4 minutes

Les méthodes des promesses

Les méthodes de l'objet `Promise`

Pour le moment, nous avons étudié les méthodes `then()`, `catch()` et `finally()` qui sont situés sur le prototype de l'objet `Promise` (nous étudierons les prototypes en détails dans un prochain chapitre).

Il existe cinq méthodes sur l'objet `Promise` que nous allons étudier une par une.

Raccourci syntaxique avec `Promise.reject()` et `Promise.resolve()`

Ces méthodes permettent de créer directement un objet `Promise` acquitté : soit tenue avec `resolve()`, soit rompue avec `reject()`.

Ainsi les deux expressions suivantes sont équivalentes :

```
Promise.resolve(2);
new Promise(resolve => resolve(2));
```

De même :

```
Promise.reject(2);
new Promise((resolve, reject) => reject(2));
```

Elles peuvent être utiles lorsqu'une `API` a besoin d'utiliser une promesse mais que vous pouvez obtenir la valeur de manière synchrone (par exemple si vous l'avez mise en cache).

La méthode `Promise.all()`

Cette méthode est très importante et souvent utile. Elle permet **d'effectuer des traitements asynchrones en parallèle et d'attendre que tous ces traitements soient terminés pour retourner une valeur**.

La méthode prend en argument un tableau de promesses et va attendre soit qu'une des promesses échoue, soit que toutes les promesses soient résolues.

Elle retourne une nouvelle promesse et il est donc possible de chaîner avec `then()`, `catch()` ou `finally()`.

Si une promesse est rejetée, elle n'attend pas les autres promesses et renvoie immédiatement une erreur.

Si toutes les promesses sont tenues, **elle renvoie un tableau de résultats qui contient les valeurs retournées par les promesses dans le même ordre que la déclaration des promesses** (et non pas dans l'ordre d'arrivée des valeurs).

Par exemple :

```
Promise.all([
  new Promise(resolve => setTimeout(() => resolve(1), 3000)),
  new Promise(resolve => setTimeout(() => resolve(2), 2000)),
  new Promise(resolve => setTimeout(() => resolve(3), 1000))
]).then(alert);
```

Voici un exemple :
https://codesandbox.io/embed/js-c14-l4-1-vbkkx

La méthode `Promise.allSettled()`

Nous avons vu qu'avec la méthode `Promise.all()` si une erreur se produit et une promesse est rejetée, alors toutes les autres promesses sont ignorées.

Parfois, nous souhaitons avoir le résultat de toutes les promesses y compris si une ou plusieurs ont échoué.

La méthode `Promise.allSettled()` renvoie une promesse qui est résolue une fois que l'ensemble des promesses passée en argument sont réussies ou rejetées.

Cette méthode prend également un tableau de promesse.

Cette méthode retourne un tableau de résultats contenant des objets.

Pour les promesses tenues, l'objet est de la forme `{status:"fulfilled", value:42}`.

Pour les promesses rejetée, l'objet est de la forme `{status:"rejected", reason: "erreur..."}`.

Voici un exemple :
https://codesandbox.io/embed/js-c14-l4-2-kvudu

La méthode `Promise.race()`

La méthode `Promise.race()` renvoie une promesse qui est résolue ou rejetée dès qu'une des promesses passée en argument est résolue ou rejetée.

Elle prend en argument un tableau de promesses comme pour les autres méthodes.

Cette méthode fait la course entre les promesses, d'où son nom !

Elle retourne uniquement le résultat de la promesse la plus rapide à être acquitté.

```
Promise.race([
  new Promise((resolve, reject) => setTimeout(() => resolve(1), 1000)),
  new Promise((resolve, reject) => setTimeout(() => reject(new Error("Aïe!")),
3000)),
  new Promise((resolve, reject) => setTimeout(() => resolve(3), 500))
]).then(res => console.log(res)); // 3
```

Attention :

```
Promise.race([
  new Promise((resolve, reject) => setTimeout(() => resolve(1), 1000)),
  new Promise((resolve, reject) => setTimeout(() => reject(new Error("Aïe!")), 500)),
  new Promise((resolve, reject) => setTimeout(() => resolve(3), 700))
]).then(res => console.log(res)); // rien
```

La raison est que la première promesse acquittée est la seconde, qui est rejetée. Comme elle est rejetée et que nous n'avons pas passé de deuxième argument à `then()` ou n'avons pas mis de `catch()` nous ne récupérons pas la valeur.

Il faut faire :

```
Promise.race([
  new Promise((resolve, reject) => setTimeout(() => resolve(1), 1000)),
  new Promise((resolve, reject) => setTimeout(() => reject(new Error("Aïe!")), 500)),
  new Promise((resolve, reject) => setTimeout(() => resolve(3), 700))
]).then(res => console.log(res)).catch(res => console.log(res)) ; // Aïe!
```

Vous pouvez bien tester avec cet exemple :
https://codesandbox.io/embed/js-c14-l4-3-21fpi