# JavaScript の GCC フロントエンドの開発
## -JavaScript の Ahead-of-time コンパイラ-

### 1. 背景

JavaScript は Web アプリケーション記述用の言語として幅広く用いられている。近年、GMail、 Google Maps といったような JavaScript を用いた複雑な Web アプリケーション、サービスの利用が拡大してきている。そのため、Web アプリケーションを実行する JavaScript 処理系の実行性能も同時に要求されている。

JavaScript では Java 言語、C++言語といった静的型付け言語と異なり、最適化の情報源となる型情報は動的に決定するためプログラム実行前に取得することが難しい。そのため、既存の静的言語と比較しコード最適化を行うことが困難である。JavaScript 処理系では効率的な実行を達成するため、インタプリタの高速化や Just-in-time コンパイラの導入によって処理系の速度を向上させている。

多くのJavaScript処理系はWebブラウザに組み込まれ、ソースコードをプログラム実行時に取得し実行する形をとる。各JavaScript処理系ではプログラム実行中の高速化にのみ着目しており、プログラム実行前における高速化手法については検討されていない。そこで本プロジェクトではJavaScript向けAhead-of-time（以下AOT）コンパイラをGCC（GNU Compiler Collection）上に構築し、その実用性について比較調査を行い、高速化手法について検討を行う。

### 2. 目的

本プロジェクトでは JavaScript のソースコードから機械語を生成する JavaScript 向け AOT コンパイラの開発を目的とした。また、生成したコードを動作させるために必要なランタイムライブラリの提供も行う。

### 3. 開発の内容

GCC 上に JavaScript の AOT コンパイラを JavaScript の GCC フロントエンド（以下 GCCJS）という形で構築した。本ソフトウェアはコード生成を行う JavaScript フロントエンドとランタイムライブラリの 2 つで構成される。主な機能は以下のとおりである。

#### 3. 1. JavaScript フロントエンド

GCC は C、 C++言語だけでなく Java 言語など様々なプログラミング言語のコンパイラを提供しており、GCC フロントエンド上では各プログラミング言語の字句解析器、 構文解析器が構成され、それぞれのソースコードが GENERIC tree に変換処理される（図 2)。本プロジェクトでは ECMAScript を参考に lex、 yacc を用いて独自に字句解析器、構文解析器を構築した。

GCCJS はこの字句解析器、構文解析器を利用し、入力として受け取ったプログラムを順次 GENERIC Tree の形式へ変換を行っている。GCCJS の構成の詳細を図 1 に示す。

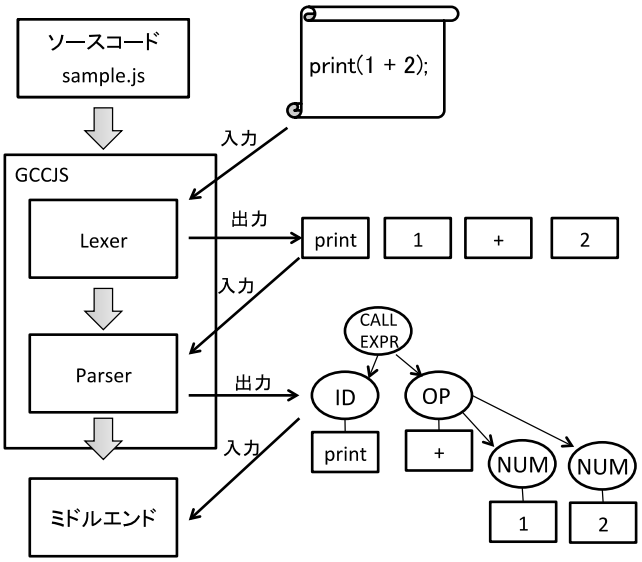


図 2. GCC の構成

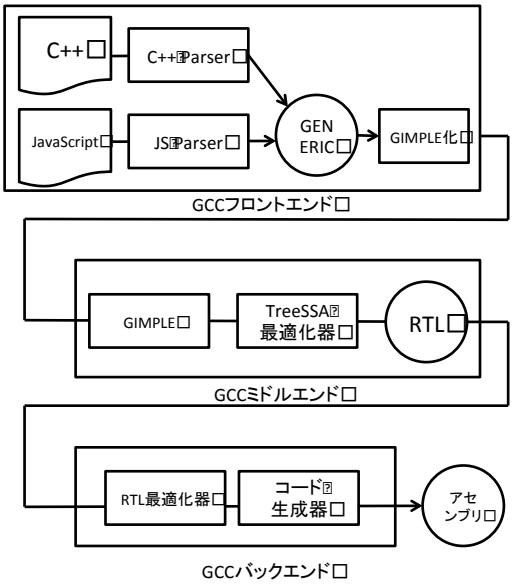


図 1. GCCJS の処理の流れ

3. 2. JavaScript ランタイムライブラリ

GCCJS ではコンパイラ部分と合わせてランタイムライブラリの設計も行った。ランタイムライブラリは以下の機能を持つライブラリである。GCCJS が生成したネイティブコードを含めたランタイムライブラリの構成を図 3 に示す。

- ライブラリ呼び出し機構（JS API）
- メモリ管理
- オブジェクト管理
- コード評価器

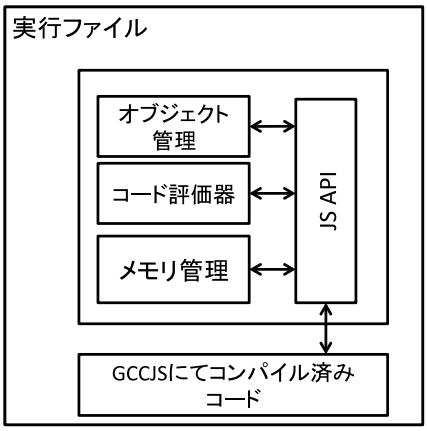


図 3. GCCJS ランタイムライブラリ

ライブラリ呼び出し機構では JavaScript からランタイムに提供される関数を呼び出す API（JS API）を提供し、関数呼び出しの補助を行っている。JS API とは C 言語などで実装されるライブラリと JavaScript コードとを結合させるためのインタフェースである。GCCJS では、JavaScript コード単体ですべての機能を記述することは困難である。また既存のライブラリの機能を利用するために、ランタイムライブラリでは JS API によって JavaScript コードとライブラリで提供される関数、機能との連携を実現している。ランタイムライブラリで提供される機能は JS API を経由してネイティブコードから呼び出される。

メモリ管理はプログラム実行中に利用されるメモリを管理する機構である。現在の実装では BohemGC を利用しメモリ管理を行っている。GCCJS では変数はすべてオブジェクトとして扱っており、数値も関数もヒープ上に確保したオブジェクトとして操作を行う。

オブジェクト管理機構では、プロトタイプやクラス構造の管理を行う。主な機能を以下に列挙する。

- プロトタイプオブジェクトの管理
- プロトタイプの継承関係管理
- プロトタイプのフィールドの管理

プログラム実行中に生成されるオブジェクトの、プロトタイプ情報からフィールドへのアクセス方法の変更を動的に行うなど、高速化のための機構となっている。

### 3. 4. 性能評価

既存の JS 処理系である Rhino、 V8 との性能比較を、マイクロベンチマークを用いた関数呼び出し、数値演算の実行速度で評価した（図 4)。なお、実験環境としては以下の環境で測定を行い、また評価は評価対象プログラムの実行時間を 3 回計測し、もっとも実行時間の短いものを計測結果とした。

- CPU Intel(R) Core(TM)i7 2.20GHz
- メモリ 8GB
- OS MacOSX10.7
- C コンパイラ GCC4.6.0
- Rhino 1.7
- V8 3.5.3

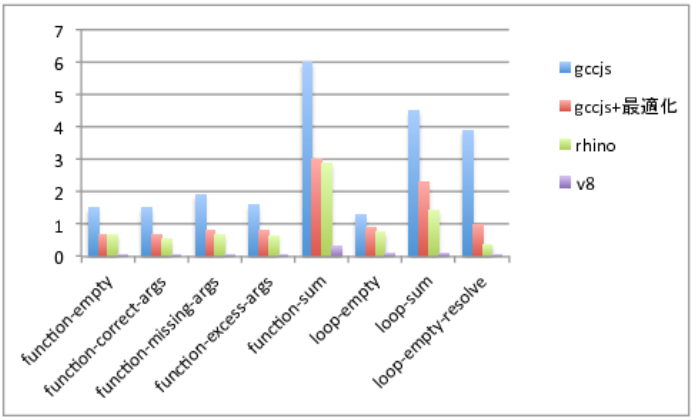


**図 4. ベンチマーク結果**

ベンチマークでは既存の処理系と比較し、全体的に性能が低下していることが確認されたが、最適化を有効にした際には平均して約 2 倍の性能向上が確認できた。近年のコンピュータアーキテクチャ上では AOT コンパイラを用いた場合でも、最適化によっては十分に性能が発揮できる可能性があるのではないかと考えられる。

## 4. 従来の技術（または機能）との相違

JavaScript 言語の JIT コンパイラについては Web ブラウザで利用されている JavaScript JIT コンパイラにて実装されている。既存の JavaScript JIT コンパイラは Web ブラウザとの連携を考えて実装されているため、JavaScript を利用するには Web ブラウザを搭載する必要がある。

一方で GCCJS では実行前にコードのコンパイルを行うため、ささいな記述ミスによる実行時エラーを抑制でき、またコンパイルされたコード単体でアプリケーションの動作が可能となる。

5. 期待される効果

本ソフトウェアを利用することによって、スマートフォンなど組み込み機器のネイティブアプリケーション開発をランタイムライブラリと組み合わせることで JavaScript のみで開発することが実現することが可能となる。また、静的解析による高速化手法を検討するための基盤ソフトウェアとして利用することが可能となる。

6. 普及（または活用）の見通し

現在、ソースコードを GPL3 のライセンスにてオープンソース公開を行っている。安定版の公開後は成果物を GCC コミュニティにレビューを求め、成果物の統合を目指し活動を行う予定である。

7. クリエータ名（所属）

井出真広（横浜国立大学工学府）

（参考）関連 URL

本プロジェクトにおける成果物は以下の URL にてオープンソースソフトウェアとして公開している。

http://github.com/imashairo/gccjs/