Nodejs

Programmation asynchrone

S.gounane
ISIL 2020-2021

Introduction

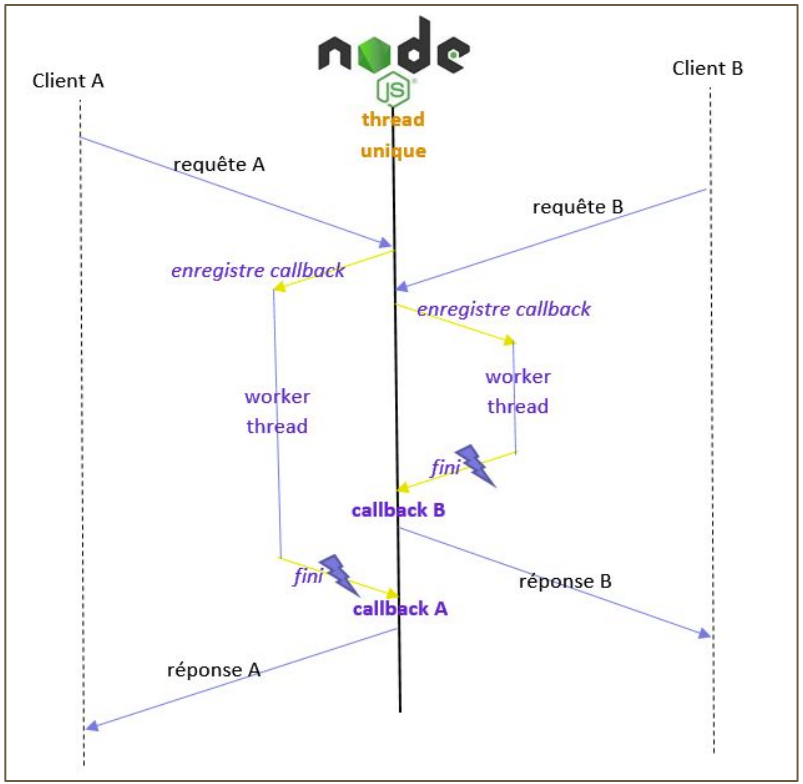
- Node.js est ***monothreadé*** : toutes les opérations sont exécutées dans un seul thread.
- Pour exécuter des opérations d'entrées-sorties bloquantes (requêtes HTTP, des lectures ou écritures de fichiers, des requêtes en base de données, etc.), Node.js s'appuie sur la "boucle d'événements" (***Event Loop***).
- Les opérations synchrones sont exécutées séquentiellement à l'aide de la "pile d'appels" (***Call Stack***). Cette pile est prioritaire, Il est donc essentiel de ne pas avoir des opérations qui monopolisent cette pile (un seul thread).
- Les opérations les plus ***pénalisantes*** ou ***bloquantes*** doivent donc être exécutées de manière ***asynchrone*** gérées par l'event loop qui communique avec le noyau du système d'exploitation hôte qui déléguer leurs l'exécution à des "threads système".
- Nodejs possède trois mécanismes: ***Callbacks***, ***Promesses*** et ***Asyn/Await***

Synchrone vs Asynchrone

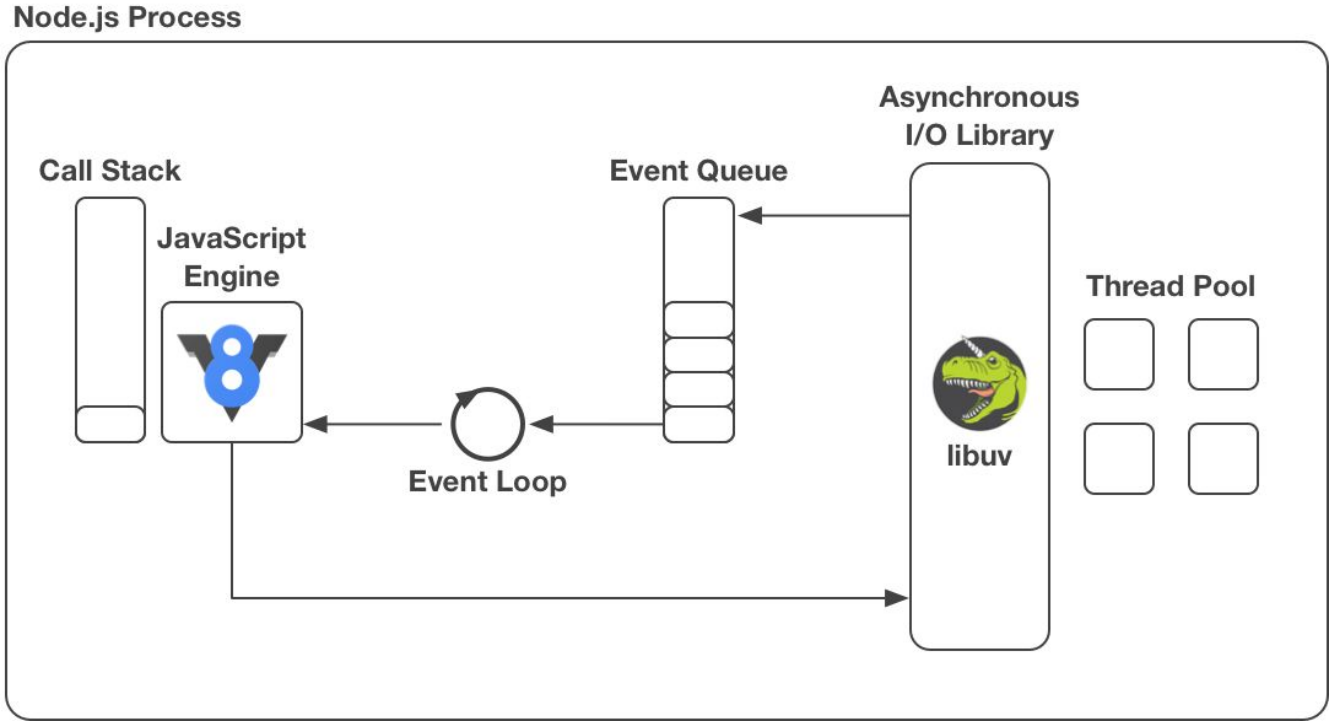
- Une tâche est dite ***synchrone*** si l'appelant de cette tâche doit attendre la fin de son exécution pour passer aux autres instructions.
- Une tâche est dite ***Asynchrone*** si l'appelant appelle l'exécution de cette tâche (à l'aide de **Libuv**) et passe immédiatement aux autres instructions. Une fois l'exécution de la tâche est terminée l'appelant sera ***notifier*** et une fonction (***callback***) ou une ***promesse*** est exécutée.
- Normalement les tâche lourdes (bloqueantes) doivent être exécutées de manière asynchrone

Principe

- Quand une opération asynchrone est terminée, le noyau émet un événement à destination de Node.js afin que le callback associée à l'opération soit ajoutée à la file des événements (event queue).
- Une callback est une fonction qui décrit le traitement à réaliser une fois que la réponse à l'opération asynchrone a été obtenue (éventuellement une erreur).
- Quand la call stack est vide, les callbacks présents dans la file d'événements peuvent être exécutés.

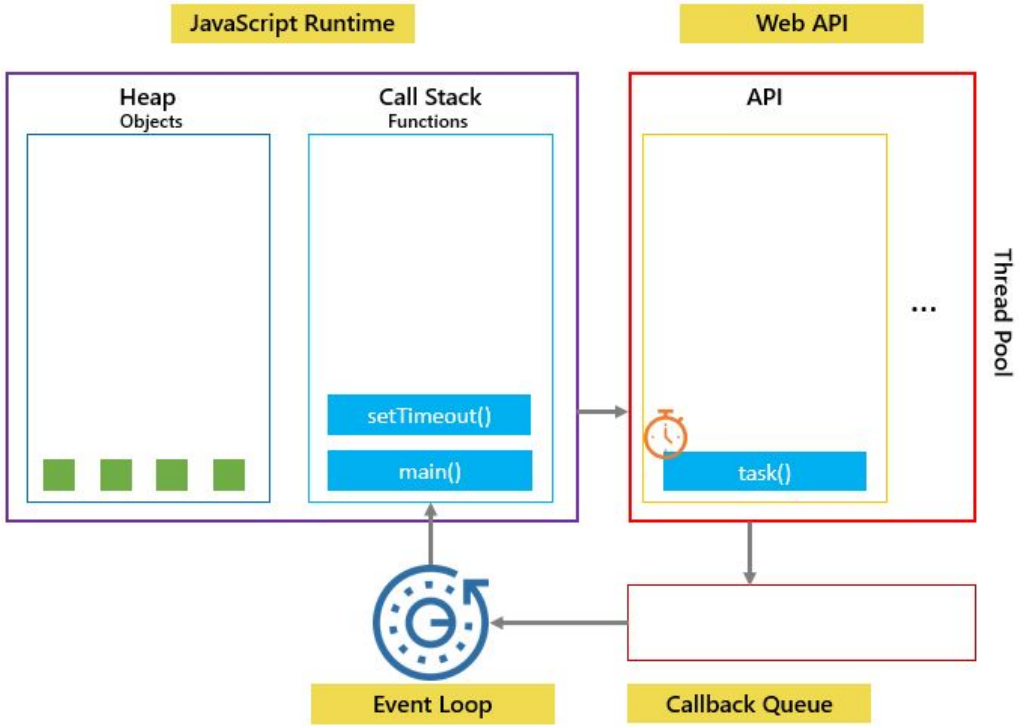


Principe



https://itnext.io/javascript-promises-with-node-js-e8ca827e0ea3

Principe



https://www.javascripttutorial.net/javascript-event-loop/

Les Callbacks

- Une fonction de callback est une fonction qui est transmise comme argument à une autre fonction pour qu'elle puisse par la suite être appelée sur certains événements ou circonstances.
- Un callbacks est utilisé pour s'assurer que le code est exécuté uniquement après la fin d'une opération asynchrone.
- Pendant longtemps, les callbacks étaient le mécanisme le plus commun pour écrire du code asynchrone, mais maintenant ils sont devenus obsolètes car ils peuvent rendre la lecture du code plus complexe. (callback hell)

Les Callbacks

Il existe de nombreuses façons d'utiliser les fonctions de callback dans une autre fonction. En général, ils ont cette structure :

*function **asynchronousTask**([Function Arguments], [**Callback Function**]) {*

    [Action]

}

Les Callbacks: Exemple

```
1   const request = require('request');
2
3   request('https://api.covid19api.com/countries', (error, response, body) => {
4       if (error) {
5           console.error(`Could not send request to API: ${error.message}`);
6           return;
7       }
8       if (response.statusCode != 200) {
9           console.error(`Expected status code 200 but received ${response.statusCode}.
10          return;
11      }
12      console.log('Processing list of Countries');
13      countries = JSON.parse(body);
14      countries.forEach(country => {
15          console.log(`${country['Country']}, ${country['Slug']}, ${country['ISO2']}`)
16      });
17  });
```

Les Callbacks: Exemple2

```
9       if (response.statusCode != 200) {
10          console.error(`Expected status code 200 but received ${response.statusCode}.
11          return;
12      }
13      console.log('Processing list of Countries');
14      countries = JSON.parse(body);
15      let str=''
16      countries.forEach(country => {
17          str+=`${country['Country']}, ${country['Slug']}, ${country['ISO2']}\n`;
18      });
19      fs.writeFile('coutries.csv', str, (error) => {
20          if (error) {
21              console.error(`Could not save the data to a file: ${error}`);
22              return;
23          }
24          console.log('data saved to countries.csv');;
25      });
26  });
```

Callback hell

Si on veut exécuter quatres opérations asynchrones, chacune ne peut s'exécuter que lorsqu'une autre est terminée:

```
function1(() => {
    function2(() => {
        function3(() => {
            function4(() => {
                // final action
            });
        });
    });
});
```

Callback hell

Exercice1:

Enregistrer les données covide de chaque pays dans un fichier csv

Utiliser les URL suivants:

- https://api.covid19api.com/countries : pour récupérer la liste des pays
- https://api.covid19api.com/country/:pays : Pour récupérer les données d'un pays

Les Promesses

- Une ***promesse*** est un ***objet JavaScript*** qui renvoie une valeur à un moment donné dans le ***futur***.
- Les fonctions ***asynchrones*** peuvent retourner des objets de promesse .
- Si on reçoit un ***résultat*** dans le futur, nous disons que la promesse a été tenue (***resolve***).
- Si on reçoit une ***erreur*** dans le futur, nous disons que la promesse a été rejetée (***reject***).
- Sinon, la promesse est toujours en cours de traitement dans un état d'attente.(***pending***)

Créer une Promesse

```
const promesse=new Promise((resolve, reject)=>{
    //tâche asynchrone à réaliser
    //appel resolve(resultat) si la promesse est tenue
    //sinon appel reject(erreur)
});
```

Créer une Promesse

```
1   function asyncFunc(a,b){
2       const promesse=new Promise((resolve, reject)=>{
3           if(b!=0) resolve(a/b);
4           else reject("div / zero error!!!") ;
5       });
6       return promesse;
7   }
8   asyncFunc(4,2)
9       .then((r)=>console.log("resultat:",r))
10      .catch((err)=> console.log(error));
11
12  asyncFunc(4,0)
13      .then((r)=>console.log("resultat:",r))
14      .catch((err)=> console.log(err));
```

Consommer une promesse

Syntax

promiseFunction()

.then([Callback Function for Fulfilled Promise])

.catch([Callback Function for Rejected Promise])

Consommer une promesse

```
1   const axios = require('axios');
2
3   axios.get('https://api.covid19ap.com/countries')
4       .then((response) => {
5           console.log('Successfully retrieved the countries list');
6           response.data.forEach(country => {
7               console.log(`${country['Country']}, ${country['Slug']}, ${country['ISO2']}`);
8           });
9       })
10      .catch((error)=> {
11          console.log(error);
12      });
```

Consommer une promesse

```
1  const axios = require('axios');
2  const fs = require('fs').promises;
3  axios.get('https://api.covid19api.com/countries')
4      .then((response) => {
5          console.log('Successfully retrieved the countries list');
6          let str=''
7          response.data.forEach(country => {
8              str+=`${country['Country']}, ${country['Slug']}, ${country['ISO2']}\n`;
9          });
10         return fs.writeFile('coutries.csv',str);
11     })
12     .then(()=> {
13         console.log("Data hase been saved ....")
14     })
15     .catch((error)=> {
16         console.log(error);
17     });
```

Consommer une promesse

Exercice:

En utilisant les promesse, enregistrer les données covide de chaque pays dans un fichier csv

Utiliser les URL suivants:

- https://api.covid19api.com/countries : pour récupérer la liste des pays
- https://api.covid19api.com/country/:pays : Pour récupérer les données d'un pays

Async/Await

- La déclaration **async *function*** et le mot clé **await** n’ajoutent pas de nouvelles fonctionnalités au langage
- permet de créer et d’utiliser des promesses avec un code plus intuitif, qui ressemble davantage à la syntaxe classique du JavaScript.
- Au lieu d'avoir le résultat d'une promesse disponible dans la méthode then(), le résultat est renvoyé sous forme de valeur comme dans toute autre fonction.

Async/Await

- On définit une fonction avec le mot-clé ***async*** pour dire à JavaScript qu'il s'agit d'une fonction asynchrone qui renvoie une promesse.
- On utilise le mot-clé ***await*** pour dire à JavaScript de retourner les résultats de la promesse au lieu de retourner la promesse elle-même.

  Syntax:

  async function myfunction() {

  await [Asynchronous Action]

  }

Async/Await

```
1   const axios = require('axios');
2
3   async function getCountries(){
4       const response=await axios.get('https://api.covid19api.com/countries');
5       console.log('Successfully retrieved the countries list');
6       response.data.forEach(country => {
7           console.log(`${country['Country']}, ${country['Slug']}, ${country['ISO2']}`);
8       });
9   }
10  getCountries();
```

Async/Await

```
const axios = require('axios');
const fs = require('fs').promises;

async function getCountries(){
    const response=await axios.get('https://api.covid19api.com/countries');
    console.log('Successfully retrieved the countries list');
    let str='';
    response.data.forEach(country => {
        str+=`${country['Country']}, ${country['Slug']}, ${country['ISO2']}\n`;
    });
    await fs.writeFile('coutries.csv',str);
    console.log("Data hase been saved ....");
}
getCountries();
```

Async/Await

```
const axios = require('axios');
const fs = require('fs').promises;

async function getCountries(){
    const response=await axios.get('https://api.covid19api.com/countries');
    console.log('Successfully retrieved the countries list');
    let str='';
    response.data.forEach(country => {
        str+=`${country['Country']}, ${country['Slug']}, ${country['ISO2']}\n`;
    });
    await fs.writeFile('coutries.csv',str);
    console.log("Data hase been saved ....");
}
getCountries();
```

Async/Await: try catch

```
const axios = require('axios');
const fs = require('fs').promises;

async function getCountries(){
    try{
        const response=await axios.get('https://api.covid19ap.com/countries');
        console.log('Successfully retrieved the countries list');
        let str='';
        response.data.forEach(country => {
            str+=`${country['Country']}, ${country['Slug']}, ${country['ISO2']}\n`;
        });
        await fs.writeFile('coutries.csv',str);
        console.log("Data hase been saved ....");
    }
    catch(error){
        console.log(error)
    }

}
getCountries();
```

Async/Await

Exercice:

En utilisant async/await, enregistrer les données covide de chaque pays dans un fichier csv

Utiliser les URL suivants:

- https://api.covid19api.com/countries : pour récupérer la liste des pays
- https://api.covid19api.com/country/:pays : Pour récupérer les données d'un pays