技術書をソフトウェア開発する

jsprimer の 10 年から学ぶ継続的メンテナンスの技術

自己紹介

- Name : **azu**
- Twitter/X : @azu_re
- Website: Web scratch, JSer.info
- Open Source:
  - 🧹textlint, 🔑🛡️ secretlint, 📚HontKit

JavaScript Primer (jsprimer) とは

- 約 10 年、6 つのメジャーバージョンを経て更新され続けている JavaScript 入門書
- 書籍版 (amazon.co.jp/dp/4048931105) も販売中
- ウェブ版 (jsprimer.net) は無料で公開
- GitHub: asciidwango/js-primer
- 書籍版とウェブ版の内容に違いはなく、オープンソースで開発中

簡単年表

ECMAScript	jsprimer
2015	開発開始
2016	ウェブで公開
…	…
2019	v1 / 書籍の第 1 版リリース

簡単年表

ECMAScript	jsprimer
2020	v2
2021	v3
2022	v4 / 書籍の第 2 版リリース
2023	v5
2024	v6
2025	v7(予定)

本日のテーマ

- **Why:** なぜ jsprimer は始まり、更新され続けるのか？
- **How:** どうやって更新を実現しているのか？
- **Learn:** 技術書を「ソフトウェア開発」することから得られる知見
- **Future:** これからの jsprimer の展望

なぜ、TSKaigi で JavaScript の話をするのか？

TSKaigi のミッション

学び、繋がり、”型”を破ろう

なぜ TSKaigi で JavaScript の話をするのか？

TypeScript と JavaScript の関係性

- TSKaigi のミッション: 「学び、繋がり、”型”を破ろう」
- TypeScript から「型」を取り除くと JavaScript になる
- JavaScript を学ぶことは、TypeScript の理解を深めることに繋がる

```
$ node --experimental-strip-types
```

- ts-blank-spaceで発見された
- TypeScript の Design Goal として JavaScript と非互換な機能や変更を入れることはしないようになっている
  - https://github.com/Microsoft/TypeScript/wiki/TypeScript-Design-Goals
- TypeScript の `erasableSyntaxOnly` や Node.js の `--experimental-strip-types` フラグなどもあり、TypeScript ファイルを JavaScript として直接実行できるようになってきた

TypeScript から型を取り除くと JavaScript になる

TypeScript コード

```
class Cat<T> {
  public whiskers: number;
  public tail: T;

  constructor(count: number, tail: T) {
    this.whiskers = count;
    this.tail = tail;
  }
}

throw Error();
```

型を削除

JavaScript コード（型を削除後）

```
class Cat<T> {
  public whiskers: number;
  public tail: T;

  constructor(count: number, tail: T) {
    this.whiskers = count;
    this.tail = tail;
  }
}

throw Error();
```

TypeScriptから型を取り除くとJavaScript

「TypeScript」から「型」を引くと、
純粋な「JavaScript」が残る！

今日、TSKaigiで話すこと

「TypeScript」から「型」に関する話題を引くと、
純粋な「JavaScript」の話題が残る！

TypeScript から型を取り除けば JavaScript

TSKaigi 2025

TypeScript から 型(の話)を
取り除けば JavaScript(の話)が
できるカンファレンス

```
$ tskaigi --experimental-strip-types
```

Why: なぜ jsprimer は書かれたのか?

時代背景 (2015 年頃)

- ECMAScript 2015 (ES6) の登場
- JavaScript の大きな転換期
- ES2015 ベースの現代的な入門書が不足
- 「これ読んでおいて！」と渡せる現代的な書籍が必要

jsprimer の目的

「書き方」「作り方」「学び方」の 3 つの原則に基づく設計

- **書き方**: 基本文法を体系的に学ぶ
- **作り方**: 学んだ知識で実際にアプリを作る
- **学び方**: JavaScript の変化に対応できる素養を身につける

Why: なぜ JSPrimer は更新され続けるのか？

JavaScript は "Living Standard"

- JavaScript の仕様である ECMAScript は毎年更新
- 実行環境 (ブラウザ, Node.js) も進化

変化に対応できる基礎を身につけるため、書籍自体が変化に対応し続ける必要がある

– *JavaScript Primer* 改訂 2 版をリリースしました！*/JavaScript Primer* はなぜ更新され続けるのか？ | *Web Scratch*

生きてるものの宿命

依存がない静的なコードは時間で変化はしにくい、依存があるアクティブなコードは、コードが変わらなくても依存関係である周りが変化します。そのため 5 年間触っていないコードが、最新の環境ではそのままは動かないのはこれが理由です。

— *Working in Public*

技術書の宿命

- JavaScript 周りのエコシステムは変化し続ける
- エコシステムを扱った書籍は、ものすごい速さで古くなる

コードの状態と時間経過の課題

- コードには「静的状態」と「アクティブ状態」がある
  - 静的状態: 依存がなく、時間が経っても内容が変わらない
  - アクティブ状態: 外部のライブラリや環境に依存し、周囲の変化に影響される
- アクティブなコードは、依存先のバージョンアップなどで「ズレ」が生じ、動かなくなるリスクが高い
- 価値を保つには、継続的なメンテナンスが不可欠

書籍もコードと同じ

- 書籍も「静的状態」と「アクティブ状態」がある
  - 静的状態: 依存がなく、時間が経っても内容が変わらない
  - アクティブ状態: 外部のライブラリや環境に依存し、周囲の変化に影響される
- 書籍が何にも依存してないことはほぼない
  - → アクティブ状態であるが、いかに依存を綺麗に保つがメンテナンス的に重要

ECMAScript のアクティブな状態と静的な状態

- ECMAScript ではアクティブな状態と静的な状態を使い分けている
- アクティブな仕様 = Living Standard
  - https://tc39.es/ecma262 常に最新のものとして公開される
- 静的な仕様 = Snapshot
  - https://ecma-international.org/technical-committees/tc39/
  - ECMA のサイトで1 年に一度更新される

静的な書籍版とアクティブなウェブ版

- ECMAScript 仕様のモデルに真似る:
- **Living Standard**: ウェブ版 (jsprimer.net)
  - 常に最新版を維持
- **Snapshot**: 書籍版
  - 安定版として時々リリース(読みやすさ重視)

Why: なぜ静的なスナップショットを作るか

- 読みやすさ
- 読んでいる途中で内容が変わらないようにするため

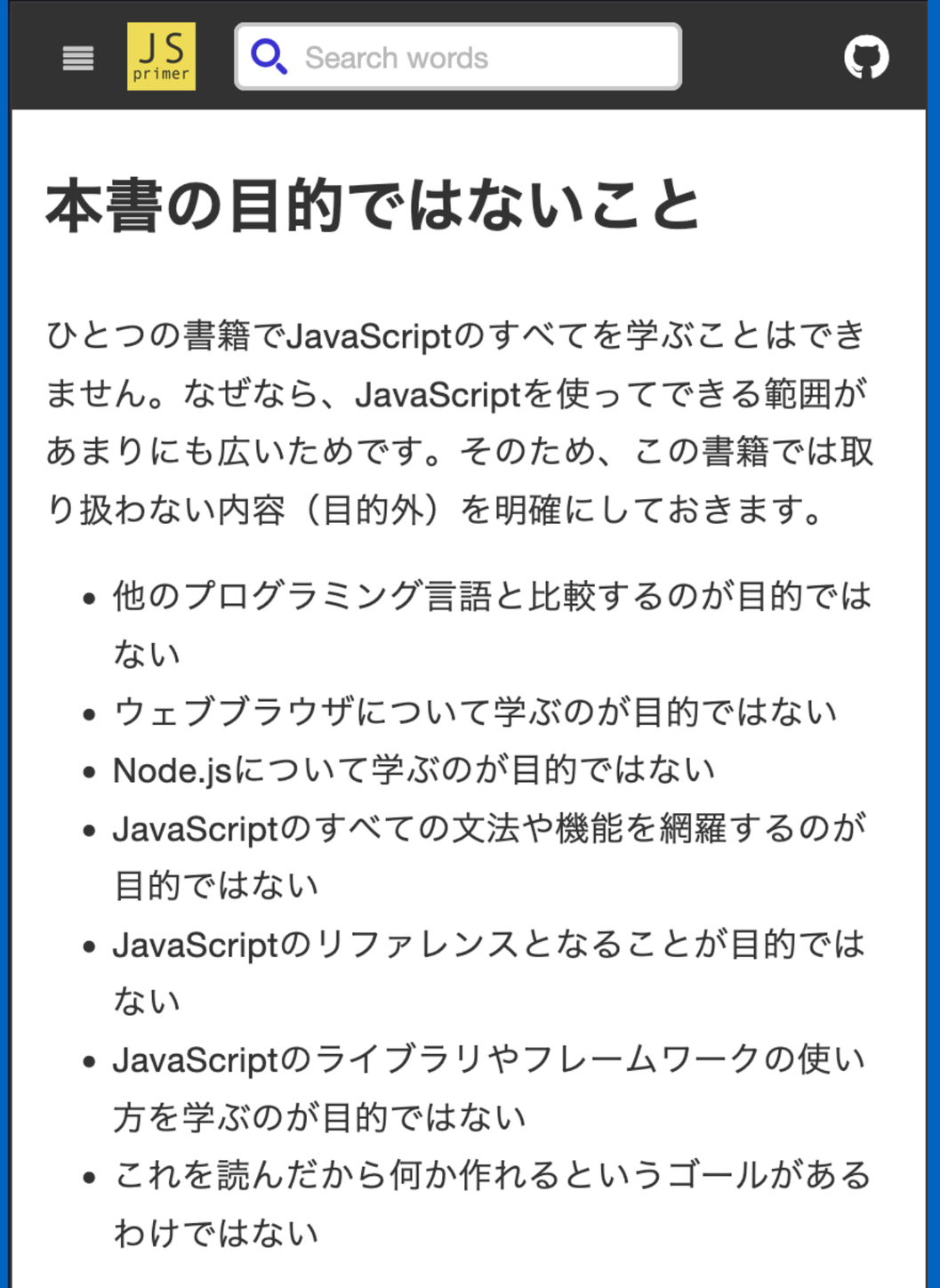
How: どうやって更新を実現しているのか？

ポイント

- 1. 変化を前提とした設計
- 2. 読みやすさを優先する
- 3. テスト
- 4. 変化の追従プロセス
- 5. オープンソース開発
- 6. コミュニティ
- 7. 経済的支援モデル

1. 変化を前提とした設計

- **スコープ管理:** 「目的ではないこと」を明確化し、**依存**を減らす。
  - ライブラリ解説ではなく、基礎的な言語機能にフォーカス
  - 依存関係の少ないコードを重視
  - 本書の目的 · Issue #103 · asciidwango/js-primer



2. 読みやすさを優先する

- 書きやすさよりも、読みやすさを重視
- 更新を継続的に行うには、何度も読む必要がある
- 書くより読む回数の方が多い
- 読むコストを下げる工夫 (構成、表現統一)

既知の言葉で未知を説明する

- 既知な情報から未知な情報へと書く[known-new contract]

「"known-new contract"とは、読者がすでに知っていること（以前に提示された情報）を最初に提示してから新しい情報を導入することで、書き手が文章間の結束をどのように達成するかを説明するために使われる言語学的概念である。議論を構築するにしても、情景を描写するにしても、概念を分析するにしても、ある文章から次の文章へと論理的に進行することが重要である。明確な文から文への進行は、読者の集中力を維持し、推論のパターンを容易に追うことを可能にする。」

---

known-new contract 英語だと"known-new contract"という概念が近いもの。既知から未知に書くから引用

既知から未知へ

具体的な例をいくつか挙げてみます。

- jsprimer という書籍は前から後ろに順番に読んでいく構造
- 未知の言葉がいきなり出てこないように、既知 -> 未知の順となるように説明

既知 から 未知 へ

- 結論を最初に、その後に説明を書く

1. まず紹介するコードの説明をする
2. その後にコードを表示する
3. その後に補足を説明する

文字へのアクセス

文字列の特定の位置にある文字にはインデックスを指定してアクセスできます。これは、配列における要素へのアクセスにインデックスを指定するのと同じです。

文字列では `文字列[インデックス]` のように指定した位置（インデックス）の文字へアクセスできます。インデックスの値は `0` 以上 `2^53 - 1` 未満の整数が指定できます。

```
const str = "文字列";
// 配列と同じようにインデックスでアクセスできる
console.log(str[0]); // => "文"
console.log(str[1]); // => "字"
console.log(str[2]); // => "列"
```

実行

また、存在しないインデックスへのアクセスでは配列やオブジェクトと同じように `undefined` を返します。

```
const str = "文字列";
// 42番目の要素は存在しない
console.log(str[42]); // => undefined
```

実行

3. テスト

- **ドキュメントの自動テスト:**
  - `textlint`: 文章校正、表現統一
  - `power-doctest`: サンプルコードの動作検証
- **CI/CD での継続的なテスト:** GitHub Actions でテストを自動化し、品質を維持
  - サンプルコードの検証、Integration Test の実装
- **読みやすさ分析:** `textstat` による文章の複雑さの可視化

textlint

- textlintは、自然言語の文章を校正するためのツール
- ESLint の自然言語向けの Linter
- 技術書むけのルールをベースに、辞書や書籍独自のルールを実装している
  - https://github.com/asciidwango/js-primer/blob/master/.textlintrc.js
- Maintainer Month: なぜ textlint を作ったか | Web Scratch

textlint を使うことで書きやすくする

- textlint は読みやすさのためのルールが色々とある
- 一方でルール決めることで書きやすさが上がる
- 特に LLM など、全ての単語を人間が書いてるわけじゃない
- こういった統一性を担保するのが Linter の役割であるので、読みやすさと書きやすさをあげてくれる

書籍独自のルール

- fix: prototype を表すのに # を使わないようにする
- 静的メソッドの表記統一 と textlint ルールの追加 #1797
- 表記揺れなども textlint のルールとして書いて吸収している

power-doctest

- 書籍におけるサンプルコードは動くのが正解なコード、エラーになることが正解のコードがある
- どちらも実行してその結果が期待通りかを確認するテストが必要
- power-doctestは、文章の中にあるコードをテストする

元の文章 ⬅️

次のコードは、数値の文字列を二つ受け取り、合計を返す関数を定義しています。

```
const sum = (a, b) => a + b;
console.log(sum(1, 2)); // => 3
```

▶️power-doctest による変換

- コードを抽出して、Assertion に変換してテストとして実行する

```
const sum = (a, b) => a + b;
assert.strictEqual(sum(1, 2), 3);
```

元の文章 ⬅️

変数名に数字を含めることはできますが、変数名を数字から開始することはできません。
これは変数名と数値が区別できなくなってしまうためです。

<!-- doctest:SyntaxError -->

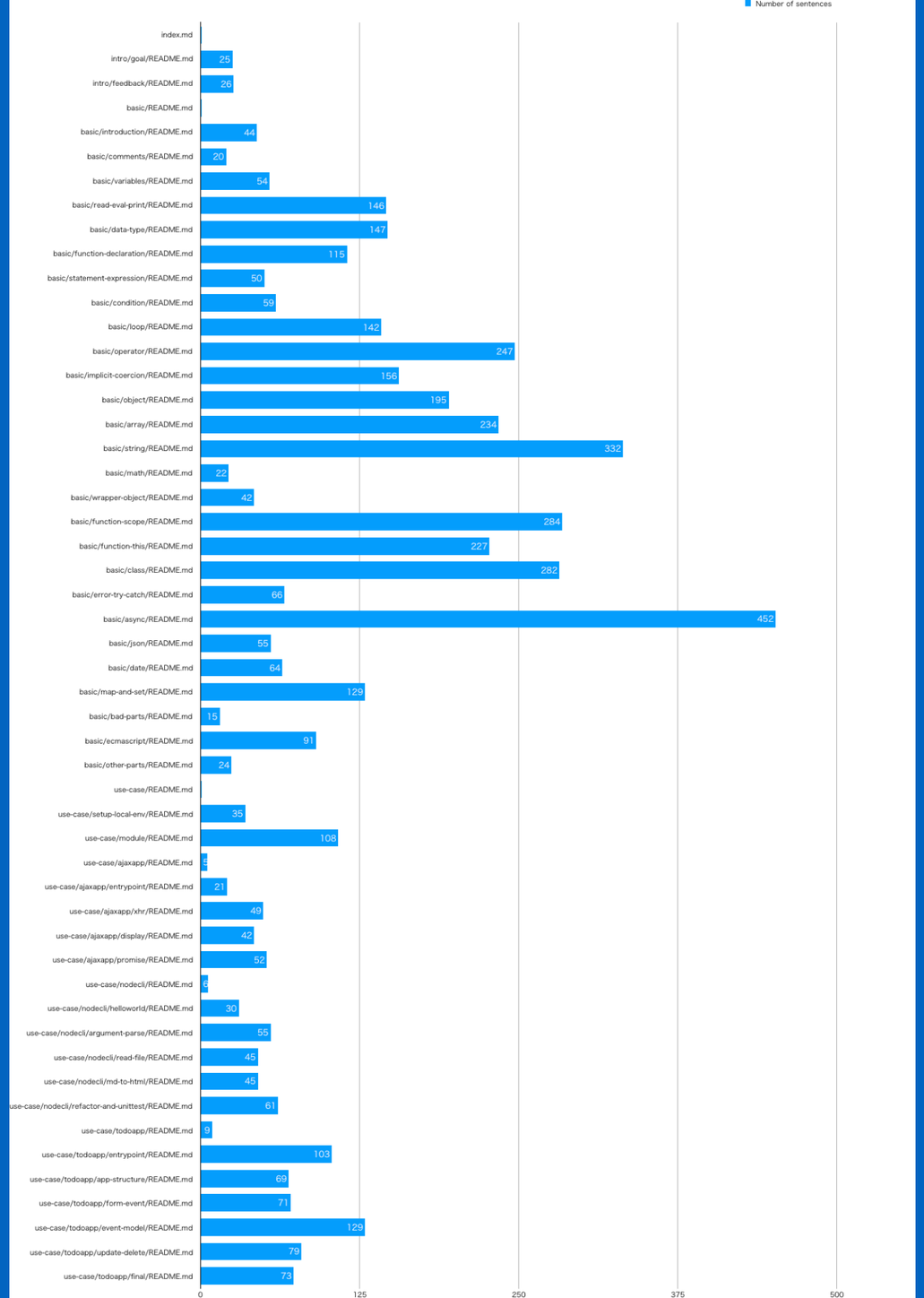
```
let 1st; // NG: 数字から始まっている
let 123; // NG: 数字のみで構成されている
```

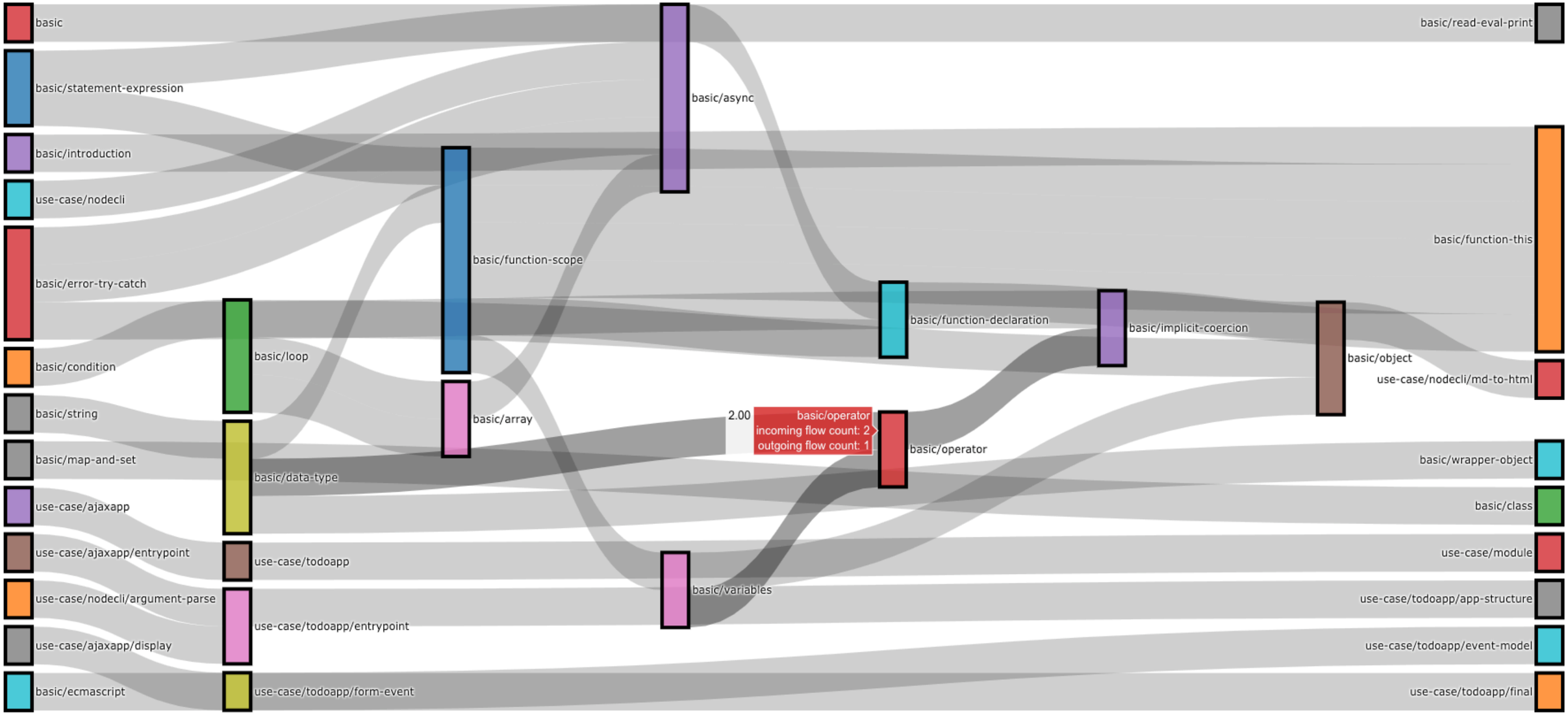
▶️power-doctest によるテスト

- コードブロックを抽出して、実行した結果の期待値をコメントから取得
- 実行した結果が `SyntaxError` になることを確認する

textstat での可視化

- 文章を書いていくと、巨大になって流れが難しくなる
- フローを分析するようなツールが必要
- → textstat を書いて文章の依存関係や文字数を可視化した
- 章ごとのページ量を可視化する · Issue #554 · asciidwango/js-primer


Number of sentences
index.md
intro/goal/README.md 25
intro/feedback/README.md 26
basic/README.md
basic/introduction/README.md 44
basic/comments/README.md 20
basic/variables/README.md 54
basic/read-eval-print/README.md 146
basic/data-type/README.md 147
basic/function-declaration/README.md 115
basic/statement-expression/README.md 50
basic/condition/README.md 59
basic/loop/README.md 142
basic/operator/README.md 247
basic/implicit-coercion/README.md 156
basic/object/README.md 195
basic/array/README.md 234
basic/string/README.md 332
basic/math/README.md 22
basic/wrapper-object/README.md 42
basic/function-scope/README.md 284
basic/function-this/README.md 227
basic/class/README.md 282
basic/error-try-catch/README.md 66
basic/async/README.md 452
basic/json/README.md 55
basic/date/README.md 64
basic/map-and-set/README.md 129
basic/bad-parts/README.md 15
basic/ecmascript/README.md 91
basic/other-parts/README.md 24
use-case/README.md
use-case/setup-local-env/README.md 35
use-case/module/README.md 108
use-case/ajaxapp/README.md
use-case/ajaxapp/entrypoint/README.md 21
use-case/ajaxapp/xhr/README.md 49
use-case/ajaxapp/display/README.md 42
use-case/ajaxapp/promise/README.md 52
use-case/nodecli/README.md
use-case/nodecli/helloworld/README.md 30
use-case/nodecli/argument-parse/README.md 55
use-case/nodecli/read-file/README.md 45
use-case/nodecli/md-to-html/README.md 45
use-case/nodecli/refactor-and-unittest/README.md 61
use-case/todoapp/README.md 9
use-case/todoapp/entrypoint/README.md 103
use-case/todoapp/app-structure/README.md 69
use-case/todoapp/form-event/README.md 71
use-case/todoapp/event-model/README.md 129
use-case/todoapp/update-delete/README.md 79
use-case/todoapp/final/README.md 73
0
125
250
375
500

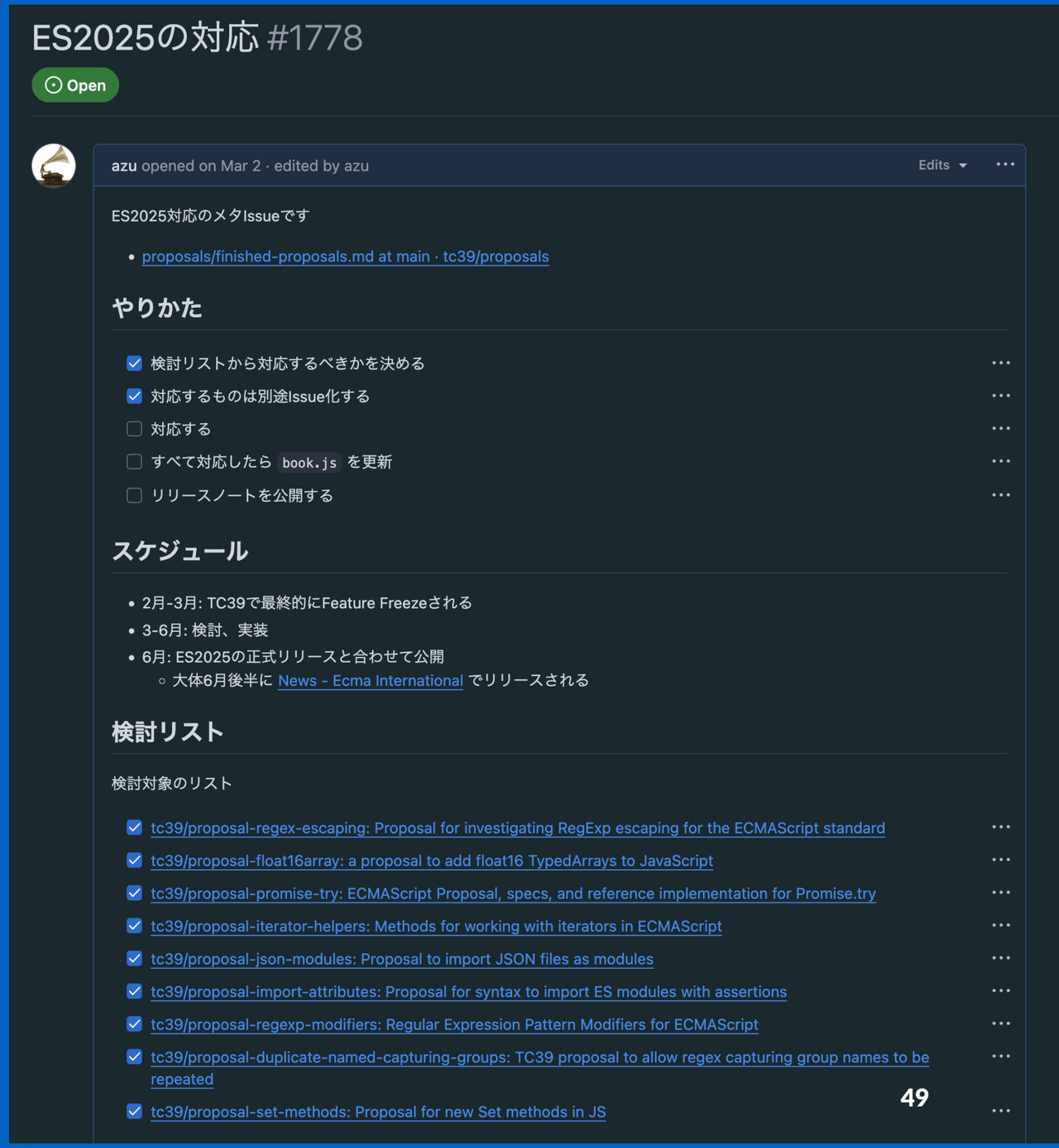

basic
basic/statement-expression
basic/introduction
use-case/nodecli
basic/error-try-catch
basic/condition
basic/string
basic/map-and-set
use-case/ajaxapp
use-case/ajaxapp/entrypoint
use-case/nodecli/argument-parse
use-case/ajaxapp/display
basic/ecmascript
basic/loop
basic/data-type
use-case/todoapp
use-case/todoapp/entrypoint
use-case/todoapp/form-event
basic/function-scope
basic/array
basic/async
basic/variables
2.00
basic/operator
incoming flow count: 2
outgoing flow count: 1
basic/operator
basic/function-declaration
basic/implicit-coercion
basic/object
basic/read-eval-print
basic/function-this
use-case/nodecli/md-to-html
basic/wrapper-object
basic/class
use-case/module
use-case/todoapp/app-structure
use-case/todoapp/event-model
use-case/todoapp/final

textstate を見てやっていること

- 順序が重要な書籍なので、滑らかに量を増やしていく
- 急に増やすとそこで諦めて止まる人が出てきてしまう
- 既知を使って未知を説明するには流れが重要
- リンクリストを見て、未知が先に出てこないかを確認してる

4. 変化の追従プロセス

- Issue でタスク管理し対応していく
  - ES2025 の対応 · Issue #1778 · asciidwango/js-primer
- ECMAScript は毎年のリリーススケジュールが決まっているので合わせたサイクル



ES2025の対応 #1778

Open

azu opened on Mar 2 · edited by azu Edits

ES2025対応のメタIssueです

- proposals/finished-proposals.md at main · tc39/proposals

やりかた

- [x] 検討リストから対応するべきかを決める
- [x] 対応するものは別途Issue化する
- [ ] 対応する
- [ ] すべて対応したら `book.js` を更新
- [ ] リリースノートを公開する

スケジュール

- 2月-3月: TC39で最終的にFeature Freezeされる
- 3-6月: 検討、実装
- 6月: ES2025の正式リリースと合わせて公開
  - 大体6月後半に News - Ecma International でリリースされる

検討リスト

検討対象のリスト

- [x] tc39/proposal-regex-escaping: Proposal for investigating RegExp escaping for the ECMAScript standard
- [x] tc39/proposal-float16array: a proposal to add float16 TypedArrays to JavaScript
- [x] tc39/proposal-promise-try: ECMAScript Proposal, specs, and reference implementation for Promise.try
- [x] tc39/proposal-iterator-helpers: Methods for working with iterators in ECMAScript
- [x] tc39/proposal-json-modules: Proposal to import JSON files as modules
- [x] tc39/proposal-import-attributes: Proposal for syntax to import ES modules with assertions
- [x] tc39/proposal-regexp-modifiers: Regular Expression Pattern Modifiers for ECMAScript
- [x] tc39/proposal-duplicate-named-capturing-groups: TC39 proposal to allow regex capturing group names to be repeated
- [x] tc39/proposal-set-methods: Proposal for new Set methods in JS

ES2025 の対応の流れ

1. Issue を立てる
2. どの Proposal を対応するかを決める
3. リポジトリで ES2025 が動くかをツールのアップデート
4. 改めて読み直して何を削るかを決める(ここで読み直してる)
5. 書く

ES2025 で対応する Issue

- import attributes
- ES2020: Dynamic Import (読み直していて増えた)
- `RegExp.escape`
- Set Methods for JavaScript)
- Iterator Helpers

大きな変更への対応

- ドキュメントに対する Design Doc (OUTLINE.md) を書く
  - 例: Iterator Helpers 対応 (#1782, OUTLINE)
- 大きな変更は与える影響範囲が広い、または書く量が多い
  - 書いているうちに目的がずれてくる
  - 最初に目的、目的ではないこと、アウトラインを決めておいてずれにくくする

Iterators and Generators

このドキュメントについて

この章では、JavaScriptにおけるイテレーション（反復処理）の仕組みであるイテレータと、イテレータを簡単に作成するためのジェネレータ関数について解説。ES2025で導入されるイテレータの新しいメソッドについても触れる。

想定読者

- JavaScriptの基本的な文法（特に配列、オブジェクト、関数、`for...of` ループ）の理解
- 非同期処理やデータストリームの扱いへの興味

背景

- ES2015でのイテレータとジェネレータ導入後、JavaScript Primerでは基本的な解説に留まっていた
- 近年の非同期処理の普及やデータストリーミング技術（例: LLMのServer-Sent Events）利用増加に伴う、イテレータやジェネレータの重要性のあがってくる
- ES2025でのイテレータの新しいメソッド導入によるイテレータ操作の簡便化・宣言的記述への対応。基礎となるイテレータとジェネレータの解説の必要性

```
// 擬似的な応答チャンクを生成するジェネレータ関数
function* generateLlmChunks() {
  yield "応答チャンク1 ";
  yield "応答チャンク2 ";
  yield "応答チャンク3";
}

// ジェネレータからチャンクを取得し、結合して表示
let fullResponse = "";
for (const chunk of generateLlmChunks()) { fullResponse += chunk; }
console.log("LLM応答(擬似):", fullResponse);
```

目的

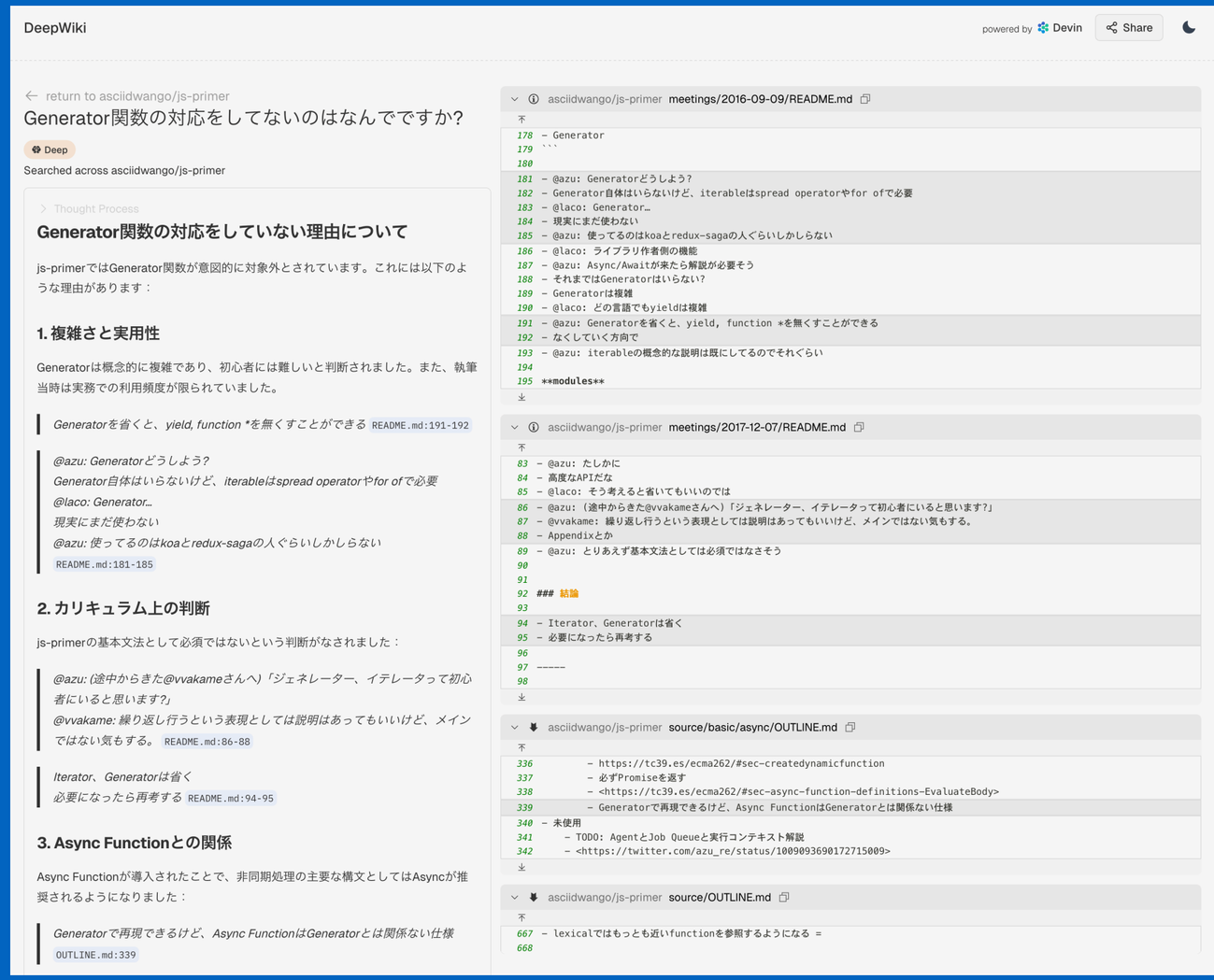
- Iterator/Generatorを見た時になんだこれ?と思わないようにする
- IterableプロトコルとIteratorプロトコルの仕組みの理解
- `for...of` ループの動作原理の理解
- なぜイテレータが必要なのか、どのように使うのかの理解

5. オープンソースとしての開発

- 書籍として出版予定のものとして書いていた
- ただし、最初からオープンソースとして公開しながら開発した
- すべての意思決定が GitHub 上にある
- すべての議事録が GitHub 上にある
- すべての文章/コードが GitHub 上にある

Deep Wiki

- Deep Wiki で検索してる例
- 議事録が全てリポジトリにあるため



6. コミュニティ

- jsprimer はオープンソースのプロジェクト
- 今までで100 人以上のコントリビューターがいる

コミュニティの力

- Contribute のハードルをどれだけ下げられるか
  - 文章の間違いに気づいたら・JavaScript Primer #jsprimer
  - 明確な Contribution Guide
  - 初めて GitHub 使う人も多いので、ブラウザだけで修正できる方法の案内をする
  - 繰り返す

誰でも Contribute できるように Contribute する

- CONTRIBUTING.md は誰もが読むわけじゃない
- ガイドを読まなくても Contribute できるようにする
- 右下のボタンから、今見てるページの Issue を立てられる
- ここからコミュニケーションを初めて、PR を作ってもらうところまでサポートする

文章の間違いに気づいたら

- バグがない書籍はない
- バグを見つけたら、すぐに報告してもらう
- 文章の間違いを見つけたら、右下のボタンから報告してもらう

文章の間違いに気づいたら

まったくバグがないプログラムはないのと同様に、まったく間違いのない技術書は存在しません。この書籍もできるだけ間違い（特に技術的な間違い）を減らせるように努力していますが、どうしても誤字脱字や技術的な間違い、コード例の間違いなどを見落としている場合があります。

そのため「この書籍には間違いが存在する」と思って読んでいくことを推奨しています。もし、読んでいて間違いを見つけたなら、ぜひ報告してください。

また、文章の意味や意図がわからないといった疑問を持つこともあるでしょう。そのような疑問もぜひ報告してください。

もし、その疑問が実際には間違いではなく勘違いであっても、回答をもらうことで自分の理解を修正できます。そのため、疑問を問い合わせても損することはないはずです。

この書籍はGitHub上で公開されているため、GitHubリポジトリのIssueとしてあなたの疑問を報告できます。

- 書籍や内容に対する質問 → こちらから質問できます
- 内容のエラーや問題の報告 → こちらからバグ報告できます
- 内容をもっと詳細に解説する提案 → こちらから提案できます
- 新しいトピックなどの提案 → こちらから提案できます
- その他のIssue → その他のIssueはこちらから

また、GitHub Discussions（掲示板）へIssueにするのは気が引けると思った質問を書いたり、JavaScript Primerを読んだ感想などを書き込めます。

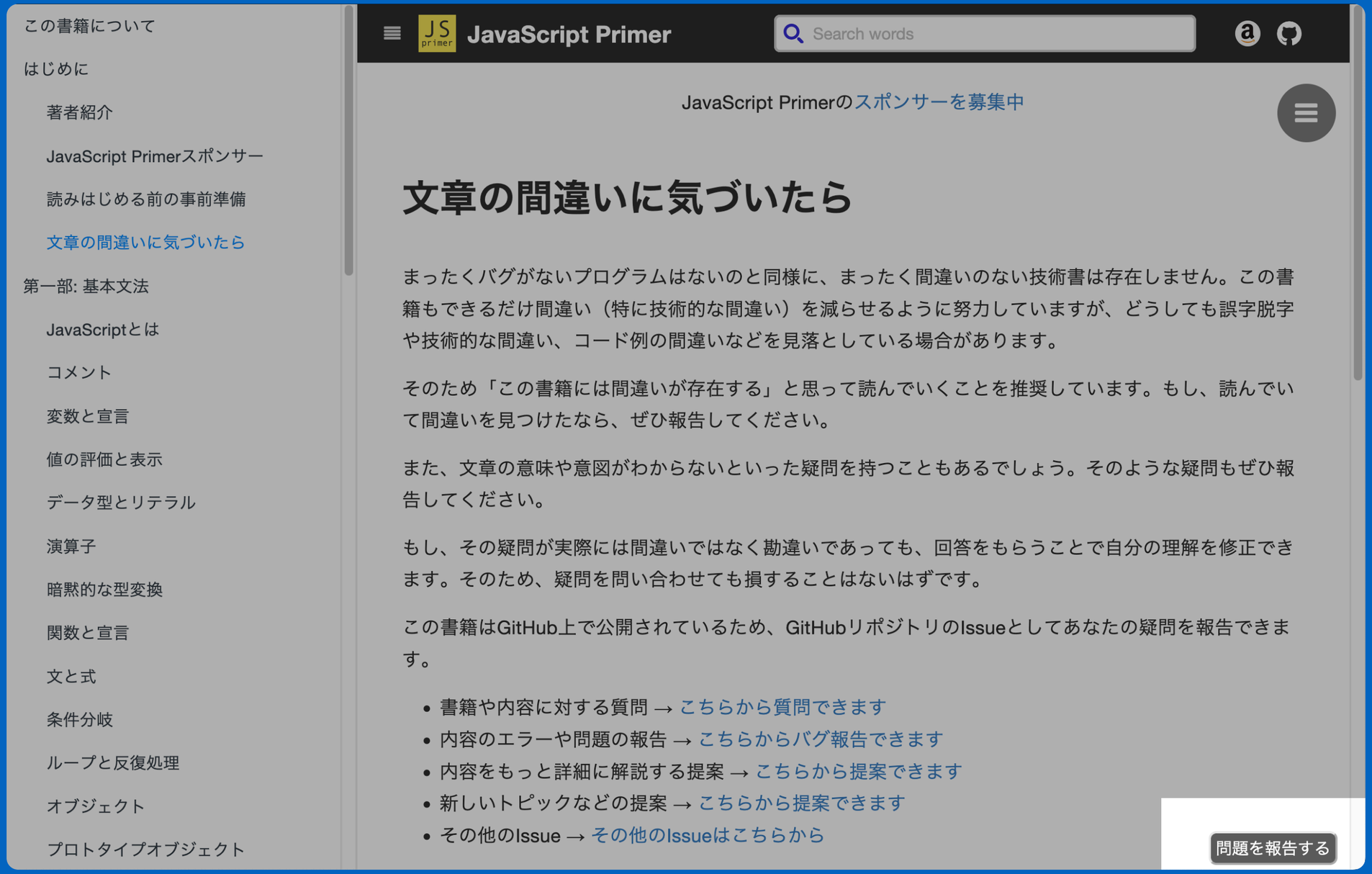
- Discussions · asciidwango/js-primer

GitHub Discussions（掲示板）の使い方については、次のスレッドにまとめています。

- 👋 ようこそ JavaScript Primer へ！ · Discussion #1304 · asciidwango/js-primer

GitHubのアカウントを持っていない方は、次のフォームから報告できます。

https://goo.gl/forms/lOx4ckFyb0fB9cBM2


この書籍について
はじめに
著者紹介
JavaScript Primerスポンサー
読みはじめる前の事前準備
文章の間違いに気づいたら
第一部: 基本文法
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
演算子
暗黙的な型変換
関数と宣言
文と式
条件分岐
ループと反復処理
オブジェクト
プロトタイプオブジェクト
JS primer
JavaScript Primer
Search words
JavaScript Primerのスポンサーを募集中
文章の間違いに気づいたら
まったくバグがないプログラムはないのと同様に、まったく間違いのない技術書は存在しません。この書籍もできるだけ間違い（特に技術的な間違い）を減らせるように努力していますが、どうしても誤字脱字や技術的な間違い、コード例の間違いなどを見落としている場合があります。
そのため「この書籍には間違いが存在する」と思って読んでいくことを推奨しています。もし、読んでいて間違いを見つけたなら、ぜひ報告してください。
また、文章の意味や意図がわからないといった疑問を持つこともあるでしょう。そのような疑問もぜひ報告してください。
もし、その疑問が実際には間違いではなく勘違いであっても、回答をもらうことで自分の理解を修正できます。そのため、疑問を問い合わせても損することはないはずです。
この書籍はGitHub上で公開されているため、GitHubリポジトリのIssueとしてあなたの疑問を報告できます。
書籍や内容に対する質問 → こちらから質問できます
内容のエラーや問題の報告 → こちらからバグ報告できます
内容をもっと詳細に解説する提案 → こちらから提案できます
新しいトピックなどの提案 → こちらから提案できます
その他のIssue → その他のIssueはこちらから
問題を報告する

目的: 変化に対応できるようにする

- なんでこんなことをしているのか
- jsprimer は変化に対応できるようにするために、jsprimer を変化させ続ける
- 現在のソフトウェアの変化の中心にはオープンソースがある
- そのため、オープンソースに関わってもらうことで変化に対応できるようにする

7. 経済的支援モデル

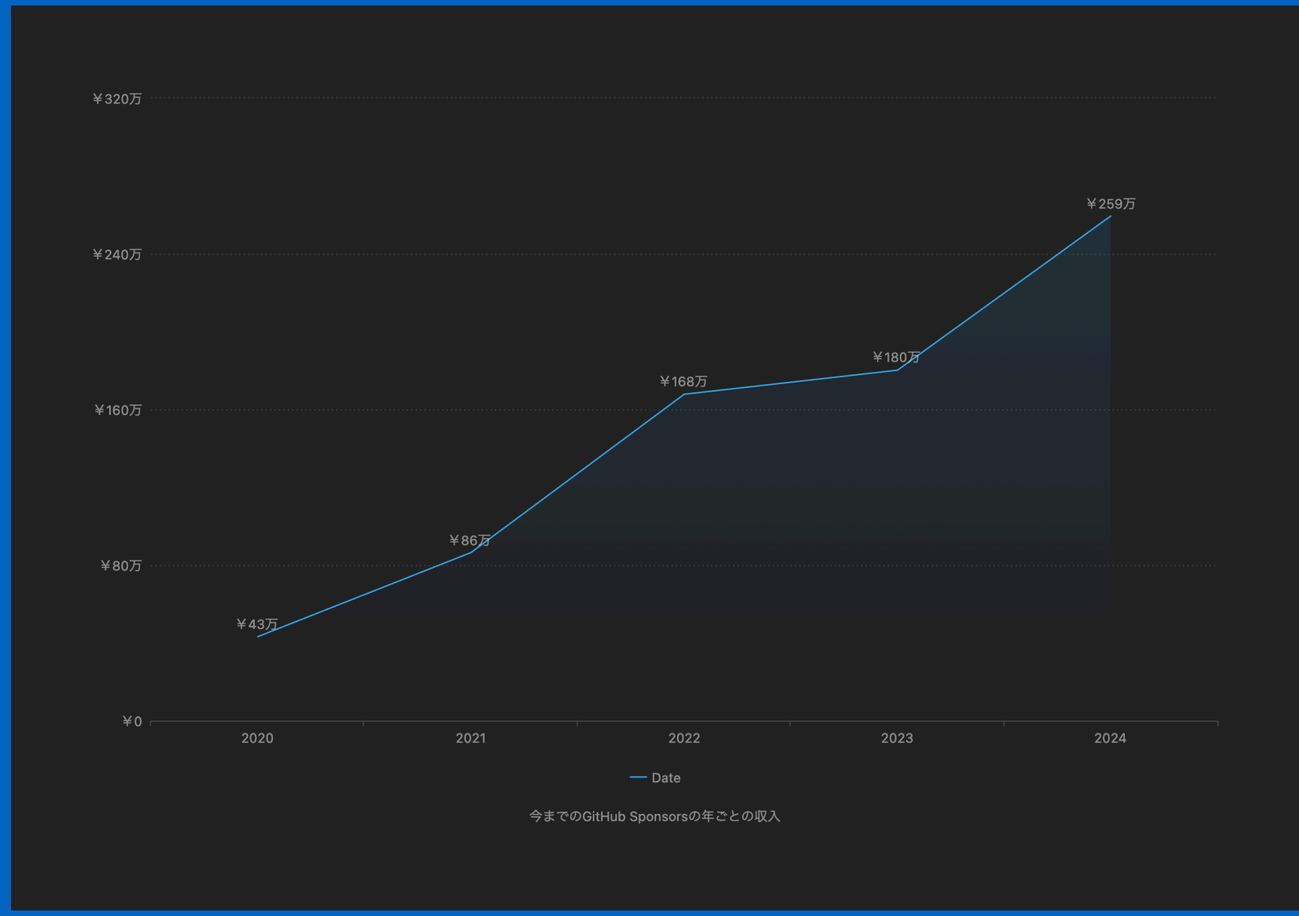
- 書籍: JavaScript Primer 改訂 2 版 迷わないための入門書
- Sponsor @azu on GitHub Sponsors
- Open Collective - jsprimer

書籍

- 書籍版 (amazon.co.jp/dp/4048931105) も販売中

GitHub Sponsors

- Sponsor @azu on GitHub Sponsors でスポンサーを募集している
- 毎年収支も公開している
  - GitHub Sponsors の募集を始めてから 2 年が経ったので振り返る | Web Scratch
  - GitHub Sponsors の収入 @ 2022 | Web Scratch
  - GitHub Sponsors の収入 @ 2023 | Web Scratch
  - オープンソース活動の振り返り/GitHub Sponsors の収入まとめ @ 2024 | Web Scratch



プロジェクトの経済モデル

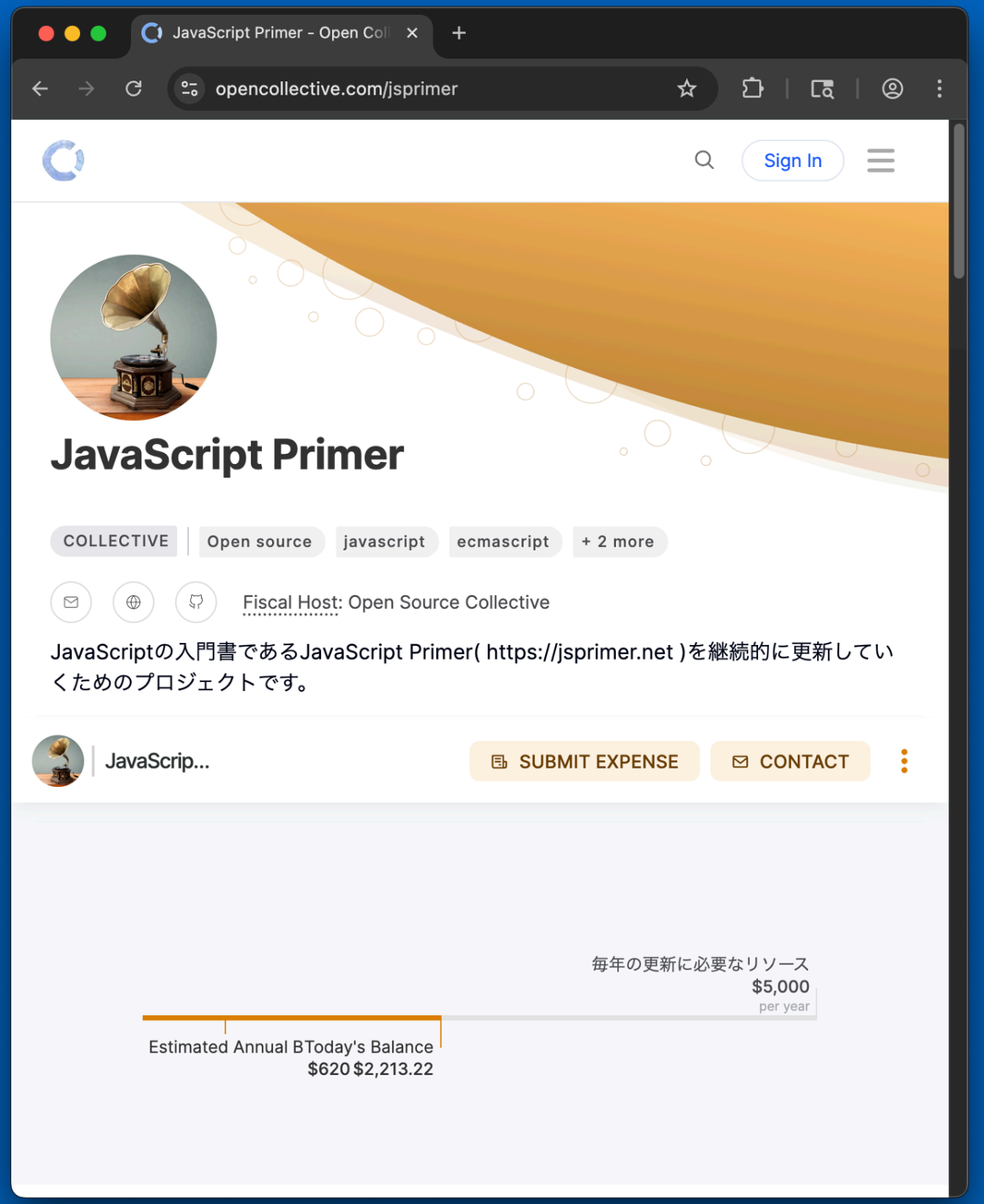
JavaScript Primer のコスト

- 1 年ごとに 1 度メジャーアップデートしている
- メジャーアップデートには大体 30 日分ぐらいの労力がかかる
- このコストの試算は 2,882 円 x 8 時間 x 30 日 = 大体 70 万円/Year [avg]
  - 詳細: JavaScript Primer 改訂 2 版をリリースしました！/JavaScript Primer はなぜ更新され続けるのか？
- このコストを補う仕組みを Open Collective で作る

[avg] 2,882 円は全国のプログラマの平均時給: 出典

Open Collective

- 継続的な更新を支えるための資金調達プラットフォーム
- 個人や企業がプロジェクトを支援可能
- 支援金は透明性を持って管理され、プロジェクトの維持や貢献者への還元に使用



支援の方法

1. **単発支援**: 自由な金額で支援

2. **定期支援**: 毎月または毎年の定額支援

3. **企業スポンサー**: ロゴ掲載や特典付きの支援プラン

全部募集中！

支援金の使い道

- jsprimer にはContributing Expenses Policyがある
- Contribute してもらったタスクの Point に応じして、Open Collective の予算から支払っている
- Contributing Expenses Policy には、計算プロセス、請求プロセス、支払いプロセスが書かれている

Point	Description
0	些細な変更
1	2 よりは簡単
2	大体 1 日分やると終わる想定
3	2 よりは難しい
5	かなり難しい
8	難易度がとても高いので、できる人は限られる

具体例: Contributing Expenses Policy

- JavaScript Primer - Open Collectiveの年間の予算は$620.00 USD(2025-05-22)
- 年間の更新コストは 30 日で、Point に直すと 60 Point が年間必要なコスト
- 1 Point あたりのコストは$10.33
- なので、2Point のタスクは大体$20 ぐらいを支払える

- Contributing Expenses Policy にこの計算ツールも入ってます

```
const yearlyEstimatedBudget = 620; // Open Collectiveの推定年間予算($ドル)
const yearlyWorkloadPoints = 60; // 1年間のPoints
const onePointCost = yearlyEstimatedBudget / yearlyWorkloadPoints;
console.log({ onePointCost }); // => $10.333333333333334

const costOfPoint = 2; /// 2 Points
const cost = onePointCost * costOfPoint;
console.log({ cost }); // => $20.666666666666668
```

詳細情報

- jsprimer の Open Collective ページ
  - https://opencollective.com/jsprimer
- JavaScript Primer スポンサーの目的や特典などの紹介ページ
  - https://jsprimer.net/intro/sponsors/

Thanks to Sponsors

Azu

管理者

Leave a public
essage (optional)

Suguru Inatomi

コアコントリビューター

Shintaro Kaneko

$1,000USD

株式会社コクチョウ (Kok...

Yearly Gold Sponsors

$1,000USD

t-cool

$120USD

RoyMcCrain

$110USD

Kesin11

$105USD

keisuke kudo

$70USD

NA

naoya7076

$50USD

Kiyohito Keet...

$50USD

Ry hei Tsu a

$50USD

me tan

$40USD

EH

Ei ha hiva

$20USD

TY

To oka Yuki

$20USD

zaq1tomo

$20USD

tacamy

$20USD

Guest

$20USD

ysaito

$10USD

Ken S.

$5USD

Yuji Yamaguchi

$5USD

sasazame

$5USD

F4

f4ah6o

$2USD

メトリクス分析

- JavaScript Primer スポンサー · JavaScript Primer #jsprimerでダッシュボードを公開している
- JavaScript Primer Dashboard › サマリー｜全体

サマリー｜全体

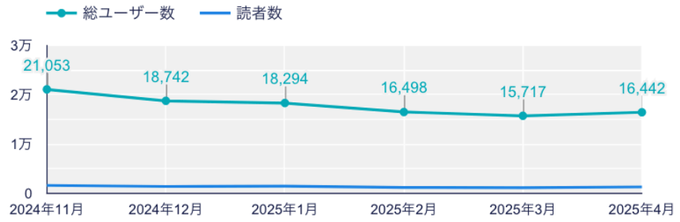
デバイス　イベント　2025/04/01 - 2025/04/30

総ユーザー数
11,880
5.9%

読者数の割合
11%
6.7%

読者数
1,334

指定期間内(デフォルト1ヶ月)で訪問したユーザーの何%が読者(3ページ以上読んでいる人)になったか



	ページタイトル	URL	総ユーザー数	エンゲージメント率	PV数
1.	JavaScript Primer - 迷わないための入門書 #jsprimer	jsprimer.net/	2,400	61%	4,404
2.	変数と宣言 · JavaScript Primer #jsprimer	jsprimer.net/bas…	1,861	62%	2,603
3.	コメント · JavaScript Primer #jsprimer	jsprimer.net/bas…	1,382	56%	1,718
4.	第一部: 基本文法 · JavaScript Primer #jsprimer	jsprimer.net/basic/	908	75%	1,587
5.	関数と宣言 · JavaScript Primer #jsprimer	jsprimer.net/bas…	916	59%	1,474
6.	JavaScriptとは · JavaScript Primer #jsprimer	jsprimer.net/bas…	904	81%	1,423
7.	非同期処理:Promise/Async Function · JavaScript Primer #jsprimer	jsprimer.net/bas…	788	54%	1,398
8.	演算子 · JavaScript Primer #jsprimer	jsprimer.net/bas…	706	65%	1,205
9.	データ型とリテラル · JavaScript Primer #jsprimer	jsprimer.net/bas…	647	71%	1,199
10.	オブジェクト · JavaScript Primer #jsprimer	jsprimer.net/bas…	778	62%	1,196
		総計	11,880	48%	38,289

1 - 100 / 404

変数

データ探索名:
イベントのセグメント

過去 30 日間
4月22日～2025年5月21日

セグメント
- 読者
- コードを実行
- 問題を報告
- タブレット トラフィック
- ウェブのトラフィック

ディメンション
- イベント名
- 性別
- 国
- デバイス カテゴリ

設定

- 読者
- 問題を報告
- コードを実行

内訳
ディメンションをドロップするか選択してください

最初の行
1

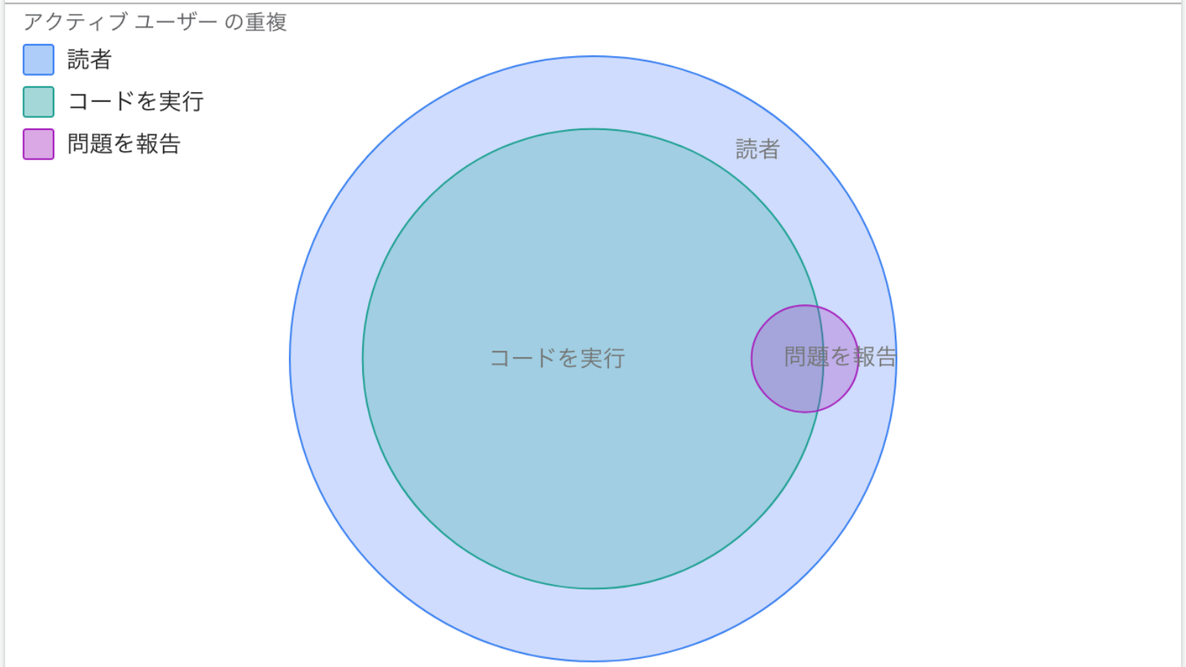
表示する行数
10

値
アクティブ ユーザー
指標をドロップするか選択してください

フィルタ
ディメンションや指標をドロップするか選択してください

セグメントの ...

アクティブ ユーザーの重複



セグメント セット	↓アクティブ ユーザー
1 読者	1,223
2 読者 のみ	972
3 コードを実行	706
4 コードを実行 のみ	454
5 読者 + コードを実行	251
6 問題を報告	38
7 問題を報告 のみ	32

アウトカム設計

- "変化に対応できる人"(jsprimer の目的) という定義を分解していき、次のようなアウトカムに分解できる

  1. 知識を得る → 読者数

  2. 試行錯誤できる → コードを実行している人の数

  3. コントリビュートできる →　　Issue 報告ボタンを押した人の数

- メトリクスを計測して、どれだけの人が変化に対応できるようになったかを計測しながら改善していく

技術書をソフトウェア開発する

技術的なメンテナンス基盤の構築

- 自動テスト・CI による品質担保
- バージョン管理による変更の追跡
- Issue と Pull Request による透明性の高い開発
- 外部からの貢献を促す環境整備

変化に強い設計思想の採用

- 依存関係を最小化する設計
- 構成の見直しと再構築
- 目的と範囲の明確化
- 「読みやすさ」を中心とした設計

継続的な改善サイクルの確立

- 読者/ユーザーからのフィードバック収集
- 定期的な更新スケジュール
- メトリクスによる改善点の可視化
- コミュニティベースの運営モデル

まとめ

真面目に技術書書くのと

真面目にソフトウェア開発するのは同じ！