Main Menu

Add Logging at Class Load Time with Java Instrumentation

April 24, 2008
Thorbjørn Ravn Andersen

{cs.r.title}

- **Contents**
-

Main Menu

- Home
- Projects
- Forums
- People
- Java User Groups
- JCP

Add Loggi at Class Load Time with Java Instru

April 24, 2008

Thorbjørn Ravn Andersen

- **Conte**
-
  - U
    th

Ja
In
A
- Th
cc
Sa
A
- Th
Sa
M
Fi
- Th
Sa
bu
Fi
- C
- R

When

you're trying to analyze why a program failed, a very valuable piece of information is what the program was actually doing when it failed. In many cases, this can be

determined with a stack trace, but frequently that information is not available, or perhaps what you need is information about the data that was being processed at the time of failure.

Traditionally this means using a logging framework like "http://loggi or the "http://java.s summary.htm Java Logging API, and then writing and maintaining all necessary log

statements manually. This is very tedious and error-prone, and well-suited for automation. Java 5 added the "http://java.s Java Instrumenta mechanism, which allows you to provide "Java agents" that

can inspect and modify the byte code of the classes as they are loaded.

This article will show how to implement such a Java agent, which transparentl will add entry and exit

logging to all methods in all your classes with the standard Java Logging API. The example used is Hello World:

```
public cla
    public
        Sy
    }
}
```

And here is the same program with entry and exit log statements added:

```
import jav
import jav
import jav

public cla
    final

    public
        if

        }
        Sy
        if
```

```
        }
    }
}
```

The default logger format generates output similar to:

```
2007-12-22
INFO: &gt;
Hello Worl
2007-12-22
INFO: &lt;
```

Note that each log statement is printed on

two lines. First, a line with a time stamp, the provided log name, and the method in which the call was made, and then a line with the provided log text.

The rest of

the article will demonstrate how to make the original Hello World program behave like the logging Hello World by manipulatin the byte code when it is loaded. The manipulatio

mechanism is the Java Instrumenta API added in Java 5.

Using the Java Instrumen API

You can invoke Java with the JVM arguments `-javaagent:` or

-
`javaagent:`
to have
Java call
the
`premain(..`
method
listed in the
manifest of
youragent.ja
before
trying to
run the
`main`
method
specified.
This
`premain(..`
method
then allows

you to register a class file *transformer* with the system class loader, which provides a `transform(` method. This method is then called as a part of loading each and every

class from then on, and may manipulate the actual byte codes before it is processed by the class loader into a real `Class`.

For this to work you must have several pieces in

place:

A class implementi

`ClassFileT`

The `transf` method will be called for each class as it is being loaded. The argume is the

comple
origina
byte
code
for the
whole
class.

A class providing a static void `premain()` method.

The `premai`
method
must
register
the

above transfo with the class loader. It can also handle the argume provide at the comma line.

A correct *MANIFEST* file.

The

MANIF
must
contain
a `Pre-Class:`
`..` line
referrin
to the
class
with
the
`premai`
method
Additio
use
`Boot-Class-Path:`
to
refer

to
externa
.jar
files.

The code must be put into a .jar file with this manifest. If not, it will silently fail.

The `com.runjv` Sample Agent

This section lists a sample agent named `com.runjva` It implements the `java.lang.` interface and provides the `premain(..` method required.

The actual

byte code manipulatio
in the `transform(`
method is done with the JBoss "http://www
library, which provides both a Java snippet compiler and high-level byte code manipulatio
routines.

The compiler allows us to do the manipulatio by creating Java string snippets that are then compiled and inserted at the appropriate location.

The signature

extraction and return value type extraction methods are rather complex and have been placed in `com.runjva` which is not listed but available in the sample code .zip file.

See the Resources section for the sample code and links to Javassist and background articles.

This is the `com.runjva` class:

```
package co

import jav
import jav
```

```
import jav

public cla

  public s
      Inst
    if (ag
      Stri
      Set

      if (
        Sy
        Ru



        })
      }
      // .
    }
    instru
  }
```

The `premain(..` method is responsible for adding the `LoggerAgen` as a class transformer. It also considers the string argument as a comma-separated list of options and if the

option `time` is given, the date is printed now and at shutdown time.

```
String d
String i

String[]

public b
    Clas
    byte

  for (i
    if (
      re
    }
```

```
        }
        return
    }
```

The `transform(` method is called for every class loaded by the system class loader just before being instantiated into a real object. Each class contains code to

load those classes it needs, so to avoid adding loggers to the runtime library classes, it is necessary to look at the class name and return library classes unmodified (note that

the separators are slashes, not dots).

```
private
  ClassP
  CtClas
  try {
    cl =
    if (

      Ct
      St

      cl

      Ct
      fo
```

```
            }
            b
        }
    } catc
        Syst

    } fina
        if (
            cl
        }
    }
    return
}
```

The `doClass(..` method uses Javassist to analyze

the byte stream provided. If it is a real class (as opposed to an interface), a logger field named `_log` is added and initialized to the name of the class. Each non-empty method is then

processed with `doMethod(.` The `finally` clause ensures that the class definition is removed again from the Javassist pools to keep memory usage down.

```
  private
      thro

    String
    String

    method
        +

    method
        +
  }
}
```

The doMethod(..
class creates

```
if
(_log.isLo
```

snippets to insert at the beginning and end of each method. This level has been chosen as it is the lowest level that generates output without any configuratio of the logging

system.

Note that the `JavassistH` class is available in the sample code .zip file (see "#resources"

The Sample *MANIFES* File

Only two lines are

needed here: one to point to the class with the `premain` method, and one to make Javassist available to the agent.

`Premain-Cl`
`Boot-Class`

Note that the *dist/loggera* needs to

lib/javassist-hence the *../lib* relative path.

The Sample *build.xml* File

The *build.xml* file contain a compilation target, a .jar target, a

traditional `HelloWorld` target, and a `HelloWorld` target with the logger agent active.

```
&lt;projec
  &lt;targ

  &lt;targ
    &lt;ja
      &lt;
    &lt;/j
  &lt;/tar

  &lt;targ
    &lt;ja
    &lt;/j
```

```
  &lt;/tar

  &lt;targ
    &lt;de
    &lt;mk
    &lt;ja

    &lt;/j
  &lt;/tar

  &lt;targ
    &lt;ja
  &lt;/tar
&lt;/proje
```

Running `ant` gives output similar to:

```
Buildfile:
```

```
compile:
    [delete
     [mkdir
     [javac

jar:
        [jar

withoutAge
      [java

withAgent:
      [java
      [java
      [java
      [java
      [java
      [java
      [java

all:
```

```
BUILD SUCC
Total time
```

The output shows that the logging statements have been added and that they actually generate output. The actual order of the statements may change

from run to run, as the log statements are written to `System.err` and the timing info, as well as the output from `HelloWorld` is written to `System.out`

Conclusio

The Java

Instrumenta
API can be
used to
transparentl
add
method-call
logging to
any Java
code at
runtime
without
changing
the source
code or the
compiled
byte code.
By
automating
the

generation of log statements, it is guaranteed that they are always up to date and the programmer are relieved of this tedious task.

Resources

- "/today code

for this article
- "http://summa Officia Java 6 Javado for `java.l`
- The Javassi home page; contain a tutorial downlc and

forums

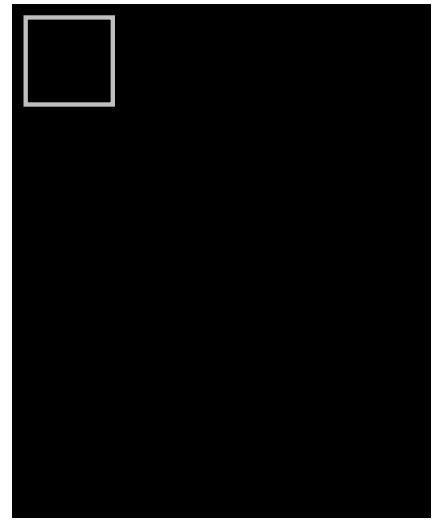
- "
"http://
Modify
Applic
with
Java 5
Class
File
Transfo
an
article
at
Javalob
by
R.J.
Lorime
outlinii
the

basics

- "http://log-messag up Log4J in Java 5" by Inigo Surguy explain how to use instrum to remove those

log
stateme
that
are not
enablec
at
class-
load
time.
Similar
to this
agent,
but
require
log
stateme
to be
present
in the

code.



Thorbjørn Ravn Andersen is a Senior Software Engineer who creates web services, frontends

and helper programs for legacy Cobol applications in Java on the IBM iSeries.

Related Topics >>

Programmir

|

Article Links

>>

- Login or registe

to post
comme

- Printer-
friendly
version
- ShareT
- 27474
reads

Comme

A simple/str and

by
kiranbhogad
- 2010-01-

29 18:01

A simple/straig and well-written article. Nice intro to how the instrumentat framework works ... with code. Thanks! Kiran Bhogadi

- Login or

registe
to post
comme

I found this article very useful and implemen an improved ...

by *dmpl* - *2011-09-02*

07:51

I found this article very useful and implemente an improved version of the LoggerAger

The class from the article requires that javasist.jar and all

other files needed to compile the inserted java code are added to the bootstrap class path. It is particularly bad if the solution is used in applications with many class loaders or if the

libraries the compiled code depend on should not be on the bootstrap class path because of any other reasons. Debugging of the transformer classes in IDE is also more difficult in this case.

Therefore I split the LoggerAgen in two different classes, the agent and the transformer. Here is my solution:

```
import jav
import jav
import jav
import jav

public cla

    privat
    privat
```

```
privat
public

    tr
    in
}

public
    tr
```

```
            }

            }
            re
        }
}
```

The manifest does not require the Boot-Class-Path: ../lib/javassi any more

The class actually used for the

transformati
should be
set as an
argument
of
premain(...)
.

`-javaagent`

This
solution
instantiates
different
transformer
objects for
each class
loader the
transform

is called.

Regards,

Dimitry Polivaev

- Login or register to post comme

I really like the article and hope to solve a

problem this ...

by *ratzlow - 2012-06-23 14:28*

I really like the article and hope to solve a problem this path. It would be very helpful if the sample project was still

available. Could you please let me know where to download it?

TIA

Frank

- Login or registe to post comme

...

by *ratzlow - 2012-06-23 14:45*

...

- Login or registe to post comme

Excellent article. Source link seems to be

messed up, ...

by *benow* - *2012-07-31 06:13*

Excellent article. Source link seems to be messed up, however. I found it here (http://today

- Login or

register
to post
comme

- Feedback
- FAQ
- Terms of Use
- Privacy
- Trademarks

Your use of this web site or any of its content or software indicates your agreement to be bound by these Terms of Participation.

Powered by Oracle, Project Kenai and Cognisync