

Why Clojure?

Erwin Rooijackers

14-12-2016

Agenda

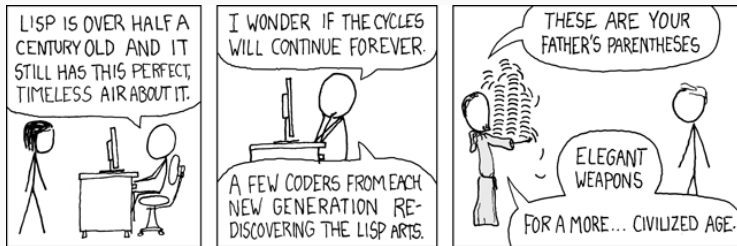
- Why Clojure is great:
 - It is a Lisp
 - Data-oriented
 - Embraces functional programming
 - JVM + JavaScript runtime
- Demo:
 - Figwheel + Reagent

Clojure is a Lisp

What is so great about Lisp?

"The most powerful programming language is Lisp. If you don't know Lisp (or its variant, Scheme), you don't know what it means for a programming language to be powerful and elegant. Once you learn Lisp, you will see what is lacking in most other languages." — Richard Stallman

Parentheses



(<http://xkcd.com/297/>)

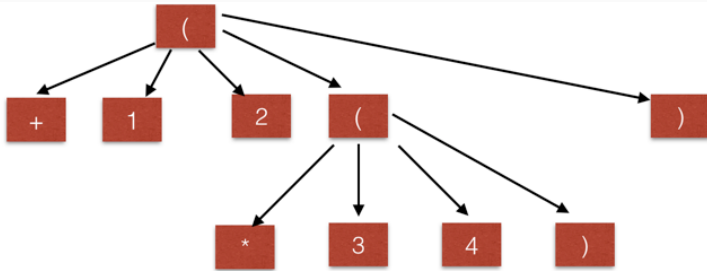
Almost all syntax

Lists	<code>'(1 2 3), '(fred kees piet)</code>
Vectors	<code>[1 2 3 4 5], [fred kees piet]</code>
Maps	<code>{:a 1 :b 2 :b 3}, {1 kees 2 piet}</code>
Sets	<code>#{1 2 3}</code>
Code	<code>(+ 1 2 3) ;; => 6</code>
Naming	<code>(def n 10)</code>
Lambda	<code>(def plus-two (fn [a] (+ a *)))</code> <code>(plus-two 2) ;; => 4</code>
Quote	<code>'(+ 1 2 3) ;; => (+ 1 2 3)</code>

- Parentheses!
- Code is data and data is code (homoiconicity)

Parentheses

(+ 1 2 (* 3 4))



AST of (+ 1 2 (* 3 4))

Manipulating the AST

```
(defmacro unless [pred a b]  
  `(if (not ~pred) ~a ~b))
```

;; Usage:

```
(unless false  
  (println "Will print")  
  (println "Will not print"))
```

;; Macro expansion:

```
(if (not false)  
  (println "Will print")  
  (println "Will not print"))
```


Programmable programming language

- **unless** impossible to implement as function.
- Second expression is not evaluated!
- "programmable programming language"
- Lisp can build any abstraction at all if you can define syntax and semantics for it.
- Clojure has idiomatic ways of doing that and a small core (no Lisp Curse).

Clojure is data oriented

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

Clojure is data oriented

Rich Hickey time

<https://youtu.be/VSdnJD0-xdg?t=49m8s>

Clojure embraces functional
programming

Functional programming

- End of Moore's law
- More cores
- Distributed
- Parallelism and concurrency
- How can we adapt our programming practices to this future?
- Parallelism and concurrency (impossibly) hard in some languages

Root of the problem

- "Non-determinism caused by concurrent threads accessing shared mutable state." - Martin Odersky

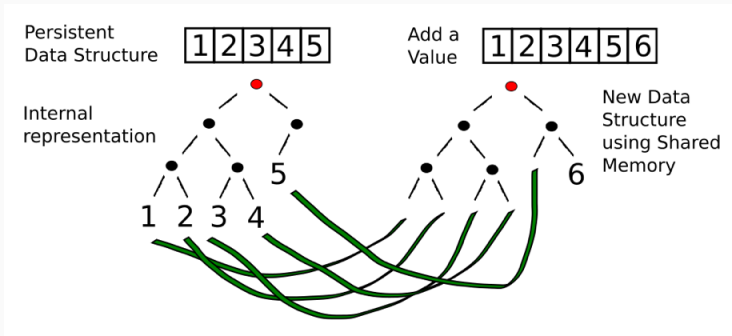
```
var x = 0
async { x = x + 1 }
async { x = x * 2 }
// Can give 0, 1, 2
```

(Martin Odersky, "Working Hard to Keep It Simple" - OSCON Java 2011)

Functional programming

- Parallel processing is a fact
- No mutable state means no problem
- Imperatively we think in variables and blocks of memory that change over **time**.
- Functionally we think in **space**: I construct this, then that, then third things out of that.
- Recursion
- **Pure functions** are clearer to reason about

Immutable data structures - Structural sharing



- Algorithms ensure expected performance characteristics of data structure

Runs on JVM and JavaScript runtime

- Boring is good.
- Clojure and ClojureScript embrace host platforms.
- Sequences implement the expected interfaces.
- Interop with host.

Figwheel and Reagent

...

Learn more:

- Rich Hickey - Clojure, Made Simple:
<https://youtu.be/VSdnJD0-xdg>
- Derek Slager - ClojureScript for Skeptics:
<https://youtu.be/gsf fg5xxFQI>
- Rich Hickey talks collection: <http://bit.ly/1KQNzBr>
- Bret Victor - Inventing on Principle:
<https://vimeo.com/36579366>