02-2015 | Sommer | www.ijug.eu

Java aktuell

Praxis. Wissen. Networking. Das Magazin für Entwickler

Aus der Community – für die Community

Java aktuell

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

JAVA IST SUPER STARK

Programmierung
JavaScript für Java-Entwickler

Cloud Computing
Software-Architekturen in wolkigen Zeiten

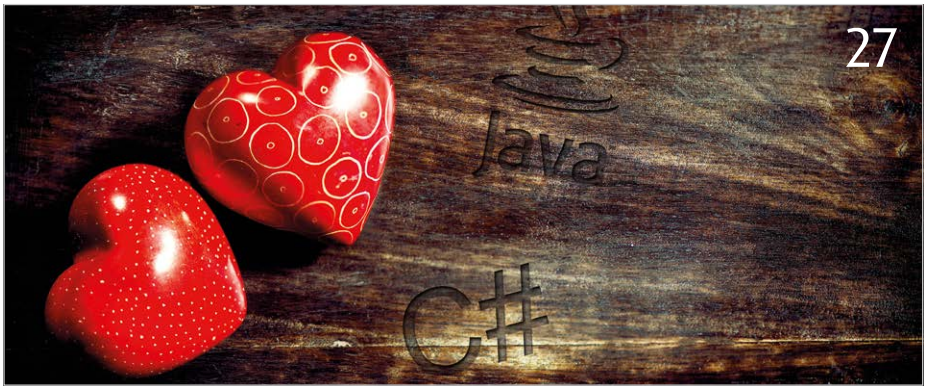
Applikationsserver
JBoss vs. WebLogic Server

JavaServer Faces
Interview mit Spec Lead Ed Burns

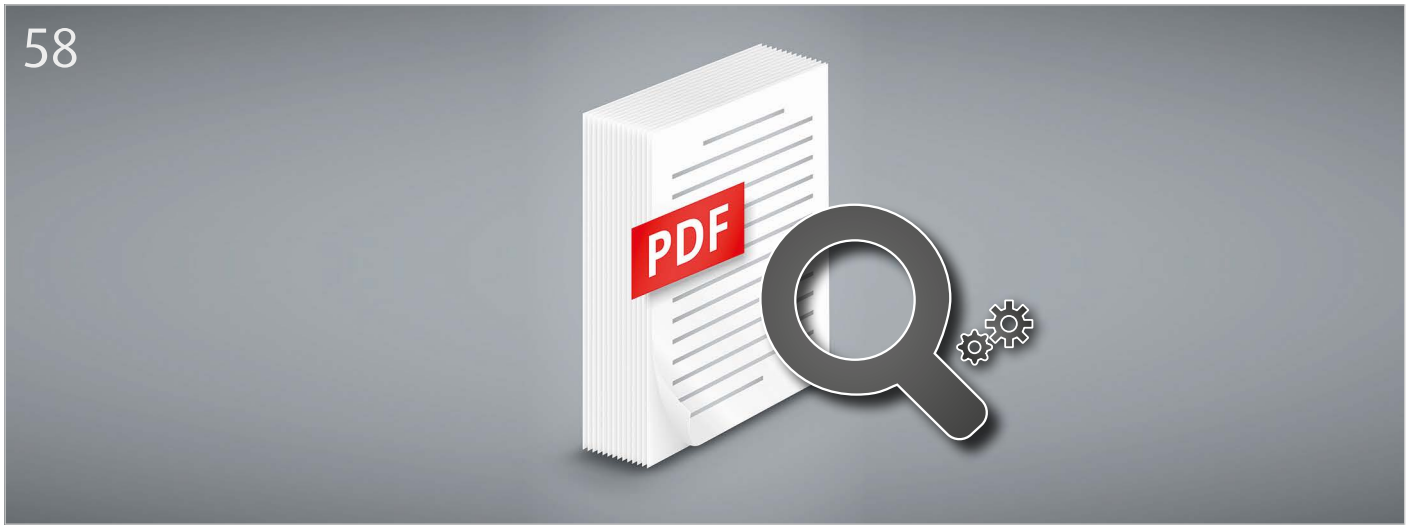
4 191978 304903 02

iJUG Verbund

In Xtend geschriebener Code ist meist kompakter und eleganter als sein Java-Äquivalent



Der Autor ist ein sehr großer Fan von Java, aber auch von C#. Er stellt einige interessante Seiten von C# vor



Die Korrektheit erzeugter PDF-Dokumente überprüfen – nicht manuell, sondern automatisiert

Unbekannte Kostbarkeiten des SDK

Heute: Bestimmung des Aufrufers

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 derartige Features des SDK vor: die unbekannten Kostbarkeiten.

Manchmal wäre die Information, wer eine Methode aufgerufen hat, sehr interessant, etwa bei der Erzeugung von Logging-Meldungen. Java besitzt keine einfache und intuitiv zugängliche Möglichkeit, dies zu realisieren. Man kann jedoch den Stack-Trace des aktuellen Threads verwenden, um an die gewünschte Information zu gelangen.

Motivation

Generell ist es für das Gesamtverständnis eines Systems wichtig zu wissen, welche Methode „A" welche Methode „B" aufruft, beziehungsweise alternativ, welche Methode „B" durch welche Methode „A" aufgerufen wird. Statische Analyse-Werkzeuge und damit insbesondere moderne IDEs unterstützen bei der Beantwortung beider Fragen. Die jeweilige Antwort basiert aber ausschließlich auf einer statischen Analyse und liefert für die zweite Fragestellung immer nur die potentiellen Kandidaten eines Aufrufs. Will man zur Laufzeit wissen, wer der Aufrufer einer bestimmten Methode ist, muss man etwas tiefer in die Trickkiste greifen.

Stack-Traces

Die Klassen „Error" und „Exception", beides Unterklassen von „Throwable", werden verwendet, um Ausnahmesituationen bei der Programmausführung anzuzeigen. Dazu wird eine Momentaufnahme des Aufruf-Stacks des ausführenden Threads in ein Array von Instanzen der Klasse „StackTraceElement" kopiert und in „Throwable" geschrieben. Die Momentaufnahme erfolgt zu dem Zeitpunkt, an dem die Ausnahmesituation aufgetreten ist. Über die Methode „getStackTrace()" des „Throwable" kann dann auf diesen Stack-Trace zugegriffen werden.

Um an einen Stack-Trace und damit eine Repräsentation des aktuellen Aufruf-Stacks zu kommen, muss aber nicht notwendigerweise ein „Error" oder eine „Exception" erzeugt werden. Die Klasse „Thread" besitzt ebenfalls eine Methode „getStackTrace()", die das Gewünschte liefert. Dazu das entsprechende Zitat aus dem Java-Doc der Methode „getStackTrace()" *[1]*: „Returns an array of stack trace elements representing the stack dump of this thread. This method will return a zero-length array if this thread has not started, has started but has not yet been scheduled to run by the system, or has terminated. If the returned array is of non-zero length then the first element of the array represents the top of the stack, which is the most recent method invocation in the sequence. The last element of the array represents the bottom of the stack, which is the least recent method invocation in the sequence. Some virtual machines may, under some circumstances, omit one or more stack frames from the stack trace. In the extreme case, a virtual machine that has no stack trace information concerning this thread is permitted to return a zero-length array from this method."

Das Beispiel in *Listing 1* macht sich dies zunutze. In der Main-Methode der Klasse „Caller" wird die Methode „foo()" der Klasse „Callee" aufgerufen. In dieser erfolgt der Zugriff auf den aktuellen Stack-Trace mit der Methode „getStackTrace()". Der Rückgabewert ist ein Array von „StackTraceElement"-Objekten. Wie dem Java-Doc zu entnehmen, repräsentiert das erste Array-Element den obersten Eintrag des Stacks und das letzte Array-Element den ersten Methodenaufruf in dieser Aufruf-Folge. *Listing 2* zeigt die Ausgabe des Beispiels. Die Zeilennummern entsprechen nicht dem Code

```
public class Caller {

  public static void main(String[] args) {
     new Callee().foo();
  }


}

public class Callee {

  public void foo() {

     StackTraceElement[] stackTraceElements =
        Thread.currentThread().getStackTrace();
     System.out.println("0: " + stackTraceElements[0]);
     System.out.println("1: " + stackTraceElements[1]);
     System.out.println("2: " + stackTraceElements[2]);
  }

}
```

Listing 1

```
0: java.lang.Thread.getStackTrace(Thread.java:1552)
1: de.pdbm.Callee.foo(Callee.java:6)
2: de.pdbm.Caller.main(Caller.java:6)
```

Listing 2

aus *Listing 1*, da dort die Import-Anweisungen weggelassen wurden. Im Beispiel enthält das Array auch nur die dargestellten drei Elemente. In einem etwas realistischeren Programm ist das Array deutlich größer. Der interessante Wert ist im dritten Array-Element (Index 2) gespeichert. Will man beispielsweise für das Logging entsprechende Informationen aufbereiten, so können die folgenden Methoden der Klasse „StackTraceElement" verwendet werden:

- String getClassName()
- String getFileName()
- String getMethodName()
- int getLineNumber()

Stack-Traces revisited

Das eigentliche Ziel ist erreicht, nämlich die Bestimmung des Aufrufers einer bestimmten Methode. Es soll aber nicht unerwähnt bleiben, dass das dargestellte Verfahren einen gewissen Aufwand bedeutet und somit bei großzügiger Verwendung zu Performance-Problemen führen kann. Generell ist die Erzeugung eines (großen) Stack-Traces mit deutlichem Aufwand verbunden *[2]*, da der ausführende Thread unterbrochen, das Array erzeugt und unter Verwendung vieler String-Konvertierungsmethoden befüllt werden muss.

Wie fast immer bei teuren Operationen, bietet die VM auch hier Kommandozeilen-Optionen für das Fein-Tuning an. So kann beispielsweise die maximale Größe des Strack-Traces, also die Länge des Arrays, mit „-XX:MaxJavaStackTraceDepth=<value>" auf den Wert „value" beschränkt werden. Der Default ist „1024". „-XX:-StackTraceInThrowable" unterbindet die Erzeugung des Stack-Trace ganz, es wird also ein Array mit null Elementen erzeugt. Dasselbe gilt ebenfalls für die erste Option mit einem Wert von „0". In beiden Fällen liefert das Beispielprogramm eine „ArrayIndexOutOfBoundsException".

Fazit

Java kennt keinen direkten Weg, um in einer Methode den Aufrufer dieser Methode zur Laufzeit zu bestimmen. Mit einem explizit erzeugten Stack-Trace innerhalb der Methode ist dies jedoch leicht möglich. Da der damit verbundene Aufwand relativ hoch ist, empfiehlt sich dieses Vorgehen jedoch nur in Ausnahmefällen.

Literatur/Videos

[1] http://docs.oracle.com/javase/8/docs/api/java/lang/Thread.html

[2] Scott Oaks, Java Performance – The Definitive Guide, O'Reilly, 2014

Bernd Müller

bernd.mueller@ostfalia.de

Bernd Müller ist Professor für Software-Technik an der Ostfalia (Hochschule Braunschweig/Wolfenbüttel). Er ist Autor mehrere Java-EE-Bücher, Sprecher auf nationalen und internationalen Konferenzen und engagiert sich in der JUG Ostfalen sowie im iJUG.

http://ja.ijug.eu/15/2/16

DEVOXX 4KIDS in Deutschland

Am 6. Juni 2015 wird es in Hamburg eine Devoxx4Kids geben. Dahinter stehen weltweit organisierte Veranstaltungen, bei denen Kinder Computerspiele entwickeln, Roboter programmieren und eine Einführung in Elektronik bekommen. Ziel ist es zu zeigen, dass es möglich ist, kreativ mit dem Computer zu arbeiten. Im Vorfeld der Veranstaltung sind Quadrocopter, Tinkerforge und NAOs geplant.

Bereits am 5. Mai 2015 findet in Hamburg bei der JUG HH ein Vorstellungsevent mit dem Karlsruher Devoxx4Kids-Team statt.

Weitere Informationen unter „http://www.devoxx4kids.org/deutschland"