JAVA 16

Oracle heeft sinds Java SE 10 een halfjaarlijks release-schema geïnstalleerd, in tegenstelling tot de globaal driejaarlijkse releases daarvoor. De deadline voor nieuwe features in Java is fixed gemaakt. Dit betekent dat het aantal JEP's (JDK Enhancement Proposals) dat geïmplementeerd kan worden per release variabel is geworden. Ook zal eens in de drie jaar een zogenaamde LTS-versie (Long-Term Support) worden gemaakt.

Ivo Woltring *is Software Architect en Codesmith bij Ordina JTech en houdt zich graag bezig met nieuwe ontwikkelingen in de softwarewereld.*

Java SE 11 is zo'n LTS-versie en zal door Oracle nog tot september 2023 ondersteund worden. Dit betekent dat Java 17 ook weer een LTS wordt. Die gaat dit jaar rond september gereleased worden! De versies die tussen LTS-versies uitkomen krijgen alleen support tot een nieuwe versie uitkomt, dus zo ook Java SE 16.
Java SE 16 is in maart 2021 uitgekomen. Dit artikel beschrijft welke nieuwe features in Java SE 16 geïmplementeerd zijn. Tijdens het schrijven van dit artikel was deze laatste releasedatum nog niet bereikt, dus er kunnen dus nog (kleine) scope-wijzigingen plaats hebben gevonden. In Java 16 staan 17 features (JEP's) gepland.
Om met wat features van Java SE 16 te kunnen spelen, zonder de early access echt te hoeven installeren, is alle code in dit artikel uitgeprobeerd binnen een Docker-container met OpenJDK 16 (listing 1).
Dit artikel is in twee hoofdsecties onderverdeeld. Het eerste deel zal gaan over nieuwe standaard-features. Het tweede deel zal ingaan op preview- en incubator-features. In versie 16 worden geen features uitgefaseerd. Bij elke feature zal het JEP-nummer vermeld worden.

{ NIEUWE STANDAARD-FEATURES }

347: Enable C++14 Language Features

C++14 features zullen geactiveerd worden in de JDK-sourcecode-builds. Dit betekent dat de Java-sourcecode gecompileerd zal worden met de C++14 (-std=c++14) opties, geactiveerd voor alle platformen (Windows, Linux, macOS en AIX). De JEP geeft ook richtlijnen over welke van de C++14-features gebruikt mogen/kunnen worden in de HotSpot-code. Tot JDK 15 zijn de features die gebruik maakten van C++-code gelimiteerd geweest tot de C++98/03-standaard. In Java SE 11 is de code geüpdatet zodat hij nieuwe versies aankon, maar er werd nog geen gebruik van gemaakt.

L1

```
$ docker run -it --rm \
      -v $(pwd)/src/main/java:/src \
      openjdk:16 /bin/bash
$ cd /src
```

357: Migrate from Mercurial to Git en 369: Migrate to GitHub

De OpenJDK Community sourcecode-repositories zijn gemigreerd van Mercurial naar Git. De drie hoofdredenen voor het migreren naar Git zijn:
1. De grootte van de versiebeheer metadata
2. Beschikbare tools
3. Beschikbare hosting

Initiële tests hebben aangetoond dat de grootte van de JDK-repository in Git ongeveer 300 MB is en in hg 1,2 GB. Daarnaast wordt Git door bijna alle teksteditors en IDE's geweldig ondersteund. Gekozen is voor het hosten op GitHub: https://github.com/openjdk [3].

376: ZGC: Concurrent Thread-Stack Processing

De Z Garbage Collector is ontworpen om GC-pauzes en schaalbaarheidsproblemen in HotSpot weg te halen. In Java SE 15 is het een standaard product-feature geworden. Java SE 16 worden een paar optimalisaties doorgevoerd om de GC nog sneller te maken.

380: Unix-Domain Socket Channels

De SocketChannel en ServerSocketChannel API's leveren 'blocking'- en 'multiplexed non-blocking'-toegang voor tcp/ip-sockets. In Java SE 16 worden deze classes extended zodat ook Unix domain (AF_UNIX) sockets en server sockets ondersteund worden. Unix-domain sockets wordt gebruikt voor communicatie tussen processen (Inter-process communication - IPC) op dezelfde host.

L2

```
public class JEP390 {
    public static void main(String[] args) {
        Double d = 20.0;
        synchronized (d) { /* do stuff */ }
// javac warning & HotSpot warning
        Object o = d;
        synchronized (o) { /* do stuff */ }
// HotSpot warning
    }
}
$ java JEP390.java
JEP390.java:4: warning: [synchronization] at-
tempt to synchronize on an instance of a va-
lue-based class
        synchronized (d) { /* do stuff */ }
// javac warning & HotSpot warning
        ^
1 warning
```

L3

```
public class JEP394 {
    public void print(Object o) {
        if (o instanceof String bs &&
            bs.length() > 5){
            System.out.println(bs);
        } else {
            System.out.println("Shorter");
        }
    }
  public static void main(String[] args) {
    JEP394 jep394 = new JEP394();
    var site = "IvoNet.nl";
    jep394.print(site);
  }
}
```

L4

```
public class JEP395 {
    static record Point(int x, int y) {};
    public static void main(String[] args) {
        final Point p1 = new Point(1, 2);
        final Point p2 = new Point(1, 2);
        if (p1.equals(p2)) {
            System.out.println(p1);
        }
    }
}
$ java JEP395.java
Point[x=1, y=2]
```

386: Alpine Linux Port

In deze JEP wordt de JDK geport naar Alpine Linux en andere Linux-distributies die musl gebruiken als hun belangrijkste C-Library. Dit gebeurt voor zowel de x64- als de AArch64-architecturen. Musl (uitgesproken in het Engels als "muscle") is een implementatie van de standard-library zoals beschreven is in de standaarden ISO C en POSIX.

388: Windows/AArch64 Port

Ports de JDK naar Windows/AArch64. Met het op de markt komen van nieuwe hardware voor eindgebruikers en AArch64 (ARM64)-servers is de vraag naar deze port erg toegenomen.

387: Elastic Metaspace

De metaspace van de HotSpot, waar class-metadata worden gemanaged, is redelijk berucht om het hoge geheugengebruik. Voor de meeste programma's is dit geen probleem, maar in sommige gevallen kon het niet (tijdig) teruggeven van ongebruikt geheugen leiden tot excessief geheugengebruik. Deze JEP zorgt ervoor dat ongebruikte metaspace tijdig weer teruggegeven wordt aan het Operating Systeem (OS).

390: Warnings for Value-Based Classes

Het Project Valhalla [4] is druk bezig met een grote aanvulling op het Java-programmeermodel in de vorm van primitive classes. Het is de bedoeling dat dit soort classes zich vergelijkbaar gaan gedragen als echte primitives. Deze classes worden ook wel Value-based classes genoemd. Denk hierbij aan classes als 'java.lang.Integer', 'java.lang.Double', etc. Alle classes die in aanmerking komen voor deze aanpassing zijn met `@ValueBased` geannoteerd.

De primitieve wrapper-classes in de JDK die geannoteerd zijn met `@jdk.internal.ValueBased` zullen voortaan een waarschuwing genereren als je er op probeert te synchroniseren. Deze zullen namelijk in de toekomst niet meer werken. Zie listing 2.

392: Packaging Tool

De Packaging Tool is een command-line tool (jpackage) om platformspecifieke packages te maken van Java-applicaties. Dit betekent deb- of rpm-bestanden op Linux, pkg- of dmg-bestanden op macOS en msi- of exe-bestanden op Windows. In het artikel over Java 14 is hier een uitgebreid codevoorbeeld van gegeven [2].

394: Pattern Matching for instanceof

Na twee incubatorfases is deze feature nu onderdeel geworden van de taal. De type-test kan gevolgd worden door een binding-variabele die daarna meteen gebruikt kan worden. In het voorbeeld van listing 3 is dat de bs-variabele. De bs-variabele wordt in de if geïntroduceerd en in hetzelfde statement al gebruikt (`bs.length()`) en in de verdere methode.

L5

```
#include <stdio.h>
void helloworld() {
    printf("Hello Java Magazine readers!\n");
}
$ microdnf install gcc
$ gcc -c -fpic helloworld.c
$ gcc -shared -o helloworld.so helloworld.o
```

395: Records

Records bieden een compacte syntaxis voor het declareren van klassen die houders zijn voor onveranderlijke ('immutable') gegevens. Ook deze feature is nu standaard na twee incubatiefases. Zie listing 4.

396: Strongly Encapsulate JDK Internals by Default

Deze JEP zorgt ervoor dat alle JDK-interne classes nu op de '--illegal-access'-optie standaard de deny (niet toegestaan) in plaats van de permit (toegestaan) krijgen. Alleen kritische API's zoals de 'sun.misc.Unsafe' zullen nog toegankelijk blijven.

{ PREVIEW EN INCUBATOR FEATURES }

338: Vector API (Incubator)

Dit is de eerste iteratie van de 'jdk.incubator.vector'-module. Deze module stelt je in staat om vectorberekeningen uit te drukken die runtime gecompileerd kunnen worden naar optimale vector-hardware-instructies voor ondersteunde CPU-architecturen en daardoor dus de optimale performance voor dit soort berekeningen haalt. Zie referentie [2] voor voorbeeldcode.

389: Foreign Linker API (Incubator)

Deze API levert een volledig Java-manier om native code aan te roepen. Het zal JNI gaan vervangen. Zie listing 5 voor een voorbeeld.

Nu we er een shared library van gemaakt hebben kunnen we het gaan aanroepen vanuit Java. In die code zoeken we de library en dan de method. Deze wordt aan een eigen method handle gebonden door een FunctionDescriptor en een MethodType aan te geven waarna de method aangeroepen kan worden. Omdat het nog een Incubator-feature is moet de module expliciet meegegeven worden en aangezet (`--add-modules jdk.incubator.foreign -Dforeign.restricted=permit`) Zie listing 6.

393: Foreign-Memory Access API (Third Incubator)

Deze API is al in Java SE 14 aangeboden als incubator-module en is in dat artikel uitgebreid behandeld geweest [2]. In Java SE 15 zijn er nog aanpassingen gedaan op basis van bevindingen en ook in Java SE 16 worden verfijningen aangebracht, waardoor een derde Incubatie-fase ingegaan wordt.

L6

```
import jdk.incubator.foreign.CLinker;
import jdk.incubator.foreign.FunctionDescriptor;
import jdk.incubator.foreign.LibraryLookup;
import java.lang.invoke.MethodType;
import java.nio.file.Path;
public class JEP389 {
  public static void main(String[] args)  {
    var lib = LibraryLookup
      .ofPath(Path.of("/src/helloworld.so"));
    var sym = lib.lookup("helloworld").get();
    var fd = FunctionDescriptor.ofVoid();
    var mt = MethodType.methodType(Void.TYPE);
    var mh = CLinker.getInstance()
      .downcallHandle(sym.address(), mt, fd);
    mh.invokeExact();
  }
}
$ java --add-modules jdk.incubator.foreign
-Dforeign.restricted=permit JEP389.java
WARNING: Using incubator modules: jdk.incubator.
foreign
warning: using incubating module(s): jdk.incuba-
tor.foreign
1 warning
Hello Java Magazine readers!
```

397: Sealed Classes (Second Preview)

Deze JEP biedt Java een manier om implementaties en overerving van classes, interfaces en records te beperken tot wat expliciet toegestaan is. In Java SE 15 is deze ter preview aangeboden en in het bijbehorende artikel behandeld [2]. Deze preview bouwt hierop voort.

{ CONCLUSIE }

In deze versie is niks verwijderd, maar wel een heleboel toegevoegd. Voor het dagelijkse werk van de meeste Software Engineers zullen Records en de Pattern matching-JEPS waarschijnlijk het meest geanticipeerd zijn. De volgende versie (17) wordt weer een LTS-versie. De kinderziektes van Java SE 9 zijn lang voorbij en sinds die versie zijn vele mooie dingen toegevoegd aan de taal. De meeste belangrijke frameworks zijn al lang over naar tenminste Java SE 11 LTS en voor bedrijven die nu nog Java SE 1.8 (8) gebruiken is er zo langzamerhand geen fatsoenlijk excuus meer om die versie te handhaven. <

REFERENTIES

1 http://openjdk.java.net/projects/jdk/16/
2 http://ivo2u.nl/oz
3 https://github.com/openjdk
4 https://openjdk.java.net/projects/valhalla/