Javascript - ajax

Auparavant utilisées via l'objet **XmlHttpRequest** des navigateurs, les requêtes partielles se font maintenant en utilisant l'api **fetch**.

Promise

Les promesses permettent d'écrire du code asynchrone, pour les méthodes qui nécessitent un temps d'exécution et ne peuvent retourner immédiatement un résultat.

Promise

Exemple : Chargement d'un script

```
function loadScript(src) {
  return new Promise(function(resolve, reject) {
    let script = document.createElement('script');
    script.src = src;

    script.onload = () => resolve(script);
    script.onerror = () => reject(new Error(`Script load error for ${src}`));

    document.head.append(script);
  });
}
```

- **resolve** permet d'indiquer que la promesse est résolue
- **reject** qu'elle n'a pas abouti

Utilisation

```
let promise =
loadScript("https://cdnjs.cloudflare.com/ajax/libs/lodash.js/4.17.11/lodash.js");

promise.then(
  script => alert(`${script.src} is loaded!`),
  error => alert(`Error: ${error.message}`)
);
```

async/await

L'utilisation de **async** et **await** permet d'utiliser les fonctions asynchrones sans obligation de chaîner les promesses (pas besoin de mettre le code dans le retour **then**).

```
function resolveAfter2Seconds() {
  return new Promise(resolve => {
    setTimeout(() => {
      resolve('resolved');
    }, 2000);
  });
}

async function asyncCall() {
  console.log('calling');
  const result = await resolveAfter2Seconds();
  console.log(result);
  // Expected output: "resolved"
}

asyncCall();
```

API Fetch

L'api fetch permet en javascript d'effectuer des requêtes partielles vers le serveur, sans changer de page, selon le mode AJAX (Asynchronous Javascript And XML).

feetch fonctionne selon la syntaxe suivante :

```
let promise = fetch(url, [options])
```

- Requête vers une **url** cible
- en passant un tableau d'**option**s (méthode, headers...)

Sans options, **fetch** effectue un simple **GET** téléchargeant le contenu.

Lors de l'utilisation de **fetch**, la requête est envoyée vers le serveur et renvoie une promise qui pourra être utilisée via un objet **Response**.

La réponse peut être analysée (via ses membres **status** ou **ok**, et on obtient son corps fourni sous la forme d'une seconde promesse :

```
let response = await fetch(url);

if (response.ok) { // Statut dans les 200
  // obtenir le corps de réponse au format JSON
  let json = await response.json();
} else {
  alert("HTTP-Error: " + response.status);
}
```

Response

Méthode	**Action**
response.headers()	Retourne les en-têtes de la réponse
response.text()	Retourne la réponse sous forme de texte

Méthode	Action
response.json()	Retourne la réponse au format JSON
response.formData()	Retourne un objet FormData
response.blob()	Retourne la réponse sous forme de blob (données binaires typées)

Exemples

Requête vers l'api Github pour obtenir la liste des messages de commits d'un repo :

Version **await** :

```
let url = 'https://api.github.com/repos/cnam-dist/nfa085-2023/commits';
let response = await fetch(url);

let commits = await response.json();

alert(commits[0].message);
```

Version **then**

```
fetch('https://api.github.com/repos/cnam-dist/nfa085-2023/commits')
  .then(response => response.json())
  .then(commits => alert(commits[0].message));
```

Chargement d'image :

```
let response = await fetch('/lib/tpl/bootstrap3/images/logo.png');

let blob = await response.blob(); // télécharger en tant qu'objet Blob

// create <img> for it
let img = document.createElement('img');
img.style = 'position:fixed;top:10px;left:10px;width:100px';
document.body.append(img);

// l'afficher
img.src = URL.createObjectURL(blob);

setTimeout(() => { // le cacher après 3 secondes
  img.remove();
  URL.revokeObjectURL(img.src);
}, 3000);
```

Test image loading