

# async와 await

async

await

async 클래스 메서드

에러 핸들링

## async

async가 붙은 함수는 반드시 프로미스를 반환하고, 프로미스가 아닌 것은 프로미스로 감싸 반환

```
1  async function f() {  
2      return 1;  
3  }  
4  
5  f().then(alert); // 1
```

결과가 1인 resolved 프로미스가 반환된다  
-> 1이 alert된다

ko.javascript.info 내용:

1

확인

## await

await 키워드를 만나면 프로미스가 처리될 때까지 기다린다(await)

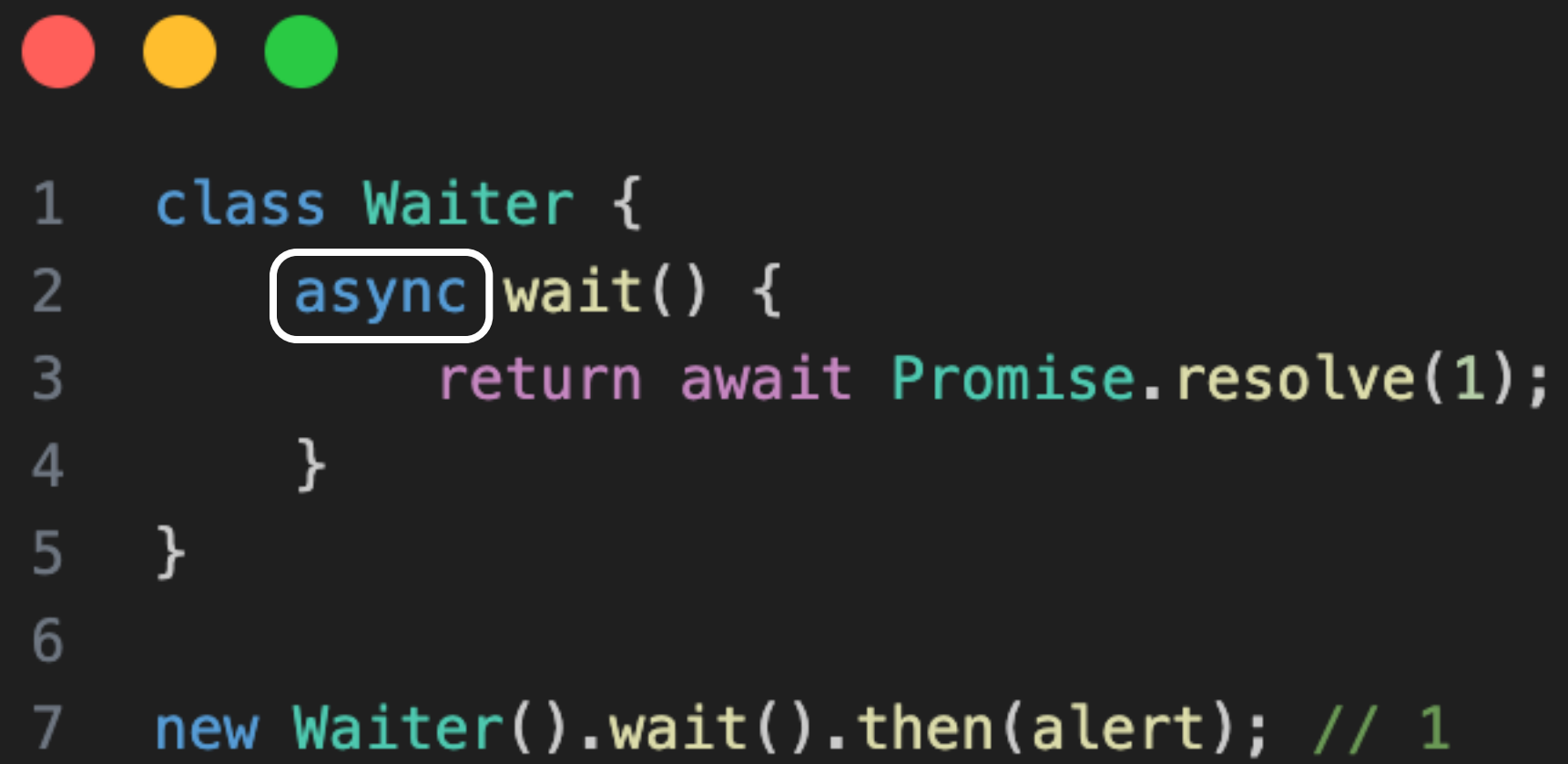
```
1  async function f() {  
2    let promise = new Promise((resolve, reject) => {  
3      setTimeout(() => resolve("완료!"), 1000);  
4    });  
5  
6    let result = await promise; // 프라미스가 이행될 때까지 기다림 (*)  
7  
8    console.log(result); // "완료!"  
9  }  
10  
11  f();
```

1초 뒤에 '완료!'가 출력된다.

프로미스가 처리되길 기다리는 동안엔 엔진이 다른 일(다른 스크립트를 실행, 이벤트 처리 등)을 할 수 있기 때문에, **CPU 리소스가 낭비되지 않습니다.**

## async 클래스 메서드

메서드 이름 앞에 `async`를 추가하면 `async` 클래스 메서드를 선언할 수 있다.



```
1 class Waiter {  
2   async wait() {  
3     return await Promise.resolve(1);  
4   }  
5 }  
6  
7 new Waiter().wait().then(alert); // 1
```

## 에러 핸들링

await가 던진 에러는 throw가 던진 에러를 잡을 때처럼 try..catch를 사용해 잡을 수 있다.

```
1  async function f() {  
2      try {  
3          let response = await fetch("http://유효하지-않은-주소");  
4          let user = await response.json();  
5      } catch (err) {  
6          // fetch와 response.json에서 발행한 에러 모두를 여기서 잡습니다.  
7          alert(err);  
8      }  
9  }  
10  
11  f();
```

## 에러 핸들링

await가 던진 에러는 throw가 던진 에러를 잡을 때처럼 try..catch를 사용해 잡을 수 있다.



```
1  async function f() {  
2      let response = await fetch("http://유효하지-않은-주소");  
3  }  
4  
5  // f()는 거부 상태의 프라미스가 됩니다.  
6  f().catch(alert); // TypeError: failed to fetch // (*)
```

try..catch가 없으면 f()를 호출해 만든  
프로미스가 rejected 상태가 된다.

-> .catch를 추가해 처리 가능

## async await

async/await를 함께 사용해 읽고, 쓰기 쉬운 비동기 코드를 작성할 수 있다.

```
1  async function showAvatar() {  
2    // JSON 읽기 (response 객체를 json화)  
3    let response = await fetch("/article/promise-chaining/user.json");  
4    let user = await response.json();  
5  
6    // github 사용자 정보 읽기  
7    let githubResponse = await fetch(`https://api.github.com/users/${user.name}`);  
8    let githubUser = await githubResponse.json();  
9  
10   // 아바타 보여주기  
11   let img = document.createElement("img");  
12   img.src = githubUser.avatar_url;  
13   img.className = "promise-avatar-example";  
14   document.body.append(img);  
15  
16   // 3초 대기  
17   await new Promise((resolve, reject) => setTimeout(resolve, 3000));  
18  
19   //후에 사라짐  
20   img.remove();  
21 }
```

1. .then 대신 await로 바꾼다.
2. function 앞에 async를 붙여 await를 사용할 수 있도록 한다

## async await

async/await를 함께 사용해 읽고, 쓰기 쉬운 비동기 코드를 작성할 수 있다.

```
10 // 아바타 보여주기
11 let img = document.createElement("img");
12 img.src = githubUser.avatar_url;
13 img.className = "promise-avatar-example";
14 document.body.append(img);
15
16 // 3초 대기
17 await new Promise((resolve, reject) => setTimeout(resolve, 3000));
18
19 //후에 사라짐
20 img.remove();
21
22 return githubUser;
23 }
24 showAvatar();
```

1. .then 대신 await로 바꾼다.
2. function 앞에 async를 붙여 await를 사용할 수 있도록 한다

이미지가 로드되면 보이고  
3초 후에 사라진다.





## async await와 promise.then

- async/await을 사용하면 await가 대기를 처리해주기 때문에 .then이 거의 필요 X
- .catch 대신 일반 try..catch를 사용할 수 있다는 장점
- 대체로, promise.then을 사용하는 것보다 async/await를 사용하는 것이 더 편리

**THANK YOU**