Curso de JavaScript avanzado

Promesas y asincronía

Uso de async/await

ÍNDICE

Funciones asíncronas

- En Javascript una función asíncrona es una función precedida de la palabra **async** que devuelve siempre una promesa
- Cualquier valor que se devuelva que no sea una promesa **será introducido en una promesa resuelta**
- Las funciones asíncronas no comparten el tipo **Function** sino que están tipadas como **AsyncFunction**

Funciones asíncronas

```
async function funcionAsync() {
  return Promise.resolve('Funciona!');
}

async function funcionAsync2() {
  return 'Funciona!';
}

let funcionAsync3= async () => { return 'Funciona!' };

funcionAsync.then(resp => console.log(resp))
// Esto sacará 'Funciona!'

funcionAsync2.then(resp => console.log(resp))
// Esto sacará 'Funciona!'
```

Uso con await

- La palabra clave **await** solo puede ser utilizada en funciones asíncronas definidas con la palabra **async**
- Esta palabra en conjunción con una promesa **congela la ejecución de la función hasta que la promesa se encuentre resuelta**
- Por lo tanto el uso de await también **puede realizarse con cualquier función que devuelva una promesa**

Uso con await

```
async function funcionAsync() {

  let promise = new Promise((resolve, reject) => {
    setTimeout(() => resolve('hecho!'), 1000)
  });

  let result = await promise; // La ejecución se parará aquí

  alert(result); // Saltará una alerta con 'hecho!'
}
```

Uso con await

```
funcionAsync();
setTimeout(() => alert('qué tal?'), 1100)
alert('hola');

// Esto sacará los diálogos en el orden:
// 'hola'
// 'hecho!'
// 'qué tal?'
```

- El hecho de que la ejecución en la función sea asíncrona **no afecta al flujo principal del programa**
- Se seguirán realizando todas las llamadas **hasta la resolución**

Reescribiendo con async/await

```
promise
.then(res => console.log(res * 2))
.then(res => console.log(res * 4))
.catch(error => console.error(error))
```

```
try {
  let res = noPromise() * 2;
  res = res * 4;
} catch (error) {
  console.error(error);
}
```

- Si reescribiéramos una promesa con encadenamiento de manera síncrona, **sería equivalente a una serie de instrucciones en un bloque try...catch**
- Con async/await **la ejecución será asíncrona** pero **el tratamiento dentro de dicha función será síncrono**

Reescribiendo con async/await

```
fetch('/user.json')
  .then(response => response.json())
  .then(user => fetch(`https://api.github.com/users/${user.name}`))
  .then(response => response.json())
  .then(githubUser => {
    let img = document.createElement('img');
    img.src = githubUser.avatar_url;
    img.className = "promise-avatar-example";
    document.body.append(img);

    setTimeout(() => img.remove(), 3000);
  })
  .catch(error => console.error(error.message))
```

Reescribiendo con async/await

```
async function peticion() {
  try {
    let response = await fetch('/user.json')
    let user = response.json();
    response = await fetch(`https://api.github.com/users/${user.name}`);
    let githubUser = response.json();

    let img = document.createElement('img')
    img.src = githubUser.avatar_url
    img.className = 'promise-avatar-example'
    document.body.append(img)
    setTimeout(() => img.remove(), 3000)
  } catch (error) {
    console.log(error.message);
  }
}
```

Reescribiendo con async/await

```
async function peticion() {
  let response = await fetch('/user.json')
  if (!response.ok) {
    throw new Error(`Error HTTP con status: ${response.status}`);
  } else {
    let user = response.json();
    response = await fetch(`https://api.github.com/users/${user.name}`);
    let githubUser = response.json();

    let img = document.createElement('img')
    img.src = githubUser.avatar_url
    img.className = 'promise-avatar-example'
    document.body.append(img)
    setTimeout(() => img.remove(), 3000)
  }
}

peticion().catch(error => console.error(error));
// Esto imprimirá el error lanzado si el status de la primera peticion no es "ok"
```

PARA RESUMIR

- Una función asíncrona en Javascript es una función de palabra clave **async que se ejecuta paralelamente y devuelve una promesa**
- En combinación con la palabra clave **await las funciones asíncronas pueden parar su ejecución hasta la resolución de sus tareas asíncronas**
- Es posible reescribir cualquier encadenamiento de promesas como **un flujo aparentemente síncrono en una función declarada async**