FALLING IN LOVE WITH ASYNC-AWAIT

Atishay Jain, Senior Computer Scientist, Adobe

Byetconf JS 2019
JavaScript
Agenda

■ About me
■ The async-await paradigm
■ A beginners walk-through of async await
  - *Callback vs promise vs async await*
  - *The program flow*
■ Intermediate async-await
  - *A detour to promises*
  - *Async guarantees*
  - *The program flow*
  - *Wrapping asynchronous functions*
■ Advanced async-await
  - *Exploiting the promises underneath*
  - *Exploiting callbacks deep down*
■ Conclusion
About Me

- Work in San Francisco.
- Crazy about the web
  - Working on web since the Flash & IE6 days.
  - Have worked on Desktop (Adobe InDesign), and on Mobile (Adobe Capture)
  - Author of VS Code’s “All Autocomplete” plugin
  - Rewrote my website for the nth time last year - https://atishay.me (Perf score 100)
  - Guest author at CSS-Tricks
- Ship node based desktop software to millions of Creative Cloud users
- Use these techniques(& much more) in daily life
- Doing asynchronous node.js since Node 0.6.
THE ASYNC-AWAIT PARADIGM
The concept of async-await

Serial Dispatch
The concept of async-await

Serial Dispatch
The concept of async-await

Serial Dispatch
The concept of async-await

Serial Dispatch
The concept of async-await

Serial Dispatch
The concept of async-await

Serial Dispatch
The concept of async-await

Serial Dispatch
The concept of async-await

```javascript
if(condition) {
    // code
}
```

Conditionals Dispatch
The concept of async-await

Conditionals Dispatch
The concept of async-await

Conditionals Dispatch
The concept of async-await

Conditionals Dispatch
The concept of async-await

Conditionals Dispatch

```javascript
if(condition) {
} else {

→

}
The concept of async-await

Conditionals Dispatch
The concept of async-await

Loop

while (i<1) {
    i++
}
The concept of async-await

Loop

while (i<1) {
    i++
}

The concept of async-await

while (i<1) {
    i++
}

Loop
The concept of async-await

```
while (i < 1) {
    i++
}
```

Loop
The concept of async-await

Loop

while (i<1) {
  i++
}

→

https://atishay.me
The concept of async-await

Loop
The concept of async-await

Loop

while (i<1) {
    i++
}

The concept of async-await

```
while (i<1) {
    i++
}
```

Loop
The concept of async-await

```javascript
while (i<1) {
  i++
}
```

Loop
The concept of async-await

Function

```javascript
function foo() {
  // Function implementation
}
```
The concept of async-await

```javascript
function foo() {
  // function body
}
```

Function
The concept of async-await

```javascript
function foo() {
}
```
The concept of async-await

```javascript
function foo()
{
    // Function body
}
```
The concept of async-await

```javascript
function foo() {
    // code
}
```
The concept of async-await

```
function foo() {
    // code here
}
```

Function
The concept of async-await

```
function foo() {
  // code
}
```
The concept of async-await

Function

```javascript
function foo() {
}
```
The concept of async-await

Function
The concept of async-await

```javascript
function foo() {
  // ...
}
```
The concept of async-await

```javascript
function foo() {
}
```

Function
The concept of async-await

Async

p = foo()
async function foo()
await p
The concept of async-await

Async

```
p = foo()

async function foo()

await p
```
The concept of async-await
The concept of async-await
The concept of async-await

Async

async function foo() {
  p = foo();

  await p;
}
The concept of async-await
The concept of async-await

Async

Falling in love with async await

Atishay Jain<contact@atishay.me>
https://atishay.me
The concept of async-await

Async

p = foo()

async function foo()

await p
The concept of async-await
The concept of async-await

Async

Async function foo()
The concept of async-await

Event Dispatch
The concept of async-await

Event Dispatch
The concept of async-await

Event Dispatch
The concept of async-await

Event Dispatch
The concept of async-await

Event Dispatch
The concept of async-await

Event Dispatch
The concept of async-await

Event Dispatch
The concept of async-await

Event Dispatch
The concept of async-await

Event Dispatch
BEGINNING ASYNC-AWAIT

A callback vs promise vs async await
A callback vs promise vs async await

**Callback**

```javascript
function work(data, cb) {
    db.connect((err) => {
        if(err) { return cb(err);}
        db.query(query1, (err, result1) => {
            if(err) { return cb(err) }
            db.query(query2, cb);
        });
    });
}
```

**Promises**

```javascript
function work() {
    return db.connect()
        .then(() => {
            return db.query(query1);
        }).then((result1) => {
            return db.query(query2);
        });
}
```

**async await**

```javascript
async function work() {
    await db.connect();
    result1 = await db.query(query1);
    return await db.query(query2);
}
```
BEGINNING
ASYNC-AWAIT
The Program Flow
The program flow

```javascript
async function foo() {
    ...
    await foo1();
    ...
    if(await foo2()) {
        ...
        await foo3()
        ...
    }
    for(...) {
        await foo4(...);
    }
    ...
}
```

The natural waterfall model
The program flow

```javascript
x = [...]
x.forEach(async y => {
    await foo(y);
});
```
The program flow

// Wrong will return promise to
// forEach and not wait
x = [...]
x.forEach(async y => {
  await foo(y);
});
The program flow

// Wrong will return promise to // forEach and not wait
x = [...]
x.forEach(async y => {
    await foo(y);
});

// Right - will wait for each // await in sequential order
x = [...];
for(y of x) {
    await foo(y)
}

Using for-of

Note: for..in does not make it to the JavaScript Good parts and should be avoided in the favor of for..of where possible.
INTERMEDIATE
ASYNC-AWAIT
A detour to promises
A detour to promises
A detour to promises

- Promises have a then/catch/finally.

```javascript
async function f() {...}
await f()
  .then((resp) => doSomething(resp))
  .catch((e) => handleError(e))
  .finally(() => cleanup());
```
A detour to promises

- There is a concept of unhandled rejection.

```javascript
process.on("unhandledRejection", ...);
window.addEventListener("unhandledrejection", ...);
```
A detour to promises

- Use Promises to convert a callback based method to async

```javascript
function x(data, callback) {
  y.doSomething(data, (err, z) => {
    if (err) { return callback(err); }
    processed = somePostProcessing(z);
    callback(null, processed)
  });
}

x(data, (err, processed) => { });
```

Conversion guide: https://atishay.me/blog/2018/08/25/from-callbacks-to-async-await/
A detour to promises

- Use Promises to convert a callback based method to async

```javascript
function x(data, callback) {
  y.doSomething(data, (err, z) => {
    if (err) { return callback(err); }
    processed = somePostProcessing(z);
    callback(null, processed)
  });
}

function x(data) {
  return new Promise((resolve, reject) => {
    y.doSomething(data, (err, z) => {
      if (err) { return reject(err); }
      processed = somePostProcessing(z);
      resolve(processed);
    });
  });
}

const processed = await x(data);
```

Conversion guide: https://atishay.me/blog/2018/08/25/from-callbacks-to-async-await/
A detour to promises

- Use Promises to convert a callback based method to async

```javascript
function x(data, callback) {
    y.doSomething(data, (err, z) => {
        if (err) { return callback(err); }
        processed = somePostProcessing(z);
        callback(null, processed)
    });
}
```

```javascript
async function x(data) {
    return new Promise((resolve, reject) => {
        y.doSomething(data, (err, z) => {
            if (err) { return reject(err); }
            processed = somePostProcessing(z);
            resolve(processed);
        });
    });
}
```

```javascript
x(data, (err, processed) => { });
```

```javascript
const processed = await x(data);
```

Conversion guide: [https://atishay.me/blog/2018/08/25/from-callbacks-to-async-await/](https://atishay.me/blog/2018/08/25/from-callbacks-to-async-await/)
INTERMEDIATE
ASYNC-AWAIT

The Program Flow
The program flow

```javascript
async foo() {
    ...
    await fetch()
    ...
}

const p = foo();
// Do some parallel tasks
... await p;
```

Delayed await
The program flow

```javascript
const tasks = [
  fetch(p1),
  fetch(p2),
  fetch(p3)
];

responses = await Promise.all(tasks);
```

Running stuff in parallel

Note: This will return quickly in case of exceptions. There are ways to workaround this, while the standards committee is working on Promise.allSettled to provide the easy solution.
The program flow

```javascript
await Promise.all([...].map(async x => {
    ...
    await foo(x);
    ...
}));
```

Using array.map
The program flow

```javascript
// setTimeout async version
const timeout = async (time) => new Promise(resolve => setTimeout(resolve, time));

// Adding a timeout to fetch
await Promise.race([fetch(...), timeout(2000)]);
```

Racing functions
The program flow

```javascript
function handleError(error) {
  console.log(error);
  return fallbackFoo2Response;
}

await foo1(...);
await foo2(...).catch(handleError);
await foo3(...)

await Promise.all([]
  .map(async () => {
    ...
  })
  .map(x => x.catch(e => e))
);
```

Errors without try...catch
ADVANCED
ASYNC-AWAIT
Async guarantees
Async guarantees
Async guarantees

```javascript
undefined = true;

foo.apply(window, arguments)

function(cb) {
  x((err) => {
    if(err) { callback(err); }
    callback(true);
  })
}
```
Async guarantees

- It always returns once
- There is only one return value
- It is always a promise

```javascript
undefined = true;
foo.apply(window, arguments)

function(cb) {
  x((err) => {
    if(err) { callback(err); }
    callback(true);
  })
}
```
Wrapping asynchronous functions
Wrapping asynchronous functions

Shared data problem
Wrapping asynchronous functions

```javascript
let mutex = false;

async lock() {
    while(mutex) { await timeout(100);}
    mutex = true;
}

unlock() {
    mutex = false;
}

async a() {
    await lock();
    await fs.readFile(file);
    await fs.writeFile(file);
    unlock()
}

async b() {
    await lock()
    await fs.readFile(file);
    await fs.writeFile(file);
    unlock();
}
```

Shared data and locks
Wrapping asynchronous functions

```javascript
async a() {
  await lock();
  await fs.readFile(file);
  await fs.writeFile(file);
}
```

Fixing forgotten lock call problem
Wrapping asynchronous functions

```javascript
async a() {
    await lock();
    await fs.readFile(file);
    await fs.writeFile(file);
}
```

```javascript
async a() {
    return await transaction(async () => {
        await fs.readFile(file);
        await fs.writeFile(file);
    });
}
```

Fixing forgotten lock call problem
Wrapping asynchronous functions

```javascript
async transaction(method) {
    await lock();
    const value = await method().catch(e => {
        unlock();
        throw e;
    });
    unlock();
    return value;
}
```

The transaction wrapper
Wrapping asynchronous functions

```javascript
async time(key, method) {
    console.time(key);
    return await method().finally(() => {
        console.timeEnd(key);
    });
}

// Usage
await time('myMethod', async () => {
    // do your work
})
```

A better way - the time wrapper
Wrapping asynchronous functions

```javascript
async time(key, method) {
  const context = {};
  console.time(key);
  return await method(context).finally(() => {
    console.timeEnd(key);
    fetch(url, {body: context});
  });
}

// Usage
await time(async (context)=> {
  //do your work
  context.x = "some result;
});
```

Include context
ADVANCED
ASYNC-AWAIT

Exploiting the promises underneath
Exploiting promises in async

The Parallel calls problem

Image from gifer
Exploiting promises in async

```javascript
function get(cb) {
    if (data) {
        cb(data);
    } else {
        callbacks.push(cb);
    }
    if (callbacks.length === 1) {
        ...
        callbacks.forEach(cb => cb(data));
    }
}
```

Doing it once
Exploiting promises in async

```javascript
let get = memoize(async () => {
    ...
    return data;
});
```

Memoize method
Exploiting promises in async

```javascript
function memoize(func, resolver) {
    const memoized = function(...args) {
        const key = resolver ? resolver.apply(this, args) : args[0]
        const cache = memoized.cache

        if (cache.has(key)) {
            return cache.get(key)
        }

        const result = func.apply(this, args)
        memoized.cache = cache.set(key, result) || cache
        return result
    }
    memoized.cache = new Map();
    return memoized
}
```

Lodash - Memoize method
https://github.com/lodash/lodash/blob/master/memoize.js
Exploiting promises in async

```javascript
if (result instanceof Promise) {
  result.state = 'pending';
  result.then(() => result.state = 'resolved')
    .catch(() => result.state = 'rejected');
}
```

```javascript
if (cache.has(key)) {
  let x = cache.get(key);
  if (x instanceof Promise) {
    if (x.state === 'pending') {
      return x;
    }
  }
}
```

Reuse the promise only if pending
Exploiting promises in async

// Convert to ES6 Decorator

function Memoize() {
    return function (target, functionName, descriptor) {
        descriptor.value = memoize(target[functionName]);
    };
}

@Memoize
async get() {
    ...
    return data;
}
ADVANCED
ASYNC-AWAIT

Exploiting callbacks deep down
Exploiting callbacks deep down

Reader-Writer Problem
Exploiting callbacks deep down

Reader-Writer Problem
Exploiting callbacks deep down

```javascript
await lock.read(async () => {
  // Do reading
});

await lock.write(async () => {
  // Do writing
});
```

Read-Write Lock API
Exploiting callbacks deep down

```javascript
class Lock {
    pendingReads = []
    pendingWrites = []

    async read(func) { // Cannot use await here
        return new Promise(resolve => {
            pendingReads.push({func, cb: resolve});
            this.perform();
        });
    }

    async write(func) { // Cannot use await here
        return new Promise(resolve => {
            pendingWrites.push({func, cb: resolve});
            this.perform();
        });
    }

    perform() {
        ...
    }
}
```

Structure of the Lock class
Exploiting callbacks deep down

```javascript
perform() {
    if (this.state === 'None' && this.pendingWrites.length > 0) {
        this.state = 'Write';
        const {func, cb} = this.pendingWrites.shift();
        func().finally(() => {
            cb(); this.state = 'None'; this.perform();
        })
    }

    if (this.state !== 'Write' && this.pendingReads.length > 0) {
        this.pendingReads.forEach(({func, cb}) => {
            this.state = 'Read';
            this.readInProgress++;
            func().finally(() => {
                cb();
                this.readInProgress--;
                if (this.readInProgress === 0) { this.state = 'None'; } 
                this.perform();
            })
        });
    }
}
```

readInProgress = 0
state = 'None'

Structure of the Lock class
CONCLUSION
Conclusion

- Where callbacks rule:
  - `addEventListener`
  - `Method Wrappers (can be @decorators as well)`.
- Where Promises rule
  - `Short hands with async`
- Where async await rule
  - `Everywhere else`
THANK YOU

Atishay Jain
https://atishay.me
contact@atishay.me