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)

2016-10-27

Three ways of understanding Promises

Labels: dev, esnext, javascript

This blog post covers three ways of understanding Promises.

This is an example of invoking a Promise-based function `asyncFunc()`:

```
function asyncFunc() {
    return new Promise((resolve, reject) => {
        setTimeout(() => resolve('DONE'), 100);
    });
}
asyncFunc()
.then(x => console.log('Result: '+x));

// Output:
// Result: DONE
```

So what is a Promise?

- Conceptually, invoking `asyncFunc()` is a blocking function call.
- A Promise is both a container for a value and an event emitter.

1. Conceptually: calling a Promise-based function is blocking

```
function asyncFunc() {
    return new Promise((resolve, reject) => {
        setTimeout(() => resolve('DONE'), 100);
    });
}
async function main() {
    const x = await asyncFunc(); // (A)
    console.log('Result: '+x);

    // Same as:
    // asyncFunc()
    // .then(x => console.log('Result: '+x));
}
main();
```

`main()` is an async function. Its body expresses well what's going on *conceptually* – how we usually think about asynchronous computations:

- Line (A): Wait until `asyncFunc()` is finished.
- Line (B): Then log its result `x`.

Prior to ECMAScript 6 and generators, you couldn't suspend and resume code, which is why, for Promises, you put everything that happens after the code is resumed into a callback. Invoking that callback is the same as resuming the code.

2. A Promise is a container for an asynchronously delivered value

Dr. Axel Rauschmayer

Free online books by Axel

Speaking JavaScript [up to ES5]

Exploring ES6

JavaScript training:
Ecmanauten

- ▶ December (1)
- ▶ November (4)
- ▼ October (4)
  - Three ways of understanding Promises
  - Tips for using async functions (ES2017)
  - ES proposal: asynchronous iteration
  - ES proposal: Rest/Spread Properties
- ▶ September (5)
- ▶ August (1)
- ▶ June (1)
- ▶ May (2)
- ▶ April (2)
- ▶ March (2)
- ▶ 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)

If a function returns a Promise then that Promise is like a blank into which the function will (usually) eventually fill in its result, once it has computed it. You can simulate a simple version of this process via an Array:

```
function asyncFunc() {
    const blank = [];
    setTimeout(() => blank.push('DONE'), 100);
    return blank;
}
const blank = asyncFunc();
// Wait until the value has been filled in
setTimeout(() => {
    const x = blank[0]; // (A)
    console.log('Result: '+x);
}, 200);
```

With Promises, you don't access the eventual value via `[0]` (as in line (A)), you use method `then()` and a callback.

3. A Promise is an event emitter

Another way to view a Promise is as an object that emits events.

```
function asyncFunc() {
    const eventEmitter = { success: [] };
    setTimeout(() => { // (A)
        for (const handler of eventEmitter.success) {
            handler('DONE');
        }
    }, 100);
    return eventEmitter;
}
asyncFunc()
.success.push(x => console.log('Result: '+x)); // (B)
```

Registering the event listener (line (B)) can be done after calling `asyncFunc()`, because the callback handed to `setTimeout()` (line (A)) is executed asynchronously, after this piece of code is finished.

Normal event emitters specialize in delivering multiple events, starting as soon as you register.

In contrast, Promises specialize in delivering exactly one value and come with built-in protection against registering too late: the result of a Promise is cached and passed to event listeners that are registered after the Promise was settled.

4. Further reading

- Chapter "Promises for asynchronous programming" in "Exploring ES6"
- Chapter "Async functions" in "Exploring ES2016 and ES2017"

hci (27)
entertainment (26)
nodejs (26)
society (26)
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)

Newer Post Home Older Post

Subscribe to: Post Comments (Atom)

chromeos (7)

english (7)

es proposal (7)

fringe (7)

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)

asmjs (3)

back to the future (3)

bitwise_ops (3)

css (3)

es2016 (3)

flattr (3)

fluentconf (3)

food (3)

foreign languages (3)

house (3)

icloud (3)

info mgmt (3)

jsfuture (3)

jsstyle (3)

linux (3)

mozilla (3)

python (3)

regexp (3)

samsung (3)

tizen (3)

traffic (3)

typedjs (3)

unix (3)

adobe (2)

angry birds (2)

angularjs (2)

astronomy (2)

audio (2)

comic (2)

design (2)

dom (2)

ecommerce (2)

eval (2)

exploring es6 (2)

facebook flow (2)

facets (2)

flash (2)

free (2)

futurama (2)

guide (2)

history (2)

hyena (2)

internet explorer (2)

iteration (2)

journalism (2)

jquery (2)

jsengine (2)

jslib (2)

law (2)

lightning (2)

markdown (2)

math (2)

meego (2)

month (2)

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)