Towards ES7 Async/Await with Koa

Jonathan Ong
https://github.com/jonathanong
https://twitter.com/jongleberry
JSDC 2014

node.js callback errors must be handled

```
fs.stat(__dirname, function (err, stats) {
  // handle errors
  if (err) return console.error(err.stack);
  console.log(stats);
});
```

node.js callbacks can not be caught!

```
try {
  setImmediate(function () {
    throw new Error('boom');
  });
} catch (err) {


}
```

The callback pyramid of doom

```
fn1(function (err) {
  if (err) return callback(err);
  fn2(function (err) {
    if (err) return callback(err);
    fn3(function (err) {
      if (err) return callback(err);
      // …
    });
  });
});
```

node.js callbacks and conditionals

Requires a lot of refactoring!

node.js callback solutions

1. A lot of refactoring.
2. Use a control flow library like **`async`**.
3. Use domains (likely to be deprecated).
4. Use ES6 promises.

ES6 Promises

Native to JavaScript as of ES6

- Most browsers and probably IE12+
- Node v0.11.13+

It’s confusing to learn, especially for beginners.

ES7 Async/Await with Promises

Write asynchronous code like it's synchronous.

```
async function example() {
  try {
    // assuming node returns promises
    var stats = await fs.stat(__filename);
    var buffer = await fs.readFile(__filename);
  } catch (err) {
    console.error(err.stack);
  }
}
```

ES6 Promises

Will be very easy to use with **async/await**.

Will be rarely created!

We’re almost there with generators

https://github.com/visionmedia/co

```
co(function* () {
  try {
    // assuming node returns promises
    var stats = yield fs.stat(__filename);
    var buffer = yield fs.readFile(__filename);
  } catch (err) {
    console.error(err.stack);
  }
})();
```

Koa - The Spiritual Successor to Express

Written by TJ Holowaychuk.

Community maintained.

Uses `co` for control flow.

Uses the same modules as Express: https://jshttp.github.io

Koa: Generators as Middleware

```
// promised version of node's API
var fs = require('mz/fs');

app.use(function* (next) {
  this.response.body = yield fs.readFile(__filename);
});
```

Koa: With Try/Catch Support

```
var fs = require('mz/fs');

app.use(function* (next) {
  try {
    this.response.body = yield fs.readFile(__filename);
  } catch (err) {
    this.response.status = 404;
    this.response.body = {
      message: 'Could not read file!'
    };
  }
});
```

Koa: Eventually with ES7 Async/Await Support

```
app.use(async function errorHandler(next) {
  try {
    await next;
  } catch (err) {
    this.response.status = 500;
    this.response.body = {
      message: 'Can not send file!'
    };
  }
});

app.use(async function sendFile(next) {
  this.response.body = await fs.readFile(__filename);
});
```

Why Async/Await?

- No nested callbacks.
- Try/catch error handling.
- No 3rd party libraries needed.
- No more arguing about control flow.

The Path to ES7 Async/Await

Write everything with promises now so we don’t have to think about **async** later.