Java 12

8 nieuwe features

Op het moment van schrijven van dit artikel is de countdown voor de Java SE 12 release nog gaande en zijn er alleen preview builds beschikbaar. Op 19 maart 2019 is het eindelijk zover en zal de final release beschikbaar komen. Voor Java 12 zijn 8 nieuwe features (JEP's) geïmplementeerd. In dit artikel zullen deze features beschreven worden.

Oracle heeft sinds Java SE 10 een 6 maanden releaseschema aangekondigd. De deadline is dus fixed gemaakt. Dit betekent dat het aantal JDK Enhancement Proposals (JEP) dat geïmplementeerd kan worden per release variabel is geworden. Eens in de drie versies zal een zogenaamde Long-term support (LTS) versie worden gemaakt. Java SE 11 is zo'n LTS versie en zal door Oracle nog tot september 2023 ondersteund worden. De versies die tussen LTS versies uitkomen krijgen alleen support tot een nieuwe versie uit komt. Laten we kijken of er in versie 12 features zitten die het interessant maken om te upgraden...

1 JEP 189: Shenandoah: Garbage Collector met korte pauze tijd

In de laatste paar releases van Java is veel aandacht aan de Garbage Collector (GC) besteed en JEP 189 is er ook een voor. Deze JEP voegt een nieuwe garbage collector toe aan de JVM met de naam Shenandoah. Shenandoah is een algoritme dat vooral geschikt is voor toepassingen waarvoor reactiesnelheid en voorspelbare korte pauzes belangrijk zijn. De korte pauze tijd wordt verkregen door meer garbage collection werk gelijktijdig (concurrent) te doen met het draaiende Java programma. Om deze GC te activeren moet je deze opties meegeven: `-XX:+UnlockExperimentalVMOptions -XX:+UseShenandoahGC`.

De Shenandoah GC is niet die ene GC die alle GC's zal gaan regeren. De Shenandoah GC legt de nadruk op responsiviteit er zijn andere GC-algoritmen die prioriteit geven aan doorvoer of geheugen gebruik. De gebruiker zal dus zelf moeten kiezen waar ze de nadruk willen leggen.

2 JEP 230: Microbenchmark Suite

JEP 230 voegt een standaardreeks micro-benchmarks toe aan de JDK-broncode en maakt het voor JDK-ontwikkelaars gemakkelijk om bestaande micro-benchmarks uit te voeren en nieuwe benchmarks te maken. Een micro-benchmark meet de prestaties van een klein codefragment.

3 JEP 325: Switch Expressions (Preview)

Ik denk dat de meeste ontwikkelaars deze JEP het interessantst vinden van alle JEP's in deze release. Een vernieuwde switch statement die gemoderniseerd is. Met twee voorbeelden zal dit geïllustreerd worden.
Het voorbeeld in **listing 1** laat zien dat je meerdere case labels kan hebben en dat alleen wat rechts van de pijl staat wordt uitgevoerd en ook als return waarde geldt.
Aan de rechterkant van de pijl kan ook een code blok staan zoals getoond in **listing 2**. Het break statement kan nu ook een argument bevatten dat als return waarde geldt.
De nieuwe switch statement is een "Preview" feature. Dat betekent dat het niet per default geactiveerd is binnen de JDK. Om deze te activeren zal de parameter `--enable-preview` meegegeven moeten worden.
De code voorbeelden in dit artikel zijn uitgeprobeerd met behulp van een OpenJDK 12 Docker image. Het commando om een jshell te starten met alle nieuwe features geactiveerd staat in **listing 3**.

4 JEP 334: JVM Constants API

Deze JEP Introduceert een API voor het modelleren van constanten in de constante pool. Deze API is nuttig voor programma's en libraries die zich bezighouden met bytecode-instructies en het manipuleren van class bestanden.

5 JEP 340: One AArch64 Port, Not Two

Niet heel veel detail is gegeven over deze JEP, maar het komt er op neer dat van de voorheen

Ivo Woltring is Software Architect en Codesmith bij JTech Ordina.

twee beschikbare ARM ports er nog maar 1 over blijft, zodat contributeurs zich nog maar op 1 64-bits ARM implementatie hoeven te concentreren.

6 JEP 341: Default CDS Archives

CDS staat voor Class-Data Sharing. Met CDS kan een reeks klassen worden voorverwerkt in een gedeeld archiefbestand. Waarom is dit nodig? De JVM doet veel magie tijdens het laden van klassen. De JVM interpreteert de klasse, slaat deze op in een interne structuur, voert een aantal controles uit, lost de symbolen op en verbindt deze, etc. Dan is de klasse klaar om te werken. Al deze stappen vergen enige tijd. Bovendien laadt elke JVM-instantie gewoonlijk dezelfde systeem klassen zoals bijvoorbeeld String, Integer en URLConnection die standaard in Java zijn opgenomen. Al deze klassen hebben geheugen nodig. Wanneer we een gedeeld archief hebben dat voorverwerkte klassen bevat, kan deze tijdens runtime worden toegewezen aan het geheugen. Als gevolg hiervan kan het de opstarttijd en geheugen voetafdruk verminderen. Met de implementatie van deze JEP zullen CDS archives standaard gegenereerd worden.

7 JEP 344: Abortable Mixed Collections for G1

Garbage Collection kan een grote invloed hebben op de performance van je applicatie. De G1 Garbage Collector kan geconfigureerd worden om niet langer dan een opgegeven tijd te pauzeren (`–XX:MaxGCPauseTimeMillis`). De G1 zal dan proberen dit doel te halen, maar dit is niet altijd mogelijk.
Om te voldoen aan het door de gebruiker geleverde pauze tijddoel, zorgt deze JEP ervoor dat de G1 Garbage Collector het garbage collection-proces afbreekt door de set met afval verzamelde gebieden te splitsen in verplichte en optionele onderdelen en de garbage collection af te breken van het optionele onderdeel als het doel voor de pauzetijd anders niet wordt bereikt. Het maakt de G1 dus weer een stukje slimmer en efficiënter.

8 JEP 346: Promptly Return Unused Committed Memory from G1

Oracle verbetert de G1 Garbage Collector zodat het automatisch Java heap geheugen naar het besturingssysteem retourneert als het niet actief is.
Tot deze JEP geïmplementeerd werd kon het voorkomen dat de G1 lang geheugen vast hield. Meestal alleen maar terug leverend na een volledige Garbage Collection en omdat G1 een volledige Garbage Collection probeert te voorkomen, kon dat dus lang duren.

```
public String dag(int day) {
    return switch (day) {
        case 1, 2, 3, 4, 5  -> "weekdag"; //geldt ook als return waarde
        case 6,7 -> "weekend";
        default -> "onbekend";
    } ;
}
```

Listing 1

```
public boolean isWeekend(String day) {
    return switch (day) {
        case "maandag", "dinsdag", "woensdag", "donderdag", "vrijdag" -> false;
        case "zaterdag","zondag" -> true;
        default -> {
            if (day.startsWith("z")) {
                System.out.println("Lijkt op weekend");
                break true;  // de break kan nu een argument bevatten
            }

            System.out.printf("Onbekende dag: %s%n", day);
            break false; // de break kan nu een argument bevatten
        }
    } ;
}
```

Listing 2

```
docker run \
       -it \
       --rm \
       --name jdk12 \
       -e JAVA_OPTS="-XX:+UnlockExperimentalVMOptions -XX:+UseShenandoahGC" \
       openjdk:12-oracle \
       jshell --enable-preview
```

Listing 3

Conclusie

In eerste instantie zat JEP 326 (Raw String Literals) ook in deze release, maar heeft de release niet gehaald. Door het niet halen van deze JEP is de enige voor ontwikkelaars interessante JEP die van de switch en deze is niet standaard geactiveerd. Het zal me dus niks verbazen als upgraden van Java 11 naar Java 12 voor velen niet hoog op de prioriteitenlijst zal komen te staan. ■

LATEN WE EENS KIJKEN OF IN VERSIE 12 FEATURES ZITTEN DIE HET INTERESSANT MAKEN OM TE UPGRADEN

REFERENTIES

https://openjdk.java.net/projects/jdk/12/
https://www.oracle.com/technetwork/java/java-se-support-roadmap.html
https://hub.docker.com/_/openjdk
https://openjdk.java.net/projects/code-tools/jmh/
http://ivo2u.nl/oa