Clojure

Een volwassen alternatief voor Java

Dit jaar vieren we het 20-jarig bestaan van Java, de populairste programmeertaal van dit moment. Tenminste, als we uitgaan van de TIOBE index (1). Populariteit verkrijg je echter niet zomaar en in het begin moesten er wel degelijk zieltjes worden gewonnen. Zo werd Java destijds gepositioneerd als een beter alternatief voor C++: "*We were after the C++ programmers. We managed to drag a lot of them about halfway to Lisp.*" -- Guy Steele, co-auteur van de Java specificatie.

En zo kwam het dus dat vele programmeurs de overstap naar Java maakten, waarmee de eerste helft van het verlichtingspad werd afgelegd. Dit artikel gaat over dat tweede gedeelte, de overstap naar Lisp – of in ieder geval: een lisp, volgens Steele het toppunt van productiviteit.

De lisp waar we het over hebben, is Clojure, de programmeertaal voor de JVM die in 2007 werd geïntroduceerd door Rich Hickey. Op de Java One van 2014 mocht Hickey een praatje (**2**) houden, waar hij begon met de volgende quote (**3**):

"A lot of the best programmers and the most productive programmers I know, are writing everything in Clojure and swearing by it, and then just producing ridiculously sophisticated things in a very short time. And that programmer productivity matters." -- Adrian Cockroft, (voormalig) Netflix

De pitch van Rich was dat je met Clojure programma's kunt schrijven die beter en flexibeler zijn en dat je er bovendien ook productiever mee bent dan met Java. Het zal zeker een reden zijn waarom Clojure her en der is omarmd als een volwassen alternatief voor Java. Naast het eerder genoemde Netflix geldt dat bijvoorbeeld ook voor de programmeurs bij WalmartLabs (**4**):

"Clojure shrinks our code base to about one-fifth the size it would be if we had written in Java" -- Anthony Marcar, architect, WalmartLabs

De taal heeft zich ondertussen bewezen in de industrie. Wellicht voor jou ook een reden om over te stappen op Clojure? We zetten hier zeven voordelen van overstappen voor je op een rijtje.

1. Clojure is simpel

Rich Hickey heeft een jaar van zijn leven gewijd aan het ontwerpen van een taal die vooral simpel moest zijn. Simpel betekent daarbij niet: gemakkelijk ('easy'). Want het was zeker niet zijn doel om een taal te ontwerpen die eenvoudig is voor Java-programmeurs, want dan kom je niet verder. Zijn doel was om Javanen te helpen om programmeursnirvana te bereiken en de weg daarheen kan voor sommigen best pijnlijk zijn, met al die haakjes en die prefix-notatie. Maar als je daar eenmaal aan gewend bent, dan ontdek je dat de syntax verbazingwekkend simpel is.

De eenvoud van Clojure uit zich bijvoorbeeld in de manier waarop je functies schrijft. Clojure dwingt je in de richting van pure functies, dus functies zonder side effects. Dat wordt bereikt door middel van *immutability*. Als je uit kunt gaan van immutable values, is je code beter te begrijpen en makkelijker te testen.

2. Clojure is data-georiënteerd

Eenvoud uit zich ook in het feit dat Clojure niet object-georiënteerd is, maar data-georiënteerd. Data wordt gerepresenteerd met immutable hashmaps, in plaats van classes zoals in Java. Deze datastructuren zijn wel eenvoudig te manipuleren met functies, maar staan daar los van. De filosofie van Clojure sluit daarom aan bij een uitspraak van Alan J. Perlis:

Martin van Amersfoorth is software engineer bij Finalist. Meer dan twintig jaar geleden, begonnen als Lisp-programmeur, stapte hij over naar 'the dark side': de curly braced OO-talen. Eerst met Objective-C, daarna met Java. Inmiddels is hij op zijn schreden teruggekeerd en ontwikkelt hij nu voornamelijk in Clojure.

```
(defn what-is-my-ip [request]
  {:status 200
   :headers {"Content-Type" "text/plain"}
   :body (:remote-addr request)})
```

Listing 1

"It is better to have 100 functions operate on one data structure than to have 10 functions operate on 10 data structures."

Als je steeds dezelfde datastructuren hergebruikt, hoef je niet telkens het wiel opnieuw uit te vinden om je data in de gewenste vorm te krijgen. Classes vormen echter een taal op zichzelf. Je moet eerst de methods leren kennen voordat je de datastructuren erachter kunt gebruiken. Omdat hashmaps in Clojure immutablezijn, hoef je bij het rekenen met hashmaps ook geen rekening te houden met eventuele side-effects.

Een klein voorbeeld. In de Ring (**5**) library, waarmee je webapplicaties kunt schrijven, is een handler niets anders dan een functie die een hashmap verwacht (de request) en een hashmap teruggeeft (de response), zie **Listing 1**).

De namespace waarin we deze handler definiëren heeft geen afhankelijkheid van de Ring-library. Het enige wat we hoeven te weten, is hoe een request er in Ring uitziet, dus welke keys we in de hashmap kunnen verwachten en welke keys we terug moeten geven om een geldige response te kunnen vormen. Alles wat tussen request en response gebeurt, is niets anders dan rekenen met normale hashmaps. Dat is toch een stuk eenvoudiger dan het werken met `HttpServletRequest` en `HttpServletResponse`.

3. Clojure vergt minder regels code

Met Clojure heb je geen last van de overhead die object-georiënteerde talen met zich meebrengen. Overhead die bestaat uit het schrijven van interface- en implementatiecode om je datastructuren te 'beschermen'. Je hebt dus minder Clojure-regels dan Java-regels nodig om precies hetzelfde te bereiken.

Maar dat is niet de enige reden waarom je minder regels code nodig hebt. Clojure is immers een functionele programmeertaal en daarvan weten we dat ze expressiever zijn dan imperatieve programmeertalen. Zo hoef je geen loops of iteraties uit te schrijven, omdat het volstaat om *higher order* functies, zoals map, filter en reduce, toe te passen op je collecties. Sinds versie 8 beschikt Java over lambda's en de Streams API. De winst van functioneel programmeren ten op zichte van Java 7 is al vrij snel voelbaar, zodra je met transformaties over collecties aan de slag gaat.

Omdat Clojure een lisp is, kun je boilerplate-code voorkomen door gebruik te maken van macro's. Macros zijn functies die compile-time worden uitgevoerd en als argumenten expressies kunnen ontvangen. Expressies worden gerepresenteerd als data, die je ook kunt transformeren met behulp van Clojure-functies. Dit komt dus neer op code-generatie, zodat je boilerplate niet zelf hoeft te schrijven.

De website **http://www.todobackend.com/** bevat vergelijkbare implementaties van een Todo-list-API. Als we de implementatie in Clojure afzetten tegen de implementaties in Java 7 en 8 zien we het volgende:

- Clojure: 168 regels Clojure (inclusief buildconfiguratie);
- Java 7 + Spring MVC: 555 regels XML, 228 regels Java, 56 regels Groovy;
- Java 8 + Spring 4 Boot: 200 regels Java, 37 regels Groovy.

We hebben hier het aantal regels code van de gebruikte libraries en frameworks niet meegeteld. Onze verwachting is dat Clojure er dan nog beter van af zou komen. De voordelen van minder regels code zijn evident. Kleinere programma's hebben minder bugs, zijn beter te begrijpen en zijn door de eenvoud ook robuuster.

4. Clojure ondersteunt meerdere platformen

Bij het werken in het Clojure-ecosysteem kun je de kennis van dezelfde programmeertaal hergebruiken in meerdere omgevingen. Voor server side ontwikkeling maak je gebruik van Clojure op de JVM (of in .NET via ClojureCLR).

Voor front-end development kun je gebruik maken van ClojureScript, wat compileert

Michiel Borkent ontdekte tijdens het afstudeerproject van zijn studie Informatica de kracht van Common Lisp. Later zag hij Clojure als praktischer alternatief voor het bedrijfsleven, omdat het te combineren is met andere JVM-technologie. Als hogeschooldocent gebruikte hij Clojure als taal voor een introductiecursus functioneel programmeren*. Bij Finalist heeft hij Clojure gebruikt voor hetontwikkelen full stack webapplicaties.

*) **http://michielborkent.nl/clojurecursus**

naar JavaScript. De ClojureScript-compiler kan naast JavaScript voor de browser ook JavaScript genereren voor NodeJS, wat je dan ook weer voor server side toepassingen kunt gebruiken. De implementaties van Clojure op diverse omgevingen zijn zoveel mogelijk hetzelfde gehouden en wijken alleen af als dat afgedwongen wordt door beperkingen van de omgeving. Een voorbeeld van zo'n beperking is de afwezigheid van multithreading in een browser.

Veel Clojure libraries bieden ondersteuning voor zowel Clojure als ClojureScript. Dus ook de kennis die je hebt van deze libraries kun je hergebruiken over omgevingen heen. Schrijvers van libraries kunnen sinds Clojure 1.7 (uitgebracht op 30 juni 2015) gebruik maken van reader conditionals. Dit geeft de mogelijkheid om binnen één bestand meerdere platformen te bedienen.

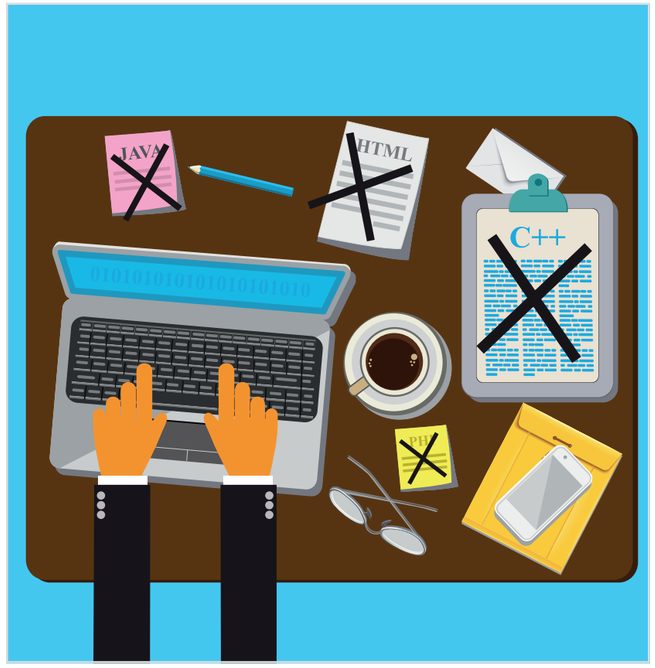
Ook als applicatie-ontwikkelaar kun je reader conditionals gebruiken om dezelfde code te gebruiken voor meerdere platformen. Een voorbeeld. **Listing 2** definieert een variabele met de waarde NaN. Op het Clojure/JVM-platform is dat Double/NaN en in JavaScript NaN uit de globale namespace.

5. Clojure is interactief

Clojure is een interactieve programmeertaal. Via een REPL (read-eval-print loop) krijg je direct feedback. Tijdens het ontwikkelen, kun je functies herdefiniëren en opnieuw testen terwijl de applicatie draait, zonder een nieuwe compileercyclus te starten. Een plug-in, zoals JRebel voor hot code reloading, is dus niet nodig. Het is zelfs mogelijk om over het netwerk in te loggen op een REPL-sessie en zo de stand van zaken te inspecteren op een productieserver.

```
(def not-a-number
  #?(:clj Double/NaN
     :cljs    js/NaN
     :default nil))
```

Listing 2



Ook ClojureScript kent het concept van een REPL waarmee je code in je browser (of NodeJS) kan testen en herdefiniëren. Een tool als Figwheel (**6**) maakt het tevens mogelijk om het gewijzigde ClojureScript meteen in je browser te zien, zonder dat je de applicatie in je browser moet verversen.

Dit alles maakt Clojure uitermate geschikt voor live coding. Je kan met Clojure zelfs live muziek maken. Bekijk hiervoor de library Overtone (**7**).

6. Clojure is full stack

Hoe je het ook wendt of keert, Java is niet full stack, het is een taal voor de backend. Wel is geprobeerd om een plekje te veroveren aan de client side, maar wie ontwikkelt er tegenwoordig nog applicaties in AWT, Swing of JavaFX? Evenzo is Clojure begonnen als alternatief voor backend, maar met de komst van ClojureScript, en een hoop front-end libraries en tools, kun je nu met recht spreken van een full stack programmeertaal. Bovendien biedt de stack interessante elementen waarmee full stack ontwikkeling aanzienlijk kan worden verbeterd.

DE TAAL HEEFT ZICH ONDERTUSSEN BEWEZEN IN DE INDUSTRIE

```
(def count-state (atom 10))

(defn counter []
  [:div
   @count-state
   [:button {:on-click #(swap! count-state inc)}
    "x"]])

(reagent/render-component [counter]
                          (js/document.getElementById "app"))
```

Listing 3

React

Met de komst van JavaScript library React is het bouwen van een SPA (single page app) in ClojureScript een fluitje van een cent. Reagent is een ClojureScript library die het schrijven van React-componenten vergemakkelijkt. Door middel van Hiccup-notatie, waarmee je uit een geneste Clojure-datastructuur HTML kan beschrijven, kun je met relatief weinig code een React-component schrijven.

Listing 3 is een voorbeeld van een component wat het aantal clicks telt op een knop en dit toont in een div.

Met Om, een andere ClojureScript wrapper rondom React, zijn zelfs betere performance benchmarks gehaald dan React zelf. Dit heeft te maken met de efficiënte manier waarop verschillen in state kan worden bepaald in ClojureScript (maar ook in Clojure zelf).

7. Clojure voorkomt callback hell

Een van de problemen waar JavaScript-developers mee worstelen, is het fenomeen callback hell. Een browser heeft maar één thread. Daarom moeten we werken met geneste callbacks. Als we callbacks diep nesten, wordt de code steeds slechter leesbaar.

Clojure geeft hiervoor een oplossing in de vorm van de core.async library, die het werken met asynchrone code vereenvoudigt. Uiteraard werkt ook deze library op zowel server als client. De bouwblokken van core.async zijn channels, buffers en go-blocks. Go-blocks zijn source-transformaties die de illusie geven van synchrone code, maar uiteindelijk geneste callbacks opleveren. Een voorbeeld. De code in **Listing 4** haalt eerst het emailadres op van een gebruiker via een REST-API call. Hiervoor gebruiken we de library http-cljs, die het maken van Ajax calls combineert met core.async. Als resultaat van een http-call krijgen we een een channel terug, waaruit we het resultaat uit kunnen lezen met de <! (take) operatie. Vervolgens gebruiken we het emailadres in een tweede call om de bestellingen van deze gebruiker op te halen. Als uiteindelijk resultaat geven we het aantal bestellingen terug.

```
(go (let [email (:body
                 (<! (http/get
                      (str "/api/users/"
                           "123"
                           "/email"))))
          orders (:body
                  (<! (http/get
                       (str
                        "/api/orders-by-email/"
                        email))))]
      (count orders)))
```

Listing 4

Slot

Java bestaat 20 jaar. Inmiddels zijn vele alternatieve programmeertalen verschenen waarmee je applicaties voor de JVM kunt schrijven. Van de Cambrische explosie van nieuwe talen die ontstond, halverwege het vorige decennium, is er een beperkt aantal die de status van volwassenheid hebben bereikt: Groovy, Scala, maar ook Clojure. In dit artikel hebben we een aantal voordelen benoemd waarmee Clojure zich onderscheidt van de rest. Mocht dit je hebben overtuigd, de beste manier om met Clojure aan de slag te gaan, is door er gewoon mee te beginnen. Om de drempel zo klein mogelijk te maken, kun je gebruik maken van onze overstapservice (**8**). ■

HET DOEL WAS OM JAVANEN TE HELPEN OM EEN PROGRAMMEURS-NIRVANA TE BEREIKEN

REFERENTIES

[1] http://www.tiobe.com/index.php/content/paperinfo/tpci/index.html
[2] https://www.youtube.com/watch?v=VSdnJDO-xdg
[3] http://thenewstack.io/the-new-stack-makers-adrian-cockcroft-on-sun-netflix-clojure-go-docker-and-more/
[4] http://blog.cognitect.com/blog/2015/6/30/walmart-runs-clojure-at-scale
[5] https://github.com/ring-clojure
[6] https://github.com/bhauman/lein-figwheel
[7] http://overtone.github.io/
[8] https://github.com/finalist/clojure-overstapservice