Java Roadmap

Hoe ziet de toekomst van Java eruit?

De Java SE 10 Development Kit is pas net uitgebracht door Oracle en Java SE 11 staat alweer bijna klaar. Sinds Java SE 10 heeft Oracle een 6 maanden releaseschema aangekondigd voor de standaardeditie van Java. In zijn blog heeft Mark Reinhold een nieuw voorstel gedaan voor versionering. Hij stelt voor om vanaf het moment van de 6 maanden release (vanaf versie 10) over te gaan op een YY.MM release versie. Dit zou Java SE 11 versie 18.9 maken, omdat het uit moet komen in september 2018. Er is een hoop te doen rondom deze manier van versioneren en dit kan dus nog veranderen. Nu zie je vaak de volgende notatie staan "Java SE 11 (18.9)". De tijd zal het leren.

Tot nu toe zijn pas een paar nieuwe features aangekondigd voor Java SE 11 (18.9). Ook gaan een paar features verloren in deze versie, zoals CORBA en Java EE (recentelijk hernoemd naar Jakarta EE, hierover later meer) modules. Ook zal JavaFX verwijderd worden.

Op dit moment zitten we op Java SE 10 (18.3) en de laatste versie van de Enterprise Edition is uitgebracht (Java EE 8).

Java SE 11 zal een zogenaamde LTS (Long Term Support) release zijn. Java SE 11 zal de hoogste level van ondersteuning genieten van Oracle tot september 2023 en aanvullende ondersteuning, zoals patches en beveiligingswaarschuwingen, tot 2026. Het is de planning van Oracle om elke drie jaar een LTS release uit te brengen. Dat betekent dus dat elke feature release, zoals Java SE 9 en Java SE 10, maar een half jaar ondersteuning van Oracle krijgen, omdat dit short-term releases zijn.

In JDK 11 zijn tot nu toe drie nieuwe features gepland, maar er worden er wellicht meer verwacht. Dit wordt bevestigd op de JEP site voor Java SE 11.

De Epsilon "No-Op" Garbage collector, zoals beschreven in JEP 318, handelt wel geheugenallocatie af, maar implementeert geen terugvorderingsmechanisme. Zodra de Java Heap "vol" is, zal de JVM stoppen. De motivatie achter deze triviale no-op garbage collector ligt vooral op het gebied van testen, JVM Interface, geheugendruk en extreem kort lopende jobs.

Nieuwe features

De Local-Variable syntax for Lambda Parameters, zoals beschreven in JEP 323, is een van de nieuwe features gepland voor Java SE 11 en zal toestaan dat het woord 'var' wordt gebruikt bij het declareren van de formele parameters van impliciet getypeerde Lambda expressies. Sinds Java SE 10 is het mogelijk om var te gebruiken voor implicit typing van lokale variabelen, maar dit kan binnen Lambda expressies nog niet worden gebruikt. Deze JEP wil dat rechttrekken (zie **Listing 1**).

Deze twee statements zijn dan gelijk aan elkaar. In deze JEP is het nog niet toegestaan om expliciete declaratie en impliciet (var) door elkaar te gebruiken (zie **Listing 2**).

```
(x, y) -> x.process(y)          // implicitly typed lambda expression
(var x, var y) -> x.process(y)  // implicit typed lambda expression
```

Listing 1

```
(var x, y) -> x.process(y)
// Cannot mix 'var' and 'no var' in implicitly typed lambda expression
(var x, int y) -> x.process(y)
// Cannot mix 'var' and manifest types in explicitly typed lambda expression
```

Listing 2

Ivo Woltring is Software Architect en Codesmith bij Ordina JTech.

Edwin Derks is Solutions Architect en CodeSmith bij Ordina JTech en heeft naast Java een passie voor cloud-driven development en serverless architecture.

JEP 309 beschrijft een uitbreiding op het class-file formaat voor ondersteuning van een nieuwe constant-pool variant. Deze variant maakt het mogelijk om de creatie te delegeren naar een zogenaamde bootstrap methode. Deze JEP is vooral gericht op de JVM en zou taalontwerpers en compiler builders meer opties moeten geven voor expressiviteit en performance.

De Java EE features, die voor het gemak voor ontwikkelaars sinds Java SE 6 ook in de Standard Edition zitten, zoals JAX-WS (Java API voor XML-gebaseerde webservices), JAXB (XML Binding), JAF (Java Activation Framework) en Common annotations, zullen verwijderd worden. In Java SE 9 zijn ze deprecated verklaard om in Java SE 11 (18.9) verwijderd te worden.

De motivatie hierachter is dat het onderhouden van deze modules in twee Java versies lastig is, doordat de verschillende versies ook verschillende evoluties doormaken. Oracle zegt eigenlijk dat met twee onafhankelijke versies er geen noodzaak meer is om deze features in Java SE te hebben. Dit kan gevolgen hebben voor applicaties, die nu afhankelijk zijn van deze "out-of-the-box" ondersteuning van de EE features. De code zal niet meer compileren op Java SE 11 (18.9).

Volgens Oracle is er geen significante interesse meer voor het ontwikkelen van nieuwe applicaties met CORBA en zijn de kosten van het onderhouden daarvan niet langer rendabel. Ook hier is er een risico voor applicaties die een (sub-)set aan CORBA API's gebruiken, dat ze niet meer zullen werken als het gecompileerd is op Java SE 11+. Voor alle EE modules, die verwijderd worden, zijn ook third party oplossingen te vinden, maar niet voor CORBA. Meerdere keren nadenken dus voor applicaties, die CORBA gebruiken, als je wilt upgraden.

Java FX gaat uit de Standaard Editie van Java verwijderd worden, zodat het niet meer mee hoeft te draaien in het halfjaarlijkse release schema.

Ondertussen... bij Java EE

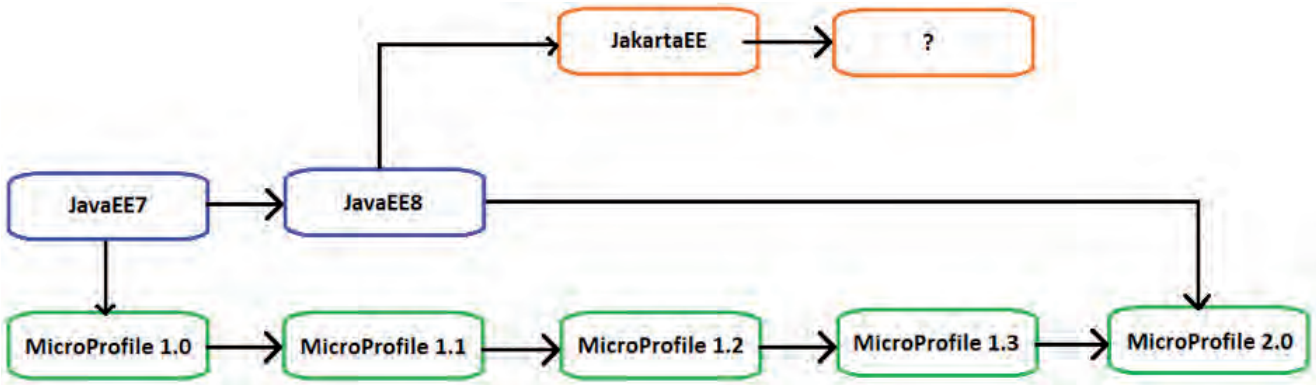
Jarenlang is de Java EE specificatie eigendom van Oracle geweest. Dat hoor je goed: geweest. Afgelopen najaar heeft Oracle de Java EE specificatie overgedragen aan de Eclipse Foundation, in dezelfde tijd dat Oracle eindelijk de definitieve en langverwachte versie van de Java EE 8 specificatie had uitgebracht.

De ontwikkeling van de Java EE specificatie maakte namelijk een moeilijke tijd door onder het bewind van Oracle. Dit had als reden dat de specificatie niet bijbleef met de ontwikkelingen van de enterprise softwaremarkt om Java EE heen. Hierdoor zag Oracle er waarschijnlijk geen heil meer in en wilden ze dit project onderbrengen bij een partij, die hier verandering in moet brengen.

Hiermee is de Java EE specificatie een open source project geworden. Men hoopt dat de community het project oppakt en deze in een hoger tempo doorontwikkelt naar moderne maatstaven voor enterprise software. Dit is helaas onder het bewind van Oracle niet gelukt, ondanks dat grote bedrijven als RedHat en IBM hier actief hun steentje aan bij hebben gedragen.

Een ander issue was de komst van Java SE 9 met zijn modulesysteem. Dit fenomeen heeft een grote impact op de adoptie van Java SE 9+ in de Java EE implementaties. Dit komt o.a. doordat de applicatieservers, die de Java EE specificatie implementeren vaak ook intern een eigen modularisatie toepassen om bijvoorbeeld snelle starttijden en lage memory footprints te realiseren. Tevens moeten ook alle referentie implementaties van de specificaties hierop aangepast worden. Ondersteuning voor Java SE 9 is daarom nog steeds niet volledig of optimaal.

Maar dat is nog niet alles: Oracle heeft dan wel afstand gedaan van de Java EE specificatie, maar niet van de naam. De community heeft de specificatie omgedoopt naar Jakarta EE, met als interne project naam: EE4J. De applicatieserver Glassfish (tevens gedoneerd en verder doorontwikkeld als Payara) blijft overigens (althans voorlopig) de referentie implementatie van de Jakarta EE specificatie.




```
if (obj instanceof Integer) {
   int intValue = ((Integer) obj).intValue();
   // use intValue
}
```

Listing 3

```
if (x matches Integer i) {
   // gebruik i hier
}
```

Listing 4

Naast Java EE en Jakarta EE is er ook nog het project MicroProfile om microservices mee te bouwen in Java EE stijl. Het huidige MicroProfile 1.3 is gebaseerd op Java EE 7 en vult met zijn specificaties het gat op, dat Java EE 7 open liet om naar moderne maatstaven microservices te bouwen. Versie 2.0 wordt uitgebracht in juni 2018 en bevat updates voor een Java EE 8 basis en gloednieuwe JSON-B specificatie.

Jakara EE heeft als doel om cloud native Java te supporten. In hoeverre dit project gaat doorpakken op zowel de bestaande Java EE als mogelijk de MicroProfile specificatie zullen we in de nabije toekomst zien.

Maar hoe nu verder?

Tot dit punt van het artikel zaten we nog redelijk op wat daadwerkelijk al gepland is voor de nabije toekomst en wat de concrete plannen zijn. Maar hoe nu verder? Een aantal zaken zitten al in de pipeline. Zo is er project

```
String html = "<html>\n" +
              "    <body>\n" +
              "        <p>Hello World.</p>\n" +
              "    </body>\n" +
              "</html>\n";
```

Listing 5

```
String html = `<html>
                 <body>
                   <p>Hello World.</p>
                 </body>
               </html>
               `;
```

Listing 6

Amber, dat als doel heeft om kleinere, op productiviteit georiënteerde Java-taalfuncties te verkennen, die zijn geaccepteerd als kandidaat JEP's. JEP 323 uit project Amber is ondertussen gepromoot naar feature van Java SE 11 (18.9).

Zo wordt in Project Amber onder andere gekeken naar **Pattern Matching** (JEP 305), **Raw String Literals** (JEP 326) en **Data Classes** (Nog geen JEP).

Bij Pattern Matching wordt onder andere bekeken of veel voorkomende constructies, (zoals in **Listing 3**) gemakkelijker gemaakt kunnen worden met een constructie als in **Listing 4**.

Bij **Raw String Literals** wordt gekeken of het gebruiken van String gemakkelijker kan.

Vervang bijvoorbeeld **Listing 5** door **Listing 6**.

Ook wordt hier gekeken of het dubbel escapen weg kan. Vervang:

```
"this".matches("\\w\\w\\w\\w");
```

Door:

```
"this".matches(`\w\w\w\w`);
```

Bij **Data Classes** wordt onderzocht of een stuk verbositeit van Java geëlimineerd kan worden door het introduceren van Data Classes. Het idee is om classes als **Listing 9** te vervangen door:

```
record Point(int x, int y) { }
```

Dit lijkt een no-brainer, waarbij de meeste Javanen zullen denken: "ja, doe maar!". Het is echter moeilijk om consensus te krijgen over zaken als muteerbaarheid, uitbreidbaarheid en constructors.

Als stukje onderhoud van de JVM is er project **Graal**. Graal biedt bijvoorbeeld ondersteuning voor het gebruik van een custom JIT (Just-in-Time) compiler. Zo'n JIT compiler is nu beschikbaar voor Java, maar eventueel ook voor Scala, Clojure of Kotlin. De Java JIT compiler is tevens geschreven in Java en niet in C++, zoals de conventionele C1 en C2 compiler (die we nu gebruiken).

Dus we krijgen een JIT Compiler, geschreven in Java, die we als zodanig kunnen ontwikkelen en onderhouden. Of een JIT Compiler nu geschreven is in Java of C++, het gaat om wat deze compiler runtime interpreteert naar machinecode. Daarom wil men vanaf nu een JIT Compiler in Java. De C1 en C2 zijn te oud en niet meer te onderhouden (zeker niet met C++). Met Java hopen ze hiermee een stap te zetten in de toekomstbestendigheid van de JVM.

Toekomstvisie

Brian Goetz is van mening dat Java op deze manier op de juiste weg is. Oracle loopt inderdaad niet voorop met het introduceren van nieuwe baanbrekende features, maar kijkt naar de ervaringen, die zijn opgedaan bij features in andere talen. Vooralsnog lijkt Java daarmee zijn kracht te behouden. Wij zijn zelf in ieder geval heel benieuwd naar de toekomst van Java en zullen deze ook op de voet blijven volgen. ■

```
final class Point {
   public final int x;
   public final int y;

   public Point(int x, int y) {
       this.x = x;
       this.y = y;
   }
   // implementaties van equals, hashCode, toString
   // verder niets
}
```

Listing 9

AFGELOPEN NAJAAR HEEFT ORACLE DE JAVA EE SPECIFICATIE OVERGEDRAGEN AAN DE ECLIPSE FOUNDATION

REFERENTIES

http://openjdk.java.net/projects/graal/
http://chrisseaton.com/truffleruby/jokerconf17/
https://mreinhold.org/blog/forward-faster
http://openjdk.java.net/jeps
https://jakarta.ee
https://projects.eclipse.org/projects/technology.microprofile/releases/microprofile-2.0
http://openjdk.java.net/projects/amber/
http://openjdk.java.net/projects/jdk/11/