ASYNC JAVASCRIPT

🌂 PATTERNS AND GOTCHAS ☔

Hugo Di Francesco

Engineer at Concrete.cc

Slides: codewithhugo.com/async-js

Twitter: @hugo__df

CONTENTS 

1. Asynchronicity in JavaScript (a history lesson)
2. Why `async/await`?
3. Gotchas
4. Patterns

ASYNCHRONICITY IN JAVASCRIPT

Primitives:

- Callbacks
- Promises
- (Observables)
- `async/await`

What's asynchronous in a web application?

What's asynchronous in a web application?

tl;dr Most things

1. any network calls (HTTP, database)

1. any network calls (HTTP, database)
2. timers (`setTimeout`, `setInterval`)

1. any network calls (HTTP, database)
2. timers (`setTimeout`, `setInterval`)
3. filesystem access

1. any network calls (HTTP, database)
2. timers (`setTimeout`, `setInterval`)
3. filesystem access

... Anything else that can be offloaded

In JavaScript, these operations are non-blocking.

HTTP Request in Python:

```
data = request(myUrl)
print(data)
```

HTTP Request in JavaScript:

```
request(myUrl, (err, data) => {
  console.log(data);
});
```

Why non-blocking I/O?

You don't want to freeze your UI while you wait.

Why non-blocking I/O?

You don't want to freeze your UI while you wait.

Non-blocking -> waiting doesn't cost you compute cycles.

How non-blocking I/O is implemented (in JavaScript):

- pass a "callback" function
- it's called with the outcome of the async operation

NODE-STYLE CALLBACKS 🍬

```
myAsyncFn((err, data) => {
  if (err) dealWithIt(err);
  doSomethingWith(data);
})
```

A callback is:

- "just" a function
- in examples, usually anonymous functions (pass `function () {}` directly)
- according to some style guides, should be an arrow function (`() => {}`)
- called when the async operation finishes

A Node-style callback is:

- called with any error(s) as the first argument/parameter, if there's no error, `null` is passed
- called with any number of "output" data as the other arguments

ie. `(err, data) => { /* more logic */ }`

ASYNC/AWAIT 

```
(async () => {
  try {
    const data = await myAsyncFn();
    const secondData = await myOtherAsyncFn(data);
    const final = await Promise.all([
      fun(data, secondData),
      fn(data, secondData),
    ]);
    /* do anything else */
  } catch (err) {
    handle(err);
  }
})();
```

WHAT HAPPENS WHEN YOU FORGET `await`? 🐙

- values are undefined
- `TypeError: x.fn is not a function`

```
async function forgotToWait() {
  try {
    const res = fetch('https://jsonplaceholer.typicode.com/tod
    const text = res.text()
  } catch (e) {
    console.log(e);
  }
}

forgotToWait()
// TypeError: res.text is not a function
```