プログラミング実験第三

TinyJavaScript コンパイラの作り直し

1311216
Rathore Amogh
岩崎研究室

Contents

1 はじめに

1.1 背景

TinyJavaScript は JavaScript の一部機能を制限んしたサブセットのことである [1]。TinyJavaScript の元のコンパイラは Mozilla の SpiderMonkey Parser API [2] を使用していた。しかし、SpiderMonkey Parser は絶えてしまって、TinyJavaScript のコンパイラの開発も続けられなくなった。だから、TinyJavaScript コンパイラを新しいパーサを使用して作りなおす必要が出てきた。このレポートは TinyJavaScript コンパイラを Node JS で作る実験について述べる。

1.2 実験の目的

この実験では TinyJavaScript のコンパイラを NodeJS と Esprima (ECMAScript Parsing Infrastructure for Multipurpose Analysis) [3] を使用して実装する。それで、コンパイラの作り方と構造について学習する。最後に、実験で作った新しいコンパイラと TinyJavaScript の [1] で述べてるコンパイラの比較を行う。

1.3 実装の方針

実装には Esprima という ECMAScript[6] Parser を使用する。JavaScript は ECMAScript の dialect であるので、ECMAScript のパーサは JavaScript の構文に対して正しくパーシングをされる。参考のため [1] で述べてる TinyJavaScript Compiler の実装を使う。もう少し具体的に言うと、Esprima を使って TinyJavaScript のソースコードを抽象構文木に変換する。それから、抽象構文木をトラバースして SSJSVM(これについて後で述べる) の命令列を生成する。

2 TinyJavaScript と SSJSVM

2.1 TinyJavaScript

TinyJavaScript は JavaScript の以下の機能をサポートしない [1]

- with 文
- delete 文
- グローバル変数宣言時の var の省略
- for in 文
- switch 文
- 名前付きの関数定義

2.2 SSJSVM

SSJSVM は Server-side JavaScript Virtual Machine の略称である。すなわち、サーバで動く JavaScript の仮想機械のことである。本実験では、TinyJavaScript のソースコードを SSJSVM の命令列に変換するコンパイラを作る。

3 パーサの選び方

実験の目的は TinyJavaScript のコンパイラを新しいパーサ (SpiderMonkey 1.6 でないパーサ) を使用して作るということである。だから、プログラミングを始める前にパーサを選ぶ必要がある。以下の特徴を持つパーサを使いたい。

1. ECMAScript の新しいバージョンをサポートする
2. 構文解析を正しくやってくれる
3. 使いやすい
4. ドキュメントとサポートがいい

いろいろ調べた結果、２つの案が出てきた。

- ECMAScript の文法を揃えて ANTLR[7] (ANother Tool for Language Recognition) を使ってパーサを生成する
- Esprima を使う

ANTLR を使用すると ANTLR 用の文法が必要となる。その文法はいろいろなソースがインターネットで提供してるが、オフィシャルではないので正しさは保証できない。あとドキュメントなども全然提供されてなくて、使いにくいと判断した。ところで、Esprima はとてもアクチブなプロジェクトであって、投稿者が多い。だから、最後に Esprima を使うと決める。

4 Esprima

Esprima [3] は JavaScript で書かれてる ECMAScript のパーシングインフラストラクチャである。Esprima の主な特徴は以下のとおりである [3]。

- ECMAScript 6 (ECMA-262 [6]) 全体をサポート
- Estree プロジェクトの標準を対応する構文木フォーマット
- よくテストされたパーサ [4]
- オープンソース [5]

5 コンパイラの設計

本実験で作るコンパイラは、ソースコードを 3 つの段階で仮想機械の命令列に変換する。その段階は以下の通りである。

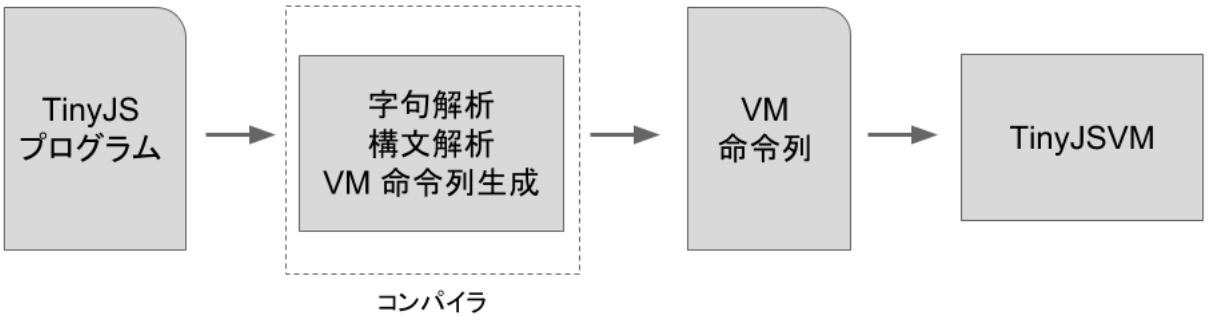

Figure 1: コンパイルの 3 つの段階

1. ソースコードの字句解析を行う
2. 字句列から抽象構文木を生成する
3. 抽象構文木を仮想機械の命令列に変換する

本実験で使う Esprima パーサは段階 1 と段階 2 の処理を行ってくれる。Esprima は抽象構文木を JS オブジェクトにして返す。

6 コンパイラの実装

7 評価

8 終わりに

References

[1] 高田 祥. *ARM* 上で動作する *JavaScript* 処理系の実装. 電気通信大学 電気通信学部情報工学科 ソフトウェア学講座. January, 2011.

[2] SpiderMonkey 1.6
http://www-archive.mozilla.org/js/spidermonkey/release-notes/

[3] Esprima
http://esprima.org/

[4] Esprima のテスト情報
http://esprima.org/test/ci.html

[5] Esprima のソースコード
https://github.com/jquery/esprima

[6] ECMAScript
http://www.ecma-international.org/publications/standards/Ecma-262.htm

[7] ANTLR
http://www.antlr.org/

[8] EStree プロジェクト
https://github.com/estree/estree