant texte, capture d’écran, logiciel, affichage

Description générée automatiquement

import React, { useState } from "react";

export const UseState = () => {

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

  console.log('UseState Render')

  return (

    <div>

      <button onClick={() => setCount((c) => c + 1)}>Count-{count}</button>

      <button onClick={() => setCount(0)}>Count to 0-{count}</button>

      <button onClick={() => setCount(5)}>Count to 5-{count}</button>

    </div>

  );

};

1. **useReducer**

import React,{useReducer} from 'react'

   const UseReducer = () => {

    const initialiseState=0;

const reducer=(state,action)=>{

    switch(action){

        case 'increment' :return state+1

        case 'decrement' :return state-1

        case 'reset' :return initialiseState

        default :return state

    }

}

    const [count,dispatch]=useReducer(reducer,initialiseState)

  console.log("UseReduecer Render")

  return (

<div>

<div>{count}</div>

<button onClick={() => dispatch("increment")}>Increment</button>

<button onClick={() => dispatch("decrement")}>Decrement</button>

<button onClick={() => dispatch('reset')}>Reset</button>

</div>

  )

}

export default UseReducer

La fonction **dispatch** d'un hook **useReducer** provoquera le nouveau rendu du composant à chaque fois, même si les valeurs sont identiques après la valeur initiale.

1. **L’immutabilité de l'état**
   1. **Differrence entre prs={…person} et prs=person**

let person = {

  name: 'John',

  age: 30,

  city: 'New York'

};

let prs = { ...person }; // Copie immuable de l'objet person

let prs1 = person;       // Référence au même objet person

console.log(person); // Affiche : { name: 'John', age: 30, city: 'New York' }

console.log(prs);    // Affiche : { name: 'John', age: 30, city: 'New York' }

console.log(prs1);   // Affiche : { name: 'John', age: 30, city: 'New York' }

// Vérification de l'immutabilité

console.log(person === prs);  // Affiche : false (copie immuable, références différentes)

console.log(person === prs1); // Affiche : true (même objet, référence identique)

En résumé, **prs** sera une copie immuable de l'objet **person**, tandis que **prs1** fera référence au même objet que **person**. Cela signifie que si vous modifiez **prs**, l'objet **person** restera inchangé, mais si vous modifiez **prs1**, l'objet **person** sera également modifié car ils pointent vers la même référence mémoire.

* 1. **L’immutabilité de l'état**

L'« immutabilité de l'état » en français se réfère au concept de ne pas modifier directement l'état (ou les données) d'un objet ou d'une variable une fois qu'il a été créé. Au lieu de cela, on crée une nouvelle copie de l'état avec les modifications nécessaires.

« Modifier directement un objet ou un tableau en tant qu'état ne déclenchera pas de nouveau rendu lorsque vous l'utilisez avec les hooks useState ou useReducer.

Pour déclencher un nouveau rendu, faites une copie de l'état existant, effectuez les modifications nécessaires, puis transmettez le nouvel état à la fonction de modification (setter function) ou lors du retour d'une fonction réductrice (reducer function).

Modifier directement l'état est une manière facile de créer des bugs dans votre application. Assurez-vous de ne pas le faire. »

ObjectUseState

import React ,{useState} from 'react'

const initState={

    fName:'Houssam',

    lName:"Mrabte"

}

export const ObjectUseState = () => {

    const [person,setPerson]=useState(initState)

    const changeName=()=>{

        let prs={...person}

        prs.fName='Abir'

        prs.lName="Lmrabte"

        setPerson(prs)

    }

    console.log("ObjectUseState Render")

  return (

    <div>

        <button onClick={()=>changeName()}>{person.fName}  {person.lName}</button>

    </div>

  )

}

ArrayUseState

import React ,{useState} from 'react'

const initState=['Bruce','Wayne']

export const ArrayUseState = () => {

    const [persons,setPersons]=useState(initState)

    const handleClick=()=>{

        let prs=[...persons]

        prs.push("Clark")

        prs.push("Test")

        setPersons(prs)

    }

    console.log("ArrayUseStateRender")

  return (

    <div>

          <button onClick={handleClick} >Click</button>

          {

            persons.map( (person,i)=>(

                <div key={i}>{person}</div>

            ))

          }

    </div>

  )

}

* 1. **Le render avec des composants enfants et parents**

Une image contenant texte

Description générée automatiquementUne image contenant texte, capture d’écran, Police, diagramme

Description générée automatiquement

Parent.js

import React, { useState } from "react";

import { Child } from "./Child";

export const Parent = () => {

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

  console.log('Parent Render')

  return (

    <div>

        <Child/>

      <button onClick={() => setCount((c) => c + 1)}>Count-{count}</button>

      <button onClick={() => setCount(0)}>Count to 0-{count}</button>

      <button onClick={() => setCount(5)}>Count to 5-{count}</button>

    </div>

  );

};

Child.js

import React from "react";

export const Child = () => {

  console.log("Cild Render");

  return <div>Child components</div>;

};

Une image contenant texte

Description générée automatiquement

1. **Optimisation**
2. **référence au même élément**

Une image contenant texte, capture d’écran, Police, logiciel

Description générée automatiquement

Une image contenant texte, capture d’écran, Police, Site web

Description générée automatiquement

Une image contenant texte

Description générée automatiquement

Solution:

GrandParents.js

import React,{useState} from 'react'

import { ChildOne } from './ChildOne'

import { ParentOne } from './ParentOne'

export const GrandParents = () => {

    const [newCount,setNewCount]=useState(0)

    console.log("GrandParent Render")

  return (

    <div>

        <button onClick={()=>setNewCount((prev)=>prev+1)}>

        GrandParents - Count :{newCount}

        </button>

        <ParentOne>

            <ChildOne/>

        </ParentOne>

        </div>

  )

}

ParentOne.js

import React, { useState } from "react";

export const ParentOne = ({ children }) => {

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

  console.log("ParentOne Render");

  return (

    <div>

      <button onClick={() => setCount((c) => c + 1)}>Count-{count}</button>

      {children}

    </div>

  );

};

ChildOne.js

import React from "react";

export const ChildOne = () => {

  console.log("ChildOne Render");

  return <div>ChildOne components</div>;

};