StrongLoop An IBM Company

Asynchronous Error Handling in Express with Promises, Generators and ES7

April 21, 2015 / in Community, Express, How-To / by Marc Harter

tldr; Callbacks have a lousy error-handling story [http://strongloop.com/strongblog/robust-node-applications-error-handling/]. Promises are better [http://strongloop.com/strongblog/promises-in-node-js-with-q-an-alternative-to-callbacks/]. Marry the built-in error handling in Express with promises and significantly lower the chances of an uncaught exception. Promises are native ES6, can be used with generators, and ES7 proposals like `async/await` [https://github.com/tc39/ecmascript-asyncawait] through compilers like Babel [http://babeljs.io].

This article focuses on effective ways to capture and handle errors using error-handling middleware [https://github.com/tc39/ecmascript-asyncawait] in Express[1] [#foot1]. The article also includes a sample repository of

Questions?

Contact us for training and professio services to see how Node accelerate your mobile, I next-generation web API

Recent Posts

Introducing Raymon Cat Person who Occ Evangelizes
January 31, 2017 - 7:0

The Future of Node "Community Focuse and Inclusive Node
January 24, 2017 - 3:2

Introducing Joe Sep Lead Developer Evan Friends Call Me Joe Can Too"
January 24, 2017 - 2:4

Categories

API Connect (15)
API Tip (11)
Arc (24)
BACN (3)
Case Studies (3)
Cloud (11)
Community (251)

these concepts on GitHub [https://github.com/strongloop-community/express-example-error-handling] .

First, let’s look at what Express handles out of the box and then we will look at using promises, promise generators and ES7 `async/await` to simplify things further.

Express has built-in synchronous handling

By default, Express will catch any exception thrown within the initial *synchronous* execution of a route and pass it along to the next error-handling middleware:

```
app.get('/', function (req, res) {
  throw new Error('oh no!')
})
app.use(function (err, req, res, next) {
  console.log(err.message) // oh no!
})
```

Yet in asynchronous code, Express cannot catch exceptions as you’ve lost your stack once you have entered a callback:

```
app.get('/', function (req, res) {
  queryDb(function (er, data) {
    if (er) throw er
  })
})
app.use(function (err, req, res, next) {
  // error never gets here
})
```

For these cases, use the `next` function to propagate errors:

```
app.get('/', function (req, res, next) {
  queryDb(function (err, data) {
    if (err) return next(err)
    // handle data
```

Espanol (7)
Events (30)
Express (21)
How-To (162)
JavaScript Language (20)
  ES2015/ES6 (6)
LoopBack (127)
Mobile (32)
News (125)
Node core (6)
Node DevOps (22)
Node.js v0.12 (17)
Performance Tip (11)
Portuguese (31)
Press (14)
Product (30)
  API Gateway (2)
Resources (8)
StrongNode (7)
Uncategorized (2)

Archives

February 2017
January 2017
December 2016
November 2016
October 2016
September 2016
August 2016
July 2016
June 2016
May 2016
April 2016
March 2016
February 2016
January 2016
December 2015
November 2015
October 2015
September 2015
August 2015
July 2015
June 2015
May 2015
April 2015

```
    makeCsv(data, function (err, csv) {
      if (err) return next(err)
      // handle csv

    })
  })
})
app.use(function (err, req, res, next) {
  // handle error
})
```

Still, this isn't bulletproof. There are two problems with this approach:

1. You must explicitly handle *every* `error` argument.
2. Implicit exceptions aren't handled (like trying to access a property that isn't available on the `data` object).

Asynchronous error propagation with promises

Promises [http://strongloop.com/strongblog/promises-in-node-js-with-q-an-alternative-to-callbacks/] handle any exception (explicit and implicit) within asynchronous code blocks (inside `then`) like Express does for us in synchronous code blocks. Just add `.catch(next)` to the end of promise chains.

```
app.get('/', function (req, res, next) {
  // do some sync stuff
  queryDb()
    .then(function (data) {
      // handle data
      return makeCsv(data)
    })
    .then(function (csv) {
      // handle csv
    })
    .catch(next)
})
app.use(function (err, req, res, next) {
  // handle error
})
```

Now all errors asynchronous and synchronous get propagated to the error middleware. Hurrah!

Well, almost. Promises are a decent asynchronous primitive, but they are kinda verbose despite the welcomed error propagation. Let's fix this using promise generators.

Cleaner code with generators

If you use io.js [http://iojs.org] or Node `>=0.12`, you can improve on this workflow using native generators [https://strongloop.com/strongblog/how-to-generators-node-js-yield-use-cases/] [2] [#foot2] . For this, let's use a helper to make promise generators called `Bluebird.coroutine`.

> This example uses bluebird [https://github.com/petkaantonov/bluebird] , but promise generators exist in all the major promise libraries

First, let's make Express compatible with promise generators by creating a little `wrap` function:

```
var Promise = require('bluebird')
function wrap (genFn) { // 1
    var cr = Promise.coroutine(genFn) // 2
    return function (req, res, next) { // 3
        cr(req, res, next).catch(next) // 4
    }
}
```

The `wrap` function:

1. Takes a generator
2. Teaches it how to yield promises (through `Promise.coroutine`)
3. Returns a normal Express route function
4. When this function executes, it will call the coroutine, catch any errors, and pass them to `next`.

This `wrap` boilerplate hopefully will go away with Express 5 custom routers [https://github.com/strongloop/express/pull/2431] but write it once and keep it as a utility. With it, we can write route functions like this:

```
app.get('/', wrap(function *(req, res) {
  var data = yield queryDb()
  // handle data
  var csv = yield makeCsv(data)
  // handle csv
}))
app.use(function (err, req, res, next) {
  // handle error
})
```

This is pretty clean and reads well. All normal control structures (like `if/else`) work the same regardless if asynchronously or synchronous executed. Just remember to `yield` the promises.

Let's look next at the ES7 `async/await` proposal and clean things up even more.

Using ES7 async/await

The `async/await` proposal [https://github.com/tc39/ecmascript-asyncawait] behaves just like a promise generator but it can be used in more places (like class methods and arrow functions).

We still need a `wrap` function but it's simpler as we don't need `Bluebird.coroutine` or generators. Below is semantically the same as the previous `wrap` function, written in ES6:

```
let wrap = fn => (...args) => fn(...args).catch(args[2])
```

Then, we make routes like this:

```
app.get('/', wrap(async function (req, res) {
  let data = await queryDb()
  // handle data
  let csv = await makeCsv(data)
  // handle csv
}))
```

Or with arrow functions:

```
app.get('/', wrap(async (req, res) => { ... }))
```

Now, to run this code, you will need the Babel [http://babeljs.io] JavaScript compiler. There are many ways to use Babel with Node, but to keep things simple, install the `babel-node` command by running:

```
npm i babel -g
```

Then run your app using:

```
babel-node --stage 0 myapp.js
```

> Bonus: Since this code compiles to ES5, you can use this solution with older versions of Node.

Throw me a party!

With error handling covered both synchronously and asynchronously you can develop Express code differently. Mainly, **DO** use `throw`. The intent of `throw` is clear. If you use `throw` it will bypass execution until it hits a `catch`. In other words, it will behave just like `throw` in synchronous code. You can use `throw` and `try/catch` meaningfully again with promises, promise generators, and `async/await`:

```
app.get('/', wrap(async (req, res) => {
  if (!req.params.id) {
    throw new BadRequestError('Missing Id')
  }
  let companyLogo
  try {
    companyLogo = await getBase64Logo(req.params.id)
  } catch (err) {
    console.error(err)
    companyLogo = genericBase64Logo
  }
}))
```

Also **DO** use custom error classes [http://dailyjs.com/2014/01/30/exception-error/] like `BadRequestError` as it makes sorting errors out easier:

```
app.use(function (err, req, res, next) {
  if (err instanceof BadRequestError) {
    res.status(400)
    return res.send(err.message)
  }
  ...
})
```

Caveats

There are two caveats with this approach:

1. You must have all your asynchronous code return promises (except emitters). Raw callbacks simply don't have the facilities [http://strongloop.com/strongblog/promises-in-node-js-with-q-an-alternative-to-callbacks/] for this to work. This is getting easier as promises are legit now in ES6. If a particular library does not return promises, it's trivial to convert using a helper function like `Bluebird.promisifyAll`.
2. Event emitters (like streams) can still cause uncaught exceptions. So make sure you are handling the `error` event properly.

```
app.get('/', wrap(async (req, res, next) => {
  let company = await getCompanyById(req.query.id)
  let stream = getLogoStreamById(company.id)
  stream.on('error', next).pipe(res)
}))
```

Alternatives to promises

An alternative to promises is to capture errors using generators and thunks [http://en.wikipedia.org/wiki/Thunk] . One way to accomplish this is using co [https://github.com/tj/co] and a `wrap` function like co-express [https://github.com/mciparelli/co-express] .

---

1. I am assuming you *are* propagating errors there. If you are not, it will save you maintenance time and code duplication to do so. ↩ [#fref1]

2. Faux generators [https://facebook.github.io/regenerator/] work in older versions of Node using a JavaScript compiler like Babel. I personally find the `async/await` syntax more compelling if I am already using a compiler. ↩ [#fref2]

You may also be interested in...

- How to Deploy Express Apps with StrongLoop Process Manager [https://strongloop.com/strongblog/best-practices-express-js-process-manager/]
- Writing Modular Node.js Projects for Express and Beyond [https://strongloop.com/strongblog/modular-node-js-express/]
- Express 3.x to 4.x Migration Guide [https://strongloop.com/strongblog/express-3-to-4-migration-guide/]

Like 20 [] Tweet [] G+ Share []

Share [https://www.addtoany.com/share#url=https%3A%2F%2Fstrongloop.com error-handling-expressjs-es7-promises-generators%2F&title=Asynchronous%20Error%20Handling%20in%20Ex

< Previous Post **Next Post >**

Compose APIs, Build, Deploy and Monitor Node

Install API Connect

```
$ npm install -g apiconnect
```

Create your API Connect project

```
$ apic loopback
```

Launch API Designer

```
$ cd your-project

$ apic edit
```

API Designer will run on your local browser

An IBM Bluemix account is required to use the Designer. Register for IBM Bluemix

Click To Get Started

It’s FREE!