Instrumentierung
— Das Werkzeug der Werkzeugmacher —

Bernd Müller

Ostfalia
Hochschule Braunschweig/Wolfenbüttel

Agenda

- Instrumentierung und Agenten
- Redefinition und Retransformation
- Beispiele

Vorstellung Referent

- Prof. Informatik (Ostfalia, HS Braunschweig/Wolfenbüttel)
- Buchautor (JSF, Seam, JPA, ...)

- Mitglied EGs JSR 344 (JSF 2.2) und JSR 338 (JPA 2.1)
- Geschäftsführer PMST GmbH
- ...
- bernd.mueller@ostfalia.de
- @berndmuller

Instrumentierung und Agenten

oder

Alt, mächtig, aber unbekannt ...

Das Package `java.lang.instrument`

- Das Package `java.lang.instrument` wurde mit Java 5 eingeführt
- Java-Doc:
  „*. . . Provides services that allow Java programming language agents to instrument programs running on the JVM. The mechanism for instrumentation is modification of the byte-codes of methods. . . .*“
- Verwendungsmöglichkeiten: Monitoring, JPA-Provider, Code-Coverage, . . .
- Allgemein: Sinnvolles Verhalten, das nicht im Code steht, nachträglich und nur bei Bedarf einbauen

Agenten

- Deployed als Jar-File
- Attribut im Manifest definiert die Agenten-Klasse
- Alternativen, um Agent zu starten
  - Auf Kommandozeile bei VM-Start (zwingend erforderlich für Kommandozeilenimplementierungen)
  - Nach VM-Start, z.B. durch nicht näher spezifiziertes Binden (optional und implementation dependent)
- Artikel Java Magazin 3/2015: *James Bond lässt grüßen - Geheimagenten (oder geheime Agenten) bei ihrer Arbeit*

Pre-Main

oder

Gibt es ein Leben vor `main()` ?

Agentenstart über Kommandozeile

- Syntax: `-javaagent:`*jarpath[=options]*
- Option mehrfach verwendbar, damit mehrere Agenten
- Manifest des Agenten-Jars muss Attribut `Premain-Class` enthalten
- Diese Agentenklasse muss `premain()`-Methode enthalten
- Nachdem VM initialisiert ist, werden alle `premain()`-Methoden in der Reihenfolge der Optionen aufgerufen, dann die `main()`-Methode
- Zwei mögliche Signaturen:

```
public static void premain(String agentArgs,
                           Instrumentation inst);
public static void premain(String agentArgs);
```

- Aufruf der zweiten Alternative nur, falls erste nicht existiert
- Agent wird über System-Class-Loader geladen

Das Attach-API

oder

Wie rede ich mit einer VM ?

Instrumentation-Package

„*. . . Implementations may also support a mechanism to start agents some time after the VM has started. For example, an implementation may provide a mechanism that allows a tool to attach to a running application, and initiate the loading of the tool's agent into the running application. The details as to how the load is initiated, is implementation dependent.*“

- Achtung: implementierungsabhängig
- Aber: in HotSpot, JRockit, IBM, SAP vorhanden
- Schnittstelle ist die abstrakte Klasse `VirtualMachine` im Package `com.sun.tools.attach`, enthalten in `tools.jar`

Java-Doc Klasse `VirtualMachine`

„ *A VirtualMachine represents a Java virtual machine to which this Java virtual machine has attached. The Java virtual machine to which it is attached is sometimes called the target virtual machine, or target VM. An application (typically a tool such as a managemet console or profiler) uses a VirtualMachine to load an agent into the target VM.* “

- Methode `attach(<pid>)`
  Fabrikmethode, um angebundene Instanz zu bekommen
- Methode `loadAgent(<agent>,<args>)`
  um Agent zu laden und zu starten (Methode `agentmain()`)

Agent mit agentmain()

- Der Agent

```
public static void agentmain(String agentArgs,
                             Instrumentation inst);
public static void agentmain(String agentArgs);
```

- Im Manifest Attribut `Agent-Class` auf Klassennamen des Agenten setzten

Retransformation und Redefinition

oder

Ändern oder neu machen ?

Retransformation und Redefinition

- Möglichkeit, bereits geladene Klassen ganz oder teilweise zu ersetzen
- Beides optionale VM-Funktionen
- `void redefineClasses(ClassDefinition... defs)`
  „Redefine the supplied set of classes using the supplied class files. This method is used to replace the definition of a class without reference to the existing class file bytes, as one might do when recompiling from source for fix-and-continue debugging.“
- `void retransformClasses(Class<?>... classes)`
  „Retransform the supplied set of classes. This function facilitates the instrumentation of already loaded classes.“

Retransformation

- Attribut `Can-Retransform-Classes` im Manifest auf `true` setzen
- Transformer registrieren:
  `Instrumentation.addTransformer(ClassFileTransformer transformer)`
- Entsprechende Methode aufrufen:
  `Instrumentation.retransformClasses(Class<?>... classes)`

Redefinition

- Attribut `Can-Redefine-Classes` im Manifest auf `true` setzen
- Entsprechende Methode aufrufen:
  `Instrumentation.redefineClasses(Class<?>... classes)`

Auszug aus Java-Doc für `redefineClasses()`

- If a redefined method has active stack frames, those active frames continue to run the bytecodes of the original method. The redefined method will be used on new invokes.
- This method does not cause any initialization except that which would occur under the customary JVM semantics. In other words, redefining a class does not cause its initializers to be run. The values of static variables will remain as they were prior to the call.
- Instances of the redefined class are not affected.

Auszug aus Java-Doc für `redefineClasses() (cont'd)`

- The redefinition may change method bodies, the constant pool and attributes. The redefinition must not add, remove or rename fields or methods, change the signatures of methods, or change inheritance. These restrictions maybe be lifted in future versions.
- The class file bytes are not checked, verified and installed until after the transformations have been applied, if the resultant bytes are in error this method will throw an exception.

Beispiele

Beispiel: *Monitoring*

Beispiel Monitoring: Aufrufhäufigkeit der Methode

```
public class ClassToMonitor {

  public void foo() {
    // beliebig
  }

}
```

- Aufrufhäufigkeit der Methode `foo()` soll (auf einfache Art) gezählt werden

Beispiel Monitoring: Zähler und Main

```
public class Monitor {
  public static int counter = 0;
}

public class Main {
  public static void main(String[] args) throws Exception {
    System.out.println("Zaehler vor Schleife: "
                       + Monitor.counter);
    ClassToMonitor classToMonitor = new ClassToMonitor();
    for (int i = 0; i < 1000; i++) {
      classToMonitor.foo();
    }
    System.out.println("Zaehler nach Schleife: " + Monitor.counter);
  }
}
```

Beispiel Monitoring: Der Agent

```
public class MonitorAgent {

  public static void premain(String agentArgs,
                             Instrumentation instrumentation) {
    instrumentation.addTransformer(new MonitorTransformer());
  }
}
```

Und die MANIFEST.MF

```
Premain-Class: de.pdbm.MonitorAgent
```

Beispiel Monitoring: Instrumentierung mit Javassist

```
public class MonitorTransformer
             implements ClassFileTransformer {

  public byte[] transform(ClassLoader loader, String className,
      Class<?> classBeingRedefined, ProtectionDomain protectionDomain,
      byte[] classfileBuffer) throws IllegalClassFormatException {

    if (className.equals("de/pdbm/ClassToMonitor")) {
      ClassPool pool = ClassPool.getDefault();
      try {
        CtClass cc = pool.get("de.pdbm.ClassToMonitor");
        CtMethod method = cc.getDeclaredMethod("foo");
        method.insertBefore("de.pdbm.Monitor.counter++;");
        return cc.toBytecode();
      } catch (NotFoundException | CannotCompileException | IOException e) {
        ...
      }
    }
    return classfileBuffer; // andere Klassen unveraendert
  }
}
```

Beispiel: *Klasse neu laden*

Beispiel Ändern einer Methode und Neuladen

```
public class ClassToBeRedefined {

  public void saySomething() {
    System.out.println("foo");
    //System.out.println("bar");
  }

}
```

Beispiel Ändern einer Methode und Neuladen (cont'd)

```
public class Agent {

  private static Instrumentation instrumentation = null;

  public static void agentmain(String agentArgument,
                               Instrumentation instrumentation) {
    Agent.instrumentation = instrumentation;
  }

  public static void redefineClasses(ClassDefinition...
                                     definitions) throws Exception {
    if (Agent.instrumentation == null) {
      throw new RuntimeException("Agent nicht gestartet. Instrumentierung ni
    }
    Agent.instrumentation.redefineClasses(definitions);
  }
```

Beispiel Ändern einer Methode und Neuladen (cont'd)

```
public class Main {

  public static void main(String[] args) throws Exception {
    ClassToBeRedefined ctbr = new ClassToBeRedefined();
    ctbr.saySomething();
    InputStream is = ctbr.getClass().getClassLoader()
                         // class ClassToBeRedefined
                         .getResourceAsStream("dummy");
    byte[] classBytes = classInputStreamToByteArray(is);
    ClassDefinition classDefinition =
        new ClassDefinition(ctbr.getClass(), classBytes);
    loadAgent();
    Agent.redefineClasses(classDefinition);
    ctbr.saySomething();
  }
```

Beispiel Ändern einer Methode und Neuladen (cont'd)

```
    ...
    private static void loadAgent() {
      String nameOfRunningVM = ManagementFactory.getRuntimeMXBean().getName()
      int p = nameOfRunningVM.indexOf('@');
      String pid = nameOfRunningVM.substring(0, p);

      try {
        VirtualMachine vm = VirtualMachine.attach(pid);
        vm.loadAgent(JAR_FILE_PATH, "");
        vm.detach();
      } catch (Exception e) {
        throw new RuntimeException(e);
      }
    }
}
```

Seeing is believing . . .

Demo

Beispiel: *Alle Unit-Tests bestehen* ;-)

Beispiel „Alle JUnit-Tests bestehen“

```
public class ClassToTest {

  public String getTheCanonicalClassName() {
    return "Falscher Name";
    //return this.getClass().getCanonicalName();
  }

  public int add(int a, int b) {
    return a * b;
    //return a + b;
  }
}
```

Die JUnit-Tests

```
public class JunitTests {

  @Test
  public void testClassName() {
    ClassToTest ctt = new ClassToTest();
    Assert.assertEquals("Falscher Klassenname",
                        ClassToTest.class.getCanonicalName(),
                        ctt.getTheCanonicalClassName());
  }
  @Test
  public void testAdd() {
    ClassToTest ctt = new ClassToTest();
    Assert.assertEquals("Falsche Summe", (3 + 4), ctt.add(3, 4));
  }
}
```

Der Transformer

```
public class JunitTransformer implements ClassFileTransformer {

  @Override
  public byte[] transform(ClassLoader loader, String className,
  Class<?> classBeingRedefined, ProtectionDomain protectionDomain,
  byte[] classfileBuffer) throws IllegalClassFormatException {
    if (className.equals("org/junit/Assert")) {
      return transformAssert(); // ohne Exception-Handling
    }
    // alle anderen Klassen unveraendert zurueckgeben
    return classfileBuffer;
  }

  ...
}
```

Der Transformer

```
private byte[] transformAssert() throws Exception {
  ClassPool pool = ClassPool.getDefault();
  CtClass cc = pool.get("org.junit.Assert");
  for (CtMethod ctMethod : cc.getMethods()) {
      if (ctMethod.getName().startsWith("assert")) {
      ctMethod.setBody("return;");
    } else {
      // die anderen (equals(), clone(), wait(), ...)
    }
  }
  return cc.toBytecode();
}
```

Der Agent

```
public class TransformerAgent {

  public static void agentmain(String agentArgs, Instrumentation instrumenta
    instrumentation.addTransformer(new JunitTransformer(), true);

    Class<?>[] classes = instrumentation.getAllLoadedClasses();
    for (Class<?> c : classes) {
      if (c.getName().equals("org.junit.Assert")) {
        try {
          instrumentation.retransformClasses(c);
        } catch (UnmodifiableClassException e) {
          e.printStackTrace();
          System.err.println(c + " laesst sich nicht modifizieren");
        }
      }
    }
  }
}
```

Und wie besteht man alle Unit-Tests?

```
public class ClassToTest {

  static {
    AgentLoader.loadAgent();
  }

  public String getTheCanonicalClassName() {
    return "Falscher Name";
    //return this.getClass().getCanonicalName();
  }

  public int add(int a, int b) {
    return a * b;
    //return a + b;
  }
}
```

Seeing is believing . . .

Demo

Beispiel: Build your own JRebel Look-Alike

Was man dazu benötigt ...

- File-System-Watcher! Gibt's seit Java 7 im JDK (Interface `java.nio.file.WatchService`)
- Immer wenn `*.class`-Datei geändert wird, diese neu laden
- Das wars schon !

Seeing is believing . . .

Demo

Beispiel: *App-Server Logging*

JBoss' Byteman (byteman.jboss.org)

- Byteman is a tool which makes it easy to trace, monitor and test the behaviour of Java application and JDK runtime code.
- It injects Java code into your application methods or into Java runtime methods without the need for you to recompile, repackage or even redeploy your application.
- Injection can be performed at JVM startup or after startup while the application is still running.
- Injected code can access any of your data and call any application methods, including where they are private. You can inject code almost anywhere you want and there is no need to prepare the original source code in advance.
- You can even remove injected code and reinstall different changes while the application continues to execute.

Seeing is believing . . .

Demo

Fragen und Anmerkungen