組み込みシステム向け JavaScript 仮想機械の文字列オブジェクトの実装戦略

1255112　近森凪沙　【 ソフトウェア検証・解析学研究室 】

Implementation Strategies of String Objects in JavaScript Virtual Machines for Embedded Systems

1255112　CHIKAMORI, Nagisa　【 Software Verification and Analysis Lab. 】

1　はじめに

近年，IoT（Internet of Things）技術の普及により，組み込みシステムの利用が広まっている．我々が開発を進めている JavaScript 処理系 eJS[1] は，JavaScript で組み込みシステムのアプリケーションが開発を行えるようにすることを目指しており，組み込みシステムで実行可能な仮想機械である eJSVM を提供している．組み込みシステムには，フラッシュメモリと RAM の両方を搭載しているものがあるが，そのようなシステムでは RAM の容量が小さい一方，フラッシュメモリは書き換え不可なものが多い．そのため，大きいプログラムを実行するためには RAM を効率的に使用する必要がある．

eJSVM では，同じ文字列を一つにまとめて共有するインターンを行っており，まとめた文字列はハッシュ表で管理している．文字列をインターンすることで，同じ文字列に割り当てるメモリが削減され，文字列の比較はアドレスで比較するため高速化される．しかし，この方式はハッシュ表などが RAM を消費する．eJSVM では，ハッシュ表の他に JavaScript オブジェクトを配置するヒープ（以下，JS ヒープ）など様々なデータを RAM に配置しており，これらが RAM を圧迫している．そこで，本研究では利用可能な RAM 領域を増やすため，ハッシュ表を用いない文字列管理方式を提案する．また，文字列の比較回数が最も多いオブジェクトのプロパティ検索を特に高速化するために，コンパイル時の解析により比較方法を決定する方法と，キャッシュに保存されたプロパティかどうか判定する際には文字列のアドレスによる比較を行う方法の，2 種類の高速化法を考案し，eJSVM に実装した．

2　ハイブリッド文字列管理方式

eJS では，プロパティ名など JavaScript プログラム実行前に生成される文字列（以下，静的文字列）は，コンパイル時にインターンしてフラッシュメモリに配置している．そのため，静的文字列同士の比較だけでもアドレスによる比較ができれば，比較を一部高速化することができると考える．

そこで，+演算子で結合した文字列など JavaScript プログラム実行中に生成される文字列（以下，動的文字列）はインターンせず個別にメモリを割り当て，静的文字列同士の比較の時だけアドレスによる比較を行う方式を提案する．この方式をハイブリッド文字列管理方式と呼ぶ．全体像を図 1 に示す．提案方式では，プログラムの実行中に文字列の比較方法を決定する．静的文字列同士の比較の場合，文字列のアドレスによる比較を行い，動的文字列同士または動的文字列と静的文字列の比較の場合，文字列の内容による比較を行う．

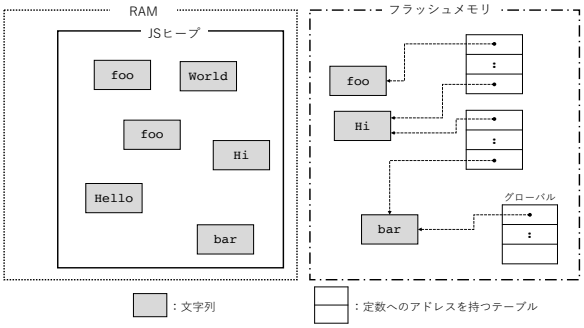

図 1　ハイブリッド文字列管理方式

ハッシュ表を用いないため，ハッシュ表が占めていた領域を他のデータが使用することができる．一方で，動的文字列同士または動的文字列と静的文字列の比較が多いと，文字列の内容による比較の増加により遅くなることが考えられる．また，文字列の比較を行う度に静的文字列同士の比較であるかどうか調べる必要があり，このことがオーバヘッドになる可能性がある．そのため，文字列の比較方法を判断する必要がない場面では，その処理を除外したい．

最も文字列の比較回数が多いオブジェクトのプロパティ検索では，キャッシュを利用することでプロパティ検索を高速化している [2]．オブジェクトのプロパティにアクセスする度にキャッシュを調べるため，キャッシュに保存されたプロパティ名との比較は頻繁に行われる．プロパティ名はほぼ静的文字列が使用されるため，実際には文字列のアドレスによる比較だけでよい場面が多々ある．そこで，キャッシュに保存されたプロパティ名と一致するかどうか調べる処理について，以下の 2 種類の高速化法を考案した．

2.1　静的な文字列比較

プロパティへのアクセスを行っている箇所は，コンパイル時の解析により知ることができる．プロパティ名に静的文字列が必ず使用されると分かれば，文字列のアドレスによる比較のみを行うだけでよい．そこで，JavaScript プログラムのコンパイル時に文字列の比較方法を決定する．この手法を静的な文字列比較と呼ぶ．常にプロパティ名に静的文字列が使用されている場合は，文字列のアドレスによる比較のみを行う．

2.2　楽観的な文字列比較

プロパティ名に動的文字列が使用されることはほとんどなく，キャッシュに保存されたプロパティ名と一致するかどうかを調べる処理では，静的文字列同士の比較が多くなる．静的文字列は既にインターンしているため，文字列のアドレスで同一の文字列かどうか判断することができる．そこで,「ほとんどの場合，プロパティ名には静的文字列が使用される」と仮定して，キャッシュに保存されているプロパティ名との比較は，常に文字列のアドレスによる比較を行う．この手法を楽観的な文字列比較と呼ぶ．プロパティ名が静的文字列であっても動的文字列であっても，キャッシュに保存されているプロパティ名との比較では，文字列のアドレスによる比較を行うようになる．

3　評価

本研究では，Mbed[3] 対応マイコンを用いて，既存の eJSVM と提案方式を実装した eJSVM の比較を行う．また，実行時間の比較ではこれらに加え，静的な文字列比較と楽観的な文字列比較をそれぞれ提案方式に実装した eJSVM の実行時間も比較する．

まず，文字列を管理するハッシュ表が無くなったことで RAM や JS ヒープの空き領域が増加したか確認するため，eJSVM 起動直後の RAM と JS ヒープを調査した．その結果，RAM の空き領域は既存の eJSVM では最大で 19.5 KiB だったが，提案方式では 39.5 KiB に増加した．これは，ハッシュ表が使用していた 20 KiB が無くなったことによる効果である．また，JS ヒープの空き領域は既存の eJSVM では最大で 121,496 B だったのに対し，提案方式では 123,884 B と 2.4 KiB の増加が確認された．その他のベンチマークでも，約 1.1〜2.2 KiB の増加が確認された．

次に，実行時間の比較を行った．結果を図 2 に示す．ベンチマークごとに，左から既存の eJSVM，提案方式，静的な文字列比較を実装した提案方式，楽観的な文字列比較を実装した提案方式の実行時間を，既存の eJSVM を 1 とする比で表している．既存の eJSVM と比べて，通常の提案方式では極端に傾向が異なるベンチマーク b10 と人工的なプログラムである b20，b21，b22 を除いて実行時間が平均で 6 %増加した．また，静的な文字列比較を実装した場合は既存の eJSVM と比べて実行時間が平均で 2 %減少し，楽観的な文字列比較を実装した場合は既存の eJSVM と同程度だった．通常の提案方式と比べると，一部を除いて実行時間は短くなっており，既存の eJSVM とほぼ同じ実行時間となった．これは，文字列の比較を判断する処理を除外したことで，実行時間を短くすることができたからであると考える．一方，ベンチマーク b21 は，既存の eJSVM と比べて実行時間が 80 %以上増加した．このプログラムは，プロパティ名に動的文字列を使用する人工的なプログラムである．このような特殊なプログラムの場合は，高速化法は効果があまりないことがわかった．しかし，プロパティ名に動的な文字列が使用されることはほぼないため，全体的に見れば高速化法を実装した提案方式は，既存の eJSVM と変わらない実行時間になると考える．

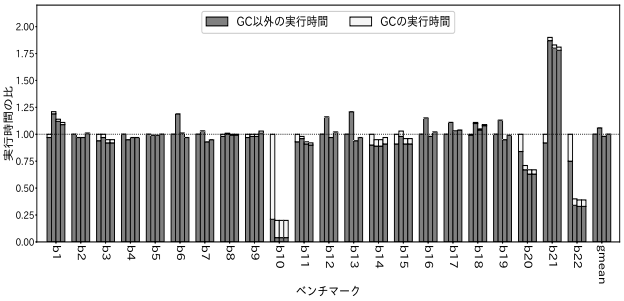

図 2　実行時間の比

4　まとめ

本研究では，利用可能な RAM 領域を増やすため，ハッシュ表を用いない文字列管理方式を提案した．また，最も比較回数が多いプロパティ検索における文字列比較を高速化するため，文字列の比較方法を判断する処理を除外する方法を考案した．コンパイル時に文字列の比較方法を決定する静的な文字列比較と，プロパティ名に静的文字列が使用されると仮定してキャッシュされたプロパティ名と比較する際に文字列のアドレスによる比較を行う楽観的な文字列比較を提案方式に実装し，高速化を図った．提案方式を eJSVM に実装した結果，RAM や JS ヒープの利用可能領域が増加した．また，静的な文字列比較と楽観的な文字列比較をそれぞれ実装した提案方式では，既存の eJSVM とほぼ変わらない実行時間になった．

参考文献

[1] T. Kataoka, T. Ugawa, and H. Iwasaki. A Framework for Constructing JavaScript Virtual Machines with Customized Datatype Representations. SAC'18, pages 1238–1247, 2018.

[2] L. Peter Deutsch, Allan M. Schiffman. Efficient Implementation of the Smalltalk-80 System, POPL'84, pages 297–302, 1984.

[3] Mbed. https://os.mbed.com/, 2020.