

Web Client - #1 (Partie 2)

[Promesses]

Cyril Rouyer

Sommaire

- Les difficultés liées aux appels asynchrones
- Les promesses
- Agrégation de promesses

Les difficultés liées aux appels asynchrones

- TD1 exercice 3 : un simple appel XHR ne permet pas de traiter les résultats en dehors de l'événement "load"
- Il devient alors difficile d'enchaîner plusieurs appels asynchrones en cascade (ex : d'abord les films, puis pour chaque film les acteurs principaux, ...)
- Les traitements multiples pour un même appel doivent être regroupés dans 1 seule callback
- Les appels multiples en parallèle sont difficiles à synchroniser (ex : n'afficher les acteurs des films trouvés qu'à partir du moment où ils ont tous, sans exception, été chargés)

Les promesses 1

- Une promesse est un **objet** de l'API standard JS : **Promise**
- Elle représente un appel asynchrone
- Elle a trois états :
 - **pending** : en attente d'un résultat
 - **fulfilled** : l'appel asynchrone s'est terminé avec succès et a retourné une valeur comme résultat de la promesse
 - **rejected** : l'appel asynchrone s'est terminé avec échec et a retourné une erreur comme résultat de la promesse

Les promesses 2

- La méthode **then(callback)** sur votre promesse permet d'enregistrer une callback qui sera exécutée quand la promesse atteindra l'état **fulfilled**.
- La méthode **catch(callback)** sur votre promesse permet d'enregistrer une callback qui sera exécutée si la promesse se termine en échec (état **rejected**)

Les promesses 3

then(callback)

- Sa **callback** reçoit en paramètre le résultat de l'appel asynchrone
- **then()** retourne elle-même une promesse. Cette promesse dépend de la **callback** passée en paramètre et de ce qu'elle retourne :
 - Soit la callback retourne une valeur (string, objet, ...) : la promesse issue de l'appel de **then()** sera alors fulfilled avec cette valeur
 - Soit elle renvoie une promesse : dans ce cas **then()** la retourne telle quelle.

Les promesses 4

/*

* On suppose que `send` envoie une requête HTTP vers l'API OMDB et

* retourne une promesse

*/

```
let pr1 = send('https://www.omdbapi.com/?apikey=xxxxxx&s=Oppenheimer')
```

//pr1 est une promesse

```
let pr2 = pr1.then(show_movies);
```

//pr2 est une promesse

/*

* `show_movies` sera appelée quand l'appel Ajax sera terminé, et que donc `pr1`

* sera fulfilled. Elle recevra en paramètre le résultat de l'appel HTTP.

* `pr2` est une promesse différente de `pr1`. Elle sera accomplie lorsque

* `show_movies` sera terminée, ou abandonnée si `show_movies` renvoie une erreur.

* Elle sera résolue avec la valeur de retour de `show_movies`.

*/

Les promesses 5

catch(callback)

- Sa **callback** reçoit en paramètre les informations sur l'erreur rencontrée durant l'appel asynchrone
- **catch()** retourne elle-même une promesse qui sera toujours rejected si la callback est appelée

Les promesses 6

```
let pr = send("https://www.omdbapi.com/?  
apikey=xxxxxx&s=Oppenheimer")
```

```
pr.then(show_movies)  
pr.catch((error) => {  
  console.log(error);  
})
```

Les promesses 7

- Comment on crée une promesse ?
- Promise propose un constructeur qui attend en paramètre une fonction dans laquelle, en théorie, du code asynchrone est exécuté
- Cette fonction attend deux paramètres : resolve, et reject, qui sont des fonctions de callback
- resolve doit être appelée en cas de succès, en lui fournissant en paramètre les résultats
- reject doit être appelée en cas d'erreur, avec l'erreur en paramètre

Les promesses 8

```
function query(maSuperCondition) {  
  return new Promise(function(resolve, reject) {  
    if(maSuperCondition === 'ok') {  
      resolve({success: true, message: "Tout roule !"})  
    }  
    else {  
      reject({success: false, message: "Une erreur est  
survenue"})  
    }  
  })  
}
```

Les promesses 9

- On peut aussi transformer n'importe quelle valeur en promesse fulfilled ou rejected
- `Promise.resolve(10).then(x => console.log(x));` //va afficher 10 dans la console
- `Promise.reject('error').catch(e => console.log(e));` //affiche 'error' dans la console.

Les promesses 10

- À quoi servent les promesses ?
- On peut enregistrer plusieurs callback sur la même promesse, cela permet d'avoir plusieurs traitements différents pour un même appel asynchrone
- On peut chaîner facilement des appels asynchrones successifs en chaînant les `.then()`, chaque callback va recevoir en paramètre le résultat de l'appel précédent
- On peut traiter les erreurs de façon globale. Dès qu'une erreur est produite, la callback du `catch` permet de la traiter

Les promesses 11

2 traitements sur le même appel asynchrone

```
let pr = getGHusers('https://api.github.com/users')  
pr.then(show_github_users)  
pr.then(count_github_users)
```

Ici, `show_github_users` affiche les utilisateur tandis que `count_github_users` se charge de les compter.

Les promesses 12

Enchaîner 2 appels successifs

```
let pr = getGHusers('https://api.github.com/users/wycats')  
pr.then(show_user)  
  .then(show_user_repos)  
  .catch(show_error)
```

Ici `show_user` va afficher les infos de l'utilisateur et va retourner une promesse relative à un nouvel appel Ajax permettant d'obtenir les repos de l'utilisateur.
`show_user_repos` va permettre d'afficher ces repos.

Agrégation de promesses 1

- **Promise.all([... promesses])**

Elle reçoit un tableau de promesses en paramètre et retourne une promesse qui est fulfilled lorsque toutes les promesses sont fulfilled, ou rejected dès qu'une promesse est rejected

- **Promise.any([... promesses])**

Elle retourne une promesse fulfilled dès qu'au moins une des promesses est fulfilled

- **Promise.race([... promesses])**

Elle retourne une promesse fulfilled dès qu'une promesse l'est aussi, ou rejected dès qu'une promesse du tableau l'est (c'est le premier qui gagne)

Agrégation de promesses 2

```
let user = query('https://api.github.com/users/mojombo');  
let repos = query('https://api.github.com/users/mojombo/  
repos');  
let follows = query('https://api.github.com/users/mojombo/  
followers');
```

```
Promise.all( [user , follows, repos ] ) .then(resp_array =>  
  console.log(resp_array)  
)  
.catch( error => console.log(error); )
```