

JPF Inspector

Abstract

The goal of this project is to provide a tool for inspection and control of JPF execution - the JPF Inspector.

JPF Inspector will support breakpoints, "single step" execution (forward and backward) at different levels of granularity. It will allow the user to examine program's state - threads, call stacks, and heap - and to modify it, like in a standard debugger (GDB). The tool will also be able to display internal state of choice generators and listeners. Basic profiling information will be captured.

JPF Inspector will provide two user interfaces: command-line and GUI based on the JPF-shell framework.

Contact

student: Pavel Jancik <Alfik.009@...> (PJ)

mentor: Pavel Parizek <pparizek81@...> (PP)

co-mentor:

Program & Timeline

This project is funded by the [Google Summer of Code \(GSoC\)](#) program. It follows the overall [GSoC timeline](#) (start 05/24, finish 08/16).

Current detailed timeline:

- 05/10 - 05/16 design stage: create models of interface and main "server" classes
- 05/17 - 05/23 design stage: Eclipse project
- 05/24 - 06/03 basic commands (run, continue), basic breakpoints
- 06/04 - 06/11 textual console + integration into JPF shell
- 06/12 - 06/19 breakpoint infrastructure - continuation, logging
- 06/20 - 07/10 various kinds of breakpoints (instruction execution, choice)
- 07/12 - 07/16 single stepping, finalizing milestone (MTE)
- 07/17 - 07/25 program state observer
- 07/26 - 08/01 state expression evaluation (for displaying variables)
- 08/02 - 08/08 configuration, saving and loading executed commands
- 08/09 - 08/15 code cleaning, testing, documentation
- 08/16 - 08/20 final evaluation, polishing

Future plans

- profiling support: collecting statistics, aggregating data ...
- better user interface: source code view, choice lists for stepping commands, program state explorer

Description

Planned basic features of the JPF Inspector tool include:

- Breakpoints. They can be set to: a specific line of code, instruction at a given position, specific instruction type (invoke, return), read or write access of a variable, object creation and destruction, garbage collection (start and stop), transition boundary (next value choice). Each breakpoint can be in any of the following three states: enabled, disabled, and log. If a breakpoint in the log state is reached, a log message is printed and the execution is not interrupted.
- Single-step execution. It will be possible to tell JPF to execute the SuT in a single-step fashion. Supported modes (granularity) both for forward and backward steps will be: by instruction, by transition, by choice generator of a specific type (data X scheduling). When the execution reaches a forward step that involves a choice, the user will be able to decide which option (value or thread) to use.
- Program state inspection. Inspector will provide a mechanism for exploring and altering program state, including values of heap objects and call stacks of individual threads. Navigation over references (object tree) will be supported too - a custom expression language will be used internally.

Advanced features may include some of the following: JPF introspection (displaying raw state of choice generators and listeners), gathering of various statistics (profiling information), batch execution (saving and loading command sequences), and configuration management (predefined breakpoints).

Textual console interface (command-line) will behave like in the GDB debugger. The GUI interface will be implemented as a new panel for [JPF shell](#). In particular, a user interface of any kind will display the current position in the SuT code (if single-step execution is turned on) and it will allow the user to manage breakpoints.

User guide

[User guide](#) describes the currently supported commands and extension points (API) of the Inspector tool. It includes many examples and some screenshots of the Inspector user interface.

Repository

The sources for this project are available from a Mercurial repository on <https://bitbucket.org/alf009/jpf-inspector>

Project Blog

2010-08-07 (PJ) - state observer: printing thread information

2010-08-07 (PJ) - single stepping over source code lines

2010-07-11 (PJ) - first part of user manual written

2010-07-10 (PJ) - more kinds of breakpoints implemented

2010-06-19 (PJ) - breakpoints attached to source code line implemented

2010-06-19 (PJ) - inspector console in the form of a shell panel created

2010-06-03 (PJ) - basic commands (run, continue) implemented

2010-05-25 (PJ) - design model added (set of class diagrams)

2010-05-15 (PP) - basic description added

2010-05-01 (PJ) - detailed timeline proposed

2010-05-07 (NR) - project page created