4 minutes

useRef() et le DOM

Manipuler le DOM avec des références

React met automatiquement à jour le DOM pour correspondre au JSX retourné par vos composants. Donc la plupart du temps vous n'avez pas besoin de manipuler directement le DOM.

Cependant, dans quelques cas, vous devez obligatoirement manipuler le DOM, par exemple pour focus un élément HTML, défiler jusqu'à l'affichage d'un élément, ou encore pour mesurer la taille ou la position d'un élément.

Dans ces cas, **vous devrez utiliser `useRef()` et l'attribut JSX `ref`**.

Pour obtenir la référence d'un élément du DOM il faut utiliser cette syntaxe :

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

const maRef = useRef(null);

<div ref={maRef}>
```

Copier

Notez bien l'attribut `ref` que nous lions à l'objet `ref` retourné par le `hook useRef()`.

Le nœud du DOM sera placé dans la propriété `current`.

Une fois l'accès à un élément du DOM effectué, vous pouvez le manipuler, par exemple :

```
maRef.current.scrollIntoView();

maRef.current.focus();
```

Copier

Donner le focus à un élément du DOM

Voici un exemple simple pour donner le focus à un élément :

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

export default function Form() {
  const inputRef = useRef(null);

  function handleClick() {
    inputRef.current.focus();
  }

  return (
    <>
```

```
      <input ref={inputRef} />
      <button onClick={handleClick}>
        Focus le champ
      </button>
    </>
  );
}
```

Copier

Retrouvez cet exemple sur Stackblitz en fin de leçon.

Utiliser une fonction de rappel pour manipuler les listes d'éléments

Parfois, vous voudrez obtenir des références à tous les éléments d'une liste, par exemple pour pouvoir défiler jusqu'à eux.

Dans ce cas, il est possible de passer une fonction de rappel à l'attribut `ref`.

Prenons un exemple :

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

export default function CatFriends() {
  const itemsRef = useRef(new Map());

  function scrollToId(id) {
    const node = itemsRef.current.get(id);
    node.scrollIntoView({
      behavior: 'smooth',
      block: 'nearest',
      inline: 'center',
    });
  }

  return (
    <>
      <nav>
        <button className="btn btn-primary m-20" onClick={() => scrollToId(0)}>
          Chat 1
        </button>
        <button className="btn btn-primary mr-15" onClick={() => scrollToId(5)}>
          Chat 6
        </button>
        <button className="btn btn-primary mr-15" onClick={() => scrollToId(9)}>
          Chat 10
        </button>
      </nav>
      <div>
        <ul className="d-flex" style={{ overflowX: 'scroll' }}>
          {catList.map((cat) => (
            <li
              style={{ minWidth: '300px' }}
              key={cat.id}
              ref={(node) => {
                itemsRef.current.set(cat.id, node);
              }}
```

```
            >
              <img src={cat.imageUrl} />
            </li>
          ))}
        </ul>
      </div>
    </>
  );
}

const catList = [];
for (let i = 0; i < 10; i++) {
  catList.push({
    id: i,
    imageUrl: 'https://placekitten.com/250/200?image=' + i,
  });
}
```

Copier

Nous allons détailler :

`const itemsRef = useRef(new Map())` : nous mettons un `Map` dans notre référence. Rappelez-vous qu'une référence peut contenir tout type de valeur.

La fonction `scrollToId(id)` permet de récupérer l'élément du `Map` qui a pour clé `id` et d'utiliser la valeur (qui est le nœud `HTML`) pour défiler jusqu'à lui.

La partie qui nous intéresse le plus est :

```
<ul className="d-flex" style={{ overflowX: 'scroll' }}>
  {catList.map((cat) => (
    <li
      style={{ minWidth: '300px' }}
      key={cat.id}
      ref={(node) => {
        itemsRef.current.set(cat.id, node);
      }}
    >
      <img src={cat.imageUrl} />
    </li>
  ))}
</ul>
```

Copier

Pour chaque élément de la liste de chats qui contient des objets du type `{id: 1, imageUrl: 'url'}`, nous créons un élément de liste.

Notez bien la fonction de rappel passée à l'attribut `JSX` `ref`. Cette fonction reçoit en argument le nœud `HTML` de l'itération en cours.

Nous utilisons ensuite notre `Map` contenu dans `itemsRef.current` pour définir en clé l'`id` du chat et en valeur le nœud (qui est l'élément de liste courant).

Grâce à cela nous pouvons ensuite accéder au nœud `HTML` de chaque élément en utilisant `itemsRef.current.get(id)`.