02-2014 | Sommer | www.ijug.eu

Java aktuell

Praxis. Wissen. Networking. Das Magazin für Entwickler

Aus der Community — für die Community

Java aktuell

Java blüht auf

Praxis
Performance richtig bewerten

Technologie
HTML5 und Java

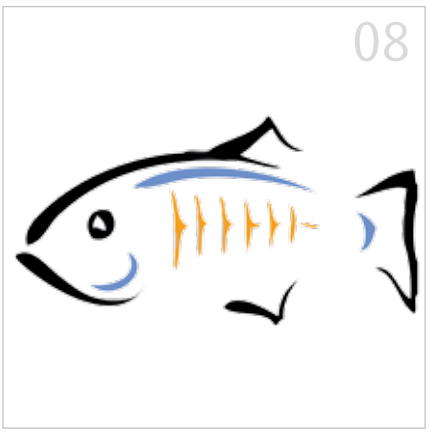
Java Server Faces
Umstieg auf 2.x

Oracle
Anwendungsentwicklung mit WebLogic

Sonderdruck

D: 4,90 EUR A: 5,60 EUR CH: 9,80 CHF Benelux: 5,80 EUR ISSN 2191-6977

iJUG Verbund



Die neue Oracle-Strategie zum GlassFish Server, Seite 8

HTML5 und Java-Technologien, Seite 18

Java und funktionale Programmierung - ein Widerspruch?, Seite 23

Eingebettete DSLs mit Clojure, Seite 26

Unbekannte Kostbarkeiten des SDK
Heute: Vor und nach der „main()“-Methode

Bernd Müller, Ostfalia

Das Java SDK enthält eine Reihe von Features, die wenig bekannt sind. Wären sie bekannt und würden sie verwendet, könnten Entwickler viel Arbeit und manchmal sogar zusätzliche Frameworks einsparen. Wir stellen in dieser Reihe solche Features des SDK vor: die unbekannten Kostbarkeiten.

Java-Programmieranfängern wird beigebracht, dass die „main()“-Methode der Einstiegspunkt eines Java-Programms ist und dass nach deren Ende die JVM beendet wird. Dies ist zwar prinzipiell richtig, jedoch gibt es die Option, sowohl vor als auch nach der „main()“-Methode beliebigen Code ausführen zu können. Der Artikel stellt diese beiden Optionen vor.

Instrumentierung

Unter Byte-Code-Instrumentierung, kurz „Instrumentierung“, versteht man das Ändern von Byte-Code, etwa um Methoden-Aufrufe zu loggen oder deren Ausführungszeit zu messen. Frameworks, die den vom Anwendungsentwickler geschriebenen Code ändern oder erweitern müssen, um eine bestimmte Funktionalität zu realisieren, machen ebenfalls von der Instrumentierung Gebrauch. Ein Beispiel dafür sind JPA-Provider, die etwa das Lazy-Loading von Assoziationen realisieren müssen. Dies ist bereits in [1] detailliert beschrieben.

Um die unterschiedlichen Ansätze der Instrumentierung zu vereinheitlichen, wurde in der Version 5 des SDK das Package „java.lang.instrument“ eingeführt. Zitat direkt aus der API-Doc des Package [2]: „Provides services that allow Java programming language agents to instrument programs running on the JVM.“

Das Ziel dieses Artikels ist jedoch nicht die Einführung in das Instrumentation-API, sondern die Möglichkeit, beliebigen Code vor der „main()“-Methode ausführen zu können. Der im API-Doc erwähnte „Agent“ ist hier das Mittel zum Zweck.

Java-Agenten

Ein Java-Agent ist eine Sammlung von Klassen, die in einem JAR gepackt sind. Die Klasse, deren Methoden vor der „main()“-Methode ausgeführt werden, ist die sogenannte „Pre Main“-Klasse. Die Methoden sind nicht beliebig wählbar, sondern durch Signaturen festgelegt (siehe Listing 1).

Die JVM versucht beim Programmstart zunächst, die erste und – falls diese nicht existiert – die zweite Methode auszuführen. Aber wie wird der JVM überhaupt der Java-Agent bekannt gemacht? Dies erfolgt zum einen über die Manifest-Datei des JAR, die für den Eintrag „Premain-Class“ die Pre-Main-Klasse enthält, etwa „Premain-Class: de.pdbm.SimpleAgent“. Außerdem muss der Aufruf der JVM mit der Option „javaagent“ erfolgen. Die Option enthält durch Doppelpunkt getrennt den Namen der JAR-Datei des Agenten, also beispielsweise „java -javaagent:/path-to/agent.jar de.pdbm.Main“.

Da wir nicht instrumentieren wollen, genügt für einen einfachen Test die zweite der oben genannten, überladenen „premain()“-Methoden. Listing 2 zeigt die Klasse „SimpleAgent“.

Bei Aufruf der JVM mit obiger Agentenoption erfolgt die Ausgabe der „premain()“-Methode vor dem Laden der Main-Klasse und dem Aufruf der dortigen „main()“-Methode. Der Parameter „agentArgs“ der Methode enthält den optionalen String, der nach dem Agenten-Jar angegeben werden kann, also etwa „java -javaagent:/path-to/agent.jar=“para1 para2“ de.pdbm.Main“.

Anwendung mit Sinn

Das vorangegangene Beispiel bewegte sich auf „Hello World“-Niveau. Wir werden deshalb etwas Sinnvolleres realisieren. Java-Agenten wurden eingeführt, um die Instrumentierung von Klassen zu ermögli-

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

Listing 1

```
package de.pdbm;

public class SimpleAgent {
      public static void premain(String agentArgs) {
             System.out.println(“Ausgabe vor ‘main()’”);
      }
}
```

Listing 2

```
public class Agent {

  static List<Class> classesBeforeMain;
  static List<Class> classesAfterMain;

  public static void premain(String agentArgs,
      final Instrumentation instrumentation) {
    Runtime.getRuntime().addShutdownHook(new Thread() {

      public void run() {
        classInfoAfterMain(instrumentation);
      }
    });
    classInfoBeforeMain(instrumentation);
  }
  private static void classInfoBeforeMain(Instrumentation instrumentation) {
    classesBeforeMain = Arrays.asList(instrumentation.getAllLoadedClasses());
    System.out.println("Anzahl geladener Klassen: " + classesBeforeMain.size());
  }
  private static void classInfoAfterMain(Instrumentation instrumentation) {
    classesAfterMain = Arrays.asList(instrumentation.getAllLoadedClasses());
    System.out.println("Anzahl geladener Klassen: " + classesAfterMain.size());
    for (Class clazz : classesAfterMain) {
      if (!classesBeforeMain.contains(clazz)) {
        System.out.println(clazz.getCanonicalName());
      }
    }
  }
}
```

Listing 3

chen. Dieser Artikel führt jedoch nicht das Instrumentation-API ein, sodass wir uns auf die folgende Anforderung festlegen: Zählung der geladenen Klassen vor und nach dem Laden der Main-Klasse und der Ausführung der „main()"-Methode. Wir sind uns darüber im Klaren, dass diese Anforderung in der täglichen Projektarbeit eher selten auftaucht und man sich eigene, sinnvollere Anforderungen überlegen kann.

Wir haben bereits gesehen, dass die „premain()"-Methode vor der „main()"-Methode ausgeführt wird. Wie kann man dann etwas nach der „main()"-Methode ausführen? Hier steht die Methode „addShutdownHook()" der Klasse „Runtime" zur Verfügung. Laut Dokumentation registriert die Methode in der JVM einen Shutdown-Hook. Dies ist ein initialisierter, aber noch nicht gestarteter Thread. Er wird gestartet, wenn die JVM heruntergefahren wird.

Obwohl wir nicht instrumentieren wollen, kommt uns der „Instrumentation"-Parameter der „premain()"-Methode beim Erreichen unseres Ziels zu Hilfe. Das Interface „Instrumentation" enthält unter anderem die Methode „getAllLoadedClasses()", die alle geladenen Klassen zurückgibt. Die Definition des Java-Agenten gestaltet sich damit recht einfach (siehe Listing 3).

System	Geladen vor „main()"	Geladen nach „main()"
Oracle Hot Spot 1.7.0_25	420	430
Oracle Hot Spot 1.7.0_40	421	429
OpenJDK 1.7.0_45	446	454
IBM 1.7.0 SR4	460	485
SAP 1.7.0_25	511	522

Tabelle

Die Methode „Instrumentation#getAllLoadedClasses()" liefert als Antwort ein Array von Klassen. Wir haben dieses Array in eine Liste konvertiert, um es bei der Berechnung der Differenz der beiden Mengen einfacher zu haben. Ansonsten ist die Klasse selbsterklärend.

Zu den geladenen Klassen gehören selbstverständlich die Main-Klasse, aber auch andere, nicht so offensichtliche Klassen wie etwa „java.lang.Void" oder „java.util.IdentityHashMap.KeySet". Erstaunlicherweise ist die Anzahl der geladenen Klassen von SDK zu SDK verschieden. Die Tabelle zeigt einige Ergebnisse.

Fazit

Mit einem Java-Agenten ist es möglich, beliebigen Code auszuführen, noch bevor die „main()"-Methode aufgerufen wird. Java-Agenten dienen primär der Instrumentierung von Klassen, lassen sich jedoch auch anderweitig verwenden. Mit einem Shutdown-Hook kann Code nach der „main()"-Methode ausgeführt werden.

Literatur

[1] Marc Steffens und Bernd Müller. Weaving, Instrumentation, Enhancement: Was ein JPA-Provider so alles macht. Java aktuell, Ausgabe 1/2012.

[2] http://docs.oracle.com/javase/7/docs/api/java/lang/instrument/package-summary.html

Bernd Müller
bernd.mueller@ostfalia.de

Bernd Müller ist Professor für Software-Technik an der Ostfalia. Er ist Autor des Buches „JavaServer Faces 2.0" und Mitglied in der Expertengruppe des JSR 344 (JSF 2.2).

www.ijug.eu

JETZT ABO BESTELLEN

Sichern Sie sich 4 Ausgaben für 18 EUR

Für Oracle-Anwender und Interessierte gibt es das Java aktuell Abonnement auch mit zusätzlich sechs Ausgaben im Jahr der Fachzeitschrift DOAG *News* und vier Ausgaben im Jahr Business News zusammen für 70 EUR. Weitere Informationen unter www.doag.org/shop/

FAXEN SIE DAS AUSGEFÜLLTE FORMULAR AN
0700 11 36 24 39

ODER BESTELLEN SIE ONLINE
go.ijug.eu/go/abo

Interessenverbund der Java User Groups e.V.
Tempelhofer Weg 64
12347 Berlin

Java aktuell

+++ AUSFÜLLEN +++ AUSSCHNEIDEN +++ ABSCHICKEN +++ AUSFÜLLEN +++ AUSSCHNEIDEN +++ ABSCHICKEN +++ AUSFÜLLEN

☐ **Ja**, ich bestelle das Abo Java aktuell – das IJUG-Magazin: 4 Ausgaben zu 18 EUR/Jahr
☐ **Ja**, ich bestelle den kostenfreien Newsletter: Java aktuell – der iJUG-Newsletter

ANSCHRIFT

Name, Vorname

Firma

Abteilung

Straße, Hausnummer

PLZ, Ort

GGF. ABWEICHENDE RECHNUNGSANSCHRIFT

Straße, Hausnummer

PLZ, Ort

E-Mail

Telefonnummer

X

Die allgemeinen Geschäftsbedingungen* erkenne ich an, Datum, Unterschrift

*Allgemeine Geschäftsbedingungen:
Zum Preis von 18 Euro (inkl. MwSt.) pro Kalenderjahr erhalten Sie vier Ausgaben der Zeitschrift "Java aktuell - das iJUG-Magazin" direkt nach Erscheinen per Post zugeschickt. Die Abonnementgebühr wird jeweils im Januar für ein Jahr fällig. Sie erhalten eine entsprechende Rechung. Abonnementverträge, die während eines Jahres beginnen, werden mit 4,90 Euro (inkl. MwSt.) je volles Quartal berechnet. Das Abonnement verlängert sich automatisch um ein weiteres Jahr, wenn es nicht bis zum 31. Oktober eines Jahres schriftlich gekündigt wird. Die Wiederrufsfrist beträgt 14 Tage ab Vertragserklärung in Textform ohne Angabe von Gründen.