Java マルチスレッドプログラムのためのアニメーションツール

山岡 裕司 寺田 実
東京大学 工学部 機械情報工学科

1 はじめに

応答性や効率の向上が望める等、プログラムの並行化によるメリットが大きいことはよく知られている。しかし、一般に並行プログラムの有効なデバッギング手法はなく、そのデバッグは非常に困難であることに変わりはない。

本研究では並行プログラムのデバッグの困難さの最大の原因が各プロセスの処理状況を把握することの大変さにあると考え、視覚化によるその大変さの軽減によってデバッグを有効に支援できると考えた。

2 目的

Java 言語によるマルチスレッドプログラムのデバッギングツールを開発し、その有効性を示す。

プログラムの実行点に対応したソースコード上での実行点を連続的で滑らかに移動させる。また、実行ログをとることによりプログラムをバグが生じた時と同様の実行順序で実行させる。これらにより、実行環境へのアクセスが可能なアニメーションデバッガを開発する。

対象言語を Java 言語としたのは、言語仕様として並行性を扱う機能を備えており、入門用の言語としても広く利用されているので、並行プログラムデバッガの需要が比較的高いと思われるからである。

3 視覚化

１スレッドに１ウィンドウを対応させ、それぞれのウィンドウにおいてソースコードを電子紙芝居 [1] として表示する。

デバッギの実行点の変化に伴って、対応するスレッドの対応するソースコード行がハイライト表示される。つまり対応するスレッドのウィンドウ内に表示されたソースコードの、対応する行が強調表示される。新たな関数が呼ばれたときは、そのウィンドウ内に紙芝居の一枚のシートにみたてたその関数部分のソースコードをアニメーションでスライドさせながら挿入する。また、はるか上方あるいは下方でウィンドウ外にある場所のソースコード行を強調すべきときには、ソースコード自体を上下にアニメーションでスライドする。これらの方法により実行点の移動の連続性を保つ。

4 再現性の考慮

並行プログラムにおけるバグには、再現性のないバグも多い。そのため、バグが生じるまで何度もアニメーションしながらデバッギを実行するのでは大変効率が悪い。従って再現性のないバグがあることがわかっているプログラムに対しては、バグが生じるまではプログラムの視覚化を行なわず、実行ログをとりながらできるだけ速やかにデバッギを実行することができるようにした。これによってログに従って再びデバッギをバグが生じた時と同じ実行順序で実行し、その様子をアニメーションするということができるようになる。またログさえ残っていれば、何度でもバグが生じた時とまったく同じ状況を再現できる。

5 可読性の向上

一般的なプログラムでは、ソースコード中の同じ部分を何度も繰り返すことが多い。プログラマにとってバグがないと既にわかっているソースコードを何度もアニメーションして表示することは、得られる情報がないばかりか余計な時間がかかったりバグのない部分に注意が多く向けられたりといった好ましくない結果をもたらすばかりである。そこで実行ログをテキストファイルにし、適当な時間的な位置に”breakpoint” という命令文を書き込めるようにした。プログラマがデバッガを使用中に continue 命令を送ると、デバッガはアニメーションを一時中断して”breakpoint” の位置までデバッギの実行点を進めることができる。また、デバッガか

Animated Debugger for multi-threaded Java Programs, Yuji Yamaoka and Minoru Terada, Department of Mechano-Informatics, School of Engineering, University of Tokyo

ら直接デバッギの現在の実行点に対応するログの位置に”breakpoint” を書き込むこともできる。

さらに、終了したスレッドウィンドウは自動的に閉じ、また、ユーザーが表示したくないウィンドウすなわちスレッドを指定することができるようにした。これらによって、画面内に沢山のウィンドウが現れてあるスレッドへの注目の妨げとなることを防ぐ。

6 実装

JPDA(Java Platform Debugger Architecture) を用いて Java 言語により実装した。

本システムでの基本的なデバッグの流れは、

1. デバッギを動かしながら自動的にその実行点をログに取る
2. バグが発見されたら一旦デバッギを終了
3. ログに従って再度デバッギを実行、バグが出る状況が視覚化されつつ再現される

となる。

デバッギの視覚化の様子を図１に示す。図１において、左中央のウィンドウが全体のコントロールウィンドウで、ユーザーのコマンドを受け付ける。右上下の２つのウィンドウは２つのスレッドを表しており、現在の実行点が下段のスレッドの notifyAll(); であることがわかる（Queue.java というタイトルの下地色が現在の実行点のソースコードであることを示している)。

なおデバッガ機能として、コマンドライン入力による、

- breakpoint 書き込み
- continue（前述）
- 変数表示
- コールツリー表示
- スレッドの状態表示

を実装した。

7 評価

現時点ではあまり客観的な評価をしていないが、Java 言語によるマルチスレッドプログラムの初心者の方に「全スレッドの実際の動きがわかりやすい」とのコメントを頂いた。今後は Java マルチスレッドプログラムで比較的生じやすいバグを調査し、それらのデバッグの効率を調べる実験を通じて評価する予定。

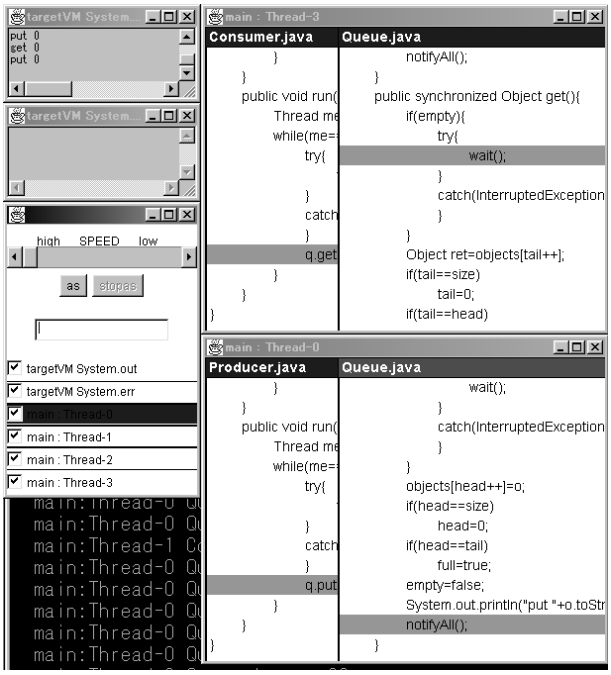

図 1: デバッギの視覚化の様子

8 関連研究

スレッド毎にウィンドウを用意することによって、スレッド切替に伴うデバッガのソースビューでの実行点の不連続的な移動をなくしたツールに ThreadViewer[2] がある。しかし、ThreadViewer はスレッド間の同期の様子を視覚化することに重点を置いており、単一スレッド内での実行点の移動が不連続である。本研究は各スレッドの実行点の移動が滑らかに目で追えるような視覚化をテーマとしており、十分に独自性が主張できると考えられる。

参考文献

[1] 寺田 実: 電子紙芝居, インタラクティブシステムとソフトウェア, 日本ソフトウェア科学会 WISS’98,pp.181-186,1998.

[2] 片山 透, 中本 幸一, 臼井 和敏: Java プログラムのスレッド視覚化ツール, http://www.nec.co.jp/japanese/product/ccsoft/java/devenv/products/tv/