

ASYNCHRONOUS NETWORKING

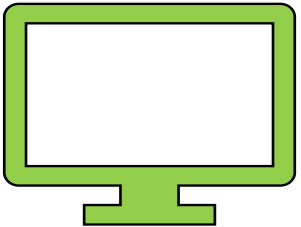
Networking with `async/await` & `fetch()`

OVERVIEW

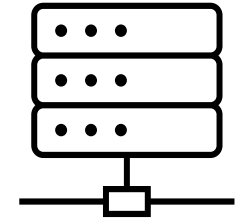
- Client-Server Model + AJAX
- Concurrency & JS
- Networking with XMLHttpRequest()
- Networking with Promises & fetch()
- Networking with `async/await` & `fetch()`

RECAP

Client

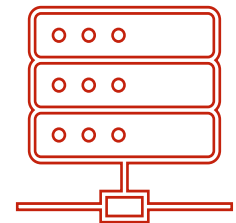


Servers



Latest News API

Super cool cats API



We've built up a nice asynchronous framework with callbacks, then Promises.

Both have a similar problem: code written does not look like the functions we are used to.

That's where `async/await` come in

ASYNC/AWAIT

```
1 // Promise version
2 fetch( input: "http://example.com/movies.json", init: {
3     method: "POST", // this object is optional
4 }) Promise<Response>
5     .then(res => res.json()) Promise<any>
6     .then(js => console.log(js));
7
8
9 // async/await version
10 async function getMovies() {
11     const url = "http://example.com/movies.json";
12
13     const res = await fetch(url);
14     const js = await res.json();
15
16     console.log(js);
17 }
18
```

- Keywords introduced in ES2017
 - Supported by virtually all browsers
 - So-called “coroutines” i.e. resumable functions
- Built on top of Promises
- `async` functions always return a Promise
- `await` expressions always unwrap a resolved Promise
- Rejections are thrown as exceptions
- Program asynchronous code in a familiar and intuitive way again

UNDER THE HOOD

AN INTUITIVE UNDERSTANDING

```
1 async function foo(v) {  
2   const w = await v;  
3   return w;  
4 }
```



```
1 resumable function foo(v) {  
2   implicit_promise = createPromise();  
3   // 1. Wrap v in a promise.  
4   promise = Promise.resolve(v)  
5   // 2. Attach handlers for resuming  
6   // foo.  
7   promise.then(() => {  
8     resume(foo);  
9   }, (err) => {  
10    throw(err);  
11  });  
12  // 3. Suspend foo and return  
13  // implicit_promise  
14  w = suspend(foo, implicit_promise);  
15  implicit_promise.resolve(w);  
16 }
```

Image Credit: [Zain Afzal](#)

- Non-promise values/thenables are awaitable like promises
- v always wrapped into a Promise
- This Promise then chained with injected logic to resume the function at the point where `await` was used
- JS runtime suspends execution of the function whilst returning a Promise to the caller that represents the suspended function's eventual result
- The returned Promise resolves once the suspended function is resumed and reaches the end of its body

N.B. It is *as-if* this is what happens. This is not real code.

ERROR-HANDLING REVISITED

```
1 // Promise version
2 function foo() {
3   // should reject with an unreachable domain error
4   const baz = fetch(input: "http://oops");
5
6   baz.catch(err => console.log(err));
7 }
8
9 // async/await version
10 async function Foo() {
11   // should reject with an unreachable domain error
12   const baz = fetch(input: "http://oops");
13
14   try {
15     await baz;
16   } catch (err) {
17     console.log(err);
18   }
19 }
20
```

- `async/await` functions always maintain their function scope
- Errors and rejections handled by `try/catch` again
- Careful:
 - `await`'ing a Promise will cause an exception
 - `async` functions called synchronously will still return a pending Promise

ASYNC/AWAIT FETCH DEMO

See [examples/asyncawait-fetch](#)



PROMISES OR ASYNC/AWAIT?

Promise

- Can pass around and attach different handlers
- Maintains a stack trace so takes more memory
- Supported more on older browsers

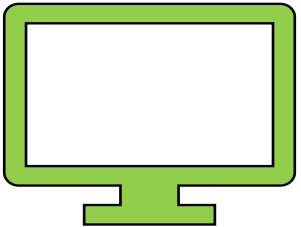
async/await

- Easier to read
- Potentially harder to debug
 - Debuggers will jump around source code
- More performant in some cases
- More familiar

Ultimately, which to use comes down to your specific needs / environment / targets, etc.

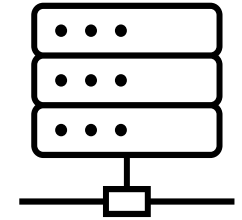
FINALLY!

Client



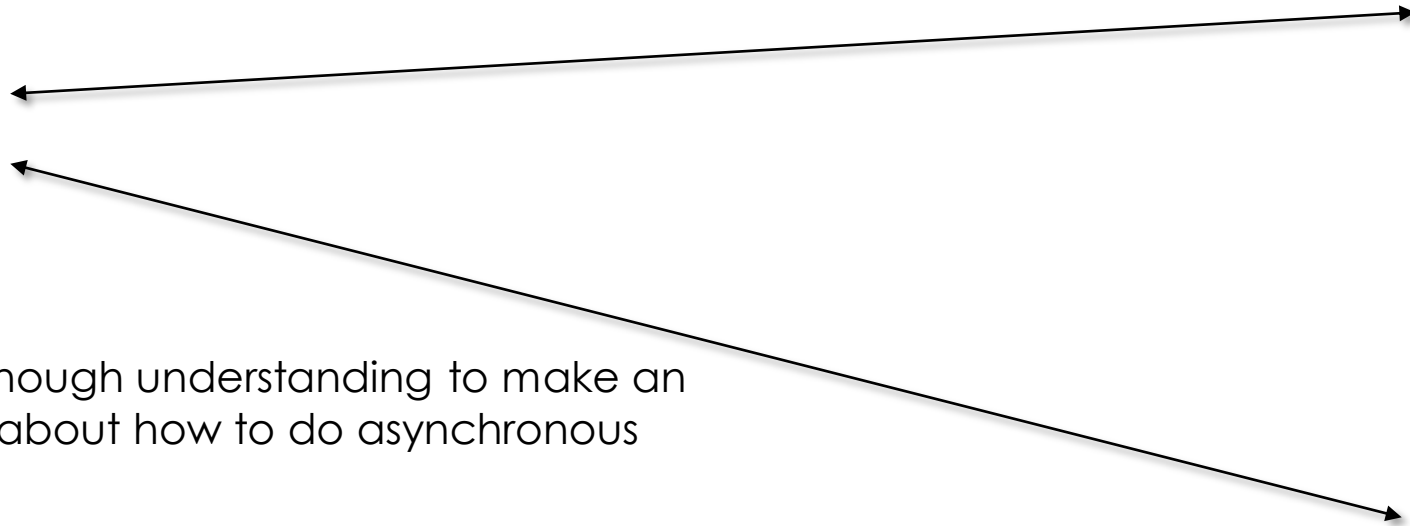
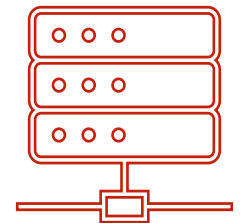
We've finally built enough understanding to make an educated decision about how to do asynchronous networking in JS!

Servers



Latest News API

Super cool cats API



SUMMARY

- Today:
 - `async/await`
 - Using `async/await` with `fetch()`
- Asynchronous networking with Javascript accomplished by:
 - `XMLHttpRequest` (legacy)
 - `fetch()` and Promises
 - `fetch()` and `async/await`
- AJAX technologies continue to evolve
- But the fundamentals rarely change