JavaScript WeakRefs and TC39 standardization

Daniel Ehrenberg
Igalia, in partnership with Bloomberg
D2 2019

igalia
littledan@igalia.com

我

- Daniel Ehrenberg
- @littledan
- Delegate in TC39


家
Les Roquetes del Garraf, Europe

工作

igalia

Embedded WebKit and Chromium development

Mesa and GStreamer drivers

CSS, ARIA, WebAssembly, MathML, JavaScript standards+implementation in web browsers

TC39

igalia

littledan@igalia.com

WeakRefs: Motivation and use cases

littledan@igalia.com

In-memory cache

- Remote resources with identifiers
- Cache locally to avoid repeat lookups
- Tradeoff: Cache hit rate vs memory usage
- Idea:
  - Hold things in the cache if the resource is being used
  - Remove from cache when it's garbage collected???
  - (May be combined with LRU)

littledan@igalia.com

WebAssembly memory management

- WebAssembly is based on a big TypedArray: malloc and free inside
- How can allocations in WebAssembly be exposed to JavaScript?
  - Object wrapping a range in the TypedArray
    - Expect user to explicitly call obj.free() method? Too hard.
    - Automatically free when no longer referenced???

littledan@igalia.com

Avoiding stranded resources

- File handles, sockets, etc
- When no one references them:
  Resource leak until end of process
- Possible solution:
  Trigger error/log warning when a resource is stranded

littledan@igalia.com

Common capability: connecting to GC

- These examples require tying into GC:
  weak references and finalizers
- JavaScript doesn't give that access!
- Why? Interoperability/stability goals

littledan@igalia.com

Client-side API Design Principles

A Collection of Interesting Ideas, 12 December 2019

Editors:

Travis Leithead (Microsoft)

Sangwhan Moon (Invited Expert)

§ 1.3. Do not expose garbage collection

Web APIs should not expose a way for author code to deduce when/if garbage collection of JavaScript objects has run.

The reason for this is somewhat subtle. The more that Web API semantics are affected by garbage collection timing (or whether objects are collected at all), the more programs will be affected by changes in this timing. But user agents differ significantly in both the timing of garbage collection

New tradeoffs with new information

- Eventually, committee was convinced that we *should* add WeakRefs
- WebAssembly use case was especially persuasive
  - JS devs have been telling us this is needed for decades
- Now, the proposal is at Stage 3 in TC39!

littledan@igalia.com

Who is TC39?

- A committee of Ecma, with…
- JS developers
- JavaScript engines
- Transpilers
- Frameworks, libraries
- Academics
- Major websites/app platforms
- Now, some Chinese companies!
- etc

Search...

TABLE OF CONTENTS

Draft ECMA-262 / April 23, 2018

ECMAScript® 2019 Language Specification

ecma INTERNATIONAL

Contributing to this Specification

This specification is developed on GitHub with the help of the ECMAScript community. There are a number of ways to contribute to the development of this specification:

GitHub Repository: https://github.com/tc39/ecma262
Issues: All Issues, File a New Issue
Pull Requests: All Pull Requests, Create a New Pull Request
Test Suite: Test262
Editor: Brian Terlson (E-mail, Twitter, GitHub)

Community:

- Mailing list: es-discuss
- IRC: #tc39 on freenode

tc39 / ecma262

Unwatch 836 Star 6,408 Fork 446

Code Issues 157 Pull requests 52 Projects 0 Wiki Insights Settings

Status, process, and documents for ECMA262 https://tc39.github.io/ecma262/

Edit

ecmascript javascript Manage topics

1,191 commits 10 branches 10 releases 85 contributors

Branch: master New pull request Create new file Upload files Find file Clone or download

jmdyck and bterlson Editorial: Misc Editorial (#1182) ... Latest commit cbe4679 20 hours ago

img	Editorial: Store and rethrow module instantiation/evaluation errors	9 months ago
scripts	Meta: Build ES2016 in a subfolder.	2 years ago
workingdocs	Editorial: Use HTTPS for ecma-international.org URLs (#1017)	6 months ago
.editorconfig	Add .editorconfig	3 years ago
.gitignore	Editorial: reword InstanceofOperator copy and rename vars (closes #894)	11 months ago
.travis.yml	meta: require node 8 to build the spec	4 months ago
CODE_OF_CONDUCT.md	Add CoC.md reference file (#1168)	13 days ago
CONTRIBUTING.md	Meta: Add paragraph about resolving conflicts with master. (#1099)	2 months ago
FAQ.md	Fix example file name in FAQ.	3 years ago
README.md	meta: Add Community section to readme.md	a year ago
figure-2.uxf	Editorial: fix size of figure 2.	2 years ago

Meetings

- Every two months
- For three days
- Discuss language changes
- Seek consensus on proposal
  Stage advancement

igalia

6. Web compatibility issues / Needs Consensus PRs

✓	timebox	topic
✓	30m	Normative: Fix extending null
✓	5m	Normative: Make super() throw after evaluating args
✓	10m	Normative: make async iterators next/return/throw not p `undefined` when value is absent
✓	25m	Normative: Eliminate extra environment for eval in para initializers redux (slides)
✓	25m	Normative: TypedArray on prototypes web reality (low pr (slides)
✓	15m	Normative: make EnumerableOwnPropertyNames ordere

7. Overflow from previous meeting

✓	timebox	topic	presenter

8. Short (≤30m) Timeboxed Discussions

✓	timebox	topic
✓	20m	async-of grammar ambiguity
✓	30m	RegExp match indices performance feedback (slides)
~	30m	Policy on published code/polyfills in proposal repos
✓	30m (could be 10m)	JSExplain demo (code, demo)

9. Proposals

✓ represents an agenda item which has been presented, and does not indic

✓	stage	timebox	topic	present
✓	3	15m	Intl.RelativeTimeFormat for stage 4 (slides)	Zibi Bra
✓	3	15m	For-in order for Stage 4 (PR, tests, slides)	Kevin Gi

TC39 stages

- Stage 1: An idea under discussion
- Stage 2: We want to do this, and we have a first draft.

  "On the roadmap"
- Stage 3: Basically final draft; ready to go
- Stage 4: 2+ implementations, tests ⇒ standard

littledan@igalia.com

Consensus-based decision-making

- TC39 doesn't vote on what the language will be
  - (rarely vote on technicalities/procedural matters)
- TC39 is *consensus-seeking*
  - We work together to meet everyone's goals
  - "Does anyone object to stage advancement?"
  - Objections must have rationale, appropriate to stage
  - Technical concerns should be raised early

littledan@igalia.com

Consensus-based decision-making

- Consensus is *established* at some point,
  but could always be revisited with new consensus.
  - Established consensus cannot be "vetoed"/outvoted later
  - Technical concerns should be raised as early as possible

- We assume *good faith*:
  True motivations are given for objections
  Working together with trust
- All TC39 delegates are considered equal

littledan@igalia.com

Stage 4 features

2+ implementations, tests $\Rightarrow$ standard

littledan@igalia.com

BigInt: Stage 4!

littledan@igalia.com

```
const x = 2 ** 53;
```

$\Rightarrow$ `x === 9007199254740992`

```
const y = x + 1;
```

$\Rightarrow$ `y === 9007199254740992`

littledan@igalia.com

```
const x = 2n ** 53n;
          ⇒ x === 9007199254740992n
const y = x + 1n;
          ⇒ y === 9007199254740993n
```


littledan@igalia.com

```
const x = 2n ** 53n;
```

⇒ `x === 9007199254740992n`

```
const y = x + 1n;
```

⇒ `y === 9007199254740993n`

n stands for BigIntn

littledan@igalia.com

67 

68 

Beta 

10.4 

Flag 

Dynamic import(): Stage 4

Domenic Denicola

littledan@igalia.com

Example

```
<script>
  const main = document.querySelector("main");
  for (const link of document.querySelectorAll("nav > a")) {
    link.addEventListener("click", e => {
      e.preventDefault();

      import(`./section-modules/${link.dataset.entryModule}.js`)
        .then(module => {
          module.loadPageInto(main);
        })
        .catch(err => {
          main.textContent = err.message;
        });
    });
  }
</script>
```

Intl.RelativeTimeFormat: Stage 4

Zibi Braniecki

littledan@igalia.com

Intl.RelativeTimeFormat

- Shipped in Chrome and Firefox

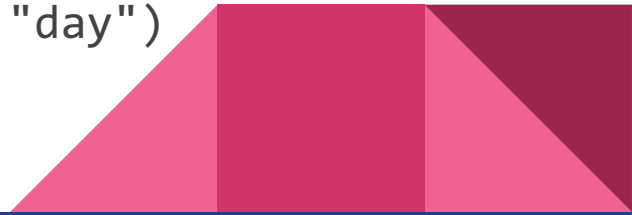
```
let rtf = new Intl.RelativeTimeFormat("en");

rtf.format(100, "day");
// "in 100 days"



new Intl.RelativeTimeFormat("zh").format(100, "day")

// "100天后"
```



Optional chaining: Stage 4

Daniel Rosenwasser
Claude Pache
Gabriel Isenberg
Dustin Savery

littledan@igalia.com

Optional Property Access

```
let x = foo?.bar;

// Equivalent to...

let x = (foo !== null && foo !== undefined) ?
  foo.bar :
  undefined;
```

Collaboration with TypeScript in TC39

- TypeScript saw many feature requests for ?.
- TC39's effort was going slowly
- TypeScript aligns with JavaScript runtime semantics
  - TypeScript type system erased at compile time
  - JS defines runtime semantics
- Daniel Rosenwasser, TypeScript PM, came in push it forward
- Based on TC39 success, ?. shipping in TS 3.7
- Now, ?. will be part of ES2020

littledan@igalia.com

Nullish coalescing: Stage 4

Daniel Rosenwasser
Gabriel Isenberg

littledan@igalia.com

Nullish Coalescing

```
let x = foo() ?? bar();

// Equivalent to...

let tmp = foo();
let x = (tmp !== null && tmp !== undefined) ?
  tmp :
  bar();
```

Stage 3 features

Basically final draft; ready to go

littledan@igalia.com

WeakRefs: Stage 3

Sathya Gunasekaran
Till Schneidereit
Mark Miller
Dean Tribble

littledan@igalia.com

Read consistency

```
const w = new WeakRef(someObject);

...

if (w.deref()) {

  w.deref().foo(); // w.deref() here can not fail

}
```

Private fields and methods: Stage 3

littledan@igalia.com

Why?

```
class Counter extends HTMLElement {
  #x = 0;

  connectedCallback() {
    this.#render();
  }
  #render() {
    this.textContent =
        this.#x.toString();
  }
}
```

- Private methods encapsulate behavior
- You can access private fields inside private methods

is the new _

for strong encapsulation

__

Why strong encapsulation?

- Not all code needs this, but
- Library/framework authors may want to provide a stable API
- Common techniques can be hacked into
  - `_properties`
  - TypeScript `private`
- In practice, users depend on these internals, and then library authors cannot evolve beyond their old details
  - Node.js and Moment.js hit these issues
- Library users benefit from good maintenance

littledan@igalia.com

```
class PublicCounter {
  _x = 0;
}

let c = new PublicCounter();
console.log(c._x);    // 0
```

```
class PrivateCounter {
  #x = 0;
}

let p = new PrivateCounter();
console.log(p.#x);    // SyntaxError
```

Stage 3

littledan@igalia.com

Why not `private` keyword?

- In languages with types:
  `obj.x` can check whether `x` is private by looking at the type of `obj`
- JavaScript is dynamically typed
- ⇒ private vs public distinction needed at access point,
  not just definition
- # as part of the name was the cleanest, simplest solution we found
- We thought about many alternatives over 20 years; ask me later about any further possibilities

littledan@igalia.com

Stage 2 features

We want to do this, and we have a first draft

littledan@igalia.com

Decorators: Stage 2

Yehuda Katz
Ron Buckton

littledan@igalia.com

Syntax abstraction for attrs/props in Web Components

```
// Polymer abstraction
class XCustom extends PolymerElement {
  @property({ type: String, reflect: true
})
  address = '';
}
```

```
// Salesforce abstraction
import { api } from '@salesforce';

class InputAddress extends HTMLElement {
    @api address = '';
}
```

Example of using decorators to improve ergonomics.

Temporal: Stage 2

Maggie Pint
Philipp Dunkel

littledan@igalia.com

What time did this presentation start in Berlin?

```
let startDateTime =                    // Temporal.DateTime
    Temporal.now.dateTime().with({ hour: 14, minute: 00 });

let startAbsolute =                    // Temporal.Absolute
    startDateTime.inTimeZone(Temporal.now.timeZone());

let localizedToBerlin =                // Temporal.DateTime
    startAbsolute.inTimeZone("Europe/Berlin");

Intl.DateTimeFormat("zh", { hour: "numeric", minute: "numeric" })
    .format(localizedToBerlin);        // 上午7:00
```


littledan@igalia.com

Stage 1 features

An idea under discussion

littledan@igalia.com

Pipeline operator: Stage 1

Gilbert Garza
J.S. Choi
James DiGioia

littledan@igalia.com

library.js

```
export function doubleSay(str) {
  return str + ", " + str;
}
export function capitalize(str) {
  return str[0].toUpperCase() +
      str.substring(1);
}
export function exclaim(str) {
  return str + "!";
}
```

ordinary.js

```
import { doubleSay, capitalize, exclaim }
                    from "./library.js";
let result =
  exclaim(capitalize(doubleSay("hello")));
// ===> "Hello, hello!"
```

with-pipeline.js

```
import { doubleSay, capitalize, exclaim }
                    from "./library.js";
let result = "hello"
  |> doubleSay
  |> capitalize
  |> exclaim;
// ===> "Hello, hello!"
```


littledan@igalia.com

Records and Tuples: Stage 1

Robin Ricard
Richard Button

littledan@igalia.com

```
const marketData = #[
    #{ ticker: "AAPL", lastPrice: 195.855 },
    #{ ticker: "SPY", lastPrice: 286.53 },
];
```

Operator overloading: Stage 1

littledan@igalia.com

Vector overloading: Usage

```
// Usage example
import { Vector } from "./vector.mjs";
with operators from Vector;

new Vector([1, 2, 3]) + new Vector([4, 5, 6])   // ==> new Vector([5, 7, 9])
3 * new Vector([1, 2, 3])                       // ==> new Vector([3, 6, 9])
new Vector([1, 2, 3]) == new Vector([1, 2, 3])  // ==> true
(new Vector([1, 2, 3]))[1]                      // ==> 2
```

Stage 0 features

Not even really at a stage!

littledan@igalia.com

BigDecimal: Stage 0

Andrew Paprocki

igalia

littledan@igalia.com

“Why are Numbers broken in JS?”

littledan@igalia.com

Problem and solution (?)

```
// Number (binary 64-bit floating point)

js> 0.1 + 0.2

                  =>
                      0.30000000000000004

// BigDecimal (???)

js> 0.1d + 0.2d

                  =>                  0.3d
```


littledan@igalia.com

WeakRef and FinalizationGroup API

littledan@igalia.com

WeakRef

- `let weakRef = new WeakRef(obj)`
- `weakRef.deref()` `// ⇒ obj or undefined`

littledan@igalia.com

In-memory cache

```
// This technique is incomplete; see below.
function makeWeakCached(f) {
  const cache = new Map();
  return key => {
    const ref = cache.get(key);
    if (ref) {
      const cached = ref.deref();
      if (cached !== undefined) return cached;
    }

    const fresh = f(key);
    cache.set(key, new WeakRef(fresh));
    return fresh;
  };
}

var getImageCached = makeWeakCached(getImage);
```


littledan@igalia.com

FinalizationGroup

- `function cleanupCallback(holdingsIterator) { /* */ }`
- `let group = new FinalizationGroup(cleanupCallback)`
- `group.register(obj, holdings, unregisterToken)`
- `group.unregister(unregisterToken)`
- `/* implicitly */`
  `cleanupCallback([holdings][Symbol.iterator]())`

littledan@igalia.com

Post-mortem finalization

- The FinalizerGroup's callback iterator has `holdings`, not object

  ```
  group.register(obj, holdings)
  ```

- API doesn't provide access to object after it is collected
- "No resurrection"/bringing-back-from-the-dead
- Bringing an object back alive is causes problems;
  this API avoids them

littledan@igalia.com

In-memory cache with tombstone cleanup

```
// Fixed version that doesn't leak memory.
function makeWeakCached(f) {
  const cache = new Map();
  const cleanup = new FinalizationGroup(iterator => {
    for (const key of iterator) {
      // See note below on concurrency considerations.
      const ref = cache.get(key);
      if (ref && !ref.deref()) cache.delete(key);
    }
  });

  return key => {
    const ref = cache.get(key);
    if (ref) {
      const cached = ref.deref();
      // See note below on concurrency considerations.
      if (cached !== undefined) return cached;
    }

    const fresh = f(key);
    cache.set(key, new WeakRef(fresh));
    cleanup.register(fresh, key, key);
    return fresh;
  };
}

var getImageCached = makeWeakCached(getImage);
```

Stranded resource errors

```
class FileStream {
  static #cleanUp(holdings) {
    for (const file of holdings) {
      console.error(`File leaked: ${file}!`);
    }
  }

  static #finalizationGroup = new FinalizationGroup(this.#cleanUp);

  #file;

  constructor(fileName) {
    this.#file = new File(fileName);
    FileStream.#finalizationGroup.register(this, this.#file, this);
    // eagerly trigger async read of file contents into this.data
  }

  close() {
    FileStream.#finalizationGroup.unregister(this);
    File.close(this.#file);
    // other cleanup
  }

  async *[Symbol.iterator]() {
    // read data from this.#file
  }
}

const fs = new FileStream('path/to/some/file');

for await (const data of fs) {
  // do something
}
fs.close();
```


littledan@igalia.com

WebAssembly memory management

```
function makeAllocator(size, length) {
  const freeList = Array.from({length}, (v, i) => size * i);
  const memory = new ArrayBuffer(size * length);
  const finalizationGroup = new FinalizationGroup(
    iterator => freeList.unshift(...iterator));
  return { memory, size, freeList, finalizationGroup };
}

function allocate(allocator) {
  const { memory, size, freeList, finalizationGroup } = allocator;
  if (freeList.length === 0) throw new RangeError('out of memory');
  const index = freeList.shift();
  const buffer = new Uint8Array(memory, index * size, size);
  finalizationGroup.register(buffer, index);
  return buffer;
}
```


littledan@igalia.com

My suggestions for contributing to TC39 at different stages

littledan@igalia.com

Contributing to existing proposals

- The thing that you want to do may already be a proposal
- Proposal list https://github.com/tc39/proposals
- Contributions welcome!
- Proposals often stalled/slow because more work needed

littledan@igalia.com

Stage 0/1

- Document use cases
- Begin seeking feedback, and keep doing it the whole time!
- Discuss the problem space/big-picture questions
- Prototype implementations:
  unstable, rough, maybe downstream

littledan@igalia.com

Stage 2

- Nail down the proposal details, from discussion and prototyping
- Draft documentation and tests in the proposal repo
- Prototype implementations:
  Ideally nearing completion, but still considered unstable
- Distribute the prototype more broadly to gather more solid feedback

littledan@igalia.com

Stage 3

- Upstream tests into test262 and fill in any gaps.
- Place documentation in MDN and fill in the gaps
- Implement in engines and consider shipping
- Integrate usage into environments (e.g., the Web, Node.js)
- Distribute high-level explanations more broadly to developers

littledan@igalia.com

Stage 4

- Ideally nothing!
- Tie up any loose ends in the spec language
- Implement in trailing engines if needed
- Further changes: separate PRs/proposals

littledan@igalia.com

WeakRefs history and development in TC39

igalia

littledan@igalia.com

[[strawman:weak_refs]]

Rationale

The accepted weak maps API serves important use cases for weak references, but not all. Most importantly, the observer pattern relies on an observable object holding weak references to its observers. If an observer becomes otherwise collectable, the object it observes should not keep it alive. The observer pattern is very relevant to and common on the web, including:

- MVC and data binding frameworks
- reactive-style libraries
- reactive-style languages that compile to JS

The publish-subscribe pattern has similar characteristics.

It's worth noting that using weak references for the observer pattern ties the behavior of notification to the non-deterministic behavior of the garbage collector. If an object hasn't been collected yet but is intended not to be used anymore, it could still receive a notification. So a well-written observer should still usually be manually disconnected before it dies. But frameworks can use weak references as a back-stop to prevent leaks when their clients forget to disconnect observers.

TODO: Unify this page and weak_references into one strawman.

Examples

```
let obj = { ... };

let ref = new WeakRef(obj);
alert(ref.get() === obj); // true

obj = null; // make sure the scope chain isn't keeping obj alive
document.getElementById("big-red-button").onclick = function() {
    // ... obj has been garbage collected in the meantime ...
    alert(ref.get()); // null
};
```

A proposal for ES6 in 2010

Security

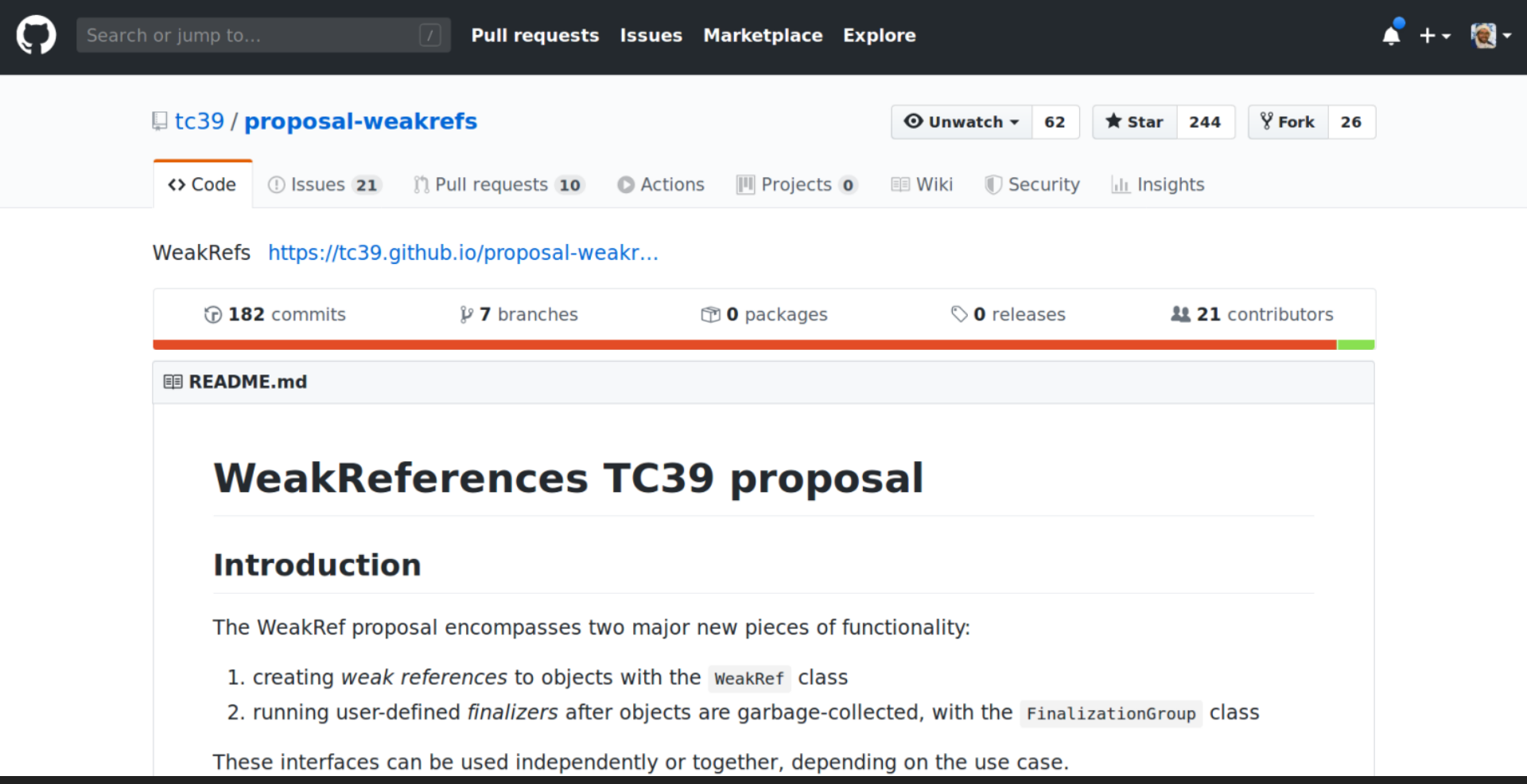
Weak references observe the behavior of the garbage collector, which can provide a channel of communication between otherwise separated partitions of the object graph. Security-sensitive code would most likely need to censor access to the WeakRef API from untrusted code. Within a realm, this could be achieved e.g. by creating custom module loaders. However, such restrictions do not enable one realm to

Progression in TC39 stage process

- Stage 1: March 2016
  - However, there was strong resistance
- More people are convinced about WebAssembly use case
- Stage 2: March 2018
- Stage 3: June 2019

- As the proposal progressed, more co-champions/helpers

littledan@igalia.com


Search or jump to...
Pull requests
Issues
Marketplace
Explore
tc39 / proposal-weakrefs
Unwatch 62
Star 244
Fork 26
Code
Issues 21
Pull requests 10
Actions
Projects 0
Wiki
Security
Insights
WeakRefs https://tc39.github.io/proposal-weakr...
182 commits
7 branches
0 packages
0 releases
21 contributors
README.md
WeakReferences TC39 proposal
Introduction
The WeakRef proposal encompasses two major new pieces of functionality:
1. creating weak references to objects with the WeakRef class
2. running user-defined finalizers after objects are garbage-collected, with the FinalizationGroup class
These interfaces can be used independently or together, depending on the use case.

Search...

TABLE OF CONTENTS

Stage 3 Draft / November 19, 2019

WeakRefs proposal

1 WeakRef Objects

A WeakRef is an object that is used to refer to a target object without preserving it from garbage collection. WeakRefs can dereference to allow access to the target object, if the target object hasn't been reclaimed by garbage collection.

1.1 The WeakRef Constructor

The WeakRef constructor:

- is the intrinsic object %WeakRef%.
- is the initial value of the **`WeakRef`** property of the global object.
- creates and initializes a new WeakRef object when called as a constructor.
- is not intended to be called as a function and will throw an exception when called in that manner.
- is designed to be subclassable. It may be used as the value in an **`extends`** clause of a class definition. Subclass constructors that intend to inherit the specified **`WeakRef`** behaviour must include a **`super`** call to the **`WeakRef`** constructor to create and initialize the subclass instance with the internal state necessary to support the **`WeakRef.prototype`** built-in methods.

WeakRef behavior invariants

littledan@igalia.com

WeakRef behavior invariants

- Problem: Different JS implementations have different GCs.
- ⇒ WeakRefs will go undefined at different times;
  ⇒ FinalizationGroup cleanup will happen at different times
- How can one program works across different JS engines?

- ~~Non-answer: Require everyone to use the same GC~~
- Answer: Define certain common properties among GCs

littledan@igalia.com

Consistency of multiple `.deref()` calls

- In a straight line of code, `.deref()` either returns the object, or undefined, not a mix
- Objects may be "collected" only when yielding to the event loop

```
const w = new WeakRef(obj);
// ...
if (w.deref()) {
  w.deref().foo();
  // w.deref() here can not fail
}
```


littledan@igalia.com

Definition of liveness

4.1.2 Liveness

For some object *obj*, a *hypothetical WeakRef-oblivious* execution with respect to *obj* is an execution whereby WeakRef.prototype.deref being called on a WeakRef whose referent is *obj* always returns **undefined**.

At any point during evaluation, an object *obj* is considered *live* if either of the following conditions is met:

- *obj* is included in any agent's [[KeptAlive]] List.
- There exists a valid future hypothetical WeakRef-oblivious execution with respect to *obj* that observes the Object value of *obj*.

Further invariants

- All the WeakRefs pointing to the same object turn to undefined at the same time, but the FinalizationGroup callback may be delayed until later.
- The unregisterToken is treated as a weak reference, not strong.
- If a strongly connected component of the object graph dies all at once, then no finalizer callbacks are called.

- See GitHub issues for details

littledan@igalia.com

Participating in TC39

littledan@igalia.com

Participating internationally

- Many TC39 members are facing similar issues:
  - English is a second language
    - You can participate with mostly written communication
  - Live outside the US
    - Many delegates live in Europe
  - Unable to attend most TC39 meetings in person
    - Instead, join by video call
- TC39's code of conduct prohibits discrimination on nationality
- Many TC39 members want to increase international participation
- TC39 participants represent ideas and organizations, not countries

littledan@igalia.com

Joining TC39

- Join TC39 as a member by joining Ecma
- Joining Ecma requires:
  - Signing IPR forms
    - Typical Ecma RAND policy
    - TC39-specific royalty-free agreement
  - Paying membership fee
- TC39 *delegates* represent member *organizations*
- Companies considering joining can provisionally attend TC39 as "prospective members"

littledan@igalia.com

Participating asynchronously on GitHub

- Most tasks don't take place in meetings:
  Most important technical work happens on GitHub
- Work on GitHub:
  - Some of the discussion about design
  - Specification text
  - Documentation
  - Tests
  - Implementations
- Non-members can contribute on GitHub! We encourage it.
  - Just sign non-member IPR form
- Meeting notes are published to GitHub
- Only members can take part in meetings and consensus process

littledan@igalia.com

TC39 changes over time

littledan@igalia.com

Older TC39 mode

- Specification: Big MS Word doc
- Communication: Meetings and es-discuss list
- Large, occasional specification; no stages

littledan@igalia.com

ECMAScript 2015 Language Specification

1 Scope

This Standard defines the ECMAScript 2015 general purpose programming language.

2 Conformance

A conforming implementation of ECMAScript must provide and support all the types, values, objects, properties, functions, and program syntax and semantics described in this specification.

A conforming implementation of ECMAScript must interpret source text input in conformance with the Unicode Standard, Version 5.1.0 or later and ISO/IEC 10646. If the adopted ISO/IEC 10646-1 subset is not otherwise specified, it is presumed to be the Unicode set, collection 10646.

A conforming implementation of ECMAScript that provides an application programming interface that supports

Search...

TABLE OF CONTENTS

Draft ECMA-262 / April 23, 2018

ECMAScript® 2019 Language Specification

ecma INTERNATIONAL

Contributing to this Specification

This specification is developed on GitHub with the help of the ECMAScript community. There are a number of ways to contribute to the development of this specification:

GitHub Repository: https://github.com/tc39/ecma262
Issues: All Issues, File a New Issue
Pull Requests: All Pull Requests, Create a New Pull Request
Test Suite: Test262
Editor: Brian Terlson (E-mail, Twitter, GitHub)

Community:

- Mailing list: es-discuss
- IRC: #tc39 on freenode

tc39 / ecma262

Unwatch 836 Star 6,408 Fork 446

Code Issues 157 Pull requests 52 Projects 0 Wiki Insights Settings

Status, process, and documents for ECMA262 https://tc39.github.io/ecma262/

Edit

ecmascript javascript Manage topics

1,191 commits 10 branches 10 releases 85 contributors

Branch: master New pull request Create new file Upload files Find file Clone or download

jmdyck and bterlson Editorial: Misc Editorial (#1182) ... Latest commit cbe4679 20 hours ago

img	Editorial: Store and rethrow module instantiation/evaluation errors	9 months ago
scripts	Meta: Build ES2016 in a subfolder.	2 years ago
workingdocs	Editorial: Use HTTPS for ecma-international.org URLs (#1017)	6 months ago
.editorconfig	Add .editorconfig	3 years ago
.gitignore	Editorial: reword InstanceofOperator copy and rename vars (closes #894)	11 months ago
.travis.yml	meta: require node 8 to build the spec	4 months ago
CODE_OF_CONDUCT.md	Add CoC.md reference file (#1168)	13 days ago
CONTRIBUTING.md	Meta: Add paragraph about resolving conflicts with master. (#1099)	2 months ago
FAQ.md	Fix example file name in FAQ.	3 years ago
README.md	meta: Add Community section to readme.md	a year ago
figure-2.uxf	Editorial: fix size of figure 2.	2 years ago

Invited experts

Ecma Rules (June 2019)

6.2.6
Individuals can participate in the work of a TC as invited experts. They participate only at the invitation of the SG at the request of the TC. They have no voting rights. Invited experts shall comply with the Ecma policies and sign the appropriate form before participating. The invitation to participate may be withdrawn by the SG at any time.

No description, website, or topics provided.

Edit

Manage topics

56 commits | **2** branches | **0** packages | **0** releases | **3** contributors

Branch: **master** | **New pull request** | **Create new file** | **Upload files** | **Find file** | **Clone or download**

bkardell Create notes-2019-11-21.md ... Latest commit `c094b13` 23 hours ago

educators	Create notes-2019-11-14.md	17 days ago
frameworks	Create notes-2019-11-21.md	23 hours ago
tools	Create notes-2019-11-26.md	23 hours ago
CODE_OF_CONDUCT.md	Create CODE_OF_CONDUCT.md	11 months ago
README.md	Add Tools and transpilers group to existing groups list	3 months ago

README.md

JS outreach groups

Some TC39 delegates are organizing meetings with various stakeholders in order to get more input in the process and share information with the community. Current calls include:

- Educators
- Frameworks
- Tools and transpilers

TC 39

Sign Up | Log In

all categories | all tags | Categories | Latest | Top

Category	Topics	Top
Announcements Welcome to TC39's focal communication hub.	0	
Proposals Discuss existing ECMAScript proposals.	2	
Ideas New ideas, proposals looking for a champion, and proposal mentoring. Get feedback on your new ideas here: what's been done, where to find prior art, the polish your proposal needs to be considered by the committee.	27	
I have questions Specifying a language is a complex affair, filled with confusing language and sometimes bewildering processes. It's okay not to understand, and we encourage you to ask questions! Spec Reading Delegates AMA	9	

Top		
	What should the categories be? Meta	18 Jun 7
	bigint enhancements. Proposals proposal	11 Oct 15
	Try-catch oneliner Ideas proposal	14 14d
	☑ Force HTTPS usage Meta	9 Apr 5
	New well-known Symbol: Symbol.typeofTag Ideas proposal	12 17d
G	JSON: Add Datetime, Timedelta and Binary Data Types	9 9d

Introduction

Like the technical community as a whole, TC39 is made up of a mixture of professionals and volunteers from all over the world. To ensure a fair and balanced standards process, to avoid communication issues and unhappiness, and to promote inclusiveness, we have a few ground rules that we ask people to adhere to.

This isn't an exhaustive list of things that you can't do. Rather, take it in the spirit in which it's intended a guide to make it easier to enrich all of us and the technical communities in which we participate and empower others to speak.

This Code of Conduct is enforced within all spaces managed by TC39. This includes IRC channels moderated by TC39, mailing lists such as esdiscuss, issue trackers on projects hosted by TC39, and TC39 events and meetings.

If you believe someone is violating the Code of Conduct, we ask that you report it by emailing tc39-conduct-reports@googlegroups.com. For more details, please see our Reporting Guidelines.

Be respectful

Respect is a fundamental value of the standardization work. Not all of us will agree all the time, but disagreement is no excuse for poor behavior and poor manners. We might all

README.md

Ecma International's TC39 is the standards committee which defines ECMAScript (JavaScript). This repository is intended to provide documentation into how TC39 works.

This repository is an early work in progress!

For an introduction to getting involved in TC39, see CONTRIBUTING.md.

Table of Contents

TC39

TC39 | Specs | Contribute | State of Proposals

Specifying JavaScript.

TC39

Ecma International's TC39 is a group of JavaScript developers, implementers, academics, and more, collaborating with the community to maintain and evolve the definition of JavaScript.

We are part of ecma INTERNATIONAL

Contribute

TC39 welcomes contributions. You can help by giving feedback on proposals, improving documentation, writing tests or implementations, or suggesting language feature ideas. See our contributor guide for details.

To participate in TC39 meetings as a member, join Ecma.

Specs

We develop the JavaScript (formally, ECMAScript) specification on GitHub and meet every two months to discuss proposals. To learn more about the process, please take a look at the four stages for new language feature proposals. See our meeting agendas and minutes to learn more.

State of Proposals

Current Candidates for the Specification

JavaScript and evidence-based language design

By Yulia Startsev

Posted on May 29, 2019 in Featured Article, JavaScript, and Standards

Author's note: Hi, I'm an engineer at Mozilla working on the Firefox DevTools server. I'm also a TC39 representative. This post focuses on some of the experiments I am trying out at the TC39, the standards body that manages the JavaScript specification. A follow up post will follow...

Integrating the traditional and the new values

- First-principles reasoning
- "The future is bigger than the past"
- Long, complete design cycles
- Rigorous debate
- Welcoming participants who join
- Language specialists/theorists

- Practicality
- Integration into the existing ecosystem
- Quick, incremental iteration
- Letting points be made without interruption
- Actively seeking out feedback
- More JS/frontend dev participation

All of these are useful,
complementary perspectives

littledan@igalia.com

Geekbang. 极客邦科技 | InfoQ 极客传媒

InfoQ官网 全新改版上线

促进软件开发领域知识与创新的传播

关注InfoQ网站
第一时间浏览原创IT新闻资讯

免费下载迷你书
阅读一线开发者的技术干货

谢谢!

https://tc39.es

littledan@igalia.com