Object instance profiling

Lukáš Marek
http://dsrg.mff.cuni.cz

CHARLES UNIVERSITY PRAGUE
Faculty of Mathematics and Physics
Introduction

• Two standard types of profiling
  ▪ Sampling
    • Pro: configurable overhead
    • Con: not precise
  ▪ Instrumentation
    • Pro: precise
    • Con: large overhead (collecting stack traces)
Motivation

• Collecting stack trace for every invocation is too expensive
  ▪ But it's the only way (we know) how to obtain context (for std. instrumentation)

• We want precise results

• Easy to use (automated as much as possible)
Object Instance Profiling

• Idea: Don't profile code, profile instances
  ▪ Similar to instrumentation
  ▪ Pro: No stack trace “needed” (but possible)
    • Context is the instance itself
  ▪ Pro: Instance-specific performance data
  ▪ Con: Harder implementation (hidden from user)
  ▪ Con: Not useful every time
    ▪ Con: Right now not as easy to use as other methods
      • But we are doing our best
Object Instance Profiling II

- Part of the JPMF framework
- Idea comes from component instrumentation
  - Useful for component instrumentation
    - No need for special instrumentation for different frameworks
  - Also suitable for non-component application
Three implementation variants
   a) Proxy object
   b) Proxy methods (single class)
   c) Proxy methods (class hierarchy)

Implementation details
Pros x Cons
Proxy object

• Proxy object is placed between reference to the original object and the original object itself
  ▪ All methods are called through the proxy object
  ▪ Call interception is done in the proxy object
Proxy object – Inheritance

- Simple
- Not always possible (final classes)

Proxy

Profiled instance

Original method

Proxy method

Call
Proxy object – Delegation

- Problem with different types
- Problem with different instances
Proxy object – Pros x Cons

- **Pro**: Simplest solution
- **Pro**: Code instrumentation needed only around an instance creation point
- **Pro**: Overhead is connected only with the measured instance
- **Con**: All creation points has to be identified
  - Problem with factory methods, multiple object creations, frameworks
Proxy methods (single class)

• All public methods (from all classes) in an instance are replaced with proxy methods
  ▪ Overridden only in topmost class

Profiled instance

Proxy method

Call

Original renamed method
Proxy methods (single class) II

- Pro: instrumentation needed only in one class
- Pro: Parents are not affected
- Con: Children are affected
- Con: Run-time check is needed
- Con: Call chaining
Proxy methods (class hierarchy)

- All public methods (from all classes) in an instance are replaced with proxy methods
  - Overridden in class where the particular method is defined

 Profiled instance

Proxy method

Call

Original renamed method
Proxy methods (class hierarchy) II

- **Pro**: No call chains
- **Con**: All classes in class hierarchy are affected by instrumentation
- **Con**: Run-time check is needed
Instance identification

- By name
  - In a field
  - As a method call
- Context identification (stack trace)
- Assigned number
  - Not persistent between invocations
Current status

• Proxy methods (class hierarchy) implemented
  ▪ Instances identified by creation time
• Part of the JPMF
  ▪ Java
  ▪ Uses ASM
• Successfully tested on smaller examples
• Prepared for q-impress e-soa showcase
Current status II

• Online instrumentation with agent supplied at start-up
  ▪ Byte code instrumentation, no source code needed

• Requires offline creation of the instrumented class hierarchy
  ▪ Special utility
  ▪ Only instances of these classes can be measured
• ... 

• Enabling/Disabling events (instances) on the fly with jconsole (MBeans)
Future work

• Measure the profiling overhead and publish a paper about it
• Consider other implementation methods
• On the fly instrumentation
  ▪ No offline utility needed
  ▪ Access jvm when it is already running
Questions?