Java aktuell

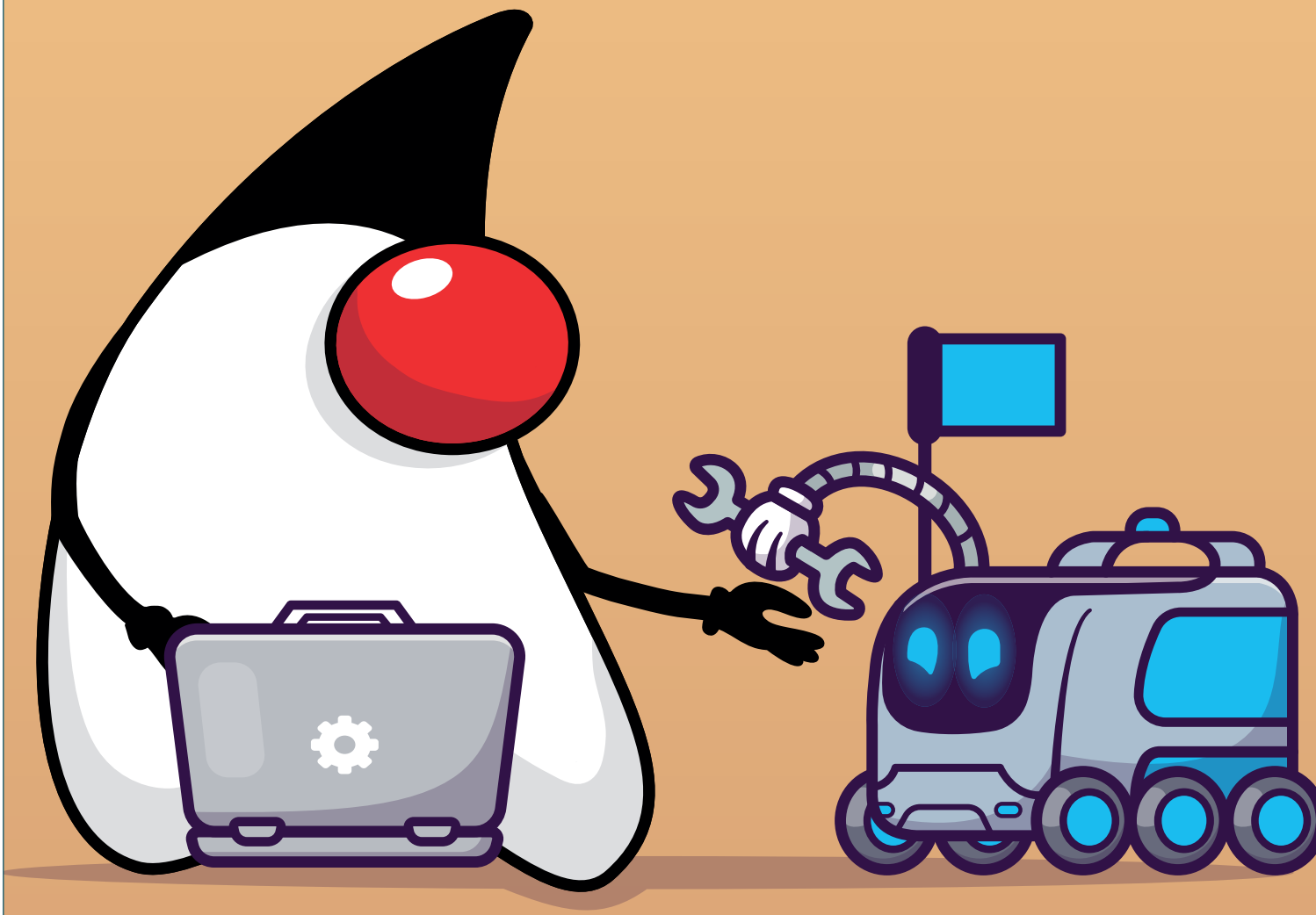
iJUG Verbund
www.ijug.eu

Tools
Shell, GitHub Actions mit Dependabot

Testing
Playwright Java, Test-First-Strategie

Cloud Native
Cloud-Native-Applikationen mit Spring Boot 3 und GraalVM

TOOLS



Ausgabe 03/2024
Java aktuell
Praxis. Wissen. Networking. Das Magazin für Entwickler
D: 4,90 EUR | CH: 9,80 CHF | A: 5,60 EUR | Benelux: 5,80 EUR
4 191978 304903 03

Unbekannte Kostbarkeiten des SDK "Heute: Der ToolProvider"

Bernd Müller – Ostfalia

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

Unsere heutige Ausgabe der unbekannten Kostbarkeiten widmet sich dem *ToolProvider*. Ein Interface, das es auf standardisiertem Weg erlaubt, bestehende Kommandozeilenwerkzeuge des JDK, aber auch eigene Werkzeuge, über ein öffentliches API aus einem laufenden Programm heraus aufzurufen. Man erspart sich also eine zweite laufende JVM.

Der ToolProvider

Das Interface *ToolProvider* im Package `java.util.spi` beschreibt sich selbst als: *„An interface for command-line tools to provide a way to be invoked without necessarily starting a new VM."* Doch welche Werkzeuge sind gemeint? Der erfahrene Java-Entwickler weiß, dass das OpenJDK in seinem `bin`-Verzeichnis eine ganze Reihe außerordentlich hilfreicher Werkzeuge bereitstellt. Neben `javac` und `java` etwa auch `jar`, `javap`, `jcmd`, `jconsole`, `jps` und viele, viele mehr.

Im OpenJDK Version 21 zählen wir 29 solcher Werkzeuge. Alle Werkzeuge sind ausführbare Binärprogramme (keine Shell-Skripte), die allerdings eine JVM starten. Über `Runtime#exec()` können Betriebssystemprogramme aus einem Java-Programm heraus gestartet werden. Wird dieser Weg genutzt, um eines der genannten Werkzeuge zu starten, laufen auf dem System jedoch zwei JVMs. Das Interface ToolProvider wurde geschaffen, um dies zu vermeiden und mit nur einer JVM auszukommen.

Das Interface im Überblick

Das Interface ToolProvider wurde mit Java 9 eingeführt und basiert auf dem durch die Klasse `ServiceLoader` definierten Mechanismus zum Laden von Diensten. Das Interface umfasst vier (überladene) Methoden, die in *Abbildung 1* dargestellt sind. Die Default-Methode `description()` liefert ein leeres Optional zurück. Die Default-Methode `run()` ruft die normale `run()`-Methode auf, die eine Instanz des Werkzeugs aufruft. Die ersten beiden Parameter sind für die Ausgaben und Fehlermeldungen des Werkzeugs gedacht. Der VarArgs-Parameter repräsentiert die Parameter der entsprechenden Kommandozeilenversion. Dies ist schon alles. Wir werden das API später an Beispielen verdeutlichen.

Stand der Dinge

Im OpenJDK Version 21 sind acht ToolProvider implementiert, die in *Tabelle 1* aufgeführt sind. Wir haben auch ihre jeweilige Implementierung angegeben, da einige so alt sind, dass ihre Package-Namen interessanterweise noch auf Sun basieren. Bis auf jpackage wurden alle ToolProvider, wie das Interface selbst, mit Java 9 eingeführt. jpackage wurde mit Java 14 nachgeschoben. Das Werkzeug war aber auch erst mit Java 14 released worden.

Der Quellcode des OpenJDK enthält auch die Klasse `JshellToolProvider`. Diese wird aber (im Augenblick noch) nicht als `ToolProvider` verfügbar gemacht. Es gab 2016 eine kurze Diskussion auf der OpenJDK-Mailing-Liste, ob `jshell` verfügbar gemacht werden sollte. Dies führte allerdings nicht zur Veröffentlichung von `jshell`.

Ein Beispiel

Nachdem wir nun wissen, was es mit dem `ToolProvider` auf sich hat, wollen wir ein kleines Beispiel entwickeln. Das Werkzeug `javadoc` wird von uns in der Regel über Maven oder Gradle aufgerufen. Als Kommandozeilenwerkzeug hat es eine ganze Reihe von Optionen beziehungsweise Parametern. Zu den wichtigsten gehören `-d` (Destination) für die Angabe des Zielverzeichnisses sowie `-sourcepath` für die Angabe des Quellcode-Verzeichnisses. Letztendlich ist das Package anzugeben, für das das *Javadoc* erzeugt werden soll. Dies sollen auch unsere Beispielparameter sein. Die Main-Methode in *Listing 1* zeigt beispielhaft die Verwendung. Die Methode `findFirst()` erwartet den Namen des Kommandozeilenwerkzeugs. Der VarArg-Parameter der `run()`-Methode ist identisch zum `args`-Parameter der Main-Methode des Kommandozeilenwerkzeugs. Im Beispiel ist dies zunächst das Parameterpaar zur Angabe des Zielverzeichnisses, gefolgt vom Parameterpaar des Quellcode-Verzeichnisses. Diese beiden Parameterpaare könnten in der Reihenfolge vertauscht werden. Der letzte Parameter ist der Package-Name, für den das *Javadoc* zu erstellen ist *(siehe Listing 1)*.

ToolProvider	Implementierende Klasse
jar	sun.tools.jar.JarToolProvider
javac	com.sun.tools.javac.main.JavacToolProvider
javadoc	jdk.javadoc.internal.tool.JavadocToolProvide
javap	com.sun.tools.javap.Main$JavapToolProvider
jdeps	com.sun.tools.jdeps.Main$JDepsToolProvider
jlink	jdk.tools.jlink.internal.Main$JlinkToolProvider
jmod	jdk.tools.jmod.Main$JmodToolProvider
jpackage	jdk.jpackage.internal.JPackageToolProvider

Tabelle 1: Liste der implementierten ToolProvider

Method Summary

All Methods | Static Methods | Instance Methods | Abstract Methods | Default Methods

Modifier and Type	Method	Description
default Optional<String>	description()	Returns a short description of the tool, or an empty Optional if no description is available.
static Optional<ToolProvider>	findFirst(String name)	Returns the first instance of a ToolProvider with the given name, as loaded by ServiceLoader using the system class loader.
String	name()	Returns the name of this tool provider.
default int	run(PrintStream out, PrintStream err, String... args)	Runs an instance of the tool, returning zero for a successful run.
int	run(PrintWriter out, PrintWriter err, String... args)	Runs an instance of the tool, returning zero for a successful run.

Abbildung 1

```
public static void main(String[] args) {
     Optional<ToolProvider> javadoc = ToolProvider.findFirst("javadoc");
     String cwd = System.getProperty("user.dir");
     String src = cwd + "/src/main/java";
     String apidoc = cwd + "/target/site/apidocs";
     javadoc.get().run(System.out, System.err,
        "-d", apidoc, "-sourcepath", src, "de.pdbm.toolprovider");
}
```

Listing 1

```
public class TrickyToolProvider implements ToolProvider {

     @Override
     public String name() {
        return "tricky";
     }

     @Override
     public Optional<String> description() {
        return Optional.of("Runs the very tricky Tool");
     }

     @Override
     public int run(PrintWriter out, PrintWriter err, String... args) {
        return Tricky.execute(args, System.out, System.err);
     }

}
```

Listing 2

Eigene ToolProvider

ToolProvider ist ein allgemein verwendbares Interface, sodass auch eigene Implementierungen erstellt werden können. Wenn Sie also ein in Java geschriebenes Werkzeug haben, das Sie sowohl von der Kommandozeile als auch aus einem laufenden Programm heraus über ein definiertes API aufrufen wollen, ist der ToolProvider das Mittel der Wahl. Nachdem wir bereits in *Abbildung 1* gesehen haben, dass das ToolProvider-API sehr schlank ist, zeigt *Listing 2* eine beispielhafte Implementierung.

Das Werkzeug selbst wird durch die Klasse `Tricky` realisiert, die wir hier aber nicht zum Besten geben. Interessanter ist der Aufruf dieser Klasse über den `ToolProvider`. Der Aufruf erfolgt in der `run()`-Methode in *Listing 2*. Die `execute()`-Methode haben wir strukturell an die im OpenJDK vorhandenen `ToolProvider`-Implementierungen angelehnt. Die Java-Main-Methode halt als einzigen Parameter ein String-Array. Die Kommunikation mit der Umgebung erfolgt über die Variablen `in`, `out` und `err` der Klasse `System`. Wenn wir über den `ToolProvider` ein derartiges Werkzeug aus einer laufenden JVM aufrufen, müssen die Standardausgabe sowie die Fehlerausgabe übergeben werden. Dies erfolgt über die Methode `execute()`, die wir, wie gesagt, an die entsprechenden `ToolProvider`-Implementierungen des OpenJDK angelehnt haben.

`ToolProvider` werden über den `ServiceLoader`-Mechanismus geladen. Daher muss noch im Verzeichnis `META-INF/services` eine Datei `java.util.spi.Tool.Provider` angelegt werden, die als einzige Zeile den Namen der `Tricky`-Klasse enthält. Bei Bedarf können Einzelheiten im *Javadoc* der Klasse `ServiceLoader` nachgelesen werden. Das ist auch schon alles. Aufrufe von der Kommandozeile oder auch aus einem laufenden Programm heraus sind nun möglich und der Aufruf aus einem laufenden Programm heraus startet keine zweite JVM *(siehe Listing 2)*.

Falls es Sie in den Fingern juckt und Sie gerne alles selbst ausprobieren wollen, enthält unser Projekt auf GitHub *[1]* den kompletten Quellcode der Beispiele dieses Artikels.

Zusammenfassung

Mit Java 9 wurde das Interface *ToolProvider* eingeführt. Es erlaubt den einfachen Aufruf von Kommandozeilenwerkzeugen, ohne eine weitere JVM starten zu müssen. Eine Reihe von Werkzeugen, die mit dem JDK oder einem JRE ausgeliefert werden, sind hierüber aufrufbar. Die Implementierung eigener `ToolProvider` ist ebenfalls leicht möglich.

Referenzen

[1] *https://github.com/BerndMuller/toolprovider*

Bernd Müller
Ostfalia
bernd.mueller@ostfalia.de

Nach seinem Studium der Informatik und der Promotion arbeitete Bernd Müller für die IBM und die HDI Informationssysteme. Er ist Professor, Geschäftsführer, Autor mehrerer Bücher zu den Themen JSF und JPA, sowie Speaker auf nationalen und internationalen Konferenzen. Er engagiert sich im iJUG und speziell in der JUG Ostfalen.

Mitglieder des iJUG

01 BED-Con e.V.
02 Clojure User Group Düsseldorf
03 DOAG e.V.
04 EuregJUG Maas-Rhine
05 JUG Augsburg
06 JUG Berlin-Brandenburg
07 JUG Bremen
08 JUG Bielefeld
09 JUG Bonn
10 JUG Darmstadt
11 JUG Deutschland e.V.
12 JUG Dortmund
13 JUG Düsseldorf rheinjug
14 JUG Erlangen-Nürnberg
15 JUG Freiburg
16 JUG Goldstadt
17 JUG Görlitz
18 JUG Hannover
19 JUG Hessen
20 JUG HH
21 JUG Ingolstadt e.V.
22 JUG Kaiserslautern
23 JUG Karlsruhe
24 JUG Köln
25 Kotlin User Group Düsseldorf
26 JUG Mainz
27 JUG Mannheim
28 JUG München
29 JUG Münster
30 JUG Oberland
31 JUG Ostfalen
32 JUG Paderborn
33 JUG Saxony
34 JUG Stuttgart e.V.
35 JUG Switzerland
36 JSUG
37 Lightweight JUG München
38 SUG Deutschland e.V.
39 JUG Thüringen
40 JUG Saarland
41 JUG Duisburg
42 JUG Frankfurt

iJUG Verbund
www.ijug.eu

Impressum

Java aktuell wird vom Interessenverbund der Java User Groups e.V. (iJUG) (Tempelhofer Weg 64, 12347 Berlin, *www.ijug.eu*) herausgegeben. Es ist das User-Magazin rund um die Programmiersprache Java im Raum Deutschland, Österreich und Schweiz. Es ist unabhängig von Oracle und vertritt weder direkt noch indirekt deren wirtschaftliche Interessen. Vielmehr vertritt es die Interessen der Anwender an den Themen rund um die Java-Produkte, fördert den Wissensaustausch zwischen den Lesern und informiert über neue Produkte und Technologien.

Java aktuell wird verlegt von der DOAG Dienstleistungen GmbH, Tempelhofer Weg 64, 12347 Berlin, Deutschland, gesetzlich vertreten durch den Geschäftsführer Fried Saacke, deren Unternehmensgegenstand Vereinsmanagement, Veranstaltungsorganisation und Publishing ist.

DOAG e.V. hält 100 Prozent der Stammeinlage der DOAG Dienstleistungen GmbH. DOAG e.V. wird gesetzlich durch den Vorstand vertreten; Vorsitzender: Björn Bröhl. DOAG e.V. informiert kompetent über alle Oracle-Themen, setzt sich für die Interessen der Mitglieder ein und führen einen konstruktiv-kritischen Dialog mit Oracle.

Redaktion:
Sitz: DOAG Dienstleistungen GmbH
ViSdP: Fried Saacke
Redaktionsleitung: Lisa Damerow
Kontakt: redaktion@ijug.eu

Redaktionsbeirat:
Andreas Badelt, Marcus Fihlon, Markus Karg, Manuel Mauky, Bernd Müller, Benjamin Nothdurft, Daniel van Ross, Bennet Schulz

Titel, Gestaltung und Satz:
Alexander Kermas,
DOAG Dienstleistungen GmbH

Bildnachweis:
Titel: Bild © Designed by catalyststuff
https://freepik.com
S. 10 + 11: Bild © Designed by vectorpocket
https://freepik.com
S. 14: Bild © Sloth McSloth
https://stock.adobe.com
S. 16 + 17: Bild © ant
https://stock.adobe.com
S. 26 + 27: Bild © IBEX.Media
https://stock.adobe.com
S. 34 + 35: Bild © semisatch
https://stock.adobe.com
S. 42 + 43: Bild © Designed by upklyak
https://freepik.com
S. 50: Bild © Open Elements
https://open-elements.com
S. 56 + 57: Bild © Designed by stories
https://freepik.com
S. 64 + 65: Bild © Designed by macrovector
https://freepik.com
S. 70 + 71: Bild © PCH.Vector
https://stock.adobe.com
S. 76 + 77: Bild © Designed by freepik
https://freepik.com
S. 80 + 81: Bild © Designed by macrovector
https://freepik.com

Anzeigen:
DOAG Dienstleistungen GmbH
Kontakt: sponsoring@doag.org
Mediadaten und Preise:
www.doag.org/go/mediadaten

Druck:
WIRmachenDRUCK GmbH
www.wir-machen-druck.de

Inserentenverzeichnis