組み込み Java 起動高速化の検討と評価

竹内俊策† 岡田英明‡ 高橋克英‡ 前田慎司‡ 橘高大造‡
三菱電機株式会社 モバイルターミナル製作所† 情報技術総合研究所‡

【1】はじめに

現在、携帯端末に Java が組み込まれ、様々なサービスが提供されている。主要なサービスとしてゲーム等のエンタテーメント系のアプリケーションがあり、我々は使用環境の向上策として JIT 動的コンパイラを適用した実行速度の高速化を検討してきた[1]。しかし、近年ビジネス系のアプリケーションが注目されてきている。ビジネス系のアプリケーションには特に即座に使用できる環境が必須であるが、現在、Java アプリケーションの起動には 2～5 秒程度の時間がかかっているためユーザが不便さを感じてしまう。Java 以外の電卓、スケジューラなどの組み込みアプリケーションは 0.5 秒以内で起動しており、Java アプリケーションに対してもこれらと同等の起動性能が求められている。

そこで本稿では、起動処理過程においてボトルネックとなっている処理を明らかにし、その高速化手法を検討、試作及びその評価を行ったので報告する。

【2】現状の JavaVM 起動方法とその性能

現状の JavaVM の起動処理過程(図 2 参照)では各初期化処理を行った後、起動時に必要となるクラスに対してクラスロードを行う。クラスロードは、要求されたクラスのクラスファイルを読み込み、揮発メモリ上に内部データ構造を構築する。次にクラスのバイトコードを検証する。また JIT 動的コンパイラ[1]が組み込まれている場合、事前コンパイル対象とされているメソッドをコンパイルする。

上記起動処理過程の内ボトルネックとなっている処理を明らかにするため、各処理時間を測定した。ビジネス系のメールアプリ(アプリ 1)を使用した場合の結果を図 1 に示す。なお、起動開始はユーザによるアプリケーション起動要求のキー入力時とし、起動終了はアプリケーションの最初の画面が表示された時とした。図 1 中の VM 起動前処理は起動開始から VM 起動処理開始までの処理である。

総起動時間 2405msec のうちボトルネックとなっている処理は VM 起動前処理(21%)、クラスロード(26%)、検証処理(15%)、JIT 動的コンパイラに関する処理(JIT 関係初期化処理と JIT 動的コンパイル)(32%)であった。起動前処理はボトルネックとなっているが、VM における処理ではないため本稿での検討対象外とする。

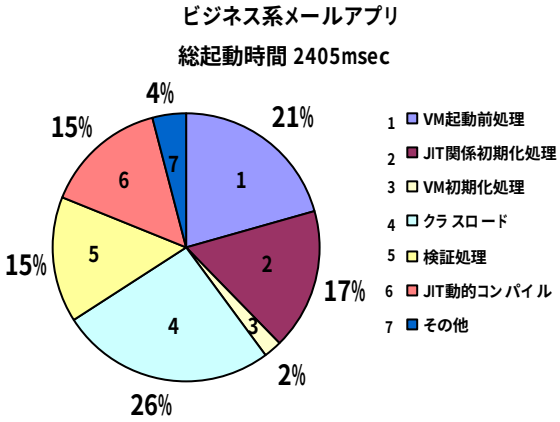

図 1 アプリケーションの起動時間とその内訳

【3】高速化手法の検討

ボトルネックであるクラスロード、検証、コンパイル処理は起動毎に毎回同じ処理を行っている。そのため揮発メモリにある処理済みのデータ構造を不揮発メモリに保存し、次回起動時に再利用することによる高速化が可能であると考えられる。コンパイル処理については、再利用する方法の他に H/W アクセラレータを利用することでコンパイル処理を行わない方法が考えられる。そのため本稿ではクラスロード、検証処理に絞って検討する。

クラスロード、検証処理を再利用するには 2 つの方法が考えられる。以下にその内容と利点、欠点を述べる。

方法 1:クラスロードを行い、検証処理を行った直後(図 2 の d 点)に揮発メモリに構築された内部データ構造を別の揮発メモリに一時的に退避し、アプリケーション終了時に一時退避しておいた内部データ構造を不揮発メモリに保存する。次回起動時に図 2 の b 点で保存しておいた内部データ構造を不揮発メモリから揮発メモリにコピーする。

Reducing Startup Time of Embedded Java
Shunsaku Takeuchi† Hideaki Okada‡ Katsuhide Takahasi‡
Shinji Maeda‡ Taizou Kittaka‡
Mitsubishi Electric Corporation
Mobile Terminal Center†
Information Technology R&D Center‡

方法 2:アプリケーション終了時(図 2 の e 点)に揮発性メモリにおいて連続した領域に格納してある内部データ構造を不揮発メモリに保存する。次回起動時の各初期化処理終了後(図 2 の a 点)に不揮発メモリに保存されているデータを揮発メモリにコピーする。

方法 1、2 ともに内部データ構造の中身にはポインタ情報が含まれているため、不揮発メモリへの保存時に格納されていた揮発メモリの先頭アドレス、内部データ構造のサイズ等の管理情報を保存し、次回起動時に使用する。

方法 1 は検証処理直後のデータ構造を保存するため、保存したデータ構造をそのまま次回起動時に使用することができる。しかし、一時退避するための揮発メモリが必要となる欠点がある。

方法 2 では揮発メモリの使用量が増加するという欠点はない。しかし検証処理終了直後とアプリケーション終了時の内部データ構造には状態差があり[2]、この状態差を解消するため保存時に以下の処理を行う。

- 内部データ構造の状態を検証済みの状態にする。
- static フィールドの領域を 0 初期化する。
- quick 命令化されたバイトコードを元に戻す。

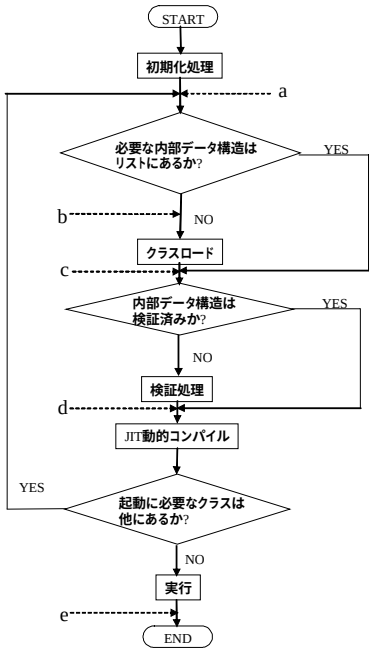

図 2　VM 起動処理フロー図

【4】試作結果

方法 1 はメモリ資源を節約したい携帯端末には適さないため、方法 2 を採用し、試作を行って起動時間の性能を評価した。評価用のアプリケーションとしてビジネス系のメールアプリケーション(アプリ 1)とゲームアプリケーション(アプリ 2)とエンタテーメント系のメールアプリケーション(アプリ 3)を用いて測定を行い、事前コンパイルをしない場合についても評価した。図 3 にその結果を示す。

JIT 処理を行う場合、アプリ 1 は 41%、アプリ 2 は 40%、アプリ 3 は 48%の高速化が達成された。また、JIT 処理を行わない場合は、アプリ 1 で 67%、アプリ 2 で 75%、アプリ 3 で 60%の高速化が達成された。

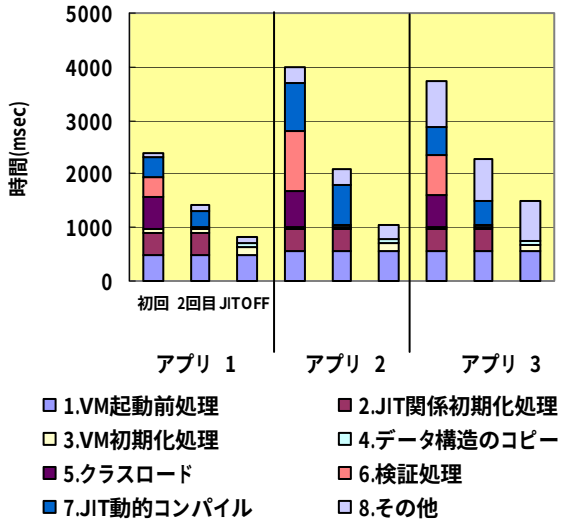

図 3　評価結果

【5】おわりに

ロードしたクラスの検証処理済みの内部データ構造を不揮発メモリに保存し、次回起動時に再利用することによる起動処理の高速化を検討し、試作及び評価を行った。JIT 処理を行わない場合、目標値には達していないもののファイル I/O の小さいビジネス系アプリであれば起動時間が 1 秒を切る性能を得られた。

今後は本稿対象外としたコンパイル処理、起動前処理の高速化について検討を行う。

【参考文献】

[1] 高橋克英 他:携帯端末向け Java 高速化手法の検討,FIT2003,pp79-80,2003

[2] Tim Lindholm,Frank Yellin:Java 仮想マシン仕様 第 2 版,ピアソン・エデュケーション,2001