APM

POSEIDON HOUSE • CASTLE PARK • CAMBRIDGE CB3 0RD UNITED KINGDOM
+44 1223 515010 • Fax +44 1223 359779 • Email: apm@ansa.co.uk • URL: http://www.ansa.co.uk

A benchmark test of Reflective Java

Takanori Ugai

Abstract

This document is a summary on the overhead of the Reflective Java's current implementation.

Doc No. (Sum Info)	**Draft / Approved** Document Type	4 February, 1997

Distribution:
Supersedes:
Superseded by:

TABLE OF CONTENTS

1 INTRODUCTION

The Reflective Java is designed and implemented in the ANSA programme. Reflective Java's goal is to provide applications with the capability to customise their behaviour in order to meet particular application requirements flexibly and transparently using reflection technology. Implementations of reflection mechanism is often very inefficient. A simple benchmark program for Reflective Java and results are presented in this document. And analysis of the results and some suggestions for improvements are described.

2 REFLECTIVE JAVA

The Reflective Java enable Java-powered systems to be customised dynamically, flexibly and transparently, by supporting the Metaobject Protocol approach. Reflective Java enables programmers to change the behaviour of method invocations by specifying what should be done before or/and after "normal" method execution: metaBefor is executed before a method is invoked, and metaAfter is executed afterwards.

3 BENCHMARK PROGRAM

3 .1 Base object

The base object has one method, which simply returns 0.

```
class Base {
 public int aMethod() throws Throwable {
   return 0 ;
 }
}
```

3 .2 meta object

```
import java.io.* ;
import metaobject.* ;
class meta_Base extends MetaObject {
 public meta_Base(Object obj, String classnm) {
   super(obj,classnm) ;
 }
 public String metaBefore(int m_id, int c_id, ArgPack args) {
  return "noException" ;
 }
 public String metaAfter(int m_id, int c_id, ArgPack args,
ArgPack rpl) {
  return "noException" ;
 }
}
```

The reflective object is empty that means that the invoked method returns immediately and metaBefore and metaAfter simply returns "NoException."

Potentially the rate between object creations represents the cost for initialisation of a reflective object and the rate between method invocations represents the costs of invocation of metaBefore and metaAfter.

3 .3 Main loop

The benchmark applet is insisted to evaluate the reflective Java's overhead.

In the program, In the demo, the main loop creates a reflective object 1000 times, invoke a method of the object 1000 times, creates the base object 100000 times, and invokes a method of the base object 100000 times. The main object gets the date at each phase and displays the run time and calculates and displays the rate.

```
public static void main(String args[])
{
 private refl_Base obj;
 private Base bobj ;
 obj = new refl_Base() ;
 try {
   for(int count = count < 1000 ; count ++) {
      obj = new refl_Base() ;
   }
 } catch (Throwable ex) {
   ex.toString() ;
   return ;
 }
 checkTime() ;
 try {
   obj = new refl_Base() ;
 } catch (Throwable ex) {
   ex.toString() ;
   return ;
 }
 try {
   for(int count = 0 ; count < 1000; count++) {
     obj.aMethod() ;
   }
 } catch (Throwable ex) {
   ex.toString() ;
 }
 checkTime() ;
 bobj = new Base() ;
 try {
   for(int count = count < 100000 ; count ++) {
      bobj = new Base() ;
   }
 } catch (Throwable ex) {
   ex.toString() ;
   return ;
 }
 checkTime() ;
 try {
   bobj = new Base() ;
 } catch (Throwable ex) {
   ex.toString() ;
   return ;
 }
 try {
   for(int count = 0 ; count < 100000; count++) {
     bobj.aMethod() ;
   }
 } catch (Throwable ex) {
   ex.toString() ;
```

```
}
checkTime() ;
```

4 RESULTS

Table 1 Running environment and results

Machine	Interpreter	cration time (1)	ivoke time(2)	creation time (3)	invocation time (4)	creation overhead (5)	invocation overhead (6)
P100, 40Mb, WinNT4.0	NN3.0	741	220	2604	40	28	550
HP9000 Model700 HP-UX	NN3.0	1197	746	5953	1395	20	53
486DX4, 75Mhz 24MB	NN3.0	1420	440	5500	110	25	400
SS20, Solaris2.5	NN3.0	609	158	1533	691	39	22
SS20, Solaris 2.5	JDK1.1 appletviewe	104	95	376	184	27	51
SS20, Solaris 2.5	JDK1.02 appletviewe	205	232	1240	708	16	32

(1) is the time (ms) for the creation of a reflective object 1000 times.

(2) is the time (ms) for the invocation of a reflective object method 1000 times.

(3) is the time (ms) for the creation of a “normal” object 100000 times

(4) is the time (ms) for the invocation of a reflective object method 100000 times.

(5) is the result of (1)*100/(3)

(6) is the result of (2)*100/(4)

NN stands for Netscape Navigator

5 ANLYSYS & IMPROVEMENT

5 .1 Reflective Object Creation

The benchmark result says a reflective object creation requires several dozen times as seconds as much as a normal object does. In the current implementation, when a reflective object is created, a metaobject class object is created, local variables are initialised and base object is created. In the initialisation phase, all information of all methods of the base object are registered. The information includes the name, the parameter's number, type, name. That implies the reflective object creation would take the time in proportion to the number of the methods.

5 .2 Reflective Method Invocation

The benchmark result says a method invocation of the reflective object takes from several dozen to several hundreds times seconds as much as a method invocation of the base object does. A method invocation consists of the unpacking the arguments, invoking the metaBefore, the check of the returned string from the metaBefore, the invocation of the base method, the invocation of metaAfter and the check of the returned string. Lots of exceptions thrown by methods executes lots of condition checks.

5 .3 Other overheads

A reflective object requires some other classes than the base class and meta class definition. That means it takes long loading time for the classes.

5 .4 The optimisation of generated codes

MetaBefore and metaAfter return string, and the reflective object check the return value. We can use a integer instead of string.

5 .5 Programmers tips

- Minimise the number of the method in a reflective object.

- Minimise the number of the arguments. If possible, it should be one object.
- Minimise the number of exceptions which the methods throw.

6 REFERENCES

[APM1931] *Reflective Java*, Zhixue Wu and Scarlet Schwiderski

[APM1877] *Reflective Java : The Design, Implementation, Applications,* Zhixue Wu and Scarlet Schwiderski

[Java] *The Java Language Specification*, James Gosling, Bill Joy, Guy Steele, URL:http://www.javasoft.com/doc/language_specification/index.html

[API] *Java API Documentation,* James Gosling, Frank Yellin, The Java Team URL:http://www.javasoft.com/products/JDK/1.0.2/api/

[APM1911] *Design of Reflective Java,* Zhixue Wu and Scarlet Schwiderski