Porting Common Lisp code to Clojure

Asked 15 years, 9 months ago Modified 13 years ago Viewed 8k times

35

How practical is it to port a Common Lisp application to Clojure? To be more specific, what features exist in Common Lisp that do not exist in Clojure, and would have to be re-written?

lisp clojure common-lisp

Share

Improve this question

Follow

edited Mar 12, 2009 at 17:39

Marko

31.3k ● 18 ● 78 ● 108

asked Feb 27, 2009 at 16:43

Paul

6,439 ● 4 ● 36 ● 45

4 Answers

Sorted by: Highest score (default)

32

There's a list on clojure.org of differences between Clojure and other Lisps. Some other things I've noticed using Clojure:




- Idiomatic Clojure leans heavily toward immutable data structures. Anywhere you see `SETF` in CL may have to be changed in Clojure to take full advantage. (You always have the option of using mutable Java data structures in Clojure, but most people don't.)
- Clojure's multimethods are similar to CL's (arguably more powerful, because you can dispatch on things other than type) but a full-blown CLOS is not available in Clojure. Clojure uses `struct` instead, which is just a fancy hashmap. Java's OOP system is also available, of course. Some people are working on porting CLOS to Clojure but I'm not sure how far along those efforts are at this point.
- Clojure macros work slightly differently than CL macros when it comes to symbol/namespace resolution. I'm not sure if I understand well enough to elucidate the differences. You don't have to mess with gensyms quite as much in Clojure though, which is nice.
- Clojure doesn't have a condition system like CL's. You have only Java's `try`/`catch`/`finally` for exception handling.
- Clojure doesn't allow user-defined reader macros.
- Clojure doesn't have multiple return values. Destructuring in Clojure is very nice (supports lists, vectors, hash-maps, sets etc.) and it's built into more places than CL by default, so this is less of an issue than it could be.

Depending on the app and how it's written, it may be practical and straightforward to port from CL to Clojure, or it may be more practical to rewrite it from the ground up in a more functional, thread-safe way to fit better with Clojure style.

Share Improve this answer Follow

edited Mar 19, 2009 at 22:29

answered Feb 27, 2009 at 20:45

Brian Carper

72.8k ● 28 ● 170 ● 169

9 Also Clojure is a Lisp 1 (with namespaces), whereas CL is a Lisp 2. This is a very important difference which effects macros hugely, and, to a lesser degree, variable and function declarations. – dsm Mar 13, 2009 at 13:06

1 CLOS multimethods can dispatch on type or identity. What else can Clojure dispatch on? – Anonymous Coward Mar 21, 2009 at 18:59

6 It can dispatch on anything. You supply your own arbitrary dispatch function. – Brian Carper Mar 21, 2009 at 20:39

1 There is a user contrib library called error kit which implements much of CLOS condition/restart handling. – David Plumpton Jul 16, 2009 at 22:38

1 @BrianCarper In, CL A generic function and its defmethods is not a multiplexer that choses which method to call. It composes a function (effective method) from all the applicable defmethods. There are a 8 built-in 'blending' strategies and one can define your own if needed. IMHO that is more powerful not less. Although the default method combination is generally good enough. I imagine that

Clojure's multimethods are good enough for the common case as well. Also generic functions dispatch on classes, not types. – PuercoPop Dec 17, 2015 at 17:03

8

I don't have a specific answer, but I'd recommend these resources:

- Rich Hickey's two part talk Clojure for Lisp Programmers
- Stuart Halloway's work on translating the examples from Peter Seibel's *Practical Common Lisp* to Clojure.

Share Improve this answer Follow

edited Feb 27, 2009 at 21:01

answered Feb 27, 2009 at 16:54

zweiterlinde

14.8k ● 2 ● 29 ● 33

I don't believe that work is by Peter Seibel. Stuart Halloway is the one porting from CL to Clojure. – Brian Carper Feb 27, 2009 at 20:29

Thanks, Brian. Fixed. I did just buy Stuart's preprint, so perhaps he'll forgive me :) – zweiterlinde Feb 27, 2009 at 21:02

There are a number of accounts of transitioning from CL to Clojure (blog, another blog, Thread on HN).

6

The biggest problem a lot of Common Lispers have with Clojure when they first check it out is its lack of **Tail Call Optimization**, which isn't possible on the the JVM.

Share Improve this answer

Follow

edited Dec 17, 2011 at 0:26

amalloy

91.7k ● 8 ● 147 ● 212

answered Feb 27, 2009 at 17:05

Brian Gianforcaro

27.1k ● 11 ● 59 ● 78

6 That's why you use recur instead. – Benjamin Confino Feb 27, 2009 at 20:51

1 and trampolines as well. However there has been some progress on getting TCO into the JVM via the OpenJDK project. – dnolen Feb 27, 2009 at 22:08

4 Many CLs don't, either (it's not required), and I don't know any program(mers) who depend on it. e.g., SBCL/CMUCL doesn't unless you use the dynamic-space optimization. – Ken Mar 1, 2009 at 17:44

1 Its not impossible to have TCO in JVM. see this discussion. stackoverflow.com/questions/1168059/… – unj2 Jul 24, 2009 at 2:59

It's true that many CLs don't do tail call optimization either, but this is less of an issue in Common Lisp because it has a more imperative nature than Clojure. – mshroyer Oct 10, 2010 at 0:11

For idiomatic CL code it's a rewrite.

3

- CL is imperative, Clojure is more purely 'functional'.
- CL is object-oriented (CLOS), Clojure not (uses Java objects and has some OO-mechanisms)
- Identifiers and syntax are mostly different.
- Data structures are different.
- CL is mostly strict (non-lazy), Clojure uses lazy computation.

Even porting the necessary infrastructure (CLOS, Error Handling, Streams) to Clojure makes little sense, since Clojure is really a different language with a different programming style.

If one does not want to rewrite the code, there is for example ABCL, which is a Common Lisp for the JVM.

Share Improve this answer

Follow

answered Nov 27, 2010 at 13:49

Rainer Joswig

139k ● 11 ● 227 ● 354