GRUNDLAGEN • PRAXIS • ARCHITEKTUR • AGILE • INNOVATION

3 / 2018

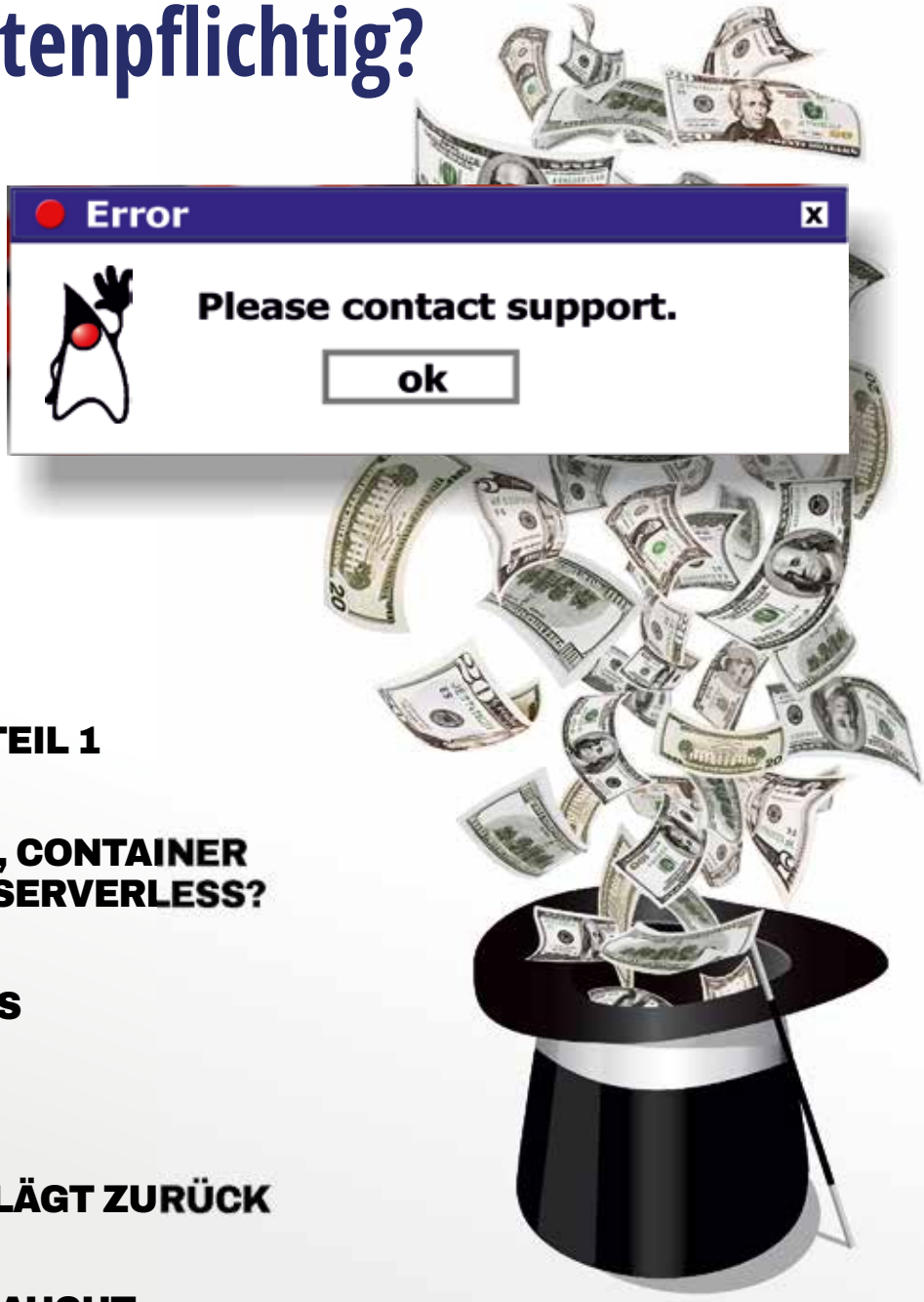
JAVAPRO

Magazin für professionelle Java Entwicklung in der Praxis

#JAVAPRO

Oracle ändert Lizenz- & Support Modell !

Java jetzt kostenpflichtig?



www.java-pro.de

www.java-pro.de

#JAVAPRO #CoreJava #JDK11 #Java

Neues in Java 11

Oracle hat Anfang 2018 für Java einen völlig neuen Release-Zyklus eingeführt und dazu jetzt ein neues Lizenz- und Support-Modell bekannt gegeben. Im Rahmen dieses Artikels werfen wir einen Blick auf die Neuerungen, die neuen Features der neuesten Java Version 11 und auf die bereits im Frühjahr 2019 erwartete Java Version 12.

Autor:

Falk Sippach hat 20 Jahre Erfahrung mit Java und ist bei der Mannheimer Firma OIO Orientation in Objects GmbH als Trainer, Software-Entwickler/Architekt tätig. Er publiziert regelmäßig in Blogs, Fachartikeln und auf Konferenzen. In seiner Wahlheimat Darmstadt organisiert er mit anderen die örtliche Java User Group.

Twitter: @sippsack

Seit Frühjahr 2018 werden Java-Entwickler mit halbjährlichen Major-Releases beglückt. In diesem Zuge wurde im vergangenen September nun bereits Java 11[1] veröffentlicht. Darüber hinaus gab es auch politisch einige Änderungen. Denn mit Java 11 führt Oracle eine neue Lizenz- und Support-Strategie ein. Freie Updates für die letzte LTS-Version 8 (Long-Term-Support) wird es ab Januar 2019 nicht mehr geben. Zudem ist ab Java 11 das Oracle JDK nur noch in der Entwicklung kostenlos nutzbar, in Produktion muss ein Support-Vertrag[2] mit Oracle gegen entsprechende Gebühren abgeschlossen werden. Das Ziel dieser Lizenzgebühr

ist die Sicherstellung der wirtschaftlichen Weiterentwicklung des JDK durch das Oracle-Engineering-Team.

Wer weiterhin auf eine freie JDK-Version aufbauen möchte, kann das mittlerweile zum Oracle JDK binär kompatible OpenJDK einsetzen, welches als Open-Source unter der GNU General Public License v2 (with the Classpath Exception – GPL v2 + CPE) veröffentlicht ist. Allerdings wird Oracle das OpenJDK immer jeweils nur ein halbes Jahr mit freien Updates versorgen. Entweder aktualisiert man dann alle sechs Monate die eingesetzte Java-Version oder man nutzt alternative Distributionen. Das AdoptOpenJDK-Projekt[3] will beispielsweise die LTS-Versionen 8 und 11 noch bis zu vier Jahre mit kostenlosen Updates versorgen. Auch für Kunden von diversen Linux-Distributionen (z. B. Red Hat Enterprise Linux[4]) wird das OpenJDK 8 noch mindestens vier Jahre im Rahmen der Betriebssystem-Support-Verträge kostenlos mit Updates unterstützt. Für Anwender der Java-Plattform gibt es also zahlreiche Optionen zwischen kommerziellen Lösungen von Oracle, Azul, IBM[5] usw. und weiterhin freien Lösungen wie dem OpenJDK oder dem AdoptOpenJDK-Projekt. Es sind somit ausreichend Alternativen vorhanden, die darauf warten ausprobiert zu werden.

Neue Features in Java 11

Insgesamt wurden für das OpenJDK 11 etwa 2400 Tickets geschlossen[6]. Fast 80 Prozent davon hat Oracle bearbeitet. An den restlichen 20 Prozent waren aber viele andere Firmen wie SAP, Red Hat, Google, IBM usw. beteiligt. Ein Großteil der Neuerungen wurde im Rahmen der folgenden Java Enhancement Proposals (JEPs) umgesetzt:

- 181: Nest-Based Access-Control
- 309: Dynamic Class-File Constants
- 315: Improve Aarch64 Intrinsics
- 318: Epsilon: A No-Op Garbage-Collector
- 320: Remove the Java-EE and CORBA-Modules
- 321: http-Client (Standard)
- 323: Local-Variable-Syntax for Lambda-Parameters
- 324: Key-Agreement with Curve25519 and Curve448
- 327: Unicode 10
- 328: Flight-Recorder
- 329: ChaCha20 and Poly1305 Cryptographic-Algorithms
- 330: Launch Single-File Source-Code Programs
- 331: Low-Overhead Heap-Profiling
- 332: Transport-Layer-Security (TLS) 1.3
- 333: ZGC: A scalable Low-Latency Garbage-Collector
- 335: Deprecate the Nashorn JavaScript-Engine
- 336: Deprecate the Pack200-Tools and -API

Aus Entwicklersicht sind nur einige wenige Punkte wirklich relevant. So wurde im JEP 323 eine Erweiterung der in Java 10 eingeführten Local-Variable Type-Inference umgesetzt. Typinferenz ist das Schlussfolgern der Datentypen aus den restlichen Angaben des Quellcodes und den Typisierungsregeln heraus. Das spart Schreibarbeit und bläht den Quellcode nicht unnötig auf, wodurch sich wiederum die Lesbarkeit erhöht.

Seit Java 10 können lokale Variablen mit dem Schlüsselwort **var** folgendermaßen deklariert werden:

```
// Funktioniert seit Java 10

var zahl = 5; // int
var string = "Hello World"; // String
var objekt = BigDecimal.ONE; // BigDecimal
```

Neu in Java 11 ist, dass man nun auch Lambda-Parameter mit **var** deklarieren kann. Das mag auf den ersten Blick nicht sonderlich sinnvoll erscheinen, da man den Typ von Lambda-Parametern sowieso weglassen und über die Typinferenz ermitteln lassen kann. Nützlich wird die Erweiterung aber für die Verwendung von Type-Annotations wie **@Nonnull** und **@Nullable**.

```
// Inference von Lambda Parametern
Consumer<String> printer = (var s) -> System.out.println(s); //
statt s -> System.out.println(s);

// aber keine Mischung von "var" und deklarierten Typen möglich
// BiConsumer<String, String> printer = (var s1, String s2)
-> System.out.println(s1 + " " + s2);

// Nützlich für Type Annotations
BiConsumer<String, String> printer = (@Nonnull var s1, @Nullable
var s2) ->
    System.out.println(s1 + (s2 == null ? "" : " " + s2));
```

Die nächste interessante Neuerung ist die Standardisierung der bisher noch experimentellen neuen HTTP Client API, die mit JDK 9 eingeführt und in JDK 10 aktualisiert wurde (JEP 110). Neben HTTP/1.1 werden nun auch HTTP/2, WebSockets, HTTP/2 Server-Push, synchrone und asynchrone Aufrufe und Reactive-Streams unterstützt. Garniert mit einem gut lesbaren Fluent-Interface wird die Verwendung von anderen HTTP-Clients (z. B. von Apache) dann in Zukunft voraussichtlich obsolet sein.

```
HttpClient client = HttpClient.newHttpClient();
HttpRequest request = HttpRequest.newBuilder()
      .uri(URI.create("http://openjdk.java.net/"))
      .build();
client.sendAsync(request, asString())
      .thenApply(HttpResponse::body)
      .thenAccept(System.out::println)
      .join();
```

Durch den JEP 330 (Launch Single-File Source-Code-Programs) können jetzt Klassen gestartet werden, die noch nicht kompiliert wurden. Programme mit einer einzigen Datei sind heutzutage beim Schreiben kleiner Hilfsprogramme üblich und insbesondere

die Domäne von Skriptsprachen. Nun kann man sich auch in Java die unnötige Arbeit sparen und das verringert zugleich die Einstiegshürde für Java-Neulinge.

```
# java HelloWorld.java
// statt
# javac HelloWorld.java
# java -cp . hello.World
```

Auf unixoiden Betriebssystemen können Java-Dateien als Shebang-Files sogar direkt ausgeführt werden:

```
#!/path/to/java --source version

[...]
```

Der Aufruf erfolgt dann als ausführbare Datei ähnlich wie bei Shell-Skripten:

```
# ./HelloWorld.java
```

Weitere erwähnenswerte Änderungen sind die Unterstützung des Unicode-10-Standards und die Integration des bisher nur mit einer kommerziellen Lizenz verwendbaren Profiling-Tools Mission Control und Flight Recorder in das OpenJDK (sie wurden bisher nur mit dem Oracle JDK ausgeliefert). Das Ziel des Flight Recorders ist das möglichst effiziente Aufzeichnen von Anwendungsdaten, um bei Problemen die Java-Anwendung und die JVM analysieren zu können. Unsinnig scheint zunächst der JEP 318 (Epsilon: A No-Op Garbage-Collector) zu sein. Dabei handelt es sich um einen neuen Garbage-Collector, der aber gar keine Garbage-Collection durchführt (No-Operation). Interessant ist dieses Verhalten aber für Serverless-Functions für das Oracle Projekt fn. Da es sich hier um sehr kurz laufende Java-Anwendungen handelt, wäre ein zwischenzeitlicher Garbage-Collector-Lauf eher kontraproduktiv und sinnlos, wenn die Anwendung kurz darauf sowieso wieder beendet wird.

API-Änderungen

An der Java-Klassenbibliothek gab es natürlich auch unzählige kleine Änderungen. Besonders viel hat sich bei String und Character getan:

```
|  Welcome to JShell -- Version 11
|  For an introduction type: /help intro
// Unicode zu String
jshell> Character.toString(100)
$1 ==> "d"
jshell> Character.toString(66)
$2 ==> "B"
// Zeichen mit Faktor multiplizieren
jshell> "-".repeat(20)
$3 ==> "--------------------"

// Enthält ein Text keine Zeichen (höchstens Leerzeichen)?
jshell> String msg = "hello"
msg ==> "hello"
jshell> msg.isBlank()
$5 ==> false
jshell> String msg = "  "
msg ==> "  "
jshell> msg.isBlank()
$7 ==> true

// Abschneiden von führenden oder nachgelagerten Leerzeichen
jshell> " hello world ".strip()
$8 ==> "hello world"
jshell> "hello world    ".strip()
$9 ==> "hello world"
jshell> "hello world    ".stripTrailing()
$10 ==> "hello world"
jshell> "        hello world    ".stripLeading()
$11 ==> "hello world    "
jshell> "    ".strip()
$12 ==> ""

// Texte zeilenweise verarbeiten
jshell> String content =  "this is a multiline content\nMostly
obtained from some file\rwhich we will break into lines\r\nusing
the new api"
content ==> "this is a multiline content\nMostly obtained fro ...
ines\r\nusing the new api"
jshell> content.lines().forEach(System.out::println)
this is a multiline content
Mostly obtained from some file
which we will break into lines
using the new api
```

Was wurde aus Java entfernt?

Die Ankündigungen in Form von Deprecations in den Versionen 9 und 10 wurden nun in Java 11 in die Tat umgesetzt. So wurden im Rahmen des JEP 320 diverse Java-Enterprise-Packages aus Java SE entfernt, dazu zählen JAX-WS (XML basierte SOAP-Webservices inklusive den Tools wsgen und wsimport), JAXB (Java-XML-Binding inklusive den Tools schemagen und xjc), JAF (Java-Beans-Activation-Framework), Common-Annotations (@PostConstruct, @Resource, ...), CORBA und JTA (Java-Transaction-API).

Neu ist auch, dass das Oracle JDK kein JavaFX mehr enthalten wird, was mit dem OpenJDK übrigens noch nie ausgeliefert wurde. Stattdessen wird JavaFX über OpenJFX[7] als separater Download angeboten und kann wie jede andere Bibliothek in beliebigen Java-Anwendungen verwendet werden. Neben JavaFX wird auch der Support für Applets und Java Web-Start eingestellt. Die Open-Source-Community plant allerdings bereits Nachfolgeprojekte[8]. Wenn man im Moment noch Java Web-Start nutzen möchte, muss man zunächst beim Oracle JDK 8 bleiben und entweder ohne Sicherheits-Updates leben oder für den

kommerziellen Support ab 2019 Geld ausgeben.
Als Deprecated markiert wurde in Java 11 zudem die JavaScript-Engine Nashorn. Es ist davon auszugehen, dass sie in zukünftigen Java-Versionen verschwinden wird. Nashorn hat sich nie so richtig als serverseitige JavaScript-Implementierung gegenüber Node.js durchsetzen können. Und mit der GraalVM geht Oracle mittlerweile alternative Wege, um andere Programmiersprachen nativ auf der JVM auszuführen.

Übrigens wird es ab Java 11 die Java-Laufzeitumgebung (JRE) nur noch in der Server-Variante und nicht mehr für Desktops geben. Allerdings kann man für Desktop-Anwendungen mit dem Modulsystem und dem Werkzeug jlink mittlerweile selbst sehr einfach in der Größe angepasste Laufzeitumgebungen erstellen.

Fazit und Ausblick

Java 11 finalisiert angefangene Arbeiten aus den beiden vorangegangenen Versionen, damit es als LTS-Release für die nächsten drei Jahre gut gewappnet ist. Dazu wurden auch einige alte Zöpfe abgeschnitten und zum Beispiel JavaFX und diverse Java-EE-Packages aus dem JDK entfernt.

Mit Java 12 steht bereits das nächste Major-Release in den Startlöchern und wird voraussichtlich im März 2019 erscheinen[9]. Die Liste der Neuerungen wächst noch und enthielt Anfang November 2018 bereits die folgenden JEPs:

- JEP 325: Switch-Expressions
- JEP 326: Raw-String-Literals
- JEP 340: One AArch64 Port, Not Two
- JEP 341: Default CDS-Archives

Während es sich bei JEP 340 um Aufräumarbeiten an der ARM-Portierung des JDK und bei JEP 341 um Erweiterungen zur Verbesserung der Performance durch Class-Data-Sharing handelt, sind für Entwickler insbesondere die ersten beiden Punkte interessant. Beide Sprach-Features stammen aus sogenannten Inkubator-Projekten (Amber[10], Valhalla[11], Loom[12]), in denen kleinere, die Entwicklerproduktivität steigernde Sprach- und VM-Features ausgebrütet und dann als Java-Enhancement-Proposals (JEPs) akzeptiert werden.

Switch-Statements können demnach in Zukunft auch als Expression verwendet werden, die das Ergebnis des entsprechenden Case-Zweigs direkt zurückliefert.

```
int numLetters = switch (day) {
    case MONDAY, FRIDAY, SUNDAY -> 6;
    case TUESDAY                -> 7;
    case THURSDAY, SATURDAY     -> 8;
    case WEDNESDAY              -> 9;
};
```

Mit Raw-String-Literals wird Java endlich die Möglichkeit bekommen, mehrzeilige Zeichenketten zu definieren, die zusätzlich Escape-Sequenzen ignorieren. Damit lässt sich viel einfacher mit regulären Ausdrücken und Windows-Dateipfaden umgehen. Einzig das Ersetzen von Variablen (String-Interpolation) ist im Moment noch nicht geplant, ein Feature, welches alternative Sprachen wie Groovy, Ruby und JavaScript schon länger unterstützt.

```
Runtime.getRuntime().exec(`"C:\Program Files\foo" bar`);
// statt
Runtime.getRuntime().exec("\"C:\\Program Files\\foo\" bar");

String script1 = `function hello() {
  print('"Hello World"');
}

hello();`;

// statt
String script2 = "function hello() {\n" +
                 "   print(\'\"Hello World\"\');\n" +
                 "}\n" +
                 "\n" +
                 "hello();\n";
```

Das sieht schon sehr vielversprechend aus und wir dürfen gespannt sein, was es bis zum Feature-Freeze beim Start der Rampdown-Phase 1 Mitte Dezember noch in das JDK 12 schaffen wird.

Das nächste LTS-Release (Java 17) wird erst im Herbst 2021 herauskommen. Bis dahin sollte in Produktion entweder weiterhin auf Java 8 oder das aktuelle Java 11 gesetzt werden. Die Änderungen der nächsten Zwischen-Releases kann man natürlich leicht verfolgen, immerhin sind die halbjährlichen Versionen gut überschaubar. Die nächsten Jahre scheinen für Java-Entwickler spannend zu bleiben und man darf sich auf regelmäßige Verbesserungen an der Sprache freuen.

Quellen:

1 JDK 11 Projektseite: http://jdk.java.net/11/
2 Java SE Support-Verträge: https://bit.ly/2Brc3Ay
3 AdoptOpenJDK: https://adoptopenjdk.net/
4 RHEL OpenJDK: https://red.ht/2adYXY6
5 IBM SDK: https://ibm.co/2r286MB
6 OpenJDK Community Zusammenarbeit: https://bit.ly/2QcIwm5
7 OpenJFX: http://openjdk.java.net/projects/openjfx/
8 Opensource Java Webstart: https://dev.karakun.com/webstart/
9 OpenJDK 12 Projektseite: https://bit.ly/2E5OLT1
10 Project Amber: http://openjdk.java.net/projects/amber/
11 Project Valhalla: http://openjdk.java.net/projects/valhalla/
12 Project Loom: http://openjdk.java.net/projects/loom/