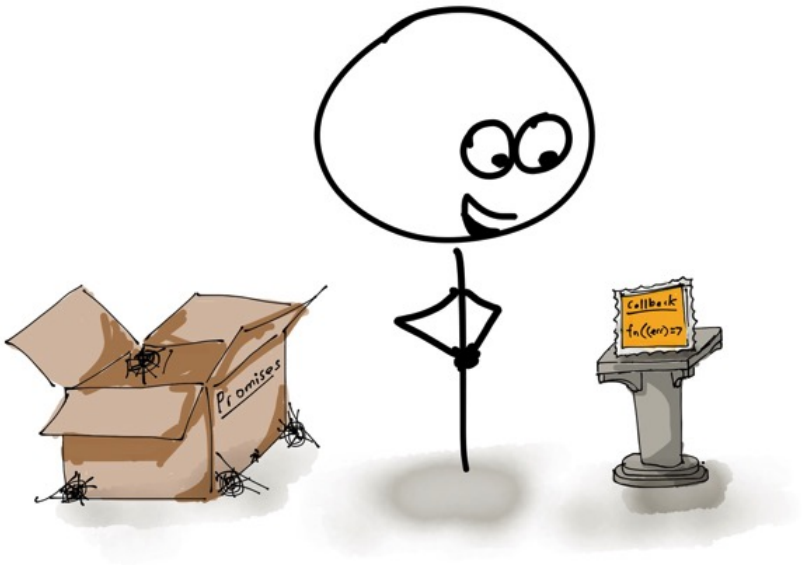
Callbacks First, Then Promises

October 5, 2016 By derickbailey

When I'm looking at asynchronous JavaScript – network calls, file system, or whatever it may be – I don't reach for promises, first. In fact, a promise is typically a last resort for me, relegated to specific scenarios.



I don't mean to say promises are not useful. They certainly are useful and they are a tool that I know well. I think you should know them – but I don't think you should use them as your default for async code.

Why not use promises, by default?

The TL;DR is that promises add another layer of complexity and potential bugs – especially around error handling.

But, I'm not going to elaborate on the complexities much. Nolan Lawson did an excellent job of pointing out the problems with promises, already.

He starts that post with a question, asking you to explain the difference in four uses of a promise.

```
1  doSomething().then(function () {
2    return doSomethingElse();
3  });
4
5  doSomething().then(function () {
6    doSomethingElse();
   });
```

```
8
9  doSomething().then(doSomethingElse());
10
11 doSomething().then(doSomethingElse);
```

1.js hosted with ❤ by GitHub view raw

This, to me, demonstrates the primary reason that I don't reach for promises by default. There are a lot of potential mistakes to make.

I reach for promises when I see the need.

That need typically arrises in one of the following scenarios:

- I need to wait for multiple async responses before continuing
- I want to cache a response and skip doing the work multiple times
- I want to chain methods together
- Working with async generators

The Promise.all method is great for waiting on multiple async returns. It lets you pass in an array of promises and wait for all of them to complete before firing the callback.

Caching a response is also useful, though there are many ways of doing this (including the memoize function of lodash / underscore, and others).

Chaining methods together can often make existing promises easier to understand and modify, though I don't typically do this unless I'm already working with promises.

And lastly, the current ES2015 (ES6) generators implementation facilitates better async patterns in our

code, but only when we add a small library around the generators.

I still prefer Node.js-style callbacks.

Having spent several years working in Node.js, I've grown to like the callback style where an error object is passed in as the first parameter.

```
doSome.work(function(err, something, stuff, etc){
  if (err) {
    // handle the error
  }

  // move on to the next thing
});
```

2.js hosted with ♥ by GitHub view raw

This code, in my experience, allows more flexibility while preventing the potential problems of swallowed and lost exceptions, that promises present.

Until we see the async / await syntax from ES(v.whatever) make it into Node and most browsers, I'll probably continue to reach for the callback style as my go-to for async code. Even when that happens, it will take a long time for this callback style to move to the side (if it ever does) due to shear momentum of existing code.

A promise is a tool you should understand, and use.

But I would not recommend using promises as your first choice for handling async JavaScript.

RELATED POST

10 Myths About Docker That Stop Developers Cold

Docker Recipes Update: Speed Up npm install In Mou...

Update and a Bonus Recipe for the Docker Recipes e...

A Sneak Peak at Docker Recipes for Node.js Develop...

Docker Recipes for Node.js: Pre-sale Dates & ...

Filed Under: Callbacks, JavaScript, Promises

About derickbailey

Derick Bailey is a developer, entrepreneur, author, speaker and technology leader in central Texas (north of Austin). He's been a professional developer since the late 90's, and has been writing code since the late 80's. In his spare time, he gets called a spamming marketer by people on Twitter, and blurts out all of the stupid / funny things he's ever done in his career on his email newsletter.

DERICK BAILEY AROUND THE WEB

Twitter: @derickbailey

Google+: DerickBailey

Screencasts: WatchMeCode.net

eBook: Building Backbone Plugins