Introduction aux requêtes HTTP (AJAX)

Temps de lecture : 3 minutes

Rappels sur l'`API fetch`

Il vaut mieux revoir les chapitres 14 et 15 de la formation `JavaScript` si vous ne connaissez pas `fetch`, cependant nous allons voir un bref rappel pour vous rafraichir la mémoire si vous connaissez déjà.

L'`API Fetch` est une interface `JavaScript` basée sur les promesses qui permet d'effectuer des requêtes `HTTP` de manière simple et flexible. Elle est utilisée pour récupérer des ressources (comme des fichiers ou des données `JSON`) via le réseau. Introduite en remplacement de l'ancienne méthode `XMLHttpRequest`, elle offre une syntaxe plus intuitive et des capacités étendues.

Voici un exemple d'utilisation :

```
fetch('https://api.example.com/data')
  .then((response) => {
    if (!response.ok) {
      throw new Error(`HTTP error! Status: ${response.status}`);
    }
    return response.json();
  })
  .then((data) => {
    console.log(data);
  })
  .catch((error) => {
    console.error('Erreur lors de la requête :', error);
  });
```

Syntaxe de `fetch()`

`fetch(url, options)` : fonction principale pour effectuer une requête.

- `url` : L'URL cible.
- `options` : Un objet de configuration (méthode, en-têtes, corps, etc.).

Les options disponibles sont :

- `method` : Spécifie la méthode `HTTP` utilisée (`GET`, `POST`, `PUT`, `DELETE`, etc.).
- `headers` : Définit les en-têtes `HTTP` comme `Content-Type` ou `Authorization`.
- `body` : Contient le corps de la requête (`JSON`, `FormData`, `Blob`, etc.), principalement utilisé avec `POST`, `PUT` ou `PATCH`.
- `mode` : Contrôle le mode de requête inter-origines (`cors`, `no-cors`, `same-origin`).
- `credentials` : Gère l'inclusion des informations d'identification (`same-origin`, `include`, `omit`).
- `cache` : Détermine le comportement de mise en cache (`default`, `no-cache`, `reload`, etc.).
- `redirect` : Spécifie comment gérer les redirections `HTTP` (`follow`, `error`, `manual`).
- `referrer` : Définit l'URL de référence envoyée avec la requête (`about:client`, une URL ou vide).
- `referrerPolicy` : Définit la politique d'envoi de l'URL de référence (`no-referrer`, `origin`, etc.).
- `integrity` : Permet de vérifier l'intégrité de la ressource à l'aide d'une somme de contrôle (par exemple, `SHA-256`).
- `keepalive` : Garde la requête active après le déchargement de la page (utile pour l'analyse ou le suivi).
- `signal` : Permet d'annuler une requête à l'aide d'un `AbortController`.
- `window` : Spécifie le contexte de la fenêtre, généralement non utilisé (valeur par défaut : `null`).

Voici un exemple :

```
fetch('https://api.example.com/data', {
  method: 'POST',
  headers: {
    'Content-Type': 'application/json',
    Authorization: 'Bearer token123',
  },
  body: JSON.stringify({ name: 'Alice', age: 25 }),
  mode: 'cors',
  credentials: 'include',
  cache: 'no-cache',
  redirect: 'follow',
  referrer: 'https://example.com/',
  referrerPolicy: 'no-referrer',
})
  .then((response) => response.json())
```

```
    .then((data) => console.log(data))
    .catch((error) => console.error('Erreur :', error));
```

Propriétés et méthodes de l'objet `Response`

- `.json()` : Convertit la réponse au format JSON.
- `.text()` : Convertit la réponse en chaîne brute.
- `.blob()` : Récupère la réponse en tant que Blob (ex. : fichiers).
- `.ok` : Boolean indiquant si le statut HTTP est compris entre 200 et 299.
- `.status` : Le code de statut HTTP de la réponse (ex. : 200, 404).
- `.headers` : Un objet représentant les en-têtes de réponse.

Introduction à l'`API resource`

Avec `Angular 19`, l'`API resource` a été introduite pour simplifier et améliorer la gestion des données asynchrones dans les applications.

Elle permet de gérer des requêtes `HTTP` ou tout autre type de chargement asynchrone tout en s'intégrant parfaitement dans l'approche réactive d'`Angular` basée sur les signaux.

Pourquoi utiliser les `resources` ?

1. **Réactivité et intégration native** : Les `resources` s'intègrent directement dans le système des signaux d'`Angular`, facilitant ainsi la gestion des changements d'état et des mises à jour d'interface utilisateur en réponse aux données.
2. **Simplification de la gestion asynchrone** : En encapsulant les requêtes dans une `API` unifiée, les `resources` offrent une approche cohérente pour traiter les états de chargement, les erreurs et les valeurs obtenues.
3. **Contrôle avancé** : L'`API` permet de définir facilement des actions comme le rechargement des données ou le traitement conditionnel des erreurs.

Fonctionnalités principales

- **Chargement réactif** : Les `resources` permettent de déclencher automatiquement des requêtes asynchrones basées sur des signaux, réduisant la complexité du code.
- **Gestion des états** : Elles fournissent des signaux pour suivre si une requête est en cours (`isLoading`), si une erreur est survenue (`error`) ou si une valeur valide est disponible (`value`).
- **Flexibilité** : Vous pouvez personnaliser les requêtes et les loaders pour répondre aux besoins spécifiques de votre application, qu'il s'agisse d'appels `API` simples ou de flux

de données complexes.