

James R Nelson

Are JavaScript Promises swallowing your errors?

posted in **ES6, Javascript** on **May 29, 2015** by **James K Nelson**

This article has kindly been turned into a [video](#) by the folks at Webucator who do [this](#).

When dealing with asynchronous code, JavaScript's ES6 promises can make your life a lot easier. No more callback pyramids, no more error handling on every second line, and no more reliance on external libraries to do things as simple as getting the result of a for loop.

But ES6 Promises can have their own pitfalls, the biggest by far being disappearing error messages. Reddit's [/u/ledp](#) put it better than I ever could:

“

What the actual flying fuck? Promises swallows errors by default!
Who's idea was that?

More specifically, any exception which is thrown within a `then` handler, a `catch` handler, or within the function passed to `new Promise`, will be silently disposed of unless manually handled.

The problem

Let's test your understanding. Which of the following would you expect to print an error to the console?

```
Promise.resolve('promised value').then(function() {  
  throw new Error('error');  
});
```

```
Promise.reject('error value').catch(function() {  
  throw new Error('error');  
});
```

```
new Promise(function(resolve, reject) {  
  throw new Error('error');  
});
```

I don't know about you, but my answer is that I'd expect *all of them* to print an error. However, the reality is that a number of modern JavaScript environments won't print errors for *any of them*.

Manually regurgitating your errors

The obvious way to go about fixing this is to add an extra catch statement to the end of each of your promise chains. For example, you could fix the first example above with:

```
Promise.resolve('promised value').then(function() {  
    throw new Error('error');  
}).catch(function(error) {  
    console.error(error.stack);  
});
```

But are you seriously going to have the self-control to type that out after every `then` call? I sure don't. Knowing this, some fine people invented the `done` method, which is functionally equivalent to the above, and can be used just like `then`, allowing you to write:

```
Promise.resolve('promised value').done(function() {  
    throw new Error('error');  
});
```

Meanwhile, back in the real world...

This seems great in theory, taking no more effort than using good ol' `then`. But in reality, you generally don't just use `then` once, you'll chain it multiple times. For example, I have a project which grabs data via HTTP, and then further processes the response data asynchronously. Using `done`, the code would look something like:

```
getSomethingViaHttp()  
    .then(function(result) {  
        // processResultAsync returns another promise  
        return processResultAsync(result);  
    })  
    .done(function(processed) {  
        showResult(processed);  
    });
```

But then what happens when my processing library's API changes at a later date to be synchronous? I might update the code to the following:

```
getSomethingViaHttp()  
    .then(function(result) {  
        // processResultAsync returns another promise  
        showResult(processResultSync(result));  
    });
```

Oops! I removed the `done` block, but then forget to swap out the preceding `then` with a `done`. The result? I ended up spending hours trying to figure out why my app was silently failing when the HTTP

server decided it didn't like me. True Story!

Tell the computer which errors to discard, not which ones to handle

The problem with being human is that if you *can* make a mistake, at some point you *will*. Keeping this in mind, it seems obvious that we should design things in such a way that mistakes hurt as little as possible, and that means handling errors by default, not discarding them.

Happily, there is a library which makes this easy! **Bluebird** provides a simple fix which easily integrates with your existing code by extending the ES6 Promises API.

After installing bluebird, you can make sure you know about unhandled rejections by adding the following three lines:

```
Promise.onPossiblyUnhandledRejection(function(error){
  throw error;
});
```

And on the odd chance you *do* want to discard a rejection, just handle it with an empty `catch`, like so:

```
Promise.reject('error value').catch(function() {});
```

And that is all you need to know to promise safely. Of course, promising safely is only the start of doing it well! And promises are only one of many tools in the skilled JavaScript developer's toolbox.

So if you value your time and want to create Single Page Applications which people love, subscribe to my newsletter! In return for your e-mail, you'll also immediately receive 3 bonus *print-optimised* PDF cheatsheets – on **ES6**, **JavaScript promises** and **React**.

I will send you useful articles, cheatsheets
and code.

I won't send you useless inbox filler. *No spam, ever.*

First Name

Email Address

Join over 5000 subscribers

Unsubscribe at any time.



ABOUT THE AUTHOR



JAMES K NELSON

Hi! I've been writing JavaScript for over half my life, and would like to share what I've learned along the way.

PREVIOUS POST

← [Rendering React components to the document body](#)

NEXT POST

[The three faces of JavaScript `this`](#) →

21 Comments

→

Are JavaScript Promises swallowing your errors?



STEFEN

I've been looking for this forever ! Nice work.

REPLY ↓



JOSH

In your async-becomes-sync example, you could have left the original code structure. One of the benefits of promises is you can return either a promise or a value and the next then() block will have the right value. Your point still stands for more complex examples, but in this case the refactor was somewhat unnecessary.

REPLY ↓



JOE ZIM

That's exactly what I was thinking. That's one of the best things about Promises is that they can treat async and sync code exactly the same.

REPLY ↓



YUSUP

Hi there. I've just tried the code in the article in Chrome dev-console and each of three statements threw unhandled errors as they should to work. I didn't experience promises swallowing errors.

REPLY ↓



JAMESKNELSON

Yes, isn't Chrome wonderful? I hear FireFox throws the errors too in newer versions. You'll still run into this problem on node.js though – where I encountered the bug myself. Also, some promise libraries can cause the errors to stop being thrown.

It is great the Chrome now handles these un-handled errors, I wish node.js would catch up.

REPLY ↓



ŠIME VIDAS

I've tested your code in the console in Firefox Nightly. All three errors are still being swallowed.

REPLY ↓



ŠIME VIDAS

Scratch that. I've tested in both Firefox stable and Firefox Nightly and they behave the same. (Demo: <http://jsbin.com/seteto/quiet>). The thing is, Firefox's console displays the errors with an **delay**. About 5 to 15 seconds on my machine. The delay could be longer on complex sites and during periods with a lot of activity (just guessing).

REPLY ↓



JOE ZIM

Yea. That's really weird. I even ran the example code from this article, and waited. 2 of the errors showed up but not the 3rd. I refreshed the page and tried again and suddenly I had 4 errors show up... the 1 from before the refresh decided to show up! I wonder what is making this so slow.



LES ORCHARD

The delay is because Firefox uses the JS garbage collection process to find orphaned Promises resolved with errors. I forget off the top of my head, but I think you can force that process to kick in earlier / more often when you're debugging



ŠIME VIDAS

Who are these fine people? And where did they specify and implement `.done()`?

REPLY ↓



JAMESKNELSON

I'm not sure who *originally* implemented `'done'`, but you can find at least one of the implementations here in [kriskowal's Q](#)

REPLY ↓



LES ORCHARD

“

But are you seriously going to have the self-control to type that out after every `then` call?

But, you don't have to type that after every `.then()`. You just have to do it at the end of a chain of promises – i.e. `.then().then().then().catch()`.

You should even skip doing it from most of your utility functions / object methods, just as long as the code that calls them eventually calls `.catch()`.

The benefit is you should be able to handle errors where you want, rather than having to catch them at every step along the way like errors in node.js-style callbacks – e.g. `function (err, result) {}`. Errors in that world are often swallowed up, too – because they're not exceptions, they're just a parameter passed to a callback. At least in Promises, there's a more consistent way to do it.

REPLY ↓

PAWEL



I'm afraid you don't understand purpose of promises. It should "swallow" errors according to the spec.

"If either onFulfilled or onRejected throws an exception e, promise must be rejected with e as the reason."

The reason of that behaviour is simple. In any point you are absolutely sure that either onFulfilled or onRejected will be called.

```
var p = Promise.resolve('promised value').then(function() {
  throw new Error('error');
});

var p = p.then(function(){ return 'OK';});
var p = p.then(function(){ return 'OK';});
//any number of chains

p.then(
  function(r){console.log('done: ', r);},
  function(e){console.log('an error happened: ', e);}
);
```

Like if you do GUI, which trigger a chain of a network operations. You don't want to know about links of that chain. But you want to show user if operation was successful. And you want to be sure if something went wrong your code would be notified. So you don't want that chain to be broken in a middle.

REPLY ↓

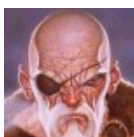


JAMESKNELSON

Interesting points you make. What do you think about the behaviour of recent browsers, where they show the exception in the console if it isn't handled by a rejection handler? This isn't swallowing, but it still follows the promises spec.

What about the case where you define your own promise using `new Promise`, but never actually call `resolve` or `reject`? This breaks the guarantee which you talk about.

REPLY ↓



PAWEL

Showing an exception in a console, in my view, is quite obscure thing. I would prefer they didn't act that way. Especially if you notice that behaviour is different for a 'normal' exception and exception in a promise. Anyway that's at least harmless.

Returning a 'pending' promise is absolutely unrelated to the exception question. When you return a promise it means you return a 'future' value which for a time being is unknown and you should wait until that future becomes present. You can do nothing with it. It's not exist yet. However on each layer you can write some custom logic like a connection timeout.

Anyway some function receive a promise, do whith it whatever it wants and pass it further. It's not of that function concern who gave that promise and where it's going

REPLY ↓



BENJAMIN GRUENBAUM

This is insane, of course you don't want to swallow errors in your code. Just think of:

```
getCall().then(function(result){
  JSON.prase(result); // typo
}).then(...);
```

Would get completely suppressed and ignored at the development stage as well – and if you forget a single `.catch` your life can become a nightmare. Of course – unhandled rejection tracking doesn't kick in if you handle errors yourself anyway.

REPLY ↓



BENJAMIN GRUENBAUM

On `io.js` (and soon `node.js`) as well as most promise libraries (like `Q` and `When`) you can do `process.on("unhandledRejection"` to access errors. See

https://iojs.org/api/process.html#process_event_unhandledrejection

REPLY ↓



GLEB BAHMUTOV

Good article, while I always suggest using `.done()` in the promise chains, see **Why promises need to be done**, your idea that the promises need to always throw the error might be overkill. The dynamic nature of promises means that someone can later attach error handler to a promise chain and handle an error.

REPLY ↓



BRIAN HUNT

See report in promises-aplus/promises-spec: <https://github.com/promises-aplus/promises-spec/issues/167>

REPLY ↓



BEN ZICHETTELO

See here before you try this with Bluebird 2.7 and up

<http://bluebirdjs.com/docs/api/error-management-configuration.html#global-rejection-events>

REPLY ↓



JASON SEBRING

You are the fucking man!

```
.catch(err => console.error(err.stack));
```

saved my ass. JESUS why didn't I know this!!!!!!

REPLY ↓

Leave a Reply

Your email address will not be published. Required fields are marked *

Comment

Name *

Email *

Website

Post Comment



James K Nelson

- **Wants to** help people create amazing things.
- **Has used** JavaScript for more than half his life.
- **Built** Memamug, an open-source React/Rails app.
- **Currently** working on Unicorn Standard.






Follow @james_k_nelson

 Follow @jamesknelson

Guides with Cheatsheets

- ES6 - The Bits You'll Actually Use
- Learn Raw React — no JSX, Flux, ES6, Webpack...
- Introduction to ES6 Promises

Recent Posts

-  Why I created junctions.js
-  Do I Even Need A Routing Library?
-  The 5 Types Of React Application State
-  Why use parenthesis on JavaScript return statements?
-  Lament of the keyboardist

Archives

- 📅 January 2017
- 📅 November 2016
- 📅 August 2016
- 📅 July 2016
- 📅 May 2016
- 📅 March 2016
- 📅 January 2016
- 📅 December 2015
- 📅 November 2015
- 📅 October 2015
- 📅 September 2015
- 📅 August 2015
- 📅 July 2015
- 📅 June 2015
- 📅 May 2015
- 📅 March 2015
- 📅 December 2014

