Intro to Clojure Macros

What are macros?

- Code that generates code
- Meta-programming
- Internally, Clojure relies heavily on macros

Why macros?

- Factor out repetitive code
- Alter core language
- DSLs (add syntax)
- Don't want *f* args evaluated
- Programmer productivity

Characteristics

- Homoiconicity
- Lisp: Code and data are the same [REPL break]
- Macros return forms not values
- Enable syntactic abstraction

- Acts at compile time not runtime
- Macros are expanded once during compilation

Abstract Syntax Tree

- Many languages are lexed and parsed to arrive at the AST
- In Lisp you program to the AST expressed as a Lisp data structure!
- Can manipulate these data structures w/ rich lisp data structure APIs!

Code

Reader

Macro System

Forms / AST

Compilation

Macros in Practice

- Writing macros is an iterative process
- defmacro similar to defn
- 2 quoting forms: ‘ `
- [repl break,unless example]

Tools

- macroexpand
- macroexpand-1

Templating

- Enter literal form and splice out
- Note: symbols are NS qualified
- Splicing: ~ ~@
- [repl break unless again]

Avoiding capture

- Symbols are NS qualified. Could interfere with context
- Macro system can generate “safe” symbols with the # suffix
- [repl break, ? macro]

More Macros

- comment
- declare
- -> ->>
- contextual-eval, JoC 8.1.1

Caveats

- Prefer functions over macros
- Premature optimization
- Gnarly. Now have to reason at 2 levels; compilation time, runtime.

- Several libs re-done to use macros less, e.g., Enlive, ClojureQL
- C. Grand, only add macros at the end to clean up ugly APIs.

Resources

- Practical Clojure
- Joy of Clojure
- Clojure in Action
- Programming Clojure
- Let over Lambda
- On Lisp