

ECMAScriptの使い方

自己紹介

- >> Name : **azu**
- >> Twitter : [@azu_re](#)
- >> Website: [Web scratch](#), [JSer.info](#)
- >> ⚡ [ECMAScript Daily](#)



目的

- >> ECMAScriptを使えるようになる
- >> 自律的なECMAScriptについての情報を検索できるようになる

以前の話したもの

>> ECMAScript as a Living Standard

>> ECMAScriptは毎年更新されるし、最新版はGitHubで毎日更新されてる

>> Living Standardになってるよ
という話

ECMAScriptとは

- » ECMAScriptはEcma Internationalという団体によって標準化されている仕様
- » Ecma内のTC39という技術委員会によって、どのような機能を仕様へ入れるかを議論、決定
- » TC39はMicroSoft、Mozilla、Google、AppleといったブラウザベンダーやECMAScriptに関心のある企業などによって構成

ECMAScriptのバージョン

バージョン

日付

1	1997年6月
2	1998年6月
3	1999年12月
4	策定されずに破棄(ES4)
5	2009年12月
5.1	2011年6月
2015	2015年6月
2016	2016年6月
2017	2017年6月

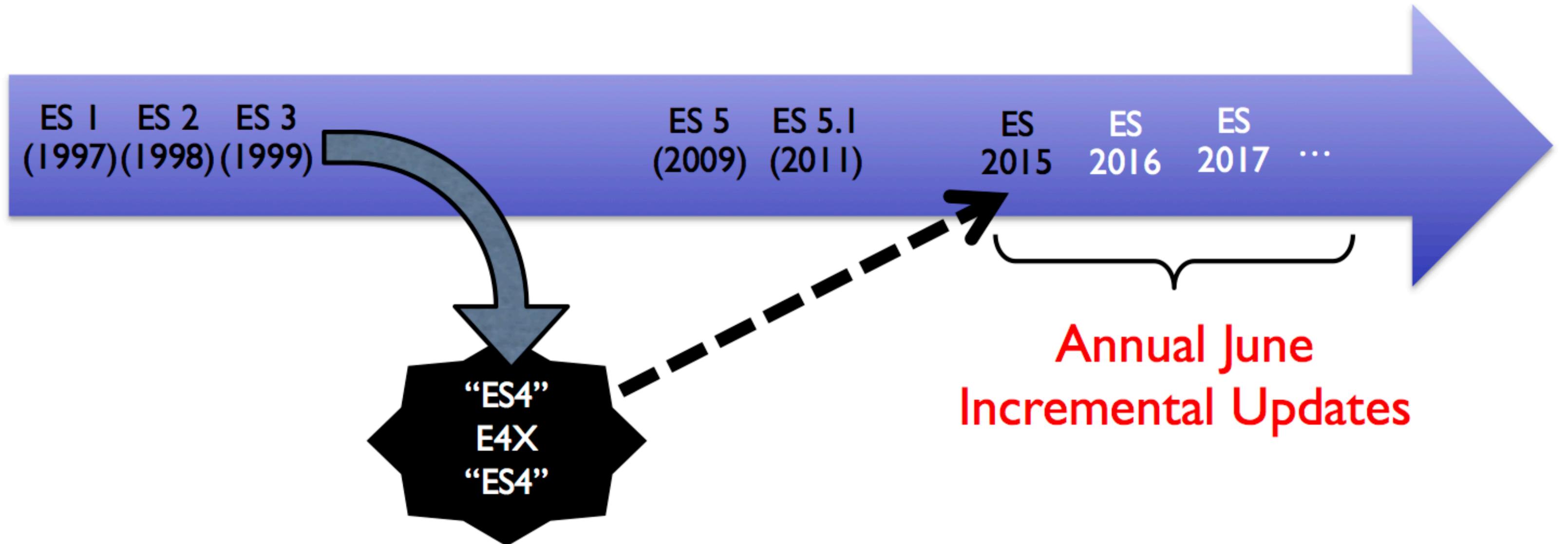
ECMAScriptのバージョンの歴史

- >> ES5.1からES2015がでるまで4年もかかっている
- >> ES2015以降は毎年リリースされている

バージョン	リリース時期
1	1997年6月
2	1998年6月
3	1999年12月
4	破棄[^ES4]
5	2009年12月
5.1	2011年6月
2015	2015年6月
2016	2016年6月
2017	2017年6月

The ECMAScript Standard Timeline

Release trains are now leaving the station



仕様策定のプロセスの変化

>> ES2015以前

>> すべての仕様の合意がとれてから一括リリース

>> ES2016以降

>> 合意がとれた仕様から順次リリース

>> ECMAScriptを毎年リリースするために策定プロセスを変更

ES2016以降の仕様策定プロセス

- ≫ 仕様に追加する機能（API、構文など）をそれぞれ個別のプロポーザル（提案書）として進める
- ≫ 現在策定中のプロポーザルは🕒 [tc39/proposals](https://tc39.org/proposals)にて公開
- ≫ 各プロポーザルは責任者であるチャンピオンとステージ（Stage）と呼ばれる0から4の5段階の状態をもつ
 - ≫ チャンピオンが責任持ってプロポーザルを進める

プロポーザルのステージ

ステージ	名前	ステージの概要
0	Strawman	アイデアの段階
1	Proposal	機能提案の段階
2	Draft	機能の仕様書ドラフトを作成した段階
3	Candidate	仕様としては完成しており、ブラウザの実装やフィードバックを求める段階
4	Finished	仕様策定が完了し、2つ以上の実装が存在している。 <u>テスト</u> がある。 正式にECMAScriptにマージできる段階

プロポーザルのステージの進み方

- >> 2ヶ月に1度行われるTC39のミーティングでプロポーザルのステージを更新
- >> ミーティングの議事録は🕒 [tc39/tc39-notes](https://tc39.github.io/tc39-notes/)で公開
- >> 毎年のECMAScriptをリリースするタイミング(6月)で、Stage 4のプロポーザルをマージ
- >> ECMAScript 20XXとしてリリース

なぜ仕様策定プロセスが変わったのか

- >> ES2015以前: すべての仕様の合意が取れてからリリース
- >> ES2016以降: 合意が取れた仕様からリリース
- >> 変更理由: ECMAScriptのリリースに長い歳月がかかり言語の進化が停滞した
- >> 歴史的失敗: ES4では多くの変更を入れることを試みたが、TC39内でも意見が分かれ最終的に合意できなかった
 - >> これにより言語の発展が数年間停滞した¹

¹ Programming Language Standardization: Patterns for Participationを参照

仕様策定プロセス変更の影響

- ≫ ECMAScriptに入れようとする機能（プロポーザル）の形も変化した
 - ≫ 最大限最小(maximally minimal)のプロポーザル
 - ≫ 最初からすべて入れることは無理なので、最小限の形から入れる提案が増えた²
- ≫ すべてのプロポーザルが入るわけではない
 - ≫ 代替のプロポーザルや後方互換性、ユースケースの問題で取り下げたプロポーザルも多い
- ≫ これはモジュール化されたプロポーザルは交換可能な事を表してる

² クラス、mixinなどがこのようなmaximally minimalで進んでいる

プロポーザルをエミュレート

- » Stage 4に「2つ以上の実装が存在している」という必須要件がある
- » 正式なECMAScriptになる前にはブラウザには実装される^{flag}
- » TranspilerやPolyfillを使ってエミュレートできる
- » Transpiler: 新しい構文を既存の機能で再現できるようにソースコードを変換するツール
 - » TypeScript、Babelなど
- » Polyfill: プロポーザルで追加された新しいメソッドや関数などを実装を提供するライブラリ
 - » core-js、babel-polyfillなど

^{flag} 多くはフラグ付き実装。一度Stableに入れると外せなくなるため。Stage 4の要件はあえて曖昧性を持たせている

⚠ TranspilerやPolyfillの注意 ⚠

- ≫ TranspilerやPolyfillはあくまで既存の機能で新しい機能を再現を試みているだけ
- ≫ 原理的に全く新しい機能は既存の機能では再現できない
 - ≫ コストの問題で再現してないこともある
- ≫ TranspilerやPolyfillをそのプロポーザルを学ぶために使うべきではない
- ≫ 先行して試せるようになったというのが大きい、プロポーザルへのフィードバックができる

Living StandardとなるECMAScript

- >> ECMAScriptの仕様書のドラフトはGitHub上の[tc39/ecma262](https://github.com/tc39/ecma262)で管理されている
- >> 本当の意味での最新のECMAScript仕様は<https://tc39.github.io/ecma262/>
- >> バージョン番号を付けずに、常に最新版を公開する仕様のことを**Living Standard**と呼ぶ
- >> 加えて、ECMAScript 2017のようにバージョン番号をつけたものも公開
 - >> バージョン付きECMAScriptは、スナップショットのようなもの

仕様や策定プロセスを知る意味

仕様や策定プロセスを知る意味

- >> 言語を学ぶため
- >> 言語が進化しているため
- >> 情報の正しい状態を調べるため

言語を学ぶため

- » JavaScriptという言語そのものを学ぶため
- » 言語の詳細を知りたい場合にはECMAScriptという仕様を参照する
- » JavaScriptにおいては言語機能に関しては[MDN Web Docs](#)などで大抵は十分

言語が進化しているため

- ≫ 言語が進化しているのは何かの問題を解決するため
- ≫ ECMAScriptはLiving Standardであり、言語仕様に新しい機能や修正などが日々行われている
 - ≫ ECMAScriptは後方互換性を尊重するため、今学んでいることが無駄になるわけではない
- ≫ この仕様はどのような経緯で入ったのかを調べる手段として策定プロセスを知る
 - ≫ 特にES2015以降はGitHubに殆どの情報があるので探しやすい

情報の正しい状態を調べるため

- >> JavaScriptは幅広く使われている言語であるため、世の中には膨大な情報がある
- >> 検索して見つかる情報には正しいものや間違っものが混在する
 - >> JSer.infoはこの問題を解決するために作った
- >> 検索できる情報と比較: 仕様やプロポーザルに関する情報は状態が明確
 - >> 仕様は安定。プロポーザルはステージという明示された状態があり、ステージ4未満の場合はまだ安定してない
- >> 問題を見つけた際に該当する仕様やプロポーザルなどよりオリジンに近いものを調べる手段を知る

具体例



実装が仕様に準拠して

いるかを調べたい

この挙動って仕様なの?というのをすべ

てのJavaScriptエンジンで比較する

eshost

- >> GoogleChromeLabs/jsvu: JavaScript (engine) Version Updater
 - >> 主要なJavaScriptエンジンをまとめてインストール
 - >> macOSならChakraを含めてほぼ対応できる
- >> bterlson/eshost-cli: Run ECMAScript code uniformly across any ECMAScript host
 - >> 指定したJavaScriptエンジンでコードを実行できる

eshost の実行例

```
$ eshost -e 'new RegExp("\n").toString()'
```

```
##### Chakra
```

```
/\n/
```

```
##### SpiderMonkey
```

```
/\n/
```

```
##### JavaScriptCore
```

```
/\n/
```

```
##### V8
```

```
/
```

```
/
```



ECMAScriptの仕様を

読む

例: BabelとTypeScriptのクラス

BabelとTypeScriptでES5相当への変換と実行結果が異なる

```
1 class MyClass {  
2     method() { }  
3 }  
4 console.log(Object.keys(MyClass.prototype));
```

Babel(preset-envのデフォルト)

```
1 "use strict";
2
3 var _createClass = function () { function defineProperties(target, props) { for (var i = 0; i < props.length; i++) { var
  descriptor = props[i]; descriptor.enumerable = descriptor.enumerable || false; descriptor.configurable = true; if ("value"
  in descriptor) descriptor.writable = true; Object.defineProperty(target, descriptor.key, descriptor); } } return function
  (Constructor, protoProps, staticProps) { if (protoProps) defineProperties(Constructor.prototype, protoProps); if
  (staticProps) defineProperties(Constructor, staticProps); return Constructor; }; }();
4
5 function _classCallCheck(instance, Constructor) { if (!(instance instanceof Constructor)) { throw new TypeError("Cannot call
  a class as a function"); } }
6
7 var MyClass = function () {
8   function MyClass() {
9     _classCallCheck(this, MyClass);
10  }
11
12  _createClass(MyClass, [{
13    key: "method",
14    value: function method() {}
15  }]);
16
17  return MyClass;
18 }();
19
20 console.log(Object.keys(MyClass.prototype)); // => []
```

TypeScript(target:es5)

```
1 var MyClass = /** @class */ (function () {
2     function MyClass() {
3     }
4     MyClass.prototype.method = function () { };
5     return MyClass;
6 }());
7 console.log(Object.keys(MyClass.prototype)); // => ["method"]
8
```

例: BabelとTypeScriptのメソッドの列挙

- >> Babelはメソッド（プロパティ）は列挙されないので []
- >> TypeScriptはメソッド（プロパティ）が列挙されるので ["method"]
- >> どっちが正しいのかを仕様を調べてみる
 - >> * ネイティブで実行するのが最速の調べ方だけど、まあたまにバグってるので...

仕様の調べ方

- >> 理由がなければ最新版を使う <https://tc39.github.io/ecma262/>
- >> class構文のメソッド定義の仕方を調べたい

大きな流れ

1. class という Syntax の定義を探す
2. class の Runtime Semantics (Syntax にはそれぞれ実行時に何をするかという定義がある)
3. Runtime Semantics でそれぞれの `method(){}` がどのように定義されているかを見ていく
4. 今回は列挙されているかなので `enumerable` を調べる

Introduction
1 Scope
2 Conformance
3 Normative References
4 Overview
5 Notational Conventions
6 ECMAScript Data Types and Values
7 Abstract Operations
8 Executable Code and Execution Contexts
9 Ordinary and Exotic Objects Behaviours
10 ECMAScript Language: Source Code
11 ECMAScript Language: Lexical Grammar
12 ECMAScript Language: Expressions
13 ECMAScript Language: Statements and Declarati...
14 ECMAScript Language: Functions and Classes
15 ECMAScript Language: Scripts and Modules
16 Error Handling and Language Extensions
17 ECMAScript Standard Built-in Objects
18 The Global Object
19 Fundamental Objects
20 Numbers and Dates
21 Text Processing
22 Indexed Collections
23 Keyed Collection
24 Structured Data
25 Control Abstraction Objects
26 Reflection

ECMAScript® 2019 Language Specification



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](#)

Refer to the [colophon](#) for more information on how this document is created.

仕様を読んだまとめ

- >> 動画で紹介したように仕様ではenumerableをfalseにしてメソッドを定義している
- >> Babelは仕様に準拠するたびにdefinePropertyで列挙しないように(enumerableをfalse)定義してる
- >> TypeScriptは単純にプロトタイプオブジェクトのメソッドを追加してるだけなので列挙される
 - >> 単純なプロパティ定義はenumerableをtrueで定義される
 - >> Object.keysで列挙されるのはその違い
- >> [Method class enumerable · Issue #15038 · Microsoft/TypeScript](#)

補助資料

- >> How to Read the ECMAScript Specification
 - >> ECMAScriptの読み方について解説してる
 - >> [[内部プロパティ]]やAbstract Operator、Runtime semanticsなど仕様書に出てくる記号や読み方を解説してる
- >> anba/es6draft: ECMAScript 2015 (ECMA-262 6th Edition) compiler and runtime
 - >> ECMAScriptのJava実装
 - >> リファレンス実装的に仕様書と対応したアルゴリズムステップで実装されているので読みやすい
 - >> ClassDefinitionEvaluationの実装もそのままある

例: js-primerの場合

js-primerとは

- » ES21015以降を基本としたJavaScriptの入門書
- » Living StandardであるECMAScriptに追従するように書いている
- » github.com/asciidwango/js-primer
- » 興味ある人はWatchしておいてください



Type to search

[この書籍について](#)

本書の読み方

本書の目的

第1部: 基礎文法

- JavaScriptとは
- コメント
- 変数と宣言
- 値の評価と表示
- データ型とリテラル
- 関数と宣言
- 文と式
- 条件分岐
- ループと反復処理
- 演算子
- 暗黙的な型変換
- オブジェクト
- 配列
- 文字列



[WIP] JavaScriptの入門書

ECMAScript 2017時代のJavaScript入門書

Tweet Watch 202 Star 644

これからJavaScriptを始める人がES2015以降をベースにして学べる本(予定)
プログラミングをやったことがあるが、今のJavaScriptがよくわからないという人が、今のJavaScriptアプリケーションを読み書きできるようになるもの。

★更新情報を購読

この本は現在実装中であるため、予告なしに変更される可能性があります。
この本の更新情報を受け取りたい方はメールアドレスを登録することで通知を受け取れます。

メールアドレス

登録

Bug Report

例: js-primerの場合

- >> 迷った場合はECMAScriptを参照する
- >> 用語、表現、動作の説明など

例1: 関数とthisの表現 – Arrow Function

*function*キーワードで定義した関数は呼び出し時に、ベースオブジェクトが暗黙的な引数のように*this*の値として渡されます。

一方、*Arrow Function*の関数は呼び出し時に*this*を受け取らないため、定義時の*Arrow Function*における*this*の参照先が静的に決定されます。

— 関数とthis・JavaScriptの入門書 #jsprimer

例1: 関数とthisの表現 – Arrow Function

- >> 「Arrow Functionはthisをbindする」という説明だと仕様に即していないため避けた
- >> 仕様ではArrowFunctionは[[ThisValue]]を持たないLexicalEnvironmentという定義
 - >> つまり原理的にthisそのものを持っていない
- >> Arrow Functionはthisの値を受け取らないからthisの値が静的に決まるという話にした
- >> 詳細は[ECMAScript 2015以降のJavaScriptのthisを理解する | Web Scratch](#)

例2: 関数とthisの表現 - thisの値

例2: 関数とthisの表現 - thisの値

- >> thisの解説をするためにあらゆる場所のthisの挙動を調べていた
- >> 次の"Module"の挙動がブラウザによって違った
 - >> V8はWindow、その他はundefined

```
<script type="module">  
const fn = () => this;  
console.log(fn());  
</script>
```

例2: 関数とthisの表現 - thisの値

- » Arrow Functionにおけるthisは外側のスコープのthisを参照する
- » トップレベルのthisとトップレベルのArrow Functionにおけるthisは同じ
 - » 以下のコードは全ブラウザで同じ挙動だった

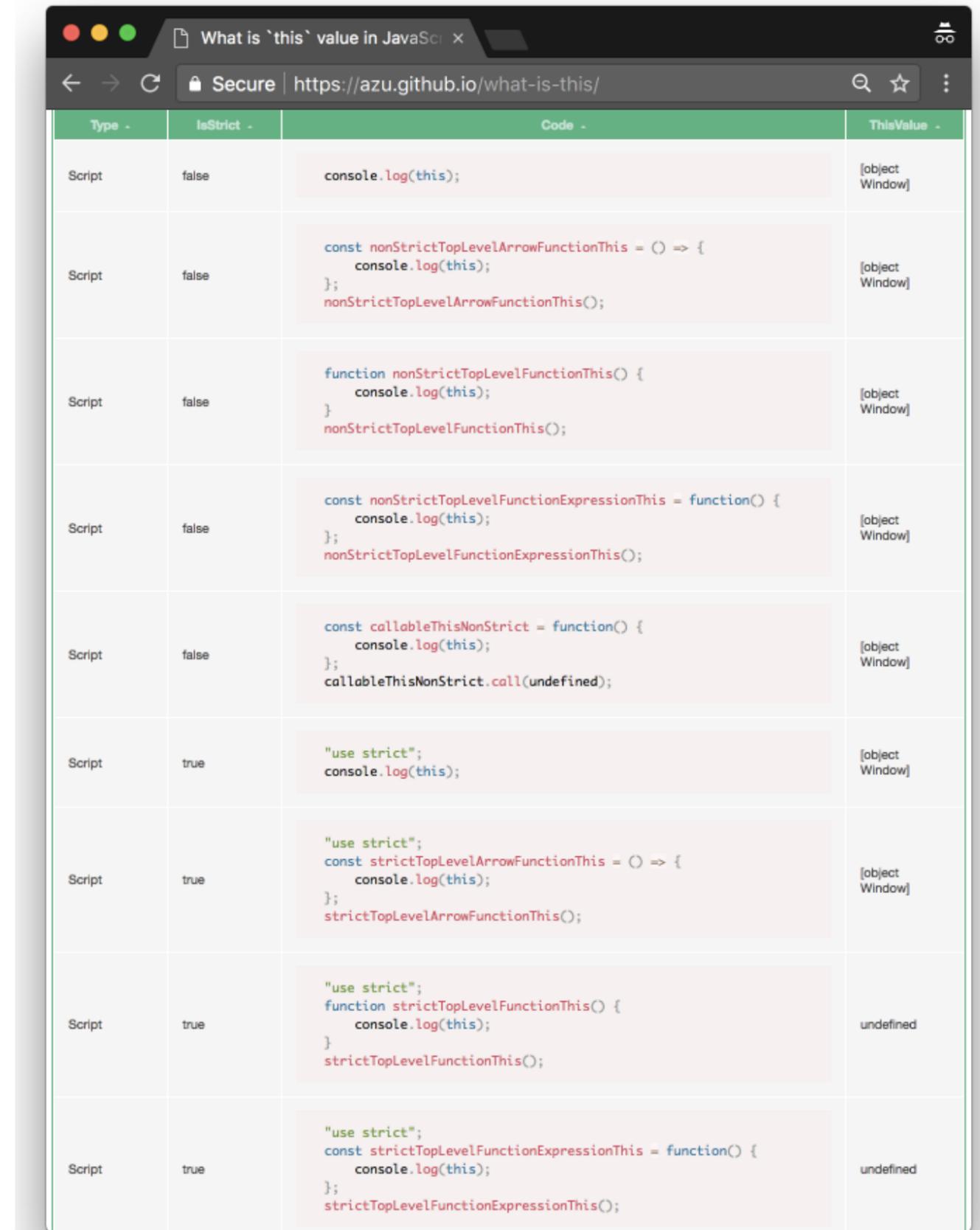
```
<script type="module">  
  const fn = () => this;  
  console.log(fn() === this); // => true  
</script>
```

例2: 関数とthisの表現 - thisの値の仕様

- ≫ トップレベルのthisの値は実行コンテキストによって違う
- ≫ 実行コンテキストは"Script"と"Module"(いわゆるES moduleのコンテキスト)
- ≫ 仕様では"Module"コンテキストのトップレベルthisは常にundefined
 - ≫ ES6 moduleのtop levelにあるthisの値は何になるのか?
- ≫ つまりはChrome/V8のバグ 🐛

例2: 関数とthisの表現 - thisの値

- >> thisのテーブル: azu.github.io/what-is-this
- >> 🐛 報告: [791334 - this in top level Arrow Function in Module Context should be undefined - chromium - Monorail](https://github.com/azu/what-is-this/issues/791334)
- >> Chrome 65で直る



Type	isStrict	Code	ThisValue
Script	false	<code>console.log(this);</code>	[object Window]
Script	false	<code>const nonStrictTopLevelArrowFunctionThis = () => { console.log(this); }; nonStrictTopLevelArrowFunctionThis();</code>	[object Window]
Script	false	<code>function nonStrictTopLevelFunctionThis() { console.log(this); } nonStrictTopLevelFunctionThis();</code>	[object Window]
Script	false	<code>const nonStrictTopLevelFunctionExpressionThis = function() { console.log(this); }; nonStrictTopLevelFunctionExpressionThis();</code>	[object Window]
Script	false	<code>const callableThisNonStrict = function() { console.log(this); }; callableThisNonStrict.call(undefined);</code>	[object Window]
Script	true	<code>"use strict"; console.log(this);</code>	[object Window]
Script	true	<code>"use strict"; const strictTopLevelArrowFunctionThis = () => { console.log(this); }; strictTopLevelArrowFunctionThis();</code>	[object Window]
Script	true	<code>"use strict"; function strictTopLevelFunctionThis() { console.log(this); } strictTopLevelFunctionThis();</code>	undefined
Script	true	<code>"use strict"; const strictTopLevelFunctionExpressionThis = function() { console.log(this); }; strictTopLevelFunctionExpressionThis();</code>	undefined



プロポージャーのステータスを知りたい

このプロポージャーの今のステータスは？

- Stage 0 Proposals
- Finished Proposals
- Inactive Proposals

ECMAScript Internationalization API Specification proposals

Active proposals

Proposals follow this process document. This list contains only stage 1 proposals and higher that have not yet been withdrawn/rejected, or become finished.

Stage 3

tc39/proposals:

Tracking ECMAScript

Proposals

	Proposal	Author	Champion	Tests
	Function.prototype.toString revision	Michael Ficarra	Michael Ficarra	✓
	global	Jordan Harband	Jordan Harband	✓
	import()	Domenic Denicola	Domenic Denicola	?
	Legacy RegExp features in JavaScript	Claude Pache	Mark Miller, Claude Pache	✓
	BigInt	Daniel Ehrenberg	Daniel Ehrenberg	🚧
	Optional catch binding	Michael Ficarra	Michael Ficarra	✓

babel/proposals: 🖋️ Tracking the status of Babel's implementation of TC39 proposals

>> Babelの実装との対応表

Babel progress on [ECMAScript](#) proposals

[Official TC39 Proposals Repo](#)

Proposals

For the official list, check the [TC39 repo](#). This list only contains all proposals higher than Stage 0 that are compilable by Babel. (If Stage 4, it will be in [preset-env](#))

Proposal Link	Current Stage	Status
Rest/Spread Properties	3	3
Asynchronous Iteration	3	3
Dynamic Import	3	3
RegExp Unicode Property Escapes	3	3
RegExp Named Capture Groups	3	3
RegExp DotAll Flag	3	3
Class Fields	3	2
function.sent	2	2
Class and Property Decorators	2	1
BigInt	2	Parsable
import.meta	2	Parsable

このプロポーザルって進んでるの？

- ≫ ECMAScriptのプロポーザルにはそれぞれチャンピオンがいる
- ≫ チャンピオンがステージを進めたい  という意志が必要
 - ≫ プロポーザルは勝手には進まない
- ≫ 何が課題で止まっているかプロポーザルリポジトリのIssueかミーティングノートを見る

例: tc39/proposal-
global

globalでグローバルオブジェクトを取
得するプロポーザル

例: tc39/proposal-global

- >> 2018年2月の段階ではStage 3で止まっている
- >> 理由はREADMEに書いてある
- >> global breaks flickr.com · Issue #20 · tc39/proposal-globalでは global という名前によって壊れるサイトがある問題がありこれを理由に止まっている

however, due to web compatibility concerns, it is on hold pending a new global identifier name.

Tips

- ≫ プロポーザルはそれぞれGitHubリポジトリを持っている
- ≫ プロポーザルのIssueには課題が書かれている
- ≫ ウェブ互換性の問題については基本的にメトリクスデータを元に話を進める
 - ≫ 壊れるウェブサイトはn%あるか
 - ≫ Chrome Platform Status、Microsoft Edge Platform Data、Firefox Data

例: tc39/proposal-
decorators

@デコレーター

例: tc39/proposal-decorators

- >> 2018年2月の段階ではStage 2
- >> ECMAScript proposal updates @ 2016-07 | ECMAScript Daily
によると2016年からずっとStage 2のまま
- >> 何が原因で進んでないのかを調べる
- >> => TC39のミーティングでどのような議論が行われてるのかを調べる
- >> => 何が課題となっているかが話し合われているはず

例: tc39/proposal-decorators Stage 2

- >> tc39/agendas (議事録のアジェンダ) を "Decorators" で検索
- >> 定期的に議論されているので何か課題があり進んでいない
 - >> Stage 3に進むには仕様としてひとまず完成した状態が求められる
- >> アジェンダと同じ月のミーティングノートを見る
 - >> 2018-01/summary.md サマリを見るのが簡単

14
10
2
14

eat sheet

14 code results in [tc39/agendas](#)

Sort: Recently

2018/01.md

Showing the top two matches Last indexed 22 days ago

```
108      1. ~[`throw` expressions](https://github.com/tc39/proposal-throw-expressi
stage 3 (Ron Buckton)~~
109      1. ~[Decorators use cases](https://docs.google.com/presentation
/d/178WoBNhXBBB1M_LlSqNKEic4SD4CFYt09Fkr9rvsHaA/view) (Diego Ferreira Val, Yehuda
Caridy Patiño)~~
...
119      1. ~[Static class features proposal](http://github.com/tc39/proposal-stat
class-features/) (Daniel Ehrenberg) ([slides](https://docs.google.com/presentation
/d/1wixI6gGD1H26xze35MKcIFyAHLQU7y9yWR06Hd37E80/edit#slide=id.p))~~
120      1. ~[Decorators](https://github.com/tc39/proposal-decorators/) discussion
towards Stage 3 (Daniel Ehrenberg) ([slides](https://docs.google.com/presentation
/d/1g6hrJp_nk_0eapuPXlkE4D_310Zbz4wQbXuIagsyoUI/edit#slide=id.p))~~
```

2017/11.md

Showing the top match Last indexed on 1 Dec 2017

```
125      1. ~[Class fields ASI](https://github.com/tc39/proposal-class-fields/issu
discussion and resolution (Daniel Ehrenberg) ([slides](https://docs.google.com
/presentation/d/1bPzE6i_Bpm6FXgzfx9XFJNHGkVcM42lux-6bUNhxpl4/edit#slide=id.p))~~
126      1. [Decorators](https://github.com/tc39/proposal-unified-class-features/)
discussion towards Stage 3 (Daniel Ehrenberg) ([slides](https://docs.google.com
/presentation/d/1g6hrJp_nk_0eapuPXlkE4D_310Zbz4wQbXuIagsyoUI/edit#slide=id.p)) *la
because slides published on Sunday*
```

2017/09.md

Showing the top match Last indexed on 14 Oct 2017

```
118      1. ~[Private methods and accessors](https://github.com/littledan/proposal
private-methods) for Stage 3 (Daniel Ehrenberg) ([slides](https://docs.google.com
/presentation/d/1aI89Jgl7CdtKV6D5-ydieUn_-kgRqAD2X8gGzh62xzc/edit#slide=id.p))~~
119      1. ~[Decorators](https://github.com/littledan/proposal-unified-class-feat
detailed discussion of proposed semantics (Daniel Ehrenberg) ([slides]
(https://docs.google.com/presentation/d/1jjGeHMW8xH49dWP8S5xk0EHaST1zmRDlbtwyLxaYW
/edit#slide=id.p))~~
```

Others

Proposals not advancing to new a new stage. e.g.: development updates, needs 1

- [Function.prototype.toString](#) still on Stage 3 with plans for a follow up work.
- [Making nullish values iterable, or at least array-spreadable](#) discussion to follow
- [new Set builtin methods](#) for further discussion before advancing to Stage 2.
- [Optional Chaining](#) updates.
- Discussion on [operator overloading](#), not advancing to Stage 1 for now to collect more feedback.
- [TC39 should endorse use of a \(one-of-several, not one specific recommended\) semicolon style](#), not any particular semicolon style.
- Discussion on [use cases of Decorators](#).
- Updates on [Decorators](#) to further advancement to Stage 3.
- Updates on [Static Class features](#), no advancement.
- Updates on [BigInt](#) status.
- [Open-ended discussion: Exploring Statements as Expressions.](#), not advancing until other issues are addressed first.
- [issue about process, coordinating with other standards bodies.](#)

例: tc39/proposal-decorators

Stage 3に向けて

- » 13.v.c Decorators: towards Stage 3 という議論が2018年1月に行われている
- » Decorators: Towards Stage 3 - Google スライド
 - » Stage 3に向けて何をサポートし、何をサポートしないかをはっきりさせる
 - » 他のクラスのプロポーザル(hard private)との協調性についての課題があり調整している
 - » 実装者、テスト作成者、ライブラリ作者に対しても意見を求めている
 - » 次のミーティング(3月)までにステークホルダーにアプローチする

Tips

- >> Stage 1はまでアイデアや実験なのでプロポーザル間でも重複する
- >> Stage 2+あたりからプロポーザル間での協調的な仕様を検討する^{note}
- >> Decoratorはclass field、privateなど色々関係する
- >> 最近のDecoratorの変更は他のプロポーザルとの協調性やDecoratorが目指す範囲を確定する作業
 - >> 今まで雰囲気動いていた部分を明示的に例外を投げるようにするなど

^{note} [Revisiting mixins-vs-protocols proposal](#)を参照

🚀 はプロポーザルのStageをChampionが進める意志がある状態か(参考程度)
まだそのStageで議論すべき課題があるかどうかを示している

Stage 2

	Proposal	Author	Champion
	Static class fields and private static methods	Daniel Ehrenberg, Kevin Gibbons, Jeff Morrison, Kevin Smith	Daniel Ehrenberg
	<code>function.sent</code> metaproperty	Allen Wirfs-Brock	Allen Wirfs-Brock
	Decorators	Daniel Ehrenberg	Yehuda Katz, Brian Terlson, Daniel Ehrenberg
	<code>throw</code> expressions	Ron Buckton	Ron Buckton
	<code>Atomics.waitAsync</code>	Lars Hansen	Shu-yu Guo, Lars Hansen
	<code>Symbol.prototype.description</code>	Michael Ficarra	Michael Ficarra



プロポーザルのステータス

の変化を知りたい

プロポーザルのステータスの変更の

タイミングを知りたい。通知欲しい

プロポーザルのステータスの変化を知りたい

>> [ECMAScript Daily](#)

>> ECMAScript関係のニュースブログ(@azu)

>> [EcmaScript.in | Stay updated about EcmaScript proposal changes](#)

>> プロポーザルのStage変化の通知メール

>> [2ality - JavaScript and more](#)

>> プロポーザルの解説

>> [bevacqua/prop-tc39: Scraping microservice for TC39 proposals](#) 🐱

>> API

今日話したこと

- >> ECMAScript · JavaScriptの入門書 #jsprimer
- >> js-primerに文章版があるよ
- >> @EcmascriptDailyをフォローしておけばとりあえず流れてくる
- >> ECMAScriptのプロポーザルはtc39/tc39-notes、tc39/proposals、
各リポジトリを見れば殆どの情報があるよ
- >> 必要になったときに自分で探してみてください

まとめ

- >> 🌟 JavaScript is ECMAScript、Browser、Node.js、WebAssembly、WebGL、WebRTC...
- >> 📖 すべてのことを知る必要はなく、すべてを知っている人もいない
- >> 🔍 知りたいと思ったときに調べる方法を持っていることが大切
- >> 🧠 なにごとも突然に新しい概念は増えない - 過程がある
- >> 🌟 日々変化するソフトウェアにおいては、自身に適切な調べ方をもつ

宿題 🏠

- ≫ ES2018で何が入るかを[tc39/proposals](https://tc39.org/proposals)から探してみよう
 - ≫ Hint: ES2018入るのはStage 4(Finished Proposals)
- ≫ 気になっているプロポーザルのリポジトリをWatchしよう
 - ≫ Hint: プロポーザルの一覧は[tc39/proposals](https://tc39.org/proposals)を見る
- ≫ ミーティングのサマリーを読んでみよう
 - ≫ [2018-01/summary.md](#)から読めます