Javaプログラムの実行負荷を解析して表示するシステムの試作

寒川 明好† 岩澤 京子‡

拓殖大学大学院 工学研究科 電子情報工学専攻† 拓殖大学 情報工学科‡

1. はじめに

Java プログラムを最適化やチューニングするとき，負荷が大きい箇所を中心におこなう．その負荷を確認するプロファイラ[2]の多くは，実行時間での計測をおこなっているが，その結果は使用している環境に左右される．そのため，正確な負荷の場所を確認しにくい．そこで動的バイトコード数などを明確にするシステムを開発した．本システムでは，基本ブロックごとの実行回数と静的なバイトコード数を融合した．

本研究の目的は，そのプロファイル結果からユーザに，自身のプログラムの実行負荷をバイトコード数で表示することである．このことによりユーザは，プログラムのチューニングや実行有無を知ることができる．また，負荷の場所を明確にすることで，最適化を容易におこなうことができるようになる．

この報告では，開発したシステムの機能と構成について記し，出力例を示す．

2. システムの機能と構成

図 1 に開発したシステムの構成を示し，次に処理の流れにしたがって(1)～(5)に概要を示す．

入力データは，Java ファイルとコンパイルした Class ファイルとする．

(1) クラスファイルを解析する[3]．バイトコードをフローグラフ化し，静的データを収集する．
(2) 基本ブロックごとにカウンタ命令を，メソッドごとにカウンタ値の出力命令を挿入する．
(3) 上記バイトコードを実行して，動的データを収集する．
(4) 上記で収集した静的データと動的データから，プロファイルを作成する．
(5) 上記の情報を元に，ユーザに分かりやすいようにまとめて出力する．

3. クラスファイル解析

入力した Class ファイルから，図 1-(1)のクラスファイル解析により，次の静的データを収集する．

(1) クラス数とクラス名
(2) メソッド数とメソッド名
(3) メソッドごとの基本ブロック数と，その静的バイトコード数
(4) ループ条件がある基本ブロック番号
(5) ソースコードとバイトコードを対応させた行番号

上記の情報を収集するには，クラスファイル内の実行処理の部分にあたるバイトコードを取り出して，フローグラフを作成する．

4. カウンタ命令の設置

実行回数を取得するために，3 章のフローグラフに次のコードを挿入する．

(1) メソッドの先頭にカウンタをゼロにして設置する．
(2) 基本ブロックごとにカウンタをインクリメントする命令を挿入する．
(3) 全ての return の直前にカウンタの値をファイルに出力する命令を挿入する．

出力するファイル名は，“クラス名＋メソッド名＋引数の型”で作成する．ファイルはメソッド数だけ作成され，データを次のように格納していく．

(1) 基本ブロック番号順に実行回数を格納していき，最後に改行をおこなう．
(2) 同一のメソッドが実行されたときは，前のファイルに追加していくように格納する．

メソッドの実行数は，上記ファイルの行数から取得する．

5. 静的データと動的データの融合

3 章で収集した静的データと，4 章で出力した実行回数を融合して，次の 3 つの情報を取得する．

(1) クラスやメソッドごとの動的バイトコード数
(2) 入力された Java プログラムの行ごとに対する実行回数
(3) ループ繰返し回数

ループ繰返し回数は，3 章で取得したループ条件がある基本ブロックと実行回数から求める．

6. 出力結果

5 章のプロファイル結果を元に，図 2 に本システムで表示した結果のスクリーンショットを示す．

Prototyping of Load Ratio Display System for Java Programs.
† Akiyoshi Samukawa, Graduate School of Electronic Information Engineering, Takushoku University.
‡ Kyoko Iwasawa, Information Engineering, Takushoku University.

図 2-(a)には，クラス名，動的バイトコード数，とその比率の円グラフ，メソッド数を表示する．

選択されたクラス名を図 2-(b)に，クラスのメソッド情報を図 2-(c)に表示する．

図 2-(c)には，メソッド名，動的バイトコード数その比率の円グラフ，実行数を表示する．

選択されたメソッド名と詳細情報，ユーザソースコードを図 2-(d)に表示する．

図 2-(d)には，選択された実行番号の基本ブロック番号，ユーザソースコードの行番号，静的バイトコード数，実行数，動的バイトコード数，ループ繰返し回数と，ユーザソースコードのメソッドの部分を表示する．またユーザソースコードは，処理がおこなわれている行ごとに実行回数を表示する．

選択されたループ条件があるソースコード行の色を変更して表示する．

7. おわりに

この報告では，開発したシステムの構成と出力例を示した．このシステムの特徴は，実行回数と静的なバイトコード数を融合して，動的負荷を測定し表示したことである．

本システムを用いることで，ユーザは自身のプログラムの実行負荷を知ることができる．また，処理の実行有無をソースコードと一緒に表示することで，容易にプログラムのチューニングや動作確認をおこなうことができる．そして，ループ繰返し回数を知ることで，ループ展開などに有効活用することができる．

今後は，サマリーの出力や，バイトコード上でループ展開による最適化をおこなうようにシステムを拡張していく．

参考文献

[1] David Eng. *Combining Static and Dynamic Data in Code Visualization.* Sable Technical Report 2002-4, McGill University, 2002-6.

[2] SourceForge.net Eclipse profiler, http://sourceforge.net/projects/eclipsecolorer/, 2008-9.

[3] Java Practice クラス解析, http://www2.mitte.nir.jp/~fuk-js/etc/class_dump.html 2008-9.

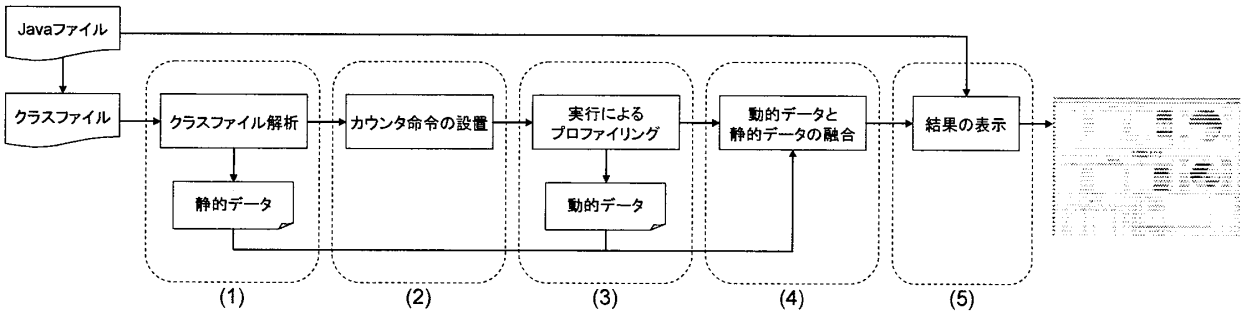

図 1　システムの全体構成

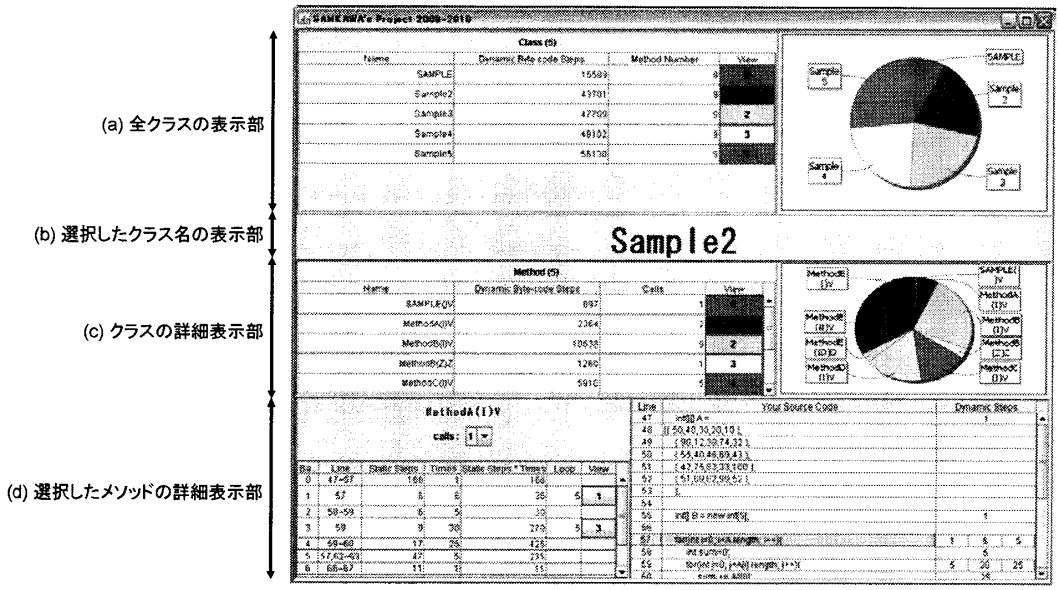

図 2　Java プログラムの実行負荷を解析して表示するシステムのスクリーンショット