ECMAScript as a Living Standard

自己紹介

- Name : **azu**
- Twitter : @azu_re
- Website: Web scratch, JSer.info

伝えたいこと

» ECMAScriptの仕様策定は特別なプロセスではない

» そこへ参加する/見ていくのはむずかしいものではない[x]

[x] 標準委員会に参加しないとできないことはあるが、何もできないわけではない

伝えたいこと

- ECMAScriptの仕様策定はただの大きな(GitHub)プロジェクト
- 普通のプロジェクトと大きな違いはない[y]
- フォーマルに物事が進む分 普通より分かりやすい
- どのように進めれば物事が進むかのパターンを学べる

[y] Ecmaへの仕様提出などは特殊だけど、それ以外は大きく変わらない

ES2016 リリース 🎉

その前にES2015では何があったんだっけ?

New syntax

- Arrow Function
- Classes
- Modules
- Block Scope (let/const)
- Extended Object Literal
- Default Params
- Rest Params
- Spread Operator
- Destructuring
- Iterator
- Generator
- Template Literal
- Tail Call Optimization

Improvement of existing classes

- String
- RegExp
- Array
- Object
- Math
- Number

New built-in classes and objects

- Promise
- Map
- Set
- WeakMap/WeakSet
- TypedArray
- Symbol
- Proxy/Reflect

一番大きな変化

- ES2015からは1年毎のリリースサイクルに変更される
- ES2015はその早いリリースサイクルを適応できるだけの基盤
  - だからまだまだ足りない機能はある
- リリースサイクルと共に仕様策定のプロセスも変更された
  - 早いサイクルと開発者/実装者からのフィードバックを得るためのプロセスへ

It's the foundation for the next 10-20 years of JavaScript evolution

— Allen Wirfs-Brock

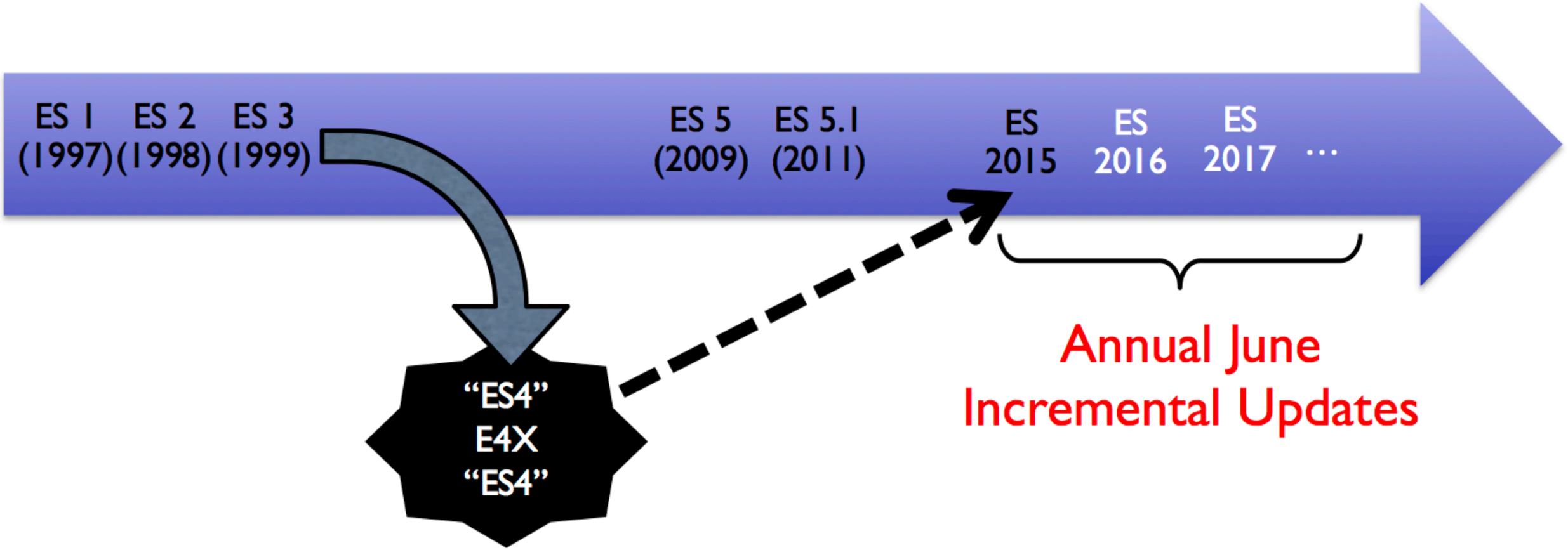
ECMAScript as a Living Standard

Living Standard

- ECMAScriptもLiving Standardへ
- tc39.github.io/ecma262/では常に最新の仕様が公開されている
- ES2015, 2016は1年毎のスナップショット

The ECMAScript Standard Timeline

Release trains are now leaving the station



Living Writing 

ES2015以降のJavaScript入門本

- github.com/asciidwango/js-primer
- ES2015以降ベースのJavaScript入門本をオープンソースで書いてる
- 仕様書はLiving Standard
- 書籍がスナップショットだけでは追いつけない
- 書籍もLivingに更新できる仕組みを作る必要がある
  - 出版は予定しているが開発中の段階から公開します

for

- プログラミングをやったことがある
- 今のJavaScriptがよくわからないという人向け
- 今のJavaScriptアプリケーションを読み書きできるようなるもの
- ミーティングノートも公開している
- js-primer/meetings at master · asciidwango/js-primer

JavaScriptの本

📖ECMAScript 2016時代のJavaScript入門書

これからJavaScriptを始める人がECMAScript 2016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なるもの。

★更新情報を購読

このサイトは予告なしに場所が変わ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。

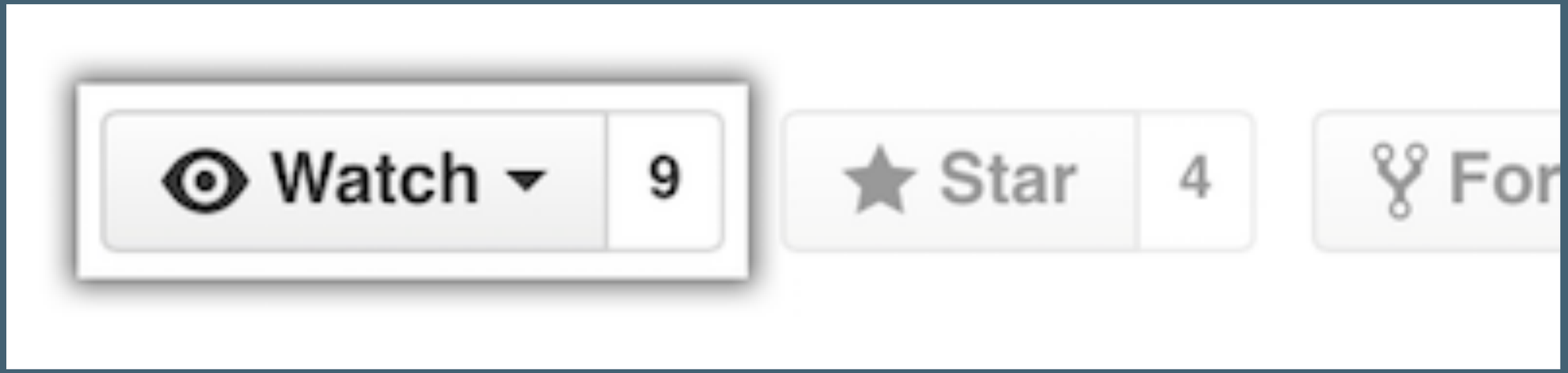
メールアドレス

登録

続きはGitHubで

https://github.com/asciidwango/js-primer

» 興味ある人はWatchしておいてください



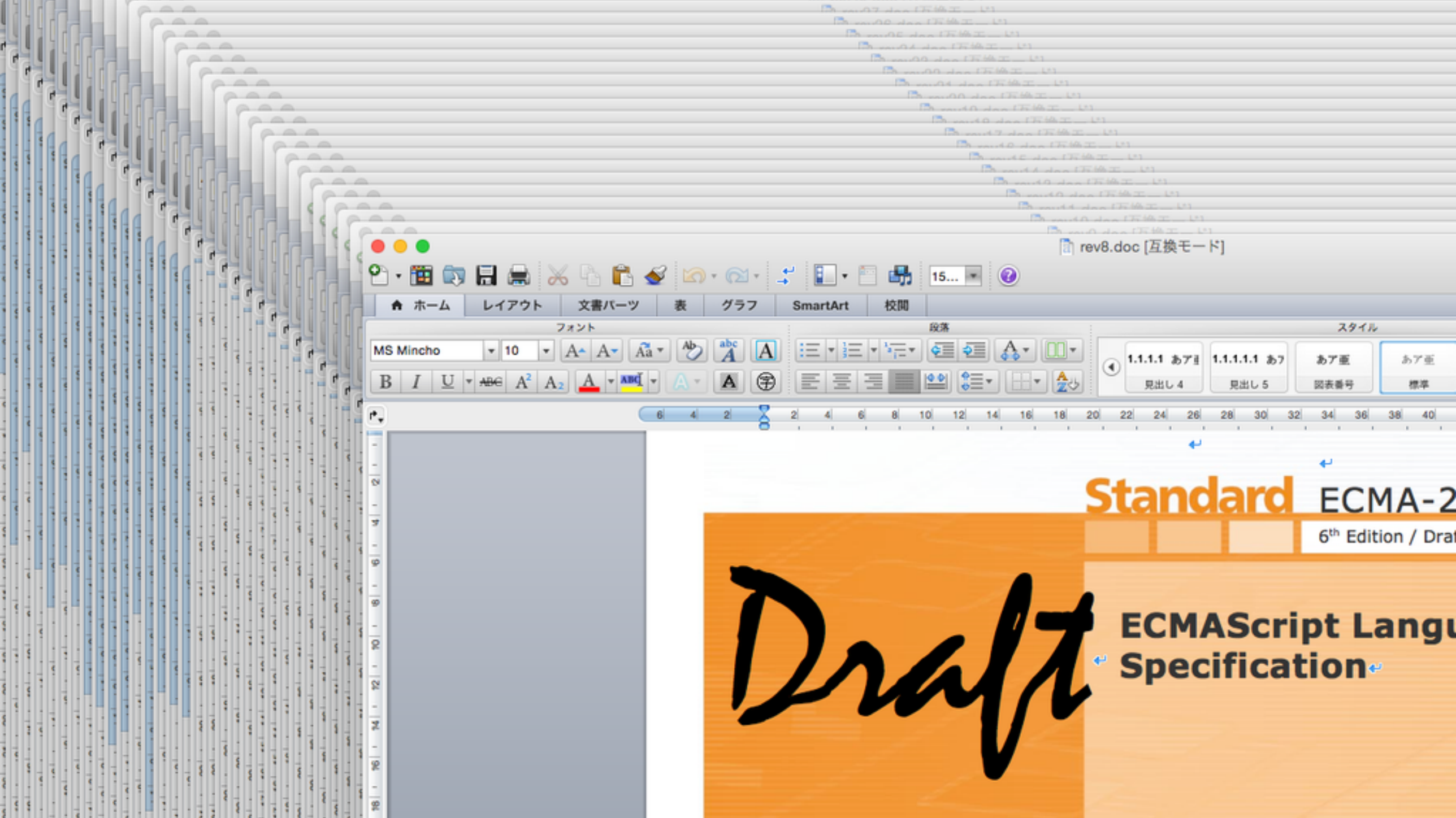
» リリースだけを知りたい人は、フォームからメールアドレスを登録しておくと通知される予定です

ECMAScript 2015 – 2016 diff points

Move to GitHub

- tc39/ecma262: Status, process, and documents for ECMA262
- GitHubで仕様書、Issue、Pull Requestで開発されるようになった

WordからEcmarkupへ変更


rev8.doc [互換モード]
ホーム
レイアウト
文書パーツ
表
グラフ
SmartArt
校閲
フォント
段落
スタイル
MS Mincho
10
Standard ECMA-2
6th Edition / Draf
Draft
ECMAScript Langu
Specification

When `Object` function is called with optional argument *value*, the following steps are taken:

1. If NewTarget is neither **undefined** nor the active function, then
    a. Return ? OrdinaryCreateFromConstructor(NewTarget, "%ObjectPrototype%").
2. If *value* is **null**, **undefined** or not supplied, return ObjectCreate(%ObjectPrototype%).
3. Return ToObject(*value*).

The `length` property of the `Object` constructor function is 1.

19.1.2 Properties of the Object Constructor

The value of the [[Prototype]] internal slot of the Object constructor is the intrinsic object %FunctionPrototype%.

Besides the `length` property, the Object constructor has the following properties:

19.1.2.1 Object.assign (*target*, ...*sources*)

The `assign` function is used to copy the values of all of the enumerable own properties from one or more source objects to a *target* object. When the `assign` function is called, the following steps are taken:

1. Let *to* be ? ToObject(*target*).
2. If only one argument was passed, return *to*.
3. Let *sources* be the List of argument values starting with the second argument.
4. For each element *nextSource* of *sources*, in ascending index order,
    a. If *nextSource* is **undefined** or **null**, let *keys* be a new empty List.
    b. Else,
        i. Let *from* be ToObject(*nextSource*).
        ii. Let *keys* be ? *from*.[[OwnPropertyKeys]]().
    c. Repeat for each element *nextKey* of *keys* in List order,
        i. Let *desc* be ? *from*.[[GetOwnProperty]](*nextKey*).
        ii. If *desc* is not **undefined** and *desc*.[[Enumerable]] is **true**, then

WordからEcmarkupへ変更

- ECMAScriptの仕様を書くためのCustom Element + Ecmarkdown
  - アルゴリズムのステップをMarkdown風に書ける
- これによりクロスリファレンスリンクが充実し読みやすくなった
  - リンクをクリックすれば宣言元へ飛べる

Layering: Unhandled Rejection

- Unhandled Rejection Tracking Browser Events
- 実行環境が`rejectionHandled` / `onrejectionhandled`イベントを実装できるためのHookポイントの追加
  - Promise Error Handling
  - Tracking unhandled rejected Promises

Normative: Remove [[Construct]] from generators

```
function * G() { construtor(){} }
new G(); // throw error
```

» Generatorはnewできなくなった

» constructorがg.next()されるまで呼ばれないという混乱を生む挙動になっていたため

» 実例とともに学ぶECMAScript 2015 〜Generator〜 - NET BIZ DIV. TECH BLOG

Normative: Add `Array.prototype.includes` and `TypedArray.prototype.includes`

- » ES2015で入らなかったのは`contains`がBreak the webであったため
- » tc39/Array.prototype.includes: Spec, tests, reference implementation, and docs for ESnext-track Array.prototype.includes
- » Introducing Break the Web: Array extra methods case // Speaker Deck

Normative: Add ** exponentiation operator

» べき乗演算子

```
// 2の3乗算
2 ** 3 == Math.pow(2, 3);
```

Normative: Require Unicode 8

» U+180Eがホワイトスペースではなくなった

  » Unicode 5.2.0では空白(Zs)の定義に\u180Eが含まれていた

```
eval("1\u180E===1"); // throw exception in ES2016
// SyntaxError: illegal character in ES2016
// `true` in ES2015
```

» [ES2016] Require Unicode 8.0.0 · Pull Request #300 · tc39/ecma262 | ECMAScript Daily

Normative: Remove Proxy `enumerate` trap and `Reflect.enumerate`

- `Reflect.enumerate`が実装上の複雑さなどの問題から削除された
- Normative: Remove [[Enumerate]] and associated reflective capabilities by bterlson · Pull Request #367 · tc39/ecma262
- Why Remove Proxy [[Enumerate]] and Reflect.enumerate? | ECMAScript Daily

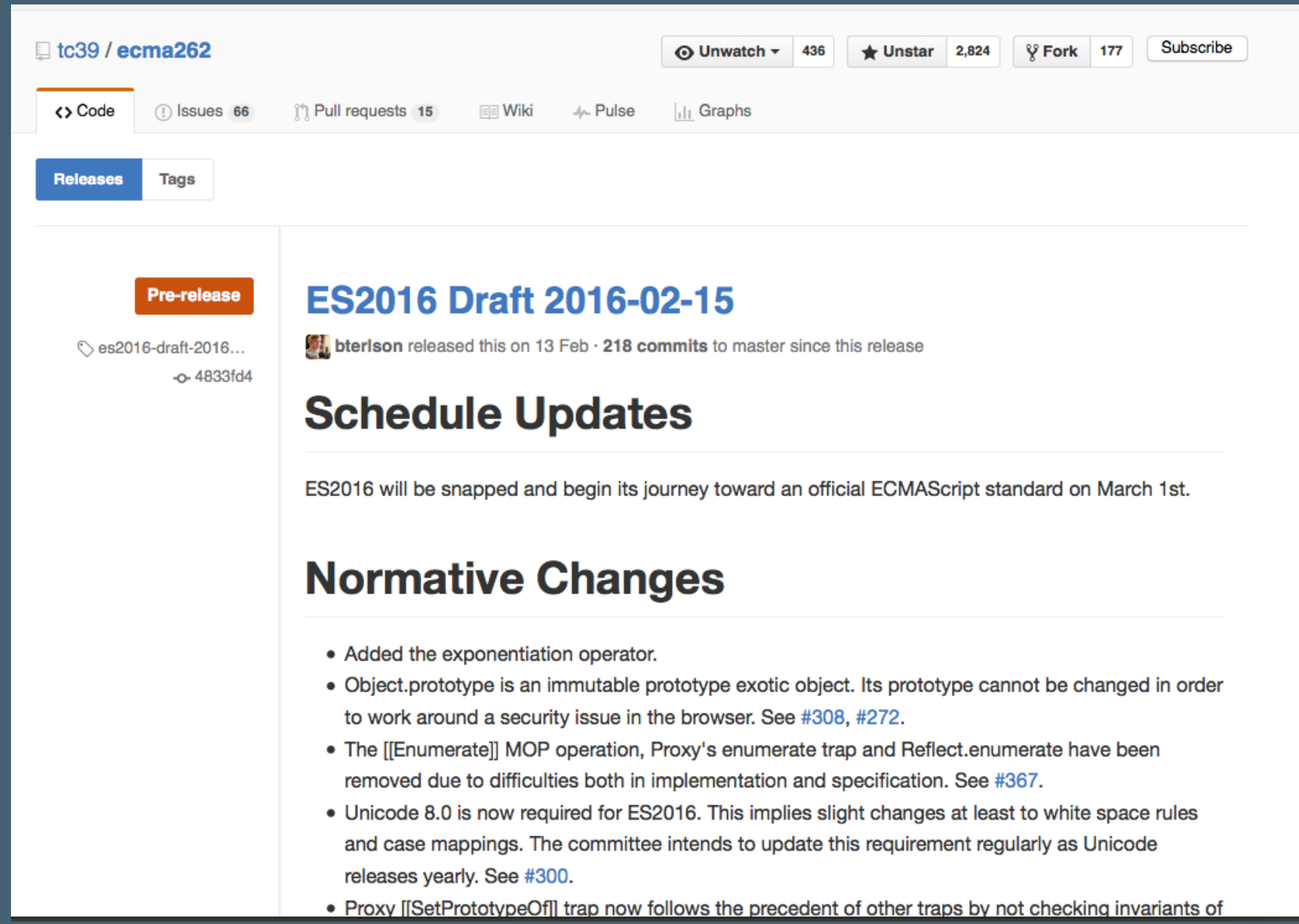
リリースノートはどこに?

リリースノート

» Releases · tc39/ecma262

» GitHub Releaseにリリースノートがある

» リリースノートには変更内容が書かれている



変更内容の種類

- Normative
  - 新しい機能の追加、仕様の変更など
- Editorial
  - 記述の修正、リファクタリング
- Layering
  - ECMAScriptと連携するHTML仕様向けの機能追加、修正など

変更内容の種類はどこから?

» tc39/ecma262はコミットメッセージの規則を持っている [1]

Prefix	Description
Normative:	Nortable change and Add new feature
Layering:	Layering ECMAScript and Other(DOM)
Editorial:	Fix and Refactoring
Meta:	Update Meta

[1] https://ecmascript-daily.github.io/ecmascript/2016/06/25/ecma262-commit-message-conventions

コミットメッセージ規則の利点

» コミットメッセージの規則を持つことで検索が簡単になる

```
$ git clone https://github.com/tc39/ecma262.git
$ cd ecma262
$ git log --grep "Normative:" es2016-draft-1...es2016-draft-20160215
```

そもそも誰が仕様決めているの?

TC39

TC39とは

- ECMAScriptを策定してる専門委員会
  - 言語仕様は法的な問題などに対処するため標準委員会で管理することが多い
- 重要な変更はTC39の中でConsensusを取ってから仕様へ反映
  - 2ヶ月に1度のミーティングで合意を取る

Editor

- Brian Terlson @ Microsoft
- @bterlson
- ES2016〜仕様書のEditor

Need Consensus 🎩

- 挙動を変更するものはConsensusが必要
  - 既存ウェブサイトに影響を与えるものはかなり難しい
  - Chrome Platform StatusやAPI usage on the web platformなど実測値が参考に使われる
- 挙動を変更しないものは、その場でPull Requestがマージされる

新しい機能 – Proposal

- 全く新しい機能も基本的には合意を得られないと追加されない
- ES2016からは機能ごとのProposalを出す
- つまり、機能ごとに仕様を決めていき毎年リリースする
- ProposalにはStage 0〜4の**5段階**のラベルがあり、Stage 4になったら仕様へマージされる [2]
  - こちらも2ヶ月に1度のミーティングでStageが変動する

---

[2] The TC39 Process

5段階のStage

- 0. Strawman - アイデア
- 1. Proposal - 提案
- 2. Draft - ドラフト
- 3. Candidate- 仕様書と同じ形式(実装のフィードバック期間)
- 4. Finished - 実装が2つ、test262にテスト => 策定完了
    - 次の年のECMAScript 201＊にマージされる

Proposalってどういうものがあるの?

Proposalの一覧

tc39/proposals: Tracking ECMAScript Proposals

Async Functions

» **Stage3**: 2つ目の実装が出たらStage4へ

```
async function gets(aURL, bURL) {
  const contentA = await getURLAsync(aURL);
  const contentB = await getURLAsync(bURL);
  console.log(contentA, contentB);
}
```

Function.prototype.toString revision

`function f(){}`	`"function f(){}"`	`"function f(){}"`	`"function f(){}"`
`(function (){})`	`"function (){}"`	`"function (){}"`	`"function (){}"`
`class A { a(){} }`	`"class A { a(){} }"`	`"function () {\n    \n}"`	`"function A() { }"`
`(class { a(){} })`	`"class { a(){} }"`	`"function () {\n    \n}"`	`"function () { }"`
`function* g(){}`	`"function* g(){}"`	`"function* g() {}"`	`"function g() {}"`
`(function* (){})`	`"function* (){}"`	`"function* (){}"`	`"function (){}"`
`a => a`	`"a => a"`	`"a => a"`	`"a => a"`
`({ *a(){} }.a)`	`"*a(){}"`	`"function* a(){}"`	`"function a(){}"`
`O.gOPD({ get a(){} }, "a").get`	`"function a(){}"`	`"function (){}"`	`"function (){}"`
`O.gOPD({ set a(b){} }, "a").set`	`"function a(b){}"`	`"function (b){}"`	`"function (b){}"`
`({ a(){} }.a)`	`"a(){}"`	`"function a(){}"`	`"function a(){}"`
`Function.prototype.toString`	`"function toString() { [native code] }"`	`"function toString() {\n[native code]\n}"`	`"function toString() {\n[native code]\n}"`
`(function f(){}.bind(0))`	`"function () { [native code] }"`	`"function f() {\n[native code]\n}"`	`"function f() {\n[native code]\n}"`
`Function("a", "b")`	`"function`	`"function anonymous(a)`	`"function anonymous(a)`

function f(){}	"function f(){}"	"function f(){}"	"function f(){}"
(function (){})	"function (){}"	"function (){}"	"function (){}"
class A { a(){} }	"class A { a(){} }"	"function () {\n \n}"	"function A() { }"
(class { a(){} })	"class { a(){} }"	"function () {\n \n}"	"function () { }"
function* g(){}	"function* g(){}"	"function* g() {}"	"function g() {}"
(function* (){})	"function* (){}"	"function* (){}"	"function (){}"
a => a	"a => a"	"a => a"	"a => a"
({ *a(){} }.a)	"*a(){}"	"function* a(){}"	"function a(){}"
O.gOPD({ get a(){} }, "a").get	"function a(){}"	"function (){}"	"function (){}"
O.gOPD({ set a(b){} }, "a").set	"function a(b){}"	"function (b){}"	"function (b){}"
({ a(){} }.a)	"a(){}"	"function a(){}"	"function a(){}"
Function.prototype.toString	"function toString() { [native code] }"	"function toString() {\n[native code]\n}"	"function toString() {\n[native code]\n}"
(function f(){}.bind(0))	"function () {\n[native code] }"	"function f() {\n[native code]\n}"	"function f() {\n[native code]\n}"
Function("a", "b")	"function	"function anonymous(a)	"function anonymous(a)

Function.prototype.toString revision

» Stage 3

» `func.toString()`が返す文字列を規定 [3]

» ネイティブな関数は `[native function]` を返すという事を規定

```
class { /* body */ }
A.toString();
/*
class { /* body */ }
*/
```

[3] https://github.com/tc39/test262/pull/553

なぜ機能ごとの策定プロセスを取るのか?

- 1年毎にリリースするため、リリース速度をあげる目的
- できるだけ小さな単位で仕様を決めていくため
- 機能毎という小さなProposalをベースにすることで、細かくリリースできる
- 他の言語やフレームワークも似たような事をやっている
  - Python、Swift、Rust、Ember

Avoid bikeshed

プログラミング言語標準化のパターンランゲージ[4]

— Allen Wirfs-Brock(ES2015 Editor)

---

[4] 📝 プログラミング言語標準化のパターン

"maximally minimal"

- 言語デザインは複雑になるほどConsensusを取るのが難しくなる
- "maximally minimal"とは反対意見がある原因やそう思える部分を最大限取り除いた最小のものを仕様にするという手法

"maximally minimal" classes

- ES6での`class`の事例
- strawman:maximally*minimal*classes [ES Wiki]
- 全体として必要という合意があるにもかかわらず、詳細の同意が得られず進まない問題へのパターン
- 全体として合意できる最大限最小のものをES6 Classesとした
- 類似: Less is more

なぜ策定プロセスが公開されているのか?

プロセスの透明性

- tc39/tc39-notesにミーティングノートが公開されている
- どのような議論が行われ、どのような意思決定がされたのか
  - なぜ、このProposalは必要なのか
  - なぜ、このProposalはStage Xなのか
  - なぜ、このProposalは廃止されたのか

一般のライブラリでも同じ

- React: React Core Meeting Notes
- Ember.js: Meeting minutes from the Ember.js core teams
- Node.js: Technical Steering Committee meeting
- jQuery: jQuery Core Team | jQuery Meetings

開発者がプロセスに参加する

- (広い意味での)開発者にとって公開されたプロセスは有益
- ECMAScriptはStage 3で2つのブラウザへの実装が必要
  - ブラウザはフラグ付きで実装できる
  - 開発者はフラグをオンにして試せる
- Babelのようなツールでもっと前に試せる

ブラウザとフラグ

- 全てのブラウザはフラグ管理して実装する
- WebKitもprefixではなくフラグで管理する宣言をした
  - Feature Policy | WebKit
- β/Nightly版を使えば簡単に利用できる
  - Chrome Canary
  - Firefox Developer Edition
  - Safari Technology Preview

フィードバック

- 仕様があっても実装されなければ意味がない
- 実装されても使われなければ意味がない
- 使う側もプロセスの理解とフィードバックが重要
- フィードバックをまともに得ずに進んだものは壊れた歴史
  - ES4で消えた2年半

フィードバックの方法

- 仕様なら
  - Proposalリポジトリへ Issueを立てる
  - ES Discussに投げる
- 実装なら
  - 各ブラウザ/実装のIssueへ
- 何がわからないのかわからないなら
  - jQuery Standards Group
  - WICG

どこから情報を得るか

情報源

- リリースノート
  - Releases · tc39/ecma262
- ミーティングノート
  - tc39/tc39-notes
- ロードマップ/Proposal
  - tc39/proposals

情報源

- Issue
  - Issues · tc39/ecma262 and Proposal's issue
- メーリングリスト/質問
  - ES Discuss
- テスト
  - tc39/test262
- 実装
  - 各ブラウザ、anba/es6draft

ECMAScript Daily

» ECMAScriptについてニュースサイト

» JSer.infoのECMAScript版

情報源を見てみると

- GitHubにリポジトリがあって、IssueやPull Requestでやり取りしていて、リリースノートがGitHub Releaseにあって、議論の結果が残っていて、未確定なロードマップみたいなものがあって、実装やテストがある
- 各実装がGitHubの該当Issueへリンクを貼りクロスリファレンスになっている
- 普通のGitHubプロジェクトと大差ないのでは❓

ECMAScriptを大きな GitHubプロジェクトとして見る

ECMAScriptがやっていること

- ✔ 更新内容は普段からコミットの段階で整理しておく
  - コミットメッセージの規約を設けるなどして、整理しておく
-  リリースノートはGitHub Releaseで公開している
  - リリースノートはアクセスしやすい/更新が分かりやすい場所へ

ECMAScriptがやっていること

- 💭 複雑な設計問題に対してはパターンを使う
  - "maximally minimal"は意思決定のデットロックを壊すツール
- 🎩 Consensusが必要なものはミーティングで意思決定をする
  - 慎重な意思決定が必要な部分は時間を取ってミーティングを行う
- ✏ ミーティングの内容は透明性のために公開している
  - 話し合った過程や内容を公開することで、部外者の"なぜ?"を解決できるようにする

ECMAScriptがやっていること

- 📌 ProposalにはStage Xというラベリングをしている
  - Proposalの現在のステータスが分かるようにラベリングしている
- 🕐 Stage 3では開発者からフィードバックを得る方法/期間を提供している
  - 仕様確定前に十分なフィードバックを得る方法や期間を開発者に提供する

6行まとめ

» ✔ 更新内容は普段からコミットの段階で整理しておく

» 📝 リリースノートはGitHub Releaseで公開している

» 💭 複雑な設計問題に対してはパターンを使う

» 🎩 慎重な意思決定が必要な部分は時間を取ってミーティングを行う

» ✏ ミーティングの内容は透明性のために公開している

» 📌 Proposalの現在のステータスが分かるようにラベリングしている

» 🕐 仕様として確定する前に開発者からフィードバックを得る方法/期間を提供している

1行まとめ

大きなGitHubプロジェクトとそんなに変わらない

おわり

参考

- V8 JavaScript Engine: ES6, ES7, and beyond
- Previewing ES6 Modules and more from ES2015, ES2016 and beyond | Microsoft Edge Dev Blog
- azu/browser-javascript-resource: Browser JavaScript Resource.
- ECMAScript 6 compatibility table
- APIデザインの極意 Java/NetBeansアーキテクト探究ノート - インプレスブックス

引用

» p4 ES6 in Practice

» ECMAScript 2015: What Took It So Long?