4 minutes

Chaîner les promesses

Chaîner des promesses

Toutes les méthodes que nous avons vues dans la leçon précédente **retournent une nouvelle promesse**.

Cela signifie que vous pouvez chaîner les promesses pour réaliser autant de traitements asynchrones que vous souhaitez à la suite.

```
new Promise((resolve, reject) => setTimeout(() => resolve(22), 3000))
.then(r => r * 2)
.then(r => r + 4)
```

La promesse initiale créée par `new Promise()` va être tenue avec `resolve()` au bout de 3 secondes et retourner 22.

La méthode `then()` suivante va ensuite récupérer la valeur, effectuer un traitement, et retourner une valeur **dans une nouvelle promesse**.

La méthode `then()` suivante va ensuite récupérer cette valeur et effectuer un traitement.

Vous pouvez faire des tests :
https://codesandbox.io/embed/js-c14-l3-1-2qovd

Attention à la manipulation de plusieurs chaînes de promesse

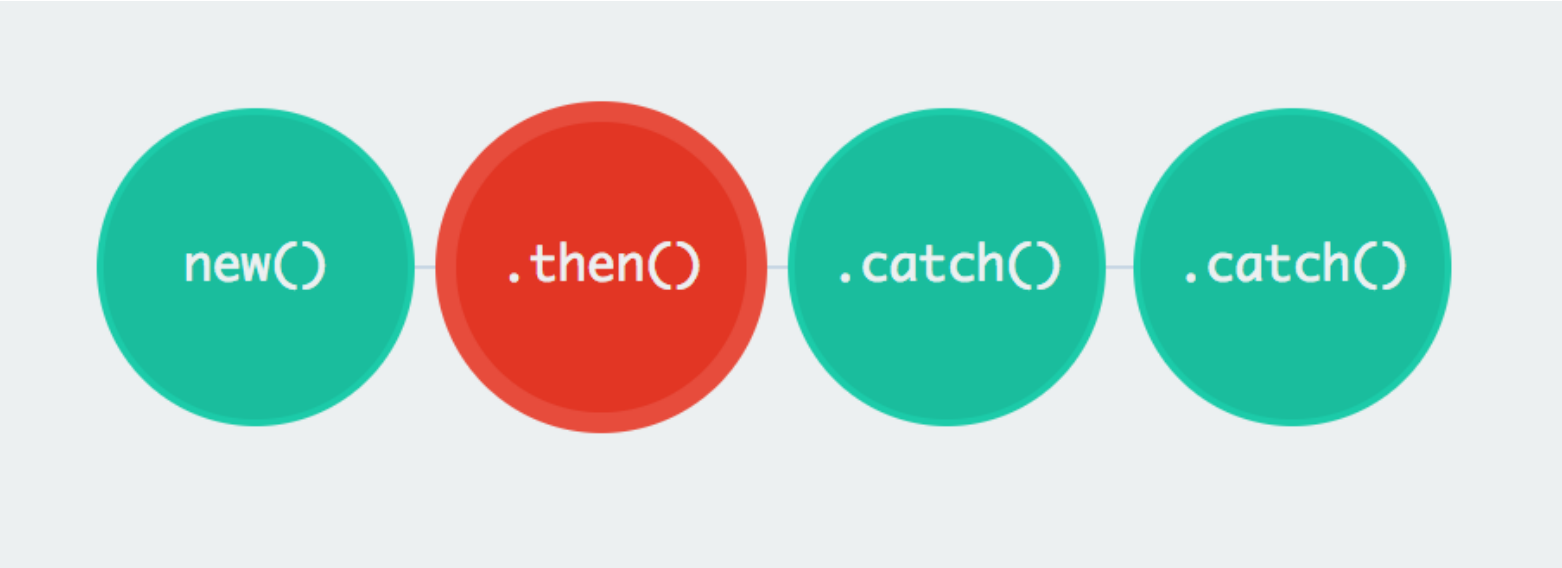
Il peut être un peu délicat de gérer les chaînes de promesses lorsque vous débutez. Nous allons prendre deux exemples pour bien comprendre.

Premier exemple

Nous allons créer une promesse et utiliser les méthodes `then()` et `catch()` :

```
new Promise(resolve => {
resolve(4);
})
.then(res => res.une.prop.qui.nexiste.pas)
.catch(err => console.error(err.message))
.catch(err => console.error(err.message));
```

Nous avons :



Nous créons une nouvelle promesse avec `new Promise()`, puis nous utilisons la méthode `then()` qui retourne une promesse, sur laquelle nous utilisons la méthode `catch()` qui retourne une promesse, sur laquelle nous utilisons la méthode `catch()`.

Autrement dit, nous avons simplement une unique chaîne de promesses.

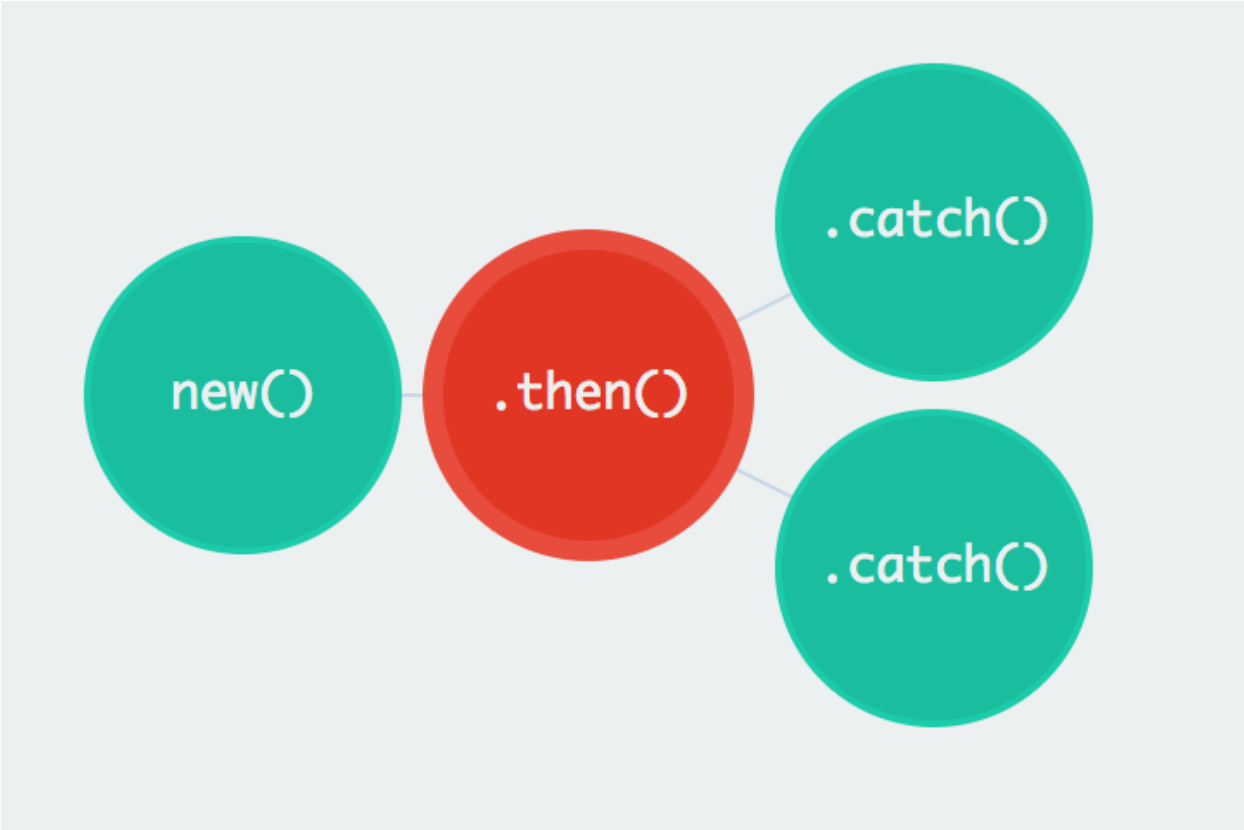
A noter que nous aurons un seul message d'erreur.

Deuxième exemple

Ici nous allons utiliser sur la même promesse plusieurs méthodes, mais dans les chaîner :

```
const p = new Promise(resolve => {
resolve(4);
}).then(res => res.une.prop.qui.nexiste.pas);
p.catch(err => console.error(err.message));
p.catch(err => console.error(err.message));
```

Ici nous avons pas la même chose :



Nous créons une promesse avec `new Promise()`, puis nous retournons une promesse avec la méthode `then()`. C'est cette dernière promesse qui est stockée dans la constante.

Sur la constante nous utilisons deux fois la méthode `catch()` mais nous ne les chaînons pas, le résultat est différent ! Nous aurons deux messages d'erreurs et non plus un seul !

Vous pouvez tester :
https://codesandbox.io/embed/js-c14-l3-2-pkmo3

Une erreur de débutant est de confondre chaînage de promesses (exemple 1) et utilisation de plusieurs méthodes `then()` ou `catch()` sur une même promesse (exemple 2).

Retourner des promesses dans les `then()`

Vous pouvez effectuer des traitements asynchrones dans les méthodes `then()` et `catch()`.

C'est à dire que vous pouvez par exemple retourner une nouvelle promesse, et dans ce cas la méthode va attendre qu'elle soit acquittée et retourner son résultat.

Voici un exemple :
https://codesandbox.io/embed/js-c14-l3-3-38ndm

La gestion des erreurs dans les chaînes de promesses

Il est simple de gérer les erreur sur une chaîne de promesse, il suffit de mettre un `catch()` à la fin pour que toutes les erreurs de la chaîne y soit transmises.

Par exemple, si une erreur survient dans la première promesse ou dans l'un des `then()` de la chaîne, la méthode `catch()` de la fin de chaîne récupérera l'erreur :

```
new Promise(resolve => {
setTimeout(() => resolve(22), 1000);
})
.then(res => /* code */)
.then(res => /* code */)
.then(res => /* code */)
.catch(err => console.error(err.message));
```

A noter que la **méthode `catch()` récupère l'erreur en amont dans la chaîne mais elle n'interrompt pas la chaîne** :

```
new Promise(resolve => {
setTimeout(() => resolve(22), 500);
})
.then(res => {
  nimp();
  return 22;
})
.catch(err => console.error(err.message))
.then(res => console.log('Je suis affiché'));
```