Rapport sur les Hooks en React Native

(Armelio de Campos)

Introduction aux Hooks

Les **hooks** sont une fonctionnalité introduite dans React 16.8 qui permet aux développeurs d'utiliser l'état et d'autres fonctionnalités de React sans avoir à créer des classes. Ils permettent de simplifier le code, de le rendre plus lisible et de réutiliser la logique d'état entre plusieurs composants. Les hooks commencent par le mot "use", ce qui est une convention pour indiquer qu'ils sont utilisés dans des composants fonctionnels.

Importance des Hooks

Les hooks sont essentiels pour plusieurs raisons :

- **Simplicité et lisibilité** : Les hooks permettent d'écrire des composants fonctionnels sans la complexité associée aux classes, ce qui rend le code plus facile à comprendre et à maintenir.
- **Réutilisation de la logique** : Ils facilitent le partage de la logique d'état entre différents composants sans avoir à modifier la hiérarchie des composants.
- **Gestion des effets secondaires** : Les hooks comme `useEffect` remplacent les méthodes de cycle de vie des classes, permettant ainsi une gestion plus efficace des effets secondaires.

Hooks Prédéfinis

Voici quelques hooks prédéfinis couramment utilisés dans React Native :

useState

- Permet de déclarer un état local dans un composant fonctionnel.
- Retourne un tableau avec la valeur actuelle de l'état et une fonction pour mettre à jour cette valeur.
- **Exemple :**

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

useEffect

- Utilisé pour gérer les effets secondaires dans les composants.
- Remplace les méthodes de cycle de vie comme `componentDidMount` et `componentDidUpdate`.
- S'exécute après chaque rendu par défaut, mais peut être configuré pour ne s'exécuter qu'en fonction de certaines dépendances.
- **Exemple :**

```
useEffect(() => {

}, [dependencies]);
```

useMemo

- Permet de mémoriser une valeur calculée afin d'optimiser les performances.
- Ne recalculera la valeur que si l'une des dépendances change.
- **Exemple :**

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

useRef

- Utilisé pour créer une référence mutable qui persiste pendant toute la durée de vie du composant.
- Ne déclenche pas un nouveau rendu lorsque sa valeur change.
- **Exemple :**

```
const inputRef = useRef(null);
```

useCallback

- Renvoie une version mémorisée d'une fonction qui ne change que si l'une des dépendances a changé.
- Utile pour optimiser les performances lors du passage de fonctions en tant que props.
- **Exemple :**

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

Hooks Personnalisés

En plus des hooks prédéfinis, il est possible de créer des **hooks personnalisés** pour encapsuler la logique réutilisable. Cela permet d'organiser le code et d'améliorer sa modularité. Par exemple, un hook personnalisé pourrait gérer la logique d'un formulaire ou récupérer des données depuis une API.

Conclusion

Les hooks en React Native représentent une avancée significative dans la manière dont les développeurs peuvent gérer l'état et les effets secondaires dans leurs applications. En favorisant l'utilisation de composants fonctionnels, ils rendent le code plus propre et plus maintenable tout en offrant une flexibilité accrue grâce à la possibilité de créer des hooks personnalisés.