Introducing Async/Await

Valeri Karpov
@code_barbarian
github.com/vkarpov15

About Me

- Lead for Mongoose, Node+MongoDB ODM
- Author, *Mastering Async/Await* (ebook)
- Blogger, thecodebarbarian.com
- Invented the term “MEAN stack”
- 3rd async/await workshop: SF, Zagreb

Workshop Schedule

- **3:00-3:10** Intro, Return Values
- **3:10-3:40** Exercise 1
- **3:40-3:50** Error Handling
- **3:50-4:20** Exercise 2
- **4:20-4:25** Wrap-up and Takeaways

What Is Async/Await?

- 2 new keywords
- Async: special function that returns a promise
- Await: pauses execution of an async function

Example 1.1

```
async function test() {
  // This function will print "Hello, World!" after 1 second.
  await new Promise(resolve => setTimeout(() => resolve(), 1000));
  console.log('Hello, World!');
}

test();
```

Callback Hell

- Error handling
- Readability

```
function getWikipediaHeaders() {
  // i. check if headers.txt exists
  fs.stat('./headers.txt', function(err, stats) {
    if (err != null) { throw err; }
    if (stats == undefined) {
      // ii. fetch the HTTP headers
      var options = { host: 'www.wikipedia.org', port: 80 };
      http.get(options, function(err, res) {
        if (err != null) { throw err; }
        var headers = JSON.stringify(res.headers);
        // iii. write the headers to headers.txt
        fs.writeFile('./headers.txt', headers, function(err) {
          if (err != null) { throw err; }
          console.log('Great Success!');
        });
      });
    } else { console.log('headers already collected'); }
  });
}
```

Async/Await Makes Async Logic Flat

```
async function getWikipediaHeaders() {
  if (await stat('./headers.txt') != null) {
    console.log('headers already collected');
  }
  const res = await get({ host: 'www.wikipedia.org', port: 80 });
  await writeFile('./headers.txt', JSON.stringify(res.headers));
  console.log('Great success!');
}
```

```
function getWikipediaHeaders() {
  // i. check if headers.txt exists
  fs.stat('./headers.txt', function(err, stats) {
    if (err != null) { throw err; }
    if (stats == undefined) {
      // ii. fetch the HTTP headers
      var options = { host: 'www.wikipedia.org', port: 80 };
      http.get(options, function(err, res) {
        if (err != null) { throw err; }
        var headers = JSON.stringify(res.headers);
        // iii. write the headers to headers.txt
        fs.writeFile('./headers.txt', headers, function(err) {
          if (err != null) { throw err; }
          console.log('Great Success!');
        });
      });
    } else { console.log('headers already collected'); }
  });
}
```

Loops, If Statements, Try/Catch Work

Example 1.5

```
async function test() {
  while (true) {
    // Convoluted way to print out "Hello, World!" once per second by
    // pausing execution for 200ms 5 times
    for (let i = 0; i < 10; ++i) {
      if (i % 2 === 0) {
        await new Promise(resolve => setTimeout(() => resolve(), 200));
      }
    }
    console.log('Hello, World!');
  }
}
```

Can Only Await Within An Async Function

TLDR; don't use **`forEach()`** with async/await*

Example 1.3

```
function test() {
  const p = new Promise(resolve => setTimeout(() => resolve(), 1000));
  // SyntaxError: Unexpected identifier
  await p;
}
```

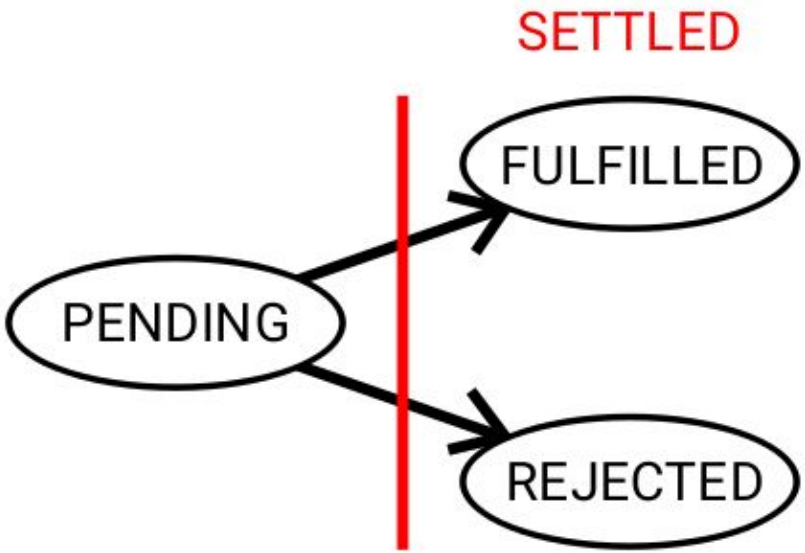
```
const assert = require('assert');

async function test() {
  const p = Promise.resolve('test');
  assert.doesNotThrow(function() {
    // "SyntaxError: Unexpected identifier" because the above function
    // is **not** marked async. "Closure" = function inside a function
    await p;
  });
}
```

Example 3.20

```
async function test() {
  const p1 = Promise.resolve(1);
  const p2 = Promise.resolve(2);
  // SyntaxError: Unexpected identifier
  [p1, p2].forEach(p => { await p; });
}
```

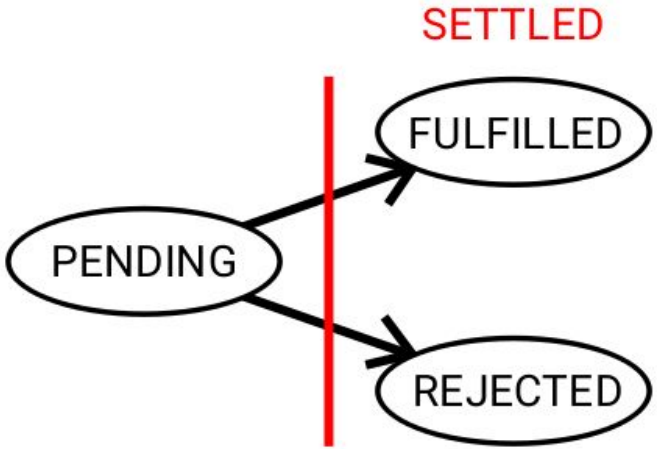
A Brief Overview of Promises

- Promise = state machine
- Represents async op
- Can *fulfill* with a value
- Or *reject* with an error
- await only handles promises



Await and Assignment

- Promise fulfilled value



Example 1.6

```
async function test() {
  // You can `await` on a non-promise without getting an error.
  let res = await 'Hello World!';
  console.log(res); // "Hello, World!"

  const promise = new Promise(resolve => {
    // This promise resolves to "Hello, World!" after 1s
    setTimeout(() => resolve('Hello, World!'), 1000);
  });
  res = await promise;
  // Prints "Hello, World!". `res` is equal to the value the
  // promise resolved to.
  console.log(res);

  // Prints "Hello, World!". You can use `await` in function params!
  console.log(await promise);
}
```

Composing Async Functions

- Async functions return a promise
- Referred to as the *returned promise*

Example 1.7

```
async function computeValue() {
  await new Promise(resolve => setTimeout(() => resolve(), 1000));
  // "Hello, World" is the _resolved value_ for this function call
  return 'Hello, World!';
}

async function test() {
  // Prints "Hello, World!" after 1s. `computeValue` returns a promise!
  console.log(await computeValue());
}
```

```
async function computeValue() {
  await new Promise(resolve => setTimeout(resolve, 1000));
  return 'Hello, World!';
}

async function run() {
  const promise = computeValue();
  await promise;
};
```

Resolved Value vs Return Value

The value you **`return`** from an async function is **not** the return value! Await *unwraps* the promise

Example 1.7

```
async function computeValue() {
  await new Promise(resolve => setTimeout(() => resolve(), 1000));
  // "Hello, World" is the _resolved value_ for this function call
  return 'Hello, World!';
}

async function test() {
  // Prints "Hello, World!" after 1s. `computeValue` returns a promise!
  console.log(await computeValue());
}
```

Exercise 1: Gather Blog Post Comments

- Suppose you have an API with 2 endpoints:
  - `/post?id=${id}`
  - `/posts`

Secure | https://us-central1-mastering-async-await.clou...

{"content":"If you've ever tried to build any kind of website, odds are you've had to create some way of validating and saving input from a form. Back in the bad old days this used to be a huge pain, because there were no good frameworks to help get the job done right. The three primary pain points that you have to deal with when trying to validate a form without the aid of a framework are:\n\n* **Pain Point #1.** How to avoid writing a wall of if-statements for validating each data item\n* **Pain Point #2.** How to handle adding and deleting data items as your code base evolves, i.e. how to avoid having to make changes in several different locations when you want to add/remove a data item\n* **Pain Point #3.** How to display validation errors to your client\n\nThankfully there are now multiple frameworks available to help make the entire form process easier. But how do

Secure | https://us-central1-mastering-async-await.clou...

[{"src":"./lib/posts/20160304_circle_ci.md","dest":{"directory":"./bin","name":"setting-up-circle-ci-with-node-js"},"title":"Setting Up Circle CI With Node.js","date":"2016-03-04T00:00:00.000Z","tags":["NodeJS"],"image":"http://i.imgur.com/3HiTMYc.jpg","id":51},{"src":"./lib/posts/20160311_superagent.md","dest":{"directory":"./bin","name":"replacing-angular-js-$http-backend-with-superagent"},"title":"Replacing AngularJS' $httpBackend With Superagent","date":"2016-03-11T00:00:00.000Z","tags":["NodeJS","AngularJS"],"image":"http://f.cl.ly/items/3d282n3A0h0Z0K2w0q2a/Screenshot

Exercise 1: Gather Blog Post Comments

- `fetch()` a list of blog posts
- `fetch()` the content of each blog post
- Find the `id` of the first post whose `content` contains “async/await hell”
- http://bit.ly/async-await-exercise-1

Part 2: Error Handling

- **`await`** on a fulfilled promise returns the value
- **`await`** on a rejected promise throws an error

Example 1.10

```
async function test() {
  try {
    const p = Promise.reject(new Error('Oops!'));
    // The below `await` throws
    await p;
  } catch (error) {
    console.log(err.message); // "Oops!"
  }
}
```

Consolidated Error Handling

- 3 different patterns to handle all CB errors

Example 1.12

```
function testWrapper(callback) {
  try {
    // There might be a sync error in `test()`
    test(function(error, res) {
      // `test()` might also call the callback with an error
      if (error) {
        return callback(error);
      }
      // And you also need to be careful that accessing `res.x` doesn't
      // throw **and** calling `callback()` doesn't throw.
      try {
        return callback(null, res.x);
      } catch (error) {
        return callback(error);
      }
    });
  }
}
```

Consolidated Error Handling

- Async function try/catch handles sync errors

Example 1.11

```
async function test() {
  try {
    const bad = undefined;
    bad.x;
    const p = Promise.reject(new Error('Oops!'));
    await p;
  } catch (error) {
    // "cannot read property 'x' of undefined"
    console.log(err.message);
  }
}
```

Unhandled Errors Become Rejections

- Throwing rejects the returned promise

Example 1.14

```
async function computeValue() {
  // `err` is the "rejected value"
  const err = new Error('Oops!');
  throw err;
}

async function test() {
  try {
    const res = await computeValue();
    // Never runs
    console.log(res);
  } catch (error) {
    console.log(error.message); // "Oops!"
  }
}
```

Rejected Value vs Sync Error

- *Rejected value* like resolved value for errors

Example 1.14

```
async function computeValue() {
  // `err` is the "rejected value"
  const err = new Error('Oops!');
  throw err;
}

async function test() {
  try {
    const res = await computeValue();
    // Never runs
    console.log(res);
  } catch (error) {
    console.log(error.message); // "Oops!"
  }
}
```

Await Throws, Not the Function Call

Example 1.1.5

```
async function computeValue() {
  throw new Error('Oops!');
};

async function test() {
  try {
    const promise = computeValue();
    // With the below line commented out, no error will be thrown
    // await promise;
    console.log("No Error");
  } catch (error) {
    console.log(error.message); // Won't run
  }
}
```

Should You Use Try/Catch?

- `catch()` works too, often a better choice

Example 1.17

```
async function computeValue() {
  throw new Error('Oops!');
};

async function test() {
  let err = null;
  await computeValue().catch(_err => { err = _err; });
  console.log(err.message);
}
```

Try/Catch vs. `catch()`

- Try/catch for specific, `catch()` for general
- Don't use try/catch to wrap the entire function

Example 1.18

```
// If you find yourself doing this, stop!
async function fn1() {
  try {
    /* Bunch of logic here */
  } catch (err) {
    handleError(err);
  }
}

// Do this instead
async function fn2() {
  /* Bunch of logic here */
}

fn2().catch(handleError);
```

Exercise 2: Retrying Failed Requests

- Exercise 1 assumed the API was reliable
- What about if every 2nd request fails?
- Need to wrap **`fetch()`** to retry 3 times
- http://bit.ly/async-await-exercise-2

Key Takeaways

- Async functions always return a promise
- `return` resolves the returned promise
- `throw` rejects the returned promise
- `await` pauses execution until promise settles
- `await p` returns the value p is fulfilled with

Further Reading

- http://bit.ly/node-promises-from-scratch
- http://bit.ly/async-await-design-patterns
- http://bit.ly/node-async-await
- *The 80/20 Guide to ES2015 Generators*

Thanks for Attending!

The *Mastering Async/Await* Ebook, June 14, 2018

asyncawait.net/wyncode