Java microframeworks shootout

Vier microframeworks onder de loep

De tijd dat wekelijks een nieuw Java Web/MVC framework verscheen, ligt inmiddels achter ons. Er lijkt echter een nieuwe trend te zijn ontstaan: de introductie van Java *microframeworks* voor met name de realisatie van HTTP/REST-gebaseerde microservices.

Voor microservices blijkt de traditionele applicatie server steeds minder geschikt. De rol van resource sharing wordt in toenemende mate ingevuld door virtualisatie- en containertechnologie. Die ontwikkeling heeft er mede aan bijgedragen dat frameworks verschijnen met een focus op het bouwen van lichtgewicht Java services die worden gepackaged in direct uitvoerbare JAR's. Deze frameworks worden - vanwege de beperkte scope en kleine footprint - ook wel aangeduid als microframeworks.

In Java Magazine 2016-03 (**http://www.nljug.org/databasejava/**) is Ratpack al aan bod gekomen. Dit is slechts één voorbeeld van zo'n jong microframework. Er zijn er echter veel meer: Spark, ActiveWeb, Jodd, Rapidoid, Pippo, Blade, Jooby, Ninja, Fathom, noem maar op. In dit artikel kijken we naar een aantal van deze, minder bekende, microframeworks en vergelijken we de geboden functionaliteit, developer ervaring en community. Gezien het grote aantal microframeworks beperken we onze selectie tot:

- Spark (**http://sparkjava.com**);
- Ninja (**http://ninjaframework.org**);
- Jooby (**http://jooby.org**);
- Pippo (**http://pippo.ro**).

Naast een onderlinge vergelijking stellen we ook de vraag: hoe verhouden deze microframeworks zich tot de meer gevestigde Java frameworks voor microservices, zoals Dropwizard en Spring Boot (beiden besproken in Java Magazine 2014-06)?

Ninja

Aan slag gaan met dit framework is eenvoudig. De enige vereisten zijn Java 7 of hoger en Maven. Beginnen met Ninja kan op basis van een Maven archetype welke een projectstructuur genereerd (**Listing 1**).

Hierna is het een kwestie van het uitvoeren van een "mvn install ninja:run" om de applicatie te starten. De gegenereerde pom.xml bevat naast Ninja ook de Maven shade plugin voor het genereren van een fat-jar. Out of the box bevat je project nu alle dependencies voor het bouwen van een applicatie op basis van REST of server-side HTML templating, met ondersteuning voor onder andere dependency injection, ORM, transactions, metrics, security, email, caching, etc.

Het Ninja framework is geïnspireerd op versie 1.x van het Play framework. Het doel is om een eenvoudig full-stack Java framework te bieden. Een aspect waar veel aandacht aan besteed is, is het maken van korte development cycles. Dit wordt ondersteund door de SuperDevMode die automatisch code wijzigingen oppakt.

SuperDevMode maakt gebruik van de door de IDE gecompileerde classes. Het is dus noodzaak dat je IDE automatisch je code kan compileren en dat deze feature is ingeschakeld. Zowel Netbeans, Eclipse als IntelliJ ondersteu-

```
mvn archetype:generate -DarchetypeGroupId=org.ninjaframework
-DarchetypeArtifactId=ninja-servlet-archetype-simple
```

Listing 1: Ninja archetype

Erwin de Gier is software architect en trainer bij de business line Open Source van Sogeti.

Richard Kettelerij is freelance software architect en co-founder van Mindloops.

	Ninja	Spark	Jooby	Pippo
REST	Java routes	Java DSL	Java DSL (of Annotaties)	Java DSL
HTTP(S)	Jetty	Jetty	Jetty, Netty, Undertow, Servlets	Jetty, Servlets
JSON	Jackson	GSON	Jackson, GSON	Jackson, GSON, Fastjson
Caching	Ehcache, Memcached	-	Ehcache, Memcached, Redis, Hazelcast, Guava cache	-
Templating / HTML	Freemarker	Freemarker, Handlebars, Jade, Pebble + 5 anderen	Freemarker, Handlebars, Jade, Pebble	Freemarker, Jade, Pebble, Groovy
Security	-	-	Pac4J	- (CSRF filter inclusief)
Database / ORM	JPA	-	Hibernate, JOOQ, RxJdbc, etc	-
Database / NoSql	-	-	Cassandra, Mongo, Couchbase	-
Validatie	Bean Validation	-	Bean Validation	Bean Validation
Metrics	Dropwizard Metrics	-	Dropwizard Metrics	Dropwizard Metrics
Configuratie	Java props	- (puur Java)	Typesafe config file	Java props
Build	Maven	Maven, Gradle	Maven	Maven
DI	Guice	-	Guice	Guice, Spring, Weld
Fat JAR (hello world app)	26 MB	2 MB	8,5 MB	3 MB

Tabel 1

```
public class Routes implements ApplicationRoutes {
  @Override
   public void init(Router router) {
        router.GET().route("/user").with(UserController.class, "user");
        ...
   }
}
```

Listing 2

```
package controllers;

@Singleton
public class UserController {

  public Result user (
      @PathParam("id") String id,
      @PathParam("email") String email) {
          return Results.ok().render(user);
  }
}
```

Listing 3

nen dit. Via het eerder getoonde "mvn ninja:run" wordt de SuperDevMode gestart. De plugin detecteert wijzigingen in de gecompileerde bestanden en voert een server herstart uit. Het is dus geen echte hotdeploy feature. Voor kleine projecten is de herstart vrij snel (minder dan een seconde).

HTTP endpoint en REST resources definieer je in Ninja via een centrale Router class (**listing 2**). Dit is enigszins vergelijkbaar met het conf/routes bestand in Play 1.x.

Via de route() methode kan een URL worden opgegeven, eventueel door gebruik te maken van een reguliere expressie. De URL mapped naar een specifieke methode in een controller class (**listing 3**).

Spark

Voor de duidelijkheid: dit is een ander framework dan Apache Spark, de big data engine. De naam is onhandig gekozen. Zoeken naar informatie over het Spark Java framework levert vaak resultaten over Apache Spark op. In tegenstelling tot Ninja heeft Spark een andere aanpak. Het framework is zo minimaal mogelijk opgezet. Voor zaken als dependency injection, metrics, database toegang en security moet je zelf een oplossing zoeken. Het voordeel hiervan is dat de minimale versie erg klein is en dat je kan kiezen welke dependencies je eventueel verder nodig hebt. Voor veel voorkomende functionaliteit, zoals het ontsluiten van databases of login functionaliteit, zijn wel tutorials en voorbeelden op de website beschikbaar.

Beginnen met Spark doe je door zelf een Maven of Gradle project te maken. Aan dit project voeg je vervolgens een dependency naar Spark toe. Deze manier van werken wijkt duidelijk af van die van Ninja en Jooby. Spark voelt meer aan als een library dan een framework. Voor mensen die ervan houden om alles zelf in de hand te hebben is dit ideaal. Als je echter verwacht dat veel voor je wordt geregeld, dan stelt Spark teleur. Het starten van een Spark applicatie gaat door het uitvoeren van een main methode. Als je een fat jar wil bouwen van je applicatie, dan zal je zelf de Maven shade plugin moeten toevoegen.

De idee achter Spark is het bieden van een puur Java alternatief voor tech-

nologieën, zoals NodeJS. Het streven is om zoveel mogelijk gebruik te maken van Java 8 lamba expressies om zodoende beknopte code te schrijven. Naast de mogelijkheden voor het bouwen van een RESTful services biedt Spark ook mogelijkheden om gebruik te maken van een template engine voor het renderen van HTML.

Net als veel andere microframeworks die worden besproken in dit artikel, biedt Spark een eenvoudige syntax voor het definieren van routes voor REST services (**listing 4**).

Voor elke HTTP methode is een Java methode beschikbaar die de koppeling maakt tussen een URL en een functie. Het is mogelijk om parameters en reguliere expressies te gebruiken in de URL mapping.

Jooby

Jooby hanteert, net als Ninja, de aanpak van een Maven archetype om mee te starten. De enige vereisten zijn Java 8 en Maven.

Dit levert een project op waarbij de POM is voorzien van de nodige dependencies en de Maven shade plugin.

Jooby heeft standaard minder dependencies en functionaliteit dan Ninja, maar meer dan Spark. Zo is er standaard ondersteuning voor REST, HTML templating, websockets en biedt Jooby ook ondersteuning voor het schrijven van applicaties in Javascript (op basis van Nashorn). Jooby pakt zaken anders aan dan de andere frameworks. Zo is de beschikbare functionaliteit sterk afhankelijk van de modules die je als ontwikkelaar kiest. Zo is er bijvoorbeeld de joob-jdbi module voor het werken met relationele databases.

Ook Jooby biedt een op lambda's gebaseerde DSL voor het definieren REST routes (**listing 5**).

Jooby biedt een hot reload functie op basis van JBoss Modules. JBoss Modules biedt een modulair class loading mechanisme met als doel sneller reloads te kunnen doen. De hot

```
public class HelloWorld {
    public static void main(String[] args) {
        get("/user", (req, res) -> "Hello User!");
    }
}
```

Listing 4

```
public class App extends Jooby {
  {
    get("/user/:name", req -> "hey " + req.param("name").value());
  }

  public static void main(final String[] args) throws Throwable {
    new App().start(args);
  }
}
```

Listing 5

reload functie kan worden gestart met: "mvn jooby:run". Wanneer class files of configuratiefiles veranderen, wordt vervolgens de applicatie opnieuw opgestart.

Bijzonder aan Jooby is dat een module beschikbaar is voor het genereren van Swagger specificaties. Swagger biedt een gestandaardiseerde beschrijving voor REST API's in HTML, JSON of YAML formaat. Deze beschrijving kan gegenereerd worden op basis van de applicatie code indien dit aan bepaalde conventies voldoet. Naast Swagger is het ook mogelijk om RAML beschrijvingen van de API de genereren.

```
 _____  _____  _____  _____  _______
( _ \(_  _)( _ \( _ \( _ )
 ) __/ _)(_  ) __/ ) __/ )(_)(
(__)  (_____)(__)  (__)  (_______)
```

Pippo

Voor Pippo is een Maven archetype beschikbaar (**listing 6**).

Dit levert een project op waarbij de pom is voorzien van dependencies zoals Jetty, de Maven assembly en jar plugins. Het idee achter Pippo is om een extreem klein (140 kb) framework te bieden wat eenvoudig uitgebreid kan worden. Hiervoor biedt Pippo een concept wat ze de Initializer noemen. Door middel van een pippo.properties bestand kunnen eigen classes worden gedefinieerd

ONTWIKKELAARS MET KENNIS VAN SPRING BOOT OF DROPWIZARD KUNNEN WAARSCHIJNLIJK ZONDER MOEITE AAN DE SLAG MET DE MICROFRAMEWORKS

```
mvn archetype:generate \
  -DarchetypeGroupId=ro.pippo \
  -DarchetypeArtifactId=pippo-quickstart \
  -DarchetypeVersion=0.8.0 \
  -DgroupId=com.mycompany \
  -DartifactId=myproject
```

Listing 6

```
public class UserInitializer implements Initializer {

    @Override
    public void init(Application application) {
        // show users page
        application.GET("/users", (routeContext) -> routeContext.render("users"));

        // show user page for the user with id specified as path parameter
        application.GET("/user/{id}", (routeContext) -> routeContext.render("user"));
    }

    @Override
    public void destroy(Application application) {
        // do nothing
    }
}
```

Listing 7

```
initializer=ro.pippo.freemarker.UserInitializer
```

die moeten worden geladen tijdens het opstarten van de applicatie. Deze class moet de Initializer interface implementeren:
Omdat vanuit de initializer toegang is tot de Pippo applicatie instantie, is het mogelijk om bijvoorbeeld HTTP routes toe te voegen in separate modules. Zo wordt het eenvoudig om de applicatie op te delen in verschillende jar files, elk met hun eigen pippo.properties bestand. Als deze jar files op het classpath staan, worden de initializer classes één voor één geinstantieerd en worden de aanwezige init() methodes uitgevoerd. **Listing 7** toont een Initializer.

Net als de andere microframeworks besproken in dit artikel, biedt Pippo een Java DSL voor het definiëren van REST routes (**listing 8**). De DSL van Spark en Jooby voelt wel eenvoudiger aan dan die van Pippo.

Op het gebied van dependency injection is Pippo heel flexibel. Het ondersteunt zowel Spring, Guice en CDI (in de vorm van Weld). Voor elk van deze oplossingen is een Pippo library beschikbaar die deze integratie verzorgt.

Wie van de vier?

Kenmerkend is dat de meeste microframeworks handig gebruik maken van de lambda mogelijkheden in Java 8 voor het definiëren van REST routes/resources. Deze "DSL"- aanpak wijkt duidelijk af van de bekende annotatie aanpak die we kennen uit JAX-RS of Spring. Verder heeft elk van de vier microframeworks zo zijn eigen sterke en zwakke punten.

Ninja biedt een "opinionated full-stack framework". Met andere woorden: je kan hier zo mee aan de slag. Alles wat je typisch nodig hebt in een CRUD webapplicatie zit erin en je hoeft zelf geen keuzes te maken welke libraries hierbij worden toegepast. Dit maakt Ninja wel het minst 'micro', aangezien je standaard een volledige stack krijgt. Dit zie je ook terug in de JAR size (zie **tabel 1**).

Spark is daarentegen het meest 'micro'. Het biedt primair een HTTP server en een aantrekkelijke syntax voor het definiëren van REST endpoints. Daarmee is Spark geschikt voor ontwikkelaars die geen full-stack nodig hebben of zelf hun stack willen opbouwen. Voordeel is tevens dat

```
GET("/contact/{id}", (routeContext) -> {
    int id = routeContext.getParameter("id").toInt(0);
    String action = routeContext.getParameter("action").toString("new");

    Map<String, Object> model = new HashMap<>();
    model.put("id", id);
    model.put("action", action)
    routeContext.render("contact", model);
});
```

Listing 8

Spark de grootste bekendheid geniet van deze vier microframeworks.

Pippo heeft veel gelijkenissen met Spark, maar waar Spark (anders dan de optionele template engine) geen enkele technische keuze aan de ontwikkelaar overlaat, doet Pippo dit wel. Zo heeft Pippo bijvoorbeeld de mogelijkheid om de instellingen van de onderliggende HTTP server te wijzigen of zelf een andere implementatie (Jetty, Undertow, etc.) te kiezen. Daarnaast biedt het modules voor zaken als dependency injection en metrics.

Jooby geeft de ontwikkelaar weer de meeste keuzevrijheid. Het is het minst opinionated microframework, aangezien het modulair is opgezet en ontwikkelaars zelf de benodigde modules kunnen kiezen. Het verschil met Pippo is dat modules beschikbaar zijn voor de gehele webapplicatie stack, inclusief persistence. Kortom, Jooby kan zowel full-stack worden ingezet (a la Ninja), maar ook enkel als REST endpoint (a la Spark) dienen.

Community

Op het moment dat je jouw applicatie baseert op een framework wil je met enige zekerheid kunnen stellen dat dit framework nog een tijd wordt onderhouden. Je applicatie migreren naar een ander framework kan een behoorlijke klus zijn. Om de parallel te trekken naar de eerder genoemde Java Web/MVC frameworks die vorig decennium sterk in opkomst waren: daarvan zijn er nu slechts enkele over. Dit heeft diverse redenen maar één reden is dat sommige frameworks een te kleine community hebben.

Voor alle besproken microframeworks geldt dat deze op dit moment actief onderhouden worden. Dit blijkt onder meer uit het feit dat recentelijk releases hebben plaatsgevonden. **Tabel 2** toont meer cijfers over de communities.

We zien dat Ninja inmiddels vier jaar in ontwikkeling is. Ook is er een behoorlijke groep mensen die bijdragen levert aan dit framework (61). Maar als we dieper graven, dan blijkt dat het gros van die bijdragen afkomstig zijn van slechts 2 personen. Dit fenomeen zien we ook bij de andere microframeworks. Wat betreft community omvang staat Spark op nummer 1. Dit framework is duidelijk het populairst. Jooby is het jongst en momenteel een one-man show. Pippo heeft wat meer tractie, maar een punt van zorg is dat het een zeer lage test coverage kent (20%).

Gebaseerd op deze statistieken raden we momenteel Spark en Ninja aan boven Jooby en Pippo. De eerste twee zijn langer beschikbaar, hebben grotere communities (met name Spark) en voor Ninja is (als je daar behoefte aan hebt) commercieel support beschikbaar.

Spring Boot, Dropwizard, etc?

Als we kijken naar de meer gevestigde frameworks, zoals Spring Boot of Dropwizard, dan geldt dat de besproken microframeworks slechts een subset van de aanwezige functionaliteit in die frameworks aanbieden. In andere woorden: met Spring Boot of Dropwizard kan je functioneel hetzelfde bereiken als met een microframework (en meer).

Mocht je dus al gebruikmaken van bijvoorbeeld Spring Boot, dan zal je de afweging moeten maken of het gebruik van een microframework - voor een nieuw te realiseren microservice - wel de moeite waard is? Aan de andere kant zijn de besproken microframeworks dermate eenvoudig te doorgronden, zodat ontwikkelaars met kennis van Spring Boot of Dropwizard waarschijnlijk zonder moeite aan de slag kunnen. Desalniettemin is het de vraag of een microframework in dit geval tot een complexiteitsreductie of -toename leidt. Dit kan per situatie verschillen. Vergeet verder niet dat de communities van de vier besproken microframeworks aanzienlijk kleiner zijn dan die van de gevestigde frameworks.

De sweet spot voor Java microframeworks lijkt dus vooral te liggen bij ontwikkelaars die:

- geen fullstack framework nodig hebben. Bijvoorbeeld voor (integratie)services waar persistence geen rol speelt, of voor eenvoudige beheer- of testtools.
- incidenteel iets met Java doen en een alternatief zoeken voor microframeworks zoals Sinatra (Ruby) of Flask (Python). De besproken Java microframeworks stellen deze groep ontwikkelaars in staat om met een minimale leercurve een schaalbare HTTP service op de JVM te realiseren.
- veel controle willen hebben over hun gehele Java stack. In deze gevallen zal men een microframework willen combineren met andere libraries/frameworks naar keuze. Denk aan een combinatie als Spark Dagger (voor IoC) + No(SQL) library.
- gecharmeerd zijn van de getoonde Java 8 "DSLs" voor het definieren van REST routes. ■

VOOR ALLE BESPROKEN MICROFRAMEWORKS GELDT DAT DEZE OP DIT MOMENT ACTIEF ONDERHOUDEN WORDEN

	Ninja	Spark	Jooby	Pippo
GitHub stars	1450	4700	250	480
Aantal commits	1827	703	805	1041
Aantal issues (open/closed)	79/133	124/201	17/374	15/89
Eerste release	Aug. 2012	Feb. 2013	April 2015	Okt. 2014
Aantal contributers	61	73	6	10
Aantal 'core' contributers	2	2	1	2

Tabel 2