Présentation des signaux

Temps de lecture : 5 minutes

Pourquoi les signaux ?

Pour développer une interface utilisateur (`UI`) complexe, les développeurs d'applications `JavaScript` doivent stocker, calculer et synchroniser l'état vers la couche de vue de l'application de manière efficace.

Les `UI` impliquent souvent plus que la simple gestion de valeurs simples, mais souvent le rendu d'un état calculé dépendant d'un arbre complexe d'autres valeurs ou d'états également calculés.

L'objectif des signaux est de fournir une infrastructure pour gérer cet état d'application afin que les développeurs puissent se concentrer sur la logique métier plutôt que sur ces détails répétitifs.

Les signaux sont utilisés dans la programmation réactive pour éliminer le besoin de gérer les mises à jour dans les applications, en offrant un modèle de programmation déclaratif basé sur les changements d'état.

En d'autre terme, les signaux **permettent d'améliorer les performances et de simplifier la réactivité.**

Voici l'ensemble des objectifs des signaux exprimés par l'équipe d' `Angular` :

- Modèle clair et unifié pour le flux de données dans une application.
- Support intégré pour l'état dérivé déclaratif.
- Synchronisation des parties de l'UI qui nécessitent une mise à jour, avec une granularité au niveau des composants individuels.
- Interopérabilité avec des bibliothèques réactives telles que `RxJS` .
- Meilleure gestion pour éviter les erreurs courantes de performance de la détection des changements et les erreurs telles que `ExpressionChangedAfterItHasBeenChecked` .
- Pouvoir créer des applications entièrement sans zone (`zoneless`), éliminant les problématiques de `zone.js` .
- Simplification de nombreux concepts du framework, tels que les requêtes et les `hooks` de cycle de vie.

Qu'est-ce qu'un signal ?

Les signaux sont des valeurs réactives qui permettent de notifier des consommateurs lorsque leur valeur change.

Les signaux peuvent contenir n'importe quelle valeur, des primitives aux structures de données complexes.

On lit la valeur d'un signal en appelant sa fonction getter, ce qui permet à `Angular` de suivre où le signal est utilisé.

Les signaux peuvent être soit modifiables, soit en lecture seule.

Nous allons voir une brève introduction générale des signaux puis rentrer en détail dans leur utilisation dans les leçons suivantes.

Création et utilisation des signaux

Création d'un signal

Pour créer un signal, on utilise la fonction `signal` en lui fournissant une valeur initiale. La valeur actuelle du signal est obtenue en l'appelant comme une fonction.

Pour modifier cette valeur, si le signal est modifiable, les méthodes `.set()` ou `.update()` sont disponibles.

```
import { signal } from '@angular/core';

const compteur = signal(0); // Création d'un signal avec une val
eur initiale de 0

console.log(compteur()); // Affiche la valeur actuelle : 0
compteur.set(1);         // Met à jour la valeur à 1
console.log(compteur()); // Affiche la nouvelle valeur : 1
```

Signaux dérivés

Les signaux dérivés, créés avec la fonction `computed`, permettent de générer une nouvelle valeur basée sur d'autres signaux.

Ils sont recalculés automatiquement lorsque leurs dépendances changent.

```
import { signal, computed } from '@angular/core';

const largeur = signal(5);
const hauteur = signal(10);
const aire = computed(() => largeur() * hauteur()); // Signal dé
```

```
rivé calculant l'aire

console.log(aire()); // Affiche 50
largeur.set(7);
console.log(aire()); // Affiche 70 après mise à jour de la large
ur
```

Effets

Les effets, définis via la fonction `effect`, exécutent une fonction chaque fois que les signaux dont ils dépendent changent.

Ils sont utiles pour des opérations telles que la journalisation ou la synchronisation avec des API externes.

```
import { signal, effect } from '@angular/core';

const message = signal('Bonjour');

effect(() => {
  console.log(`Message mis à jour : ${message()}`);
});

message.set('Bonsoir'); // Déclenche l'effet et affiche "Message
mis à jour : Bonsoir"
```

Synchroniser les signaux avec des sources externes

Le `linkedSignal` est une fonctionnalité qui permet de lier un signal à une source externe.

Cela signifie que le signal est automatiquement mis à jour lorsqu'une fonction ou une source de données change. Il est utile pour synchroniser l'état local avec des propriétés calculées ou d'autres états réactifs.

Nous le verrons plus en détail dans le chapitre.

Gérer des données asynchrones

La fonction `resource` d'`Angular` est conçue pour gérer des données asynchrones, comme des appels `API` ou des fichiers chargés dynamiquement.

Elle offre un moyen de suivre l'état de la ressource (chargement, réussite ou échec) et d'intégrer facilement ces états dans les templates.

Une `resource` est créée à l'aide de la fonction `resource` et peut écouter un signal ou une fonction asynchrone.

Même chose, nous la verrons dans ce chapitre.