Wie Sie immer alle Unit-Tests bestehen ;-)

Das Instrumentierungs-API

Bernd Müller

Java Forum Stuttgart

9.7.2015

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
- Aktiv in JUG Ostfalen
- ...

Motivation

Motivation

- Die schlechte Nachricht:
  - Für Anwendungsentwickler wenig direkt Verwendbares
  - Framework-Entwickler kennen das schon ;-)

Motivation

- Die schlechte Nachricht:
  - Für Anwendungsentwickler wenig direkt Verwendbares
  - Framework-Entwickler kennen das schon ;-)
- Die gute Nachricht:
  - Als interessierter Java-Entwickler lohnt es sich, hinter die Kulissen zu schauen
  - Man versteht einiges besser und wird damit besser
  - Eventuell könnten Sie etwas Spaß haben (siehe Unit-Tests)

Agenda

- Instrumentierung
  - Was ist das?
  - Agenten, Start von Agenten
  - Redefinition, Retransformation

Agenda

- Instrumentierung
  - Was ist das?
  - Agenten, Start von Agenten
  - Redefinition, Retransformation
- Beispiele
  - Monitoring
  - Hot Code Replacement
  - Unit Tests ;-)
  - ?

Instrumentierung

Was ist Instrumentierung ?

Das Package `java.lang.instrument`

- Java-Doc Package `java.lang.instrument`:
  „*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.*“

Das Package `java.lang.instrument`

- Java-Doc Package `java.lang.instrument`:
  „*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.*“
- Wikipedia *Instrumentation* (computer programming):
  „*. . . instrumentation refers to an ability to monitor or measure the level of a product's performance, to diagnose errors and to write trace information. . . .*“

Das Package `java.lang.instrument`

- Java-Doc Package `java.lang.instrument`:
  „*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.*“
- Wikipedia *Instrumentation* (computer programming):
  „*. . . instrumentation refers to an ability to monitor or measure the level of a product's performance, to diagnose errors and to write trace information. . . .*“
- Lässt sich aber für beliebiges nutzen (JPA-Provider, Code-Coverage, . . .)

Das Package `java.lang.instrument` (cont'd)

„An agent is deployed as a JAR file. An attribute in the JAR file manifest specifies the agent class which will be loaded to start the agent. For implementations that support a command-line interface, an agent is started by specifying an option on the command-line. 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.“

Wie geht's ?

- Zentral: der Agent

Wie geht's ?

- Zentral: der Agent
- Deployed als Jar-File

Wie geht's ?

- Zentral: der Agent
- Deployed als Jar-File
- Attribut im Manifest definiert die Agent-Klasse

Wie geht's ?

- Zentral: der Agent
- Deployed als Jar-File
- Attribut im Manifest definiert die Agent-Klasse
- Alternativen, um Agent zu starten

Wie geht's ?

- Zentral: der Agent
- Deployed als Jar-File
- Attribut im Manifest definiert die Agent-Klasse
- Alternativen, um Agent zu starten
  - Auf Kommandozeile bei VM-Start (zwingend erforderlich für Kommandozeilenimplementierungen)

Wie geht's ?

- Zentral: der Agent
- Deployed als Jar-File
- Attribut im Manifest definiert die Agent-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)

Was kann man damit machen?

- Allgemein: Instrumentieren, z.B. um zu

Was kann man damit machen?

- Allgemein: Instrumentieren, z.B. um zu
- Monitoren (Visual VM, . . .)

Was kann man damit machen?

- Allgemein: Instrumentieren, z.B. um zu
- Monitoren (Visual VM, . . .)
- Proxies bauen (JPA: Assoziationen, Automatic Dirty Checking,. . .)

Was kann man damit machen?

- Allgemein: Instrumentieren, z.B. um zu
- Monitoren (Visual VM, ...)
- Proxies bauen (JPA: Assoziationen, Automatic Dirty Checking,...)
- Code-Coverage-Werkzeuge bauen (EMMA, JaCoCo, Clover, ...)

Was kann man damit machen?

- Allgemein: Instrumentieren, z.B. um zu
- Monitoren (Visual VM, . . .)
- Proxies bauen (JPA: Assoziationen, Automatic Dirty Checking,. . .)
- Code-Coverage-Werkzeuge bauen (EMMA, JaCoCo, Clover, . . .)
- Aspektorientierte Programmierung

Was kann man damit machen?

- Allgemein: Instrumentieren, z.B. um zu
- Monitoren (Visual VM, . . .)
- Proxies bauen (JPA: Assoziationen, Automatic Dirty Checking,. . .)
- Code-Coverage-Werkzeuge bauen (EMMA, JaCoCo, Clover, . . .)
- Aspektorientierte Programmierung
- . . .

Was kann man damit machen?

- Allgemein: Instrumentieren, z.B. um zu
- Monitoren (Visual VM, ...)
- Proxies bauen (JPA: Assoziationen, Automatic Dirty Checking,...)
- Code-Coverage-Werkzeuge bauen (EMMA, JaCoCo, Clover, ...)
- Aspektorientierte Programmierung
- ...
- Allgemein: Sinnvolles Verhalten, das nicht im Code steht, (nachträglich und nur bei Bedarf) einbauen

Agentenstart über Komandozeile — Pre-Main

Agentenstart über Kommandozeile

- Syntax: `-javaagent:`*jarpath*`[=`*options*`]`

Agentenstart über Kommandozeile

- Syntax: `-javaagent:`*jarpath[=options]*
- Option mehrfach verwendbar, damit mehrere Agenten

Agentenstart über Kommandozeile

- Syntax: `-javaagent:`*jarpath[=options]*
- Option mehrfach verwendbar, damit mehrere Agenten
- Manifest des Agenten-Jars muss Attribut `Premain-Class` enthalten

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

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

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

Agentenstart über Kommandozeile

- Optional `agentmain()`-Methode zur Verwendung nach VM-Start

Agentenstart über Kommandozeile

- Optional `agentmain()`-Methode zur Verwendung nach VM-Start
- Falls Start über Kommandozeile, wird `agentmain()` nicht aufgerufen

Agentenstart über Kommandozeile

- Optional `agentmain()`-Methode zur Verwendung nach VM-Start
- Falls Start über Kommandozeile, wird `agentmain()` nicht aufgerufen
- Agent wird über System-Class-Loader geladen

Agentenstart über Kommandozeile

- Optional `agentmain()`-Methode zur Verwendung nach VM-Start
- Falls Start über Kommandozeile, wird `agentmain()` nicht aufgerufen
- Agent wird über System-Class-Loader geladen
- Jeder Agent bekommt seine Parameter über `agentArgs`-Parameter als *ein* String, d.h. Agent muss selbst parsen

Agentenstart über Kommandozeile

- Optional `agentmain()`-Methode zur Verwendung nach VM-Start
- Falls Start über Kommandozeile, wird `agentmain()` nicht aufgerufen
- Agent wird über System-Class-Loader geladen
- Jeder Agent bekommt seine Parameter über `agentArgs`-Parameter als *ein* String, d.h. Agent muss selbst parsen
- Falls Agent nicht geladen werden kann oder `premain()`-Methode nicht existert, wird VM beendet

Agentenstart über Kommandozeile

- Optional `agentmain()`-Methode zur Verwendung nach VM-Start
- Falls Start über Kommandozeile, wird `agentmain()` nicht aufgerufen
- Agent wird über System-Class-Loader geladen
- Jeder Agent bekommt seine Parameter über `agentArgs`-Parameter als *ein* String, d.h. Agent muss selbst parsen
- Falls Agent nicht geladen werden kann oder `premain()`-Methode nicht existert, wird VM beendet
- Exceptions der `premain()`-Methode führen ebenfalls zum Beenden der VM

Agentenstart über Attach-API — Agent-Main

Wiederholung 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 SDK, SAP SDK 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. For example, a profiler tool written in the Java Language might attach to a running application and load its profiler agent to profile the running application.* “

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

Agentenstart über Attach-API

- Manifest des Agenten-Jars muss Attribut `Agent-Class` enthalten

Agentenstart über Attach-API

- Manifest des Agenten-Jars muss Attribut `Agent-Class` enthalten
- Diese Agentenklasse muss `agentmain()`-Methode enthalten

Agentenstart über Attach-API

- Manifest des Agenten-Jars muss Attribut `Agent-Class` enthalten
- Diese Agentenklasse muss `agentmain()`-Methode enthalten
- Zwei mögliche Signaturen:

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

- Aufruf der zweiten Alternative nur, falls erste nicht existiert
- Im Manifest Attribut `Can-Redefine-Classes` auf `true`, falls redefiniert wird (neue Klassendefinition)
- Im Manifest Attribut `Can-Retransform-Classes` auf `true`, falls transformiert wird (Byte-Code-Enhancer)

Redefinition

Auszug aus Java-Doc für `redefineClasses()` I

- 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. Where the existing class file bytes are to be transformed (for example in bytecode instrumentation) retransformClasses should be used.
- This method operates on a set in order to allow interdependent changes to more than one class at the same time (a redefinition of class A can require a redefinition of class B).

Auszug aus Java-Doc für `redefineClasses()` II

- 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()` III

- 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.

Retransformation

Retransformation

- Retransformation auch möglich

Retransformation

- Retransformation auch möglich
- Dazu `Can-Retransform-Classes` im Manifest auf `true` setzen

Retransformation

- Retransformation auch möglich
- Dazu `Can-Retransform-Classes` im Manifest auf `true` setzen
- Transformer registrieren:
  `Instrumentation.addTransformer(ClassFileTransformer transformer)`

Retransformation

- Retransformation auch möglich
- Dazu `Can-Retransform-Classes` im Manifest auf `true` setzen
- Transformer registrieren:
  `Instrumentation.addTransformer(ClassFileTransformer transformer)`
- Und entsprechende Methode aufrufen:
  `Instrumentation.retransformClasses(Class<?>... classes)`

Retransformation

- Retransformation auch möglich
- Dazu `Can-Retransform-Classes` im Manifest auf `true` setzen
- Transformer registrieren:
  `Instrumentation.addTransformer(ClassFileTransformer transformer)`
- Und entsprechende Methode aufrufen:
  `Instrumentation.retransformClasses(Class<?>... classes)`
- Ja, so einfach ist es wirklich ;-)

Beispiele

Beispiel Monitoring: Aufrufhäufigkeit einer Methode

Beispiel Monitoring: Aufrufhäufigkeit einer 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 Hot Code Replacement/Patching

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
    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 m
        }
      }
    }
  }
}
```

Wie aktivieren?

```
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

Letztes Beispiel ???

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`)

Was man dazu benötigt . . .

- File-System-Watcher! Gibt's seit Java 7 im JDK
  (Interface `java.nio.file.WatchService`)
- Artikel Java aktuell 3/2015:
  Unbekannte Kostbarkeiten des SDK: Dateisystem-Überwachung

Was man dazu benötigt ...

- File-System-Watcher! Gibt's seit Java 7 im JDK
  (Interface `java.nio.file.WatchService`)
- Artikel Java aktuell 3/2015:
  Unbekannte Kostbarkeiten des SDK: Dateisystem-Überwachung
- Immer wenn `*.class`-Datei geändert wird, diese neu laden

Was man dazu benötigt . . .

- File-System-Watcher! Gibt's seit Java 7 im JDK
  (Interface `java.nio.file.WatchService`)
- Artikel Java aktuell 3/2015:
  Unbekannte Kostbarkeiten des SDK: Dateisystem-Überwachung
- Immer wenn `*.class`-Datei geändert wird, diese neu laden
- Das wars schon !

Was man dazu benötigt ...

- File-System-Watcher! Gibt's seit Java 7 im JDK
  (Interface `java.nio.file.WatchService`)
- Artikel Java aktuell 3/2015:
  Unbekannte Kostbarkeiten des SDK: Dateisystem-Überwachung
- Immer wenn `*.class`-Datei geändert wird, diese neu laden
- Das wars schon !
- Jetzt die Demo ...

Zusammenfassung

- Seit Java 5 kann man Klassen instrumentieren, d.h. Klassen verändern

Zusammenfassung

- Seit Java 5 kann man Klassen instrumentieren, d.h. Klassen verändern
- Entweder beim initialen Laden, aber auch bereits geladene Klassen

Zusammenfassung

- Seit Java 5 kann man Klassen instrumentieren, d.h. Klassen verändern
- Entweder beim initialen Laden, aber auch bereits geladene Klassen
- Was kann man damit machen?

Zusammenfassung

- Seit Java 5 kann man Klassen instrumentieren, d.h. Klassen verändern
- Entweder beim initialen Laden, aber auch bereits geladene Klassen
- Was kann man damit machen?
  - Sinnvolles Verhalten, das nicht im Code steht, (nachträglich und nur bei Bedarf) einbauen

Zusammenfassung

- Seit Java 5 kann man Klassen instrumentieren, d.h. Klassen verändern
- Entweder beim initialen Laden, aber auch bereits geladene Klassen
- Was kann man damit machen?
  - Sinnvolles Verhalten, das nicht im Code steht, (nachträglich und nur bei Bedarf) einbauen
  - Oder was Ihnen sonst so einfällt ...

Fragen und Anmerkungen