Iterator Helpers update

Michael Ficarra · TC39 · November 2022

with special thanks to Gus Caplan, Kevin Gibbons, and Yulia Startsev

resolutions to previously discussed topics

Search or jump to... / Pull requests Issues Marketplace Explore

tc39 / proposal-iterator-helpers Public

Unwatch Fork Starred 868

Code Issues 2 Pull requests Actions Settings

fixes #207: add a counter parameter to Array [illegible]gues that have indices #211

Edit Code

Merged main ← GH-207 6 minutes ago

Conversation 7 Commits 3 Checks [illegible] Files changed

+48 −54

michaelficarra commented on Aug 9 Member

Fixes #207. Adds "counter" parameters to [illegible] methods that have an `Array.prot[illegible]` [illegible]gue that passes an index to its callback. Also remov[illegible] `.index[illegible]` [illegible] that it's equivalent to `[illegible] => [i, a])`.

Feedback on #207 has been mixed. I am not yet convinced [illegible] is an improvement, but I am also not opposed to this change if that's what the committee d[illegible]

👎 2

ljharb commented on Aug 9 Member

The PR does what it claims to do, but I'm even less enthusiastic about it than "unconvinced".

Reviewers – review now

bakkot

Assignees

No one—assign yourself

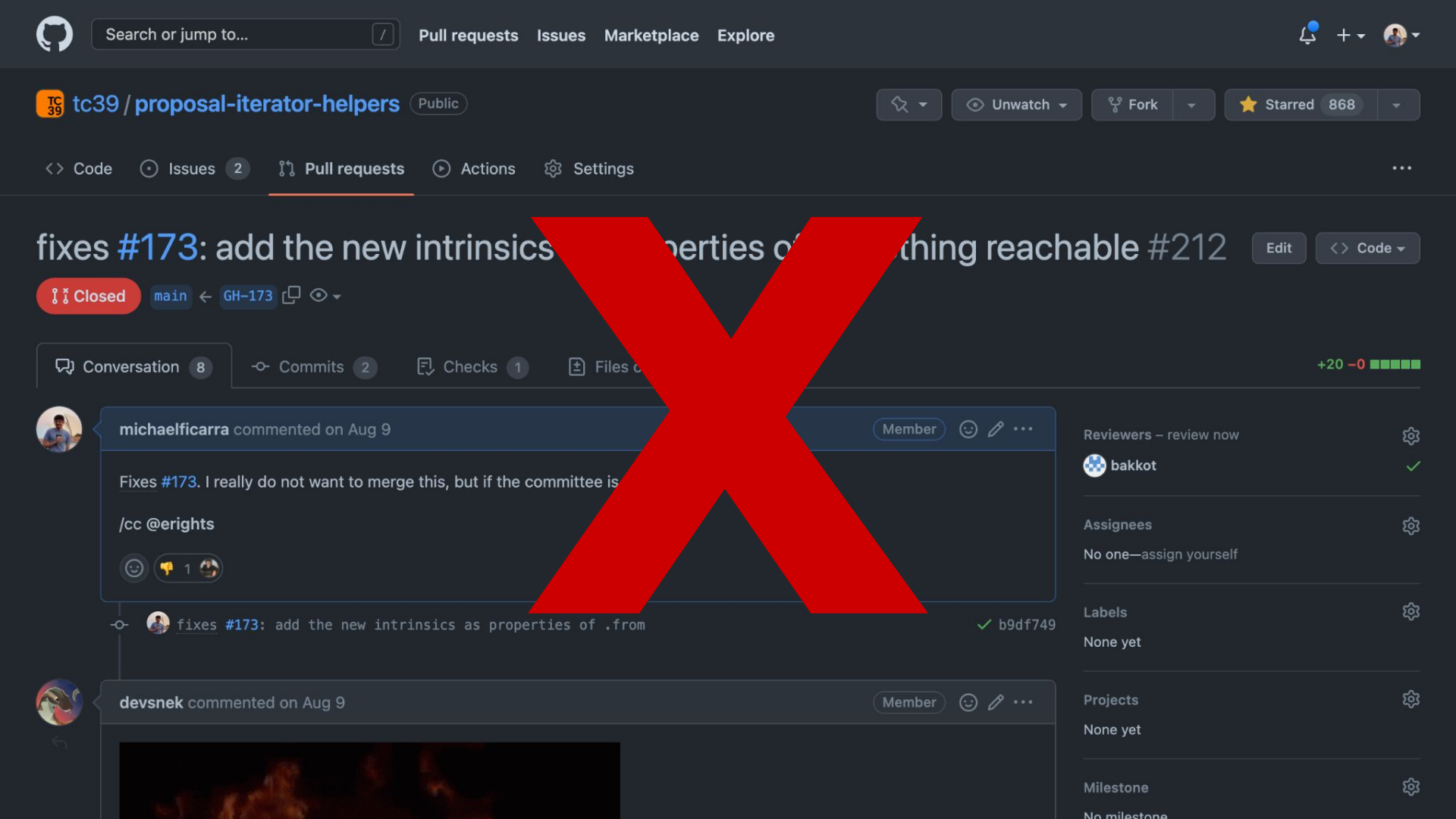
Labels

None yet

Projects

None yet


tc39 / proposal-iterator-helpers
Public
Code
Issues 7
Pull requests 4
Discussions
Actions
fixes #115: @@toStringTag is an accessor no
Open
Conversation 2
Files changed 1
michaelficarra comm
Fixes #115. Option 1: make
or.
1
2
fixes #1
gTag
r now
ljhark
ays ago
spec.html
Add more commits by pushing to the GH-115 branch on tc39/proposal-iterator-helper
This branch has not been deployed
No deployments
tc39 / proposal-iterator-helpers
Public
Code
Issues 7
Pull requests 4
Discussions
Actions
fixes #115: @@toStringTag is writable now #2
Open
main
GH-115-2
Conversation 0
Checks 1
1
michaelficarra commented
#115: @@toStringTag
now
commits by pushing to the GH-115-2 branch on tc39/proposal-iterator-help
This branch has not been deployed
No deployments
At least 1 approving review is required by reviewers with write acc
1 successful check

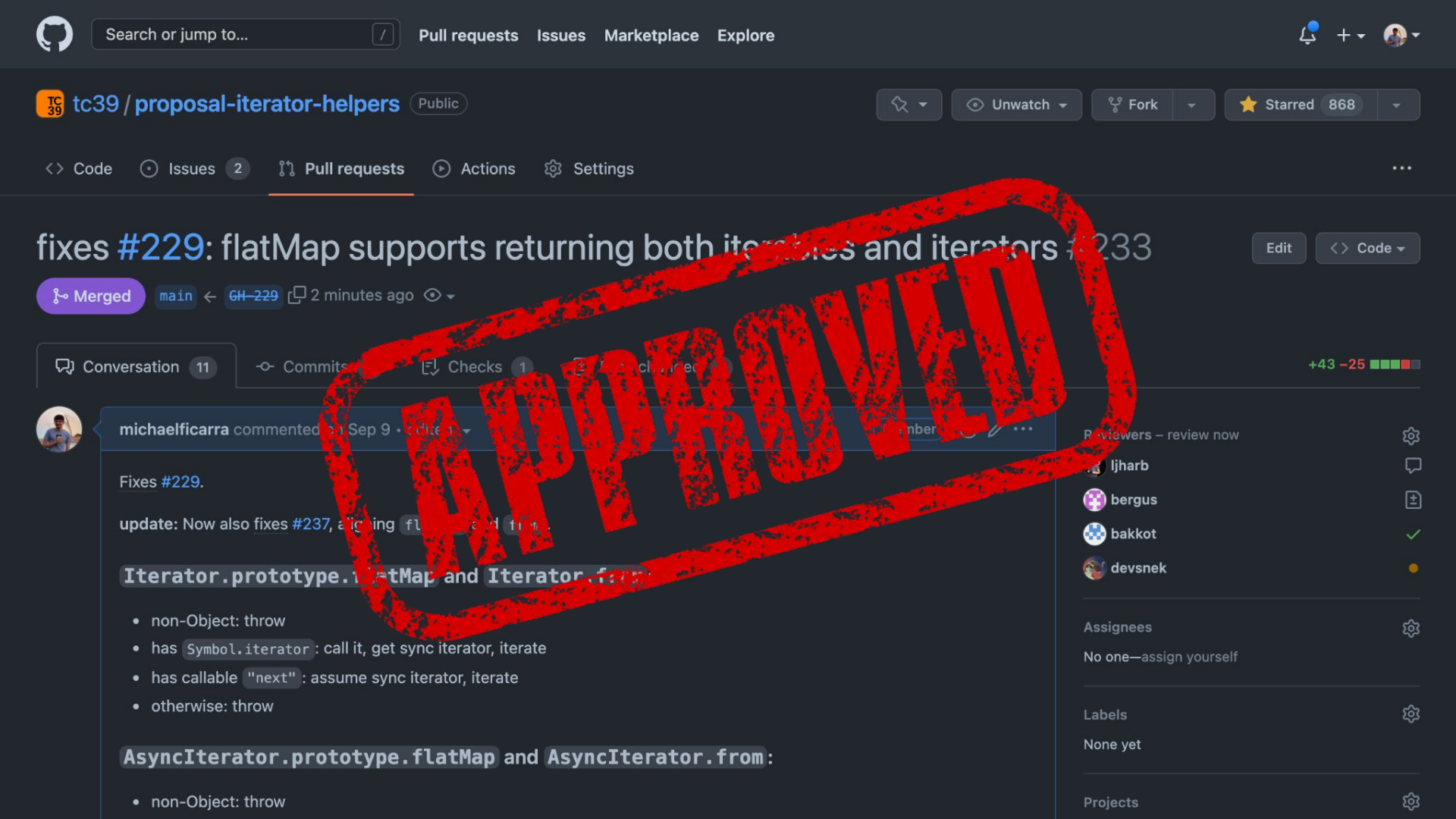

Search or jump to...
Pull requests
Issues
Marketplace
Explore
tc39 / proposal-iterator-helpers
Public
Unwatch
Fork
Starred 868
Code
Issues 2
Pull requests
Actions
Settings
fixes #173: add the new intrinsics
perties o
thing reachable #212
Edit
Code
Closed
main ← GH-173
Conversation 8
Commits 2
Checks 1
Files c
+20 −0
michaelficarra commented on Aug 9
Member
Fixes #173. I really do not want to merge this, but if the committee is
/cc @erights
1
fixes #173: add the new intrinsics as properties of .from
b9df749
devsnek commented on Aug 9
Member
Reviewers – review now
bakkot
Assignees
No one—assign yourself
Labels
None yet
Projects
None yet
Milestone

should flatMap flatten iterables or iterators? (#229)

1. `Iterator.prototype.flatMap` aligns with `Iterator.from`
    a. non-Object: throw
    b. has `Symbol.iterator`: call it, get sync iterator, iterate
    c. has callable `"next"`: assume sync iterator, iterate
    d. otherwise: throw
2. `AsyncIterator.prototype.flatMap` aligns with `AsyncIterator.from`
    a. non-Object: throw
    b. has `Symbol.asyncIterator`: call it, get async iterator, async iterate
    c. has `Symbol.iterator`: call it, get sync iterator, lift to async iterator, async iterate
    d. has callable `"next"`: assume async iterator, async iterate
    e. otherwise: throw
3. notably:
    a. flatMap now supports both iterators and iterables
        - behaviour aligns with respective `"from"` functions
        - because sync/async iterators are not distinguishable, no auto-lifting sync iterators to async
    b. Strings and other iterable primitives are not iterated
        - exception to be made for Tuples in the future

handling bare sync iterators in async from/flatMap

- solution: don't worry about it
  `for await (item of AsyncIterator.from(syncIterOfPromise)) print(item)`
  will see promises (iff `syncIterOfPromises` does not have Symbol.iterator)
- unexplored alternative:
  - await IteratorResult object to see if it's a promise (i.e. `(await x) !== x`)
    - if it is, assume we have an async iterator, use the `.value` property from the awaited value
    - if it is not, await the `.value` property and use the result


Search or jump to...
Pull requests
Issues
Marketplace
Explore
tc39 / proposal-iterator-helpers
Public
Unwatch
Fork
Starred 868
Code
Issues 2
Pull requests
Actions
Settings
fixes #229: flatMap supports returning both iterables and iterators #233
Edit
Code
Merged
main
GH-229
2 minutes ago
Conversation 11
Commits
Checks 1
+43 −25
michaelficarra commented on Sep 9
Fixes #229.
update: Now also fixes #237, aligning
Iterator.prototype.flatMap and Iterator.from
non-Object: throw
has Symbol.iterator: call it, get sync iterator, iterate
has callable "next": assume sync iterator, iterate
otherwise: throw
AsyncIterator.prototype.flatMap and AsyncIterator.from:
non-Object: throw
Reviewers – review now
ljharb
bergus
bakkot
devsnek
Assignees
No one—assign yourself
Labels
None yet
Projects
APPROVED

Does this pass through `return`/t synchron #223

Edit New issue

Closed 9 comments

glasser (David Glasser) on Aug 25

One challenge I have found with using async iteration is that it is challen ionality like `map` in a way that works properly with `return` / `throw`. For example, it would look like o ke:

```
async function* map(it, f) {
  for await (const x of it) {
    yield f(x);
  }
}
```

but calling `.return` on the value returned from `map` doesn't actually call `it.return` immediately if it's blocked on reading from `it`, which can have poor results. You can build a `map` that does pass through `it.return` immediately but it's much harder than writing an async generator. Lots of discussion about that on this issue.

So my question, having recently discovered this proposal, is: does this proposal help with this concern? If you `return` an async iterator returned by `.map(fn)` for example, is that passed through to the original iterator immediately, or only if it's currently yielding? I can't figure out the answer myself from README.md or DETAILs.md, and I don't understand the

Assignees
No one—assign yourself

Labels
needs resolution ×

Projects
None yet

Milestone
No milestone

Development
Create a branch for this issue or link a pull request.

proposal summary

- Iterator()
- Iterator.from(O)
- Iterator.prototype
  - .constructor
  - .map(mapperWithCounter)
  - .filter(predicateWithCounter)
  - .take(limit)
  - .drop(limit)
  - .flatMap(mapperWithCounter)
  - .reduce(reducerWithCounter [, initialValue])
  - .toArray()
  - .forEach(effectWithCounter)
  - .some(predicateWithCounter)
  - .every(predicateWithCounter)
  - .find(predicateWithCounter)
  - [Symbol.toStringTag]
  - .toAsync()

- AsyncIterator()
- AsyncIterator.from(O)
- AsyncIterator.prototype
  - .constructor
  - .map(mapperWithCounter)
  - .filter(predicateWithCounter)
  - .take(limit)
  - .drop(limit)
  - .flatMap(mapperWithCounter)
  - .reduce(reducerWithCounter [, initialValue])
  - .toArray()
  - .forEach(effectWithCounter)
  - .some(predicateWithCounter)
  - .every(predicateWithCounter)
  - .find(predicateWithCounter)
  - [Symbol.toStringTag]

other noteworthy changes

fixes #230: don't await non-objects returned from functions passed to AsyncIterator HoFs #239

Edit Code

Merged main ← GH-230 22 seconds ago

Conversation 6 Commits 4 Checks 1 Files changed 1 +22 −8

michaelficarra commented 23 days ago Member

Fixes #230. This is only observable in the number of ticks taken to complete the method. It is always at least 1 because the async iteration protocol itself requires awaiting the IteratorResult object, so no Zalgo issues.

fixes #230: don't await non-objects returned from AsyncIterator HoFs 2ae4395

bakkot approved these changes 23 days ago View changes

bakkot left a comment Member

This is a little bit weird - we don't have any other fast paths like this - but it doesn't seem harmful, and I don't see a reason to take more time than necessary here.

Reviewers – review now

bakkot

Assignees

No one—assign yourself

Labels

None yet

Projects

None yet

Search or jump to... /

Pull requests Issues Marketplace Explore

tc39 / ecma262 Public

Unwatch Fork Starred 13.7k

Code Issues 290 Pull requests 96 Actions Projects 1 Wiki 1 Settings Releases 21

Editorial: support built-in async functions #2942

Edit Code

Open main ← add-builtin-async-function-objects

Conversation 1 Commits 3 Checks 6 Files changed 1 +65 −5

michaelficarra 1 hour ago Member

This adds support for built-in async functions, as needed by the iterator helpers and `Array.fromAsync` proposals.

Editorial: support built-in async functions ✓ 0667b31

michaelficarra added the editorial change label 1 hour ago

michaelficarra requested a review from **bakkot** 1 hour ago

This was referenced 44 minutes ago

remove built-in async function infrastructure from this proposal tc39/proposal-iterator-helpers#245 Merged

Reviewers – review now

jmdyck

bakkot

Still in progress? Convert to draft

Assignees – assign yourself

Labels

editorial change ×

Projects

Milestone

10.4 Built-in Async Function Objects

Built-in async function objects are built-in function objects that provide alternative [[Call]] and [[Construct]] internal methods that conform to the following definitions:

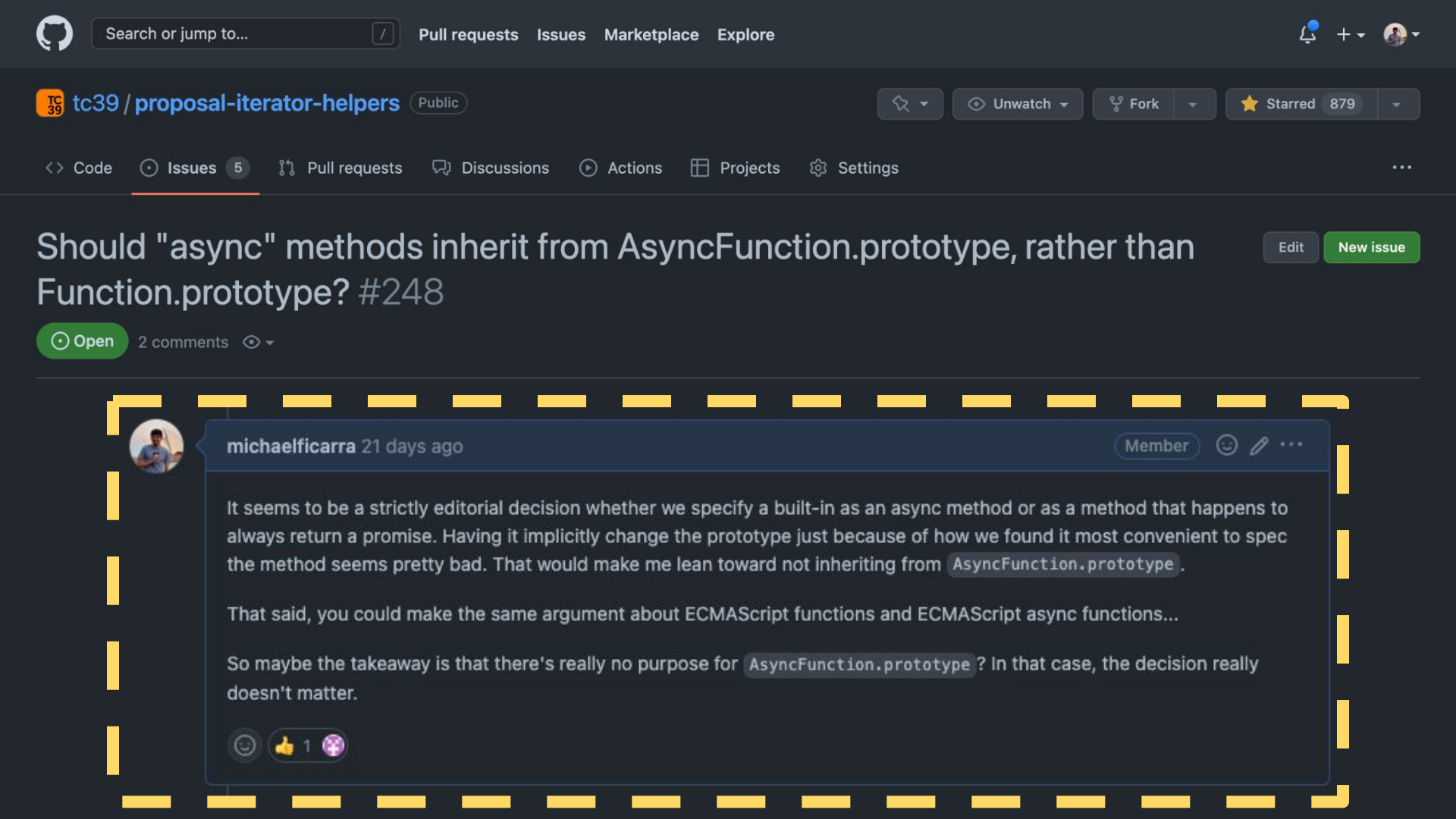
10.4.1 [[Call]] (*thisArgument*, *argumentsList*)

The [[Call]] internal method of a built-in async function object *F* takes arguments *thisArgument* (an ECMAScript language value) and *argumentsList* (a List of ECMAScript language values) and returns a normal completion containing an ECMAScript language value. It performs the following steps when called:

1. Let *callerContext* be the running execution context.
2. If *callerContext* is not already suspended, suspend *callerContext*.
3. Let *calleeContext* be a new execution context.
4. Set the Function of *calleeContext* to *F*.
5. Let *calleeRealm* be *F*.[[Realm]].
6. Set the Realm of *calleeContext* to *calleeRealm*.
7. Set the ScriptOrModule of *calleeContext* to **null**.
8. Perform any necessary implementation-defined initialization of *calleeContext*.
9. Push *calleeContext* onto the execution context stack; *calleeContext* is now the running execution context.
10. Let *promiseCapability* be ! NewPromiseCapability(%Promise%).
11. Let *resultsClosure* be a new Abstract Closure that captures *F*, *thisArgument*, and *argumentsList* and performs the following steps when called:
    a. Return the result of evaluating *F* in a manner that conforms to the specification of *F*. *thisArgument* is the **this** value, *argumentsList* provides the named parameters, and the NewTarget value is **undefined**.
12. Perform AsyncFunctionStart(*promiseCapability*, *resultsClosure*).
13. Remove *calleeContext* from the execution context stack and restore *callerContext* as the running execution context.
14. Return *promiseCapability*.

10.4.2 [[Construct]] (*argumentsList*, *newTarget*)

new open questions


Search or jump to...
Pull requests
Issues
Marketplace
Explore
tc39 / proposal-iterator-helpers
Public
Unwatch
Fork
Starred 879
Code
Issues 5
Pull requests
Discussions
Actions
Projects
Settings
Should "async" methods inherit from AsyncFunction.prototype, rather than Function.prototype? #248
Edit
New issue
Open
2 comments
michaelficarra 21 days ago
Member
It seems to be a strictly editorial decision whether we specify a built-in as an async method or as a method that happens to always return a promise. Having it implicitly change the prototype just because of how we found it most convenient to spec the method seems pretty bad. That would make me lean toward not inheriting from AsyncFunction.prototype.
That said, you could make the same argument about ECMAScript functions and ECMAScript async functions...
So maybe the takeaway is that there's really no purpose for AsyncFunction.prototype? In that case, the decision really doesn't matter.
1

stage 3?