Treball Dirigit Clojure

Llenguatges de Programació | UPC FIB

Curs 2024/25 Quadrimestre Primavera

Hash: 1745

Índex

1 Introducció

Clojure és un llenguatge de programació funcional de la família Lisp que combina la simplicitat sintàctica i la potència de la programació funcional amb la solidesa de la màquina virtual Java (JVM).

Clojure aposta per dades immutables, funcions pures i primitives de concurrència segures que faciliten l'escriptura de codi robust i fàcilment paral·lelitzable. A més, l'interoperabilitat completa amb l'ecosistema Java, juntament amb les variants ClojureScript (per a JavaScript) i ClojureCLR (per a .NET), amplien el seu abast a pràcticament qualsevol entorn de desenvolupament.

El nom Clojure és un joc de paraules amb el terme anglès *closure* (clausura) i alhora incorpora les lletres C, L i J en referència a C#, Lisp i Java, tres influències clau en el seu disseny.

2 Història de Clojure

Rich Hickey, un desenvolupador amb gairebé 20 anys d'experiència en C++, Java i C#, volia una manera més senzilla de programar aplicacions multithread en entorns comercials. Després de constatar el poder expressiu de Lisp i la dificultat de gestionar la concurrència amb llenguatges orientats a objectes, va decidir dissenyar un nou llenguatge funcional, dinàmic i "simpàtic"amb la plataforma Java. Els seus objectius inicials eren:

- Oferir un Lisp "industrial", amb sintaxi mínima però rendiment proper al codi Java.
- Facilitar la concurrència sense bloquejos, mitjançant memòria transaccional (STM) i atòmica amb operacions compare-and-swap (CAS).
- Mantenir un nucli petit i simple que pogués ser ampliat gràcies al sistema de macros de Lisp.

Entre 2005 i 2007, Hickey va dedicar prop de dos anys i mig a experimentar amb aquesta idea d'un dialecte de Lisp pragmàtic per a la indústria. L'objectiu era disposar d'un entorn que aconseguís la corba de productivitat d'un llenguatge dinàmic, però sense renunciar al rendiment i a l'ecosistema Java.

L'octubre de 2007 va publicar la primera versió beta (v1.0-Beta) de Clojure, capaç d'executar-se sobre la JVM 1.5. La versió 1.0 estable va arribar el 2009, després de guanyar visibilitat en la JVM Language Summit. Al llarg dels anys s'han afegit característiques com protocols (versió 1.2), reducers, transducers, la llibreria spec, etc..., mantenint el llenguatge al dia amb les necessitats modernes.

3 Ús actual de Clojure

Tot i ser un llenguatge relativament jove, Clojure ha trobat el seu lloc en diversos àmbits professionals. Inicialment va atreure principalment desenvolupadors Java interessats en la programació funcional. La seva adopció dins l'ecosistema JVM ha anat incrementant amb els anys: el fan servir sobretot equips reduïts però experts, que valoren la programació funcional i la concurrència segura que ofereix.

Avui dia, Clojure s'utilitza en producció en nombroses empreses i projectes. Un estudi de l'any 2018 sobre l'ecosistema JVM va revelar que Clojure era el segon llenguatge més usat sobre la JVM com a llenguatge principal, només per darrere de Java. Empreses de primer nivell com Apple, Atlassian, Netflix, Nubank, Puppet o Walmart, entre d'altres, han informat d'usar Clojure en els seus sistemes crítics; fins i tot la NASA ha utilitzat Clojure en alguns projectes.

Malgrat no ser un llenguatge massiu en termes de nombre de programadors, Clojure gaudeix d'una comunitat molt activa i lleial. En l'enquesta de desenvolupadors de Stack Overflow de 2023, Clojure va aparèixer com el quart llenguatge més estimat per part dels qui el coneixen, ja que el 68% dels enquestats amb experiència en Clojure volien continuar utilitzant-lo. No obstant això, només un percentatge petit (al voltant del 2%) de tots els enquestats el van indicar com a llenguatge que desitgen aprendre, reflectint el seu caràcter de nínxol i especialitzat. Com a curiositat, Clojure acostuma a aparèixer també entre els llenguatges de programació millor pagats (per exemple, en les enquestes de Stack Overflow i HackerRank) perquè combina l'alta demanda en posicions especialitzades amb una oferta relativament baixa de professionals.

4 Paradigmes de programació de Clojure

Clojure és un llenguatge de programació multiparadigma, amb un focus principal en el paradigma funcional però que també integra de manera idiomàtica metaprogramació, concurrència i interoperabilitat amb Java.

Definit com un "llenguatge de programació funcional dinàmic" de la família Lisp, la seva sintaxi es basa en S-expressions (expressions prefixades) i una notació mínima, on el codi es representa mitjançant llistes que el reader de Lisp analitza i transforma internament en estructures de dades.

```clojure
;; Definició d'una funció amb S-expression
(defn square [x]
  (* x x)) ;; Notació prefix

;; Aplicació de la funció
(square 5)
; => 25

;; Les expressions com (+ 1 2 3) i (defn square [x] ...) són
;; S-expressions: la primera posició és la funció i la resta arguments.
```

Una de les característiques més poderoses de Clojure és el seu sistema de macros, que permet tractar el codi com a dades i escriure programari que genera programari (meta-programació).

```clojure
;; Definició d'una macro "unless" (el contrari d'un if)
(defmacro unless [pred a b]
  `(if (not ~pred)
     ~a
     ~b))

;; Exemple d'ús
(unless false
  (println "Això es mostra perquè la condició és false")
  (println "Això no es mostrarà"))
; => Això es mostra perquè la condició és false
```

Clojure aprofita al màxim la programació funcional amb dades immutables, funcions de primera classe i ordres superiors. Alguns exemples bàsics:

```clojure
;; map: aplica inc a cada element
(map inc [1 2 3])
; => (2 3 4)

;; filter: filtra per nombres senars
(filter odd? [1 2 3 4])
; => (1 3)

;; reduce: suma tots els elements
(reduce + [1 2 3 4])
; => 10

;; composició de funcions
((comp inc inc) 5)
; => 7

;; partial: fixant el primer argument de +
(def add5 (partial + 5))
(add5 10)
; => 15
```

Clojure incorpora un model de concurrència ric i sense bloquejos per escriure aplicacions multithread de manera segura. El llenguatge proporciona:

- **Atoms**: valors independents que es modifiquen de manera atòmica mitjançant `swap!`.
- **Refs**: transaccions coordinades amb `dosync`, `ref`, `alter` i `commute`.
- **Agents**: estat asíncron actualitzat mitjançant missatges i processat per un thread pool intern.
- **Futures**: computacions en segon pla, on el resultat es recupera quan es dereferencia.

```clojure
;; 1. Atom: comptador concurrent
(def counter (atom 0))
(dotimes [_ 100]
  (future (dotimes [_ 100]
            (swap! counter inc))))
(Thread/sleep 1000)
@counter
; => 10000

;; 2. Ref: transaccions coordinades
(def account-a (ref 100))
(def account-b (ref 200))

(dosync
  (alter account-a - 50)
  (alter account-b + 50))

(println @account-a @account-b)
; => 50 250

;; 3. Agent: estat asíncron actualitzat per un thread pool intern
(def my-agent (agent 0))
(send-off my-agent
  (fn [state]
    (Thread/sleep 100)
    (inc state)))
(await my-agent)
@my-agent
; => 1

;; 4. Future: computació en segon pla
(def slow-result (future (Thread/sleep 500) (* 2 21)))
@slow-result
; => 42
```

5 Sistema d'execució

Clojure s'executa sobre la Java Virtual Machine (JVM), aprofitant-ne tant la infraestructura de rendiment (JIT, garbage collector) com tot l'ecosistema de llibreries Java. El procés bàsic és:

1. **Read**: el REPL llegeix la S-expression i la converteix en estructures de dades internes.
2. **Eval**: aquestes estructures s'avaluen com a codi Clojure.
3. **Compile**: en aquest pas, les expressions es transformen en bytecode Java estàndard.
4. **JIT**: la JVM, al seu torn, optimitza i compila just-in-time aquest bytecode a codi natiu.

Aquest model "Read-Eval-Compile-JIT" és totalment transparent per al desenvolupador. Gràcies al REPL pots escriure qualsevol expressió i veure'n immediatament el resultat. El bytecode generat per Clojure és estàndard de Java, de manera que s'aprofita el compilador JIT (Just-In-Time) de la JVM i el garbage collector de la plataforma sense haver de construir aquests mecanismes des de zero.

Gràcies a això, es combina la flexibilitat d'un llenguatge interpretat amb l'eficiència d'un llenguatge compilat a bytecode JVM.

Un possible inconvenient és el temps d'arrencada de la JVM, que fa que aplicacions molt petites (scripts) tinguin una latència inicial més alta que si s'escrivissin en un llenguatge interpretat de sistema; però eines com Babashka resolen aquest problema permetent executar codi Clojure sense arrencar una JVM completa.

Tanmateix, cal notar que el dinamisme de Clojure té un cost: algunes operacions que involucren reflexió o invocació dinàmica poden ser una mica més lentes que en un llenguatge totalment estàtic. Per mitigar-ho, el compilador de Clojure permet l'ús de type hints (pistes de tipus) en codi crític perquè certs trams s'executin sense reflexió.

```
(defn string-length [^String s]
  ;; Ara la crida a .length evita reflexió
  (.length s))
```

6 Sistema de Tipus

Clojure adopta un sistema de tipus dinàmic. Això vol dir que les variables i funcions no declaren tipus estàtics per als seus valors o paràmetres, i la verificació de tipus es realitza en temps d'execució. En paraules simples, una funció de Clojure pot retornar qualsevol tipus de dada i assignar-se a una variable; només en intentar operar amb aquest valor es comprovarà si suporta l'operació (llençant-se un error en temps d'execució si no és adequat).

La flexibilitat proporcionada per aquest model és alta, ja que permet escriure funcions genèriques que treballen sobre diferents tipus de dades sense acoblament fort. No obstant això, també implica que alguns errors de tipus no es detecten fins que s'executa o prova el codi.

La decisió de Rich Hickey va ser dotar Clojure d'un tipatge dinàmic per mantenir el llenguatge el més simple i flexible possible, seguint la tradició de Lisp. De fet, Clojure no aplica cap comprovació de tipus estàtica pel seu compte: "el sistema de tipus és enteramente dinàmic". Això contrasta amb altres llenguatges funcionals com Haskell o OCaml, que són de tipatge estàtic fort i comproven la correcció de tipus en compilar. A Clojure, en canvi, la responsabilitat de la coherència de tipus recau en el desenvolupador i en les proves.

7 Estudi bibliogràfic (Fonts i qualitat de la informació)

Abans de presentar les referències, cal destacar que, tot i que és fàcil trobar gran quantitat d'informació sobre Clojure –articles, blogs, tutorials i discussions en fòrums– resulta sovint difícil destriar quines són les fonts oficials i més fiables. Per això, consultar sempre la documentació oficial (`https://clojure.org/reference`) i els materials directament publicats o revisats pel seu autor (Rich Hickey) és fonamental per assegurar la validesa i l'actualitat de la informació, ja que el llenguatge es va actualitzant.

- Official Clojure Documentation. (s. d.). Clojure: Reference. `https://clojure.org/reference`

  La pàgina oficial recull assaigs i guies de Rich Hickey i l'equip del core (p. ex.Rationale o Values and Change). Proporciona explicacions sobre disseny, immutabilitat i model d'identitat/estat, a més d'exemples de codi i notes de versió.
  - Qualitat: excel·lent. La documentació oficial és la font principal i amb més autoritat de qualsevol llenguatge de programació, i aquesta està actualitzada i ben documentada amb exemples i descripcions detallades. Com a únic punt negatiu només està amb anglès.

- "Clojure" (2025, gener 10). Wikipedia. `https://en.wikipedia.org/wiki/Clojure`

  Explicació històrica i tècnica: creador, any d'aparició, versions, adopció i disseny, amb referències externes. Ha estat útil per recopilar dates, noms d'empreses i xifres d'enquestes.
  - Qualitat: Bona, però font terciària. Cal contrastar cada dada amb les fonts citades, ja que qualsevol la pot editar, el contingut és força actualitzat (fins 2024) i ofereix una vista general del llenguatge i història.

- Hickey, R. (2009). *The A–Z of Programming Languages: Clojure* [Entrevista]. Computerworld. `https://cse.sc.edu/~mgv/csce330f16/pres/az.pdf`

  Entrevista resposta pel propi Rich Hickey on explica motivacions, decisions de disseny i reptes inicials, aportant context històric directe i cites autèntiques.
  - Qualitat: alta. Font periodística de prestigi i declaracions originals; com a punt negatiu, és del 2009.

- Stack Overflow. (2023). Developer Survey Results. `https://survey.stackoverflow.co/2023/`

  Enquesta anual (>70 000 respostes) que posiciona Clojure com el 4t llenguatge "most loved" i mesura el seu interès d'aprenentatge i també analitza els salaris.
  - Qualitat: bona. Cobertura global i sèrie longitudinal; perceptiva més que acadèmica.

- Emerick, C., Carper, B., & Grand, C. (2012). *Clojure Programming.* Sebastopol, CA: O'Reilly Media. ISBN 978-1449309154.

  Manual extens que cobreix fonaments, seqüències, macros i exemples complets fins a la versió 1.4.
  - Qualitat: alta. Autors comunitaris i editorial tècnica solvent; una mica desactualitzat però essencial per al nucli del llenguatge.

- Fogus, M., & Houser, C. (2014). *The Joy of Clojure* (2a ed.). Shelter Island, NY: Manning Publications. ISBN 978-1935182643.

  Anàlisi idiomàtica profunda sobre filosofia funcional, concurrència (agents/STM) i patrons avançats.
  - Qualitat: excel·lent. Text rigorós i crític, escrit per contribuïdors destacats; lectura exigent.