Most popular (last 30 days)

ECMAScript 2017: the final feature set

ECMAScript 6 modules: the final syntax

ES proposal: import() – dynamically importing ES modules

Making transpiled ES modules more spec-compliant

ES proposal: Shared memory and atomics

Classes in ECMAScript 6 (final semantics)

Communicating between Web Workers via MessageChannel

Most popular (all time)

ECMAScript 6 modules: the final syntax

Classes in ECMAScript 6 (final semantics)

Iterating over arrays and objects in JavaScript

ECMAScript 6's new array methods

The final feature set of ECMAScript 2016 (ES7)

WebAssembly: a binary format for the web

Basic JavaScript for the impatient programmer

Google Dart to "ultimately ... replace JavaScript"

Six nifty ES6 tricks

Google's Polymer and the future of web UI frameworks

Blog archive

► 2017 (4)

▼ 2016 (38)

2016-03-25

Promise-based functions should not throw exceptions

Labels: async, dev, esnext, javascript, promises

This blog post gives tips for error handling in asynchronous, Promise-based functions.

1. Operational errors vs. programmer errors

In programs, there are two kinds of errors:

- *Operational errors* happen when a correct program encounters an exceptional situation that requires deviating from the "normal" algorithm. For example, a storage device may run out of memory while the program is writing data to it. This kind of error is expected.
- *Programmer errors* happen when code does something wrong. For example, a function may require a parameter to be a string, but receives a number. This kind of error is unexpected.

1.1. Operational errors: don't mix rejections and exceptions

For operational errors, each function should support exactly one way of signaling errors. For Promise-based functions that means not mixing rejections and exceptions, which is the same as saying that they shouldn't throw exceptions.

1.2. Programmer errors: fail quickly

For programmer errors, it usually makes sense to fail as quickly as possible:

```
function downloadFile(url) {
    if (typeof url !== 'string') {
        throw new Error('Illegal argument: ' + url);
    }
    return new Promise(···).
}
```

Note that this is not a hard and fast rule. You have to decide whether or not you can handle exceptions in a meaningful way in your asynchronous code.

2. Handling exceptions in Promise-based functions

If exceptions are thrown inside the callbacks of `then()` and `catch()` then that's not a problem, because these two methods convert them to rejections.

However, things are different if you start your async function by doing something synchronous:

```
function asyncFunc() {
    doSomethingSync(); // (A)
    return doSomethingAsync()
    .then(result => {
        ···
    });
}
```


Dr. Axel Rauschmayer

Free online books by Axel

Speaking JavaScript [up to ES5]

Exploring ES6

JavaScript training:
Ecmanauten

- December (1)
- November (4)
- October (4)
- September (5)
- August (1)
- June (1)
- May (2)
- April (2)
- March (2)
  - Promise-based functions should not throw exception...
  - The need for multi-platform npm packages
- February (8)
- January (8)

- 2015 (65)
- 2014 (55)
- 2013 (98)
- 2012 (177)
- 2011 (380)
- 2010 (174)
- 2009 (68)
- 2008 (46)
- 2007 (12)
- 2006 (1)
- 2005 (2)

Labels

dev (592)
javascript (400)
computers (316)
life (194)
jslang (179)
esnext (156)
apple (107)
webdev (95)
mobile (83)
scitech (50)
hack (49)
mac (47)
google (39)
java (37)
ios (33)
business (32)
video (32)
clientjs (31)
hci (27)
entertainment (26)
nodejs (26)
society (26)

If an exception is thrown in line A then the whole function throws an exception. There are two solutions to this problem.

2.1. Solution 1: returning a rejected Promise

You can catch exceptions and return them as rejected Promises:

```
function asyncFunc() {
    try {
        doSomethingSync();
        return doSomethingAsync()
        .then(result => {
            ...
        });
    } catch (err) {
        return Promise.reject(err);
    }
}
```

2.2. Solution 2: executing the sync code inside a callback

You can also start a chain of `then()` method calls via `Promise.resolve()` and execute the synchronous code inside a callback:

```
function asyncFunc() {
    return Promise.resolve()
    .then(() => {
        doSomethingSync();
        return doSomethingAsync();
    })
    .then(result => {
        ...
    });
}
```

An alternative is to start the Promise chain via the Promise constructor:

```
function asyncFunc() {
    return new Promise((resolve, reject) => {
        doSomethingSync();
        resolve(doSomethingAsync());
    })
    .then(result => {
        ...
    });
}
```

This approach saves you a tick (the synchronous code is executed right away), but it makes your code less regular.

3. Async functions and exceptions

Brian Terlson points out that async functions reflect a preference for not mixing exceptions and rejections: Originally, if an async function had a default value that threw an exception then the function would throw an exception. Now, the function rejects the Promise it returns.

4. Further reading

- "Error Handling in Node.js" by Joyent
- ES proposal: async functions
- Promises for asynchronous programming [chapter in "Exploring ES6"]

Acknowledgements: this post was inspired by a post by user Mörre Noseshine in the "Exploring ES6" Google Group. Im also thankful for the feedback to a tweet asking whether it is OK to throw exceptions from Promise-based functions.

browser (25)
firefox (25)
html5 (24)
ipad (24)
movie (23)
psychology (22)
2ality (18)
tv (18)
android (17)
social (17)
chrome (16)
fun (16)
jsmodules (16)
tablet (16)
humor (15)
politics (15)
web (15)
cloud (14)
hardware (14)
microsoft (14)
software engineering (13)
blogging (12)
gaming (12)
eclipse (11)
gwt (11)
numbers (11)
programming languages (11)
app store (10)
media (10)
nature (10)
security (10)
semantic web (10)
software (10)
twitter (10)
webos (10)
12quirks (9)
education (9)
jstools (9)
photo (9)
webcomponents (9)
windows 8 (9)
async (8)
idea (8)
iphone (8)
itunes (8)
scifi-fantasy (8)
app (7)
babel (7)
bookmarklet (7)
chromeos (7)
english (7)
es proposal (7)
fringe (7)

Damien Lebrun • 10 months ago

when dealing with a mix of sync and cb functions, you can also use a Promise constructor:

```
function asyncFunc() {
  return New Promise((resolve, reject) => {
    doSomethingThatMightThrow();
    doSomethingAsync(err => (err ? reject(err) : resolve()))
  });
}
```

1 • Reply • Share ›

Katavic ➔ Damien Lebrun • 10 months ago

That's the way I use it now, Dr. Axel's function would look like this in my code:

```
function asyncFunc() {
        return new Promise((resolve,reject) => {
                doSomethingSync(); // (A)
                doSomethingAsync()
                .then(result => {
                        ...
                       resolve();
                 }).catch(reject);
        });
}
```

• Reply • Share ›

Damien Lebrun ➔ Katavic • 10 months ago

In that case, to keep things on the same level, you could do:

```
function asyncFunc() {
  return New Promise((resolve, reject) => {
    doSomethingThatMightThrow();
    doSomethingAsync.then(resolve, reject)
  }).then(result => {
    ...
  });
}
```

• Reply • Share ›

Dmitry Shimkin ➔ Damien Lebrun • 10 months ago

Also it can be simplified to:

```
function asyncFunc() {
  return new Promise((resolve) => {
    doSomethingThatMightThrow();
    resolve(doSomethingAsync());
  });
}
```

• Reply • Share ›

Katavic ➔ Dmitry Shimkin • 10 months ago

This is the first example of resolving Promise with a

html (7)
jsint (7)
jsshell (7)
thunderbolt (7)
webapp (7)
advancedjs (6)
blogger (6)
crowdsourcing (6)
latex (6)
lion (6)
promises (6)
ted (6)
book (5)
environment (5)
gadget (5)
googleio (5)
intel (5)
jsarrays (5)
jshistory (5)
layout (5)
light peak (5)
michael j. fox (5)
music (5)
pdf (5)
polymer (5)
shell (5)
tc39 (5)
template literals (5)
underscorejs (5)
vlc (5)
__proto__ (4)
coffeescript (4)
concurrency (4)
dart (4)
facebook (4)
gimp (4)
googleplus (4)
health (4)
howto (4)
hp (4)
javafx (4)
kindle (4)
leopard (4)
macbook (4)
motorola (4)
münchen (4)
occupy (4)
pl fundamentals (4)
presenting (4)
publishing (4)
series (4)
textbook (4)
web design (4)
amazon (3)

promise that I'm gonna use in my code. thx :)

Reply • Share ›

Kenneth Brubaker • 24 days ago

Maybe too late to ask, but what about async/await functions themselves? You cannot reject so exceptions are the only thing you can do, right?

If so, to be parallel when thening, you should throw exceptions and always have a catch all that rejects like in your example 2.1.

Thoughts?

Reply • Share ›

Ricardo Nolde • 4 months ago

If you are using **bluebird**, you can use *Promise.try* and *Promise.fromCallback*:

```
function asyncFunc() {
  return Promise
      .try(doSomethingThatMightThrow)
      .then(() => Promise.fromCallback(cb => doSomethingAsync(cb)))
      .then
          ...;
}
```

Reply • Share ›

Justin Hyland • 4 months ago

I agree that promise based functions shouldn't *throw* exceptions... But its fine to reject the promise with an exception, correct? Heres an example of what I mean: http://pastebin.com/ATqdr3hA

Reply • Share ›

Dmitri Pavlutin • 10 months ago

If there is a chance that the function can throw something, it's better to use in a separated resolve.
This way it's possible to solve the sync exception independently from the async functions and even continue the chain if the error is recoverable.

```
function asyncFunc() {
  return Promise.resolve({
        then: function(res) {
            doSomethingSync();
            res(true);
        }
    })
    .catch(function (exc) {
        //catch from sync()
        //try to recover and continue the chain
        return true;
        //..or throw the error to final handler
        return Promise.reject(exc);
     })
    .then(() => {
       return doSomethingAsync();
   }).then(result => {
        ...
   }).catch(...);
}
```

Reply • Share ›

LionessLover • 10 months ago

My other post was detected as spam because it contained two links I guess, but disqus promises to fix that. So I just add this in a new comment:

The Railway Oriented Programming talk nicely demonstrates the tracks analogy (instead of talking about monads directly).

Promises have two tracks for "result" and "exception". However, I found that I want THREE tracks:

1. normal result
2. not found (null)
3. exception

With two tracks I still need lots of "if" in the code - either in the `then` or in the `catch` part. Either I push `null` results on the exception track - then I need "if" to separate actual exceptions from mere "not found". Or I return them as a regular result - in which case I still have the entire "Cannot read property xyz of null" problem which the monad style is supposed to solve.

---

see more

Reply • Share ›

Marvin Richter ➚ LionessLover • 8 months ago

the "not found" case is a total valid response and should never be an exception. Because it is expected to happen!

Many people are using exceptions for implementing actual business functionality. That is a bad practice and is an indicator for having missed something in the architectural design.

The solution should be to take a step back and fix the architecture.

1 Reply • Share ›

Michael Burkman ➚ LionessLover • 8 months ago

A little late to the game, but if you use something like Bluebird for your promise handling, then you can choose to catch certain types of errors, while letting other errors propagate further.
http://bluebirdjs.com/docs/api...

Reply • Share ›

Alexander Glukhovtsev ➚ LionessLover • 10 months ago

You might find this useful
What does it mean to be #Reactive by Erik Meijer
it is all about #side #effects #types (it cover #Futures too)

Reply • Share ›

Newer Post Home Older Post

Subscribe to: Post Comments (Atom)

nike (2)

nokia (2)

npm (2)

programming (2)

raffle (2)

repl (2)

servo (2)

sponsor (2)

steve jobs (2)

travel (2)

typescript (2)

usb (2)

winphone (2)

wwdc (2)

airbender (1)

amdefine (1)

aol (1)

app urls (1)

architecture (1)

atscript (1)

basic income (1)

biology (1)

blink (1)

bluetooth (1)

canada (1)

clip (1)

coding (1)

community (1)

cross-platform (1)

deutsch (1)

diaspora (1)

distributed-social-network (1)

dsl (1)

dvd (1)

dzone (1)

emacs (1)

emberjs (1)

energy (1)

esnext news (1)

esprop (1)

example (1)

facetator (1)

feedback (1)

firefly (1)

firefoxos (1)

fritzbox (1)

german (1)

git (1)

guest (1)

guice (1)

h.264 (1)

home entertainment (1)

hosting (1)

htc (1)

ical (1)

jsdom (1)

jsmyth (1)

library (1)

location (1)

marketing (1)

mars (1)

meta-data (1)

middle east (1)

mpaa (1)

msl (1)

mssurface (1)

netflix (1)

nsa (1)

obama (1)

openoffice (1)

opinion (1)

oracle (1)

organizing (1)

philosophy (1)

pixar (1)

pnacl (1)

prism (1)

privacy (1)

proxies (1)

puzzle (1)

raspberry pi (1)

read (1)

rodney (1)

rust (1)

safari (1)

sponsoring (1)

star trek (1)

static generation (1)

talk (1)

technique (1)

theora (1)

thunderbird (1)

typography (1)

unicode (1)

v8 (1)

voice control (1)

webassembly (1)

webkit (1)

webm (1)

webpack (1)

yahoo (1)