Razvoj programskog jezika Clojure

Seminarski rad u okviru kursa
Dizajn programskih jezika
Matematički fakultet

Luka Tomašević
mv16352@alas.matf.bg.ac.rs

21. decembar 2019.

Sažetak

Sadržaj ovog rada se bazira na razvoju programskog jezika Clojure. Navedeni su osnovni koncepti jezika kao i ideje koje su uticale na njegov razvoj. Dat je uvid u osnovne uticaje starijih programskih jezika koji su bili inspiracija za njegovo stvaranje, a to su: ML, Scheme, Common Lisp, Erlang, Haskell i Java, kao i vizuelni prikaz uticaja u vidu razvojnog stabla.

Sadržaj

1 Uvod

Poslednjih godina industrijom su vladali objekto-orjentisani i skript jezici, koji zbog svojih nedostataka primoravaju veliki broj programera da žude za novim, potpunijim i interesantnijim jezicima. Jezik koji pratimo u tekstu koji sledi za inspiraciju uzima neke od najstarijih jezika, pionira ideja koje su inspirisale nebrojane programske jezike a jedan od njih je *Clojure*. Clojure je nastao 2007. godine i brzo postaje prihvaćen i gradi veliku zajednicu Clojure programera. Jednostavnost zapisa programa i redukcija koda bili su inicijalni razlozi njegove popularnosti, pored odlične podrške za skalabilnost. Logo jezika Clojure prikazan je na slici 1.

Slika 1: Logo programskog jezika Clojure

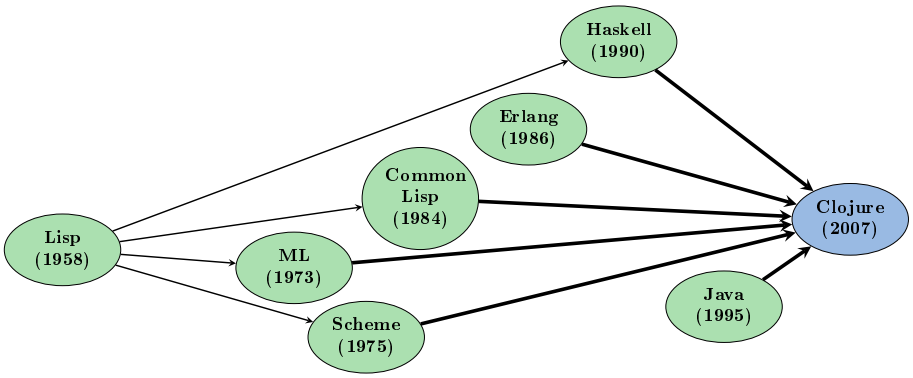
2 Osnovno o Clojure-u

Nakon više od dvadeset godina iskustva u programiranju u jezicima poput: C, C++, Java i C#, Rič Hiki (engl. *Rich Hickey*) odlučuje da razvije sopstveni programski jezik inspirisan lošim iskustvom sa objektno-orjentisanim programskim jezicima. 2007. godine izlazi inicijalna verzija jezika i nazvan je Clojure [2].

Clojure je moderni funkcionalni dijalekt porodice jezika Lisp. Clojure je dizajniran da bude hostovan, tako da koristi **JVM** (engl. *Java Virtual Machine*), takođe takva implementacija jezika dovodi Clojure u blisku vezu sa Javom. Clojure je dinamički jezik opšte namene i vodi se filozofijom, poput mnogih Lispova, **code-as-data**. Pri dizajnu jezika dosta pažnje posvećeno je konkurentnosti što ovaj programski jezik čini odličnim izborom za pisanje programa koji imaju mogućnost iskorišavanja resursa mašine, odnosno programi pisani u ovom jeziku su performantni bez obzira na količinu podataka koja se obrađuje. Clojure je kompajliran jezik, sadrži Lispovski makro sistem, kao pretežno funkcionalni jezik dosta pažnje posvećuje se rekurziji i funkcijama višeg reda odnosno funkcijama koje primaju kao argument druge funkcije, odnosno povratna vrednost im je takođe funkcija. Koncept lenjog izračunavanja je prisutan, izračunavanje izraza se odlaže do same upotrebe, kao i princip imutabilnosti.

3 Razvojno stablo

Imajući u vidu da Clojure pripada Lisp familiji programskih jezika, veliki uticaj na njega imaju ranije implementacije Lispa poput jezika **Common Lisp** i **Scheme**. Jezik je usko povezan sa jezikom **Java**, zbog same činjenice da koristi JVM, a dosta inspiracije crpi iz jezika **Haskell**, **ML** i **Erlang**. Razvojno stablo jezika Clojure može se videti na slici 2.



Slika 2: Razvojno stablo jezika Clojure

3.1 ML

ML (engl. Meta Language) je multi-paradigmatski jezik opšte namene razvijen je od strane Robina Milnera (engl. *Robin Milner*) i njegovih kolega sedamdesetih godina 20. veka tokom rada na automatskom dokazivaču teorema [7]. Pripada funkcionalnoj i imperativnoj paradigmi sa sintaksom baziranom na programskim jeziku **ISWIM** [6]. Nije čisto funkcionalan što znači da dozvoljava mutabilnost podataka. ML je staticki tipiziran jezik koji koristi Hindli-Milnerov tipski sistem (engl. *Hindley-Milner*) [9] što se ogleda u strogoj tipizaciji i kao rezultat čini, dobro napisan, ML program nepodložan greškama tokom izvršavanja. ML takođe sadrži automatski sistem upravljanja memorijom preko sakupljača smeća (engl. *garbage collection*) gde se automatski oslobađa korišćena memorija nakon završetka procesa. Za razliku od čisto imperativnih jezika, gde je implementiranje funkcija na osnovu ulaznih parametara koje su funkcije komplikovano, u ML-u je to pojednostavljeno jer su sve funkcije u lambda računu višeg reda. Još jedna od odlika ML-a je parametarski polimorfizam čime je omogućena upotreba generičkih funkcija i generičkih tipova podataka. ML, kao i Clojure pripada Lisp-1 [1]modelu podele Lispova. ML na Clojure ima najveći uticaj u vidu upotrebe referentnih tipova.

```
1000 val r = ref 5;
1002 !r;
1004 r := !r + 4;
1006 !r;
```

Listing 1: ML

```
1000 (def r (ref 5))
1002 @r
1004 (dosync (alter r + 4))
1006 @r
```

Listing 2: Clojure

Slika 3: Kôd referentnih tipova u ML-u i Clojure-u

[1]Lisp-1 i Lisp-2 su modeli koji se koriste za opisivanje načina korišćenja identifikatora, tj. Lisp-1 opisuje model u kom jedan identifikator može biti vezan samo za promenljivu ili samo za funkciju, dok u Lisp-2 modelu jedan identifikator se može koristiti i za promenljivu i za fukciju.

3.2 Scheme

Još jedan dijalekt Lisp-a, programski jezik **Scheme** imao je veliki uticaj na Clojure. Scheme je jezik opšte namene, on je multiparadigmatski, uticaj Lisp-a ga čini funkcionalnim, a uticaj programskog jezika ALGOL 60 [2] ga čini imperativnim [3]. Razvijen je od strane Gaja Stilija (engl. *Guy Steele*) i Džeralda Susmana (engl. *Gerald Sussman*) tokom sedamdesetih godina prošlog veka. Jezik Scheme je minimalistički, trenutni standard sadrži samo pedeset stranica. Kao i ostali dijalekti Lisp-a, Scheme ima vrlo malo sintakse. Ima makro sistem koji omogućuje programeru da doda nove sintaksne konstrukcije u sam jezik tokom pisanja koda, što čini jezik veoma fleksibilnim. Najveći uticaj programskoj jezika Scheme na jezik Clojure predstavlja upotreba Lispovske **S-notacije**, odnosno principa smeštanja samog izvornog koda, pored podataka, u binarno stablo.

3.3 Common Lisp

Common Lisp je dinamički tipiziran multiparadigmatski jezik opšte namene, pripada proceduralnoj, funkcionalnoj i objektno-orjentisanoj paradigmi [8]. Nastao je kao potreba standardizacije već postojećih Lisp dijalekata: Lisp Machine Lisp, Spice Lisp, New Implementation of LISP (NIL) i S-1 Lisp, odnosno kao poboljšanje verzije MacLisp-a[3].Common Lisp sadrži CLOS [4]. Moguća je upotreba Lisp makroa, transformacije koda, kao kod većine implementacija Lisp-a, kao i upotreba čitačkih makroa (engl. reader macros) koji parsiraju ulazne karaktere. Common Lisp je kompatibilan sa starom verzijom standarda, odnosno sa MacLisp-om kao i sa prvom verzijom jezika Lisp. Konačna verzija Common Lisp-a objavljena je 1994. godine pod ANSI [1] standardom. Common Lisp koristi S notaciju. Najveći uticaj na Clojure od svih jezika navedenih u ovom radu definitivno ima Common Lisp. Pored sintakse inspirisane originalnim Lisp jezikom, a kasnije standardizovane pojavom Common Lisp-a, koncepta makroa, upotrebe **REPL**-a [5], itd. Common Lisp je jedan od ključnih blokova iskorišćenih za razvoj Clojure-a.

3.4 Erlang

Erlang je funkcionalni jezik opšte namene razvijen od strane kompanije Erikson (engl. *Ericsson*), njegovi tvorci su: Džo Armstrong (engl. *Joe Armstrong*), Robert Virding (engl. *Robert Virding*) i Majk Vilijams (engl. *Mike Williams*) [10]. Dizajn Erlang-a je velikim delom bio baziran na konkurentnosti i kao takav pogodan je za pisanje programa koji manipulišu velikim brojem procesa istovremeno. Takođe osnovne odlike programa pisanih u Erlang-u su: laka distribucija na računare koji se nalaze na mreži, tolerancija na greske, odnosno sigurnost na softverske i hardverske greške, skalabilnost i laka nadogradivost, tj. mogućnost menjanja delova programa bez potrebe za gašenjem procesa. U Erlang-u podaci su imutabilni

[2] ALGOL 60 (skraćenica od engl. *Algorithmic Language 1960*) je proceduralni, imperativni jezik iz familijie ALGOL jezika. Prvi je jezik koji implementira ugnježdene funkcije sa pravljenjem zasebnog steka. Nastao je 1960. godine pod uticajem jezika ALGOL 58

[3] MacLisp je dijalekt Lisp-a razvijen od strane Ričarda Grinblata (*engl. Richard Greenblatt*) na MIT-u 1966. godine.

[4] CLOS - (engl. *Common Lisp Object System*) je potpora za objektno-orjentisano programiranje koja je deo ANSI Common Lisp-a

[5] REPL (engl. *read–eval–print loop*) je jednostavno interaktivno okruženje koje od korisnika uzima input i nakon izvršavanja vraća na izlaz obrađen output

i podržan je „pohlepni“ način evaluacije. Clojure je preuzeo najbolje od Erlang-a a to je implementacija distributivnih programa. Imajući u vidu da je Erlang pogodan za obradu velikog skupa podataka mnoge kompanije poput: Fejsbuka (engl. *Facebook*), Ti-Mobajla (nem. *T-Mobile*) i Votsapa (engl. *Whatsapp*) koriste ga na svakodnevnom nivou.

3.5 Haskell

Haskell je statički tipiziran funkcionalni jezik baziran an semantici programskog jezika Miranda[6] [4]. Jezik je baziran na alfa računu i pored toga jedna od osnovnih karakteristika Haskell-a je „lenjost“, preciznije lenjo izračunavanje. Lenjo izračunavanje se ogleda u tome što se izrazi neće izračunati ako u datom trenutku to nije neophodno. Haskell je modularan, tj. program pisan u Haskell-u je niz funkcija, samim tim može se reći da je Haskell program kolekcija manjih Haskell programa. Clojure i Haskell dele gotovo isti zapis osnovnih funkcija 4, ali ono po čemu Haskell inspiriše Clojure je upravo lenjo izračunavanje.

```
take 2 [1,2,3,4,5]
drop 2 [1,2,3,4,5]
sum [1,2,3,4,5]
product [1,2,3,4,5]
cycle [1,2,3]
repeat 9
iterate (1+) 0
takeWhile (<3) [1,2,3,4,5]
dropWhile (<3) [1,2,3,4,5]
splitAt 3 [1,2,3,4,5]
any (<5) [1,2,3,4,5]
all even [2,4,6,8] [x*2 | x <-
    [0..], x*2 < 9]
zip [1,2,3] [10,11,12]
irb>Concat.new("Hello").join("world!
    ")
irb> => "Hello world!"
```

Listing 3: Haskell

```
(take 2 [1,2,3,4,5])
(drop 2 [1,2,3,4,5])
(reduce + [1,2,3,4,5])
(reduce * [1,2,3,4,5])
(cycle [1,2,3])
(repeat 9)
(iterate + 0)
(take-while #(< 3) [1,2,3,4,5])
(drop-while #(< 3) [1,2,3,4,5])
(split-at 3 [1,2,3,4,5])
(some #(< 5) [1,2,3,4,5])
(every? even? [2,4,6,8])
(for [x (range) :let [y (* 2 x)]
    :while (< y 9)] y)
(map vector [1,2,3] [10,11,12])
```

Listing 4: Clojure

Slika 4: Kôd osnovnih funkcija u jezicima Haskell i Clojure

3.6 Java

Java je objekto-orjentisan jezik opšte namene koji je klasno implmentiran. Dizajniran je od strane Džejmsa Goslinga (engl. *James Gosling*) za kompaniju San Mikrosistemi (engl. *Sun Microsystems*) 1995. godine. Java je sintaksno bazirana na jezicima C [7][11] i C++ [8][5], na umu dizajnera bila je portabilnost, odnosno kompajliran java kod se može pokrenuti sa bilo koje platforme koja podržava Javu. Java ima implementiran automatski sakupljač podataka (engl. *garbage collector*) koji oslobađa memoriju korišćenu za vreme života objekta. Java je statički tipiziran jezik, što znači

[6] Miranda je čisto funkcionalni programski jezik koji favorizuje lenjost. Dizajner ovog jezika je Dejvid Tarner (engl. *David Turner*), jezik je objavljen 1985. godine i prvi je čisto funkcionalni jezik komercijalne prirode.

[7] C je proceduralni programski jezik opšte namene nastao 1972. godine u Belovim laboratorijama(engl. *Bell Labs*). Njegov tvorac je Denis Riči (engl. *Dennis Ritchie*)

[8] C++ je objektno-orjentisan, proceduralan, generički programski jezik opšte namene. Nastao je 1985. godine kao nadogradnja na programski jezik C

da tip svake promenljive i tip svakog izraza je poznat za vreme kompilacije, svaka promenljiva posle inicijalizacije može primiti promenljivu istog tog tipa. Java ima bogate biblioteke, što je upravo jedan od razloga zašto je ušla u osnovu Clojure-a.

4 Zaključak

Imali smo priliku da vidimo razvoj programskog jezika Clojure, kao i njegove najveće uticaje. Svi jezici spomenuti u ovom radu samo su opisani svojim najosnovnijim karakteristikama i svojstvima, za više informacija za svaki od njih pogledati literaturu.

Literatura

[1] ANSI. Zvanična stranica ANSI instituta. https://www.ansi.org/.

[2] Clojure. Zvanična stranica programskog jezika Clojure. https://clojure.org/.

[3] R. Kent Dybvig. *The Scheme Programming Language*. The MIT Press, 2009.

[4] Haskell. Zvanična stranica programskog jezika Haskell. https://www.haskell.org/.

[5] ISOC++. Zvanična stranica C++ jezika. https://isocpp.org/.

[6] P.J. Landin. The next 700 programming languages. *Magazine Communications of the ACM*, March 1966.

[7] Chin-Liang Chang; Richard Char-Tung Lee. *Symbolic Logic and Mechanical Theorem Proving*. Academic Press, 1978.

[8] Common Lisp. Zvanična stranica programskog jezika Common Lisp. https://common-lisp.net/.

[9] Robin Milner. A theory of type polymorphism in programming. *Journal of Computer and System Sciences*, December 1978.

[10] Erlang OTP. Zvanična stranica programskog jezika Erlang. http://erlang.org.

[11] Brian W. Kernighan; Dennis M. Ritchie. *The C Programming Language*. Prentice Hall Software Series, 1978.