FREE eBook

LEARNING bluebird

Free unaffiliated eBook created from
Stack Overflow contributors.

#bluebird

Table of Contents

About

You can share this PDF with anyone you feel could benefit from it, downloaded the latest version from: bluebird

It is an unofficial and free bluebird ebook created for educational purposes. All the content is extracted from Stack Overflow Documentation, which is written by many hardworking individuals at Stack Overflow. It is neither affiliated with Stack Overflow nor official bluebird.

Use the content presented in this book at your own risk; it is not guaranteed to be correct nor accurate, please send your feedback and corrections to info@zzzprojects.com

Chapter 1: Getting started with bluebird

Remarks

This section provides an overview of what bluebird is, and why a developer might want to use it.

It should also mention any large subjects within bluebird, and link out to the related topics. Since the Documentation for bluebird is new, you may need to create initial versions of those related topics.

Examples

Installation or Setup

Node.js

```
npm install bluebird
```

Then:

```
var Promise = require("bluebird");
```

Browsers

There are many ways to use bluebird in browsers:

- Direct downloads
  - Full build bluebird.js
  - Full build minified bluebird.min.js
  - Core build bluebird.core.js
  - Core build minified bluebird.core.min.js
- You may use browserify on the main export
- You may use the bower package.

When using script tags the global variables `Promise` and `P` (alias for `Promise`) become available. Bluebird runs on a wide variety of browsers including older versions. We'd like to thank BrowserStack for giving us a free account which helps us test that.

Read Getting started with bluebird online: https://riptutorial.com/bluebird/topic/5638/getting-started-with-bluebird

Chapter 2: Converting a callback API to promises.

Remarks

Promises have state, they start as pending and can settle to:

- **fulfilled** meaning that the computation completed successfully.
- **rejected** meaning that the computation failed.

Promise returning functions *should never throw*, they should return rejections instead. Throwing from a promise returning function will force you to use both a `} catch {` *and* a `.catch`. People using promisified APIs do not expect promises to throw. If you're not sure how async APIs work in JS - please see this answer first.

Examples

Converting a whole NodeJS module at once

Let's say you have a library that returns callbacks, for example the `fs` module in NodeJS:

```
const fs = require("fs");
fs.readFile("/foo.txt", (err, data) => {
  if(err) throw err;
  console.log(data);
});
```

We want to convert it to a promise returning API, with bluebird - we can do this using `promisifyAll` which converts an entire API to use promises:

```
const Promise = require("bluebird");
const fs = Promise.promisifyAll(require("fs"));
// this automatically adds `Async` postfixed methods to `fs`.
fs.readFileAsync("/foo.txt").then(console.log);
```

Which lets you use the whole module as promises.

Here are some common ecxamples on how to promisify certain modules:

```
// The most popular redis module
var Promise = require("bluebird");
Promise.promisifyAll(require("redis"));
// The most popular mongodb module
var Promise = require("bluebird");
Promise.promisifyAll(require("mongodb"));
// The most popular mysql module
var Promise = require("bluebird");
```

```
// Note that the library's classes are not properties of the main export
// so we require and promisifyAll them manually
Promise.promisifyAll(require("mysql/lib/Connection").prototype);
Promise.promisifyAll(require("mysql/lib/Pool").prototype);
// Mongoose
var Promise = require("bluebird");
Promise.promisifyAll(require("mongoose"));
// Request
var Promise = require("bluebird");
Promise.promisifyAll(require("request"));
// Use request.getAsync(...) not request(..), it will not return a promise
// mkdir
var Promise = require("bluebird");
Promise.promisifyAll(require("mkdirp"));
// Use mkdirp.mkdirpAsync not mkdirp(..), it will not return a promise
// winston
var Promise = require("bluebird");
Promise.promisifyAll(require("winston"));
// rimraf
var Promise = require("bluebird");
// The module isn't promisified but the function returned is
var rimrafAsync = Promise.promisify(require("rimraf"));
// xml2js
var Promise = require("bluebird");
Promise.promisifyAll(require("xml2js"));
// jsdom
var Promise = require("bluebird");
Promise.promisifyAll(require("jsdom"));
// fs-extra
var Promise = require("bluebird");
Promise.promisifyAll(require("fs-extra"));
// prompt
var Promise = require("bluebird");
Promise.promisifyAll(require("prompt"));
// Nodemailer
var Promise = require("bluebird");
Promise.promisifyAll(require("nodemailer"));
// ncp
var Promise = require("bluebird");
Promise.promisifyAll(require("ncp"));
// pg
var Promise = require("bluebird");
Promise.promisifyAll(require("pg"));
```

Converting a single NodeJS function

You can convert a single function with a callback argument to a `Promise`-returning version with `Promise.promisify`, so this:

```
const fs = require("fs");
fs.readFile("foo.txt", (err, data) => {
   if(err) throw err;
   console.log(data);
});
```

becomes:

```
const promisify = require("bluebird");
const readFile = promisify(require("fs").readFile));
readFile("foo.txt").then(console.log); // promisified version
```

Converting any other callback API

In order to convert any callback API to promises assuming the `promisify` and `promisifyAll` version doesn't fit - you can use the promise constructor.

Creating promises generally means specifying when they settle - that means when they move to the fulfilled (completed) or rejected (errored) phase to indicate the data is available (and can be accessed with `.then`).

```
new Promise((fulfill, reject) => { // call fulfill/reject to mark the promise
   someCallbackFunction((data) => {
      fulfill(data); // we mark it as completed with the value
   })
});
```

As an example, let's convert `setTimeout` to use promises:

```
function delay(ms) { // our delay function that resolves after ms milliseconds
  return new Promise((resolve, reject) => { // return a new promise
    setTimeout(resolve, ms); // resolve it after `ms` milliseconds have passed
  })
}
// or more concisely:
const delay = ms => new Promise(r => setTimeout(r, ms));
```

We can now use it like a regular promise returning function:

```
delay(1000).then(() => console.log("One second passed")).
            then(() => delay(1000)).
            then(() => console.log("Another second passed"));
```

Read Converting a callback API to promises. online:
https://riptutorial.com/bluebird/topic/5655/converting-a-callback-api-to-promises-

Chapter 3: Promise.all

Remarks

```
Promise.all(
  Iterable<any> | Promise<Iterable<any>> input
) -> Promise
```

This method is useful for when you want to wait for more than one promise to complete.

Given an `Iterable`(arrays are `Iterable`), or a promise of an `Iterable`, which produces promises (or a mix of promises and values), iterate over all the values in the `Iterable` into an array and return a promise that is fulfilled when all the items in the array are fulfilled. The promise's fulfillment value is an array with fulfillment values at respective positions to the original array. If any promise in the array rejects, the returned promise is rejected with the rejection reason.

```
var files = [];
for (var i = 0; i < 100; ++i) {
    files.push(fs.writeFileAsync("file-" + i + ".txt", "", "utf-8"));
}
Promise.all(files).then(function() {
    console.log("all the files were created");
});
```

This method is compatible with `Promise.all` from native promises.

Examples

Waiting for two things to happen

```
var firstItem = fetch("/api1").then(x => x.json());
var secondItem = fetch("/api2").then(x => x.json());
Promise.all([firstItem, secondItem]).spread((first, second) => {
  // access both results here, both requests completed at this point
});
```

Read Promise.all online: https://riptutorial.com/bluebird/topic/5656/promise-all

Credits

S. No	Chapters	Contributors
1	Getting started with bluebird	Ben Fortune, Community
2	Converting a callback API to promises.	Andy Pan, Benjamin Gruenbaum
3	Promise.all	Benjamin Gruenbaum