3C-5

携帯端末向け Java 動的コンパイラのチューニング方法の検討

岡田　英明　　高橋　克英　　清原　良三
三菱電機（株）　情報技術総合研究所

1. はじめに

多様なハードウェアおよびソフトウェアプラットフォームが存在する携帯端末では、Java 実行環境を搭載することによりそのプラットフォーム依存性を解決し、第三者ソフトウェアの導入を容易にすることが一般的になりつつある。携帯端末上の Java 仮想マシン[1]は、当初は中間言語であるバイトコードをインタプリタで実行することが一般的であったが、バイトコードを直接ハードウェアで実行するアクセラレータや、バイトコードをネイティブコードにコンパイルするネイティブコンパイラを用いた高速化方式の採用が進んでいる。

ネイティブコンパイラによる高速化はコンパイルしたネイティブコードの格納領域を要求するため、一般にメモリ容量が限られる携帯端末では、コンパイルするメソッドを絞り込み、かつキャッシュ制御を行って当該領域を有効に利用することが必要である。また、携帯端末上での Java アプリケーションはゲーム等のユーザとの対話を必要とすることが多く、ネイティブコードへのコンパイル処理によりアプリケーションの応答性を阻害しないようにする必要がある。そこで、我々はコンパイルを行うメソッドとそのタイミングを静的および動的判断によって行う動的コンパイラを検討、評価してきた[2,3]。

動的コンパイラを適切に使用するためには、組み合わせるネイティブコンパイラやメモリ容量、クラスライブラリの特性に応じたチューニングがかかせない。本論文では、動的コンパイラのチューニング手法について検討する。

2. 動的コンパイラのチューニングパラメータ

動的コンパイラにおけるメソッドのコンパイル判断基準およびそのタイミングを簡単に記す。なお、ここでシステムクラスとは携帯端末に組み込み済み、かつアプリケーションを構築するための共通クラスであり、アプリクラスとはアプリケーション固有のクラスを指す。

- **システムクラスの静的判断による事前コンパイル**
  Java 実行環境の起動時に、頻繁に実行されると考えられるメソッドをあらかじめコンパイルしておく。
- **システムクラスの動的判断による実行時コンパイル**
  プロファイリング（一定時間内にメソッドが呼び出される回数を計測する）により、メソッドが頻繁に呼び出されたと判断した場合にコンパイルする。
  なお、システムクラスには事前コンパイル対象でも動的判断対象でもないメソッドが存在する。
- **アプリクラスの静的判断によるコンパイル**
  クラスロード時に、コンパイルすることによる効果が大きいと考えられるメソッドをコンパイルする。
- **アプリクラスの動的判断によるコンパイル**
  システムクラスと同様にプロファイリングを用いた動的判断に基づくコンパイルを行う。すべてのアプリクラスのメソッドが動的判断の対象となる。

表１に動的コンパイラのチューニングパラメータ示す。

表 1 動的コンパイラのチューニングパラメータ一覧

パラメータ
ネイティブコード増加率
コードキャッシュサイズ
静的判断対象アプリクラスのメソッドサイズ
動的判断対象アプリクラスのメソッドサイズ
コンパイル対象システムクラス

表 2 チューニングのための条件

条件	関連するパラメータ
ネイティブコンパイラ	コードキャッシュサイズ ネイティブコード増加率 静的判断対象アプリクラスのメソッドサイズ 動的判断対象アプリクラスのメソッドサイズ
CPU 速度性能	動的判断対象アプリクラスのメソッドサイズ
メモリ容量	コードキャッシュサイズ
システムクラス	コンパイル対象システムクラス コードキャッシュサイズ
アプリケーションサイズ(Sa)	コードキャッシュサイズ
ネイティブコンパイルによる実行中断許容時間(Ti)	動的判断対象アプリクラスのメソッドサイズ

１）ネイティブコード増加率

ネイティブコンパイラによりバイトコードをネイティブコードに変換したときのコードサイズの増加率である。ネイティブコンパイラが出力可能な最大のネイティブコードサイズを超えた場合、コンパイル処理が無駄になるため、無駄なコンパイルを回避するために使用する。

２）コードキャッシュサイズ

ネイティブコンパイラによりコンパイルしたネイティブコードの格納領域（キャッシュ制御あり）のサイズである。

３）静的判断対象アプリクラスのメソッドサイズ

ゲームアプリケーションなどでは、キー入力判定およびそれに対応する処理を１つのメソッドで記述することが多い。このようなメソッドの場合にはメソッドサイズが大きくなるため優先的にコンパイルする。

４）動的判断対象アプリクラスのメソッドサイズ

動的判断によりネイティブコードへのコンパイルが発生した場合、アプリケーションの実行が一時中断するとアプリケーションの応答性を阻害する。したがって、動的判断対象となるメソッドのサイズを制限する。

５）コンパイル対象システムクラス

事前コンパイルおよび動的判断対象となるメソッドを選択する。携帯端末における Java の利用はますます拡大しており、多種多様なシステムクラスが搭載されるようになっている。したがって、そのすべてのメソッドのコンパイル設定を行う工数は大きく、動的コンパイラのチューニングにおいて最も時間のかかる作業である。

従来のコンパイル対象システムクラスの設定方法（以下、基準手法と呼ぶ）を説明する。全メソッドに対して、高速化の必要性を判断し、個別にコンパイル設定を行う。さらに、上記でコンパイル対象としなかったすべてのメソッドを動的判断対象として各種性能測定およびチューニング用

Study of Tuning Java Dynamic Compiler for Hand-held Devices
Hideaki OKADA
Mitsubishi Electric Corporation Information Technology R&D Center

に作成したベンチマークを実行し、頻繁に呼ばれたメソッドを抽出する。さらに抽出されたメソッドについて、それぞれコンパイル対象の ON/OFF を切り替えて同じベンチマークを実行して比較し、効果のあったものを抽出する。

3. チューニングのための条件

表２に動的コンパイラをチューニングするための条件を示す。ネイティブコンパイラは、対象とするネイティブコードの差異により出力するネイティブコードのサイズが異なる。また、CPU 速度性能との組合せによりコンパイル時間にも影響を及ぼす。アプリケーションサイズ(Sa)は、最大または典型的なアプリケーションの Jar ファイルのサイズである。ネイティブコンパイルによる実行中断許容時間(Ti)は、アプリケーションの応答性の阻害性を表す条件である。例えば、30FPS(Frames Per Second)の描画性能を目標とし、実行中断時間を最悪でも 1 フレームの描画時間とすると、33(=1000/30)ms となる。

これらの入力条件に基づく動的コンパイラのチューニングについて、システムクラスのコンパイル対象設定以外の部分とシステムクラスのコンパイル対象に分けて述べる。

4. 動的コンパイラのチューニング

１）ネイティブコード増加率の設定

システムクラスをすべてコンパイルするなどして、使用するネイティブコンパイラがバイトコードをネイティブコードにコンパイルしたときにどのようにサイズが増加するか測定し、回帰分析により増加率（α）を決定する。

同時にそのコンパイル時間(τ)を計測し、バイトコードあたりのコンパイル時間を測定する。

２）コードキャッシュサイズの設定

コードキャッシュサイズに対する要求は、次のようになる。各種アプリケーションの Jar ファイルのサイズに対して、多くの場合その 0.7 倍から 1.4 倍のバイトコードが存在する。したがって、アプリクラスに対しては、

$$Sa \times \alpha \times 1.4$$

を要求する。システムクラスについては、５章の結果から得ることができる。上記要求に対し、ハードウェアおよび周囲のソフトウェア条件が許容するメモリ容量を使用する。

３）静的判断対象アプリクラスのメソッドサイズの設定

ネイティブコンパイラが出力できるネイティブコードサイズが制限されている場合、コンパイル可能サイズ(Sc)に対して、以下のように設定する。

$$Sc \div \alpha$$

４）動的判断対象アプリクラスのメソッドサイズの設定

１）の結果から、以下のように設定する。

$$Ti \div \tau$$

5. コンパイル対象システムクラスの設定

システムクラスのコンパイル設定について述べる。本設定を効率よく行うため、以下の手順による設定手法を提示する。

1. 速度性能を要求しないためにコンパイルが必要でないと考えられるパッケージおよびクラスのメソッドをコンパイル対象外とする。
2. 速度性能および汎用性(java.lang パッケージ等)から原則的にコンパイルするパッケージまたはクラスのメソッドをすべてコンパイル対象とする。ただし、例外クラスのメソッドやコンストラクタは対象外とする。
3. 残ったクラスのうち、アプリケーションに公開されている API であるものについて、そのメソッド個別にコンパイル設定を行う。
4. コンパイル対象漏れ防止のため、上記手順までにコンパイル対象とされたメソッドが呼び出すメソッドを、一旦コンパイル対象外と判断したものも含めてすべてコンパイル対象とする。例外クラスのメソッドは除く。

6. 評価

提示したシステムクラスのコンパイル設定手法について評価する。今回、評価に用いたシステムクラスライブラリのボリュームを表３に示す。

提示した手法に基づき、まず 15 のパッケージをコンパイル対象外とし、118 のクラスを同じく対象外とし、46 のクラスを原則的にコンパイルするクラスとした。この時点で残った 574 メソッドについて個別に設定した。結果として設定されたコンパイル対象を表４に示す。

今回提示した手法では基準手法による結果と比べて、表５に示す差異が生じた。コンパイル対象設定漏れの原因の多くは、コンストラクタをコンパイル対象外としたことであった。また、コンパイル対象が多すぎた原因は、クラス単位でコンパイル対象としたが、一部メソッドは基準手法ではコンパイル不要としたものであった。

表 3 評価対象システムクラス

項目	値
パッケージ数	36
クラス数	372
メソッド数	4119

表４　コンパイル対象メソッド数

項目	値
パッケージ数	13
クラス数	76
メソッド数	632

表５　提示手法と基準手法の結果の違い

項目	値
コンパイル対象設定漏れメソッド	19
コンパイル対象超過メソッド	7

7. おわりに

携帯端末向けの Java 動的コンパイラにおけるチューニング手法を検討した。特に、今後増大が見込まれるシステムクラスライブラリに対するチューニングについて、チューニング工数を軽減するための手法を提示した。本手法と工数の多い基準手法による結果を比較し、本手法がある程度妥当なチューニング結果を導くことを確認した。

しかし、上記設定手法の比較は１つのシステムクラスライブラリでのみ評価したものであり、妥当性を十分示せていない。今後は、提示手法による設定の妥当性の評価方法および提示手法による結果を近似値とした最適化手法について検討を進める予定である。

[1]Lindholm, The Java Virtual Machine Specification, 2nd Edition, Sun Microsystems, Inc.
[2]高橋他,携帯端末 Java の高速化手法の検討と評価,情報科学技術フォーラム 2003,M-020,Sep.2003
[3]高橋他,携帯端末 Java の高速化手法の検討と評価,情報科学技術フォーラム 2003,M-020,Sep.2003
[4]高橋他,携帯端末 Java アプリケーションに特化した高速化手法,第 67 回情報処理学会全国大会