

8월 26일 (목)

1. Codecademy - Async JavaScript and HTTP Requests - Learn JavaScript Syntax : Async-Await

Article 1. Making Asynchronous Programming Easier with async and await

출처 : https://developer.mozilla.org/ko/docs/Learn/JavaScript/Asynchronous/Async_await

- async await 의 단점
- async await class 메서드
- 브라우저 지원
- 결론

- async await 의 단점

async/await의 단점

앞서 봤듯이 async/await은 매우 유용하지만 고려해야 할 몇 가지 단점이 있습니다.

Async/await 는 우리의 코드를 마치 동기식 코드처럼 보이게 합니다. 그리고 어떤 면에서는 정말로 동기적으로 행동합니다. 함수 블록에 여러 개의 `await` 키워드를 사용하면 Promise가 fulfilled되기 전 까지 다음 `await` 을 차단합니다. 그 동안 다른 태스크는 계속 실행이 되지만 정의한 함수 내에서는 동기적으로 작동할 것 입니다.

이 말은 우리가 작성한 코드가 바로 이어지는 수 많은 Promise에 의해 느려질 수 있다는 것을 의미합니다. 각 `await` 는 이전의 작업이 끝날 때 까지 기다립니다(Promise 체이닝과 혼동하지 마세요). 그런데 우리가 원하는건 기다리는게 아니고 일제히 실행되는 것 입니다.

이 문제를 완화할 수 있는 패턴이 있습니다. — 모든 `Promise` 오브젝트를 변수에 저장하여 미리 실행되게 하고 변수가 사용 가능할 때 꺼내서 쓰는 것 입니다. 어떻게 작동하는지 한번 살펴봅시다.

두 가지 예시를 보여 드리겠습니다. — 느린 비동기 작업 [slow-async-await.html](#) (소스 코드) 그리고 빠른 비동기 작업 [fast-async-await.html](#) (소스 코드)입니다. 두 예제에서 마치 비동기 작업인 것 처럼 보이기 위해 `setTimeout(.)` 을 사용했습니다. :

```

async function makeResult(items) {
  let newArr = [];
  for(let i=0; i < items.length; i++) {
    newArr.push('word_'+i);
  }
  return newArr;
}

async function getResult() {
  let result = await makeResult(items); // Blocked on this line
  useThatResult(result); // Will not be executed before makeResult() is done
}

```

그리고 세 가지 `timeoutPromise()` 함수를 호출하는 `timeTest()` 함수를 만들었습니다.

```

async function timeTest() {
  ...
}

```

그리고 두 개 예제 모두 시작 시간을 기록하고, `timeTest()` Promise가 fulfilled된 시간을 저장하여 두 시간의 차를 계산해 작업이 얼마나 걸렸는지 사용자에게 보여줍니다. :

```

let startTime = Date.now();
timeTest().then(() => {
  let finishTime = Date.now();
  let timeTaken = finishTime - startTime;
  alert("Time taken in milliseconds: " + timeTaken);
})

```

`timeTest()` 함수만 두 예제에서 차이가 있습니다.

`slow-async-await.html` 예제에서, `timeTest()` 함수는 아래와 같이 생겼습니다. :

```
async function timeTest() {  
  await timeoutPromise(3000);  
  await timeoutPromise(3000);  
  await timeoutPromise(3000);  
}
```

아주 간단하게 `timeoutPromise()` 함수를 직접 호출했습니다. 각 작업은 3초씩 걸립니다. 그리고 `await` 키워드를 사용했기 때문에 이전 `await` 작업이 끝나야 다음으로 진행됩니다. — 첫 번째 예제를 실행하면, `alert` 박스에서 약 9초 (9000밀리초)가 걸렸음을 확인할 수 있습니다.

다음으로 `fast-async-await.html` 예제에서, `timeTest()` 은 아래와 같이 생겼습니다. :

```
async function timeTest() {  
  const timeoutPromise1 = timeoutPromise(3000);  
  const timeoutPromise2 = timeoutPromise(3000);  
  const timeoutPromise3 = timeoutPromise(3000);  
  
  await timeoutPromise1;  
  await timeoutPromise2;  
  await timeoutPromise3;  
}
```

여기선 세 가지 `Promise` 오브젝트를 변수에 저장하여 동시에 작업을 시작하도록 했습니다.

그리고 그 변수에 `await`을 사용하여 결과를 호출합니다. — 작업이 거의 동시에 시작됐기 때문에, `Promise`도 거의 동시에 fulfilled될 것 입니다. 두 번째 예제를 실행하면 거의 3초(3000밀리초) 만에 작업이 끝났음을 확인할 수 있습니다.

코드를 주의깊게 테스트 하고, 성능이 떨어지기 시작하면 위의 상황을 의심해봐야 합니다.

다른 아주 사소한 단점은 비동기로 실행될 `Promise`가 있다면 `async`함수 안에 항상 `await`을 써야한다는 것 입니다.

- **async await class 메서드**

Async/await class 메서드

마지막으로 보여줄 내용은 `async` 키워드를 class/object의 메서드에 사용하여 Promise를 반환하게 만들 수 있다는 것입니다. 그리고 `await` 를 그 안에 넣을 수도 있습니다. 다음 문서를 살펴보세요 > [ES class code we saw in our object-oriented JavaScript article](#), 그리고 보이는 코드를 `async` 메서드로 수정한 아래의 내용과 비교 해보세요 :

```
class Person {
  constructor(first, last, age, gender, interests) {
    this.name = {
      first,
      last
    };
    this.age = age;
    this.gender = gender;
    this.interests = interests;
  }

  async greeting() {
    return await Promise.resolve(`Hi! I'm ${this.name.first}`);
  };

  farewell() {
    console.log(`${this.name.first} has left the building. Bye for now!`);
  };
}

let han = new Person('Han', 'Solo', 25, 'male', ['Smuggling']);
```

이제 클래스의 첫 번째 메서드를 아래와 같이 사용할 수 있습니다. :

```
han.greeting().then(console.log);
```

• 브라우저 지원

브라우저 지원

`async/await` 사용 여부를 결정할 때 고려해야 할 한가지 사항은 이전 브라우저에 대한 지원입니다. `promises`와 마찬가지로 대부분의 최신 브라우저에서 사용할 수 있습니다. 주요 지원 문제는 Internet Explorer 그리고 Opera Mini에서 발생합니다.

`async/await`을 사용하는데 브라우저 지원이 걱정되는 경우 [BabelJS](#) 라이브러리를 사용하는 것을 고려해 볼 수 있습니다. BabelJS는 최신 자바스크립트를 사용하여 애플리케이션을 작성하고 사용자 브라우저에 필요한 변경사항을 Babel이 파악할 수 있도록 지원합니다. `async/await`를 지원하지 않는 브라우저를 만나면 Babel은 이전 브라우저에서 작동하는 `polyfill`를 자동으로 제공합니다.

- 결론

결론

async/await를 사용하면 읽기 쉽고 유지보수가 편리한 비동기 코드를 간단하게 작성할 수 있습니다. 브라우저 지원이 다른 비동기 코드에 비해 제한적이기는 하지만 현재는 물론 미래에도 사용을 위해 배울 가치는 충분합니다.

Article 2. Choosing the Right Approach

- Asynchronous callbacks
- `setTimeout()`
- `setInterval()`
- `requestAnimationFrame()`
- Promises
- `Promise.all()`
- Async await

- Asynchronous callbacks

콜백함수가 유용한 상황은 다음과 같다.

Useful for...			
Single delayed operation	Repeating operation	Multiple sequential operations	Multiple simultaneous operations
No	Yes (recursive callbacks)	Yes (nested callbacks)	No

```
function loadAsset(url, type, callback) {
  let xhr = new XMLHttpRequest();
  xhr.open('GET', url);
  xhr.responseType = type;

  xhr.onload = function() {
    callback(xhr.response);
  };

  xhr.send();
}

function displayImage(blob) {
  let objectURL = URL.createObjectURL(blob);

  let image = document.createElement('img');
  image.src = objectURL;
  document.body.appendChild(image);
}

loadAsset('coffee.jpg', 'blob', displayImage);
```

위험성⇒

- 중첩된 콜백은 읽기 힘들다.
- 실패한 콜백에 대해 하나하나 예러문을 작성해야한다.
- 우아하지 못하다.
- 실행순서가 뒤죽박죽 될 수 있다.

• `setTimeout()`

Useful for...

Single delayed operation	Repeating operation	Multiple sequential operations	Multiple simultaneous operations
Yes	Yes (recursive timeouts)	Yes (nested timeouts)	No

```
let myGreeting = setTimeout(function() {
  alert('Hello, Mr. Universe!');
}, 2000)
```

Browser compatibility

[Report problems with this compatibility data on GitHub](#)

	Desktop						Mobile						Other	
	Chrome	Edge	Firefox	Internet Explorer	Opera	Safari	WebView Android	Chrome Android	Firefox for Android	Opera Android	Safari on iOS	Samsung Internet	Deno	Node.js
setTimeout	30	12	1	4	4	1	4.4	30	4	10.1	1	3.0	1.0	Partial ★ ▼
Supports parameters for callback	Yes	12	Yes	10	Yes	?	Yes	Yes	?	?	?	Yes	1.0	Yes

- **setInterval()**

Useful for...

Single delayed operation	Repeating operation	Multiple sequential operations	Multiple simultaneous operations
No	Yes	No (unless they are the same)	No

```
function displayTime() {
  let date = new Date();
  let time = date.toLocaleTimeString();
  document.getElementById('demo').textContent = time;
}

const createClock = setInterval(displayTime, 1000);
```

브라우저 지원은 위 setTimeout () 과 같다.

재귀적 setTimeout() 과 같은 기능이다.

- **requestAnimationFrame()**

Useful for...

Single delayed operation	Repeating operation	Multiple sequential operations	Multiple simultaneous operations
No	Yes	No (unless it is the same one)	No

```
const spinner = document.querySelector('div');
let rotateCount = 0;
let startTime = null;
let rAF;

function draw(timestamp) {
  if(!startTime) {
    startTime = timestamp;
  }

  rotateCount = (timestamp - startTime) / 3;

  if(rotateCount > 359) {
    rotateCount %= 360;
  }

  spinner.style.transform = 'rotate(' + rotateCount + 'deg)';

  rAF = requestAnimationFrame(draw);
}

draw();
```

위험성⇒

- 만약 너의 애니메이션의 프레임 속도를 느리게 실행해야한다면, setInterval() 또는 재귀적 setTimeout() 을 사용해야할 것이다.

모든 브라우저 지원

- **Promises**

Useful for...

Single delayed operation	Repeating operation	Multiple sequential operations	Multiple simultaneous operations
No	No	Yes	See Promise.all(), below

```
fetch('coffee.jpg')
.then(response => response.blob())
.then(myBlob => {
  let objectURL = URL.createObjectURL(myBlob);
  let image = document.createElement('img');
  image.src = objectURL;
  document.body.appendChild(image);
})
.catch(e => {
  console.log('There has been a problem with your fetch operation: ' + e.message);
});
```

위험성 ⇒

- 프로미스를 중첩할 가능성이 있다. 중첩해선 안된다.

브라우저 지원 (익스플로러 지원안됨)

	🖥️						📱						📦	
	Chrome	Edge	Firefox	Internet Explorer	Opera	Safari	WebView Android	Chrome Android	Firefox for Android	Opera Android	Safari on iOS	Samsung Internet	Deno	Node.js
Promise	32	12	29	No	19	8	4.4.3	32	29	19	8	2.0	1.0	0.12.0
Promise() constructor	32	12	29 ★ ▼	No	19	8 ★ ▼	4.4.3	32	29 ★ ▼	19	8 ★ ▼	2.0	1.0	0.12.0 ★ ▼
all()	32	12	29	No	19	8	4.4.3	32	29	19	8	2.0	1.0	0.12.0
allSettled()	76	79	71	No	63	13	76	76	79	54	13	12.0	1.0	12.9.0
any	85	85	79	No	No	14	85	85	79	No	14	14.0	1.2	15.0.0
catch()	32	12	29	No	19	8	4.4.3	32	29	19	8	2.0	1.0	0.12.0
finally()	63	18	58	No	50	11.1	63	63	58	46	11.3	8.0	1.0	10.0.0
Incumbent settings object tracking	No	No	50	No	No	No	No	No	50	No	No	No	No	No
race()	32	12	29	No	19	8	4.4.3	32	29	19	8	2.0	1.0	0.12.0
reject()	32	12	29	No	19	8	4.4.3	32	29	19	8	2.0	1.0	0.12.0
resolve()	32	12	29	No	19	8	4.4.3	32	29	19	8	2.0	1.0	0.12.0
then()	32	12	29	No	19	8	4.4.3	32	29	19	8	2.0	1.0	0.12.0

- **Promise.all()**

Useful for...

Single delayed operation	Repeating operation	Multiple sequential operations	Multiple simultaneous operations
No	No	No	Yes

```

function fetchAndDecode(url, type) {
  // Returning the top level promise, so the result of the entire chain is returned out of the function
  return fetch(url).then(response => {
    // Depending on what type of file is being fetched, use the relevant function to decode its contents
    if(type === 'blob') {
      return response.blob();
    } else if(type === 'text') {
      return response.text();
    }
  })
  .catch(e => {
    console.log(`There has been a problem with your fetch operation for resource "${url}": ` + e.message);
  });
}

// Call the fetchAndDecode() method to fetch the images and the text, and store their promises in variables
let coffee = fetchAndDecode('coffee.jpg', 'blob');
let tea = fetchAndDecode('tea.jpg', 'blob');
let description = fetchAndDecode('description.txt', 'text');

// Use Promise.all() to run code only when all three function calls have resolved
Promise.all([coffee, tea, description]).then(values => {
  console.log(values);
  // Store each value returned from the promises in separate variables; create object URLs from the blobs
  let objectURL1 = URL.createObjectURL(values[0]);
  let objectURL2 = URL.createObjectURL(values[1]);
  let descText = values[2];

  // Display the images in <img> elements
  let image1 = document.createElement('img');
  let image2 = document.createElement('img');
  image1.src = objectURL1;
  image2.src = objectURL2;
  document.body.appendChild(image1);
  document.body.appendChild(image2);

  // Display the text in a paragraph
  let para = document.createElement('p');
  para.textContent = descText;
  document.body.appendChild(para);
});

```

위험성⇒

- 프로미스 하나라도 리젝트가 되면 나머지의 결과의 resolve 여부를 알 수 없다.

브라우저 지원⇒

익스플로러 제외하고 모두 가능

- **Async await**

Useful for...

Single delayed operation	Repeating operation	Multiple sequential operations	Multiple simultaneous operations
No	No	Yes	Yes (in combination with <code>Promise.all()</code>)







```
async function myFetch() {  
  let response = await fetch('coffee.jpg');  
  let myBlob = await response.blob();  
  
  let objectURL = URL.createObjectURL(myBlob);  
  let image = document.createElement('img');  
  image.src = objectURL;  
  document.body.appendChild(image);  
}  
  
myFetch();
```

오늘의 단어

- Pitfall : 위험, 곤란, 어려움, 문제
- elapse : 경과하다, 지나다

Browser compatibility

[Report problems with this compatibility data on GitHub](#)

	<div></div>						<div></div>						<div></div>	
	<div>Chrome</div> 	<div>Edge</div> 	<div>Firefox</div> 	<div>Internet Explorer</div> 	<div>Opera</div> 	<div>Safari</div> 	<div>WebView Android</div> 	<div>Chrome Android</div> 	<div>Firefox for Android</div> 	<div>Opera Android</div> 	<div>Safari on iOS</div> 	<div>Samsung Internet</div> 	<div>Deno</div> 	<div>Node.js</div> 
setTimeout	30	12	1	4	4	1	4.4	30	4	10.1	1	3.0	1.0	Partial ★ ▼
Supports parameters for callback	Yes	12	Yes	10	Yes	?	Yes	Yes	?	?	?	Yes	1.0	Yes