High level await

🪛Mise en place

Découvrons l'utilisation des promesses qui ont remplacé les fonctions callbacks.

Créez un fichier  index.html.

index.html 

```
1. <!DOCTYPE html>
2. <html lang="en">
3.
4. <head>
5.   <meta charset="UTF-8">
6.   <meta name="viewport" content="width=device-width, initial-scale=1.0">
7.   <title>Document</title>
8.   <script type="module" src="./module2.mjs"></script>
9. </head>
10. <body>
11. </body>
12. </html>
```

Créez un module  module1.mjs. Notez l'extension en mjs.

 module1.mjs

```
1. export async function asyncFunction1() {
2.     return new Promise((resolve) => {
3.         setTimeout(() => resolve('Data from asyncFunction1'), 2000);
```

```
    });
}
```

Créez un module module2.mjs

module2.mjs

```
import { asyncFunction1 } from './module1.mjs';

export async function asyncFunction2() {
    const data = await asyncFunction1();
    document.body.innerHTML = data;
}

asyncFunction2();
```

Dans module1.mjs, asyncFunction1 est une fonction asynchrone qui renvoie une promesse. Cette promesse se résout après 2 secondes avec la chaîne "Data from asyncFunction1".

Dans module2.mjs, nous importons *asyncFunction1* du module *module1.mjs*.

Nous définissons une autre fonction asynchrone, asyncFunction2. Dans cette fonction, nous utilisons le mot-clé await pour mettre en pause l'exécution de asyncFunction2 jusqu'à ce que la promesse renvoyée par asyncFunction1 soit résolue.

Une fois la promesse résolue, sa valeur est assignée à la variable data, puis affichée dans le DOM.

Enfin, nous appelons asyncFunction2. Comme il s'agit d'une fonction asynchrone, elle retourne immédiatement, et l'instruction console.log est exécutée une fois que la promesse de asyncFunction1 est résolue.

😈N'oubliez pas que await ne peut être utilisé qu'à l'intérieur d'une fonction asynchrone. Il fait attendre jusqu'à ce que la promesse renvoie un résultat. Il ne faut attendre que le bloc de la fonction asynchrone et non l'ensemble de l'exécution du programme.

🆒Amélioration

Définition : ([Top-level await · V8](#)) Top level await enables developers to use the await keyword outside of async functions. It acts like a big async function causing other modules who import them to wait before they start evaluating their body.

🔧Nous pouvons donc modifier le module1.mjs ainsi :

module1.mjs

```
1. async function asyncFunction1() {
2.   return new Promise((resolve) => {
```

```
3.      setTimeout(() => resolve("Data from asyncFunction1"), 2000);
4.    });
5.  }
6.  export const data = asyncFunction1();
```

Lig. 6 le module exporte une promesse data.

Nous modifions également le module2.mjs pour prendre en compte le nouvel export.

module2.mjs

```
1.  import { data } from "./module1.mjs";
2.
3.  document.body.innerHTML = await data;
```

Lig.3 le code est en attente du résultat de l'importation.

🥷Dans ce code, module1.mjs exporte une promesse data qui se résout avec le résultat de asyncFunction1. Dans module2.mjs, vous pouvez alors importer data et utiliser await au niveau supérieur pour attendre la résolution de la promesse.

Le résultat est finalement défini comme HTML interne du corps du document.

https://github.com/dupontdenis/top-level-await.git

🪛On pourrait également écrire

module1.mjs

```
1.  export async function asyncFunction1() {
```

```
2.    // Your async code here...
3.  }
4.
5.  // Await the result of asyncFunction1 at the top level
6.  export const data = await asyncFunction1();
```


module2.mjs

```
1.  import { data } from "./module1.mjs";
2.
3.  // data is already the resolved value of the promise
4.  const result = data;
5.  document.body.innerHTML = result;
```