Promises reading list

> This article was originally published at https://gist.github.com/joepie91/791640557e3e5fd80861.

This is a list of examples and articles, in roughly the order you should follow them, to show and explain how promises work and why you should use them. I'll probably add more things to this list over time.

This list primarily focuses on Bluebird, but the basic functionality should also work in ES6 Promises, and some examples are included on how to replicate Bluebird functionality with ES6 promises. You should still use Bluebird where possible, though - they are faster, less error-prone, and have more utilities.

> I'm available for tutoring and code review :)

If you get stuck

I've made a brief FAQ of common questions that people have about Promises, and how to use them. If you don't understand something listed here, or you're wondering how to implement a specific requirement, chances are that it'll be answered in that FAQ.

Compatibility

Bluebird will **not** work correctly (in client-side code) in older browsers. If you need to support older browsers, and you're using Webpack or Browserify, you should use the `es6-promise` module instead, and reimplement behaviour where necessary.

Introduction

- Start reading here, to understand why Promises matter.
- If it's not quite clear yet, some code that uses callbacks, and its equivalent using Bluebird.
- A demonstration of how promise chains can be 'flattened'

Promise.try

Many guides and examples fail to demonstrate Promise.try, or to explain why it's important. This article will explain it.

Error handling

- A quick introduction
- An illustration of error bubbling: step 1, step 2
- Implementing 'fallback' values (ie. defaults for when an asynchronous operation fails)
- bluebird-tap-error, a module for intercepting and looking at errors, without preventing propagation. Useful if you need to do the actual error handling elsewhere.
- Handling errors in Express, using Promises

Many examples on the internet don't show this, but you should **always** start a chain of promises with Promise.try, and if it is within a function or callback, you should always **return** your promises chain. Not doing so, will result in less reliable error handling and various other issues (eg. code executing too soon).

Promisifying

- Promisifying functions and modules that use nodebacks (Node.js callbacks)
- An example of manually promisifying an EventEmitter
- Promisifying `fs.exists` (which is async, but doesn't follow the nodeback convention)

Functional (map, filter, reduce)

- Functional programming in Javascript: map, filter and reduce (an introduction, not Bluebird-specific, but important to understand)
- (Synchronous) examples of map, filter, and reduce in Bluebird
- Example of using map for retrieving a (remote) list of URLs with bhttp

Nesting

- Example of retaining scope through nesting
- Example of 'breaking out' of a chain through nesting
- Example of a nested Promise.map

- An example with increasing complexity, implementing an 'error-tolerant' Promise.map: part 1, part 2, part 3

ES6 Promises

- Documentation on MDN
- Promise.try using ES6 Promises
- Promise.delay using ES6 Promises

Odds and ends

Some potentially useful snippets:

- Flattening an array of arrays, when using promises

You're unlikely to need any of these things, if you just stick with either Bluebird or ES6 promises:

- How to test whether a Promises implementation handles callbacks correctly
- Why this matters.

Revision #2
Created 2024-12-11 14:38:05 UTC by joepie91
Updated 2024-12-11 18:44:19 UTC by joepie91