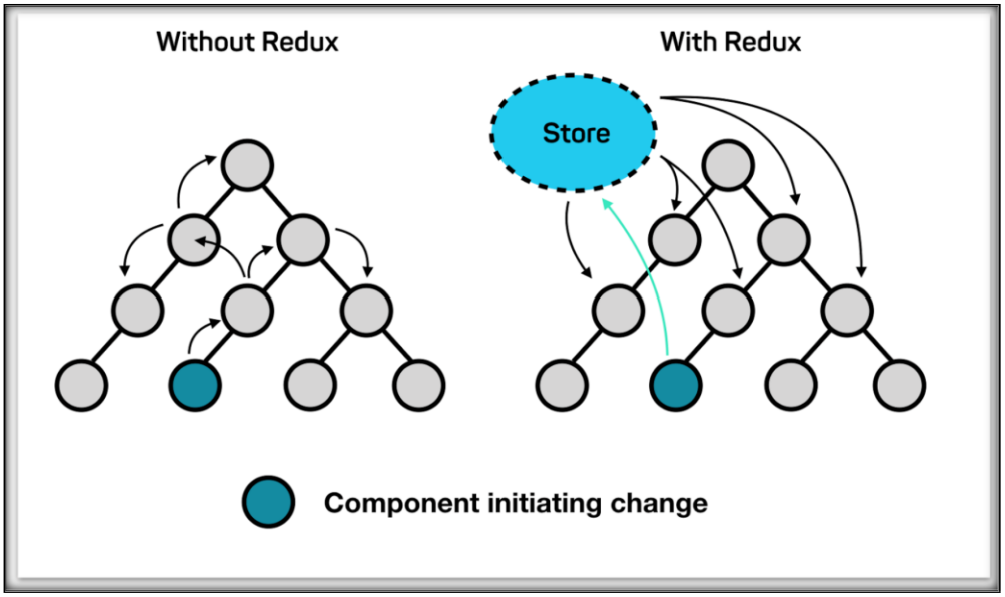
Chapitre 12 : Redux

Table des matières

I. Définition

Redux est une bibliothèque JavaScript open source pour gérer et centraliser l'état des applications code avec JavaScript.



Le plus souvent Redux est utilisé avec des bibliothèques telles que React ou Angular pour créer des interfaces utilisateur.

II. Installation de Redux

Voici les deux packages qu'il faut installer pour travailler avec Redux dans une application React :

- **npm install redux**
- **npm install react-redux**

III. Les concepts de Redux

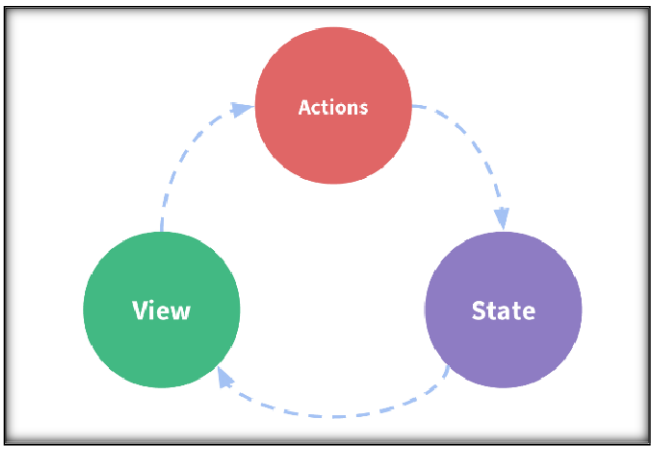
a. Les actions

L'état dans Redux est en lecture seule. Cela vous permet de restreindre toute partie de la vue ou tout appel réseau pour écrire/mettre à jour l'état directement.

La seule façon de changer l'état est d'émettre une action.

Une action qui est un objet JavaScript décrivant quelque chose qui s'est produit dans l'application.

Autrement dit, si quelqu'un veut changer l'état de l'application, il devra exprimer son intention de le faire en émettant ou en envoyant une action.



Un objet action possède deux champs :

- **Type :** Ce champ est de type chaine de caractères, il permet de décrire le type de l'action.
- **Payload :** ce champ n'est pas obligatoire, il contient des informations supplémentaires sur l'action.

La syntaxe de déclaration d'une action est :

```
const Actions = {
      type: '',
      payload: ''
}
```

Exemple 1 :

```
const action1 = {
      type: 'INCREMENT_COUNTER',
}
```

Exemple 2 :

```
const action2 = {
      type: 'INCREMENT_COUNTER',
```

```
        payload: 2
    }
```

Exemple 3:

```
    const action3 = {
        type: "ADD_USER",
        payload: { id: "1", name: "Mary" },
    }
```

Exemple 4 :

```
 const action3 = {
    type: "ADD_USERS",
    payload: [
      { id: "1", name: "Mary" },
      { id: "2", name: "Jane" },
    ],
  };
```

b. Action Creators

Action Creator est une fonction qui permet de créer et retourner une action.

Exemple 1:

```
    const incrementer = () => {
        return {
            type: 'INCREMENT_COUNTER',
        }
    }
```

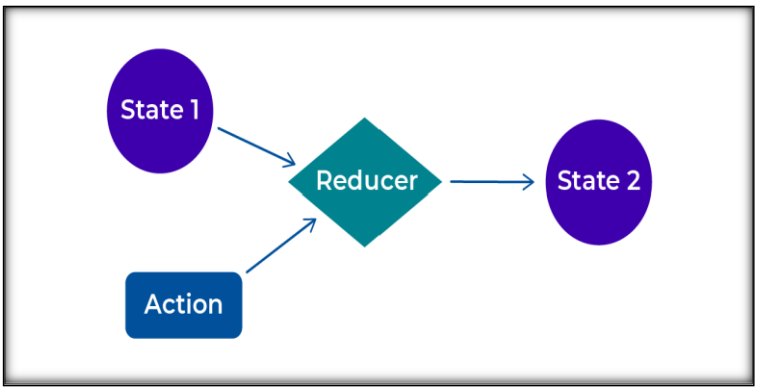
Exemple 2:

```
const doAddToDoItem = (text) => {
  return { type: "TODO_ADD", payload: text };
};
```

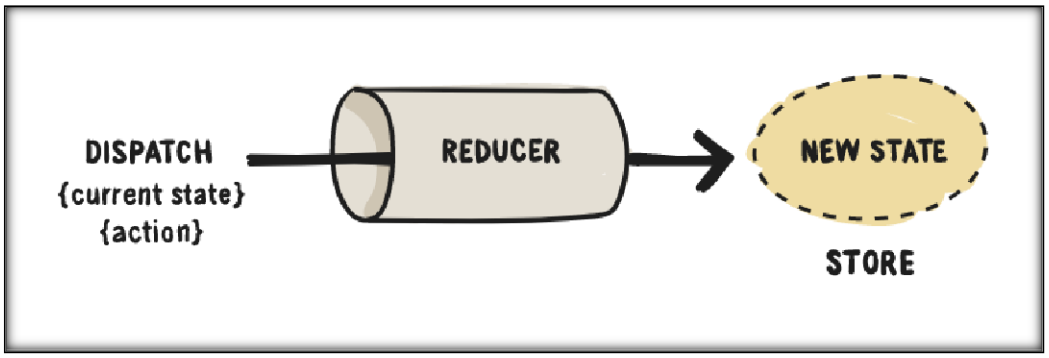
c. Reducers

Le Reducer, c'est le cerveau de Redux.

Un Reducer Redux est une fonction qui reçoit le state et une action en paramètre, et qui retourne un nouveau state.



Les réducteurs sont le seul moyen de changer d'état dans Redux. C'est le seul endroit où vous pouvez écrire de la logique et des calculs.



Exemple 1 :

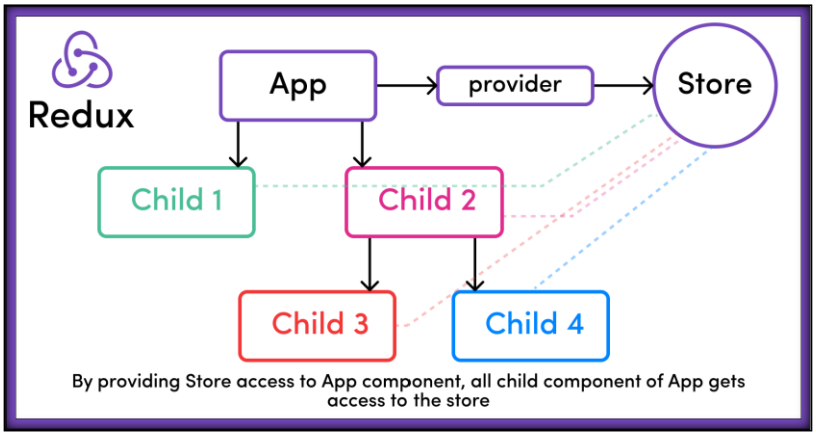
```
let initial_state = {
  counter: 10,
  step: 1,
};

const ReducerCounter = (state = initial_state, action) => {
  let newState = { ...state };
  switch (action.type) {
    case "INCREMENTER":
      newState.counter = newState.counter + newState.step;
      break;
    case "DECREMENTER":
      newState.counter = newState.counter - newState.step;
      break;
    case "INITIALISER":
      newState.counter = 0;
      break;
  }
  return newState;
}
```

d. Store

Le Store Redux rassemble l'état, les actions et les réducteurs qui composent votre application.

Le Store est le seau principal et central qui stocke tous les états d'une application. Il doit être considéré et maintenu comme une source unique de vérité pour l'état de l'application.



Si le store est fourni à App.js (**en enveloppant le composant App dans la balise <Provider> </Provider>**) comme indiqué dans l'extrait de code ci-dessous, tous ses enfants (composants enfants de App.js) peuvent également accéder à l'état de l'application à partir du magasin. Cela lui permet d'agir comme un état global.

```
const store = createStore(ReducerCounter)

root.render(
      <Provider store={store}>
            <App />
      </Provider>
);
```

La méthode getStore : Renvoie l'arbre d'état actuel de votre application. Il est égal à la dernière valeur retournée par le reducer du magasin.

e. useDispatch et useSelector

Dans react-redux, le hook **useDispatch** nous donne accès à la méthode dispatch de notre store. La méthode dispatch est utilisée pour envoyer des actions dans notre magasin redux et c'est le seul moyen d'affecter le magasin à partir d'un composant.

Le hook **useSelector** nous permet de lire des données du store.

Exemple :

```
import React from 'react'
import type { RootState } from '../../app/store'
import { useSelector, useDispatch } from 'react-redux'
import { decrement, increment } from './counterSlice'

export function Counter() {
  const count = useSelector(state => state.counter)
  const dispatch = useDispatch()

  return (
    <div>
      <div>
        <button
          aria-label="Increment value"
          onClick={() => dispatch(increment())}
        >
          Increment
        </button>
        <span>{count}</span>
        <button
          aria-label="Decrement value"
          onClick={() => dispatch(decrement())}
        >
          Decrement
        </button>
      </div>
    </div>
  )
}
```

IV. Flux de données d'application Redux

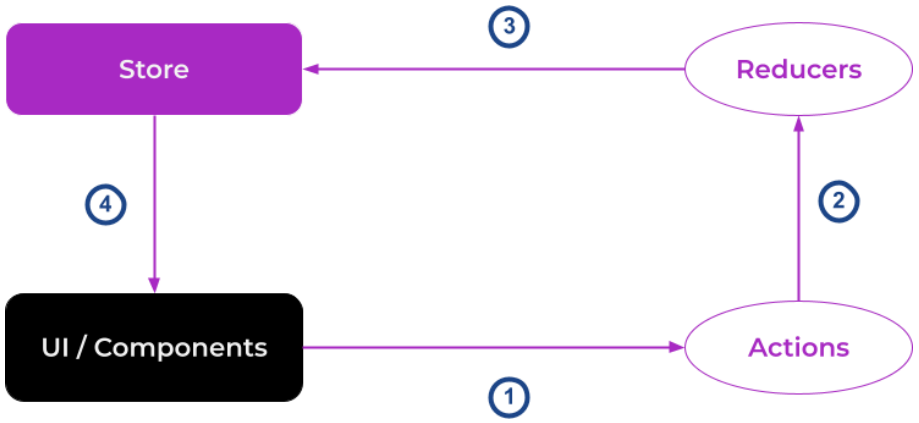
Quatre concepts fondamentaux régissent le flux de données dans les applications React-Redux.

Magasin Redux : Le magasin Redux, en termes simples, est un objet qui contient l'état de l'application. Un magasin redux peut être constitué de petits objets d'état qui sont combinés en un seul grand objet. Tout composant de l'application peut facilement accéder à cet état (magasin).

Créateurs d'action : les créateurs d'action, comme leur nom l'indique, sont des fonctions qui renvoient des actions (objets). Les créateurs d'action sont appelés lorsque l'utilisateur interagit avec l'application via son interface utilisateur (clic de bouton, soumission de formulaire, etc.) ou à certains moments du cycle de vie d'un composant (montage de composant, démontage de composant, etc.).

Actions : les actions sont des objets simples qui ont classiquement deux propriétés : le type et payload. La propriété type est généralement une chaine de caractère qui spécifie l'identification de l'action, et le champ payload utile est une propriété facultative qui contient certaines données requises pour effectuer une tâche particulière. La fonction principale de l'action est d'envoyer des données de l'application au magasin Redux.

Réducteurs : les réducteurs sont des fonctions pures qui mettent à jour l'état de l'application en réponse à des actions. Les réducteurs prennent un état précédent et une action comme entrée et renvoient une version modifiée de l'état. Comme l'état est immuable, un réducteur renvoie toujours un nouvel état, qui est une version mise à jour de l'état précédent.



1. Le flux de données dans une application React-Redux commence au niveau du composant lorsque l'utilisateur interagit avec l'interface utilisateur de l'application. Cette interaction conduit les créateurs d'action à envoyer une action.
2. Lorsqu'une action est envoyée, elle est reçue par le Reducer.

3. Ainsi, il devient la tâche du réducteur de déterminer s'il doit mettre à jour l'état en fonction de l'action envoyée. Ceci est vérifié en utilisant une simple instruction switch pour filtrer les actions requises. Il est essentiel de noter ici que l'état ne change jamais réellement dans redux. Au lieu de cela, le réducteur génère toujours un nouvel état qui est une copie de l'ancien état, mais avec quelques modifications.
4. Le magasin informe ensuite le composant du nouvel état qui à son tour récupère l'état mis à jour et restitue le composant. Une autre observation importante ici est que le flux de données dans une application React-Redux est unidirectionnel, c'est-à-dire qu'il ne va que dans une seule direction.