Java デバッガによる式の監視機能の必要性と実現に関して

久米 出[1,a)]　新田 直也[2,b)]　柴山 悦哉[3,c)]　中村 匡秀[4,d)]

概要：Java プログラムのデバッグではしばしば式の実行時の値の調査が必要となる。標準的なデバッガはこの調査を支援するために式を**評価**する機能を実装している。しかしながら式の評価に副作用が伴う場合には、プログラムの本来の実行を変えてしまわないようにソースコードの書き換え等の措置が必要となる。こうした措置に掛かる労力が評価機能を用いる利点を帳消しにしてしまう事も珍しくない。
式の副作用に関するこの問題を解決するために、我々は式の評価に代わるものとして式の値を**監視**する機能を提案し、現在その実装を進めている。本稿では式の監視がもたらす利点、式の評価との原理的な相違点、そして監視機能の開発を進める上で明らかになった実装上の問題点を紹介する。

キーワード：Java デバッガ，式，評価，コンパイラ

On the Necessity and Implementation about Expression Watching by Java Debuggers

Abstract: Developers who debug Java programs often need to examine runtime values of expressions. A standard Java debugger has a feature to *evaluate* expressions in order to support such examinations. However, evaluating an expression with side effects requires additional efforts such as modification of original source code so that the debugged process doesn't differ from the original process. Such additional efforts often cancel out the merits to use the evaluation feature.
In order to solve this problem, we propose and develop a new debug feature to *watch* runtime values of expressions instead of their evaluations. In this paper, we explain the merits of this watching feature, its fundamental difference from evaluation, and implementation issues we have found in our development.

Keywords: Java Debugger，Expression，Evaluation，Compiler

1. 研究の背景と動機

Java プログラムのデバッグ時には式 (expression) の実行時の値を調べる作業が不可欠である。複雑な式を構成する部分式の値を段階的に調査したり、ループ内の特定の式に対してその繰り返し毎の値の一覧を取得する等、様々な要望に対してデバッガが支援出来る事が望ましい。

```
1  while(list.next().num() < line++) {
2    if(indent()+size > max ||
3        match.group(i).length() != text()+add){
4      // Body statements.
5    }
6  }
```

上記のコード中には `Iterator` インスタンスから値を取得する式 (`list.next()`) と変数値を増加する式 (`line++`) が含まれている。いずれも `while` 命令によって繰り返し評価される。これらの式の評価には副作用が伴う。

標準的な Java デバッガは式を評価する機能を有しており、条件付きブレイクポイントと組み合わせる事によって作業者の様々な要望に応じた値の表示が可能となる。式の

[1] 奈良先端科学技術大学院大学
Nara Institute of Science and Technology
[2] 甲南大学
Konan University
[3] 東京大学
The University of Tokyo
[4] 神戸大学
Kobe University
[a)] kume@is.naist.jp
[b)] n-nitta@konan-u.ac.jp
[c)] etsuya@ecc.u-tokyo.ac.jp
[d)] masa-n@cs.kobe-u.ac.jp

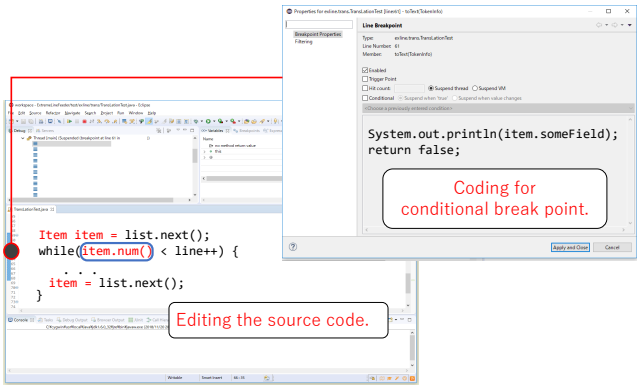

図 1　式の評価
Fig. 1　Evaluation of Expression

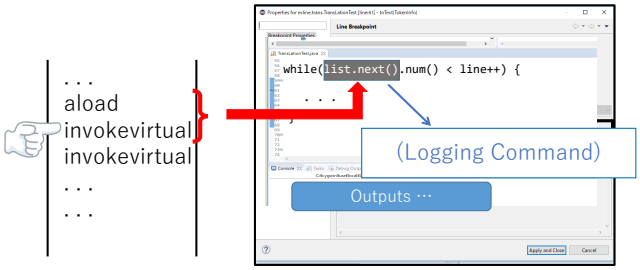

図 2　式の監視
Fig. 2　Watch of Expression

評価が副作用を発生させなければ悪名高い printf デバッグのようなソースコードの書き換えをする事無く、繰り返し評価される式の値のログを取得する事が可能となる。

しかしながら、評価される式が副作用を有する場合にはソースコードの変更が必要となる場合がある。例えば `list.next()` によって実行時に取得される各オブジェクトに対してある特定の属性値の一覧を取得するためには、図 1 に示すようなコードの編集が避けられない[*1]。

一般にデバッガはブレイクポイントで指定されたコードを単純に実行する。ブレイクポイントで `list.next()` を評価すると、本来の実行には存在しなかった余計な副作用が発生してしまう。これを避けるためにソースコードの書き換えが必要となった。これでは悪名高い printf デバッグを採用するのと本質的に何も変わらない事になる。

2.　式の監視

Java コンパイラは各バイトコード命令にソースコード上の行番号を与える。デバッガはこの行番号を用いてプログラム実行がブレイクポイントに到達している事を認識し、そのブレイクポイントに設定されたコードを実行する。この時コード中の式は評価される。

第 1 節で述べたように式の評価で副作用が発生する時には、それによって本来の実行の内容が変わってしまわないように、ソースコードの変更等の措置が必要となる。デバッガによる式の評価を利用する限りこの問題は避けられない。我々はこの問題を抜本的に解決するために、評価に代えてデバッガによる式の監視機能を提案する。

デバッガによる式の監視とは (1) ソースコード中の各式に対してその式の評価を実施するバイトコード命令を特定し、(2) 該当する命令が実行される際に生成する値をその式の値とする機能である (図 2)。

デバッガはデバッグ対象プロセスに対して余計な実行 (式の評価) を挿入する代わりに、プロセスの内容を監視する事によって指定された式の値を取得する。副作用に対する対処を必要としないため、例えばソースコード上の式を選択するだけでその式の実行時の値を出力するような仕組みの実装が可能となる (図 2)。さらに式単位でのブレイクポイントの指定やステップ実行も原理的には可能となる。

3.　監視機能の実現

我々は標準的なデバッグの枠組み [2] 上に式の監視機能を実現する事を研究の最終目標としている。標準的な Java コンパイラはバイトコード命令に対して行番号を与える事によってソースコードと関連付けている。

我々はソースコードを変換する事によって、バイトコード命令の対応する式を (もし式に対応しているのであれば)、行番号から自動的に決定する手法を開発中である [1]。現在の手法では幾つかの種類のバイトコード命令に対して対応付けの誤りが発生する。我々は誤りを自動的に補正する仕組みを実装するために、バイトコードの種類毎に誤りのパターン分析を進めている。

4.　議論

式の監視の応用を考える上で、デバッグ時の式の値の調査に関する状況や要求の分析が必要である。真偽値を計算する関係式 (Relational Expression) 等に対して、しばしばコンパイラは式の値を実行時に生成しないようなバイトコードを生成する。このようなソースコード上の式とそのバイトコードの隔りの存在を効率的に発見し、対処を実現するための支援環境が必要である。またこうした問題やその対処は個別のコンパイラに依存するため、なるべく依存性の少ない解決の枠組みの考察も必要である。

参考文献

[1] Kume, I., Nakamura, M. and Nitta, N.: Revealing Implicit Correspondence between Bytecode Instructions and Expressions Determined by Java Compilers, *25th Australasian Software Engineering Conference (ASWEC) and Australasian Software Week (ASW)*, IEEE (2018).
[2] Oracle and its affiliates: Java Platform Debugger Architecture (JPDA), https://docs.oracle.com/javase/6/docs/technotes/guides/jpda/index.html.

*1　ブレイクポイントで一旦実行は中断されるのだが、設定されたコードが `false` を返すためにそのまま実行が継続される。

*1　本研究は科研費による助成を受けている。(15K12009, 17H00731,16H02908)