Clojure in Action

Creighton Kirkendall

Architect/Snapping Turtle Wrangler

Principal Consultant SEI

Email: ckirkendall@gmail.com

Twitter: @crkirkendall

Clojure in Action

λCincyf(P)

- Clojure Syntax (lisp)
- Building our First Project
  - Leiningen & Counterclockwise
- Channels (Aleph/Lamina)
- Log Server
- Java Interop
- Log4j Appender

Clojure in Action

Clojure Syntax (lisp)

Name	Clojure Syntax	Java Equivalent
Symbols	atom, foo-bar, *foo*, etc.	Variables Names
Literals	42, "foo", nil, true, false, \c, :foo	Same
Keywords	:foo (like symbols, prefixed with colon)	None
Lists	(a b c) & '(a b c)	LinkedList
Vectors	[a b c]	Array
Maps (hashes)	{:a 1 :b 1} or {:a 1, :b 2}	Map
Sets	#{:a :b :c}	Set

Clojure in Action

Clojure Syntax (lisp)

Name	Clojure Syntax	Java Equivalent
Variable Definition	(def a “test”)	String a=”test”;

Function Definition	(fn [x] (println x))
	
	#(println %1)
	
	(def tmp (fn [x] (println x))
	
	(defn tmp [x] (println x))
	
	(defn- tmp[x] (println x))
		No parallel for in-line functions
		
		public static void tmp(Object x){
		System.out.println(x);
		}
		
		private static void tmp(Object x){
		System.out.prinln(x);
		}

Calling a function	(tmp “test”)	tmp(test);

Clojure in Action

Building our First Project

Leiningen (Clojure build tool similar to maven)

Description	Syntax
Create New Project	**lein new <project name>**
Download Dependencies	**lein deps**
Run Unit Tests	lein test
Jar up project files	lein jar
Launch a repl with project in classpath	lein repl
Clean project	lein clean
Create Eclipse Project Files – note this requires a dev dependency	**lein eclipse**
Build the coolest thing ever!	**lein uberjar**

Clojure in Action

Building Our First Project

Leiningen – project.clj

```
(defproject aleph-server "1.0.0-SNAPSHOT"
  :description "FIXME: write description"
  :dependencies [[org.clojure/clojure "1.2.1"]
                 [aleph "0.2.0-alpha1"]]
  :dev-dependencies[[lein-eclipse "1.0.0"]]
)
```

Clojure in Action

Building Our First Project

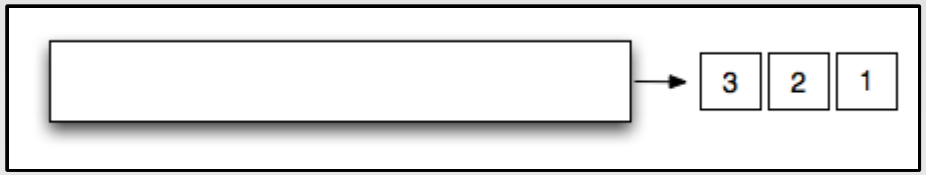
Counterclockwise (Clojure Eclipse plugin)

- Source code editing:
  - syntax coloring, rainbow parens
  - interaction with the REPL
  - paredit mode (80% done)
  - auto-indentation
  - Navigation
- Debug
  - Ability to add breakpoints to Clojure source code
- REPL
  - syntax coloring
  - code completion

Clojure in Action

Channels - Lamina

- Channels are event message queues



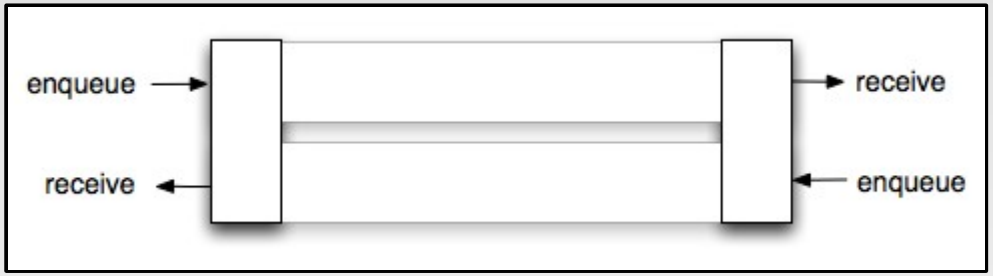

```
> (def ch (channel 1))
<== [1]
> (enqueue ch 2 3)
true
> ch
<== [1 2 3]
> (receive-all ch #(println "a:" %1))
a:1
a:2
a:3
```

Other functions
(receive ch) – one message
(receive-all ch) – all messages
(fork ch) – duplicate stream
(enqueue ch msg) – add msg
(close ch) – close channel
(close? ch) – check if closed

Clojure in Action

Channels - Aleph

- Communication framework based on channels




```
(defn handler [ch client-info]
  (receive-all ch
    #(enqueue ch (str "You said " %))))

(start-tcp-server handler {:port 10000,
                           :frame (string :utf-8 :delimiters ["\r\n"])})
```

Clojure in Action

Logging Server

```
(ns org.cincyfp.log.server.core
  (:import  (java.io.File)))

(use 'lamina.core 'aleph.tcp 'gloss.core 'gloss.io 'clojure.java.io)

(def log-writer (writer "/home/creighton/cool.log" :append true))

(def log-channel (channel))

(defn log-handler [channel client-info]
  (enqueue channel "ok")
  (receive-all channel (fn [msg] (enqueue channel "ok")
                       (enqueue log-channel msg))))

(def f (future
     (doseq [msg (lazy-channel-seq log-channel)]
       (.write log-writer (str msg))
       (.flush log-writer))))

(def msg-frame (string :utf-8 :delimiters ["_EM_"]))

(start-tcp-server log-handler {:port 10000, :frame msg-frame})
```

Clojure in Action

Logging Server (Making it more functional)

```
(ns org.cincyfp.log.server.logserver
  (:import  (java.io.File)))

(use 'lamina.core 'aleph.tcp 'gloss.core 'gloss.io 'clojure.java.io)

(def log-writer (writer "/home/creighton/cool.log" :append true))

(def log-channel (channel))

(defn create-handler-func [output-channel]
(fn [channel client-info]
  (enqueue channel "ok")
  (receive-all channel (fn [msg] (enqueue channel "ok")
                     (enqueue output-channel msg)))))

(defn create-logging-future [input-channel print-writer]
  (future
     (doseq [msg (lazy-channel-seq input-channel)]
       (.write print-writer (str msg))
       (.flush print-writer))))


(def log-handler (create-handler-func log-channel))

(def msg-frame (string :utf-8 :delimiters ["_EM_"]))

(start-tcp-server log-handler {:port 10000, :frame msg-frame})

(def logging-future (create-logging-future log-channel log-writer) )
```

Clojure in Action

Java Interop – Clojure to Java

Calling a Method

```
>(.toUpper “fred”)
“FRED”

>(. (System/out) (println “test”))
test

>(.. System (getProperties) (get “os.name”))
“Linux”

>(doto (new java.util.HashMap) (.put "a" 1) (.put "b" 2))
{a=1, b=2}
```

Creating Objects

```
>(new Thread #(println “tmp))

>(Thread. #(println “tmp))
```

Clojure in Action

Java Interop – Java to Clojure

:gen-class (note :gen-interface is similar)

```
(gen-class :name TstAppender :extends org.apache.log4j.AppenderSkeleton)

(defn -append [this event]
  (println (.getMessage event)))

(defn -close [this]) ;nothing to clean up

(defn -requireLayout [this] false)
```

proxy

```
(defn add-mousepressed-listener
  [component f & args]
  (let [listener (proxy [MouseAdapter] []
                   (mousePressed [event]
                                 (apply f event args)))]
    (.addMouseListener component listener)
    listener))
```

Clojure in Action

Log4j Appender

Defining the class and imports

```
(ns org.cincyfp.log4j.appender.core
  (:import  (org.apache.log4j.AppenderSkeleton)
            (org.apache.log4j.spi.LoggingEvent)
            (org.apache.log4j.spi.ErrorCode)
            (org.apache.log4j.Layout)
            (org.apache.log4j.helpers.LogLog))
  (:gen-class
    :name "org.cincyfp.log4j.appender.CljAppender"
    :extends org.apache.log4j.AppenderSkeleton
    :state        state
    :init         myinit
    :methods [[getHost [] String]
              [setHost [String] void]
              [getPort [] int]
              [setPort [int] void]
              [createAsyncHandler [] void]
              [getHandlerFunc [] Runnable]]))

(use 'lamina.core 'aleph.tcp 'gloss.core)
```

Clojure in Action

Log4j Appender

Defining the constructor

```
; retures a vector containing a vector of args to be
; passed to the super constructor and the state
(defn -myinit []
  (let [ch (channel)]
  [[] (ref {:name "clj-append" :channel ch})]))
```

Defining setters and getters

```
(defn -getHost [this]
  (:host @(.state this)))

(defn -setHost [this nm]
  (let [state (.state this)]
        (dosync (alter state assoc :host nm))))
```

Clojure in Action

Important Links

UC Berkeley's – Functional Programming Lectures

http://bit.ly/jfVvGd - first three lectures

Leiningen (Maven like build tool)

https://github.com/technomancy/leiningen/downloads

Counterclockwise (Clojure Eclipse plugin)

http://ccw.cgrand.net/updatesite

Clojure Documentation

http://clojuredocs.org

http://clojure.org

Clojure Reading List

http://clojure.com/reading.html