OCaml vs Clojure

Yang Shouxun (yang.shouxun AT)
FLTRP

March 4, 2012

Outline

- Why OCaml in 2002?
- Introduction to OCaml
- Pain Points in OCaml
- Why Clojure now?
- Pain Points in Clojure
- What is better in OCaml?

Why OCaml in 2002?

Choose a principal language for myself:

Perl and Common Lisp were too slow for my project.

- high-level, garbage collection is a must
- interactive development
- free/open-source implementation
- small footprint, high performance
- active development, a promising future
- good documentation/books available for learning

The winner is OCaml in 2002.

Its version was 3.02 on Debian Potato.

Introduction to OCaml

- created by Xavier Leroy and others in 1996
- one of two major dialects of ML
- History: Caml -> Caml Light -> Objective Caml
- Homepage: http://caml.inria.fr
- Current version: 3.12.1 (July 4, 2011)

Introduction to OCaml

- official documentation
- English translation of "Developing Applications With Objective Caml"

  a free on-line book covering OCaml 2.04
- other on-line resources

  e.g. Jason Hickey’s "Introduction to Objective Caml"

Introduction to OCaml

- F# based on OCaml

  possible to program in both OCaml/F# with minimal changes
- GODI package manager

  not tried, using Debian system instead
- ocsigen

  a web server and programming framework in OCaml
- OCaml-Java project

  the site basically inaccessible in China
- ocamlbuild

  introduced in OCaml 3.10
- ocamlcore.org
- OCamlForge

Introduction to OCaml

- interactive toplevel
- bytecode compiler and native code compiler
- Emacs mode: caml-mode, tuareg-mode, typerex

in your .emacs file:

```
;; Loading TypeRex mode for OCaml files
(add-to-list 'load-path "/usr/local/share/emacs/site-lisp")
(add-to-list 'auto-mode-alist '("\.ml[iylp]?" . typerex-mode))
(autoload 'typerex-mode "typerex" "Major mode for editing Caml code" t)

;; TypeRex mode configuration
(setq ocp-server-command "/usr/local/bin/ocp-wizard")
```

M-x typerex-run-caml will open a new *caml-toplevel buffer in Emacs

Pain Points in OCaml

Two sets of operators for integers and floats:

```
      Objective Caml version 3.12.1

4 + 3.5;;
Characters 4-7:
  4 + 3.5;;
      ^^^
Error: This expression has type float but an expression was expected of type int
```

Pain Points in OCaml

Very easy to forget using the float version

```
(float 4) +. 3.5;;
=> - : float = 7.5
```

Maybe a little bit faster, but a bit low level

Pain Points in OCaml

Satisfying the typing system when working with complex object hierarchy is not fun

e.g. GDome, LablGTK2

Pain Points in OCaml

No built-in heterogeneous collection data types

You have to define a sum type and a set of functions handling the sum type or go through the object-oriented approach.

Pain Points in OCaml

Integer Overflow

On 64-bit: -2^62 .. 2^62 - 1

On 32-bit: -2^30 .. 2^30 - 1

```
min_int - 1;;
=> - : int = 4611686018427387903

max_int + 1;;
=> - : int = -4611686018427387904
```

You have to use the arbitrary-precision library early on and another set of operators.

Pain Points in OCaml

OCaml Standard Library issue

- extlib: extended standard library for OCaml

  minimal changes
- core: Jane Street's standard library for OCaml
- batteries included: community-driven effort

  to standardize on an uniform, documented, and comprehensive OCaml development platform

Why OCaml revisited

- lack a leader like Torvalds Linus to Linux or Rich Hickey to Clojure
- success of the language is not the key objective of the development team
- conflicts/split inside development team and community
  - the standard syntax vs the revised syntax
  - camlp4 vs camlp5
  - stdlib, extlib, core, batteries-included
- some commercial use in industry, but losing ground to new languages?

  time window for popularity was closed

Introduction to Clojure

- created by Rich Hickey
- first public release in Oct. 2007
- a modern Lisp dialect
- designed for the JVM, but ported to orther platforms
- homepage: http://www.clojure.org
- current version: 1.3 (September 23, 2011)

Why Clojure?

To me:

- attracts my attention when it's first announced
- I love Lisp languages
- Clojure is Lisp reloaded

  with cleaner syntax and persistent data structures
- JVM and interoperabilty with Java
- Lisp macros

  CamlP4 works on grammar tree, not as easy to understand as Lisp macros, and the syntax is not as nice
- wonderful community
- concurrency programming support

Why Clojure?

According to Stuart Halloway:

- power: hosted (jvm, .net, ...)
- robustness: functional
- concurrency: identity, state, and time
- abstraction: oo done right
- focus: lisp

Why Clojure?

OCaml compared with Clojure:

- power: not hosted, need wrapper to C libraries
- robustness: functional, static typing
- concurrency: traditional approach
- abstraction: algebraic data typing
- focus: static typing and type inference

Pain Points in Clojure

- no tail call optimization
- Comparatively slow
- Numbers
- Loss of multiparadigm programming

What is better in OCaml?

- pattern matching
- typing: algebraic, static, automatic; structural subtyping
- interface/implementation separation
- efficient implementation
- smaller footprint
- faster startup
- prefix notation is not most intuitive for some cases

What is better in OCaml?

static typing

if a then b
=> b must produce a value of unit

if a then b else c
=> b and c must produce values of a compatible type

static typing helps catch some bugs

What is better in OCaml?

a bug in clojure/string.clj

```
(defn- replace-first-by
  [^CharSequence s ^Pattern re f]
  (let [m (re-matcher re s)]
    (let [buffer (StringBuffer. (.length s))]
      (if (.find m)
        (let [rep (f (re-groups m))]
          (.appendReplacement m buffer rep)
          (.appendTail m buffer)
          (str buffer))))))
```

called by replace-first

No else branch

What is better in OCaml?

a bug in clojure/string.clj

```
(require ’[clojure.string :as s])

(defn encode [w]
  (s/replace-first w #"[0-9a-fA-F]+" #(str (Integer/parseInt % 16))))

(map encode (s/split "This is a long sentence." #" "))

=> (nil nil "10" nil "s14ntence.")
```

The bug shows up when calling replace-first with pattern and a function of match and no match is found.

What is better in OCaml?

With Sum types, pattern matching is very powerful, and destructuring is not as good as real pattern matching

What is better in OCaml?

```
let rec ack m n =
     match m,n with
  | 0,n -> n + 1
  | m,0 -> ack (m - 1) 1
  | m,n -> ack (m - 1) (ack m (n - 1))

=>  val ack : int -> int -> int = <fun>
```

in Clojure:

```
(defn ack
  [m n]
  (if (zero? m)
    (inc n)
    (if (zero? n)
      (ack (dec m) 1)
      (ack (dec m) (ack m (dec n))))))
```

Interesting Projects in Clojure

- Typed Clojure
- Clojure-in-Clojure
- ClojureCLR
- ClojureScript
- ClojureScript/One
- Clojure-Py