EBook Gratuito

APPRENDIMENTO

bluebird

Free unaffiliated eBook created from
Stack Overflow contributors.

#bluebird

Sommario

Di

Capitolo 1: Iniziare con Bluebird

Osservazioni

Questa sezione fornisce una panoramica di cosa sia Bluebird e perché uno sviluppatore potrebbe volerlo utilizzare.

Dovrebbe anche menzionare eventuali soggetti di grandi dimensioni all'interno di bluebird e collegarsi agli argomenti correlati. Poiché la Documentation for bluebird è nuova, potrebbe essere necessario creare versioni iniziali di tali argomenti correlati.

Examples

Installazione o configurazione

Node.js

```
npm install bluebird
```

Poi:

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

browser

Esistono molti modi per utilizzare bluebird nei browser:

- Download diretti
  - Completa compilazione bluebird.js
  - Completa configurazione bluebird.min.js minificata
  - Core build bluebird.core.js
  - Core build minified bluebird.core.min.js
- Puoi usare browserify sull'esportazione principale
- È possibile utilizzare il pacchetto Bower .

Quando si utilizzano tag di script, diventano disponibili le variabili globali `Promise` e `P` (alias per `Promise`). Bluebird funziona su un'ampia varietà di browser incluse le versioni precedenti. Vorremmo ringraziare BrowserStack per averci fornito un account gratuito che ci aiuta a provarlo.

Leggi Iniziare con Bluebird online: https://riptutorial.com/it/bluebird/topic/5638/iniziare-con-bluebird

Capitolo 2: Conversione di un'API di callback in promesse.

Osservazioni

Le promesse hanno lo stato, iniziano come in sospeso e possono accontentarsi di:

- significato **compiuto** che il calcolo è stato completato con successo.
- significato **rifiutato** che il calcolo non è riuscito.

Promettere che le funzioni di ritorno *non dovrebbero mai essere lanciate* , dovrebbero invece restituire i rifiuti. Lanciare da una funzione di ritorno promessa ti costringerà ad usare sia un `} catch {` *che* un `.catch` . Le persone che utilizzano le API promesse non si aspettano che le promesse vengano lanciate. Se non sei sicuro di come funzionano le API asincrone in JS, consulta prima questa risposta .

Examples

Convertire un intero modulo NodeJS in una sola volta

Supponiamo che tu abbia una libreria che restituisce i callback, ad esempio il modulo `fs` in NodeJS:

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

Vogliamo convertirlo in API di ritorno promettenti, con bluebird: possiamo farlo utilizzando `promisifyAll` che converte un'intera API per utilizzare le promesse:

```
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);
```

Che ti consente di utilizzare l'intero modulo come promesse.

Ecco alcuni ecxample comuni su come promuovere determinati moduli:

```
// 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"));
```

Conversione di una singola funzione NodeJS

È possibile convertire una singola funzione con un argomento di callback in una versione `Promise` -returning con `Promise.promisify` , quindi questo:

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

diventa:

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

Conversione di qualsiasi altra API di callback

Per convertire qualsiasi API di callback in promesse presupponendo che la versione `promisify` e `promisifyAll` non si adatti, è possibile utilizzare il costruttore di promesse .

Generare promesse significa in genere specificare quando si stabiliscono - ciò significa che quando si passa alla fase completata (completata) o rifiutata (errata) per indicare che i dati sono disponibili (e che è possibile accedere con `.then`).

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

Ad esempio, convertiamo `setTimeout` per usare le promesse:

```
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));
```

Ora possiamo usarlo come una normale funzione di ritorno promessa:

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

Leggi Conversione di un'API di callback in promesse. online:
https://riptutorial.com/it/bluebird/topic/5655/conversione-di-un-api-di-callback-in-promesse-

Capitolo 3: Promise.all

Osservazioni

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

Questo metodo è utile quando si desidera attendere il completamento di più di una promessa.

Dato un `Iterable` (gli array sono `Iterable`), o una promessa di un `Iterable` , che produce promesse (o un mix di promesse e valori), itera su tutti i valori in `Iterable` in un array e restituisci una promessa che si adempie quando tutte le gli elementi dell'array sono soddisfatti. Il valore di adempimento della promessa è un array con valori di adempimento nelle rispettive posizioni dell'array originale. Se una qualsiasi promessa nell'array viene respinta, la promessa restituita viene respinta con il motivo del rifiuto.

```
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");
});
```

Questo metodo è compatibile con `Promise.all` da promesse native.

Examples

Aspettando che succedano due cose

```
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
});
```

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

Titoli di coda

S. No	Capitoli	Contributors
1	Iniziare con Bluebird	Ben Fortune, Community
2	Conversione di un'API di callback in promesse.	Andy Pan, Benjamin Gruenbaum
3	Promise.all	Benjamin Gruenbaum