報道各位

2018年5月2日
株式会社インプレスR&D
https://nextpublishing.jp/

ソースを解析・加工して生産性に差をつける！

『JavaScript AST 入門』発行

技術書典シリーズ最新刊！

インプレスグループで電子出版事業を手がける株式会社インプレスR&Dは、『JavaScript AST 入門』(著者:佐々木 俊介)を発行いたしました。

『JavaScript AST入門』
https://nextpublishing.jp/isbn/9784844398226

著者:佐々木 俊介
小売希望価格:電子書籍版 1600円(税別)／印刷書籍版 1800円(税別)
電子書籍版フォーマット:EPUB3／Kindle Format8
印刷書籍版仕様:B5判／カラー／本文72ページ
ISBN:978-4- 8443-9822-6
発行:インプレスR&D

＜＜発行主旨・内容紹介＞＞

【JavaScriptのASTを理解して開発生産性をUP！】

本書はJavaScriptのソースコードを扱いやすいように加工されたデータ構造であるASTとそのツールエコシステムの解説書です。他の言語と異なりトランスパイルなども一般的なJavaScriptでは、ASTを自由に操作することでそのエコシステムを使いこなし、ソースコードの解析・加工が簡単に行うことができます。

JavaScriptのASTを理解して、プログラム開発の生産性を大幅に向上させましょう。

(本書は、次世代出版メソッド「NextPublishing」を使用し、出版されています。)

JavaScript の AST でできること、導入方法を詳しく解説

第1章　JavaScript ASTがなぜ簡単なのか？

本来のASTはコンパイラの内部表現に過ぎないため、言語利用者の大半には縁のないものです。しかしJavaScriptは他の処理系とは違う歴史を持っているためASTが身近です。それはブラウザの互換性との戦いの歴史により、トランスパイルと呼ばれるソースコードをソースコードに変換するのが一般的だという特殊な事情からくるものです。開発時には最新版の言語仕様を使いつつ、トランスパイルを行ってWebブラウザ上で問題ないソースコードを動かすのです。

最近のJavaScript開発ではBabel[1]というトランスパイラを使うのが定番です。Babelはプラグインを使って、色々な拡張がされたソースコードを自動的にブラウザで動くソースコードに変換してくれます。特に最近は`babel-preset-env`[2]を使えば、難しいことを考えなくても最適なコードに変換できるようになりました。

Babelはトランスパイラですが、同時にASTを使ったエコシステム・ライブラリ群でもあります。Babelのプラグインはまさにの変換をする小さなプログラムです。Babelを使ったメタプログラミングはBabelのプラグインを書けばだいたい達成できます。

さらにJavaScriptのASTはBabelの専売特許ではありません。もともとESTreeという標準仕様があるため、Babelに限らずJavaScript ASTエコシステムでは知識や経験を使い回せます。そういった手厚いASTのエコシステムのおかげで、他の言語では考えられないくらい簡単にASTを使ってソースコードの解析・加工などができてしまうのです。

ASTを使ったメタプログラミングには、分厚いコンパイラの本を読む必要はありません。是非本書を読んでカジュアルにJavaScriptをハックしてみましょう。

1.1　ASTでできること

ASTを使って何ができるでしょうか？ ASTを使うとソースコードの解析や加工ができます。対象のソースコードに手を付けずにハックできます。

1.1.1　デバッグやテストに便利なツールを作る

第3章では、対象のソースコードを一切変更せずに、動的に依存性注入（Dependency Injection）するというハックをたったの100行程度で実装しています。

これにより、ユニットテストに向いていないソースコードの一部分を切り出してテストコードを書いたり、モックを注入したりできます。テストの無いソースコードにテストを導入するのは一般的には面倒ですが、動的にソースコードを書き換えれば、その一部を切り取って簡単にテストを順次追加していけるのです。

1.1.2　ソースコードの整形

JavaScriptでは、ESlintsやprettierというソースコードの整形ツールがよく利用されますが、これらにもASTが使われます。

1.1.3　コメントの活用

ASTではコメントを簡単に取得できるため、そのコメントをさまざまな目的で活用できます。JavaDocのような、コメントに関数やクラスの仕様ドキュメントを書く習慣がありますが、そういったツールもJavaScript ASTを使えば簡単に作れます。これは付録Aで軽く触れています。

1.1.4　ソースコードの静的解析や最適化

一般的に、テストのカバレッジの取得やlintツールなど、さまざまな静的解析ツールにはASTが使われています。ASTのエコシステムの中にはソースコードの静的解析をしてくれるもの、支援するものなどもあります。JavaScriptが小規模の目的にしか使えない“おもちゃ”だった時代はとっくの昔に終わりました。カジュアルさを残しつつも、バグを減らすための仕組みを使いましょう。

第5章では50行以内でできるお手軽なソースコードの最適化を実践しています。さらに50行ほどを追加してさらなる最適化についても書いています。

1.1.5　ソースコードの難読化

静的解析や最適化の技術を応用したものですが、Webサーバーから JavaScriptのソースコードを取得して動かすというWebブラウザの仕組み上、ソースコードそのままを配信したくないこともあります。ASTによりソースコードを読みづらくする難読化や、配信するファイルサイズを減らす為の軽量化などが可能です。

1.1.6　生産性の圧倒的向上

エンジニアの3大美徳の1つに「怠惰」というものがあります。コンピュータにできることは頑張って人間がしてはいけない。自動化して自分自身は楽をしようというものです。

Railsで有名になったDRY（Don't Repeat Yourself）という言葉があります。人間が同じものを何回も書くと、大なり小なりあれど必ず生産性を阻害します。繰り返しは人間が手作業でするものではありません。自動化することの大切さをDRY思想では説いています。人力で頑張って運用するのはエンジニアとしてはとても恥ずべきことです。

できるエンジニアの生産性は1桁も2桁も違うという言葉もありますが、ASTをうまく使いこなせば動作速度やソースコード管理のしやすさを犠牲にせずにそれが可能です。

1.https://babeljs.io/
2.https://babeljs.io/docs/plugins/preset-env/

AST の使い方を Babel 系パーサの Babylon を使って紹介

第2章　AST解説

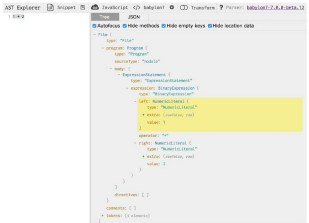
この章ではAST[1]について解説します。前半部分ではASTの取得と使い方の説明、後半部分では実際にASTを使った実践をそれぞれ説明します。

2.1　ASTを実際に眺めてみよう

ASTを実際に眺めるにはAST Explorerhttp://astexplorer.net/を使いましょう。babylon以外のASTパーサーや他の言語にも色々対応しています。AST関連の開発ではこれを使うのが定番です。

babylon7を選んで、`1 + 2`を読み込んでみましょう。

図2.1: AST explorer



ASTはノード[2]が集まって出来あがったツリーです。図2.2は先程の`1 + 2`を表しています。

図2.2: ASTのツリー構造

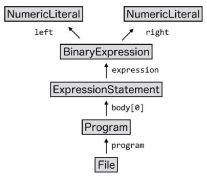


図2.2はツリーというには貧弱ですが、なんとなく木っぽいなと思ってください。ツリーの根本にあるのは`File`というノードです。根本から1つ上の幹として`Program`があります。この2つは、いってみればコード全体を指すものです。

JavaScriptのソースコードは、何らかのStatementの集まりです。Statementはプログラミング言語の世界では「文」と呼ばれます。関数呼び出しや変数の書き換え、クラス定義なども文です。JavaScriptの場合は上から順に文を実行[3]します。`Program`の`body`はまさにその`Statement`の配列で枝分かれしていきますが、今回は`body`の中身が1つです。

`ExpressionStatement`はこのプログラム唯一の文です。式そのものが文になる場合のプレースホルダーのようなものです。`ExpressionStatement`の中身は`BinaryExpression`、つまり二項演算子[4]です。

`BinaryExpression`には3つの重要なプロパティがあります。演算子を指す`operator`で今回の場合`'+'`という足し算を指す文字列が入っています。残りは`left`と`right`で`operator`の左と右です。今回は`1 + 2`という式なので、`left`には`1`を指すリテラルである`NumericLiteral`が入っています。リテラルというのはプログラミングの世界では、ソースコード上に直接書かれる数値や文字列のことです。

`NumericLiteral`はそこで完結していてそれ以上先を辿れない先端、つまり行き止まりです。

2.1.1　JavaScriptにおけるASTとは

JavaScriptにおいてASTエコシステムの歴史はMozilla Firefoxから始まります。Firefoxのリフレクション[5]用にParser APIが生み出されたのですが、このAPIが返すオブジェクトを標

1.Abstract Syntax Treeの略称で日本語では抽象構文木といいます。
2.ノードとは節（ふし）です。プログラミングの世界ではノードというと根本のルートノードや、葉っぱのリーフノードや、その間にあるものもすべてノードとして扱います。
3.関数宣言やクラス宣言は、登録するという命令を実行しています。中身は呼び出されるまで実行されません。
4.a + bとか1 * 2とかhoge == fugaのような、「左」「演算子」「右」の構造のものを二項演算子といいます。
5.プログラム が自分自身の情報を読み取ったり書き換えたりすることです
6.https://developer.mozilla.org/en-US/docs/Mozilla/Projects/SpiderMonkey/Parser_API

Babel プラグインの作り方と実際の作例を紹介

4.3　プラグインオプションの取得方法

プラグインのオプションはPluginPass型のoptsに含まれているので、メインのビジター関数であれば、`state.opts`で取得できます。

リスト4.8: chapter3/state-opts.js

```
1: const {transform} = require('@babel/core')
2:
3: const plugin = babel => {
4:   return {
5:     visitor: {
6:       Program: (nodePath, state) => {
7:         console.log(state.opts) // --> { hoge: 'hoge' }
8:       },
9:     },
10:  }
11: }
12:
13: const plugins = [
14:   [plugin, {hoge: 'hoge'}]
15: ]
16:
17: transform('1 + 2', {plugins})
```

babelプラグインのオプションは少し特殊な指定方法をしています。配列の1つめにプラグイン、2つめのオプションを渡すのです。。

4.4　BabelプラグインとしてInjectorプラグインを作ってみる

Babelプラグインの作り方を説明してきました。それでは実際にプラグインを作ってみましょう。

今回作るInjectorプラグインはDIを実現するもので、指定したソースコードに手を加えず、中の変数宣言・関数宣言・クラス宣言を置換したり、コードの最後に新しいコードを追加できるというものです。テストやデバッグに便利です。

Injectorプラグインでは、まさにASTという題材をお手軽に楽しめると思います。

4.4.1　DI（Dependency Injection）

Dependency Injectionを日本語で表現すると依存性注入ですが、依存性というのは、実際に

1 他の言語であればタプルというデータ構造の用意されていたりするのですが……。

は何らかのオブジェクトやプリミティブ値や関数です。対象を動かすときに依存する物だから依存性です。

たとえばNode.jsでファイルを読み込むには、`const fs = require('fs')`でfsのモジュールを読み込んで、`fs.readFile('hoge.js')`のようにfsモジュールのファイルを読み込む関数を叩くのが一般的です。

このとき`const fs = require('dummy-fs')`というように、実際のファイルI/Oを発生させないダミー（モック）を流し込めれば、対象のソースコードを変更しなくてもユニットテストしやすくなります。これを動的書き換えやASTを使わずに行うとどうなるでしょうか？たとえばクラスのコンストラクタにfsを渡すようにする仕組みなどがよくあるパターンですね。DIはもともとデザインパターンの一種です。

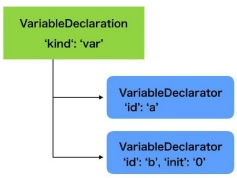
JavaScriptにはASTというとても強い味方がいるので、スマートにDIを実現できます。

4.4.2　変数定義を置換してみる

まずは変数定義を置き換えてみます。

変数定義はVariableDeclarationとVariableDeclaratorの二段構成です。`var a, b = 0`のような定義をした場合、1つのVariableDeclaration（`kind`が`'var'`）の下にaをIDとしてもつVariableDeclaratorとbをIDとしてもつVariableDeclaratorがそれぞれぶら下がります。

図4.1: VariableDeclarationとVariableDeclarator



VariableDeclaratorのinitには変数の初期化のNodeが入っているので、ここを置換すれば変数定義を変更できます。

リスト4.9: chapter3/replace-variable-init.js

```
1: const {transform} = require('@babel/core')
2: const {parseExpression} = require('babylon')
3:
```


＜＜目次＞＞

4.7 Babel プラグインの自動テスト
4.8 require hack
第 5 章 最適化プラグインを簡単に作ってみよう
5.1 超お手軽実装編
5.2 変数の静的解析情報を使って、もう少しがんばってみる

＜＜著者紹介＞＞

佐々木 俊介
高校生のときにパソ通にハマリ、その後紆余曲折を経てテキストエディタや MSX エミュレータその他を開発。技術者として勤務した後、現在はフリーで Javascript 関連のプログラマー。著書に『最新 JavaScript 開発～ES2017 対応モダンプログラミング』(インプレス R&D)。

＜＜販売ストア＞＞

電子書籍:
Amazon Kindle ストア、楽天 kobo イーブックストア、Apple iBookstore、紀伊國屋書店 Kinoppy、Google Play Store、honto 電子書籍ストア、Sony Reader Store、BookLive!、BOOK☆WALKER
印刷書籍:
Amazon.co.jp、三省堂書店オンデマンド、honto ネットストア、楽天ブックス
※ 各ストアでの販売は準備が整いしだい開始されます。
※ 全国の一般書店からもご注文いただけます。

【株式会社インプレス R&D】 https://nextpublishing.jp/
株式会社インプレス R&D（本社：東京都千代田区、代表取締役社長：井芹昌信）は、デジタルファーストの次世代型電子出版プラットフォーム「NextPublishing」を運営する企業です。また自らも、NextPublishing を使った「インターネット白書」の出版など IT 関連メディア事業を展開しています。
※NextPublishing は、インプレス R&D が開発した電子出版プラットフォーム(またはメソッド)の名称です。電子書籍と印刷書籍の同時制作、プリント・オンデマンド(POD)による品切れ解消などの伝統的出版の課題を解決しています。これにより、伝統的出版では経済的に困難な多品種少部数の出版を可能にし、優秀な個人や組織が持つ多様な知の流通を目指しています。

【インプレスグループ】 https://www.impressholdings.com/
株式会社インプレスホールディングス(本社:東京都千代田区、代表取締役:唐島夏生、証券コード:東証 1 部 9479)を持株会社とするメディアグループ。「IT」「音楽」「デザイン」「山岳・自然」「モバイルサービス」を主要テーマに専門性の高いコンテンツ＋サービスを提供するメディア事業を展開しています。

【お問い合わせ先】

株式会社インプレス R&D NextPublishing センター
〒101-0051 東京都千代田区神田神保町 1-105
TEL 03-6837-4820
電子メール: np-info@impress.co.jp