ES7 async functions

Posted 27 March 2014 - hey at least it's not another progressive enhancement article

They're brilliant. They're brilliant and I want laws changed so I can marry them.

Update: This feature is now shipping in browsers. I've written a more up-to-date and in-depth guide.

Async with promises

In the HTML5Rocks article on promises, the final example show how you'd load some JSON data for a story, then use that to fetch more JSON data for the chapters, then render the chapters in order as soon as they arrived.

The code looks like this:

```
function loadStory() {
  return getJSON('story.json').then(function(story) {
    addHtmlToPage(story.heading);

    return story.chapterURLs.map(getJSON)
      .reduce(function(chain, chapterPromise) {
        return chain.then(function() {
          return chapterPromise;
        }).then(function(chapter) {
          addHtmlToPage(chapter.html);
        });
      }, Promise.resolve());
  }).then(function() {
    addTextToPage("All done");
  }).catch(function(err) {
    addTextToPage("Argh, broken: " + err.message);
  }).then(function() {
    document.querySelector('.spinner').style.display = 'none';
  });
}
```

Not bad, but…

This time with ES7 async functions...

```
async function loadStory() {
  try {
    let story = await getJSON('story.json');
    addHtmlToPage(story.heading);
    for (let chapter of story.chapterURLs.map(getJSON)) {
      addHtmlToPage((await chapter).html);
    }
    addTextToPage("All done");
  } catch (err) {
    addTextToPage("Argh, broken: " + err.message);
  }
  document.querySelector('.spinner').style.display = 'none';
}
```

With async functions (full proposal), you can `await` on a promise. This halts the function in a non-blocking way, waits for the promise to resolve & returns the value. If the promise rejects, it throws with the rejection value, so you can deal with it using `catch`.

Edit: I originally used `await` within an arrow function, apparently that's not allowed so I've replaced it with a `for` loop. Domenic gave me a knowledge smack-down on why `await` can't be used in arrow functions.

`loadStory` returns a promise, so you can use it in other async functions.

```
(async function() {
  await loadStory();
  console.log("Yey, story successfully loaded!");
}());
```

Until ES7 arrives...

You can use async functions and other ES6/7 features today using the Traceur transpiler. Also, you can use ES6 generators to create something akin to async functions.

You need a small bit of library code, a spawn function. Then you can use generators similar to async functions:

```
function loadStory() {
```

```
  return spawn(function *() {
    try {
      let story = yield getJSON('story.json');
      addHtmlToPage(story.heading);
      for (let chapter of story.chapterURLs.map(getJSON)) {
        addHtmlToPage((yield chapter).html);
      }
      addTextToPage("All done");
    } catch (err) {
      addTextToPage("Argh, broken: " + err.message);
    }
    document.querySelector('.spinner').style.display = 'none';
  });
}
```

In the above, I'm passing a generator function to `spawn`, you can tell it's a generator because for the asterisk in `function *()`. The `spawn` function calls `.next()` on the generator, receives the promise at the `yield` call, and waits for it to resolve before calling `.next()` with the result (or `.throw()` if it rejects).

ES7 brings the spawn function into the spec and makes it even easier to use. Having a standard way to simplify async coding like this is fantastic!

Further reading

- JavaScript promises, there and back again - guide to promises
- `Promise.resolve()` is not the opposite of `Promise.reject()` - a common misunderstanding with promises

Reply • Share ›

Белов Александр ➚ Jofferson Ramirez Tiquez • 8 months ago

It's not a future! It's a present!

3 Reply • Share ›

Claudenir Freitas ➚ Белов Александр • 6 months ago

it's true!

Reply • Share ›

dan dan ➚ Claudenir Freitas • 5 months ago

it's !false

1 Reply • Share ›

Claudenir Freitas ➚ dan dan • 5 months ago

ok

Reply • Share ›

Jofferson Ramirez Tiquez ➚ Белов Александр • 4 months ago

The future is now. :p

Reply • Share ›

Mathias Bynens • 3 years ago

For the record, the relevant ES7 proposal is here: https://github.com/lukehoba...

10 Reply • Share ›

Dan Dascalescu ➚ Mathias Bynens • a year ago

The URL changed to https://github.com/tc39/ecm...

Reply • Share ›

Kjellski • 3 years ago

It's really fun to see how close these are to what C# brings with async and await ;) even the same syntax!

8 Reply • Share ›

Matthias Götzke ➚ Kjellski • 2 years ago

yeah .. that was a great feature in c# .. sadly it wasnt really usable in large applications for us, since most libraries one wanted to use where blocking in nature ...

Reply • Share ›

naholyr • 3 years ago

The example does not seem equivalent to me, I'm pretty sure Promise#map runs all executions concurrently, whereas the await-ish equivalent will run in sequence.

More, this reduce boilerplate seems weird to me, I don't get why the example is not as simple as that (edit: wrong not working code) :

```
return story.chapterURLs.map(getJSON).map(function (chapter) {
  return addHtmlToPage(chapter.html)
```

```
})
```

and to avoid useless indentation, the whole beginning of the code could have been (using a quite widespread shortcut to get property from a promise):

```
var story = getJSON('story.json');
var storyHtml = story.get('html').then(addHtmlToPage);
var chaptersHtml = story.get('chapterURLs').map(getJSON).map(function (p) {
  return addHtmlToPage(p.html);
});
Promise.all([storyHtml, chaptersHtml])
.then(…)
```

But the real issue here may be the lack of concurrency in execution, resulting in a waste of time.

3 ^ v • Reply • Share ›

jaffathecake Mod ➚ naholyr • 3 years ago

Unfortunately there's isn't a Promise#map.

Your first example:

```
return story.chapterURLs.map(getJSON).map(function (chapter) {
  return addHtmlToPage(chapter.html)
});
```

This doesn't work as `chapter` is a promise, so it has no `.html`. If you changed that line to `chapter.then(function(data) { addHtmlToPage(data.html); })` then the chapters would appear in the order they load, which may not be the correct order.

I'm not sure of your second example, as it uses a number of promise methods that promises don't have, but yeah, it won't render chapters as soon as possible.

2 ^ v • Reply • Share ›

getify ➚ naholyr • 3 years ago

Do you have documentation (link) on calling `.get(..)` on promises? Is that part of the standard that I missed, or is that from some library?

1 ^ v • Reply • Share ›

naholyr ➚ naholyr • 3 years ago

Yep I used some methods common to most implementations : Promise#map, Promise#get, and Promise.all, which are anyway quite easy to implement from spec-only promises.

That's the whole point, tooling ;) You may not need new syntactic sugar in the core (even more if they make concurrent calls harder to write) if a dozen lines of codes can provide this in user-land. That's what I'm not so neatly arguing for ^^

The main point being this `await` thing makes concurrent calls hard to implement, and I'd like this point managed before it lands in the core.

(ping **@jaffathecake** & **@getify**)

About first example, I've written it too fast in fact, it should be (still based on Promise.all and Promise#map of course ;)):

```
return Promise.all(story.chapterURLs.map(getJSON)).map(function (chapter) {
  return addHtmlToPage(chapter.html)
})
```

3 ^ v • Reply • Share ›

C. Scott Ananian ➚ naholyr • 3 years ago

I've started the tooling:

https://github.com/cscott/p... adds map/get/etc on top of ES6 promises (via es6-shim or built-in to your engine, when that occurs).

^ v • Reply • Share ›

Domenic Denicola ➚ naholyr • 3 years ago

The execution is concurrent, since the calls to `getJSON` all happen at once. It simply waits for the first result to be available before adding the HTML; then waits for the second result; etc. But all the HTTP requests were already fired by the time the `for` loop starts.

8 ^ v • Reply • Share ›

naholyr ➚ Domenic Denicola • 3 years ago

OK got it, quite elegant indeed. You made your point, convinced ;)

^ v • Reply • Share ›

Vignesh shanmugam ➚ naholyr • 3 years ago

So in that case, Does the current requests are blocked till the previous response arrives?.

^ v • Reply • Share ›

jaffathecake Mod ➚ Vignesh shanmugam • 3 years ago

They're blocked from displaying, but not from downloading. getJSON starts the downloads.

All chapters download at the same time, but the 3rd will never appear on screen before the 2nd.

2 ^ v • Reply • Share ›

Vignesh shanmugam ➚ jaffathecake • 3 years ago

Got it Jake. Thanks :)

^ v • Reply • Share ›

benjamn • 3 years ago

Just submitted an Esprima pull request to parse async/await syntax: https://github.com/ariya/es...

Once that's in, transpiling to Promises + https://github.com/facebook... should be pretty easy!

3 ^ v • Reply • Share ›

Sylvain Pollet • 3 years ago

Does that mean ES7 will make yield much less useful ?

1 ^ v • Reply • Share ›

> **Domenic Denicola** ➚ Sylvain Pollet • 3 years ago
>
> Using `yield` for asynchronicity is actually kind of a hack; it's meant for lazy sequences and iterators. `await` nicely separates out the concerns by allowing `yield` to be used for its original purpose, and `await` for asynchronicity.
>
> 10 ^ v • Reply • Share ›

Esailija • 3 years ago

That's pretty bad but it can be much better if you play to promise strengths
http://pastebin.com/NfK5pKUi

1 ^ v • Reply • Share ›

> **jaffathecake** Mod ➚ Esailija • 3 years ago
>
> I don't think you're using JavaScript promises there. JavaScript promises don't have map or finally.
>
> 1 ^ v • Reply • Share ›
>
> > **Kris Kowal** ➚ jaffathecake • 3 years ago
> >
> > Allow me to introduce **@Esailija** , author of Bluebird. His implementation of Promise is a strict superset of ES6's, and extends the prototype with certain Q methods and some others. Q does not implement map, but one can `Q(array).invoke("map", relation).all()`. I expect that we will see many Promise prollyfills.
> >
> > https://github.com/petkaant...
> >
> > 2 ^ v • Reply • Share ›
> >
> > > **jaffathecake** Mod ➚ Kris Kowal • 3 years ago
> > >
> > > Yeah, we cleared this up on IRC. It wasn't clear that the article was dealing with platform features, not library ones.
> > >
> > > Totally agree that map, finally, get etc make stuff way better. Hopefully they'll appear in JS.
> > >
> > > ^ v • Reply • Share ›

Alberto Luiz Souza • 3 years ago

Nice feature!. Scala has this feature too :). It is so much better than have to handle with callback hell!

^ v • Reply • Share ›

Marc-André Lafortune • 3 years ago

Checkout IcedCoffeescript...

• Reply • Share ›

Joe Zimmerman • 3 years ago

So just adding "async" in front of "function" means it'll return a promise? How do you get it to resolve/reject with a certain value? Just return a value or throw an error?

• Reply • Share ›

> **Kris Kowal** ➚ Joe Zimmerman • 3 years ago
>
> Exactly.
>
> 1 • Reply • Share ›
>
> > **Joe Zimmerman** ➚ Kris Kowal • 3 years ago
> >
> > In that case, does a rejection handler receive the error object or the error message string as the rejection value?
> >
> > • Reply • Share ›
> >
> > > **Kris Kowal** ➚ Joe Zimmerman • 3 years ago
> > >
> > > It receives whatever was thrown, typically an Error.
> > >
> > > 3 • Reply • Share ›

Jeff Ingle • 3 years ago

Isn't this just the same async syntax from C#?

1 • Reply • Share ›

> **Michael Melanson** ➚ Jeff Ingle • 2 years ago
>
> Yes it is, this is clearly cribbed directly from C#'s async/await. It's one of the greatest innovations in C#, so I'm glad to see it getting adopted elsewhere.
>
> 1 • Reply • Share ›

Joe Zimmerman • 3 years ago

I think this can be optimized better:

```
for (let chapter of story.chapterURLs.map(getJSON)) {
addHtmlToPage((await chapter).html);
}
```

If you do a map, and then iterate over the results of the map, you end looping twice. If you do this, you'll only loop once:

```
for (let chapter of story.chapterURLs) {
addHtmlToPage((await getJSON(chapter)).html);
}
```

It could also potentially be easier to read, depending on the reader.

• Reply • Share ›

> **jaffathecake** Mod ➚ Joe Zimmerman • 3 years ago
>
> getJSON starts the download, so although your code does less looping, it downloads the chapters in series rather than in parallel, which is much slower.

2 ^ v • Reply • Share ›

Joe Zimmerman ➚ jaffathecake • 3 years ago

You're right. I realized that before, but then when I came back here and looked at it again, my brain broke. :P

^ v • Reply • Share ›

Tom Yandell • 3 years ago

Hey Jake. Nice post. Do you know if top-level scopes will be equivalent to async function bodies - i.e. will you be able to await at the top level? Seems to me that that would remove the need for a done method for promises (as libraries like Q have).

^ v • Reply • Share ›

Tom Yandell ➚ Tom Yandell • 3 years ago

nm, is covered here: https://github.com/lukehoba...

^ v • Reply • Share ›

Yohei Munesada • 3 years ago

I didn't know that!! These are really cool!

^ v • Reply • Share ›

agamemnus • 2 years ago

Or...

http://jsfiddle.net/agamemn...

^ v • Reply • Share ›

jaffathecake Mod ➚ agamemnus • 2 years ago

I don't get this. It's longer & it hides the spinner before all the stories have arrived. What's the benefit?

^ v • Reply • Share ›

agamemnus ➚ jaffathecake • 2 years ago

I was talking about the format/syntax: I don't like the idea of adding more and more syntactical methods when the current syntax will work. It just needs a different engine functionality.

And it's not longer... I deleted the original fiddle, but here is one way:

```
function loadStory () {
var result = await (getJSON('story.json')) // This will wait until getJSON, an async function, runs the callback.

// We're now in a virtual callback function. "story" is an object with either a success or error property.
if (story.error) {
// code...
return
}
var story = result.success
```

var story = result.success
// ...code...
}

• Reply • Share ›

Matt • 2 years ago

You can try start using async and await right now using https://www.npmjs.com/packa... Enjoy!

• Reply • Share ›

jaffathecake Mod → Matt • 2 years ago

The implementations in https://6to5.org/ and https://github.com/google/t... feel more standard.

• Reply • Share ›

Matt → jaffathecake • 2 years ago

Really? In what way? Last time I looked at 6to5 didn't have either async or await, and traceur requires runtime support. Obviously the implementation is different, but the semantics should be very close to the standard (as far as es7 is a standard).

• Reply • Share ›

jaffathecake Mod → Matt • 2 years ago

6to5 requires the regenerator runtime, but I'm using it in production.

https://github.com/jakearch...

• Reply • Share ›

Matt → jaffathecake • 2 years ago

That's good to know. I'm interested to know how the implementations compare - it's true nodent is very minimalist (just async/await). I'm putting together some benchmarks to see how they all compare. I'll keep you posted

• Reply • Share ›