Koa - Asynchronous Decorators as Middleware

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

Express

Express is a linear sequence of callbacks:

```
http.createServer(function (req, res) {
  middleware1(req, res, function (err) {
    if (err) handleError(err);
    middleware2(req, res, handleError);
  });

  function handleError(err) {}
});
```

Express

You must handle all errors manually:

```
app.use(function (req, res, next) {
  fs.stat(__filename, function (err) {
    if (err) return next(err);
    req.stats = stats;
    next();
  });
});
```

Express

Middleware can’t change the response.

- You can’t minify HTML responses.
- You can’t cache and re-serve responses.

Express

Can’t execute asynchronous functions after a response is set.

- Save a session to the database.

HapiJS

Plugin based:

- Options vs. logic.
- More overhead to handle options and use-cases.
- Requires creating and using plugins.

Uses (deprecated) domains for error handling.

Bakes in many features not in Express.

A Truly Expressive Framework

allows you to write your app without relying on the features of framework.

No depending on “an API for that.”

Koa: Middleware are Decorators

Decorators are functions that wrap other functions.

Middleware are functions.

Read about the “**Decorator Pattern**”.

Koa: Middleware wrap Middleware

Middleware wrap all subsequent middleware.

```
app.use(function* minifier(next) {
  // we call this downstream
  yield next;
  // we call this upstream
  this.response.body = minify(this.response.body);
});

app.use(function* main() {
  this.response.body = '<html></html>';
});
```

Koa: Middleware are Apps

Middleware are apps that wrap other apps.

```
app.use(function* (subapp) {
  // before subapp is executed
  yield subapp;
  // after subapp is executed
  this.response.body = minify(this.response.body);
});

app.use(function* subapp() {
  this.response.body = '<html></html>';
});
```

Different Middleware Philosophies

Express

Pass the control flow to the next middleware.

Koa

Wrap all subsequent middleware.

Koa: Error Handlers are Decorators

```
app.use(function* errorHandler(next) {
  try {
    yield next;
  } catch (err) {
    this.response.status = 500;
    this.response.body = err.message;
  }
});

app.use(function* () {
  throw new Error('boom');
});
```

Koa: With Better Stream Handling

Koa handles all stream errors and leaks!

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

Via:

- https://github.com/stream-utils/destroy
- https://github.com/jshttp/on-finished

Different Error Handling Philosophies

Express

Let’s catch any error and try to do something about it.

Hapi

I’ve got this covered for you.

Koa

You handle it however and whenever you’d like.

Koa: Build From the Ground, Up

```
app.use(myBusinessLogic);
```

Start with your business logic.

Now, I want to serve static files.

Koa: Build From the Ground, Up

```
app.use(require('koa-static')());
app.use(myBusinessLogic);
```

Now, I want to handle compression..

Koa: Build From the Ground, Up

```
app.use(require('koa-compress')());
app.use(require('koa-static')());
app.use(myBusinessLogic);
```

Koa: Apps are like Christmas Trees

Start with your business logic. Keep decorating!

Express

“Where do I place put this middleware?”

Koa vs. Express

1. Error handling.
2. Async/Await-style control flow.
3. Decorator pattern.

Koa is minimal and has fewer features.

Koa and Express use the same modules.

Koa requires ES6+, **--harmony-generators** in node v0.11.13.

Koa vs. Hapi

- Koa is lean, Hapi is robust.
- Convention (Hapi) vs. Configuration (Koa).

Use-Cases for Koa

- APIs
- Promise-based Model (of MVC)
- Complex and/or unconventional sites

But steeper learning curve!

- Promises
- Generators
- Modular
- HTTP